ABSTRACT

Title of dissertation: LEARNING WITH MINIMAL SUPERVISION: NEW META-LEARNING AND REINFORCEMENT LEARNING ALGORITHMS

Amr Mohamed Nabil Aly Aly Sharaf
Doctor of Philosophy, 2020

Dissertation directed by: Professor Hal Daumé III
Department of Computer Science
University of Maryland

Standard machine learning approaches thrive on learning from huge amounts of labeled training data, but what if we don't have access to large amounts of labeled datasets? Humans have a remarkable ability to learn from only a few examples. To do so, they either build upon their prior learning experiences, or adapt to new circumstances by observing sparse learning signals. In this dissertation, we promote algorithms that learn with minimal amounts of supervision inspired by these two ideas. We discuss two families for minimally supervised learning algorithms based on meta-learning (or learning to learn) and reinforcement learning approaches.

In the first part of the dissertation, we discuss meta-learning approaches for learning with minimal supervision. We present three meta-learning algorithms for few-shot adaptation of neural machine translation systems, promoting fairness in learned models by learning to actively learn under fairness parity constraints, and learning better exploration policies in the interactive contextual bandit setting. All of these algorithms simulate

settings in which the agent has access to only a few labeled samples. Based on these simulations, the agent learns how to solve future learning tasks with minimal supervision.

In the second part of the dissertation, we present learning algorithms based on reinforcement and imitation learning. In many settings the learning agent doesn't have access to fully supervised training data, however, it might be able to leverage access to a sparse reward signal, or an expert that can be queried to collect the labeled data. It is important then to utilize these learning signals efficiently. Towards achieving this goal, we present three learning algorithms for learning from very sparse reward signals, leveraging access to noisy guidance, and solving structured prediction learning tasks under bandit feedback. In all cases, the result is a minimally supervised learning algorithm that can effectively learn given access to sparse reward signals.

Learning with Minimal Supervision: New Meta-Learning and Reinforcement Learning Algorithms

by

Amr Mohamed Nabil Aly Aly Sharaf

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Advisory Committee:
Professor Hal Daumé III, Chair/Advisor
Professor Jordan Boyd-Graber
Professor Philip Resnik
Professor Soheil Feizi
Professor Tom Goldstein
Professor Yisong Yue

Dedication

This thesis is dedicated to my family, who went above and beyond the call of duty in their everlasting support. To my mom, my dad, my brother, and most importantly my niece and nephew Farida and Mohamed.

Acknowledgments

First and foremost, I am tremendously grateful for my advisor Hal Daumé III for his continuous support, guidance, and patience throughout my PhD. I want to thank Hal for providing me with the freedom to follow my passion and work on the variety of different research directions I am interested in. Hal introduced me to the wonderful world of reinforcement learning and structured prediction during my graduate courses, and inspired me to pursue further research in these areas. Hal taught me all aspects of conducting great research: formulating research questions, designing experiments, analyzing results, and writing and editing research papers. I also enjoyed teaching the computational linguistics course with Hal, as well as attending his machine learning and reinforcement learning courses during my first year in the program. I am deeply indebted to Hal for his invaluable and inspirational insight, as well as endless kindness and encouragement.

I would like to thank members of my dissertation committee: Jordan Boyd-Graber, Philip Resnik, Soheil Feizi, Tom Goldstein, and Yisong Yue, many of whom also served in my oral candidacy examination, for supervising this work and providing feedback and guidance for its research directions. Tom Goldstein deserves a special mention for his continuous guidance and support. Tom welcomed me to his research group and helped me tremendously during my job search. I am grateful for Tom for teaching me everything I know about optimization algorithms and techniques. I also enjoyed working with Tom as a teaching assistant for the discrete structures course. I am also thankful for Philip Resnik for serving as the dean's representative, and for Yisong Yue for agreeing to be an external member on my committee.

During my PhD, I was fortunate to have the opportunity to join Microsoft Research and Amazon as a research intern. I am grateful to my internship mentors: Hany Hassan (Microsoft Research), Asli Celikyilmaz (Microsoft Research), Lambert Mathias (Amazon), and Markus Dreyer (Amazon). I am especially grateful to Hany who was supportive and patient while I built my background in Machine Translation. Hany was also extremely

supportive during my job search. The Department of Computer Science at UMD truly offers a world-class research and learning environment. I would like to thank faculty members of the CLIP lab: Marine Carpuat, Doug Oard, Louiqa Raschid, and Naomi Feldman. I also enjoyed taking graduate courses with Rama Chellappa, James Reggia, Aravind Srinivasan, and David Jacobs. I would like to thank the teaching instructors I worked with: Clyde Kruskal, Tom Reinhardt, and Nelson Padua-Perez. I am also grateful to my Master's advisors from Alexandria University: Mohamed Hussein and Mohamed Ismail.

Of course, I could not have completed this journey without the other students in UMD and the CLIP lab. I am grateful to Moustafa Meshry, Kianté Brantley, Khanh Nguyen, Shi Feng, Chen Zhao, Trista Cao, Sudha Rao, Mohit Iyyer, Anna Sotnikova, Yancy Liao, Joe Barrow, Yogarshi Vyas, Ahmed Elgohary, Ahmed Abdelkader, Mahmoud Sayed, Micah Goldblum, Aya Abdelsalam, Candice Schumann, Emily Gong, Yuan su, Suraj Nair, and Pedro Rodriguez. I would also like to thank Tom Hurst, Jennifer Story, Jodie Gray, Arlene Schenk, and Janice Perrone at UMD for making all the administrative tasks easy. I am especially grateful to Moustafa Meshry and Kianté Brantley for their continuous encouragement. I am also extremely grateful to Lauren Williams for her guidance and support.

I am extremely grateful to my family and friends who supported me every step of the way during this journey. I am grateful to my parents and brother for their tremendous support. I would also like to thank my family and my dearest little nephew and niece for helping me keep my calm through their cuteness. I am thankful to my cousins Mohamed, Omar, and Tamer Abouelseoud. Special thanks for Mohamed and Kareem Abouelsoud for the weekend hangouts in Washington D.C. that minimized the stress of graduate studies. As always, I am grateful to all of my childhood and college friends, who have helped me keep my sanity during the grind of graduate school: Ahmed Kotb, Ahmed Gamal, Mohamed El-Gharably, Mohamed Gaber, Ahmed Emara, Ahmed Elshaarany, Esin Durmus, Karim

Zaytoun, Haitham Khedr, Mohamed Dyab, Ahmed Mohsen, and Ahmed Eshra. I want to give a shout-out to Ahmed Kotb who was extremely supportive and made the time I spent in Seattle during my summer internships much more enjoyable. I am really grateful to my childhood friend Ahmed Gamal for his support, I will always remember the spring breaks we spent together in Florida.

I am very fortunate to have spent five lovely years at UMD. Throughout my journey in graduate school, many people have helped me mature in both academics and life. This dissertation would not have been possible without their support. I dedicate this dissertation for my parents and my family.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

Standard machine learning approaches thrive on learning from huge amounts of labeled training data, but what if we don't have access to large amounts of labeled datasets? Humans have a remarkable ability to learn from only a few examples. To do so, they either build upon their prior learning experiences, or adapt to new circumstances by observing sparse learning signals. In this dissertation, we promote algorithms that learn with minimal amounts of supervision inspired by these two ideas. We discuss two families for minimally supervised learning algorithms based on meta-learning (or learning to learn) and reinforcement learning approaches.

In the first part of the dissertation, we present meta-learning approaches for learning with minimal supervision. We present three meta-learning algorithms for few-shot adaptation of neural machine translation systems (§1.3.1), promoting fairness in learned models by learning to actively learn under fairness parity constraints (§1.3.2), and learning better exploration policies in the interactive contextual bandit setting (§1.3.3). All of these algorithms simulate settings in which the learner has access to only a few labeled samples. Based on these simulations, the agent learns how to solve future learning tasks given only few labeled examples. As a result, these algorithms provide a method to promote the learning of fair and adaptive models given a minimal amount of supervision.

In the second part of the dissertation, we study learning algorithms based on reinforcement and imitation learning. In many settings the learning agent doesn't have access

to fully supervised training data, however, it might be able to leverage access to a sparse reward signal, or an expert that can be queried to collect the labeled data. It is important then to be able to utilize these learning signals efficiently. Towards achieving this goal, we present three learning algorithms for learning from very sparse reward signals (§1.4.1), leveraging access to noisy guidance (§1.4.2), and solving structured prediction learning tasks under bandit feedback (§1.4.3). In all cases, the result is a minimally supervised learning algorithm that can effectively learn given access to sparse reward signals.

1.2 Learning with Minimal Supervision

Several approaches have been proposed to tackle the problem of learning with minimal supervision. In this dissertation, we study three of these approaches: optimization based meta-learning (Part I), reinforcement learning, and imitation learning (Part II).

Table 1.1 presents some of the most common approaches for learning with minimal supervision, as well as the most important pros and cons for each approach. This dissertation introduces new algorithms from the first three approaches in the table. In meta-learning (Part I), the agent assumes access to related learning tasks on which it can run simulations for what the agent will observe at test time. From these simulations, the agent can leverage its previous learning experience to solve future learning task more efficiently at test time. The advantage for this approach is that the agent learns to optimize for the test time behavior based on the training simulations, i.e. we don't have a mismatch between training and testing objectives. The disadvantage however is that this form of learning necessitates the availability of similar tasks to learn from at training time.

In reinforcement learning (Part II), the agent learns directly by observing reward signals from the environment. However, this comes at the cost of increasing the sample complexity required for learning. Imitation learning is more sample efficient than reinforcement learning, however, it requires access to an expert strategy to imitate at training time. Recently (Brown et al., 2020; Devlin et al., 2018, 2019), unsupervised pre-training

approaches have proven to be able to learn with minimal amounts of supervision in few-shot learning settings. However, the training objective for these approaches are usually generic. The transferability of such approaches to new learning tasks have recently been investigated in Vu et al. (2020). These approaches are less useful for the applications we consider in this dissertation, where there is a mismatch between the unsupervised training objective and the targeted learning task. For instance, it is unclear how to leverage pre-trained language models to efficiently address the task of few-shot domain adaptation for translation models (chapter 3).

Approach	**Advantages**	**Disadvantages**
Meta-Learning	Training objective matches testing behavior	Requires access to related training tasks
Reinforcement Learning	Learning directly from reward signals	Higher sample complexity
Imitation Learning	Lower sample complexity than Reinforcement Learning	Requires access to expert simulator at training time
Unsupervised Pre-Training	Doesn't require labeled training data	Training objective may not generalize to testing behavior

Table 1.1: Some of the most common approaches for learning with minimal supervision.

1.3 Part I: Meta-Learning Algorithms

Our goal in this dissertation is to promote algorithms that learn with minimal supervision. In Part I of the dissertation, we focus on learning algorithms based on meta-learning. Conventional learning algorithms expose the agent to a single learning task, in contrast, meta-learning approaches expose the agent to multiple learning tasks at training time. The goal is to leverage these learning tasks to build experiences that enable the agent to learn more efficiently in future learning scenarios with minimal supervision.

Table 1.2 shows the outline for this part of the dissertation. We study three different forms of learning with minimal supervision: few-shot learning (chapter 3), active learning

(chapter 4), and finally contextual bandit learning (chapter 5). We investigate and present learning algorithms for answering the following research questions:

1. How can we use meta-learning to adapt a Neural Machine Translation (NMT) model to new domains with very few in-domain data? (§1.3.1)

2. Can we promote the learning of fair models via meta-learning? (§1.3.2)

3. Can an agent learn better exploration strategies in a "Contextual Bandits" setting via meta-learning? (§1.3.3)

We discuss each of these three forms of minimal supervision, and provide an overview for each of the three algorithms individually below.

Minimal Supervision Setting	**Application**	**Algorithm**	**Chapter**
Few-Shot Learning	Domain Adaptation for NMT Systems	META-MT	chapter 3
Active Learning	Promoting Fairness in Learned Models	PANDA	chapter 4
Contextual Bandit Learning	Better Exploration Strategies	MÊLÉE	chapter 5

Table 1.2: Outline of Part I: Meta-Learning Approaches

1.3.1 Meta-Learning for Few-Shot NMT Adaptation

In chapter 3, we study the "few-shot" learning setting as a form of learning with minimal supervision. In few-shot learning, the agent is presented with only a handful of training examples. The number of these training examples per label is known as the "shot". We present META-MT, a meta-learning approach for adapting Neural Machine Translation (NMT) systems in a few-shot setting. META-MT provides a new approach to adapt NMT models to target domains with the minimal amount of in-domain data. We frame the adaptation of NMT systems as a meta-learning problem, where we learn to adapt to new unseen domains based on simulated offline meta-training domain adaptation tasks. We evaluate the proposed meta-learning strategy on ten domains with general large scale NMT systems. We show that META-MT significantly outperforms classical domain

adaptation when very few in-domain examples are available. Our experiments shows that META-MT can outperform classical fine-tuning by up to 2.5 BLEU points after seeing only $4,000$ translated words (300 sentences), even in a zero-shot learning setting.

1.3.2 Learning to Active Learn under Parity Constraints

In chapter 4, we study the "Active Learning" setting as a form of learning with minimal supervision, where the learning agent has control over which samples to query for labels. Machine learning models can have consequential effects, and disparities in error rate can lead to harms suffered more by some groups than others. Past algorithmic approaches mitigate such disparities for fixed training data; we ask: what if we can gather more data? We develop a meta-learning algorithm for parity-constrained active learning that learns a policy to decide which labels to query so as to maximize accuracy subject to parity constraints, using forward-backward splitting at the meta-learning level. Empirically, across three classification tasks and different parity metrics, our approach outperforms alternatives by a large margin.

1.3.3 Meta-Learning for Contextual Bandit Exploration

In chapter 5, we study the "Contextual Bandit" setting as a form of learning with minimal supervision. We describe MÊLÉE, a meta-learning algorithm for learning an exploration policy in the contextual bandit setting. Here, an algorithm must take actions based on contexts, and learn based only on a reward signal from the action taken, thereby generating an exploration/exploitation trade-off. MÊLÉE addresses this trade-off by learning a good exploration strategy for offline tasks based on synthetic data, on which it can simulate the contextual bandit setting. Based on these simulations, MÊLÉE uses an imitation learning strategy to learn a good exploration policy that can then be applied to true contextual bandit tasks at test time. We compare MÊLÉE to seven strong baseline contextual bandit algorithms on a set of three hundred real-world datasets, on which it

outperforms alternatives in most settings, especially when differences in rewards are large. Finally, we demonstrate the importance of having a rich feature representation for learning how to explore.

1.4 Part II: Reinforcement-Learning Algorithms

In Part II of this dissertation we discuss minimally supervised learning algorithms based on reinforcement learning. Part I studies a setting in which the agent has access to fully supervised datasets on which it can run simulations at training time, but what if we don't have access to such datasets? This is the setting we study in Part II, instead of observing a fully labeled dataset to learn from, the agent learns by observing a "reward signal".

Table 1.3 shows the outline for this part of the dissertation. We study three different forms of learning with minimal supervision: Reinforcement Learning Reward Signals (chapter 6), Active Imitation Learning (chapter 7), and Structured Contextual Bandits (chapter 8). We observe these different forms of learning supervision and design algorithms to solve a wide set of different structured prediction and gaming applications. We investigate the following research questions:

1. How can we solve reinforcement learning problems with very sparse reward signals observed only at the end of an episode? (§1.4.1)

2. Can we leverage access to a noisy heuristic that provides noisy guidance to minimize the annotation cost in an imitation learning setting? (§1.4.2)

3. Can we solve structured prediction problems given only access to partial feedback? (§1.4.3)

We provide an overview for the three algorithms addressing these research questions individually below.

Minimal Supervision Setting	Application	Algorithm	Chapter
Reinforcement Learning Reward Signal	Games, Dependency Parsing	RESLOPE	chapter 6
Active Imitation Learning	Keyphrase Extraction, POS Tagging	LEAQI	chapter 7
Structured Contextual Bandits	Structured Prediction	BLS	chapter 8

Table 1.3: Outline of Part II: Reinforcement Learning Approaches

1.4.1 Reinforcement Learning With No Incremental Feedback

In chapter 6 we consider reinforcement learning and bandit structured prediction problems with very sparse loss feedback - only at the end of an episode - as a form of learning with minimal supervision . We introduce a novel algorithm, RESIDUAL LOSS PREDICTION (RESLOPE), that solves such problems by automatically learning an internal representation of a denser reward function. RESLOPE operates as a reduction to contextual bandits, using its learned loss representation to solve the credit assignment problem, and a contextual bandit oracle to trade-off exploration and exploitation. RESLOPE enjoys a no-regret reduction-style theoretical guarantee and outperforms state of the art reinforcement learning algorithms in MDP environments and bandit structured prediction settings.

1.4.2 Active Imitation Learning with Noisy Guidance

In chapter 7 we consider active imitation learning as a form of learning with minimal supervision. Imitation learning algorithms provide state-of-the-art results on many structured prediction tasks by learning near-optimal search policies. Such algorithms assume training-time access to an expert that can provide the optimal action at any queried state; unfortunately, the number of such queries is often prohibitive, frequently rendering these approaches impractical. To combat this query complexity, we consider an active learning setting in which the learning algorithm has additional access to a much cheaper *noisy heuristic* that provides noisy guidance. Our algorithm, LEAQI, learns a *difference classifier* that predicts when the expert is likely to disagree with the heuristic, and queries the expert only when necessary. We apply LEAQI to three sequence labeling tasks, demonstrating

significantly fewer queries to the expert and comparable (or better) accuracies over a passive approach.

1.4.3 Structured Prediction Under Bandit Feedback

In chapter 8 we demonstrate the importance of learning from different feedback signals for a bandit Structured prediction task. We present an algorithm for structured prediction under online bandit feedback. The learner repeatedly predicts a sequence of actions, generating a structured output. It then observes feedback for that output and no others. We consider two cases: a pure bandit setting in which it only observes a loss, and more fine-grained feedback in which it observes a loss for every action. We find that the fine-grained feedback is necessary for strong empirical performance, because it allows for a robust variance-reduction strategy. We empirically compare a number of different algorithms and exploration methods and show the efficacy of our approach (BLS) on sequence labeling and dependency parsing tasks.

1.5 Thesis Statement and Contributions

We now make the main statement of this thesis:

Meta-Learning and reinforcement learning algorithms provide a useful class of algorithms for learning fair, adaptive, and robust models with minimal supervision.

We validate this claim by providing the following contributions:

1. Amr Sharaf, Hany Hassan, and Hal Daumé III. Meta-learning for few-shot NMT adaptation. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*. Association for Computational Linguistics, 2020.

2. Amr Sharaf and Hal Daumé III. Promoting fairness in learned models by learning to active learn under parity constraints. In *Workshop on Real World Experiment Design and Active Learning*. International Conference on Machine Learning, 2020.

3. Amr Sharaf and Hal Daumé III. Meta-learning contextual bandit exploration. In *Workshop on Meta-Learning*. Advances in Neural Information Processing Systems (NeurIPS), 2019.

4. Hal Daumé III, John Langford, and Amr Sharaf. Residual loss prediction: Reinforcement learning with no incremental feedback. In *International Conference on Learning Representations (ICLR)*, 2018[1].

5. Kianté Brantley, Amr Sharaf, and Hal Daumé III. Active imitation learning with noisy guidance. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

6. Amr Sharaf and Hal Daumé, III. Structured prediction via learning to search under bandit feedback. In *Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing*. Association for Computational Linguistics, 2017.

1.6 Dissertation Outline

We begin by introducing the meta-learning and reinforcement learning background needed to understand the remainder of this dissertation in chapter 2. To coherently present the thesis, we discuss prior work related to each application in its own chapters, instead of putting them all in a single chapter. Therefore, we include a section of related work with each application. Chapter 9 concludes with a summary and future work.

The thesis is divided into two parts, each consisting of three chapters. Part I presents algorithms for learning with minimal supervision based on meta learning. This part includes the following algorithms.

1. Chapter 3 presents our meta-learning algorithm META-MT for adapting Neural Machine Translation (NMT) systems to new domains in a few-shot learning setting

[1] Authors are listed alphabetically.

where the agent has access to only few hundred parallel sentences from the targeted domain.

2. Chapter 4 presents PANDA, a meta-learning algorithm for promoting the learning of fair models via Meta-learning.

3. Chapter 5 introduces MÊLÉE, a meta-learning algorithm that allows an agent to learn better exploration strategies in a contextual bandit setting.

All of these approaches assume that we have fully supervised data on which we can run simulations at training time. Part II studies a different form of learning with minimal supervision. In this part, we introduce learning algorithms based on reinforcement learning, where an agent learns by observing reward (or loss) signals for the actions executed by the agent. This part includes the following algorithms:

1. Chapter 6 presents RESLOPE, a reinforcement learning algorithm for learning with very sparse reward signals observed only at the end of a learning episode.

2. Chapter 7 introduces LEAQI, an active imitation learning algorithm that minimizes the cost of querying an expensive expert by leveraging access to a possibly noisy guidance from a weaker heuristic.

3. Chapter 8 presents BLS, a learning algorithm for solving structured prediction problems in a bandit setting. In contrast to RESLOPE where the agent observes the reward signal at the end of the episode, BLS studies the setting when the agent also has access to partial reward signals for each selected action.

We start the discussion by providing the necessary background in chapter 2.

Chapter 2: Background

In this dissertation, we promote the learning of models that require minimal amount of supervision. We study two family of algorithms based on meta-learning and reinforcement learning techniques. This chapter presents an overview for the three main knowledge areas this dissertation touches upon: meta-learning (§2.1), reinforcement learning (§2.2), and imitation learning (§2.3). The meta-learning background is most relevant to Part I of the dissertation, where we study minimally supervised learning algorithms based on meta-learning. While the background on reinforcement and imitation learning are more relevant to Part II.

2.1 Meta-Learning Background

The goal of meta-learning is to train a model that can quickly learn a new task using only a few data points and training iterations. To accomplish this, the agent is trained during a meta-training phase on a set of similar learning tasks, such that the trained agent can quickly learn the new tasks using only a small number of examples and iterations. In effect, the meta-learning problem treats an entire learning task as training examples. The main idea is to simulate at training time a setting in which the agent gets exposed to only a handful of examples to learn from. Throughout these simulations, the agent effectively learns to generalize to new unseen learning tasks at test time, where it is required to learn these new and previously unseen tasks in a minimally supervised setting given just a few training samples. We formalize this meta-learning problem setting in a general manner

below, and provide more concrete instantiations for this generic formulation in Part I of this dissertation (see chapter 3, chapter 4, and chapter 5 for a more concrete discussion).

2.1.1 Meta-Learning Formulation

In a standard fully supervised machine learning setting, we are interested in learning model parameters θ on data points sampled from a distribution D. The data points from the distribution D are usually split into two subsets: we optimize the parameters θ on a training set D_{train} and evaluate its generalization on the test set D_{test}. In meta-learning, however, we deal with meta-sets $\mathcal{D}$ containing multiple regular datasets, where each $D \in \mathcal{D}$ has a split of D_{train} and D_{test}.

In meta-learning, we thus have different meta-sets for meta-training, meta-validation, and meta-testing ($\mathcal{D}_{\text{meta-train}}$, $\mathcal{D}_{\text{meta-validation}}$, and $\mathcal{D}_{\text{meta-test}}$ respectively). On $\mathcal{D}_{\text{meta-train}}$ we are interested in training a learning procedure (the meta-learner) that can take as input one of its training sets D_{train} and produce a learner that achieves high average performance on its corresponding test set D_{test}. Using $\mathcal{D}_{\text{meta-validation}}$, we can perform hyper-parameter selection of the meta-learner, and finally we can evaluate the meta-learner's generalization performance on $\mathcal{D}_{\text{meta-test}}$.

More formally, we consider a model, denoted f, that maps observations $\boldsymbol{x}$ to outputs $\boldsymbol{y}$. During meta-learning, the model is trained to be able to learn on a large number of tasks. Since we would like to apply this framework to a variety of learning problems, from adapting a Neural Machine Translation (NMT) model (see chapter 3) to Active Learning (see chapter 4), we introduce a generic notion of a learning task below.

Formally, each task $\mathrm{T} = \{\ell(\theta, \mathcal{D}), \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}\}$ consists of a loss function ℓ that takes as input the model's parameters θ and a dataset $\mathcal{D}$, a training dataset $\mathcal{D}_{\text{train}}$, and finally a testing dataset $\mathcal{D}_{\text{test}}$. The loss function $\ell(\theta, \mathcal{D}) \rightarrow \mathbb{R}$ provides task specific feedback for the model f_θ, which might be in the form of a misclassification or a cross-entropy loss.

In our meta-learning scenarios, we consider a distribution over tasks $\mathrm{P}(\mathrm{T})$ that we want our model to be able to learn from. In the NMT model adaptation scenario (chapter 3), the model is trained to adapt to a new domain T_i drawn from $\mathrm{P}(\mathrm{T})$ from only very few in-domain data from $\mathcal{D}_{\mathrm{T}_i}^{\text{train}}$. We use the cross-entropy loss function ℓ_{T_i} to evaluate the adaptation performance on the task specific test split $\mathcal{D}_{\mathrm{T}_i}^{\text{test}}$. During meta-training, a task T_i is sampled from $\mathrm{P}(\mathrm{T})$, the model is adapted with a very small dataset $\mathcal{D}_{\mathrm{T}_i}^{\text{train}}$ using feedback from the cross-entropy loss ℓ_{T_i}, and then tested on samples from $\mathcal{D}_{\mathrm{T}_i}^{\text{test}}$. The model f is then improved by considering how the test error on new data $\mathcal{D}_{\mathrm{T}_i}^{\text{test}}$ changes with respect to the parameters. In effect, the test error on sampled tasks T_i serves as the training error of the meta-learning process. At the end of meta-training, new tasks are sampled from $\mathrm{P}(\mathrm{T})$ to construct the meta-test set $\mathcal{D}^{\text{meta-test}}$, and meta-performance is measured by the model's performance after adapting to new domains from $\mathcal{D}^{\text{meta-test}}$. Generally, adaptation tasks used for meta-testing are held out during meta-training.

Following notation from Finn et al. (2017), Table 2.1 shows an overview for the meta-learning terminology and notation used in this dissertation. In essence, meta-learning algorithms learn to learn tasks using data from tasks in the meta-training set $\mathcal{D}_{\text{meta-train}}$. After meta-learning, the learned learning algorithm is evaluated in its ability to learn new tasks in the meta-test set $\mathcal{D}_{\text{meta-test}}$. We use the term "Task" broadly to encapsulate a concept to be learned, a domain to be adapted to, or combinations thereof. We provide concrete examples with task definitions in chapter 3, chapter 4, and chapter 5.

2.1.2 Optimization Based Meta-Learning

The meta-learning question is: how can we learn the parameters θ for the meta-learner f_θ by observing learning tasks sampled from the distribution $\mathrm{P}(\mathrm{T})$? In Part I of this dissertation we use an optimization based approach to learn the meta-learner's parameters θ. Formally, we consider a meta-learner model represented by a parametrized function f_θ with parameters θ. When learning from a new task T_i, the model's parameters θ become ϕ_i.

Symbol	Terminology	Examples / More Details
T	Task	Entity being learned or adapted to, corresponds to an objective, domain, environment, or combinations thereof
$\mathrm{P}(\mathrm{T})$	Task Distribution	Distribution of tasks from which the meta-training and meta-testing tasks are drawn.
$\{\mathrm{T}_i\} \sim \mathrm{P}(\mathrm{T})$	meta-train tasks	Set of tasks used for meta-learning
$\{\mathcal{D}_{\mathrm{T}_i}\}$	meta-train set	Set of datasets corresponding to the meta-training tasks; the algorithm will learn to learn from data in these datasets
$\{\mathrm{T}_j\} \sim \mathrm{P}(\mathrm{T})$	meta-test tasks	Set of tasks used for evaluation; the learned learning procedure will be evaluated on its ability to learn these tasks
$\{\mathcal{D}_{\mathrm{T}_j}\}$	meta-test set	Set of datasets corresponding to meta-test tasks
$\mathcal{D}_{\mathrm{T}}^{\text{train}}$	training set (support set)	Training data for task T
$\mathcal{D}_{\mathrm{T}}^{\text{test}}$	test set (query set)	Test data for task T, sampled from $\mathcal{D}_{\mathrm{T}}$

Table 2.1: Summary of meta-learning terminology used in this dissertation.

The updated parameter vector ϕ_i is computed using one or more gradient descent updates on the training data for task T_i. For example, when using one gradient update:

$$\phi_i = \theta - \alpha \nabla_\theta \ell(\theta, \mathcal{D}_{\mathrm{T}_i}^{\text{train}}) \tag{2.1}$$

The step size α may be fixed as a hyper-parameter or meta-learned. For simplicity of notation, we consider one gradient update for the rest of this section, but using multiple gradient updates is a straightforward extension.

The meta-learner model parameters are trained by optimizing the performance of f_{ϕ_i} with respect to θ across tasks sampled from $\mathrm{P}(\mathrm{T})$. More concretely, the meta-objective is:

$$\min_\theta \sum_{\mathrm{T}_i \sim \mathrm{P}(\mathrm{T})} \ell(\phi_i, \mathcal{D}_{\mathrm{T}_i}^{\text{test}}) = \min_\theta \sum_{\mathrm{T}_i \sim \mathrm{P}(\mathrm{T})} \ell(\theta - \alpha \nabla_\theta \ell(\theta, \mathcal{D}_{\mathrm{T}_i}^{\text{train}}), \mathcal{D}_{\mathrm{T}_i}^{\text{test}}) \tag{2.2}$$

Note that the meta-optimization is performed over the meta-learner model parameters θ, whereas the objective is computed using the updated model parameters ϕ. In effect, optimization based meta-learning aims to optimize the model parameters such that one or

1 Optimization Based Meta-Learning

Require: $\mathrm{P}(\mathrm{T})$: distribution over tasks
Require: α, β: step size hyper-parameters
1: Randomly initialize θ
2: **while** Not Done **do**
3: Sample batch of tasks $\mathrm{T}_i \sim \mathrm{P}(\mathrm{T})$
4: **for** each T_i **do**
5: Sample task specific training dataset $\mathcal{D}_{\mathrm{T}_i}^{\text{train}} \sim \mathcal{D}_{\mathrm{T}_i}$
6: Sample task specific testing dataset $\mathcal{D}_{\mathrm{T}_i}^{\text{test}} \sim \mathcal{D}_{\mathrm{T}_i}$
7: Evaluate $\nabla_\theta \ell(\theta, \mathcal{D}_{\mathrm{T}_i}, \mathcal{D}_{\tau_i}^{\text{train}})$
8: Compute adapted parameters with gradient descent: $\phi_i = \theta - \alpha \nabla_\theta \ell(\theta, \mathcal{D}_{\mathrm{T}_i}^{\text{train}})$
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathrm{T}_i \sim \mathrm{P}(\mathrm{T})} \ell(\phi_i, \mathcal{D}_{\mathrm{T}_i}^{\text{test}})$
11: **end while**

small number of gradient steps on a new task will produce maximally effective behavior on that task.

The meta-optimization across tasks is performed via stochastic gradient descent (SGD), such that the model parameters θ are updated as follows:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathrm{T}_i \sim \mathrm{P}(\mathrm{T})} \ell(\phi_i, \mathcal{D}_{\mathrm{T}_i}^{\text{test}}) \tag{2.3}$$

where β is the meta-step size. This optimization based meta-learning algorithm is adapted from the Model Agnostic Meta-Learning (MAML) algorithm from Finn et al. (2017), and is outlined in 1. The gradient of the meta-objective update involves a gradient through a gradient. Computationally, this requires an additional backward pass through f to compute Hessian vector products, which is supported by standard deep learning libraries.

2.2 Reinforcement Learning Background

In Part I of this dissertation we focus on algorithms for learning with minimal supervision based on meta-learning. However, for these algorithms to work, we need

access to similar learning tasks on which we can run simulations for learning with minimal supervision, but what if we don't have access to these simulations? In Part II we show that we can still learn with minimal supervision using reinforcement learning reward signals.

Reinforcement learning is learning what to do – i.e. how to map situations to actions– so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In many cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics of trial-and-error search and delayed reward are the two most important distinguishing features in reinforcement learning.

In this section we introduce a mathematically idealized form of the reinforcement learning problem. We introduce the formal problem of finite Markov decision process, or finite MDPs. MDPs are a classical formulation of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards. We introduce key elements of the problem's mathematical structure such as states, actions, rewards, returns, and value functions. We follow the notation from Sutton and Barto (1998).

2.2.1 The Agent-Environment Interface

MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision maker is called the agent. The system is interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selects actions and the environment responds to these actions and presents new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions.

Formally, the agent and the environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \cdots$. At each time step t, the agent receives some representation of the environment's **state**, $S_t \in \mathcal{S}$, and on that basis selects an **action**, $A_t \in \mathcal{A}(s)$. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathbb{R}$, and finds itself in a new state, S_{t+1}. The MDP and agent together thereby give rise to a sequence of trajectory that begins like:

$$S_0, A_0, R_1, S_1, A_1, S_2, A_2, R_3, \cdots \tag{2.4}$$

In a finite MDP, the sets of states, actions, and rewards $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ all have a finite number of elements. In this case, the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action. That is, for particular values of these random variables, $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at time t, given particular values of the preceding state and action:

$$p(s', r|s, a) = \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{2.5}$$

for all $s', s \in \mathcal{S}, r \in \mathbb{R}$, and $a \in \mathcal{A}$. The function p defines the dynamics of the MDP. In a Markov decision process, the probabilities given by p completely characterize the environment's dynamics. That is, the probability of each possible value for S_t and R_t depends only on the immediately proceeding state and action, S_{t-1} and A_{t-1}, and, given them, not at all on earlier states and actions. The state must include information about all aspects of the past agent-environment interaction that make a difference for the future. If it does, then the states is said to have the Markov property.

Rewards In reinforcement learning, the purpose of the agent is formalized in terms of special signal, called the reward, passing from the environment to the agent. At each time step, the reward is a simple number, $R_t \in \mathbb{R}$. The agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but

cumulative reward in the long run. The agent seeks to maximize the expected return, where the return, denoted G_t is defined as the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T \tag{2.6}$$

where T is the final time step.

2.2.2 Policies and Value Functions

Almost all reinforcement learning algorithms involve estimating value functions - functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of "how good" here is defined in terms of future rewards that can be expected in terms of expected return.

A **policy** is a mapping from states to probabilities of selecting each possible action. If the agent is following policy π at time t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$.

The value function of a state s under a policy π, denoted $v_\pi(s)$ is the expected return when starting in s and following π thereafter. For MDPs, we can denote v_π formally by:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \forall s \in \mathcal{S} \tag{2.7}$$

where $\mathbb{E}_\pi[.]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step.

2.3 Imitation Learning Background

Reinforcement learning takes the trial-and-error approach and uses the end loss / reward as supervised signal to evaluate how good a policy is. However, sometimes, it is much harder to quantify the value of a certain behavior than to demonstrate the desired

behavior. For example, it is not clear exactly how bad it is to drive slightly off the road, but it is easy to show a good driving path.

Imitation learning assumes access to an expert who shows good actions to take in any given state. During policy learning, examples of state / action pairs generated by the expert are used as supervised signals. Instead of minimizing the cumulative loss, in imitation learning, we minimize the difference from the expert actions. By mimicking the oracle actions, our ultimate goal is to minimize the task loss defined over the entire sequence. In chapter 5 and chapter 7 we show how imitation learning could be used to design algorithms for learning with minimal supervision.

In a sequential decision making process, at each time step t, the system is in some state $s \in \mathcal{S}$, an agent chooses an action $a = \pi(s)$ from the action set $\mathcal{A}$ using policy π. A policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, is a mapping from a state (usually a feature representation of that state) to an action. After taking the action, it then transitions to a new state s', inducing loss $L(s, a)$. The induced loss indicates the goodness of taking action a in state s. The system repeats this process until it reaches the terminal state. A trajectory is a complete sequence of $\langle s_t, a_t, L(s_t, a_t)\rangle$ tuples from the starting state $(t = 1)$ to the terminal state $(t = T)$.

Let d_π^t be the state distribution at time t after executing π from time 1 to $t - 1$, and d_π be the average state distribution of states over T steps. The task loss is defined as the T-step expected loss of π : $J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s\sim d_\pi^t}[L(s, \pi(s))] = T\mathbb{E}_{s\sim d_\pi}[L(s, \pi(s))]$. An optimal policy π^* is a policy that minimizes the loss $J(\pi)$.

2.3.1 Behavior Cloning

A straight forward approach to imitation learning is to use the oracle's trajectories as supervised data and learn a policy (multi-class classifier) that predicts the oracle action. This approach is known as behavioral cloning. At each step t, we collect a training example $(s_t, \pi^*(s_t))$, where $\pi^*(s_t)$ is the oracle's action (class label) in state s_t and s_t is the state. Let $l(s, \pi, \pi^*(s))$ denote the surrogate loss of executing π in state s with respect to $\pi^*(s)$.

This can be any convex loss function used for training the classifier, for example, hinge loss in Support Vector Machines (SVM), or logistic loss in logistic regression. Using any standard supervised learning algorithm, we can learn a policy:

$$\hat{\pi} = \underset{\pi \in \Pi}{\operatorname{argmin}} \, \mathbb{E}_{s \sim d_{\pi^*}}[l(s, \pi, \pi^*(s))] \tag{2.8}$$

where Π is the policy space and d_π is the distribution of states generated by executing the expert policy. We can bound the task loss $J(\pi)$ based on how well the learner imitates the oracle. Assuming $l(s, \pi, \pi^*(s))$ is an upper bound on the $0-1$ loss and $L(s, a)$ is bounded in $[0, 1]$, Ross and Bagnell (2014) have shown that:

Theorem 1. *Let* $\mathbb{E}_{s \sim d_{\pi^*}}[l(s, \pi, \pi^*(s))] = \epsilon$*, then* $J(\pi) \leq J(\pi^*) + T^2\epsilon$

One drawback of this approach is that it ignores the fact that the state distribution is different for the oracle and the learner. When the learner cannot mimic the oracle perfectly, i.e. classification error occurs, the wrong action will change the following state distribution. Thus, the learner policy is not able to handle situations where the learner follows a wrong path that is never chosen by the oracle. This lead to compounding errors causing the quadratically increasing loss.

2.3.2 Dataset Aggregation

The above problem of insufficient exploration can be alleviated by iteratively learning a policy trained under states visited by both the oracle and the learner. For example, during training one can execute a "mixture policy" that at times takes an action given by the previous learned policy. Alternatively, at each iteration one can learn a policy from trajectories generated by all previous policies, including the expert policy.

The Dataset Aggregation (DAgger) algorithm Ross and Bagnell (2014) works as follows. Let s_π denote that state visited by executing policy π. In the first iteration, we collect a training set $\mathcal{D}_1 = \{(s_{\pi^*}, \pi^*(s_{\pi*}))\}$ from the expert $(\pi_1 = \pi^*)$, and then learn a

policy π_2. This is the same as the behavior Cloning approach to imitation. In iteration i, we collect trajectories by executing the previous policy π_i and form the training set $\mathcal{D}_i$ by labeling s_{π_i} with the oracle action $\pi^*(s_{\pi_i})$; π_{i+1} is then learned on $\mathcal{D}_0 \cup \cdots \mathcal{D}_i$. Intuitively, this enables the learner to make up for past failures to mimic the oracle. Thus, we can obtain a policy that performs well under its own induced state distribution.

Unlike behavior cloning which yields loss quadratically growing with T, DAgger guarantees loss linear in T. Formally, assume that $l(s, \pi, \pi^*(s))$ is a strongly convex loss in π upper bounding the $0-1$ loss. We denote the sequence of learned policies by $\pi_1, \pi_2, \cdots, \pi_N$. Let $\epsilon_N = \min_{\pi \in \Pi} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{s \sim d_{\pi_i}}[l(s, \pi, \pi^*(s))]$ be the minimum loss we can achieve in the policy space Π, and let $Q_t^{\pi'}(s, \pi)$ denote the t-step loss of executing π in the initial state and then running π' thereafter. It can be shown that:

Theorem 2. *If N is $iO(uT \log T)$ and $Q^{\pi^*}_{T-t+1}(s, \pi) - Q^{\pi^*}_{T-1+1}(s, \pi^*) \leq u$, there exists a policy $\pi \in \pi_{1:N}$ such that $J(\pi) \leq J(\pi^*) + uT\epsilon_N + O(1)$*

The theorem says that if the training error of the supervised classification problem is ϵ_N, then the task loss relative to the oracle is $O(uT\epsilon_N)$, given that the cumulative cost is bounded by u when π and π^* chooses different actions at any time step t.

2.4 Summary

In this chapter we introduced the basics of meta-learning, reinforcement learning, and imitation learning algorithms. In the rest of this dissertation we demonstrate how these families of algorithms could be used to design learning algorithms with minimal supervision. In Part I we present algorithms based on meta-learning, while in Part II we study learning algorithms based on reinforcement learning. Imitation learning is usually more efficient than reinforcement learning, however, it requires access to an expert oracle at training time. Whenever we have access to such expert, we take advantage of the

imitation learning efficiency in comparison to reinforcement learning. We present two learning algorithms that utilize imitation learning in chapter 5 and chapter 7.

Part I

Meta-Learning Algorithms

Chapter 3: Meta-Learning for Few-Shot NMT Adaptation

3.1 Introduction

In this part of the dissertation we present minimally supervised learning algorithms based on meta-learning. In this chapter, we study the few-shot learning setting as a form of minimal supervision. We present a minimally supervised meta-learning approach that *learns* to adapt neural machine translation systems to new domains given only a small amount of training data in the targeted domain. To achieve this, we simulate many domain adaptation tasks, on which we use a *meta-learning* strategy to learn how to adapt. Based on these simulations, our approach, META-MT (Meta-learning for Machine Translation), learns model parameters that generalize to future (real) adaptation tasks (§3.4.1).

Neural Machine Translation (NMT) systems (Bahdanau et al., 2016b; Sutskever et al., 2014) are usually trained on large general-domain parallel corpora to achieve state-of-the-art results (Barrault et al., 2019). Unfortunately, these generic corpora are often qualitatively different from the targeted domain of the translation system. Moreover, NMT models trained on one domain tend to perform poorly when translating sentences in a significantly different domain (Koehn and Knowles, 2017; Chu and Wang, 2018).

A widely used approach for adapting NMT systems is *domain adaptation by fine-tuning* (Luong and Manning, 2015; Freitag and Al-Onaizan, 2016; Sennrich et al., 2016), where a model is first trained on general-domain data and then adapted by continuing the training on a smaller amount of in-domain data. This approach often leads to empirical improvements in the targeted domain; however, it fails when the amount of in-domain data

is insufficient, leading to model over-fitting and catastrophic forgetting, where adapting to a new domain leads to a degradation on the general-domain (Thompson et al., 2019).

Ideally, we would like to have a model that is easily adaptable to many domains with minimal amount of in-domain data. Towards this goal, at training time (§ 3.4.2), META-MT simulates many small-data domain adaptation tasks from a large pool of data. Using these tasks, META-MT simulates what would happen after fine-tuning the model parameters to each such task. It then uses this information to compute parameter updates that will lead to efficient adaptation during deployment. We optimize these parameters using the Model Agnostic Meta-Learning algorithm (MAML) (Finn et al., 2017).

The contributions of this chapter are as follows:

1. First, we propose a new approach that enables NMT systems to effectively adapt to a new domain using few-shot learning.

2. Second, we show what models and conditions enable meta-learning to be useful for NMT adaptation.

3. Finally, We evaluate META-MT on ten different domains, showing the efficacy of our approach.

To the best of our knowledge, this is the first work on adapting large scale NMT systems in a few-shot learning setup [1].

3.2 Related Work

Our goal for few-shot NMT adaptation is to adapt a pre-trained NMT model (e.g. trained on general domain data) to new domains (e.g. medical domain) with a small amount of training examples. Chu et al. (2018) surveyed several recent approaches that address the shortcomings of traditional fine-tuning when applied to domain adaptation.

[1] **Code Release:** https://bit.ly/34KJOKv

Our work distinguishes itself from prior work by learning to fine-tune with tiny amounts of training examples.

Most recently, Bapna et al. (2019) proposed a simple approach for adaptation in NMT. The approach consists of injecting task specific adapter layers into a pre-trained model. These adapters enable the model to adapt to new tasks as it introduces a bottleneck in the architecture that makes it easier to adapt. Our approach uses a similar model architecture, however, instead of injecting a new adapter for each task separately, META-MT uses a single adapter layer, and meta-learns a better initialization for this layer that can easily be fine-tuned to new domains with very few training examples.

