Principles of Learning in Multitask Settings:
A Probabilistic Perspective

Maruan Al-Shedivat

May 2021

CMU-ML-21-104

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Eric Xing, Chair
Ameet Talwalkar
Ruslan Salakhutdinov
Pieter Abbeel (UC Berkeley)
Rich Caruana (Microsoft Research)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was supported in part by a Google PhD Fellowship in Machine Learning, a CMLH Fellowship in Digital Health, an AWS Research Grant, and the National Institute of Health under award number R01GM114311.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: machine learning, multitask learning, federated learning, meta-learning, few-shot learning, zero-shot generalization, personalization, probabilistic graphical models

To my dearest grandmother, Elvira Romanova,
who taught me to be relentless in the pursuit of excellence.

Abstract

Today, machine learning is transitioning from research to widespread deployment. This transition requires algorithms that can learn from heterogeneous datasets and models that can operate in complex, often multitask settings. So, is there a set of principles we could follow when designing models and algorithms for such settings? In this thesis, we approach this question from a probabilistic perspective, develop a declarative framework for representing, analyzing, and solving different multitask learning problems, and consider multiple case studies ranging from multi-agent games, to multilingual translation, to federated learning and personalization.

The ideas presented in this thesis are organized as follows. First, we introduce our core probabilistic multitask modeling framework. Starting with a general definition of a learning task, we show how multiple related tasks can be assembled into and represented by a joint probabilistic model. We then define different notions of generalization in multitask settings and demonstrate how to derive practical learning algorithms and consistent objective functions that enable certain types of generalization using techniques from probabilistic learning and inference. Next, we illustrate our proposed framework thorough multiple concrete case studies. Each of our case studies is an independent vignette that focuses on a particular domain and showcases the versatility of our framework. Not only we reinterpret different problems from a probabilistic standpoint, but we also develop new learning algorithms and inference techniques that improve upon the current state-of-the-art in each of the considered domains.

Acknowledgments

I find myself extremely fortunate and privileged to have so many fantastic people to thank, without whom the work behind this thesis would not have been possible.

First, I am thankful to my advisor, Eric Xing, for his guidance and support throughout my Ph.D. journey. Eric gave me the freedom to explore and pursue many of my own ideas, challenged my assumptions, strongly encouraged to venture outside the "research comfort zone," and always reminded myself and other members of his lab to focus on developing a taste for selecting and solving research problems that matter. Eric has also fostered a rich academic environment in SAILING Lab, where I have had a great time, learned a lot from many talented group members, and have matured and grown as a scientist and engineer over the years.

I am also indebted to my thesis committee: Pieter Abbeel, Rich Caruana, Russ Salakhutdinov, and Ameet Talwalkar. I have had a great privilege to work with all of them on different projects and have numerous intellectually stimulating discussions, where many of the ideas and intuitions underlying this thesis were born. Many thanks to Pieter for giving me an opportunity to work together at OpenAI during the early years of my Ph.D.—not only our work culminated with a best paper award, but introduced me along the way to many fascinating problems at the frontier of reinforcement learning and learning-to-learn research and helped sharpen my own perspective on the future of ML algorithms and the role of probabilistic modeling. I am also grateful to Ameet, Russ, and Rich for all our discussions and their always sharp insights on topics ranging from basic research to career, life, and beyond.

This thesis would have not been possible without many individuals, friends, and collaborators who I had a great fortune to get to know over the years, across the boundaries of multiple organizations. Together, we studied, worked, worked out, ran and cycled, traveled to conferences and went on weeks-long crazy road trips, organized reading groups, workshops, socials, and events, brainstormed random startup ideas, contemplated life, and did a million other things that made my Ph.D. experience so much more joyful and memorable!

Thank you to CMU faculty I had a pleasure to learn from and interact with inside and outside the classroom: Aaditya Ramdas, Alex Smola, Barnabás Póczos, Geoff Gordon, Larry Wasserman, Matt Gormley, Nihar Shah, Ryan Tibshirani, Tom Mitchell, Zachary Lipton. Many thanks to all my dear CMU friends: Abulhair Saparov, Ahmed Hefni, Andrew Gordon Wilson, Anthony Platanios, Amanda Coston, Avinava Dubey, Ben Eysenbach, Ben Lengerich, Bhuwan Dhingra, Brandon Amos, Carlton Downey, Chris Dann, Eric Wang, Helen Zhou, Josue Orellana, Ivan Stelmakh, Kirthevasan Kandasamy, Lisa Lee, Liam Li, Paul Liang, Po-Wei Wang, Mariya Toneva, Misha Khodak, Mrinmaya Sachan, Otilia Stretcu, Roberto Iriondo, Sashank Reddi, Willie Neiswanger, Xun Zheng, Yifei Ma, YJ Choi, Zhiting Hu, and many others who I had a pleasure to interact with. As every MLD Ph.D. student, I am indebted beyond the words to Diane Stidle whose support of the MLD community is immeasurable.

Thank you to Jakob Foerster, Aditya Grover, Aravind Rajeswaran, Harri Edwards, Yura Burda, Igor Mordatch, Ankur Handa, Cathy Wu, Ryan Lowe, Trapit Bansal, Yuhuai Wu for being great friends and collaborators during my time at OpenAI and beyond. Thank you to Ankur Parikh, Afshin Rostamizadeh, and Jenny Gillenwater for being fantastic hosts during my internships at Google and strong believers in my crazy research ideas, as well as other fellow Googlers I had a pleasure to interact with and learn from: Brendan McMahan, Emily Pitler, Ian Tenney, Jacob Devlin, Manaal Faruqui, Slav Petrov, Zachary Charles, and Zachary Garrett.

Last but not least, I am extremely grateful to my family, my parents Karina and Faisal, siblings Muammar, Raduan, and Darina, my grandmother Elvira, my wonderful wife Nouran and her family for all their unconditional love, support, and belief in me that kept me going throughout all these years.

Contents

List of Figures

List of Tables

List of Algorithms

List of Theorems

Abbreviations

CEN Contextual Explanation Network. 7, 81, 82, 84, 86–101, 103–106, 145, 146, 148–151, 153

CRF Conditional Random Field. 84, 87, 104

CV Computer Vision. 2, 19

EP Expectation Propagation. 14

ERM Empirical Risk Minimization. 18

FL Federated Learning. 5, 6, 14, 32–36, 39, 42, 43, 47, 48

KL Kullback-Leibler. 14, 26, 35, 37, 89

LIME Local Interpretable Model-agnostic Explanations. 83, 86, 87, 92–94, 98, 100, 101

MAML Model-agnostic Meta-Learning. 6, 15, 24, 28, 29, 34, 42, 43, 47, 51–53, 63, 64, 112, 113

MCMC Markov Chain Monte Carlo. 14, 48

MDP Markov Decision Process. 16, 51, 64

MoE Mixture of Experts. 88, 89

MT Machine Translation. 65, 69, 72

NLP Natural Language Processing. 2, 19

NMT Neural Machine Translation. 65, 66, 68, 69, 73

NP Neural Process. 15, 16, 28, 29, 42

PGM Probabilistic Graphical Models. 2, 22

PMM Probabilistic Multitask Modeling. 2–7, 17, 22, 28, 32, 42, 49, 51, 64, 65, 69, 80

ProtoNet Prototypical Network. 15, 16, 24, 28, 29, 42, 112, 113

RL Reinforcement Learning. 2, 6, 16, 25, 49–53, 58, 63, 64, 109

VCM Varying-Coefficient Models. 89

Chapter 1

Introduction

Machine learning is one of the most successful approaches to building intelligent systems today. At its core, *learning* amounts to extracting statistical patterns from data and building models that exhibit *generalization* (*i.e.*, can accurately predict unseen data). This is typically done through optimization of an objective function computed on a training dataset. In other words, machine learning enables *inductive inference* (*i.e.*, allows to discover functional dependencies from examples) in a distributional sense—the learned models are accurate on unseen examples as long as these examples originate from the same underlying distribution as the training data.

With the advent of deep learning (LeCun et al., 2015), training models on massive datasets has led to many breakthroughs across different areas of machine intelligence. Each of the modern successes pertains to a single, well-defined learning problem within a specific, narrow domain (*e.g.*, machine translation between a pair of languages). As machine learning is transitioning from research to widespread deployment in the real world, however, it has to face significantly more complex scenarios (Stoica et al., 2017), ranging from environments with dynamically changing objectives to settings where a high degree of personalization is necessary. However, many settings in which we wish to deploy machine learning models, unfortunately, often break some of the assumptions required for distributional inductive inference to work.

In this thesis, we argue that many of the complex learning problems that arise in the real world can be decomposed into parts and approached using *multitask learning* (Caruana, 1998). As such, our main question of interest is the following:

> *Is there a set of common principles that we could follow when designing models and algorithms for learning in a variety multitask settings that arise in practice?*

Learning models that can handle multiple tasks has been a long-standing problem in machine intelligence, with many different formulations, approaches, and settings of interest. Back in the late 80s and early 90s, Sejnowski and Rosenberg (1987) proposed the first multitask neural network for speech synthesis, Ring (1994) introduced continual learning, and Caruana (1998) explored the effects and demonstrated the power of training models on multiple prediction tasks simultaneously. Fast forward to today, and multitask learning has almost become the default approach with a rapidly growing toolbox of different techniques (*e.g.*, see Ruder, 2017).

To name just a few: instead of training models from scratch, today they are often initially pre-trained on massive datasets and then fine-tuned for the task of interest, which has become a best practice in Computer Vision (CV) (Yosinski et al., 2014; Kolesnikov et al., 2019), Natural Language Processing (NLP) (Mikolov et al., 2013; Peters et al., 2018; Devlin et al., 2018; Radford et al., 2019), and other domains; introducing auxiliary tasks and corresponding losses has shown to be effective in improving performance on the main task of interest and used successfully in Reinforcement Learning (RL) (Ng et al., 1999; Jaderberg et al., 2016); training models on large collections of similar tasks has enabled extremely data-efficient learning, which has been successfully used in robotics (*e.g.*, Duan et al., 2017) and beyond (Hospedales et al., 2020); methods for reducing interference and inducing transfer between tasks is yet another very active area of research (*e.g.*, Chen et al., 2018; Zamir et al., 2018; Chen et al., 2020).

While the toolbox of different multitask learning techniques is extremely rich and keeps rapidly growing, a coherent methodological framework behind these developments is lacking. Unfortunately, this makes most of the existing approaches ad-hoc, application-specific and narrow, and it is often quite nontrivial to adapt them from one problem domain to another.[1] Moreover, what does it mean "to generalize" in a multitask setting? Without a formal framework, the notion of generalization becomes somewhat vague and unclear.

1.1 Goals

This thesis aims to develop a *declarative framework* for representing and solving a variety of learning problems that can be approached from a multitask perspective. Our proposed framework is based on Probabilistic Graphical Models (PGM) (Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Koller and Friedman, 2009), which allow us to separate the declarative representation (or description) of a multitask learning problem in a particular application domain from the general-purpose algorithms used for learning and inference. In the past, graphical models have entirely redefined the way how pattern discovery and data analysis are approached today, not only giving rise to a variety of popular models (such as factor, topic, mixed membership, etc.) but also providing a general algorithmic toolkit that has been applied across different domains. Similarly, as a step toward approaching multitask learning systematically, our goal is to develop a Probabilistic Multitask Modeling (PMM) framework that can be used for many applications.

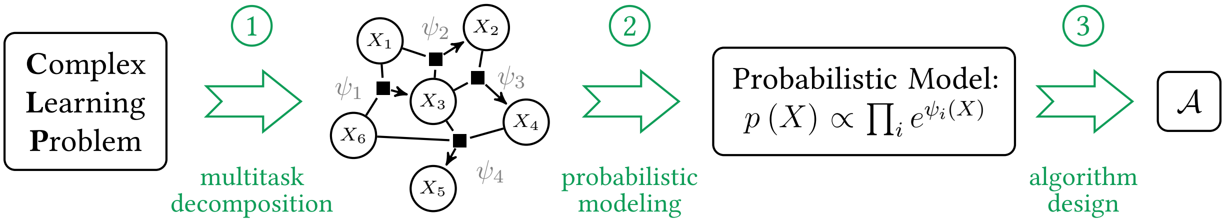


Figure 1.1: The three steps of probabilistic multitask modeling.

[1] For example, after ImageNet pre-training had become standard in computer vision, it required developing new architectures, designing custom loss functions, inventing new training techniques, and over more than 4 years of research before pre-training started to work similarly well for natural language tasks (Ruder, 2018).

Thesis statement. The following two claims are at the core of my thesis.

1. *Many complex learning problems can be decomposed into multiple related learning tasks.*
2. *Using a joint probabilistic model over all tasks allows to define different notions of multitask generalization as well as derive consistent objective functions for learning in such settings.*

As we will see throughout this thesis, many complex learning scenarios—ranging from learning under nonstationarity to personalization and model interpretability—can all be cast as multitask learning problems. Once the problem is decomposed into multiple related tasks, as depicted in Figure 1.1, we can represent them using a probabilistic graphical model and define a joint probability distribution over all the variables across all these tasks. Using this joint distribution, we can further formally define different notions of generalization in each setting of interest (*e.g.*, zero-shot generalization, cross-task generalization, etc.). Further, using probabilistic inference as a general-purpose tool, we will be able to derive objective functions optimizing which will yield multitask models that exhibit the desired notions of generalization.

1.2 Contributions

The main contributions of this thesis are two-fold.

1. Framework. First, it formalizes and presents the PMM framework as we sketched out above. Starting with the general definition of a *task*, we show how multiple related tasks can be assembled into and represented by probabilistic graphical models, in particular, *factor graphs*. We then define different notions of generalization (*i.e.*, formulate the classical inductive inference with respect to conditionals or marginals of the joint multitask distribution) and demonstrate how to derive consistent objective functions as well as design algorithms for optimizing them using approximate probabilistic inference techniques.

2. Vignettes. Second, the framework is illustrated through multiple concrete case studies. Each of the case studies is an independent vignette that focuses on a multitask setting within a specific domain (ranging from natural language generation to multi-agent reinforcement learning), showcasing the versatility of the proposed PMM framework. Each case study not only reinterprets the problem from a probabilistic standpoint, but also develops new learning algorithms and inference techniques that improve upon the state-of-the-art in the given domain.

The Scope. Multitask learning is vast. This thesis by no means attempts to capture and reinterpret all possible aspects, settings, and scenarios that appear, or may appear in the future, in real world applications. Instead, the scope of this thesis is limited to the design principles of the objective functions that enable a few select notions of (out-of-distribution) generalization in multitask settings. Apart from the objective function design, there are other techniques that have been shown to improve learning and generalization in multitask settings (*e.g.*, data augmentation, Andreas, 2019). Interpreting such other techniques from a probabilistic standpoint, while likely possible and potentially fruitful, is beyond the scope of this thesis.

In the following section, we provide a brief overview of each of the contributions, including the framework, each of the case studies, as well as highlight the key results.

1.3 Overview

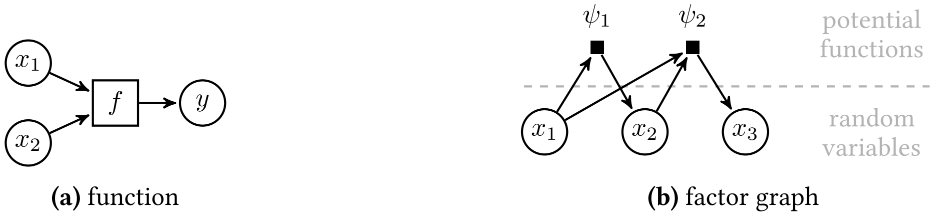
This thesis is organized into three parts. The first part describes the PMM framework in its full generality and connects it with the related work in the multitask learning literature. The second part presents four independent case studies focused on concrete applications of our framework. And the third and final part concludes this thesis, summarizes the key takeaways, and outlines interesting directions of future work. Below, we briefly overview each of the parts.

1.3.1 Overview of the Framework (Part I)

The first part provides the necessary background on probabilistic graphical modeling and probabilistic inference, introduces and discusses in detail each of the three steps of our framework, illustrated in Figure 1.1, and concludes with a brief survey of modern multitask learning literature, which is discussed from the perspective of the introduced framework.

Representing Multitask Learning Problems with Factor Graphs

The first two questions that we ask in this part are: "What is a learning task?" and "How can we describe and represent arbitrary multitask learning problems?" Following Mitchell et al. (2018), we associate a learning task with a function (Figure 1.2a), *i.e.*, a (not necessarily deterministic) mapping from some inputs to outputs, where the goal is to find a function that fits the data well. A multitask learning problem, therefore, consists in learning a collection of such functions (finite or infinite) from a given dataset. Since these functions often share variables, a good representation of a multitask learning problem must capture such relationships. Moreover, it must also allow us to evaluate the "goodness of fit" of the entire collection of functions jointly.



(a) function

(b) factor graph

Figure 1.2: Examples of graphs that represent (a) a function and (b) a factor graph.

We show that *factor graphs* (Kschischang et al., 2001; Frey, 2002) are particularly suitable for representing multitask learning problems. Factor graphs is a class of graphical models that has a bipartite structure with a set of nodes that corresponds to random variables and another set that corresponds to potential functions (Figure 1.2b). Such graphs allow us to explicitly and unambiguously represent the factorization of the underlying joint distribution over the variables. Assigning each learning task a potential function lets us define a join probability distribution over all the variables, tie together multiple tasks, and use standard techniques from probabilistic inference for constructing objective functions and designing new multitask learning algorithms.

Defining Generalization in Multitask Settings and Deriving Consistent Loss Functions

Further, we show how different kinds of multitask learning problems that arise in practice—ranging from fixed collections to distributions over tasks or even subsets of related tasks—can all be represented with factor graphs that define the corresponding joint probabilistic models. Moreover, we show that different types of generalization in multitask settings (such as zero-shot, few-shot, etc.) can be defined with respect to specific conditional and marginal distributions.

As a result, we get a very simple recipe for constructing loss functions for multitask learning: use (variational) bounds or computationally tractable approximations to various log-conditional and log-marginal likelihoods of interest. Such loss functions can be derived mechanistically from the corresponding factor graph representations using standard tools from probabilistic inference literature (Koller and Friedman, 2009). Moreover, we show that optimizing such loss functions often provably leads to desired generalization properties of the resulting models, while optimizing more common losses often does not.

Finally, we conclude this part with a discussion of alternative learning strategies beyond optimization, such as posterior inference over model parameters (MacKay, 1992). Such inference can be done tractably by minimizing a divergence between the true posterior and a variational approximation using message passing algorithms (Minka, 2005), which can be executed very efficiently in distributed settings and can scale to massive numbers of tasks.

1.3.2 Overview of the Case Studies (Part II)

The second part grounds the ideas behind our PMM framework by illustrating them through multiple case studies. Each chapter in this part is focused on a concrete machine learning application that can be cast as a multitask learning problem and is independent of other chapters. Below, we overview the key contributions of each chapter.

Chapter 4: An Inferential Approach to Federated Learning

In our first case study, we focus on Federated Learning (FL)—a framework that enables learning statistical models from heterogeneous datasets scattered across a large number of different entities (*e.g.*, organizations, users, mobile devices, etc., all termed *clients*) without a direct access to their data. The ultimate goal of FL is to provide a best possible model for each client. As such, we can view FL as a *massively multitask learning* problem where each client corresponds to a learning task, the tasks are different but related to each other, and the goal can be achieved by learning a collection of optimal personalized models, one for each client. Depending on the setting, the number of clients can be fixed or unlimited; in the latter case, we have to deal with a distribution over tasks which requires an ability to generalize to new tasks.

FL is not only an interesting problem to tackle due to its growing popularity and the increasing number of real-world deployments (Kairouz et al., 2019), but also because it has simple multitask structure that allows us to illustrate the key ideas and demonstrate the advantages of our probabilistic approach before moving on to more complex scenarios in the chapters that follow.

Canonically, FL is formulated as a distributed optimization problem (McMahan et al., 2017). While convenient, it turns out that this formulation has multiple disadvantages, such as issues with convergence in heterogeneous settings. Much of the recent FL literature has focused on different ways to address some of these shortcomings without reconsidering the original formulation, *e.g.*, by introducing various regularization losses, learning rate schedules, etc.

In this chapter, we introduce a probabilistic view of FL that allows us to reformulate the problem as distributed Bayesian inference rather than optimization. In this new formalism, personalization corresponds to inference of the local parameters from the client data, while learning corresponds to global posterior inference. Next, we design a new algorithm for efficient posterior inference in federated settings, termed federated posterior averaging (FedPA) which generalizes the commonly used in practice federated averaging (FedAvg) (McMahan et al., 2017) and addresses some of its limitations. We also show that FedPA is a special case of *generalized message-passing* (Minka, 2005) and can be seen as a minimization of a divergence between the true posterior and a variational approximation. Finally, we discuss inferential approaches to personalization and empirically study combinations of FedPA with meta-learning algorithms.

Chapter 5: Reinforcement Learning under Nonstationarity

Learning under nonstationarity is a canonical example of a learning problem that we call *complex*. Due to nonstationarity, the data distribution changes or drifts over time, and thus the classical notions of learning and generalization no longer apply in such settings. In this chapter, we study RL in nonstationary environments that gradually change over time and develop methods for training agents that can adapt to such changes.

Noting that any nonstationary process that gradually changes over time can be seen as stationary on a certain time scale, following our main thesis, we propose to decompose such nonstationary problems into sequences of stationary learning tasks represented with Markov chains. As long as the transitions between tasks are predictable from the past observations, such multitask representation allows us to solve nonstationary environments by slightly updating (*i.e.*, *adapting*) the policy before using it for each subsequent task. Further, since nonstationarity allows for only a limited amount of interaction with each task before it changes, it immediately puts learning into the *few-shot regime*, which requires data efficiency.

In this chapter, we formalize these requirements and formulate few-shot generalization in sequential multitask settings using our PMM framework. To do so, we first provide a probabilistic re-interpretation of Model-agnostic Meta-Learning (MAML) (Finn et al., 2017), where our exposition focuses on RL. Then, we extend our formalism to meta-RL on chains of tasks and derive a new loss that can be directly plugged into MAML and optimized using policy gradients (Sutton et al., 2000) to enable continuous adaptation from task to task in a few-shot regime. The proposed approach is extensively validated through an empirical study that compares it with multiple baselines on nonstationary locomotion problems and complex multi-agent games and shows that meta-learning-enabled continuous adaptation is superior to other methods. Finally, we conclude this chapter with a discussion of some notable follow-up work that either builds on, extends, or complements our contributions.

Chapter 6: Zero-shot Consistent Multilingual Machine Translation

Typically, learning models that can effectively generalize to new tasks requires having access to at least a few data samples from each target task, as it was the case in the previous chapters. In this chapter, we focus on *zero-shot generalization*—a setting in which the goal is to train models capable of solving new tasks without any access to the target data. In this case study, we consider multilingual machine translation where tasks correspond to translation directions between different pairs of languages. Zero-shot generalization not only is desirable but critical for building multilingual systems as the data for some tasks (*i.e.*, pairs of languages) can often be either extremely scarce or unavailable, which makes multilingual translation a perfect application as well as test bed.

While multilingual translation is trivial to decompose into multiple tasks, representing all these tasks with a joint model that enables zero-shot generalization is non-trivial. Again, applying our PMM framework, we represent multiple translation tasks with a single factor graph and a corresponding joint distribution over the so-called *equivalent sentences*. Computing or optimizing the marginal likelihood under this model turns out to be intractable. So, we design a tractable objective function, called *agreement-based loss*, which lower bounds the likelihood. We show that optimizing our lower bound provably enables zero-shot generalization, while optimizing the classical cross-entropy objective does not. Finally, we demonstrate the effectiveness of our method on multiple zero-shot translation benchmark tasks, showing substantial improvement of up to +2-3 BLEU points. We conclude with a brief discussion of recent follow up work that extends our approach to unsupervised machine translation.

Chapter 7: A Multitask View of Model Interpretability

When a learning problem admits a natural multitask decomposition (as in Chapters 4 to 6), PMM provides a systematic way to represent the problem, derive a loss function, and learn models that exhibit the desired notions of generalization. In this chapter, we argue that, given additional considerations such as *interpretability of the model predictions*, it might be advantageous to view even classical single-task learning problems from a multitask perspective. But how so?

Many modern single-task learning problems (*e.g.*, image classification) rely on large datasets, learning from which requires using a rich enough family of models that can represent the complexity of patterns observed in the data. Using models that are too simple (*e.g.*, linear), albeit interpretable, is likely to result in sub-par performance, while complex models such as deep neural nets are typically not designed to be human-intelligible. So, in one way or the other, we have to cope with the *accuracy vs. interpretability* trade-off if we approach the problem as stated.

Alternatively, we can decompose the original, complex learning problem into multiple "tasks" each of which is simple enough to be solved with an interpretable model. In this case study, we follow this alternative and decompose various prediction problems into tasks that correspond to the same problem but on a *sub-space* of possible inputs, which we call *the context.* To this end, we introduce a new class of model architectures, called Contextual Explanation Networks (CENs) and show that if each task is solvable by different contexts can be identified reliably, our approach enables overall accurate prediction along with interpretability within each context.

1.4 Bibliographic Notes

Some of the work presented in this thesis has been published in the following venues.

Chapter 3 expands on and generalizes some of the ideas from:

- Maruan Al-Shedivat and Ankur Parikh. "Consistency by Agreement in Zero-Shot Neural Machine Translation". In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2019
- Maruan Al-Shedivat, Jennifer Gillenwater, Eric Xing, and Afshin Rostamizadeh. "Federated Learning via Posterior Averaging: A New Perspective and Practical Algorithms". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021
- Maruan Al-Shedivat, Liam Li, Eric Xing, and Ameet Talwalkar. "On Data Efficiency of Meta-learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2021

Chapter 4 is based on:

- Maruan Al-Shedivat, Jennifer Gillenwater, Eric Xing, and Afshin Rostamizadeh. "Federated Learning via Posterior Averaging: A New Perspective and Practical Algorithms". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021

Chapter 5 is based on:

- Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. "Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments". In: *International Conference on Learning Representations (ICLR)*. 2018

Chapter 6 is based on:

- Maruan Al-Shedivat and Ankur Parikh. "Consistency by Agreement in Zero-Shot Neural Machine Translation". In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2019

Chapter 7 is based on:

- Maruan Al-Shedivat, Avinava Dubey, and Eric Xing. "Contextual explanation networks". In: *The Journal of Machine Learning Research (JMLR)* (2020)

We note that Chapters 4 to 7 present some the previously published results by the author in light of the new unifying probabilistic multitask modeling framework (Chapter 3). As such, our exposition here is more general. In addition, we also briefly summarize and discuss notable follow up work that builds on or extends our ideas at the end of each chapter. We hope that even readers quite familiar with the original papers find our discussion interesting and insightful.

Open Source Contributions

The work on this thesis resulted in multiple open source contributions, including the following:

- **keras-gp** (https://github.com/alshedivat/keras-gp): An library that provides GP layers compatible with the Keras API, which can be used with arbitrary deep neural architectures.
- **robosumo** (https://github.com/openai/robosumo): A set of custom competitive multi-agent MuJoCo environments and pretrained models used for testing continuous adaptation.
- **meta-blocks** (https://github.com/alshedivat/meta-blocks): An efficient implementation of popular meta-learning algorithms in TensorFlow along with tools for easy benchmarking.
- **cen** (https://github.com/alshedivat/cen): Contextual explanation layers for TensorFlow (Keras API), compatible with arbitrary arbitrary deep neural architectures.
- **fedpa** (https://github.com/alshedivat/fedpa): A mini-library for running and visualizing federated learning algorithms. A scalable version was also open sourced through Google.

Excluded Research Contributions

Besides the primary contributions presented in this thesis, there are multiple other notable research developments that have been made throughout the author's PhD journey, which were excluded for clarity and summarized below:

- **Deep Gaussian processes and spectral learning methods.** During the first 1.5 years of PhD, some of our initial work was focused on spectral learning of non-parametric models (Kandasamy et al., 2016) as well as probabilistic (Bayesian) deep learning, with an emphasis on predictive uncertainty quantification in temporal prediction using Gaussian Processes (GPs) with recurrent neural kernels (Al-Shedivat et al., 2017c).
- **Learning in multi-agent settings.** Starting with the internship at OpenAI, we have made multiple contributions to the multi-agent learning literature: enabling agents to learn to reciprocate in general-sum games (Foerster et al., 2018a), learning to model behavior of other agents with policy embeddings (Grover et al., 2018b; Grover et al., 2018a), new algorithms for learning from humans (Platanios et al., 2020). Along the way, new methods for estimating high-order derivatives of stochastic objective functions have been designed (Foerster et al., 2018b; Mao et al., 2019), which turned out to be instrumental in the aforementioned projects.
- **Interpretability of predictive models with applications to healthcare.** In this thesis we will discuss model interpretability in some capacity (in particular, a multitask perspective on the problem). Apart from that, we have also explored the properties and implications of different automatically generated explanations (Al-Shedivat et al., 2017b), applied them to problems in digital healthcare (Al-Shedivat et al., 2017a; Lengerich et al., 2020), and developed a new class of regularizers that can be used for arbitrary models and significantly improve post-hoc explanations of their predictions (Plumb et al., 2020).
- **Miscellaneous.** Finally, there are several other side-projects we had a pleasure to contribute to: quantification of complexity of exploration in reinforcement learning in navigation environments (Al-Shedivat et al., 2018b), methods for controllable and progressive generation of long text passages using large scale pretrained language models (Tan et al., 2021), and the field guide to federated learning (Wang et al., 2021).

1.5 How to Read This Thesis?

Each chapter of this thesis is written to be as self-contained as possible. Chapter 3 is written in a tutorial style: it defines different types of multitask learning problems, present the key ideas behind our probabilistic approach, and connects them to the existing literature. As such, we keep our discussion in Chapter 3 to be mostly application-agnostic and provide only simple examples designed for illustration purposes. Throughout Chapter 3, however, we refer to case studies presented in Chapters 4 to 7, each of which is independent of the other chapters and dives deeper into a particular technique used in the context of a specific application.

The recommended order of reading the reset of this thesis is the following: Chapter 3, then Chapters 4 to 7 in any order (skipping any, depending on the reader's interests and preferences), and finally the concluding Chapter 8. Readers are not expected to have detailed knowledge of probabilistic machine learning or any of the subareas that we dive into in the case studies, as we try to provide the necessary background in Chapter 2 and refer to it throughout.

Part I

Foundations

Probability theory is nothing but common sense reduced to calculation.

Lierre Simon Laplace

When we try to pick out anything by itself, we find that it is bound fast by a thousand invisible cords that cannot be broken, to everything in the universe.

John Muir

In this part, we introduce the probabilistic multitask modeling framework and discuss its connections to some of the classical and modern developments in multitask and meta-learning literature. Chapter 2 provides the necessary background on probabilistic graphical modeling and probabilistic inference, and establishes the notation that we use throughout the rest of this thesis. Then, Chapter 3 introduces and discusses in detail the three steps of our framework, defines different notions of generalization in multitask settings, and presents some formal guarantees for our methods, and briefly surveys some of the popular and successful multitask learning techniques from the literature, connecting them to the introduced framework.

Chapter 2

Background & Preliminaries

2.1 Summary

This chapter provides a minimal technical background on probabilistic modeling—touching upon graphical representations of probability distribution and summarizing different learning and inference techniques—which is necessary to introduce the main contributions of this thesis in the subsequent chapters. We also briefly introduce reinforcement learning and some aspects of deep learning such as encoder-decoder architectures. Readers familiar with most of these concepts are encouraged to skip directly to Chapter 3.

2.2 Probability Distributions and Graphical Models

Throughout this thesis, we will work with probability distributions over random variables. The data (*i.e.*, observations), different latent states (if any), and parameters of the model will all be represented with random variables, which are assumed to be high-dimensional vectors. Probability distributions over these variables will be represented with graphs (Figure 2.1).

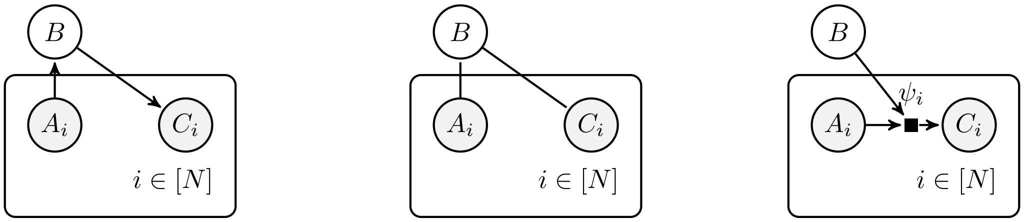


Figure 2.1: Directed (left), undirected (middle), and factor (right) graph representations of $p\,(A, B, C)$.

In the graphical representation of probabilistic models, random variables are represented with nodes, where shaded nodes correspond to observed variables and blank nodes correspond to unobserved (or latent) variables. Groups of indexed variables (*e.g.*, those that correspond to different data points) are scoped with *plates*. We will often times refer to variables within some plates as *local* and outside the plates as *global*, where local variables will pertain to a particular context (*e.g.*, a user, an environment, or even a single data point) while global variables (which will typically be latent) will be relevant across all contexts represented by the model.

While there are multiple different types of graphical models (Koller and Friedman, 2009), throughout this thesis we will use *factor graphs* which represent functional dependencies between random variables explicitly using additional nodes (small black squares, "■"). Although factor graphs generalize directed and undirected graphical models (Frey, 2002), in this thesis we mainly use them for clarity—representing mutltitask learning problems using factor graphs would make it transparent which functions and in what capacity are shared across different tasks and on which variables they operate on. Finally, factor graphs will allow us to define joint probability distributions using the exponential form (Wainwright and Jordan, 2008). For example, the joint distribution for the factor graph in Figure 2.1 can be written as follows:

$$p\left(A, B, C\right) = \exp\left\{\sum_i \psi_i(A_i, C_i, B) - \log Z\right\}, \tag{2.1}$$

where Z is the normalizing constant. Note that factors ψ are arbitrary functions which can be implemented with neural networks.

A note on terminology. In machine learning, the word "model" can be ambiguous and used to refer either to a *probabilistic model* (*i.e.*, an object for which we can compute or estimate the likelihood given some data) or a *computational/predictive model* (*i.e.*, a function that maps some inputs to some outputs). To avoid ambiguity, unless specified otherwise, we will use "model" to refer to probabilistic models, and refer to predictive models explicitly or use "predictor."

2.3 Probabilistic Inference and Learning

Given a probabilistic model, we will often have two primary goals: (i) process the data and estimate the posterior distribution (or its mode) over the latent variables (including local and global parameters of the model), which entails *learning*, and (ii) estimate the predictive distribution and compute predictions, which entails *inference*. While for certain types of models both learning and inference can be done exactly and tractably (see Koller and Friedman (2009) for background), in the majority of practical settings neither is computationally tractable and requires approximations.

2.3.1 Intractability and Approximations

The main source of intractability in probabilistic modeling comes from integration—computing and optimizing the likelihood of the model, estimating posterior distributions, or computing predictive distributions almost always require computation of intractable integrals. Notice that even the joint likelihood in our simple example in Equation 2.1 already requires computation of the normalizing constant Z—this, in turn, requires integration (or summation) over all possible configurations of random variables A, B, C. We will give more concrete examples in Chapter 3.

Monte Carlo Integration and Markov Chains

Monte Carlo methods allow to approximately compute intractable integrals using sampling. In our example with two observed variables A and C and a latent variable B, computing the likelihood $p\left(A, C\right)$ requires integrating out the latent variable: $p\left(A, C\right) = \int_{B \in \mathcal{B}} p\left(A, B, C\right) dB$. If $p\left(A, B, C\right)$ factorizes into $p\left(B \mid A\right) p\left(A\right) p\left(C \mid B\right)$, the integral is equivalent to computing an expectation $\mathbb{E}_{B \mid A}\left[p\left(C \mid B\right) p\left(A\right)\right]$, which can be approximated with sampling:

$$p\left(A, C\right) \approx \sum_{m=1}^{M} p\left(C \mid B = b_m\right) p\left(A\right), \quad \text{where} \quad b_m \sim p\left(B \mid A\right) \tag{2.2}$$

Note that this approximation requires us to be able to draw samples from $p\left(B \mid A\right)$. From uniform, standard normal, and other simple distributions we can draw samples directly. However, for more complex distributions it is not the case. Markov Chain Monte Carlo (MCMC) allows to produce (approximate) samples from arbitrary distributions. In a nutshell, the main idea of the method is to construct a Markov chain with a stationary distribution that matches the one that we would like to sample from. Then, after running the chain for long enough, it should mix and start producing samples from the target distribution.

In Chapter 4, we will use a variant of Langevin dynamics (Welling and Teh, 2011) for MCMC which relies on stochastic gradient descent and will allow us to reinterpret popular distributed optimization algorithms from an inferential perspective.

Variational Approximations

Another way to approximate intractable integrals that come up in probabilistic inference is to convert integration into an optimization problem. The key idea is to approximate an intractable distribution p that requires integration (such as $p\left(A, C\right)$ in our running example) with another distribution from a tractable family $q \in \mathcal{Q}$ by minimizing a divergence between the two:

$$q^{\star} := \arg\min_{q \in \mathcal{Q}} \mathbb{D}[p \parallel q], \tag{2.3}$$

where $\mathbb{D}[\cdot \parallel \cdot]$ is some divergence measure. In this thesis, we will mainly work with Kullback-Leibler (KL), but using some other divergence measure is also possible and would usually come with some advantages and tradeoffs (for a contemporary overview, see Li, 2018).

2.3.2 Distributed Inference and Message Passing

In Chapter 3, we will design a multitask modeling framework where variational inference is the cornerstone of learning in multitask settings as well as inference of accurate task-specific models. Interestingly, using certain types of divergence measures, such as exclusive $\mathrm{KL}[p \parallel q]$, makes the optimization in Equation 2.3 nicely decompose into subproblems which can be solved efficiently in a distributed manner using Expectation Propagation (EP) and message passing algorithms. This will be particularly useful for scaling probabilistic methods to massively multitask settings such as FL, as we will show in Chapter 4. For brevity, we omit description of message passing approaches to inference and refer interested readers to an excellent tutorial by Minka (2005).

2.4 Learning to Learn (or Meta-learning)

In Chapters 3 to 5, some of our discussion will focus on meta-learning, where the goal is to learn a *learning algorithm* from a collection of training tasks that can produce accurate predictive models for unseen tasks given only few data points per task. For this to be possible, all tasks have to be related to each other in some sense. Typically, tasks are assumed to be sampled from a common underlying *task distribution* (Baxter, 2000). We will give a general definition of a task and show how a distribution of tasks can be represented using graphical models further in Chapter 3. For now, assuming that tasks are defined by small datasets of labeled points, $D_i := \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_i}$, and there are M training tasks, we provide a brief background on popular gradient-based and metric-based meta-learning algorithms.

2.4.1 Gradient-based Meta-learning

Model-agnostic Meta-Learning (MAML) (Finn et al., 2017) is one of the most popular gradient-based methods. The key idea of the method is simple: given a predictive model $f_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta}$, the method searches for an initialization $\boldsymbol{\theta}_0^\star$ of the parameters such that a few gradient updates on a few data points from a new task can yield a highly accurate model for that task. Mathematically, MAML can be formulated as a nested optimization with inner and outer loops:

$$\begin{aligned}
&\boldsymbol{\theta}_0^\star = \arg\min_{\boldsymbol{\theta}_0} \frac{1}{M} \sum_{i=1}^{n} \mathcal{L}(\boldsymbol{\theta}_0, Q_i), \quad \text{where} \quad \mathcal{L}(\boldsymbol{\theta}_0, D) = \frac{1}{|D|} \sum_{\mathbf{x}, y \in D} \ell(f_{\boldsymbol{\theta}_i(\boldsymbol{\theta}_0)}(\mathbf{x}), y), \\
&\boldsymbol{\theta}_i(\boldsymbol{\theta}_0) = \boldsymbol{\theta}_0 - \sum_{t=1}^{T} \alpha_t \nabla_{\boldsymbol{\theta}_t} \mathcal{L}(f_{\boldsymbol{\theta}_t}, S_i), \quad \text{and} \quad D_i = S_i \sqcup Q_i
\end{aligned} \tag{2.4}$$

In the outer loop, the algorithm searches for optimal initialization $\boldsymbol{\theta}_0^\star$ by optimizing a loss $\mathcal{L}(\boldsymbol{\theta}_0, Q_i)$ computed on a subset of the data Q_i for each task, averaged over all M training tasks. In the inner loop, for each training task i, it takes one or more gradient steps on the loss function computed on another data subset S_i to compute $\boldsymbol{\theta}_i$ as a function of the current initialization $\boldsymbol{\theta}_0$.

One of the advantages of MAML and many other gradient-based methods is their model-agnostic nature: they can be used with arbitrary models and loss functions, and as a results applied to supervised, self-supervised, semi-supervised, and reinforcement learning settings. We will also see in Chapters 3 and 4 that inner and outer loops in these methods can be generalized and reinterpreted from a probabilistic inference standpoint.

2.4.2 Metric-based Meta-learning

Another popular class of meta-learning methods is known as metric-based, which include Prototypical Networks (ProtoNets) (Snell et al., 2017) and Neural Processes (NPs) (Garnelo et al., 2018). The key idea of these methods is to learn a function that maps small datasets that define tasks to vectors in an embedding space. Then, these embedding vectors can be used to construct task-specific predictive models. In particular, if each task is a classification problem, ProtoNets compute embedding vectors for each class (called *prototypes*) and then define predictive models as a soft nearest neighbor classifiers. Similarly, NPs map task data to one or more embeddings and then uses these embeddings as extra inputs to the task-specific predictive model.

In practice, metric-based meta-learning methods are easier to train than gradient-based ones, they are often more data efficient (*i.e.*, can work well when trained on fewer tasks and fewer data points per task), but require committing to a particular model (*e.g.*, soft nearest neighbor). Note that metric-based methods can be also viewed from as algorithms that have "inner" and "outer" loops, where the inner loop computes task-specific embeddings and the outer loop optimizes parameters of the predictive model that are shared across all tasks.

2.5 Reinforcement Learning

Even though most of the subsequent chapters are focused on supervised learning problems, ideas presented in this thesis are not limited to it and, *e.g.*, can be used for multitask learning in Reinforcement Learning (RL) settings. In RL, instead of learning a predictive model from a static dataset, the goal is often to learn a *policy* (*i.e.*, a function that maps *observations* to *actions*) which an agent can follow to interact with an environment and to achieve certain goals. Such interactive setting is often formalized using Markov Decision Process (MDP) that defines a state space $\mathcal{X}$, an action space $\mathcal{A}$, an initial state distribution $p\left(\mathbf{x}_0\right)$, a probability $p\left(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{a}_t\right)$ of transitioning to the next state $\mathbf{x}_{t+1}$ given the current state $\mathbf{x}_t$ and the taken action $\mathbf{a}_t$, and a reward function $r(\cdot)$ which measures the quality of a *trajectory* (*i.e.*, a state-action sequence).

MDP essentially defines a learning task, in which we are not given any data but an opportunity to interact with some environment in order to collect data (in the form of trajectories observation, action, and reward sequences) and use that data to learn a policy that maximizes the rewards. Thus, similar to how meta-learning is defined for a distribution of supervised tasks (Baxter, 1998), *meta reinforcement learning* (or meta-RL) can be defined for a distribution of MDPs. In meta-RL, the goal is to learn a *reinforcement learning algorithm* which can be used to solve new MDPs (unseen at training time, sampled from the same distribution as training MDPs) having access to a limited number of interactions with the corresponding environments.

The interactive nature of the problem adds quite a bit of complexity to learning and can be approached from many different angles, giving rise to many different RL and meta-RL techniques: on-policy vs. off-policy, online vs. offline, and model-based vs. model-free methods. Analyzing and discussing all these methods is far beyond the scope of this thesis. In Chapter 5, we will focus on on-policy, model-free meta-RL and present a probabilistic take on the problem.

Chapter 3

Structure & Interpretation of Learning in Multitask Settings

3.1 Summary

In this chapter, we introduce Probabilistic Multitask Modeling (PMM) framework. We start by defining what a learning task is, what it means to solve a task (or multiple tasks), and introduce three types of multitask learning problems. Further, we define different notions of generalization that arise in each of the three settings. Then, we discuss how to represent multitask learning problems using factor graphs and how to use these representations to derive consistent objective functions and design algorithms for learning and inference. The chapter concludes with a brief survey the modern multitask learning techniques and a discussion of the key takeaways and the limitations of the presented framework.

3.2 What is Multitask Learning?

What is a *learning task*? To build a formal framework, we need a definition that is, on the one hand, general enough to encompass many different problems and, on the other hand, precise enough to be operational and allow us to distinguish between simple learning problems that correspond to a single task from those that are more complex. Similar to Mitchell et al. (2018), we say that "solving a learning task" means finding a function that: (1) maps some inputs to some outputs and (2) has a high utility according to some utility measure on the data sampled i.i.d. from a specified distribution. Formally, we can give the following definition.

Definition 3.1: Learning Task

Learning task is defined with a tuple, $T := (\mathcal{Z}, p(\mathcal{Z}), \mathcal{L})$, where $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$ is the space of inputs, $\mathcal{X}$, and outputs, $\mathcal{Y}$, $p(\mathcal{Z})$ is a data distribution over $\mathcal{Z}$, and $\mathcal{L} : (f : \mathcal{X} \mapsto \mathcal{Y}) \times \mathcal{Z} \mapsto \mathbb{R}$ is the loss functional that measures the utility of a task solution f.

Definition 3.1 neither specifies the family of functions nor the search algorithm that is used to find functions of high utility, both of which are our design choices. For example, consider supervised learning, where the goal is to learn a function that can predict some outputs given some inputs. If we select a family of functions parameterized by a vector $\boldsymbol{\theta} \in \Theta$ and search for functions of high utility by minimizing the average loss $\mathcal{L}$ on a dataset $D := \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$, this corresponds to solving the task using the well-known Empirical Risk Minimization (ERM) principle (Vapnik, 1999):

$$\hat{\boldsymbol{\theta}} \leftarrow \operatorname*{arg\,min}_{\boldsymbol{\theta} \in \Theta} \left\{ \mathcal{R}_{\text{emp}} \left[\boldsymbol{\theta}, D\right] := \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(f(\cdot; \boldsymbol{\theta}), \mathbf{x}_n, \mathbf{y}_n) \right\} \tag{3.1}$$

Another approach is to define a *probabilistic model* (also parametrized by some vector $\boldsymbol{\theta} \in \Theta$) that specifies the conditional probability distribution $p\left(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta}\right)$, and then search for high-utility[1] functions of the form $f(\mathbf{x}; \boldsymbol{\theta}) := \arg\max_{\mathbf{y} \in \mathcal{Y}} p\left(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta}\right)$ by maximizing the log likelihood of the probabilistic model given the data (*e.g.*, see Murphy, 2012):

$$\hat{\boldsymbol{\theta}} \leftarrow \operatorname*{arg\,min}_{\boldsymbol{\theta} \in \Theta} \left\{ \ell \left[\boldsymbol{\theta}, D_n\right] := -\sum_{i=1}^{n} \log p\left(\mathbf{y}_i \mid \mathbf{x}_i; \boldsymbol{\theta}\right) \right\} \tag{3.2}$$

Finally, we could take a fully-Bayesian approach (MacKay, 1992), define a prior distribution over the space of parameter vectors $p\left(\boldsymbol{\theta}\right)$, and select the solution of the task to be a function derived from the *predictive posterior distribution* $p\left(\mathbf{y} \mid \mathbf{x}, D\right)$ (see Section 2.2):

$$\hat{f}_{\text{Bayes}}(\mathbf{x}; \theta) := \operatorname*{arg\,max}_{\mathbf{y} \in \mathcal{Y}} \left\{ p\left(\mathbf{y} \mid \mathbf{x}, D_n\right) := \int_{\Theta} p\left(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \mid D\right) d\boldsymbol{\theta} \right\} \tag{3.3}$$

Note that in this latter case, we integrate over the parameter space instead of optimizing.[2]

All three methods described above are valid approaches to solving single-task learning problems, and all of them have advantages and disadvantages (*e.g.*, computational complexity, the number manually specified hyper-parameters, etc.) and sound theory behind them. In this chapter, we mainly focus on the probabilistic approaches and extend them to multiple tasks.

3.2.1 Different Types of Multitask Learning Problems

We distinguish between three broad classes of multitask learning problems:

① A finite/fixed collection of tasks somehow related to each other.

② An infinite/growing collection of tasks sampled i.i.d. from an underlying task distribution.

③ An infinite/growing collection of generally related tasks (sampled non-i.i.d.).

[1] In this example, we assume that functions that predict outputs from inputs more accurately have higher utility.

[2] How do we select a prior distribution $p\left(\boldsymbol{\theta}\right)$? This is a good question and there are many methods, *e.g.*, empirical Bayes (Robbins et al., 1956). But it is not important for our discussion in this thesis. Throughout, whenever the Bayesian approach comes up, we will simply assume that some prior is selected (*e.g.*, uniform, etc.).

Many problems in NLP, CV, and robotics known just as "multitask" correspond to our first category since often there is a fixed collection of tasks of interest being solved (Ruder, 2017; Zamir et al., 2018; Yu et al., 2020b). The second category corresponds to the typical formulation of *meta-learning* or *learning-to-learn* (Thrun and Pratt, 1998; Baxter, 1998; Finn et al., 2017), where after training on a collection of training tasks sampled i.i.d. from some task distribution, we would like to be able to (meta-)generalize to future tasks that come from the same distribution. The last category pertains to the so-called *continual*, *lifelong*, or *never-ending* learning (Ring, 1994; Mitchell et al., 2018), where new tasks keep arriving over time (generally, in a non-i.i.d. fashion), all tasks may somehow depend on each other, and we are interested maintaining the best possible performance either on all tasks or on the newly arrived tasks.

In this thesis, we mainly focus on the first two types of multitask learning problems. Note that combinations of these types are also possible—*e.g.*, in Chapter 5 we introduce the problem of *continuous adaptation* (a special case of continual learning), which is a combination of ② and ③. We discuss related work on in more detail in the chapters that describe our case studies.

3.2.2 Generalization in Multitask Settings

In machine learning, we say that a model generalizes if its performance on new (unseen) data is close to performance on the training data when both datasets come from the same underlying distribution. Now, we extend this classical notion of generalization to multitask settings.

First, we define two quantities: risk and its empirical estimate.

Definition 3.2: Risk

Given a task T, a dataset D of size N, a function f that solves T:

- The expected loss with respect to the data distribution is called (population) risk:

$$\mathcal{R}\left(f, T\right) := \mathbb{E}_{\mathbf{x},\mathbf{y}\sim T}\left[\mathcal{L}_T\left(f, \mathbf{x}, \mathbf{y}\right)\right] \tag{3.4}$$

- The average loss of f on the dataset D is called empirical risk:

$$\mathcal{R}_{\text{emp}}\left(f, T, D\right) := \frac{1}{|D|}\sum_{(\mathbf{x},\mathbf{y})\in D}\mathcal{L}_T\left(f, \mathbf{x}, \mathbf{y}\right) \tag{3.5}$$

  which is the empirical estimate of the risk.

Using these two quantities, we can define generalization error of a learning algorithm.

Definition 3.3: Generalization Error

Let $\mathcal{A}$ be a learning algorithm, which processes data and produces a function, $\mathcal{A}\left(D\right) \to f$. The generalization error of algorithm $\mathcal{A}$ on task T given dataset D is defined as follows:

$$\mathcal{G}\left(\mathcal{A}, T, D\right) := \mathcal{R}_{\text{emp}}\left(\mathcal{A}\left(D\right), T, D\right) - \mathcal{R}\left(\mathcal{A}\left(D\right), T\right) \tag{3.6}$$

Further, we say that a learning algorithm $\mathcal{A}$ *generalizes* on task T if its generalization error on that task vanishes as the amount of training data increases, *i.e.*, $\lim_{|D| \to \infty} \mathcal{G}(\mathcal{A}, T, D) = 0$. Note that generalization error is not computable exactly since it requires computation of the expectation with respect to the unknown data distribution. In practice, instead, we compute stochastic estimates of the population risk and generalization error based on a holdout dataset.

In all multitask settings introduced in Section 3.2.1, we have multiple datasets and multiple loss functions. This allows us to define and measure generalization in a few different ways.

Generalization to Fixed Collections of Tasks

Given M tasks $T_1, \ldots, T_M$, consider a learning algorithm $\mathcal{A}$ that processes data from all these tasks, $D_{1:M} = D_1 \cup \cdots \cup D_M$ and returns a functions $f_1, \ldots, f_M$ that solve the corresponding tasks. Denoting $\mathcal{A}_i(D_{1:M}) := f_i$, we can define multitask generalization of the algorithm $\mathcal{A}$ with respect each individual task T_i in the collection as follows.

Definition 3.4: Multitask Generalization Error

$$\mathcal{G}_{\mathrm{MT}}(\mathcal{A}, T_i, D_{1:M}) := \mathcal{R}_{\mathrm{emp}}(\mathcal{A}_i(D_{1:M}), T_i, D_i) - \mathcal{R}(\mathcal{A}_i(D_{1:M}), T_i) \tag{3.7}$$

Moreover, at training time, we may be given data for only a subset of tasks, while interested in measuring generalizing on one of the unseen tasks. This is known as zero-shot generalization to unseen tasks. Denoting the indices of tasks seen at training as $\mathcal{I}_s \subset [M]$, we can formally define zero-shot generalization of algorithm $\mathcal{A}$ to an unseen task $T_i (i \notin \mathcal{I}_s)$ as follows.

Definition 3.5: Zero-shot Generalization Error

$$\mathcal{G}_{\mathrm{ZS}}(\mathcal{A}, T_i, D_{\mathcal{I}_s}) := \mathcal{R}_{\mathrm{emp}}(\mathcal{A}_i(D_{\mathcal{I}_s}), T_i, D_i) - \mathcal{R}(\mathcal{A}_i(D_{\mathcal{I}_s}), T_i) \tag{3.8}$$

We note a couple of important points:

1. Zero-shot generalization is not always feasible, *i.e.*, increasing the amount of data in the training tasks $D_{\mathcal{I}_s}$ may not improve (or even affect) performance an unseen task. For zero-shot generalization to be possible, not only tasks must be strongly coupled, but we may also need a loss function that enables such generalization, as we show in Chapter 6.
2. Learning from multiple tasks may improve or worsen the generalization error when compared to single-task learning. The worsening can happen due to *task interference* or *negative transfer* (McCloskey and Cohen, 1989; Schaul et al., 2019).

Both Definitions 3.4 and 3.5 also apply to continual learning problems as well where the collection of tasks grows over time. In that case, we can look at generalization error (or its zero-shot version) on a particular task T_i as a function of time. The temporal ordering of tasks also allows us to consider *forward transfer* (*i.e.*, zero-shot generalization on task T_{t+1} given $T_1, \ldots, T_t$) and *backward transfer* (*i.e.*, generalization on one of the $T_1, \ldots, T_t$ tasks after adding T_{t+1}) as proposed by Lopez-Paz and Ranzato (2017).

Generalization to Distributions of Tasks

In the distributional setting, all tasks are related to each other not arbitrarily but through the underlying distribution. This allows us to define meta-generalization (Baxter, 2000), or the ability of learning algorithms to generalize to new tasks from the same distribution.

To define it formally, we first need the notion of transfer risk and its empirical estimate.

Definition 3.6: Transfer Risk

Let $\mathbb{P}$ be a distribution of tasks, $D_1, \ldots, D_M$ be datasets that correspond to tasks $T_1, \ldots, T_M$ sampled from $\mathbb{P}$, and $\mathcal{A}$ is a learning algorithm. Then:

- The expected risk of the functions produced by $\mathcal{A}$ on tasks from $\mathbb{P}$ is called (population) transfer risk:
$$\mathfrak{R}\left(\mathcal{A}, \mathbb{P}\right) := \mathbb{E}_{T \sim \mathbb{P}}\left[\mathbb{E}_{D \sim T}\left[\mathcal{R}\left(\mathcal{A}(D), T\right)\right]\right] \tag{3.9}$$
- The estimate of the transfer risk on $D_1, \ldots, D_M$ is called empirical transfer risk:
$$\mathfrak{R}_{\text{emp}}\left(\mathcal{A}, T_{1:M}, D_{1:M}\right) := \frac{1}{M}\sum_{i=1}^{M} \mathcal{R}_{\text{emp}}\left(\mathcal{A}(D_i), T_i, D_i\right) \tag{3.10}$$

Note that transfer risk is defined for an algorithm that produces functions. Having transfer risk defined, meta-generalization becomes a straightforward extension of classical generalization:

Definition 3.7: Meta Generalization Error

Let $\mathbb{A}$ be a meta-learning algorithm, which processes data from multiple tasks and produces a learning algorithm, $\mathbb{A}\left(T_{1:M}, D_{1:M}\right) \to \mathcal{A}$. The meta-generalization error of $\mathbb{A}$ on task distribution $\mathbb{P}$ given datasets $D_1, \ldots, D_M$ from tasks $T_1, \ldots, T_M$ is defined as follows:

$$\mathfrak{G}\left(\mathbb{A}, \mathbb{P}\right) := \mathfrak{R}_{\text{emp}}\left(\mathbb{A}\left(T_{1:M}, D_{1:M}\right), T_{1:M} D_{1:M}\right) - \mathfrak{R}\left(\mathbb{A}\left(T_{1:M}, D_{1:M}\right), \mathbb{P}\right) \tag{3.11}$$

Definitions 3.6 and 3.7 may seem cumbersome, but all they do is simply define another level of learning on top of learning from data (hence, “meta” in the name). In other words, given a set of training tasks $T_1, \ldots, T_M$ and the corresponding training datasets $D_1, \ldots, D_M$, we first run some meta-learning algorithm $\mathbb{A}$ to get a learning algorithm $\mathcal{A}$. The obtained learning algorithm is expected to be more suitable for learning from tasks that come from the same distribution as $T_1, \ldots, T_M$. In our probabilistic framework, things will become even more intuitive: both the meta-algorithm and the algorithms it produces will correspond to inference of global and local parameters in a multitask probabilistic model, respectively.

► To summarize, we have defined three types of learning problems in multitask settings. We also introduced different notions of generalization that can be used to measure the quality of different learning algorithms in each of these settings. All our definitions so far have been model- and algorithm-agnostic. Next, we present a probabilistic framework that will allow to design learning algorithms for a variety multitask problems systematically.

3.3 Probabilistic Multitask Modeling Framework

Our framework (PMM) is based on Probabilistic Graphical Models (PGM) (Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Koller and Friedman, 2009). We emphasize that PMM does not contribute anything new to PGM itself, which is a well-established area at the intersection of probability, statistics, and computer science. Instead, we present a simple way to represent different multitask learning problems with PGMs, use such representations to encode our knowledge about the relationships between the tasks, and derive loss functions and learning algorithms for solving such problems more systematically.

3.3.1 Representing Multiple Tasks with Factor Graphs

We propose to represent tasks with factor graphs (Kschischang et al., 2001; Frey, 2002). Just as an example, consider a collection of three *independent* tasks, T_1, T_2, T_3. Representing each individual task in a collection with a factor graph is straightforward as give on Figure 3.1 below.

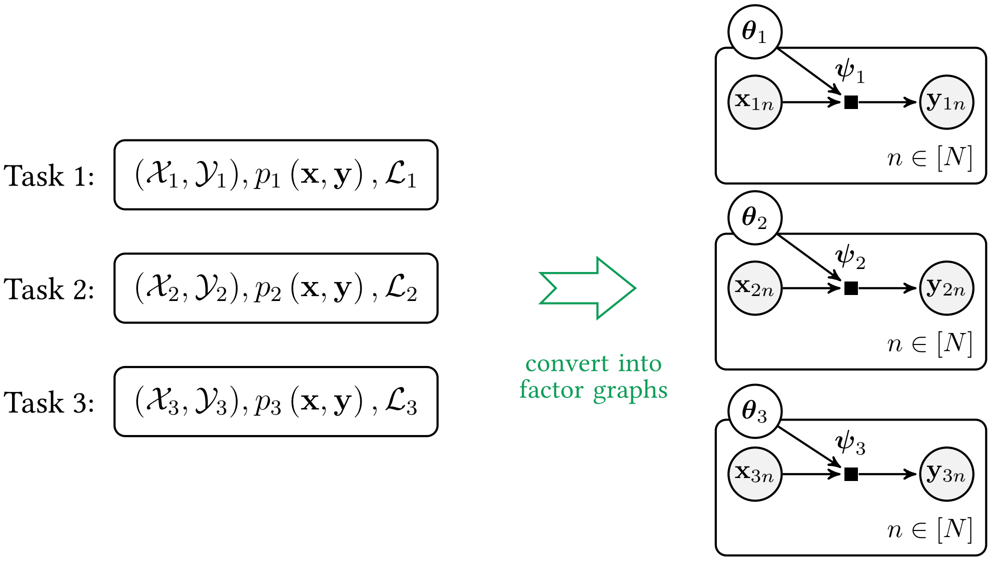


Figure 3.1: Representation of individual tasks with factor graphs.

Each factor graph here represents a conditional probability distribution:

$$p_i(\mathbf{y}_{in} \mid \mathbf{x}_{in}, \boldsymbol{\theta}_i) \propto \exp\{\psi_i(\mathbf{x}_{in}, \mathbf{y}_{in}, \boldsymbol{\theta}_i)\}, \text{ for } i = 1, 2, 3 \tag{3.12}$$

Note that when tasks are independent, none of the variables is shared between the tasks. To couple the tasks, the main mechanism that we will be used in our probabilistic multitask modeling framework is through sharing random variables between the factor graphs. Again, for the sake of this example, assume that all three tasks share their inputs $\mathbf{x}$ and a subset of parameters $\boldsymbol{\theta}$. Then, we can represent this multitask learning problem with a join factor graph (Figure 3.2).

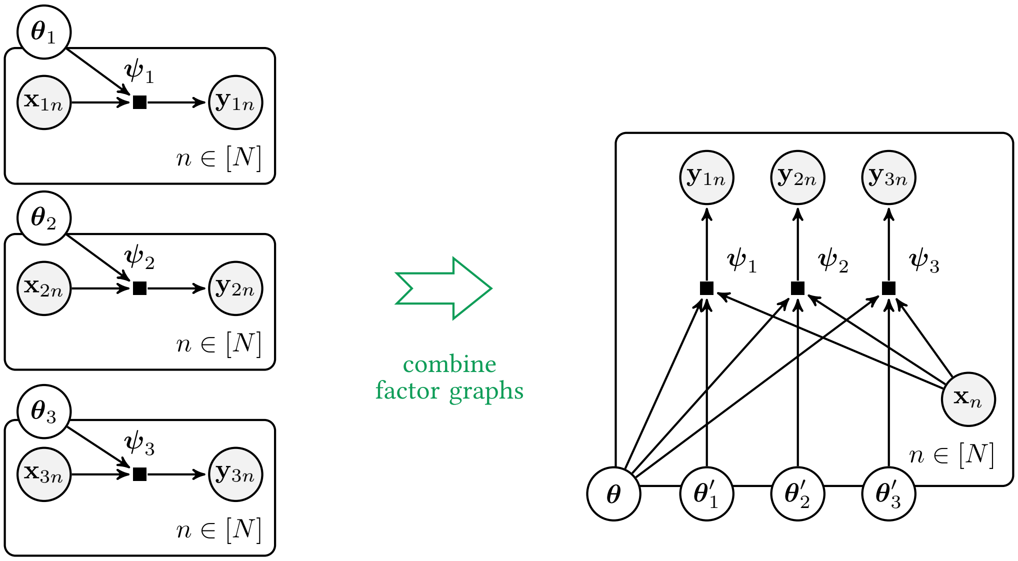


Figure 3.2: Representation of tasks with a joint factor graph.

Note that the shared inputs got collapsed into $\mathbf{x}$ while each $\boldsymbol{\theta}_i$ got split into the shared $\boldsymbol{\theta}$ and individual $\boldsymbol{\theta}'_i$ parameters. Our resulting factor graph essentially corresponds to a multitask learning problem that can be solved with a multi-output (or multi-head) neural net, originally described by Caruana (1998). The conditional probability distribution of all outputs $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$ given the input $\mathbf{x}$ and all parameters takes the following form:

$$p\left(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3 \mid \mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\theta}'_1, \boldsymbol{\theta}'_2, \boldsymbol{\theta}'_3\right) \propto \exp\left\{\sum_{i=1}^{3} \psi_i\left(\mathbf{x}, \mathbf{y}_i, \boldsymbol{\theta}, \boldsymbol{\theta}'_i\right)\right\} \propto \prod_{i=1}^{3} p\left(\mathbf{y}_i \mid \mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\theta}'_i\right) \tag{3.13}$$

Note that the distribution factorizes into a product of three conditionals that correspond to each task since each factor ψ_i depends only on a single output. As a result, the log conditional likelihood of this probabilistic multitask model decomposes into a sum of individual terms:

$$\log p\left(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3 \mid \mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\theta}'_1, \boldsymbol{\theta}'_2, \boldsymbol{\theta}'_3\right) = \sum_{i=1}^{3} \log p\left(\mathbf{y}_i \mid \mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\theta}'_i\right) \tag{3.14}$$

In other words, the log conditional probability of the outputs given the inputs in the resulting model recovers the classical multitask learning objective function: a linear combination of individual task losses,[3] which correspond to log conditional likelihoods of individual outputs. Learning parameters in this setting can be done either though likelihood maximization or posterior inference.

While this example is trivial, the problem becomes much more interesting when outputs of some tasks become inputs to other tasks and/or not all inputs and outputs are observed. In

[3] In practice, one would typically use a *weighted* linear combination of individual task losses to emphasize the “importance” of each task. In our probabilistic framework, the weights of individual likelihoods are proportional to the number of training examples n_i that have the corresponding output $\mathbf{y}_i$. In other words, by default, the task “importance” is proportional to the number of times the task has been seen at training time.

such cases, we will not be able to factorize the joint probability of the observed outputs and the corresponding loss function will not be as straightforward to optimize or even compute. We explore such non-trivial situations in the context of multilingual machine translation in Chapter 6 and in the context of interpretability of predictive models in Chapter 7.

Representing distributions of tasks. Graphical models allow to easily represent task distributions using the plate notation and additional latent variables. Figure 3.3 provides an example of such representation. In this example, parameters $\boldsymbol{\theta}$ are the only variables shared across the tasks and $\boldsymbol{\phi}_i$'s represent task-specific latent variables shared across data points within each task. The plates simply denote replication of the corresponding variables (see Section 2.2) that are sampled i.i.d. from the underlying distribution. Assuming that there are M training tasks, the log likelihood of the training data $D_{1:M} = D_1 \cup D_2 \cup \cdots \cup D_M$ takes the following form:

$$\log p\left(D \mid \boldsymbol{\theta}\right) = \sum_{i=1}^{M} \log \int p\left(D_i \mid \boldsymbol{\phi}_i, \boldsymbol{\theta}\right) p\left(\boldsymbol{\phi}_i\right) d\boldsymbol{\phi}_i \qquad (3.15)$$

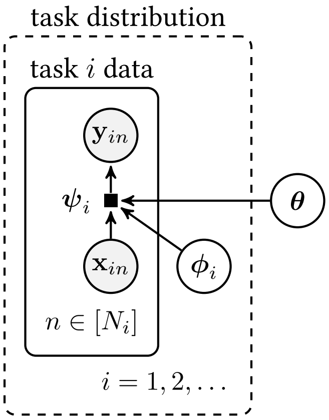


Figure 3.3: Representation of a distribution of i.i.d. tasks.

where $p\left(D_i \mid \boldsymbol{\phi}_i, \boldsymbol{\theta}\right) = \prod_{n=1}^{N_i} p\left(\mathbf{y}_{in} \mid \mathbf{x}_{in}, \boldsymbol{\phi}_i, \boldsymbol{\theta}\right)$. The tasks are tied together through the prior over the local latent variables, $p\left(\boldsymbol{\phi}_i\right)$, induced by the task distribution (*i.e.*, there are no direct dependencies between the task-specific variables). Due to marginalization of these local variables, computing or maximizing the likelihood directly is non-trivial in this model and requires approximation. Further, obtaining a function that solves a new task T_j, *i.e.*, $f_j(\mathbf{x}; \boldsymbol{\theta}) = \arg\max_{\mathbf{y} \in \mathcal{Y}_j} p\left(\mathbf{y} \mid \mathbf{x}, D_j, \boldsymbol{\theta}\right)$, requires inference of the posterior predictive:

$$p\left(\mathbf{y} \mid \mathbf{x}, D_j, \boldsymbol{\theta}\right) = \int p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\phi}, \boldsymbol{\theta}\right) p\left(\boldsymbol{\phi} \mid D_j, \boldsymbol{\theta}\right) d\boldsymbol{\phi} \qquad (3.16)$$

which again involves generally intractable integral and requires approximation.

From this probabilistic perspective, we can now reinterpret meta-learning algorithms $\mathbb{A}$ as procedures that approximately infer the global parameters $\boldsymbol{\theta}$; the learning algorithms $\mathcal{A}$ (produced by $\mathbb{A}$) are approximately inferring the task-specific predictive posterior $p\left(\mathbf{y} \mid \mathbf{x}, D_j, \boldsymbol{\theta}\right)$. In that sense, different meta-learning methods, such as MAML (Finn et al., 2017) or ProtoNets (Snell et al., 2017), can all be viewed as different ways to approximate the marginalization of the local latent variables. The connection between Bayesian inference and learning-to-learn has been noticed and used in multiple previous works (Lawrence and Platt, 2004; Al-Shedivat et al., 2018a; Yoon et al., 2018; Finn et al., 2018; Grant et al., 2018).

We will further explore learning from distributions of tasks in the context of federated learning in Chapter 4 and reinforcement learning in Chapter 5.

