UNIVERSITY OF CALIFORNIA, SAN DIEGO

Learning from Temporally-Structured Human Activities Data

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Zachary C. Lipton

2017

The dissertation of Zachary C. Lipton is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Co-Chair

Co-Chair

University of California, San Diego

2017

DEDICATION

To Issachar Miron.

EPIGRAPH

Simple experiments, simple theorems, are the building blocks that help us understand more complicated systems.

— Ali Rahimi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am grateful to my advisors Julian McAuley and Charles Elkan. Prior to entering PhD, I had never written (or even read) a machine learning paper or implemented a machine learning algorithm. I would also like to thank my mentors in industry Anima Anandkumar, Li Deng, Jianfeng Gao, Lihong Li, and Alex Smola as well as my many collaborators. Any success that I have had owes significantly to these people who took chances on me, as a student, an intern, an industry scientist, and ultimately as a faculty candidate. I am also grateful to my parents Miriam and Chet and to my sister Gabrielle for their support throughout my PhD and for looking the other way when I stole our family's 2004 Toyota Corolla in order to move to California and pursue a career in computer science. Finally, I am always indebted to my partner, Annie Hui-Hsin Hsieh, who reminds me every day that research has limited significance when compared to love, sushi, and ice cream.

Most of the chapters in this dissertation resulted from collaborations with other researchers and all of them profited from inspiring conversations and critical feedback. Per-chapter acknowledgements follow:

Chapter 2, consisting of background on recurrent neural networks was written in collaboration with John Berkowitz and Charles Elkan. This review also benefited from insightful comments from Julian MacAuley, Balakrishnan Narayanaswamy, Stefanos Poulis, and Sharad Vikram. The dissertation author was the primary investigator and author of this paper.

Chapter 3, *Learning to Diagnose with LSTM RNNs*, was written in collaboration with David Kale, Charles Elkan, and Randall Wetzel. We acknowledge the NVIDIA Corporation for Tesla K40 GPU hardware donation and Professors Julian McAuley and Greg Ver Steeg for their support and advice. Finally, we thank the anonymous ICLR reviewers for their feedback, which helped us to make significant improvements to this work and manuscript. The dissertation author was the primary investigator and author of this paper.

Chapter 4, *Modeling Missing Data in Clinical Time Series with RNNs*, was written in collaboration with David Kale and Randall Wetzell. Thanks to Professors Charles Elkan, Julian McAuley and Greg Ver Steeg for their support and advice. The dissertation author was the primary investigator and author of this paper.

Chapter 5, *Evaluating Multilabel Classifiers*, was written in collaboration with Charles Elkan and Balakrishnan "Murali" Narayanaswamy. The dissertation author was the primary investigator and author of this paper.

Chapter 6, *Predicting Surgery Duration with Neural Heteroscedastic Regression* was written in collaboration with Nathan Ng, Rodney Gabriel, Charles Elkan, and Julian McAuley. The dissertation author was the primary investigator and author of this paper.

Chapter 7, *Improving Factor-Based Quantitative Investing by Forecasting Company Fundamentals*, was written in collaboration with John Alberg. The dissertation author was the primary investigator and author of this paper.

Chapter 8, *Efficient Dialogue Policy Learning with BBQ-Networks* was written in collaboration with Xiujun Li, Lihong Li, Jianfeng Gao, Faisal Ahmed, and Li Deng. The dissertation author was the primary investigator and author of this paper.

Chapter 9, *Combating Deep Reinforcement Learning's Sisyphean Curse with Intrinsic Fear* was written in collaboration with Kamyar Azizzadenesheli, Abhishek Kumar, Lihong Li, Jianfeng Gao, and Li Deng. The dissertation author was the primary investigator and author of this paper.

Chapter 10, *The Mythos of Model Interpretability*, was written solely by the dissertation author. Thanks to Charles Elkan, Julian McAuley, Maggie Makar, David Kale, Been Kim, Lihong Li, Rich Caruana, Sepp Hochreiter, Daniel Fried, and Jack Berkowitz, for helpful conversations and critical feedback. The dissertation author was the primary investigator and author of this paper.