Similar to our goal, Michel and Neubig (2018) proposed a space efficient approach to adaptation that learns domain specific biases to the output vocabulary. This enables large-scale personalization for NMT models when small amounts of data are available for a lot of different domains. However, this approach assumes that these domains are static and known at training time, while META-MT can dynamically generalize to totally new domains, previously unseen at meta-training time.

Several approaches have been proposed for lightweight adaptation of NMT systems. Vilar (2018) introduced domain specific gates to control the contribution of hidden units feeding into the next layer. However, Bapna et al. (2019) showed that this introduced a limited amount of per-domain capacity; in addition, the learned gates are not guaranteed to be easily adaptable to unseen domains. Khayrallah et al. (2017) proposed a lattice search algorithm for NMT adaptation, however, this algorithm assumes access to lattices generated from a phrase based machine translation system.

Our meta-learning strategy mirrors that of Gu et al. (2018) in the low resource translation setting, as well as Wu et al. (2019) for cross-lingual named entity recognition with minimal resources, Mi et al. (2019) for low-resource natural language generation in task-oriented dialogue systems, and Dou et al. (2019) for low-resource natural language

understanding tasks. To the best of our knowledge, this is the first work using meta-learning for few-shot NMT adaptation.

3.3 Background

3.3.1 Neural Machine Translation

Neural Machine Translation is a sequence to sequence model that parametrizes the conditional probability of the source and target sequences as a neural network following encoder-decoder architecture (Bahdanau et al., 2016b; Sutskever et al., 2014). Initially, the encoder-decoder architecture was represented by recurrent networks. Currently, this has been replaced by self-attention models aka Transformer models (Vaswani et al., 2017)). Currently, Transformer models achieves state-of-the-art performance in NMT as well as many other language modeling tasks. While transformers models are performing quite well on large scale NMT tasks, the models have huge number of parameters and require large amounts of training data which is really prohibitive for adaptation tasks especially in few-shot setup like ours.

3.3.2 Few-Shot Domain Adaptation

Traditional domain adaptation for NMT models assumes the availability of relatively large amounts of in domain data. For instance, most of the related work utilizing traditional fine-tuning experiment with hundred-thousand sentences in-domain. This setup is quite prohibitive, since practically the domain can be defined by few examples. In this work we focus on few-shot adaptation scenario where we can adapt to a new domain not seen during training time using just couple of hundreds of in-domain sentences. This introduces a new challenge where the models have to be responsive to adaptation as well as robust to domain shift. Since we focus on the setting in which very few in-domain data is available, this renders many traditional domain adaptation approaches inapplicable.

3.4 Approach: Meta-Learning for Few-Shot NMT Adaptation

Neural Machine Translation systems are not robust to domain shifts Chu and Wang (2018). It is a highly desirable characteristic of the system to be adaptive to any domain shift using weak supervision without degrading the performance on the general domain. This dynamic adaptation task can be viewed naturally as a learning-to-learn (meta-learning) problem: how can we train a global model that is capable of using its previous experience in adaptation to learn to adapt faster to unseen domains? A particularly simple and effective strategy for adaptation is fine-tuning: the global model is adapted by training on in-domain data. One would hope to improve on such a strategy by decreasing the amount of required in-domain data. META-MT takes into account information from previous adaptation tasks, and aims at learning how to update the global model parameters, so that the resulting learned parameters after meta-learning can be adapted faster and better to previously unseen domains via a weakly supervised fine-tuning approach on a tiny amount of data.

Our goal in this chapter is to learn how to adapt a neural machine translation system from experience. The training procedure for META-MT uses offline simulated adaptation problems to learn model parameters θ which can adapt faster to previously unseen domains. In this section, we describe META-MT, first by describing how it operates at test time when applied to a new domain adaptation task (§3.4.1), and then by describing how to train it using offline simulated adaptation tasks (§3.4.2).

3.4.1 Test Time Behavior of META-MT

At test time, META-MT adapts a pre-trained NMT model to a new given domain. The adaptation is done using a small in-domain data that we call the *support set* and then tested on the new domain using a *query set*. More formally, the model parametrized by θ takes as input a new adaptation task T. This is illustrated in Figure 3.1: the adaptation task T consists of a standard domain adaptation problem: T includes a support set $\mathrm{T}_{\text{support}}$

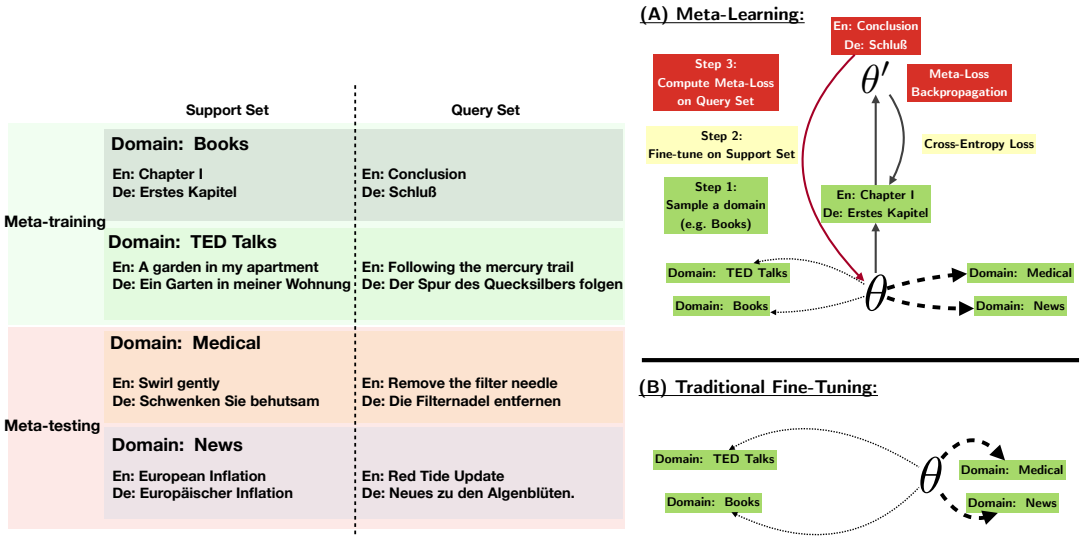


Figure 3.1: (Left) Example meta-learning set-up for few-shot NMT adaptation. The top represents the meta-training set $\mathcal{D}_{\text{meta-train}}$, where inside each box is a separate dataset T that consists of the support set $\mathrm{T}_{\text{support}}$ (left side of dashed line) and the query set $\mathrm{T}_{\text{query}}$ (right side of dashed line). In this illustration, we are considering the books and TED talks domains for meta-training. The meta-test set $\mathcal{D}_{\text{meta-test}}$ is defined similarly, but with a different set of domains not in any of the datasets in $\mathcal{D}_{\text{meta-train}}$: Medical and News. (Right) **[Top-A]** a training step of META-MT. **[Bottom-B]** Differences between meta-learning and Traditional fine-tuning. Wide lines represent high resource domains (Medical, News), while thin lines represent low-resource domains (TED, Books). Traditional fine-tuning may favor high-resource domains over low-resource ones while meta-learning aims at learning a good initialization that can be adapted to any domain with minimal training samples.

used for training the fine-tuned model, and a query set $\mathrm{T}_{\text{query}}$ used for evaluation. We are particularly interested in the distribution of tasks $\mathrm{P}(\mathrm{T})$ where the support and query sets are very small. In our experiments, we restrict the size of these sets to only few hundred parallel training sentences. We consider support sets of sizes: 4k to 64k source words (i.e. $\sim$ 200 to 3200 sentences). At test time, the meta-learned model θ interacts with the world as follows (Figure 3.1):

1. **Step 1:** The world draws an adaptation task T from a distribution P, $\mathrm{T} \sim \mathrm{P}(\mathrm{T})$;
2. **Step 2:** The model adapts from θ to θ' by fine-tuning on the task's support set $\mathrm{T}_{\text{support}}$;
3. **Step 3:** The fine-tuned model θ' is evaluated on the query set $\mathrm{T}_{\text{query}}$.

Intuitively, meta-training should optimize for a representation θ that can quickly adapt to new tasks, rather than a single individual task.

3.4.2 Training META-MT via Meta-learning

The meta-learning challenge is: how do we learn a good representation θ? We initialize θ by training an NMT model on global-domain data. In addition, we assume access to meta-training tasks on which we can train θ; these tasks must include support/query pairs, where we can simulate a domain adaptation setting by fine-tuning on the support set and then evaluating on the query. This is a weak assumption: in practice, we use purely simulated data as this meta-training data. We construct this data as follows: given a parallel corpus for the desired language pair, we randomly sample training examples to form a few-shot adaptation task. We build tasks of 4k, 8k, 16k, 32k, and 64k training words. Under this formulation, it is natural to think of θ's learning process as a process to learn a good parameter initialization for fast adaptation, for which a class of learning algorithms to consider are Model-agnostic Meta-Learning (MAML) and its first order approximations like First-order MAML Finn et al. (2017) and Reptile Nichol et al. (2018).

[1] colorblind friendly palette was selected from Neuwirth and Brewer (2014).

2 META-MT (trained model f_θ, meta-training dataset $\mathcal{D}_{\text{meta-train}}$, learning rates α, β)

while not done **do**
  Sample a batch of domain adaptation tasks $\mathrm{T} \sim \mathcal{D}_{\text{meta-train}}$
  for all $\mathrm{T}_i \in \mathrm{T}$ **do**
    Evaluate $\nabla_\theta L_{\mathrm{T}_i}(f_\theta)$ on the support set $\mathrm{T}_{i,\text{support}}$
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta L_{\mathrm{T}_i}(f_\theta)$
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \Sigma_{\mathrm{T}_i \in \mathrm{T}} L_{\mathrm{T}_i}(f_{\theta_i'})$ on the query set $\mathrm{T}_{i,\text{query}} \forall \mathrm{T}_i \in \mathrm{T}$
end while

Informally, at training time, META-MT will treat one of these simulated domains T as if it were a domain adaptation dataset. At each time step, it will update the current model representation from θ to θ' by fine-tuning on $\mathrm{T}_{\text{support}}$ and then ask: what is the meta-learning loss estimate given θ, θ', and T_{query}? The model representation θ is then updated to minimize this meta-learning loss. More formally, in meta-learning, we assume access to a distribution P over different tasks T. From this, we can sample a meta-training dataset $\mathcal{D}_{\text{meta-train}}$. The meta-learning problem is then to estimate θ to minimize the meta-learning loss on $\mathcal{D}_{\text{meta-train}}$.

The meta-learning algorithm we use is MAML by Finn et al. (2017), and is instantiated for the meta-learning to adapt NMT systems in 2. MAML considers a model represented by a parametrized function f_θ with parameters θ. When adapting to a new task T, the model's parameters θ become θ'. The updated vector θ' is computed using one or more gradient descent updates on the task T. For example, when using one gradient update:

$$\theta' = \theta - \alpha \nabla_\theta L_{\mathrm{T}}(f_\theta) \tag{3.1}$$

where α is the learning rate and L is the task loss function. The model parameters are trained by optimizing for the performance of $f_{\theta'}$ with respect to θ across tasks sampled

from $\mathrm{P(T)}$. More concretely, the meta-learning objective is:

$$\min_{\theta} \Sigma_{\mathrm{T}\sim\mathrm{P(T)}} L_{\mathrm{T}}(f_{\theta'}),$$
$$L_{\mathrm{T}}(f_{\theta'}) = L_{\mathrm{T}}(f_{\theta - \alpha\nabla_{\theta} L_{\mathrm{T}}(f_{\theta})}) \tag{3.2}$$

Following the MAML template, META-MT operates in an iterative fashion, starting with a trained NMT model f_θ and improving it through optimizing the meta-learning loss from Eq 3.2 on the meta-training dataset $\mathcal{D}_{\text{meta-train}}$. Over learning rounds, META-MT selects a random batch of training tasks from the meta-training dataset and simulates the test-time behavior on these tasks (Line 2). The core functionality is to observe how the current model representation θ is adapted for each task in the batch, and to use this information to improve θ by optimizing the meta-learning loss (Line 7). META-MT achieves this by simulating a domain adaptation setting by fine-tuning on the task specific support set (Line 4). This yields, for each task T_i, a new adapted set of parameters θ'_i (Line 5). These parameters are evaluated on the query sets for each task $\mathrm{T}_{i,\text{query}}$, and a meta-gradient w.r.t the original model representation θ is used to improve θ (Line 7).

Our pre-trained baseline NMT model f_θ is a sequence to sequence model that parametrizes the conditional probability of the source and target sequences as an encoder-decoder architecture using self-attention Transformer models Vaswani et al. (2017)).

3.5 Experimental Setup and Results

We seek to answer the following questions experimentally:

1. How does META-MT compare empirically to alternative adaptation strategies? (§3.5.4)
2. What is the impact of the support and the query sizes used for meta-learning? (§3.5.5)
3. What is the effect of the NMT model architecture on performance? (§3.5.6)

In our experiments, we train META-MT only on simulated data, where we simulate a few-shot domain adaptation setting as described in § 3.4.2. This is possible because META-MT learns model parameters θ that can generalize to future adaptation tasks by optimizing the meta-objective function in Eq 3.2.

We train and evaluate META-MT on a collection of ten different datasets. All of these datasets are collected from the Open Parallel Corpus (OPUS) Tiedemann (2012), and are publicly available online. The datasets cover a variety of diverse domains that should enable us to evaluate our proposed approach. The datasets we consider are:

1. Bible: a parallel corpus from translations of the Bible Christodouloupoulos and Steedman (2015).
2. European Central Bank: website and documentations from the European Central Bank.
3. KDE: a corpus of KDE4 localization files.
4. Quran: a collection of Quran translations compiled by the Tanzil project.
5. WMT news test sets: a parallel corpus of News Test Sets provided by WMT.
6. Books: a collection of copyright free books.
7. European Medicines Agency (EMEA): a parallel corpus made out of PDF documents from the European Medicines Agency.
8. Global Voices: parallel news stories from the Global Voices web site.
9. Medical (ufal-Med): the UFAL medical domain dataset from Yepes et al. (2017).
10. TED talks: talk subtitles from Duh (2018).

We simulate the few-shot NMT adaptation scenarios by randomly sub-sampling these datasets with different sizes. We sample different data sets with sizes ranging from 4k to 64k training words (i.e. $\sim$ 200 to 3200 sentences). This data is the only data used for any given domain across all adaptation setups. It is worth noting that different datasets have a wide range of sentence lengths. We opted to sample using number of words instead

of number of sentences to avoid introducing any advantages for domains with longer sentences.

3.5.1 Domain Adaptation Approaches

Our experiments aim to determine how META-MT compares to standard domain adaptation strategies. In particular, we compare to:

(A) **No fine-tuning:** The non-adaptive baseline. Here, the pre-trained model is evaluated on the meta-test and meta-validation datasets (see Figure 3.1) without any kind of adaptation.

(B) **Fine-tuning on a single task:** The domain adaptation by fine-tuning baseline. For a single adaptation task T, this approach performs domain adaptation by fine-tuning only on the support set $\mathrm{T}_{\text{support}}$. For instance, if $|\mathrm{T}_{\text{support}}| = K$ words, we fine tune the pre-trained model $f_\theta$only on K training words to show how classical fine-tuning behaves in few-shot settings.

(C) **Fine-tuning on meta-train:** Similar to (B), however, this approach fine-tunes on much more data. This approach fine-tunes on all the support sets used for meta-training: $\{\mathrm{T}_{\text{support}}, \forall \mathrm{T} \in \mathcal{D}_{\text{meta-train}}\}$. The goal of this baseline is to ensure that META-MT does not get an additional advantage by training on more data during the meta-training phase. For instance, if we are using N adaptation tasks each with a support set of size K, this will be using $N * K$ words for classical fine-tuning. This establishes a fair baseline to evaluate how classical fine-tuning would perform using the same data albeit in a different configuration.

(D) **META-MT** Our proposed approach from 2. In this setup, we use N adaptation tasks T in $\mathcal{D}_{\text{meta-train}}$, each with a support set of size K words to perform Meta-Learning. Second order meta-gradients are ignored to decrease the computational complexity.

3.5.2 Model Architecture and Implementation Details

We use the Transformer Model Vaswani et al. (2017) implemented in fairseq Ott et al. (2019). In this work, we use a transformer model with a modified architecture that can facilitate better adaptation. We use *"Adapter Modules"* Houlsby et al. (2019); Bapna et al. (2019) which introduce an extra layer after each transformer block that can enable more efficient tuning of the models. Following Bapna et al. (2019), we augment the Transformer model with feed-forward adapters: simple single hidden-layer feed-forward networks, with a nonlinear activation function between the two projection layers. These adapter modules are introduced after the Layer Norm and before the residual connection layers. It is composed of a down projection layer, followed by a ReLU, followed by an up projection layer. This bottle-necked module with fewer parameters is very attractive for domain adaptation as we will discuss in §3.5.6. These modules are introduced after every layer in both the encoder and the decoder. All experiments are based on the "base" transformer model with six blocks in the encoder and decoder networks. Each encoder block contains a self-attention layer, followed by two fully connected feed-forward layers with a ReLU non-linearity between them. Each decoder block contains self-attention, followed by encoder-decoder attention, followed by two fully connected feed-forward layers with a ReLU non-linearity between them.

We use word representations of size 512, feed-forward layers with inner dimensions $2,048$, multi-head attention with 8 attention heads, and adapter modules with 32 hidden units. We apply dropout Srivastava et al. (2014) with probability 0.1. The model is optimized with Adam Kingma and Ba (2014) using $\beta_1 = 0.9, \beta_2 = 0.98$, and a learning rate $\alpha = 7e-4$. We use the same learning rate schedule as Vaswani et al. (2017) where the learning rate increases linearly for $4,000$ steps to $7e-4$, after which it is decayed proportionally to the inverse square root of the number of steps. For meta-learning, we

used a meta-batch size of 1. We optimized the meta-learning loss function using Adam with a learning rate of $1e-5$ and default parameters for β_1, β_2.

All data is pre-processed with joint sentence-pieces Kudo and Richardson (2018) of size 40k. In all cases, the baseline machine translation system is a neural English to German (En-De) transformer model Vaswani et al. (2017), initially trained on 5.2M sentences filtered from the standard data (Europarl-v9, CommonCrawl, NewsCommentary-v14, wikititles-v1 and Rapid-2019) from the WMT19 shared task Barrault et al. (2019). We use WMT14 and WMT19 newtests as validation and test sets respectively. The baseline system scores 37.99 BLEU on the full WMT19 newstest which compares favorably with strong baselines at WMT19 shared task Ng et al. (2019); Junczys-Dowmunt (2019).

For meta-learning, we use the MAML algorithm as described in 2. To minimize memory consumption, we ignored the second order gradient terms from Eq 3.2. We implement the First-Order MAML approximation (FoMAML) as described in Finn et al. (2017). We also experimented with the first-order meta-learning algorithm Reptile Nichol et al. (2018). We found that since Reptile does not directly account for the performance on the task query set, along with the large model capacity of the Transformer architecture, it can easily over-fit to the support set, thus achieving almost perfect performance on the support, while the performance on the query degrades significantly. Even after performing early stopping on the query set, Reptile did not account correctly for learning rate scheduling, and finding suitable learning rates for optimizing the meta-learner and the task adaptation was difficult. In our experiments, we found it essential to match the behavior of the dropout layers when computing the meta-objective function in Eq 3.2 with the test-time behavior described in §3.4.1. In particular, the model has to run in *"evaluation mode"* when computing the loss on the task query set to match the test-time behavior during evaluation.

In contrast to training the generic NMT model which takes days to train, META-MT requires only few hours to train the meta-learner on a GeForce 1080Ti GPU, 2.1GHz Intel

Domain	A. No tuning	B. Tune on task	C. Tune on meta-train	D. META-MT
Books	11.338 ± 0.25	11.34 ± 0.24	$\underline{12.49 \pm 0.15}$	$\mathbf{12.92 \pm 0.94}$
Tanzil	11.25 ± 0.04	11.33 ± 0.04	$\underline{13.62 \pm 0.05}$	$\mathbf{15.16 \pm 0.94}$
Bible	12.93 ± 0.93	12.95 ± 0.94	17.19 ± 0.54	$\mathbf{24.70 \pm 0.61}$
KDE4	20.53 ± 0.34	20.54 ± 0.32	$\underline{26.61 \pm 0.16}$	$\mathbf{27.26 \pm 0.36}$
Med	19.30 ± 0.24	19.53 ± 0.28	28.31 ± 0.04	$\mathbf{29.59 \pm 0.05}$
GlobalVoices	25.10 ± 0.11	25.17 ± 0.23	25.83 ± 0.25	$\mathbf{26.03 \pm 0.13}$
WMT-News	26.93 ± 0.36	26.92 ± 0.48	$\mathbf{27.26 \pm 0.55}$	$\underline{27.23 \pm 0.12}$
TED	27.69 ± 0.05	27.85 ± 0.06	28.78 ± 0.03	$\mathbf{29.37 \pm 0.03}$
EMEA	27.81 ± 0.01	27.79 ± 0.05	29.77 ± 0.59	$\mathbf{32.38 \pm 0.01}$
ECB	29.18 ± 0.03	29.21 ± 0.04	31.18 ± 0.01	$\mathbf{33.23 \pm 0.40}$

Table 3.1: BLEU scores on meta-test split for different approaches evaluated across ten domains. Best results are highlighted in bold, results with-in two standard-deviations of the best value are underlined.

Xeon CPU, and 32GB of memory. This is due to the smaller size of each adaptation task, which requires only a handful of gradient descent steps to converge. The meta-learner needs to be trained only once and then it can be used simultaneously across all of the targeted ten domains.

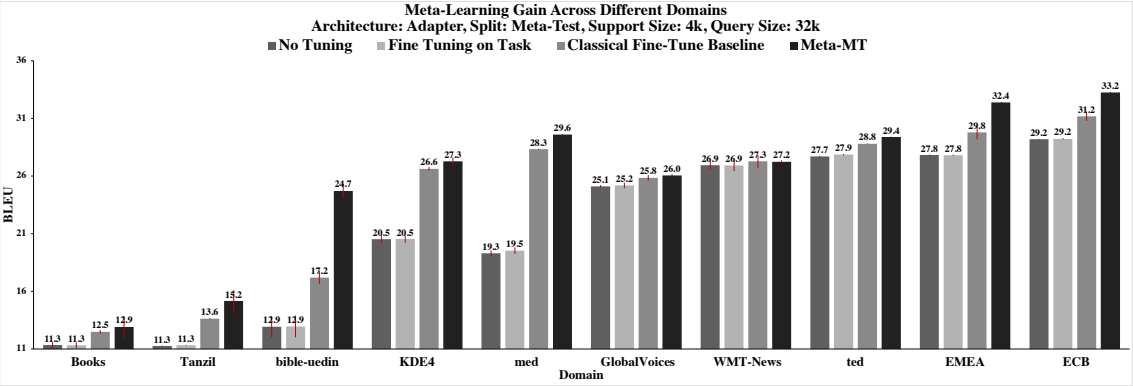


Figure 3.2: BLEU scores for different approaches evaluated across ten domains.

3.5.3 Evaluation Tasks and Metrics

Our experimental setup operates as follows: using a collection of simulated machine translation adaptation tasks, we train an NMT model f_θ using META-MT (2). This model learns to adapt faster to new domains, by fine-tuning on a tiny support set. Once f_θ

is learned and fixed, we follow the test-time behavior described in §3.4.1. We evaluate META-MT on the collection of ten different domains described in §3.5. We simulate domain adaptation problems by sub-sampling tasks with 4k English tokens for the support set, and 32k tokens for the query set. We study the effect of varying the size of the query and the support sets in §3.5.5. We use $N = 160$ tasks for the meta-training dataset $\mathcal{D}_{\text{meta-train}}$, where we sample 16 tasks from each of the ten different domains. We use a meta-validation $\mathcal{D}_{\text{meta-validation}}$ and meta-test $\mathcal{D}_{\text{meta-test}}$ sets of size 10, where we sample a single task from each domain. We report the mean and standard-deviation over three different meta-test sets. For evaluation, we use BLEU Papineni et al. (2002). We measure case-sensitive de-tokenized BLEU with SacreBLEU Post (2018). All results use beam search with a beam of size five.

3.5.4 Experimental Results

We describe our experimental results comparing the several algorithms from §3.5.1. The overall results are shown in Table 3.1 and Figure 3.2. Table 3.1 shows the BLEU scores on the meta-test dataset for all the different approaches across the ten domains. From these results we draw the following conclusions:

1. The pre-trained En-De NMT model performs well on general domains. For instance, BLEU for WMT-News [2], GlobalVoices, and ECB is at least 26 points. However, performance degrades on closed domains like Books, Quran, and Bible. [Column A].
2. Domain adaptation by fine-tuning on a single task does not improve the BLEU score. This is expected, since we are only fine-tuning on 4k tokens (i.e. $\sim 200 - 300$ sentences) [A vs B].
3. Significant leverage is gained by increasing the amount of fine-tuning data. Fine-tuning on all the available data used for meta-learning improves the BLEU score

[2] This is subset of the full test set to match the sizes of query sets from other domains

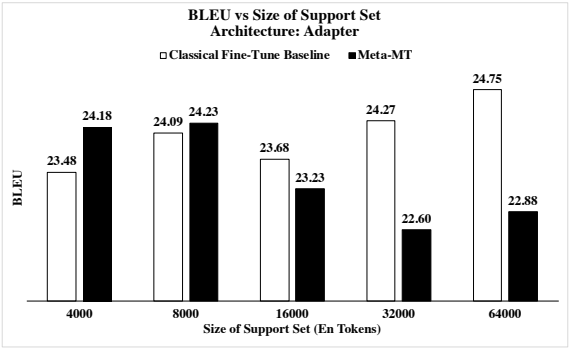


Figure 3.3: META-MT and fine-tuning adaptation performance on the meta-test set $\mathcal{D}_{\text{meta-test}}$ vs different support set sizes per adaptation task.

significantly across all domains. [B vs C]. To put this into perspective, this setup is tuned on all data aggregated from all tasks: $160 * 4k$ words which is approximately $40K$ sentences.

4. META-MT outperforms alternative domain adaptation approaches on all domains with negligible degradation on the baseline domain. META-MT is better than the non-adaptive baseline [A vs D], and succeeds in learning to adapt faster given the same amount of fine-tuning data [B vs D, C vs D]. Both **Fine-tuning on meta-train** [C] and **META-MT** [D] have access to exactly the same amount of training data, and both use the same model architecture. The difference however is in the learning algorithm. META-MT uses MAML (2) to optimize the meta-objective function in Eq 3.2. This ensures that the learned model initialization can easily be fine-tuned to new domains with very few examples.

3.5.5 Impact of Adaptation Task Size

To evaluate the effectiveness of META-MT when adapting with small in-domain corpora, we further compare the performance of META-MT with classical fine-tuning on varying amounts of training data per adaptation task. In Figure 3.3 we plot the overall adaptation performance on the ten domains when using different data sizes for the support set. In this experiment, the only parameter that varies is the size of the task support set

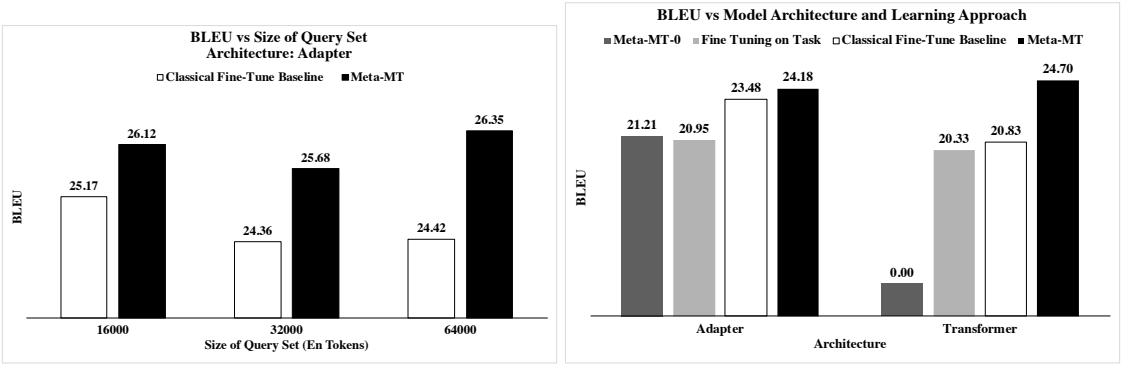


Figure 3.4: (Left) META-MT and fine-tuning adaptation performance on the meta-test set $\mathcal{D}_{\text{meta-test}}$ vs different query set sizes per adaptation task. (Right) BLEU scores reported for two different model architectures: Adapter Transformer (Left), and the Transformer base architecture (Right).

$\text{T}_{\text{support}}$. We fix the size of the query set per task to $16k$ tokens, and we vary the size of the support set from $4k$ to $64k$. To ensure that the total amount of meta-training data $\mathcal{D}_{\text{meta-train}}$ is the same, we use $N = 160$ tasks for meta-training when the support size $\text{T}_{\text{support}}$ is $4k$, $N = 80$ tasks when the support size is $8k$, $N = 40$ tasks for support size of $16k$, $N = 20$ tasks when the support size is $32k$, and finally $N = 10$ meta-training tasks when the support size is $64k$. This controlled setup ensures that no setting has any advantage by getting access to additional amounts of training data. We notice that for reasonably small size of the support set ($4k$ and $8k$), META-MT outperforms the classical fine-tuning baseline. However, when the data size increase ($16k$ to 64), META-MT is outperformed by the fine-tuning baseline. This happens because for a larger support size, e.g. $64k$, we only have access to 10 meta-training tasks in $\mathcal{D}_{\text{meta-train}}$, this is not enough to generalize to new unseen adaptation tasks, and META-MT over-fits to the training tasks from $\mathcal{D}_{\text{meta-train}}$, however, the performance degrades and does not generalize to $\mathcal{D}_{\text{meta-test}}$.

To understand more directly the impact of the query set on META-MT's performance, in Figure 3.4 we show META-MT and fine-tuning adaptation performance on the meta-test set $\mathcal{D}_{\text{meta-test}}$ on varying sizes for the query set. We fix the support size to $4k$ and vary the query set size from $16k$ to $64k$. We observe that the edge of improvement of META-MT over fine-tuning adaptation increases as we increase the size of the query set. For instance,

when we use a query set of size $64k$, META-MT outperforms fine-tuning by 1.93 BLEU points, while the improvement is only 0.95 points when the query set is $16k$.

3.5.6 Impact of Model Architecture

In our experiments, we used the Adapter Transformer architecture Bapna et al. (2019). This architecture fixes the parameters of the pre-trained Transformer model, and only adapts the feed-forward adapter module. Our model included $\sim 66M$ parameters, out of which we adapt only $405K$ (only 0.6%). We found this adaptation strategy to be more robust to meta-learning. To better understand this, Figure 3.4 shows the BLEU scores for the two different model architectures. We find that while the meta-learned Transformer architecture (Right) slightly outperforms the Adapter model (Left), it suffers from catastrophic forgetting: **META-MT-0** shows the zero-shot BLEU score before fine-tuning the task on the support set. For the Transformer model, the score drops to zero and then quickly improves once the parameters are tuned on the support set. This is undesirable, since it hurts the performance of the pre-trained model, even on the general domain data. We notice that the Adapter model does not suffer from this problem.

3.5.7 Impact of Zero-Shot Learning

To evaluate the effectiveness of META-MT when adapting to a totally unseen domain at test time, we additionally compare META-MT to classical fine-tuning in a zero-shot learning setting. In this zero-shot learning setting, we only use data from the *Tanzil* domain for meta-training, and then evaluate the meta-learned model on all of the ten domains from §3.5. This contrast with the multi-domain learning setting in §3.5.4, where we use adaptation tasks from all the ten domains at meta-training time. The results are presented in Figure 3.5. Here we see that META-MT still outperforms classical fine-tuning on seven out of the ten domains. As expected, META-MT significantly outperforms fine-tuning on the *Tanzil* domain with a $0.86(13.79 - 12.93)$ BLEU point improvement. The gains

however are less significant when compared to the multi-domain setting in §3.5.4. This is expected, Yin et al. (2019) showed that when the meta-training tasks are not mutually-exclusive, the meta-learner can ignore the task training data and learn a single model that performs all of the meta-training tasks simultaneously, but does not adapt effectively to new tasks.

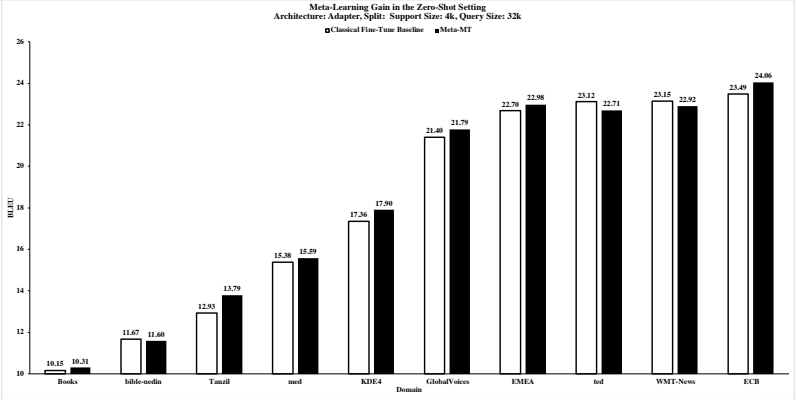


Figure 3.5: BLEU scores for META-MT (Black) vs Fine-Tuning (White) evaluated across ten domains in a zero-shot learning setting where only data from the Tanzil domain is used at training time.

3.6 Conclusion

In this chapter, we studied few-shot learning as a form of minimal supervision. We presented META-MT, a meta-learning approach for few-shot NMT adaptation. We formulated few-shot NMT adaptation as a meta-learning problem, and presented a strategy that learns better parameters for NMT systems that can be easily adapted to new domains. We validated the superiority of META-MT to alternative domain adaptation approaches. In terms of BLEU scores, META-MT outperforms alternative strategies in most domains using only a small fraction of fine-tuning data. There are several potential next steps:

1. Extending the analysis to include human evaluation for generations from the META-MT system.

2. Analyzing the syntactic and semantic divergences learned by the meta-learner when adapting to the targeted domains.

3. Providing bounds and theoretical guarantees, and understanding the theory of learning from different domains in a meta-learning setting. (Ben-David et al., 2010).

In chapter 4 we validate our thesis statement (see §1.5) by studying a different form of learning with minimal supervision: Active Learning, and present a meta-learning based algorithm for enforcing fairness parity constraints on the active learning data selection process.

Chapter 4: Promoting Fairness by Learning to Active Learn under Parity Constraints

4.1 Introduction

In chapter 3 we studied few-shot adaptation as a form of minimal supervision. In this chapter, we study a different form of supervision: active learning. In active learning an agent can interactively query an oracle to label new data points with the ground truth outputs. We propose to solve a parity-constrained active learning problem using a meta-learning approach, very much in the style of recent work on meta-learning for active learning (Konyushkova et al., 2017a; Bachman et al., 2017; Fang et al., 2017). Our algorithm, PARITY-CONSTRAINED META ACTIVE LEARNING (PANDA; see §4.3), uses data to learn a selection policy that picks which examples to have labeled under a constraint on fairness parity. The data on which it learns this selection policy is the pre-existing (possibly biased!) dataset from which it will continue learning.

Machine learning models often lead to harms due to disparities in behavior across social groups: an automated hiring system may be more likely to recommend hiring people of privileged races, genders, or age groups (Wachter-Boettcher, 2017; Giang, 2018). These disparities are typically caused by biases in historic data (society is biased); a substantial literature exists around "de-biasing" methods for algorithms, predictions, or models (, i.a.). Such approaches always assume that the training data is fixed, leading to a false choice between efficacy (e.g., accuracy, AUC) and "fairness" (typically measured by a metric of parity across subgroups (Chen et al., 2018; Kallus and Zhou, 2018)).

This is in stark contrast to how machine learning *practitioners* address disparities in model performance: they *collect more data* that's more relevant or representative of the populations of interest (Veale and Binns, 2017; Holstein et al., 2019). This disconnect leads to a mismatch between sources of bias, and the algorithmic interventions developed to mitigate them (Zarsky, 2016).

We consider a different trade-off: given a pre-existing dataset, which may have been collected in a highly biased manner, how can we manage an efficacy vs *annotation cost* trade-off under a target parity constraint? We call this problem *parity-constrained active learning*, where a maximal disparity (according to any of a number of different measures, see Table 4.1) is enforced during a data collection process. We specifically consider the case where some "starting" dataset has already been collected, distinguishing our procedure from more standard active learning settings in which we typically start from no data ((Settles, 2009), see §4.2). The goal then is to collect as little data as is needed to keep accuracy high while maintaining a constraint on parity (as a measure of fairness). As an example, consider disparities in pedestrian detection by skin tone (Wilson et al., 2019): A pedestrian detector is trained based on a dataset of 100k images, but an analysis shows that it performs significantly better at detecting people with light skin than people with dark skin. Our goal is to label few *additional* samples while achieving a high accuracy under a constraint on the disparity between these groups.[1]

To achieve this, PANDA simulates many parity-constrained active learning tasks on this pre-existing dataset, to learn the selection policy. Technically, PANDA formulates the parity-constrained active learning task as a bi-level optimization problem. The inner level corresponds to training a classifier on a subset of labeled examples. The outer level corresponds to updating the selection policy choosing this subset to achieve a desired fairness and accuracy behavior on the trained classifier. To solve this constrained bi-level optimization problem, PANDA employs the *Forward-Backward Splitting* (FBS, Lions

[1]Code: https://www.dropbox.com/sh/sbao1hdrxvgmdfw/AAC0LsyQsIxNIYxVaolLhKj_a?dl=0

and Mercier (1979); Combettes and Wajs (2005); Goldstein et al. (2014)) optimization method (also known as the proximal gradient method). Despite its apparent simplicity, FBS can handle non-differentiable objectives with possibly non-convex constraints while maintaining the simplicity of gradient-descent methods. At test time, PANDA does not require enforcing a parity constrain on the classifier used for training the selected samples, leading to a simpler and more efficient approach for enforcing the parity constrains by learning a better representation for the queried samples.

Through exhaustive empirical experiments (§4.4), we show the following:

1. PANDA is effective: it outperforms alternative active learning algorithms by a large margin under the same setting while enforcing the desired behavior on fairness.
2. PANDA is general-purpose: it learns the selection policy end-to-end and can handle a wide set of non-differentiable and non-convex constraints on fairness parity using Gumbel-Softmax reparameterization (Gumbel, 1948; Jang et al., 2016; Maddison et al., 2016) and differentiable approximations.
3. PANDA is powerful: it employs a Transformer model architecture (Vaswani et al., 2017) to represent the learned selection policy. This architecture has achieved state-of-the-art performance in many tasks including language modeling (Dai et al., 2019), machine translation (Dehghani et al., 2018), and unsupervised pre-training (Devlin et al., 2018).

4.2 Background and Related Work

Concerns about biased or disparate treatment of groups or individuals by computer systems has been raised since the 1990s Friedman and Nissenbaum (1996). Machine learning and other statistical techniques provide ample opportunity for pre-existing societal bias to be incorporated into computer systems through data, leading to a burgeoning of research studying disparities in machine learning (Abdollahi and Nasraoui, 2018; Crawford and Calo, 2016, i.a.).

METRIC	DESCRIPTION & MATHEMATICAL DEFINITION
Demographic Parity	Prediction $h(x)$ is statistically independent of the group $g(x)$ (Feldman et al., 2015): $\Delta^{\text{DP}}(h) \triangleq \max_a \mid \mathbb{E}[h(x) \mid g(x)=a] - \mathbb{E}[h(x)] \mid$
Equalized Odds	Prediction $h(x)$ is independent of the group $g(x)$ given the true label y (Hardt et al., 2016): $\Delta^{\text{EO}}(h) \triangleq \max_{a,y} \mid \mathbb{E}[h(x) \mid g(x)=a, Y=y] - \mathbb{E}[h(x) \mid Y=y] \mid$
Error-rate Balance	False positive and false negative error rates are equal across groups (Chouldechova, 2017): $\Delta^{\text{EB}}(h) \triangleq \max_{a,a',y} \mid \mathbb{E}[h(x) \mid g(x)=a, Y=y] - \mathbb{E}[h(x) \mid g(x)=a', Y=y] \mid$

Table 4.1: Three common measures of disparity for binary classification (extensions to multiclass are generally straightforward), expressed in terms of differences in expected values of predictions (where we take $h : \mathcal{X} \to \{0, 1\}$). We denote by $g(x)$ the group to which the example x belongs. In some work, disparities are taken to be *ratios* of expectations, rather than differences.

Much technical machine learning research has gone into defining what disparate treatment means formally, leading to a zoo of parity metrics (Narayanan, 2018) (see Table 4.1 for examples), proofs of their incompatibilities (Chouldechova, 2017; Kleinberg et al., 2016), and analyses of how they conform to normative notions of fairness (Srivastava et al., 2019). This has led to machine learning algorithms that optimize not just for accuracy, but rather for accuracy subject to a constraint on parity between known groups (Agarwal et al., 2018).

A parallel line of research has considered the human side of analyzing disparities in machine learning systems, including visualization (Cabrera et al., 2019), debugging (Chen et al., 2018), and needs-finding (Veale and Binns, 2017; Holstein et al., 2019). One finding of the latter is that machine learning practitioners and data scientists often have control over training data, which is not taken into account in most technical machine learning research. For instance, Holstein et al. (2019)'s results show that 78% of practitioners who had attempted to address disparities did so by trying to collect more data, despite the lack of tools that support this.

Curating more data is not a foreign concept in the machine learning research: active learning—the learning paradigm in which the learner itself chooses which examples to

have labeled next—has been studied extensively over the past five decades (Settles, 2009; Fedorov, 2013; Angluin, 1988; Cohn et al., 1994; Jiang and Ip, 2008). Most active learning approaches select samples to label based on some notion of uncertainty (e.g., entropy of predictions). Most relevant to our work are recent active learning approaches based on meta-learning (Bachman et al., 2017; Fang et al., 2017): here, instead of designing the selection strategy by hand, the selection strategy is learned based on offline, simulated active learning problems. So long as those offline problems are sufficiently similar to the target, real, active learning problem, there is hope that the learned strategy will generalize well.

We are aware of only one paper that considers active learning in the context of fairness: Fair Active Learning (FAL) by Anahideh et al. (2020). FAL uses a handselection strategy that interpolates between an uncertainty-based selection criteria, and a "fairness" criteria that estimates the impact on disparity if the label of a given point were queried (by computing expected disparity over all possible labels). FAL selects data points to be labeled to balance a convex combination of model accuracy and parity, with the trade-off specified by a hyperparameter. Empirically, Anahideh et al. (2020) showed a significant reduction in disparity while maintaining accuracy. Our setting is slightly different than FAL—we assume pre-existing data—but we compare extensively to this approach experimentally under similar conditions (§4.4).

4.3 Problem Definition and Proposed Approach

In this section we define *parity-constrained active learning* and describe our algorithm, PANDA.

4.3.1 Problem Definition: Parity-Constrained Active Learning

We consider the following model. We have collected a dataset of N *labeled* examples, $D^0 = (\boldsymbol{x}_n, y_n)_{n=1}^N$ over an input space $\mathcal{X}$ (e.g., images) and output space $\mathcal{Y}$ (e.g., pedestrian bounding boxes), and have access to a collection of M-many *unlabeled* examples, $U = (\boldsymbol{x}_m)_{m=1}^M$. Each input example $x \in \mathcal{X}$ is associated with a unique group $g(x)$ (e.g., skin tone). We fix a hypothesis class $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ and learning algorithm $\mathcal{A}$ that maps a labeled sample D to a classifier $h \in \mathcal{H}$. Finally, we have a loss function $\ell(y, \hat{y}) \in \mathbb{R}^{\geq 0}$ (e.g., squared error, classification error, etc.) and a target *disparity metric*, $\Delta(h) \in \mathbb{R}^{\geq 0}$ (such as those in Table 4.1). The goal is to label as few images from U as possible to learn a classifier h with high accuracy subject to a constraint that $\Delta(h) < \rho$ for a given threshold $\rho > 0$. We assume access to a (small) validation set V of labeled examples (which can be taken to be a subset of D). We will denote population expectations and disparities by $\mathbb{E}$ and Δ, respectively, and their estimates on a finite sample by $\hat{\mathbb{E}}_A$ and $\hat{\Delta}_A$, where A is the sample.