Representing growing collections of inter-dependent tasks. Finally, in continual learning (Ring, 1994; Mitchell et al., 2018), we have to solve an unbounded collection of tasks that grows over time, where the newly added tasks somehow depend on (a subset of) previously seen tasks (*i.e.*, tasks are sampled non-i.i.d.). This can be modeled as a succession of multitask

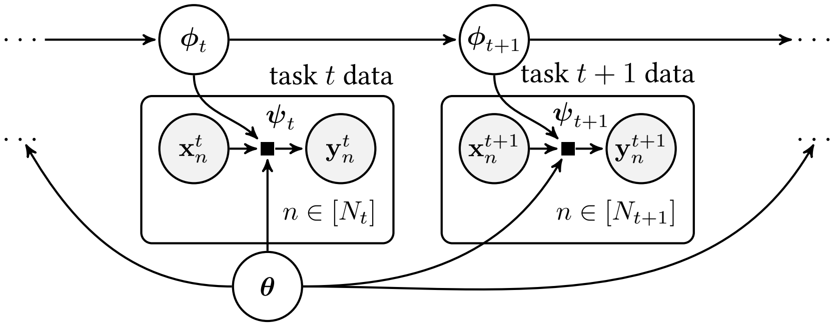


Figure 3.4: Representation of a chain of dependent tasks.

learning problems: at each instance of time t, we receive a new task T_t, add it to the collection, and solve a new multitask learning problem with $T_1, \ldots, T_t$. Depending on the application, we might be interested in the solution that attains maximum utility on the most recent task T_t or on all tasks up until T_t. In either case, at each time step, the problem reduces to multitask learning with a fixed collection of tasks, but with a caveat: at each time step, we are not solving the problem from scratch, but do have access to the previous solution(s), and can/must take advantage of that for efficiency purposes.

As an example, Figure 3.4 represents a chain of sequentially dependent tasks with a factor graph that assumes markovian structure of dependencies between the task-specific local latent variables (*i.e.*, $p\left(\boldsymbol{\phi}_{t+1} \mid \boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_t\right) = p\left(\boldsymbol{\phi}_{t+1} \mid \boldsymbol{\phi}_t\right)$ for all t). Assuming that we have previously obtained $\hat{\boldsymbol{\theta}}_t$ that solves tasks $T_1, \ldots, T_t$ (as a fixed collection of dependent tasks), given a new task T_{t+1}, our goal is to compute $\hat{\boldsymbol{\theta}}_{t+1} \leftarrow \mathrm{Update}\left(\hat{\boldsymbol{\theta}}_t, D_{t+1}\right)$. If we were to take the fully Bayesian approach, instead of working with point-estimates of $\boldsymbol{\theta}$, we would be maintaining and updating (approximately!) the posterior distribution $p\left(\boldsymbol{\theta} \mid D_1, \ldots, D_t\right) \to p\left(\boldsymbol{\theta} \mid D_1, \ldots, D_t, D_{t+1}\right)$. We will dive into the details of this approach in the context of RL under nonstationarity in Chapter 5.

▶ To summarize, probabilistic graphical models (and factor graphs in particular) provide an elegant way to specify probability distributions that satisfy the assumptions of different multitask settings. To construct a joint probabilistic model for a multitask learning problem, we can follow three simple steps summarized in Algorithm 3.1. Note that the first step of the algorithm (*i.e.*, decomposition of the learning problem into multiple tasks) is problem-specific and based on our design choices.

Algorithm 3.1 Build PMM representation.

```
Decompose the given learning problem
  into multiple tasks.
Represent each individual task with its
  own factor graph.
Combine individual factor graphs into
  the joint by merging shared variables
  and adding necessary dependencies.
```

Constructing factor-graph-based representations of a complex learning problem is only the first step of our framework. The next step is to use the representations for algorithm design.

3.3.2 Solving Multiple Tasks: A Probabilistic Approach

Given the probabilistic representations, how do we solve all these multitask learning problems? We propose to use variational Bayes (Jordan et al., 1999), *i.e.*, infer an approximate posterior distribution over all the latent variables via divergence minimization:

$$\min_{q \in \mathcal{Q}} \mathbb{D}[p(\mathbf{l} \mid \mathbf{e}) \parallel q(\mathbf{l})] \tag{3.17}$$

where $\mathbf{e}$ denotes all observable variables (stands for "**e**vidence"), $\mathbf{l}$ denotes all unobserved variables and parameters (stands for "**l**atents"), $\mathcal{Q}$ is a variational family of distributions, and $\mathbb{D}$ is a divergence measure (*e.g.*, KL divergence, but can be some other divergence).

Why variational posterior inference? From a Bayesian perspective, once a probabilistic model is defined, posterior is really all we need—it accurately summarizes all the information contained in the data about the parameters and latent variables. Exact posterior inference is almost never tractable, and therefore we have to use approximate inference.

To better understand Equation 3.17, we show how it is related to likelihood maximization and then illustrate advantages of our probabilistic approach with a couple concrete examples.

Converting Posterior Inference into Maximum Likelihood

If we set $\mathbb{D}$ to be the exclusive KL divergence (*i.e.*, $\mathrm{KL}[q \parallel p]$), solving Equation 3.17 becomes equivalent to maximizing the variational free energy:

$$\min_{q \in \mathcal{Q}} \left\{ \mathcal{F}(q) := -\mathbb{E}_{q(\mathbf{l})} \left[\log p(\mathbf{e} \mid \mathbf{l}) + \log p(\mathbf{l})\right] - \mathbb{H}(q) \right\} \tag{3.18}$$

where $\mathbb{H}$ denotes Shannon entropy. Note that we may often be interested in obtaining an approximate posterior distribution for only a subset of latent variables (typically, some hidden states in the model) and a point estimate for the rest (typically, model parameters). Denoting these subsets $\mathbf{h}$ and $\boldsymbol{\theta}$, respectively, such that $\mathbf{l} = (\mathbf{h}, \boldsymbol{\theta})$, we set $q(\mathbf{h}, \boldsymbol{\theta}) := \tilde{q}(\mathbf{h})\, \delta(\boldsymbol{\theta} - \boldsymbol{\theta}')$, where δ is the Dirac delta function, which reduces Equation 3.18 to the following:

$$\min_{\boldsymbol{\theta}, \tilde{q} \in \mathcal{Q}} -\mathbb{E}_{\tilde{q}(\mathbf{h})} \left[\log p(\mathbf{e}, \mathbf{h} \mid \boldsymbol{\theta})\right] - \log p(\boldsymbol{\theta}) - \mathbb{H}(\tilde{q}) \tag{3.19}$$

Equation 3.19 is a variational lower bound on the log posterior $\log p(\boldsymbol{\theta} \mid \mathbf{e})$. If we assume a uniform prior on $\boldsymbol{\theta}$ (*i.e.*, $\log p(\boldsymbol{\theta}) \propto 1$), it becomes a bound on the log likelihood $\log p(\mathbf{e} \mid \boldsymbol{\theta})$. Finally, if we also do not have any other latent variables apart from model parameters (as in one of our previous examples in Figure 3.2), Equation 3.19 reduces to likelihood maximization.

In other words, likelihood maximization is a special case of variational posterior inference. But if in the end, all we have to do is optimize a likelihood that factorizes into independent terms for each task (again, as in Figure 3.2), why go through all this trouble?

Solving a Collection of Tasks with Shared Variables

It turns out that in many interesting multitask settings the joint likelihood does not factorize. Interestingly, substituting the joint likelihood with a weighted combination of independent likelihoods for each task—which is known as the *composite likelihood* (Besag, 1975; Lindsay, 1988) and typically used in practice—might be a suboptimal decision. Here is why.

Consider a multitask learning problem with three variables $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$, where tasks correspond to mappings from $\mathbf{x}_i \mapsto \mathbf{x}_j$. In computer vision, these variables may correspond to different representations of a scene (*e.g.*, pixels, surface normals, depth, shading, texture, occlusion, etc.) (Zamir et al., 2018). In natural language, these could be representations of the meaning of a sentence in different languages (Al-Shedivat and Parikh, 2019). More broadly, any problem of translation between different data modalities can be represented this way. Figure 3.5 illustrates the factor graph representation of this problem.

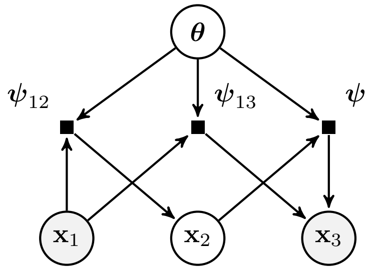


Figure 3.5: Learning to translate between data modalities.

If all variables $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ were always observed, we would have been able to optimize the log likelihood, which decomposes as follows:

$$\log p\left(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \mid \boldsymbol{\theta}\right) \propto \underbrace{\log p\left(\mathbf{x}_2 \mid \mathbf{x}_1, \boldsymbol{\theta}\right)}_{\psi_{12}(\mathbf{x}_1,\mathbf{x}_2,\boldsymbol{\theta})} + \underbrace{\log p\left(\mathbf{x}_3 \mid \mathbf{x}_1, \boldsymbol{\theta}\right)}_{\psi_{13}(\mathbf{x}_1,\mathbf{x}_3,\boldsymbol{\theta})} + \underbrace{\log p\left(\mathbf{x}_3 \mid \mathbf{x}_2, \boldsymbol{\theta}\right)}_{\psi_{23}(\mathbf{x}_2,\mathbf{x}_3,\boldsymbol{\theta})} \tag{3.20}$$

However, in practice, obtaining multi-parallel data that always contains all data modalities is expensive, and we often end up having data samples of only a pair of variables $(\mathbf{x}_i, \mathbf{x}_j)$ observed. In Figure 3.5, modality $\mathbf{x}_2$ is unobserved, which results in the following log likelihood:

$$\begin{aligned} \log p\left(\mathbf{x}_1, \mathbf{x}_3 \mid \boldsymbol{\theta}\right) &\propto \log \int p\left(\mathbf{x}_2 \mid \mathbf{x}_1, \boldsymbol{\theta}\right) p\left(\mathbf{x}_3 \mid \mathbf{x}_1, \boldsymbol{\theta}\right) p\left(\mathbf{x}_3 \mid \mathbf{x}_2, \boldsymbol{\theta}\right) d\mathbf{x}_2 && (3.21) \\ &= \log p\left(\mathbf{x}_3 \mid \mathbf{x}_1, \boldsymbol{\theta}\right) + \boxed{\log \int p\left(\mathbf{x}_2 \mid \mathbf{x}_1, \boldsymbol{\theta}\right) p\left(\mathbf{x}_3 \mid \mathbf{x}_2, \boldsymbol{\theta}\right) d\mathbf{x}_2} && (3.22) \end{aligned}$$

Note the additional term besides $\log p\left(\mathbf{x}_3 \mid \mathbf{x}_1, \boldsymbol{\theta}\right)$ makes the objective sensitive to the quality of a chain of predictions, $\mathbf{x}_1 \mapsto \mathbf{x}_2 \mapsto \mathbf{x}_3$, which enforces consistency of the model.

Such consistency-enforcing losses have been first introduced in machine translation by Al-Shedivat and Parikh (2019), which we discuss in more detail in Chapter 6. Later, similar consistency losses were proposed for computer vision by Zamir et al. (2020) in an ad-hoc manner (*i.e.*, the additional terms were not derived from a probabilistic model, but played the same role).

These works suggest that optimizing the joint log likelihood results in a somehow more "consistent" model. But what does consistency mean and why do we need consistent models? One of the major advantages is that the resulting models can provably generalize zero-shot, which is not the case for optimizing composite likelihoods, as stated in the following theorem.

Theorem 3.1: Zero-shot Generalization to Unseen Translation Tasks (Informal)

Given a total of K data modalities $\mathbf{x}_1, \ldots, \mathbf{x}_K$ and training data for only $M < K(K-1)$ tasks T_{ij} that correspond to $\mathbf{x}_i \mapsto \mathbf{x}_j$, let $\mathcal{A}_{\text{joint}}$ be an algorithm that maximizes the joint likelihood and $\mathcal{A}_{\text{comp}}$ be an algorithm that maximizes the composite likelihood. Then:

- $\mathcal{A}_{\text{joint}}$ generalizes zero-shot on the unseen tasks T_{ik} and T_{kj} for all intermediate k.
- $\mathcal{A}_{\text{comp}}$ does not exhibit zero-shot generalization on the unseen tasks.

We prove a version of Theorem 3.1 in Chapter 6. Note a couple of important points:

- First, surprisingly, the result is fully agnostic to way the model is parametrized. In deep learning, it is a common belief that generalization in multitask settings requires parameter sharing between functions that solve different tasks. This turns out not to be true for *zero-shot* generalization. Not only it additionally requires a loss that ensures consistency (*i.e.*, parameter sharing alone is not sufficient), but even if parameters are not shared at all, maximizing the joint likelihood will still provably lead to zero-shot generalization.[4]
- While our exposition here focused the problem of translation between different data modalities (which is already a very broad class of problems), generally, the same argument applies to any setting that has hidden variables used as inputs or outputs of multiple tasks.

Finally, note that Theorem 3.1 requires optimization of the joint log likelihood which is intractable due to marginalization of the latent variables. In Chapter 6, we design a variational lower bound, called *agreement-based loss*, which can be computed and optimized efficiently. Here, we move on to our next example and show how to use PMM to solve distributions of tasks.

Solving Distributions of Tasks

In Section 3.3.1, we have shown how to represent distributions of tasks with a graphical model (Figure 3.3) and provided an expression for the joint likelihood (Equation 3.15), which we noted requires further approximation. Again, let $T_1, \dots, T_M$ be our training tasks, with the corresponding datasets $D_1, \dots, D_M$. Assuming a uniform prior on $\boldsymbol{\theta}$, the variational free energy (or the lower bound on the join log likelihood) takes the following form:

$$\mathcal{F}\left(\boldsymbol{\theta}, q_1, \dots, q_M\right) = \sum_{i=1}^{M} \left\{ \mathcal{F}_i\left(\boldsymbol{\theta}, q_i\right) := -\sum_{n=1}^{N_i} \mathbb{E}_{q_i(\boldsymbol{\phi}_i)}\left[\log p\left(\mathbf{y}_{in}, \boldsymbol{\phi}_i \mid \mathbf{x}_{in}, \boldsymbol{\theta}\right)\right] + \mathbb{H}\left(q_i\right) \right\} \tag{3.23}$$

where each $q_i\left(\boldsymbol{\phi}_i\right)$ is a variational distributions of the local latent variable $\boldsymbol{\phi}_i$ and is selected to approximate the local posterior $p\left(\boldsymbol{\phi}_i \mid D_i, \boldsymbol{\theta}\right)$.

Adaptation (or personalization) via local posterior inference. For now, assuming that $\boldsymbol{\theta}$ is already learned, given a new task T_j with dataset D_j, we can solve it in two steps:

1. Get a variational approximation $q_j\left(\boldsymbol{\phi}_j\right)$ of $p\left(\boldsymbol{\phi}_j \mid D_j, \boldsymbol{\theta}\right)$.
2. Compute the approximate predictive distribution $p\left(\mathbf{y} \mid \mathbf{x}, T_j\right) := \mathbb{E}_{q_j\left(\boldsymbol{\phi}_j\right)}\left[p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\phi}_j, \boldsymbol{\theta}\right)\right]$, which provides us a function that solves T_j: $f_j(\mathbf{x}) := \arg\max_{\mathbf{y} \in \mathcal{Y}_j} p\left(\mathbf{y} \mid \mathbf{x}, T_j\right)$.

The adaptation (or personalization) procedure of any learning-to-learn method can be viewed as a way to approximate these two steps. For example, MAML (Finn et al., 2017) substitutes $q_j\left(\boldsymbol{\phi}_j\right)$ with a point estimate $\hat{\boldsymbol{\phi}}_j$ obtained by running (stochastic) gradient descent on the local log likelihood $\log p\left(D_j \mid \boldsymbol{\theta}\right)$. Similarly, ProtoNets (Snell et al., 2017) and NPs (Garnelo et al., 2018) can be viewed as *amortized inference* methods that compute point estimates of the local parameters using a function applied to the local dataset, $g_{\mathbf{w}}\left(D_j\right) \to \hat{\boldsymbol{\phi}}_j$, shared across tasks.

[4] The lack of parameter sharing might worsen sample complexity, *i.e.*, with the increase of data in supervised tasks, the zero-shot error might decrease at a slower rate than for models that share parameters. Our proof does not provide a convergence rate, however, and quantifying such an effect would require further analysis.

Meta-learning via variational free energy minimization. Learning the global parameters $\boldsymbol{\theta}$ can be done by optimizing the variational free energy (Equation 3.23), using local posterior inference as a sub-routine. The approach is summarized in the following three steps:

1. Select a subset $B \subset [M]$ of training tasks.
2. For each task in B, compute its variational approximation $q_i\left(\boldsymbol{\phi}_i\right)$.
3. Compute $\mathcal{F}$ for the selected tasks and optimize it with respect to the global parameters.

Note that variational approximations of $q_i\left(\boldsymbol{\phi}_i\right)$ used in practice by MAML, ProtoNets, NPs, and other methods are typically different point estimates $\hat{\boldsymbol{\phi}}_i$, and hence the variational free energy simply reduces to $-\sum_{i=1}^{M}\sum_{n=1}^{N_i}\log p\left(\mathbf{y}_{in} \mid \mathbf{x}_{in}, \hat{\boldsymbol{\phi}}_i, \boldsymbol{\theta}\right)$.

► To sum up, starting from a generic graphical model that represents a distribution of tasks, optimizing the variational free energy of such probabilistic model allowed us to recover many of the popular meta-learning algorithms. In our probabilistic framework, "personalization" (or "model adaptation") corresponds to approximate local posterior inference done in the inner loop, while "meta-learning" corresponds to the variational free energy minimization with respect to the global parameters in the outer loop.

Finally, there is one interesting (and subtle!) question that remains: Computation of the approximate local posterior $q_i\left(\boldsymbol{\phi}_i\right)$ in the inner loop and the free energy $\mathcal{F}_i\left(\boldsymbol{\theta}, q_i\right)$ in the outer loop for task T_i both need some data from that task. So, given a dataset D_i, should it be re-used twice (in the inner and outer loops) or split into two disjoint subsets each of which is used only once (per iteration of the outer loop)? It turns out that both approaches are valid, but the meta-generalization error as a function of the number of tasks and the number of data points per task behaves differently for these methods, as given in the following theorem.

Theorem 3.2: Meta-generalization Bounds

Let the variational free energy loss $\mathcal{F}$ be L-Lipschitz and γ-smooth. Given M training tasks with at least m data points per task, if the meta-learning algorithm $\mathbb{A}$ is a stochastic gradient method that optimizes $\mathcal{F}$ by taking K steps with non-increasing step sizes $\alpha_k \leq c/k$, with probability at least $1-\delta$, we have the following:

1. If we use disjoint subsets data for local inference in the inner loop and computation of the loss in the outer loop, then the following meta-generalization bound holds:

$$\mathfrak{G}\left(\mathbb{A}, \mathbb{P}\right) \leq O\left(L^2 K \sqrt{\frac{\ln(1/\delta)}{M}}\right) \tag{3.24}$$

2. If we re-use the same data and the inner loop inference is also done by running stochastic gradient method on L'-Lipschitz and γ'-smooth loss by taking K' steps with step sizes $\alpha_{k'} \leq c'/k'$, the corresponding bound is as follows:

$$\mathfrak{G}\left(\mathbb{A}, \mathbb{P}\right) \leq O\left(L^2 K \sqrt{\frac{\ln(1/\delta)}{M}} + L'^2 K' \frac{1}{m}\right) \tag{3.25}$$

We prove the bounds in Appendix A.1. Interestingly, splitting the data of each task in to two disjoint subsets—known as *support* (used in the inner loop) and *query* (used in the outer loop) originally proposed by Vinyals et al. (2016)—leads to a bound that does not depend on the amount of training data points per task additively, and hence enables learning in few-shot settings. The theorem has multiple other practical implications about the data efficiency of meta-learning, which have been confirmed empirically (Al-Shedivat et al., 2021b).

Algorithms Beyond Likelihood Maximization

In the previous two examples, we have shown how to solve fixed collections and distributions of tasks by maximizing (a lower bound on) the joint likelihood using stochastic gradient methods. But recall that we started with the general posterior inference formulation (Equation 3.17), and maximum likelihood is only a special case. What other alternatives do we have?

There is another broad class of so-called message-passing algorithms that can be used for solving variational inference, which include *expectation propagation* (EP) (Minka, 2001) and *variational message-passing* (VMP) (Winn and Bishop, 2005). The key idea of these algorithms is to solve Equation 3.17 using a fixed point iteration method that refines a variational approximation of the posterior over the latents represented as a product of sub-posteriors:

$$p\left(\mathbf{l} \mid \mathbf{o}\right) \propto p_0\left(\mathbf{l}\right) \prod_{i=1}^{M} p_i\left(\mathbf{l} \mid \mathbf{o}\right) \quad \text{approximated with} \quad q\left(\mathbf{l}\right) \propto q_0\left(\mathbf{l}\right) \prod_{i=1}^{M} q_i\left(\mathbf{l}\right) \tag{3.26}$$

At each iteration, the algorithm selects a subset of sub-posteriors $\{p_i\left(\mathbf{l} \mid \mathbf{e}\right)\}_{i \in B}$, solves the inference problem for each them, then aggregates them, and the process repeats. The approach is called message-passing because the underlying computation can be represented in terms of passing messages between the nodes in the factor graph. The generalized version of such algorithm was described by Minka (2005) and is summarized below.

Algorithm 3.2 Generalized Message Passing.

```
Initialize $q_0\left(\mathbf{l}\right) = p_0\left(\mathbf{l}\right)$ and $q_i\left(\mathbf{l}\right) \propto 1$ for $i = 1, \dots, N$.
repeat
    Pick a batch of random factors $B \subset [M]$.
    Compute $\log q_{-B}\left(\mathbf{l}\right) \propto \log q\left(\mathbf{l}\right) - \sum_{i \in B} \log q_i\left(\mathbf{l}\right)$.
    for each factor $i \in B$ do
        Update $q_i'\left(\mathbf{l}\right) \leftarrow \arg\min_{q_i \in \mathcal{Q}} \mathbb{D}[p_i\left(\mathbf{l} \mid \mathbf{e}\right) q_{-B}\left(\mathbf{l}\right) \parallel q_i\left(\mathbf{l}\right) q_{-B}\left(\mathbf{l}\right)]$.
    end for
    Incorporate updated factors into $q\left(\mathbf{l}\right)$: $\log q\left(\mathbf{l}\right) \propto \log q_{-B}\left(\mathbf{l}\right) + \sum_{i \in B} \log q_i'\left(\mathbf{l}\right)$.
until convergence
```

Message passing algorithms somewhat fell out of fashion after introduction of black-box variational inference methods (Ranganath et al., 2014), but recently were shown to be particularly suitable for distributed settings (*e.g.*, Hasenclever et al., 2017; Vehtari et al., 2020). We continue this discussion in Chapter 4, where we design computation and communication-efficient algorithms akin to message passing for learning and personalization in federated settings.

Part II

Case Studies

> I think that it is a relatively good approximation to truth—which is much too complicated to allow anything but approximations—that mathematical ideas originate in empirics.
>
> John von Neumann

In this part, we present four case studies that illustrate our framework. In fact, these studies stem from four independent research projects that have led us to notice the similarities between different multitask problems in the first place and distill them into a framework. In Chapter 4, we discuss federated learning and present an inferential perspective that allows us to develop new efficient algorithms both for learning and personalization in massively multitask settings. Chapter 5 focuses on reinforcement learning in nonstationary environments, how learning in such environments can be decomposed into multiple tasks, and how agents can learn to adapt to changes using approximate inference. Chapter 6 discusses zero-shot generalization in the context of multilingual machine translation. Finally, Chapter 7 presents a non-traditional take on the problem of interpretability in machine learning where we approach it from a multitask perspective.

Chapter 4

Federated Learning: An Inferential Approach

4.1 Summary

Our first case study focuses on Federated Learning (FL)—a framework for learning statistical models from heterogeneous data scattered across multiple entities (or clients) under the coordination of a central server that has no direct access to the local data (Kairouz et al., 2019). Motivated by some of the limitations of distributed optimization methods in this setting, we present a probabilistic approach to FL, viewing it as a distribution of learning tasks. Leveraging our PMM framework, we design scalable and efficient approximate posterior inference algorithms for learning global and personalized models in federated settings.

4.2 Motivation and Goals

Canonically, FL is formulated as a distributed optimization problem with a few distinctive properties such as unbalanced and non-i.i.d. data distribution across the clients and limited communication. The *de facto* standard algorithm for solving federated optimization is federated averaging (FedAvg) (McMahan et al., 2017), which proceeds in rounds of communication between the server and a random subset of clients, synchronously updating the server model after each round (Bonawitz et al., 2019). By allowing the clients perform *multiple* local SGD steps (or epochs) at each round, FedAvg can reduce the required communication by orders of magnitude compared to mini-batch (MB) SGD, and as a result dramatically speed up training.

The conceptual simplicity of FedAvg is very appealing—not only it can be used for learning a single global model for all clients, but such model can be easily personalized via fine-tuning. However, a closer look at the problem reveals multiple fundamental trade offs and limitations that optimization-based techniques run into, both when the goal is to learn a single global model for all clients or a collection of personalized models, one for each client.

Limitations of Federated Optimization

Trade offs when learning a single global model. In practice, due to statistical heterogeneity of the client data, it turns out that more local computation often leads to biased client updates and makes FedAvg stagnate at inferior optima. An example of such behavior is demonstrated on Figure 4.1 on a toy 2D federated linear regression problem with two clients: as we increase the number of local steps each client takes at every round, the rate of initial convergence improves, but FedAvg also starts saturating at a point further away from the global optimum.

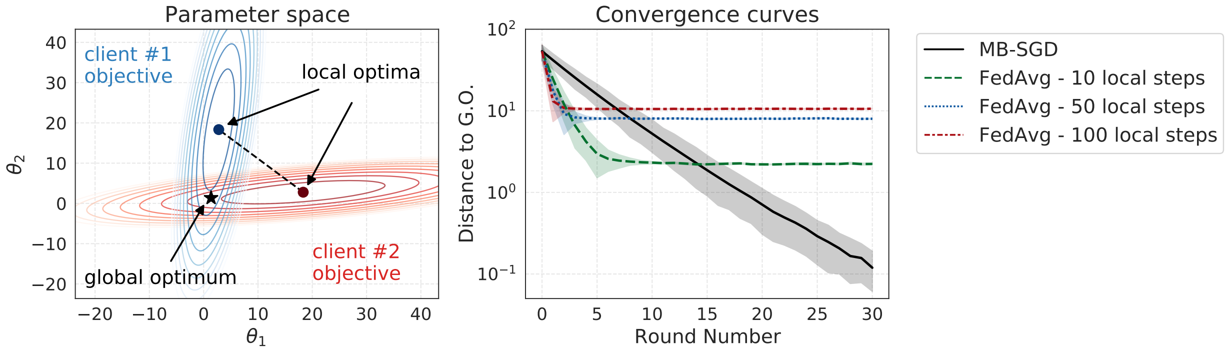


Figure 4.1: An illustration of the FedAvg behavior on a toy 2D federated linear regression problem. **Left:** Contour plots of the client objectives, their local optima, as well as the corresponding global optimum. **Right:** Learning curves for MB-SGD and FedAvg with 10 and 100 steps per round. FedAvg makes fast progress initially, but converges to a point far away from the global optimum. Shaded regions denote bootstrapped 95% CI based on 5 runs with different initializations and random seeds. Best viewed in color.

This behavior has been previously observed in multiple empirical studies (*e.g.*, Charles and Konecny, 2020; Pathak and Wainwright, 2020), and different ways to remedy such convergence issues have been proposed, ranging from regularizing local objective functions (*e.g.*, Li et al., 2018; Zhang et al., 2020; Acar et al., 2021) to using different control variate methods (Karimireddy et al., 2019; Pathak and Wainwright, 2020). However, most of these mitigation strategies intentionally have to limit the optimization progress clients can make at each round. In other words, approaching FL as a distributed optimization problem runs into a trade off between the amount of local progress allowed and the quality of the final solution. As we will see later in this chapter, taking an inferential approach would allow us to combat this issue by utilizing the available local computational resources more efficiently.

Limitations of fine-tuning-based personalization. A single global model may not be as useful in practice if it is inferior compared to models that clients can obtain individually by training on their own data, which may disincentivize the majority of clients from participating in federated training (Yu et al., 2020a). Thus, federated learning of personalized models is often highly desirable. The current go-to approach to personalization is simple fine-tuning of a global model on the local client's data. However, this approach makes training and evaluation disconnected, makes it harder to incorporate side information (*e.g.*, meta-data) that can be available on the clients, as well as requires all clients to be able to run local model optimization, which might be computationally prohibitive for some client devices (*e.g.*, with low memory).

In this chapter, we present a probabilistic approach to federated learning that helps us address the outlined limitations of federated optimization. The major part of this chapter focuses on learning a single global model, *i.e.*, the non-personalized setting, which we extend to personalized FL at the end. Our main contributions can be summarized as follows:

- We introduce a new perspective on federated learning through the lens of posterior inference which broadens the design space for FL algorithms beyond purely optimization techniques.
- With this perspective, first, we design a computation- and communication-efficient approximate posterior inference algorithm—*federated posterior averaging* (FedPA). Our algorithm works with stateless clients, has the computational complexity and memory footprint similar to FedAvg, but most importantly significantly benefits from an increased amount of local computation without stagnating at inferior optima.
- Next, we analyze convergence of FedAvg and FedPA and show that FedAvg with many local steps is in fact a special case of FedPA that estimates local posterior covariances with identities. These biased estimates are the source of inconsistent updates and explain why FedAvg has suboptimal convergence even in simple quadratic settings.
- Further, we compare FedPA with multiple strong baselines on realistic FL benchmarks introduced by Reddi et al. (2020) and achieve state-of-the-art results with respect to multiple metrics of interest, such the test accuracy of the final model and convergence speed in terms the number of communication rounds.
- Finally, we propose multiple ways to extend our framework to personalized FL, and demonstrate that FedPA used in conjunctions with simple fine-tuning or MAML improves the quality of personalized models as well on our benchmarks.

4.3 Preliminaries

Federated learning is typically formulated as the following distributed optimization problem:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \left\{ \mathcal{L}(\boldsymbol{\theta}) := \sum_{i=1}^{N} w_i \ell_i(\boldsymbol{\theta}) \right\}, \quad \ell_i(\boldsymbol{\theta}) := \frac{1}{N_i} \sum_{j=1}^{N_i} \ell(\boldsymbol{\theta}, \mathbf{z}_{ij}) \tag{4.1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$, the global objective function $\mathcal{L}(\boldsymbol{\theta})$ is a weighted average of the local objectives $\ell_i(\boldsymbol{\theta})$ over N clients. Each client's objective is the empirical risk with some loss $\ell(\boldsymbol{\theta}; \mathbf{z})$ computed on the local data D_i and the weights $\{w_i\}$ are typically set proportional to the sizes of the local datasets $\{N_i\}$, which makes $\mathcal{L}(\boldsymbol{\theta})$ coincide with the training objective of the centralized setting.

In federated learning, we distinguish between two different settings, called *cross-device* and *cross-silo* (Kairouz et al., 2019). In the cross-device setting, clients are typically mobile or edge devices and the number of clients could be extremely large (millions). In cross-silo, clients correspond to larger entities, such as organizations (*e.g.*, banks, hospitals, etc.). In this work, we are interested in the cross-device FL, which relies on the following modeling assumption: because of the very large number of clients, each client participates in training at most once (the majority of clients does not participate at all). As a consequence, we are interested in algorithms that can work with stateless clients and do not require revisiting each client multiple times.

Because the total number of clients N can be extremely large, optimization of $\mathcal{L}(\boldsymbol{\theta})$ is done over multiple rounds of communication between the server and a small random subset of M clients at every round. Reddi et al. (2020) reformulated FedAvg in a generalized form (Algorithm 4.1) using two nested optimization loops, where the inner loop corresponded to multiple steps (or epochs) of local optimization done by each client participating in a given round (ClientUpdate), and the outer loop corresponded to the server updating its state by averaging and applying of the updates (or deltas) returned by the clients (ServerUpdate). This formulation gives an additional flexibility of using arbitrary functions for client and server updates. As a result, Algorithm 4.1 can represent not only FedAvg, but a variety of other federated optimization algorithms (Wang et al., 2021). The algorithms designed in this chapter will be also represented in this general form.

Algorithm 4.1 Generalized FedAvg

```
input  initial $\boldsymbol{\theta}$, ClientUpdate, ServerUpdate
for each round $t = 1, \dots, T$ do
    Sample a subset $\mathcal{S}$ of clients
    communicate $\boldsymbol{\theta}$ to all clients $i \in \mathcal{S}$
    for each client $i \in \mathcal{S}$ in parallel do
        $\boldsymbol{\Delta}_i^t, w_i \leftarrow$ ClientUpdate($\boldsymbol{\theta}$)
    end for
    communicate $\{\boldsymbol{\Delta}_i^t, w_i\}_{i \in \mathcal{S}}$ to the server
    $\boldsymbol{\Delta}^t \leftarrow \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} w_i \boldsymbol{\Delta}_i^t$
    $\boldsymbol{\theta} \leftarrow$ ServerUpdate($\boldsymbol{\theta}, \boldsymbol{\Delta}^t$)
end for
output  final $\boldsymbol{\theta}$
```

4.4 Approach

In practice, the loss function $\ell(\boldsymbol{\theta}; \mathbf{z})$ is typically a negative log likelihood of $\mathbf{z}$ under some probabilistic model parametrized by $\boldsymbol{\theta}$, *i.e.*, $f(\boldsymbol{\theta}; \mathbf{z}) := -\log p\left(\mathbf{z} \mid \boldsymbol{\theta}\right)$. For example, least squares loss corresponds to likelihood under a Gaussian model, cross entropy loss corresponds to likelihood under a categorical model, etc. (Murphy, 2012). Thus, solving Equation 4.1 corresponds to *maximum likelihood estimation* (MLE) of the model parameters $\boldsymbol{\theta}$.

But what is the probabilistic model here?

Federated learning can be seen as a massively multitask problem, where each task corresponds to a client and we are interested in learning from a subset of clients that participate in training and generalizing to the rest of the client population. Thus, FL is natural to represent as a distribution of tasks (Figure 4.2). Note that when we are interested in learning a single global model for all clients, we have no local latent variables, only the global parameters $\boldsymbol{\theta}$.

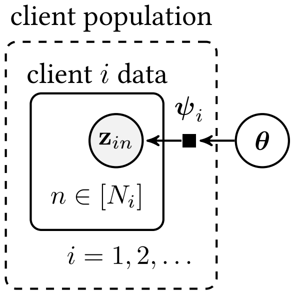


Figure 4.2: Factor graph representation of the non-personalized FL problem.

Instead of solving this problem using likelihood maximization (with FedAvg or some other federated optimization algorithm), we take an alternative (Bayesian) approach and propose to infer an approximate posterior distribution $p\left(\boldsymbol{\theta} \mid D \equiv D_1 \cup \dots \cup D_N\right)$ by minimizing the exclusive KL divergence (*cf.* Section 3.3.2):

$$\min_{q \in \mathcal{Q}} \mathrm{KL}[p\left(\boldsymbol{\theta} \mid D\right) \parallel q\left(\boldsymbol{\theta}\right)] \tag{4.2}$$

The posterior is proportional to the product of the likelihood and a prior, $p(\boldsymbol{\theta} \mid D) \propto p(D \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$. If the prior is uninformative (uniform), the modes of the global posterior coincide with MLE solutions or optima of $\mathcal{L}(\boldsymbol{\theta})$. While this simple observation establishes an equivalence between the inference of the posterior mode and optimization, the advantage of this perspective comes from the fact that the global posterior *exactly* decomposes into a product of local posteriors.

Proposition 4.1: Global Posterior Decomposition

Under the uniform prior, any global posterior distribution that exists decomposes into a product of local posteriors: $p\left(\boldsymbol{\theta} \mid D\right) \propto \prod_{i=1}^{N} p\left(\boldsymbol{\theta} \mid D_i\right)$.

Proof. Under the uniform prior, the following equivalence holds for $p\left(\boldsymbol{\theta} \mid D\right)$ as a function of $\boldsymbol{\theta}$:

$$p\left(\boldsymbol{\theta} \mid D\right) \propto p\left(D \mid \boldsymbol{\theta}\right) = \prod_{z \in D} p\left(z \mid \boldsymbol{\theta}\right) = \prod_{i=1}^{N} \underbrace{\prod_{z \in D_i} p\left(z \mid \boldsymbol{\theta}\right)}_{\text{local likelihood}} \propto \prod_{i=1}^{N} p\left(\boldsymbol{\theta} \mid D_i\right) \tag{4.3}$$

The proportionality constant between the left and right hand side is $\prod_{i=1}^{N} p\left(D_i\right) / p\left(D\right)$. □

Proposition 4.1 suggests to represent posterior approximation $q\left(\boldsymbol{\theta}\right)$ also in the form of a product, $q\left(\boldsymbol{\theta}\right) = \prod_{i=1}^{N} q_i\left(\boldsymbol{\theta}\right)$, where each component $q_i\left(\boldsymbol{\theta}\right)$ would approximate the corresponding local posterior $p\left(\boldsymbol{\theta} \mid D_i\right)$, which would enable distributed inference. As we mentioned at the end of Chapter 3, we can generally solve Equation 4.2 using expectation propagation (EP) algorithm (Minka, 2001; Hasenclever et al., 2017, see Section 3.3.2). However, we would need to make sure that the resulting algorithm can be efficiently executed in cross-device FL settings.

To that end, we propose to approximate local posteriors with Gaussians, $q_i\left(\boldsymbol{\theta}\right) = \mathcal{N}\left(\boldsymbol{\theta}; \boldsymbol{\mu}_i \boldsymbol{\Sigma}_i\right)$, which implies that the global approximations $q\left(\boldsymbol{\theta}\right)$ will also be Gaussian. Moreover, note that in the cross-device setting, we see each client at most once, and hence there is no need to explicitly store estimates of each component $q_i\left(\boldsymbol{\theta}\right)$ on the server (as EP does), since they will not be re-estimated during training more than once. This allows us to simplify EP (Algorithm 4.2).

Algorithm 4.2 Simplified EP for posterior inference in cross-devie FL

```
Initialize $q^{(0)}\left(\boldsymbol{\theta}\right) \propto 1$.
for each round $t = 1, \ldots, T$ do
    Sample a subset $\mathcal{S}_t$ of $M$ clients
    for each client $i \in \mathcal{S}_t$ in parallel do do
        $q_i^{(t)}\left(\boldsymbol{\theta}\right) \leftarrow \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\left[p_i\left(\boldsymbol{\theta}\right) q^{(t-1)}\left(\boldsymbol{\theta}\right) \,\middle\|\, q\left(\boldsymbol{\theta}\right)\right]$
    end for
    Aggregate the updates: $q^{(t)}\left(\boldsymbol{\theta}\right) \leftarrow q^{(t-1)}\left(\boldsymbol{\theta}\right) \prod_{i \in \mathcal{S}_t} \frac{q_i^{(t)}\left(\boldsymbol{\theta}\right)}{q^{(t-1)}\left(\boldsymbol{\theta}\right)}$
end for
output $q^{(T)}\left(\boldsymbol{\theta}\right)$
```

Note that Algorithm 4.2 is still impractical for realistically large models, because, naïvely, it would require communicating covariance (or precision) matrices from the clients to the server.

4.4.1 Federated Posterior Averaging: A Practical Algorithm

Note that we are only interested in computing the mode of $q(\boldsymbol{\theta})$ (which is the mean for Gaussians), and hence can further improve the efficiency of the distributed posterior inference algorithm. Specifically, the inclusive KL minimization and aggregation steps in Algorithm 4.2 result in the following updated mean of $q^{(t)}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$:

$$\boldsymbol{\mu}_t = \left(\boldsymbol{\Sigma}_{t-1}^{-1} + \sum_{i \in \mathcal{S}_t} \boldsymbol{\Sigma}_i^{-1}\right)^{-1} \left(\boldsymbol{\Sigma}_{t-1}^{-1}\boldsymbol{\mu}_{t-1} + \sum_{i \in \mathcal{S}_t} \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i\right) \tag{4.4}$$

We can see from Equation 4.4 that Algorithm 4.2 is an online filtering algorithm—given information about the local posteriors of a new subset of clients $\mathcal{S}_t$, it is incorporated in the moment estimates of the global posterior approximation. Note that after a sufficient number of rounds, $\boldsymbol{\mu}_t$ converges to the following approximate global posterior mode:

$$\boldsymbol{\mu}^\star = \left(\sum_{i=1}^{N} \boldsymbol{\Sigma}_i^{-1}\right)^{-1} \left(\sum_{i=1}^{N} \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i\right) \tag{4.5}$$

Directly computing $\boldsymbol{\mu}_t$ at each round is infeasible as it require $\mathcal{O}(d^2)$ space and $\mathcal{O}(d^3)$ computation, both on the clients and on the server, which is very expensive for the typical cross-device FL setting. Similarly, the communication cost would be $\mathcal{O}(d^2)$. For comparison, FedAvg requires only $\mathcal{O}(d)$ computation, communication, and storage. To arrive at a similarly efficient algorithm for posterior inference, we focus two questions: (a) how to estimate posterior moments efficiently? (b) how to communicate local statistics to the server efficiently?

(1) Efficient global posterior inference. There are two issues with computing an estimate of the global posterior mode directly using filtering (Equation 4.4). First, it requires computing the inverse of a $d \times d$ matrix on the server, which is an $\mathcal{O}(d^3)$ operation. Second, it relies on acquiring local means and inverse covariances, which would require $\mathcal{O}(d^2)$ communication from each client. We propose to solve both issues by converting the global posterior estimation into an equivalent optimization problem.

Proposition 4.2: Global Posterior Inference

The global posterior mode $\boldsymbol{\mu}^\star$ given in Equation 4.5 is the minimizer of a quadratic $Q(\boldsymbol{\theta}) := \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta} - \mathbf{b}^\top\boldsymbol{\theta}$, where $\mathbf{A} := \sum_{i=1}^{N} w_i \boldsymbol{\Sigma}_i^{-1}$ and $\mathbf{b} := \sum_{i=1}^{N} w_i \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i$.

Proposition 4.2 allows us to obtain a good estimate of $\boldsymbol{\mu}^\star$ by running stochastic optimization of the quadratic objective $Q(\boldsymbol{\theta})$ on the server. The stochastic gradient of $Q(\boldsymbol{\theta})$ is as follows:

$$\nabla Q(\boldsymbol{\theta}) := \sum_{i \in \mathcal{S}_t} w_i \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_i), \tag{4.6}$$

This suggests that we can obtain an approximate global posterior mode $\boldsymbol{\mu}^\star$ using the same generic Algorithm 4.1 as FedAvg but with slightly different client updates: $\boldsymbol{\Delta}_i := \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_i)$. Importantly, as long as clients are able to compute $\boldsymbol{\Delta}_i$'s efficiently, this approach will result in $\mathcal{O}(d)$ communication and $\mathcal{O}(d)$ server computation cost per round.

(2) Efficient local posterior inference. To compute $\boldsymbol{\Delta}_i$, each client needs to be able to estimate the local posterior means and covariances. We propose to use stochastic gradient Markov chain Monte Carlo (SG-MCMC, Welling and Teh, 2011; Ma et al., 2015) for approximate sampling from local posteriors on the clients, so that these samples can be used to estimate $\hat{\boldsymbol{\mu}}_i$'s and $\hat{\boldsymbol{\Sigma}}_i$'s. Specifically, we use a variant of SG-MCMC with iterate averaging (IASG, Mandt et al., 2017), which involves: (a) running local SGD for some number of steps to mix in the Markov chain, then (b) continued running of SGD for more steps to periodically produce samples via Polyak averaging (Polyak and Juditsky, 1992) of the intermediate iterates (Algorithm 4.3). The more computation is available locally on the clients each round, the more posterior samples can be produced, resulting in better estimates of the local moments.

Algorithm 4.3 IASG Sampling (ClientMCMC)

```
input initial $\boldsymbol{\theta}$, loss $f_i(\boldsymbol{\theta})$, optimizer ClientOpt$(\alpha)$,
    $B$: burn-in steps, $K$: steps per sample, $\ell$: # samples.
    // Burn-in
for step $t = 1, \dots, B$ do
    $\boldsymbol{\theta} \leftarrow$ ClientOpt$(\boldsymbol{\theta}, \hat{\nabla} f_i(\boldsymbol{\theta}))$
end for
    // Sampling
for sample $s = 1, \dots, \ell$ do
    $\mathcal{S}_{\boldsymbol{\theta}} \leftarrow \varnothing$                    // Initialize iterates
    for step $t = 1, \dots, K$ do
        $\boldsymbol{\theta} \leftarrow$ ClientOpt$(\boldsymbol{\theta}, \hat{\nabla} f_i(\boldsymbol{\theta}))$
        $\mathcal{S}_{\boldsymbol{\theta}} \leftarrow \mathcal{S}_{\boldsymbol{\theta}} \cup \{\boldsymbol{\theta}\}$
    end for
    $\boldsymbol{\theta}_s \leftarrow$ Average$(\mathcal{S}_{\boldsymbol{\theta}})$            // Average iterates
end for
output samples $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_\ell\}$
```

(3) Efficient computation of the deltas. Even if we can obtain samples $\{\hat{\boldsymbol{\theta}}_1, \dots, \hat{\boldsymbol{\theta}}_s\}$ via MCMC and use them to estimate local moments, $\hat{\boldsymbol{\mu}}_i$ and $\hat{\boldsymbol{\Sigma}}_i$, computing $\boldsymbol{\Delta}_i$ naïvely would still require inverting a $d \times d$ matrix, *i.e.*, $\mathcal{O}(d^3)$ compute and $\mathcal{O}(d^2)$ memory. The good news is that we can compute $\boldsymbol{\Delta}_i$'s much more efficiently, in $\mathcal{O}(d)$ time and memory, using a dynamic programming algorithm and appropriate mean and covariance estimators.

Theorem 4.1: Linear-time Computation of Client Deltas

Given s approximate posterior samples $\{\hat{\boldsymbol{\theta}}_1, \dots, \hat{\boldsymbol{\theta}}_s\}$, let $\hat{\boldsymbol{\mu}}_s$ be the sample mean, $\hat{\mathbf{S}}_s$ be the sample covariance, and $\hat{\boldsymbol{\Sigma}}_s := \rho_s \mathbf{I} + (1 - \rho_s)\hat{\mathbf{S}}_s$ be a shrinkage estimator (Ledoit and Wolf, 2004a) of the covariance with $\rho_s := 1/(1 + (s-1)\rho)$ for some $\rho \in [0, +\infty)$. Then, for any $\boldsymbol{\theta}$, we can compute $\hat{\boldsymbol{\Delta}}_s = \hat{\boldsymbol{\Sigma}}_s^{-1}(\boldsymbol{\theta} - \hat{\boldsymbol{\mu}}_s)$ in $\mathcal{O}(s^2 d)$ time and using $\mathcal{O}(sd)$ memory.

Proof sketch. We give a constructive proof by designing an efficient algorithm for computing $\hat{\boldsymbol{\Delta}}_s$. Our approach is based on two key ideas:

1. We prove that the shrinkage estimator of the covariance has a recursive decomposition into rank-1 updates, *i.e.*, $\hat{\boldsymbol{\Sigma}}_t = \hat{\boldsymbol{\Sigma}}_{t-1} + c_t \cdot \mathbf{x}_t^\top \mathbf{x}_t$, where c_t is a constant and $\mathbf{x}_t$ is some vector. This allows us to leverage the Sherman-Morrison formula for computing $\hat{\boldsymbol{\Sigma}}_s^{-1}$.
2. Further, we design a dynamic programming algorithm for computing $\hat{\boldsymbol{\Delta}}_s$ exactly without storing the covariance matrix or its inverse. Our algorithm is online and allows efficient updates of $\hat{\boldsymbol{\Delta}}_s$ as more posterior samples become available.

See Appendix B.3 for the full proof and derivation of the algorithm. □

Algorithm 4.4 Client Update (FedAvg)

```
input  initial $\boldsymbol{\theta}_0$, loss $f_i(\boldsymbol{\theta})$, optimizer ClientOpt
for $k = 1, \dots, K$ do
    $\boldsymbol{\theta}_k \leftarrow$ ClientOpt$(\boldsymbol{\theta}_{k-1}, \hat{\nabla} f_i(\boldsymbol{\theta}_{k-1}))$
end for
output  $\boldsymbol{\Delta} := \boldsymbol{\theta}_0 - \boldsymbol{\theta}_K$, client weight $w_i$
```

Algorithm 4.5 Client Update (FedPA)

```
input  initial $\boldsymbol{\theta}_0$, loss $f_i(\boldsymbol{\theta})$, sampler ClientMCMC
for $k = 1, \dots, K$ do
    $\boldsymbol{\theta}_k \sim$ ClientMCMC$(\boldsymbol{\theta}_{k-1}, f_i)$
end for
output  $\boldsymbol{\Delta} := \hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\theta}_0 - \hat{\boldsymbol{\mu}})$, client weight $w_i$
```

► To summarize, starting from a general posterior inference formulation (Equation 4.2), we proposed to use an EP-style filtering algorithm (Algorithm 4.2), which converges to the mode of the approximate global posterior (Equation 4.5). While running Algorithm 4.2 is infeasible in cross-device FL, we were able to further simplify it and improve its efficiency by converting computation of $\boldsymbol{\mu}^\star$ into a stochastic optimization problem. Putting all pieces together, we arrive at a new algorithm—*federated posterior averaging* (FedPA)—which is a variant of generalized federated optimization (Algorithm 4.1) with a new ClientUpdate procedure (Algorithm 4.5). The resulting algorithm is as efficient as FedAvg in terms of computation, communication, and memory footprint.

4.4.2 Analysis of the Algorithm

Does FedPA actually work? First, we test it on the toy 2D problem (Figure 4.1) which we initially used to illustrate the convergence issues of FedAvg. The results are presented on Figure 4.3. We can clearly see the reverse trends: as we increase the amount of local computation on the clients (used for producing more local posterior samples), FedPA converges faster *and* to a point much closer to the global optimum. In the rest of this section, we discuss convergence, computational and statistical efficiency, comparing FedPA and FedAvg with each other, before moving to larger experiments on realistic FL benchmark datasets.

Table 4.1: Computational complexity of the client updates for methods that use 5 local epochs measured in milliseconds (% denotes relative increase).

Dim	$\hat{\boldsymbol{\Delta}}_{\text{FedAvg}}$	$\hat{\boldsymbol{\Delta}}_\ell$ (DP)		$\hat{\boldsymbol{\Delta}}_\ell$ (exact)	
100	72	91	+26%	82	+12%
1K	76	92	+21%	104	+36%
10K	80	93	+16%	797	+896%
100K	149	155	+4%		—

Analysis of the computational overhead. Note that the computational cost of $\hat{\boldsymbol{\Delta}}_s$ consists of two components: (i) the cost of producing s approximate local posterior samples using IASG and (ii) the cost of solving a linear system using dynamic programming. How much of an overhead does it add compared to simply running local SGD? It turns out that in practical settings the overhead is almost negligible. Table 4.1 shows the time it takes a client to compute the updates based on 5 local epochs (100 steps per epoch) using different algorithms (FedAvg vs. our approach with exact or dynamic programming (DP) matrix inversion) on high-dimensional synthetic problems. As the dimensionality grows, computational complexity of DP-based estimation of $\hat{\boldsymbol{\Delta}}_s$ becomes nearly identical to FedAvg, which indicates that the majority of the cost in practice would come from SGD steps rather than our dynamic programming procedure.

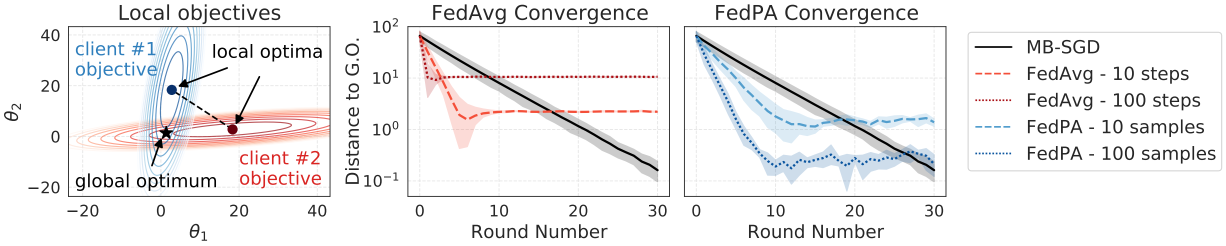


Figure 4.3: An illustration of federated learning in a toy 2D setting with two clients and quadratic objectives. **Left:** Contour plots of the client objectives, their local optima, as well as the corresponding global optimum. **Middle:** Learning curves for MB-SGD and FedAvg with 10 and 100 steps per round. FedAvg makes fast progress initially, but converges to a point far away from the global optimum. **Right:** Learning curves for FedPA with 10 and 100 posterior samples per round and shrinkage $\rho = 1$. More posterior samples (*i.e.*, more local computation) results in faster convergence and allows FedPA to come closer to the global optimum. Shaded regions denote 95% CI based on 5 runs. Best viewed in color.

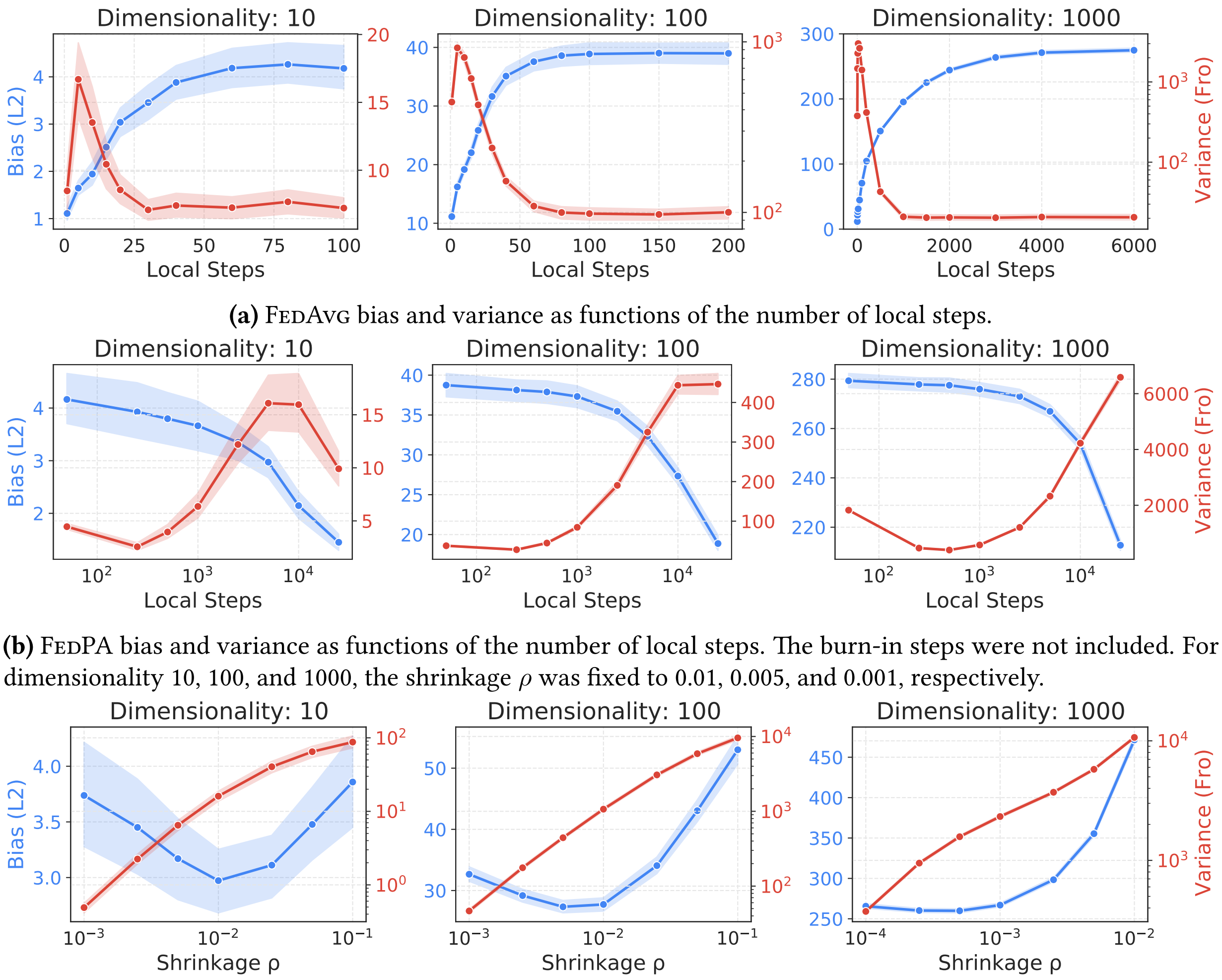
Convergence analysis of FedAvg vs. FedPA. Thanks to the similarity of the algorithms, note that FedAvg can be viewed as a posterior inference algorithm that aims to converge to $\boldsymbol{\mu}^\star$ (Equation 4.4), but estimates local posterior covariances $\boldsymbol{\Sigma}_i$ with an identity and means with the last iterate of multiple stochastic optimization steps. As a result, FedAvg obtains client deltas $\boldsymbol{\Delta}_{\text{FedAvg}} := \mathbf{I}(\boldsymbol{\theta} - \boldsymbol{\theta}_K)$, which are biased estimates of the gradients of the objective $Q(\boldsymbol{\theta})$ (Equation 4.6). On the other hand, FedPA uses local posterior samples that help de-bias the gradients, although may contribute to the variance. Thus, from our probabilistic perspective, the main difference between FedAvg and FedPA is the bias-variance trade off in the server gradient estimates $\nabla Q(\boldsymbol{\theta})$. As such, we can view both methods as *biased* SGD (Ajalloeian and Stich, 2020) on the $Q(\boldsymbol{\theta})$ objective and reason about their convergence rates as well as distances between their fixed points and correct global optima as functions of the gradient bias. We further discuss formal convergence analysis in Appendix B.1.

Quantifying empirically the bias and variance of $\hat{\boldsymbol{\Delta}}$ for FedPA and FedAvg. To better understand the differences in bias-variance trade offs of both algorithms, we measure exactly the empirical bias and variance of the client deltas computed by each method on the synthetic least squares linear regression problems.[1] The problems were generated as follows: for each dimensionality (10, 100, and 1000 features), we generated 10 random least squares problems, each of which consisted of 500 synthetic data points. Next, for each of the problems we generated 10 random initial model parameters $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_{10}\}$ and for each of the parameters we computed the exact $\boldsymbol{\Delta}_i$ as well as $\hat{\boldsymbol{\Delta}}_{\text{FedAvg},i}$ and $\hat{\boldsymbol{\Delta}}_{\text{FedPA},i}$ for different numbers of local steps. Using these sample estimates, we computed the L_2-norm of the bias and the Frobenius norm of the covariance matrices of the produced deltas as functions of the number of local steps.[2]

The results are presented on Figure 4.4. From Figure 4.4a, we see that as the amount of local computation increases, the bias in FedAvg delta estimates grows and the variance reduces. For FedPA (Figure 4.4b), the trends turn out to be the opposite: as the number of local steps

[1] The problems were generated according to Guyon (2003) using the `make_regression` function from scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html

[2] For $\hat{\boldsymbol{\Delta}}_{\text{FedPA}}$ we also varied the shrinkage hyperparameter ρ.



(a) FEDAVG bias and variance as functions of the number of local steps.

(b) FEDPA bias and variance as functions of the number of local steps. The burn-in steps were not included. For dimensionality 10, 100, and 1000, the shrinkage ρ was fixed to 0.01, 0.005, and 0.001, respectively.

(c) FEDPA bias and variance as functions of the shrinkage parameter. For dimensionality 10, 100, and 1000, the number of local steps was fixed to 5,000, 10,000, and 50,000, respectively.

Figure 4.4: The bias and variance trade offs for FEDAVG and FEDPA.

increases, the bias consistently reduces; the variance initially goes up, but with enough samples joins the downward trend. Note that the initial upward trend in the variance is due to the fact that we used the same *fixed* shrinkage ρ regardless of the number of local steps just to avoid varying multiple parameters at once. To avoid sharp increases in the variance, ρ must be selected for each number of local steps separately; Figure 4.4c demonstrates how the bias and variance depend on the shrinkage hyperparameter for some fixed number of local steps.[3]

► To summarize, our analysis suggests that both FEDAVG and FEDPA can be seen as biased stochastic gradient methods that aim to converge to the mode of the approximate global posterior. While FEDAVG can only reduce the bias by decreasing the amount of local optimization progress made by the clients, FEDPA de-biases its updates using local posterior samples (*i.e.*, better utilizing available local computational resources).

[3] One could also use posterior samples to estimate the best possible ρ that balances the bias-variance tradeoff (*e.g.*, Chen et al., 2010) and avoids sharp increases in the variance.

4.4.3 Posterior Inference for Personalization

So far, we have focused on federated learning of a single global model, then proposed to approach the problem from a probabilistic standpoint (as a distribution of tasks, Figure 4.2), and designed and analyzed a new algorithm for efficient approximate posterior inference in federated settings. But what about personalization? How does it fit into this picture?

Our PMM framework introduced in Chapter 3 makes things extremely straightforward. First, we have to slightly change the representation of the problem (Figure 4.5) and introduce client-specific local latent variables, $\boldsymbol{\phi}_i$. Then, these additional variables allow us to model local data of each clients with different distributions $p\left(D_i \mid \boldsymbol{\phi}_i, \boldsymbol{\theta}\right)$ due to conditioning on the client-specific latents. Now, we can formulate personalized FL as approximate inference of the posterior distribution over both local $\{\boldsymbol{\phi}_i\}$ and global $\boldsymbol{\theta}$ latent variables. In particular, at training time, we approximate $p\left(\boldsymbol{\theta} \mid D\right)$ with some $q\left(\boldsymbol{\theta}\right)$ by solving the same divergence minimization problem as before (Equation 4.2), where $p\left(\boldsymbol{\theta} \mid D\right)$ would still factorize into a product of local posteriors $p\left(\boldsymbol{\theta} \mid D_i\right) \propto p\left(D_i \mid \boldsymbol{\theta}\right)$, for which we have the following:

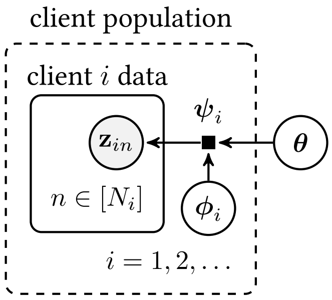


Figure 4.5: Factor graph for personalized FL.

$$\log p\left(D_i \mid \boldsymbol{\theta}\right) = \overbrace{\mathbb{E}_{q_i(\boldsymbol{\phi}_i|\boldsymbol{\theta})}\left[\log p\left(D_i \mid \boldsymbol{\phi}_i, \boldsymbol{\theta}\right) p\left(\boldsymbol{\phi}_i \mid \boldsymbol{\theta}\right)\right] + \mathbb{H}\left(q_i\left(\boldsymbol{\phi}_i \mid \boldsymbol{\theta}\right)\right)}^{\text{negative variational free energy } -\mathcal{F}_{q_i}(\boldsymbol{\theta})} + \underbrace{\mathrm{KL}[q_i\left(\boldsymbol{\phi}_i \mid \boldsymbol{\theta}\right) \,\|\, p\left(\boldsymbol{\phi}_i \mid D_i, \boldsymbol{\theta}\right)]}_{\text{divergence from the local posterior}} \tag{4.7}$$

Local variational free energy $\mathcal{F}_{q_i}\left(\boldsymbol{\theta}\right)$ lower bounds the local log likelihood $\log p\left(D_i \mid \boldsymbol{\theta}\right)$ and is tight when $q_i\left(\boldsymbol{\phi}_i \mid \boldsymbol{\theta}\right)$ coincides with $p\left(\boldsymbol{\phi}_i \mid D_i, \boldsymbol{\theta}\right)$. Thus, given access to a good enough $q_i\left(\boldsymbol{\phi}_i \mid \boldsymbol{\theta}\right)$, we can drop-in replace the local log likelihood with $\mathcal{F}_{q_i}\left(\boldsymbol{\theta}\right)$ and use it as the local objective $\ell_i(\boldsymbol{\theta})$ with FedPA or FedAvg for learning global parameters $\boldsymbol{\theta}$.

The final question is how do we get $q_i\left(\boldsymbol{\phi}_i \mid \boldsymbol{\theta}\right)$ that approximates $p\left(\boldsymbol{\phi}_i \mid D_i, \boldsymbol{\theta}\right)$? Recall that the factor graph model in Figure 4.5 is identical to the one we would use to represent a distribution of few-shot learning tasks that can be solved with meta-learning (Section 3.3.1). Also, recall that "model adaptation" of different meta-learning methods corresponds to different ways to approximate $p\left(\boldsymbol{\phi}_i \mid D_i, \boldsymbol{\theta}\right)$ (Section 3.3.2). Therefore, we can simply borrow adaptation from any off-the-shelf meta-learning method and use it to compute $q_i\left(\boldsymbol{\phi}_i \mid \boldsymbol{\theta}\right)$. Most meta-learning methods approximate $p\left(\boldsymbol{\phi}_i \mid D_i, \boldsymbol{\theta}\right)$ with a point estimate $\boldsymbol{\phi}_i(\boldsymbol{\theta}, D_i)$, which is a function of $\boldsymbol{\theta}$ and D_i. As a result, the variational free energy simplifies to the following:

$$\hat{\mathcal{F}}_i\left(\boldsymbol{\theta}\right) := -\log p\left(D_i \mid \hat{\boldsymbol{\phi}}_i(\boldsymbol{\theta}, D_i), \boldsymbol{\theta}\right) \tag{4.8}$$

► To sum up, in our framework, personalization corresponds to inference of the local latent variables, which can be done approximately using any model adaptation technique from meta-learning (*e.g.*, MAML, ProtoNets, NPs, etc.). Training can be still performed with FedPA (or FedAvg) using local variational free energy as the local objective.

4.5 Experiments

In this final part of the chapter, we empirically evaluate FedPA on a suite of FL benchmarks introduced by Reddi et al. (2020) against several competitive baselines: the best versions of FedAvg with adaptive optimizers as well as MIME (Karimireddy et al., 2020)—a recently-proposed FedAvg variant that also works with stateless clients, but uses control-variates and server-level statistics to improve convergence. Our evaluation is mainly focused on learning a single global model (*i.e.*, non-personalized FL), but we also provide a brief comparison of FedPA vs. FedAvg combined with MAML against fine-tuning baselines on one of the benchmarks at the very end.

Table 4.2: Statistics on the data and tasks. The number of examples per client are given with one standard deviation across the corresponding set of clients (denoted with ±). See description of the tasks in the text.

Dataset	Task	# classes	# clients (train/test)	# examples p/ client (train/test)
EMNIST-62	CR	62	3,400 / 3,400	198 ± 77 / 23 ± 9
CIFAR-100	IR	100	500 / 100	100 ± 0 / 100 ± 0
StackOverflow	LR	500	342,477 / 204,088	397 ± 1279 / 81 ± 301
	NWP	10,000		

4.5.1 The Setup

Datasets and tasks. The four benchmark tasks are based on the following three datasets (Table 4.2): EMNIST (Cohen et al., 2017), CIFAR100 (Krizhevsky and Hinton, 2009), and StackOverflow (StackOverflow, 2016). EMNIST (handwritten characters) and CIFAR100 (RGB images) are used for multi-class image classification tasks. StackOverflow (text) is used for next-word prediction (also a multi-class classification task, historically denoted NWP) and tag prediction (a multi-label classification task, historically denoted LR because a logistic regression model is used). EMNIST was partitioned by authors (Caldas et al., 2018), CIFAR100 was partitioned randomly into 600 clients with a realistic heterogeneous structure (Reddi et al., 2020), and StackOverflow was partitioned by its unique users. All datasets were preprocessed using the code provided by Reddi et al. (2020).

Methods and models. We use a generalized framework for federated optimization (Algorithm 4.1), which admits arbitrary adaptive server optimizers and expects clients to compute model deltas. As a baseline, we use federated averaging with adaptive optimizers (or with momentum) on the server and refer to it as FedAvg-1E or FedAvg-ME, which stands for 1 or multiple local epochs performed by clients at each round, respectively.[4] The number of local epochs in the multi-epoch versions is a hyperparameter. We use the same framework for federated posterior averaging and refer to it as FedPA-ME. As our clients use IASG to produce approximate posterior samples, collecting a single sample per epoch is optimal (Mandt et al., 2017). Thus FedPA-ME uses M samples to estimate client deltas and has the same local and global

[4] Reddi et al. (2020) referred to federated averaging with adaptive server optimizers as FedAdam, FedYogi, etc. Instead, we select the best optimizer for each task and refer to the corresponding method simply as FedAvg.

computational complexity as FedAvg-ME but with two extra hyperparameters: the number of burn-in rounds and the shrinkage coefficient ρ from Theorem 4.1. As in Reddi et al. (2020), we use the following model architectures for each task: CNN for EMNIST-62, ResNet-18 for CIFAR-100, LSTM for StackOverflow NWP, and multi-label logistic regression on bag-of-words vectors for StackOverflow LR (for details see Appendix B.4).

Hyperparameters. For hyperparameter tuning, we first ran small grid searches for FedAvg-ME using hyperparameter grids from Reddi et al. (2020). Then, we used the best FedAvg-ME configuration and did a small grid search to tune the additional hyperparameters of FedPA-ME, which turned out not to be very sensitive (*i.e.*, many configurations provided results superior to FedAvg). More hyperparameter details can be found in Appendix B.4.

Metrics. Since both speed of learning as well as final performance are important quantities for federated learning, we measure: (i) the number of rounds it takes the algorithm to attain a desired level of an evaluation metric and (ii) the best performance attained within a specified number of rounds. For EMNIST-62, we measure the number of rounds it takes different methods to achieve 84% and 86% evaluation accuracy[5], and the best validation accuracy attained within 500 and 1500 rounds. For CIFAR-100, we use the same metrics but use 30% and 40% as evaluation accuracy cutoffs and 1000 and 1500 as round number cutoffs. Finally, for StackOverflow, we measure the the number of rounds it takes to the best performance and evaluation accuracy (for the NWP task) and precision, recall at 5, macro- and micro-F1 (for the LR task) attained by round 1500. We note that the total number of rounds was selected based on computational considerations (to ensure reproducibility within a reasonable amount of computational cost) and the intermediate cutoffs were selected qualitatively to highlight some performance points of interest. In addition, we provide plots of the evaluation loss and other metrics for all methods over the course of training which show a much fuller picture of the behavior of the algorithms.