Chapter 11, *Does Mitigating ML's Disparate Impact Require Disparate Treatment?*, was written with Alexandra Chouldechova and Julian McAuley. Thanks to Charles Elkan and Yu-Xiang Wang for critical feedback. The dissertation author was the primary investigator and author of this paper.

This research has been funded partially by the Division of Biomedical Informatics at UCSD via National Library of Medicine training grant (T15LM011271). This work was made also possible by generous gifts of research credits from Amazon Web Services and four GPUs from NVIDIA.

VITA

Starting 2018	Assistant Professor at Carnegie Mellon University
2017	Applied Scientist at Amazon AI
2017	Ph. D. in Computer Science, University of California, San Diego
2017	Graduate Teaching Assistant, University of California, San Diego
2015	M. S. in Computer Science, University of California, San Diego
2007	B. A. in Economics and Mathematics *cum laude*, Columbia University

PUBLICATIONS

Zachary C. Lipton, Jianfeng Gao, Lihong Li, Xiujun Li , Faisal Ahmed, Li Deng, “Efficient Exploration for Dialogue Policy Learning with BBQ Networks”, *AAAI*, 2018.

Zachary C. Lipton, “The Doctor Just Won’t Accept That”, *NIPS Symposium on Interpretable ML*, 2017.

John Alberg, Zachary C. Lipton, “Improving Factor-Based Quantitative Investing by Forecasting Company Fundamentals”, *NIPS Time Series Workshop*, 2017.

Jianmo Ni, Zachary C. Lipton, Sharad Vikram, Julian McAuley “Estimating Reactions and Recommending Products with Generative Models of Reviews”, *International Joint Conference on Natural Language Processing (IJCNLP)*, 2017.

Zachary C. Lipton, Alexandra Chouldechova, “Can ML Reduce Disparate Impact Without Disparate Treatment?”, *(in preparation)*, 2017.

Yanyao Shen, Hyokun Yun, Zachary C. Lipton, Yakov Kronrod, Anima Anandkumar, “Deep Active Learning for Named Entity Recognition”, *ACL Workshop on Representation Learning for NLP (#REPL4NLP)*, 2017.

Ashish Khetan Kumar, Zachary C. Lipton, Anima Anandkumar, “Learning From Noisy Singly-labeled Data by Bootstrapping EM with Model Predictions”, *(under review)*, 2017.

Guneet Dhillon Singh, Kamyar Azizzadenesheli, Aran Khanna, Jeremy Bernstein, Jean Kossaifi, Zachary C. Lipton, Anima Anandkumar, “Stochastic Activation Pruning for Robust Adversarial Defense”, *NIPS Workshop on Machine Deception*, 2017.

Tommaso Furlanello, Zachary C. Lipton, Laurent Itti, Anima Anandkumar, “Born Again Networks”, *(under review)*, 2017.

Jean Kossaifi, Aran Khanna, Zachary C. Lipton, Tommasso Furlanello, Anima Anandkumar, “Tensor Contraction Layers for Parsimonious Deep Nets”, *CVPR Workshop on Tensor Methods in Computer Vision*, 2017.

Chris Donahue, Julian McAuley, Zachary C. Lipton, “Semantically Decomposing the Latent Spaces of Generative Adversarial Networks”, *NIPS Workshop - Learning Disentangled Features: from Perception to Control*, 2017.

Chris Donahue, Zachary C Lipton, Julian McAuley, “Dance Dance Convolution (conference version)”, *ICML*, 2017.

Nathan Ng, Rodney A Gabriel, Julian McAuley, Charles Elkan, Zachary C. Lipton, “Predicting Surgery Duration with Neural Heteroscedastic Regression”, *Machine Learning for Healthcare*, 2017.

Zachary C. Lipton, Subarna Tripathi, “Precise Recovery of Latent Vectors for Generative Adversarial Networks”, *ICLR Workshop Track*, 2017.

Chris Donahue, Zachary C. Lipton, Julian McAuley, “Dance Dance Convolution (workshop version)”, *ICLR Workshop Track*, 2017.

Zachary C. Lipton, Kamyar Azizzadenesheli, Abhishek Kumar, Jianfeng Gao, Lihong Li, Li Deng, “Combating Deep Reinforcement Learning’s Sisyphean Curse with Intrinsic Fear”, *NIPS Workshop on Reliable ML in the Wild*, 2016.