The specific interaction model we assume is akin to standard active learning with labeling budget B:

1: train the initial classifier on the pre-existing dataset: $h^0 = \mathcal{A}(D^0)$.
2: **for** round $b = 1 \ldots B$ **do**
3: generate categorical probability distribution $Q = \pi(h^{b-1}, U)$ over U using policy π.
4: sample an unlabeled example $\boldsymbol{x} \sim Q$, query its label y, and set $D^b = D^{b-1} \cup \{(\boldsymbol{x}, y)\}$.
5: train/update classifier: $h^b = \mathcal{A}(D^b)$.
6: **end for**
7: **return** classifier h^B, validation loss $\hat{\mathbb{E}}_V \ell(y, h^B(\boldsymbol{x}))$ and validation disparity $\hat{\Delta}_V(h^B)$.

Under this model, the active learning strategy is summarized in the example selection policy π.[2] The goal in parity constraint actively learning is to construct a π with minimal expected loss subject to the constraint that $\Delta(h) < \rho$.

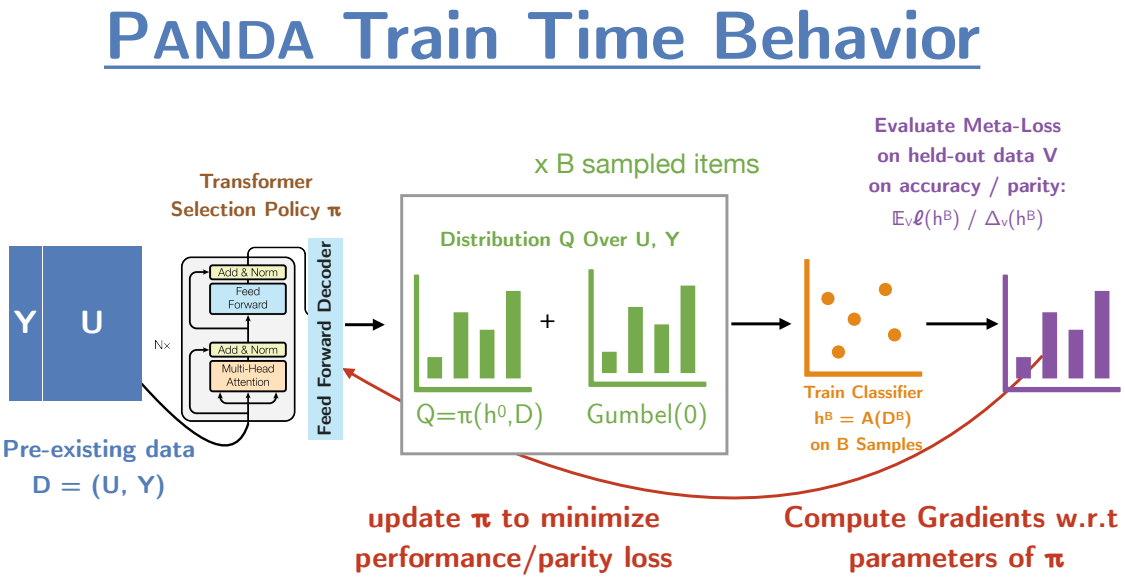


Figure 4.1: Train time behavior of PANDA. The figure shows a training step of PANDA. At training time, we have access to the labels Y for simulating the parity-constrained active learning setting. We model the selection policy π using a transformer encoder followed by a feed-forward decoder. Each layer in the transformer encoder has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. The model is trained end-to-end where a Gumbel-Softmax reparameterization trick is used to avoid back-propagating through the sampling procedure from the distribution Q.

4.3.2 PANDA: Learning to Actively Learn under Parity Constraints

We develop a meta-learning approach, PANDA (Figure 4.1), to address the parity-constrained active learning problem: the selection policy π is trained to choose samples that, if labeled, are likely to optimize accuracy subject to a parity constraint. This learning happens on D itself: by simulating many possible ways additional data could be selected

[2]For example, margin-based active learning (Roth and Small, 2006) can be realized by setting $\pi(h, U)$ to assign a $Q(\boldsymbol{x}) = \mathbf{1}[\boldsymbol{x} = \boldsymbol{x}^\star]$ where $\boldsymbol{x}^\star = \operatorname{argmin}_{\boldsymbol{x} \in U} |h(\boldsymbol{x})|$, where h returns the margin.

on the historic data, PANDA learns how to select additional examples, even if D itself was sampled in a biased manner.

To learn π, we construct a distribution of meta-training tasks, $\mathfrak{M}$; samples $(L, V) \sim \mathfrak{M}$ consist of a labeled dataset L (to simulate unlabeled data U) and a validation set V. We form $\mathfrak{M}$ by repeatedly reshuffling D, and produce a finite sample of meta-training tasks $\mathfrak{D}$ i.i.d. from $\mathfrak{M}$. The meta-learning problem is then to optimize π on $\mathfrak{D}$ to achieve high accuracy subject to a constraint on disparity. We begin by first writing the parity-constrained problem according to its characteristic function:

$$\hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H} \,:\, \hat{\Delta}_V(h) < \rho} \hat{\mathbb{E}}_V \, \ell(\boldsymbol{x}, h(\boldsymbol{x})) \quad \Longleftrightarrow \quad \hat{h} \in \operatorname*{argmin}_{h \in \mathcal{H}} \hat{\mathbb{E}}_V \, \ell(\boldsymbol{x}, h(\boldsymbol{x})) + \chi_{\hat{\Delta}, \rho, V}(h) \quad (4.1)$$

where $\chi_{\Delta,\rho,V}(h) = 0$ if $\hat{\Delta}_V(h) < \rho$ and $+\infty$ otherwise; for brevity we write $\chi(h)$ when $(\hat{\Delta}, \rho, V)$ is clear from context. Under reasonable assumptions, both minimizers are finite.

Given this, the meta-learning optimization problem is:

$$\min_{\pi \in \Pi} \hat{\mathbb{E}}_{(L,V) \sim \mathfrak{D}} \left[\hat{\mathbb{E}}_V \ell(\boldsymbol{x}, h_L^\pi(\boldsymbol{x})) + \chi(h_L^\pi) \right] \text{ where } h_L^\pi = \textsc{ActiveLearnSim}(\mathcal{A}, D, L, \pi) \quad (4.2)$$

Here, $\textsc{ActiveLearnSim}(\mathcal{A}, D, L, \pi)$ is the interactive algorithm in §4.3.1, where U is taken to be L (with labels hidden) and when a label is queried, it is retrieved from L.

When $\mathcal{A}$ is, itself, an optimizer—as it is in most machine learning settings—then formulation Eq 4.2 is a constrained bilevel optimization problem. The outer optimization is over the sampling policy π, and the inner optimization is over the optimization over h in ACTIVELEARNSIM. We assume that $\mathcal{A}$ can be written as a computational graph, in which case the outer objective can be optimized by unrolling the computational graph of $\mathcal{A}$. This introduces second-order gradient terms, but remains computationally feasible so long as

the unrolled graph of $\mathcal{A}$ is not too long: we ensure this by only running a few steps of SGD inside $\mathcal{A}$ and using a simple hypothesis class for $\mathcal{H}$.

There remain two challenges to solve Eq 4.2. The first is to address the discontinuous nature of the characteristic function χ; we use forward-backward splitting to address this. The second is that the unrolling of ACTIVELEARNSIM yields a computational graph that has stochasticity (due to the sampling of unlabeled examples); we use the Gumbel reparameterization trick to address this.

Forward-Backward Splitting (FBS) is a class of optimization methods (Lions and Mercier, 1979), which provide the simplicity of gradient descent methods while being able to enforce possibly non-differentiable constraints. In FBS, the objective is separated into a differentiable part $f(x)$ and an arbitrary (not even necessarily smooth) part $g(x)$. The algorithm operates iteratively by first taking a gradient step just with respect to f to an intermediate value: $x' = x - \eta \nabla f(x)$. Next, it computes a proximal step that chooses the next iterate x to minimize $\eta g(x) + ||x - x'||^2/2$. When f is convex, FBS converges rapidly; for non-convex problems (like Eq 4.2), theoretical convergence rates are unknown, but the algorithm works well in practice.

To apply FBS to our problem, we choose our policy class Π to be a differentiable neural network (see § 4.3.3). We set f to be the expected loss term in Eq 4.2, and g to be the characteristic function on the disparity. The gradient step with respect to f can be computed by automatic differentiation of the unrolled computational graph. The proximal step requires projecting onto χ; for complex Π there is no closed-form solution; instead, we run a separate approximate projection step by running several stops of gradient descent on a smoothed version of χ, which takes values 0 when the constraint is satisfied, and takes value $\hat{\Delta}_V(h)$ otherwise. This remains non-continuous, but (sub)differentiable—empirically, this approximate projection always finds an iterate that satisfies the original constraint.

3 Parity-constrained Active Learning via PANDA

Input: pre-existing datasets D, budget B, loss function ℓ, disparity metric Δ, threshold ρ, meta-learning learning rate schedule $\langle \eta^k \rangle_{k \geq 1}$, and inner learning rate η

Output: Selection policy π

1: Initialize selection policy $\pi(\cdot; \theta^0)$ parameterized by θ^0
2: **for** iteration $k = 1 \ldots$ convergence **do**
3: Split D into pool L and validation set V
4: $\hat{\theta}^{k+1} = \theta^k - \eta^k \nabla_\theta \hat{\mathbb{E}}_V \ell\big(y, \textsc{ActiveLearnSim}(\mathcal{A}, D, L, \pi(\cdot; \theta^k)(\boldsymbol{x})\big)$
5: $\theta^{k+1} = \operatorname{argmin}_\theta \eta^k \chi_{\hat{\Delta}, \rho, V}\big(\textsc{ActiveLearnSim}(\mathcal{A}, D, L, \pi(\cdot; \theta))\big) + 1/2||\theta - \hat{\theta}^{k+1}||^2$
6: **end for**
7: **return** $\pi(\cdot; \theta^{\text{final}})$

8: **function** $\textsc{ActiveLearnSim}(\mathcal{A}, D, L, \pi)$
9: Let $\langle \boldsymbol{x}_i, y_i \rangle_{i=1}^{|L|}$ be an indexing of L
10: **for** $b = 1 \ldots B$ **do**
11: set $\tilde{Q}_i = \pi(h^{b-1}, \boldsymbol{x}_i) + \textsc{Gumbel}(0)$ for all i and pick $j = \arg\max_i \tilde{Q}_i$
12: take (a/several) gradient step(s) of the form: $h^b = h^{b-1} - \eta \nabla_h \ell(y_j, h(\boldsymbol{x}_j))$
13: **end for**
14: return h^B
15: **end function**

Gumbel Reparameterization is a generic technique to avoid back-propagating through stochastic sampling nodes in the computational graph (Gumbel, 1948; Jang et al., 2016; Kool et al., 2019; Maddison et al., 2016). This trick allows us to sample from the categorical distribution Q by independently perturbing the log-probabilities Q_i with Gumbel noise and finding the largest element, thus enabling end-to-end differentiation through ACTIVELEARNSIM, so long as $\mathcal{A}$ is differentiable.

The **Full Training Algorithm** for PANDA is summarized in 3. Following the Forward-Backward Splitting template, PANDA operates in an iterative fashion. Over iterations, PANDA simulates a parity-constrained active learning setting for the current model parameters θ^k. Line 4 performs a simple forward gradient descent step to maximize the classifier performance. This step begins at iterate θ^k, and then moves in the direction of the (negative) gradient of the performance loss, which is the direction of steepest descent. Line 5 is the proximal (or backward) step, which enforces the parity constraint; this works

even when the parity metric is non-differentiable. In both the gradient descent step and the proximal step, PANDA performs bilevel optimization. For example, the gradient step is a gradient with respect to the parameters of the selection policy, of the computational graph defined by ACTIVELEARNSIM. That function, itself, performs an optimization of the classifier h.

4.3.3 Network Structure of Selection Policy

The selection policy π takes as input the current classifier h and unlabeled dataset U, and produces a distribution Q over elements of U. We explore policies that are *agnostic* to changes in h, meaning that Q at round b is identical for all b. This introduces a limitation that π cannot directly adapt to changes in h; however, since π is optimized end-to-end, we empirically found this to be a minor limitation. A significant advantage of this choice is that it means that ACTIVELEARNSIM can be unrolled much more easily: the simple Gumbel softmax can be replaced with Gumbel-top-B (Vieira, 2014; Kool et al., 2019) and unrolled in a single step, rather than a sequence of B-many steps.

Because π must effectively make all selections in a single step, it is important that π consider each $\boldsymbol{x}$ in the context of all other items in U, and not consider each $\boldsymbol{x}$ individually. We implement this using a Transformer architecture (Vaswani et al., 2017), in which a self-attention mechanism essentially combines information across different $\boldsymbol{x}$s in U. Specifically, we treat the examples in U as an unordered sequence as input to the Transformer encoder[3]. The Transformer architecture employs several layers of self-attention across U with independent feed-forward networks for each position. The final layer of the Transformer can be interpreted as a contextual representation for each $\boldsymbol{x} \in U$, where the context is "the rest of U." We use a final linear softmax layer to map these contextual representations to the probability distribution Q.

[3]Recall that although Transformers are typically used for *ordered* problems like language modeling (Dai et al., 2019) and machine translation (Dehghani et al., 2018), this is not how they "naturally" work: ordered inputs to Transformers require additional "position" tags.

Because this model architecture is flexible, it is also data-hungry, and training all of its parameters based just on a small set of B examples is unlikely to be sufficient. This is where the initial dataset D^0 comes in: we pretrain the parameters of the Transformer on D^0 and use the B-many actively selected samples to fine-tune the final layer, thus keeping the required sample complexity small.

4.4 Experiments

We conduct experiments in the standard active learning manner: pretend that a labeled dataset is actually unlabeled, and use its labels to answer queries. Experimentally, given a complete dataset, we first split it 50/50 into meta-training and meta-testing sections. We use meta-training to pretrain the Transformer model (see §4.3.3), and also to train PANDA. All algorithms us the same Transformer representation. The meta-testing section is split again 50/50 into the "unlabeled" set and the test set.

Picking a good dataset for parity-constrained active learning is challenging: it needs to contain a protected attribute, be sufficiently large that an active sample from unlabeled portion is representative (i.e., as the size of the sample approaches the size of the unlabeled data, all algorithms will appear to perform identically), and be sufficiently rich that learning does not happen "too quickly."

We considered three standard datasets: COMPAS (Angwin et al., 2016), Adult Income (Dua and Graff, 2017), and Law School (Wightman, 1998). Law School has only two features and we found only a few examples are needed to learn; and COMPAS we found to be too small[4]. This left only the Adult Income dataset for experiments.

[4] COMPAS consists of just under $8k$ samples, so after two splits, each set contains only $2k$ samples. We anticipated that this would be too small for three reasons. First, with a budget $B = 400$, many algorithms will end up sampling very similar sets, resulting in difficulties telling approaches apart. Second, we found that after pre-training, 15–20 completely random samples suffice to learn a classifier that is as good as one trained on all the remaining data. Nonetheless, we performed experiments on COMPAS for all baselines and found that while PANDA can fit the meta-training data well, and this generalizes well with respect to *loss*, it has poor generalization with respect to *disparity*. We also ran Fairlearn (described below) on this dataset randomly sampled subsets of the training data, and found that, while it eventually is able to achieve a target

This dataset consists of $48,842$ examples and 251 features (with one-hot encodings of categorical variables) and the binary prediction task is whether someone makes more than 50k per year, with binary race as the group attribute (white versus non-white).

4.4.1 Baseline Active Learning Approaches

Our experiments aim to determine how PANDA compares to alternative active learning strategies, including those that explicitly take disparity into account as well as those that do not. Among those that do not consider disparity, we compare to:

Random Sampling – select examples to label uniformly at random.

Margin Sampling – uncertainty-based active learning that selects the example closest to the current decision boundary (Roth and Small, 2006).

Entropy Sampling – uncertainty-based active learning that selects the example with highest entropy of predicted label (Shannon, 1948; Settles, 2009).

We also consider alternate approaches that take groups and/or disparity into account explicitly.

Group Aware Random Sampling – select examples to label uniformly at random, restricted to the group on which worse performance is achieved by h^0.[5]

Fair Active Learning (FAL) – the fair active learning approach described in § 4.2 that optimizes an interpolation between Entropy Sampling and expected disparity.

Fairlearn – select examples to label uniformly at random, and then run fairlearn to train a classifier to optimize accuracy subject to a parity constraint (Agarwal et al., 2018).

disparity level of 0.04 once $B = 400$, at any point with $B < 300$ the test-time disparity is significantly larger. We therefore drop COMPAS from consideration; it seems ill-suited to a warm-start active learning paradigm.

[5] Closely related to active learning in domain adaptation (Shi et al., 2008; Rai et al., 2010; Wang et al., 2014).

Fairlearn+AL – combine uncertainty-based active learning with a fairlearn classifier, select examples closest to the decision boundary to label, and then run fairlearn to train a classifier to optimize accuracy subject to a parity constraint.

4.4.2 Implementation Details and Hyperparameter Tuning

We use a Transformer Model (Vaswani et al., 2017) implemented in PyTorch (Paszke et al., 2019). We use the standard transformer encoder with successive encoder layers. Each layer contains a self-attention layer, followed by a fully connected feed-forward layer. We use the feed-forward layer for decoding, where we sample B items from the predicted probability distribution in a single decoding step. To ensure a fair-comparison among all approaches, we use the same Transformer architecture as a feature extractor for all approaches. This ensures that PANDA has no additional advantage by observing more training data.

The model is optimized with Adam (Kingma and Ba, 2014). We optimize all hyper-parameters with the Bayes search algorithm implemented in comet.ml using an adaptive Parzen-Rosenblatt estimator. We search for the best parameters for learning rate (10^{-2} to 10^{-7}), number of layers in the transformer encoder $(1, 3, 5)$, embedding dimensions for the encoder hidden-layer $(16, 32, 64)$, as well as the initial value for the Gumbel-Softmax temperature parameter (1 to 10^{-6}) which is then learned adaptively as meta-training progresses. The sampled examples are used to train a linear classifier, again we optimize the hyper-parameters for the learning rate and batch size using Bayes search. For active learning model selection, we sweep over parameters using the random sampling active learning method. We found that hyper-parameters for random sampling work well for other alternative approaches too. We scale all the features to have a mean zero and unit standard deviation.

Computationally, at test time, PANDA is the fastest of all the active learning algorithms we compare to from §4.4.1 with matching runtime performance to random sampling.

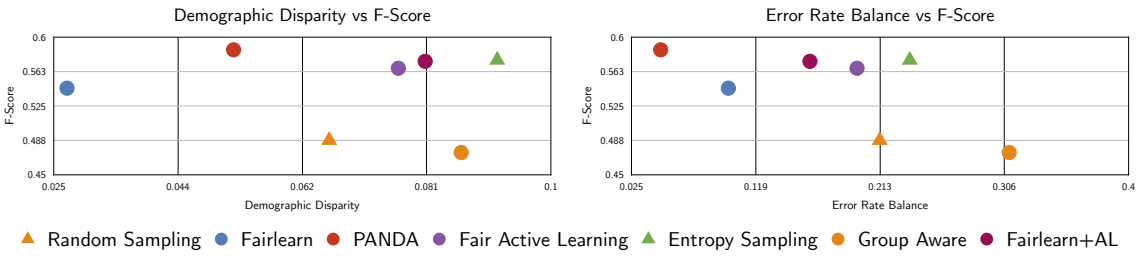


Figure 4.2: (Left) A scatterplot of demographic disparity versus F-score for a fixed budget $B = 400$, for PANDA and baseline approaches. (Right) A similar scatterplot for error rate balance versus F-score. In both cases, the upper-left is optimal behavior. Overall, we see that fairlearn and PANDA are the most competitive algorithms, with flipped behavior with respect to disparity on the two metrics. Dotted curves are algorithms unaware of parity/groups; solid lines are algorithms that are.

This is because at test time we only need a single forward pass through the selection policy to select the B samples to label. Entropy sampling requires computing the entropy in every time step. Fairlearn is much slower as the learning reduction refits a mixture of expert models with different weights. Fair Active Learning is the slowest approach as it needs to compute the "expected fairness" that requires learning a new classifier for every data point in the pool of unlabeled data. For meta-training, learning the policy for PANDA converged after few hours of training on a GeForce GTX 1080 Ti GPU, 2.1GHz Intel Xeon CPU, and 32GB of memory.

4.4.3 Evaluation Metrics and Results

We evaluate the performance of the learned classifiers using the overall F-score on the evaluation set V, and report violations for parity-constrains in terms of demographic disparity and error rate balance (Table 4.1), as these account for different ends of the constrained spectrum of parity metrics. In order to set trade-off parameters (the convex combination α for FAL and the constraints for fairlearn and PANDA), we first run FAL with several different trade-off parameters to find a value for α large enough that disparity matters but small enough that a non-zero F-score is possible. All results are with $\alpha = 0.6$. We then observed the final disparity for FAL of 0.08 and set a constraint for PANDA

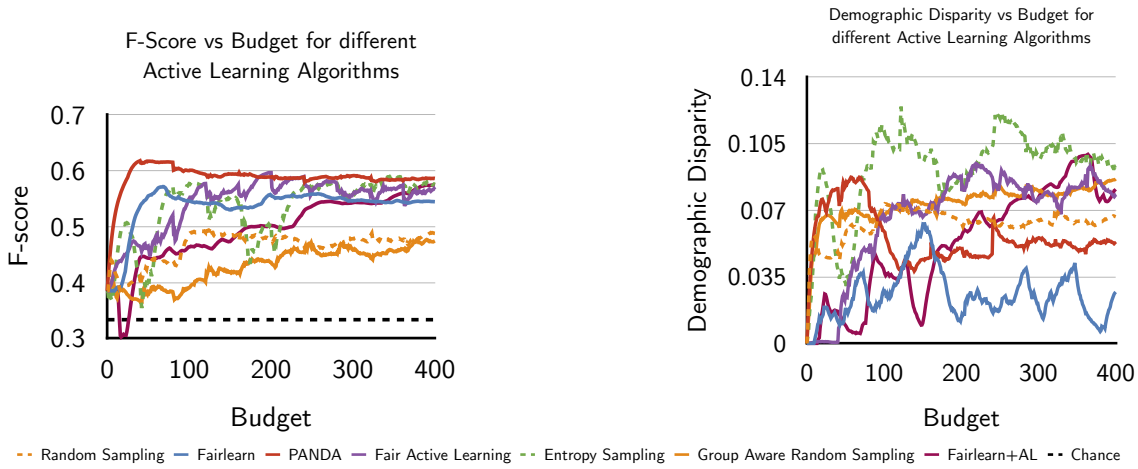


Figure 4.3: Learning curves for all algorithms, with (Left) budget (x-axis) versus F-score (y-axis) and (Right) budget (x-axis) versus demographic disparity (y-axis). The constraint value for fairlearn and PANDA is 0.04. Overall, we see that PANDA and fairlearn are able to (approximately) achieve the target parity, with PANDA achieving a higher F-score even than FAL (which has higher disparity). The black dotted line shows the F-score for a random classifier. The recall is always 0.5, since only 25% of the samples have a positive label, the precision for the random classifier is 0.25, leading to an F-score of 0.33. This random classifier has zero disparity.

and FAL of half of that: 0.04. This choice was made to ensure that FAL has an overall advantage over PANDA.

The main results are shown in Figure 4.2, where we consider performance for a fixed budget. Here, we first observe (unsurprisingly) that the baselines that do not take parity into account (Random Sampling and Entropy Sampling) do quite poorly (we do not plot margin-based sampling as it was dominated by Entropy sampling in all experiments). For example, while entropy sampling gets a very high F-score, it has quite poor disparity. Somewhat surprisingly, group-aware random sampling does worse in terms of disparity than even plain random sampling. FAL is able to achieve higher accuracy than random sampling, but, again, it's disparity is no better despite the fact that it explicitly optimizes for the trade-off. Finally, fairlearn and PANDA dominate in terms of the trade-off, with PANDA achieving higher accuracy, better error rate balance, but worth demographic disparity.

We also wish to consider the dynamic nature of these algorithms as they collect more data. In Figure 4.3, we plot budget versus f-score and disparity for a fixed parity

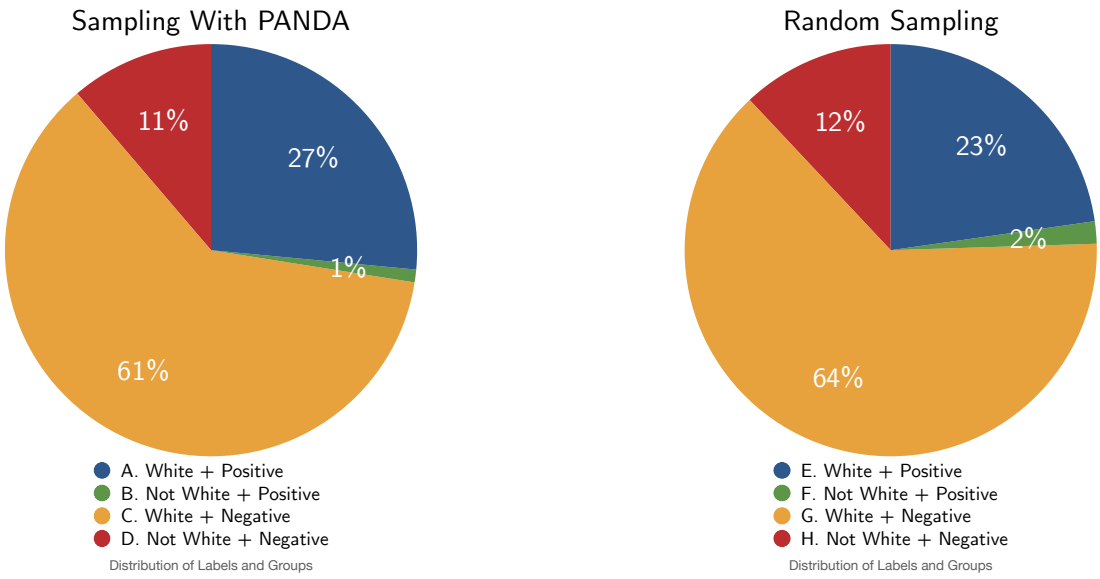


Figure 4.4: Distribution of labels (+ vs -) and groups (white vs not white) for samples selected by PANDA (left) and random sampling (right).

constraint of 0.04. Unsurprisingly, we see that entropy sampling outperforms random sampling (in F-score), though they perform essentially the same for disparity. We also see a clear trade-off in FAL between F-score (goes up as the budget increases) and disparity (also goes up).

Here, we see that both fairlearn and PANDA are able to keep the disparity low (after an initial peak for PANDA). There is a generalization gap between PANDA's training disparity (which always exactly satisfies the 0.04 constraint) and its validation disparity, which is somewhat higher, as anticipated by concentration bounds on disparity like those of Agarwal et al. (2018). The initial peak in disparity (where it does not satisfy the constraint) for PANDA is not surprising: it is trained end-to-end to pick a good sample of 400 points; it is not optimized for smaller budgets. Similarly, in terms of F-score, PANDA achieves a very high initial F-score, essentially a zero-shot learning type effect. However, as it lowers the disparity, the F-score also drops slightly. In all cases, the "Fairlearn+AL" baseline achieves better F-score in comparison to the "Fairlearn" baseline, however, this comes at the expense of both the demographic disparity and the error rate balance.

4.4.4 Distribution of Labels and Groups for Samples Selected by PANDA

To further understand the behavior of PANDA, Figure 4.4 shows the distribution of labels ("positive" versus "negative") as well as groups ("white" versus "not white") for samples selected by PANDA (left) in contrast to the 400 samples selected via random sampling (right). From this figure we draw the following conclusions:

1. PANDA queries more "positive" labeled examples (A+B versus E+F): unsurprisingly, PANDA learns to query more positive examples in contrast to random sampling (28% vs 25%). PANDA learns to sample the more informative examples to query in a highly imbalanced dataset where only 25% of the examples have positive labels.

2. This comes at the expense of sampling "negatively" labeled examples (C+D vs G+H): PANDA samples less examples with negative labels, only 72% vs 76% for random sampling.

3. Percentage of examples sampled from the marginalized group drops slightly (B+D vs F+H): similar to the behavior we observe with the "Marginalized" group sampling baseline, selecting more samples from the marginalized group does not necessarily lead to better disparity. Instead, PANDA relies more on the feature representation learned by the Transformer encoder to represents samples that lead to learning a classifier satisfying the parity constrains, regardless of the demographic group for the selected samples.

4.5 Broader Impacts

The motivation of this work is precisely to have positive broader impacts, by giving machine learning practitioners who care about fairness in machine learning another tool in their toolbox to build models with fewer disparities. Our primary target stakeholder population is such machine learning practitioners and data scientists. Secondarily, as that

primary stakeholder population builds and deploys algorithms, those who are impacted by those algorithms through direct or indirect use will, we hope, suffer fewer disparities as a result.

There are several risks. The first is a false sense of security. For instance, we do not prove formally that this approach is guaranteed to work in all cases, and our empirical results are limited to a small number of tests over a single dataset. On the positive side, Agarwal et al. (2018) prove a generalization bound for disparity that applies to our algorithm (as well as any other algorithm); thus, so long as practitioners properly test the resulting disparities of their models, they can consult these generalization bounds to get estimates of worst case behavior.

A second risk is around, if deployed, how the new data is collected. We have seen news stories recently around unethical practices for data collection. Any additional labeling that is performed as a result of running this or similar algorithms should be done consistent with standard ethical guidelines for data collection.

Overall, while there are real concerns about how this technology might be deployed, our hope is that the positive impacts outweigh the negatives, specifically because standard best-use practices should mitigate most of the risks.

4.6 Discussion, Limitations and Conclusion

In this chapter, we presented PANDA, a meta-learning approach for learning to active learn under parity constraints, motivated by the desire to build an algorithm to mitigate unfairness in machine learning by collecting more data. We have seen that empirically PANDA is effective experimentally, even in a setting in which it essentially has to choose all 400 points to label at once, rather than one at a time. An obvious direction of future work is to incorporate features of the underlying classifier into the selection policy as well as increasing the capacity of the transformer decoder; the major challenge here is the computational expense of unrolling the corresponding computational graph.

One major advantage of PANDA over all other alternatives is that in principle it does not need access to group information at test time. So long as it can be trained with group information available (for measuring disparities on the meta-test data), there is nothing in the algorithm that requires this information at test time. The only other setting in which this is possible is FAL with demographic disparity (precisely because demographic disparity does not need access to *labels*). Exploring this experimentally is a potential next step. Finally, there is the broader question of: how does one know what is the right intervention to mitigate disparities? Should we constrain our classifier? Should we collect more data? More features? Change the architecture? These are all important questions that are only beginning to be explored (Chen et al., 2018; Galhotra et al., 2017; Udeshi et al., 2018; Angell et al., 2018).

Other directions for future research include modeling annotator agreement and disagreement when labeling samples (Donmez and Carbonell, 2008; Yan et al., 2011; Zhang and Chaudhuri, 2015), as well as exploring diversity sampling for active learning, where the goal here is to cluster points based on diversity in feature distribution or representation and then sample examples from each cluster. This minimizes the cost of context switching between examples for the annotators (Sener and Savarese, 2017; Ash et al., 2019; Yuan et al., 2020).

In chapter 5, we study the "contextual bandit" setting as a different form of learning with minimal supervision to validate our claim in the thesis statement in § 1.5. We present a meta-learning algorithm that effectively learns better exploration strategies in this minimally supervised setting.

Chapter 5: Meta-Learning for Contextual Bandit Exploration

5.1 Introduction

In chapter 3 and chapter 4 we studied few-shot adaptation and active learning as forms of providing a learning agent with minimal supervision. In this chapter, we investigate a different form of supervision: contextual bandits. In a contextual bandit problem, an agent attempts to optimize its behavior over a sequence of rounds based on limited feedback (Kaelbling, 1994; Auer, 2003; Langford and Zhang, 2008). In each round, the agent chooses an action based on a context (features) for that round, and observes a reward for that action but no others (§5.2). Contextual bandit problems arise in many real-world settings like online recommendations and personalized medicine. As in reinforcement learning, the agent must learn to balance *exploitation* (taking actions that, based on past experience, it believes will lead to high instantaneous reward) and *exploration* (trying actions that it knows less about).

In this chapter, we present a meta-learning approach to automatically learn a good exploration mechanism from data. To achieve this, we use synthetic supervised learning data sets on which we can simulate contextual bandit tasks in an offline setting. Based on these simulations, our algorithm, MÊLÉE (MEta LEarner for Exploration)[1], learns a good heuristic exploration strategy that should ideally generalize to future contextual bandit problems. MÊLÉE contrasts with more classical approaches to exploration (like ϵ-greedy or LinUCB; see §5.5), in which exploration strategies are constructed by expert

[1] **Code release:** the code is available online https://www.dropbox.com/sh/dc3v8po5cbu8zaw/AACu1f_4c4wlZxD1e7W0KVZ0a?dl=0

algorithm designers. These approaches often achieve provably good exploration strategies in the worst case, but are potentially overly pessimistic and are sometimes computationally intractable.

At training time (§ 5.2.3), MÊLÉE simulates many contextual bandit problems from fully labeled synthetic data. Using this data, in each round, MÊLÉE is able to counterfactually simulate what would happen under all possible action choices. We can then use this information to compute regret estimates for each action, which can be optimized using the AggreVaTe imitation learning algorithm (Ross and Bagnell, 2014). Our imitation learning strategy mirrors that of the meta-learning approach of Bachman et al. (2017) in the active learning setting. We present a simplified, stylized analysis of the behavior of MÊLÉE to ensure that our cost function encourages good behavior (§5.3). Empirically, we use MÊLÉE to train an exploration policy on only synthetic datasets and evaluate the resulting bandit performance across three hundred (simulated) contextual bandit tasks (§5.4.4), comparing to a number of alternative exploration algorithms, and showing the efficacy of our approach (§5.4.7).

5.2 Meta-Learning for Contextual Bandits

Contextual bandits is a model of interaction in which an agent chooses actions (based on contexts) and receives immediate rewards for that action alone. For example, in a simplified news personalization setting, at each time step t, a user arrives and the system must choose a news article to display to them. Each possible news article corresponds to an action a, and the user corresponds to a context x_t. After the system chooses an article a_t to display, it can observe, for instance, the amount of time that the user spends reading that article, which it can use as a reward $r_t(a_t)$. The goal of the system is to choose articles to display that maximize the cumulative sum of rewards, but it has to do this without ever being able to know what the reward would have been had it shown a different article a'_t.

Formally, we largely follow the setup and notation of Agarwal et al. (2014). Let $\mathcal{X}$ be an input space of contexts (users) and $[K] = \{1, \ldots, K\}$ be a finite action space (articles). We consider the statistical setting in which there exists a fixed but unknown distribution $\mathcal{D}$ over pairs $(x, \boldsymbol{r}) \in \mathcal{X} \times [0,1]^K$, where $\boldsymbol{r}$ is a vector of rewards (for convenience, we assume all rewards are bounded in $[0, 1]$). In this setting, the world operates iteratively over rounds $t = 1, 2, \ldots$. Each round t:

1. The world draws $(x_t, \boldsymbol{r}_t) \sim \mathcal{D}$ and reveals context x_t.
2. The agent (randomly) chooses action $a_t \in [K]$ based on x_t, and observes reward $r_t(a_t)$.

The goal of an algorithm is to maximize the cumulative sum of rewards over time. Typically the primary quantity considered is the *average regret* of a sequence of actions $a_1, \ldots, a_T$ to the behavior of the best possible function in a prespecified class $\mathcal{F}$:

$$\text{Reg}(a_1, \ldots, a_T) = \max_{f \in \mathcal{F}} \frac{1}{T} \sum_{t=1}^{T} \Big[r_t(f(x_t)) - r_t(a_t) \Big] \tag{5.1}$$

An agent is call *no-regret* if its average regret is zero in the limit of large T.

5.2.1 Policy Optimization over Fixed Histories

To produce a good agent for interacting with the world, we assume access to a function class $\mathcal{F}$ and to an *oracle policy optimizer* for that function class. For example, $\mathcal{F}$ may be a set of single layer neural networks mapping user features (e.g., IP, browser, etc.) $x \in \mathcal{X}$ to predicted rewards for actions (articles) $a \in [K]$, where K is the total number of actions. Formally, the observable record of interaction resulting from round t is the tuple $(x_t, a_t, r_t(a_t), p_t(a_t)) \in \mathcal{X} \times [K] \times [0,1] \times [0,1]$, where $p_t(a_t)$ is the probability that the agent chose action a_t, and the full history of interaction is $h_t = \langle (x_i, a_i, r_i(a_i), p_i(a_i)) \rangle_{i=1}^{t}$. The oracle policy optimizer, POLOPT, takes as input a *history* of user interactions with the news recommendation system and outputs an $f \in \mathcal{F}$ with low expected regret.

A standard example of a policy optimizer is to combine inverse propensity scaling (IPS) with a regression algorithm (Dudik et al., 2011). Here, given a history h, each tuple (x, a, r, p) in that history is mapped to a multiple-output regression example. The input for this regression example is the same x; the output is a vector of K costs, all of which are zero except the a^{th} component, which takes value r/p. For example, if the agent chose to show to user x article 3, made that decision with 80% probability, and received a reward of 0.6, then the corresponding output vector would be $\langle 0, 0, 0.75, 0, \ldots, 0\rangle$. This mapping is done for all tuples in the history, and a supervised learning algorithm on the function class $\mathcal{F}$ is used to produce a low-regret regressor f. This is the function returned by the policy optimizer.

IPS has this nice property that it is an unbiased estimator; unfortunately, it tends to have large variance especially when some probabilities p are small. In addition to IPS, there are several standard policy optimizers that mostly attempt to reduce variance while remaining unbiased: the direct method (which estimates the reward function from given data and uses this estimate in place of actual reward), the double-robust estimator, and multitask regression. In our experiments, we use the direct method because we found it best on average, but in principle any could be used.

5.2.2 Test Time Behavior of MÊLÉE

In order to have an effective approach to the contextual bandit problem, one must be able to both optimize a policy based on historic data and make decisions about how to explore. After all, in order for the example news recommendation system to learn whether a particular user is interested in news articles on some topic is to try showing such articles to see how the user responds (or to generalize from related articles or users). The exploration/exploitation dilemma is fundamentally about long-term payoffs: is it worth trying something potentially suboptimal *now* in order to learn how to behave better in the future? A particularly simple and effective form of exploration is ϵ-greedy: given a function

f output by POLOPT, act according to $f(x)$ with probability $(1-\epsilon)$ and act uniformly at random with probability ϵ. Intuitively, one would hope to improve on such a strategy by taking more (any!) information into account; for instance, basing the probability of exploration on f's uncertainty.

Our goal is to *learn* how to explore from experience. The training procedure for MÊLÉE will use offline supervised learning problems to learn an *exploration policy* π, which takes *two inputs*: a function $f \in \mathcal{F}$ and a context x, and outputs an action. In our example, f will be the output of the policy optimizer on all historic data, and x will be the current user. This is used to produce an agent which interacts with the world, maintaining an initially empty history buffer h, as:

1. The world draws $(x_t, \boldsymbol{r}_t) \sim \mathcal{D}$ and reveals context x_t.
2. The agent computes $f_t \leftarrow \text{POLOPT}(h)$ and a greedy action $\tilde{a}_t = \pi(f_t, x_t)$.
3. The agent plays $a_t = \tilde{a}_t$ with probability $(1-\mu)$, and a_t uniformly at random otherwise.
4. The agent observes $r_t(a_t)$ and appends $(x_t, a_t, r_t(a_t), p_t)$ to the history h, where $p_t = \mu/K$ if $a_t \neq \tilde{a}_t$; and $p_t = 1 - \mu + \mu/K$ if $a_t = \tilde{a}_t$.

Here, f_t is the function optimized on the historical data, and π uses it and x_t to choose an action. Intuitively, π might choose to use the prediction $f_t(x_t)$ most of the time, unless f_t is quite uncertain on this example, in which case π might choose to return the second (or third) most likely action according to f_t. The agent then performs a small amount of additional μ-greedy-style exploration: most of the time it acts according to π but occasionally it explores some more. In practice (§5.4), we find that setting $\mu = 0$ is optimal in aggregate, but non-zero μ is necessary for our theory (§5.3).

5.2.3 Training MÊLÉE by Imitation Learning

The meta-learning challenge is: how do we learn a good exploration policy π? We assume we have access to *fully labeled* data on which we can train π; this data must

include context/reward pairs, but where the reward for *all* actions is known. This is a weak assumption: in practice, we use purely synthetic data as this training data; one could alternatively use any fully labeled classification dataset (this is inspired by Beygelzimer and Langford (2009)). Under this assumption about the data, it is natural to think of π's behavior as a sequential decision making problem in a simulated setting, for which a natural class of learning algorithms to consider are imitation learning algorithms (Daumé et al., 2009; Ross et al., 2011b; Ross and Bagnell, 2014; Chang et al., 2015).[2] Informally, at training time, MÊLÉE will treat one of these synthetic datasets as if it were a contextual bandit dataset. At each time step t, it will compute f_t by running POLOPT on the historical data, and then ask: for *each* action, what would the long time reward look like if I were to take this action. Because the training data for MÊLÉE is fully labeled, this can be evaluated for each possible action, and a policy π can be learned to maximize these rewards.

Importantly, we wish to train π using one set of tasks (for which we have fully supervised data on which to run simulations) and apply it to wholly different tasks (for which we only have bandit feedback). To achieve this, we allow π to depend representationally on f_t in arbitrary ways: for instance, it might use features that capture f_t's uncertainty on the current example (see §5.4.1 for details). We additionally allow π to depend in a *task-independent* manner on the history (for instance, which actions have not yet been tried): it can use features of the actions, rewards and probabilities in the history but *not* depend directly on the contexts x. This is to ensure that π only learns to explore and not also to solve the underlying task-dependent classification problem.

More formally, in imitation learning, we assume training-time access to an *expert*, $\pi^\star$, whose behavior we wish to learn to imitate at test-time. From this, we can define an optimal reference policy $\pi^\star$, which effectively "cheats" at training time by looking at the true labels. The learning problem is then to estimate π to have as similar behavior to $\pi^\star$ as

[2] In other work on meta-learning, such problems are often cast as full *reinforcement-learning* problems. We opt for imitation learning instead because it is computationally attractive and effective when a simulator exists.

4 MÊLÉE (supervised training sets $\{S_m\}$, hypothesis class $\mathcal{F}$, exploration rate $\mu = 0.1$, number of validation examples $N_{\textit{Val}}$), feature extractor Φ

for round $n = 1, 2, \ldots, N$ **do**
  initialize meta-dataset $D = \{\}$ and choose dataset S at random from $\{S_m\}$
  partition and permute S randomly into train *Tr* and validation *Val* where $|\textit{Val}| = N_{\textit{Val}}$
  set history $h_0 = \{\}$
  for round $t = 1, 2, \ldots, |\textit{Tr}|$ **do**
    let $(x_t, \boldsymbol{r}_t) = \textit{Tr}_t$
    for each action $a = 1, \ldots, K$ **do**
      optimize $f_{t,a} = \textsc{PolOpt}(\mathcal{F}, h_{t-1} \oplus (x_t, a, r_t(a), 1\text{-}(K\text{-}1)\mu))$ on augmented history
      roll-out: estimate $\hat{\rho}_a$, the value of a, using $r_t(a)$ and a roll-out policy π^{out}
    end for
    compute $f_t = \textsc{PolOpt}(\mathcal{F}, h_{t-1})$
    aggregate $D \leftarrow D \oplus (\Phi(f_t, x_t, h_{t-1}, \textit{Val}), \langle \hat{\rho}_1, \ldots, \hat{\rho}_K \rangle)$
    roll-in: $a_t \sim \frac{\mu}{K}\mathbf{1}_K + (1-\mu)\pi_{n-1}(f_t, x_t)$ with probability p_t, $\mathbf{1}$ is an indicator function
    append history $h_t \leftarrow h_{t-1} \oplus (x_t, a_t, r_t(a_t), p_t)$
  end for
  update $\pi_n = \textsc{Learn}(D)$
end forreturn $\{\pi_n\}_{n=1}^N$

possible, but without access to those labels. Suppose we wish to learn an exploration policy π for a contextual bandit problem with K actions. We assume access to M supervised learning datasets $S_1, \ldots, S_M$, where each $S_m = \{(x_1, \boldsymbol{r}_1), \ldots, (x_{N_m}, \boldsymbol{r}_{N_m})\}$ of size N_m, where each x_n is from a (possibly different) input space $\mathcal{X}_m$ and the reward vectors are all in $[0, 1]^K$. We wish to learn an exploration policy π with maximal reward: *therefore*, π should imitate a $\pi^\star$ that always chooses its action optimally.

We additionally allow π to depend on a *very small* amount of fully labeled data from the task at hand, which we use to allow π to calibrate f_t's predictions.Because π needs to learn to be task independent, we found that if f_ts were uncalibrated, it was very difficult for π to generalize well to unseen tasks. In our experiments we use only 30 fully labeled examples, but alternative approaches to calibrating f_t that do not require this data would be ideal.