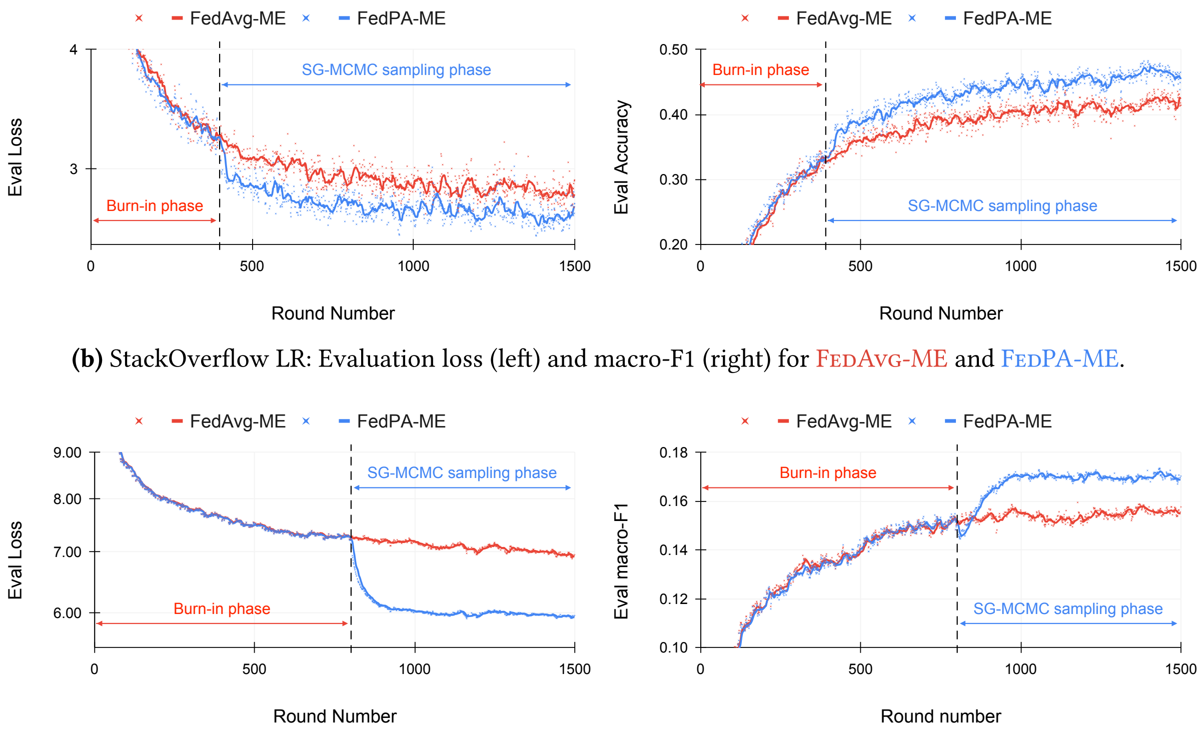
4.5.2 Results on Benchmark Tasks

The effects of posterior correction of client deltas. As we demonstrated in Section 4.4.2, FedPA essentially generalizes FedAvg and only differs in the computation done on the clients, where we compute client deltas using an estimator of the local posterior inverse covariance matrix, $\mathbf{\Sigma}_i^{-1}$, which requires sampling from the posterior. To be able to use SG-MCMC for local sampling, we first run FedPA in the *burn-in regime* (which is identical to FedAvg) for a number of rounds to bring the server state closer to the clients' local optima,[6] after which we "turn on" the local posterior sampling. The effect of switching from FedAvg to FedPA for CIFAR-100 (after 400 burn-in rounds) and StackOverflow LR (after 800 burn-in rounds) is presented on Figures 4.6a and 4.6b, respectively.[7] During the burn-in phase, evaluation performance is identical for both methods, but once FedPA starts computing client deltas using local posterior samples, the loss

[5] Centralized optimization of the CNN model on EMNIST-62 attains the evaluation accuracy of 88%.

[6] If SGD cannot reach the vicinity of clients' local optima within the specified number of local steps or epochs, estimated local means and covariances based on the SGD iterates can be arbitrarily poor.

[7] The number of burn-in rounds is a hyperparamter and was selected for each task to maximize performance. See more details in Appendix B.4.



(a) CIFAR-100: Evaluation loss (left) and accuracy (right) for FedAvg-ME and FedPA-ME.

(b) StackOverflow LR: Evaluation loss (left) and macro-F1 (right) for FedAvg-ME and FedPA-ME.

Figure 4.6: Evaluation metrics for FedAvg and FedPA computed at each training round on (a) CIFAR-100 and (b) StackOverflow LR. During the initial rounds ("burn-in"), FedPA computes deltas the same way as FedAvg; after that, FedPA computes deltas using Algorithm 4.5 and approximate posterior samples.

immediately drops and the convergence trajectory changes, indicating that FedPA is able to avoid stagnation and make progress towards a better optimum. Similar effects are observed across all other tasks (see Appendix B.5).

While the improvement of FedPA over FedAvg on some of the tasks is visually apparent (Figure 4.6), we provide a more detailed comparison of the methods in terms of the speed of learning and the attained performance on all four benchmark tasks, summarized in Table 4.3 and discussed below.

Results on EMNIST-62 and CIFAR-100. In Tables 4.3a and 4.3b, we present a comparison of FedPA against: tuned FedAvg with a fixed client learning rate (denoted FedAvg-1E and FedAvg-ME), the best variation of adaptive FedAvg from Reddi et al. (2020) with exponentially decaying client learning rates (denoted AFO), and MIME of Karimireddy et al. (2020). With more local epochs, we see significant improvement in terms of speed of learning: both FedPA-ME and FedAvg-ME achieve 84% accuracy on EMNIST-62 in under 100 rounds (similarly, both methods attain 30% on CIFAR-100 by round 350). However, more local computation eventually hurts FedAvg leading to worse optima: on EMNIST-62, FedAvg-ME is not able to consistently achieve 86% accuracy within 1500 rounds; on CIFAR-100, it takes extra 350 rounds for FedAvg-ME to get to 40% accuracy.

Table 4.3: Comparison of FedPA with baselines. All metrics were computed on the evaluation sets and averaged over the last 100 rounds before the round limit was reached. The "number of rounds to accuracy" was determined based on the 10-round running average crossing the threshold for the first time. The arrows indicate whether higher (↑) or lower (↓) value is better. The best performance is denoted in **bold**.

(a) EMNIST-62

	accuracy (%, ↑)		**rounds (#, ↓)**	
Method \@	500R	1500R	84%	86%
AFO †	80.4	86.8	546	1291
MIME ‡	83.1	*84.9	464	*—
FedAvg-1E	83.9	86.5	451	1360
FedAvg-ME	85.8	85.9	86	—
FedPA-ME	**86.5**	**87.3**	**84**	**92**

(b) CIFAR-100

	accuracy (%, ↑)		**rounds (#, ↓)**	
Method \@	1000R	1500R	30%	40%
AFO †	31.9	41.1	898	1401
MIME ‡	33.2	*33.9	680	*—
FedAvg-1E	24.2	31.7	1379	—
FedAvg-ME	40.2	42.1	**348**	896
FedPA-ME	**44.3**	**46.3**	**348**	**543**

(c) StackOverflow

	NWP		**LR (all metrics in %, ↑)**			
Method \ **Metric**	**accuracy (%, ↑)**	**rounds (#, ↓)**	**precision**	**recall@5**	**ma-F1**	**mi-F1**
AFO †	**23.4**	1049	—	68.0	—	—
FedAvg-1E	22.8	1074	74.58	**69.1**	14.9	43.8
FedAvg-ME	23.0	870	**78.65**	68.7	15.6	43.3
FedPA-ME	**23.4**	**805**	72.8	68.6	**17.3**	**44.0**

† the best results taken from (Reddi et al., 2020).
‡ the best results taken from (Karimireddy et al., 2020).
* results were only available for the method trained to 1000 rounds.

Finally, federated posterior averaging achieves the best performance on both tasks in terms of evaluation accuracy within the specified limit on the number of training rounds. On EMNIST-62 in particular, the final performance of FedPA-ME after 1500 training rounds is 87.3%, which, while only a 0.5% absolute improvement, bridges **41.7%** of the gap between the centralized model accuracy (88%) and the best federated accuracy from previous work (86.8%, Reddi et al., 2020).

Results on StackOverflow NWP and LR. Results for StackOverflow are presented in Table 4.3c. Although not as pronounced as for image datasets, we observe some improvement of FedPA over FedAvg here as well. For NWP, we have an accuracy gain of 0.4% over the best baseline. For the LR task, we compare methods in terms of average precision, recall at 5, and macro-/micro-F1. The first two metrics have appeared in some prior FL work, while the latter two are the primary evaluation metrics typically used in multi-label classification work (Gibaja and Ventura, 2015). Interestingly, while FedPA underperforms in terms of precision and recall, it substantially outperforms in terms of micro- and macro-averaged F1, especially the macro-F1. This indicates that while FedAvg learns a model that can better predict high-frequency labels, FedPA learns a model that better captures rare labels (Yang, 1999; Yang and Liu, 1999). Interestingly, note while FedPA improves on F1 metrics and has almost the same recall at 5, it's precision after 1500 rounds is worse than FedAvg. A more detailed discussion along with training curves for each evaluation metric are provided in Appendix B.5.

4.5.3 Evaluating Personalization

Finally, we evaluate FedAvg and FedPA with different personalization strategies on EMNIST-62.

The setup. To evaluate for personalization, we randomly split 3,400 clients available for EMNIST-62 into 2,500 training and 900 testing clients, each of which has a small training set and a small testing set. At test time, clients receive a model from the server and either apply it directly (baseline) or allowed to adapt it by fine-tuning it on their small training sets for 5 epochs.[8] At training time, we run FedAvg and FedPA on the training clients only for 1500 rounds with 5 client epochs per round; the algorithms either run as usual or use the variational free energy loss (Equation 4.8) computed on test subset of the client data after the model parameters are adapted using MAML on the training subset. We refer to the latter as FedAvg-MAML and FedPA-MAML. All in all, we have 4 different training methods (FedAvg and FedPA with and without MAML) and 2 different evaluation methods (model with and without fine-tuning).

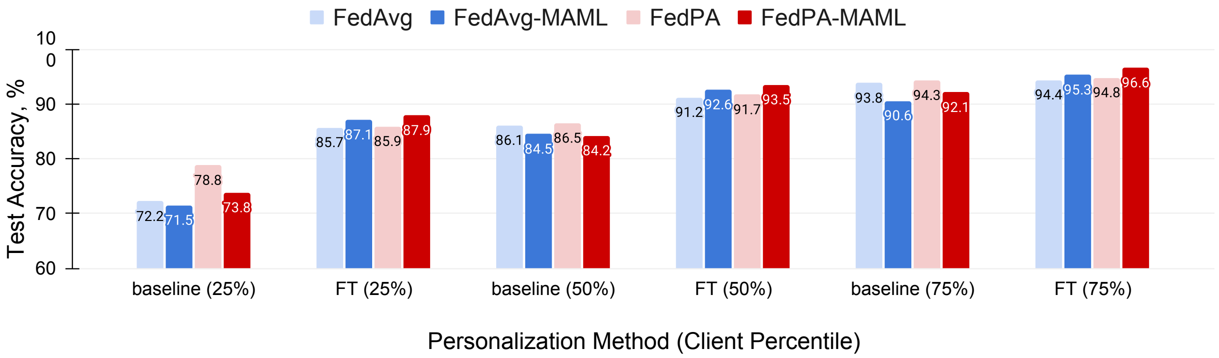


Figure 4.7: Results for personalized FL experiments. For each type of training (FedAvg and FedPA with or without MAML), we evaluate the baseline and the fine-tuned models on all test clients and report 25-th, 50-th, and 75-th test accuracy percentiles.

Results. The results are presented on Figure 4.7. Observe the following. First, fine-tuning significantly improves performance for all clients and clients that benefit the most are the those for which the baseline model is least accurate (the 25-th percentile in terms of test accuracy). Next, using MAML objective for training further leads to improved performance post-fine-tuning for both FedAvg and FedPA, but results in worse performance of the baseline model, *i.e.*, slightly better initialization for fine-tuning may not necessarily be a good model on its own. Finally, and most interestingly, fine-tuned FedPA models dominate FedAvg across all settings. Not only FedPA is able to converge to a better initialization using MAML losses on the clients, it is surprising to see that fine-tuning a model obtained by FedPA outperforms FedAvg even with standard losses—while the folklore belief (at least in the meta-learning community) is that the fixed point of FedAvg, also known as *Reptile* (Nichol et al., 2018), is a better initialization for subsequent fine-tuning, our results (on EMNIST-62) suggest that fine-tuning from a point closer to the global optimum might actually be an equal or better strategy, at least in FL.

[8] Note that the setting is different from few-shot learning in that client data are extremely heterogeneous and we neither control for the sizes nor the class balance of the training and testing sets.

4.6 Conclusion

In this chapter, we presented a new perspective on FL based on the idea of global posterior inference via local posterior averaging. Applying this perspective, we designed and analyzed a new algorithm that generalizes federated averaging, is similarly practical, and able to utilize local computation much more efficiently, yielding state-of-the-art results on multiple challenging FL benchmarks, both in non-personalized and personalized settings.

Our work sits at the intersection of two sub-fields of machine learning—the younger field of federated optimization and a more established filed of probabilistic inference—and attempts to bridge the two by reinterpreting and analyzing federated optimization from a probabilistic standpoint. We note that message passing algorithms (Minka, 2005) have been known to be effective for distributed or even asynchronous inference with sparse or infrequent communication. However, these techniques have been mainly applied to inference in small scale settings until more recently (Hasenclever et al., 2017; Vehtari et al., 2020). Even though we derived FedPA algorithm in this chapter as a special case of expectation propagation (EP) that can scale to cross-device FL (Section 4.4), the algorithm was originally inspired by the sub-posterior aggregation methods for scalable MCMC (Neiswanger et al., 2013; Scott et al., 2016), as we were not aware of this connection while working on the original paper (Al-Shedivat et al., 2021a).

Our algorithm required a number of specific approximations and design choices and has a few limitations which we discuss below. Nevertheless, we believe that the underlying approach has potential to significantly broaden the design space for FL algorithms beyond purely optimization techniques and hopefully inspire further research at the intersection of the two subfields.

4.6.1 Limitations and Future Work

- Convergence of FedPA highly depends on the quality of local posterior samples and the robustness of the covariance estimator. Methods that can compute better estimates of client deltas from fewer samples have the potential to dramatically improve convergence speed. Our choice of client delta estimator was mainly driven by simplicity and scalability considerations, and can be certainly improved.
- In experiments, we ran FedPA in the "burn-in" mode for a few rounds before switching to local posterior sampling. Exploring more adaptive strategies that gradually transition from FedAvg to FedPA over the course of training would be an interesting direction to explore.
- FedPA approximates the global posterior distribution with a Gaussian, which is extremely crude and holds only locally within a neighborhood of different local optima for neural networks. While computing a mode of a Gaussian posterior is perhaps the best we can do given the computation and communication constraints of federated learning, it would be interesting try to represent multi-modal posteriors using ensemble methods or other scalable techniques from Bayesian deep learning.
- Finally, while it is known, that posterior samples can be differentially private for free (Wang et al., 2015), better understanding of privacy implications of posterior inference in federated settings is an important and interesting direction of future work.

Chapter 5

Learning under Nonstationarity

5.1 Summary

In this chapter, we consider the problem of Reinforcement Learning (RL) in nonstationary environments. We observe that a nonstationary learning task can be decomposed into a sequence of stationary few-shot learning tasks. This decomposition allows us to reformulate the problem, leverage our PMM framework, and design a gradient-based meta-learning algorithm that enables continuous adaptation under nonstationarity. The approach is demonstrated on simulated nonstationary locomotion tasks and on challenging competitive multi-agent games.

5.2 Motivation and Goals

Recent progress in reinforcement learning (RL) has achieved very impressive results ranging from playing games (Mnih et al., 2015; Silver et al., 2016), to applications in dialogue systems (Li et al., 2016), to robotics (Levine et al., 2016). Despite the progress, the learning algorithms for solving many of these tasks are designed to deal with stationary environments. On the other hand, real-world is often nonstationary either due to complexity, changes in the dynamics or the objectives in the environment over the life-time of a system, or presence of multiple learning actors. Nonstationarity breaks the standard assumptions and requires agents to continuously adapt, both at training and execution time, in order to succeed.

Learning under nonstationary conditions is challenging. The classical approaches to dealing with nonstationarity are usually based on context detection (Da Silva et al., 2006) and tracking (Sutton et al., 2007), i.e., reacting to the already happened changes in the environment by continuously fine-tuning the policy. Unfortunately, modern deep RL algorithms, while able to achieve super-human performance on certain tasks, are known to be sample inefficient. Nevertheless, nonstationarity allows only for limited interaction before the properties of the environment change. Thus, it immediately puts learning into the few-shot regime and often renders simple fine-tuning methods impractical.

A nonstationary environment can be seen as a sequence of stationary tasks, and hence we propose to tackle it as a multi-task learning problem (Caruana, 1998). The learning-to-learn (or meta-learning) approaches (Schmidhuber, 1987; Thrun and Pratt, 1998) are particularly appealing in the few-shot regime, as they produce flexible learning rules that can generalize from only a handful of examples. Meta-learning has shown promising results in the supervised domain and have gained a lot of attention from the research community recently (e.g., Santoro et al., 2016; Ravi and Larochelle, 2016, *inter alia*). In this chapter, our goal is to develop a gradient-based meta-learning algorithm similar to (Finn et al., 2017) and suitable for continuous adaptation of RL agents in nonstationary environments. More concretely, our agents meta-learn to anticipate the changes in the environment and update their policies accordingly.

While virtually any changes in an environment could induce nonstationarity (e.g., changes in the physics or characteristics of the agent), environments with multiple agents are particularly challenging due to complexity of the emergent behavior and are of practical interest with applications ranging from multiplayer games (Peng et al., 2017) to coordinating self-driving fleets (Cao et al., 2013). Multi-agent environments are nonstationary from the perspective of any individual agent since all actors are learning and changing concurrently (Lowe et al., 2017). Thus, one of the central problems that we focus on solving in this chapter is the problem of *continuous adaptation to a learning opponent* in a competitive multi-agent setting.

5.3 Related Work

The problem of *continuous adaptation* considered in this work is a variant of *continual learning* (Ring, 1994) and is related to *lifelong* (Thrun and Pratt, 1998; Silver et al., 2013) and *never-ending* (Mitchell et al., 2015) learning. Life-long learning systems aim at solving a growing number of tasks sequentially by efficiently transferring and utilizing knowledge from already learned tasks to new tasks while minimizing the effect of catastrophic forgetting (McCloskey and Cohen, 1989; Kirkpatrick et al., 2017). Never-ending learning is concerned with mastering a fixed set of tasks in iterations, where the set keeps growing over time and we are interested improving performance on all the tasks in the set from iteration to iteration.

The scope of continuous adaptation is narrower and more precise. While life-long and never-ending learning settings are defined as general multi-task problems (Silver et al., 2013; Mitchell et al., 2015), continuous adaptation targets to solve a single but complex and nonstationary learning problem. While nonstationarity in the former two settings may exist, it is dictated by some unknown process that generated the sequence of incoming tasks and, generally, is not assumed to be predictable. In the case of continuous adaptation, we assume that nonstationarity is caused by some underlying dynamics in the properties of a given environment (e.g., changes in the behavior of other agents in a multi-agent setting), which is supposed to be more predictable. Finally, in the life-long and never-ending scenarios the boundary between training and execution is blurred as such systems constantly operate in the training regime. In continuous adaptation, on the other hand, we distinguish between training and execution phases, and expect a trained agent to have learned to adapt to the changes in a nonstationary environment at execution time under the pressure of limited data or interaction experience between the changes.

Nonstationarity of multi-agent environments is a well known issue that has been extensively studied in the context of learning in simple multi-player iterated games (such as rock-paper-scissors) where each episode is one-shot interaction (Singh et al., 2000; Bowling, 2005; Conitzer and Sandholm, 2007). In such games, discovering and converging to a Nash equilibrium strategy is a success for the learning agents. Modeling and exploiting opponents (Zhang and Lesser, 2010; Mealing and Shapiro, 2013) or even their learning processes (Foerster et al., 2018a) is advantageous as it improves convergence or helps to discover equilibria of certain properties (e.g., leads to cooperative behavior). In contrast to much of the prior work, each episode in the environments that we consider here consists of multiple steps, happens in continuous time, and requires learning a good intra-episodic controller. Finding Nash equilibria in such a setting is hard. Thus, fast adaptation becomes one of the few viable strategies against changing opponents.

5.4 Approach

The problem of continuous adaptation in nonstationary environments immediately puts learning into the few-shot regime: the agent must learn from only limited amount of experience that it can collect before its environment changes. Therefore, we build our method upon the previous work on Model-agnostic Meta-Learning (MAML) that has been shown successful in the few-shot settings (Finn et al., 2017). Importantly, we show how to approach the problem using our PMM framework, re-derive MAML for distributions of RL tasks from a probabilistic standpoint, and then extend it to continuous adaptation under nonstationarity.

5.4.1 A Probabilistic View of Meta-RL and MAML

In this section, we show how to represent distributions of RL tasks using our PMM framework and reinterpret meta reinforcement learning (meta-RL) as a way to approximate inference under the assumed probabilistic model. Our discussion is particularly focused on MAML, which is later extended to enable continuous adaptation.

Assume that we are given a distribution over tasks $\mathbb{P}$ where each task $T_i \sim \mathbb{P}$ is a Markov Decision Process (MDP), which can be defined by a tuple (*cf.* Definition 3.1 in Section 3.2):

$$T_i := (\mathcal{X}_i, \mathcal{Y}_i, p_i, \mathcal{L}_i) \tag{5.1}$$

Here, $\mathcal{X}_i$ and $\mathcal{Y}_i$ are the state and action spaces, respectively (both are typically vector spaces). p_T is a probability distribution defined over sequences of states and actions of length H (stands for *horizon*), called trajectories, denoted $\boldsymbol{\tau}_i := (\mathbf{x}_{i0}, \mathbf{a}_{i1}, \mathbf{x}_{i1}, \ldots, \mathbf{x}_{iH-1}, \mathbf{a}_{iH}) \in \mathcal{T}$, which defines the Markovian dynamics of the environment in task T. The loss function $\mathcal{L}_i$ is defined for a trajectory $\boldsymbol{\tau}_i$ through a reward function $r_i : \mathcal{X} \mapsto \mathbb{R}$ as the negative cumulative reward of the states visited along the trajectory, $\mathcal{L}_i(\boldsymbol{\tau}) := -\sum_{t=1}^{H} r_i(\mathbf{x}_t)$.

The goal of meta-RL is to find an algorithm $\mathcal{A}$ which, given access to a limited experience on a task sampled from $\mathbb{P}$, can produce a good policy for solving it. While solving a distribution of RL tasks is conceptually similar to solving a distribution of supervised tasks (Chapter 4), there is an important difference. In the supervised setting, we assumed that tasks come with (or represented by) a small labeled dataset. In RL, however, tasks do not come with data but give the agent an opportunity to interact with an environment and *generate* data (*i.e.*, a few trajectories) under some policy.[1] In other words in RL, the solution of a task (*i.e.*, the policy) and the data it is learned from are coupled. Thus, to represent a distribution of RL tasks, we need a slightly different graphical model than what we used in Chapters 3 and 4.

Figure 5.1 presents a factor graph for a distribution of RL tasks. Because trajectories depend on the policy used to interact with the environment, to enable policy adaptation during the interaction, we have to explicitly split trajectories into two subsets, $\boldsymbol{\tau}_i$ and $\boldsymbol{\tau}'_i$, where $\boldsymbol{\tau}_i$ are obtained under the initial policy and $\boldsymbol{\tau}'_i$ under the adapted (task-specific) policy. Then, the adapted policy, represented by factor π_i, can depend on the the initial trajectories $\boldsymbol{\tau}_i$, the local latent variables $\boldsymbol{\phi}_i$, and the global parameters $\boldsymbol{\theta}$. Note that without rolling out $\boldsymbol{\tau}_i$ under some initial policy, we would not have any information about the corresponding RL task.[2] Finally, note that while $\boldsymbol{\tau}_i$ and $\boldsymbol{\tau}'_i$ depend on the dynamics and the reward function specified by the task T_i, we do not model them explicitly (*i.e.*, consider model-free RL), assuming that local latent variables $\boldsymbol{\phi}_i$ can capture the sufficient information about the task.

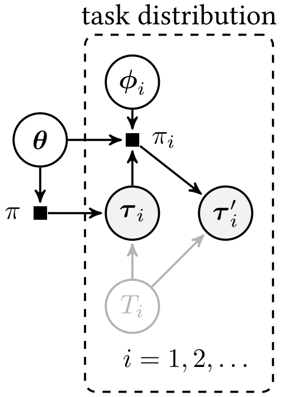


Figure 5.1: Factor graph for a distribution of RL tasks.

Under the specified probabilistic model, we can represent meta-learning as inference of the global parameters $\boldsymbol{\theta}$ and policy adaptation as inference of the local latent variables $\boldsymbol{\phi}_i$. More precisely, following Section 3.3.2, for a set of training tasks $T_1, \ldots, T_N$, we can infer an approximate local posterior $q_i\left(\boldsymbol{\phi}_i\right)$ for each task and obtain an estimate of $\boldsymbol{\theta}$ by solving the following general variational inference problem:

$$\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N}\left\{\mathbb{E}_{q_i(\boldsymbol{\phi}_i)}\left[\mathbb{E}_{\boldsymbol{\tau}'_i\sim p_i(\boldsymbol{\tau}\mid\boldsymbol{\theta},\boldsymbol{\phi}_i)}\left[\mathcal{L}_i\left(\boldsymbol{\tau}'_i\right)\right]\right]+\mathbb{H}\left(q_i\right)\right\} \tag{5.2}$$

where $q_i\left(\boldsymbol{\phi}_i\right)$ is selected to approximate $p\left(\boldsymbol{\phi}_i \mid \boldsymbol{\tau}_i, \boldsymbol{\theta}\right)$. MAML makes further approximations:

1. First, we assume that $\boldsymbol{\phi}_i$ are parameters of the task-specific policy π_i and $\boldsymbol{\theta}$ is a common initialization of these parameters. In other words, we have the same functional representation of the initial policy π and the task-specific policy π_i and the difference is only in the set of parameters being is used. To make these dependencies explicit in our notation, we further refer to these policies as $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\phi}_i}$ and to the trajectories rolled out under these policies as $\boldsymbol{\tau}_{i,\boldsymbol{\theta}}$ and $\boldsymbol{\tau}_{i,\boldsymbol{\phi}_i}$, respectively.
2. Second, we approximate $q_i\left(\boldsymbol{\phi}_i\right)$ with a point estimate $\boldsymbol{\phi}_i\left(\boldsymbol{\theta}\right)$, which is a function of $\boldsymbol{\theta}$ obtained by taking a gradient step from $\boldsymbol{\theta}$ on the loss computed on K trajectories $\boldsymbol{\tau}_i^{1:K}$

[1] In this chapter, we do not consider off-line reinforcement learning, where the agent is expected to learn a policy from a fixed set of trajectories, without interacting with an environment.

[2] An alternative to having to interact with the environment under an initial policy is to have access to some meta-data that describes the task T_i. Such meta-data could be, for example, a natural language description, which we could condition the task-specific policy on instead. We do not consider such an extended setup here.

sampled under the initial policy $\pi_{\boldsymbol{\theta}}$:

$$\boldsymbol{\phi}_i\left(\boldsymbol{\theta}\right) := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \underbrace{\mathbb{E}_{\boldsymbol{\tau}^{1:K}_{i,\boldsymbol{\theta}} \sim p_i(\boldsymbol{\tau} \mid \boldsymbol{\theta})^K}\left[\mathcal{L}_i\left(\boldsymbol{\tau}^{1:K}_{i,\boldsymbol{\theta}}\right)\right]}_{\text{loss over } \boldsymbol{\tau}_{i,\boldsymbol{\theta}} \text{ sampled from } T_i \text{ under } \pi_{\boldsymbol{\theta}}} \tag{5.3}$$

where $\mathcal{L}_i\left(\boldsymbol{\tau}^{1:K}_{i,\boldsymbol{\theta}}\right) := \frac{1}{K}\sum_{k=1}^{K} \mathcal{L}_i\left(\boldsymbol{\tau}^{k}_{i,\boldsymbol{\theta}}\right)$. We call Equation 5.3 the *adaptation update*. As a result, MAML applied to a distribution of RL tasks approximately solves Equation 5.2 by optimizing the following objective function end-to-end with respect to $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \left\{ \mathbb{L}_i\left(\boldsymbol{\theta}\right) := \mathbb{E}_{\boldsymbol{\tau}_{i,\boldsymbol{\theta}} \sim p_i(\boldsymbol{\tau} \mid \boldsymbol{\theta})} \left[\mathbb{E}_{\boldsymbol{\tau}_{i,\boldsymbol{\phi}_i} \sim p_i(\boldsymbol{\tau} \mid \boldsymbol{\phi}_i(\boldsymbol{\theta}))} \left[\mathcal{L}_i\left(\boldsymbol{\tau}_{i,\boldsymbol{\phi}_i}\right) \mid \boldsymbol{\tau}^{1:K}_{i,\boldsymbol{\theta}} \right] \right] \right\} \tag{5.4}$$

where $\boldsymbol{\phi}_i\left(\boldsymbol{\theta}\right)$ is computed in the inner loop as given in Equation 5.3. In practice, the expectations are approximated by sampling, *i.e.*, interacting with the environment and rolling out trajectories under the specified policies. The objective can be optimized using the policy gradient method (Williams, 1992), where the gradient of $\mathbb{L}_i\left(\boldsymbol{\theta}\right)$ is as follows:

$$\nabla_{\boldsymbol{\theta}} \mathbb{L}_i\left(\boldsymbol{\theta}\right) = \mathbb{E}_{\substack{\boldsymbol{\tau}^{1:K}_{i,\boldsymbol{\theta}} \sim p_i(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \\ \boldsymbol{\tau}_{i,\boldsymbol{\phi}} \sim p_i(\boldsymbol{\tau} \mid \boldsymbol{\phi}_i)}} \left[\mathcal{L}_i\left(\boldsymbol{\tau}_{i,\boldsymbol{\phi}_i}\right) \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\phi}_i}(\boldsymbol{\tau}_{i,\boldsymbol{\phi}_i}) + \nabla_{\boldsymbol{\theta}} \sum_{k=1}^{K} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}^{(k)}_{i,\boldsymbol{\theta}}) \right] \right] \tag{5.5}$$

The meta-losses $\mathbb{L}_i$ can be optimized using trust-region policy (TRPO) (Schulman et al., 2015a) or proximal policy (PPO) (Schulman et al., 2017) optimization methods. For details and derivations please refer to Appendix C.1.

5.4.2 Continuous Adaptation via Meta-learning

Up to this point, we discussed how use meta-learning for solving distributions of stationary RL tasks. But is it possible to use a similar technique for solving distributions of nonstationary RL tasks? Observe that we can view a nonstationary environment as stationary on a certain timescale, and thus decompose a nonstationary RL problem into a sequence of stationary tasks that depend on each other. Hence, we can exploit this dependence between consecutive tasks and meta-learn a rule that keeps updating the policy in a way that minimizes the total expected loss encountered during the interaction with the changing environment. For instance, in a competitive multi-agent game, when playing against an opponent that changes its strategy incrementally (*e.g.*, due to learning), our agent should ideally meta-learn to anticipate the changes and update its policy accordingly.

In the probabilistic language, a nonstationary environment can be represented by a Markov chain of tasks. Moreover, our probabilistic representation makes it easy to extend a distribution of RL tasks sampled i.i.d. (Figure 5.1) to a distribution over chains of tasks (Figure 5.2). Under this new model, we can define *continuous adaptation* as inference of $p\left(\boldsymbol{\phi}_i^{t+1} \mid \boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_t, \boldsymbol{\theta}\right)$, which we approximate in the same manner as MAML, *i.e.*, with a point estimate obtained by taking a few gradient steps on the inner loop objective starting from the common initialization.

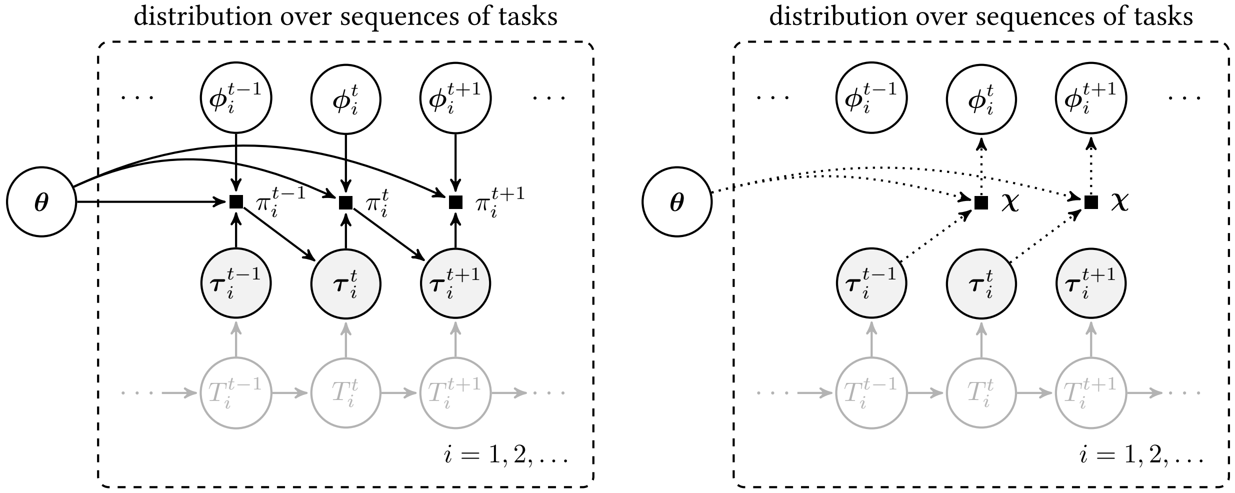


Figure 5.2: A probabilistic model that represents a distribution of nonstationary RL problems. **Left:** A factor graph that represents a nonstationary problem decomposed into a sequence of stationary RL tasks $(T_i^1, T_i^2, \ldots)$. **Right:** A graph that illustrates amortized inference of the task-specific variables.

To construct adapted parameters of the policy for task T_{i+1}, we start from $\boldsymbol{\theta}$ and do multiple meta-gradient steps with adaptive step sizes as follows (assuming the number of steps is M):[3]

$$
\begin{aligned}
\boldsymbol{\phi}_{i,0}^{t+1} &:= \boldsymbol{\theta}, \quad m = 1, \ldots, (M-1), \\
\boldsymbol{\phi}_{i,m}^{t+1} &:= \boldsymbol{\phi}_{i,m-1}^{t+1} - \alpha_m \nabla_{\boldsymbol{\phi}_{i,m-1}^{t+1}} \mathcal{L}_i^t \left(\boldsymbol{\tau}_{i,\boldsymbol{\phi}_{i,m-1}^{t+1}}^{1:K} \right), \\
\boldsymbol{\phi}_i^{t+1} &:= \boldsymbol{\phi}_{i,M-1}^{t+1} - \alpha_M \nabla_{\boldsymbol{\phi}_i^{M-1}} \mathcal{L}_i^t \left(\boldsymbol{\tau}_{i,\boldsymbol{\phi}_i^{M-1}}^{1:K} \right)
\end{aligned}
\tag{5.6}
$$

where $\{\alpha_m\}_{m=1}^M$ is a set of meta-gradient step sizes that are optimized jointly with $\boldsymbol{\theta}$. Note that our procedure makes a simplifying approximation and computes $\boldsymbol{\phi}_i^{t+1}$ by taking gradient steps on the $\mathcal{L}_i^t$ objective instead of $\mathcal{L}_i^{t+1}$, which implicitly relies on the similarity between $\mathcal{L}_i^t$ and $\mathcal{L}_i^{t+1}$ and is likely to work only when consecutive tasks in the Markov chain are not too different.

Under our adaptation update, given N sequences of L consecutive tasks each, Equation 5.2 reduces to the following optimization problem:

$$
\min_{\boldsymbol{\theta}} \frac{1}{NL} \sum_{i=1}^{N} \sum_{t=1}^{L-1} \mathbb{L}_i^{t,t+1}(\boldsymbol{\theta})
\tag{5.7}
$$

where, exploiting the Markovian assumption, $\mathbb{L}_i^{t,t+1}(\boldsymbol{\theta})$ is defined on pairs of consecutive tasks:

$$
\mathbb{L}_i^{t,t+1}(\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{\tau}_{i,t}^{1:K} \sim p_i^t(\boldsymbol{\tau}_{i,t}|\boldsymbol{\theta})^K} \left[\mathbb{E}_{\boldsymbol{\tau}_{i,t+1} \sim p_i^{t+1}(\boldsymbol{\tau}|\boldsymbol{\phi}_i)} \left[\mathcal{L}_i^{t+1}(\boldsymbol{\tau}_{i,t+1}) \mid \boldsymbol{\tau}_{i,t}^{1:K}, \boldsymbol{\theta} \right] \right]
\tag{5.8}
$$

The main difference between the loss in Equation 5.4 and Equation 5.8 is that trajectories $\boldsymbol{\tau}_{i,t}^{1:K}$ come from the current task, T_i^t, and are used to compute adapted policy parameters $\boldsymbol{\phi}_i^{t+1}$ for the

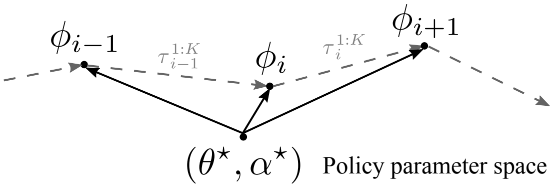
[3] Constructing $\boldsymbol{\phi}$ via multiple gradient steps with adaptive step sizes tends to yield better results in practice.

Algorithm 5.1 Meta-learning at training time.

```
input  Distribution over pairs of tasks, $\mathcal{P}(T_i, T_{i+1})$,
       learning rate, $\beta$.
Randomly initialize $\theta$ and $\alpha$.
repeat
    Sample task pairs, $\{(T_i, T_{i+1})\}_{i=1}^{n}$.
    for all task pairs $(T_i, T_{i+1})$ in the batch do
        Sample traj. $\boldsymbol{\tau}_\theta^{1:K}$ from $T_i$ using $\pi_\theta$.
        Compute $\phi = \phi(\boldsymbol{\tau}_\theta^{1:K}, \theta, \alpha)$.
        Sample traj. $\boldsymbol{\tau}_\phi$ from $T_{i+1}$ using $\pi_\phi$.
    end for
    Compute $\nabla_\theta \mathcal{L}_{T_i, T_{i+1}}$ and $\nabla_\alpha \mathcal{L}_{T_i, T_{i+1}}$ using
    sampled trajectories $\boldsymbol{\tau}_\theta^{1:K}$ and $\boldsymbol{\tau}_\phi$.
    Update $\theta \leftarrow \theta + \beta \nabla_\theta \mathcal{L}_T(\theta, \alpha)$.
    Update $\alpha \leftarrow \alpha + \beta \nabla_\alpha \mathcal{L}_T(\theta, \alpha)$.
until Convergence
output  Optimal $\theta^*$ and $\alpha^*$.
```

Algorithm 5.2 Adaptation at execution time.

```
input  A stream of tasks, $T_1, T_2, T_3, \ldots$.
Initialize $\phi = \theta$.
while there are new incoming tasks do
    Get a new task, $T_i$, from the stream.
    Solve $T_i$ using $\pi_\phi$ policy.
    While solving $T_i$, collect trajectories, $\boldsymbol{\tau}_{i,\phi}^{1:K}$.
    Update $\phi \leftarrow \phi(\boldsymbol{\tau}_{i,\phi}^{1:K}, \theta^*, \alpha^*)$ using
    importance-corrected meta-update.
end while
```



upcoming task T_{i+1}. Hence, optimizing $\mathbb{L}_i^{t,t+1}(\boldsymbol{\theta})$ is equivalent to truncated backpropagation through time with a unit lag in the chain of tasks. To optimize $\mathbb{L}_i^{t,t+1}(\boldsymbol{\theta})$, we can use policy gradient method as well. The analog of the policy gradient theorem (Sutton et al., 2000) for our setting is formulated and proven in Appendix C.1.

Meta-learning at training time. Given a sequences of tasks generated by a nonstationary environment, $T_i^1, T_i^2, T_i^3, \ldots, T_i^L$, we use the set of all pairs of consecutive tasks, $\{(T_i^t, T_i^{t+1})\}_{t=1}^{L}$, to construct the meta-loss. Then, we can meta-learn the adaptation updates by optimizing $\boldsymbol{\theta}$ and α jointly with a gradient method, as given in Algorithm 5.1. We use $\pi_{\boldsymbol{\theta}}$ to collect trajectories from T_i and π_ϕ when interacting with T_{i+1}. Intuitively, the algorithm is searching for $\boldsymbol{\theta}$ and α such that the adaptation update Equation 5.6 computed on the trajectories from T_i brings us to a policy, π_ϕ, that is good for solving T_{i+1}. The main assumption here is that the trajectories from T_i contain some information about T_{i+1}. Note that we treat adaptation steps as part of the computation graph and optimize $\boldsymbol{\theta}$ and α via backpropagation through the entire graph, which requires computing second order derivatives.

Adaptation at execution time. Note that to compute unbiased adaptation gradients at training time, we have to collect experience in T_i^t using $\pi_{\boldsymbol{\theta}}$. At test time, due to environment nonstationarity, we usually do not have the luxury to access to the same task multiple times. Thus, we keep acting according to $\pi_{\boldsymbol{\phi}}$ and re-use past experience to compute updates of $\boldsymbol{\phi}$ for each new incoming task, as summarized in Algorithm 5.2. To adjust for the fact that the past experience was collected under a policy different from $\pi_{\boldsymbol{\theta}}$, we use importance weight correction. In case of single step meta-update, we have:

$$\boldsymbol{\phi}_i^{t+1} := \boldsymbol{\theta} - \alpha \frac{1}{K} \sum_{k=1}^{K} \left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{\tau}_{i,t}^k)}{\pi_{\boldsymbol{\phi}_i^t}\left(\boldsymbol{\tau}_{i,t}^k\right)} \right) \nabla_{\boldsymbol{\theta}} \mathcal{L}_i^t(\boldsymbol{\tau}_{i,t}^k), \quad \boldsymbol{\tau}_{i,t}^{1:K} \sim p_i^t\left(\boldsymbol{\tau} \mid \boldsymbol{\phi}_i^t\right), \tag{5.9}$$

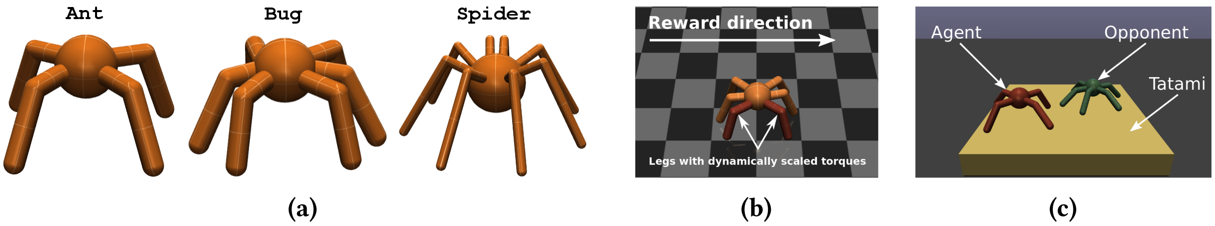


Figure 5.3: Agents and environments used in our experiments. (a) The three types of agents used in experiments. The robots differ in the anatomy: the number of legs, their positions, and constraints on the thigh and knee joints. (b) The nonstationary locomotion environment. The torques applied to red-colored legs are scaled by a dynamically changing factor. (c) RoboSumo environment.

where $\pi_{\phi_i^t}$ is used to rollout from T_i^t. Extending importance weight correction to multi-step updates is straightforward and requires simply adding importance weights to each of the intermediate steps in Equation 5.6.

5.5 Environments

We have designed a set of environments for testing different aspects of continuous adaptation methods in two scenarios: (i) simple environments that change from episode to episode according to some underlying dynamics, and (ii) a competitive multi-agent environment, RoboSumo, that allows different agents to play sequences of games against each other and keep adapting to incremental changes in each other's policies. All our environments are based on MuJoCo physics simulator (Todorov et al., 2012), and all agents are simple multi-leg robots (Figure 5.3a).

5.5.1 Nonstationary Locomotion

First, we consider the problem of robotic locomotion in a changing environment. We use a six-leg agent (Figure 5.3b) that observes the absolute position and velocity of its body, the angles and velocities of its legs, and it acts by applying torques to its joints. The agent is rewarded proportionally to its moving speed in a fixed direction. To induce nonstationarity, we select a pair of legs of the agent and scale down the torques applied to the corresponding joints by a factor that linearly changes from 1 to 0 over the course of 7 episodes. In other words, during the first episode all legs are fully functional, while during the last episode the agent has two legs fully paralyzed (even though the policy can generate torques, they are multiplied by 0 before being passed to the environment). The goal of the agent is to learn to adapt from episode to episode by changing its gait so that it is able to move with a maximal speed in a given direction despite the changes in the environment (cf. Cully et al., 2015). Also, there are 15 ways to select a pair of legs of a six-leg creature which gives us 15 different nonstationary environments. This allows us to use a subset of these environments for training and a separate held out set for testing. The training and testing procedures are described in the next section.

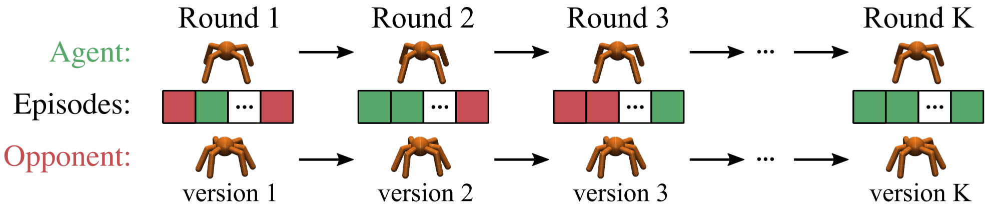


Figure 5.4: An agent competes with an opponent in an iterated adaptation games that consist of multi-episode rounds. The agent wins a round if it wins the majority of episodes (wins and losses illustrated with color). Both agents may update their policies from round to round (denoted by the version number).

5.5.2 Competitive Multi-agent Games

Our multi-agent environment, RoboSumo, allows agents to compete in the 1-vs-1 regime following the standard sumo rules[4]. We introduce three types of agents, Ant, Bug, and Spider, with different anatomies (Figure 5.3a). During the game, each agent observes positions of itself and the opponent, its own joint angles, the corresponding velocities, and the forces exerted on its own body (i.e., equivalent of tactile senses). The action spaces are continuous.

Iterated adaptation games. To test adaptation, we define the *iterated adaptation game* (Figure 5.4)—a game between a pair of agents that consists of K rounds each of which consists of one or more fixed length episodes (500 time steps each). The outcome of each round is either win, loss, or draw. The agent that wins the majority of rounds (with at least 5% margin) is declared the winner of the game. There are two distinguishing aspects of our setup: First, the agents are trained either via pure self-play or versus opponents from a fixed training collection. At test time, they face a new opponent from a testing collection. Second, the agents are allowed to learn (or adapt) at test time. In particular, an agent should exploit the fact that it plays against the same opponent multiple consecutive rounds and try to adjust its behavior accordingly. Since the opponent may also be adapting, the setup allows to test different continuous adaptation strategies, one versus the other.

Reward shaping. In RoboSumo, rewards are naturally sparse: the winner gets +2000, the loser is penalized for -2000, and in case of a draw both opponents receive -1000 points. To encourage fast learning at the early stages of training, we shape the rewards given to agents in the following way: the agent (i) gets reward for staying closer to the center of the ring, for moving towards the opponent, and for exerting forces on the opponent's body, and (ii) gets penalty inversely proportional to the opponent's distance to the center of the ring. At test time, the agents continue having access to the shaped reward as well and may use it to update their policies. Throughout our experiments, we use discounted rewards with the discount factor, $\gamma = 0.995$. More details are in Appendix C.4.2.

Calibration. To study adaptation, we need a well-calibrated environment in which none of the agents has an initial advantage. To ensure the balance, we increased the mass of the weaker agents (Ant and Spider) such that the win rates in games between one agent type versus the other type in the non-adaptation regime became almost equal (for details, see Appendix C.4.3).

[4] To win, the agent has to push the opponent out of the ring or make the opponent's body touch the ground.

5.6 Experiments

Our goal is to test different adaptation strategies in the proposed nonstationary RL settings. However, it is known that the test-time behavior of an agent may highly depend on a variety of factors besides the chosen adaptation method, including training curriculum, training algorithm, policy class, etc. Hence, we first describe the precise setup that we use in our experiments to eliminate irrelevant factors and focus on the effects of adaptation. Most of the low-level details are deferred to appendices. Video highlights of our experiments are available online.

5.6.1 The setup

Policies. We consider 3 types of policy networks: (i) a 2-layer MLP, (ii) embedding (i.e., 1 fully-connected layer replicated across the time dimension) followed by a 1-layer LSTM, and (iii) RL^2 (Duan et al., 2016) of the same architecture as (ii) which additionally takes previous reward and done signals as inputs at each step, keeps the recurrent state throughout the entire interaction with a given environment (or an opponent), and resets the state once the latter changes. For advantage functions, we use networks of the same structure as for the corresponding policies and have no parameter sharing between the two. Our meta-learning agents use the same policy and advantage function structures as the baselines and learn a 3-step meta-update with adaptive step sizes as given in Equation 5.6. Details on the architectures are given in Appendix C.2.

Meta-learning. We compute meta-updates via gradients of the negative discounted rewards received during a number of previous interactions with the environment. At training time, meta-learners interact with the environment twice, first using the initial policy, $\pi_{\boldsymbol{\theta}}$, and then the meta-updated policy, $\pi_{\boldsymbol{\phi}}$. At test time, the agents are limited to interacting with the environment only once, and hence always act according to $\pi_{\boldsymbol{\phi}}$ and compute meta-updates using importance-weight correction (see Section 5.4.2 and Algorithm 5.2). Additionally, to reduce the variance of the meta-updates at test time, the agents store the experience collected during the interaction with the test environment (and the corresponding importance weights) into the experience buffer and keep re-using that experience to update $\pi_{\boldsymbol{\phi}}$ as in Equation 5.6. The size of the experience buffer is fixed to 3 episodes for nonstationary locomotion and 75 episodes for RoboSumo. More details are given in Appendix C.3.1.

Adaptation baselines. We consider the following three baseline strategies:

(i) naive (or no adaptation),

(ii) implicit adaptation via RL^2, and

(iii) adaptation via tracking (Sutton et al., 2007) using PPO at execution time.

Training in nonstationary locomotion. We train all methods on the same collection of nonstationary locomotion environments constructed by choosing all possible pairs of legs whose joint torques are scaled except 3 pairs that are held out for testing (i.e., 12 training and 3 testing environments for the six-leg creature). The agents are trained on the environments concurrently, i.e., to compute a policy update, we rollout from all environments in parallel and then compute, aggregate, and average the gradients (for details, see Appendix C.3.2). LSTM policies retain

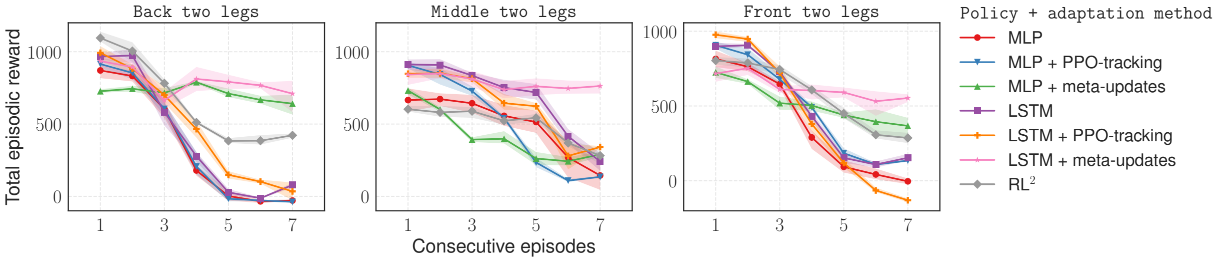


Figure 5.5: Episodic rewards for 7 consecutive episodes in 3 held out nonstationary locomotion environments. We evaluate adaptation strategies in each environment for 7 episodes followed by a full reset of the environment, policy, and meta-updates (repeated 50 times). Shaded regions correspond to 95% CI.

their state over the course of 7 episodes in each environment. Meta-learning agents compute meta-updates for each nonstationary environment separately.

Training in RoboSumo. To ensure consistency of the training curriculum for all agents, we first pre-train a number of policies of each type for every agent type via pure self-play with the PPO algorithm (Schulman et al., 2017; Bansal et al., 2018). We snapshot and save versions of the pre-trained policies at each iteration. This lets us train other agents to play against versions of the pre-trained opponents at various stages of mastery. Next, we train the baselines and the meta-learning agents against the pool of pre-trained opponents[5] concurrently. At each iteration k we (a) randomly select an opponent from the training pool, (b) sample a version of the opponent's policy to be in $[1, k]$ (this ensures that even when the opponent is strong, sometimes an undertrained version is selected which allows the agent learn to win at early stages), and (c) rollout against that opponent. All baseline policies are trained with PPO; meta-learners also used PPO as the outer loop for optimizing $\boldsymbol{\theta}$ and α parameters. We retain the states of the LSTM policies over the course of interaction with the same version of the same opponent and reset it each time the opponent version is updated. Similarly to the locomotion setup, meta-learners compute meta-updates for each opponent in the training pool separately. A more detailed description of the distributed training is given in Appendix C.3.2.

Experimental design. We design our experiments to answer the following questions:

- When the interaction with the environment before it changes is strictly limited to one or very few episodes, what is the behavior of different adaptation methods in nonstationary locomotion and competitive multi-agent environments?
- What is the sample complexity of different methods, i.e., how many episodes is required for a method to successfully adapt to the changes? We test this by controlling the amount of experience the agent is allowed to get form the same environment before it changes.

Additionally, we ask the following questions specific to the competitive multi-agent setting:

[5] In competitive multi-agent environments, besides self-play, there are plenty of ways to train agents, e.g., train them in pairs against each other concurrently, or randomly match and switch opponents each few iterations. We found that concurrent training often leads to an unbalanced population of agents that have been trained under vastly different curricula and introduces spurious effects that interfere with our analysis of adaptation. Hence, we leave the study of adaptation in naturally emerging curricula in multi-agent settings to the future work.

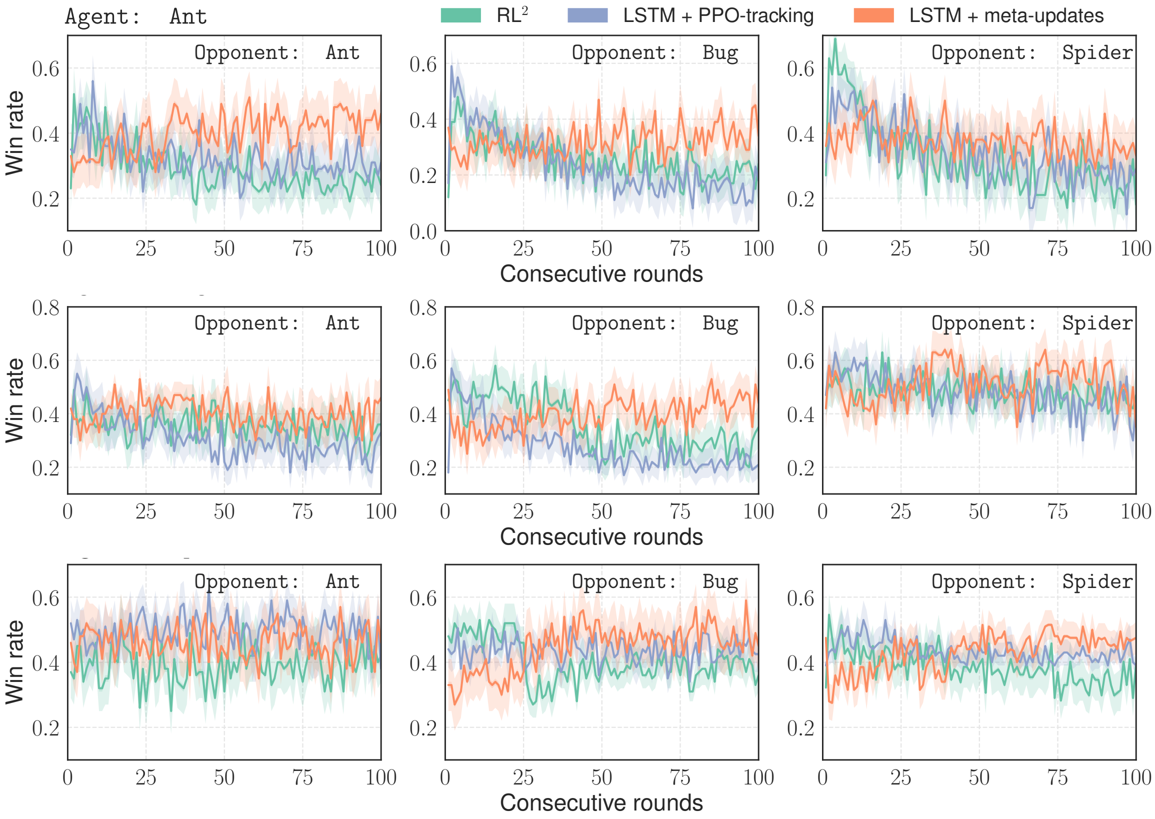


Figure 5.6: Win rates for different adaptation strategies in iterated games versus 3 different pre-trained opponents. At test time, both agents and opponents started from versions 700. Opponents' versions were increasing with each consecutive round as if they were learning via self-play, while agents were allowed to adapt only from the limited experience with a given opponent. Each round consisted of 3 episodes. Each iterated game was repeated 100 times; shaded regions denote bootstrapped 95% CI; no smoothing.

- Given a diverse population of agents that have been trained under the same curriculum, how do different adaptation methods rank in a competition versus each other?
- When the population of agents is evolved for several generations—such that the agents interact with each other via adaptation games, and those that lose disappear while the winners get duplicated—what happens with the population?

5.6.2 Adaptation in the Few-shot Regime and Sample Complexity

Few-shot adaptation in nonstationary locomotion environments. Having trained baselines and meta-learning policies as described in Section 5.6.1, we selected 3 testing environments that corresponded to disabling 3 different pairs of legs of the six-leg agent: back, middle, and front legs. The results are presented on Figure 5.5. Three observations: First, during the very first episode, the meta-learned initial policy, $\pi_{\theta^\star}$, turns out to be suboptimal for the task (it underper-

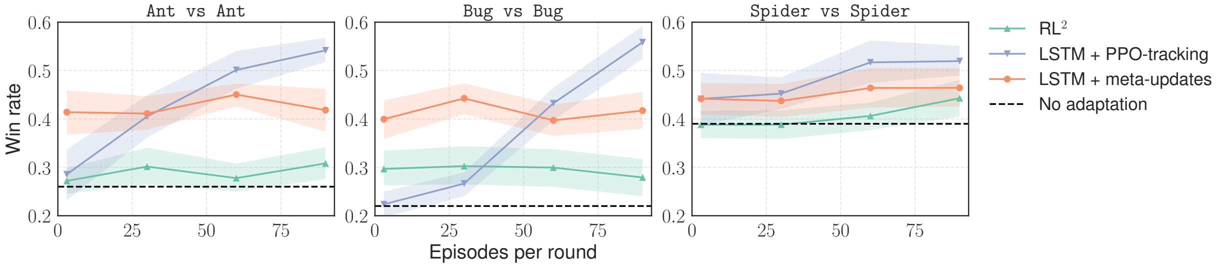


Figure 5.7: The effect of increased number of episodes per round in the iterated adaptation games.

forms compared to other policies). However, after 1-2 episodes (and environment changes), it starts performing on par with other policies. Second, by the 6th and 7th episodes, meta-updated policies perform much better than the rest. Note that we use 3 gradient meta-updates for the adaptation of the meta-learners; the meta-updates are computed based on experience collected during the previous 2 episodes. Finally, tracking is not able to improve upon the baseline without adaptation and sometimes leads to even worse results.

Adaptation in RoboSumo under the few-shot constraint. To evaluate different adaptation methods in the competitive multi-agent setting consistently, we consider a variation of the iterated adaptation game, where changes in the opponent's policies at test time are pre-determined but unknown to the agents. In particular, we pre-train 3 opponents (1 of each type, Figure 5.3a) with LSTM policies with PPO via self-play (the same way as we pre-train the training pool of opponents, see Section 5.6.1) and snapshot their policies at each iteration. Next, we run iterated games between our trained agents that use different adaptation algorithms versus policy snapshots of the pre-trained opponents. Crucially, the policy version of the opponent keeps increasing from round to round as if it was training via self-play[6]. The agents have to keep adapting to increasingly more competent versions of the opponent (see Figure 5.4). This setup allows us to test different adaptation strategies consistently against the same learning opponents.

The results are given on Figure 5.6. We note that meta-learned adaptation strategies, in most cases, are able to adapt and improve their win-rates within about 100 episodes of interaction with constantly improving opponents. On the other hand, performance of the baselines often deteriorates during the rounds of iterated games. Note that the pre-trained opponents were observing 90 episodes of self-play per iteration, while all our agents (baselines and meta-learners) had access to only 3 episodes per round.

Sample complexity of adaptation in RoboSumo. Meta-learning helps to find an update suitable for fast or few-shot adaptation. However, how do different adaptation methods behave when more experience is available? To answer this question, we employ the same setup as previously and vary the number of episodes per round in the iterated game. Each iterated game is repeated 20 times, and we measure the win-rates during the last 25 rounds of the game.

The results are presented on Figure 5.7. When the number of episodes per round goes above 50, adaptation via tracking technically turns into "learning at test time," and it is able

[6] At the beginning of the iterated game, both agents and their opponent start from version 700, i.e., from the policy obtained after 700 iterations (PPO epochs) of learning to ensure that the initial policy is reasonable.

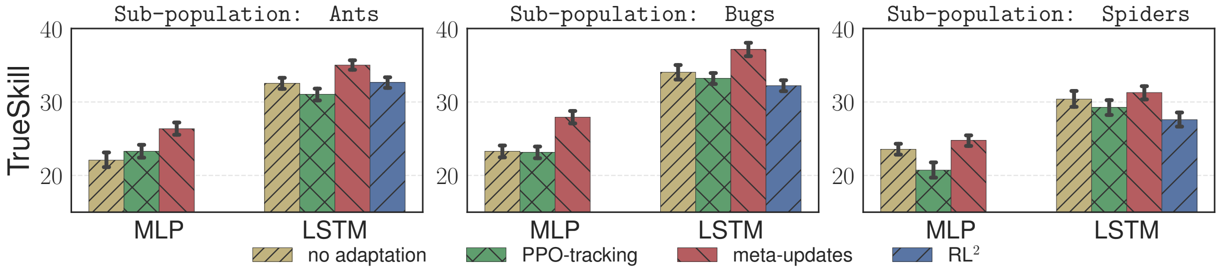


Figure 5.8: TrueSkill for the top-performing MLP- and LSTM-based agents computed based on outcomes (win, loss, or draw) in 1000 iterated adaptation games (100 consecutive rounds per game, 3 episodes per round) between randomly selected pairs of opponents from a population of 105 pre-trained agents.

to learn to compete against the self-trained opponents that it has never seen at training time. The meta-learned adaptation strategy performed near constantly the same in both few-shot and standard regimes. This suggests that the meta-learned strategy acquires a particular bias at training time that allows it to perform better from limited experience but also limits its capacity of utilizing more data. Note that, by design, the meta-updates are fixed to only 3 gradient steps from $\boldsymbol{\theta}^\star$ with step-sizes $\alpha^\star$ (learned at training), while tracking keeps updating the policy with PPO throughout the iterated game. Allowing for meta-updates that become more flexible with the availability of data can potentially help to overcome this limitation.

5.6.3 Evaluation on the Population-level

Combining different adaptation strategies with different policies and agents of different morphologies puts us in a situation where we have a diverse population of agents which we would like to rank according to the level of their mastery in adaptation (or find the "fittest"). To do so, we use TrueSkill (Herbrich et al., 2007)—a metric similar to the ELO rating.

In this experiment, we consider a population of 105 trained agents: 3 agent types, 7 different policy and adaptation combinations, and 5 different stages of training (from 500 to 2000 training iterations). First, we assume that the initial distribution of any agent's skill is $\mathcal{N}(25, 25/3)$ and the default distance that guarantees about 76% of winning, $\beta = 4.1667$. Next, we randomly generate 1000 matches between pairs of opponents and let them adapt while competing with each other in 100-round iterated adaptation games. After each game, we record the outcome and updated our belief about the skill of the corresponding agents using the TrueSkill algorithm[7]. The distributions of the skill for the agents of each type after 1000 iterated adaptation games between randomly selected players from the pool are visualized in Figure 5.8.

There are a few observations we can make: First, recurrent policies were dominant. Second, adaptation via RL^2 tended to perform equally or a little worse than plain LSTM with or without tracking in this setup. Finally, agents that meta-learned adaptation rules at training time, consistently demonstrated higher skill scores in each of the categories corresponding to different policies and agent types.

[7] We used an implementation from http://trueskill.org/.

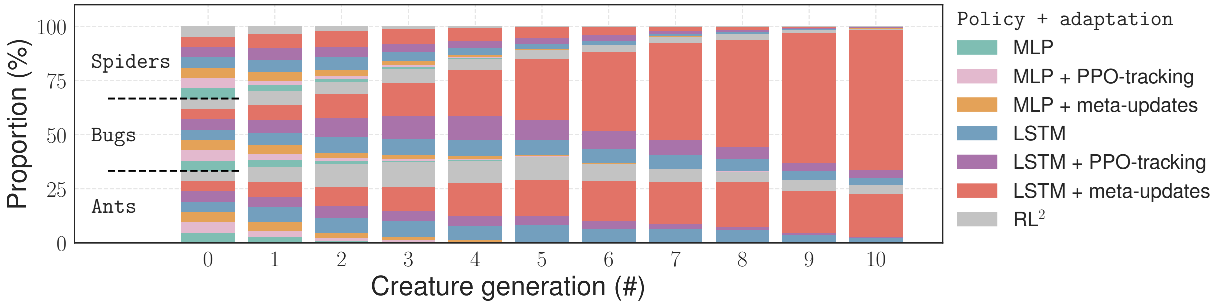


Figure 5.9: Evolution of a population of 1050 agents for 10 generations (see also a video explanation).

Finally, we enlarge the population from 105 to 1050 agents by duplicating each of them 10 times and evolve it (in the "natural selection" sense) for several generations as follows. Initially, we start with a balanced population of different creatures. Next, we randomly match 1000 pairs of agents, make them play iterated adaptation games, remove the agents that lost from the population and duplicate the winners. The same process is repeated 10 times. The result is presented in Figure 5.9. We see that many agents quickly disappear form initially uniform population and the meta-learners end up dominating.

5.7 Conclusion

In this chapter, we considered the problem reinforcement learning and adaptation in nonstationary environments and designed a simple gradient-based meta-learning algorithm that enabled continuous adaptation to Markovian nonstationarity. The key idea of the method is to regard nonstationarity as a sequence of stationary tasks and train agents to exploit the dependencies between consecutive tasks such that they can handle similar nonstationarities at execution time. Our approach was enabled by a probabilistic reformulation of meta-RL (and MAML in particular), which we further extended to distributions over sequences of dependent tasks. Finally, we conducted an extensive empirical study, testing our algorithm in simulation on nonstationary locomotion and on challenging competitive multi-agent games. In both cases, our meta-learned continuous adaptation updates were more efficient than the baselines in the few-shot regime. Additionally, agents that meta-learned to adapt demonstrated the highest level of skill when competing in iterated games against each other.

5.7.1 Limitations

The problem of continuous adaptation in nonstationary and competitive environments is far from being solved, and this work is the first attempt to use meta-learning in a such setup. Our meta-learning algorithm has a few limiting assumptions and design choices, many of which have been made either due to simplicity or computational considerations. First, our continuous adaptation update (Equation 5.6) is based on only one step lookahead and is computationally similar to backpropagation through time with a unit time lag. This could potentially be extended to fully recurrent meta-updates that take into account the full history of interaction with the changing environment. Additionally, our meta-updates were based on the gradients of a surrogate loss function. While such updates explicitly optimized the loss, they required computing second order derivatives at training time, slowing down the training process by an order of magnitude compared to baselines. Utilizing information provided by the loss but avoiding explicit backpropagation through the gradients would be more appealing and scalable. Finally, our approach is unlikely to work with sparse rewards as the meta-updates use policy gradients and heavily rely on the reward signal. Introducing auxiliary dense rewards designed to enable meta-learning is a potential way to overcome this issue.

5.7.2 Notable Follow-up Work

Since publication of the original paper that this chapter is based on (Al-Shedivat et al., 2018a), multiple work have made further progress on reinforcement learning under nonstationarity, either building on and/or extending ideas presented here in many interesting ways, or proposing complementary approaches. We highlight a few notable directions that the author is aware of:

- While our work has been focused on *model-free* RL (and policy gradient methods in particular), Nagabandi et al. (2018a) and Nagabandi et al. (2018b) proposed (online) meta-learning methods for continuous adaptation in the context of *model-based* RL.
- Chandak et al. (2020) proposed a complementary (non-meta-learning) approach for solving sequences of dependent MDPs using regret minimization and showed that their method can compensate for the performance lag, which meta-learning does not fully address.
- Probabilistic (or Bayesian) interpretation of MAML (which we discussed more broadly within our PMM framework in Chapter 3) was proposed concurrently for supervised learning by Grant et al. (2018) and for reinforcement learning by Al-Shedivat et al. (2018a). Using such probabilistic formulation, Finn et al. (2018) and Yoon et al. (2018) extended MAML beyond point estimate approximation of the local posterior distributions, and showed that representing uncertainty can be useful for solving ambiguous few-shot learning problems, improving robustness, and enabling active learning at adaptation time.
- Finally, while we considered multi-agent games as a benchmark in this work, our evaluation was centered on adaptation of a single agent, treating opponents as a nonstationary part of the environment. Kim et al. (2020) extended our continuous adaptation method to fully multi-agent setting and applied it to competitive and cooperative games.