Zachary C. Lipton, Jianfeng Gao, Lihong Li, Xiujun Li , Faisal Ahmed, Li Deng, “Efficient Exploration for Dialogue Policy Learning with BBQ Networks & Replay Buffer Spiking”, *NIPS Workshop on Deep Reinforcement Learning*, 2016.

Zachary C. Lipton, David C. Kale, Randall Wetzel, “Modeling Missing Data in Clinical Time Series with RNNs”, *Machine Learning for Healthcare*, 2016.

Subarna Tripathi, Zachary C. Lipton, Serge Belongie, Truong Nguyen, “Context Matters: Refining Object Detection in Video with Recurrent Neural Networks”, *British Machine Vision Conference*, 2016.

Zachary C. Lipton, David C. Kale, Randall Wetzel, “Modeling Missing Data in Clinical Time Series with RNNs”, *Machine Learning for Healthcare*, 2016.

Zachary C. Lipton, “The Mythos of Model Interpretability”, *ICML Workshop on Human Interpretability in Machine Learning*, 2016.

Zachary C. Lipton, David C. Kale, Charles Elkan, Randall Wetzel, “Learning to Diagnose with LSTM Recurrent Neural Networks ”, *ICLR*, 2016.

Zachary C. Lipton, Charles Elkan, “Playing the Imitation Game with Deep Learning”, *IEEE Spectrum*, 2016.

Zachary C. Lipton, Sharad Vikram, Julian McAuley, “Generative Concatenative Nets Jointly Learn to Write and Classify Reviews”, *arXiv*, 2015.

Zachary C. Lipton, David C. Kale, Randall Wetzel, “Phenotyping of Clinical Time Series with LSTM Recurrent Neural Networks”, *NIPS Workshop on Machine Learning for Healthcare*, 2015.

Zachary C. Lipton, John Berkowitz, Charles Elkan, “A Critical Review of Recurrent Neural Networks for Sequence Learning”, *arXiv*, 2015.

Zachary C. Lipton and Charles Elkan, “Efficient Elastic Net Regularization for Sparse Linear Models”, *arXiv*, 2015.

Zhanglong Ji, Zachary C. Lipton, Charles Elkan, “Differential Privacy and Machine Learning: A Survey and Review”, *arXiv*, 2014.

Zachary C. Lipton, Charles Elkan, Balakrishnan Narayaswamy, “Optimal Thresholding of Classifiers to Maximize F1 Measure”, *ECML*, 2014.

ABSTRACT OF THE DISSERTATION

Learning from Temporally-Structured Human Activities Data

by

Zachary C. Lipton

Doctor of Philosophy in Computer Science

University of California, San Diego, 2017

Assistant Professor Julian McAuley, Co-Chair
Professor Charles Elkan, Co-Chair

Despite the extraordinary success of deep learning on diverse problems, these triumphs are too often confined to large, clean datasets and well-defined objectives. Face recognition systems train on millions of perfectly annotated images. Commercial speech recognition systems train on thousands of hours of painstakingly-annotated data. But for applications addressing human activity, data can be noisy, expensive to collect, and plagued by missing values. In electronic health records, for example, each attribute might be observed on a different time scale. Complicating matters further, deciding precisely what objective warrants optimization requires critical consideration of both algorithms and the application domain. Moreover, deploying human-interacting systems requires careful consideration of societal demands such as safety, interpretability, and fairness.

The aim of this thesis is to address the obstacles to mining temporal patterns in human activity data. The primary contributions are: (1) the first application of RNNs to multivariate clinical time series data, with several techniques for bridging long-term dependencies and modeling missing data; (2) a neural network algorithm for forecasting surgery duration while simultaneously modeling heteroscedasticity; (3) an approach to quantitative investing that uses RNNs to forecast company fundamentals; (4) an exploration strategy for deep reinforcement learners that significantly speeds up dialogue policy learning; (5) an algorithm to minimize the number of catastrophic mistakes made by a reinforcement learner; (6) critical works addressing model interpretability and fairness in algorithmic decision-making.