The imitation learning algorithm we use is AggreVaTe (Ross and Bagnell, 2014) (closely related to DAgger (Ross et al., 2011b)), and is instantiated for the contextual bandits meta-learning problem in 4. AggreVaTe learns to choose actions to minimize the cost-to-go of the expert rather than the zero-one classification loss of mimicking its actions. On the first iteration AggreVaTe collects data by observing the expert perform the task, and in each trajectory, at time t, explores an action a in state s, and observes the cost-to-go Q of the expert after performing this action.

Each of these steps generates a cost-weighted training example (s, t, a, Q) and AggreVaTe trains a policy π_1 to minimize the expected cost-to-go on this dataset. At each following iteration n, AggreVaTe collects data through interaction with the learner as follows: for each trajectory, begin by using the current learner's policy π_n to perform the task, interrupt at time t, explore a roll-in action a in the current state s, after which control is provided back to the expert to continue up to time-horizon T. This results in new examples of the cost-to-go (roll-out value) of the expert (s, t, a, Q), under the distribution of states visited by the current policy π_n. This new data is aggregated with all previous data to train the next policy π_{n+1}; more generally, this data can be used by a no-regret online learner to update the policy and obtain π_{n+1}. This is iterated for some number of iterations N and the best policy found is returned. AggreVaTe optionally allow the algorithm to continue executing the expert's actions with small probability β, instead of always executing π_n, up to the time step t where an action is explored and control is shifted to the expert.

Similarly, MÊLÉE operates in an iterative fashion, starting with an arbitrary π and improving it through interaction with an expert. Over N rounds, MÊLÉE selects random training sets and simulates the test-time behavior on that training set. The core functionality is to generate a number of states (f_t, x_t) on which to train π, and to use the supervised data to estimate the value of every action from those states. MÊLÉE achieves this by sampling a random supervised training set and setting aside some validation data from it (Line 3).

It then simulates a contextual bandit problem on this training data; at each time step t, it tries *all* actions and "pretends" like they were appended to the current history (Line 8) on which it trains a new policy and evaluates it's **roll-out value** (Line 9, described below). This yields, for each t, a new training example for π, which is added to π's training set (Line 12); the features for this example are features of the classifier based on true history (Line 11) (and possibly statistics of the history itself), with a label that gives, for each action, the corresponding value of that action (the ρ_as computed in Line 9). MÊLÉE then must commit to a **roll-in action** to *actually* take; it chooses this according to a roll-in policy (Line 13), described below.

The two key questions are: how to choose roll-in actions and how to evaluate roll-out values.

Roll-in actions. The distribution over states visited by MÊLÉE depends on the actions taken, and in general it is good to have that distribution match what is seen at test time as closely as possible. This distribution is determined by a *roll-in* policy (Line 13), controlled in MÊLÉE by exploration parameter $\mu \in [0, 1/K]$. As $\mu \to 1/K$, the roll-in policy approaches a uniform random policy; as $\mu \to 0$, the roll-in policy becomes deterministic. When the roll-in policy does not explore, it acts according to $\pi(f_t, .)$.

Roll-out values. The ideal value to assign to an action (from the perspective of the imitation learning procedure) is that total reward (or advantage) that would be achieved in the long run if we took this action and then behaved according to our final learned policy. Unfortunately, during training, we do not yet know the final learned policy. Thus, a surrogate roll-out policy π^{out} is used instead. A convenient, and often computationally efficient alternative, is to evaluate the value assuming all future actions were taken by the expert (Langford and Zadrozny, 2005; Daumé et al., 2009; Ross and Bagnell, 2014). In our setting, at any time step t, the expert has access to the fully supervised reward vector $\boldsymbol{r}_t$ for the context $\boldsymbol{x}_t$. When estimating the roll-out value for an action a, the expert will

return the true reward value for this action $r_t(a)$ and we use this as our estimate for the roll-out value.

5.3 Theoretical Guarantees

We analyze MÊLÉE, showing that the no-regret property of AGGREVATE can be leveraged in our meta-learning setting for learning contextual bandit exploration. In particular, we first relate the regret of the learner in line 16 to the overall regret of π. This will show that, *if* the underlying classifier improves sufficiently quickly, MÊLÉE will achieve sublinear regret. We then show that for a specific choice of underlying classifier (BANDITRON), this is achieved.

MÊLÉE is an instantiation of AGGREVATE (Ross and Bagnell, 2014); as such, it inherits AGGREVATE's regret guarantees. Let $\hat{\epsilon}_{class}$ denote the empirical minimum expected cost-sensitive classification regret achieved by policies in the class Π on all the data over the N iterations of training when compared to the Bayes optimal regressor, for $U(T)$ the uniform distribution over $\{1, \ldots, T\}$, d_π^t the distribution of states at time t induced by executing policy π, and $Q^\star$ the cost-to-go of the expert:

$$\hat{\epsilon}_{class}(T) = \min_{\pi \in \Pi} \frac{1}{N} \hat{\mathbb{E}}_{t \sim U(T), s \sim d_{\pi_i}^t} \sum_{i=1}^{N} \left[Q^\star_{T-t+1}(s, \pi) - \min_a Q^\star_{T-t+1}(s, a) \right]$$

Theorem 3 (Thm 2.2 of Ross and Bagnell (2014), adapted)**.** *After N rounds in the parameter-free setting, if a* LEARN *(Line 16) is no-regret algorithm, then as $N \to \infty$, with probability* 1*, it holds that $J(\bar{\pi}) \leq J(\pi^\star) + 2T\sqrt{K\hat{\epsilon}_{class}(T)}$, where $J(\cdot)$ is the reward of the exploration policy, $\bar{\pi}$ is the average policy returned, and $\hat{\epsilon}_{class}(T)$ is the average regression regret for each π_n accurately predicting $\hat{\rho}$.*

This says that if we can achieve low regret at the problem of learning π on the training data it observes ("D" in MÊLÉE), i.e. $\hat{\epsilon}_{class}(T)$ is small, then this translates into low regret in the contextual-bandit setting.

At first glance this bound looks like it may scale linearly with T. However, the bound in Theorem 3 is dependent on $\hat{\epsilon}_{class}(T)$. Note however, that s is a combination of the context vector x_t and the classification function f_t. As $T \to \infty$, one would hope that f_t improves significantly and $\hat{\epsilon}_{class}(T)$ decays quickly. Thus, sublinear regret may still be achievable when f learns sufficiently quickly as a function of T. For instance, if f is optimizing a strongly convex loss function, online gradient descent achieves a regret guarantee of $O(\frac{\log T}{T})$ (e.g., Theorem 3.3 of Hazan et al. (2016)), potentially leading to a regret for MÊLÉE of $O(\sqrt{(\log T)/T})$.

The above statement is informal (it does not take into account the interaction between learning f and π). However, we can show a specific concrete example: we analyze MÊLÉE's test-time behavior when the underlying learning algorithm is BANDITRON. BANDITRON is a variant of the multiclass Perceptron that operates under bandit feedback. Details of this analysis (and proofs, which directly follow the original BANDITRON analysis) are given in Theorem 5.3; here we state the main result. Let $\gamma_t = \Pr[r_t(\pi(f_t, x_t) = 1)|x_t] - \Pr[r_t(f_t(x_t)) = 1|x_t]$ be the edge of $\pi(f_t, .)$ over f, and let $\Gamma = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\frac{1}{1+K\gamma_t}$ be an overall measure of the edge. For instance if π simply returns f's prediction, then all $\gamma_t = 0$ and $\Gamma = 1$. We can then show the following:

Stylized test-time analysis for Banditron: Details

The BANDITRONMÊLÉE algorithm is specified in 5. The is exactly the same as the typical test time behavior, except it uses a BANDITRON-type strategy for learning the underlying classifier f in the place of POLOPT. POLICYELIMINATIONMETA takes as arguments: π (the learned exploration policy) and $\mu \in (0, 1/(2K))$ an added uniform exploration parameter. The BANDITRON learns a linear multi-class classifier parameterized by a weight matrix of size $K \times D$, where D is the input dimensionality. The BANDITRON assumes a pure multi-class setting in which the reward for one ("correct") action is 1 and the reward for all other actions is zero.

At each round t, a prediction $\hat{a}_t$ is made according to f_t (summarized by W^t). We then define an exploration distribution that "most of the time" acts according to $\pi(f_t, .)$, but smooths each action with μ probability. The chosen action a_t is sampled from this distribution and a binary reward is observed. The weights of the BANDITRON are updated according to the BANDITRON update rule using $\tilde{U}^t$.

---

5 BANDITRONMÊLÉE (g, μ)

---

initialize $W^1 = \mathbf{0} \in \mathbb{R}^{K \times D}$
for rounds $t = 1 \dots T$: **do**
  observe $x_t \in \mathbb{R}^D$
  compute $\hat{a}_t = f_t(x_t) = \text{argmax}_{k \in K} \left(W^t x_t\right)_k$
  define $Q^\mu(a) = \mu + (1 - K\mu)\mathbf{1}[a = \pi(W^t, x_t)]$
  sample $a_t \sim Q^\mu$
  observe reward $r_t(a_t) \in \{0, 1\}$
  define $\tilde{U}^t \in \mathbb{R}^{K \times D}$ as:
  $\tilde{U}^t_{a,\cdot} = x_t \left(\frac{\mathbf{1}[r_t(a_t)=1]\mathbf{1}[a_t=a]}{Q^\mu(a)} - \mathbf{1}[\hat{a}_t = a] \right)$
  update $W^{t+1} = W^t + \tilde{U}^t$
end for

The *only* difference between BANDITRONMÊLÉE and the original BANDITRON is the introduction of π in the sampling distribution. The original algorithm achieves the following mistake bound shown below, which depends on the notion of multi-class hinge-loss. In particular, the hinge-loss of W on $(x, \boldsymbol{r})$ is $\ell(W, (x, \boldsymbol{r})) = \max_{a \neq a^\star} \max \left\{0, 1 - (Wx)_{a^\star} + (Wx)_a\right\}$, where $a^\star$ is the a for which $r(a) = 1$. The overall hinge-loss L is the sum of ℓ over the sequence of examples.

Theorem 4 (Thm. 1 and Corr. 2 of Kakade et al. (2008))**.** *Assume that for the sequence of examples, $(x_1, \boldsymbol{r}_1), (x_2, \boldsymbol{r}_2), \dots, (x_T, \boldsymbol{r}_T)$, we have, for all t, $||x_t|| \leq 1$. Let $W^\star$ be any matrix, let L be the cumulative hinge-loss of $W^\star$, and let $D = 2\,||W^\star||_F^2$ be the complexity of $W^\star$. The number of mistakes M made by the* BANDITRON *satisfies*

$$\mathbb{E}M \leq L + K\mu T + 3\max\left\{\frac{D}{\mu}, \sqrt{DTK\mu}\right\} + \sqrt{DL/\mu} \tag{5.2}$$

where the expectation is taken with respect to the randomness of the algorithm. Furthermore, in a low noise setting (there exists $W^\star$ with fixed complexity d and loss $L \leq O(\sqrt{DKT})$), then by setting $\mu = \sqrt{D/(TK)}$, we obtain $\mathbb{E}M \leq O(\sqrt{KDT})$.

We can prove an analogous result for BANDITRONMÊLÉE. The key quantity that will control how much π improves the execution of BANDITRONMÊLÉE is how much π improves on f_t when f_t is wrong. In particular, let $\gamma_t = \Pr[r_t(\pi(f_t, x_t) = 1)|x_t] - \Pr[r_t(f_t(x_t)) = 1|x_t]$ be the edge of $\pi(f_t, .)$ over f, and let $\Gamma = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\frac{1}{1+K\gamma_t}$ be an overall measure of the edge. (If π does nothing, then all $\gamma_t = 0$ and $\Gamma = 1$.) Given this quantity, we can prove the following Theorem 5.

Proof: [sketch] The proof is a small modification of the original proof of Theorem 4. The only change is that in the original proof, the following bound is used: $\mathbb{E}_t\tilde{/}x_t{}^2 = 1 + 1/\mu \leq 2/\mu$. $\mathbb{E}_t||\tilde{U}^t||^2/||x_t||^2 = 1 + 1/\mu \leq 2/\mu$. We use, instead: $\mathbb{E}_t||\tilde{U}^t||^2/||x_t||^2 \leq 1 + \mathbb{E}_t\frac{1}{\mu+\gamma_t} \leq \frac{2\mathbb{E}_t\frac{1}{1+\gamma_t}}{\mu}$. The rest of the proof goes through identically. □

Theorem 5. *Assume that for the sequence of examples, $(x_1, \boldsymbol{r}_1), (x_2, \boldsymbol{r}_2), \ldots, (x_T, \boldsymbol{r}_T)$, we have, for all t, $||x_t|| \leq 1$. Let $W^\star$ be any matrix, let L be the cumulative hinge-loss of $W^\star$, let μ be a uniform exploration probability, and let $D = 2\,||W^\star||_F^2$ be the complexity of $W^\star$. Assume that $\mathbb{E}\gamma_t \geq 0$ for all t. Then the number of mistakes M made by* MÊLÉE *with* BANDITRON *as* POLOPT *satisfies:*

$$\mathbb{E}M \leq L + K\mu T + 3\max\left\{D\Gamma/\mu, \sqrt{DTK\Gamma\mu}\right\} + \sqrt{DL\Gamma/\mu} \tag{5.3}$$

where the expectation is taken with respect to the randomness of the algorithm.

Note that under the assumption $\mathbb{E}\gamma_t \geq 0$ for all t, we have $\Gamma \leq 1$. The analysis gives the same mistake bound for BANDITRON but with the factor of Γ, hence this result improves upon the BANDITRON analysis only when $\Gamma < 1$.

This result is highly stylized and the assumption that $\mathbb{E}\gamma_t \geq 0$ is overly strong. This assumption ensures that π never decreases the probability of a "correct" action. It does,

however, help us understand the behavior of MÊLÉE, qualitatively: First, the quantity that matters in Theorem 5, $\mathbb{E}_t \gamma_t$ is (in the 0/1 loss case) exactly what MÊLÉE is optimizing: the expected improvement for choosing an action against f_t's recommendation. Second, the benefit of using π within BANDITRON is a *local* benefit: because π is trained with expert rollouts, as discussed in §5.3, the primary improvement in the analysis is to ensure that π does a better job predicting (in a single step) than f_t does. An obvious open question is whether it is possible to base the analysis on the *regret* of π (rather than its error) and whether it is possible to extend beyond the simple BANDITRON setting.

5.4 Experimental Setup and Results

Our experimental setup operates as follows: Using a collection of synthetically generated classification problems, we train an exploration policy π using MÊLÉE (4). This exploration policy learns to explore on the basis of calibrated probabilistic predictions from f together with a predefined set of exploration features (§5.4.1). Once π is learned and fixed, we follow the test-time behavior described in §5.2.2 on a set of 300 "simulated" contextual bandit problems, derived from standard classification tasks. In all cases, the underlying classifier f is a linear model trained with a policy optimizer that runs stochastic gradient descent.

We seek to answer two questions experimentally: (1) How does MÊLÉE compare empirically to alternative (expert designed) exploration strategies? (2) How important are the additional features used by MÊLÉE in comparison to using calibrated probability predictions from f as features?

5.4.1 Training Details for the Exploration Policy

Exploration Features. In our experiments, the exploration policy is trained based on features Φ (4, Line 12). These features are allowed to depend on the current classifier

f_t, and on any part of the history *except* the inputs x_t in order to maintain task independence. We additionally ensure that its features are independent of the *dimensionality* of the inputs, so that π can generalize to datasets of arbitrary dimensions. The specific features we use are listed below; these are largely inspired by Konyushkova et al. (2017b) but adapted and augmented to our setting. The **features of** f_t that we use are:

a) predicted probability $p(a_t|f_t, \boldsymbol{x}_t)$; **b)** entropy of the predicted probability distribution; **c)** a one-hot encoding for the predicted action $f_t(\boldsymbol{x}_t)$. The **features of** h_{t-1} that we use are: **a)** current time step t; **b)** normalized counts for all previous actions predicted so far; **c)** average observed rewards for each action; **d)** empirical variance of the observed rewards for each action in the history.

In our experiments, we found that it is essential to calibrate the predicted probabilities of the classifier f_t. We use a very small held-out dataset, of size 30, to achieve this. We use Platt's scaling (Platt, 1999; Lin et al., 2007) method to calibrate the predicted probabilities. Platt's scaling works by fitting a logistic regression model to the classifier's predicted scores.

Training Datasets. In our experiments, we follow Konyushkova et al. (2017b) (and also Peters et al. (2014), in a different setting) and train the exploration policy π only on *synthetic data*. This is possible because the exploration policy π never makes use of x explicitly and instead only accesses it via f_t's behavior on it. We generate datasets with uniformly distributed class conditional distributions. The datasets are always two-dimensional. Details are in §5.4.2.

5.4.2 Details of Synthetic Datasets

We generate datasets with uniformly distributed class conditional distributions. We generate 2D datasets by first sampling a random variable representing the Bayes classification error. The Bayes error is sampled uniformly from the interval 0.0 to 0.5. Next, we generate a balanced dataset where the data for each class lies within a unit

rectangle and sampled uniformly. We overlap the sampling rectangular regions to generate a dataset with the desired Bayes error selected in the first step.

5.4.3 Implementation Details.

Our implementation is based on scikit-learn (Pedregosa et al., 2011). We fix the training time exploration parameter μ to 0.1 (Line 13). We train the exploration policy π on 82 synthetic datasets each of size 3000 with uniform class conditional distributions, a total of $246k$ samples (§5.4.2). We train π using a linear classifier Breiman (2001) and set the hyper-parameters for the learning rate, and data scaling methods using three-fold cross-validation on the whole meta-training dataset. For the classifier class $\mathcal{F}$, we use a linear model trained with stochastic gradient descent. We standardize all features to zero mean and unit variance, or scale the features to lie between zero and one. To select between the two scaling methods, and tune the classifier's learning rate, we use three-fold cross-validation on a small fully supervised training set of size 30 samples. The same set is used to calibrate the predicted probabilities of f_t.

5.4.4 Evaluation Tasks and Metrics

Following Bietti et al. (2018), we use a collection of 300 binary classification datasets from openml.org for evaluation; the precise list and download instructions is in §5.4.5. These datasets cover a variety of different domains including text & image processing, medical diagnosis, and sensory data. We convert multi-class classification datasets into cost-sensitive classification problems by using a $0/1$ encoding. Given these fully supervised cost-sensitive multi-class datasets, we simulate the contextual bandit setting by only revealing the reward for the selected actions. For evaluation, we use progressive validation (Blum et al., 1999), which is exactly computing the reward of the algorithm. Specifically, to evaluate the performance of an exploration algorithm $\mathcal{A}$ on a dataset S of size n, we compute the progressive validation return $G(\mathcal{A}) = \frac{1}{n}\sum_{t=1}^{n} r_t(a_t)$

as the average reward up to n, where a_t is the action chosen by the algorithm $\mathcal{A}$ and r_t is the true reward vector.

5.4.5 List of Datasets

The datasets we used can be accessed at https://www.openml.org/d/<id>. The list of (id, size) pairs below shows the (<id> for the datasets we used and the dataset size in number of examples:

(46,100)	(878, 100)	(924, 130)	(1026, 155)	(1488, 195)	(877, 250)	(778, 252)	(925, 323)
(909, 400)	(1153, 484)	(920, 500)	(947, 559)	(1464, 748)	(813, 1000)	(983, 1473)	(1067, 2109)
(1021,5473)	(716, 100)	(916, 100)	(1075, 130)	(745, 159)	(446, 200)	(911, 250)	(1442, 253)
(1140, 324)	(1025,400)	(742, 500)	(926, 500)	(949, 559)	(37, 768)	(837, 1000)	(1128, 1545)
(772,2178)	(1069,5589)	(726, 100)	(922, 100)	(1141, 130)	(756, 159)	(721, 200)	(918, 250)
(1449,253)	(1144, 329)	(1071, 403)	(749, 500)	(936, 500)	(950, 559)	(1014, 797)	(845, 1000)
(1130,1545)	(948,2178)	(980,5620)	754, 100)	(932, 100)	(885, 131)	(1085, 159)	(1124, 201)
(933, 250)	(1159, 259)	(1011,336)	(1123, 405)	(750, 500)	(937, 500)	(951, 559)	(970, 841)
(849, 1000)	(1138,1545)	(958,2310)	(847, 6574)	(762, 100)	(1473, 100)	(444, 132)	(1054, 161)
(1132, 203)	(935, 250)	(450,264)	(1147, 337)	(1160, 410)	(766, 500)	(943, 500)	(826, 576)
(994, 846)	(866,1000)	(1139,1545)	(312,2407)	(1116,6598)	(768, 100)	(965, 101)	(921, 132)
(748, 163)	(40, 208)	(1136, 250)	(811,264)	(1133, 347)	(1126, 412)	(779, 500)	(987, 500)
(1004, 600)	(841, 950)	(903,1000)	(1142,1545)	(1487,2534)	(803,7129)	(775, 100)	(1064, 101)
(974, 132)	(747, 167)	(733, 209)	(746, 250)	(336, 267)	(337,349)	(1122, 413)	(792, 500)
(1470, 500)	(334, 601)	(50, 958)	(904, 1000)	(1146,1545)	(737,3107)	(1496,7400)	(783, 100)
(956, 106)	(719, 137)	(973, 178)	(796, 209)	(763, 250)	(1152,267)	(59, 351)	(1127, 421)
(805, 500)	(825, 506)	(1158, 604)	(1016, 990)	(910, 1000)	(1161,1545)	(953,3190)	(725, 8192)
(789, 100)	(1061, 107)	(1013, 138)	(463, 180)	(996, 214)	(769, 250)	(53, 270)	(1135, 355)
(764, 450)	(824, 500)	(853, 506)	(770, 625)	(31, 1000)	(912, 1000)	(1166, 1545)	(3, 3196)
(735, 8192)	(808, 100)	(771, 108)	(1151, 138)	(801, 185)	(1005, 214)	(773, 250)	(1073, 274)
(1143, 363)	(1065, 458)	(838, 500)	(872, 506)	(997, 625)	(715, 1000)	(913, 1000)	(1050, 1563)
(1038, 3468)	(752,8192)	(812, 100)	(736, 111)	(784, 140)	(1164, 185)	(895, 222)	(776, 250)
(1156, 275)	(1048,369)	(1149, 458)	(855, 500)	(717, 508)	(1145, 630)	(718, 1000)	(917, 1000)
(991, 1728)	(871,3848)	(761, 8192)	(828, 100)	(448, 120)	(1045, 145)	(788, 186)	(1412, 226)
(793, 250)	(880,284)	(860, 380)	(1498, 462)	(869, 500)	(1063, 522)	(1443, 661)	(723, 1000)
(741, 1024)	(962,2000)	(728, 4052)	(807, 8192)	(829, 100)	(782, 120)	(1066, 145)	(1154, 187)
(820, 235)	(794, 250)	(1121, 294)	(1129, 384)	(724, 468)	(870, 500)	(954, 531)	(774, 662)
(740, 1000)	(1444,1043)	(971,2000)	(720, 4177)	(850, 100)	(1455, 120)	(1125, 146)	(941, 189)
(851, 240)	(830, 250)	(43, 306)	(1163, 386)	(814, 468)	(879, 500)	(1467, 540)	(795, 662)
(743, 1000)	(1453,1077)	(978,2000)	(1043,4562)	(865, 100)	(1059, 121)	(902, 147)	(1131, 193)
(464, 250)	(832, 250)	(818, 310)	(900, 400)	(1148, 468)	(884, 500)	(1165, 542)	(827, 662)
(751, 1000)	(1068,1109)	(995,2000)	(44, 4601)	(868, 100)	(1441, 123)	(1006, 148)	(753, 194)
(730, 250)	(834, 250)	(915, 315)	(906, 400)	(1150, 470)	(886, 500)	(1137, 546)	(931, 662)
(797, 1000)	(934,1156)	(1020,2000)	(979,5000)	(875, 100)	(714, 125)	(969, 150)	(1012, 194)
(732, 250)	(863, 250)	(1157, 321)	(907, 400)	(765, 475)	(888, 500)	(335, 554)	(292, 690)
(799, 1000)	(1049,1458)	(1022,2000)	(1460,5300)	(876,100)	(867, 130)	(955, 151)	(1155, 195)
(744, 250)	(873, 250)	(1162, 322)	(908, 400)	(767, 475)	(896, 500)	(333, 556)	(1451, 705)
(806, 1000)	(1454, 1458)	(914, 2001)	(1489, 5404)				

Because our evaluation is over 300 datasets, we report aggregate results in two forms. The simpler one is **Win/Loss Statistics:** We compare two exploration methods on a given dataset by counting the number of statistically significant wins and losses. An exploration algorithm $\mathcal{A}$ wins over another algorithm $\mathcal{B}$ if the progressive validation return $G(\mathcal{A})$ is statistically significantly larger than B's return $G(\mathcal{B})$ at the 0.01 level using a paired sample t-test.

We also report **cumulative distributions** of rewards for each algorithm. In particular, for a given relative reward value ($x \in [0, 1]$), the corresponding CDF value for a given

algorithm is the fraction of datasets on which this algorithm achieved reward at least x. We compute relative reward by Min-Max normalization. Min-Max normalization linearly transforms reward y to $x = \frac{y-\min}{\max-\min}$, where min & max are the minimum & maximum rewards among all exploration algorithms.

5.4.6 Baseline Exploration Algorithms

Our experiments aim to determine how MÊLÉE compares to other standard exploration strategies. In particular, we compare to:

ϵ-greedy: With probability ϵ, explore uniformly at random; with probability $1 - \epsilon$ act greedily according to f_t (Sutton, 1996). Experimentally, we found $\epsilon = 0$ optimal on average, consistent with the results of Bietti et al. (2018).

ϵ-decreasing: selects a random action with probabilities ϵ_i, where $\epsilon_i = \epsilon_0/t$, $\epsilon_0 \in]0, 1]$ and t is the index of the current round. In our experiments we set $\epsilon_0 = 0.1$. (Sutton and Barto, 1998)

Exponentiated Gradient ϵ-greedy: maintains a set of candidate values for ϵ-greedy exploration. At each iteration, it runs a sampling procedure to select a new ϵ from a finite set of candidates. The probabilities associated with the candidates are initialized uniformly and updated with the Exponentiated Gradient (EG) algorithm. Following Li et al. (2010), we use candidate set $\{\epsilon_i = 0.05 \times i + 0.01, i = 1, \cdots, 10\}$ for ϵ.

LinUCB: Maintains confidence bounds for reward payoffs and selects actions with the highest confidence bound. It is impractical to run "as is" due to high-dimensional matrix inversions. We use diagonal approximation to the covariance when the dimensions exceeds 150. (Li et al., 2010)

τ-first: Explore uniformly on the first τ fraction of the data; after that, act greedily.

Cover: Maintains a uniform distribution over a fixed number of policies. The policies are used to approximate a covering distribution over policies that are good for both exploration and exploitation (Agarwal et al., 2014).

Cover Non-Uniform: similar to Cover, but reduces the level of exploration of Cover to be more competitive with the Greedy method. Cover-Nu doesn't add extra exploration beyond the actions chose by the covering policies (Bietti et al., 2018).

In all cases, we select the best hyperparameters for each exploration algorithm following Bietti et al. (2018). These hyperparameters are: the choice of ϵ in ϵ-greedy, τ in τ-first, the number of bags, and the tolerance ψ for Cover and Cover-NU. We set $\epsilon = 0.0$, $\tau = 0.02$, bag size $= 16$, and $\psi = 0.1$.

5.4.7 Experimental Results: Simulated Contextual Bandit Tasks

The overall results are shown in Figure 5.1. In the left-most figure, we see the CDFs for the different algorithms. To help read this, note that at $x = 1.0$, we see that MÊLÉE has a relative reward at least 1.0 on more than 40% of datasets, while ϵ-decreasing and ϵ-greedy achieve this on about 30% of datasets. We find that the two strongest baselines are ϵ-decreasing and ϵ-greedy (better when reward differences are small, toward the left of the graph). The two curves for ϵ-decreasing and ϵ-greedy coincide. This happens because the exploration probability ϵ_0 for ϵ-decreasing decays rapidly approaching zero with a rate of $\frac{1}{t}$, where t is the index of the current round. MÊLÉE outperforms the baselines in the "large reward" regimes (right of graph) but underperforms ϵ-decreasing and ϵ-greedy in low reward regimes (left of graph). In Figure 5.2a, we show statistically-significant win/loss differences for each of the algorithms. MÊLÉE is the only algorithm that always wins more than it loses against other algorithms.

To understand more directly how MÊLÉE compares to ϵ-decreasing, in the middle figure of Figure 5.1, we show a scatter plot of rewards achieved by MÊLÉE (x-axis) and ϵ-decreasing (y-axis) on each of the 300 datasets, with statistically significant differences highlighted in red and insignificant differences in blue. Points below the diagonal line correspond to better performance by MÊLÉE (147 datasets) and points above to ϵ-decreasing (124 datasets). The remaining 29 had no significant difference.

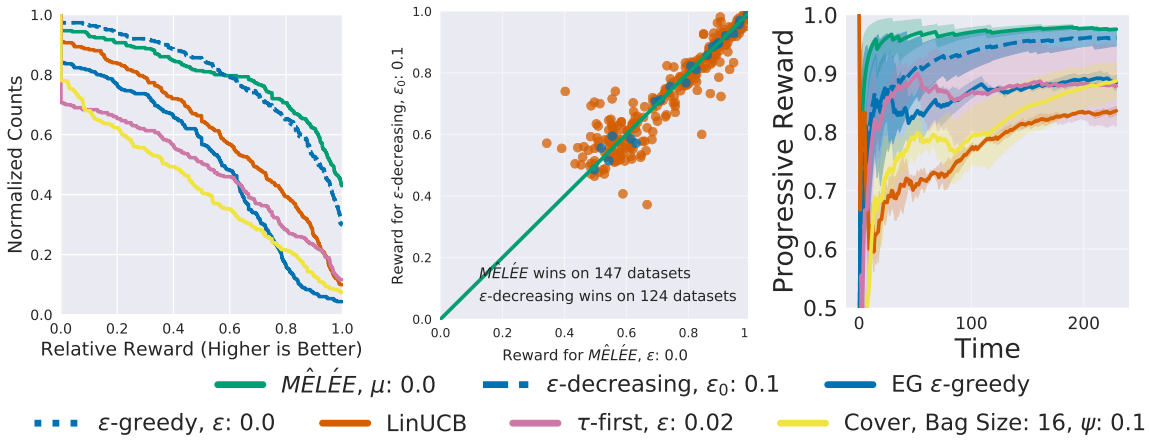


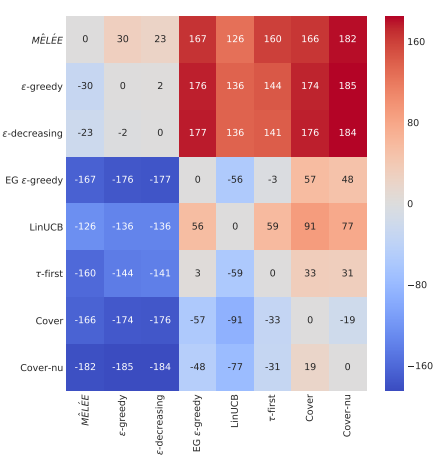
Figure 5.1: Comparison of algorithms on 300 classification problems. **(Left)** Comparison of all exploration algorithms using the empirical cumulative distribution function of the relative progressive validation return G (upper-right is optimal). The curves for ϵ-decreasing & ϵ-greedy coincide. **(Middle)** Comparison of MÊLÉE to the second best performing exploration algorithm (ϵ-decreasing), every data point represents one of the 300 datasets, x-axis shows the reward of G(MÊLÉE), y-axis show the reward of G(ϵ-decreasing), and red dots represent statistically significant runs. **(Right)** A representative learning curve on dataset #1144.

In the right-most graph in Figure 5.1, we show a representative example of learning curves for the various algorithms. Here, we see that as more data becomes available, all the approaches improve (except τ-first, which has ceased to learn after 2% of the data).

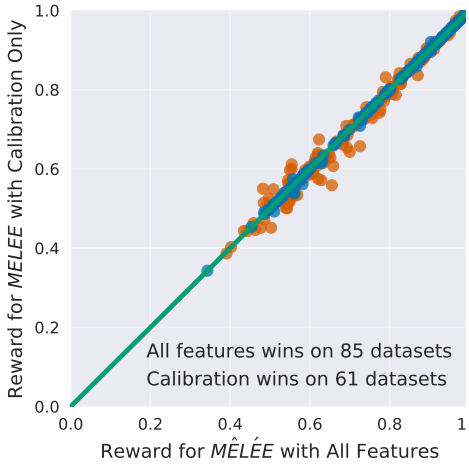
Finally, we consider the effect that the additional features have on MÊLÉE's performance. In particular, we consider a version of MÊLÉE with all features (this is the version used in all other experiments) with an ablated version that only has access to the (calibrated) probabilities of each action from the underlying classifier f. The comparison is shown as a scatter plot in Figure 5.2b. Here, we can see that the full feature set *does* provide lift over just the calibrated probabilities, with a win-minus-loss improvement of 24.

5.4.8 Experimental Results: Learning to Rank

We additionally evaluate MÊLÉE on a natural learning to rank dataset. The dataset we consider is the Microsoft Learning to Rank dataset, variant MSLR-10K from Qin and



(a) Win statistics: each (row, column) entry shows the number of times the row algorithm won against the column, minus the number of losses.



(b) Comparison of training MÊLÉE with all the features (§5.4.1, y-axis) vs training using only the calibrated prediction probabilities (x-axis). MÊLÉE gets an additional leverage when using all the features.

Liu (2013) [3]. The dataset consists of feature vectors extracted from query-url pairs along with relevance judgment labels. The relevance judgments are obtained from a retired labeling set of a commercial web search engine (Microsoft Bing), which take 5 values from 0 (irrelevant) to 4 (perfectly relevant) . In our experiments, we limit the number of labels to the two extremes: 0 and 4, and we drop the queries not labelled as any of the two extremes. A query-url pair is represented by a 136-dimensional feature vector. The dataset is highly imbalanced as the number of irrelevant queries is much larger than the number of relevant ones. To address this, we sample the number of irrelevant queries to match that of the relevant ones. To avoid correlations between the observed query-url pairs, we group the queries by the query ID, and sample a single query from each group. We convert relevance scores to losses with 0 indicating a perfectly relevant document, and 1 an irrelevant one.

Figure 5.3 shows the evaluation results on a subset of the MSLR-10K dataset. Since the performance is closely matched between the different exploration algorithms, we repeat the experiment 16 times with randomly shuffled permutations of the MSLR-10K dataset. Figure 5.3 (left) shows the learning curve of the trained policy π as well as the baselines.

[3] https://www.microsoft.com/en-us/research/project/mslr/

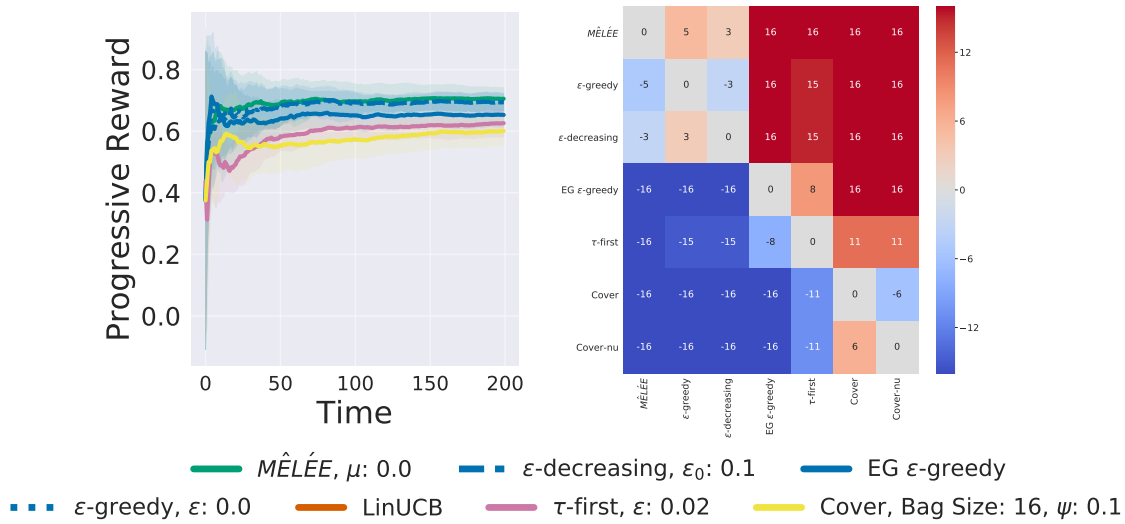


Figure 5.3: Results for the Learning to Rank task. **(Left)** Learning curve on the MSLR-10K dataset: x-axis shows the number of queries observed, and y-axis shows the progressive reward. **(Right)** Win/Loss counts for all pairs of algorithms over 16 random shuffles for the MSLR-10K dataset.

Here, we see that MÊLÉE quickly achieves high reward, after about 100 examples the two strongest baselines catch up. By 200 examples all approaches have asymptoted. We exclude LinUCB from these runs because the required matrix inversions made it too computationally expensive.[4] Figure 5.3 (right) shows statistically-significant win/loss differences for each of the algorithms, across these 16 shuffles. Each row/column entry shows the number of times the row algorithm won against the column, minus the number of losses. MÊLÉE is the only algorithm that always wins more than it loses against other algorithms, and outperforms the nearest competition (ϵ-decreasing) by 3 points.

5.5 Related Work and Discussion

The field of meta-learning is based on the idea of replacing hand-engineered learning heuristics with heuristics learned from data. One of the most relevant settings for meta-learning to ours is active learning, in which one aims to learn a decision function to decide which examples, from a pool of unlabeled examples, should be labeled. Past

[4] In a single run of LinUCB we observed that its performance is on par with ϵ-greedy.

approaches to meta-learning for active learning include reinforcement learning-based strategies (Woodward and Finn, 2017; Fang et al., 2017), imitation learning-based strategies (Bachman et al., 2017), and batch supervised learning-based strategies (Konyushkova et al., 2017b). Similar approaches have been used to learn heuristics for optimization (Li and Malik, 2016; Andrychowicz et al., 2016), multiarm (non-contextual) bandits Maes et al. (2012), and neural architecture search (Zoph and Le, 2016), recently mostly based on (deep) reinforcement learning. While meta-learning for contextual bandits is most similar to meta-learning for active learning, there is a fundamental difference that makes it significantly more challenging: in active learning, the goal is to select as few examples as you can to learn, so by definition the horizon is short; in contextual bandits, learning to explore is fundamentally a long-horizon problem, because what matters is not immediate reward but long term learning.

In reinforcement learning, Gupta et al. (2018) investigated the task of meta-learning an exploration strategy for a distribution of related tasks by learning a latent exploration space. Similarly, Xu et al. (2018) proposed a teacher-student approach for learning to do exploration in off-policy reinforcement learning. While these approaches are effective if the distribution of tasks is very similar and the state space is shared among different tasks, they fail to generalize when the tasks are different. Our approach targets an easier problem than exploration in full reinforcement learning environments, and can generalize well across a wide range of different tasks with completely unrelated features spaces.

There has also been a substantial amount of work on constructing "good" exploration policies, in problems of varying complexity: traditional bandit settings (Karnin and Anava, 2016), contextual bandits (Féraud et al., 2016) and reinforcement learning (Osband et al., 2016). In both bandit settings, most of this work has focused on the learning theory aspect of exploration: what exploration distributions *guarantee* that learning will succeed (with high probability)? MÊLÉE, lacks such guarantees: in particular, if the data distribution of the observed contexts ($\phi(f_t)$) in some test problem differs substantially from that on

which MÊLÉE was trained, we can say nothing about the quality of the learned exploration. Nevertheless, despite fairly substantial distribution mismatch (synthetic $\rightarrow$ real-world), MÊLÉE works well in practice, and our stylized theory (§ 5.3) suggests that there may be an interesting avenue for developing strong theoretical results for contextual bandit learning with learned exploration policies, and perhaps other meta-learning problems.

In conclusion, we presented MÊLÉE, a meta-learning algorithm for learning exploration policies in the contextual bandit setting. MÊLÉE enjoys no-regret guarantees, and empirically it outperforms alternative exploration algorithm in most settings. One limitation of MÊLÉE is the computational resources required during the offline training phase on the synthetic datasets. In the future, we will work on improving the computational efficiency for MÊLÉE in the offline training phase and scale the experimental analysis to problems with larger number of classes. This concludes Part I of the dissertation where we focused on studying minimally supervised learning algorithms based on meta-learning. This approach mainly depends on being able to simulate learning tasks at training time, but what if these simulations are not possible? In Part II we show that it is still possible to design minimally supervised learning algorithm using reinforcement learning when a reward signal could be observed by the learning agent.

Part II

Reinforcement-Learning Algorithms

Chapter 6: Reinforcement Learning With No Incremental Feedback

6.1 Introduction

In Part I we studied minimally supervised learning algorithms based on meta-learning. These approaches typically require access to a distribution of learning tasks on which we can run simulations at training time. However, these simulations are not always accessible. In Part II we discuss a different approach for learning with minimal supervision based on reinforcement and imitation learning. We start the discussion for this second part by presenting a reinforcement learning algorithm designed specifically for addressing the case where the learning agent observes a very sparse learning signal: a reward or loss observed only at the end of the learning episode.

Current state of the art learning-based systems require enormous, costly datasets on which to train supervised models. To progress beyond this requirement, we need learning systems that can interact with their environments, collect feedback (a loss or reward), and improve continually over time. In most real-world settings, such feedback is sparse and delayed: most decisions made by the system will not immediately lead to feedback. Any sort of interactive system like this will face at least two challenges: the credit assignment problem (which decision(s) did the system make that led to the good/bad feedback?) ; and the exploration/exploitation problem (in order to learn, the system must try new things, but these could be bad).

We consider the question of how to learn in an extremely sparse feedback setting: the environment operates episodically, and the only feedback comes at the *end* of the

episode, with *no incremental feedback* to guide learning. This setting naturally arises in many classic reinforcement learning problems (paragraph 6.5): a barista robot will only get feedback from a customer after their cappuccino is finished[1]. It also arises in the context of bandit structured prediction (Sokolov et al., 2016a; Chang et al., 2015) (§6.2.3), where a structured prediction system must produce a single output (e.g., translation) and observes only a scalar loss.

We introduce a novel reinforcement learning algorithm, RESIDUAL LOSS PREDICTION (RESLOPE) (§6.3), which aims to learn *effective representations of the **loss** signal*. By effective we mean effective in terms of credit assignment. Intuitively, RESLOPE attempts to learn a decomposition of the episodic loss into a sum of per-time-step losses. This process is akin to how a person solving a task might realize before the task is complete when and where they are likely to have made suboptimal choices. In RESLOPE, the per-step loss estimates are conditioned on all the information available up to the current point in time, allowing it to learn a highly non-linear representation for the episodic loss (assuming the policy class is sufficiently complex; in practice, we use recurrent neural network policies). When the system receives the final episodic loss, it uses the *difference* between the observed loss and the cumulative predicted loss to update its parameters.

Algorithmically, RESLOPE operates as a *reduction* (§6.4) to contextual bandits (Langford and Zhang, 2008), allowing the bandit algorithm to handle exploration/exploitation and focusing only on the credit assignment problem. RESIDUAL LOSS PREDICTION is theoretically motivated by the need for variance reduction techniques when estimating counterfactual costs (Dudík et al., 2014) and enjoys a no-regret bound (§6.4) when the underlying bandit algorithm is no-regret. Experimentally, we show the efficacy of RESLOPE on four benchmark reinforcement problems and three bandit structured prediction

[1]This problem can be—and to a large degree *has* been—mitigated through the task-specific and complex process of reward engineering and reward shaping. Indeed, we were surprised to find that many classic RL algorithms fail badly when incremental rewards disappear. We aim to make such problems disappear.

problems (§6.6.1), comparing to several reinforcement learning algorithms: Reinforce, Proximal Policy Optimization and Advantage Actor-Critic.

6.2 Problem Formulation and Background

We focus on finite horizon, episodic Markov Decision Processes (MDPs) in this chapter, which captures *both* traditional reinforcement learning problems (paragraph 6.5) *and* bandit structured prediction problems (§6.2.3). Our solution to this problem, RESIDUAL LOSS PREDICTION (§6.3) operates in a *reduction* framework. Specifically, we assume there exists "some" machine learning problem that we know how to solve, and can treat as an oracle. Our reduction goal is to develop a procedure that takes the reinforcement learning problem described above and map it to this oracle, so that a good solution to the oracle guarantees a good solution to our problem. The specific oracle problem we consider is a contextual bandit learning algorithm, relevant details of which we review in §6.2.1.