Chapter 6

Consistent Zero-shot Generalization

6.1 Summary

In this chapter, we consider the problem of multilingual machine translation, focusing on zero-shot generalization—a challenging setup that tests models on translation directions they have not been optimized for at training time. First, we reformulate multilingual translation as probabilistic inference and show how to represent multiple translation tasks using our PMM framework. Then, we define the notion of zero-shot consistency and show why standard training often results in models unsuitable for unseen tasks. Finally, we introduce a zero-shot consistent training method, which encourages the model to produce equivalent translations of parallel sentences in auxiliary languages, effectively optimizing a upper bound on the log likelihood of the joint multilingual probabilistic model. Our method yields significant improvements in performance (up to 2-3 BLEU points, or +20% of relative improvement) over strong baselines without any loss in performance on supervised translation directions.

6.2 Motivation and Goals

Machine Translation (MT) has made remarkable advances with the advent of deep learning approaches (Bojar et al., 2016; Wu et al., 2016; Crego et al., 2016; Junczys-Dowmunt et al., 2016). The progress was largely driven by the encoder-decoder framework (Sutskever et al., 2014; Cho et al., 2014), typically supplemented with an attention mechanism (Bahdanau et al., 2014; Luong et al., 2015b). Compared to the traditional phrase-based MT systems (Koehn, 2009), Neural Machine Translation (NMT) requires very large amounts of data in order to reach high performance (Koehn and Knowles, 2017). Using NMT in a multilingual setting exacerbates the problem by the fact that given k languages translating between all pairs would require $O(k^2)$ parallel training corpora (and $O(k^2)$ models, if everything trained separately).

In an effort to address the problem, multiple different multilingual NMT approaches have been proposed. Luong et al. (2015a) and Firat et al. (2016a) proposed to use $O(k)$ encoders/decoders

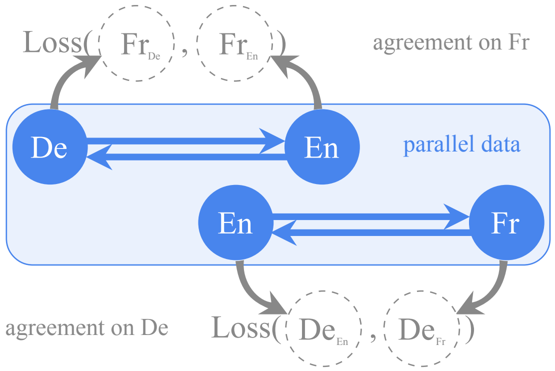


Figure 6.1: Illustration of agreement-based training of a multilingual NMT. At training time, given English-French (En ↔ Fr) and English-German (En ↔ De) parallel sentences, the model not only is trained to translate between the pair but also to agree on translations into a third language.

that are then intermixed to translate between language pairs. Johnson et al. (2016) proposed to use a single model and prepend special symbols to the source text to indicate the target language, which has later been extended to other text preprocessing approaches (Ha et al., 2017) as well as used in conjunction with language-conditional parameter generation (Platanios et al., 2018).

Johnson et al. (2016) also showed that a single multilingual system could potentially enable *zero-shot* translation, *i.e.*, translate between language pairs not seen in training. For example, given 3 languages—German (De), English (En), and French (Fr)—and parallel data only for (De, En) and (En, Fr), at test time, the system could additionally translate between (De, Fr).

Zero-shot translation is an important problem. Solving the problem could significantly improve data efficiency—a single multilingual model would be able to generalize and translate between any of the $O(k^2)$ language pairs after being trained only on $O(k)$ parallel corpora. However, zero-shot performance of NMT has been observed to be unstable and significantly lagging behind the supervised directions. Moreover, attempts to improve zero-shot performance by fine-tuning often negatively impact other directions (Firat et al., 2016b; Sestorain et al., 2018).

In this chapter, we take a different approach: instead of designing new architectures for NMT, we aim to improve the *training procedure* of Johnson et al. (2016). First, we analyze multilingual translation problem from a probabilistic perspective and define the notion of *zero-shot consistency* that gives insights as to why the vanilla training method may not yield models with good zero-shot performance. Next, we propose a novel training objective and a modified learning algorithm that achieves consistency via agreement-based learning (Liang et al., 2006; Liang et al., 2008) and improves zero-shot translation. Our training method encourages the model to produce equivalent translations of parallel training sentences into an auxiliary language (Figure 6.1) and provably leads to zero-shot generalization, as if we were to maximize the intractable joint log likelihood (see Theorem 3.1 in Section 3.3.2). In addition, we make a simple change to the neural decoder to make the agreement losses fully differentiable, which enables end-to-end training.

We demonstrate the efficacy of our approach experimentally on IWSLT17 (Mauro et al., 2017), UN corpus (Ziemski et al., 2016), and Europarl (Koehn, 2017), carefully removing complete pivots from the training corpora. Agreement-based learning results in up to +3 BLEU zero-shot improvement over the baseline, compares favorably (up to +2.4 BLEU) to other approaches in the literature (Cheng et al., 2017; Sestorain et al., 2018), is competitive with pivoting, and does not lose in performance on supervised directions.

6.3 Preliminaries

We start by establishing more formal notation and briefly reviewing some background on encoder-decoder multilingual machine translation from a probabilistic point of view.

6.3.1 Notation

Languages. We assume that we are given a fixed collection of k languages, $L_1, \dots, L_k$, that share a common vocabulary, V. A language, L_i, is defined by the marginal probability $p(\mathbf{x}_i)$ it assigns to sentences (*i.e.*, sequences of tokens from the vocabulary), denoted $\mathbf{x}_i := (x^1, \dots, x^l)$, where l is the length of the sequence. All languages together define a joint probability distribution, $p(\mathbf{x}_1, \dots, \mathbf{x}_k)$, over k-tuples of *equivalent sentences*.[1]

Corpora. While each sentence may have an equivalent representation in all languages, we assume that we have access to only partial sets of equivalent sentences, which form *corpora*. In this work, we consider *bilingual* corpora, denoted $\mathcal{C}_{ij}$, that contain pairs of sentences sampled from $p(\mathbf{x}_i, \mathbf{x}_j)$ and *monolingual* corpora, denoted $\mathcal{C}_i$, that contain sentences sampled from $p(\mathbf{x}_i)$.

Translation. Finally, we define a *translation task* from language L_i to L_j as learning to model the conditional distribution $p(\mathbf{x}_j \mid \mathbf{x}_i)$. The set of k languages along with translation tasks can be represented as a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with a set of k nodes, $\mathcal{V}$, that represent languages and edges, $\mathcal{E}$, that indicate translation directions. We further distinguish between two disjoint subsets of edges: (i) supervised edges, $\mathcal{E}_s$, for which we have parallel data, and (ii) zero-shot edges, $\mathcal{E}_0$, that correspond to zero-shot translation tasks. Figure 6.2 presents an example translation graph with supervised edges (En ↔ Es, En ↔ Fr, En ↔ Ru) and zero-shot edges (Es ↔ Fr, Es ↔ Ru, Fr ↔ Ru). We will use this graph as our running example.

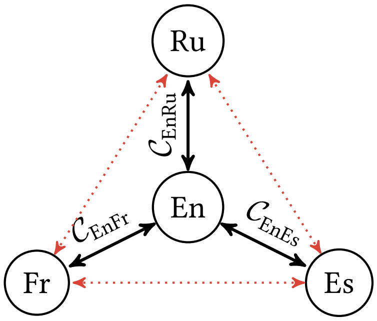

Ru
$\mathcal{C}_{\text{EnRu}}$
En
$\mathcal{C}_{\text{EnFr}}$
$\mathcal{C}_{\text{EnEs}}$
Fr
Es

Figure 6.2: Illustration of a translation graph for En, Es, Fr, Ru.

6.3.2 Encoder-decoder Models

Consider a purely bilingual setting, where we learn to translate from a source language L_s to a target language L_t. We can train a translation model by optimizing the conditional log-likelihood of the bilingual data under the model:

$$\hat{\boldsymbol{\theta}} := \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\theta}} \sum_{\mathbf{x}_s, \mathbf{x}_t \in \mathcal{C}_{st}} -\log p(\mathbf{x}_t \mid \mathbf{x}_s, \boldsymbol{\theta}) \tag{6.1}$$

where $\hat{\boldsymbol{\theta}}$ are the estimated parameters of the model. The encoder-decoder framework introduces a latent sequence, $\mathbf{u}$, and represents the model as:

$$p(\mathbf{x}_t \mid \mathbf{x}_s, \boldsymbol{\theta}) := p_{\boldsymbol{\theta}}^{\text{dec}}(\mathbf{x}_t \mid \mathbf{u} = f_{\boldsymbol{\theta}}^{\text{enc}}(\mathbf{x}_s)) \tag{6.2}$$

[1] We say that sentences are equivalent if they carry the same meaning.

where $f_{\boldsymbol{\theta}}^{\text{enc}}(\mathbf{x}_s)$ is the encoder that maps a source sequence to a sequence of latent representations, $\mathbf{u}$, and the decoder defines $p_{\boldsymbol{\theta}}^{\text{dec}}\left(\mathbf{x}_t \mid \mathbf{u}\right)$.[2] Note that $\mathbf{u}$ is usually deterministic with respect to $\mathbf{x}_s$ and accurate representation of the conditional distribution highly depends on the decoder. In neural machine translation, the exact forms of encoder and decoder are specified using RNNs (Sutskever et al., 2014), CNNs (Gehring et al., 2016), and attention (Bahdanau et al., 2014; Vaswani et al., 2017) as building blocks. The decoding distribution $p_{\boldsymbol{\theta}}^{\text{dec}}\left(\mathbf{x}_t \mid \mathbf{u}\right)$ is typically modeled autoregressively, *i.e.*, $p_{\boldsymbol{\theta}}^{\text{dec}}\left(\mathbf{x}_t \mid \mathbf{u}\right) = \prod_i p_{\boldsymbol{\theta}}^{\text{dec}}\left(\mathbf{x}_t^i \mid \mathbf{u}, \mathbf{x}_t^{0:i-1}\right)$.

6.3.3 Multilingual Neural Machine Translation

In multilingual settings, we would like to learn to translate in *all directions* having access to only few parallel bilingual corpora. In other words, we would like to learn a collection of models, $\{p\left(\mathbf{x}_j \mid \mathbf{x}_i, \boldsymbol{\theta}\right)\}_{i,j\in\mathcal{E}}$. We can assume that models are independent and choose to learn them by maximizing the following objective:

$$\mathcal{L}^{\text{ind}}(\boldsymbol{\theta}) = \sum_{(i,j)\in\mathcal{E}_s} \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{C}_{ij}} - \log p\left(\mathbf{x}_j \mid \mathbf{x}_i, \boldsymbol{\theta}\right) \tag{6.3}$$

In the statistics literature, this estimation approach is called *maximum composite likelihood* (Besag, 1975; Lindsay, 1988) as it composes the objective out of (sometimes weighted) terms that represent conditional sub-likelihoods (in our example, $p\left(\mathbf{x}_j \mid \mathbf{x}_i, \boldsymbol{\theta}\right)$). Composite likelihoods are easy to construct and tractable to optimize as they do not require representing the full likelihood, which would involve integrating out variables unobserved in the data (as we mentioned in Section 3.3.2).

Johnson et al. (2016) proposed to train a multilingual NMT systems by optimizing a composite likelihood objective (Equation 6.3) while representing all conditional distributions, $p\left(\mathbf{x}_j \mid \mathbf{x}_i, \boldsymbol{\theta}\right)$, with a *shared* encoder and decoder and using language tags, l_t, to distinguish between translation directions:

$$p\left(\mathbf{x}_t \mid \mathbf{x}_s\right) = p_{\boldsymbol{\theta}}^{\text{dec}}\left(\mathbf{x}_t \mid \mathbf{u}_{st} = f_{\boldsymbol{\theta}}^{\text{enc}}(\mathbf{x}_s, l_t)\right) \tag{6.4}$$

This approach has numerous advantages including: (a) the simplicity of training and the architecture (by slightly changing the training data, we seamlessly convert a bilingual NMT model into a multilingual one), and (b) sharing parameters of the model between different translation tasks that may lead to better and more robust representations.[3] Johnson et al. (2016) also showed that the resulting models seem to exhibit some degree of zero-shot generalization enabled by the parameter sharing. However, since we lack data for zero-shot directions, composite likelihood (Equation 6.3) misses the terms that correspond to the zero-shot models, and hence has no guarantees what so ever for performance on zero-shot tasks.[4]

[2] Slightly abusing the notation, we use $\boldsymbol{\theta}$ to denote all parameters of the model: embeddings, encoder, decoder.

[3] It may also lead to destructive interference and negative transfer between the translation tasks, which Johnson et al. (2016) did not observe in practice.

[4] In fact, since the objective in Equation 6.3 assumes that the models are independent, plausible zero-shot performance would be more indicative of the limited capacity of the model or artifacts in the data (*e.g.*, presence of multi-parallel sentences that the model picks up) rather than consistent zero-shot generalization.

6.4 Approach

Multilingual translation is a multitask learning problem. Thus we can define the notion of zero-shot generalization (see Definition 3.5 in Section 3.2.2) and evaluate multilingual MT systems in terms of *zero-shot performance*, or quality of translation along the directions they have not been optimized for (*e.g.*, due to lack of data).

Recall that we said that a learning algorithm generalizes if the corresponding generalization error vanishes with the increase in the amount of training data. Here, we further refine that definition and introduce the notion of zero-shot consistency of a learning algorithm.

Definition 6.1: Zero-shot Consistency

Let $\mathcal{E}_s$ and $\mathcal{E}_0$ be supervised and zero-shot translation tasks, respectively. Let $\ell(\cdot)$ be a non-negative loss function and $\mathcal{M}$ be a model with maximum expected supervised loss bounded by some $\varepsilon > 0$, *i.e.*:

$$\max_{(i,j)\in\mathcal{E}_s} \mathbb{E}_{\mathbf{x}_i,\mathbf{x}_j}\left[\ell(\mathcal{M})\right] < \varepsilon$$

We call $\mathcal{M}$ zero-shot consistent with respect to $\ell(\cdot)$ if there exists a function $\kappa(\cdot)$ such that:

$$\max_{(i,j)\in\mathcal{E}_0} \mathbb{E}_{\mathbf{x}_i,\mathbf{x}_j}\left[\ell(\mathcal{M})\right] < \kappa(\varepsilon)$$

where $\kappa(\varepsilon) > 0$ and $\kappa(\varepsilon) \to 0$ as $\varepsilon \to 0$.

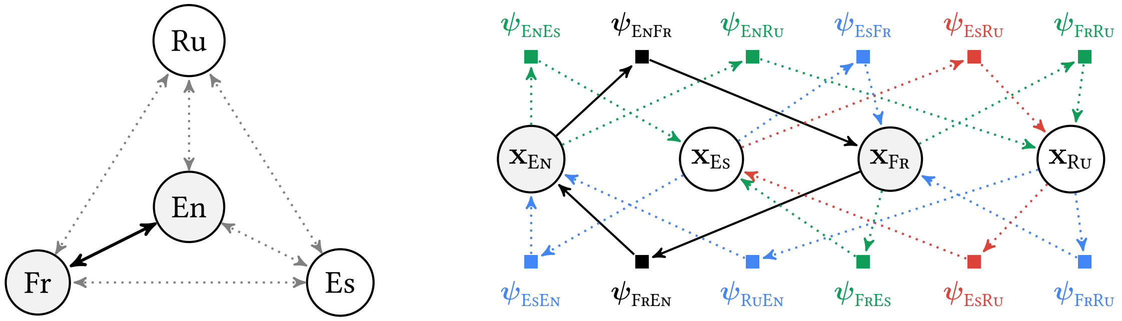
In other words, we say that an MT system is zero-shot consistent if low error on supervised tasks implies a low error on zero-shot tasks in expectation. This notion of consistency somewhat resembles error bounds in the domain adaptation literature (Ben-David et al., 2010).

Note that training NMT systems on more supervised data is known to reduce the error on the corresponding supervised translation directions (*i.e.*, $\varepsilon \to 0$). In practice, it is attractive to have NMT systems that are zero-shot consistent, which means that we can improve their zero-shot performance without having to collect data for the zero-shot translation tasks, which could be expensive or unavailable. While the training method of Johnson et al. (2016) does not guarantee zero-shot consistency, we will show that our proposed approach does.

6.4.1 Composite vs. Joint Likelihood

Following the PMM framework introduced in Chapter 3, we can define a join probabilistic model that represents the distribution over tuples of k equivalent sentences as follows:

$$p\left(\mathbf{x}_1, \ldots, \mathbf{x}_k \mid \boldsymbol{\theta}\right) := \frac{1}{Z} \prod_{(i,j)\in\mathcal{E}} p\left(\mathbf{x}_j \mid \mathbf{x}_i, \boldsymbol{\theta}\right) = \frac{1}{Z} \exp\left\{\sum_{(i,j)\in\mathcal{E}} \psi_{ij}(\mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\theta})\right\} \tag{6.5}$$



(a) translation graph

(b) factor graph

Figure 6.3: Translation graph (a) and the corresponding factor graph (b) of the joint probabilistic model. The En-Fr language pair is observed, and the corresponding translation directions En ↔ Fr are denoted with black solid arrows. Translation directions from an observed language (En or Fr) to an unobserved language (Es or Ru) are denoted in green; the reverse (unobserved to observed) are denoted in blue; directions between unobserved languages are denoted in red. Dependencies of the factors on the parameters $\boldsymbol{\theta}$ are omitted for clarity. Best viewed in color.

Again, consider our running example in Figure 6.3a, which factor graph representation is given in Figure 6.3b. Since the supervision is available only for pairs of languages at a time (*e.g.*, En-Fr), the log marginal likelihood objective under this model takes the following form:

$$\begin{aligned}
&-\mathcal{L}^{\star}(\boldsymbol{\theta}) \\
&\quad= \log p\left(\mathbf{x}_{\text{En}}, \mathbf{x}_{\text{Fr}} \mid \boldsymbol{\theta}\right) \\
&\quad= \log \sum_{\mathbf{x}_{\text{Es}}, \mathbf{x}_{\text{Ru}}} p\left(\mathbf{x}_{\text{En}}, \mathbf{x}_{\text{Es}}, \mathbf{x}_{\text{Fr}}, \mathbf{x}_{\text{Ru}} \mid \boldsymbol{\theta}\right) \\
&\quad= \text{const} + \log p\left(\mathbf{x}_{\text{En}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right) + \log p\left(\mathbf{x}_{\text{Fr}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) + \\
&\qquad \log \sum_{\mathbf{x}_{\text{Es}}, \mathbf{x}_{\text{Ru}}} p\left(\mathbf{x}_{\text{Es}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{Ru}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{Es}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{Ru}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right) \times \\
&\qquad\qquad p\left(\mathbf{x}_{\text{En}} \mid \mathbf{x}_{\text{Es}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{En}} \mid \mathbf{x}_{\text{Ru}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{Fr}} \mid \mathbf{x}_{\text{Es}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{Fr}} \mid \mathbf{x}_{\text{Ru}}, \boldsymbol{\theta}\right) \times \\
&\qquad\qquad p\left(\mathbf{x}_{\text{Es}} \mid \mathbf{x}_{\text{Ru}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{Ru}} \mid \mathbf{x}_{\text{Es}}, \boldsymbol{\theta}\right)
\end{aligned} \tag{6.6}$$

As in Figure 6.3, we denoted terms that correspond to translation directions with only one observed sequences in blue and those that have both sequences unobserved in red.

The standard objective $\mathcal{L}^{\text{ind}}(\boldsymbol{\theta})$ (Equation 6.3) includes only two factors with both observed input and output sequences and ignores the rest. Thus, increasing the amount of training data for En-Fr pair and minimizing $\mathcal{L}^{\text{ind}}(\boldsymbol{\theta})$ can certainly improve performance on the supervised tasks En ↔ Fr, but since $\mathcal{L}^{\text{ind}}(\boldsymbol{\theta})$ is not sensitive to other directions, it does not guarantee zero-shot consistency of the resulting model. On the other hand, thanks to the additional terms, $\mathcal{L}^{\star}(\boldsymbol{\theta})$ is sensitive to the quality of *all* translation directions. However, $\mathcal{L}^{\star}(\boldsymbol{\theta})$ is intractable to compute since the summation is taken over *all probable* sentences in Es and Ru.

In the following section, we design a tractable objective that upper bounds $\mathcal{L}^{\star}(\boldsymbol{\theta})$, ignoring the terms marked blue and red and approximating the summation of the terms denoted in green.

6.4.2 Agreement-based Likelihood

In this section, we derive a new objective function, called *agreement-based likelihood*. Consider the composite likelihood $\mathcal{L}^{\text{ind}}_{\text{EnFr}}(\boldsymbol{\theta})$ (Equation 6.3) for a pair of En-Fr sentences:

$$\begin{aligned}\mathcal{L}^{\text{ind}}_{\text{EnFr}}(\boldsymbol{\theta}) &= -\log\left[p\left(\mathbf{x}_{\text{Fr}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{En}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)\right] \\ &= -\log\left[\sum_{\mathbf{z}'_{\text{Es}}, \mathbf{z}'_{\text{Ru}}} p\left(\mathbf{x}_{\text{Fr}}, \mathbf{z}'_{\text{Es}}, \mathbf{z}'_{\text{Ru}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) \times \sum_{\mathbf{z}''_{\text{Es}}, \mathbf{z}''_{\text{Ru}}} p\left(\mathbf{x}_{\text{En}}, \mathbf{z}''_{\text{Es}}, \mathbf{z}''_{\text{Ru}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)\right]\end{aligned} \tag{6.7}$$

where we introduced latent translations into Spanish (Es) and Russian (Ru) and marginalized them out (by virtually summing over all sequences in the corresponding languages). Again, note that this objective still assumes independence of En → Fr and Fr → En models, even though the models share parameters $\boldsymbol{\theta}$.

Following Liang et al. (2008), we propose to tie together the single prime and the double prime latent variables, $\mathbf{z}_{\text{Es}}$ and $\mathbf{z}_{\text{Ru}}$, to encourage agreement between $p\left(\mathbf{x}_{\text{En}}, \mathbf{z}_{\text{Es}}, \mathbf{z}_{\text{Ru}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)$ and $p\left(\mathbf{x}_{\text{Fr}}, \mathbf{z}_{\text{Es}}, \mathbf{z}_{\text{Ru}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right)$ on the latent translations. We interchange the sum and the product operations inside the $\log$ in Equation 6.7, denote $\mathbf{z} := \left(\mathbf{z}_{\text{Es}}, \mathbf{z}_{\text{Ru}}\right)$ to simplify notation, and arrive at the following new objective function:

$$\mathcal{L}^{\text{agree}}_{\text{EnFr}}(\boldsymbol{\theta}) := -\log\sum_{\mathbf{z}} p\left(\mathbf{x}_{\text{Fr}}, \mathbf{z} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) p\left(\mathbf{x}_{\text{En}}, \mathbf{z} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right) \tag{6.8}$$

Next, we factorize each term as $p\left(\mathbf{x}, \mathbf{z} \mid \mathbf{y}\right) = p\left(\mathbf{x} \mid \mathbf{z}, \mathbf{y}\right) p\left(\mathbf{z} \mid \mathbf{y}\right)$. Assuming $p\left(\mathbf{x}_{\text{Fr}} \mid \mathbf{z}, \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) \approx p\left(\mathbf{x}_{\text{Fr}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right)$,[5] the objective in Equation 6.8 decomposes into two terms:

$$\begin{aligned}\mathcal{L}^{\text{agree}}_{\text{EnFr}}(\boldsymbol{\theta}) \approx\ & -\underbrace{\log p\left(\mathbf{x}_{\text{Fr}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) - \log p\left(\mathbf{x}_{\text{En}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)}_{\text{composite likelihood terms}} \\ & \underbrace{-\log\sum_{\mathbf{z}} p\left(\mathbf{z} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) p\left(\mathbf{z} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)}_{\text{agreement term}}\end{aligned} \tag{6.9}$$

We call the expression given in Equation 6.9 the negative agreement-based likelihood. Intuitively, this objective is the likelihood of observing parallel sentences $\left(\mathbf{x}_{\text{En}}, \mathbf{x}_{\text{Fr}}\right)$ *and* having sub-models $p\left(\mathbf{z} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right)$ and $p\left(\mathbf{z} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)$ agree on all translations into Es and Ru at the same time.

Upper bound. Summation in the agreement term over $\mathbf{z}$ (*i.e.*, over all probable translations into Es and Ru in our case) is intractable. Switching back from $\mathbf{z}$ to $\left(\mathbf{z}_{\text{Es}}, \mathbf{z}_{\text{Ru}}\right)$ notation and using Jensen's inequality, we upper bound it with cross-entropy:[6]

$$\begin{aligned}& -\log\sum_{\mathbf{z}} p\left(\mathbf{z} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right) p\left(\mathbf{z} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right) \\ & \quad \leq -\mathbb{E}_{\mathbf{z}_{\text{Es}} \mid \mathbf{x}_{\text{En}}}\left[\log p\left(\mathbf{z}_{\text{Es}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)\right] - \mathbb{E}_{\mathbf{z}_{\text{Ru}} \mid \mathbf{x}_{\text{En}}}\left[\log p\left(\mathbf{z}_{\text{Ru}} \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right)\right]\end{aligned} \tag{6.10}$$

[5] This means that it is sufficient to condition on a sentence in one of the languages to determine probability of a translation in any other language.

[6] Note that the expectations in Equation 6.10 are conditional on $\mathbf{x}_{\text{En}}$. Symmetrically, we can have an upper bound with expectations conditional on $\mathbf{x}_{\text{Fr}}$. In practice, we symmetrize the objective.

We can estimate the expectations in the upper bound on the agreement terms by sampling $\mathbf{z}_{\text{Es}} \sim p\left(\mathbf{z}_{\text{Es}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right)$ and $\mathbf{z}_{\text{Ru}} \sim p\left(\mathbf{z}_{\text{Ru}} \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right)$. In practice, instead of sampling we use greedy, continuous decoding (with a fixed maximum sequence length), which also makes $\mathbf{z}_{\text{Es}}$ and $\mathbf{z}_{\text{Ru}}$ differentiable with respect to parameters of the model.

To summarize, we have introduced the agreement-based likelihood which complements the composite likelihood objective with an agreement term. The obtained loss $\mathcal{L}^{\text{agree}}$ upper bounds the full negative log likelihood objective $\mathcal{L}^{\star}$ (see Appendix D.1). To make it computable, we further upper bound it with cross-entropy.

6.4.3 Consistency by Agreement

We argue that models produced by maximizing agreement-based likelihood are zero-shot consistent. Informally, consider again our running example from Figure 6.3. Given a pair of parallel sentences in (En, Fr), agreement loss encourages translations from En to $\{\text{Es}, \text{Ru}\}$ and translations from Fr to $\{\text{Es}, \text{Ru}\}$ to coincide. Note that $\text{En} \to \{\text{Es}, \text{Fr}, \text{Ru}\}$ are supervised directions, *i.e.*, we have data for these directions. Therefore, agreement ensures that translations along the zero-shot edges in the graph match supervised translations. Formally, we state it as follows.

Theorem 6.1: Zero-shot Consistency of the Agreement-based Learning

Let L_1, L_2, and L_3 be a collection of languages with $L_1 \leftrightarrow L_2$ and $L_2 \leftrightarrow L_3$ be supervised while $L_1 \leftrightarrow L_3$ be zero-shot directions. Let $p\left(\mathbf{x}_j \mid \mathbf{x}_i, \boldsymbol{\theta}\right)$ be sub-models represented by a multilingual MT model. If the expected agreement-based loss, $\mathbb{E}_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3}\left[\mathcal{L}_{12}^{\text{agree}}(\boldsymbol{\theta}) + \mathcal{L}_{23}^{\text{agree}}(\boldsymbol{\theta})\right]$, is bounded by some $\varepsilon > 0$, then, under some mild technical assumptions on the true distribution of the equivalent translations, the zero-shot cross-entropy loss $\mathbb{E}_{\mathbf{x}_1,\mathbf{x}_3}\left[-\log p_{\boldsymbol{\theta}}\left(\mathbf{x}_3 \mid \mathbf{x}_1\right)\right]$ is bounded, too, by $\kappa(\varepsilon) > 0$, where $\kappa(\varepsilon) \to 0$ as $\varepsilon \to 0$.

A precise expression for $\kappa(\varepsilon)$ along with a detailed proof are given in Appendix D.2. Note that Theorem 6.1 is straightforward to extend from triplets of languages to arbitrary connected graphs, as given in the following corollary.

Corollary 6.1. *Agreement-based learning yields zero-shot consistent translation models (w.r.t. cross-entropy) for arbitrary translation graphs as long as the supervised directions span the graph.*

Alternative ways to ensure consistency. Note that there are other ways to ensure zero-shot consistency, *e.g.*, by fine-tuning or post-processing a trained multilingual model. For instance, *pivoting*, or translating through an intermediate (pivot) language, is also zero-shot consistent, but the proof requires stronger assumptions about the quality of the supervised source-pivot model.[7] Similarly, using model distillation (Kim and Rush, 2016; Chen et al., 2017) would be also provably consistent under the same assumptions as given in Theorem 6.1, but for a single, pre-selected zero-shot direction. Note that our proposed agreement-based learning framework is provably consistent for *all* zero-shot directions and does not require any post-processing. For discussion of the alternative approaches and consistency proof for pivoting, see Appendix D.3.

[7] Intuitively, we have to assume that source-pivot model does not assign high probabilities to unlikely translations as the pivot-target model may react to those unpredictably.

Algorithm 6.1 Agreement-based multilingual NMT training

```
input  Architecture (GNMT), agreement coefficient ($\gamma$)
Initialize: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$
while not (converged or step limit reached) do
    Get a mini-batch of parallel source-target pairs, $(\mathbf{X}_s, \mathbf{X}_t)$
    Compute the supervised cross-entropy loss: $\mathcal{L}^{\text{sup}}(\boldsymbol{\theta}) \leftarrow \log p\left(\mathbf{X}_t \mid \mathbf{X}_s, \boldsymbol{\theta}\right)$
    // compute agreement loss
    Sample the auxiliary language: $L_a \sim \text{Unif}(\{1, \dots, k\})$
    Compute auxiliary translations:
        $\mathbf{Z}_{a\leftarrow s} \leftarrow \text{Decode}\left(\mathbf{Z}_a \mid f^{\text{enc}}_{\boldsymbol{\theta}}(\mathbf{X}_s, L_a)\right), \quad \mathbf{Z}_{a\leftarrow t} \leftarrow \text{Decode}\left(\mathbf{Z}_a \mid f^{\text{enc}}_{\boldsymbol{\theta}}(\mathbf{X}_t, L_a)\right)$
    Compute agreement log-probabilities:
        $\ell^t_{a\leftarrow s} \leftarrow \log p\left(\mathbf{Z}_{a\leftarrow s} \mid \mathbf{X}_t, \boldsymbol{\theta}\right), \quad \ell^s_{a\leftarrow t} \leftarrow \log p\left(\mathbf{Z}_{a\leftarrow t} \mid \mathbf{X}_s, \boldsymbol{\theta}\right)$
    Apply stop-gradients to $\ell^t_{a\leftarrow s}$ or $\ell^s_{a\leftarrow t}$ if either correspond to a supervised direction
    // update parameters
    Compute the total loss: $\mathcal{L}^{\text{total}}(\boldsymbol{\theta}) \leftarrow \mathcal{L}^{\text{sup}}(\boldsymbol{\theta}) + \gamma(\ell^t_{a\leftarrow s} + \ell^s_{a\leftarrow t})$
    Update: $\boldsymbol{\theta} \leftarrow \text{optimizer\_update}(\mathcal{L}^{\text{total}}, \boldsymbol{\theta})$
end while
output  $\boldsymbol{\theta}$
```

6.4.4 A Practical Learning Algorithm

Having derived a new objective function (Equation 6.9), we can now learn consistent multilingual NMT models using stochastic gradient method with a couple of extra tricks (Algorithm 6.1). The computation graph for the agreement loss is given in Figure 6.4. We note a few critical points:

Subsampling auxiliary languages. Computing agreement over *all* languages for each pair of sentences at training time would be quite computationally expensive (to agree on k translations, we would need to encode-decode the source and target sequences k times each). However, since the agreement upper bound is a sum over expectations (Equation 6.10), we can approximate it by subsampling: at each training step (and for each sample in the mini-batch), we pick an auxiliary language uniformly at random and compute stochastic approximation of the agreement upper bound for that language only. This stochastic approximation is simple, unbiased, and reduces the computational overhead per step for the agreement term from $O(k)$ to $O(1)$.[8]

Overview of the agreement loss computation. Given a pair of parallel sentences, $\mathbf{x}_{\text{En}}$ and $\mathbf{x}_{\text{Fr}}$, and an auxiliary language, say Es, an estimate of the upper bound on the agreement term (Equation 6.10) is computed as follows: First, we concatenate Es language tags to both $\mathbf{x}_{\text{En}}$ and $\mathbf{x}_{\text{Fr}}$ and encode the sequences so that both can be translated into Es (the encoding process is depicted in Figure 6.4-A). Next, we decode each of the encoded sentences and obtain auxiliary translations, $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{En}})$ and $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{Fr}})$, depicted as blue blocks in Figure 6.4-B. This allows us to

[8] In practice, note that there is still a constant factor overhead due to extra encoding-decoding steps to/from auxiliary languages, which is about ×4 when training on a single GPU.

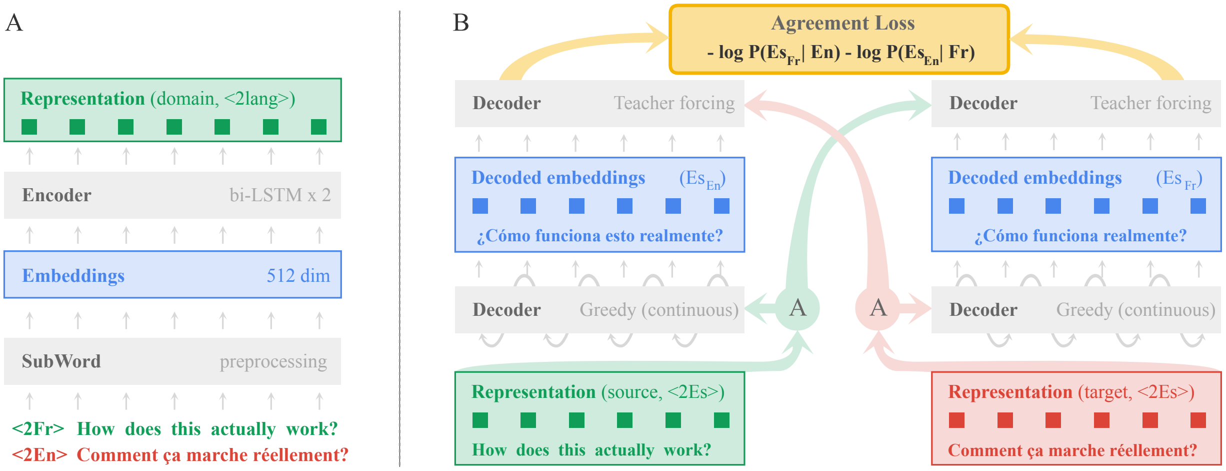

A
Representation (domain, <2lang>)
Encoder
bi-LSTM x 2
Embeddings
512 dim
SubWord
preprocessing
<2Fr> How does this actually work?
<2En> Comment ça marche réellement?
B
Agreement Loss
- log P(Es_Fr| En) - log P(Es_En| Fr)
Decoder
Teacher forcing
Decoded embeddings
(Es_En)
¿Cómo funciona esto realmente?
Decoder
Greedy (continuous)
A
Representation (source, <2Es>)
How does this actually work?
Decoder
Teacher forcing
Decoded embeddings
(Es_Fr)
¿Cómo funciona realmente?
A
Decoder
Greedy (continuous)
Representation (target, <2Es>)
Comment ça marche réellement?

Figure 6.4: (A) Computation graph for the encoder. The representations depend on the input sequence and the target language tag. (B) Computation graph for the agreement loss. First, encode source and target sequences with the auxiliary language tags. Next, decode $\mathbf{z}_{\text{Es}}$ from both $\mathbf{x}_{\text{En}}$ and $\mathbf{x}_{\text{Fr}}$ using continuous greedy decoder. Finally, evaluate log probabilities, $\log p_\theta\left(\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{En}}) \mid \mathbf{x}_{\text{Fr}}\right)$ and $\log p_\theta\left(\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{Fr}}) \mid \mathbf{x}_{\text{En}}\right)$, and compute a sample estimate of the agreement loss.

treat pairs $(\mathbf{x}_{\text{Fr}}, \mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{En}}))$ and $(\mathbf{x}_{\text{En}}, \mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{Fr}}))$ as new parallel data for En → Es and Fr → Es. Finally, using these pairs, we can compute two agreement terms:

$$\ell^{\text{Fr}}_{\text{Es}\leftarrow\text{En}} := \log p\left(\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{Fr}}) \mid \mathbf{x}_{\text{En}}, \boldsymbol{\theta}\right), \quad \ell^{\text{En}}_{\text{Es}\leftarrow\text{Fr}} := \log p\left(\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{En}}) \mid \mathbf{x}_{\text{Fr}}, \boldsymbol{\theta}\right) \tag{6.11}$$

using encoding-decoding with teacher forcing (same way as typically done for the supervised directions). Crucially, note that $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{En}})$ corresponds to a supervised direction, En → Es, while $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{Fr}})$ corresponds to zero-shot, Fr → Es. We these losses to (i) improve the zero-shot direction while (ii) minimally affecting the supervised direction. To achieve (i), we use continuous decoding, and for (ii) we use stop-gradient-based protection of the supervised directions. Both techniques are described below.

Greedy continuous decoding. In order to make $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{En}})$ and $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{Fr}})$ differentiable with respect to $\boldsymbol{\theta}$, at each decoding step t, we treat the output of the RNN, $\mathbf{h}^t$, as the key and use dot-product attention over the vocabulary of embeddings, $\mathbf{V}$, to construct $\mathbf{z}^t_{\text{Es}}$:

$$\mathbf{z}^t_{\text{Es}} := \operatorname{softmax}\left\{(\mathbf{h}^t)^\top \mathbf{V}\right\} \mathbf{V} \tag{6.12}$$

In other words, auxiliary translations, $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{En}})$ and $\mathbf{z}_{\text{Es}}(\mathbf{x}_{\text{Fr}})$, are fixed length sequences of differentiable embeddings computed in a greedy fashion.

Protecting supervised directions. Algorithm 6.1 scales agreement losses by a small coefficient γ. We found experimentally that training could be sensitive to this hyperparameter since the agreement loss also affects the supervised sub-models. For example, agreement of En → Es (supervised) and Fr → Es (zero-shot) may push the former towards a worse translation, especially at the beginning of training. To stabilize training, we apply the `stop_gradient` operator to the log probabilities and samples produced by the supervised sub-models before computing the agreement terms (Equation 6.11), to zero-out the undesirable gradient updates.

6.5 Experiments

We evaluate agreement-based training against baselines from the literature on three public datasets that have multi-parallel *evaluation data* that allows assessing zero-shot performance. We report results in terms of the BLEU score (Papineni et al., 2002).[9]

6.5.1 Datasets

UN corpus. Following the setup introduced by Sestorain et al. (2018), we use two datasets, *UNcorpus-1* and *UNcorpus-2*, derived from the United Nations Parallel Corpus (Ziemski et al., 2016).[10] *UNcorpus-1* consists of data in three languages, En, Es, Fr, where *UNcorpus-2* has Ru as the 4th language. For training, we use parallel corpora between En and the rest of the languages, each containing about 1 million sentences, sub-sampled from the official training data in a way that ensures no multi-parallel training data. The *dev* and *test* sets contain 4,000 sentences and are all multi-parallel, which is necessary for evaluation purposes only.

Europarl v7. We consider the following languages: De, En, Es, Fr. For training, again, we use parallel data between En and the rest of the languages (about 1 million sentences per parallel corpus), preprocessed to avoid multi-parallel sentences, as was also done by Cheng et al. (2017) and Chen et al. (2017) and described below. The *dev* and *test* sets contain 2,000 multi-parallel sentences, again used for evaluation purposed only.

IWSLT17. We use data from the official multilingual task: 5 languages (De, En, It, Nl, Ro), 20 translation tasks of which 4 zero-shot (De $\leftrightarrow$ Nl and It $\leftrightarrow$ Ro) and the rest 16 supervised. Note that this dataset has a significant overlap between parallel corpora in the supervised directions (up to 100K sentence pairs per direction). This implicitly makes the dataset multi-parallel and defeats the purpose of zero-shot evaluation (Dabre et al., 2017). To avoid spurious effects, we also derived **IWSLT17*** dataset from the original one by restricting supervised data to only $\text{En} \leftrightarrow \{\text{De}, \text{Nl}, \text{It}, \text{Ro}\}$ and eliminating multi-parallel data by removing overlapping pivoting sentences. We report results on both the official and preprocessed datasets.

Preprocessing. To properly evaluate systems in terms of zero-shot generalization, we preprocess Europarl and IWSLT* to avoid multi-lingual parallel sentences of the form *source-pivot-target*, where *source-target* is a zero-shot direction. To do so, we follow Cheng et al. (2017) and Chen et al. (2017) and randomly split the overlapping pivot sentences of the original *source-pivot* and *pivot-target* corpora into two parts and merge them separately with the non-overlapping parts for each pair. Along with each parallel training sentence, we save information about source and target tags, after which all the data is combined and shuffled. Finally, we use a shared multilingual subword vocabulary (Sennrich et al., 2015) on the training data (with 32K merge ops), separately for each dataset. Data statistics are provided in Appendix D.4.2.

[9] The score was computed using mteval-v13a.perl.

[10] Description of the data is available at https://github.com/liernisestorain/zero-shot-dual-MT.

6.5.2 Training and Evaluation

Additional details on the hyperparameters can be found in Appendix D.4.1.

Models. We use a smaller version of the GNMT architecture (Wu et al., 2016) in all our experiments: 512-dimensional embeddings (separate for source and target sides), 2 bidirectional LSTM layers of 512 units each for encoding, and GNMT-style, 4-layer, 512-unit LSMT decoder with residual connections from the 2nd layer onward. Multilingual translation was enabled by prepending target language tags (of the form $\langle \text{fr} \rangle$) to the source-side sequences as was originally proposed by Johnson et al. (2016).

Training. We trained the above model using the *standard method* of Johnson et al. (2016) and using our proposed *agreement-based* training (Algorithm 6.1). In both cases, the model was optimized using Adafactor (Shazeer and Stern, 2018) on a machine with four P100 GPUs for up to 500K steps, with early stopping on the dev set.

Evaluation. We focus our evaluation on zero-shot performance of the following methods:

(a) BASIC, which stands for directly evaluating a multilingual GNMT model after standard training Johnson et al., 2016.

(b) PIVOT, which performs pivoting-based inference using a multilingual GNMT model (after standard training); often regarded as gold-standard.

(c) AGREE, which applies a multilingual GNMT model trained with agreement losses directly to zero-shot directions.

To ensure a fair comparison in terms of model capacity, all the techniques above use the same multilingual GNMT architecture described in the previous section. All other results provided in the tables are as reported in the literature.

Implementation. All our methods were implemented using TensorFlow (Abadi et al., 2016) on top of the `tensor2tensor` library (Vaswani et al., 2018). Our code has been maid available through the public repository of open source projects from the Google AI Language team.

6.5.3 Results on UN Corpus and Europarl

UN Corpus. Tables 6.1 and 6.2 show results on the UNCorpus datasets. Our approach consistently outperforms BASIC and DUAL-0, despite the latter being trained with additional monolingual data (Sestorain et al., 2018). We see that models trained with agreement perform comparably to PIVOT, outperforming it in some cases, *e.g.*, when the target is Russian, perhaps because it is quite different linguistically from the English pivot.

Furthermore, unlike DUAL-0, AGREE maintains high performance in the supervised directions (within 1 BLEU point compared to BASIC), indicating that our agreement-based approach is effective as a part of a single multilingual system.

Table 6.1: Results on UNCorpus-1.

	Sestorain et al. (2018)†			Our baselines		
	PBSMT	NMT-0	Dual-0	Basic	Pivot	Agree
En → Es	61.26	51.93	—	56.58	56.58	56.36
En → Fr	50.09	40.56	—	44.27	44.27	44.80
Es → En	59.89	51.58	—	55.70	55.70	55.24
Fr → En	52.22	43.33	—	46.46	46.46	46.17
Supervised (avg.)	55.87	46.85	—	50.75	50.75	50.64
Es → Fr	52.44	20.29	36.68	34.75	**38.10**	37.54
Fr → Es	49.79	19.01	39.19	37.67	**40.84**	40.02
Zero-shot (avg.)	51.11	19.69	37.93	36.21	**39.47**	38.78

Table 6.2: Results on UNCorpus-2.

	Sestorain et al. (2018)†			Our baselines		
	PBSMT	NMT-0	Dual-0	Basic	Pivot	Agree
En → Es	61.26	47.51	44.30	55.15	55.15	54.30
En → Fr	50.09	36.70	34.34	43.42	43.42	42.57
En → Ru	43.25	30.45	29.47	36.26	36.26	35.89
Es → En	59.89	48.56	45.55	54.35	54.35	54.33
Fr → En	52.22	40.75	37.75	45.55	45.55	45.87
Ru → En	52.59	39.35	37.96	45.52	45.52	44.67
Supervised (avg.)	53.22	40.55	36.74	46.71	46.71	46.27
Es → Fr	52.44	25.85	34.51	34.73	35.93	**36.02**
Fr → Es	49.79	22.68	37.71	38.20	39.51	**39.94**
Es → Ru	39.69	9.36	24.55	26.29	27.15	**28.08**
Ru → Es	49.61	26.26	33.23	33.43	**37.17**	35.01
Fr → Ru	36.48	9.35	22.76	23.88	24.99	**25.13**
Ru → Fr	43.37	22.43	26.49	28.52	**30.06**	29.53
Zero-shot (avg.)	45.23	26.26	29.88	30.84	**32.47**	32.29

†Source: https://openreview.net/forum?id=ByecAoAqK7.

Table 6.3: Zero-shot results on Europarl. Note that Soft and Distill are not multilingual systems.

	Previous work		Our baselines		
	Soft‡	Distill†	Basic	Pivot	Agree
En → Es	—	—	34.69	34.69	33.80
En → De	—	—	23.06	23.06	22.44
En → Fr	31.40	—	33.87	33.87	32.55
Es → En	31.96	—	34.77	34.77	34.53
De → En	26.55	—	29.06	29.06	29.07
Fr → En	—	—	33.67	33.67	33.30
Supervised (avg.)	—	—	31.52	31.52	30.95
Es → De	—	—	18.23	20.14	**20.70**
De → Es	—	—	20.28	**26.50**	22.45
Es → Fr	30.57	**33.86**	27.99	32.56	30.94
Fr → Es	—	—	27.12	**32.96**	29.91
De → Fr	23.79	**27.03**	21.36	25.67	24.45
Fr → De	—	—	18.57	**19.86**	19.15
Zero-shot (avg.)	—	—	22.25	26.28	24.60

†Soft pivoting (Cheng et al., 2017). ‡Distillation (Chen et al., 2017).

Europarl. Table 6.3 shows the results on the Europarl corpus. On this dataset, our approach consistently outperforms Basic by 2-3 BLEU points but lags a bit behind Pivot on average (except on Es → De where it is better). Cheng et al. (2017)[11] and Chen et al. (2017) have reported zero-resource results on a subset of these directions and our approach outperforms the former but not the latter on these pairs. Note that both Cheng et al. (2017) and Chen et al. (2017) train separate models for each language pair and the approach of Chen et al. (2017) would require training $O(k^2)$ models to encompass all the pairs. In contrast, we use a single multilingual architecture which has more limited model capacity (although in theory, our approach is also compatible with using separate models for each direction).

6.5.4 Analysis of IWSLT17 zero-shot tasks

Table 6.4a presents results on the original IWSLT17 task. We note that because of the large amount of data overlap and presence of many supervised translation pairs (16) the vanilla training method (Johnson et al., 2016) achieves very high zero shot performance, even outperforming Pivot. While our approach gives small gains over these baselines, we believe the dataset's pecularities make it not reliable for evaluating zero-shot generalization.

On the other hand, on our proposed preprocessed IWSLT17* that eliminates the overlap and reduces the number of supervised directions, there is a considerable gap between the supervised and zero-shot performance of Basic. Agree performs better than Basic and slightly worse than Pivot. Note that results on IWSLT17 and IWSLT17* cannot be compared directly.[12]

[11] We only show their best zero-resource results since some of their methods require direct parallel data.

[12] The IWSLT17 and IWSLT17* benchmarks contain different tasks (IWSLT17* has fewer translation directions) and even for the same tasks the amount of data is different due to preprocessing.

Table 6.4: Results on the official IWSLT17 and our preprocessed IWSLT17⋆ multilingual tasks. The BLEU scores are averaged over the supervised and zero-shot tasks, respectively.

(a) IWSLT17

	Previous work		Our baselines		
	SOTA†	CPG‡	Basic	Pivot	Agree
Supervised	24.10	19.75	24.63	24.63	23.97
Zero-shot	20.55	11.69	19.86	19.26	**20.58**

(b) IWSLT17⋆

	Basic	Pivot	Agree
Supervised	28.72	28.72	**29.17**
Zero-shot	12.61	**17.68**	15.23

†From Dabre et al. (2017). ‡From Platanios et al. (2018).

6.5.5 Small data regime

To better understand the dynamics of different methods in the small data regime, we also trained all our methods on subsets of the Europarl for 200K steps and evaluated on the dev set. The training set size varied from 50 to 450K parallel sentences. From Figure 6.5, Basic tends to perform extremely poorly while Agree is the most robust (also in terms of variance across zero-shot directions). We see that Agree generally upper-bounds Pivot, except for the (Es, Fr) pair, perhaps due to fewer cascading errors along these directions.

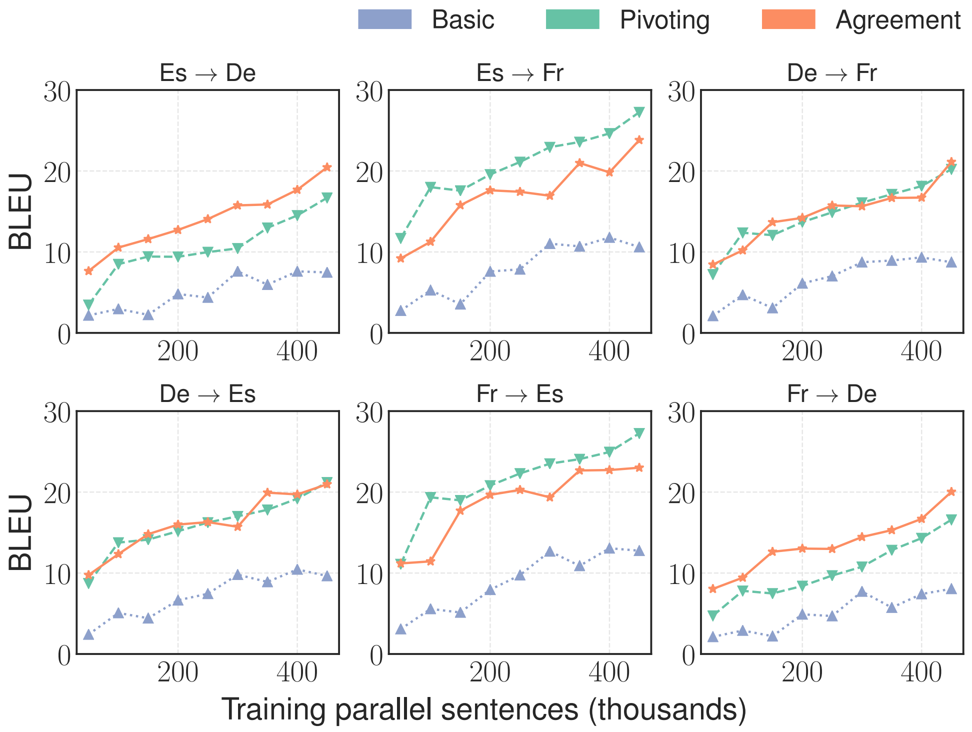


Figure 6.5: BLEU on the dev set for Agree and the baselines trained on smaller subsets of Europarl.

6.6 Conclusion

In this chapter, we studied zero-shot generalization in the context of multilingual neural machine translation. First, we introduced the concept of zero-shot consistency that implies generalization. Next, we developed a provably consistent agreement-based learning approach for zero-shot translation, which optimizes an upper bound on the negative log likelihood under a joint probabilistic model of multilingual data which we constructed using the PMM framework. Empirical results on three benchmark datasets showed that agreement-based learning yields up to +3 BLEU zero-shot improvement over the Johnson et al. (2016) baseline, compares favorably to other approaches in the literature (Cheng et al., 2017; Sestorain et al., 2018), is competitive with pivoting, and does not lose in performance on supervised directions.

We emphasize that, while our case study was focused on multilingual machine translation, our ideas and methods could be useful beyond translation, especially in arbitrary multi-modal settings. For instance, it could be applied to tasks such as cross-lingual natural language inference Conneau et al., 2018, style-transfer (Shen et al., 2017; Fu et al., 2017; Prabhumoye et al., 2018), or multilingual image or video captioning. Another interesting future direction would be to explore different hand-engineered or learned data representations, which one could use to encourage models to agree on during training (*e.g.*, make translation models agree on latent semantic parses, summaries, or potentially other data representations available at training time).

6.6.1 Limitations

The probabilistic framework developed in this chapter is fairly universal, but the approximations that we had to make to design a computationally efficient learning algorithm have limitations. First, computing agreement-based loss terms requires two extra encoding-decoding steps, which adds a constant factor overhead. Further, in order to make the losses differentiable, we had opt for a suboptimal greedy decoding strategy. Note that while it is possible to avoid decoding altogether using heuristic losses on the encoded representations instead (*e.g.*, Arivazhagan et al., 2019), empirically, we found that agreement-based losses even with suboptimal greedy decoding gave better results. An interesting direction to pursue next would be to find a way to substitute the extra encoding-decoding steps in our approach architecture that does a single (multilingual) encoding-decoding pass that computes comparable or better approximations of the agreement terms in the loss.

6.6.2 Notable Follow-up Work

Since publication of the original paper that this chapter is based on (Al-Shedivat and Parikh, 2019), our probabilistic approach has been extended in unsupervised translation:

- Garcia et al. (2020) and Li et al. (2020) extended our probabilistic framework to encompass unsupervised machine translation, *i.e.*, a setting where the translation graph (Figure 6.2) is disjoint and some languages have only mono-lingual corpora available.

Chapter 7

A Multitask View of Interpretability

7.1 Summary

This final case study is focused on the problem of interpretability of machine learning models. We approach this problem in a non-traditional way: instead of trying to explain predictions of a complex model trained to solve a single learning task, we show how to decompose the problem into multiple tasks and *learn to infer* (or generate) interpretable models for each sub-task. To this end, we introduce Contextual Explanation Networks (CENs)—a class of end-to-end trainable neural architectures that enables such inference of task-specific interpretable models. Our approach offers multiple advantages, including no loss in predictive performance and up to three orders of magnitude faster generation of faithful explanations for each prediction compared to commonly used alternative post-hoc model explanation methods.

7.2 Motivation and Goals

Model interpretability is a long-standing problem in machine learning that has become quite acute with the accelerating pace of the widespread adoption of complex predictive algorithms. While high performance often supports our belief in the predictive capabilities of a system, perturbation analysis reveals that black-box models can be easily broken in an unintuitive and unexpected manner (Szegedy et al., 2013; Nguyen et al., 2015). Therefore, for a machine learning system to be used in a social context (*e.g.*, in healthcare) it is imperative to provide sound reasoning for each prediction or decision it makes.

To design such systems, we may restrict the class of models to only *human-intelligible* (Caruana et al., 2015). However, such an approach is often limiting in modern practical settings. Alternatively, we may fit a complex model and explain its predictions *post-hoc*, *e.g.*, by searching for linear local approximations of the decision boundary (Ribeiro et al., 2016). While such methods achieve their goal, explanations are generated *a posteriori* require additional computation per data instance and, most importantly, are never the basis for the predictions made in the first place, which may lead to erroneous interpretations, as we show in this chapter.

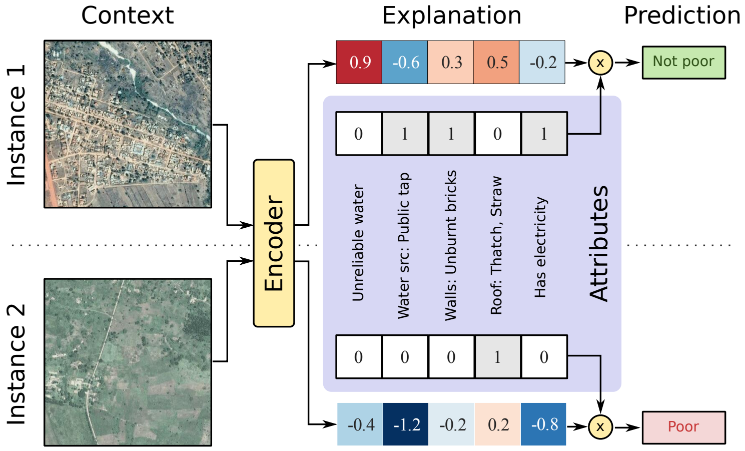


Figure 7.1: Illustration of the high-level functionality of CENs: The context is represented by satellite imagery and used to generate instance-specific linear models (explanations). The latter act on a set of interpretable attributes from regional survey data to produce predictions.

Inspired by the human learning and decision process (Lombrozo, 2006), we introduce CENs—a class of architectures that learn to predict and to explain jointly, alleviating the drawbacks of the post-hoc methods. To make a prediction, CENs operate as follows (Figure 7.1). First, they process a subset of inputs and generate parameters for a simple probabilistic model (*e.g.*, sparse linear model) which is regarded interpretable by a domain expert. Then, the generated model is applied to another subset of inputs and produces a prediction. To motivate such an architecture, we consider the following example.

A motivating illustration: One of the tasks we consider in this paper is classification of households into poor and not poor having access to satellite imagery and categorical data from surveys (Jean et al., 2016). If a human were to solve this task, to make predictions, they might assign weights to features in the categorical data and explain their predictions in terms of the most relevant variables (*i.e.*, come up with a linear model). Moreover, depending on the type of the area (as seen from the imagery), they might select slightly different weights for different areas (*e.g.*, when features indicative of poverty are different for urban and rural areas).

The CEN architecture given in Figure 7.1 imitates this process by making predictions using sparse linear models applied to interpretable categorical features. The weights of the linear models are contextual, generated by a learned encoder that maps images (the contexts) to weight vectors (the explanations). The learned encoder is sensitive to the infrastructure presented in the input images and generates different linear models for urban and rural areas. CENs can represent complex model classes by using powerful encoders. At the same time, by offsetting complexity into the encoding process, we achieve simplicity of the generated explanations and can interpret predictions in terms the variables of interest.

The proposed architecture opens a number of questions: What are the fundamental advantages and limitations of CEN? How much of the performance should be attributed to the context encoder and how much to the explanations? Finally, how do CEN-generated explanations compare to alternatives? In the rest of this chapter, we formalize our intuitions and answer these questions theoretically and experimentally.

7.3 Preliminaries

We start by introducing the notation and reviewing post-hoc model explanations, with a focus on Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016) as one of the most popular frameworks to date.

Given a collection of data where each instance is represented by inputs, $\mathbf{c} \in \mathcal{C}$, and targets, $\mathbf{y} \in \mathcal{Y}$, our goal is to learn an accurate predictive model, $f : \mathcal{C} \mapsto \mathcal{Y}$. To explain predictions, we can assume that each data point has another set of features, $\mathbf{x} \in \mathcal{X}$. Ribeiro et al. (2016) originally proposed to construct explanations in the form of simpler models, $g_{\mathbf{c}} : \mathcal{X} \mapsto \mathcal{Y}$, so that they are consistent with the original model in the neighborhood of the corresponding data instance, *i.e.*, $g_{\mathbf{c}}(\mathbf{x}) = f(\mathbf{c})$. While the original inputs, $\mathbf{c}$, can be of complex, low-level, unstructured data types (*e.g.*, text, image pixels, sensory inputs), we assume that $\mathbf{x}$ are high-level, meaningful variables (*e.g.*, categorical features).

In the post-hoc explanation literature, it is assumed that $\mathbf{x}$ are *derived* from $\mathbf{c}$ and are often binary (Lundberg and Lee, 2017) (*e.g.*, $\mathbf{c}$ can be images, while $\mathbf{x}$ can be vectors of binary indicators over the corresponding super-pixels). We consider a more general setting where $\mathbf{c}$ and $\mathbf{x}$ are not necessarily derived from each other. Throughout the chapter, we call $\mathbf{c}$ the *context* and $\mathbf{x}$ the *attributes* or *variables of interest.*

Locally Interpretable Model-agnostic Explanations (LIME)

Given a trained model, f, and a data instance with features $(\mathbf{c}, \mathbf{x})$, LIME constructs an explanation, $g_{\mathbf{c}}$, as follows:

$$g_{\mathbf{c}} = \operatorname*{arg\,min}_{g \in \mathcal{G}} \mathcal{L}(f, g, \pi_{\mathbf{c}}) + \Omega(g) \tag{7.1}$$

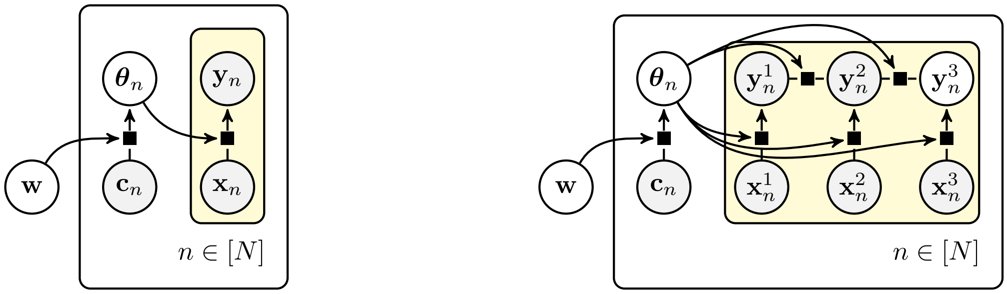
where $\mathcal{L}(f, g, \pi_{\mathbf{c}})$ is the loss that measures how well g approximates f in the neighborhood defined by the similarity kernel, $\pi_{\mathbf{c}} : \mathcal{X} \mapsto \mathbb{R}_+$, in the space of attributes, $\mathcal{X}$, and $\Omega(g)$ is the penalty on the complexity of explanation.[1] Now more specifically, Ribeiro et al. (2016) assume that $\mathcal{G}$ is the class of linear models, $g_{\mathbf{c}}(\mathbf{x}) := b_{\mathbf{c}} + \mathbf{w}_{\mathbf{c}} \cdot \mathbf{x}$, and define the loss and the similarity kernel as follows:

$$\mathcal{L}(f, g, \pi_{\mathbf{c}}) := \sum_{\mathbf{x}' \in \mathcal{X}} \pi_{\mathbf{c}}(\mathbf{x}') \left(f(\mathbf{c}') - g(\mathbf{x}')\right)^2, \quad \pi_{\mathbf{c}}(\mathbf{x}') := \exp\left\{-D(\mathbf{x}, \mathbf{x}')^2/\sigma^2\right\} \tag{7.2}$$

where the data instance of interest is represented by $(\mathbf{c}, \mathbf{x})$, $\mathbf{x}'$ and the corresponding $\mathbf{c}'$ are the perturbed features, $D(\mathbf{x}, \mathbf{x}')$ is some distance function, and σ is the scale parameter of the kernel. The regularizer, $\Omega(g)$, is often chosen to favor sparse explanations.

The model-agnostic property is the key advantage of LIME (and variations)—we can solve Equation 7.1 for any trained model, f, any class of explanations, $\mathcal{G}$, at any point of interest, $(\mathbf{c}, \mathbf{x})$. While elegant, predictive and explanatory models in this framework are learned independently and hence never affect each other. In the next section, we propose a class of models that ties prediction and explanation together in a joint probabilistic framework.

[1] Ribeiro et al. (2016) argue that only simple models of low complexity (*e.g.*, sufficiently sparse linear models) are human-interpretable and support that by human studies.



(a) CEN for scalar prediction

(b) CEN for structured prediction

Figure 7.2: Probabilistic graphical representations of different CENs: (a) CEN for a scalar prediction task with linear explanations and a context encoder parameterized by $\mathbf{w}$. (b) CEN for a structured prediction task with linear CRF-based explanations and a context encoder parameterized by $\mathbf{w}$.

7.4 Approach

We consider the same problem of learning from a collection of data represented by context variables, $\mathbf{c} \in \mathcal{C}$, attributes, $\mathbf{x} \in \mathcal{X}$, and targets, $\mathbf{y} \in \mathcal{Y}$. Our goal is to learn a model, $p\left(\mathbf{y} \mid \mathbf{x}, \mathbf{c}, \mathbf{w}\right)$, parametrized by $\mathbf{w}$ that can predict $\mathbf{y}$ from $\mathbf{x}$ and $\mathbf{c}$. We define contextual explanation networks as probabilistic models that assume the following form (Figure 7.2):[2]

$$p\left(\mathbf{y} \mid \mathbf{x}, \mathbf{c}, \mathbf{w}\right) = \int_{\boldsymbol{\theta} \in \Theta} p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right) d\boldsymbol{\theta} \tag{7.3}$$

where $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)$ is a predictive distribution parametrized by $\boldsymbol{\theta}$. We call such predictors *explanations*, since they explicitly relate interpretable attributes, $\mathbf{x}$, to the targets, $\mathbf{y}$. For example, when the targets are scalar and binary, explanations may take the form of linear logistic models; when the targets are more complex, dependencies between the components of $\mathbf{y}$ can be represented by a graphical model, *e.g.*, Conditional Random Field (CRF) (Lafferty et al., 2001).

CENs assume that each explanation is context-specific: $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right)$ defines a conditional probability of an explanation $\boldsymbol{\theta}$ being valid in the context $\mathbf{c}$. To make a prediction, we marginalize out $\boldsymbol{\theta}$. To interpret a prediction, $\hat{\mathbf{y}}$, for a given data instance with inputs $(\mathbf{x}, \mathbf{c})$, we must infer the (approximate) posterior, $p\left(\boldsymbol{\theta} \mid \hat{\mathbf{y}}, \mathbf{x}, \mathbf{c}, \mathbf{w}\right)$.

The main advantage of our approach is to allow modeling conditional probabilities $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right)$ in a black-box fashion while keeping the family of explanations that define $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)$ simple and interpretable. For instance, when the context is given as raw text, we may choose $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right)$ to be represented with a recurrent neural network, while $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)$ can still be linear models.

Implications of these assumptions are discussed in Section 7.5. Here, we continue with a discussion of a number of practical choices for $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right)$ and $p\left(\mathbf{v} \mid \mathbf{x}, \boldsymbol{\theta}\right)$ (see Table 7.1).

[2] While we focus on predictive modeling, CENs are applicable beyond that. For example, instead of learning a predictive distribution, $p\left(\mathbf{y} \mid \mathbf{x}, \mathbf{c}, \mathbf{w}\right)$, we may want to learn a contextual marginal distribution, $p\left(\mathbf{x} \mid \mathbf{c}, \mathbf{w}\right)$, over a set random variables $\mathbf{x}$, where $p\left(\mathbf{x} \mid \boldsymbol{\theta}\right)$ is defined by an arbitrary graphical model.

Table 7.1: Different types of encoders and explanations used in CENs.

Encoder	**Parameter distribution,** $p\left(\boldsymbol{\theta} \mid \mathbf{c}\right)$
Deterministic	$\delta\left(\phi(\mathbf{c}), \boldsymbol{\theta}\right)$ where $\phi(\mathbf{c})$ is arbitrary
Constrained	$\delta\left(\phi(\mathbf{c}), \boldsymbol{\theta}\right)$ where $\phi(\mathbf{c}) := \boldsymbol{\alpha}(\mathbf{c})^\top \mathbf{D}$
MoE	$\sum_{k=1}^{K} p\left(k \mid \mathbf{c}\right) \delta(\boldsymbol{\theta}, \boldsymbol{\theta}_k)$

Explanation	**Predictive distribution,** $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)$
Linear	softmax $\left(\boldsymbol{\theta}^\top \mathbf{x}\right)$
Structured	$\propto \exp\left\{-E_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{y})\right\}$ where $E_{\boldsymbol{\theta}}(\cdot, \cdot)$ is some energy function linear in $\boldsymbol{\theta}$

7.4.1 Context Encoders

In practice, we represent $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right)$ with a neural network parameterized by $\mathbf{w}$ that encodes the context into the parameter space of the explanation models. There are two simple ways to construct an encoder, which we describe below.

Deterministic encoding

Let $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right) := \delta\left(\phi_{\mathbf{w}}(\mathbf{c}), \boldsymbol{\theta}\right)$, where $\delta(\cdot, \cdot)$ is the Dirac delta-function and $\phi_{\mathbf{w}}(\cdot)$ is a network parametrized by $\mathbf{w}$ that maps $\mathbf{c}$ to $\boldsymbol{\theta}$. Collapsing the conditional distribution to a delta-function makes $\boldsymbol{\theta}$ depend deterministically on $\mathbf{c}$ and yields the following predictive distribution:

$$p\left(\mathbf{y} \mid \mathbf{x}, \mathbf{c}, \mathbf{w}\right) = \int_{\boldsymbol{\theta} \in \Theta} p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right) \delta\left(\phi_{\mathbf{w}}(\mathbf{c}), \boldsymbol{\theta}\right) d\boldsymbol{\theta} = p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta} = \phi_{\mathbf{w}}(\mathbf{c})\right) \tag{7.4}$$

Modeling $p\left(\boldsymbol{\theta} \mid \mathbf{c}\mathbf{w}\right)$ with a delta-function is convenient since the posterior, $p\left(\boldsymbol{\theta} \mid \mathbf{y}, \mathbf{x}, \mathbf{c}, \mathbf{w}\right) \propto p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right) \delta\left(\phi_{\mathbf{w}}(\mathbf{c}), \boldsymbol{\theta}\right)$ also collapses to $\boldsymbol{\theta}^\star = \phi_{\mathbf{w}}(\mathbf{c})$, and hence the inference is done via a single forward pass and the posterior can be regularized by imposing L_1 or L_2 losses on $\phi_{\mathbf{w}}(\mathbf{c})$.