Formally, we consider a (possibly virtual) learning agent that interacts directly with its environment. The interaction between the agent and the environment is governed by a restricted class of finite-horizon Markov Decision Processes (MDP), defined as a tuple $\{\mathcal{S}, s_0, \mathcal{A}, \mathcal{P}, \mathcal{L}, H\}$ where:

$\mathcal{S}$ is a large but finite state space, typically $\mathcal{S} \subset \mathbb{R}^d$; $s_0 \in \mathcal{S}$ is a start state; $\mathcal{A}$ is a finite action space[2] of size K; $\mathcal{P} = \{\, \mathcal{P}(s'|s,a) : s, s' \in \mathcal{S}, a \in \mathcal{A} \,\}$ is the set of Markovian transition probabilities; $\mathcal{L} \in \mathbb{R}^{|\mathcal{S}|}$ is the state dependent loss function, defined only at terminal states $s \in \mathcal{S}$; H is the horizon (maximum length of an episode).

The goal is to learn a policy π, which defines the behavior of the agent in the environment. We consider policies that are potentially functions of entire trajectories[3], and potentially produce distributions over actions: $\pi(s) \in \Delta^{\mathcal{A}}$, where $\Delta^{\mathcal{A}}$ is the $\mathcal{A}$-dimensional probability simplex. However, to ease exposition, we will present the background in terms

[2] In some problems the set of actions available will depend on the current state.

[3] Policies could choose to ignore all but the most recent state, for instance in fully observable environments, though this may be insufficient in partially observable environments (Littman and Sutton, 2002).

of policies that depend only on states; this can be accomplished by simply blowing up the state space.

Let d_h^π denote the distribution of states visited at time step h when starting at state s_0 and operating according to π: $d_{h+1}^\pi(s') = \mathbb{E}_{s_h \sim d_h^\pi, a_h \sim \pi(s_h)} \mathcal{P}(s' \mid s = s_h, a = a_h)$ The quality of the policy π is quantified by its value function or q-value function: $V^\pi(s) \in \mathbb{R}$ associates each state with the expected future loss for starting at this state and following π afterwards; $Q^\pi(s, a) \in \mathbb{R}$ associates each state/action pair with the same expected future loss: $V^\pi(s_h) = \mathbb{E}_{s_H \sim d_H^\pi \mid s_h} \mathcal{L}(s_H)$ and $Q^\pi(s_h, a_h) = \mathbb{E}_{s_H \sim d_H^\pi \mid s_h, a_h} \mathcal{L}(s_H)$ The learning goal is to estimate a policy π from a hypothesis class of policies Π with minimal expected loss: $J(\pi) = V^\pi(s_0)$.

6.2.1 Contextual Bandits

The contextual bandit learning problem (Langford and Zhang, 2008) can be seen as a tractable special case of reinforcement learning in which the time horizon $H = 1$. In particular, the world operates episodically. At each round t, the world reveals a context (i.e. feature vector) $\boldsymbol{x}_t \in \mathcal{X}$; the system chooses an action a_t; the world reveals a scalar loss $\ell_t(\boldsymbol{x}_t, a_t) \in \mathbb{R}^+$, a loss *only* for the selected action that may depend stochastically on $\boldsymbol{x}_t$ and a_t. The total loss for a system over T rounds is the sum of losses: $\sum_{t=1}^T \ell_t(\boldsymbol{x}_t, a_t)$. The goal in policy optimization is to learn a policy $\pi : \boldsymbol{x} \to \mathcal{A}$ from a policy class Π that has low *regret* with respect to the best policy in this class. Assuming the learning algorithm produces a sequence of policies $\pi_1, \pi_2, \ldots, \pi_T$, its regret is: $\textit{Regret}\left(\langle \pi_t \rangle_{t=1}^T\right) = \sum_{t=1}^T \ell(\boldsymbol{x}_t, \pi_t(\boldsymbol{x}_t)) - \min_{\pi^* \in \Pi} \sum_{t=1}^T \ell(\boldsymbol{x}_t, \pi^*(\boldsymbol{x}_t))$. The particular contextual bandit algorithms we will use in this chapter perform a second level of reduction: they assume access to an oracle supervised learning algorithm that can optimize a cost-sensitive loss (§6.2.5), and transform the contextual bandit problem to a cost-sensitive classification problem. Algorithms in this family typically vary along two axes: how to explore (faced with a new $\boldsymbol{x}$ how does the algorithm choose which action to take); and

how to update (Given the observed loss ℓ_t, how does the algorithm construct a supervised training example on which to train). More details are in §6.2.2.

6.2.2 More Details on Contextual Bandit Algorithms

We assume that contexts are chosen i.i.d from an unknown distribution $\mathcal{D}(\boldsymbol{x})$, the actions are chosen from a finite action set $\mathcal{A}$, and the distribution over loss $\mathcal{D}(\ell|a, \boldsymbol{x})$ is fixed over time, but is unknown. In this context, the key challenge in contextual bandit learning is the exploration/exploitation problem. Classic algorithms for the contextual bandit problem such as EXP4.P (Beygelzimer et al., 2011) can achieve a $\sqrt{T}$ regret bound; in particular:

$$R\left(\text{EXP4}\right) \in O\left(\sqrt{TK\log|\Pi|}\right) \tag{6.1}$$

where $K = |A|$. When the regret is provably sublinear in T, such algorithms are often called "no regret" because their average regret per time step goes to zero as $T \to \infty$.

The particular contextual bandit algorithms we will use in this chapter perform a second level of reduction: they assume access to an oracle supervised learning algorithm that can optimize a cost-sensitive loss, and transform the contextual bandit problem to a cost-sensitive classification problem. Algorithms in this family typically vary along two axes:

1. How to explore? I.e., faced with a new $\boldsymbol{x}$ how does the algorithm choose which action to take;

2. How to update? Given the observed loss ℓ_t, how does the algorithm construct a supervised training example on which to train.

As a simple example, an algorithm might explore uniformly at random on 10% of the examples and return the best guess action on 90% of examples (ϵ-greedy exploration). A single round to such an algorithm consists of a tuple $(\boldsymbol{x}, a, p)$, where p is the probability

with which the algorithm took action a. (In the current example, this would be $\frac{0.1}{K}$ for all actions except $\pi(\boldsymbol{x})$ and $0.9 + \frac{0.1}{K}$ for $a = \pi(\boldsymbol{x})$.) If the update rule were "inverse propensity scaling" (IPS) (Horvitz and Thompson, 1952), the generated cost-sensitive learning example would have $\boldsymbol{x}$ as an input, and a cost vector $\boldsymbol{c} \in \mathbb{R}^K$ with zeros everywhere except in position a where it would take value $\frac{\ell}{p}$. The justification for this scaling is that in expectation over $a \sim p$, the expected value of this cost vector is equal to the true costs for each action. Neither of these choices is optimal (IPS has very high variance as p gets small); we discuss alternative exploration strategies and variance reduction strategies (paragraph 6.3.2).

6.2.3 Bandit Structured Prediction via Learning to Search

In structured prediction, we observe structured input sequences $\boldsymbol{x}^{\text{SP}} \in \mathcal{X}$ and the goal is to predict a set of correlated output variables $\boldsymbol{y}^{\text{SP}} \in \mathcal{Y}$. For example, in machine translation, the input $\boldsymbol{x}^{\text{SP}}$ is a sentence in an input language (e.g., Tagalog) and the output $\boldsymbol{y}^{\text{SP}}$ is a sentence in an output language (e.g., Chippewa). In the fully supervised setting, we have access to samples $(\boldsymbol{x}^{\text{SP}}, \boldsymbol{y}^{\text{SP}})$ from some distribution $\mathcal{D}$ over input/output pairs. Structured prediction problems typically come paired with a structured loss $\ell(\boldsymbol{y}^{\text{SP}}, \hat{\boldsymbol{y}}^{\text{SP}}) \in \mathbb{R}^+$ that measures the fidelity of a predicted output $\hat{\boldsymbol{y}}^{\text{SP}}$ to the "true" output $\boldsymbol{y}^{\text{SP}}$. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ with low expected loss under $\mathcal{D}$: $\mathbb{E}_{(\boldsymbol{x}^{\text{SP}}, \boldsymbol{y}^{\text{SP}}) \sim \mathcal{D}} \ell(\boldsymbol{y}^{\text{SP}}, f(\boldsymbol{x}^{\text{SP}}))$. Recently, it has become popular to solve structured prediction problems incrementally using some form of recurrent neural network (RNN) model. When the output $\boldsymbol{y}^{\text{SP}}$ contains multiple parts (e.g., words in a translation), the RNN can predict each word in sequence, conditioning each prediction on all previous decisions. Although typically such models are trained to maximize cross-entropy with the gold standard output (in a fully supervised setting), there is mounting evidence that this has similar drawbacks to pre-RNN techniques, such as overfitting to gold standard prefixes (the model never learns what to do once it has made an error) and sensitivity to errors of different severity (due to

error compounding). In order to achieve this we must formally map from the structured prediction problem to the MDP setting; this mapping is natural and described in detail in §6.2.4.

Our focus in this chapter is on the recently proposed *bandit* structured prediction setting (Chang et al., 2015; Sokolov et al., 2016a), at training time, we only have access to input $\boldsymbol{x}^{\text{SP}}$ from the marginal distribution $\mathcal{D}^{\mathcal{X}}$. For example, a Chippewa speaker sees an article in Tagalog, and asks for a translation. A system then produces a *single* translation $\hat{\boldsymbol{y}}^{\text{SP}}$, on which a single "bandit" loss $\ell(\hat{\boldsymbol{y}}^{\text{SP}} \mid \boldsymbol{x}^{\text{SP}})$ is observed. Given only this bandit feedback, without ever seeing the "true" translation, the system must learn.

6.2.4 Bandit Structured Prediction

Recently, it has become popular to solve structured prediction problems incrementally using some form of recurrent neural network (RNN) model. When the output $\boldsymbol{y}$ contains multiple parts (e.g., words in a translation), the RNN can predict each word in sequence, conditioning each prediction on all previous decisions. Although typically such models are trained to maximize cross-entropy with the gold standard output (in a fully supervised setting), there is mounting evidence that this has similar drawbacks to pre-RNN techniques, such as overfitting to gold standard prefixes (the model never learns what to do once it has made an error) and sensitivity to errors of different severity (due to error compounding).

By casting the structured prediction problem explicitly as a sequential decision making problem (Daumé and Marcu, 2005; Daumé et al., 2009; Ross et al., 2011b; Neu and Szepesvári, 2009), we can avoid these problems by applying imitation-learning style algorithms to their solution. This "Learning to Search" framework (Figure 6.1) solves structured prediction problems by:

1. converting structured and control problems to search problems by defining a search space of states $\mathcal{S}$ and an action set $\mathcal{A}$;

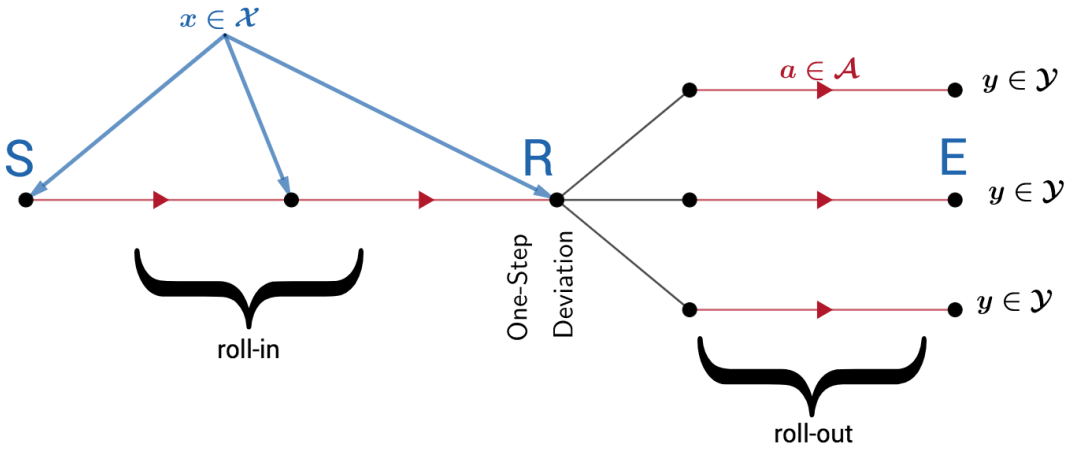


Figure 6.1: An example for a search space defined by a Learning to Search (L2S) algorithm. A search space is defined in terms of the set of states $\mathcal{X}$, and the set of actions $\mathcal{A}$. The agent starts at the initial state S, and queries the roll-in policy π^{in} twice, next, at state R, the agent considers all three actions as possible one-step deviations. The agent queries the roll-out policy π^{out} to generate three different trajectories from the set of possible output structures $\mathcal{Y}$.

2. defining structured features over each state to capture the inter-dependency between output variables;

3. constructing a reference policy π^{ref} based on the supervised training data;

4. learning a policy π^{learn} that imitates or improves upon the reference policy.

In the *bandit* structured prediction setting, this maps nicely to the type of MDPs described at the beginning of this section. The formal reduction, following (Daumé and Marcu, 2005) is to ignore the first action a_0 and to transition to an "initial state" s_1 by drawing an input $\boldsymbol{x}^{\text{SP}} \sim \mathcal{D}^{\mathcal{X}}$. The search space of the structured prediction task then generates the remainder of the state/action space for this example. The episode terminates when a state, s_H that corresponds to a "final output" is reached, at which point the structured prediction loss $\ell(\hat{\boldsymbol{y}}_{s_H} \mid \boldsymbol{x}^{\text{SP}})$ is computed on the output that corresponds to s_H. This then

becomes the loss function $\mathcal{L}$ in the MDP. Clearly, learning a good policy under this MDP is equivalent to learning a structured prediction model with low expected loss.

6.2.5 Cost-sensitive Classification

Many of the contextual bandit approaches we use in turn reduce the contextual bandit problem to a cost-sensitive classification problem. Cost-sensitive classification problems are defined by inputs $\boldsymbol{x}$ and cost vectors $\boldsymbol{y} \in \mathbb{R}^K$, where $\boldsymbol{y}(i)$ is the cost of choosing class i on this example. The goal in cost-sensitive classification is to learn a classifier $f : \boldsymbol{x} \to [K]$ such that $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\big[\boldsymbol{y}(f(\boldsymbol{x}))\big]$ is small. A standard strategy for solving cost-sensitive classification is via reduction to regression in a one-against-all framework (Beygelzimer et al., 2005). Here, a regression function $g(\boldsymbol{x}, i) \in \mathbb{R}$ is learned that predicts costs given input/class pairs. A predicted class on an input $\boldsymbol{x}$ is chosen as $\operatorname{argmin}_i g(\boldsymbol{x}, i)$. This cost-sensitive one-against-all approach achieves low regret when the underlying regressor is good. In practice, we use regression against Huber loss.

6.3 Proposed Approach

Our goal is to learn a good policy in a Markov Decision Process (§6.2) in which losses only arrive at the end of episodes. Our solution, RESIDUAL LOSS PREDICTION (RESLOPE), automatically deduces per-step losses based only on the episodic loss. To gain an intuition for how this works, suppose you are at work and want to meet a colleague at a nearby coffee shop. In hopes of finding a more efficient path to the coffee shop, you take a different path than usual. While you're on the way, you run into a friend and talk to them for a few minutes. You then arrive at the coffee shop and your colleague tells you that you are ten minutes late. To estimate the value of the different path, you wonder: how much of this ten minutes is due to taking the different path vs talking to a friend. If you

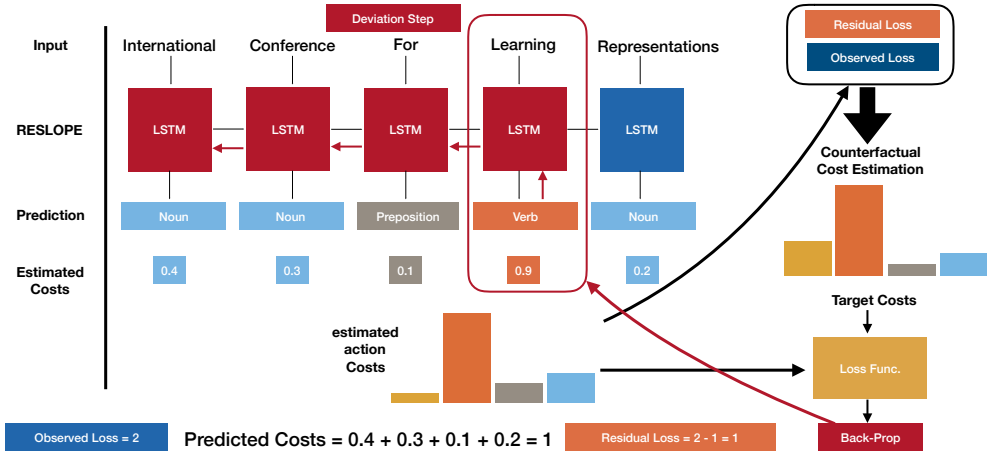


Figure 6.2: RESIDUAL LOSS PREDICTION: the system assigns a part-of-speech tag sequence to the sentence "International Conference for Learning Representations". Each state represents a partial labeling. The end state e = [Noun, Noun, Preposition, Verb, Noun]. The end state e is associated with an episodic loss $\ell(e)$, which is the total hamming loss in comparison to the optimal output structure e^*=[Adjective, Noun, Preposition, Noun, Noun]. We emphasize that our algorithm doesn't assume access to neither the optimal output structure, nor the hamming loss for every time step. Only the total hamming loss is observed in this case ($\ell(e) = 2$).

can accurately estimate that you spent seven minutes talking to your friend (you lost track of time), you can conclude that the disadvantage for the different path is three minutes.

RESLOPE addresses the problem of **sparse reward signals** and **credit assignment** by learning a decomposition of the reward signal, essentially doing automatic reward shaping (evaluated in §6.6.4). Finally, it addresses the problem of **exploration vs exploitation** by relying on a strong underlying contextual bandit learning algorithm with provably good exploration behavior.

6.3.1 Key Idea: RESIDUAL LOSS PREDICTION

Akin to the coffee shop example, RESLOPE learns a decomposition of the episodic loss (i.e total time spent from work to the coffee shop) into a sum of per-time-step losses (i.e. timing activities along the route). RESLOPE operates as a reduction from reinforcement learning with episodic loss to contextual bandits. In this way, RESLOPE solves the *credit assignment* problem by predicting residual losses, and relies on the underlying contextual

bandit oracle to solve explore/exploit. RESLOPE operates online, incrementally updating a policy π^{learn} once per episode. In the structured contextual bandit setting, we assume access to a *reference policy*, π^{ref}, that was perhaps pretrained on supervised data, and which we wish to improve; a hyperparameter β controls how much we trust π^{ref}. As π^{learn} improves, we replace π^{ref} with π^{learn}. In the RL setting, we set $\beta = 0$.

We initially present a simplified variant of RESLOPE that mostly follows the learned policy (and the reference policy as appropriate), except for a *single* deviation per episode. This algorithm closely follows the bandit version of the Locally Optimal Learning to Search (LOLS) approach of Chang et al. (2015), with three key differences: (1) residual loss prediction; (2) alternative exploration strategies; (3) alternative parameter update strategies. We assume access to a contextual bandit oracle CB that supports the following API:

1. CB.ACT$(\pi^{\text{learn}}, \boldsymbol{x})$, where $\boldsymbol{x}$ is the input example; this returns a tuple (a, p), where a is the selected action, and p is the probability with which that action was selected.
2. CB.COST$(\pi^{\text{learn}}, \boldsymbol{x}, a)$ returns the estimated cost of taking action a in the context.
3. CB.UPDATE$(\pi^{\text{learn}}, \boldsymbol{x}, a, p, c)$, where $\boldsymbol{x}$ is the input example, $a \in [K]$ is the selected action, $p \in (0, 1]$ is the probability of that action, and $c \in \mathbb{R}$ is the target cost.

The requirement that the contextual bandit algorithm also predicts costs (CB.COST) is somewhat non-standard, but is satisfied by many contextual bandit algorithms in practice, which often operate by regressing on costs and picking the minimal predicted cost action. We describe the specific contextual bandit approaches we use in §6.3.2.

Algorithm 6 shows how our reduction is constructed formally. It uses a MAKEENVIRONMENT(t) function to construct a new environment (randomly in RL and by selecting the tth example in bandit structured prediction). To learn a good policy, RESLOPE reduces long horizon trajectories to single-step contextual bandit training examples. In each episode, RESLOPE picks a single time step to deviate. Prior to the deviation step, it

6 RESIDUAL LOSS PREDICTION (RESLOPE) with *single* deviations

Require: Reference policy π^{ref}, mixture parameter β, contextual bandit oracle CB, MAKEENVIRONMENT to build new enviornments

Initialize a policy π_0^{learn} {either randomly or from a pretrained model}
for all episodes $t = 1 \dots T$ **do**
  env $\leftarrow$ MAKEENVIRONMENT(t)
  Initialize variables: example $\boldsymbol{x}^{\text{dev}}$, action a^{dev}, probability p^{dev}
  Initialize cost vector $\hat{c}_h^{\text{dev}} = 0$ for $h = 1 \dots \text{env}.H$
  Choose deviation step $h^{\text{dev}} \leftarrow$ UNIFORM(env.H)
  Choose rollout policy π^{mix} to be π^{ref} with probability β or π_{t-1}^{learn} with probability $1-\beta$
  for all time steps $h = 1 \dots \text{env}.H$ **do**
    $\boldsymbol{x} \leftarrow$ env.STATEFEATURES {computed by an RNN}
    if $h \neq h^{\text{dev}}$ { no deviation } **then**
      $a \leftarrow \begin{cases} \pi_{t-1}^{\text{learn}}(\boldsymbol{x}) & \text{if } h < h^{\text{dev}} \\ \pi^{\text{mix}}(\boldsymbol{x}) & \text{if } h > h^{\text{dev}} \end{cases}$
    else if $h = h^{\text{dev}}$ { deviation } **then**
      $(a^{\text{dev}}, p^{\text{dev}}) \leftarrow \text{CB.ACT}(\pi^{\text{learn}}, \boldsymbol{x})$
      $\boldsymbol{x}^{\text{dev}} \leftarrow \boldsymbol{x}$
      $a \leftarrow a^{\text{dev}}$
    end if
    $\hat{c}_h^{\text{dev}} \leftarrow \text{CB.COST}(\pi_{t-1}^{\text{learn}}, \boldsymbol{x}, a)$
    env.STEP(a) {updates environment and internal state of the RNN }
  end for
  $\ell^{\text{residual}} \leftarrow$ env.FINALLOSS $- \sum_{h \neq h^{\text{dev}}} \hat{c}_h^{\text{dev}}$
  $\pi_t^{\text{learn}} \leftarrow \text{CB.UPDATE}(\pi_{t-1}^{\text{learn}}, \boldsymbol{x}^{\text{dev}}, a^{\text{dev}}, p^{\text{dev}}, \ell^{\text{residual}})$
end for
Return average policy $\bar{\pi} = \frac{1}{T} \sum_t \pi_t^{\text{learn}}$

executes π^{learn} as a roll-in policy and after the deviation step, it executes a β mixture of π^{learn} and π^{ref} (Figure 6.1). *At* the deviation step, it calls CB.ACT to handle the exploration and choose an action. At *every* step, it calls CB.COST to estimate the cost of that action. Finally, it constructs a single contextual bandit training example for the deviation step, whose input was the observation at that step, whose action and probability are those that were selected by CB.ACT, and whose cost is the observed total cost *minus* the cost of every *other* action taken in this trajectory. This example is sent to CB.UPDATE. When the contextual bandit policy is an RNN (as in our setting), this will then compute a loss which is back-propagated through the RNN.

6.3.2 Contextual Bandit Oracle

The contextual bandit oracle receives examples where the cost for only one predicted action is observed, but no others. It learns a policy for predicting actions minimizing expected loss by estimating the unobserved target costs for the unpredicted actions and exploring different actions to balance the exploitation exploration trade-off (paragraph 6.3.2). The contextual bandit oracle then calls a cost-sensitive multi-class oracle (§6.2.5) given the target costs and the selected action.

CB.UPDATE: Cost Estimation Techniques. The update procedure for our contextual bandit oracles takes an example $\boldsymbol{x}$, action a, action probability p and cost c as input and updates its policy. We do this by reducing to a cost-sensitive classification oracle (§6.2.5), which expects an example $\boldsymbol{x}$ and a cost vector $\boldsymbol{y} \in \mathbb{R}^K$ that specifies the cost for *all* actions (not just the selected one). The reduction challenge is constructing this cost-sensitive classification example given the input to CB.UPDATE. We consider three methods: inverse propensity scoring (Horvitz and Thompson, 1952), doubly robust estimation (Dudík et al., 2014) and multitask regression (Langford and Agarwal, 2017).

Inverse Propensity Scoring (IPS): IPS uses the selected action probability p to correct for the shift in action proportions predicted by the policy π^{learn}. IPS estimates the target cost vector $\boldsymbol{y}$ as: $\boldsymbol{y}(i) = \frac{c}{p}\mathbf{1}[i = a]$, where $\mathbf{1}$ is an indicator function and where a is the selected action and c is the observed cost. While IPS yields an unbiased estimate of costs, it typically has a large variance as $p \to 0$.

Doubly Robust Cost Estimation (DR): The doubly robust estimator uses both the observed cost c as well as its own predicted costs $\hat{y}(i)$ for *all* actions, forming a target that combines these two sources of information. DR estimates the target cost vector $\boldsymbol{y}$ as: $\boldsymbol{y}(i) = \hat{y}(i) + \mathbf{1}[i = a](c - \hat{y}(i))/p$. The DR estimator remains unbiased, and the estimated loss $\boldsymbol{y}$ helps decrease its variance.

Multitask Regression (MTR): The multitask regression estimator functions somewhat differently from IPS and DR. Instead of reducing to to cost-sensitive classification, MTR reduces directly to importance-weighted regression. MTR maintains K different regressors for predicting costs given input/action pairs. Given $\boldsymbol{x}, a, c, p$, MTR constructs a *regression* example, whose input is $(\boldsymbol{x}, a)$, whose target output is c and whose importance weight is $1/p$.

CB.ACT: Exploration Strategies. We experiment with three exploration strategies:

Uniform: explores randomly with probability ϵ and otherwise acts greedily (Sutton and Barto, 1998).

Boltzmann: varies action probabilities where action a is chosen with probability proportional to $\exp[-c(a)/\text{temp}]$, where temp $\in \mathbb{R}^+$ is the temperature, and $c(a)$ is the predicted cost of action a.

Bootstrap Exploration: (Agarwal et al., 2014) trains a bag of multiple policies simultaneously. Each policy in the bag votes once on its predicted action, and an action is sampled from this distribution. To train, each example gets passed to each policy Poisson($\lambda = 1$)-many times, which ensures diversity . Bootstrap can operate in "greedy update" and "greedy prediction" mode (Bietti et al., 2017). In greedy update, we always update the first policy in the bag exactly once. In greedy prediction, we always predict the action from the first policy during exploitation.

6.4 Theoretical Analysis

For simplicity, we first consider the case where we have access to a good reference policy π^{ref} but do not have access to good Q-value estimates under π^{ref}. The only way one can obtain a Q-value estimate is to do a roll-out, but in a non-resettable environment, we can only do this once. We will subsequently consider the case of suboptimal (or missing)

reference policies, in which the goal of the analysis will change from competing with π^{ref} to competing with both π^{ref} and a local optimality guarantee.

Theorem 6. *Setting $\beta = 1$, running* RESLOPE *for N episodes with a contextual bandit algorithm, the average returned policy $\bar{\pi} = \mathbb{E}_n \pi_n$ has regret equal to the suboptimality of π^{ref}, namely:*

$$\textit{Regret}(\bar{\pi}) \leq \textit{Regret}(\pi^{ref}) + \frac{1}{N}\epsilon_{CB}(N) + \epsilon_{approx} \tag{6.2}$$

where $\epsilon_{CB}(N)$ is the cumulative regret of the underlying contextual bandit algorithm after N steps, and ϵ_{approx} is an approximation error term that goes to zero as $N \to \infty$ so long as the contextual bandit algorithm is no-regret and assuming all costs are realizable under the hypothesis class used by RESLOPE.

In particular, when the problem is realizable and the contextual bandit algorithm is no-regret, RESLOPE is also no-regret. The realizability assumption is unfortunate, but does not appear easy to remove (see §6.4.1 for the proof).

6.4.1 Proof of Theorem 6

In a now-classic lemma, Kakade et al. (2003) and Bagnell et al. (2004) show that the difference in total loss between two policies can be computed exactly as a sum of per-time-step advantages of one over the other:

Lemma 1 (Bagnell et al. (2004); Kakade et al. (2003))**.** For all policies π and π':

$$J(\pi) - J(\pi') = \sum_{h=1}^{H} \mathbb{E}_{s_h \sim d_\pi^h} \left[Q^{\pi'}(s_h, \pi) - V^{\pi'}(s_h) \right] \tag{6.3}$$

Proof: [Proof of Theorem 6] Let π_n be the nth learned policy and $\bar{\pi}$ be the average learned policy. We wish to bound $J(\bar{\pi}) - J(\pi^*)$. We proceed as follows, largely following the AggreVaTe analysis (Ross and Bagnell, 2014). We begin by noting that $J(\bar{\pi}) - J(\pi^*) = J(\bar{\pi}) - J(\pi^{\text{ref}}) + J(\pi^{\text{ref}}) - J(\pi^*)$ and will concern ourselves with bounding the first

difference.

$$J(\bar{\pi}) - J(\pi^{\text{ref}}) = \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) \right] \tag{6.4}$$

Fix an n, and consider the sum above for a fixed deviation time step h^{dev}. In what follows, we consider π_n to represent both the learned policy as well as the contextual bandit cost estimator, CB.COST.

$$\sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) \right] \tag{6.5}$$

$$= \mathbb{E}_{s \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) \right] + \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) \right] \tag{6.6}$$

$$= \mathbb{E}_{s \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - \left(Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}}) - \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s' \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s', \pi_n) - Q^{\pi^{\text{ref}}}(s', \pi^{\text{ref}}) \right] \right) \right] \tag{6.7}$$

$$= \mathbb{E}_{s \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - \left(\mathbb{E}_{s_H \sim \pi^{\text{ref}} \mid s_{h^{\text{dev}}} = s} \ell(s_H) - \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s' \sim d^h_{\pi_n}} \left[\text{CB.COST}(\pi_n, s', \pi_n(s')) + \epsilon_{\text{approx}}(n, s') \right] \right) \right] \tag{6.8}$$

$$= \mathbb{E}_{s \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - \left(\mathbb{E}_{s_H \sim \pi^{\text{ref}} \mid s_{h^{\text{dev}}} = s} \ell(s_H) - \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s' \sim d^h_{\pi_n}} \text{CB.COST}(\pi_n, s', \pi_n(s')) \right) \right] + \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s' \sim d^h_{\pi_n}} \epsilon_{\text{approx}}(n, s') \tag{6.9}$$

$$= \mathbb{E}_{s \sim d^{h^{\text{dev}}}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - \text{Residual}(\pi_n, h^{\text{dev}}, s) \right] + \sum_{h \neq h^{\text{dev}}} \mathbb{E}_{s' \sim d^h_{\pi_n}} \epsilon_{\text{approx}}(n, s') \tag{6.10}$$

where $\text{Residual}(\pi_n, h^{\text{dev}}, s)$ is the estimated residual on this example.

Since the above analysis holds for an arbitrary n, it holds in expectation over n; thus:

$$J(\bar{\pi}) - J(\pi^{\text{ref}}) = \mathbb{E}_n \mathbb{E}_{s \sim d^{h^{\text{dev}}_n}_{\pi_n}} \left[Q^{\pi^{\text{ref}}}(s, \pi_n) - \text{Residual}(\pi_n, h^{\text{dev}}, s) \right] + \mathbb{E}_n \sum_{h \neq h^{\text{dev}}_n} \mathbb{E}_{s' \sim d^h_{\pi_n}} \epsilon_{\text{approx}}(n, s') \tag{6.11}$$

$$= \frac{1}{N} \epsilon_{\text{CB}}(N) + \mathbb{E}_n \sum_{h \neq h^{\text{dev}}_n} \mathbb{E}_{s' \sim d^h_{\pi_n}} \epsilon_{\text{approx}}(n, s') \tag{6.12}$$

In the first line, the term in square brackets is exactly the cost being minimized by the contextual bandit algorithm and thus reduces to the regret of the CB algorithm.

In Eq (6.12), we have H-many regret minimizing online learners: one estimating the policy and one estimating estimating the $H - 1$-many costs. Cesa-Bianchi and Lugosi (2006) (Theorem 7.3) proves that in a K-player game, if each player minimizes its *internal regret*, then the overall values convergence in time-average to the value of the game. In order to apply this result to our setting we need to convert from external regret (which we are assuming about the underlying learners) to internal regret (which the theorem requires).

This can be done using, for instance, the algorithm of which gives a general reduction from an algorithm that minimizes internal regret to one that minimizes external regret.

From there, by the strong realizability assumption, and the fact that multiple no-regret minimizers will achieve a time-averaged minimax value, we can conclude that as $N \to \infty$, the approximation error term will vanish. Moreover, the term in the round parentheses $(\dots)$ is exactly the expected value of the target of the contextual bandit cost. Therefore, If the CB algorithm has regret sublinear in N, both $\epsilon_{\text{CB}}(N)$ and the approximation error term go to zero as $N \to \infty$. This completes the proof that the overall algorithm is no-regret. □

In the case that π^{ref} is not known to be optimal, or not available, we follow the LOLS analysis and obtain a regret to a convex combination of π^{ref} and the learned policy's one-step deviations (a form of local optimality) and can additionally show the following (proof in §6.4.2):

Theorem 7. *For arbitrary β, define the combined regret of $\bar{\pi}$ as: $\text{Regret}_\beta(\bar{\pi}) = \beta[J(\bar{\pi}) - J(\pi^{ref})] + (1-\beta)\sum_h [J(\bar{\pi}) - \min_{\pi \in \Pi} \mathbb{E}_{s \sim d^h_{\bar{\pi}}} Q^{\bar{\pi}}(s, \pi)]$. The first term is suboptimality to π^{ref}; the second term is suboptimality to the policy's own realizable one-step deviations. Given a contextual bandit learning algorithm, and under a realizability assumption, the combined regret of $\bar{\pi}$ satisfies: $\text{Regret}_\beta(\bar{\pi}) \leq \frac{1}{N}\epsilon_{CB}(N) + \epsilon_{approx}$*

Again, if the contextual bandit algorithm is no regret, then $\epsilon_{\text{CB}}/N \to 0$ as $N \to \infty$; see §6.4.2 for the proof.

6.4.2 Proof of Theorem 7

Proof: [Proof of Theorem 7] The proof follows a combination of the proof of Theorem 6 with the LOLS analysis. Using the same notation as before, additionally let π_n^{out} be the mixture of π_n with π^{ref} for rollout.

First, we observe (LOLS Eq 6):

$$J(\bar{\pi}) - J(\pi^{\text{ref}}) = \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} [Q^{\pi^{\text{ref}}}(s, \pi_n) - Q^{\pi^{\text{ref}}}(s, \pi^{\text{ref}})] \tag{6.13}$$

Then (LOLS Eq 7):

$$\sum_h \left[J(\bar{\pi}) - \min_{\pi \in \Pi} \mathbb{E}_{s \sim d^h_{\bar{\pi}}} Q^{\bar{\pi}}(s, \pi) \right] \leq \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi_n}(s, \pi_n) - \min_a Q^{\pi_n}(s, a) \right] \tag{6.14}$$

So far nothing has changed. It will be convenient to define $Q^{\pi_n}_\beta(s) = \beta \min_a Q^{\pi^{\text{ref}}}(s, a) + (1-\beta) \min_a Q^{\pi_n}(s, a)$. For each n fix the deviation time step h^{dev}_n. We plug these together ala LOLS and get:

$$\beta \Big(J(\bar{\pi}) - J(\pi^{\text{ref}}) \Big) + (1-\beta) \Big(J(\bar{\pi}) - \min_{\pi \in \Pi} \mathbb{E}_{s \sim d^h_{\bar{\pi}}} Q^{\bar{\pi}}(s, \pi) \Big) \tag{6.15}$$

$$\leq \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s, \pi_n) - \beta \min_a Q^{\pi^{\text{ref}}}(s, a) - (1-\beta) \min_a Q^{\pi_n}(s, a) \right] \tag{6.16}$$

$$= \mathbb{E}_n \sum_h \mathbb{E}_{s \sim d^h_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s, \pi_n) - Q^{\pi_n}_\beta(s) \right] \tag{6.17}$$

$$= \mathbb{E}_n \mathbb{E}_{s^{\text{dev}} \sim d^{h^{\text{dev}}_n}_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s^{\text{dev}}, \pi_n) - \left(Q^{\pi_n}_\beta(s^{\text{dev}}) - \sum_{h \neq h^{\text{dev}}_n} \mathbb{E}_{s_h \sim d^h_{\pi_n}} \left(Q^{\pi^{\text{out}}_n}(s_h, \pi_n) - Q^{\pi_n}_\beta(s_h) \right) \right) \right] \tag{6.18}$$

$$= \mathbb{E}_n \mathbb{E}_{s^{\text{dev}} \sim d^{h^{\text{dev}}_n}_{\pi_n}} \left[Q^{\pi^{\text{out}}_n}(s^{\text{dev}}, \pi_n) - \left(\mathbb{E}_{s_H \sim d^H_{\pi_n} \mid s_{h^{\text{dev}}_n} = s^{\text{dev}}} \mathcal{L}_n(s_H) - \sum_{h \neq h^{\text{dev}}} \text{CB.COST}(\pi_n, s_h) \right) \right] \tag{6.19}$$

The final step follows because the inner-most expectation is exactly what the contextual bandit algorithm is estimating, and $Q^{\pi_n}_\beta(s^{\text{dev}})$ is exactly the expectation of the observed loss. At this point the rest of the proof follows that of Theorem 6, relying on the same internal-to-external regret transformation, and the joint no-regret minimization of all "players." □

6.4.3 Multi-deviation RESIDUAL LOSS PREDICTION

Finally, we present the multiple deviation variant of RESLOPE. Algorithm 7 shows how RESLOPE operates under multiple deviations. The difference between the single and multiple deviation mode is twofold:

1. Instead of deviating at a single time step, *multi*-dev RESLOPE performs deviations at each time step in the horizon; 2. Instead of generating a single contextual bandit example per episode, *multi*-dev RESLOPE generates H examples, where H is the length of the time horizon, effectively updating the policy H times.

7 RESIDUAL LOSS PREDICTION (RESLOPE) with *multiple* deviations

Require: Contextual bandit oracle CB, MAKEENVIRONMENT to build new enviornments

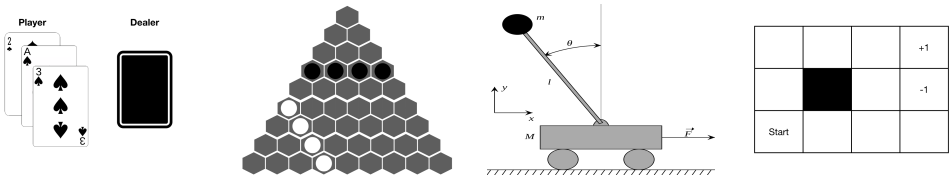
Initialize a policy π_0^{learn} {either randomly or from a pretrained model}
for all episodes $t = 1 \dots T$ **do**
  env $\leftarrow$ MAKEENVIRONMENT(t)
  Initialize variables: examples $\boldsymbol{x}_h^{\text{dev}}$, actions a_h^{dev}, probabilities p_h^{dev}
    and costs $\hat{c}_h^{\text{dev}} = 0$ for $h = 1 \dots \text{env}.H$
  for all time steps $h = 1 \dots \text{env}.H$ **do**
    $\boldsymbol{x}_h^{\text{dev}} \leftarrow$ env.STATEFEATURES {computed by an RNN}
    $(a_h^{\text{dev}}, p_h^{\text{dev}}) \leftarrow \text{CB.ACT}(\pi^{\text{learn}}, \boldsymbol{x}_h^{\text{dev}})$
    $\hat{c}_h^{\text{dev}} \leftarrow \text{CB.COST}(\pi_{t-1}^{\text{learn}}, \boldsymbol{x}_h^{\text{dev}}, a_h^{\text{dev}})$
    env.STEP(a_h^{dev}) {updates environment and internal state of the RNN }
  end for
  $\ell_h^{\text{residual}} \leftarrow$ env.FINALLOSS $- \sum_{h' \neq h} \hat{c}^{\text{dev}}(h')$ for all h
  $\pi_t^{\text{learn}} \leftarrow \text{CB.UPDATE}(\pi_{t-1}^{\text{learn}}, \boldsymbol{x}_h^{\text{dev}}, a_h^{\text{dev}}, p_h^{\text{dev}}, \ell_h^{\text{residual}})$ for all h
end for
Return average policy $\bar{\pi} = \frac{1}{T} \sum_t \pi_t^{\text{learn}}$

These two changes means that we update the learned policy π^{learn} multiple times per episode. Empirically, we found this to be crucial for achieving superior performance. Although, the generated samples for the same episode are not independent, this is made-up for by the huge increase in the number of available samples for training (i.e. $T \times H$ samples for multiple deviations versus only T samples in the single deviation mode). The theoretical analysis that precedes still holds in this case, but only makes sense when $\beta = 0$ because there is no longer any distinction between roll-in and roll-out, and so the guarantee reduces to a local optimality guarantee.

6.5 Experimental Setup

We conduct experiments on both reinforcement learning and structured prediction tasks. Our goal is to evaluate how quickly different learning algorithms learn from episodic loss. We implement our models on top of the DyNet neural network optimization package (Neubig et al., 2017). [4]

[4] The code is available at https://github.com/hal3/macarico,https://github.com/hal3/reslope



(a) Blackjack (b) Hex (c) Cart Pole (d) Grid World

Figure 6.3: Reinforcement Learning Tasks

Reinforcement Learning Environments We perform experiments in four standard reinforcement learning environments: Blackjack (classic card game), Hex (two-player board game), Cartpole (aka "inverted pendulum") and Gridworld. Our implementations of these environments are described in §6.5.1 and largely follows the AI Gym (Brockman et al., 2016) implementations. We report results in terms of *cumulative loss*, where loss is $-1\times$*reward* for consistency with the loss-based exposition above and the loss-based evaluation of bandit structured prediction (§6.2.3).

6.5.1 Details on Reinforcement Learning Environments

Blackjack is a card game where the goal is to obtain cards that sum to as near as possible to 21 without going over. Players play against a fixed dealer who hits until they have at least 17. Face cards (Jack, Queen, King) have a point value of 10. Aces can either count as 11 or 1, and a card is called "usable" at 11. The reward for winning is $+1$, drawing is 0, and losing is -1. The world is partially visible: the player can see only their own cards and one of the two initial dealer cards.

Hex is a classic two-player board game invented by Piet Hein and independently by John Nash (Hayward and Van Rijswijck, 2006; Nash, 1952). The board is an $n\times n$ rhombus of hexagonal cells. Players alternately place a stone of their color on any empty cell. To win, a player connects her two opposing sides with her stones. We use $n = 5$; the world is fully visible to the agent, with each hexagon showing as unoccupied, occupied with white or occupied with black. The reward is $+1$ for winning and -1 for losing.

Cart Pole is a classic control problem variously referred to as the "cart-pole", "inverted pendulum", or "pole balancing" problem (Barto et al., 1983). Is is an example of an inherently unstable dynamic system, in which the objective is to control translational forces that position a cart at the center of a finite width track while simultaneously balancing a pole hinged on the cart's top. In this task, a pole is attached by a joint to a cart which moves along a frictionless track (Figure 6.3c). The system is controlled by applying a force of $+1$ or -1 to the cart, thus, we operate in a discrete action space with only two actions. The pendulum starts upright, and the goal is to prevent it from falling over. The episode ends when the pole is more than 15 degrees from the vertical axis, or the cart moves more than 2.4 units from the center. The state is represented by four values indicating the poles position, angle to the vertical axis, and the linear and angular velocities. The total cumulative reward at the end of the episode is the total number of time steps the pole remained upright before the episode terminates.

Grid World consists of a simple 3×4 grid, with a $+1$ reward in the upper-right corner and -1 reward immediately below it; the cell at $(1, 1)$ is blocked (Figure 6.3d). The agent starts at a random unoccupied square. Each step costs 0.05 and the agent has a 10% chance of misstepping. The agent only gets partial visibility of the world: it gets an indicator feature specifying which directions it can step. The only reward observed is the complete sum of rewards over an episode.