Constrained deterministic encoding

The downside of deterministic encoding is the lack of constraints on the generated explanations. There are multiple reasons why this might be an issue: (i) when the context encoder is unrestricted, it might generate unstable, overfitted local models, (ii) when we want to reason about the patterns in the data as a whole, local explanations are not enough. To address these issues, we constrain the space of explanations by introducing a context-independent, *global dictionary*, $\mathbf{D} := \{\boldsymbol{\theta}_k\}_{k=1}^{K}$, where each atom, $\boldsymbol{\theta}_k$, is sparse. The encoder generates context-specific explanations using *soft attention* over the dictionary (Figure 7.3):

$$\phi_{\mathbf{w},\mathbf{D}}(\mathbf{c}) := \sum_{k=1}^{K} \boldsymbol{\alpha}_{\mathbf{w}}^{(k)}(\mathbf{c}) \boldsymbol{\theta}_k = \boldsymbol{\alpha}_{\mathbf{w}}(\mathbf{c})^\top \mathbf{D}, \quad \sum_{k=1}^{K} \boldsymbol{\alpha}_{\mathbf{w}}^{(k)}(\mathbf{c}) = 1, \quad \text{for all } k : \boldsymbol{\alpha}_{\mathbf{w}}^{(k)}(\mathbf{c}) \geq 0 \tag{7.5}$$

where $\boldsymbol{\alpha}_{\mathbf{w}}(\mathbf{c})$ is the attention over the dictionary produced by the encoder. Attention-based construction of explanations using a global dictionary (i) forces the encoder to produce models shared across different contexts, (ii) allows us to interpret the learned dictionary atoms as global "explanation modes." Again, since $p_{\mathbf{w}}\left(\boldsymbol{\theta} \mid \mathbf{c}\right)$ is a delta-distribution, the likelihood is the same as given in Equation 7.4 and inference is conveniently done via a forward pass.

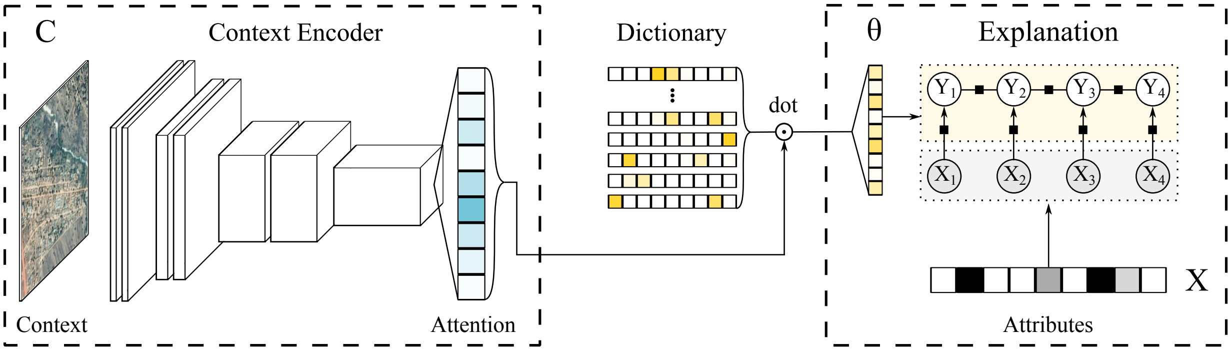


Figure 7.3: An example of a CEN architecture. In this example, the context is represented by an image and transformed by a convolutional neural network encoder into an attention vector, which is used to softly select parameters for a contextual linear probabilistic model.

The two proposed context encoders represent $p\left(\boldsymbol{\theta} \mid \mathbf{c}\right)$ with delta-functions, which simplifies learning, inference, and interpretation of the model, and are used in our experiments. Other ways to represent $p\left(\boldsymbol{\theta} \mid \mathbf{c}\right)$ include: (i) using a mixture of delta-functions (which makes CEN function similar to a mixture-of-experts model and further discussed in Section 7.5.1), or (ii) using variational autoencoding. More complex approaches are left to future research.

7.4.2 Explanations

We consider two types of explanations: *linear* that can be used for regression or classification and *structured* that are suitable for structured prediction.

Linear Explanations

When solving classification problems, linear explanations take the following form:

$$p\left(\mathbf{y} = i \mid \mathbf{x}, \boldsymbol{\theta}\right) := \frac{\exp\left\{(\mathbf{W}\mathbf{x} + \mathbf{b})_i\right\}}{\sum_{j \in \mathcal{Y}} \exp\left\{(\mathbf{W}\mathbf{x} + \mathbf{b})_j\right\}}, \tag{7.6}$$

where $\boldsymbol{\theta} := (\mathbf{W}, \mathbf{b})$ and i, j index classes in $\mathcal{Y}$. If $\mathbf{x}$ is d-dimensional and we are given m-class classification problem, then $\mathbf{W} \in \mathbb{R}^{m \times d}$ and $\mathbf{b} \in \mathbb{R}^m$. The case of regression is similar.

In Section 7.5.4, we show that if we apply LIME to interpret CEN with linear explanations, the local linear models inferred by LIME are guaranteed to recover the original CEN-generated explanations. In other words, linear explanations generated by CEN have similar properties, *e.g.*, such as *local faithfulness* (Ribeiro et al., 2016). However, we emphasize the key difference between LIME (or any post-hoc explanation technique) and CEN: the former regards explanation as a post-processing step (done after training) while the latter integrates explanation into the learning process. We analyze the implications of this difference empirically later in this chapter.

Structured Explanations

While post-hoc methods, such as LIME, can easily generate local linear explanations for scalar outputs, using such methods for structured outputs is non-trivial. At the same time, CENs let us represent $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)$ using arbitrary graphical models. To be concrete, we consider the case where the targets are binary vectors, $\mathbf{y} \in \{0, 1\}^m$, and explanations are represented by CRFs with linear potential functions (Lafferty et al., 2001).

The predictive distribution $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)$ represented by a CRF takes the following form:

$$p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right) := \frac{1}{Z_{\boldsymbol{\theta}}(\mathbf{x})} \prod_{a \in \mathcal{A}} \Psi_a(\mathbf{y}_a, \mathbf{x}_a; \boldsymbol{\theta}) \tag{7.7}$$

where $Z_{\boldsymbol{\theta}}(\mathbf{x})$ is the normalizing constant and $a \in \mathcal{A}$ indexes subsets of variables in $\mathbf{x}$ and $\mathbf{y}$ that correspond to the factors:

$$\Psi_a(\mathbf{y}_a, \mathbf{x}_a, \boldsymbol{\theta}) := \exp\left\{\sum_{k=1}^{K} \boldsymbol{\theta}_{ak} f_{ak}(\mathbf{x}_a, \mathbf{y}_a)\right\}, \tag{7.8}$$

where $\{f_{ak}(\mathbf{x}_a, \mathbf{y}_a)\}_{k=1}^{K}$ is a collection of feature vectors associated with factor $\Psi_a(\mathbf{y}_a, \mathbf{x}_a, \boldsymbol{\theta})$. For interpretability purposes, we are interested in CRFs with feature vectors that are linear or bi-linear in $\mathbf{x}$ and $\mathbf{y}$. There is a variety of application-specific CRF models developed in the literature (*e.g.*, see Sutton, McCallum, et al., 2012). While in the following section, we discuss learning and inference more generally, in Section 7.6.3 we develop a CEN model with linear chain CRF explanations for solving survival analysis tasks.

7.4.3 Inference and Learning

CENs with deterministic encoders are convenient since the posterior, $p\left(\boldsymbol{\theta} \mid \mathbf{y}, \mathbf{x}, \mathbf{c}\right)$, collapses to a point $\boldsymbol{\theta}^\star = \phi_{\mathbf{w}}(\mathbf{c})$. Inference in such models is done in two steps: (1) first, compute $\boldsymbol{\theta}^\star$, then (2) using $\boldsymbol{\theta}^\star$ as parameters, compute the predictive distribution, $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^\star\right)$. To train the model, we can optimize its log likelihood on the training data. To make a prediction using a trained CEN model, we infer $\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^\star\right)$. For classification (and regression) computing predictions is straightforward. Below, we show how to compute predictions for CEN with CRF-based explanations.

Inference for CEN with Structured Explanations

Given a CRF model Equation 7.7, we can make a prediction $\hat{\mathbf{y}}$ for inputs $(\mathbf{c}, \mathbf{x})$ by performing inference:

$$\hat{\mathbf{y}}(\boldsymbol{\theta}^\star) = \underset{\mathbf{y} \in \mathcal{Y}}{\arg\max}\, p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^\star\right) = \underset{\mathbf{y} \in \mathcal{Y}}{\arg\max} \sum_{a=1}^{A} \sum_{k=1}^{K} \boldsymbol{\theta}^\star_{ak} f_{ak}(\mathbf{x}_a, \mathbf{y}_a) \tag{7.9}$$

Depending on the structure of the CRF model (*e.g.*, linear chain, tree-structured model, etc.), we could use different inference algorithms, such the Viterbi algorithm or variational inference, in order to solve Equation 7.9 (for an overview and examples see Ch. 4, Sutton, McCallum, et al., 2012). The key point here is that having $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^{\star}\right)$ or $\hat{\mathbf{y}}(\boldsymbol{\theta}^{\star})$ computable in an (approximate) functional form, lets us construct different objective functions, *e.g.*, $\mathcal{L}(\{\mathbf{y}_i, \mathbf{x}_i, \mathbf{c}_i\}_{i=1}^{N}, \mathbf{w})$, and learn parameters of the CEN model end-to-end using gradient methods, which are standard in deep learning. In Section 7.6.3, we construct a specific objective function for survival analysis.

Learning via Likelihood Maximization and Posterior Regularization

In this chapter, we use the negative log likelihood (NLL) objective for learning CEN models:

$$\mathcal{L}\left(\{\mathbf{y}_i, \mathbf{x}_i, \mathbf{c}_i\}_{i=1}^{N}, \mathbf{w}\right) := -\frac{1}{N}\sum_{i=1}^{N} \log p\left(\mathbf{y}_i \mid \mathbf{x}_i, \boldsymbol{\theta} = \phi_{\mathbf{w}}(\mathbf{c}_i)\right) \tag{7.10}$$

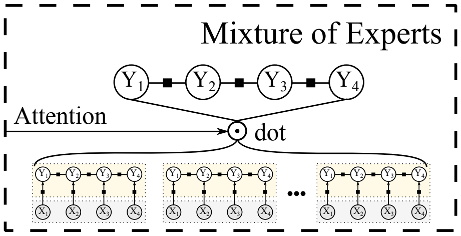
L_1, L_2, and other types of regularization imposed on $\boldsymbol{\theta}$ can be added to the objective Equation 7.10. Such regularizers, as well as the dictionary constraint introduced in Section 7.4.1, can be seen as a form of *posterior regularization* (Ganchev et al., 2010) and are important for achieving the best performance and interpretability.

7.5 Analysis

In this section, we dive into the analysis of CEN as a class of probabilistic models. First, we mention special cases of CEN model class known in the literature, such as mixture-of-experts (Jacobs et al., 1991) and varying-coefficient models (Hastie and Tibshirani, 1993). Then, we discuss implications of the CEN structure, a potential failure mode of CEN with deterministic encoders and how to rectify it using conditional entropy regularization, and finally analyze relationship between CEN-generated and post-hoc explanations. Readers who are mostly interested in empirical properties and applications may skip this section.

7.5.1 Special Cases of CEN

Mixtures of Experts. So far, we have represented $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right)$ with a delta-function centered around the output of the encoder. It is natural to extend $p\left(\boldsymbol{\theta} \mid \mathbf{c}, \mathbf{w}\right)$ to a mixture of delta-distributions, in which case CEN recovers the Mixture of Experts (MoE) (Jacobs et al., 1991). To see this, let $\mathbf{D}$ be a dictionary of experts, and define $p_{\mathbf{w},\mathbf{D}}\left(\boldsymbol{\theta} \mid \mathbf{c}\right) := \sum_{k=1}^{K} p_{\mathbf{w}}\left(k \mid \mathbf{c}\right)\delta(\boldsymbol{\theta}, \boldsymbol{\theta}_k)$. The log-likelihood for CEN in such case is the same as for MoE:



$$\begin{aligned}
&\log p_{\mathbf{w},\mathbf{D}}\left(\mathbf{y}_i \mid \mathbf{x}_i, \mathbf{c}_i\right) \\
&\qquad = \log \int p\left(\mathbf{y}_i | \mathbf{x}_i, \boldsymbol{\theta}\right) p_{\mathbf{w},\mathbf{D}}\left(\boldsymbol{\theta} | \mathbf{c}_i\right) d\boldsymbol{\theta} \\
&\qquad = \log \sum_{k=1}^{K} p_{\mathbf{w}}\left(k | \mathbf{c}_i\right) p\left(\mathbf{y}_i | \mathbf{x}_i, \boldsymbol{\theta}_k\right)
\end{aligned} \tag{7.11}$$

As in Section 7.4.1, $p_{\mathbf{w}}(k \mid \mathbf{c})$ is represented with a soft attention over the dictionary, $\mathbf{D}$, which is now used to combine predictions of the experts with parameters $\{\boldsymbol{\theta}_k\}_{k=1}^{K}$ instead of constructing a single context-specific explanation. Learning of MoE models is done either by optimizing the likelihood or via expectation maximization. Note another difference between CEN and MoE is that the latter assumed that $\mathbf{c} \equiv \mathbf{x}$ and that both $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$ and $p(\boldsymbol{\theta} \mid \mathbf{c})$ can be represented by arbitrary complex model classes, ignoring interpretability.

Varying-Coefficient Models. In statistics, there is a class of (generalized) regression models, called Varying-Coefficient Modelss (VCMs) (Hastie and Tibshirani, 1993), in which coefficients of linear models are allowed to be smooth deterministic functions of other variables (called the "effect modifiers"). Interestingly, the motivation for VCM was to increase flexibility of linear regression. In the original work, Hastie and Tibshirani (1993) focused on simple dynamic (temporal) linear models and on nonparametric estimation of the varying coefficients, where each coefficient depended on a different effect variable. CEN generalizes VCM by (i) allowing parameters, $\boldsymbol{\theta}$, to be random variables that depend on the context, $\mathbf{c}$, nondeterministically, (ii) letting the "effect modifiers" to be high-dimensional context variables (not just scalars), and (iii) modeling the effects using deep neural networks. In other words, CEN alleviates the limitations of VCM by leveraging the probabilistic graphical models and deep learning frameworks.

7.5.2 Implications of the Structure of CENs

CENs represent the predictive distribution in a compound form (Lindsay, 1995):

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{c}) = \int p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathbf{c})\, d\boldsymbol{\theta}$$

and we assume that the data is generated according to $\mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$, $\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \mathbf{c})$. We would like to understand:

> *Can CEN represent any conditional distribution,* $p(\mathbf{y} \mid \mathbf{x}, \mathbf{c})$, *when the class of explanations is limited* (e.g., *to linear models)? If not, what are the limitations?*

Generally, CEN can be seen as a mixture of predictors. Such mixture models could be quite powerful as long as the mixing distribution, $p(\boldsymbol{\theta} \mid \mathbf{c})$, is rich enough. In fact, even a finite mixture exponential family regression models can approximate any smooth d-dimensional density at a rate $O(m^{-4/d})$ in the KL-divergence (Jiang and Tanner, 1999). This result suggests that representing the predictive distribution with contextual mixtures should not limit the representational power of the model. However, there are two caveats:

(i) In practice, $p(\boldsymbol{\theta} \mid \mathbf{c})$ is limited, since we represent it either with a delta-function, a finite mixture, or a simple distribution parametrized by a deep network.

(ii) Classical predictive mixtures (including MoE) do not separate input features into two subsets, $\mathbf{c}$ and $\mathbf{x}$. We do this intentionally to produce explanations in terms of specific variables of interest that could be useful for interpretability or model diagnostics down the line. However, it could be the case that $\mathbf{x}$ contains only some limited information about $\mathbf{y}$, which could limit the predictive power of the full model.

To address point (i), we consider $p(\boldsymbol{\theta} \mid \mathbf{c})$ that fully factorizes over the dimensions of $\boldsymbol{\theta}$: $p(\boldsymbol{\theta} \mid \mathbf{c}) = \prod_j p(\theta_j \mid \mathbf{c})$, and assume that explanations, $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$, also factorize according to some underlying graph, $\mathcal{G}_{\mathbf{y}} = (\mathcal{V}_{\mathbf{y}}, \mathcal{E}_{\mathbf{y}})$. The following proposition shows that in such case $p(\mathbf{y} \mid \mathbf{x}, \mathbf{c})$ inherits the factorization properties of the explanation class.

Proposition 7.1: Factorization of the CEN's predictive distribution

Let $p(\boldsymbol{\theta} \mid \mathbf{c}) := \prod_j p(\theta_j \mid \mathbf{c})$ and let $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$ factorize according to some graph $\mathcal{G}_{\mathbf{y}} = (\mathcal{V}_{\mathbf{y}}, \mathcal{E}_{\mathbf{y}})$. Then, $p(\mathbf{y} \mid \mathbf{x}, \mathbf{c})$ defined by CEN with $p(\boldsymbol{\theta} \mid \mathbf{c})$ encoder and $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$ explanations also factorizes according to $\mathcal{G}$.

Proof. The statement directly follows from the definition of CEN (see Appendix E.1.1). □

Remark 7.1. *All encoders, $p(\boldsymbol{\theta} \mid \mathbf{c})$, considered in this paper, including delta functions and their mixtures, fully factorize over the dimensions of $\boldsymbol{\theta}$.*

Remark 7.2. *The proposition has no implications for the case of scalar targets, $\mathbf{y}$. However, in case of structured prediction, regardless of how good the context encoder is, CEN will strictly assume the same set of independencies as given by the explanation class, $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$.*

As indicated in point (ii), CENs assume a fixed split of the input features into context, $\mathbf{c}$, and variables of interest, $\mathbf{x}$, which has interesting implications. Ideally, we would like $\mathbf{x}$ to be a good predictor of $\mathbf{y}$ in any context $\mathbf{c}$. For instance, following our motivation example (see Figure 7.1), if $\mathbf{c}$ distinguishes between urban and rural areas, $\mathbf{x}$ must encode enough information for predicting poverty *within* urban or rural neighborhoods. However, since the variables of interest are often manually selected (*e.g.*, by a domain expert) and limited, we may encounter the following (not mutually exclusive) situations:

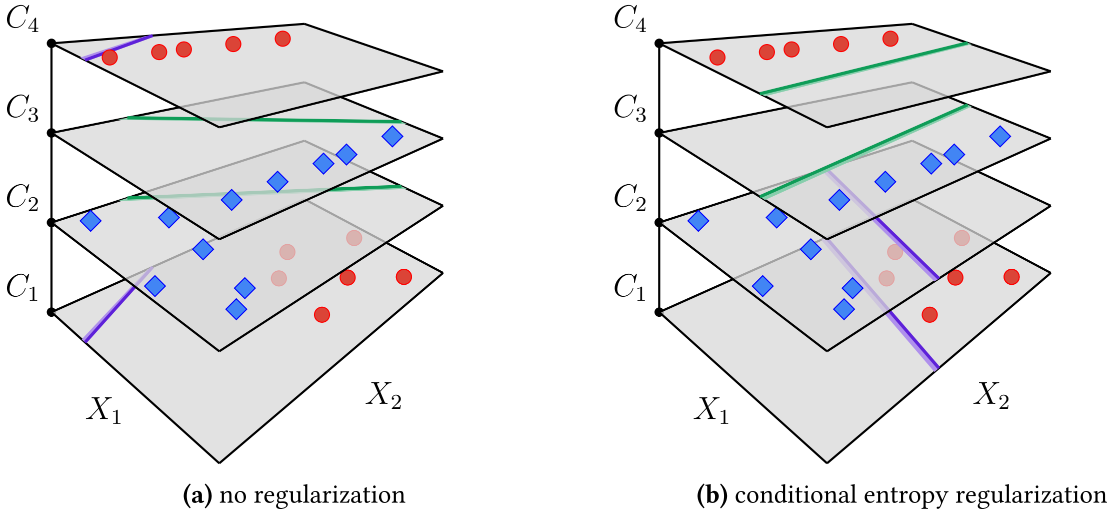
(a) $\mathbf{c}$ may happen to be a strong predictor of $\mathbf{y}$ and already contain information available in $\mathbf{x}$ (*e.g.*, it is the case when $\mathbf{x}$ is derived from $\mathbf{c}$).

(b) $\mathbf{x}$ may happen to be a poor predictor of $\mathbf{y}$, even within the context specified by $\mathbf{c}$.

In both cases, CEN may learn to ignore $\mathbf{x}$, leading to essentially meaningless explanations. In the next section, we show that, if (a) is the case, regularization can help eliminate such behavior. Additionally, if (b) is the case, *i.e.*, $\mathbf{x}$ are bad features for predicting $\mathbf{y}$ (and for seeking explanation in terms of these features), CEN must indicate that. It turns out that the accuracy of CEN depends on the quality of $\mathbf{x}$, as empirically shown in Section 7.6.2.

7.5.3 Conditional Entropy Regularization

CEN has a failure mode: when the context $\mathbf{c}$ is highly predictive of the targets $\mathbf{y}$ and the encoder is represented by a powerful model, CEN may learn to rely entirely on the context variables. In such case, the encoder would generate spurious explanations, one for each target class. For example, for binary targets, $\mathbf{y} \in \{0, 1\}$, CEN may learn to always map $\mathbf{c}$ to either $\boldsymbol{\theta}_0$ or $\boldsymbol{\theta}_1$ when $\mathbf{y}$ is 0 or 1, respectively. In other words, $\boldsymbol{\theta}$ (as a function of $\mathbf{c}$) would become highly predictive of $\mathbf{y}$ on its own, and hence $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) \approx p(\mathbf{y} \mid \boldsymbol{\theta})$, *i.e.*, $\mathbf{y}$ would be (approximately) conditionally



(a) no regularization

(b) conditional entropy regularization

Figure 7.4: A toy synthetic dataset and two linear explanations (green and purple) produced by a CEN model trained (a) with no regularization or (b) with conditional entropy regularization.

independent of x given $\boldsymbol{\theta}$. This is problematic since explanations would become spurious, *i.e.*, no longer actually used to make predictions from the variables of interest.

Note that such a model would be accurate only when the generated $\boldsymbol{\theta}$ is always highly predictive of $\mathbf{y}$, *i.e.*, when the conditional entropy $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta})$ is low. Following this observation, we propose to regularize the model by approximately *maximizing* $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta})$. For a CEN with a deterministic encoder (Section 7.4.1), we can compute an unbiased estimate of $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta})$ given a mini-batch of samples from the dataset as follows:

$$\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta}) = \int p\left(\mathbf{y}, \boldsymbol{\theta}\right) \log p\left(\mathbf{y} \mid \boldsymbol{\theta}\right) d\mathbf{y} d\boldsymbol{\theta} \tag{7.12}$$

$$= \mathbb{E}_{\mathbf{c},\mathbf{x} \sim p(\mathbf{c},\mathbf{x})} \left[\int p\left(\mathbf{y} \mid \mathbf{x}, \phi(\mathbf{c})\right) \log \mathbb{E}_{\mathbf{x}' \sim p(\mathbf{x} \mid \mathbf{c})} \left[p\left(\mathbf{y} \mid \mathbf{x}', \phi(\mathbf{c})\right)\right] d\mathbf{y} \right] \tag{7.13}$$

$$\approx \frac{1}{|B|} \sum_{i \in B} \int p\left(\mathbf{y} \mid \mathbf{x}_i, \phi(\mathbf{c}_i)\right) \log \left[\sum_{\mathbf{x}' \sim p(\mathbf{x} \mid \mathbf{c}_i)} p\left(\mathbf{y} \mid \mathbf{x}', \phi(\mathbf{c}_i)\right) \right] d\mathbf{y} \tag{7.14}$$

In the given expressions, elements of B index training samples (*e.g.*, B represents a mini-batch), Equation 7.13 is obtained by using the definition of CEN and marginalizing out $\boldsymbol{\theta}$, Equation 7.14 is a stochastic estimate that approximates expectations using a mini-batch and samples from $p\left(\mathbf{x} \mid \mathbf{c}_i\right)$. In practice, approximate samples $\mathbf{x}'$ from the latter distribution can be obtained either by simply perturbing $\mathbf{x}_i$ or first learning $p\left(\mathbf{x} \mid \mathbf{c}\right)$ and then sampling from it. Intuitively, if the predictions are accurate while $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta})$ is high, we can be sure that CEN learned to generate contextual $\boldsymbol{\theta}$'s that are uncorrelated with the targets but result into accurate conditional models.

An illustration on synthetic data. To illustrate the problem, we consider a toy synthetic 3D dataset with 2 classes that are not separable linearly (Figure 7.4). The coordinates along the vertical axis C correspond to different contexts, and (X_1, X_2) represent variables of interest.

Note we can perfectly distinguish between the two classes by using only the context information. CEN with a dictionary of size 2 learns to select one of the two linear explanations for each of the contexts. When trained without regularization (Figure 7.4a), selected explanations are spurious hyperplanes since each of them is used for points of a single class only. Adding entropy regularization (Figure 7.4b) makes CEN select hyperplanes that meaningfully distinguish between the classes within different contexts.

Quantifying contribution of the explanations. Starting from the introduction, we have argued that explanations are meaningful when they are used for prediction. In other words, we would like explanations have a non-zero contribution to the overall accuracy of the model. The following proposition quantifies the contribution of explanations to the predictive performance of entropy-regularized CEN.

Proposition 7.2: Bound on the predictive contribution of explanations

Let CEN with linear explanations have the expected predictive accuracy

$$\mathbb{E}_{\mathbf{x},\boldsymbol{\theta}\sim p(\mathbf{x},\boldsymbol{\theta})}\left[p\left(\hat{\mathbf{y}} = \mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)\right] \geq 1 - \varepsilon, \tag{7.15}$$

where $\varepsilon \in (0, 1)$ is small. Let also the conditional entropy be $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta}) \geq \delta$ for some $\delta \geq 0$. Then, the expected contribution of the explanations to the predictive performance of CEN is given by the following lower bound:

$$\mathbb{E}_{\mathbf{x},\boldsymbol{\theta}\sim p(\mathbf{x},\boldsymbol{\theta})}\left[p\left(\hat{\mathbf{y}} = \mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right) - p\left(\hat{\mathbf{y}} = \mathbf{y} \mid \boldsymbol{\theta}\right)\right] \geq \frac{\delta - 1}{\log |\mathcal{Y}|} - \varepsilon, \tag{7.16}$$

where $|\mathcal{Y}|$ denotes the cardinality of the target space.

Proof. The statement follows from Fano's inequality. For details, see Appendix E.1.2. □

Remark 7.3. *The proposition states that explanations are meaningful (as contextual models) only when CEN is accurate (*i.e.*, the expected predictive error is less than ε) and the conditional entropy $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta})$ is high. High accuracy and low entropy imply spurious explanations. Low accuracy and high entropy imply that $\mathbf{x}$ features are not predictive of $\mathbf{y}$ within the class of explanations, suggesting to reconsider our modeling assumptions.*

7.5.4 CEN-generated vs. Post-hoc Explanations

In this section, we analyze the relationship between CEN-generated and LIME-generated *post-hoc* explanations. Given a trained CEN, we can use LIME to approximate its decision boundary and compare the explanations produced by both methods. The question we ask:

> *How does the local approximation, $\hat{\boldsymbol{\theta}}$, relate to the actual explanation, $\boldsymbol{\theta}^{\star}$, generated and used by CEN to make a prediction in the first place?*

For the case of binary classification,[3] it turns out that when the context encoder is deterministic and the space of explanations is *linear*, local approximations, $\hat{\boldsymbol{\theta}}$, obtained by solving Equation 7.1 recover the original CEN-generated explanations, $\boldsymbol{\theta}^\star$. The result is stated formally as follows.

Theorem 7.1

Let explanations and local approximations be linear models, $p\left(\mathbf{y}=1 \mid \mathbf{x}, \boldsymbol{\theta}\right) \propto \exp\left\{\mathbf{x}^\top \boldsymbol{\theta}\right\}$. Further, let the encoder be L-Lipschitz and pick a sampling distribution $\pi_{\mathbf{x},\mathbf{c}}$ that concentrates around the point $(\mathbf{x}, \mathbf{c})$ such that $p_{\pi_{\mathbf{x},\mathbf{c}}}\left(\|\mathbf{z}' - \mathbf{z}\| > t\right) < \varepsilon(t)$, where $\mathbf{z} := (\mathbf{x}, \mathbf{c})$ and $\varepsilon(t) \to 0$ as $t \to \infty$. Then, if for $(\mathbf{x}_k, \mathbf{c}_k) \sim \pi_{\mathbf{x},\mathbf{c}}$ the loss function is defined as:

$$\mathcal{L} = \frac{1}{K} \sum_{k=1}^{K} \left(\operatorname{logit}\left\{p\left(\mathbf{y}=1 \mid \mathbf{x}_k, \mathbf{c}_k\right)\right\} - \operatorname{logit}\left\{p\left(\mathbf{y}=1 \mid \mathbf{x}_k, \boldsymbol{\theta}\right)\right\}\right)^2 \tag{7.17}$$

the solution of Equation 7.1 concentrates around $\boldsymbol{\theta}^\star$ as:

$$\mathbb{P}_{\pi_{\mathbf{x},\mathbf{c}}}\left(\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^\star\| > t\right) \le \delta_{K,L}(t),\ \delta_{K,L} \xrightarrow[t\to\infty]{} 0 \tag{7.18}$$

Intuitively, by sampling from a distribution sharply concentrated around $(\mathbf{x}, \mathbf{c})$, we ensure that $\hat{\boldsymbol{\theta}}$ will recover $\boldsymbol{\theta}^\star$ with high probability. A detailed proof is given in Appendix E.1.3.

This result establishes an equivalence between the explanations generated by CEN and those produced by LIME post-hoc *when approximating CEN*. Note that when LIME is applied to a model other than CEN, equivalence between explanations is not guaranteed. Moreover, as we further show experimentally, certain conditions such as incomplete or noisy interpretable features may lead LIME to produce inconsistent and erroneous explanations.

7.6 Experiments and Results

In this section, we move to a number of case studies where we empirically analyze properties of the proposed CEN framework on classification and survival analysis tasks. In particular, we evaluate CEN with linear explanations on a few classification tasks that involve different data modalities of the context (*e.g.*, images or text). For survival prediction, we design CEN architectures with structured explanations, derive learning and inference algorithms, and showcase our models on problems from the healthcare domain.

7.6.1 Solving Classification using CEN with Linear Explanations

We start by examining the properties of CEN with linear explanations (Table 7.1) on a few classification tasks. Our experiments are designed to answer the following questions:

[3] Analysis of the multi-class case can be reduced to the binary in the one-vs-all fashion.

(i) When explanation is a part of the learning and prediction process, how does that affect performance of the final predictive model *quantitatively*?

(ii) *Qualitatively*, what kind of insight can we gain by inspecting explanations?

(iii) Finally, we analyze *consistency* of linear explanations generated by CEN versus those generated using LIME (Ribeiro et al., 2016), a popular post-hoc method.

Details on our experimental setup, all hyperparameters, and training procedures are given in the tables in Appendix E.2.3.

Poverty Prediction

We consider the problem of poverty prediction for household clusters in Uganda from satellite imagery and survey data. Each household cluster is represented by a collection of 400×400 satellite images (used as the context) and 65 categorical variables from living standards measurement survey (used as the interpretable attributes). The task is binary classification of the households into being either below or above the poverty line.

We follow the original study of Jean et al. (2016) and use a VGG-F network (pre-trained on nightlight intensity prediction) to compute 4096-dimensional embeddings of the satellite images on top of which we build contextual models. Note that this datasets is fairly small (500 training and 142 test points), and so we keep the VGG-F part frozen to avoid overfitting.

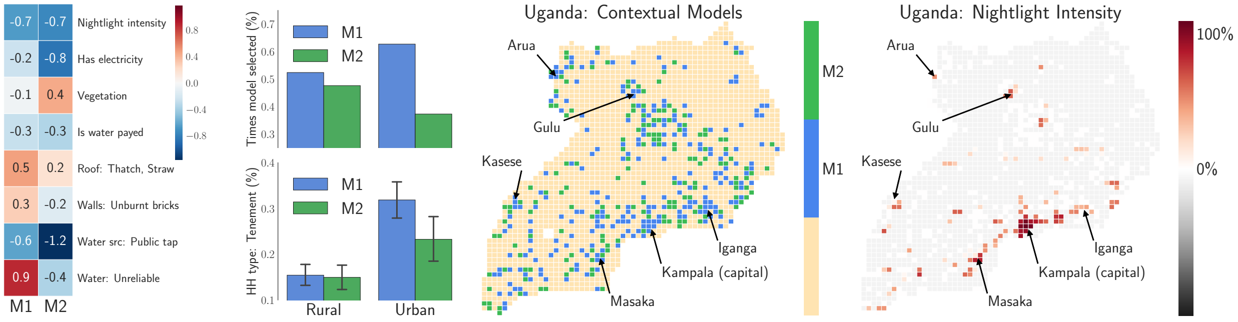
Models. For baselines, we use logistic regression (LR) and multi-layer perceptrons (MLP) with 1 hidden layer. The LR uses either VGG-F embeddings (LR_{emb}) or the categorical attributes ($\text{LR}_{\texttt{att}}$) as inputs. The input of the MLP is concatenated VGG-F embeddings and categorical attributes. Context encoder of the CEN model uses VGG-F to process images, followed by an attention layer over a dictionary of 16 trainable linear explanations defined over the categorical features (Figure 7.3). Finally, we evaluate a mixture-of-experts (MoE) model of the same architecture as CEN, since it is a special case (see Section 7.5.1). Both CEN and MoE are trained with the dictionary constraint and L_1 regularization over the dictionary elements to encourage sparse explanations.

Table 7.2: Performance on the poverty prediction task.

	Acc (↑)	**AUC (↑)**
LR_{emb}	62.5%	68.1%
$\text{LR}_{\texttt{att}}$	75.7%	82.2%
MLP	77.4%	78.7%
$\text{MoE}_{\texttt{att}}$	77.9%	**85.4%**
$\text{CEN}_{\texttt{att}}$	**81.5%**	84.2%

Performance. The results are presented in Table 7.2. Both in terms of accuracy and AUC, CEN models outperform both simple logistic regression and vanilla MLP. Even though the results suggest that categorical features are better predictors of poverty than VGG-F embeddings of images, note that using embeddings to *contextualize* linear models reduces the error. This indicates that *different* linear models are optimal in different contexts.

Qualitative analysis. We have discovered that, on this task, CEN encoder tends to sharply select one of the two explanations from the dictionary (denoted M1 and M2) for different household clusters in Uganda (Figure 7.5a). In the survey data, each household cluster is marked as either urban or rural. Conditional on a satellite image, CEN tends to pick M1 more often for urban areas and M2 for rural (Figure 7.5b). Notice that different explanations weigh categorical



(a) **(b)** **(c)** **(d)**

Figure 7.5: Qualitative results for the Satellite dataset: (a) Weights given to a subset of features by the two models (M1/M2) discovered by CEN. (b) How frequently M1/M2 are selected for areas marked rural or urban (top) and the average proportion of Tenement-type households in an urban/rural area for which M1/M2 was selected. (c) M1/M2 models selected for different areas on the Uganda map. M1 tends to be selected for urban areas while M2 is selected for the rest. (d) Nightlight intensity of different areas.

features, such as *reliability of the water source* or the *proportion of houses with walls made of unburnt brick*, quite differently. When visualized on the map, we see that CEN selects M1 more frequently around the major city areas (Figure 7.5c), which also correlates with high nightlight intensity in those areas (Figure 7.5d).

The estimated approximate conditional entropy of the binary targets (poor vs. not poor) given the selected model: $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta} = \mathrm{M1}) \approx 77\%$ and $\mathcal{H}(\mathbf{y} \mid \boldsymbol{\theta} = \mathrm{M2}) \approx 72\%$. The high performance of CEN along with high conditional entropy makes us confident in the produced explanations (Section 7.5.3) and allows us to draw conclusions about *what causes the model* to classify certain households in different neighborhoods as poor in terms of interpretable categorical variables.

Sentiment Analysis

The next problem we consider is sentiment prediction of IMDB reviews (Maas et al., 2011a). The reviews are given in the form of English text (sequences of words) and the sentiment labels are binary (good/bad movie). This dataset has 25k labelled reviews used for training and validation, 25k labelled reviews that are held out for test, and 50k unlabelled reviews.

Models. Following Johnson and Zhang (2016), we use a bi-directional LSTM with max-pooling as our baseline that predicts sentiment directly from text sequences. The same architecture is used as the context encoder in CEN that produces parameters for linear explanations. The explanations are applied to either (a) a bag-of-words (BoW) features (with a vocabulary limited to 2,000 most frequent words excluding English stop-words) or (b) a 200-dimensional topic representation produced by a separately trained off-the-shelf topic model (Blei et al., 2003).

Performance. Table 7.3 compares CEN with other models from the literature. Not only CEN achieves the state-of-the-art accuracy on this dataset in the supervised setting, it also outperforms or comes close to many of the semi-supervised methods. This indicates that the inductive biases

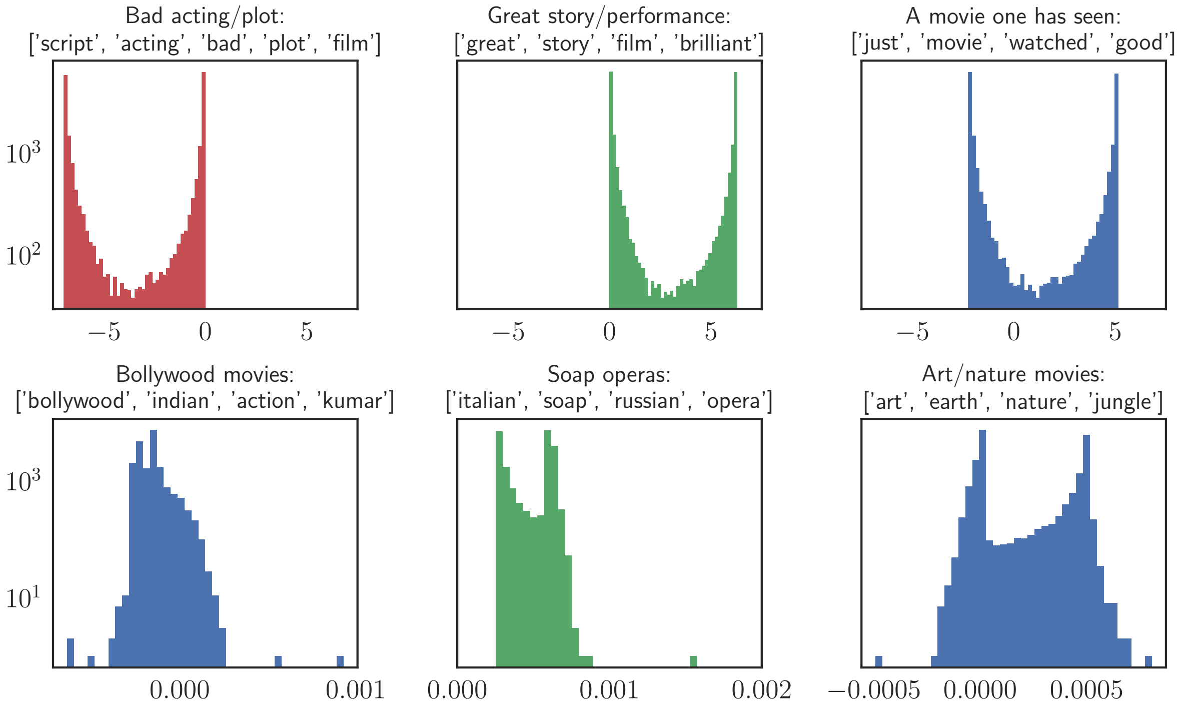


Figure 7.6: Histograms of test weights assigned by CEN to six different topics.

provided by the CEN architecture lead to a more significant performance improvement than most of the semi-supervised training methods on this dataset. We also remark that classifiers derived from large-scale language models pretrained on massive unsupervised corpora (*e.g.*, Gray et al., 2017; Howard and Ruder, 2018; Xie et al., 2019) have become popular and now dominate the leaderboard for this task.

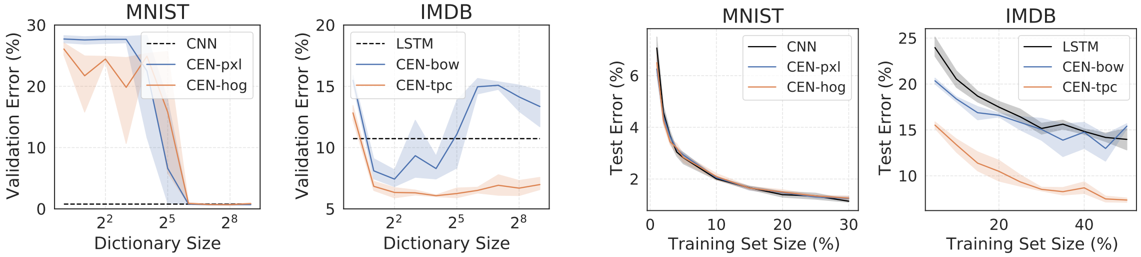
Qualitative analysis. After training CEN-`tpc` with linear explanations in terms of topics on the IMDB dataset, we generate explanations for each test example and visualize histograms of the weights assigned by the explanations to the 6 selected topics in Figure 7.6. The 3 topics in the top row are acting- and plot-related (and intuitively have positive, negative, or neutral connotation), while the 3 topics in the bottom are related to particular genre of the movies. Note that acting-related topics turn out to be bimodal, *i.e.*, contributing either positively, negatively, or neutrally to the sentiment prediction in different contexts. CEN assigns a high negative weight to the topic related to "bad acting/plot" and a high positive weight to "great story/performance" in most of the contexts (and treats those neutrally conditional on some of the reviews). Interestingly, genre-related topics almost always have a negligible contribution to the sentiment which indicates that the learned model does not have any particular bias towards or against a given genre.

Table 7.3: Sentiment classification error rate on IMDB dataset. The standard error (±) is based on 5 different runs. It is interesting to note that CENs establishes a new state of the art performance on the supervised prediction task while also outperforming or coming close to many of the semi-supervised methods that used additional 50k unlabeled reviews for pretraining. All current state of the art methods leverage large-scale pretraining (the bottom section of the table); these results are not directly comparable with methods trained on IMDB data only and included for completeness.

Reference	Method	Error (↓, %)
Supervised (trained on 25K labeled reviews only)		
Maas et al. (2011b)	Full + BoW (bnc)	11.67
Dahl et al. (2012)	WRRBM + BoW (bnc)	10.77
Wang and Manning (2012)	NBSVM-bi	8.78
Johnson and Zhang (2015a)	seq2-bow$n$-CNN	7.67
Johnson and Zhang (2015b)	oh-CNN (best)	8.39
Johnson and Zhang (2016)	oh-2LSTMp (best)	7.33
This work	CEN_{bow}	6.52 ± 0.15
	CEN_{tpc}	$\mathbf{6.24} \pm 0.12$
Semi-supervised (trained on 25K labeled + 50K unlabeled only)		
Maas et al. (2011b)	Full + Unlabeled + BoW	11.11
Le and Mikolov (2014)	Paragraph vectors	7.42
Dai and Le (2015)	wv-LSTM	7.24
Johnson and Zhang (2015b)	oh-CNN	6.51
Johnson and Zhang (2016)	oh-2LSTMp	5.94
Dieng et al. (2017)	TopicRNN	6.28
Miyato et al. (2016)	Virtual adversarial	5.94
This work	CEN_{bow}	—
	CEN_{tpc}	$\mathbf{5.48} \pm 0.09$
Semi-supervised via large-scale pre-training (massive external data)		
Gray et al. (2017)	block-sparse LSTM	5.01
Howard and Ruder (2018)	ULMFiT	4.60
Sachan et al. (2019)	Mixed-objective LSTM	4.32
Xie et al. (2019)	BERT-large	4.20
Haonan et al. (2019)	Graph Star	4.00

Table 7.4: Prediction error of the models on image classification tasks (averaged over 5 runs). The subscripts denote the features on which the linear models are built: pixels and HOG features.

MNIST (Error ↓, %)							**CIFAR10 (Error ↓, %)**						
LR_{px1}	LR_{hog}	CNN	MoE_{px1}	MoE_{hog}	CEN_{px1}	CEN_{hog}	LR_{px1}	LR_{hog}	VGG	MoE_{px1}	MoE_{hog}	CEN_{px1}	CEN_{hog}
8.00	2.98	**0.75**	1.23	1.10	**0.76**	**0.73**	60.1	48.6	9.4	13.0	11.7	9.6	**9.2**



(a) Validation error vs. dictionary size.

(b) Test error vs. training data size.

Figure 7.7: Analysis of the behavior of different CEN models with different dictionary sizes (varied between 1 and 512), feature types, trained on full or on a subset of the data. Shaded regions denote 95% CI based on 5 runs with different random seeds. (a) CEN is sensitive to the size of the dictionary—there is a critical size such that models with dictionaries smaller than that tend to significantly underperform. (b) Sample complexity of CENs. Models are trained with early stopping based on validation performance.

Image Classification

For the purpose of completeness, we also provide results on two classical image datasets: MNIST and CIFAR-10. For CEN, full images are used as the context; to imitate high-level features, we use (a) the original images cubically downscaled to 20×20 pixels, gray-scaled and normalized, and (b) HOG descriptors computed using 3×3 blocks (Dalal and Triggs, 2005). For each task, we use linear regression and vanilla convolutional networks as baselines (a small convnet for MNIST and VGG-16 for CIFAR-10). The results are reported in Table 7.4. CENs are competitive with the baselines and do not exhibit deterioration in performance. Visualization and analysis of the learned explanations is given in Appendix E.2.2 and the details on the architectures, hyperparameters, and training are given in Appendix E.2.3

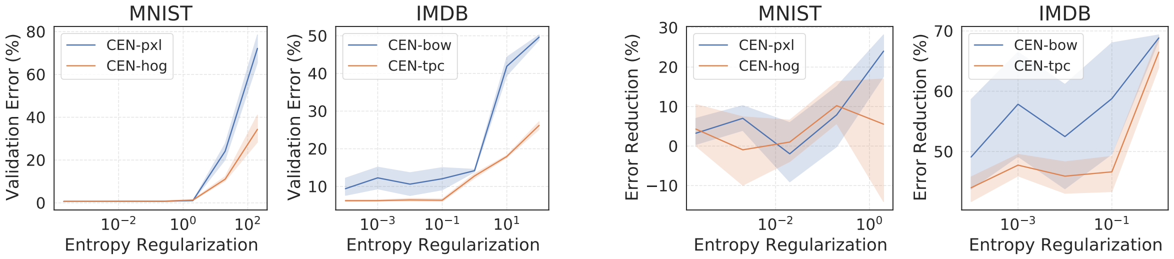
7.6.2 Properties of Explanations

In this section, we look at the explanations from the regularization and consistency point of view. As we show next, prediction via explanation not only has a strong regularization effect, but also always produces consistent locally linear models. Additionally, we analyze the effect of entropy regularization, quantify how much CEN's performance relies on explanations, and discuss computational considerations and tradeoffs for CEN and LIME.

Explanations as a Regularizer

By controlling the dictionary size, we can control the expressivity of the model class specified by CEN. For example, when the dictionary size is 1, CEN becomes equivalent to a linear model.[4] For larger dictionaries, CEN becomes as flexible as a deep network (Figure 7.7a). Adding a small

[4] Note that CENs with the dictionary size of 1 is still trained using stochastic optimization method as a neural network, which tends to yield a somewhat worse performance than the vanilla logistic regression.



(a) Validation error vs. entropy regularization.

(b) Expected contribution of the explanations.

Figure 7.8: The effects of entropy regularization on (a) the predictive performance of a CEN model and (b) the lower bound on the contribution of the explanations to the relative predictive error reduction. Shaded regions are 95% CI based on 5 runs with different random seeds.

sparsity penalty to each element of the dictionary (between 10^{-6} and 10^{-3}, see Appendix E.2.3) helps to avoid overfitting for very large dictionary sizes, so that the model learns to use only a few dictionary atoms for prediction while shrinking the rest to zero. Generally, dictionary size is a hyperparameter which optimal value depends on the data and the type of the interpretable features (*cf.*, CEN_{bow} and CEN_{tpc} on Figure 7.7a).

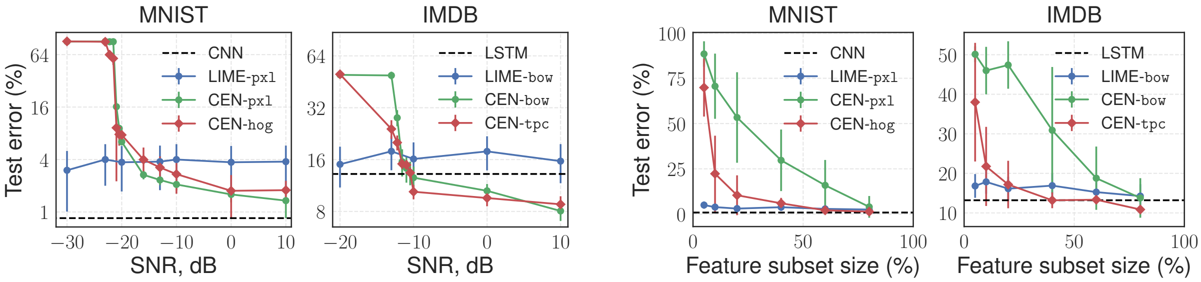
If explanations can act as a proper regularizer, we must observe improved sample efficiency of the model. To verify this, we trained CEN models on subsets of the data (size varied between 1% and 30% for MNIST and 2% and 50% for IMDB) with early stopping based on the validation performance. The test error on MNIST and IMDB for different training set sizes is presented on Figure 7.7b. On the IMBD dataset, CEN $_{\text{tpc}}$ required an order of magnitude fewer samples to match the baseline's performance, indicating efficient use of explanations for prediction. Note that such drastic sample efficiency gains were observed on IMDB only for CEN $_{\text{tpc}}$ (*i.e.*, when using topics as interpretable features); gains for CEN $_{\text{bow}}$ were noticeable but moderate; no sample efficiency gains were observed on MNIST for any of our CEN models.

Quantifying Contribution of the Explanations

Even though improved sample efficiency and regularizing effects of explanations indicate their non-trivial contribution indirectly, we wish to further quantify such contribution of explanations to the predictive performance of CEN. To do so, we run a set of experiments where we vary conditional entropy regularization coefficient and measure (a) performance of CEN on the validation set and (b) expected lower bound on the relative reduction of predictive error due to explanations, defined as:

$$[p\left(\hat{\mathbf{y}} \neq \mathbf{y} \mid \mathbf{c}\right) - p\left(\hat{\mathbf{y}} \neq \mathbf{y} \mid \mathbf{x}, \mathbf{c}\right)] \,/\, p\left(\hat{\mathbf{y}} \neq \mathbf{y} \mid \mathbf{c}\right) \tag{7.19}$$

As we have shown in Section 7.5.3, conditional entropy regularization encourages CEN models to learn context representations that are minimally correlated with the targets, and hence makes the model rely on the explanations rather than contextual information only. Figure 7.8a shows that entropy regularization generally does not affect predictive performance of a CEN



(a) Explanation test error vs. feature noise.

(b) Explanation test error vs. feature size.

Figure 7.9: The effect of feature quality the explanations: (a) Test error vs. the level of the noise added to the interpretable features. (b) Test error vs. the total number of interpretable features. Error bars: 95% CI.

model, unless the regularization coefficient becomes too large (*e.g.*, an order of magnitude larger than the predictive cross-entropy loss). Increasing conditional entropy regularization leads to CEN models whose performance relies more on explanations (Figure 7.8b). However, note that even without entropy regularization, explanations have a significant relative contribution to the reduction of the predictive error of CEN, ranging between 10-20% on MNIST and 40-60% on IMDB. This indicates that, while conditional entropy regularization is beneficial, even without it CEN still learns to generate meaningful, non-spurious explanations.

Consistency of Explanations

While regularization is a useful aspect, the main use case for explanations is model diagnostics. Linear explanations assign weights to the interpretable features, $\mathbf{x}$, and thus the quality of explanations depends on the quality of the selected features. In this section, we evaluate explanations generated by CEN and LIME (a post-hoc method). In particular, we consider two cases: (a) the features are corrupted with additive noise, and (b) the selected features are incomplete. For analysis, we use MNIST and IMDB datasets. Our key question is:

> *Can we trust the explanations built on noisy or incomplete features?*

The effect of noisy features. In this experiment, we inject noise[5] into the features $\mathbf{x}$ and ask LIME and CEN to fit explanations to the corrupted features. Note that after injecting noise, each data point has a noiseless representation $\mathbf{c}$ and a noisy $\mathbf{x}$. LIME constructs explanations by approximating the decision boundary of the baseline model trained to predict $\mathbf{y}$ from $\mathbf{c}$ features only. CEN is trained to construct explanations given $\mathbf{c}$ and then make predictions by applying explanations to $\mathbf{x}$. The predictive performance of the produced explanations on noisy features is given on Figure 7.9a. Since baselines take only $\mathbf{c}$ as inputs, their performance stays the same (dashed line). Regardless of the noise level, LIME overfits explanations—it is able to almost perfectly approximate the decision boundary of the baselines essentially using pure noise. On the other hand, performance of CEN degenerates with the increasing noise level indicating that the model fails to learn when the selected interpretable representation is of very low quality.

[5] We use Gaussian noise with zero mean and select variance for each signal-to-noise ratio level appropriately.

The effect of feature selection. Using the same setup, instead of injecting noise into $\mathbf{x}$, we construct $\mathbf{x}$ by randomly subsampling a set of dimensions.[6] Figure 7.9b demonstrates that while performance of CENs degrades proportionally to the size of $\mathbf{x}$ (*i.e.*, less informative features imply worse performance for CEN), we see that, again, LIME is again able to perfectly fit explanations to the decision boundary of the original models, despite the loss of information in the interpretable features $\mathbf{x}$.

These two experiments indicate a major drawback of explaining predictions post-hoc: when constructed on poor, noisy, or incomplete features, such explanations can overfit an arbitrary decision boundary of a predictor and are likely to be meaningless or misleading. For example, predictions of a perfectly valid model might end up getting absurd explanations which is unacceptable from the decision support point of view.[7] On the other hand, if we use CEN to generate explanations, high predictive performance would indicate presence of a meaningful signal in the selected interpretable features and explanations.

Computational Overhead and Considerations

Given all the advantages of CEN, such as often improved performance and consistency of linear explanations, what is the added computational overhead? It turns out that CEN compares quite favorably against the typical bundle solution: *a vanilla deep network plus a post-hoc explanation system* (e.g., *LIME*). The CEN architecture essentially adds a single bi-linear layer to the top of a network, resulting in a mild overhead of $O(D \times |\mathcal{X}|)$ multiplication and addition operations during the forward pass through the model. The training time overhead in aggregate does not exceed 20% when compared to a vanilla deep network of the same architecture (Table 7.5). Note that the models we used in our experiments are tiny by the modern standards, and we expect CEN's relative compute overhead to be even smaller for modern large-scale architectures. Also note that CENs generate explanations more than three orders of magnitude faster than LIME, manly because the latter has to solve an optimization problem for each instance of interest to obtain an explanation.

Table 7.5: Compute overhead.

Dataset	**CEN**	**LIME**
	Training time overhead	
MNIST	$18.6 \pm 1.7\%$	–
IMDB	$1.8 \pm 0.5\%$	–
Satellite	$0.4 \pm 0.1\%$	–
	Explanation time per instance	
MNIST	0.05 ± 0.03 ms	77 ± 9 ms
IMDB	0.07 ± 0.03 ms	38 ± 5 ms
Satellite	0.01 ± 0.01 ms	22 ± 6 ms

7.6.3 Solving Survival Analysis using CEN with Structured Outputs

In this final case study, we design CENs with structured explanations for survival prediction. We provide some background on survival analysis and the structured prediction approach proposed by Yu et al. (2011), then introduce CENs with linear CRF-based explanations for survival analysis, and conclude with experimental results on two datasets from the healthcare domain.

[6] Subsampling dimensions from $\mathbf{x}$ is done to resemble human subjectivity in selecting semantically meaningful features for model interpretation.

[7] Similar behavior has been observed in recent work that studied post-hoc explanation systems in adversarial settings (Dombrowski et al., 2019; Lakkaraju and Bastani, 2019).

Background on Survival Analysis via Structured Prediction

In survival time prediction, our goal is to estimate the risk and occurrence time of an undesirable event in the future (*e.g.*, death of a patient, earthquake, hard drive failure, customer turnover, etc.). A common approach is to model the *survival time*, T, either for a population (i.e., average survival time) or for each instance. Classical approaches, such as Aalen additive hazard (Aalen, 1989) and Cox proportional hazard (Cox, 1972) models, view survival analysis as continuous time prediction and hence a regression problem.

Alternatively, the time can be discretized into intervals (*e.g.*, days, weeks, etc.), and the survival time prediction can be converted into a multi-task classification problem (Efron, 1988). Taking this approach one step further, Yu et al. (2011) noticed that the output space of such a multitask classifier is structured in a particular way, and proposed a model called *sequence of dependent regressors*. The model is essentially a CRF with a particular structure of the pairwise potentials between the labels. We introduce the setup in our notation below.

Let the data instances be represented by tuples $(\mathbf{c}, \mathbf{x}, \mathbf{y})$, where targets are now sequences of m binary variables, $\mathbf{y} := (y^1, \dots, y^m)$, that indicate occurrence of an event at the corresponding time intervals.[8] If the event occurred at time $t \in [t_i, t_{i+1})$, then $y^j = 0,\ \forall j \leq i$ and $y^k = 1,\ \forall k > i$. If the event was *censored* (*i.e.*, we lack information for times after t), we represent targets $(y^{i+1}, \dots, y^m)$ with latent variables. Importantly, only $m + 1$ sequences are valid under these conditions, *i.e.*, assigned non-zero probability by the model. This suggests a linear CRF model defined as follows:

$$p\left(\mathbf{y} = (y^1, y^2, \dots, y^m) \mid \mathbf{x}, \boldsymbol{\theta}^{1:m}\right) \propto \exp\left\{\sum_{t=1}^{m} y^i(\mathbf{x}^\top \boldsymbol{\theta}^t) + \omega(y^t, y^{t+1})\right\} \tag{7.20}$$

The potentials between $\mathbf{x}$ and $y^{1:m}$ are linear functions parameterized by $\boldsymbol{\theta}^{1:m}$. The pairwise potentials between targets, $\omega(y_i, y_{i+1})$, ensure that non-permissible configurations where $(y_i = 1, y_{i+1} = 0)$ for some $i \in \{0, \dots, m-1\}$ are improbable (*i.e.*, $\omega(1, 0) = -\infty$ and $\omega(0, 0) = \omega_{00}$, $\omega(0, 1) = \omega_{01}$, $\omega(1, 1) = \omega_{10}$ are learnable parameters).

To train the model, Yu et al. (2011) optimize the following objective:

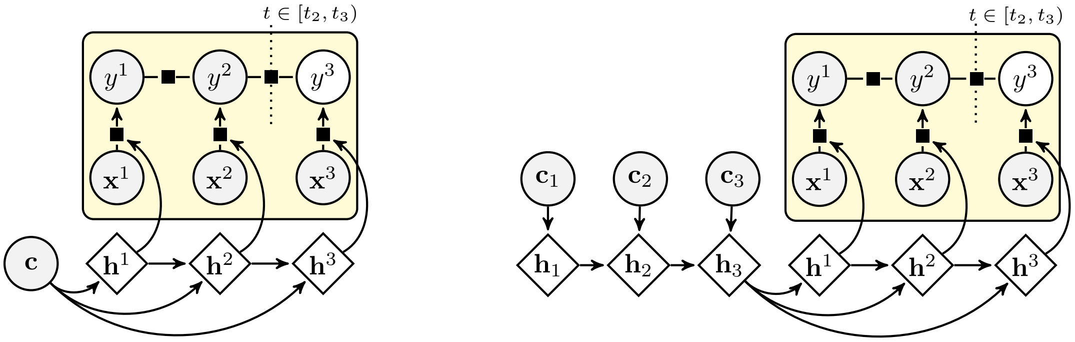
$$\min_{\boldsymbol{\Theta}} C_1 \sum_{t=1}^{m} \|\theta^t\|^2 + C_2 \sum_{t=1}^{m-1} \|\theta^{t+1} - \theta^t\|^2 - \mathcal{L}(\mathbf{y}, \mathbf{x}; \boldsymbol{\theta}^{1:m}) \tag{7.21}$$

where the first two terms are regularization and the last term is the log of the likelihood:

$$\mathcal{L}(\mathbf{y}, \mathbf{x}; \boldsymbol{\Theta}) = \sum_{i \in \text{NC}} \log p\left(T = t_i \mid \mathbf{x}_i, \boldsymbol{\Theta}\right) + \sum_{j \in \text{C}} \log p\left(T > t_j \mid \mathbf{x}_j, \boldsymbol{\Theta}\right) \tag{7.22}$$

where NC denotes the set of non-censored instances (for which we know the outcome times, t_i) and C is the set of censored inputs (for which we only know the censorship times, t_j).

[8] We assume that the occurrence time is lower bounded by $t_0 = 0$, upper bounded by some $t_m = T$, and discretized into intervals $[t_i, t_{i+1})$, where $i \in \{0, \dots, m-1\}$.



(a) Graphical model used for SUPPORT2.

(b) Graphical model used for PhysioNet.

Figure 7.10: CEN architectures used in our survival analysis experiments. Context encoders were (a) single hidden layer MLP and (b) single hidden layer LSTM. Encoders produced inputs for another LSTM over the output time intervals (denoted with $\mathbf{h}^1$, $\mathbf{h}^2$, $\mathbf{h}^3$ hidden states respectively).

The likelihood of an uncensored and a censored event at time $t \in [t_j, t_{j+1})$ are as follows:

$$\begin{aligned} p\left(T = t \mid \mathbf{x}, \boldsymbol{\theta}^{1:m}\right) &= \exp\left\{\sum_{i=j}^{m} \mathbf{x}^\top \boldsymbol{\theta}^i\right\} \Bigg/ \sum_{k=0}^{m} \exp\left\{\sum_{i=k+1}^{m} \mathbf{x}^\top \boldsymbol{\theta}^i\right\} \\ p\left(T \geq t \mid \mathbf{x}, \boldsymbol{\theta}^{1:m}\right) &= \sum_{k=j+1}^{m} \exp\left\{\sum_{i=k+1}^{m} \mathbf{x}^\top \boldsymbol{\theta}^i\right\} \Bigg/ \sum_{k=0}^{m} \exp\left\{\sum_{i=k+1}^{m} \mathbf{x}^\top \boldsymbol{\theta}^i\right\} \end{aligned} \tag{7.23}$$

CEN with Structured Explanations for Survival Analysis

To construct CEN for survival analysis, we follow the structured survival prediction setup described in the previous section. We define CEN with linear CRF explanations as follows:

$$\begin{aligned} &\boldsymbol{\theta}^t \sim p_{\mathbf{w}}\left(\boldsymbol{\theta}^t \mid \mathbf{c}\right), \mathbf{y} \sim p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^{1:m}\right), \\ &p\left(\mathbf{y} = (y^1, y^2, \ldots, y^m) \mid \mathbf{x}, \boldsymbol{\theta}^{1:m}\right) \propto \exp\left\{\sum_{t=1}^{m} y^i(\mathbf{x}^\top \boldsymbol{\theta}^t) + \omega(y^t, y^{t+1})\right\}, \\ &p_{\mathbf{w}}\left(\boldsymbol{\theta}^t \mid \mathbf{c}\right) := \delta(\boldsymbol{\theta}^t, \phi^t_{\mathbf{w},\mathbf{D}}(\mathbf{c})),\ \phi^t_{\mathbf{w},\mathbf{D}}(\mathbf{c}) := \boldsymbol{\alpha}(\mathbf{h}^t)^\top \mathbf{D},\ \mathbf{h}^t := \mathrm{RNN}(\mathbf{h}^{t-1}, \mathbf{c}) \end{aligned} \tag{7.24}$$

Note that an RNN-based context encoder generates different explanations for each time point, $\boldsymbol{\theta}^t$ (Figure 7.10). All $\boldsymbol{\theta}^t$ are generated using context- and time-specific attention $\boldsymbol{\alpha}(\mathbf{h}^t)$ over the dictionary $\mathbf{D}$. We adopt the training objective from Equation 7.21 with the same likelihood Equation 7.22. The model is a special case of CENs with structured explanations (Section 7.4.2).

Survival Analysis of Patients in Intense Care Units

We evaluate the proposed model against baselines on two survival prediction tasks.

Table 7.6: Performance of the baselines and CENs with structured explanations. The numbers are averages from 5-fold cross-validation; the std. are on the order of the least significant digit. "Acc@K" denotes accuracy at the K-th temporal quantile (see main text for explanation).

SUPPORT2					**PhysioNet Challenge 2012**				
Model	**Acc@25**	**Acc@50**	**Acc@75**	**RAE**	**Model**	**Acc@25**	**Acc@50**	**Acc@75**	**RAE**
Cox	84.1	73.7	47.6	0.90	Cox	93.0	69.6	49.1	0.24
Aalen	87.1	66.2	45.8	0.98	Aalen	93.3	78.7	57.1	0.31
CRF	84.4	89.3	79.2	0.59	CRF	93.2	85.1	65.6	0.14
MLP-CRF	**87.7**	89.6	80.1	0.62	LSTM-CRF	93.9	86.3	68.1	**0.11**
MLP-CEN	84.4	**96.2**	**83.3**	**0.52**	LSTM-CEN	**94.8**	**87.5**	**70.1**	**0.09**

Datasets. We use two publicly available datasets for survival analysis of of the intense care unit (ICU) patients: (a) SUPPORT2,[9] and (b) data from the PhysioNet 2012 challenge.[10] The data was preprocessed and used as follows:

- SUPPORT2: The data had 9105 patient records (7105 training, 1000 validation, 1000 test) and 73 variables. We selected 50 variables for both $\mathbf{c}$ and $\mathbf{x}$ features (*i.e.*, the context and the variables of interest were identical). Categorical features (such as *race* or *sex*) were one-hot encoded. The values of all features were non-negative, and we filled the missing values with -1 to preserve the information about missingness. For CRF-based predictors, we capped the survival timeline at 3 years and converted it into 156 discrete 7-day intervals.
- PhysioNet: The data had 4000 patient records, each represented by a 48-hour irregularly sampled 37-dimensional time-series of different measurements taken during the patient's stay at the ICU. We resampled and mean-aggregated the time-series at 30 min frequency. This resulted in a large number of missing values that we filled with 0. The resampled time-series were used as the context, $\mathbf{c}$. For the attributes, $\mathbf{x}$, we took the values of the last available measurement for each variable in the series. For CRF-based predictors, we capped the survival timeline at 60 days and converted into 60 discrete intervals.

Models. For baselines, we use classical Aalen and Cox models[11] and a CRF (Yu et al., 2011). All the baselines used $\mathbf{x}$ as their inputs. We combine CRFs with neural encoders in two ways:

(i) We apply CRFs to the outputs from the neural encoders (the models denoted MLP-CRF and LSTM-CRF).[12] Note that parameters of such CRF layer assign weights to the latent features and are not interpretable in terms of the attributes of interest.

(ii) We use CENs with CRF-based explanations, that process the context variables, $\mathbf{c}$, using the same neural networks as in (i) and output the sequence of parameters $\boldsymbol{\theta}^{1:m}$ for CRFs, while the latter act on the attributes, $\mathbf{x}$, to make structured predictions.

[9] http://biostat.mc.vanderbilt.edu/wiki/Main/DataSets.

[10] https://physionet.org/challenge/2012/.

[11] Implementation based on https://github.com/CamDavidsonPilon/lifelines.

[12] Similar models have been very successful in the natural language applications (Collobert et al., 2011).

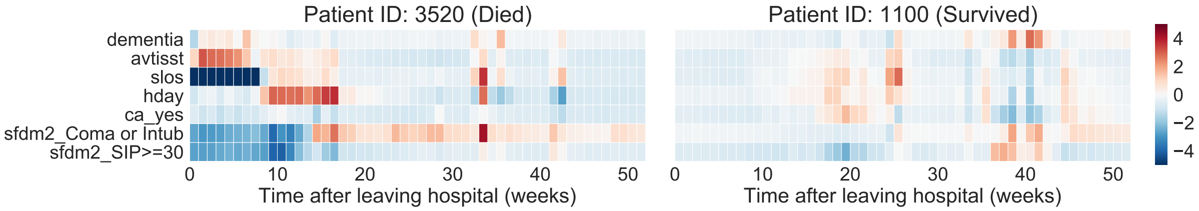


Figure 7.11: Weights of the CEN-generated CRF explanations for two patients from SUPPORT2 dataset for a set of the most influential features: `dementia` (comorbidity), `avtisst` (avg. TISS, days 3-25), `slos` (days from study entry to discharge), `hday` (day in hospital at study admit), `ca_yes` (the patient had cancer), `sfdm2_Coma or Intub` (intubated or in coma at month 2), `sfdm2_SIP` (sickness impact profile score at month 2). Higher weight values correspond to higher contributions to the risk.

More details on the architectures and training are given in Appendix E.2.3.

Metrics. Following Yu et al. (2011), we use two metrics specific to survival analysis:

(a) Accuracy of correctly predicting survival of a patient at times that correspond to 25%, 50%, 75% population-level temporal quantiles (*i.e.*, the time points such that the corresponding % of the population in the data were discharged from the study due to censorship or death).

(b) The relative absolute error (RAE) between the predicted and actual time of death for non-censored patients.