Bandit Structured Prediction Environments We also conduct experiments on structured prediction tasks. The evaluation framework we consider is the fully online setup described in (§6.2.3), measuring the degree to which various algorithms can effectively improve by observing only the episodic loss, and effectively balancing exploration and exploitation. We learn from one structured example at a time and we do a single pass over the available examples. We measure performance in terms of average cumulative loss on the online examples as well as on a held-out evaluation dataset. The loss on the online examples

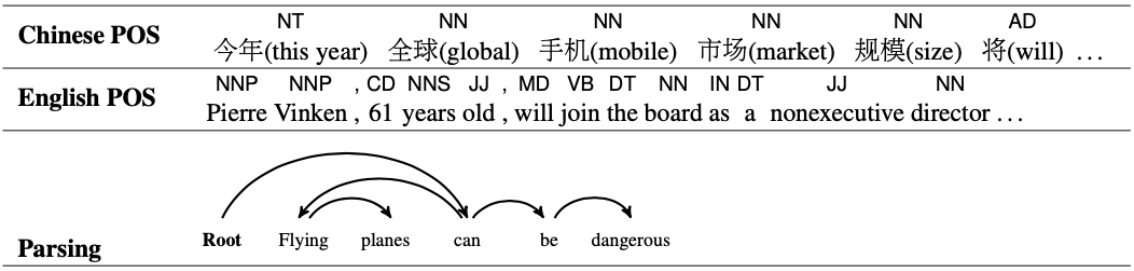


Figure 6.4: Example inputs for part of speech tagging and dependency parsing.

measures how much the algorithm is penalized for unnecessary exploration. We perform experiments on the three tasks described in detail in §6.5.2: English Part of Speech Tagging, English Dependency Parsing and Chinese Part of Speech Tagging.

6.5.2 Structured Prediction Data Sets

English POS Tagging we conduct POS tagging experiments over the 45 Penn Treebank (Marcus et al., 1993) tags. We simulate a domain adaptation setting by training a reference policy on the TweetNLP dataset (Owoputi et al., 2013) which achieves good accuracy in domain, but performs badly out of domain. We simulate bandit episodic loss over the entire Penn Treebank Wall Street Journal (sections $02 \rightarrow 21$ and 23), comprising 42k sentences and about one million words. The measure of performance is the average Hamming loss. We define the search space by sequentially selecting greedy part-of-speech tags for words in the sentence from left to right.

Chinese POS Tagging we conduct POS tagging experiments over the Chinese Penn Treebank (3.0) (Xia, 2000) tags. We simulate a domain adaptation setting by training a reference policy on the Newswire domain from the Chinese Treebank Dataset (Xue et al., 2005) and simulate bandit episodic feedback from the spoken conversation domain. We simulate bandit episodic loss over 40k sentences and about 300k words. The measure of performance is the average Hamming loss. We define the search space by sequentially selecting greedy part-of-speech tags for words in the sentence from left to right.

English Dependency Parsing For this task, we assign a grammatical head (i.e. parent) for each word in the sentence. We train an arc-eager dependency parser (Nivre, 2003) which chooses among (at most) four actions at each state: Shift, Reduce, Left or Right. The reference policy is trained on the TweetNLP dataset and evaluated on the Penn Treebank corpus. The loss is the unlabeled attachment score (UAS), which measures the fraction of words that are assigned the correct parent.

In all structured prediction settings, the feature representation begins with pretrained (and non-updated) embeddings. For English, these are the 6gb Glove embeddings (Pennington et al., 2014); for Chinese, these are the FastText embeddings (Joulin et al., 2016). We then run a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) over the input sentence. The input features for labeling the nth word in POS tagging experiments are the biLSTM representations at position n. The input features for dependency actions are a concatenation of the biLSTM features of the next word on the buffer and the two words on the top of the stack.

6.5.3 Comparative Algorithms

We compare against three common reinforcement learning algorithms: Reinforce (Williams, 1992) with a baseline whose value is an exponentially weighted running average of rewards; Proximal Policy Optimization (PPO) (Schulman et al., 2017); and Advantage Actor-Critic (A2C) (Mnih et al., 2016). For the structured prediction experiments, since the bandit feedback is simulated based on labeled data, we can also estimate an "upper bound" on performance by running a supervised learning algorithm that uses full information (thus forgoing issues of both exploration/exploitation and credit assignment). We run supervised DAgger to obtain such an upper bound.

6.5.4 Policy Architecture

In all cases, our policy is a recurrent neural network (Elman, 1990) that maintains a real-valued hidden state and combines: (a) its previous hidden state, (b) the features from the environment (described for each environment in the preceding sections), and (c) an embedding of its previous action. These form a new hidden state, from which a prediction is made. Formally, at time step h, $\boldsymbol{v}_h$ is the hidden state representation, $f(\text{state}_h)$ are the features from the environment and a_h is the action taken. The recursion is:

$$\boldsymbol{v}_0 = \textbf{const} \quad ; \quad \boldsymbol{v}_{h+1} = \mathit{ReLU}\left(\mathbf{A}\left[\, \boldsymbol{v}_h \;,\; \boldsymbol{f}(\text{state}_h) \;,\; \textbf{emb}(a_h) \,\right]\right) \tag{6.20}$$

Here, $\mathbf{A}$ is a learned matrix, **const** is an initial (learned) state, **emb** is a (learned) action embedding function, and *ReLU* is a rectified linear unit applied element-wise.

Given the hidden state $\boldsymbol{v}_h$, an action must be selected. This is done using a simple feedforward network operating on $\boldsymbol{v}_h$ with either no hidden layers (in which case the output vector is $\boldsymbol{o}_h = \mathbf{B}\boldsymbol{v}_h$) or a single hidden layer (where $\boldsymbol{o}_h = \mathbf{B}_2\, \mathit{ReLU}(\mathbf{B}_1\boldsymbol{v}_h)$). In the case of RESLOPE and DAgger, which expect *cost estimates* as the output of the policy, the output values $\boldsymbol{o}_h$ are used as the predicted costs (and a_h might be the argmin of these costs when operating greedily). In the case of Reinforce, PPO and A2C, which expect action probabilities, these are computed as $\mathit{softmax}(-\boldsymbol{o}_h)$ from which, for instance, an action a_h is sampled.

Details on optimization, hyperparameters and "deep learning tricks" are reported in §6.5.5.

6.5.5 Optimization, Hyperparameter Selection and "Tricks"

We optimize all parameters of the model using the Adam[5] optimizer (Kingma and Ba, 2014), with a tuned learning rate, a moving average rate for the mean of $\beta_1 = 0.9$ and for the variance of $\beta_2 = 0.999$; epsilon (for numerical stability) is

[5] We initially experimented also with RMSProp (Tieleman and Hinton, 2012) and AdaGrad (Duchi et al., 2011) but Adam consistently performed as well or better than the others on all tasks.

fixed at $1e-8$ (these are the DyNet defaults). The learning rate is tuned in the range $\{0.050.01, 0.005, 0.001, 0.0005, 0.0001\}$.

For the structured prediction experiments, the following input features hyperparameters are tuned:

- Word embedding dimension $\in \{50, 100, 200, 300\}$ (for the Chinese embeddings, which come only in 300 dimensional versions, we took the top singular vectors to reduce the dimensionality).
- BiLSTM dimension $\in \{50, 150, 300\}$
- Number of BiLSTM layers $\in \{1, 2\}$
- Pretraining: DAgger or AggreVaTe initialization with probability of rolling in with the reference policy $\in \{0.0, 0.999^N, 0.99999^N, 1.0\}$, where N is the number of examples
- Policy RNN dimension $\in \{50, 150, 300\}$
- Number of policy layers $\in \{1, 2\}$
- Roll-out probability $\beta \in \{0.0, 0.5, 1.0\}$

For each task, the network architecture that was optimal for supervised pretraining was fixed and used for all bandit learning experiments[6].

For the reinforcement learning experiments, we tuned:

- Policy RNN dimension $\in \{20, 50, 100\}$
- Number of policy layers $\in \{1, 2\}$

[6]English POS tagging and dependency parsing: DAgger 0.99999^N, 300 dim embeddings, 300 dim 1 layer LSTM, 2 layer 300 dimensional policy; Chinese POS tagging: DAgger 0.999^N, 300 dim embeddings, 50 dim 2 layer LSTM, 1 layer 50 dimensional policy).

Some parameters we do not tune: the nonlinearities used, the size of the action embeddings (we use 10 in all cases), the input RNN form for the text experiments (we always use LSTM instead of RNN or GRU based on preliminary experiments). We do not regularize our models (weight shrinkage only reduced performance in initial experiments) nor do we use dropout. Pretraining of the structured prediction models ran for 20 passes over the data with early stopping based on held-out loss. The state of the optimizer was reset once bandit learning began.

The variance across difference configurations was relatively small across RL tasks, so we chose a two layer policy with 20 dimensional vectors for all RL tasks.

Each algorithm also has a set of hyperparameters; we tune them as below:

- Reinforce: with baseline or without baseline
- A2C: a multiplier τ on the relative importance of actor loss and critic loss:
$$\tau \in \{0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$$
- PPO: with baseline or without baseline; and epsilon parameter
$$\epsilon \in \{0.01, 0.05, 0.1, 0.2, 0.4, 0.8\}$$
- RESLOPE: update strategy (IPS, DR, MTR) and exploration strategy (uniform, Boltzmann or Bootstrap)

In each reinforcement/bandit experiment, we *optimistically* pick algorithm hyperparameters and learning rate based on final evaluation criteria, noting that this likely provides unrealistically optimistic performance for *all* algorithms. We perform 100 replicates of every experiment in the RL setting and 20 replicates in the structured prediction setting. We additionally ablate various aspects of RESLOPE in §6.6.2.

We employ only two "tricks," both of which are defaults in dynet: gradient clipping (using the default dynet settings) and smart parameter initialization (dynet uses Glorot initialization (Glorot and Bengio, 2010)).

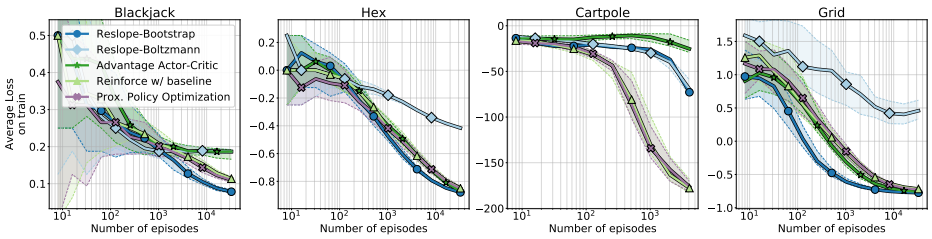


Figure 6.5: Average loss during learning on the four RL problems. Shaded regions are empirical quartiles over the experimental replicates with different random seeds.

6.6 Experimental Results

We study several questions empirically: 1. How does RESIDUAL LOSS PREDICTION compare to policy gradient methods on reinforcement learning and bandit structured prediction tasks? (§6.6.1) 2. What's the effect of ablating various parts of the RESLOPE approach, including multiple deviations? (§6.6.2) 3. Does RESLOPE succeed in learning a good representation of the loss? (§6.6.4)

6.6.1 Reinforcement Learning and Bandit Structured Prediction Results

In our first set of experiments, we compare RESLOPE to the competing approaches on the four reinforcement learning tasks described above. Figure 6.5 shows the results. In Blackjack, Hex and Grid, RESLOPE outperforms all the competing approaches with lower loss earlier in the learning process (though for Hex and Grid they all finish at the same near-optimal policy). For Cartpole, RESLOPE significantly *underperforms* both Reinforce and PPO.[7] Furthermore, in both Blackjack and Grid, the bootstrap exploration significantly improves upon Boltzmann exploration. In general, both RESLOPE performs quite well.In these experiments, PPO performs nearly identically to Reinforce. This happens because all of our experiments use a minibatch size of one. When PPO is run with a minibatch

[7]It is not entirely clear to us yet why this happens. We found that RESLOPE performs as well as Reinforce and PPO if we (a) replace the loss with one centered around zero and (b) replace the RNN policy with a simpler feed-forward network, but we do not include these results in the figure to keep the experiments consistent.

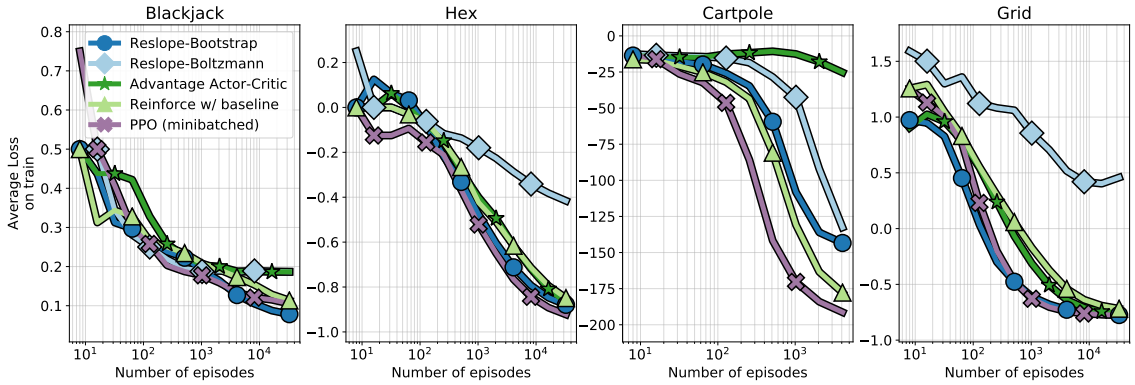


Figure 6.6: Average loss during learning on the four RL problems, including PPO with minibatching. (None of the other algorithms use minibatching, so the comparison is somewhat unfair.)

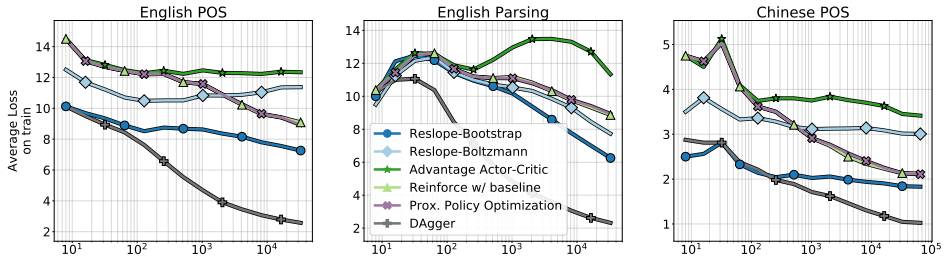


Figure 6.7: Average loss during learning for three bandit structured prediction problems. Also included are supervised learning results with DAgger.

size of one, it reduces to *exactly* Reinforce. We also have conducted experiments with PPO with larger minibatches; these results are reported in the appendix in Figure 6.6. In those experiments, we adjusted the minibatch size and number of epochs to match exactly with the PPO algorithm described in Schulman et al. (2017). In each iteration, each of N actors collect T timesteps of data. Then we construct the surrogate loss on these NT time steps of data, and optimize it with minibatch Adam for K epochs. With these adjustments, PPO's performance falls between RESLOPE and Reinforce on Blackjack, slightly superior to RESLOPE on Hex, better than everything on Cartpole, and roughly equivalent to RESLOPE on Gridworld. We were, unfortunately, unable to conduct these experiments in the structured prediction setting, because the state memoization necessary to implement PPO with large/complex environments overflowed our system's memory quite quickly.

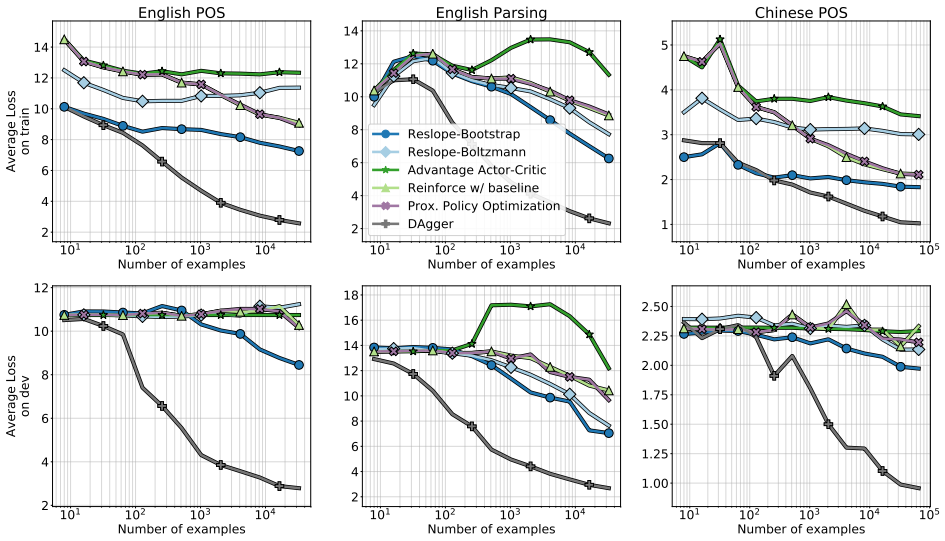


Figure 6.8: Average loss (top) and heldout loss (bottom) during learning for three bandit structured prediction problems. Also included are supervised learning results with DAgger. The main difference between the training loss and the development loss is that in the development data, the system needn't explore, and so the gaps in algorithms which explore different amounts (e.g., especially on English POS tagging) disappear.

In our second set of experiments, we compare the same algorithms *plus* the fully supervised DAgger algorithm on the three structured prediction problems; the results are in Figure 6.7. Here, we can observe RESLOPE significantly outperforming all alternative algorithms (except, of course, DAgger) on training loss (also on heldout (development) loss; see Figure 6.8 in the appendix). There is still quite a gap to fully supervised learning, but nonetheless RESLOPE is able to reduce training error significantly on all tasks: by over 25% on English POS, by about half on English dependency parsing, and by about 10% on Chinese POS tagging.

6.6.2 Ablation of RESIDUAL LOSS PREDICTION

In our construction of RESLOPE, there are several tunable parameters: which contextual bandit learner to use (IPS, DR, MTR), which exploration strategy (Uniform, Boltzmann, Bootstrap), and, for Bootstrap, whether to do greedy prediction and/or greedy

	Reinforcement Learning				Bandit SP		
	Blackjack	**Cartpole**	**Grid**	**Hex**	**Zh-POS**	**En-Dep**	**En-POS**
total loss	0.17	-28.0	0.69	-0.88	1.8	6.3	7.3
loss std	0.021	23.0	0.74	0.008	0.019	0.58	0.77
→ MTR	-1.55	-0.105	-0.783	2.88	0.023	1.56	0.661
→ IPS	-1.81	0.77	-0.28	0.427	282.0	13.2	17.6
→ Boltzmann	2.85	0.263	0.184	54.8	275.0	14.1	18.3
→ Uniform	10.8	0.28	0.566	104.0	285.0	16.1	13.8
– g-predict	-0.638	0.362	-0.31	-0.151	0.236	0.314	0.596
– g-update	1.03	0.508	-0.158	2.24	7.11	3.87	2.79

Table 6.1: Results of ablating various parts of the RESLOPE approach. Columns are tasks. The first two rows are the cumulative average loss over multiple runs and its standard deviation. The numbers in the rest of the column measure how much it hurts (positive number) or helps (negative number) to ablate the corresponding parameter. To keep the numbers on a similar scale, the changes are reported as *multiples* of the standard deviation. So a value of 2.0 means that the cumulative loss gets worse by an additive factor of two standard deviations.

update. In Table 6.1 (in the Appendix), we show the results on all tasks for ablating these various parameters. For the purpose of the ablation, we *fix* the "baseline" system as: DR, Bootstrap, and with both greedy prediction and greedy updates, though this is not uniformly the optimal setting (and therefore these numbers may differ slightly from the preceding figures). The primary take-aways from these results are: (1) MTR and DR are competitive, but IPS is *much* worse; (2) Bootstrap is much better than either other exploration method (especially uniform, not surprisingly); (3) Greedy prediction is a bit of a wash, with only small differences either way; (4) Greedy update is important. In §6.6.3, we consider the effect of single vs multiple deviations and observe that significant importance of multiple deviations for all algorithms, with Reinforce and PPO behaving quite poorly with only single deviations.

6.6.3 Effect of Single vs Multiple Deviations

Next, we consider the single-deviation version of RESLOPE (6) versus the multiple-deviation version (7). To enable comparison with alternative algorithms, we also experi-

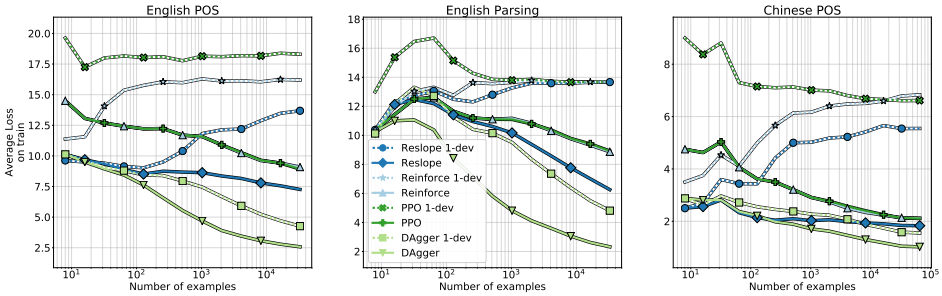


Figure 6.9: The empirical effect of multiple deviations for different algorithms.

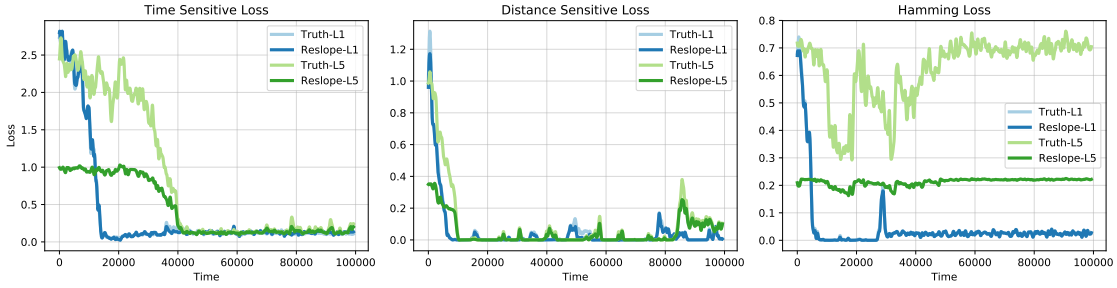


Figure 6.10: Empirical effect of additive vs non-additive loss functions. Performance is better when the loss is additive (blue) vs non-additive (green). The x-axis shows the number of episodes and the y-axis measures the incremental loss using the true loss function (light colors) and using RESLOPE (dark colors). If RESLOPE worked perfectly, these would coincide.

ment with variants of Reinforce, PPO and DAgger that are only allowed single deviations as well (also chosen uniformly at random). The results are shown in Figure 6.9. Not surprisingly, all algorithms suffer when only allowed single deviations. PPO makes things worse over time (likely because its updates are very conservative, such that even in the original PPO paper the authors advocate multiple runs over the same data), as does Reinforce. DAgger still learns, though more slowly, when only allowed a single deviation. RESLOPE behaves similarly though not quite as poorly. Overall, this suggests that even though the samples generated with multiple deviations by RESLOPE are no longer independent, the gain in number of samples more than makes up for this.

6.6.4 Evaluating the Learned Loss Representation

In our final set of experiments, we study RESLOPE's performance under different—and especially non-additive—loss functions. Our goal is to investigate RESLOPE's ability to learn good representations for the episodic loss. We consider the following different

incremental loss functions for each time step: Hamming (0/1 loss at each position), Time-Sensitive (cost for an error at position h is equal to h) and Distance-Sensitive (cost for predicting $\hat{a}$ instead of a is $|\hat{a} - a|$). To combine these per-stop losses into a per-trajectory loss $\boldsymbol{\tau}$ of length H, we compute the H-dimensional loss vector $\boldsymbol{\ell}$ suffered by RESLOPE along this trajectory. To consider both additive and non-additive combinations, we consider Lp norms of this loss vector. When the norm is L1, this is simple additive loss. More generally, we consider $\ell(\boldsymbol{\tau}) = \sqrt[p]{\sum_{t=1}^{t=H} \boldsymbol{\ell}^p(t)}$ for any $p > 0$.

6.6.4.1 Synthetic data for Evaluating the Learned Loss Representation

Experiments were conducted on a synthetic sequence labeling dataset. Input sequences are random integers (between one and ten) of length 6. The ground truth label for the hth word is the corresponding input mod 4. We generate 16k training sequences for this experiment. We run RESLOPE with bootstrap sampling in multiple deviation mode. We use the MTR cost estimator, and optimize the policies using ADAM with a learning rate of 0.01.

We run six different experiments using different incremental and episodic loss functions. For each incremental loss function (i.e. hamming, time sensitive, distance sensitive) we run two experiments: using the total hamming loss (additive) and an Lp norm of five (non-additive). Results are presented in Figure 6.10. We observe the following. RESLOPE can always learn the optimal representation for the incremental loss when the episodic loss function is additive. This is the case for all the three incremental loss functions: hamming, time sensitive, and distance sensitive. Learning is faster when the episodic loss function is additive. While RESLOPE is still able to learn a good representation even when using the L5 norm loss, this happens much later in comparison to the additive loss function (40k time steps for L5 norm vs 20k for total hamming loss). Not surprisingly, performance degrades as the episodic loss function becomes non-additive. This is most acute when using L-5 norm with the incremental hamming loss. This is expected as in the distance

and time sensitive loss functions, RESLOPE observes a smoother loss function and learns to distinguish between different time steps based on the implicit encoding of time and distance information in the observed loss. RESLOPE can still learn a good representation for smoother episodic loss functions. This is shown empirically for time and distance sensitive loss functions.

6.6.4.2 Evaluating the Learned Loss Representation for Grid World

In this section, we study RESLOPE's performance under different—and especially non-additive—loss functions. This experiment is akin to the experimental setting in section 6.6.4, however it's performed on the grid world reinforcement learning environment, where the quantitative aspects of the loss function is well understood.

We study a simple 4×4 grid, with a $+1$ reward in the upper-right corner and -1 reward immediately below it; the cells at $(1,1)$ and $(2,1)$ are blocked. The agent starts at a random position in the grid. Each step costs $+0.05$ and the probability of success is 0.9. The agent has full visibility of the world: it knows its horizontal and vertical position in the grid.

We consider two different episodic reward settings:

1. The only reward observed is the complete sum of losses over an episode. (additive setting);

2. The only reward observed is the L5 norm of the vector of losses over an episode (non-additive setting).

Results are shown in Figure 6.11. Results are very similar to the structured prediction setting (section 6.6.4). Performance is better when the loss is additive (blue) vs non-additive (green).

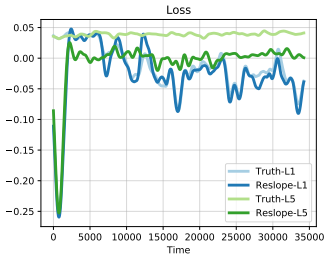


Figure 6.11: Empirical effect of additive vs non-additive loss functions. Performance is better when the loss is additive (blue) vs non-additive (green). The x-axis shows the number of episodes and the y-axis measures the incremental loss using the true loss function (light colors) and using RESLOPE (dark colors). If RESLOPE worked perfectly, these would coincide.

6.7 Related Work and Discussion

RESIDUAL LOSS PREDICTION builds most directly on the bandit learning to search frameworks LOLS (Chang et al., 2015) and BLS (Sharaf and Daumé, 2017). The "bandit" version of LOLS was analyzed theoretically but not empirically in the They addressed this by requiring additional feedback from the user, which worked well empirically but did not enjoy any theoretical guarantees. RESLOPE achieves the best of both worlds: a strong regret guarantee, good empirical performance, and no need for additional feedback. The key ingredient for making this work is using the residual loss structure together with strong base contextual bandit learning algorithms.

A number of recent algorithms have updated "classic" learning to search papers with deep learning underpinnings (Wiseman and Rush, 2016; Leblond et al., 2017). These aim to incorporate sequence-level global loss function to mitigate the mismatch between training and test time discrepancies, but only apply in the fully supervised setting. Mixing of supervised learning and reinforcement signals has become more popular in structured prediction recently, generally to do a better job of tuning for a task-specific loss using either Reinforce (Ranzato et al., 2015) or Actor-Critic (Bahdanau et al., 2016a). The bandit variant of the structured prediction problem was studied by Sokolov et al. (2016a), who proposed a reinforce method for optimizing different structured prediction models under bandit feedback in a log-linear structured prediction model.

A standard technique for dealing with sparse and episodic reward signals is reward shaping (Ng et al., 1999): supplying additional rewards to a learning agent to guide its learning process, beyond those supplied by the underlying environment. Typical reward shaping is hand-engineered; RESLOPE essentially learns a good task-specific reward shaping automatically. The most successful baseline approach we found is Proximal Policy Optimization (PPO, (Schulman et al., 2017)), a variant of Trust Region Policy Optimization (TRPO, (Schulman et al., 2015)) that is more practical. Experimentally we have seen RESLOPE to typically learn more quickly than PPO. Theoretically both have useful guarantees of a rather incomparable nature.

Since RESLOPE operates as a reduction to a contextual bandit oracle, this allows it to continually improve as better contextual bandit algorithms become available, for instance work of Syrgkanis et al. (2016b) and Agarwal et al. (2014). Although RESLOPE is quite effective, there are a number of shortcomings that need to be addressed in future work. For example, the bootstrap sampling algorithm is prohibitive in terms of both memory and time efficiency. One approach for tackling this would be using the amortized bootstrap approach by Nalisnick and Smyth (2017), which uses amortized inference in conjunction with implicit models to approximate the bootstrap distribution over model parameters. There is also a question of whether the reduction to contextual bandits creates "reasonable" contextual bandit problems in conjunction with RNNs. While some contextual bandit algorithms assume strong convexity or linearity, the ones we employ operate on arbitrary policy classes, provided a good cost-sensitive learner exists. The degree to which this is true will vary by neural network architecture, and what can be guaranteed (e.g., no regret full-information online neural learning). A more significant problem in the multi-deviation setting is that as RESLOPE learns, the residual costs will change, leading to a shifting distribution of *costs*; in principle this could be addressed using CB algorithms that work in adversarial settings (Syrgkanis et al., 2016a,b), but largely remains an open challenge.

RESLOPE is currently designed for discrete action spaces. Extension to continuous action spaces (Levine et al., 2016; Lillicrap et al., 2015) remains an open problem.

In the next chapter (chapter 7), we validate our thesis statement in §1.5 by studying a different form of learning with minimal supervision: active imitation learning. We present a learning algorithm that leverages access to a noisy heuristic to minimize the cost of querying a more expensive expert for labels. Imitation learning provides a more efficient learning approach in comparison to reinforcement learning by leveraging access to an expert at training time.

Chapter 7: Active Imitation Learning with Noisy Guidance

7.1 Introduction

In chapter 6 we presented a reinforcement learning algorithm for learning with very sparse reward signals. Reinforcement learning takes the trial-and-error approach and uses the end loss / reward as supervised signal to evaluate how good a policy is. However, sometimes, it is much harder to quantify the value of a certain behavior than to demonstrate the desired behavior. For example, it is not clear exactly how bad it is to drive slightly off the road, but it is easy to show a good driving path. Imitation learning assumes access to an expert who shows good actions to take in any given state. In this chapter we present another minimally supervised learning algorithm based on imitation learning. Our main goal is to minimize the labeling cost for the expert by leveraging access to a noisy heuristic.

Structured prediction methods learn models to map inputs to complex outputs with internal dependencies, typically requiring a substantial amount of expert-labeled data. To minimize annotation cost, we focus on a setting in which an expert provides labels for *pieces* of the input, rather than the complete input (e.g., labeling at the level of words, not sentences). A natural starting point for this is imitation learning-based "learning to search" approaches to structured prediction (Daumé et al., 2009; Ross et al., 2011a; Bengio et al., 2015a; Leblond et al., 2018). In imitation learning, training proceeds by incrementally producing structured outputs one piece at a time and, at every step, asking the expert "what would you do here?" and learning to mimic that choice. This interactive model comes at a

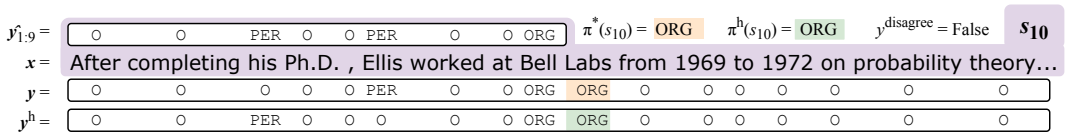


Figure 7.1: A named entity recognition example (from the Wikipedia page for Clarence Ellis). $\boldsymbol{x}$ is the input sentence and $\boldsymbol{y}$ is the (unobserved) ground truth. The predictor π operates left-to-right and, in this example, is currently at state s_{10} to tag the 10th word; the state s_{10} (highlighted in purple) combines $\boldsymbol{x}$ with $\hat{\boldsymbol{y}}_{1:9}$. The heuristic makes two errors at $t = 4$ and $t = 6$. The heuristic label at $t = 10$ is y^h_{10} =ORG. Under Hamming loss, the cost at $t = 10$ is minimized for $a =$ ORG, which is therefore the expert action (if it were queried). The label that would be provided for s_{10} to the difference classifier is 0 because the two policies agree.

substantial cost: the expert demonstrator must be continuously available and must be able to answer a potentially large number of queries.

We reduce this annotation cost by only asking an expert for labels that are truly needed; our algorithm, Learning to Query for Imitation (LEAQI) [1] achieves this by capitalizing on two factors. First, as is typical in active learning (see §5.5), LEAQI only asks the expert for a label when it is uncertain. Second, LEAQI assumes access to a *noisy heuristic* labeling function (for instance, a rule-based model, dictionary, or inexpert annotator) that can provide low-quality labels. LEAQI operates by always asking this heuristic for a label, and only querying the expert when it thinks the expert is likely to disagree with this label. It trains, simultaneously, a *difference classifier* (Zhang and Chaudhuri, 2015) that predicts disagreements between the expert and the heuristic (see Figure 7.1).

The challenge in learning the difference classifier is that it must learn based on one-sided feedback: if it predicts that the expert is likely to agree with the heuristic, the expert is not queried and the classifier cannot learn that it was wrong. We address this one-sided feedback problem using the Apple Tasting framework (Helmbold et al., 2000), in which errors (in predicting which apples are tasty) are only observed when a query is made (an apple is tasted). Learning in this way particularly important in the general case

[1] Code is available at: https://github.com/xkianteb/leaqi

where the heuristic is likely not just to have high variance with respect to the expert, but is also statistically biased.

Experimentally (§6.6), we consider three structured prediction settings, each using a different type of heuristic feedback. We apply LEAQI to: English named entity recognition where the heuristic is a rule-based recognizer using gazetteers (Khashabi et al., 2018); English scientific keyphrase extraction, where the heuristic is an unsupervised method (Florescu and Caragea, 2017); and Greek part-of-speech tagging, where the heuristic is a small dictionary compiled from the training data (Zesch et al., 2008; Haghighi and Klein, 2006). In all three settings, the expert is a simulated human annotator. We train LEAQI on all three tasks using fixed BERT (Devlin et al., 2019) features, training only the final layer (because we are in the regime of small labeled data). The goal in all three settings is to minimize the number of words the expert annotator must label. In all settings, we're able to establish the efficacy of LEAQI, showing that it can indeed provide significant label savings over using the expert alone and over several baselines and ablations that establish the importance of both the difference classifier and the Apple Tasting paradigm.

7.2 Background and Related Work

We review first the use of imitation learning for structured prediction, then online active learning, and finally applications of active learning to structured prediction and imitation learning problems.

7.2.1 Learning to Search

The learning to search approach to structured prediction casts the joint prediction problem of producing a complex output as a sequence of smaller classification problems (Ratnaparkhi, 1996; Collins and Roark, 2004a; Daumé et al., 2009). For instance, in the named entity recognition example from Figure 7.1, an input sentence $\boldsymbol{x}$ is labeled one

8 DAgger(Π, N, $\langle\beta_i\rangle_{i=0}^{N}$, $\pi^\star$)

initialize dataset $D = \{\}$
initialize policy $\hat{\pi}_1$ to any policy in Π
for $i = 1 \ldots N$ **do**
    ▷ *stochastic mixture policy*
    Let $\pi_i = \beta_i \pi^\star + (1 - \beta_i)\hat{\pi}_i$
    Generate a T-step trajectory using π_i
    Accumulate data $D \leftarrow D \cup \{(s, \pi^\star(s))\}$ for all s in those trajectories
    Train classifier $\hat{\pi}_{i+1} \in \Pi$ on D
end for
return best (or random) $\hat{\pi}_i$

word at a time, left-to-right. At the depicted state (s_{10}), the model has labeled the first nine words and must next label the tenth word. Learning to search approaches assume access to an oracle policy $\pi^\star$, which provides the optimal label at every position.

In (interactive) imitation learning, we aim to imitate the behavior of the expert policy, $\pi^\star$, which provides the true labels. The learning to search view allows us to cast structured prediction as a (degenerate) imitation learning task, where states are (input, prefix) pairs, actions are operations on the output, and the horizon T is the length of the sequence. States are denoted $s \in \mathcal{S}$, actions are denoted $a \in [K]$, where $[K] = \{1, \ldots, K\}$, and the policy class is denoted $\Pi \subseteq [K]^{\mathcal{S}}$. The goal in learning is to find a policy $\pi \in \Pi$ with small loss on the distribution of states that it, itself, visits.

A popular imitation learning algorithm, DAgger (Ross et al., 2011a), is summarized in 8. In each iteration, DAgger executes a mixture policy and, at each visited state, queries the expert's action. This produces a classification example, where the input is the state and the label is the expert's action. At the end of each iteration, the learned policy is updated by training it on the accumulation of all generated data so far. DAgger is effective in practice and enjoys appealing theoretical properties; for instance, if the number of iterations N is $\tilde{O}(T^2 \log(1/\delta))$ then with probability at least $1 - \delta$, the generalization error of the learned policy is $O(1/T)$ (Ross et al., 2011a, Theorem 4.2).

7.2.2 Active Learning

Active learning has been considered since at least the 1980s often under the name "selective sampling" (Rendell, 1986; Atlas et al., 1990). In agnostic online active learning for classification, a learner operates in rounds (e.g. Balcan et al., 2006; Beygelzimer et al., 2009, 2010a). At each round, the learning algorithm is presented an example $\boldsymbol{x}$ and must predict a label; the learner must decide whether to query the true label. An effective margin-based approach for online active learning is provided by Cesa-Bianchi et al. (2006) for linear models. Their algorithm defines a sampling probability $\rho = b/(b+z)$, where z is the margin on the current example, and $b > 0$ is a hyperparameter that controls the aggressiveness of sampling. With probability ρ, the algorithm requests the label and performs a perceptron-style update.

Our approach is inspired by Zhang and Chaudhuri's (2015) setting, where two labelers are available: a free weak labeler and an expensive strong labeler. Their algorithm minimizes queries to the strong labeler, by learning a difference classifier that predicts, for each example, whether the weak and strong labelers are likely to disagree. Their algorithm trains this difference classifier using an example-weighting strategy to ensure that its Type II error is kept small, establishing statistical consistency, and bounding its sample complexity.

This type of learning from one-sided feedback falls in the general framework of *partial-monitoring games*, a framework for sequential decision making with imperfect feedback. Apple Tasting is a type of partial-monitoring game (Littlestone and Warmuth, 1989), where, at each round, a learner is presented with an example $\boldsymbol{x}$ and must predict a label $\hat{y} \in \{-1, +1\}$. After this prediction, the true label is revealed *only* if the learner predicts $+1$. This framework has been applied in several settings, such as spam filtering and document classification with minority class distributions (Sculley, 2007). Sculley (2007) also conducts a through comparison of two methods that can be used to address the

one-side feedback problem: label-efficient online learning (Cesa-Bianchi et al., 2006) and margin-based learning (Vapnik, 1982).

7.2.3 Active Imitation & Structured Prediction

In the context of structured prediction for natural language processing, active learning has been considered both for requesting full structured outputs (e.g. Thompson et al., 1999; Culotta and McCallum, 2005; Hachey et al., 2005) and for requesting only pieces of outputs (e.g. Ringger et al., 2007; Bloodgood and Callison-Burch, 2010). For sequence labeling tasks, Haertel et al. (2008) found that labeling effort depends both on the number of words labeled (which we model), plus a fixed cost for reading (which we do not).

In the context of imitation learning, active approaches have also been considered for at least three decades, often called "learning with an external critic" and "learning by watching" (Whitehead, 1991). More recently, Judah et al. (2012) describe *RAIL*, an active learning-for-imitation-learning algorithm akin to our ACTIVEDAGGER baseline, but which in principle would operate with any underlying i.i.d. active learning algorithm (not just our specific choice of uncertainty sampling).

7.3 Our Approach: LEAQI

Our goal is to learn a structured prediction model with minimal human expert supervision, effectively by combining human annotation with a noisy heuristic. We present LEAQI to achieve this. As a concrete example, return to Figure 7.1: at s_{10}, π must predict the label of the tenth word. If π is confident in its own prediction, LEAQI can avoid any query, similar to traditional active learning. If π is not confident, then LEAQI considers the label suggested by a noisy heuristic (here: `ORG`). LEAQI predicts whether the true expert label is likely to disagree with the noisy heuristic. Here, it predicts no disagreement and avoids querying the expert.

7.3.1 Learning to Query for Imitation

Our algorithm, LEAQI, is specified in 9. As input, LEAQI takes a policy class Π, a hypothesis class $\mathcal{H}$ for the difference classifier (assumed to be symmetric and to contain the "constant one" function), a number of episodes N, an expert policy $\pi^\star$, a heuristic policy π^{h}, and a confidence parameter $b > 0$. The general structure of LEAQI follows that of DAgger, but with three key differences:

(a) roll-in (Line 7) is according to the learned policy (not mixed with the expert, as that would require additional expert queries),

(b) actions are queried only if the current policy is uncertain at s (Line 12), and

(c) the expert $\pi^\star$ is only queried if it is predicted to disagree with the heuristic π^{h} at s by the difference classifier, or if apple tasting method switches the difference classifier label (Line 15; see §7.3.2).

In particular, at each state visited by π_i, LEAQI estimates z, the certainty of π_i's prediction at that state (see §7.3.3). A sampling probability ρ is set to $b/(b+z)$ where z is the certainty, and so if the model is very uncertain then ρ tends to zero, following (Cesa-Bianchi et al., 2006). With probability ρ, LEAQI will collect *some* label.

When a label is collected (Line 12), the difference classifier h_i is queried on state s to predict if $\pi^\star$ and π^{h} are likely to disagree on the correct action. (Recall that h_1 always predicts disagreement per Line 4.) The difference classifier's prediction, $\hat{d}_i$, is passed to an *apple tasting* method in Line 15. Intuitively, most apple tasting procedures (including the one we use, STAP; see §7.3.2) return $\hat{d}_i$, unless the difference classifier is making many Type II errors, in which case it may return $\neg\hat{d}_i$.

A target action is set to $\pi^{\text{h}}(s)$ if the apple tasting algorithm returns "agree" (Line 17), and the expert $\pi^\star$ is only queried if disagreement is predicted (Line 20). The state and target action (either heuristic or expert) are then added to the training data. Finally, if the

expert was queried, then a new item is added to the difference dataset, consisting of the state, the heuristic action on that state, the difference classifier's prediction, and the ground truth for the difference classifier whose input is s and whose label is whether the expert and heuristic *actually* disagree. Finally, π_{i+1} is trained on the accumulated action data, and h_{i+1} is trained on the difference dataset (details in §7.3.3).

There are several things to note about LEAQI:

- ◇ If the current policy is already very certain, a expert annotator is *never* queried.
- ◇ If a label is queried, the expert is queried only if the difference classifier predicts disagreement with the heuristic, or the apple tasting procedure flips the difference classifier prediction.
- ◇ Due to apple tasting, most errors the difference classifier makes will cause it to query the expert unnecessarily; this is the "safe" type of error (increasing sample complexity but not harming accuracy), versus a Type II error (which leads to biased labels).
- ◇ The difference classifier is only trained on states where the policy is uncertain, which is exactly the distribution on which it is run.