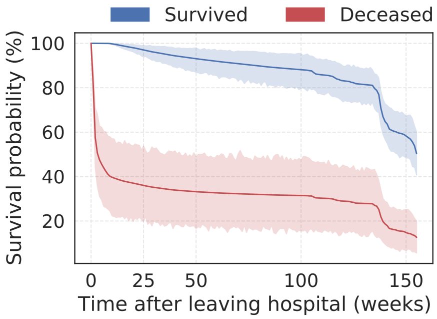


Figure 7.12: CEN-predicted survival curves for 100 random test patients from SUPPORT2. Color indicates death within 1 year after leaving the hospital. Shaded regions are 99% CI.

Performance. The results for all models are given in Table 7.6. Our implementation of the CRF baseline slightly improves upon the performance reported by Yu et al. (2011). MLP-CRF and LSTM-CRF improve upon plain CRFs but, as we noted, can no longer be interpreted in terms of the original variables. CENs outperform or closely match neural CRF models on all metrics while providing interpretable explanations for the predicted risk for each patient at each point in time.

Qualitative analysis. To inspect predictions of CENs qualitatively, for any given patient, we can visualize the weights assigned by the corresponding explanation to the respective attributes. Figure 7.11 shows weights of the explanations for a subset of the most influential features for two patients from SUPPORT2 dataset who were predicted as survivor/non-survivor. These temporal charts help us (a) to better understand which features the model selects as the most influential at each point in time, and (b) to identify potential inconsistencies in the model or the data—for example, using a chart as in Figure 7.11 we identified and excluded a feature (`hospdead`) from SUPPORT2 data, which initially was included but leaked information about the outcome as it directly indicated in-hospital death. Finally, explanations also allow us to better understand patient-specific temporal dynamics of the contributing factors to the survival rates predicted by the model (Figure 7.12).

7.7 Conclusion

In this chapter, we have introduced contextual explanation networks (CENs)—a class of models that learn to predict by generating and leveraging intermediate context-specific explanations. We have formally defined CENs as a class of probabilistic models, considered a number of special cases (*e.g.*, the mixture-of-experts model), and derived learning and inference algorithms within the encoder-decoder framework for simple and sequentially-structured outputs. We have shown that there are certain conditions when post-hoc explanations are erroneous and misleading. Such cases are hard to detect unless explanation is a part of the prediction process itself, as in CEN. Finally, learning to predict and to explain jointly turned out to have a number of benefits, including strong regularization, consistency, and ability to generate explanations with no computational overhead, as shown in our case studies.

7.7.1 Limitations

We would like to point out a few limitations of our approach and potential ways of addressing those in the future work. Firstly, while each prediction made by CEN comes with an explanation, the process of conditioning on the context is still uninterpretable. Ideas similar to context selection (Liu et al., 2017) or rationale generation (Lei et al., 2016) may help improve interpretability of the conditioning. Secondly, the space of explanations considered in this work assumes the same graphical model structure and parameterization for *all* explanations and uses a simple sparse dictionary constraint. This might be limiting, and one could imagine using a more hierarchically structured space of explanations instead, so that different types of explanations can be generated for different contexts. Nonetheless, we believe that the proposed class of models is useful not only for improving prediction capabilities, but also for model diagnostics, pattern discovery, and general data analysis, especially when machine learning is used for decision support in high-stakes applications.

7.7.2 Notable Follow-up Work

Since the first preprint of the original paper that this chapter is based on, multiple work have build on and/or extended CENs in various interesting ways:

- CENs have been successfully applied to digital pathology and enabled accurate discriminative subtyping of cancers (Lengerich et al., 2020).
- Multiple works have used the same mechanism of parameter generation as CEN for purposes other than interpretability (*e.g.*, Platanios et al., 2018; Stoica et al., 2020).
- Alvarez-Melis and Jaakkola (2018) introduced the so-called self-explaining networks, which is a simple extension of CEN that allows to learn interpretable features in addition to learning to generate explanations. Raghu et al. (2021) further extend interpretable features to arbitrary "supporting evidence" and tailored the model for clinical risk prediction.

Part III

Conclusion

Alice: Would you tell me, please, which way I ought to go from here?
The Cheshire Cat: That depends a good deal on where you want to get to.
Alice: I don't much care where.
The Cheshire Cat: Then it doesn't much matter which way you go.
Alice: ...So long as I get somewhere.
The Cheshire Cat: Oh, you're sure to do that, if only you walk long enough.

Lewis Carroll (Alice in Wonderland)

Chapter 8

Discussion

With a fast growing adoption of machine learning across many different application areas, the focus of the research community has shifted from solving well-defined statistical learning problems to building models that can be useful in challenging, real-world scenarios. Recognizing the importance and a central role that multitask learning plays today in solving such complex learning problems, we started this thesis with a simple question:

> *Is there a set of common principles that we could follow when designing models and algorithms for learning in a variety multitask settings that arise in practice?*

To answer this question constructively, we have developed a probabilistic framework (Chapter 3) that allowed us to represent arbitrary multitask learning problems using probabilistic graphical models and then solve them using the same posterior inference principle (or maximum likelihood as a special case). Further, we illustrated the generality of the proposed approach by applying it to multiple areas of machine intelligence, ranging from federated learning (Chapter 4), to multi-agent reinforcement learning (Chapter 5), to multilingual translation (Chapter 6), and even demonstrated that it can be used to improve interpretability of complex predictive models in settings that are classically considered non-multitask (Chapter 7).

On the one hand, the developed framework is general enough to encompass many existing multitask learning formulations and approaches, allowing us to reinterpret them from a probabilistic perspective and giving us a sense of the common underlying structure of different methods. On the other hand, it is concrete enough and provides with a general blueprint for solving a wide variety of multitask learning problems, which we used for designing new scalable learning algorithms, consistent loss functions, and accurate inference techniques that helped us improve upon the current state-of-the-art in each of the considered domains.

In this final chapter, we briefly summarize the key takeaways from the results of this thesis. We also step back and reflect on the philosophy behind the proposed probabilistic modeling approach, its advantages, limitations, and alternative perspectives. Finally, we conclude with a discussion of interesting directions for future work.

8.1 Takeaways, Advantages, and Limitations

In this thesis, we claimed that representing complex learning problems as collections of learning tasks and modeling them probabilistically is worthwhile endeavor. To support our claim, we presented a general probabilistic multitask modeling framework and four case studies with multiple different theoretical, algorithmic, and empirical contributions. Without enumerating all the contributions (for details, see the respective chapters), we highlight two key advantages:

1. First, our probabilistic approach provides a level of abstraction that allows us to initially reason about a multitask learning problem in a domain-agnostic manner. At this level of abstraction, we only need to specify all the variables that describe the problem and the presence or absence of functional dependencies between them. Without committing to any specific functional representations or computational techniques, our modeling framework can be used to derive loss functions and learning algorithms with certain desirable properties (*e.g.*, consistent zero-shot generalization, convergence guarantee, etc.).
2. Second, the loss functions (or generic learning algorithms) obtained initially are would be typically computationally intractable. Thus, the next step is always to figure out a tractable approximation of the initial loss function (or algorithm), which would typically require domain-specific techniques (*e.g.*, a particular neural architecture, a data preprocessing technique, etc.). In other words, instead of innovating on the problem formulation end (*e.g.*, designing ad-hoc loss functions), our framework provides a consistent formulation and allows to redirect the research effort into figuring out the important computational aspects.

Our framework also has multiple limitations. Perhaps the biggest limitation of our approach is that it requires someone to specify the tasks and dependencies between them before using framework. When tasks are unknown *a priori* or hard to specify (*e.g.*, specifying reward functions in RL is non-trivial), the framework may not be as useful or even applicable.

8.2 Where Do We Go from Here?

There are many directions we and others can pursue in the follow up work, including some concrete ideas and open problems listed at the end of each chapter in Part II. On a higher level, however, we see further *automation of multitask learning* as one of the most exciting directions.

What do we mean by *automation*? Our probabilistic framework can be seen as a tool for systematizing and partially automating some of the efforts of researchers and practitioners when solving multitask learning problems (specifically, our framework, by and large, takes care of the problem formulation). As a next step, we can imagine a fully automated system: given a problem formulation (*e.g.*, in natural language form or as a spec in some structured domain-specific language), it is translated automatically into a probabilistic representation, then "compiled" into a learning algorithm that satisfies certain properties (*e.g.*, generalization guarantees) and is ready to be executed on a particular hardware. Another step in that direction of automation would be to automate decomposition of different complex learning problems into multiple tasks.

Appendices

Appendix A

Probabilistic Multitask Modeling: Additional Details

A.1 Proofs

The meta-generalization bounds provided in Theorem 3.2 in Section 3.3.2 directly extend a classical result by Maurer (2005), which in turn uses meta-learning formulation of Baxter (2000) and is a direct adaptation of the algorithmic stability bounds of Bousquet and Elisseeff (2002). The result is provided in the theorem below.[1]

Theorem A.1: Maurer (2005), Theorem 1

Let the meta-algorithm $\mathbb{A}$ and ℓ loss satisfy the following two conditions:

C1. For every pair of meta-samples $\mathbb{S} = \{S_1, \dots, S_n\}$, $\mathbb{S}^{-i} := \mathbb{S} \setminus \{S_i\}$, and for any sample S, we have $|\hat{R}(\mathbb{A}(\mathbb{S}), S) - \hat{R}(\mathbb{A}(\mathbb{S}^{-i}), S)| \leq \beta'$.

C2. For any pair of samples $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$, $S^{-j} := S \setminus \{(x_j, y_j)\}$, any algorithm A produced by $\mathbb{A}$, and any (x, y), we have $|\ell(A(S)(x), y) - \ell(A(S^{-j})(x), y)| \leq \beta$.

Then for any task distribution $\mathbb{P}$, with probability at least $1 - \delta$ the following holds:

$$\mathfrak{R}(\mathbb{A}(\mathbb{S}), \mathbb{P}) - \mathcal{L}_{\text{emp}}(\mathbb{A}(\mathbb{S}), \mathbb{S}) \leq 2\beta' + (4n\beta' + M)\sqrt{\frac{\ln(1/\delta)}{2n}} + 2\beta, \tag{A.1}$$

where $\mathcal{L}_{\text{emp}}(\mathbb{A}(\mathbb{S}), \mathbb{S}) := \frac{1}{n}\sum_{i=1}^{n} \hat{R}(\mathbb{A}(\mathbb{S}), S_i)$, $\hat{R}(A, S_i) := \frac{1}{|S_i|}\sum_{(x,y)\in S_i} \ell(A(S_i)(x), y)$ with the loss function ℓ bounded by M.

Conditions C1 and C2 in Theorem A.1 define uniform stability (*i.e.*, sensitivity of the algorithm to removal of an arbitrary point from the training sample (Bousquet and Elisseeff, 2002)) and state

[1] In Section 3.3.2, our discussion was focused on optimizing the variational free energy of a probabilistic model of a distribution of tasks. Here, theorems are formulated in terms of general continuous and smooth loss functions.

that the bound holds if the meta-algorithm $\mathbb{A}$ and every algorithm A it produces are uniformly β'- and β-stable with respect to the empirical risk $\hat{R}$ and a loss function ℓ, respectively. The bound becomes non-trivial when $\beta' = o(1/n^a), a \geq 1/2$ and $\beta = o(1/m^b), b \geq 0$.

Theorem A.1 provides a bound on the difference between the transfer risk $\mathcal{R}[\mathbb{A}(\mathbb{S}), \mathbb{P}]$ and its empirical estimator $\mathcal{L}_{\text{emp}}(\mathbb{A}(\mathbb{S}), \mathbb{S})$ based on meta-sample $\mathbb{S}$, implying that a small $\mathcal{L}_{\text{emp}}(\mathbb{A}, \mathbb{S})$ guarantees meta-generalization within the bound. Denoting $\mathcal{A} \equiv \mathbb{A}(\mathbb{S})$ to simplify our notation, the bound is obtained as follows:

$$\mathcal{R}(\mathcal{A}, \mathbb{P}) - \mathcal{L}_{\text{emp}}(\mathcal{A}, \mathbb{S}) = \mathbb{E}_{\mathcal{D} \sim \mathbb{P}(\mathcal{T})}\left[\mathbb{E}_{S \sim \mathcal{D}^m}\left[\hat{R}(\mathcal{A}, S)\right]\right] - \frac{1}{n}\sum_{i=1}^{n} \hat{R}(\mathcal{A}, S_i) + \tag{A.2}$$

$$\mathbb{E}_{\mathcal{D} \sim \mathbb{P}(\mathcal{T})}\left[\mathbb{E}_{S \sim \mathcal{D}^m}\left[R(\mathcal{A}(S), \mathcal{D}) - \hat{R}(\mathcal{A}, S)\right]\right] \tag{A.3}$$

The term (A.2) is the difference between the expected empirical risk over the true distribution of tasks and its estimate $\mathcal{L}_{\text{emp}}(\mathcal{A}, \mathbb{S})$ based on the meta-sample $\mathbb{S}$. As long as $\mathbb{A}$ is β'-uniformly stable with respect to $\hat{R}(\mathcal{A}, S)$ (C1, Theorem A.1), this term is bounded by $2\beta' + (4n\beta' + M)\sqrt{\ln(1/\delta)/2n}$, which follows from the classical result of Bousquet and Elisseeff (2002).

The term in Equation A.3 is the estimation error of a model $f(\cdot) = \mathcal{A}(S)$ learned by $\mathcal{A}$ from S with respect to the data distribution, computed in expectation over the distribution of tasks $\mathbb{P}$. Stability of the inner-loop (C2, Theorem A.1) directly implies a bound of 2β on this term (see Maurer (2005), Theorem 6). Putting together bounds of terms in Equations A.2 and A.3, we arrive at Equation A.1.

Bounding Meta-generalization of Modern Meta-learning Algorithms

The bound given in Equation A.1 is on the generalization error, *i.e.*, the deviation of the true transfer risk $\mathcal{R}$ from the empirical estimator $\mathcal{L}_{\text{emp}}$, and has meaningful practical implications only when the meta-algorithm $\mathbb{A}$ minimizes $\mathcal{L}_{\text{emp}}$. Note that $\mathcal{L}_{\text{emp}}(\mathcal{A}, \mathbb{S})$ is the meta-training objective function optimized by methods such as FedAvg (McMahan et al., 2017) or Reptile (Nichol et al., 2018), and thus the bound from Theorem A.1 applies to them directly. However, MAML, ProtoNets, and their variations optimize $\mathcal{L}_Q(\mathcal{A}, \mathbb{S})$, so we have to bound $\mathcal{R}(\mathcal{A}, \mathbb{P}) - \mathcal{L}_Q(\mathcal{A}, \mathbb{S})$ instead, which can be decomposed into two terms similar to Equations A.2 and A.3, where $\hat{R}$ is replaced by $\hat{R}_Q$ and S is replaced by $S \setminus Q$ (since samples from the query set Q are not used in the inner-loop). The bound on the first term will not change much as we can still directly apply results from stability theory with the only caveat that we would require β'_Q-uniform stability of the meta-algorithm with respect to $\hat{R}_Q$. The second term, however, vanishes:

$$\begin{aligned}
&\mathbb{E}_S\left[R(\mathcal{A}(S \setminus Q), T) - \hat{R}_Q(\mathcal{A}, S)\right] \\
&= \mathbb{E}_{S \setminus Q}\left[R(\mathcal{A}(S \setminus Q), T) - \mathbb{E}_Q\left[\frac{1}{|Q|}\sum_{(x,y) \in Q} \ell(\mathcal{A}(S \setminus Q)(x), y)\right]\right] \equiv 0
\end{aligned} \tag{A.4}$$

This allows us to reformulate Theorem A.1 and obtain the following generalization bound applicable to any meta-learning method that optimizes $\hat{R}_Q$ in the outer loop, including MAML and ProtoNets.

Theorem A.2

Let the meta-algorithm $\mathbb{A}$ and ℓ loss satisfy the following two conditions:

C1. For every pair of meta-samples $\mathbb{S} = \{S_1, \dots, S_n\}$, $\mathbb{S}^{-i} := \mathbb{S} \setminus \{S_i\}$, and for any sample S, we have $|\hat{R}_Q(\mathbb{A}(\mathbb{S}), S) - \hat{R}_Q(\mathbb{A}(\mathbb{S}^{-i}), S)| \leq \beta'_Q$.

C2. For any pair of samples $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$, $S^{-j} := S \setminus \{(x_j, y_j)\}$, any algorithm A produced by $\mathbb{A}$, and any (x, y), we have $|\ell(A(S)(x), y) - \ell(A(S^{-j})(x), y)| \leq \beta$.

Then for any task distribution $\mathbb{P}$, with probability at least $1 - \delta$ the following holds:

$$\mathcal{R}(\mathbb{A}(\mathbb{S}), \mathbb{P}) - \mathcal{L}_Q(\mathbb{A}(\mathbb{S}); \mathbb{S}) \leq 2\beta'_Q + (4n\beta'_Q + M)\sqrt{\frac{\ln(1/\delta)}{2n}}, \tag{A.5}$$

where $\mathcal{L}_Q(\mathbb{A}(\mathbb{S}); \mathbb{S}) := \frac{1}{n}\sum_{i=1}^{n} \hat{R}_Q(\mathbb{A}(\mathbb{S}), S_i)$, $\hat{R}_Q(A, S_i) := \frac{1}{|Q_i|}\sum_{(x,y)\in Q_i} \ell(A(S_i\, Q_i)(x), y)$ with the loss function ℓ bounded by M.

Since MAML, Reptile, and ProtoNets use stochastic gradient method (SGM) for solving the outer loop optimization problem, and Reptile additionally uses SGM in the inner loop as well, we further adopt the following result from stability theory of SGM due to Hardt et al. (2015).

Lemma A.1: Hardt et al. (2015), Theorem 3.12

Let $\ell(\cdot, z) \in [0, 1]$ be L-Lipschitz and γ-smooth loss function for every z. Suppose that we optimize $\frac{1}{n}\sum_{i=1}^{n} \ell(\theta, z_i)$ by running SGM for T steps with monotonically non-increasing step sizes $\alpha_t \leq c/t$. Then, SGM is β-uniformly stable with

$$\beta \leq \frac{1 + 1/(\gamma c)}{n - 1}(2cL^2)^{1/(\gamma c+1)} T^{1-1/(\gamma c+1)} \tag{A.6}$$

Combining Theorems A.1 and A.2 and Lemma A.1 we finally arrive at the meta-generalization error bounds for modern meta-learning algorithms.

Theorem A.3

Let the meta-algorithm $\mathbb{A}$ be an SGM that optimizes an L'-Lipschitz and γ'-smooth loss $\mathcal{L}(\mathcal{A}, \mathbb{S})$ by taking T' steps with non-increasing step sizes $\alpha'_t \leq c'/t$. With probability at least $1 - \delta$, we have the following:

1. If $\mathcal{L}(\mathcal{A}, \mathbb{S})$ is Q-estimator of the transfer risk, then the following bound holds:

$$\mathfrak{R}[\mathcal{A}, \mathbb{P}] - \mathcal{L}(\mathcal{A}, \mathbb{S}) \leq B'(n, T', L', \gamma', c') \approx O\left(L'^2 T' \sqrt{\frac{\ln(1/\delta)}{n}}\right) \tag{A.7}$$

2. If $\mathcal{L}(A; \mathbb{S})$ is the empirical estimator of the transfer risk and the inner loop learning algorithm A is an SGM that optimizes L-Lipschitz and γ-smooth loss $\ell(f(x), y)$ by

taking T steps with non-increasing step sizes $\alpha_t \leq c/t$, then:

$$\begin{aligned} \mathfrak{R}[\mathcal{A}, \mathbb{P}] - \mathcal{L}(\mathcal{A}, \mathbb{S}) &\leq B'(n, T', L', \gamma', c') + B(m, T, L, \gamma, c) \\ &\leq O\left(L'^2 T' \sqrt{\frac{\ln(1/\delta)}{n}} + L^2 T \frac{1}{m} \right) \end{aligned} \tag{A.8}$$

Proof. Conditions of the theorem and Lemma A.1 imply that $\mathbb{A}$ is β'-(or β'_Q-)uniformly stable and the coefficient can be expressed through the Lipschitz and smoothness constants of $\mathcal{L}_{\text{emp}}$ (or $\mathcal{L}_Q$). This leads to the following expression for $B'(n, T', L', \gamma', c')$:

$$B'(n, T', L', \gamma', c') = \frac{2C}{n} \left(1 + \frac{1}{n-1} \right) + 2C \sqrt{\frac{2 \ln(1/\delta)}{n}} \left(1 + \frac{1}{n-1} + \frac{M}{4C} \right), \tag{A.9}$$

where $C := (1 + 1/(\gamma' c'))(2c' L'^2)^{1/(\gamma' c' + 1)} T'^{1 - 1/(\gamma' c' + 1)}$. The simplified expression given in Equation A.7 upper-bounds the one given in Equation A.9. Similarly, if each algorithm A produced by the meta-algorithm $\mathbb{A}$ is an SGM on the $\mathcal{L}_{\text{emp}}$ objective, using Lemma A.1 we arrive at the following expression for $B(m, T, L, \gamma, c)$:

$$B(m, T, L, \gamma, c) = 2\beta \leq 2 \frac{1 + 1/(\gamma c)}{m - 1} (2cL^2)^{1/(\gamma c + 1)} T^{1 - 1/(\gamma c + 1)} \approx O\left(L^2 T \frac{1}{m} \right) \tag{A.10}$$

where the approximation ignores terms associated with c and γ. The statement of the theorem now follows from Theorems A.1 and A.2 and the derived expressions. □

Appendix B

Inferential Federated Learning: Additional Details

B.1 Preliminary Analysis and Ablations

In Chapter 4, we derived federated posterior averaging (FedPA) starting with the global posterior decomposition (Proposition 4.1, which is exact) and applying the following three approximations:

1. The Laplace approximation of the local and global posterior distributions.
2. The shrinkage estimation of the local moments.
3. Approximate sampling from the local posteriors using MCMC.

We have also observed that FedAvg is a special case of FedPA (from the algorithmic point of view), since it can be viewed as also using the Laplace approximation for the posteriors, but estimating local covariances $\hat{\Sigma}_i$'s with identities and local means using the final iterates of local SGD.

In this section, we analyze the effects of approximations 2 and 3 on the convergence of FedPA. Specifically, we first discuss the convergence rates of FedAvg and FedPA as *biased* stochastic gradient optimization methods (Ajalloeian and Stich, 2020). We show how the bias and variance of the client deltas behave for FedAvg and FedPA as functions of the number samples. We also analyze the quality of samples produced by IASG (Mandt et al., 2017) and how they depend on the amount of local computation and hyperparameters. Our analyses are conducted empirically on synthetic data.

B.1.1 Discussion of the Convergence of FedPA vs. FedAvg

First, observe that if each client is able to perfectly estimate their $\boldsymbol{\Delta}_i = \Sigma_i^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu})$, the problem solved by Algorithm 4.1 simply becomes an optimization of a quadratic objective using unbiased stochastic gradients, $\boldsymbol{\Delta} := \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{\Delta}_i$. The noise in the gradients in this case comes from the fact that the server interacts with only a small subset of M out of N clients in each round. This is a classical stochastic optimization problem with well-known convergence rates under some assumptions on the norm of the stochastic gradients (*e.g.*, Nemirovski et al., 2009). The rate of convergence for SGD with a $\mathcal{O}(t^{-1})$ decaying learning rate used on the server is $\mathcal{O}(1/\sqrt{t})$. It can be further improved to $\mathcal{O}(1/t)$ using Polyak momentum (Polyak, 1964) or iterate averaging (Polyak and Juditsky, 1992).

In reality, both FedAvg and FedPA produce biased estimates $\hat{\mathbf{\Delta}}_{\text{FedAvg}}$ and $\hat{\mathbf{\Delta}}_{\text{FedPA}}$, respectively. Thus, we can analyze the problem as SGD with biased stochastic gradient estimates and let $\hat{\mathbf{\Delta}}_t := \nabla F(\boldsymbol{\theta}_t) + \mathbf{b}(\boldsymbol{\theta}_t) + \mathbf{n}(\boldsymbol{\theta}_t)$ where $\mathbf{b}(\boldsymbol{\theta}_t)$ and $\mathbf{n}(\boldsymbol{\theta}_t, \xi)$ are bias and noise terms. Following Ajalloeian and Stich (2020), we can further assume that the bias and noise terms are norm-bounded as follows.

Assumption B.1 (Bounded bias). *There exist constants* $0 \le m < 1$ *and* $\zeta^2 \ge 0$ *such that*

$$\|\mathbf{b}(\boldsymbol{\theta})\|^2 \le m\|\nabla F(\boldsymbol{\theta})\|^2 + \zeta^2, \quad \forall \boldsymbol{\theta} \in \mathbb{R}^d. \tag{B.1}$$

Assumption B.2 (Bounded noise). *There exist constants* $0 \le M < 1$ *and* $\sigma^2 \ge 0$ *such that*

$$\mathbb{E}_\xi\left[\|\mathbf{n}(\boldsymbol{\theta}, \xi)\|^2\right] \le M\|\nabla F(\boldsymbol{\theta})\|^2 + \sigma^2, \quad \forall \boldsymbol{\theta} \in \mathbb{R}^d. \tag{B.2}$$

Under these general assumptions, the following convergence result holds.

Theorem B.1: Ajalloeian and Stich (2020), Theorem 2

Let $F(\boldsymbol{\theta})$ be L-smooth. Then SGD with a learning rate $\alpha := \min\left\{\frac{1}{L}, \frac{1-m}{2ML}, \left(\frac{LF}{\sigma^2 T}\right)^{1/2}\right\}$ and gradients that satisfy Assumptions B.1 and B.2 achieves the vicinity of a stationary point, $\mathbb{E}\left[\|\nabla F(\boldsymbol{\theta})\|^2\right] = \mathcal{O}\left(\varepsilon + \frac{\zeta^2}{1-m}\right)$, in T iterations, where

$$T = \mathcal{O}\left(\frac{1}{\varepsilon}\left[1 + \frac{M}{1-m} + \frac{\sigma^2}{\varepsilon(1-m)}\right]\right)\frac{LF}{1-m}. \tag{B.3}$$

Note that SGD with biased gradients is able to converge to a vicinity of the optimum determined by the bias term $\zeta^2/(1-m)$. For FedAvg, since the bias is not countered, this term determines the distance between the stationary point and the true global optimum. For FedPA, since $\hat{\mathbf{\Delta}}_{\text{FedPA}} \to \mathbf{\Delta}$ with more local samples, the bias should vanish as we increase the amount of local computation.

Determining the precise statistical dependence of the gradient bias on the local samples is beyond the scope of this work. However, to gain more intuition about the differences in behavior of FedPA and FedAvg, below we conduct an empirical analysis of the bias and variance of the estimated client deltas on synthetic least squares problems, for which exact deltas can be computed analytically.

B.1.2 Analysis of the Quality of IASG-based Sampling and Covariance

The more and better samples we can obtain locally, the lower the bias and variance of the gradients of $Q(\boldsymbol{\theta})$ will be, resulting in faster convergence to a fixed point closer to the global optimum. For local sampling, we proposed to use a variant of SG-MCMC called Iterate Averaged Stochastic Gradient (IASG) developed by Mandt et al. (2017), given in Algorithm 4.3. The algorithm generates samples by simply averaging every K intermediate iterates produced by a client optimizer (typically, SGD with some a fixed learning rate α) after skipping the first B iterates as a burn-in phase.[1]

How good are the samples produced by IASG and how do different parameters of the algorithm affect the quality of the samples? To answer this question, we run IASG on synthetic least squares problems, for which we can compute the actual posterior distribution and measure the quality of the samples by evaluating the effective sample size (ESS, Liu, 1996; Owen, 2013). Given ℓ approximate posterior samples $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_\ell\}$, the ESS statistic can be computed as follows:

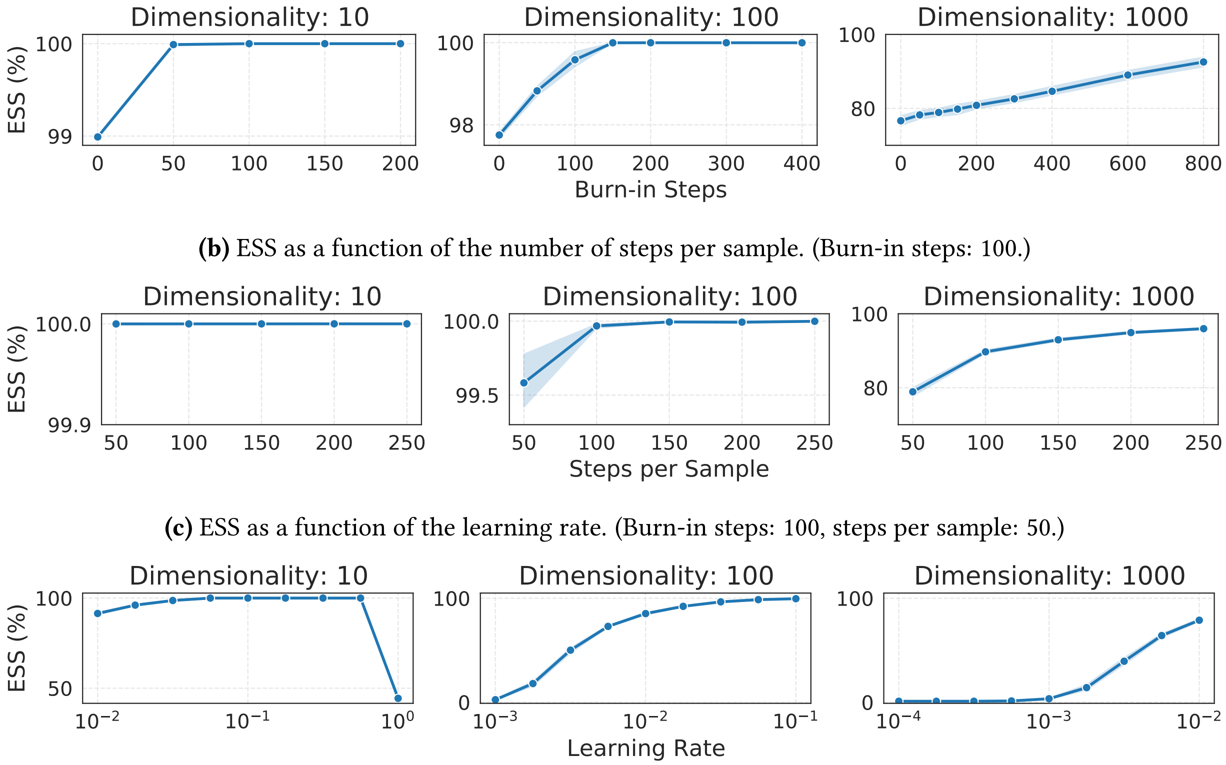
$$\mathrm{ESS}\left(\{\boldsymbol{\theta}_i\}_{j=1}^{\ell}\right) := \left(\sum_{j=1}^{\ell} w_j\right)^2 \bigg/ \sum_{j=1}^{\ell} w_j^2 \,,$$

where weights w_j must be proportional to the posterior probabilities, or equivalently to the loss.

Effects of the dimensionality, the number of data points, and IASG parameters on ESS. The results of our synthetic experiments are presented below in Figure B.1. The takeaways are as follows:

- More burn-in steps (or epochs) generally improve the quality of samples.
- The larger the number of steps per sample the better (less correlated) the samples are.
- The learning rate is the most sensitive and important hyperparameter—if too large, IASG might diverge (happened in the 1000 dimensional case); if too small, the samples become correlated.
- Finally, the quality of the samples deteriorates with the increase in the number of dimensions.

[1] Note that in our experiments in Section 4.5, instead of using B local steps for burn-in at each round, we used several *initial rounds* as burn-in-only rounds, running FedPA in the FedAvg regime.



(a) ESS as a function of the number of burn-in steps. (Steps per sample: 50.)

(b) ESS as a function of the number of steps per sample. (Burn-in steps: 100.)

(c) ESS as a function of the learning rate. (Burn-in steps: 100, steps per sample: 50.)

Figure B.1: The ESS statistics for samples produced by IASG on random synthetic least squares linear regression problems of dimensionality 10, 100, 1000. Total number of data points per problem: 500, batch size: 10. In (a) and (b) the learning rate was set to 0.1 for 10 and 100 dimensions, and 0.01 for 1000 dimensions.

B.2 Additional Proofs

Proposition 4.2: Global Posterior Inference

The global posterior mode $\boldsymbol{\mu}^\star$ given in Equation 4.5 is the minimizer of a quadratic $Q(\boldsymbol{\theta}) := \frac{1}{2}\boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta} - \mathbf{b}^\top\boldsymbol{\theta}$, where $\mathbf{A} := \sum_{i=1}^N w_i \boldsymbol{\Sigma}_i^{-1}$ and $\mathbf{b} := \sum_{i=1}^N w_i \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i$.

Proof. The statement of the proposition (implicitly) assumes that all matrix inverses exist. Then, the quadratic $Q(\boldsymbol{\theta})$ is positive definite (PD) since $\mathbf{A}$ is PD as a convex combination of PD matrices $\boldsymbol{\Sigma}_i^{-1}$. Thus, the quadratic has a unique solution $\boldsymbol{\theta}^\star$ where the gradient of the objective vanishes:

$$\mathbf{A}\boldsymbol{\theta}^\star - \mathbf{b} = 0 \quad \Rightarrow \quad \boldsymbol{\theta}^\star = \mathbf{A}^{-1}\mathbf{b} = \left(\sum_{i=1}^N q_i \boldsymbol{\Sigma}_i^{-1}\right)^{-1} \sum_{i=1}^N q_i \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i \equiv \boldsymbol{\mu}, \tag{B.4}$$

which implies that $\boldsymbol{\mu}$ is the unique minimizer of $Q(\boldsymbol{\theta})$. □

B.3 Efficient Computation of Client Deltas

In this section, we provide a constructive proof for the following theorem by designing an efficient algorithm for computing $\hat{\boldsymbol{\Delta}}_\ell := \hat{\boldsymbol{\Sigma}}_\ell^{-1}(\boldsymbol{\theta} - \hat{\boldsymbol{\mu}}_\ell)$ on the clients in time and memory linear in the number of dimensions d of the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$.

Theorem 4.1: Linear-time Computation of Client Deltas

Given s approximate posterior samples $\{\hat{\boldsymbol{\theta}}_1, \dots, \hat{\boldsymbol{\theta}}_s\}$, let $\hat{\boldsymbol{\mu}}_s$ be the sample mean, $\hat{\mathbf{S}}_s$ be the sample covariance, and $\hat{\boldsymbol{\Sigma}}_s := \rho_s \mathbf{I} + (1-\rho_s)\hat{\mathbf{S}}_s$ be a shrinkage estimator (Ledoit and Wolf, 2004a) of the covariance with $\rho_s := 1/(1+(s-1)\rho)$ for some $\rho \in [0, +\infty)$. Then, for any $\boldsymbol{\theta}$, we can compute $\hat{\boldsymbol{\Delta}}_s = \hat{\boldsymbol{\Sigma}}_s^{-1}(\boldsymbol{\theta} - \hat{\boldsymbol{\mu}}_s)$ in $\mathcal{O}(s^2 d)$ time and using $\mathcal{O}(sd)$ memory.

The naïve computation of update vectors (*i.e.*, where we first estimate $\hat{\boldsymbol{\mu}}_\ell$ and $\hat{\boldsymbol{\Sigma}}_\ell$ from posterior samples and use them to compute deltas) requires $\mathcal{O}(d^2)$ storage and $\mathcal{O}(d^3)$ compute on the clients and is both computationally and memory intractable. We derive an algorithm that, given ℓ posterior samples, computes $\hat{\boldsymbol{\Delta}}_\ell$ using only $\mathcal{O}(\ell d)$ memory and $\mathcal{O}(\ell^2 d)$ compute.

The algorithm makes use of the following two components:

1. The shrinkage estimator of the covariance (Ledoit and Wolf, 2004a), which is known to be well-conditioned even in high-dimensional settings (*i.e.*, when the number of samples is smaller than the number of dimensions) and is widely used in econometrics (Ledoit and Wolf, 2004b) and computational biology (Schäfer and Strimmer, 2005).
2. Incremental computation of $\hat{\boldsymbol{\Sigma}}_\ell^{-1}(\boldsymbol{\theta}_\ell - \hat{\boldsymbol{\mu}}_\ell)$ that exploits the fact that each new posterior sample only adds a rank-1 component to $\hat{\boldsymbol{\Sigma}}_\ell$ and applies the Sherman-Morrison formula to derive a dynamic program for updating $\hat{\boldsymbol{\Delta}}_\ell$.

Notation. For the sake of this discussion, we denote $\boldsymbol{\theta}$ (*i.e.*, the server state broadcasted to the clients at round t) as $\mathbf{x}_0$, drop the client index i, denote posterior samples as $\mathbf{x}_j$, sample mean as $\bar{\mathbf{x}}_\ell := \frac{1}{\ell}\sum_{j=1}^{\ell} \mathbf{x}_j$, and sample covariance as $\hat{\mathbf{S}}_\ell := \frac{1}{\ell-1}\sum_{j=1}^{\ell}(\mathbf{x}_j - \bar{\mathbf{x}}_\ell)(\mathbf{x}_j - \bar{\mathbf{x}}_\ell)^\top$.

B.3.1 The Shrinkage Estimator of the Covariance

Ledoit and Wolf (2004a) proposed to estimate a high-dimensional covariance matrix using a convex combination of identity and sample covariance (known as the shrinkage estimator):

$$\hat{\boldsymbol{\Sigma}}_\ell(\rho_\ell) := \rho_\ell \mathbf{I} + (1-\rho_\ell)\mathbf{S}_\ell, \tag{B.5}$$

where ρ_ℓ is a scalar parameter that controls the bias-variance tradeoff of the estimator. As an aside, while ρ_ℓ can be arbitrary and the optimal ρ_ℓ requires knowing the true covariance $\boldsymbol{\Sigma}$, there are near-optimal ways to estimate $\hat{\rho}_\ell$ from the samples (Chen et al., 2010), which we discuss at the end of this section.

In this section, we focus on deriving an expression for ρ_t as a function of $t = 1, \dots, \ell$ that ensures that the difference between $\hat{\boldsymbol{\Sigma}}_t$ and $\hat{\boldsymbol{\Sigma}}_{t-1}$ is a rank-1 matrix.

Derivation of a shrinkage estimator that admits rank-1 updates. Consider a matrix:

$$\tilde{\boldsymbol{\Sigma}}_t := \mathbf{I} + \beta_t \hat{\mathbf{S}}_t, \tag{B.6}$$

where β_t is a scalar function of $t = 1, 2, \dots, \ell$. We would like to find β_t such that $\tilde{\boldsymbol{\Sigma}}_t = \tilde{\boldsymbol{\Sigma}}_{t-1} + \gamma_t \mathbf{U}_t$, where $\mathbf{U}_t$ is a rank-1 matrix, *i.e.*, the following equality should hold:

$$\beta_t \hat{\mathbf{S}}_t = \beta_{t-1}\hat{\mathbf{S}}_{t-1} + \gamma_t \mathbf{U}_t \tag{B.7}$$

To determine the functional form of β_t, we need recurrent relationships for $\bar{\mathbf{x}}_t$ and $\hat{\mathbf{S}}_t$. For the former, note that the following relationship holds for two consecutive estimates of the sample mean, $\bar{\mathbf{x}}_{t-1}$ and $\bar{\mathbf{x}}_t$:

$$\bar{\mathbf{x}}_t = \frac{(t-1)\bar{\mathbf{x}}_{t-1} + \mathbf{x}_t}{t} = \bar{\mathbf{x}}_{t-1} + \frac{1}{t}(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1}) \tag{B.8}$$

This allows us to expand $\hat{\mathbf{S}}_t$ as follows:

$$\begin{aligned}
(t-1)\hat{\mathbf{S}}_t &= \sum_{j=1}^{t}(\mathbf{x}_j - \bar{\mathbf{x}}_t)(\mathbf{x}_j - \bar{\mathbf{x}}_t)^\top \\
&= \sum_{j=1}^{t}\left(\mathbf{x}_j - \bar{\mathbf{x}}_{t-1} - \frac{\mathbf{x}_t - \bar{\mathbf{x}}_{t-1}}{t}\right)\left(\mathbf{x}_j - \bar{\mathbf{x}}_{t-1} - \frac{\mathbf{x}_t - \bar{\mathbf{x}}_{t-1}}{t}\right)^\top \\
&= \underbrace{\sum_{j=1}^{t-1}\left(\mathbf{x}_j - \bar{\mathbf{x}}_{t-1}\right)\left(\mathbf{x}_j - \bar{\mathbf{x}}_{t-1}\right)^\top}_{=(t-2)\hat{\mathbf{S}}_{t-1}} - 2\frac{\mathbf{x}_t - \bar{\mathbf{x}}_{t-1}}{t}\underbrace{\sum_{j=1}^{t-1}\left(\mathbf{x}_j - \bar{\mathbf{x}}_{t-1}\right)^\top}_{=0} + \qquad \text{(B.9)} \\
&\quad \frac{t-1}{t^2}(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})^\top + \left(\frac{t-1}{t}\right)^2(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})^\top \\
&= (t-2)\hat{\mathbf{S}}_{t-1} + \frac{t-1}{t}(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})^\top
\end{aligned}$$

Thus, we have the following recurrent relationship between $\hat{\mathbf{S}}_t$ and $\hat{\mathbf{S}}_{t-1}$:

$$\hat{\mathbf{S}}_t = \left(\frac{t-2}{t-1}\right)\hat{\mathbf{S}}_{t-1} + \frac{1}{t}(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})^\top \tag{B.10}$$

Now, we can plug (B.10) into (B.7) and obtain the following equation:

$$\beta_t\left(\frac{t-2}{t-1}\right)\hat{\mathbf{S}}_{t-1} + \frac{\beta_t}{t}(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})^\top = \beta_{t-1}\mathbf{S}_{t-1} + \gamma_t\mathbf{U}_t, \tag{B.11}$$

which implies that $\mathbf{U}_t := (\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})^\top$, $\gamma_t := \beta_t/t$, and the following telescoping expressions for β_t:

$$\beta_t = \left(\frac{t-1}{t-2}\right)\beta_{t-1} = \left(\frac{t-1}{\cancel{t-2}}\cdot\frac{\cancel{t-2}}{t-3}\right)\beta_{t-2} = \cdots = (t-1)\beta_2, \tag{B.12}$$

where we set $\beta_2 \equiv \rho \in [0, +\infty)$ to be a constant. Thus, if we define $\tilde{\mathbf{\Sigma}}_t := \mathbf{I} + \rho(t-1)\hat{\mathbf{S}}_t$, then the following recurrent relationships will hold:

$$\begin{aligned}
\tilde{\mathbf{\Sigma}}_1 &= \mathbf{I}, \\
\tilde{\mathbf{\Sigma}}_2 &= \mathbf{I} + \rho\hat{\mathbf{S}}_2 = \tilde{\mathbf{\Sigma}}_1 + \frac{\rho}{2}(\mathbf{x}_2 - \bar{\mathbf{x}}_1)(\mathbf{x}_2 - \bar{\mathbf{x}}_1)^\top, \\
\tilde{\mathbf{\Sigma}}_3 &= \mathbf{I} + 2\rho\hat{\mathbf{S}}_3 = \tilde{\mathbf{\Sigma}}_2 + \frac{2\rho}{3}(\mathbf{x}_3 - \bar{\mathbf{x}}_2)(\mathbf{x}_3 - \bar{\mathbf{x}}_2)^\top, \\
&\dots \\
\tilde{\mathbf{\Sigma}}_t &= \mathbf{I} + (t-1)\rho\hat{\mathbf{S}}_{t-1} = \tilde{\mathbf{\Sigma}}_{t-1} + \frac{(t-1)\rho}{t}(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})(\mathbf{x}_t - \bar{\mathbf{x}}_{t-1})^\top
\end{aligned} \tag{B.13}$$

Finally, we obtain a shrinkage estimator of the covariance from $\tilde{\mathbf{\Sigma}}_n$ by normalizing coefficients:

$$\hat{\mathbf{\Sigma}}_t := \underbrace{\frac{1}{1+(t-1)\rho}}_{\rho_t}\mathbf{I} + \underbrace{\frac{(t-1)\rho}{1+(t-1)\rho}}_{1-\rho_t}\hat{\mathbf{S}}_t = \rho_t\tilde{\mathbf{\Sigma}}_t \tag{B.14}$$

Note that $\hat{\mathbf{\Sigma}}_1 \equiv \mathbf{I}$ and $\hat{\mathbf{\Sigma}}_t \to \mathbf{S}_t$ as $t \to \infty$.

B.3.2 Computing Deltas using Dynamic Programming

Since $\hat{\mathbf{\Sigma}}_\ell$ is proportional to $\tilde{\mathbf{\Sigma}}_\ell$ and the latter satisfies recurrent rank-1 updates given in Equation B.13, denoting $\mathbf{u}_\ell := \mathbf{x}_\ell - \bar{\mathbf{x}}_{\ell-1}$, we can express $\hat{\mathbf{\Sigma}}_\ell^{-1} = \tilde{\mathbf{\Sigma}}_\ell^{-1}/\rho_\ell$ using the Sherman-Morrison formula:

$$\tilde{\mathbf{\Sigma}}_\ell^{-1} = \tilde{\mathbf{\Sigma}}_{\ell-1}^{-1} - \frac{\gamma_\ell\left(\tilde{\mathbf{\Sigma}}_{\ell-1}^{-1}\mathbf{u}_\ell\mathbf{u}_\ell^\top\tilde{\mathbf{\Sigma}}_{\ell-1}^{-1}\right)}{1+\gamma_\ell\left(\mathbf{u}_\ell^\top\tilde{\mathbf{\Sigma}}_{\ell-1}^{-1}\mathbf{u}_\ell\right)} \tag{B.15}$$

Note that we would like to estimate $\hat{\boldsymbol{\Delta}}_\ell := \hat{\boldsymbol{\Sigma}}_\ell^{-1}(\mathbf{x}_0 - \bar{\mathbf{x}}_\ell)$, which can be done without computing or storing any matrices if we know $\tilde{\boldsymbol{\Sigma}}_{\ell-1}^{-1}\mathbf{u}_\ell$ and $\tilde{\boldsymbol{\Sigma}}_{\ell-1}^{-1}(\mathbf{x}_0 - \bar{\mathbf{x}}_\ell)$.

Denoting $\tilde{\boldsymbol{\Delta}}_t := \tilde{\boldsymbol{\Sigma}}_t^{-1}(\mathbf{x}_0 - \bar{\mathbf{x}}_t)$, and knowing that $\mathbf{x}_0 - \bar{\mathbf{x}}_\ell = (\mathbf{x}_0 - \bar{\mathbf{x}}_{\ell-1}) - \mathbf{u}_\ell/\ell$ (which follows from Equation B.8), we can compute $\hat{\boldsymbol{\Delta}}_\ell$ using the following recurrence:

$$\tilde{\boldsymbol{\Delta}}_1 := \mathbf{x}_0 - \bar{\mathbf{x}}_1, \quad \mathbf{v}_{1,2} := \mathbf{x}_2 - \bar{\mathbf{x}}_1, \qquad \text{// initial conditions} \tag{B.16}$$

$$\mathbf{u}_t := \mathbf{x}_t - \bar{\mathbf{x}}_{t-1}, \quad \mathbf{v}_{t-1,t} := \tilde{\boldsymbol{\Sigma}}_{t-1}^{-1}\mathbf{u}_t \qquad \text{// recurrence for } \mathbf{u}_t \text{ and } \mathbf{v}_{t-1,t} \tag{B.17}$$

$$\tilde{\boldsymbol{\Delta}}_t = \tilde{\boldsymbol{\Delta}}_{t-1} - \left[1 + \frac{\gamma_t\left(t\mathbf{u}_t^\top\tilde{\boldsymbol{\Delta}}_{t-1} - \mathbf{u}_t^\top\mathbf{v}_{t-1,t}\right)}{1 + \gamma_t\left(\mathbf{u}_t^\top\mathbf{v}_{t-1,t}\right)}\right]\frac{\mathbf{v}_{t-1,t}}{t} \qquad \text{// recurrence for } \tilde{\boldsymbol{\Delta}}_t \tag{B.18}$$

$$\hat{\boldsymbol{\Delta}}_t = \tilde{\boldsymbol{\Delta}}_t/\rho_t \qquad \text{// final step for } \hat{\boldsymbol{\Delta}}_t \tag{B.19}$$

Remember that our goal is to avoid storing $d \times d$ matrices throughout the computation. In the above recursive equations, all expressions depend only on vector-vector products except the one for $\mathbf{v}_{t-1,t}$ which needs a matrix-vector product. To express the latter one in the form of vector-vector products, we need another 2-index recurrence on $\mathbf{v}_{i,j} := \tilde{\boldsymbol{\Sigma}}_i^{-1}\mathbf{u}_j$:

$$\mathbf{v}_{1,2} = \mathbf{u}_2, \quad \mathbf{v}_{1,3} = \mathbf{u}_3, \quad \ldots \quad \mathbf{v}_{1,t} = \mathbf{u}_t \qquad \text{// initial conditions} \tag{B.20}$$

$$\mathbf{v}_{t-1,t} = \left[\tilde{\boldsymbol{\Sigma}}_{t-2}^{-1} - \frac{\gamma_{t-1}\left(\tilde{\boldsymbol{\Sigma}}_{t-2}^{-1}\mathbf{u}_{t-1}\mathbf{u}_{t-1}^\top\tilde{\boldsymbol{\Sigma}}_{t-2}^{-1}\right)}{1 + \gamma_{t-1}\left(\mathbf{u}_{t-1}^\top\tilde{\boldsymbol{\Sigma}}_{t-2}^{-1}\mathbf{u}_{t-1}\right)}\right]\mathbf{u}_t \qquad \text{// Sherman-Morrison} \tag{B.21}$$

$$= \mathbf{v}_{t-2,t} - \frac{\gamma_{t-1}\left(\mathbf{v}_{t-2,t-1}^\top\mathbf{u}_t\right)}{1 + \gamma_{t-1}\left(\mathbf{v}_{t-2,t-1}^\top\mathbf{u}_{t-1}\right)}\mathbf{v}_{t-2,t-1} \tag{B.22}$$

$$= \mathbf{v}_{1,t} - \sum_{k=2}^{t-1}\frac{\gamma_k\left(\mathbf{v}_{k-1,k}^\top\mathbf{u}_t\right)}{1 + \gamma_k\left(\mathbf{v}_{k-1,k}^\top\mathbf{u}_k\right)}\mathbf{v}_{k-1,k} \qquad \text{// final expression for } \mathbf{v}_{t-1,t} \tag{B.23}$$

Now, equipped with these two recurrences, given a stream of samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots$, we compute $\hat{\boldsymbol{\Delta}}_t$ for $t \geq 2$ based on $\mathbf{x}_t$, $\{\mathbf{u}_k\}_{k=1}^{t-1}$, $\{\mathbf{v}_{k-2,k-1}\}_{k=1}^{t-1}$ and $\hat{\boldsymbol{\Delta}}_{t-1}$ using two steps:

1. Compute $\mathbf{u}_t$ and $\mathbf{v}_{t-1,t}$ using the second recurrence.
2. Compute $\hat{\boldsymbol{\Delta}}_t$ from $\mathbf{u}_t$, $\mathbf{v}_{t-1,t}$, and $\hat{\boldsymbol{\Delta}}_{t-1}$ using the first recurrence.

For each new sample in the sequence, we repeat the two steps to obtain the updated $\hat{\boldsymbol{\Delta}}_t$ estimate, until we have processed all ℓ samples. Note that the first step requires $\mathcal{O}(t)$ vector-vector multiplies, *i.e.*, $\mathcal{O}(td)$ compute, and $\mathcal{O}(d)$ memory, and the second step a $\mathcal{O}(1)$ number of vector-vector multiplies. As a result, the computational complexity of estimating $\hat{\boldsymbol{\Delta}}_\ell$ is $\mathcal{O}(\ell^2 d)$ and the storage needed for the dynamic programming state represented by a tuple $\left(\{\mathbf{u}_k\}_{k=1}^{t-1}, \{\mathbf{v}_{k-2,k-1}\}_{k=1}^{t-1}, \hat{\boldsymbol{\Delta}}_{t-1}\right)$ is $\mathcal{O}(\ell d)$.

The any-time property of the resulting algorithm. Interestingly, the above algorithm is online as well as *any-time* in the following sense: as we keep sampling more from the posterior, the estimate of $\hat{\boldsymbol{\Delta}}$ keeps improving, but if stopped at any time, the algorithm still produces the best possible estimate under the given time constraint. If the posterior sampler is stopped during the burn-in phase or after having produced only 1 posterior sample, the returned delta will be identical to FedAvg. By spending more compute on the clients (and a bit of extra memory), with each additional posterior sample $\mathbf{x}_t$, we have $\hat{\boldsymbol{\Delta}}_t \xrightarrow[t\to\infty]{} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_0 - \boldsymbol{\mu})$.

Optimal selection of ρ. Note that to be able to run the above described algorithm in an online fashion, we have to select and commit to a ρ before seeing any samples. Alternatively, if the online and any-time properties of the algorithm are unnecessary, we can first obtain ℓ posterior samples $\{\mathbf{x}_k\}_{k=1}^{\ell}$, then infer a near-optimal $\hat{\rho}_\star$ from these samples—*e.g.*, using the Rao-Blackwellized version of the LW estimator (RBLW) or the oracle approximating shrinkage (OAS), both proposed and analyzed by Chen et al. (2010)—and then use the inferred $\hat{\rho}_\star$ to compute the corresponding delta using our dynamic programming algorithm.

B.4 Details on the Experimental Setup

In this part, we provide additional details on our experimental setup, including a more detailed description of the datasets and tasks, models, methods, and hyperparameters.

B.4.1 Datasets, Tasks, and Models

Statistics of the datasets used in our empirical study can be found in Table 4.2. All the datasets and tasks considered in our study are a subset of the tasks introduced by Reddi et al. (2020).

EMNIST-62. The dataset is comprised of 28×28 images of handwritten digits and lower and upper case English characters (62 different classes total). The federated version of the dataset was introduced by Caldas et al. (2018), and is partitioned by the author of each character. The heterogeneity of the dataset is coming from the different writing style of each author. We use this dataset for the character recognition task, termed EMNIST CR in Reddi et al. (2020) and the same model architecture, which is a 2-layer convolutional network with 3×3 kernel, max pooling, and dropout, followed by a 128-unit fully connected layer. The model was adopted from the TensorFlow Federated library: https://bit.ly/3l41LKv.

CIFAR-100. The federated version of CIFAR-100 was introduced by Reddi et al. (2020). The training set of the dataset is partitioned among 500 clients, 100 data points per client. The partitioning was created using a two-step latent Dirichlet allocation (LDA) over to “coarse” to “fine” labels which created a label distribution resembling a more realistic federated setting. For the model, also following Reddi et al. (2020), we used a modified ResNet-18 with group normalization layer instead of batch normalization, as suggested by Hsieh et al. (2019). The model was adopted from the TensorFlow Federated library: https://bit.ly/33jMv6g.

StackOverflow. The dataset consists of text (questions and answers) asked and answered by the total of 342,477 unique users, collected from https://stackoverflow.com. The federated version of the dataset partitions it into clients by the user. In addition, questions and answers in the dataset

Table B.1: Selected optimizers for each task. For SGD, m is momentum. For Adam, $\beta_1 = 0.9, \beta_2 = 0.99$.

Hyperparameter	**EMNIST-62**	**CIFAR-100**	**StackOverflow NWP**	**StackOverflow LR**
ServerOpt	SGD ($m = 0.9$)	SGD ($m = 0.9$)	Adam ($\tau = 10^{-3}$)	Adagrad ($\tau = 10^{-5}$)
ClientOpt	SGD ($m = 0.9$)	SGD ($m = 0.9$)	SGD ($m = 0.0$)	SGD ($m = 0.9$)
# clients p/round	100	20	10	10

have associated metadata, which includes tags. We consider two tasks introduced by Reddi et al. (2020): the next word prediction task (NWP) and the tag prediction task via multi-label logistic regression. The vocabulary of the dataset is restricted to 10,000 most frequently used words for each task (*i.e.*, the NWP task becomes a multi-class classification problem with 10,000 classes). The tags are similarly restricted to 500 most frequent ones (*i.e.*, the LR task becomes a multi-label classification proble with 500 labels).

For tag prediction, we use a simple linear regression model where each question or answer are represented by a normalized bag-of-words vector. The model was adopted from the TensorFlow Federated library: https://bit.ly/2EXjAeY.

For the NWP task, we restrict each client to the first 128 sentences in their dataset, perform padding and truncation to ensure that sentences have 20 words, and then represent each sentence as a sequence of indices corresponding to the 10,000 frequently used words, as well as indices representing padding, out-of-vocabulary (OOV) words, beginning of sentence (BOS), and end of sentence (EOS). We note that accuracy of next word prediction is measured only on the content words and *not* on the OOV, BOS, and EOS symbols. We use an RNN model with 96-dimensional word embeddings (trained from scratch), 670-dimensional LSTM layer, followed by a fully connected output softmax layer. The model was adopted from the TensorFlow Federated library: https://bit.ly/2SoSi3X.

B.4.2 Methods

As mentioned in the main text, we used FedAvg with adaptive server optimizers with 1 or multiple local epochs per client as our baselines. For each task, we selected the best server optimizer based on the results reported by Reddi et al. (2020), given in Table B.1. We emphasize, even though we refer to all our baseline methods as FedAvg, the names of the methods as given by Reddi et al. (2020) should be FedAvgM for EMNIST-62 and CIFAR-100, FedAdam for StackOverflow NWP and FedAdagrad for StackOverflow LR. Another difference between our baselines and Reddi et al. (2020) is that we ran SGD *with momentum* on the clients for EMNIST-62, CIFAR-100, and StackOverflow LR, as that improved performance of the methods with multiple epochs per client.

Our FedPA methods used the same configurations as FedAvg baselines; moreover, FedPA and FedAvg were identical (algorithmically) during the burn-in phase and only different in the client-side computation during the sampling phase of FedPA.

Table B.2: Hyperparameter grids for each task.

Hyperparameter	**EMNIST-62**	**CIFAR-100**	**StackOverflow NWP**	**StackOverflow LR**
Server learning rate	$\{0.01, 0.05, 0.1, 0.5, 1, 5\}$		$\{0.01, 0.05, 0.1, 0.5, 1\}$	$\{0.1, 0.5, 1, 5, 10\}$
Client learning rate	$\{0.001, 0.005, 0.01, 0.05, 0.1\}$		$\{0.01, 0.05, 0.1\}$	$\{1, 5, 10, 50, 100\}$
Client epochs	$\{2, 5, 10, 20\}$			
FedPA burn-in	$\{100, 200, 400, 600, 800\}$			
FedPA shrinkage	$\{0.0001, 0.001, 0.01, 0.1, 1\}$			

Table B.3: The best selected hyperparameters for each task.

Hyperparameter	**EMNIST-62**	**CIFAR-100**	**StackOverflow NWP**	**StackOverflow LR**
Server learning rate	0.5	0.5	1.0	5.0
Client learning rate	0.01	0.01	0.1	50.0
Client epochs	5	10	5	5
FedPA burn-in	100	400	800	800
FedPA shrinkage	0.1	0.01	0.01	0.01

B.4.3 Hyperparameters and Grids

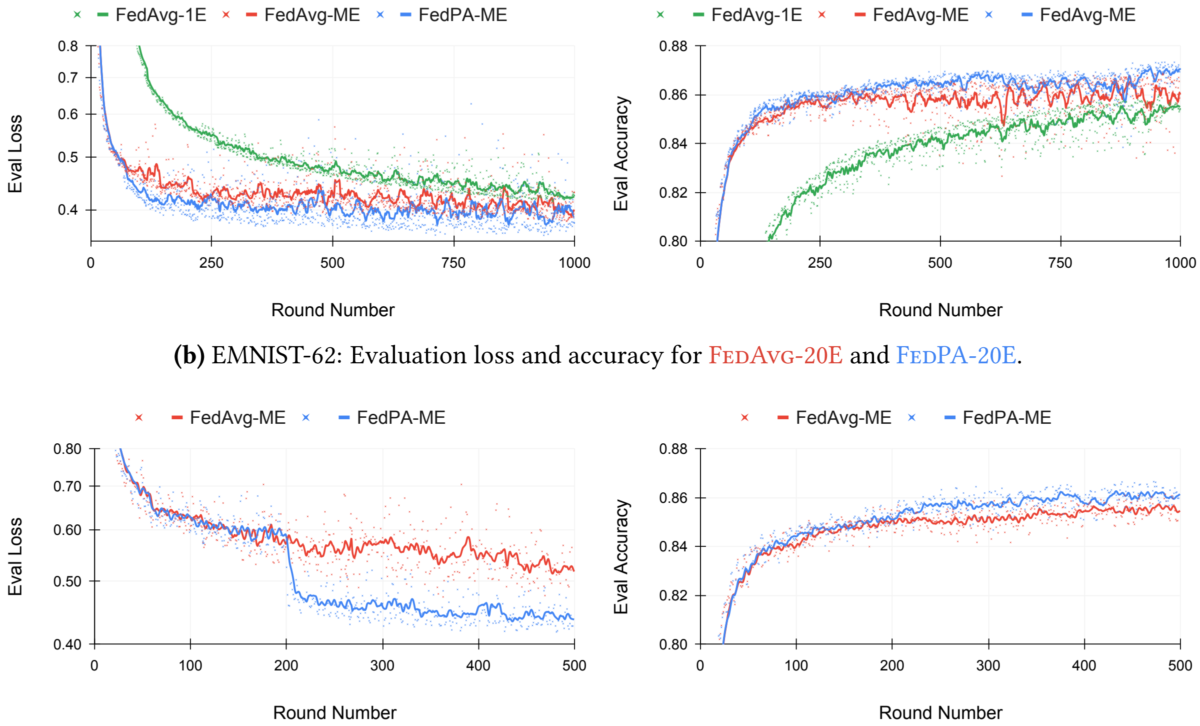
All hyperparameter grids are given in Table B.2. The best server and client learning rates were selected based on the FedAvg performance and used for FedPA. The best selected hyperparameters are given in Table B.3.

B.5 Additional Experimental Results

We provide additional experimental results. As mentioned in the main text, the results presented in Table 4.3 were selected to highlight the differences between the methods with respect to two metrics of interest: (i) the number of rounds until the desired performance, and (ii) the performance achievable within a fixed number of rounds. A much fuller picture is given by the learning curves of each method. Therefore, we plot evaluation losses, accuracies, and metrics of interest over the course of training. On the plots, individual values at each round are indicated with $\times$-markers and the 10-round running average with a line of the corresponding color.

EMNIST-62. Learning curves for FedAvg and FedPA on EMNIST-62 are given in Figure B.2. Figure B.2a shows the best FedAvg-1E, FedAvg-5E, and FedPA-5E models and Figure B.2b shows the best FedAvg-20E, and FedPA-20E. Apart from the fact that multi-epoch versions converge significantly faster than FedAvg-1E, note that the effect of bias reduction when switching from the burn-in to sampling in FedPA becomes much more pronounced in the 20-epoch version.

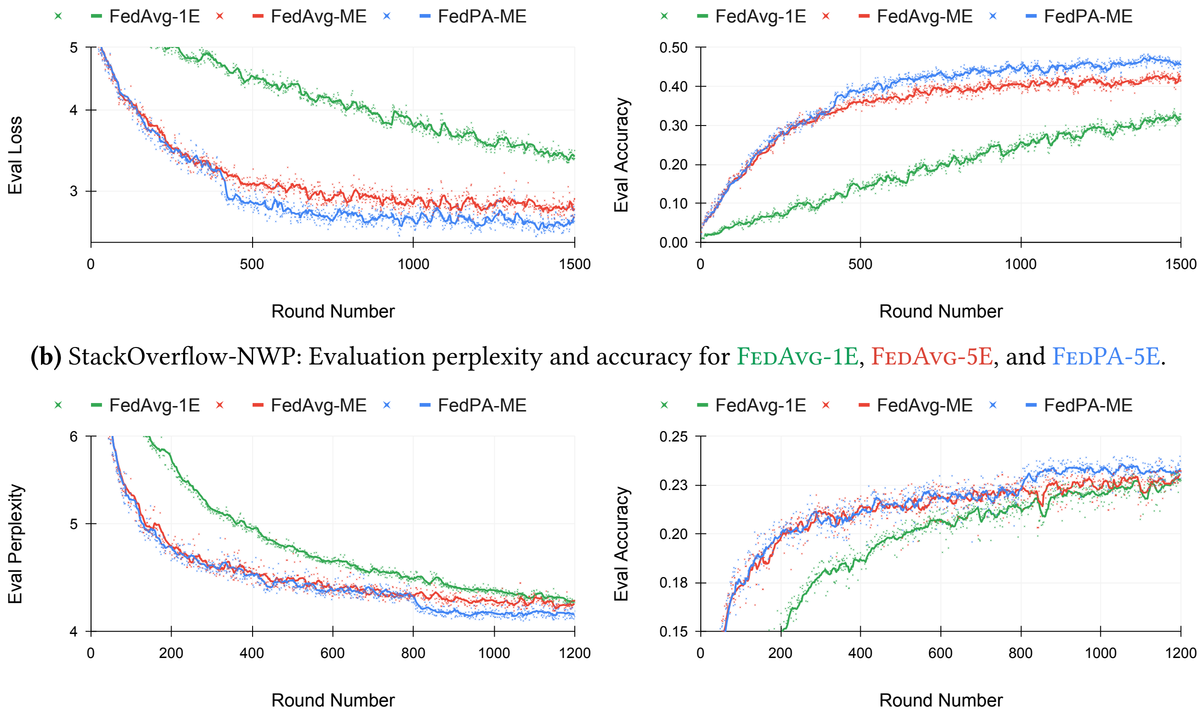
CIFAR-100 and StackOverflow. Learning curves for various models on CIFAR-100 and StackOverflow tasks are presented in Figures B.3 and B.4. The takeaways for CIFAR-100 and StackOverflow NWP are essentially the same as for EMNIST-62—much faster convergence with the increased number of local epochs and visually noticeable improvement in losses and accuracies due to sampling-based bias correction in client deltas after the burn-in phase is over. Interestingly, we see that on StackOverflow LR task FedAvg-1E clearly dominates multi-epoch methods in terms of the loss and recall at 5, losing in precision and macro-F1. Even more puzzling is the significant drop in the average precision of FedPA-ME after the switching to sampling, while at the same time a jump in recall and F1 metrics. This indicates that the global model moves to a different fixed point where it over-predicts positive labels (*i.e.*, less precise) but also less likely to miss rare labels (*i.e.*, higher recall on rare labels, and as a result a jump in macro-F1). The reason why this happens, however, is unclear.



(a) EMNIST-62: Evaluation loss and accuracy for FedAvg-1E, FedAvg-5E, and FedPA-5E.

(b) EMNIST-62: Evaluation loss and accuracy for FedAvg-20E and FedPA-20E.

Figure B.2: Evaluation metrics for FedAvg and FedPA computed at each training round on EMNIST-62.



(a) CIFAR-100: Evaluation loss and accuracy for FedAvg-1E, FedAvg-10E, and FedPA-10E.

(b) StackOverflow-NWP: Evaluation perplexity and accuracy for FedAvg-1E, FedAvg-5E, and FedPA-5E.

Figure B.3: Evaluation metrics for FedAvg and FedPA computed at each training round on (a) CIFAR-100 and (b) StackOverflow NWP tasks.

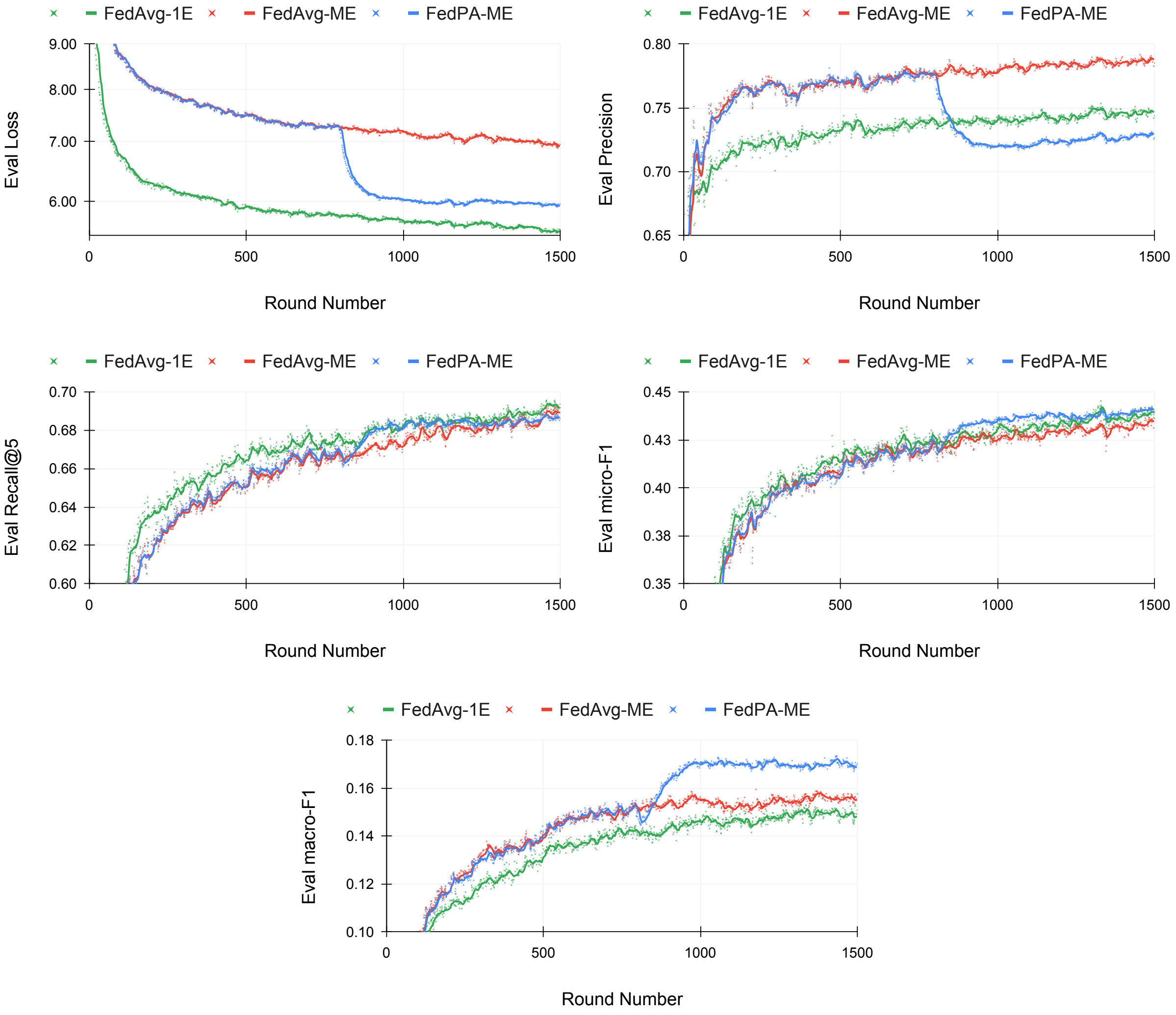


Figure B.4: Evaluation metrics for FedAvg and FedPA computed at each training round on StackOverflow LR. Evaluation loss, average precision and recall, and micro- and macro-averaged F1 for FedAvg-1E, FedAvg-5E, and FedPA-5E.