7.3.2 Apple Tasting for One-Sided Learning

The difference classifier $h \in \mathcal{H}$ must be trained (line 27) based on one-sided feedback (it only observes errors when it predicts "disagree") to minimize Type II errors (it should only very rarely predict "agree" when the truth is "disagree"). This helps keep the labeled data for the learned policies unbiased. The main challenge here is that the feedback to the difference classifier is *one-sided*: that is, if it predicts "disagree" then it gets to see the truth, but if it predicts "agree" it never finds out if it was wrong. We use one of (Helmbold et al., 2000)'s algorithms, STAP (see 10), which works by random sampling from apples that are predicted to not be tasted and tasting them anyway (line 12). Formally, STAP

tastes apples that are predicted to be bad with probability $\sqrt{(m+1)/t}$, where m is the number of mistakes, and t is the number of apples tasted so far.

We adapt Apple Tasting algorithm STAP to our setting for controlling the number of Type II errors made by the difference classifier as follows. First, because some heuristic actions are much more common than others, we run a separate apple tasting scheme *per* heuristic action (in the sense that we count the number of error *on this heuristic action* rather than globally). Second, when there is significant action imbalance[2] we find it necessary to skew the distribution from STAP more in favor of querying. We achieve this by sampling from a *Beta* distribution (generalizing the uniform), whose mean is shifted toward zero for more frequent heuristic actions. This increases the chance that Apple Tasting will have on finding bad apples error for each action (thereby keeping the false positive rate low for predicting disagreement).

7.3.3 Measuring Policy Certainty

In step 11, LEAQI must estimate the certainty of π_i on s. Following Cesa-Bianchi et al. (2006), we implement this using a margin-based criteria. To achieve this, we consider π as a function that maps actions to scores and then chooses the action with largest score. The certainty measure is then the difference in scores between the highest and second highest scoring actions:

$$\text{certainty}(\pi, s) = \max_a \pi(s,a) - \max_{a' \neq a} \pi(s,a')$$

7.3.4 Analysis

Theoretically, the main result for LEAQI is an interpretation of the main DAgger result(s). Formally, let d_π denote the distribution of states visited by π, $C(s,a) \in [0,1]$

[2]For instance, in named entity recognition, both the heuristic and expert policies label the majority of words as O (not an entity). As a result, when the heuristic says O, it is very likely that the expert will agree. However, if we aim to optimize for something other than accuracy—like F1—it is precisely these disagreements that we need to find.

be the immediate cost of performing action a in state s, $C_\pi(s) = \mathbb{E}_{a\sim\pi(s)}C(s,a)$, and the total expected cost of π to be $J(\pi) = T\mathbb{E}_{s\sim d_\pi}C_\pi(s)$, where T is the length of trajectories. C is not available to a learner in an imitation setting; instead the algorithm observes an expert and minimizes a surrogate loss $\ell(s,\pi)$ (e.g., ℓ may be zero/one loss between π and $\pi^\star$). We assume ℓ is strongly convex and bounded in $[0,1]$ over Π.

Given this setup assumptions, let $\epsilon_{\text{pol-approx}} = \min_{\pi\in\Pi}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim d_{\pi_i}}\ell(s,\pi)$ be the true loss of the best policy in hindsight, let:

$$\epsilon_{\text{dc-approx}} = \min_{h\in\mathcal{H}}\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{s\sim d_{\pi_i}} \mathit{err}(s,h,\pi^\star(s)\neq\pi^{\text{h}}(s)) \tag{7.1}$$

,

be the true error of the best difference classifier in hindsight, and assuming that the regret of the policy learner is bounded by $\mathit{reg}_{\text{pol}}(N)$ after N steps, Ross et al. (2011a) shows the following[3]:

Theorem 8 (Thm 4.3 of Ross et al. (2011a))**.** *After N episodes each of length T, under the assumptions above, with probability at least $1-\delta$ there exists a policy $\pi\in\pi_{1:N}$ such that:*

$$\mathbb{E}_{s\sim d_\pi}\ell(s,\pi) \leq$$
$$\epsilon_{\textit{pol-approx}} + \mathit{reg}_{\textit{pol}}(N) + \sqrt{(2/N)\log(1/\delta)}$$

This holds regardless of how $\pi_{1:N}$ are trained (Line 26). The question of how well LEAQI performs becomes a question of how well the combination of uncertainty-based sampling and the difference classifier learn. So long as those do a good job on their individual *classification* tasks, DAgger guarantees that the *policy* will do a good job. This is formalized below, where $Q^\star(s,a)$ is the best possible cumulative cost (measured by C) starting in state s and taking action a:

Theorem 9 (Theorem 2.2 of Ross et al. (2011a))**.** *Let u be such that*

$$Q^\star(s,a) - Q^\star(s,\pi^\star(s)) \leq u \tag{7.2}$$

[3]Proving a stronger result is challenging: analyzing the sample complexity of an active learning algorithm that uses a difference classifier—even in the non-sequential setting—is quite involved (Zhang and Chaudhuri, 2015).

Task	Named Entity Recognition	Keyphrase Extraction	Part of Speech Tagging
Language	English (en)	English (en)	Modern Greek (el)
Dataset	CoNLL'03 (Tjong Kim Sang and De Meulder, 2003)	SemEval 2017 Task 10 (Augenstein et al., 2017)	Universal Dependencies Nivre (2018)
# Ex	$14,987$	$2,809$	$1,662$
Avg. Len	14.5	26.3	25.5
# Actions	5	2	17
Metric	Entity F-score	Keyphrase F-score	Per-tag accuracy
Features	English BERT (Devlin et al., 2019)	SciBERT (Beltagy et al., 2019)	M-BERT (Devlin et al., 2019)
Heuristic	String matching against an offline gazeteer of entities from Khashabi et al. (2018)	Output from an unsupervised keyphrase extraction model Florescu and Caragea (2017)	Dictionary from Wiktionary, similar to Zesch et al. (2008) and Haghighi and Klein (2006)
Heur Quality	P 88%, R 27%, F 41%	P 20%, R 44%, F 27%	10% coverage, 67% acc

Table 7.1: An overview of the three tasks considered in our experiments.

for all a and all s with $d_\pi(s) > 0$; then for some $\pi \in \pi_{1:N}$, as $N \to \infty$:

$$J(\pi) \leq J(\pi^\star) + uT\epsilon_{\textit{pol-approx}}$$

Here, u captures the most long-term impact a single decision can have; for example, for average Hamming loss, it is straightforward to see that $u = \frac{1}{T}$ because any single mistake can increase the number of mistakes by at most 1. For precision, recall and F-score, u can be as large as one in the (rare) case that a single decision switches from one true positive to no true positives.

7.4 Experiments

The primary research questions we aim to answer experimentally are:

Q1 Does uncertainty-based active learning achieve lower query complexity than passive learning in the learning to search settings?

Q2 Does learning a difference classifier improve query efficiency over active learning alone?

Q3 Does Apple Tasting successfully handle the problem of learning from one-sided feedback?

Q4 Is the approach robust to cases where the noisy heuristic is uncorrelated with the expert?

Q5 Is casting the heuristic as a policy more effective than using its output as features?

To answer these questions, we conduct experiments on three tasks (see Table 7.1): English named entity recognition, English scientific keyphrase extraction, and low-resource part of speech tagging on Modern Greek (el), selected as a low-resource setting.

7.4.1 Algorithms and Baselines

In order to address the research questions above, we compare LEAQI to several baselines. The baselines below compare our approach to previous methods:

DAGGER. Passive DAgger (8)

ACTIVEDAGGER. An active variant of DAgger that asks for labels only when uncertain. (This is equivalent to LEAQI, but with neither the difference classifier nor apple tasting.)

DAGGER+FEAT. DAGGER with the heuristic policy's output appended as an input feature.

ACTIVEDAGGER+FEAT. ACTIVEDAGGER with the heuristic policy as a feature.

The next set of comparisons are explicit ablations:

LEAQI+NOAT LEAQI with no apple tasting.

LEAQI+NOISYHEUR. LEAQI, but where the heuristic returns a label uniformly at random.

The baselines and LEAQI share a linear relationship. DAGGER is the baseline algorithm used by all algorithms described above but it is very query inefficient with respect to an expert annotator. ACTIVEDAGGER introduces active learning to make DAGGER more query efficient; the delta to the previous addresses Q1. LEAQI+NOAT introduces the difference classifier; the delta addresses Q2. LEAQI adds apple tasting to deal with one-sided learning; the delta addresses Q3. Finally, LEAQI+NOISYHEUR. (vs LEAQI) addresses Q4 and the +FEAT variants address Q5.

7.4.2 Data and Representation

For *named entity recognition*, we use training, validation, and test data from CoNLL'03 (Tjong Kim Sang and De Meulder, 2003), consisting of IO tags instead of BIO tags (the "B" tag is almost never used in this dataset, so we never attempt to predict it) over four entity types: Person, Organization, Location, and Miscellaneous. For *part of speech tagging*, we use training and test data from modern Greek portion of the Universal Dependencies (UD) treebanks (Nivre, 2018), consisting of 17 universal tags[4]. For *keyphrase extraction*, we use training, validation, and test data from SemEval 2017 Task 10 (Augenstein et al., 2017), consisting of IO tags (we use one "I" tag for all three keyphrase types).

In all tasks, we implement both the policy and difference classifier by fine-tuning the last layer of a BERT embedding representation (Devlin et al., 2019). More specifically, for a sentence of length T, $w_1, \ldots, w_T$, we first compute BERT embeddings for each word, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ using the appropriate BERT model: English BERT and M-BERT[5] for named entity and part-of-speech, respectively, and SciBERT (Beltagy et al., 2019) for keyphrase

[4] ADJ, ADP, ADV, AUX, CCONJ, DET, INTJ, NOUN, NUM, PART, PRON, PROPN, PUNCT, SCONJ, SYM, VERB, X.
[5] Multilingual BERT (Devlin et al., 2019)

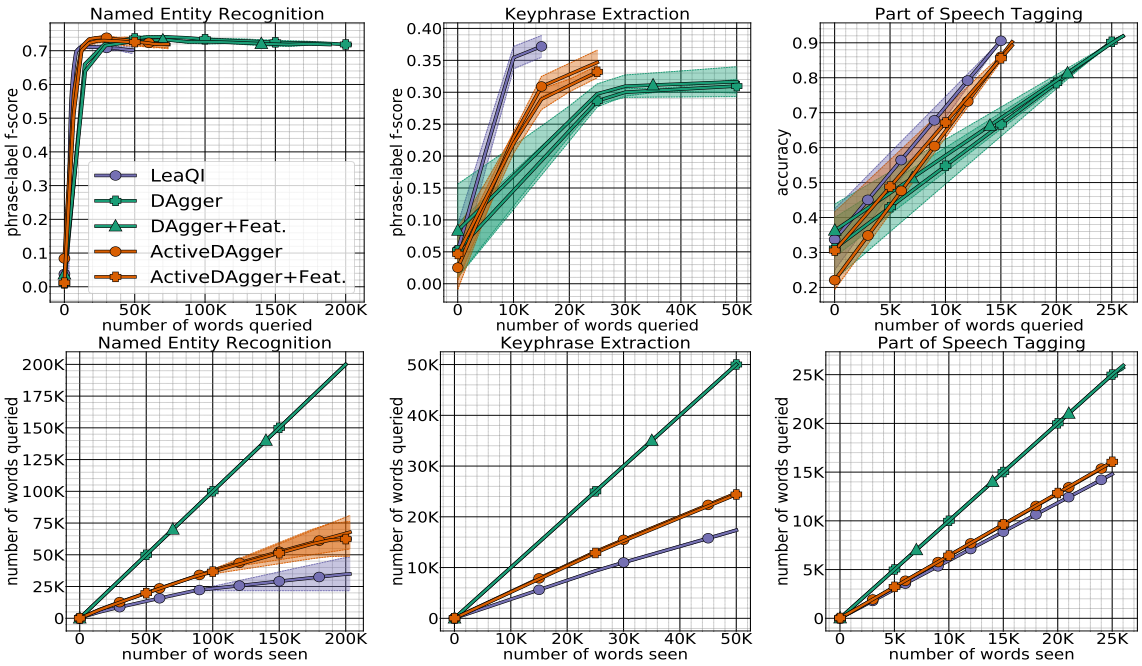


Figure 7.2: Empirical evaluation on three tasks: (left) named entity recognition, (middle) keyphrase extraction and (right) part of speech tagging. The top rows shows performance (f-score or accuracy) with respect to the number of queries to the expert. The bottom row shows the number of queries as a function of the number of words seen.

extraction. We then represent the state at position t by concatenating the word embedding at that position with a one-hot representation of the previous action: $s_t = [\boldsymbol{w}_t; \text{onehot}(a_{t-1})]$. This feature representation is used both for learning the labeling policy and also learning the difference classifier.

7.4.3 Expert Policy and Heuristics

In all experiments, the expert $\pi^\star$ is a simulated human annotator who annotates one word at a time. The expert returns the optimal action for the relevant evaluation metric (F-score for named entity recognition and keyphrase extraction, and accuracy for part-of-speech tagging). We take the annotation cost to be the total number of words labeled.

The heuristic we implement for named entity recognition is a high-precision gazeteer-based string matching approach. We construct this by taking a gazeteer from Wikipedia

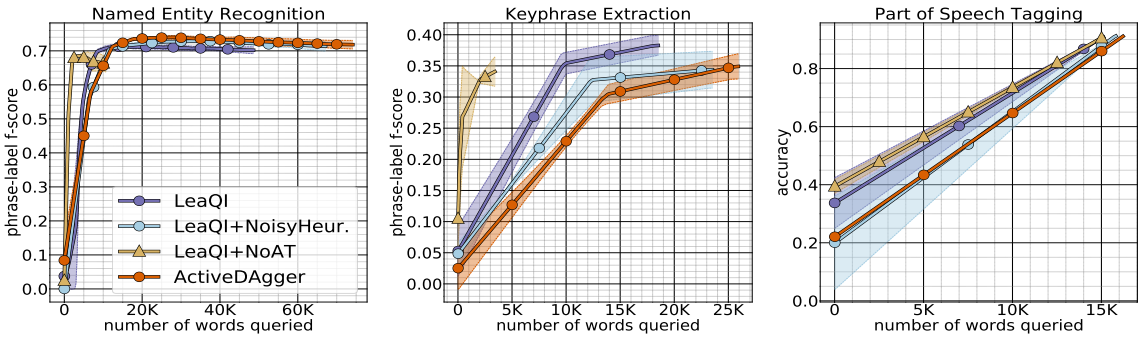


Figure 7.3: Ablation results on (left) named entity recognition, (middle) keyphrase extraction and (right) part of speech tagging. In addition to LEAQI and DAgger (copied from Figure 7.2), these graphs also show LEAQI+NOAT (apple tasting disabled), and LEAQI+NOISYHEUR. (a heuristic that produces labels uniformly at random).

using the CogComp framework (Khashabi et al., 2018), and use FlashText (Singh, 2017) to label the dataset. This heuristic achieves a precision of 0.88, recall of 0.27 and F-score of 0.41 on the training data.

The keyphrase extraction heuristic is the output of an "unsupervised keyphrase extraction" approach (Florescu and Caragea, 2017). This system is a graph-based approach that constructs word-level graphs incorporating positions of all word occurrences information; then using PageRank to score the words and phrases. This heuristic achieves a precision of 0.20, recall of 0.44 and F-score of 0.27 on the training data.

The part of speech tagging heuristic is based on a small dictionary compiled from Wiktionary. Following Haghighi and Klein (2006) and Zesch et al. (2008), we extract this dictionary using Wiktionary as follows: for word w in our training data, we find the part-of-speech y by querying Wiktionary. If w is in Wikitionary, we convert the Wikitionary part of speech tag to a Universal Dependencies tag (see §7.4.6.1), and if word w is not in Wiktionary, we use a default label of "X". Furthermore, if word w has multiple parts of speech, we select the first part of speech tag in the list. The label "X" is chosen 90% of the time. For the remaining 10%, the heuristic achieves an accuracy of 0.67 on the training data.

7.4.4 Experimental Setup

Our experimental setup is online active learning. We make a single pass over a dataset, and the goal is to achieve an accurate system as quickly as possible. We measure performance (accuracy or F-score) after every 1000 words (≈ 50 sentences) on heldout test data, and produce error bars by averaging across three runs and reporting standard deviations.

Hyperparameters for DAGGER are optimized using grid-search on the named entity recognition training data and evaluated on development data. We then fix DAGGER hyperparameters for all other experiments and models. The difference classifier hyperparameters are subsequently optimized in the same manner. We fix the difference classifier hyperparameters for all other experiments.[6]

7.4.5 Experimental Results

The main results are shown in the top two rows of Figure 7.2; ablations of LEAQI are shown in Figure 7.3. In Figure 7.2, the top row shows traditional learning curves (performance vs number of queries), and the bottom row shows the number of queries made to the expert as a function of the total number of words seen.

Active vs Passive (Q1). In all cases, we see that the active strategies improve on the passive strategies; this difference is largest in keyphrase extraction, middling for part of speech tagging, and small for NER. While not surprising given previous successes of active learning, this confirms that it is also a useful approach in our setting. As expected, the active algorithms query far less than the passive approaches, and LEAQI queries the least.

[6] We note that this is a somewhat optimistic hyperparameter setting: in the real world, model selection for active learning is extremely challenging. Details on hyperparameter selection and LEAQI's robustness across a rather wide range of choices are presented in §7.4.6.2, §7.4.7 and §7.4.8 for keyphrase extraction and part of speech tagging.

Heuristic as Features vs Policy (Q5). We see that while adding the heuristic's output as a feature can be modestly useful, it is not uniformly useful and, at least for keyphrase extraction and part of speech tagging, it is not as effective as LEAQI. For named entity recognition, it is not effective at all, but this is also a case where all algorithms perform essentially the same. Indeed, here, LEAQI learns quickly with few queries, but never quite reaches the performance of ActiveDAgger. This is likely due to the difference classifier becoming overly confident too quickly, especially on the "O" label, given the (relatively well known) oddness in mismatch between development data and test data on this dataset.

Difference Classifier Efficacy (Q2). Turning to the ablations (Figure 7.3), we can address Q2 by comparing the ActiveDAgger curve to the LeaQI+NoAT curve. Here, we see that on NER and keyphrase extraction, adding the difference classifier without adding apple tasting results in a far worse model: it learns very quickly but plateaus much lower than the best results. The exception is part of speech tagging, where apple tasting does not seem necessary (but also does not hurt). Overall, this essentially shows that without controlling Type II errors, the difference classifier on it's own does not fulfill its goals.

Apple Tasting Efficacy (Q3). Also considering the ablation study, we can compare LeaQI+NoAT with LeaQI. In the case of part of speech tagging, there is little difference: using apple tasting to combat issues of learning from one sided feedback neither helps nor hurts performance. However, for both named entity recognition and keyphrase extraction, removing apple tasting leads to faster learning, but substantially lower final performance (accuracy or f-score). This is somewhat expected: without apple tasting, the training data that the policy sees is likely to be highly biased, and so it gets stuck in a low accuracy regime.

Robustness to Poor Heuristic (Q4). We compare LeaQI+NoisyHeur to ActiveDAgger. Because the heuristic here is useless, the main hope is that it does not *degrade* performance

below ActiveDAgger. Indeed, that is what we see in all three cases: the difference classifier is able to learn quite quickly to essentially ignore the heuristic and only rely on the expert.

7.4.6 Experimental Details

7.4.6.1 Wiktionary to Universal Dependencies

Table 7.2 shows the conversion we used between the Greek, Modern (el) Wiktionary POS tags and the Universal Dependencies POS tags.

POS Tag Source	Greek Wiktionary	Universal Dependencies
	adjective	ADJ
	adposition	ADP
	preposition	ADP
	adverb	ADV
	auxiliary	AU
	coordinating conjunction	CCONJ
	determiner	DET
	interjection	INTJ
	noun	NOUN
	numeral	NUM
	particle	PART
	pronoun	PRON
	proper noun	pROPN
	punctuation	PUNCT
	subordinating conjunction	SCONJ
	symbol	SYM
	verb	VERB
	other	X
	article	DET
	conjunction	PART

Table 7.2: Conversion between Greek, Modern (el) Wiktionary POS tags and Universal Dependencies POS tags.

7.4.6.2 Hyperparameters

Here we provide a table of all the hyperparameters we considered for LEAQI and baselines models. (see §7.4.4, Table 7.3)

Hyperparameter	Values Considered	Final Value
Policy Learning rate	10^{-3} to 10^{-6}	10^{-6}
Difference Classifier Learning rate h	$0.1, 0.01, 0.001, 0.0001$	0.01
Confidence parameter (b)	0.5, 1, 1.5	0.5

Table 7.3: Hyperparameters for LEAQI

7.4.7 Ablation Study: Difference Classifier Learning Rate

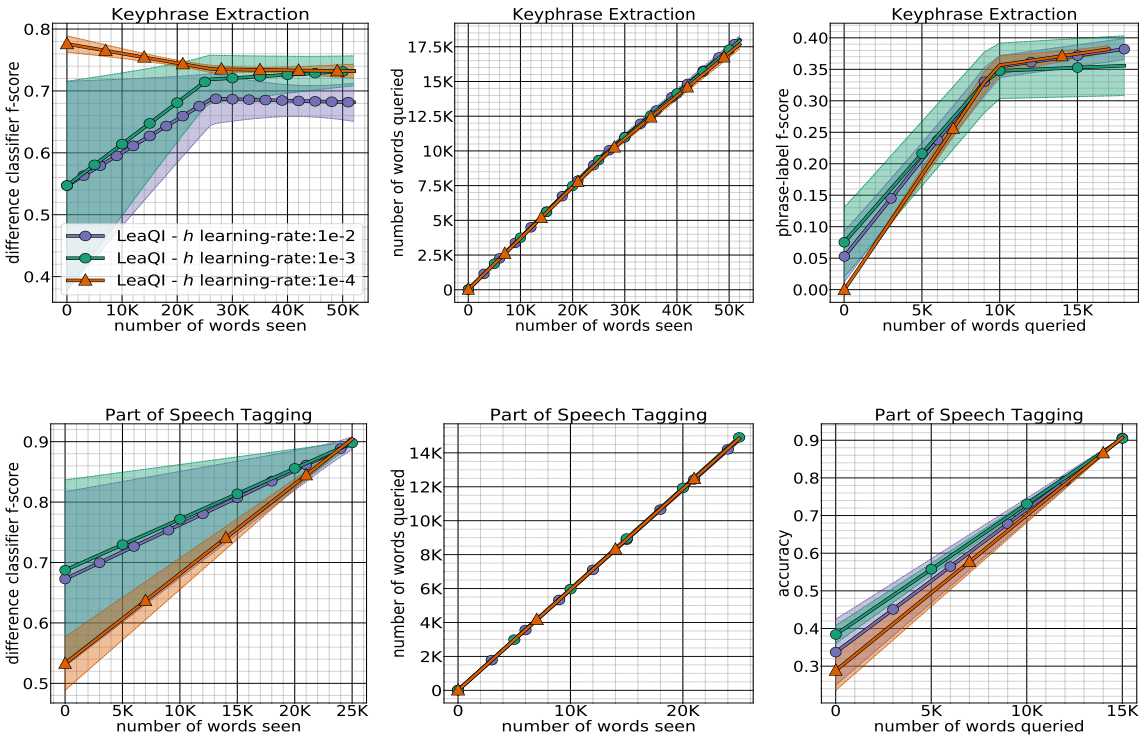


Figure 7.4: (top-row) English keyphrase extraction and (bottom-row) low-resource language part of speech tagging on Greek, Modern (el). We show the performance of using different learning rates for the difference classifier h. These plots indicate that their is small difference in performance depending on the difference classifier learning rate.

7.4.8 Ablation Study: Confidence Parameter b

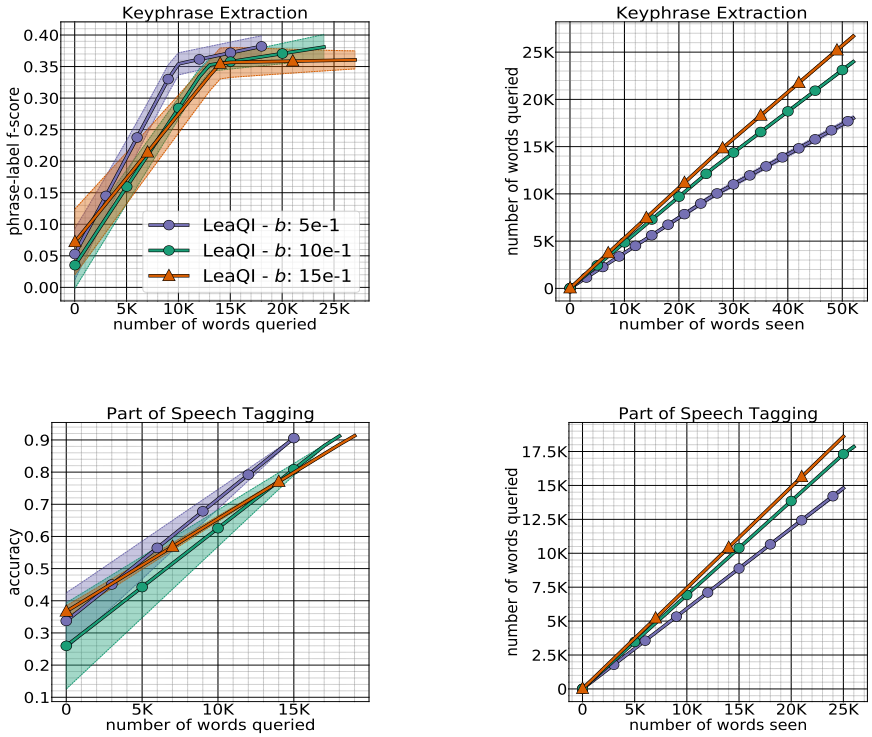


Figure 7.5: (top-row) English keyphrase extraction and (bottom-row) low-resource language part of speech tagging on Greek, Modern (el). We show the performance of using difference confidence parameters b. These plots indicate that our model is robust to difference confidence parameters.

7.5 Discussion and Limitations

In this chapter, we considered the problem of reducing the number of queries to an expert labeler for structured prediction problems. We took an imitation learning approach and developed an algorithm, LEAQI, which leverages a source that has low-quality labels: a heuristic policy that is suboptimal but free. To use this heuristic as a policy, we learn a difference classifier that effectively tells LEAQI when it is safe to treat the heuristic's action as if it were optimal. We showed empirically—across Named Entity Recognition, Keyphrase Extraction and Part of Speech Tagging tasks—that the active learning approach

improves significantly on passive learning, and that leveraging a difference classifier improves on that.

We highlight some limitation for our approach below:

1. In some settings, learning a difference classifier may be as hard or harder than learning the structured predictor; for instance if the task is binary sequence labeling (e.g., word segmentation), minimizing its usefulness.

2. The true labeling cost is likely more complicated than simply the number of individual actions queried to the expert.

Despite these limitations, we hope that LEAQI provides a useful (and relatively simple) bridge that can enable using rule-based systems, heuristics, and unsupervised models as building blocks for more complex supervised learning systems. This is particularly attractive in settings where we have very strong rule-based systems, ones which often outperform the best statistical systems, like coreference resolution (Lee et al., 2011), information extraction (Riloff and Wiebe, 2003), and morphological segmentation and analysis (Smit et al., 2014).

In this chapter we studied the usual "supervised" structured prediction setting, where at training time, we have access to ground truth outputs (e.g., dependency trees) on which to build a predictor. In chapter 8 we consider the substantially harder case of online *bandit* structured prediction, in which the system never sees supervised training data, but instead must make predictions and then only receives feedback about that *single* prediction.

9 LEAQI$(\Pi, \mathcal{H}, N, \pi^\star, \pi^{\text{h}}, b)$

initialize dataset $D = \{\}$
initialize policy π_1 to any policy in Π
initialize difference dataset $S = \{\}$
initialize difference classifier $h_1(s) = 1$ $(\forall s)$
for $i = 1 \dots N$ **do**
    Receive input sentence $\boldsymbol{x}$
    ▷ *generate a T-step trajectory using π_i*
    Generate output $\hat{\boldsymbol{y}}$ using π_i
    for each s in $\hat{\boldsymbol{y}}$ **do**
        ▷ *draw bernouilli random variable*
        $Z_i \sim \text{Bern}\left(\frac{b}{b+\text{certainty}(\pi_i, s)}\right)$; see §7.3.3
        if $Z_i = 1$ **then**
            ▷ *set difference classifier prediction*
            $\hat{d}_i = h_i(s)$
            if AppleTaste(s, $\pi^{\text{h}}(s)$, $\hat{d}_i$) **then**
                ▷ *predict agree query heuristic*
                $D \leftarrow D \cup \left\{ \left(s, \pi^{\text{h}}(s)\right) \right\}$
            else
                ▷ *predict disagree query expert*
                $D \leftarrow D \cup \left\{ \left(s, \pi^\star(s)\right) \right\}$
                $d_i = \mathbb{1}\left[\pi^\star(s) = \pi^{\text{h}}(s)\right]$
                $S \leftarrow S \cup \left\{ \left(s, \pi^{\text{h}}(s), \hat{d}_i, d_i\right) \right\}$
            end if
        end if
    end for
    Train policy $\pi_{i+1} \in \Pi$ on D
    Train difference classifier $h_{i+1} \in \mathcal{H}$ on S to minimize Type II errors (see §7.3.2)
end for
return best (or random) π_i

10 AppleTaste_STAP($S, a_i^{\mathrm{h}}, \hat{d}_i$)

▷ *count examples that are action* a_i^h
let $t = \sum_{(\_,a,\_,\_)\in S} \mathbb{1}[a_i^{\mathrm{h}} = a]$
▷ *count mistakes made on action* a_i^h
let $m = \sum_{(\_,a,\hat{d},d)\in S} \mathbb{1}[\hat{d} \neq d \wedge a_i^{\mathrm{h}} = a]$
$w = \frac{t}{|S|}$ ▷ *percentage of time* a_i^h *was seen*
if $w < 1$ **then**
    ▷ *skew distribution*
    draw $r \sim$ ***Beta***$(1 - w, 1)$
else
    draw $r \sim$ ***Uniform***$(0, 1)$
end if
return $(d = 1) \wedge (r \leq \sqrt{(m + 1)/t})$

Chapter 8: Structured Prediction Under Bandit Feedback

8.1 Introduction

In structured prediction the goal is to jointly predict the values of a collection of variables that interact. In the usual "supervised" setting (see chapter 7), at training time, you have access to ground truth outputs (e.g., dependency trees) on which to build a predictor. In this chapter we consider the substantially harder case of online *bandit* structured prediction, in which the system never sees supervised training data, but instead must make predictions and then only receives feedback about the quality of that *single* prediction. The model we simulate is (Figure 8.1) :

1. the world reveals an input (e.g., a sentence)
2. the algorithm produces a *single* structured prediction (e.g., full parse tree);
3. the world provides a loss (e.g., overall quality rating) and possibly a small amount of additional feedback;
4. the algorithm updates itself

The goal of the system is to minimize it's cumulative loss over time, using the feedback to improve itself. This introduces a fundamental exploration-versus-exploitation trade-off, in which the system must try new things in hopes that it will learn something useful, but also in which it is penalized (by high cumulative loss) for exploring too much.[1]

One natural question we explore in this chapter is: in addition to the loss, what forms of feedback are both easy for a user to provide and useful for a system to utilize? At

[1] Unlike active learning—in which the system chooses which examples it wants feedback on—in our setting the system is beholden to the human's choice in inputs.

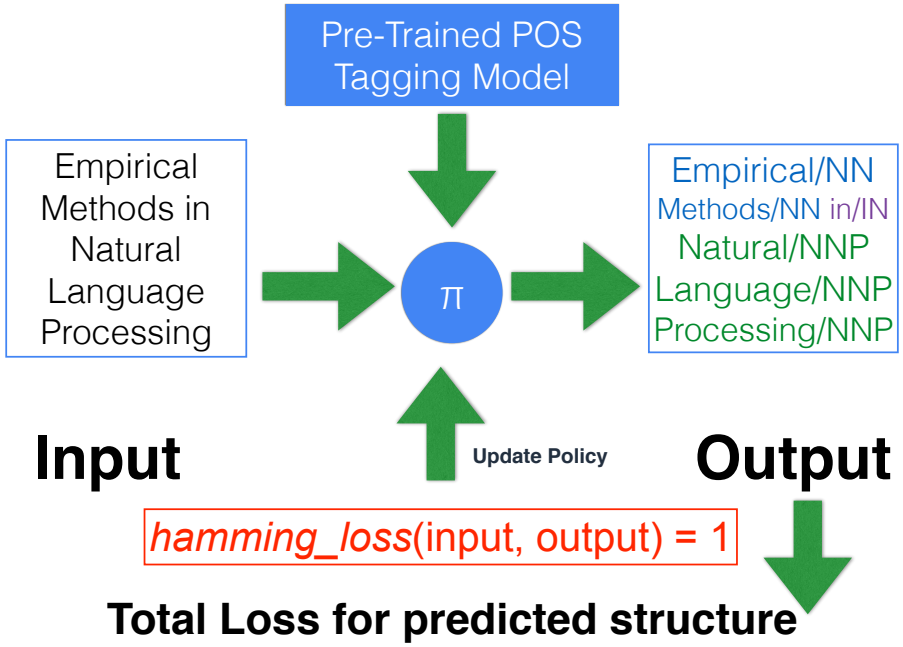


Figure 8.1: BLS for learning POS tagging. We learn a policy π, whose output a user sees. The user views predicted tags and provides a loss (and possibly additional feedback, such as which words are labeled incorrectly). This is used to update π.

one extreme, one can solicit no additional feedback, which makes the learning problem very difficult. We describe *Bandit Learning to Search*, BLS, an approach for improving joint predictors from different types of bandit feedback. The algorithm predicts outputs and observes the loss of the predicted structure; but then it uses a regression strategy to estimate *counterfactual* costs of (some) other structures that it did *not* predict. This variance reduction technique (§8.2.2) is akin to doubly-robust estimation in contextual bandits. The trade-off is that in order to accurately train these regressors, BLS requires per-action feedback from the user (e.g., which words were labeled incorrectly). It appears that this feedback is necessary; with out it, accuracy *degrades* over time. Additionally, we consider several forms of exploration beyond a simple ϵ-greedy strategy, including Boltzmann exploration and Thompson sampling (§8.2.4). We demonstrate the efficacy of these developments on POS tagging, syntactic chunking and dependency parsing

(§8.3), in which we show improvements over both LOLS Chang et al. (2015) and Policy Gradient (Sutton et al., 1999).

8.2 Learning with Bandit Feedback

We operate in the learning to search framework, a family of algorithms for solving structured prediction problems, which essentially train a *policy* to make sequence of decisions that are stitched together into a final structured output. Such algorithms decompose a joint prediction task into a sequence of action predictions, such as predicting the label of the next word in sequence labeling or predicting a shift/reduce action in dependency parsing[2]; these predictions are tied by features and/or internal state. Algorithms in this family have recently met with success in neural networks (Bengio et al., 2015b; Wiseman and Rush, 2016), though date back to models typically based on linear policies (Collins and Roark, 2004b; Daumé and Marcu, 2005; Xu et al., 2007; Daumé et al., 2009; Ross et al., 2011b; Ross and Bagnell, 2014; Doppa et al., 2014; Chang et al., 2015).

Most learning to search algorithms operate by considering a search space like that shown in Figure 8.2. The learning algorithm first *rolls-in* for a few steps using a *roll-in* policy π^{in} to some state R, then considers all possible actions available at state R, and then *rolls out* using a *roll-out* policy π^{out} until the end of the sequence. In the fully supervised case, the learning algorithm can then compute a loss for all possible outputs, and use this loss to drive learning at state R, by encouraging the learner to take the action with lowest cost, updating the learned policy from $\hat{\pi}_i$ to $\hat{\pi}_{i+1}$.

In the bandit setting, this is not possible: we can only evaluate one loss; nevertheless, we can follow a similar algorithmic structure. Our specific algorithm, BLS, is summarized in 11. We start with a pre-trained reference policy π^{ref} and seek to improve it with bandit feedback. On each example, an exploration algorithm (§8.2.4) chooses whether to explore

[2]Although the decomposition is into a sequence of predictions, such approaches are not limited to "left-to-right" style prediction tasks (Ross et al., 2011b; Stoyanov et al., 2011).

11 BLS (BLS)

Require: examples $\{x_i\}_{i=1}^N$, reference policy π^{ref}, exploration algorithm $explorer$, and rollout-parameter $\beta \geq 0$

$\pi_0 \leftarrow$ initial policy;
$\mathcal{I} \leftarrow \emptyset$;
$\rho \leftarrow$ initial cost estimator;
for each x_i in training examples **do**
  if explorer chooses not to explore **then**
    $\pi \leftarrow$ *Unif*$(\mathcal{I})$ // pick policy;
    $y_i \leftarrow$ predict using π;
    $c \leftarrow$ bandit loss of y_i;
  else
    $t \leftarrow$ *Unif*$(0, T-1)$ // deviation time;
    $\tau \leftarrow$ roll-in with $\hat{\pi}_i$ for t rounds;
    $s_t \leftarrow$ final state in τ;
    $a_t =$ *explorer*(s_t) // deviation action;
    $\pi^{\text{out}} \leftarrow \pi^{\text{ref}}$ with prob β, else $\hat{\pi}_i$;
    $y_i \leftarrow$ roll-out with π^{out} from $\tau + a_t$;
    $c \leftarrow$ bandit loss of y_i;
    $\hat{c} \leftarrow$ *est_cost*$(s_t, \tau, \rho, A(s_t), a_t, c)$;
    $\hat{\pi}_{i+1} \leftarrow$ *update*$(\hat{\pi}_i, (\Phi(x_i, s_t), \hat{c}))$;
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{\hat{\pi}_{i+1}\}$;
    update cost estimator ρ;
  end if
end for

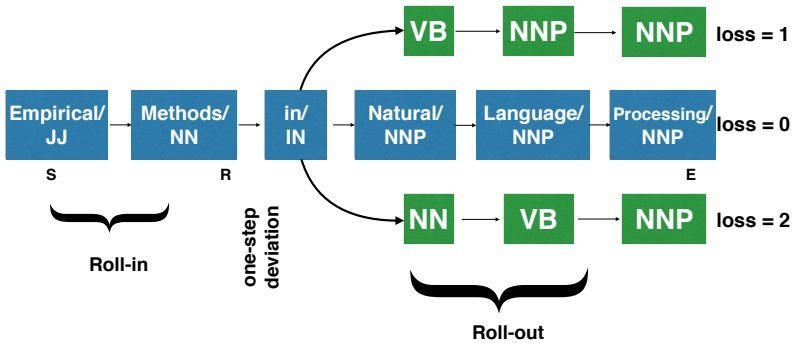


Figure 8.2: Search space for part of speech tagging, explored by a policy that chooses to "explore" at state R.

or exploit. If it chooses to exploit, a random learned policy is used to make a prediction and no updates are made. If, instead, it chooses to explore, it executes a roll-in as usual, a *single* deviation at time t according to the exploration policy, and then a roll-out. Upon completion it suffers a bandit loss for the *entire* complete trajectory. It then uses a *cost estimator* ρ to guess the costs of the un-taken actions. From this it forms a complete cost vector, and updates the underlying policy based on this cost vector. Finally, it updates the cost estimator ρ.

8.2.1 Cost Estimation by Importance Sampling

The simplest possible method of cost estimation is importance sampling Horvitz and Thompson (1952); Chang et al. (2015). If the third action is the one explored with probability p_3 and a cost $\hat{c}_3$ is observed, then the cost vector for all actions is set to $\langle 0, 0, \hat{c}_3/p_3, 0, \ldots, 0 \rangle$. This yields an unbiased estimate of the true cost vector because in expectation over all possible actions, the cost vector equals $\langle \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_K \rangle$.

Unfortunately, this type of cost estimate suffers from huge variance (see experiments in § 8.3). If actions are explored uniformly at random, then all cost vectors look like $\langle 0, 0, K\hat{c}_3, 0, \ldots 0 \rangle$, which varies quite far from its expectation when K is large. To better understand the variance reduction issue, consider the part of speech tagging example from Figure 8.2. As in the figure, suppose that the deviation occurs at time step 3 and that during

roll-in, the first two words are tagged correctly by the roll-in policy. At $t = 3$, there are 45 possible actions (each possible part of speech) to take from the deviation state, of which three are shown; each action (under uniform exploration) will be taken with probability $1/45$. If the first is taken, a loss of one will be observed, if the second, a loss of zero, and if the third, a loss of two. When a fair coin is flipped, perhaps the third choice is selected, which will induce a cost vector of $\boldsymbol{c} = \langle 0, 0, 90, 0, \dots \rangle$. In expectation over this random choice, we have $\mathbb{E}_a[c]$ is correct, implying unbiasedness, but the variance is very large: $\mathcal{O}((Kc_{\text{max}})^2)$.

This problem is exacerbated by the fact that many learning to search algorithms define the cost of an action a to be the *difference* between the cost of a and the minimum cost. This is desirable because when the policy is predicting greedily, it should choose the action that *adds* the least cost; it should not need to account for already-incurred cost. For example, suppose the first two words in Figure 8.2 were tagged incorrectly. This would add a loss of 2 to any of the estimated costs, but could be very difficult to fit because this loss was based on past actions, not the current action.

8.2.2 Doubly Robust Cost Estimation

To address the challenge of high variance cost estimates, we adopt a strategy similar to the doubly-robust estimation used in the (non-structured) contextual bandit setting (Dudik et al., 2011). In particular, we train a separate set of regressors to estimate the total costs of any action, which we use to impute a counterfactual cost for untaken actions.

Algorithm 12 spells this out, taking advantage of an action-to-cost regressor, ρ. To estimate the cost of an un-taken action a' at a deviation, we simulate the execution of a', followed by the execution of the current policy through the end. The cost of that entire trajectory is estimated by summing ρ over all states along the path. For example, in the part of speech tagging example above, we learn 45 regressors: one for each part of speech.

12 *est_cost*

Require: current state: s_t; roll-in trajectory: τ; K regression functions (one for every action): ρ; set of allowed actions: $A(s_t)$; roll-out policy: π^{out}; explored action: a_t; bandit loss: c

$t \leftarrow |\tau|$;

Initialize $\hat{c}$: a vector of size $|A(s_t)|$;

$\hat{c}_0 \leftarrow 0$;

for $(a, s) \in \tau$ **do**

    $\hat{c}_0 \leftarrow \hat{c}_0 + \rho_a(\Phi(s))$;

end for

for $a \in A(s_t)$ **do**

    if $a = a_t$ **then**

        $\hat{c}(a) \leftarrow c$;

    else

        $\hat{c}(a) \leftarrow \hat{c}_0$;

        $y \leftarrow$ roll-out with π^{out} from $\tau + a$;

        for (a', s') in y **do**

            $\hat{c}(a) \leftarrow \hat{c}(a) + \rho_{a'}(\Phi(s'))$;

        end for

    end if

end for

return $\hat{c}$: a vector of size $|A(s_t)|$, where $\hat{c}(a)$ is the estimated cost for action a at state s_t.

The cost of a roll-out is estimated as the sum of these regressors over the entire predicted sequence.

Using this doubly-robust estimate strategy addresses *both* of the problems mentioned in §8.2.1. First, this is able to provide a cost estimate for *all* actions. Second, because ρ is deterministic, it will give the same cost to the common prefix of all trajectories, thus resolving credit assignment.

The remaining challenge is: how to estimate these regressors. Unfortunately, this currently comes at an additional cost to the user: we must observe per-action feedback. In particular, when the user views a predicted output (e.g., part of speech sequence), we ask for a binary signal for each action whether the predicted action was right or wrong. Thus, for a sentence of length T, we generate T training examples for every time step $1 \leq t \leq T$. Each training example has the form: (a_t, c_t), where a_t is the predicted action at time t, and c_t is a binary cost, either 1 if the predicted action was correct, or zero otherwise. This amounts to a user "crossing out" errors, which hopefully incurs low overhead. Using these T training examples, we can effectively train the K regressors for estimating the cost of unobserved actions.