Appendix C

Learning under Nonstationarity: Additional Details

C.1 Policy Gradient Theorem for Meta-RL

In this section, we derive the policy gradient update for MAML as well as formulate and equivalent of the policy gradient theorem (Sutton et al., 2000) in the learning-to-learn setting.

Our derivation is not bound to a particular form of the adaptation update. In general, we are interested in meta-learning a *procedure*, f_θ, parametrized by θ, which, given access to a limited experience on a task, can produce a good policy for solving it. Note that f_θ is responsible for both collecting the initial experience and constructing the final policy for the given task. For example, in case of MAML (Finn et al., 2017), f_θ is represented by the initial policy, π_θ, and the adaptation update rule Equation 5.5 that produces π_ϕ with $\phi := \theta - \alpha \nabla_\theta L_T(\boldsymbol{\tau}_\theta^{1:K})$.

More formally, after querying K trajectories, $\boldsymbol{\tau}_\theta^{1:K}$, we want to produce π_ϕ that minimizes the expected loss w.r.t. the distribution over tasks:

$$\mathcal{L}(\theta) := \mathbb{E}_{T \sim \mathcal{D}(T)} \left[\mathbb{E}_{\boldsymbol{\tau}_\theta^{1:K} \sim p_T(\boldsymbol{\tau} \mid \theta)} \left[\mathbb{E}_{\boldsymbol{\tau}_\phi \sim p_T(\boldsymbol{\tau} \mid \phi)} \left[L_T(\boldsymbol{\tau}_\phi) \mid \boldsymbol{\tau}_\theta^{1:K} \right] \right] \right] \tag{C.1}$$

Note that the inner-most expectation is conditional on the experience, $\boldsymbol{\tau}_\theta^{1:K}$, which our meta-learning procedure, f_θ, collects to produce a task-specific policy, π_ϕ. Assuming that the loss $L_T(\boldsymbol{\tau}_\theta^{1:K})$ is linear in trajectories, and using linearity of expectations, we can drop the superscript $1 : K$ and denote the trajectory sampled under ϕ_θ for task T_i simply as $\boldsymbol{\tau}_{\theta,i}$. At training time, we are given a finite sample of tasks from the distribution $\mathcal{D}(T)$ and can search for $\hat{\theta}$ close to optimal by optimizing over the empirical distribution:

$$\hat{\theta} := \arg\min_\theta \hat{\mathcal{L}}(\theta), \text{ where } \hat{\mathcal{L}}(\theta) := \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\boldsymbol{\tau}_{\theta,i} \sim p_{T_i}(\boldsymbol{\tau} \mid \theta)} \left[\mathbb{E}_{\boldsymbol{\tau}_{\phi,i} \sim p_{T_i}(\boldsymbol{\tau} \mid \phi)} \left[L_{T_i}(\boldsymbol{\tau}_{\phi,i}) \mid \boldsymbol{\tau}_{\theta,i} \right] \right] \tag{C.2}$$

We re-write the objective function for task T_i in Equation C.2 more explicitly by expanding the expectations:

$$\begin{aligned}\mathcal{L}_{T_i}(\theta) &:= \mathbb{E}_{\boldsymbol{\tau}_{\theta,i}\sim p_{T_i}(\boldsymbol{\tau}|\theta)}\left[\mathbb{E}_{\boldsymbol{\tau}_{\phi,i}\sim p_{T_i}(\boldsymbol{\tau}|\phi)}\left[L_{T_i}(\boldsymbol{\tau}_{\phi,i}) \mid \boldsymbol{\tau}_{\theta,i}\right]\right] = \\ &\int L_{T_i}(\boldsymbol{\tau}_{\phi,i})\; p_{T_i}\left(\boldsymbol{\tau}_{\phi,i} \mid \phi\right)\; p_{T_i}\left(\phi \mid \theta, \boldsymbol{\tau}_{\theta,i}\right)\; p_{T_i}\left(\boldsymbol{\tau}_{\theta,i} \mid \theta\right)\; d\boldsymbol{\tau}_{\phi,i}\, d\phi\, d\boldsymbol{\tau}_{\theta,i}\end{aligned} \tag{C.3}$$

Trajectories, $\boldsymbol{\tau}_{\phi,i}$ and $\boldsymbol{\tau}_{\theta,i}$, and parameters ϕ of the policy π_ϕ can be thought as random variables that we marginalize out to construct the objective that depends on θ only. The adaptation update rule Equation 5.5 assumes the following $p_{T_i}\left(\phi \mid \theta, \boldsymbol{\tau}_{\theta,i}\right)$:

$$p_{T_i}\left(\phi \mid \theta, \boldsymbol{\tau}_{\theta,i}\right) := \delta\left(\theta - \alpha\nabla_\theta \frac{1}{K}\sum_{k=1}^{K} L_{T_i}(\boldsymbol{\tau}_{\theta,i}^k)\right) \tag{C.4}$$

Note that by specifying $p_{T_i}\left(\phi \mid \theta, \boldsymbol{\tau}_{\theta,i}\right)$ differently, we may arrive at different meta-learning algorithms. After plugging Equation C.4 into Equation C.3 and integrating out ϕ, we get the following expected loss for task T_i as a function of θ:

$$\begin{aligned}\mathcal{L}_{T_i}(\theta) &= \mathbb{E}_{\boldsymbol{\tau}_{\theta,i}\sim p_{T_i}(\boldsymbol{\tau}|\theta)}\left[\mathbb{E}_{\boldsymbol{\tau}_{\phi,i}\sim p_{T_i}(\boldsymbol{\tau}|\phi)}\left[L_{T_i}(\boldsymbol{\tau}_{\phi,i}) \mid \boldsymbol{\tau}_{\theta,i}\right]\right] = \\ &\int L_{T_i}(\boldsymbol{\tau}_{\phi,i})\; p_{T_i}\left(\boldsymbol{\tau}_{\phi,i} \mid \theta - \alpha\nabla_\theta \frac{1}{K}\sum_{k=1}^{K} L_{T_i}(\boldsymbol{\tau}_{\theta,i}^k)\right)\; p_{T_i}\left(\boldsymbol{\tau}_{\theta,i} \mid \theta\right)\; d\boldsymbol{\tau}_{\phi,i}\, d\boldsymbol{\tau}_{\theta,i}\end{aligned} \tag{C.5}$$

The gradient of Equation C.5 will take the following form:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{T_i}(\theta) = &\int \left[L_{T_i}(\boldsymbol{\tau}_{\phi,i})\; \nabla_\theta \log p_{T_i}\left(\boldsymbol{\tau}_{\phi,i} \mid \phi\right)\right] p_{T_i}\left(\boldsymbol{\tau}_{\phi,i} \mid \phi\right)\; p_{T_i}\left(\boldsymbol{\tau}_{\theta,i} \mid \theta\right)\; d\boldsymbol{\tau}\; d\boldsymbol{\tau}_{\theta,i} + \\ &\int \left[L_{T_i}(\boldsymbol{\tau})\; \nabla_\theta \log p_{T_i}\left(\boldsymbol{\tau}_{\theta,i} \mid \theta\right)\right] p_{T_i}\left(\boldsymbol{\tau} \mid \phi\right)\; p_{T_i}\left(\boldsymbol{\tau}_{\theta,i} \mid \theta\right)\; d\boldsymbol{\tau}\; d\boldsymbol{\tau}_{\theta,i}\end{aligned} \tag{C.6}$$

where $\phi = \phi(\theta, \boldsymbol{\tau}_{\theta,i}^{1:K})$ as given in Equation C.5. Note that the expression consists of two terms: the first term is the standard policy gradient w.r.t. the updated policy, π_ϕ, while the second one is the policy gradient w.r.t. the original policy, π_ϕ, that is used to collect $\boldsymbol{\tau}_{\theta,i}^{1:K}$. If we were to omit marginalization of $\boldsymbol{\tau}_{\theta,i}^{1:K}$ (as it was done in the original paper (Finn et al., 2017)), the terms would disappear. Finally, the gradient can be re-written in a more succinct form:

$$\nabla_\theta \mathcal{L}_{T_i}(\theta) = \mathbb{E}_{\substack{\boldsymbol{\tau}_{\theta,i}^{1:K}\sim p_{T_i}(\boldsymbol{\tau}|\theta)\\ \boldsymbol{\tau}\sim p_{T_i}(\boldsymbol{\tau}|\phi)}}\left[L_{T_i}(\boldsymbol{\tau})\left[\nabla_\theta \log \pi_\phi(\boldsymbol{\tau}) + \nabla_\theta \sum_{k=1}^{K} \log \pi_\theta(\boldsymbol{\tau}_k)\right]\right] \tag{C.7}$$

The update given in Equation C.7 is an unbiased estimate of the gradient as long as the loss L_{T_i} is simply the sum of discounted rewards (i.e., it extends the classical REINFORCE algorithm (Williams, 1992) to meta-learning). Similarly, we can define L_{T_i} that uses a value or advantage function and extend the policy gradient theorem Sutton et al., 2000 to make it suitable for meta-learning.

Theorem C.1: Meta Policy Gradient Theorem

For any MDP, gradient of the value function w.r.t. θ takes the following form:

$$\begin{aligned}
&\nabla_\theta V_T^\theta(\mathbf{x}_0) = \\
&\quad \mathbb{E}_{\boldsymbol{\tau}_{1:K} \sim p_T(\boldsymbol{\tau} \mid \theta)} \left[\sum_{\mathbf{x}} d_T^\phi(\mathbf{x}) \sum_a \frac{\partial \pi_\phi(a \mid \mathbf{x})}{\partial \theta} Q_T^\phi(a, \mathbf{x}) \right] + \\
&\quad \mathbb{E}_{\boldsymbol{\tau}_{1:K} \sim p_T(\boldsymbol{\tau} \mid \theta)} \left[\left(\frac{\partial}{\partial \theta} \sum_{k=1}^K \log \pi_\theta(\boldsymbol{\tau}_k) \right) \sum_a \pi_\phi(a \mid \mathbf{x}_0) Q_T^\phi(a, \mathbf{x}_0) \right],
\end{aligned} \tag{C.8}$$

where $d_T^\phi(\mathbf{x})$ is the stationary distribution under policy π_ϕ.

Proof. We define task-specific value functions under the generated policy, π_ϕ, as follows:

$$\begin{aligned}
V_T^\phi(\mathbf{x}_0) &= \mathbb{E}_{\boldsymbol{\tau} \sim p_T(\boldsymbol{\tau} \mid \phi)} \left[\sum_{t=k}^H \gamma^t R_T(\mathbf{x}_t) \mid \mathbf{x}_0 \right], \\
Q_T^\phi(\mathbf{x}_0, a_0) &= \mathbb{E}_{\boldsymbol{\tau} \sim p_T(\boldsymbol{\tau} \mid \phi)} \left[\sum_{t=k}^H \gamma^t R_T(\mathbf{x}_t) \mid \mathbf{x}_0, a_0 \right],
\end{aligned} \tag{C.9}$$

where the expectations are taken w.r.t. the dynamics of the environment of the given task, T, and the policy, π_ϕ. Next, we need to marginalize out $\boldsymbol{\tau}_{1:K}$:

$$V_T^\theta(\mathbf{x}_0) = \mathbb{E}_{\boldsymbol{\tau}_{1:K} \sim p_T(\boldsymbol{\tau} \mid \theta)} \left[\mathbb{E}_{\boldsymbol{\tau} \sim p_T(\boldsymbol{\tau} \mid \phi)} \left[\sum_{t=k}^H \gamma^t R_T(\mathbf{x}_t) \mid \mathbf{x}_0 \right] \right], \tag{C.10}$$

and after the gradient w.r.t. θ, we arrive at:

$$\begin{aligned}
&\nabla_\theta V_T^\theta(\mathbf{x}_0) = \\
&\quad \mathbb{E}_{\boldsymbol{\tau}_{1:K} \sim p_T(\boldsymbol{\tau} \mid \theta)} \left[\sum_a \frac{\partial \pi_\phi(a \mid \mathbf{x}_0)}{\partial \theta} Q_T^\phi(a, \mathbf{x}_0) + \pi_\phi(a \mid \mathbf{x}_0) \frac{\partial Q_T^\phi(a, \mathbf{x}_0)}{\partial \theta} \right] + \\
&\quad \mathbb{E}_{\boldsymbol{\tau}_{1:K} \sim p_T(\boldsymbol{\tau} \mid \theta)} \left[\left(\sum_{k=1}^K \frac{\partial}{\partial \theta} \log \pi_\theta(\boldsymbol{\tau}_k) \right) \sum_a \pi_\phi(a \mid \mathbf{x}_0) Q_T^\phi(a, \mathbf{x}_0) \right],
\end{aligned} \tag{C.11}$$

where the first term is similar to the expression used in the original policy gradient theorem (Sutton et al., 2000) while the second one comes from differentiating trajectories $\boldsymbol{\tau}_{1:K}$ that depend on θ. Following Sutton et al. (2000), we unroll the derivative of the Q-function in the first term and arrive at the following final expression for the policy gradient:

$$\begin{aligned}
&\nabla_\theta V_T^\theta(\mathbf{x}_0) = \\
&\quad \mathbb{E}_{\boldsymbol{\tau}_{1:K} \sim p_T(\boldsymbol{\tau}|\theta)} \left[\sum_{\mathbf{x}} d_T^\phi(\mathbf{x}) \sum_a \frac{\partial \pi_\phi(a \mid \mathbf{x})}{\partial \theta} Q_T^\phi(a, \mathbf{x}) \right] + \\
&\quad \mathbb{E}_{\boldsymbol{\tau}_{1:K} \sim p_T(\boldsymbol{\tau}|\theta)} \left[\left(\frac{\partial}{\partial \theta} \sum_{k=1}^{K} \log \pi_\theta(\boldsymbol{\tau}_k) \right) \sum_a \pi_\phi(a \mid \mathbf{x}_0) Q_T^\phi(a, \mathbf{x}_0) \right]
\end{aligned} \tag{C.12}$$

□

Remark C.1. *The same theorem is applicable to the continuous adaptation setting with the only changes in the distributions used to compute expectations in Equations C.8 and C.9. In particular, the outer expectation in Equation C.8 should be taken w.r.t. $p_{T_i}(\boldsymbol{\tau} \mid \theta)$ while the inner expectation w.r.t. $p_{T_{i+1}}(\boldsymbol{\tau} \mid \phi)$.*

C.1.1 Multiple Adaptation Gradient Steps

All our derivations so far assumed single step gradient-based adaptation update. Experimentally, we found that the multi-step version of the update often leads to a more stable training and better test time performance. In particular, we construct ϕ via intermediate M gradient steps:

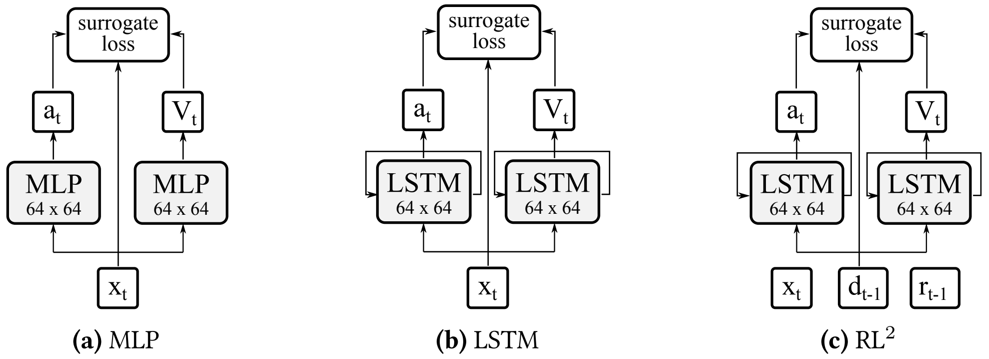
$$\begin{aligned}
\phi^0 &:= \theta, \quad \boldsymbol{\tau}_\theta^{1:K} \sim P_T(\boldsymbol{\tau} \mid \theta), \\
\phi^m &:= \phi^{m-1} - \alpha_m \nabla_{\phi^{m-1}} L_T\left(\boldsymbol{\tau}_{\phi^{m-1}}^{1:K}\right), \quad m = 1, \ldots, M-1, \\
\phi &:= \phi^{M-1} - \alpha_M \nabla_{\phi^{M-1}} L_T\left(\boldsymbol{\tau}_{\phi^{M-1}}^{1:K}\right)
\end{aligned} \tag{C.13}$$

where ϕ^m are intermediate policy parameters. Note that each intermediate step, m, requires interacting with the environment and sampling intermediate trajectories, $\boldsymbol{\tau}_{\phi^m}^{1:K}$. To compute the policy gradient, we need to marginalize out all the intermediate random variables, π_{ϕ^m} and $\boldsymbol{\tau}_{\phi^m}^{1:K}$, $m = 1, \ldots, M$. The objective function Equation C.3 takes the following form:

$$\begin{aligned}
&\mathcal{L}_T(\theta) = \\
&\quad \int L_T(\boldsymbol{\tau})\, p_T(\boldsymbol{\tau} \mid \phi)\; p_T\left(\phi \mid \phi^{M-1}, \boldsymbol{\tau}_{\phi^{M-1}}^{1:K}\right) d\boldsymbol{\tau} d\phi \times \\
&\quad\quad \prod_{m=1}^{M-2} p_T\left(\boldsymbol{\tau}_{\phi^{m+1}}^{1:K} \mid \phi^{m+1}\right) p_T\left(\phi^{m+1} \mid \phi^m, \boldsymbol{\tau}_{\phi^m}^{1:K}\right) d\boldsymbol{\tau}_{\phi^{m+1}}^{1:K} d\phi^{m+1} \times \\
&\quad\quad p_T\left(\boldsymbol{\tau}^{1:K} \mid \theta\right) d\boldsymbol{\tau}^{1:K}
\end{aligned} \tag{C.14}$$

Since $p_T\left(\phi^{m+1} \mid \phi^m, \boldsymbol{\tau}_{\phi^m}^{1:K}\right)$ at each intermediate steps are delta functions, the final expression for the multi-step MAML objective has the same form as Equation C.5, with integration taken w.r.t. all intermediate trajectories. Similarly, an unbiased estimate of the gradient of the objective gets M additional terms:

$$\nabla_\theta \mathcal{L}_T = \mathbb{E}_{\{\boldsymbol{\tau}_{\phi^m}^{1:K}\}_{m=0}^{M-1}, \boldsymbol{\tau}} \left[L_T(\boldsymbol{\tau}) \left[\nabla_\theta \log \pi_\phi(\boldsymbol{\tau}) + \sum_{m=0}^{M-1} \nabla_\theta \sum_{k=1}^{K} \log \pi_{\phi^m}(\boldsymbol{\tau}_{\phi^m}^k) \right] \right], \tag{C.15}$$



(a) MLP **(b)** LSTM **(c)** RL^2

Figure C.1: Policy and value function architectures.

where the expectation is taken w.r.t. trajectories (including all intermediate ones). Again, note that at training time we do not constrain the number of interactions with each particular environment and do rollout using each intermediate policy to compute updates. At testing time, we interact with the environment only once and rely on the importance weight correction as described in Section 5.4.2.

C.2 Additional details on the architectures

The neural architectures used for our policies and value functions are illustrated in Figure C.1. Our MLP architectures were memory-less and reactive. The LSTM architectures had used a fully connected embedding layer (with 64 hidden units) followed by a recurrent layer (also with 64 units). The state in LSTM-based architectures was kept throughout each episode and reset to zeros at the beginning of each new episode. The RL^2 architecture additionally took reward and done signals from the previous time step and kept the state throughout the whole interactions with a given environment (or opponent). The recurrent architectures were unrolled for $T = 10$ time steps and optimized with PPO via backprop through time.

C.3 Additional details on meta-learning and optimization

C.3.1 Meta-updates for continuous adaptation

Our meta-learned adaptation methods were used with MLP and LSTM policies (Figure C.1). The meta-updates were based on 3 gradient steps with adaptive step sizes α were initialized with 0.001. There are a few additional details to note:

1. θ and ϕ parameters were a concatenation of the policy and the value function parameters.
2. At the initial stages of optimization, meta-gradient steps often tended to "explode", hence we clipped them by values norms to be between -0.1 and 0.1.

3. We used different surrogate loss functions for the meta-updates and for the outer optimization. For meta-updates, we used the vanilla policy gradients computed on the negative discounted rewards, while for the outer optimization loop we used the PPO objective.

C.3.2 On PPO and its distributed implementation

As mentioned in the main text and similar to (Bansal et al., 2018), large batch sizes were used to ensure enough exploration throughout policy optimization and were critical for learning in the competitive setting of RoboSumo. In our experiments, the epoch size of the PPO was set 32,000 episodes and the batch size was set to 8,000. The PPO clipping hyperparameter was set to $\epsilon = 0.2$ and the KL penalty was set to 0. In all our experiments, the learning rate (for meta-learning, the learning rate for θ and α) was set to 0.0003. The generalized advantage function estimator (GAE) (Schulman et al., 2015b) was optimized jointly with the policy (we used $\gamma = 0.995$ and $\lambda = 0.95$).

To train our agents in reasonable time, we used a distributed implementation of the PPO algorithm. To do so, we versioned the agent's parameters (i.e., kept parameters after each update and assigned it a version number) and used a versioned queue for rollouts. Multiple worker machines were generating rollouts in parallel for the most recent available version of the agent parameters and were pushing them into the versioned rollout queue. The optimizer machine collected rollouts from the queue and made a PPO optimization steps (see (Schulman et al., 2017) for details) as soon as enough rollouts were available.

We trained agents on multiple environments simultaneously. In nonstationary locomotion, each environment corresponded to a different pair of legs of the creature becoming dysfunctional. In RoboSumo, each environment corresponded to a different opponent in the training pool. Simultaneous training was achieved via assigning these environments to rollout workers uniformly at random, so that the rollouts in each mini-batch were guaranteed to come from all training environments.

C.4 Additional details on the environments

C.4.1 Observation and action spaces

Both observation and action spaces in RoboSumo continuous. The observations of each agent consist of the position of its own body (7 dimensions that include 3 absolute coordinates in the global cartesian frame and 4 quaternions), position of the opponent's body (7 dimensions), its own joint angles and velocities (2 angles and 2 velocities per leg), and forces exerted on each part of its own body (6 dimensions for torso and 18 for each leg) and forces exerted on the opponent's torso (6 dimensions). All forces were squared and clipped at 100. Additionally, we normalized observations using a running mean and clipped their values between -5 and 5. The action spaces had 2 dimensions per joint. Table C.1 summarizes the observation and action spaces for each agent type.

Table C.1: Dimensionality of the observation and action spaces of the agents in RoboSumo.

Agent	Observation space					Action space
	Self			Opponent		
	Coordinates	Velocities	Forces	Coordinates	Forces	
Ant	15	14	78	7	6	8
Bug	19	18	114	7	6	12
Spider	23	22	150	7	6	16

Note that the agents observe neither any of the opponents velocities, nor positions of the opponent's limbs. This allows us to keep the observation spaces consistent regardless of the type of the opponent. However, even though the agents are blind to the opponent's limbs, they can sense them via the forces applied to the agents' bodies when in contact with the opponent.

C.4.2 Shaped rewards

In RoboSumo, the winner gets 2000 reward, the loser is penalized for -2000, and in case of draw both agents get -1000. In addition to the sparse win/lose rewards, we used the following dense rewards to encourage fast learning at the early training stages:

- **Quickly push the opponent outside.** The agent got penalty at each time step proportional to $e^{-d_{\text{opp}}}$ where d_{opp} was the distance of the opponent from the center of the ring.
- **Moving towards the opponent.** Reward at each time step proportional to magnitude of the velocity component towards the opponent.
- **Hit the opponent.** Reward proportional to the square of the total forces exerted on the opponent's torso.
- **Control penalty.** The l_2 penalty on the actions to prevent jittery/unnatural movements.

C.4.3 RoboSumo calibration

To calibrate the RoboSumo environment we used the following procedure. First, we trained each agent via pure self-play with LSTM policy using PPO for the same number of iterations, tested them one against the other (without adaptation), and recorded the win rates (Table C.2). To ensure the balance, we kept increasing the mass of the weaker agents and repeated the calibration procedure until the win rates equilibrated.

Table C.2: Win rates for the first agent in RoboSumo *without adaptation* before/after calibration.

Masses (Ant, Bug, Spider)	Ant vs. Bug	Ant vs. Spider	Bug vs. Spider
Initial (10, 10, 10)	$25.2 \pm 3.9\%$	$83.6 \pm 3.1\%$	$90.2 \pm 2.7\%$
Calibrated (13, 10, 39)	$50.6 \pm 5.6\%$	$51.6 \pm 3.4\%$	$51.7 \pm 2.8\%$

C.5 Additional details on experiments

C.5.1 Average win rates

Table C.3 gives average win rates for the last 25 rounds of iterated adaptation games played by different agents with different adaptation methods (win rates for each episode are visualized in Figure 5.6).

Table C.3: Average win-rates (95% CI) in the last 25 rounds of the 100-round iterated adaptation games between different agents and different opponents. The base policy and value function were LSTMs with 64 hidden units.

Agent	**Opponent**	**Adaptation Strategy**		
		RL^2	LSTM + PPO-tracking	LSTM + meta-updates
Ant	Ant	24.9 (5.4)%	30.0 (6.7)%	**44.0** (7.7)%
	Bug	21.0 (6.3)%	15.6 (7.1)%	**34.6** (8.1)%
	Spider	24.8 (10.5)%	27.6 (8.4)%	35.1 (7.7)%
Bug	Ant	33.5 (6.9)%	26.6 (7.4)%	**39.5** (7.1)%
	Bug	28.6 (7.4)%	21.2 (4.2)%	**43.7** (8.0)%
	Spider	45.8 (8.1)%	42.6 (12.9)%	52.0 (13.9)%
Spider	Ant	40.3 (9.7)%	48.0 (9.8)%	45.3 (10.9)%
	Bug	38.4 (7.2)%	43.9 (7.1)%	48.4 (9.2)%
	Spider	33.9 (7.2)%	42.2 (3.9)%	46.7 (3.8)%

C.5.2 TrueSkill rank of the top agents

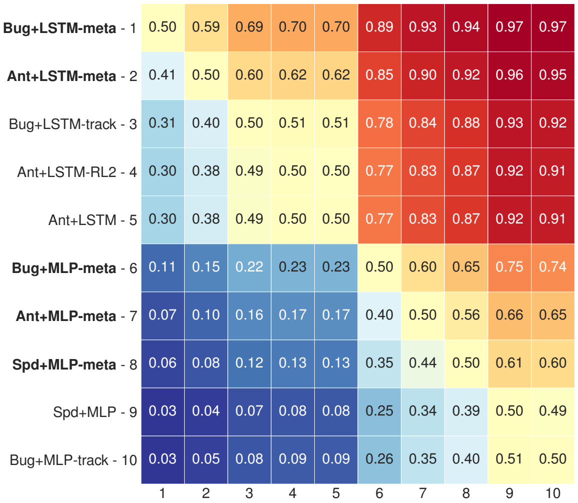
Since TrueSkill represents the belief about the skill of an agent as a normal distribution (i.e., with two parameters, μ and σ), we can use it to infer *a priori* probability of an agent, a, winning against its opponent, o, as follows (Herbrich et al., 2007):

$$P(a \text{ wins } o) = \Phi\left(\frac{\mu_a - \mu_o}{\sqrt{2\beta^2 + \sigma_a^2 + \sigma_o^2}}\right), \text{ where } \Phi(x) := \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{C.16}$$

The ranking of the top-5 agents with MLP and LSTM policies according to their TrueSkill is given in Tab. 1 and the *a priori* win rates in **??**. Note that within the LSTM and MLP categories, the best meta-learners are 10 to 25% more likely to win the best agents that use other adaptation strategies.

Rank	Agent	TrueSkill rank*
1	Bug + LSTM-meta	31.7
2	Ant + LSTM-meta	30.8
3	Bug + LSTM-track	29.1
4	Ant + RL^2	28.6
5	Ant + LSTM	28.4
6	Bug + MLP-meta	23.4
7	Ant + MLP-meta	21.6
8	Spider + MLP-meta	20.5
9	Spider + MLP	19.0
10	Bug + MLP-track	18.9

* The rank is a conservative estimate of the skill, $r = \mu - 3\sigma$, to ensure that the actual skill of the agent is higher with 99% confidence.



	1	2	3	4	5	6	7	8	9	10
Bug+LSTM-meta - 1	0.50	0.59	0.69	0.70	0.70	0.89	0.93	0.94	0.97	0.97
Ant+LSTM-meta - 2	0.41	0.50	0.60	0.62	0.62	0.85	0.90	0.92	0.96	0.95
Bug+LSTM-track - 3	0.31	0.40	0.50	0.51	0.51	0.78	0.84	0.88	0.93	0.92
Ant+LSTM-RL2 - 4	0.30	0.38	0.49	0.50	0.50	0.77	0.83	0.87	0.92	0.91
Ant+LSTM - 5	0.30	0.38	0.49	0.50	0.50	0.77	0.83	0.87	0.92	0.91
Bug+MLP-meta - 6	0.11	0.15	0.22	0.23	0.23	0.50	0.60	0.65	0.75	0.74
Ant+MLP-meta - 7	0.07	0.10	0.16	0.17	0.17	0.40	0.50	0.56	0.66	0.65
Spd+MLP-meta - 8	0.06	0.08	0.12	0.13	0.13	0.35	0.44	0.50	0.61	0.60
Spd+MLP - 9	0.03	0.04	0.07	0.08	0.08	0.25	0.34	0.39	0.50	0.49
Bug+MLP-track - 10	0.03	0.05	0.08	0.09	0.09	0.26	0.35	0.40	0.51	0.50

Table C.4 & Figure C.2: Top-5 agents with MLP and LSTM policies from the population ranked by TrueSkill. The heatmap shows *a priori* win-rates in iterated games based on TrueSkill for the top agents against each other.

Appendix D

Zero-shot Consistency: Additional Details

D.1 Consistency of the Full Likelihood

(Throughout this appendix chapter, we slightly alter our notation: instead of writing $p\left(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}\right)$, we write $p_{\boldsymbol{\theta}}\left(\mathbf{y} \mid \mathbf{x}\right)$, which means the same thing and slightly simplifies notation.)

Recall from Section 6.4.1 that, a set of conditional models, $\{p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)\}$, the full likelihood over equivalent translations, $(\mathbf{x}_1, \dots, \mathbf{x}_k)$, can be written as follows:

$$p_{\boldsymbol{\theta}}\left(\mathbf{x}_1, \dots, \mathbf{x}_k\right) := \frac{1}{Z} \prod_{i,j \in \mathcal{E}} p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right) \tag{D.1}$$

where $Z := \sum_{\mathbf{x}_1, \dots, \mathbf{x}_k} \prod_{i,j \in \mathcal{E}} p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)$ is the normalizing constant and $\mathcal{E}$ denotes *all* edges in the graph (Figure D.1). Given only bilingual parallel corpora, $\mathcal{C}_{ij}$ for $i, j \in \mathcal{E}_s$, we can observe only certain pairs of variables. Therefore, the log-likelihood of the data can be written as:

$$\mathcal{L}(\boldsymbol{\theta}) := \sum_{i,j \in \mathcal{E}_s} \sum_{\mathbf{x}_i, \mathbf{x}_j \in \mathcal{C}_{ij}} \log \sum_{\mathbf{z}} p_{\boldsymbol{\theta}}\left(\mathbf{x}_1, \dots, \mathbf{x}_k\right) \tag{D.2}$$

The outer sum iterates over available corpora. The middle sum iterates over parallel sentences in a corpus. The most inner sum marginalizes out unobservable sequences, denoted $\mathbf{z} := \{\mathbf{x}_l\}_{l \neq i,j}$, which are sentences equivalent under this model to $\mathbf{x}_i$ and $\mathbf{x}_j$ in languages other than L_i and L_j. Note that due to the inner-most summation, computing the log likelihood is intractable. We claim the following.

Claim D.1

Maximizing the full log-likelihood yields zero-shot consistent models (Definition 6.1).

Proof. To better understand why this is the case, let us consider example in Figure D.1 and

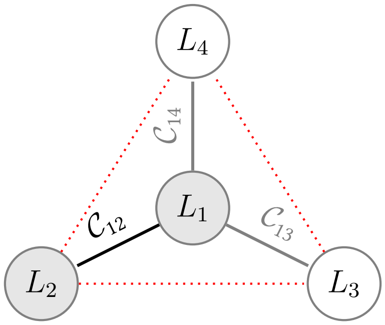


Figure D.1: Probabilistic graphical model for a multilingual system with four languages (L_1, L_2, L_3, L_4). Variables can only be observed only in pairs (shaded in the graph).

compute the log-likelihood of $(\mathbf{x}_1, \mathbf{x}_2)$:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}_1, \mathbf{x}_2) = \log \sum_{\mathbf{x}_3, \mathbf{x}_4} p_{\boldsymbol{\theta}}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \\
= \text{const} + \log p_{\boldsymbol{\theta}}(\mathbf{x}_1 \mid \mathbf{x}_2) + \log p_{\boldsymbol{\theta}}(\mathbf{x}_2 \mid \mathbf{x}_1) + \\
\log \sum_{\mathbf{x}_3, \mathbf{x}_4} p_{\boldsymbol{\theta}}(\mathbf{x}_3 \mid \mathbf{x}_1)\, p_{\boldsymbol{\theta}}(\mathbf{x}_3 \mid \mathbf{x}_2)\, p_{\boldsymbol{\theta}}(\mathbf{x}_4 \mid \mathbf{x}_1)\, p_{\boldsymbol{\theta}}(\mathbf{x}_4 \mid \mathbf{x}_2) \times \\
p_{\boldsymbol{\theta}}(\mathbf{x}_1 \mid \mathbf{x}_3)\, p_{\boldsymbol{\theta}}(\mathbf{x}_2 \mid \mathbf{x}_3)\, p_{\boldsymbol{\theta}}(\mathbf{x}_1 \mid \mathbf{x}_4)\, p_{\boldsymbol{\theta}}(\mathbf{x}_2 \mid \mathbf{x}_4) \times \\
p_{\boldsymbol{\theta}}(\mathbf{x}_3 \mid \mathbf{x}_4)\, p_{\boldsymbol{\theta}}(\mathbf{x}_4 \mid \mathbf{x}_3)
\end{aligned}$$

Note that the terms that encourage agreement on the translation into L_3 are colored in green (similarly, terms that encourage agreement on the translation into L_4 are colored in blue). Since all other terms are probabilities and bounded by 1, we have:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}_1, \mathbf{x}_2) + \log Z \le \log p_{\boldsymbol{\theta}}(\mathbf{x}_1 \mid \mathbf{x}_2) + \log p_{\boldsymbol{\theta}}(\mathbf{x}_2 \mid \mathbf{x}_1) + \\
\log \sum_{\mathbf{x}_3, \mathbf{x}_4} p_{\boldsymbol{\theta}}(\mathbf{x}_3 \mid \mathbf{x}_1)\, p_{\boldsymbol{\theta}}(\mathbf{x}_3 \mid \mathbf{x}_2)\, p_{\boldsymbol{\theta}}(\mathbf{x}_4 \mid \mathbf{x}_1)\, p_{\boldsymbol{\theta}}(\mathbf{x}_4 \mid \mathbf{x}_2) \\
\equiv \mathcal{L}^{\text{agree}}(\boldsymbol{\theta})
\end{aligned}$$

In other words, the full log likelihood lower-bounds the agreement objective (up to a constant $\log Z$). Since optimizing for agreement leads to consistency (Theorem 6.1), and maximizing the full likelihood would necessarily improve the agreement, the claim follows. □

Remark D.1. *Note that the other terms in the full likelihood also have a non-trivial purpose: (a) the terms $p_{\boldsymbol{\theta}}(\mathbf{x}_1 \mid \mathbf{x}_3)$, $p_{\boldsymbol{\theta}}(\mathbf{x}_1 \mid \mathbf{x}_4)$, $p_{\boldsymbol{\theta}}(\mathbf{x}_2 \mid \mathbf{x}_3)$, $p_{\boldsymbol{\theta}}(\mathbf{x}_2 \mid \mathbf{x}_4)$, encourage the model to correctly reconstruct $\mathbf{x}_1$ and $\mathbf{x}_2$ when back-translating from unobserved languages, L_3 and L_4, and (b) terms $p_{\boldsymbol{\theta}}(\mathbf{x}_3 \mid \mathbf{x}_4)$, $p_{\boldsymbol{\theta}}(\mathbf{x}_4 \mid \mathbf{x}_3)$ enforce consistency between the latent representations. In other words, full likelihood accounts for a combination of agreement, back-translation, and latent consistency.*

D.2 Proof of Theorem 6.1

The statement of Theorem 6.1 mentions an assumption on the true distribution of the equivalent translations. The assumption is as follows.

Assumption D.1. *Let $p\left(\mathbf{x}_i \mid \mathbf{x}_j, \mathbf{x}_k\right)$ be the ground truth conditional distribution that specifies the probability of $\mathbf{x}_i$ to be a translation of $\mathbf{x}_j$ and $\mathbf{x}_k$ into language L_i, given that $\left(\mathbf{x}_j, \mathbf{x}_k\right)$ are correct translations of each other in languages L_j and L_k, respectively. We assume:*

$$0 \leq \delta \leq \mathbb{E}_{\mathbf{x}_k \mid \mathbf{x}_i, \mathbf{x}_j}\left[p\left(\mathbf{x}_i \mid \mathbf{x}_j, \mathbf{x}_k\right)\right] \leq \xi \leq 1$$

This assumption means that, even though there might be multiple equivalent translations, there must be not too many of them (implied by the δ lower bound) and none of them must be much more preferable than the rest (implied by the ξ upper bound). Given this assumption, we can prove the following simple lemma.

Lemma D.1

Let $L_i \to L_j$ be one of the supervised directions, $\mathbb{E}_{\mathbf{x}_i, \mathbf{x}_j}\left[-\log p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)\right] \leq \varepsilon$. Then the following holds:

$$\mathbb{E}_{\mathbf{x}_i \mid \mathbf{x}_j, \mathbf{x}_k}\left[\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)}{p\left(\mathbf{x}_j \mid \mathbf{x}_i, \mathbf{x}_k\right)}\right] \geq \log \frac{1}{\xi} - \varepsilon\delta$$

Proof. First, using Jensen's inequality, we have:

$$\log \mathbb{E}_{\mathbf{x}_i \mid \mathbf{x}_j, \mathbf{x}_k}\left[\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)}{p\left(\mathbf{x}_j \mid \mathbf{x}_i, \mathbf{x}_k\right)}\right] \geq \mathbb{E}_{\mathbf{x}_i \mid \mathbf{x}_j, \mathbf{x}_k}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right) - \log p\left(\mathbf{x}_j \mid \mathbf{x}_i, \mathbf{x}_k\right)\right]$$

The bound on the supervised direction implies that

$$\mathbb{E}_{\mathbf{x}_i \mid \mathbf{x}_j, \mathbf{x}_k}\left[\log p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)\right] \geq -\varepsilon\delta$$

To bound the second term, we use Assumption D.1:

$$\mathbb{E}_{\mathbf{x}_i \mid \mathbf{x}_j, \mathbf{x}_k}\left[-\log p\left(\mathbf{x}_j \mid \mathbf{x}_i, \mathbf{x}_k\right)\right] \geq \log \frac{1}{\xi}$$

Putting these together yields the bound. □

Now, using Lemma D.1, we can prove Theorem 6.1.

Proof. By assumption, the agreement-based loss is bounded by ε. Therefore, expected cross-entropy on all supervised terms, $L_1 \leftrightarrow L_2$, is bounded by ε. Moreover, the agreement term (which is part of the objective) is also bounded:

$$-\mathbb{E}_{\mathbf{x}_i, \mathbf{x}_j}\left[\sum_{\mathbf{x}_k} p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_j\right) \log p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_i\right)\right] \leq \varepsilon$$

Expanding this expectation, we have:

$$\begin{aligned}
&\sum_{\mathbf{x}_i,\mathbf{x}_j} p\left(\mathbf{x}_i,\mathbf{x}_j\right) \sum_{\mathbf{x}_k} p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_j\right) \log p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_i\right)] \\
&\quad= \sum_{\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k} p\left(\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k\right) \frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_j\right)}{p\left(\mathbf{x}_k \mid \mathbf{x}_i,\mathbf{x}_j\right)} \log p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_i\right) \\
&\quad= \sum_{\mathbf{x}_i,\mathbf{x}_k} \mathbb{E}_{\mathbf{x}_j \mid \mathbf{x}_i,\mathbf{x}_k}\left[\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_j\right)}{p\left(\mathbf{x}_k \mid \mathbf{x}_i,\mathbf{x}_j\right)}\right] p\left(\mathbf{x}_i,\mathbf{x}_k\right) \log p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_i\right)
\end{aligned}$$

Combining that with Lemma D.1, we have:

$$\mathbb{E}_{\mathbf{x}_i,\mathbf{x}_k}\left[-\log p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_i\right)\right] \leq \frac{\varepsilon}{\log\frac{1}{\xi} - \delta\varepsilon} \equiv \kappa(\varepsilon)$$

Since by Assumption D.1, δ and ξ are some constants, $\kappa(\varepsilon) \to 0$ as $\varepsilon \to 0$. □

D.3 Consistency of Distillation and Pivoting

As we mentioned in Chapter 6, distillation (Chen et al., 2017) and pivoting yield zero-shot consistent models. Let us understand why this is the case.

In our notation, given $L_1 \to L_2$ and $L_2 \to L_3$ as supervised directions, distillation optimizes a KL-divergence between $p_{\boldsymbol{\theta}}\left(\mathrm{x}_3 \mid \mathrm{x}_2\right)$ and $p_{\boldsymbol{\theta}}\left(\mathrm{x}_3 \mid \mathrm{x}_1\right)$, where the latter is a zero-shot model and the former is supervised. Noting that KL-divergence lower-bounds cross-entropy, it is a loser bound on the agreeement loss. Hence, by ensuring that KL is low, we also ensure that the models agree, which implies consistency (a more formal proof would exactly follow the same steps as the proof of Theorem 6.1).

To prove consistency of pivoting, we need an additional assumption on the quality of the source-pivot model.

Assumption D.2. *Let $p_{\boldsymbol{\theta}}\left(\mathrm{x}_j \mid \mathrm{x}_i\right)$ be the source-pivot model. We assume the following bound holds for each pair of equivalent translations, $\left(\mathbf{x}_j, \mathbf{x}_k\right)$:*

$$\mathbb{E}_{\mathbf{x}_i \mid \mathbf{x}_j,\mathbf{x}_k}\left[\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)}{p\left(\mathbf{x}_j \mid \mathbf{x}_i,\mathbf{x}_k\right)}\right] \leq C$$

where $C > 0$ is some constant.

Theorem D.1: Consistency of Pivoting

Given the conditions of Theorem 6.1 and Assumption D.2, pivoting is zero-shot consistent.

Proof. We can bound the expected error on pivoting as follows (using Jensen's inequality and the conditions from our assumptions):

$$
\begin{aligned}
\mathbb{E}_{\mathbf{x}_i,\mathbf{x}_k}\left[-\log\sum_{\mathbf{x}_j} p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right) p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_j\right)\right] &\le \mathbb{E}_{\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k}\left[-p_{\boldsymbol{\theta}}\left(\mathbf{x}_j \mid \mathbf{x}_i\right)\log p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_j\right)\right] \\
&\le \sum_{\mathbf{x}_i,\mathbf{x}_k} \mathbb{E}_{\mathbf{x}_j \mid \mathbf{x}_i,\mathbf{x}_k}\left[\frac{p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_j\right)}{p\left(\mathbf{x}_k \mid \mathbf{x}_i, \mathbf{x}_j\right)}\right] \times \\
&\qquad p\left(\mathbf{x}_i, \mathbf{x}_k\right)\log p_{\boldsymbol{\theta}}\left(\mathbf{x}_k \mid \mathbf{x}_i\right) \\
&\le C\varepsilon
\end{aligned}
$$

□

D.4 Experimental details

D.4.1 Details on the models and training

Architecture. All our NMT models used the GNMT (Wu et al., 2016) architecture with Luong attention (Luong et al., 2015b), 2 bidirectional encoder, and 4-layer decoder with residual connections. All hidden layers (including embeddings) had 512 units. Additionally, we used separate embeddings on the encoder and decoder sides as well as tied weights of the softmax that produced logits with the decoder-side (*i.e.*, target) embeddings. Standard dropout of 0.2 was used on all hidden layers. Most of the other hyperparameters we set to default in the T2T (Vaswani et al., 2018) library for the text2text type of problems.

Training and hyperparameters. We scaled agreement terms in the loss by $\gamma = 0.01$. The training was done using Adafactor (Shazeer and Stern, 2018) optimizer with 10,000 burn-in steps at 0.01 learning rate and further standard square root decay (with the default settings for the decay from the T2T library). Additionally, implemented agreement loss as a subgraph as a loss was not computed if γ was set to 0. This allowed us to start training multilingual NMT models in the burn-in mode using the composite likelihood objective and then switch on agreement starting some point during optimization (typically, after the first 100K iterations; we also experimented with 0, 50K, 200K, but did not notice any difference in terms of final performance). Since the agreement subgraph was not computed during the initial training phase, it tended to accelerate training of agreement models.

D.4.2 Details on the datasets

Statistics of the IWSLT17 and IWSLT17* datasets are summarized in Table D.1. UNCorpus and and Europarl datasets were exactly as described by Sestorain et al. (2018) and Chen et al. (2017) and Cheng et al. (2017), respectively.

Table D.1: Data statistics for IWSLT17 and IWSLT17*.

Corpus	**Directions**	**Train**	**Dev** (dev2010)	**Test** (tst2010)
IWSLT17	De → En	206k	888	1568
	De → It	205k	923	1567
	De → Nl	0	1001	1567
	De → Ro	201k	912	1677
	En → De	206k	888	1568
	En → It	231K	929	1566
	En → Nl	237k	1003	1777
	En → Ro	220k	914	1678
	It → De	205k	923	1567
	It → En	231k	929	1566
	It → Nl	205k	1001	1669
	It → Ro	0	914	1643
	Nl → De	0	1001	1779
	Nl → En	237k	1003	1777
	Nl → It	233k	1001	1669
	Nl → Ro	206k	913	1680
	Ro → De	201k	912	1677
	Ro → En	220k	914	1678
	Ro → It	0	914	1643
	Ro → Nl	206k	913	1680
IWSLT17*	De → En	124k	888	1568
	De → It	0	923	1567
	De → Nl	0	1001	1567
	De → Ro	0	912	1677
	En → De	124k	888	1568
	En → It	139k	929	1566
	En → Nl	155k	1003	1777
	En → Ro	128k	914	1678
	It → De	0	923	1567
	It → En	139k	929	1566
	It → Nl	0	1001	1669
	It → Ro	0	914	1643
	Nl → De	0	1001	1779
	Nl → En	155k	1003	1777
	Nl → It	0	1001	1669
	Nl → Ro	0	913	1680
	Ro → De	0	912	1677
	Ro → En	128k	914	1678
	Ro → It	0	914	1643
	Ro → Nl	0	913	1680

Appendix E

Contextual Explanation Networks: Additional Details

E.1 Proofs

E.1.1 Proof of Proposition 7.1

Assume that $p\left(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}\right)$ factorizes as $\prod_{\mathbf{a} \in \mathcal{V}_{\mathbf{Y}}} p\left(\mathbf{Y}_{\mathbf{a}} \mid \mathbf{Y}_{\mathrm{MB}(\mathbf{a})}, \mathbf{X}, \boldsymbol{\theta}_{\mathbf{a}}\right)$, where $\mathbf{a}$ denotes subsets of the $\mathbf{Y}$ variables and $\mathrm{MB}(\mathbf{a})$ stands for the corresponding Markov blankets. Using the definition of CEN given in **??**, we have:

$$\begin{aligned} p\left(\mathbf{Y} \mid \mathbf{X}, \mathbf{C}\right) &= \int p\left(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}\right) p\left(\boldsymbol{\theta} \mid \mathbf{C}\right) d\boldsymbol{\theta} \\ &= \int \prod_{\mathbf{a} \in \mathcal{V}_{\mathbf{Y}}} p\left(\mathbf{Y}_{\mathbf{a}} \mid \mathbf{Y}_{\mathrm{MB}(\mathbf{a})}, \mathbf{X}, \boldsymbol{\theta}_{\mathbf{a}}\right) \prod_{j} p\left(\theta_j \mid \mathbf{C}\right) d\boldsymbol{\theta} \\ &= \prod_{\mathbf{a} \in \mathcal{V}_{\mathbf{Y}}} \left[\int p\left(\mathbf{Y}_{\mathbf{a}} \mid \mathbf{Y}_{\mathrm{MB}(\mathbf{a})}, \mathbf{X}, \boldsymbol{\theta}_{\mathbf{a}}\right) \prod_{j \in \mathbf{a}} p\left(\theta_j \mid \mathbf{C}\right) d\boldsymbol{\theta}_{\mathbf{a}}\right] \\ &= \prod_{\mathbf{a} \in \mathcal{V}_{\mathbf{Y}}} p\left(\mathbf{Y}_{\mathbf{a}} \mid \mathbf{Y}_{\mathrm{MB}(\mathbf{a})}, \mathbf{X}, \mathbf{C}\right) \end{aligned} \tag{E.1}$$

E.1.2 Proof of Proposition 7.2

To derive the lower bound on the contribution of explanations in terms of expected accuracy, we first need to bound the probability of the error when only $\boldsymbol{\theta}$ are used for prediction:

$$\mathbb{P}_e := p\left(\hat{\mathbf{Y}}(\boldsymbol{\theta}) \neq \mathbf{Y}\right) = \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta})}\left[p\left(\hat{\mathbf{Y}} \neq \mathbf{Y} \mid \boldsymbol{\theta}\right)\right],$$

which we bound using the Fano's inequality (Ch. 2.11, Cover and Thomas, 2012):

$$\mathcal{H}\left(\mathbb{P}_e\right) + \mathbb{P}_e \log\left(|\mathcal{Y}| - 1\right) \geq \mathcal{H}\left(\mathcal{Y} \mid \boldsymbol{\theta}\right) \tag{E.2}$$

Since the error $(\hat{\mathbf{Y}}(\boldsymbol{\theta}) \neq \mathbf{Y})$ is a binary random variable, then $\mathcal{H}(\mathbb{P}_e) \leq 1$. After weakening the inequality and using $\mathcal{H}(\mathcal{Y} \mid \boldsymbol{\theta}) \geq \delta$ from the proposition statement, we get:

$$\mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta})}\left[p\left(\hat{\mathbf{Y}} \neq \mathbf{Y} \mid \boldsymbol{\theta}\right)\right] \geq \frac{\mathcal{H}(\mathcal{Y} \mid \boldsymbol{\theta}) - 1}{\log |\mathcal{Y}|} \geq \frac{\delta - 1}{\log |\mathcal{Y}|} \tag{E.3}$$

The claimed lower bound Equation 7.16 follows after we combine Equation E.3 and the assumed bound on the accuracy of the model in terms of ε given in Equation 7.15.

E.1.3 Proof of Theorem 7.1

To prove the theorem, consider the case when f is defined by a CEN, instead of $\mathbf{x}$ we have $(\mathbf{c}, \mathbf{x})$, and the class of approximations, G, coincides with the class of explanations, and hence can be represented by $\boldsymbol{\theta}$. In this setting, we can pose the same problem as:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(f, \boldsymbol{\theta}, \pi_{\mathbf{c},\mathbf{x}}) + \Omega(\boldsymbol{\theta}) \tag{E.4}$$

Suppose that CEN produces $\boldsymbol{\theta}^{\star}$ explanation for the context $\mathbf{c}$ using a deterministic encoder, ϕ. The question is whether and under which conditions $\hat{\boldsymbol{\theta}}$ can recover $\boldsymbol{\theta}^{\star}$. Theorem 7.1 answers the question in affirmative and provides a concentration result for the case when hypotheses are linear. Here, we prove Theorem 7.1 for a little more general class of log-linear explanations: $\operatorname{logit}\{p(Y = 1 \mid \mathbf{x}, \theta)\} = \mathbf{a}(\mathbf{x})^{\top}\boldsymbol{\theta}$, where $\mathbf{a}$ is a C-Lipschitz vector-valued function whose values have a zero-mean distribution when $(\mathbf{x}, \mathbf{c})$ are sampled from $\pi_{\mathbf{x},\mathbf{c}}$[1]. For simplicity of the analysis, we consider binary classification and omit the regularization term, $\Omega(g)$. We define the loss function, $\mathcal{L}(f, \boldsymbol{\theta}, \pi_{\mathbf{x},\mathbf{c}})$, as:

$$\mathcal{L} = \frac{1}{K} \sum_{k=1}^{K} \left(\operatorname{logit}\{p(Y = 1 \mid \mathbf{x}_k - \mathbf{x}, \mathbf{c}_k)\} - \operatorname{logit}\{p(Y = 1 \mid \mathbf{x}_k - \mathbf{x}, \boldsymbol{\theta})\}\right)^2 \tag{E.5}$$

where $(\mathbf{x}_k, \mathbf{c}_k) \sim \pi_{\mathbf{x},\mathbf{c}}$ and $\pi_{\mathbf{x},\mathbf{c}} := \pi_{\mathbf{x}}\pi_{\mathbf{c}}$ is a distribution concentrated around $(\mathbf{x}, \mathbf{c})$. Without loss of generality, we also drop the bias terms in the linear models and assume that $\mathbf{a}(\mathbf{x}_k - \mathbf{x})$ are centered.

Proof. The optimization problem Equation E.4 reduces to the least squares linear regression:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{K} \sum_{k=1}^{K} \left(\operatorname{logit}\{p(Y = 1 \mid \mathbf{x}_k - \mathbf{x}, \mathbf{c}_k)\} - \mathbf{a}(\mathbf{x}_k - \mathbf{x})^{\top}\boldsymbol{\theta}\right)^2 \tag{E.6}$$

We consider deterministic encoding, $p(\boldsymbol{\theta} \mid \mathbf{c}) := \delta(\boldsymbol{\theta}, \boldsymbol{\phi}(\mathbf{c}))$, and hence we have:

$$\begin{aligned} \operatorname{logit}\{p(Y = 1 \mid \mathbf{x}_k - \mathbf{x}, \mathbf{c}_k)\} &= \operatorname{logit}\{p(Y = 1 \mid \mathbf{x}_k - \mathbf{x}, \boldsymbol{\theta} = \boldsymbol{\phi}(\mathbf{c}_k))\} \\ &= \mathbf{a}(\mathbf{x}_k - \mathbf{x})^{\top}\boldsymbol{\phi}(\mathbf{c}_k) \end{aligned} \tag{E.7}$$

[1] In case of logistic regression, $\mathbf{a}(\mathbf{x}) = [1, x_1, \ldots, x_d]^{\top}$.

To simplify the notation, we denote $\mathbf{a}_k := \mathbf{a}(\mathbf{x}_k - \mathbf{x})$, $\boldsymbol{\phi}_k := \boldsymbol{\phi}(\mathbf{c}_k)$, and $\boldsymbol{\phi} := \boldsymbol{\phi}(\mathbf{c})$. The solution of Equation E.6 now can be written in a closed form:

$$\hat{\boldsymbol{\theta}} = \left[\frac{1}{K}\sum_{k=1}^{K} \mathbf{a}_k \mathbf{a}_k^\top\right]^+ \left[\frac{1}{K}\sum_{k=1}^{K} \mathbf{a}_k \mathbf{a}_k^\top \boldsymbol{\phi}_k\right] \tag{E.8}$$

Note that $\hat{\boldsymbol{\theta}}$ is a random variable since $(\mathbf{x}_k, \mathbf{c}_k)$ are randomly generated from $\pi_{\mathbf{x},\mathbf{c}}$. To further simplify the notation, denote $M := \frac{1}{K}\sum_{k=1}^{K} \mathbf{a}_k \mathbf{a}_k^\top$. To get a concentration bound on $\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^\star\|$, we will use the continuity of $\phi(\cdot)$ and $\mathbf{a}(\cdot)$, concentration properties of $\pi_{\mathbf{x},\mathbf{c}}$ around $(\mathbf{x}, \mathbf{c})$, and some elementary results from random matrix theory. To be more concrete, since we assumed that $\pi_{\mathbf{x},\mathbf{c}}$ factorizes, we further let $\pi_{\mathbf{x}}$ and $\pi_{\mathbf{c}}$ concentrate such that $p_{\pi_{\mathbf{x}}}\left(\|\mathbf{x}' - \mathbf{x}\| > t\right) < \varepsilon_{\mathbf{x}}(t)$ and $p_{\pi_{\mathbf{c}}}\left(\|\mathbf{c}' - \mathbf{c}\| > t\right) < \varepsilon_{\mathbf{c}}(t)$, respectively, where $\varepsilon_{\mathbf{x}}(t)$ and $\varepsilon_{\mathbf{c}}(t)$ both go to 0 as $t \to \infty$, potentially at different rates.

First, we have the following bound from the convexity of the norm:

$$\begin{align} p\left(\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^\star\| > t\right) &= p\left(\left\|\frac{1}{K}\sum_{k=1}^{K}\left[M^+ \mathbf{a}_k \mathbf{a}_k^\top (\boldsymbol{\phi}_k - \boldsymbol{\phi})\right]\right\| > t\right) \tag{E.9} \\ &\leq p\left(\frac{1}{K}\sum_{k=1}^{K}\left\|M^+ \mathbf{a}_k \mathbf{a}_k^\top (\boldsymbol{\phi}_k - \boldsymbol{\phi})\right\| > t\right) \tag{E.10} \end{align}$$

By making use of the inequality $\|Ax\| \leq \|A\|\|x\|$, where $\|A\|$ denotes the spectral norm of the matrix A, the L-Lipschitz property of $\phi(\mathbf{c})$, the C-Lipschitz property of $\mathbf{a}(\mathbf{x})$, and the concentration of $\mathbf{x}_k$ around $\mathbf{x}$, we have

$$\begin{align} p\left(\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^\star\| > t\right) &\leq p\left(L\frac{1}{K}\sum_{k=1}^{K}\left\|M^+ \mathbf{a}_k \mathbf{a}_k^\top\right\| \|\mathbf{c}_k - \mathbf{c}\| > t\right) \tag{E.11} \\ &\leq p\left(CL\left\|M^+\right\|\frac{1}{K}\sum_{k=1}^{K}\left\|\mathbf{a}_k \mathbf{a}_k^\top\right\| \|\mathbf{c}_k - \mathbf{c}\| > t\right) \tag{E.12} \\ &\leq p\left(\frac{CL}{\lambda_{\text{min}}(M)}\frac{1}{K}\sum_{k=1}^{K}\|\mathbf{x}_k - \mathbf{x}\|\|\mathbf{c}_k - \mathbf{c}\| > t\right) \tag{E.13} \\ &\leq p\left(\frac{CL\tau^2}{\lambda_{\text{min}}(M)} > t\right) + p\left(\|\mathbf{x}_k - \mathbf{x}\|\|\mathbf{c}_k - \mathbf{c}\| > \tau^2\right) \tag{E.14} \\ &\leq p\left(\lambda_{\text{min}}\left(M/(C\tau)^2\right) < \frac{L}{C^2 t}\right) + \varepsilon_{\mathbf{x}}(\tau) + \varepsilon_{\mathbf{c}}(\tau) \tag{E.15} \end{align}$$

Note that we used the fact that the spectral norm of a rank-1 matrix, $\mathbf{a}(\mathbf{x}_k)\mathbf{a}(\mathbf{x}_k)^\top$, is simply the norm of $\mathbf{a}(\mathbf{x}_k)$, and the spectral norm of the pseudo-inverse of a matrix is equal to the inverse of the least non-zero singular value of the original matrix: $\|M^+\| \leq \lambda_{\text{max}}(M^+) = \lambda_{\text{min}}^{-1}(M)$.

Finally, we need a concentration bound on $\lambda_{\text{min}}\left(M/(C\tau)^2\right)$ to complete the proof. Note that $\frac{M}{C^2\tau^2} = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{\mathbf{a}_k}{C\tau}\right)\left(\frac{\mathbf{a}_k}{C\tau}\right)^\top$, where the norm of $\left(\frac{\mathbf{a}_k}{C\tau}\right)$ is bounded by 1. If we denote $\mu_{\text{min}}(C\tau)$

the minimal eigenvalue of Cov $\left[\frac{\mathbf{a}_k}{C\tau}\right]$, we can write the matrix Chernoff inequality (Tropp, 2012) as follows:

$$p\left(\lambda_{\min}\left(M/(C\tau)^2\right) < \alpha\right) \leq d\exp\left\{-KD(\alpha \| \mu_{\min}(C\tau))\right\}, \quad \alpha \in [0, \mu_{\min}(C\tau)]$$

where d is the dimension of $\mathbf{a}_k$, $\alpha := \frac{L}{C^2 t}$, and $D(a\|b)$ denotes the binary information divergence:

$$D(a\|b) = a\log\left(\frac{a}{b}\right) + (1-a)\log\left(\frac{1-a}{1-b}\right).$$

The final concentration bound has the following form:

$$p\left(\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^\star\| > t\right) \leq d\exp\left\{-KD\left(\frac{L}{C^2 t}\|\mu_{\min}(C\tau)\right)\right\} + \varepsilon_{\mathbf{x}}(\tau) + \varepsilon_{\mathbf{c}}(\tau) \tag{E.16}$$

We see that as $\tau \to \infty$ and $t \to \infty$ all terms on the right hand side vanish, and hence $\hat{\boldsymbol{\theta}}$ concentrates around $\boldsymbol{\theta}^\star$. Note that as long as $\mu_{\min}(C\tau)$ is far from 0, the first term can be made negligibly small by sampling more points around $(\mathbf{x}, \mathbf{c})$. Finally, we set $\tau \equiv t$ and denote the right hand side by $\delta_{K,L,C}(t)$ that goes to 0 as $t \to \infty$ to recover the statement of the original theorem. □

Remark E.1. *We have shown that $\hat{\boldsymbol{\theta}}$ concentrates around $\boldsymbol{\theta}^\star$ under mild conditions. With more assumptions on the sampling distribution, $\pi_{\mathbf{x},\mathbf{c}}$, (e.g., sub-gaussian) one could derive precise convergence rates. Note that we are in total control of any assumptions we put on $\pi_{\mathbf{x},\mathbf{c}}$ since precisely that distribution is used for sampling. This is a major difference between the local approximation setup here and the setup of linear regression with random design; in the latter case, we have no control over the distribution of the design matrix, and any assumptions we make could potentially be unrealistic.*

Remark E.2. *Note that concentration analysis of a more general case when the loss $\mathcal{L}$ is a general convex function and $\Omega(g)$ is a decomposable regularizer could be done by using results from the M-estimation theory (Negahban et al., 2009), but would be much more involved and unnecessary for our purposes.*

E.2 Experimental Details

This section provides details on the experimental setups including architectures, training procedures, etc. Additionally, we provide and discuss qualitative results for CENs on the MNIST and IMDB datasets.

E.2.1 Additional Details on the Datasets and Experiment Setups

MNIST. We used the classical split of the dataset into 50k training, 10k validation, and 10k testing points. All models were trained for 100 epochs using the AMSGrad optimizer (Reddi et al., 2019) with the learning rate of 10^{-3}. No data augmentation was used in any of our experiments. HOG representations were computed using 3×3 blocks.

CIFAR10. For this set of experiments, we followed the setup given by Zagoruyko (2015), reimplemented in Keras (Chollet et al., 2015) with TensorFlow (Abadi et al., 2016) backend. The input images were global contrast normalized (a.k.a. GCN whitened) while the rescaled image representations were simply standardized. Again, HOG representations were computed using 3×3 blocks. No data augmentation was used in our experiments.

IMDB. We considered the labeled part of the data only (50,000 reviews total). The data were split into 20,000 train, 5,000 validation, and 25,000 test points. The vocabulary was limited to 20,000 most frequent words (and 5,000 most frequent words when constructing BoW representations). All models were trained with the AMSGrad optimizer () with 10^{-2} learning rate. The models were initialized randomly; no pre-training or any other unsupervised/semi-supervised technique was used.

Satellite. As described in the main text, we used a pre-trained VGG-16 network[2] to extract features from the satellite imagery. Further, we added one fully connected layer network with 128 hidden units used as the context encoder. For the VCEN model, we used dictionary-based encoding with Dirichlet prior and logistic normal distribution as the output of the inference network. For the decoder, we used an MLP of the same architecture as the encoder network. All models were trained with Adam optimizer with 0.05 learning rate. The results were obtained by 5-fold cross-validation.

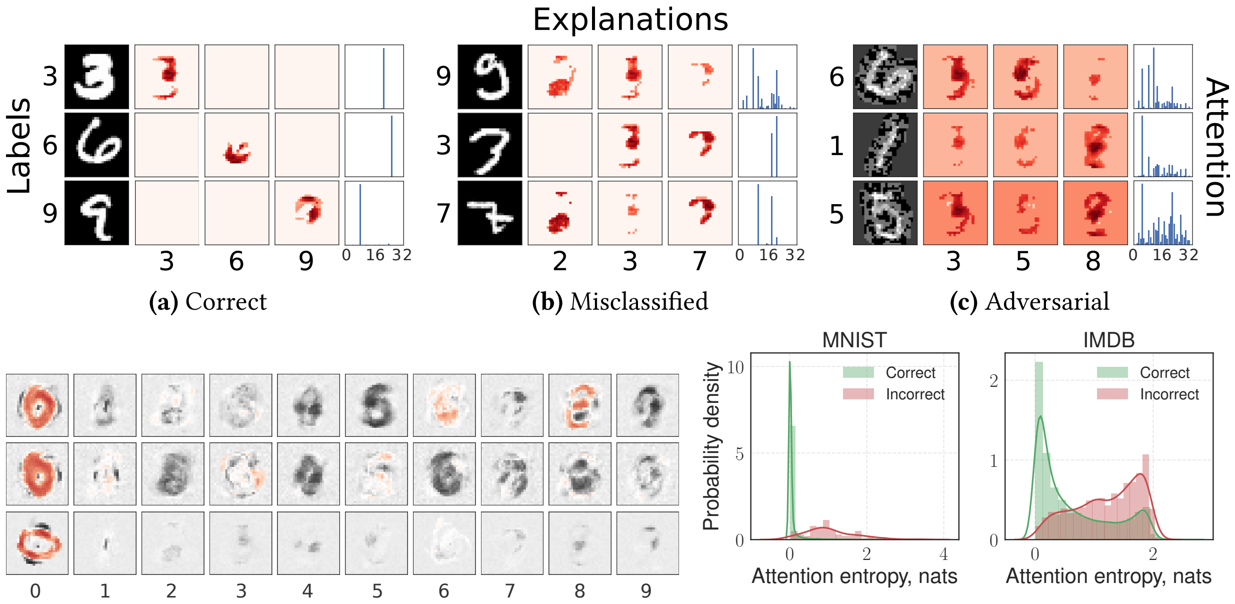
Medical data. We have used minimal pre-processing of both SUPPORT2 and PhysioNet datasets limited to standardization and missing-value filling. We found that denoting missing values with negative entries (-1) often led a slightly improved performance compared to any other NA-filling techniques. PhysioNet time series data was irregularly sampled across the time, so we had to resample temporal sequences at regular intervals of 30 minutes (consequently, this has created quite a few missing values for some of the measurements). All models were trained using Adam optimizer with 10^{-2} learning rate.

E.2.2 More on Qualitative Analysis

MNIST

Figures E.1a to E.1c visualize explanations for predictions made by CEN_{pxl} on MNIST. The figures correspond to 3 cases where CEN (a) made a correct prediction, (b) made a mistake, and (c) was applied to an adversarial example (and made a mistake). Each chart consists of the following columns: true labels, input images, explanations for the top 3 classes (as given by the activation of the final softmax layer), and attention vectors used to select explanations from the global dictionary. A small subset of explanations from the dictionary is visualized in Figure E.1d (the full dictionary is given in Figure E.2), where each image is a weight vector used to construct the pre-activation for a particular class. Note that different elements of the dictionary capture different patterns in the data (in Figure E.1d, different styles of writing the 0 digit) which CEN actually uses for prediction.

[2] The model was taken form https://github.com/nealjean/predicting-poverty.



(a) Correct

(b) Misclassified

(c) Adversarial

(d) Selected elements of the explanation dictionary

(e) Attention

Figure E.1: Explanations generated by CEN for the 3 top classes and the corresponding attention vectors for (a) correctly classified, (b) misclassified, and (c) adversarially constructed images. Adversarial examples were generated using the fast gradient sign method (FGSM) (Papernot et al., 2016). (d) Elements from the learned 32-element dictionary that correspond to different writing styles of 0 digits. (e) Histogram of the attention entropy for correctly and incorrectly classified test instances for CEN-`px1` on MNIST and CEN_{tpc} on IMDB.

Also note that confident correct predictions (Figure E.1a) are made by selecting a single explanation from the dictionary using a sharp attention vector. However, when the model makes a mistake, its attention is often dispersed (Figures E.1b and E.1c), i.e., there is uncertainty in which pattern it tries to use for prediction. Figure E.1e further quantifies this phenomenon by plotting histogram of the attention entropy for all test examples which were correctly and incorrectly classified. While CENs are certainly not adversarial-proof, high entropy of the attention vectors is indicative of ambiguous or out-of-distribution examples which is helpful for model diagnostics.