8.2.3 Theoretical Analysis

In order to analyze the variance of the BLS estimator (in particular to demonstrate that it has lower variance than importance sampling), we provide a reduction to contextual bandits in an i.i.d setting. Dudík et al. (2014) studied the contextual bandit setting, which is similar to out setting but without the complexity of *sequences* of actions. (In particular, if $T = 1$ then our setting is exactly the contextual bandit setting.) They studied the task of off-policy evaluation and optimization for a target policy ν using doubly robust estimation given historic data from an exploration policy μ consisting of contexts, actions, and received rewards. They prove that this approach yields accurate value estimates when

there is either a good (but not necessarily consistent) model of rewards or a good (but not necessarily consistent) model of past policy. In particular, they show:

Theorem 10. *Let $\Delta(x,a)$ and $\rho_k(x,a)$ denote, respectively, the additive error of the reward estimator $\hat{r}$ and the multiplicative error of the action probability estimator $\hat{\mu}_k$. If exploration policy μ and the estimator $\hat{\mu}_k$ are stationary, and the target policy ν is deterministic, then the variance of the doubly robust estimator $\mathbb{V}_\mu[\hat{V}_{DR}]$ is:*

$$\begin{aligned}
&\frac{1}{n}(\mathbb{V}_{(x,a)\sim\nu}[r^*(x,a)+(1-\rho_1(x,a))\Delta(x,a)]) \\
&\quad +\mathbb{E}_{(x,a)\sim\nu}\big[\frac{1}{\hat{\mu}_1(a|x)}\rho_1(x,a)\mathbb{V}_{r\sim D(\cdot|x,a)}[r]\big] \\
&\quad +\mathbb{E}_{(x,a)\sim\nu}\big[\frac{1-\mu_1(a|x)}{\hat{\mu}_1(a|x)}\rho_1(x,a)\Delta(x,a)^2\big]]
\end{aligned}$$

The theorem show that the variance can be decomposed into three terms. The first term accounts for the randomness in the context features. The second term accounts for randomness in rewards and disappears when rewards are deterministic functions of the context and actions. The last term accounts for the disagreement between actions taken by the exploration policy μ and the target policy ν. This decomposition shows that doubly robust estimation yields accurate value estimates when there is either a good model of rewards or a good model of the exploration policy.

We can build on this result for BLS to show an *identical* result under the following reduction. The exploration policy μ in our setting is defined as follows: for every exploration round, a randomly selected time-step is assigned a randomly chosen action, and a deterministic reference policy is used to generate the roll-in and roll-out trajectories. Our goal is to evaluate and optimize a better target policy ν. Under this setting, and assuming that the structures are generated i.i.d from a fixed but unknown distribution, the structured prediction problem will be equivalent to a contextual bandit problem were we consider the roll-in trajectory as part of the context.

8.2.4 Options for Exploration Strategies

In addition to the ϵ-greedy exploration algorithm, we consider the following exploration strategies:

Boltzmann (Softmax) Exploration. Boltzmann exploration varies the action probabilities as a graded function of estimated value. The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their cost estimates; action a is chosen with probability proportional to $\exp\left[\frac{1}{\text{temp}}c(a)\right]$, where "temp" is a positive parameter called the temperature, and $c(a)$ is the current predicted cost of taking action a. High temperatures cause the actions to be all (nearly) equiprobable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates.

Thompson Sampling estimates the following elements: a set Θ of parameters μ; a prior distribution $P(\mu)$ on these parameters; past observations D consisting of observed contexts and rewards; a likelihood function $P(r|b,\mu)$, which gives the probability of reward given a context b and a parameter μ; In each round, Thompson Sampling selects an action according to its posterior probability of having the best parameter μ. This is achieved by taking a sample of parameter for each action, using the posterior distributions, and selecting that action that produces the best sample (Agrawal and Goyal, 2013; Komiyama et al., 2015). We use Gaussian likelihood function and Gaussian prior for the Thompson Sampling algorithm. In addition, we make a linear payoff assumption similar to Agrawal and Goyal (2013), where we assume that there is an unknown underlying parameter $\mu_a \in R^d$ such that the expected cost for each action a, given the state s_t and context x_i is $\Phi(x_i, s_t)^T \mu_a$.

8.3 Experimental Results

The evaluation framework we consider is the fully online setup described in the introduction, measuring the degree to which various algorithms can effectively improve upon a reference policy by observing only a partial feedback signal, and effectively balancing exploration and exploitation. We learn from one structured example at every time step, and we do a single pass over the available examples. We measure loss as the average cumulative loss over time, thus algorithms are appropriately "penalized" for unnecessary exploration.

8.3.1 Tasks, Policy Classes and Data Sets

We experiment with the following three tasks. For each, we briefly define the problem, describe the policy class that we use for solving that problem in a learning to search framework (we adopt a similar setting to that of Chang et al. (2016), who describes the policies in more detail), and describe the data sets that we use. The regression problems are solved using squared error regression, and the classification problems (policy learning) is solved via cost-sensitive one-against-all.

Part-Of-Speech Tagging over the 45 Penn Treebank (Marcus et al., 1993) tags. To simulate a domain adaptation setting, we train a reference policy on the TweetNLP dataset (Owoputi et al., 2013), which achieves good accuracy in domain, but does poorly out of domain. We simulate bandit feedback over the entire Penn Treebank Wall Street Journal (sections 02–21 and 23), comprising $42k$ sentences and about one million words. (Adapting *from* tweets *to* WSJ is nonstandard; we do it here because we need a large dataset on which to simulate bandit feedback.) The measure of performance is average per-word accuracy (one minus Hamming loss).

Noun Phrase Chunking is a sequence segmentation task, in which sentences are divided into base noun phrases. We solve this problem using a sequence span identification

predictor based on Begin-In-Out encoding, following Ratinov and Roth (2009), though applied to chunking rather than named-entity recognition. We used the CoNLL-2000 dataset for training and testing. We used the smaller test split ($2,012$ sentences) for training a reference policy, and used the training split ($8,500$ sentences) for online evaluation. Performance was measured by F-score over predicted noun phrases (for which one has to predict the entire noun phrase correctly to get any points).

Dependency Parsing is a syntactic analysis task, in which each word in a sentence gets assigned a grammatical head (or "parent"). The experimental setup is similar to part-of-speech tagging. We train an arc-eager dependency parser (Nivre, 2003), which chooses among (at most) four actions at each state: Shift, Reduce, Left or Right. As in part of speech tagging, the reference policy is trained on the TweetNLP dataset (using an oracle due to (Goldberg and Nivre, 2013)), and evaluated on the Penn Treebank corpus (again, sections $02-21$ and section 23). The loss is unlabeled attachment score (UAS), which measures the fraction of words that pick the correct parent.

8.3.2 Main Results

Here, we describe experimental results (Table 8.1) comparing several algorithms: (line B) The bandit variant of the LOLS algorithm, which uses importance sampling and ϵ-greedy exploration; (lines C-F) BLS, with bandit feedback and per-word error correction, with variance reduction and four exploration strategies: ϵ-greedy, Boltzmann, Thompson, and "oracle" exploration in which case the oracle action is always chosen during exploration; (line G) The Policy Gradient reinforcement learning algorithm, with ϵ-greedy exploration on one-step deviations; and (line H) A fully supervised "upper bound" trained with DAgger.

From these results, we draw the following conclusions (the rest of this section elaborates on these conclusions in more detail):

	Algorithm	Variant	POS Acc	DepPar UAS	Chunk F-Scr
A.	Reference	-	47.24	44.15	74.73
B.	LOLS	ϵ-greedy	2.29	18.55	31.76
C.	BLS	ϵ-greedy	86.55	56.04	90.03
D.		Boltz.	**89.62**	**57.20**	**90.91**
E.		Thomp.	89.37	56.60	90.06
F.		Oracle	89.23	56.60	90.58
G.	Policy∇	ϵ-greedy	75.10	-	90.07
H.	DAgger	Full Sup	96.51	90.64	95.29

Table 8.1: Total progressive accuracies for the different algorithms on the three natural language processing tasks. LOLS uniformly *decreases* performance over the Reference baseline. BLS, which integrates cost regressors, uniformly improves, often quite dramatically. The overall effect of the exploration mechanism is small, but in all cases Boltzmann exploration is statistically significantly better than the other options at the $p < 0.05$ level (because the sample size is so large). Policy Gradient for dependency parsing is missing because after processing $\frac{1}{4}$ of the data, it was substantially subpar.

1. The original LOLS algorithm is ineffective at improving the accuracy of a poor reference policy (A vs B);
2. Collecting additional per-word feedback in BLS allows the algorithm to drastically improve on the reference (A vs C) and on LOLS (B vs C); we show in §8.3.3 that this happens because of variance reduction;
3. Additional leverage can be gained by varying the exploration strategy, and in general Boltzmann exploration is effective (C,D,E), but the Oracle exploration strategy is *not* optimal (F vs D); see §8.3.4;
4. For large action spaces like POS tagging, the BLS-type updates outperform Policy Gradient-type updates, when the exploration strategy is held constant (G vs D), see §8.3.5.
5. Bandit feedback is less effective than full feedback (H vs D) (§8.3.6).

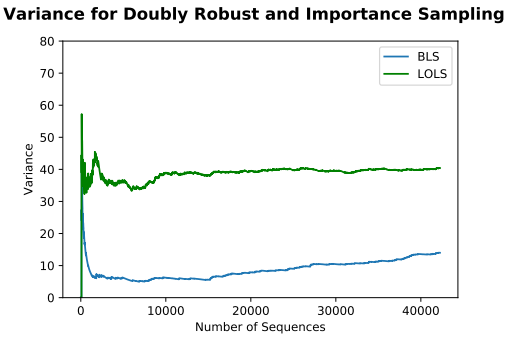


Figure 8.3: Analyzing the variance of the cost estimates from LOLS and BLS over a run of the algorithm for POS; the x-axis is number of sentences processed, y-axis is empirical variance.

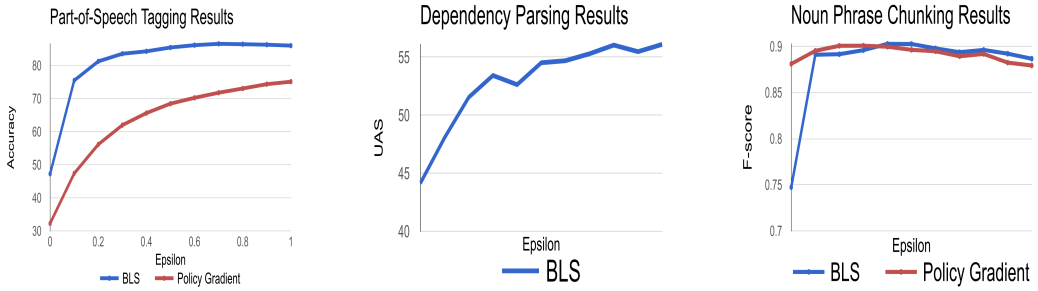


Figure 8.4: Analyzing the effect of ϵ in exploration/exploitation trade-off. Overall, large values of ϵ are strongly preferred.

8.3.3 Effect of Variance Reduction

Table 8.1 shows the progressive validation accuracies for all three tasks for a variety of algorithmic settings. To understand the effect of variance, it is enough to compare the performance of the Reference policy (the policy learned from the out of domain data) with that of LOLS. In all of these cases, running LOLS substantially decreases performance. Accuracy drops by 45% for POS tagging, 26% for dependency parsing and 43% for noun phrase chunking. For POS tagging, the LOLS accuracy falls *below* the accuracy one would get for random guessing (which is approximately 14% on this dataset for `NN`)!

When the underlying algorithm changes from LOLS to BLS, the overall accuracies go up significantly. Part of speech tagging accuracy increases from 47% to 86%; dependency parsing accuracy from 44% to 57%; and chunking F-score from 74% to 90%. These

numbers naturally fall below state of the art for *fully supervised* learning on these data sets, precisely because these results are based only on bandit feedback (see §8.3.6).

8.3.4 Effect of Exploration Strategy

Figure 8.4 shows the effect of the choice of ϵ for ϵ-greedy exploration in BLS. Overall, best results are achieved with *remarkably* high epsilon, which is possibly counter-intuitive. The reason this happens is because BLS only explores on one out of T time steps, of which there are approximately 30 in each of these experiments (the sentence lengths). This means that even with $\epsilon = 1$, we only take a random action roughly 3.3% of the time. It is therefore not surprising that large ϵ is the most effective strategy. Overall, although the differences are small, the best choice of ϵ across these different tasks is ≈ 0.6.

Returning to Table 8.1, we can consider the effect of different exploration mechanisms: ϵ-greedy, Boltzmann (or softmax) exploration, and Thompson sampling. Overall, Boltzmann exploration was the most effective strategy, gaining about 3% accuracy in POS tagging, just over 1% in dependency parsing, and just shy of 1% in noun phrase chunking. Although the latter two effects are small, they are statistically significant, which is measurable due to the fact that the evaluation sets are very large. In general, Thompson sampling is also effective, though worse than Boltzmann.

Finally, we consider a variant in which whenever BLS requests exploration, the algorithm "cheats" and chooses the gold standard decision at that point. This is the "oracle exploration" line in Table 8.1. We see that this does *not* improve overall quality, which suggests that a good exploration strategy is not one that always does the right thing, but one that also explored bad—but useful-to-learn-from—options.

8.3.5 Policy Gradient Updates

A natural question is: how does bandit structured prediction compare to more standard approaches to reinforcement learning (we revisit the question of how these

problems differ in § 8.4). We chose Policy Gradient Sutton et al. (1999) as a point of comparison. The main question we seek to address is how the BLS update rule compares to the Policy Gradient update rule. In order to perform this comparison, we hold the exploration strategy *fixed*, and implement the Policy Gradient update rule inside our system.

More formally, the policy gradient optimization is similar to that used in BLS. PG maintains a policy π_θ, which is parameterized by a set of parameters θ. Features are extracted from each state s_t to construct the feature vectors $\phi(s_t)$, and linear function approximation models the probability of selecting action a_t at state s_t under π_θ: $\pi_\theta(a_t|s_t) \propto \exp(\theta_{a_t}^T \phi(s_t))$, where K is the total number of actions. PG maximizes the total expected return under the distribution of trajectories sampled from the policy π_θ.

To balance the exploration / exploitation tradeoff, we use exactly the same epsilon greedy technique used in BLS (Algorithm 11). For each trajectory τ sampled from π_θ, a state is selected uniformly at random, and an action is selected greedily with probability ϵ. The policy π_θ is used to construct the roll-in and roll-out trajectories. For every trajectory τ, we collect the same binary grades from the user as in BLS, and use them to train a regression function to estimate the per-step reward. These estimates are then be summed up to compute the total return G_t from time step t onwards (Algorithm 12).

We use standard policy gradient update for optimizing the policy θ based on the observed rewards:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log(\pi_\theta(s_t, a_t)) G_t \tag{8.1}$$

The results of this experiment are shown in line G of Table 8.1. Here, we see that on POS tagging, where the number of actions is very large, PG significantly underperforms BLS. Our initial experiments in dependency parsing showed the PG significantly underperformed BLS after processing $\frac{1}{4}$ of the data. The difference is substantially smaller in chunking, where PG is on part with BLS with ϵ-greedy exploration. Figure 8.4 shows the effect of ϵ

on PG, where we see that it also prefers large values of ϵ, but its performance saturates as $\epsilon \to 1$.

8.3.6 Bandit Feedback vs Full Feedback

Finally, we consider the trade-off between bandit feedback in BLS and full feedback. To make this comparison, we run the fully supervised algorithm DAgger Ross et al. (2011b) which is effectively the same algorithm as LOLS and BLS under full supervision. In Table 8.1, we can see that full supervision dramatically improves performance from around 90% to 97% in POS tagging, 57% to 91% in dependency parsing, and 91% to 95% in chunking. Of course, achieving this improved performance comes at a high labeling cost: a human has to provide exact labels for each decision, not just binary "yes/no" labels.

8.4 Discussion & Conclusion

The most similar algorithm to ours is the bandit version of LOLS Chang et al. (2015) (which is analyzed theoretically but not empirically); the key differences between BLS and LOLS are: (a) BLS employs a doubly-robust estimator for "guessing" the costs of counterfactual actions; (b) BLS employs alternative exploration strategies; (c) BLS is effective in practice at improving the performance of an initial policy.

In the NLP community, Sokolov et al. (2016a) and Sokolov et al. (2016b) have proposed a policy gradient-like method for optimizing log-linear models like conditional random fields Lafferty et al. (2001) under bandit feedback. Their evaluation is most impressive on the problem of domain adaptation of a machine translation system, in which they show that their approach is able to learn solely from bandit-style feedback, though requiring a large number of samples.

In the learning-to-search setting, the difference between *structured prediction under bandit feedback* and *reinforcement learning* gets blurry. A distinction in the problem

definition is that the world is typically assumed to be fixed and stochastic in RL, while the world is both deterministic and *known* (conditioned on the input, which is random) in bandit structured prediction: given a state and action, the algorithm always knows what the next state will be. A difference in solution is that there has been relatively little work in reinforcement learning that explicitly begins with a reference policy to improve and often assumes an *ab initio* training regime. In practice, in large state spaces, this makes the problem almost impossible, and practical settings like AlphaGo (Silver et al., 2016) require imitation learning to initialize a good policy, after which reinforcement learning is used to improve that policy.

Learning from partial feedback has generated a vast amount of work in the literature, dating back to the seminal introduction of multi-armed bandits by (Robbins, 1985). However, the vast number of papers on this topic does not consider joint prediction tasks; see Auer et al. (2002); Auer (2002); Langford and Zhang (2008); Srinivas et al. (2009); Li et al. (2010); Beygelzimer et al. (2010b); Dudik et al. (2011); Chapelle and Li (2011); Valko et al. (2013) and references *inter alia*. There, the system observes (bandit) feedback for every decision.

Other forms of contextual bandits on structured problems have been considered recently. Kalai and Vempala (2005) studied the structured problem of online shortest paths, where one has a directed graph and a fixed pair of nodes (s, t). Each period, one has to pick a path from s to t, and then the times on all the edges are revealed. The goal of the learner is to improve it's path predictions over time. Relatedly, Krishnamurthy et al. (2015) studied a variant of the contextual bandit problem, where on each round, the learner plays a sequence of actions, receives a score for each individual action, and obtains a final reward that is a linear combination to those scores.

In this chapter, we presented a computationally efficient algorithm for structured contextual bandits, BLS, by combining: locally optimal learning to search (to control the structure of exploration) and doubly robust cost estimation (to control the variance

of the cost estimation). This provides the first practically applicable learning to search algorithm for learning from bandit feedback. Unfortunately, this comes at a cost to the user: they must make more fine-grained judgments of correctness than in a full bandit setting. In particular, they must mark each decision as correct or incorrect: it is an open question whether this feedback can be removed without incurring a substantially larger sample complexity. A second large open question is whether the time step at which to deviate can be chosen more intelligently, similar to selective sampling (Shi et al., 2015), using active learning.

This concludes our discussion for Part II of this dissertation. In this part, we presented minimally supervised learning algorithms based on reinforcement and imitation learning. In chapter 9 we conclude the discussion with a summary and pointers for future work.

Chapter 9: Conclusion

9.1 Summary

We now conclude by discussing the contributions of this thesis with regards to our original thesis statement:

> Meta-Learning and reinforcement learning algorithms provide a useful class of algorithms for learning fair, adaptive, and robust models with minimal supervision.

In this dissertation we presented algorithms for learning with minimal supervision based on meta-learning and reinforcement learning. In Part I we focused on learning algorithms based on meta-learning. We studied the following set of problems:

1. Few-shot adaptation of Neural Machine Translation (NMT) systems (chapter 3).
2. Learning to actively learn under fairness parity constraints (chapter 4).
3. Learning better exploration strategies in contextual bandits (chapter 5).

In all these settings, the key idea is to construct a meta-training dataset by sampling from a distribution of learning tasks. At training time, the agent uses this learning tasks to simulate what would happen when presented with a new minimally supervised problem.

In Part II we studied minimally supervised learning algorithms that could be used whenever the agent has access to reward signals, even in the case where we can't run full simulations on supervised learning tasks. We studied the following set of problems:

1. Reinforcement learning with no incremental feedback (chapter 6).

2. Active imitation learning with noisy guidance (chapter 7).

3. Structured prediction problems under bandit feedback (chapter 8).

9.2 Open Problems and Future Directions

In this section, we discuss some potential future directions of research in the area of learning with minimal supervision.

9.2.1 Selection of Meta-Training Tasks

Typical meta-learning algorithms assume access to a distribution of learning tasks. At training time, the meta-learner samples learner tasks from this distribution to simulate learning at inference time. In many cases, it is not clear how to construct this distribution over the learning tasks. The choice of the best distribution of learning task to learn from at meta-training time remains an open research question.

9.2.2 Addressing the Vanishing Gradient Problem in Gradient Based Meta-Learning

Optimization based meta-learning (1) aims to optimize the model parameters such that one or small number of gradient steps on a new task will produce maximally effective behavior on that task. The meta-optimization is performed over the meta-learner model parameters, whereas the objective is computed using the updated model parameters after fine-tuning. In effect, optimization based meta-learning aims to optimize the model parameters such that one or small number of gradient steps on a new task will produce maximally effective behavior on that task.However, this requires running a back propagation procedure through the full computational graph for the updated model parameters. This leads to a vanishing gradient problem, where the gradients with respect to the original meta-learning parameters vanish as the computational graph becomes larger. Exploring

algorithms for mitigating this issue, and scaling up existing meta-learning algorithms for larger computational graphs remains an open research problem.

9.2.3 Selecting Interventions for Mitigating Disparities

In chapter 4 we presented a meta-learning algorithm for promoting fairness in learned models by learning to active learn under fairness parity constraints. However, this is not the only possible method for mitigating disparities in machine learning models. Other methods include: constraining the machine learning model, collecting more data, extracting more features, or changing the model architecture. The selection of the best method of mitigation remains an open research question.

9.2.4 Solving Non-Increment Reinforcement Learning Problems in a Continuous Action Space

In chapter 6 we presented a reinforcement learning algorithm for learning with non-incremental reward signals observed at the end of a learning episode. Our proposed algorithm requires access to discrete action spaces as it maintains estimates for each action in the space to tackle the credit assignment problem. Extending this approach to continuous action spaces remain an open research problem.

Bibliography

Behnoush Abdollahi and Olfa Nasraoui. Transparency in fair machine learning: The case of explainable recommender systems. In *Human and Machine Learning*, pages 21–35. Springer, 2018.

Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E. Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *In Proceedings of the 31st International Conference on Machine Learning (ICML-14*, pages 1638–1646, 2014.

Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. *arXiv preprint arXiv:1803.02453*, 2018.

Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 127–135, 2013.

Hadis Anahideh, Abolfazl Asudeh, and Saravanan Thirumuruganathan. Fair active learning. *arXiv preprint arXiv:2001.01796*, 2020.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

R. Angell, B. Johnson, Y. Brun, and A. Meliou. Themis: Automatically testing software for discrimination. In *Joint Meeting on European Software Engineering*, 2018.

Dana Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1988. ISSN 0885-6125.

Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. propublica. *See https://www. propublica. org/article/machine-bias-risk-assessments-in-criminal-sentencing*, 2016.

Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. *arXiv preprint arXiv:1906.03671*, 2019.

Les E Atlas, David A Cohn, and Richard E Ladner. Training connectionist networks with queries and selective sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1990.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

Isabelle Augenstein, Mrinal Das, Sebastian Riedel, Lakshmi Vikraman, and Andrew McCallum. Semeval 2017 task 10: Scienceie - extracting keyphrases and relations from scientific publications. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017.

Philip Bachman, Alessandro Sordoni, and Adam Trischler. Learning algorithms for active learning. In *ICML*, pages 301–310. JMLR. org, 2017.

J. A. Bagnell, Sham M Kakade, Jeff G. Schneider, and Andrew Y. Ng. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems 16*, pages 831–838. MIT Press, 2004.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016a.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2016b.

Nina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2006.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*, 2019.

Loïc Barrault, Ondřej Bojar, Marta R Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, et al. Findings of the 2019 conference on machine translation (wmt19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, 2019.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. ISSN 0018-9472.

Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: Pretrained language model for scientific text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015a.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, Advances in Neural Information Processing Systems (NeurIPS), pages 1171–1179. MIT Press, 2015b.

Alina Beygelzimer and John Langford. The offset tree for learning with partial labels. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 129–138. ACM, 2009.

Alina Beygelzimer, John Langford, and Bianca Zadrozny. Weighted one-against-all. In *AAAI*, pages 720–725, 2005.

Alina Beygelzimer, Sanjoy Dasgupta, , and John Langford. Importance weighted active learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009.

Alina Beygelzimer, Daniel Hsu, John Langford, and Tong Zhang. Agnostic active learning without constraints. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2010a.

Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert E Schapire. Contextual bandit algorithms with supervised learning guarantees. *arXiv preprint arXiv:1002.4058*, 2010b.

Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandit algorithms with supervised learning guarantees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 19–26. PMLR, 2011.

Alberto Bietti, Alekh Agarwal, and John Langford. Vowpal wabbit. *VW*, 2017.

Alberto Bietti, Alekh Agarwal, and John Langford. A Contextual Bandit Bake-off. working paper or preprint, 2018.

Michael Bloodgood and Chris Callison-Burch. Bucking the trend: Large-scale cost-focused active learning for statistical machine translation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, 2010.

Avrim Blum, Adam Kalai, and John Langford. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 203–208. ACM, 1999.

Kianté Brantley, Amr Sharaf, and Hal Daumé III. Active imitation learning with noisy guidance. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. ISSN 0885-6125.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Ángel Alexander Cabrera, Will Epperson, Fred Hohman, Minsuk Kahng, Jamie Morgenstern, and Duen Horng Chau. Fairvis: Visual analytics for discovering intersectional bias in machine learning. *arXiv preprint arXiv:1904.05419*, 2019.

Nicolò Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Worst-case analysis ofselective sampling for linear classification. *Journal of Machine Learning Research (JMLR)*, 2006.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé, III, and John Langford. Learning to search better than your teacher. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML, pages 2058–2066. JMLR.org, 2015.

Kai-Wei Chang, He He, Hal Daumé, III, John Langford, and Stéphane Ross. A credit assignment compiler for joint prediction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011.

Irene Chen, Fredrik D Johansson, and David Sontag. Why is my classifier discriminatory? In *Advances in Neural Information Processing Systems*, pages 3539–3550, 2018.

Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data*, 5(2):153–163, 2017.

Christos Christodouloupoulos and Mark Steedman. A massively parallel corpus: the bible in 100 languages. *Language resources and evaluation*, 49(2):375–395, 2015.

Chenhui Chu and Rui Wang. A survey of domain adaptation for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1304–1319. Association for Computational Linguistics, 2018.

Chenhui Chu, Raj Dabre, and Sadao Kurohashi. A comprehensive empirical comparison of domain adaptation methods for neural machine translation. *Journal of Information Processing*, 26:529–538, 2018.

David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine learning*, 15(2):201–221, 1994.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, 2004a.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics, 2004b.

Patrick L Combettes and Valérie R Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

Kate Crawford and Ryan Calo. There is a blind spot in ai research. *Nature*, 538(7625): 311–313, 2016.

Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2005.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Hal Daumé, III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 169–176. ACM, 2005. ISBN 1-59593-180-5.

Hal Daumé, III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning Journal*, 2009.

Hal Daumé, III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009. ISSN 1573-0565.

Hal Daumé III, John Langford, and Amr Sharaf. Residual loss prediction: Reinforcement learning with no incremental feedback. In *International Conference on Learning Representations (ICLR)*, 2018.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Pinar Donmez and Jaime G Carbonell. Proactive learning: cost-sensitive active learning with multiple imperfect oracles. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 619–628, 2008.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Hc-search: A learning framework for search-based structured prediction. *J. Artif. Intell. Res.(JAIR)*, 50:369–407, 2014.

Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. Investigating meta-learning algorithms for low-resource natural language understanding tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1192–1197. Association for Computational Linguistics, 2019.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12: 2121–2159, 2011. ISSN 1532-4435.

Miroslav Dudik, Daniel Hsu, Satyen Kale, Nikos Karampatziakis, John Langford, Lev Reyzin, and Tong Zhang. Efficient optimal learning for contextual bandits. *arXiv preprint arXiv:1106.2369*, 2011.

Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. Doubly robust policy evaluation and optimization. *Statist. Sci.*, 29(4):485–511, 2014.

Kevin Duh. The multitarget ted talks task. http://www.cs.jhu.edu/~kevinduh/a/multitarget-tedtalks/, 2018.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990. ISSN 0364-0213.

Meng Fang, Yuan Li, and Trevor Cohn. Learning how to active learn: A deep reinforcement learning approach. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 595–605. Association for Computational Linguistics, 2017.

Valerii Vadimovich Fedorov. *Theory of optimal experiments*. Elsevier, 2013.

Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *KDD*, 2015.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

Corina Florescu and Cornelia Caragea. PositionRank: An unsupervised approach to keyphrase extraction from scholarly documents. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, 2017.

Markus Freitag and Yaser Al-Onaizan. Fast domain adaptation for neural machine translation. *ArXiv*, abs/1612.06897, 2016.

Batya Friedman and Helen Nissenbaum. Bias in computer systems. *ACM Transactions on Information Systems (TOIS)*, 14(3):330–347, 1996.

Raphaël Féraud, Robin Allesiardo, Tanguy Urvoy, and Fabrice Clérot. Random forest for the contextual bandit problem. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 93–101. PMLR, 2016.

Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: Testing software for discrimination. In *Joint Meeting on Foundations of Software Engineering*, 2017.

Vivian Giang. The potential hidden bias in automated hiring systems. *Fast Company*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 2010.

Yoav Goldberg and Joakim Nivre. Training deterministic parsers with non-deterministic oracles. *Transactions of the ACL*, 1, 2013.

Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a fasta implementation. *arXiv preprint arXiv:1411.3406*, 2014.

Jiatao Gu, Yong Wang, Yun Chen, Victor O. K. Li, and Kyunghyun Cho. Meta-learning for low-resource neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3622–3631. Association for Computational Linguistics, 2018.

Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1948.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Advances in Neural Information Processing Systems*, pages 5307–5316, 2018.

Ben Hachey, Beatrice Alex, and Markus Becker. Investigating the effects of selective sampling on the annotation task. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, 2005.

Robbie Haertel, Eric K. Ringger, Kevin D. Seppi, James L. Carroll, and Peter McClanahan. Assessing the costs of sampling methods in active learning for annotation. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, 2008.

Aria Haghighi and Dan Klein. Prototype-driven learning for sequence models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*, 2006.

Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323, 2016.

Ryan B Hayward and Jack Van Rijswijck. Hex and combinatorics. *Discrete Mathematics*, 306(19):2515–2528, 2006.

Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

David P. Helmbold, Nicholas Littlestone, and Philip M. Long. Apple tasting. *Information and Computation*, 2000.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. Improving fairness in machine learning systems: What do industry practitioners need? In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2019.

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzkebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751, 2019.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jun Jiang and Horace Ho-Shing Ip. Active learning for the prediction of phosphorylation sites. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 3158–3165. IEEE, 2008.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

Kshitij Judah, Alan Paul Fern, and Thomas Glenn Dietterich. Active imitation learning via reduction to iid active learning. In *AAAI*, 2012.

Marcin Junczys-Dowmunt. Microsoft translator at wmt 2019: Towards large-scale document-level neural machine translation. In *WMT*, 2019.

Leslie Pack Kaelbling. Associative reinforcement learning: Functions ink-dnf. *Machine Learning*, 15(3):279–298, 1994.

Sham M Kakade, Shai Shalev-Shwartz, and Ambuj Tewari. Efficient bandit algorithms for online multiclass prediction. In *Proceedings of the 25th international conference on Machine learning*, pages 440–447, 2008.

Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

Nathan Kallus and Angela Zhou. Residual unfairness in fair machine learning from prejudiced data. *arXiv preprint arXiv:1806.02887*, 2018.

Zohar S Karnin and Oren Anava. Multi-armed bandits: Competing with optimal sequences. In *Advances in Neural Information Processing Systems 29*, pages 199–207. Curran Associates, Inc., 2016.

Daniel Khashabi, Mark Sammons, Ben Zhou, Tom Redman, Christos Christodoulopoulos, Vivek Srikumar, Nicholas Rizzolo, Lev Ratinov, Guanheng Luo, Quang Do, Chen-Tse Tsai, Subhro Roy, Stephen Mayhew, Zhili Feng, John Wieting, Xiaodong Yu, Yangqiu Song, Shashank Gupta, Shyam Upadhyay, Naveen Arivazhagan, Qiang Ning, Shaoshi Ling, and Dan Roth. CogCompNLP: Your swiss army knife for NLP. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*, 2018.

Huda Khayrallah, Gaurav Kumar, Kevin Duh, Matt Post, and Philipp Koehn. Neural lattice search for domain adaptation in machine translation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 20–25. Asian Federation of Natural Language Processing, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*, 2016.

Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39. Association for Computational Linguistics, 2017.

Junpei Komiyama, Junya Honda, and Hiroshi Nakagawa. Optimal regret analysis of thompson sampling in stochastic multi-armed bandit problem with multiple plays. *arXiv preprint arXiv:1506.00779*, 2015.

Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data. In *Advances in Neural Information Processing Systems*, pages 4225–4235, 2017a.

Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data. In *Advances in Neural Information Processing Systems*, pages 4225–4235, 2017b.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. *arXiv preprint arXiv:1903.06059*, 2019.

Akshay Krishnamurthy, Alekh Agarwal, and Miroslav Dudik. Efficient contextual semi-bandit learning. *arXiv preprint arXiv:1502.05890*, 2015.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71. Association for Computational Linguistics, 2018.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289, 2001.

John Langford and Alekh Agarwal. Vowpal wabbit online learning project, 2017.

John Langford and Bianca Zadrozny. Relating reinforcement learning performance to classification performance. In *Proceedings of the 22nd international conference on Machine learning*, pages 473–480. ACM, 2005.

John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems 20*, pages 817–824. Curran Associates, Inc., 2008.

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. Searnn: Training rnns with global-local losses. *arXiv preprint arXiv:1706.04499*, 2017.

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. SEARNN: Training RNNs with global-local losses. In *International Conference on Learning Representations (ICLR)*, 2018.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, 2011.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, 2016. ISSN 1532-4435.

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 661–670. ACM, ACM, 2010. ISBN 978-1-60558-799-8.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Hsuan-Tien Lin, Chih-Jen Lin, and Ruby C. Weng. A note on platt's probabilistic outputs for support vector machines. *Machine Learning*, 68(3):267–276, 2007. ISSN 1573-0565.

Pierre-Louis Lions and Bertrand Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

N. Littlestone and M. K. Warmuth. The weighted majority algorithm. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, 1989.

Michael L. Littman and Richard S Sutton. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pages 1555–1561. MIT Press, 2002.

Minh-Thang Luong and Christopher D. Manning. Stanford neural machine translation systems for spoken language domain. In *International Workshop on Spoken Language Translation*, 2015.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Francis Maes, Louis Wehenkel, and Damien Ernst. Meta-learning of exploration / exploitation strategies: The multi-armed bandit case. In *International Conference on Agents and Artificial Intelligence*, pages 100–115. Springer, 2012.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2): 313–330, 1993.

Fei Mi, Minlie Huang, Jiyong Zhang, and Boi Faltings. Meta-learning for low-resource natural language generation in task-oriented dialogue systems. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pages 3151–3157. AAAI Press, 2019. ISBN 978-0-9992411-4-1.

Paul Michel and Graham Neubig. Extreme adaptation for personalized neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 312–318. Association for Computational Linguistics, 2018.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

Eric Nalisnick and Padhraic Smyth. The amortized bootstrap. In *ICML 2017 Workshop on Implicit Models.*, 2017.

Arvind Narayanan. Translation tutorial: 21 fairness definitions and their politics. In *Proc. Conf. Fairness Accountability Transp., New York, USA*, 2018.

John F Nash. Some games and machines for playing them. *Technical Report, D-1164*, 1952.

Gergely Neu and Csaba Szepesvári. Training parsers by inverse reinforcement learning. *Mach. Learn.*, 77(2-3):303–337, 2009. ISSN 0885-6125.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

Erich Neuwirth and R Color Brewer. Colorbrewer palettes. *R package version*, pages 1–1, 2014.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287. Morgan Kaufmann Publishers Inc., 1999. ISBN 1-55860-612-2.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. Facebook FAIR's WMT19 news translation task submission. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 314–319. Association for Computational Linguistics, 2019.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *IWPT*, pages 149–160, 2003.

Joakim et. al Nivre. Universal dependencies v2.5, 2018. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems 29*, pages 4026–4034. Curran Associates, Inc., 2016.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Olutobi Owoputi, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *In Proceedings of NAACL*, 2013.

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*, pages 311–318. Association for Computational Linguistics, 2002.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

Jonas Peters, Joris M Mooij, Dominik Janzing, and Bernhard Schölkopf. Causal discovery with continuous additive noise models. *The Journal of Machine Learning Research*, 15 (1):2009–2053, 2014.

John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

Matt Post. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*, 2018.

Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013. URL http://arxiv.org/abs/1306.2597.

Piyush Rai, Avishek Saha, Hal Daumé III, and Suresh Venkatasubramanian. Domain adaptation meets active learning. In *Proceedings of the NAACL HLT 2010 Workshop on Active Learning for Natural Language Processing*, pages 27–32. Association for Computational Linguistics, 2010.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *CoNLL*, 2009.

Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *EMNLP*, 1996.

Larry Rendell. A general framework for induction and a study of selective induction. *Machine Learning Journal*, 1986.

Ellen Riloff and Janyce Wiebe. Learning extraction patterns for subjective expressions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2003.

Eric Ringger, Peter McClanahan, Robbie Haertel, George Busby, Marc Carmen, James Carroll, Kevin Seppi, and Deryle Lonsdale. Active learning for part-of-speech tagging: Accelerating corpus annotation. In *Proceedings of the Linguistic Annotation Workshop*, 2007.

Herbert Robbins. Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, pages 169–177. Springer, 1985.

Stéphane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Stéphane Ross, Geoff J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Workshop on Artificial Intelligence and Statistics (AI-Stats)*, 2011a.

Stéphane Ross, Geoffrey Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635. PMLR, 2011b.

Dan Roth and Kevin Small. Margin-based active learning for structured output spaces. In *European Conference on Machine Learning*, pages 413–424. Springer, 2006.

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)*, pages 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Sculley. Practical learning from one-sided feedback. In *KDD*, 2007.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96. Association for Computational Linguistics, 2016.

Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

Claude E Shannon. A note on the concept of entropy. *Bell System Tech. J*, 27(3):379–423, 1948.

Amr Sharaf and Hal Daumé, III. Structured prediction via learning to search under bandit feedback. In *Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing*. Association for Computational Linguistics, 2017.

Amr Sharaf and Hal Daumé III. Meta-learning contextual bandit exploration. In *Workshop on Meta-Learning*. Advances in Neural Information Processing Systems (NeurIPS), 2019.

Amr Sharaf and Hal Daumé III. Promoting fairness in learned models by learning to active learn under parity constraints. In *Workshop on Real World Experiment Design and Active Learning*. International Conference on Machine Learning, 2020.

Amr Sharaf, Hany Hassan, and Hal Daumé III. Meta-learning for few-shot NMT adaptation. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*. Association for Computational Linguistics, 2020.

Tianlin Shi, Jacob Steinhardt, and Percy Liang. Learning where to sample in structured prediction. In *Proceedings of the Workshop on Artificial Intelligence and Statistics (AI-Stats)*, 2015.

Xiaoxiao Shi, Wei Fan, and Jiangtao Ren. Actively transfer domain knowledge. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 342–357. Springer, 2008.

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. *Nature*, 529(7587):484–489, 2016.

Vikash Singh. Replace or retrieve keywords in documents at scale. *CoRR*, abs/1711.00046, 2017.

Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *Proceedings of the Conference of the European Association for Computational Linguistics (EACL)*, 2014.

Artem Sokolov, Julia Kreutzer, Christopher Lo, and Stefan Riezler. Learning structured predictors from bandit feedback for interactive NLP. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics (ACL), 2016a.

Artem Sokolov, Stefan Riezler, and Tanguy Urvoy. Bandit structured prediction for learning from partial feedback in statistical machine translation. *arXiv preprint arXiv:1601.04468*, 2016b.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

Megha Srivastava, Hoda Heidari, and Andreas Krause. Mathematical notions vs. human perception of fairness: A descriptive approach to fairness for machine learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2459–2468, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of the Workshop on Artificial Intelligence and Statistics (AI-Stats)*, pages 725–733, 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1st edition, 1998. ISBN 0262193981.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 99, pages 1057–1063, 1999.

Vasilis Syrgkanis, Akshay Krishnamurthy, and Robert Schapire. Efficient algorithms for adversarial contextual learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2159–2168. PMLR, 2016a.

Vasilis Syrgkanis, Haipeng Luo, Akshay Krishnamurthy, and Robert E Schapire. Improved regret bounds for oracle-based adversarial contextual bandits. In *Advances in Neural Information Processing Systems 29*, pages 3135–3143. Curran Associates, Inc., 2016b.

Brian Thompson, Jeremy Gwinnup, Huda Khayrallah, Kevin Duh, and Philipp Koehn. Overcoming catastrophic forgetting during domain adaptation of neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2062–2068. Association for Computational Linguistics, 2019.

Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. Active learning for natural language parsing and information extraction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1999.

Jorg Tiedemann. Parallel data, tools and interfaces in opus. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Ugur Dogan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*. European Language Resources Association (ELRA), 2012. ISBN 978-2-9517408-7-7.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technology (NAACL/HLT)*, 2003.

Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. Automated directed fairness testing. In *International Conference on Automated Software Engineering*, 2018.

Michal Valko, Nathaniel Korda, Rémi Munos, Ilias Flaounas, and Nelo Cristianini. Finite-time analysis of kernelised contextual bandits. *arXiv preprint arXiv:1309.6869*, 2013.

Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag, 1982.

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*, pages 5998–6008, 2017.

Michael Veale and Reuben Binns. Fairer machine learning in the real world: Mitigating discrimination without collecting sensitive data. *Big Data & Society*, 4(2): 2053951717743530, 2017.

Tim Vieira. Gumbel-max trick and weighted reservoir sapling. https://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weightedreservoir-sampling/, 2014.

David Vilar. Learning hidden unit contribution for adapting neural machine translation models. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 500–505. Association for Computational Linguistics, 2018.

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. Exploring and predicting transferability across nlp tasks. *arXiv preprint arXiv:2005.00770*, 2020.

Sara Wachter-Boettcher. Ai recruiting tools do not eliminate bias. *Time Magazine*, 2017.

Xuezhi Wang, Tzu-Kuo Huang, and Jeff Schneider. Active transfer learning under model shift. In *International Conference on Machine Learning*, pages 1305–1313, 2014.

Steven Whitehead. A study of cooperative mechanisms for faster reinforcement learning. Technical report, University of Rochester, 1991.

Linda F Wightman. Lsac national longitudinal bar passage study. lsac research report series. *ERIC*, 1998.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, 1992. ISSN 0885-6125.

Benjamin Wilson, Judy Hoffman, and Jamie Morgenstern. Predictive inequity in object detection. *arXiv preprint arXiv:1902.11097*, 2019.

Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306. Association for Computational Linguistics, 2016.

Mark Woodward and Chelsea Finn. Active one-shot learning. *arXiv preprint arXiv:1702.06559*, 2017.

Qianhui Wu, Zijia Lin, Guoxin Wang, Hui Chen, Börje F Karlsson, Biqing Huang, and Chin-Yew Lin. Enhanced meta-learning for cross-lingual named entity recognition with minimal resources. *arXiv preprint arXiv:1911.06161*, 2019.

Fei Xia. The part-of-speech tagging guidelines for the penn chinese treebank (3.0). *Technical Report*, 2000.

Tianbing Xu, Qiang Liu, Liang Zhao, Wei Xu, and Jian Peng. Learning to explore with meta-policy gradient. *arXiv preprint arXiv:1803.05044*, 2018.

Yuehua Xu, Alan Fern, and Sung Wook Yoon. Discriminative learning of beam-search heuristics for planning. In *IJCAI*, pages 2041–2046, 2007.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2): 207–238, 2005.

Yan Yan, Romer Rosales, Glenn Fung, and Jennifer G Dy. Active learning from crowds. In *ICML*, 2011.

Antonio Jimeno Yepes, Aurélie Névéol, Mariana Neves, Karin Verspoor, Ondrej Bojar, Arthur Boyer, Cristian Grozea, Barry Haddow, Madeleine Kittner, Yvonne Lichtblau, et al. Findings of the wmt 2017 biomedical translation shared task. In *Proceedings of the Second Conference on Machine Translation*, pages 234–247, 2017.

Mingzhang Yin, George Tucker, Mingyuan Zhou, Sergey Levine, and Chelsea Finn. Meta-learning without memorization. *arXiv preprint arXiv:1912.03820*, 2019.

Michelle Yuan, Hsuan-Tien Lin, and Jordan Boyd-Graber. Cold-start active learning through self-supervised language modeling. *arXiv preprint arXiv:2010.09535*, 2020.

Tal Zarsky. The trouble with algorithmic decisions: An analytic road map to examine efficiency and fairness in automated and opaque decision making. *Science, Technology, & Human Values*, 41(1):118–132, 2016.

Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting lexical semantic knowledge from Wikipedia and Wiktionary. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*, 2008.

Chicheng Zhang and Kamalika Chaudhuri. Active learning from weak and strong labelers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.