IMDB

Similar to MNIST, we train CEN_{tpc} with linear explanations in terms of topics on the IMDB dataset. Then, we generate explanations for each test example and visualize histograms of the weights assigned by the explanations to 6 selected topics in ??. The 3 topics in the top row are acting- and plot-related (and intuitively have positive, negative, or neutral connotation), while the 3 topics in the bottom are related to particular genre of the movies.

Note that acting-related topics turn out to be bi-modal, i.e., contributing either positively, negatively, or neutrally to the sentiment prediction in different contexts. As expected intuitively, CEN assigns highly negative weight to the topic related to "bad acting/plot" and highly positive weight to "great story/performance" in most of the contexts (and treats those neutrally conditional on some of the reviews). Interestingly, genre-related topics almost always have a negligible contribution to the sentiment (i.e., get almost 0 weights assigned by explanations) which indicates that the learned model does not have any particular bias towards or against a given genre. Importantly, inspecting summary statistics of the explanations generated by CEN allows us to explore the biases that the model picks up from the data and actively uses for prediction[3].

Figure E.3 visualizes the full dictionary of size 16 learned by CEN_{tpc}. Each column corresponds to a dictionary atom that represents a typical explanation pattern that CEN attends to before making a prediction. By inspecting the dictionary, we can find interesting patterns. For instance, atoms 5 and 11 assign inverse weights to topics `[kid, child, disney, family]` and `[sexual, violence, nudity, sex]`. Depending on the context of the review, CEN may use one of these patterns to predict the sentiment. Note that these two topics are negatively correlated across all dictionary elements, which again is quite intuitive.

Satellite

We visualize the two explanations, M1 and M2, learned by CEN_{att} on the Satellite dataset in full in Figures E.4a and provide additional correlation plots between the selected explanation and values of each survey variable in Figure E.4b.

E.2.3 Model Architectures

Architectures of the model used in our experiments are summarized in Tables E.1 to E.3.

[3] If we wish to enforce or eliminate certain patterns from explanations (e.g., to ensure fairness), we may impose additional constraints on the dictionary. However, this is beyond the scope of this work.

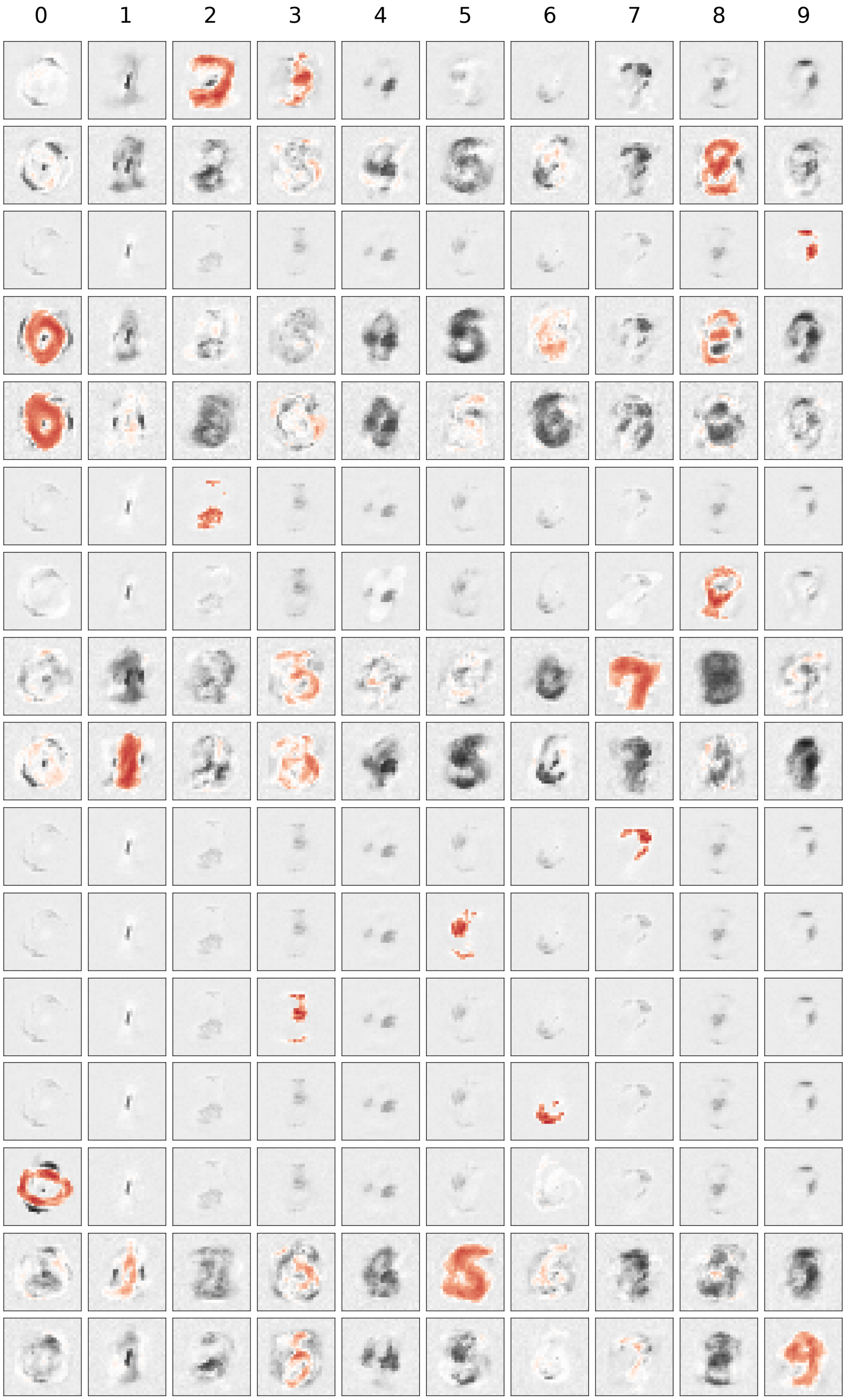
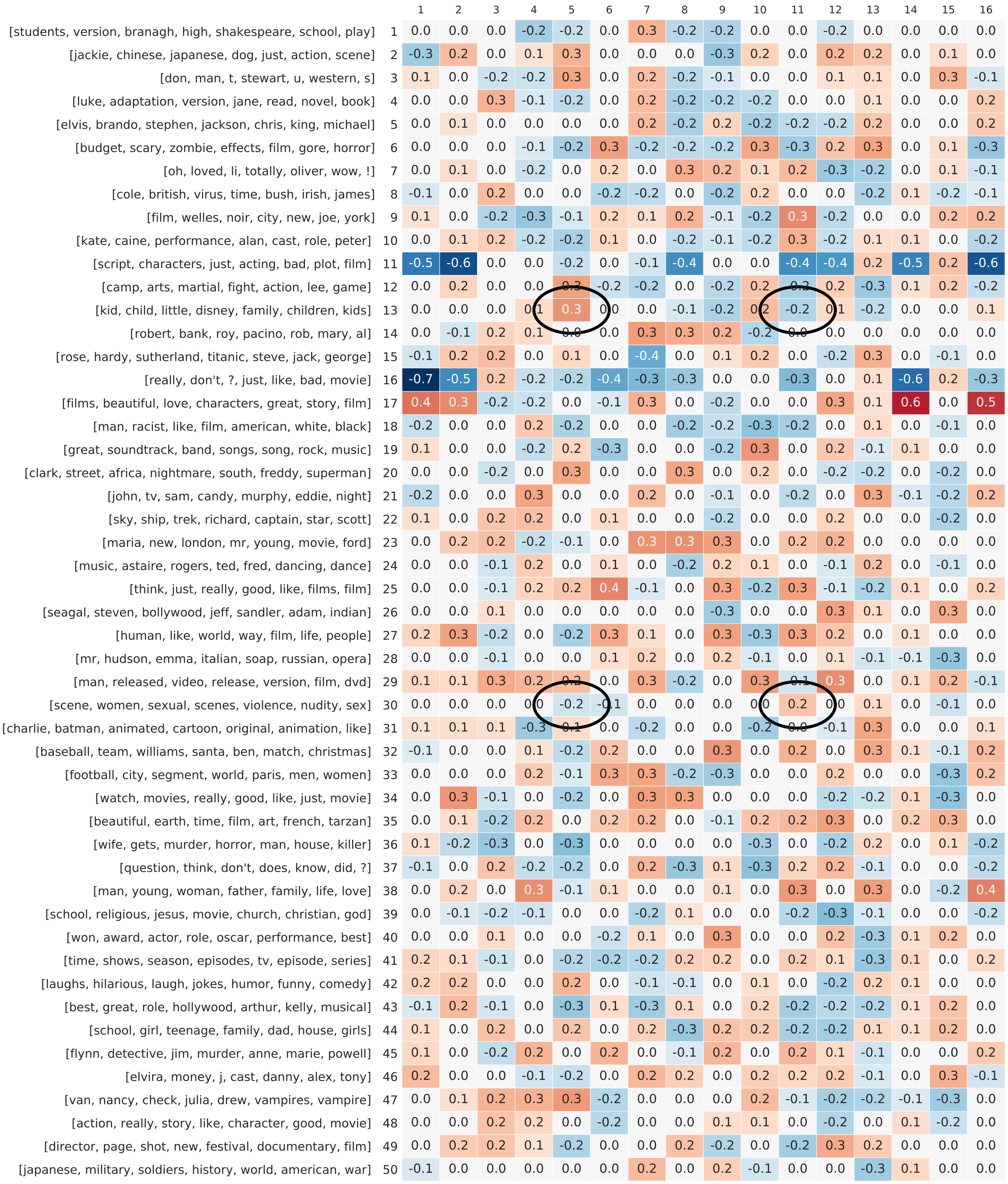
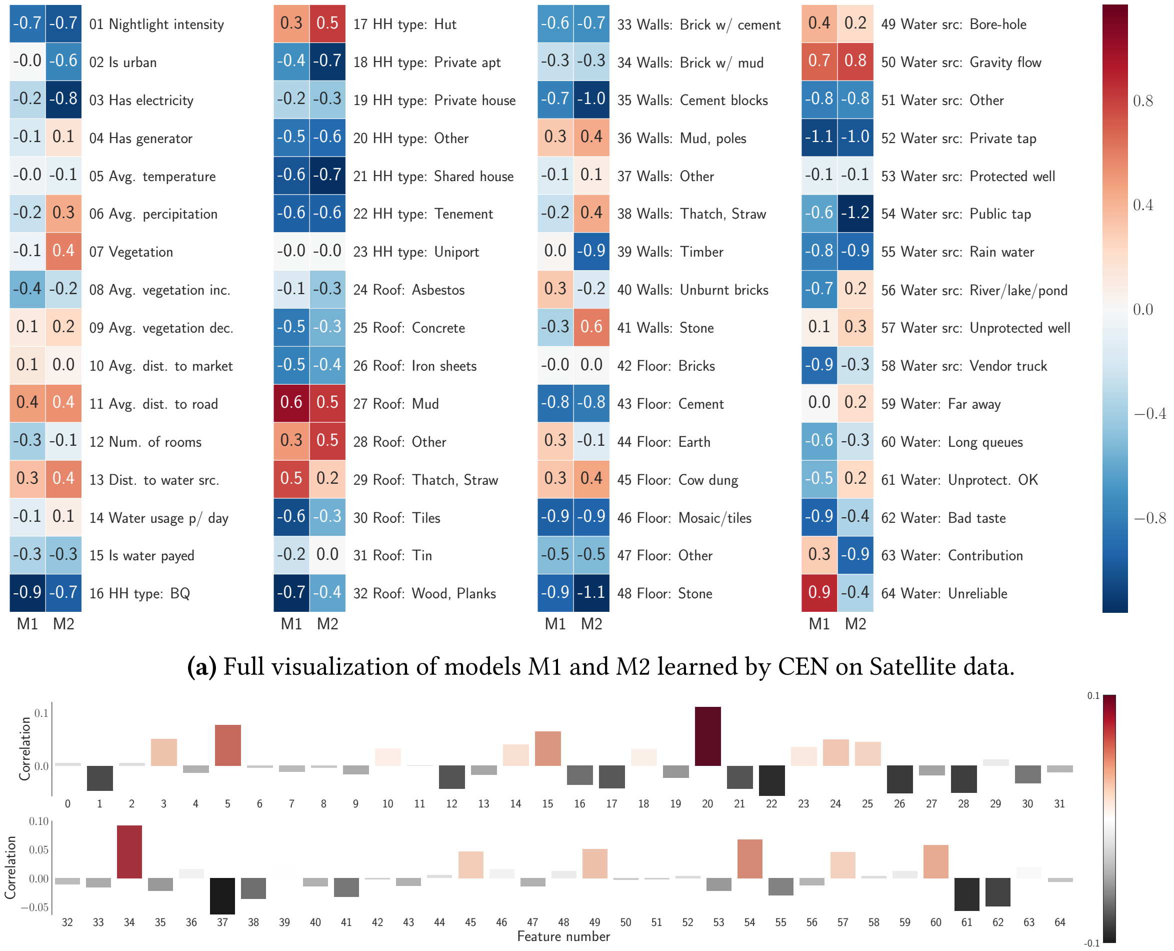


Figure E.2: Visualization of the model dictionary learned by CEN on MNIST. Each row corresponds to a dictionary element, and each column corresponds to the weights of the model voting for each class of digits. Images visualize the weights of the models. Red corresponds to high positive values, dark gray to high negative values, and white to values that are close to 0.



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
[students, version, branagh, high, shakespeare, school, play]	1	0.0	0.0	0.0	-0.2	-0.2	0.0	0.3	-0.2	-0.2	0.0	0.0	-0.2	0.0	0.0	0.0	0.0
[jackie, chinese, japanese, dog, just, action, scene]	2	-0.3	0.2	0.0	0.1	0.3	0.0	0.0	0.0	-0.3	0.2	0.0	0.2	0.2	0.0	0.1	0.0
[don, man, t, stewart, u, western, s]	3	0.1	0.0	-0.2	-0.2	0.3	0.0	0.2	-0.2	-0.1	0.0	0.0	0.1	0.1	0.0	0.3	-0.1
[luke, adaptation, version, jane, read, novel, book]	4	0.0	0.0	0.3	-0.1	-0.2	0.0	0.2	-0.2	-0.2	-0.2	0.0	0.0	0.1	0.0	0.0	0.2
[elvis, brando, stephen, jackson, chris, king, michael]	5	0.0	0.1	0.0	0.0	0.0	0.0	0.2	-0.2	0.2	-0.2	-0.2	-0.2	0.2	0.0	0.0	0.2
[budget, scary, zombie, effects, film, gore, horror]	6	0.0	0.0	0.0	-0.1	-0.2	0.3	-0.2	-0.2	-0.2	0.3	-0.3	0.2	0.3	0.0	0.1	-0.3
[oh, loved, li, totally, oliver, wow, !]	7	0.0	0.1	0.0	-0.2	0.0	0.2	0.0	0.3	0.2	0.1	0.2	-0.3	-0.2	0.0	0.1	-0.1
[cole, british, virus, time, bush, irish, james]	8	-0.1	0.0	0.2	0.0	0.0	-0.2	-0.2	0.0	-0.2	0.2	0.0	0.0	-0.2	0.1	-0.2	-0.1
[film, welles, noir, city, new, joe, york]	9	0.1	0.0	-0.2	-0.3	-0.1	0.2	0.1	0.2	-0.1	-0.2	0.3	-0.2	0.0	0.0	0.2	0.2
[kate, caine, performance, alan, cast, role, peter]	10	0.0	0.1	0.2	-0.2	-0.2	0.1	0.0	-0.2	-0.1	-0.2	0.3	-0.2	0.1	0.1	0.0	-0.2
[script, characters, just, acting, bad, plot, film]	11	-0.5	-0.6	0.0	0.0	-0.2	0.0	-0.1	-0.4	0.0	0.0	-0.4	-0.4	0.2	-0.5	0.2	-0.6
[camp, arts, martial, fight, action, lee, game]	12	0.0	0.2	0.0	0.0	0.3	-0.2	-0.2	0.0	-0.2	0.2	-0.2	0.2	-0.3	0.1	0.2	-0.2
[kid, child, little, disney, family, children, kids]	13	0.0	0.0	0.0	0.1	0.3	0.0	0.0	-0.1	-0.2	0.2	-0.2	0.1	-0.2	0.0	0.0	0.1
[robert, bank, roy, pacino, rob, mary, al]	14	0.0	-0.1	0.2	0.1	0.0	0.0	0.3	0.3	0.2	-0.2	0.0	0.0	0.0	0.0	0.0	0.0
[rose, hardy, sutherland, titanic, steve, jack, george]	15	-0.1	0.2	0.2	0.0	0.1	0.0	-0.4	0.0	0.1	0.2	0.0	-0.2	0.3	0.0	-0.1	0.0
[really, don't, ?, just, like, bad, movie]	16	-0.7	-0.5	0.2	-0.2	-0.2	-0.4	-0.3	-0.3	0.0	0.0	-0.3	0.0	0.1	-0.6	0.2	-0.3
[films, beautiful, love, characters, great, story, film]	17	0.4	0.3	-0.2	-0.2	0.0	-0.1	0.3	0.0	-0.2	0.0	0.0	0.3	0.1	0.6	0.0	0.5
[man, racist, like, film, american, white, black]	18	-0.2	0.0	0.0	0.2	-0.2	0.0	0.0	-0.2	-0.2	-0.3	-0.2	0.0	0.1	0.0	-0.1	0.0
[great, soundtrack, band, songs, song, rock, music]	19	0.1	0.0	0.0	-0.2	0.2	-0.3	0.0	0.0	-0.2	0.3	0.0	0.2	-0.1	0.1	0.0	0.0
[clark, street, africa, nightmare, south, freddy, superman]	20	0.0	0.0	-0.2	0.0	0.3	0.0	0.0	0.3	0.0	0.2	0.0	-0.2	-0.2	0.0	-0.2	0.0
[john, tv, sam, candy, murphy, eddie, night]	21	-0.2	0.0	0.0	0.3	0.0	0.0	0.2	0.0	-0.1	0.0	-0.2	0.0	0.3	-0.1	-0.2	0.2
[sky, ship, trek, richard, captain, star, scott]	22	0.1	0.0	0.2	0.2	0.0	0.1	0.0	0.0	-0.2	0.0	0.0	0.2	0.0	0.0	-0.2	0.0
[maria, new, london, mr, young, movie, ford]	23	0.0	0.2	0.2	-0.2	-0.1	0.0	0.3	0.3	0.3	0.0	0.2	0.2	0.0	0.0	0.0	0.0
[music, astaire, rogers, ted, fred, dancing, dance]	24	0.0	0.0	-0.1	0.2	0.0	0.1	0.0	-0.2	0.2	0.1	0.0	-0.1	0.2	0.0	-0.1	0.0
[think, just, really, good, like, films, film]	25	0.0	0.0	-0.1	0.2	0.2	0.4	-0.1	0.0	0.3	-0.2	0.3	-0.1	-0.2	0.1	0.0	0.2
[seagal, steven, bollywood, jeff, sandler, adam, indian]	26	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	-0.3	0.0	0.0	0.3	0.1	0.0	0.3	0.0
[human, like, world, way, film, life, people]	27	0.2	0.3	-0.2	0.0	-0.2	0.3	0.1	0.0	0.3	-0.3	0.3	0.2	0.0	0.1	0.0	0.0
[mr, hudson, emma, italian, soap, russian, opera]	28	0.0	0.0	-0.1	0.0	0.0	0.1	0.2	0.0	0.2	-0.1	0.0	0.1	-0.1	-0.1	-0.3	0.0
[man, released, video, release, version, film, dvd]	29	0.1	0.1	0.3	0.2	0.2	0.0	0.3	-0.2	0.0	0.3	-0.1	0.3	0.0	0.1	0.2	-0.1
[scene, women, sexual, scenes, violence, nudity, sex]	30	0.0	0.0	0.0	0.0	-0.2	-0.1	0.0	0.0	0.0	0.0	0.2	0.0	0.1	0.0	-0.1	0.0
[charlie, batman, animated, cartoon, original, animation, like]	31	0.1	0.1	0.1	-0.3	0.1	0.0	-0.2	0.0	0.0	-0.2	0.0	-0.1	0.3	0.0	0.0	0.1
[baseball, team, williams, santa, ben, match, christmas]	32	-0.1	0.0	0.0	0.1	-0.2	0.2	0.0	0.0	0.3	0.0	0.2	0.0	0.3	0.1	-0.1	0.2
[football, city, segment, world, paris, men, women]	33	0.0	0.0	0.0	0.2	-0.1	0.3	0.3	-0.2	-0.3	0.0	0.0	0.2	0.0	0.0	-0.3	0.2
[watch, movies, really, good, like, just, movie]	34	0.0	0.3	-0.1	0.0	-0.2	0.0	0.3	0.3	0.0	0.0	0.0	-0.2	-0.2	0.1	-0.3	0.0
[beautiful, earth, time, film, art, french, tarzan]	35	0.0	0.1	-0.2	0.2	0.0	0.2	0.2	0.0	-0.1	0.2	0.2	0.3	0.0	0.2	0.3	0.0
[wife, gets, murder, horror, man, house, killer]	36	0.1	-0.2	-0.3	0.0	-0.3	0.0	0.0	0.0	0.0	-0.3	0.0	-0.2	0.2	0.0	0.1	-0.2
[question, think, don't, does, know, did, ?]	37	-0.1	0.0	0.2	-0.2	-0.2	0.0	0.2	-0.3	0.1	-0.3	0.2	0.2	-0.1	0.0	0.0	-0.2
[man, young, woman, father, family, life, love]	38	0.0	0.2	0.0	0.3	-0.1	0.1	0.0	0.0	0.1	0.0	0.3	0.0	0.3	0.0	-0.2	0.4
[school, religious, jesus, movie, church, christian, god]	39	0.0	-0.1	-0.2	-0.1	0.0	0.0	-0.2	0.1	0.0	0.0	-0.2	-0.3	-0.1	0.0	0.0	-0.2
[won, award, actor, role, oscar, performance, best]	40	0.0	0.0	0.1	0.0	0.0	-0.2	0.1	0.0	0.3	0.0	0.0	0.2	-0.3	0.1	0.2	0.0
[time, shows, season, episodes, tv, episode, series]	41	0.2	0.1	-0.1	0.0	-0.2	-0.2	-0.2	0.2	0.2	0.0	0.2	0.1	-0.3	0.1	0.0	0.2
[laughs, hilarious, laugh, jokes, humor, funny, comedy]	42	0.2	0.2	0.0	0.0	0.2	0.0	-0.1	-0.1	0.0	0.1	0.0	-0.2	0.2	0.1	0.0	0.0
[best, great, role, hollywood, arthur, kelly, musical]	43	-0.1	0.2	-0.1	0.0	-0.3	0.1	-0.3	0.1	0.0	0.2	-0.2	-0.2	-0.2	0.1	0.2	0.0
[school, girl, teenage, family, dad, house, girls]	44	0.1	0.0	0.2	0.0	0.2	0.0	0.2	-0.3	0.2	0.2	-0.2	-0.2	0.1	0.1	0.2	0.0
[flynn, detective, jim, murder, anne, marie, powell]	45	0.1	0.0	-0.2	0.2	0.0	0.2	0.0	-0.1	0.2	0.0	0.2	0.1	-0.1	0.0	0.0	0.2
[elvira, money, j, cast, danny, alex, tony]	46	0.2	0.0	0.0	-0.1	-0.2	0.0	0.2	0.2	0.0	0.2	0.2	0.2	-0.1	0.0	0.3	-0.1
[van, nancy, check, julia, drew, vampires, vampire]	47	0.0	0.1	0.2	0.3	0.3	-0.2	0.0	0.0	0.0	0.2	-0.1	-0.2	-0.2	-0.1	-0.3	0.0
[action, really, story, like, character, good, movie]	48	0.0	0.0	0.2	0.2	0.0	-0.2	0.0	0.0	0.1	0.1	0.0	-0.2	0.0	0.1	-0.2	0.0
[director, page, shot, new, festival, documentary, film]	49	0.0	0.2	0.2	0.1	-0.2	0.0	0.0	0.2	-0.2	0.0	-0.2	0.3	0.2	0.0	0.0	0.0
[japanese, military, soldiers, history, world, american, war]	50	-0.1	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.2	-0.1	0.0	0.0	-0.3	0.1	0.0	0.0

Figure E.3: The full dictionary learned by CEN_{tpc} model: rows correspond to topics and columns correspond to dictionary atoms. Very small values were thresholded for visualization clarity. Different atoms capture different prediction patterns; for example, atom 5 assigns a highly positive weight to the `[kid, child, disney, family]` topic and down-weighs `[sexual, violence, nudity, sex]`, while atom 11 acts in an opposite manner. Given the context of the review, CEN combines just a few atoms to make a prediction.



(a) Full visualization of models M1 and M2 learned by CEN on Satellite data.

(b) Correlation between the selected explanation and the value of a particular survey variable.

Figure E.4: Additional visualizations for CENs trained on the Satellite data.

Table E.1: Top-performing architectures used in our experiments on MNIST and IMDB datasets.

(a) MNIST

Convolutional Encoder		
Convolutional Block	layer	Conv2D
	# filters	32
	kernel size	3×3
	strides	1×1
	padding	valid
	activation	ReLU
	layer	Conv2D
	# filters	32
	kernel size	3×3
	strides	1×1
	padding	valid
	activation	ReLU
	layer	MaxPoo2D
	pooling size	2×2
	dropout	0.25
	layer	Dense
	units	128
	dropout	0.50
# blocks		1
# params		1.2M

Contextual Explanations	
model	Logistic regr.
features	HOG (3, 3)
# features	729
standardized	Yes
dictionary	256
l_1 penalty	$5 \cdot 10^{-5}$
l_2 penalty	$1 \cdot 10^{-6}$
model	Logistic reg.
features	Pixels (20, 20)
# features	400
standardized	Yes
dictionary	64
l_1 penalty	$5 \cdot 10^{-5}$
l_2 penalty	$1 \cdot 10^{-6}$
Contextual VAE	
prior	Dir(0.2)
sampler	LogisticNormal

(b) IMDB

Squential Encoder	
layer	Embedding
vocabulary	20k
dimension	1024
layer	LSTM
bidirectional	Yes
units	256
max length	200
dropout	0.25
rec. dropout	0.25
layer	MaxPool1D
# params	23.1M

Contextual Explanations	
model	Logistic reg.
features	BoW
# features	20k
Dictionary	32
l_1 penalty	$5 \cdot 10^{-5}$
l_2 penalty	$1 \cdot 10^{-6}$
model	Logistic reg.
features	Topics
# features	50
Dictionary	16
l_1 penalty	$1 \cdot 10^{-6}$
l_2 penalty	$1 \cdot 10^{-8}$
Contextual VAE	
Prior	Dir(0.1)
Sampler	LogisticNormal

Table E.2: Top-performing architectures used in our experiments on CIFAR10 and Satellite datasets. VGG-16 architecture for CIFAR10 was taken from https://github.com/szagoruyko/cifar.torch but implemented in Keras with TensorFlow backend. Weights of the pre-trained VGG-F model for the Satellite experiments were taken from https://github.com/nealjean/predicting-poverty.

(a) CIFAR10

Convolutional Encoder		
VGG-16	model	VGG-16
	pretrained	No
	fixed weights	No
MLP	layer	Dense
	pretrained	No
	fixed weights	No
	units	16
	dropout	0.25
	activation	ReLU
# params		20.0M

Contextual Explanations	
model	Logistic reg.
features	HOG (3, 3)
# features	1024
dictionary	16
l_1 penalty	$1 \cdot 10^{-5}$
l_2 penalty	$1 \cdot 10^{-6}$
Contextual VAE	
prior	Dir(0.2)
sampler	LogisticNormal

(b) Satellite

Convolutional Encoder		
VGG-F	model	VGG-F
	pretrained	Yes
	fixed weights	Yes
MLP	layer	Dense
	pretrained	No
	fixed weights	No
	units	128
	dropout	0.25
	activation	ReLU
# trainable params		0.5M

Contextual Explanations	
model	Logistic reg.
features	Survey
# features	64
dictionary	16
l_1 penalty	$1 \cdot 10^{-3}$
l_2 penalty	$1 \cdot 10^{-4}$
# params	
Contextual VAE	
prior	Dir(0.2)
sampler	LogisticNormal

Table E.3: Top-performing architectures used in our experiments on SUPPORT2 and PhysioNet.

(a) SUPPORT2

MLP Encoder		
MLP	layer	Dense
	pretrained	No
	fixed weights	No
	units	64
	dropout	0.50
	activation	ReLU

Contextual Explanations	
model	Linear CRF
features	Measurements
# features	50
dictionary	16
l_1 penalty	$1 \cdot 10^{-3}$
l_2 penalty	$1 \cdot 10^{-4}$

(b) PhysioNet Challenge 2012

Sequential Encoder		
LSTM	layer	LSTM
	bidirectional	No
	units	32
	max length	150
	dropout	0.25
	rec. dropout	0.25

Contextual Explanations	
model	Linear CRF
features	Statistics
# features	111
dictionary	16
l_1 penalty	$1 \cdot 10^{-3}$
l_2 penalty	$1 \cdot 10^{-4}$

Bibliography

[1] Herbert Robbins et al. "An Empirical Bayes Approach to Statistics". In: *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California. 1956 (cit. on p. 18).

[2] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17 (cit. on p. 116).

[3] DR Cox. "Regression Models and Life-Tables". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1972), pp. 187–220 (cit. on p. 102).

[4] Julian Besag. "Statistical analysis of non-lattice data". In: *The statistician* (1975), pp. 179–195 (cit. on pp. 26, 68).

[5] Jurgen Schmidhuber. "Evolutionary principles in self-referential learning". In: *On learning how to learn: The meta-meta-... hook.) Diploma thesis, Institut f. Informatik, Tech. Univ. Munich* (1987) (cit. on p. 50).

[6] Terrence J Sejnowski and Charles R Rosenberg. "Parallel networks that learn to pronounce English text". In: *Complex systems* 1.1 (1987), pp. 145–168 (cit. on p. 1).

[7] Bradley Efron. "Logistic regression, survival analysis, and the Kaplan-Meier curve". In: *Journal of the American statistical Association* 83.402 (1988), pp. 414–425 (cit. on p. 102).

[8] Steffen L Lauritzen and David J Spiegelhalter. "Local computations with probabilities on graphical structures and their application to expert systems". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 50.2 (1988), pp. 157–194 (cit. on pp. 2, 22).

[9] Bruce G Lindsay. "Composite likelihood methods". In: *Contemporary mathematics* 80.1 (1988), pp. 221–239 (cit. on pp. 26, 68).

[10] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988 (cit. on pp. 2, 22).

[11] O.O. Aalen. "A linear regression model for the analysis of life time". In: *Statistics in Medicine, 8(8):907–925* (1989) (cit. on p. 102).

[12] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation.* Vol. 24. Elsevier, 1989, pp. 109–165 (cit. on pp. 20, 50).

[13] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. "Adaptive mixtures of local experts". In: *Neural computation* 3.1 (1991), pp. 79–87 (cit. on p. 88).

[14] David JC MacKay. "Bayesian methods for adaptive models". PhD thesis. California Institute of Technology, 1992 (cit. on pp. 5, 18).

[15] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: *SIAM journal on control and optimization* 30.4 (1992), pp. 838–855 (cit. on pp. 38, 116).

[16] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256 (cit. on pp. 53, 131).

[17] Trevor Hastie and Robert Tibshirani. "Varying-coefficient models". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1993), pp. 757–796 (cit. on pp. 88, 89).

[18] Mark B Ring. "Continual learning in reinforcement environments". PhD thesis. University of Texas at Austin Austin, Texas 78712, 1994 (cit. on pp. 1, 19, 24, 50).

[19] Bruce G Lindsay. "Mixture models: theory, geometry and applications". In: *NSF-CBMS regional conference series in probability and statistics.* JSTOR. 1995, pp. i–163 (cit. on p. 89).

[20] Jun S Liu. "Metropolized independent sampling with comparisons to rejection sampling and importance sampling". In: *Statistics and computing* 6.2 (1996), pp. 113–119 (cit. on p. 117).

[21] Jonathan Baxter. "Theoretical models of learning to learn". In: *Learning to learn.* Springer, 1998, pp. 71–94 (cit. on pp. 16, 19).

[22] Rich Caruana. "Multitask learning". In: *Learning to learn.* Springer, 1998, pp. 95–133 (cit. on pp. 1, 23, 50).

[23] Sebastian Thrun and Lorien Pratt. *Learning to learn.* Springer, 1998 (cit. on pp. 19, 50).

[24] Wenxin Jiang and Martin A Tanner. "Hierarchical mixtures-of-experts for exponential family regression models: approximation and maximum likelihood estimation". In: *Annals of Statistics* (1999), pp. 987–1011 (cit. on p. 89).

[25] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. "An introduction to variational methods for graphical models". In: *Machine learning* 37.2 (1999), pp. 183–233 (cit. on p. 25).

[26] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. Vol. 99. 1999, pp. 278–287 (cit. on p. 2).

[27] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer Science & Business Media, 1999 (cit. on p. 18).

[28] Yiming Yang. "An evaluation of statistical approaches to text categorization". In: *Information retrieval* 1.1-2 (1999), pp. 69–90 (cit. on p. 46).

[29] Yiming Yang and Xin Liu. "A re-examination of text categorization methods". In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval.* 1999, pp. 42–49 (cit. on p. 46).

[30] Jonathan Baxter. "A model of inductive bias learning". In: *Journal of artificial intelligence research* 12 (2000), pp. 149–198 (cit. on pp. 15, 21, 111).

[31] Satinder Singh, Michael Kearns, and Yishay Mansour. “Nash convergence of gradient dynamics in general-sum games”. In: *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2000, pp. 541–548 (cit. on p. 51).

[32] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063 (cit. on pp. 6, 55, 130–132).

[33] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. “Factor graphs and the sum-product algorithm”. In: *IEEE Transactions on information theory* 47.2 (2001), pp. 498–519 (cit. on pp. 4, 22).

[34] John Lafferty, Andrew McCallum, Fernando Pereira, et al. “Conditional random fields: Probabilistic models for segmenting and labeling sequence data”. In: *Proceedings of the eighteenth international conference on machine learning, ICML*. Vol. 1. 2001, pp. 282–289 (cit. on pp. 84, 87).

[35] Thomas P Minka. “Expectation Propagation for approximate Bayesian inference”. In: *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*. 2001, pp. 362–369 (cit. on pp. 30, 36).

[36] Olivier Bousquet and André Elisseeff. “Stability and generalization”. In: *The Journal of Machine Learning Research* 2 (2002), pp. 499–526 (cit. on pp. 111, 112).

[37] Brendan J Frey. “Extending factor graphs so as to unify directed and undirected graphical models”. In: *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. 2002, pp. 257–264 (cit. on pp. 4, 13, 22).

[38] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. “BLEU: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318 (cit. on p. 75).

[39] David M Blei, Andrew Y Ng, and Michael I Jordan. “Latent dirichlet allocation”. In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022 (cit. on p. 95).

[40] Isabelle Guyon. “Design of experiments of the NIPS 2003 variable selection benchmark”. In: *NIPS 2003 workshop on feature extraction and feature selection*. Vol. 253. 2003 (cit. on p. 40).

[41] Neil D Lawrence and John C Platt. “Learning to learn with the informative vector machine”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 65 (cit. on p. 24).

[42] Olivier Ledoit and Michael Wolf. “A well-conditioned estimator for large-dimensional covariance matrices”. In: *Journal of multivariate analysis* 88.2 (2004), pp. 365–411 (cit. on pp. 38, 119, 120).

[43] Olivier Ledoit and Michael Wolf. “Honey, I shrunk the sample covariance matrix”. In: *The Journal of Portfolio Management* 30.4 (2004), pp. 110–119 (cit. on p. 119).

[44] Michael Bowling. “Convergence and no-regret in multiagent learning”. In: *Advances in neural information processing systems*. 2005, pp. 209–216 (cit. on p. 51).

[45] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893 (cit. on p. 98).

[46] Andreas Maurer. "Algorithmic stability and meta-learning". In: *Journal of Machine Learning Research* 6.Jun (2005), pp. 967–994 (cit. on pp. 111, 112).

[47] Tom Minka. *Divergence measures and message passing*. Tech. rep. Citeseer, 2005 (cit. on pp. 5, 6, 14, 30, 48).

[48] Juliane Schäfer and Korbinian Strimmer. "A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics". In: *Statistical applications in genetics and molecular biology* 4.1 (2005) (cit. on p. 119).

[49] John Winn and Christopher M Bishop. "Variational message passing." In: *Journal of Machine Learning Research* 6.4 (2005) (cit. on p. 30).

[50] Bruno C Da Silva, Eduardo W Basso, Ana LC Bazzan, and Paulo M Engel. "Dealing with non-stationary environments using context detection". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 217–224 (cit. on p. 49).

[51] Percy Liang, Ben Taskar, and Dan Klein. "Alignment by agreement". In: *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*. Association for Computational Linguistics. 2006, pp. 104–111 (cit. on p. 66).

[52] Tania Lombrozo. "The structure and function of explanations". In: *Trends in cognitive sciences* 10.10 (2006), pp. 464–470 (cit. on p. 82).

[53] Vincent Conitzer and Tuomas Sandholm. "AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents". In: *Machine Learning* 67.1-2 (2007), pp. 23–43 (cit. on p. 51).

[54] Ralf Herbrich, Tom Minka, and Thore Graepel. "TrueSkill™: a Bayesian skill rating system". In: *Advances in neural information processing systems*. 2007, pp. 569–576 (cit. on pp. 62, 137).

[55] Richard S Sutton, Anna Koop, and David Silver. "On the role of tracking in stationary environments". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 871–878 (cit. on pp. 49, 58).

[56] Percy S Liang, Dan Klein, and Michael I Jordan. "Agreement-based learning". In: *Advances in Neural Information Processing Systems*. 2008, pp. 913–920 (cit. on pp. 66, 71).

[57] Martin J Wainwright and Michael I Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008 (cit. on p. 13).

[58] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009 (cit. on p. 65).

[59] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009 (cit. on pp. 2, 5, 13, 22).

[60] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: (2009) (cit. on p. 43).

[61] Sahand Negahban, Bin Yu, Martin J Wainwright, and Pradeep K Ravikumar. "A unified framework for high-dimensional analysis of m-estimators with decomposable regulariz-

ers". In: *Advances in Neural Information Processing Systems*. 2009, pp. 1348–1356 (cit. on p. 148).

[62] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. "Robust stochastic approximation approach to stochastic programming". In: *SIAM Journal on optimization* 19.4 (2009), pp. 1574–1609 (cit. on p. 116).

[63] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. "A theory of learning from different domains". In: *Machine learning* 79.1-2 (2010), pp. 151–175 (cit. on p. 69).

[64] Yilun Chen, Ami Wiesel, Yonina C Eldar, and Alfred O Hero. "Shrinkage algorithms for MMSE covariance estimation". In: *IEEE Transactions on Signal Processing* 58.10 (2010), pp. 5016–5029 (cit. on pp. 41, 120, 123).

[65] Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al. "Posterior regularization for structured latent variable models". In: *Journal of Machine Learning Research* 11.Jul (2010), pp. 2001–2049 (cit. on p. 88).

[66] Chongjie Zhang and Victor R Lesser. "Multi-Agent Learning with Policy Prediction." In: *AAAI*. 2010 (cit. on p. 51).

[67] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. "Natural language processing (almost) from scratch". In: *Journal of machine learning research* 12.ARTICLE (2011), pp. 2493–2537 (cit. on p. 104).

[68] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. "Learning word vectors for sentiment analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 142–150 (cit. on p. 95).

[69] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 142–150. ISBN: 978-1-932432-87-9. URL: http://dl.acm.org/citation.cfm?id=2002472.2002491 (cit. on p. 97).

[70] Max Welling and Yee W Teh. "Bayesian learning via stochastic gradient Langevin dynamics". In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. Citeseer. 2011, pp. 681–688 (cit. on pp. 14, 38).

[71] Chun-Nam J Yu, Russell Greiner, Hsiu-Chin Lin, and Vickie Baracos. "Learning patient-specific cancer survival distributions as a sequence of dependent regressors". In: *Advances in Neural Information Processing Systems*. 2011, pp. 1845–1853 (cit. on pp. 101, 102, 104, 105).

[72] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012 (cit. on p. 145).

[73] George E. Dahl, Ryan P. Adams, and Hugo Larochelle. "Training Restricted Boltzmann Machines on Word Observations". In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress,

2012, pp. 1163–1170. ISBN: 978-1-4503-1285-1. URL: http://dl.acm.org/citation.cfm?id=3042573.3042723 (cit. on p. 97).

[74] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012 (cit. on pp. 18, 35).

[75] Charles Sutton, Andrew McCallum, et al. “An introduction to conditional random fields”. In: *Foundations and Trends® in Machine Learning* 4.4 (2012), pp. 267–373 (cit. on pp. 87, 88).

[76] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on.* IEEE. 2012, pp. 5026–5033 (cit. on p. 56).

[77] Joel A Tropp. “User-friendly tail bounds for sums of random matrices”. In: *Foundations of computational mathematics* 12.4 (2012), pp. 389–434 (cit. on p. 148).

[78] Sida Wang and Christopher D. Manning. “Baselines and Bigrams: Simple, Good Sentiment and Topic Classification”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2.* ACL ’12. Jeju Island, Korea: Association for Computational Linguistics, 2012, pp. 90–94. URL: http://dl.acm.org/citation.cfm?id=2390665.2390688 (cit. on p. 97).

[79] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. “An overview of recent progress in the study of distributed multi-agent coordination”. In: *IEEE Transactions on Industrial informatics* 9.1 (2013), pp. 427–438 (cit. on p. 50).

[80] Richard Mealing and Jonathan L Shapiro. “Opponent Modelling by Sequence Prediction and Lookahead in Two-Player Games”. In: *International Conference on Artificial Intelligence and Soft Computing.* Springer. 2013, pp. 385–396 (cit. on p. 51).

[81] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems.* 2013, pp. 3111–3119 (cit. on p. 2).

[82] Willie Neiswanger, Chong Wang, and Eric Xing. “Asymptotically exact, embarrassingly parallel MCMC”. In: *arXiv preprint arXiv:1311.4780* (2013) (cit. on p. 48).

[83] Art B. Owen. *Monte Carlo theory, methods and examples.* 2013 (cit. on p. 117).

[84] Daniel L Silver, Qiang Yang, and Lianghao Li. “Lifelong Machine Learning Systems: Beyond Learning Algorithms.” In: *AAAI Spring Symposium: Lifelong Machine Learning.* Vol. 13. 2013, p. 05 (cit. on p. 50).

[85] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013) (cit. on p. 81).

[86] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on pp. 65, 68).

[87] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on p. 65).

[88] Quoc Le and Tomas Mikolov. “Distributed representations of sentences and documents”. In: *International Conference on Machine Learning*. 2014, pp. 1188–1196 (cit. on p. 97).

[89] Rajesh Ranganath, Sean Gerrish, and David Blei. “Black box variational inference”. In: *Artificial intelligence and statistics*. PMLR. 2014, pp. 814–822 (cit. on p. 30).

[90] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112 (cit. on pp. 65, 68).

[91] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*. 2014, pp. 3320–3328 (cit. on p. 2).

[92] Rich Caruana et al. “Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 1721–1730 (cit. on p. 81).

[93] François Chollet et al. *Keras*. https://keras.io. 2015 (cit. on p. 149).

[94] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), pp. 503–507 (cit. on p. 56).

[95] Andrew M Dai and Quoc V Le. “Semi-supervised sequence learning”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 3079–3087 (cit. on p. 97).

[96] Eva Gibaja and Sebastián Ventura. “A tutorial on multilabel learning”. In: *ACM Computing Surveys (CSUR)* 47.3 (2015), pp. 1–38 (cit. on p. 46).

[97] Moritz Hardt, Benjamin Recht, and Yoram Singer. “Train faster, generalize better: Stability of stochastic gradient descent”. In: *arXiv preprint arXiv:1509.01240* (2015) (cit. on p. 113).

[98] Rie Johnson and Tong Zhang. “Effective Use of Word Order for Text Categorization with Convolutional Neural Networks”. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2015, pp. 103–112 (cit. on p. 97).

[99] Rie Johnson and Tong Zhang. “Semi-supervised convolutional neural networks for text categorization via region embedding”. In: *Advances in neural information processing systems*. 2015, pp. 919–927 (cit. on p. 97).

[100] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444 (cit. on p. 1).

[101] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. “Multi-task sequence to sequence learning”. In: *arXiv preprint arXiv:1511.06114* (2015) (cit. on p. 65).

[102] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. “Effective approaches to attention-based neural machine translation”. In: *arXiv preprint arXiv:1508.04025* (2015) (cit. on pp. 65, 143).

[103] Yi-An Ma, Tianqi Chen, and Emily Fox. “A complete recipe for stochastic gradient MCMC”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2917–2925 (cit. on p. 38).

[104] Tom M Mitchell, William W Cohen, Estevam R Hruschka Jr, Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, et al. "Never Ending Learning." In: *AAAI*. 2015, pp. 2302–2310 (cit. on p. 50).

[105] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533 (cit. on p. 49).

[106] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 427–436 (cit. on p. 81).

[107] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 1889–1897 (cit. on p. 53).

[108] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015) (cit. on p. 135).

[109] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units". In: *arXiv preprint arXiv:1508.07909* (2015) (cit. on p. 75).

[110] Yu-Xiang Wang, Stephen Fienberg, and Alex Smola. "Privacy for free: Posterior sampling and stochastic gradient monte carlo". In: *International Conference on Machine Learning*. 2015, pp. 2493–2502 (cit. on p. 48).

[111] Sergey Zagoruyko. *92.45% on CIFAR-10 in Torch*. http://torch.ch/blog/2015/07/30/cifar.html. Blog. 2015 (cit. on p. 149).

[112] Martn Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: a system for large-scale machine learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283 (cit. on pp. 76, 149).

[113] Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. "Findings of the 2016 Conference on Machine Translation." In: *ACL 2016 FIRST CONFERENCE ON MACHINE TRANSLATION (WMT16)*. The Association for Computational Linguistics. 2016, pp. 131–198 (cit. on p. 65).

[114] Josep Crego, Jungi Kim, Guillaume Klein, Anabel Rebollo, Kathy Yang, Jean Senellart, Egor Akhanov, Patrice Brunelle, Aurelien Coquard, Yongchao Deng, et al. "SYSTRAN's Pure Neural Machine Translation Systems". In: *arXiv preprint arXiv:1610.05540* (2016) (cit. on p. 65).

[115] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779* (2016) (cit. on p. 58).

[116] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. "Multi-way, multilingual neural machine translation with a shared attention mechanism". In: *arXiv preprint arXiv:1601.01073* (2016) (cit. on p. 65).

[117] Orhan Firat, Baskaran Sankaran, Yaser Al-Onaizan, Fatos T Yarman Vural, and Kyunghyun Cho. “Zero-resource translation with multi-lingual neural machine translation”. In: *arXiv preprint arXiv:1606.04164* (2016) (cit. on p. 66).

[118] Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. “A convolutional encoder model for neural machine translation”. In: *arXiv preprint arXiv:1611.02344* (2016) (cit. on p. 68).

[119] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. “Reinforcement learning with unsupervised auxiliary tasks”. In: *arXiv preprint arXiv:1611.05397* (2016) (cit. on p. 2).

[120] Neal Jean, Marshall Burke, Michael Xie, W Matthew Davis, David B Lobell, and Stefano Ermon. “Combining satellite imagery and machine learning to predict poverty”. In: *Science* 353.6301 (2016), pp. 790–794 (cit. on pp. 82, 94).

[121] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. “Google’s multilingual neural machine translation system: enabling zero-shot translation”. In: *arXiv preprint arXiv:1611.04558* (2016) (cit. on pp. 66, 68, 69, 76, 78, 80).

[122] Rie Johnson and Tong Zhang. “Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings”. In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016, pp. 526–534 (cit. on pp. 95, 97).

[123] Marcin Junczys-Dowmunt, Tomasz Dwojak, and Hieu Hoang. “Is neural machine translation ready for deployment? A case study on 30 translation directions”. In: *arXiv preprint arXiv:1610.01108* (2016) (cit. on p. 65).

[124] Kirthevasan Kandasamy, Maruan Al-Shedivat, and Eric P Xing. “Learning HMMs with nonparametric emissions via spectral decompositions of continuous matrices”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016 (cit. on p. 9).

[125] Yoon Kim and Alexander M Rush. “Sequence-level knowledge distillation”. In: *arXiv preprint arXiv:1606.07947* (2016) (cit. on p. 72).

[126] Tao Lei, Regina Barzilay, and Tommi Jaakkola. “Rationalizing neural predictions”. In: *arXiv preprint arXiv:1606.04155* (2016) (cit. on p. 106).

[127] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research* 17.39 (2016), pp. 1–40 (cit. on p. 49).

[128] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. “Deep reinforcement learning for dialogue generation”. In: *arXiv preprint arXiv:1606.01541* (2016) (cit. on p. 49).

[129] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. “Adversarial training methods for semi-supervised text classification”. In: *arXiv preprint arXiv:1605.07725* (2016) (cit. on p. 97).

[130] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. “Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples”. In: *arXiv preprint arXiv:1602.02697* (2016) (cit. on p. 150).

[131] Sachin Ravi and Hugo Larochelle. “Optimization as a model for few-shot learning”. In: (2016) (cit. on p. 50).

[132] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why Should I Trust You?: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, pp. 1135–1144 (cit. on pp. 81, 83, 86, 94).

[133] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. “Meta-learning with memory-augmented neural networks”. In: *International conference on machine learning*. 2016, pp. 1842–1850 (cit. on p. 50).

[134] Steven L Scott, Alexander W Blocker, Fernando V Bonassi, Hugh A Chipman, Edward I George, and Robert E McCulloch. “Bayes and big data: The consensus Monte Carlo algorithm”. In: *International Journal of Management Science and Engineering Management* 11.2 (2016), pp. 78–88 (cit. on p. 48).

[135] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489 (cit. on p. 49).

[136] StackOverflow. *Stack Overflow Data*. 2016. URL: https://www.kaggle.com/stackoverflow/stackoverflow (cit. on p. 43).

[137] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. “Matching networks for one shot learning”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 3637–3645 (cit. on p. 30).

[138] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In: *arXiv preprint arXiv:1609.08144* (2016) (cit. on pp. 65, 76, 143).

[139] Michal Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. “The United Nations Parallel Corpus v1. 0.” In: *LREC*. 2016 (cit. on pp. 66, 75).

[140] Yun Chen, Yang Liu, Yong Cheng, and Victor OK Li. “A teacher-student framework for zero-resource neural machine translation”. In: *arXiv preprint arXiv:1705.00753* (2017) (cit. on pp. 72, 75, 78, 142, 143).

[141] Yong Cheng, Qian Yang, Yang Liu, Maosong Sun, and Wei Xu. “Joint training for pivot-based neural machine translation”. In: *Proceedings of IJCAI*. 2017 (cit. on pp. 66, 75, 78, 80, 143).

[142] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. “EMNIST: Extending MNIST to handwritten letters”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2921–2926 (cit. on p. 43).

[143] Raj Dabre, Fabien Cromieres, and Sadao Kurohashi. “Kyoto University MT System Description for IWSLT 2017”. In: *Proc. of IWSLT, Tokyo, Japan* (2017) (cit. on pp. 75, 79).

[144] Adji B. Dieng, Chong Wang, Jianfeng Gao, and John William Paisley. “TopicRNN: A Recurrent Neural Network with Long-Range Semantic Dependency”. In: *International Conference on Learning Representations*. 2017 (cit. on p. 97).

[145] Yan Duan, Marcin Andrychowicz, Bradly C Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. “One-shot imitation learning”. In: *arXiv preprint arXiv:1703.07326* (2017) (cit. on p. 2).

[146] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1126–1135 (cit. on pp. 6, 15, 19, 24, 28, 50, 51, 130, 131).

[147] Zhenxin Fu, Xiaoye Tan, Nanyun Peng, Dongyan Zhao, and Rui Yan. “Style transfer in text: Exploration and evaluation”. In: *arXiv preprint arXiv:1711.06861* (2017) (cit. on p. 80).

[148] Scott Gray, Alec Radford, and Diederik P Kingma. “Gpu kernels for block-sparse weights”. In: *arXiv preprint arXiv:1711.09224* 3 (2017) (cit. on pp. 96, 97).

[149] Thanh-Le Ha, Jan Niehues, and Alexander Waibel. “Effective Strategies in Zero-Shot Neural Machine Translation”. In: *arXiv preprint arXiv:1711.07893* (2017) (cit. on p. 66).

[150] Leonard Hasenclever, Stefan Webb, Thibaut Lienart, Sebastian Vollmer, Balaji Lakshminarayanan, Charles Blundell, and Yee Whye Teh. “Distributed Bayesian learning with stochastic natural gradient expectation propagation and the posterior server”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 3744–3780 (cit. on pp. 30, 36, 48).

[151] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* (2017), p. 201611835 (cit. on p. 50).

[152] Philip Koehn. *Europarl: A Parallel Corpus for Statistical Machine Translation.* 2017 (cit. on p. 66).

[153] Philipp Koehn and Rebecca Knowles. “Six challenges for neural machine translation”. In: *arXiv preprint arXiv:1706.03872* (2017) (cit. on p. 65).

[154] Liping Liu, Francisco Ruiz, and David Blei. “Context Selection for Embedding Models”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4817–4826 (cit. on p. 106).

[155] David Lopez-Paz and Marc’Aurelio Ranzato. “Gradient episodic memory for continual learning”. In: *arXiv preprint arXiv:1706.08840* (2017) (cit. on p. 20).

[156] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *arXiv preprint arXiv:1706.02275* (2017) (cit. on p. 50).

[157] Scott Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *arXiv preprint arXiv:1705.07874* (2017) (cit. on p. 83).

[158] Stephan Mandt, Matthew D Hoffman, and David M Blei. “Stochastic gradient descent as approximate bayesian inference”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 4873–4907 (cit. on pp. 38, 43, 115, 117).

[159] Cettolo Mauro, Federico Marcello, Bentivogli Luisa, Niehues Jan, Stüker Sebastian, Sudoh Katsuitho, Yoshino Koichiro, and Federmann Christian. “Overview of the IWSLT 2017 Evaluation Campaign”. In: *International Workshop on Spoken Language Translation.* 2017, pp. 2–14 (cit. on p. 66).

[160] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282 (cit. on pp. 6, 32, 112).

[161] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. “Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games”. In: *arXiv preprint arXiv:1703.10069* (2017) (cit. on p. 50).

[162] Sebastian Ruder. “An overview of multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1706.05098* (2017) (cit. on pp. 1, 19).

[163] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal Policy Optimization Algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on pp. 53, 59, 135).

[164] Maruan Al-Shedivat, Avinava Dubey, and Eric P Xing. “Personalized survival prediction with contextual explanation networks”. In: *Machine Learning for Healthcare Workshop (NeurIPS)*. 2017 (cit. on p. 9).

[165] Maruan Al-Shedivat, Avinava Dubey, and Eric P Xing. “The intriguing properties of model explanations”. In: *Symposium on Interpretable Machine Learning (NeurIPS)*. 2017 (cit. on p. 9).

[166] Maruan Al-Shedivat, Andrew Gordon Wilson, Yunus Saatchi, Zhiting Hu, and Eric P Xing. “Learning scalable deep kernels with recurrent structure”. In: *The Journal of Machine Learning Research (JMLR)* (2017) (cit. on p. 9).

[167] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. “Style transfer from non-parallel text by cross-alignment”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6830–6841 (cit. on p. 80).

[168] Jake Snell, Kevin Swersky, and Richard Zemel. “Prototypical networks for few-shot learning”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 4080–4090 (cit. on pp. 15, 24, 28).

[169] Ion Stoica, Dawn Song, Raluca Ada Popa, David Patterson, Michael W Mahoney, Randy Katz, Anthony D Joseph, Michael Jordan, Joseph M Hellerstein, Joseph E Gonzalez, et al. “A berkeley view of systems challenges for ai”. In: *arXiv preprint arXiv:1712.05855* (2017) (cit. on p. 1).

[170] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008 (cit. on p. 68).

[171] David Alvarez-Melis and Tommi S Jaakkola. “Towards robust interpretability with self-explaining neural networks”. In: *arXiv preprint arXiv:1806.07538* (2018) (cit. on p. 106).

[172] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. “Emergent Complexity via Multi-Agent Competition”. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=Sy0GnUxCb (cit. on pp. 59, 135).

[173] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecny, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. “Leaf: A benchmark for federated settings”. In: *arXiv preprint arXiv:1812.01097* (2018) (cit. on pp. 43, 123).

[174] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. “Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 794–803 (cit. on p. 2).

[175] Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R Bowman, Holger Schwenk, and Veselin Stoyanov. “Xnli: Evaluating cross-lingual sentence representations”. In: *arXiv preprint arXiv:1809.05053* (2018) (cit. on p. 80).

[176] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on p. 2).

[177] Chelsea Finn, Kelvin Xu, and Sergey Levine. “Probabilistic model-agnostic meta-learning”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 9537–9548 (cit. on pp. 24, 64).

[178] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. “Learning with Opponent-Learning Awareness”. In: *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018 (cit. on pp. 9, 51).

[179] Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. “Dice: The infinitely differentiable monte carlo estimator”. In: *International Conference on Machine Learning (ICML)*. 2018 (cit. on p. 9).

[180] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. “Conditional neural processes”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1704–1713 (cit. on pp. 15, 28).

[181] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. “Recasting gradient-based meta-learning as hierarchical bayes”. In: *arXiv preprint arXiv:1801.08930* (2018) (cit. on pp. 24, 64).

[182] Aditya Grover, Maruan Al-Shedivat, Jayesh Gupta, Yuri Burda, and Harrison Edwards. “Learning policy representations in multiagent systems”. In: *International Conference on Machine Learning (ICML)*. PMLR. 2018 (cit. on p. 9).

[183] Aditya Grover, Maruan Al-Shedivat, Jayesh K Gupta, Yuri Burda, and Harrison Edwards. “Evaluating generalization in multiagent systems using agent-interaction graphs”. In: *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. 2018 (cit. on p. 9).

[184] Jeremy Howard and Sebastian Ruder. “Universal language model fine-tuning for text classification”. In: *arXiv preprint arXiv:1801.06146* (2018) (cit. on pp. 96, 97).

[185] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. “Federated optimization in heterogeneous networks”. In: *arXiv preprint arXiv:1812.06127* (2018) (cit. on p. 33).

[186] Yingzhen Li. “Approximate inference: New visions”. PhD thesis. University of Cambridge, 2018 (cit. on p. 14).

[187] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bishan Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matt Gardner, Bryan Kisiel, et al. “Never-ending learning”. In: *Communications of the ACM* 61.5 (2018), pp. 103–115 (cit. on pp. 4, 17, 19, 24).

[188] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. “Learning to adapt in dynamic, real-world environments

through meta-reinforcement learning". In: *arXiv preprint arXiv:1803.11347* (2018) (cit. on p. 64).

[189] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. "Deep online learning via meta-learning: Continual adaptation for model-based rl". In: *arXiv preprint arXiv:1812.07671* (2018) (cit. on p. 64).

[190] Alex Nichol, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms". In: *arXiv preprint arXiv:1803.02999* (2018) (cit. on pp. 47, 112).

[191] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep contextualized word representations". In: *arXiv preprint arXiv:1802.05365* (2018) (cit. on p. 2).

[192] Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. "Contextual Parameter Generation for Universal Neural Machine Translation". In: *arXiv preprint arXiv:1808.08493* (2018) (cit. on pp. 66, 79, 106).

[193] Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. "Style transfer through back-translation". In: *arXiv preprint arXiv:1804.09000* (2018) (cit. on p. 80).

[194] Sebastian Ruder. "NLP's ImageNet moment has arrived". In: *The Gradient* (2018) (cit. on p. 2).

[195] Lierni Sestorain, Massimiliano Ciaramita, Christian Buck, and Thomas Hofmann. "Zero-Shot Dual Machine Translation". In: *arXiv preprint arXiv:1805.10338* (2018) (cit. on pp. 66, 75–77, 80, 143).

[196] Noam Shazeer and Mitchell Stern. "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost". In: *arXiv preprint arXiv:1804.04235* (2018) (cit. on pp. 76, 143).

[197] Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. "Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments". In: *International Conference on Learning Representations (ICLR)*. 2018 (cit. on pp. 8, 24, 64).

[198] Maruan Al-Shedivat, Lisa Lee, Ruslan Salakhutdinov, and Eric Xing. "On the complexity of exploration in goal-driven navigation". In: *arXiv preprint arXiv:1811.06889* (2018) (cit. on p. 9).

[199] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. "Tensor2tensor for neural machine translation". In: *arXiv preprint arXiv:1803.07416* (2018) (cit. on pp. 76, 143).

[200] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. "Bayesian model-agnostic meta-learning". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 7343–7353 (cit. on pp. 24, 64).

[201] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. "Taskonomy: Disentangling task transfer learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3712–3722 (cit. on pp. 2, 19, 27).

[202] Jacob Andreas. “Good-enough compositional data augmentation”. In: *arXiv preprint arXiv:1904.09545* (2019) (cit. on p. 3).

[203] Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Roee Aharoni, Melvin Johnson, and Wolfgang Macherey. “The missing ingredient in zero-shot neural machine translation”. In: *arXiv preprint arXiv:1903.07091* (2019) (cit. on p. 80).

[204] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. “Towards federated learning at scale: System design”. In: *arXiv preprint arXiv:1902.01046* (2019) (cit. on p. 32).

[205] Ann-Kathrin Dombrowski, Maximillian Alber, Christopher Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. “Explanations can be manipulated and geometry is to blame”. In: *Advances in Neural Information Processing Systems.* 2019, pp. 13567–13578 (cit. on p. 101).

[206] Lu Haonan, Seth H Huang, Tian Ye, and Guo Xiuyan. “Graph Star Net for Generalized Multi-Task Learning”. In: *arXiv preprint arXiv:1906.12330* (2019) (cit. on p. 97).

[207] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B Gibbons. “The non-IID data quagmire of decentralized machine learning”. In: *arXiv preprint arXiv:1910.00189* (2019) (cit. on p. 123).

[208] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. “Advances and open problems in federated learning”. In: *arXiv preprint arXiv:1912.04977* (2019) (cit. on pp. 5, 32, 34).

[209] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. “Scaffold: Stochastic controlled averaging for on-device federated learning”. In: *arXiv preprint arXiv:1910.06378* (2019) (cit. on p. 33).

[210] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. “Big transfer (BiT): General visual representation learning”. In: *arXiv preprint arXiv:1912.11370* 6 (2019) (cit. on p. 2).

[211] Himabindu Lakkaraju and Osbert Bastani. “”How do I fool you?”: Manipulating User Trust via Misleading Black Box Explanations”. In: *arXiv preprint arXiv:1911.06473* (2019) (cit. on p. 101).

[212] Jingkai Mao, Jakob Foerster, Tim Rocktäschel, Maruan Al-Shedivat, Gregory Farquhar, and Shimon Whiteson. “A baseline for any order gradient estimation in stochastic computation graphs”. In: *International Conference on Machine Learning (ICML).* 2019 (cit. on p. 9).

[213] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9 (cit. on p. 2).

[214] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. “On the convergence of adam and beyond”. In: *arXiv preprint arXiv:1904.09237* (2019) (cit. on p. 148).

[215] Devendra Singh Sachan, Manzil Zaheer, and Ruslan Salakhutdinov. “Revisiting LSTM networks for semi-supervised text classification via mixed objective function”. In: *Pro-*

ceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 2019, pp. 6940–6948 (cit. on p. 97).

[216] Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. "Ray interference: a source of plateaus in deep reinforcement learning". In: *arXiv preprint arXiv:1904.11455* (2019) (cit. on p. 20).

[217] Maruan Al-Shedivat and Ankur Parikh. "Consistency by Agreement in Zero-Shot Neural Machine Translation". In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2019 (cit. on pp. 8, 27, 80).

[218] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. "Unsupervised data augmentation". In: *arXiv preprint arXiv:1904.12848* (2019) (cit. on pp. 96, 97).

[219] Ahmad Ajalloeian and Sebastian U Stich. "Analysis of SGD with biased gradient estimators". In: *arXiv preprint arXiv:2008.00051* (2020) (cit. on pp. 40, 115, 116).

[220] Yash Chandak, Georgios Theocharous, Shiv Shankar, Martha White, Sridhar Mahadevan, and Philip Thomas. "Optimizing for the future in non-stationary mdps". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1414–1425 (cit. on p. 64).

[221] Zachary Charles and Jakub Konecny. "On the outsized importance of learning rates in local update methods". In: *arXiv preprint arXiv:2007.00878* (2020) (cit. on p. 33).

[222] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. "Just Pick a Sign: Optimizing Deep Multitask Models with Gradient Sign Dropout". In: *Advances in Neural Information Processing Systems* 33 (2020) (cit. on p. 2).

[223] Xavier Garcia, Pierre Foret, Thibault Sellam, and Ankur P Parikh. "A multilingual view of unsupervised machine translation". In: *arXiv preprint arXiv:2002.02955* (2020) (cit. on p. 80).

[224] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. "Meta-learning in neural networks: A survey". In: *arXiv preprint arXiv:2004.05439* (2020) (cit. on p. 2).

[225] Sai Praneeth Karimireddy, Martin Jaggi, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. "Mime: Mimicking Centralized Stochastic Algorithms in Federated Learning". In: *arXiv preprint arXiv:2008.03606* (2020) (cit. on pp. 43, 45, 46).

[226] Dong-Ki Kim, Miao Liu, Matthew Riemer, Chuangchuang Sun, Marwa Abdulhai, Golnaz Habibi, Sebastian Lopez-Cot, Gerald Tesauro, and Jonathan P How. "A Policy Gradient Algorithm for Learning to Learn in Multiagent Reinforcement Learning". In: *arXiv preprint arXiv:2011.00382* (2020) (cit. on p. 64).

[227] Benjamin J Lengerich, Maruan Al-Shedivat, Amir Alavi, Jennifer Williams, Sami Labbaki, and Eric P Xing. "Discriminative Subtyping of Lung Cancers from Histopathology Images via Contextual Deep Learning". In: *medRxiv* (2020) (cit. on pp. 9, 106).

[228] Zuchao Li, Hai Zhao, Rui Wang, Masao Utiyama, and Eiichiro Sumita. "Reference language based unsupervised neural machine translation". In: *arXiv preprint arXiv:2004.02127* (2020) (cit. on p. 80).

[229] Reese Pathak and Martin J Wainwright. "FedSplit: An algorithmic framework for fast federated optimization". In: *arXiv preprint arXiv:2005.05238* (2020) (cit. on p. 33).

[230] Emmanouil Antonios Platanios, Maruan Al-Shedivat, Eric Xing, and Tom Mitchell. “Learning from Imperfect Annotations”. In: *arXiv preprint arXiv:2004.03473* (2020) (cit. on p. 9).

[231] Gregory Plumb, Maruan Al-Shedivat, Ángel Alexander Cabrera, Adam Perer, Eric Xing, and Ameet Talwalkar. “Regularizing black-box models for improved interpretability”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2020) (cit. on p. 9).

[232] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konecny, Sanjiv Kumar, and H Brendan McMahan. “Adaptive Federated Optimization”. In: *arXiv preprint arXiv:2003.00295* (2020) (cit. on pp. 34, 35, 43–46, 123, 124).

[233] Maruan Al-Shedivat, Avinava Dubey, and Eric Xing. “Contextual explanation networks”. In: *The Journal of Machine Learning Research (JMLR)* (2020) (cit. on p. 8).

[234] George Stoica, Otilia Stretcu, Emmanouil Antonios Platanios, Tom Mitchell, and Barnabás Póczos. “Contextual Parameter Generation for Knowledge Graph Link Prediction”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 03. 2020, pp. 3000–3008 (cit. on p. 106).

[235] Aki Vehtari, Andrew Gelman, Tuomas Sivula, Pasi Jylänki, Dustin Tran, Swupnil Sahai, Paul Blomstedt, John P Cunningham, David Schiminovich, and Christian P Robert. “Expectation Propagation as a Way of Life: A Framework for Bayesian Inference on Partitioned Data.” In: *Journal of Machine Learning Research* 21.17 (2020), pp. 1–53 (cit. on pp. 30, 48).

[236] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. “Salvaging federated learning by local adaptation”. In: *arXiv preprint arXiv:2002.04758* (2020) (cit. on p. 33).

[237] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning”. In: *Conference on Robot Learning*. 2020, pp. 1094–1100 (cit. on p. 19).

[238] Amir R Zamir, Alexander Sax, Nikhil Cheerla, Rohan Suri, Zhangjie Cao, Jitendra Malik, and Leonidas J Guibas. “Robust learning through cross-task consistency”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11197–11206 (cit. on p. 27).

[239] Xinwei Zhang, Mingyi Hong, Sairaj Dhople, Wotao Yin, and Yang Liu. “FedPD: A federated learning framework with optimal rates and adaptivity to non-IID data”. In: *arXiv preprint arXiv:2005.11418* (2020) (cit. on p. 33).

[240] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N Whatmough, and Venkatesh Saligrama. “Federated learning based on dynamic regularization”. In: *International Conference on Learning Representations*. 2021 (cit. on p. 33).

[241] Aniruddh Raghu, John Guttag, Katherine Young, Eugene Pomerantsev, Adrian V Dalca, and Collin M Stultz. “Learning to predict with supporting evidence: applications to clinical risk prediction”. In: *Proceedings of the Conference on Health, Inference, and Learning*. 2021, pp. 95–104 (cit. on p. 106).

[242] Maruan Al-Shedivat, Jennifer Gillenwater, Eric Xing, and Afshin Rostamizadeh. “Federated Learning via Posterior Averaging: A New Perspective and Practical Algorithms”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021 (cit. on pp. 8, 48).

[243] Maruan Al-Shedivat, Liam Li, Eric Xing, and Ameet Talwalkar. “On Data Efficiency of Meta-learning”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2021 (cit. on pp. 8, 30).

[244] Bowen Tan, Zichao Yang, Maruan Al-Shedivat, Eric P Xing, and Zhiting Hu. “Progressive Generation of Long Text with Pretrained Language Models”. In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2021 (cit. on p. 9).

[245] Jianyu Wang, Zach Charles, Gauri Joshi, Brendan McMahan, Zheng Xu, Blaise Aguera y Arcas, Maruan Al-Shedivat, et al. “Field Guide to Federated Optimization”. In: (2021) (cit. on pp. 9, 35).