Chapter 1

Introduction

Since the earliest conception of artificial intelligence, we have sought to build systems that interact with humans in real-time. In Alan Turing's groundbreaking paper *Computing Machinery and Intelligence*, he proposes an "Imitation Game" which judges a machine's intelligence by its ability to convincingly engage in dialogue [Tur50]. The same year, science fiction writer Isaac Asimov published *I, Robot*, a collection of short stories about artificially intelligent robots that he wrote over the course of the preceding decade. These stories describe robots that play with children, provide manual labor on a mining operation on Mercury, pilot spaceships, and organize the world economy. Already today some of the most widely anticipated applications artificial intelligence relate to medical care, self-driving cars, and dialogue systems.

While Turing's test remains unpassed and Asimov's prognostications remain unfulfilled, the field of artificial intelligence entered a period of accelerated development. Over the past decade, deep neural networks have significantly advanced the state of the art in pattern recognition. And in constrained settings, neural networks can now outperform humans. For example, given large datasets of images where each image has a fixed size and belongs to one of a limited set of categories, where each image is labeled, and where each class is well-represented in the training data, and where the training and test data are samples from the same underlying distribution, modern deep neural networks can approach human-level performance.

Unfortunately, the real world does not always oblige, especially when learning from the dynamics of human activities. Consider the hospital setting, where we might want to design an artificially intelligent diagnostician. Real data collected in the hospital might consist of complicated objects, like trajectories through time. Moreover the robo-diagnositician can only access ground-truth labels at certain points in time, while at other times labels are might be corrupted or missing altogether. Moreover, even the input data are frequently missing. Take, for example, a patient who goes for a walk during hourly rounds or momentarily detaches from a heart rate sensor. Complicating things further, the set of available diagnosis codes and the protocol according to which they are applied may change over time.

The limitations of current machine learning techniques become more pronounced when we move from considering an inference-engine that only makes predictions to an agent that takes actions in the real world, altering its environment in the process. Following the success of deep reinforcement learning (DRL) agents on video games and board games [MKS$^+$13, SHM$^+$16], we might hope to apply DRL directly to human-interacting systems. However, subjecting human subjects to millions of failed dialogues, for example, might exceed the budget of even a wealthy organization.

Finally, taking consequential actions based on the outputs of machine learning systems in any social context raises several issues. First, if a machine learning agent, such as a self-driving car can take potentially catastrophic actions, we need to mitigate any potential harm that might come. Second, the law is concerned with the reasons for which decisions are made. However, offering explanations for the output generated by a pattern-recognition system is not straightforward. Finally, actions taken by an algorithmic decision-maker might affect various subpopulations differently, even those delineated according to legally protected characteristics. Deploying decision-making systems based on such models could potentially run afoul of legal and ethical principles. Building human-interacting systems requires a sophisticated understanding of the trade-offs between utility and various criteria for fairness.

1.1 Overview of Contributions

This dissertation presents a collection of works all of which aim to address the challenges of building artificially intelligent agents that learn from human activities and interact with humans in time. The primary contributions address several distinct challenges: Learning patterns given long-term temporal dependencies; accounting for the presence and significance of missing data; forecasting the future; modeling uncertainty owing to heteroscedasticity in real-world datasets; exploring as efficiently as possible to produce high-performing reinforcement learning policies. This thesis also contributes critical explorations on the issues of safety, interpretability, and fairness.

The thesis is organized as follows: Chapter 2 provides a brief background on feedforward and recurrent neural networks. Additional background on reinforcement learning, dialogue systems, etc will be introduced in subsequent chapters as necessary. A brief overview of the main contributions in each subsequent chapter follows:

Learning to Diagnose with LSTM Recurrent Neural Networks In this chapter, we present the first paper to apply modern recurrent neural networks to recognizing diagnoses in multivariate time series of clinical medical data. Focusing on the 13 most frequently sampled features widely available in the electronic health records of intensive care unit patients, we show that we can recognize a large number of diagnoses with surprising accuracy.

Modeling Missing Data in Clinical Time Series with RNNs Medical data tends to be plagued by missing values. Each measurement is only observed for some patients. Moreover re-sampling to discrete time series results in additional missing values - not every measurement is observed in every time window. In this chapter, we present a simple technique for capturing the signal in missing data showing substantial improvements in predictive performance.

Optimal Thresholding of Multilabel Classifiers to Maximize F1 Score In many real-world datasets, many labels are available. For example in the previous chapters, we build classifiers that simultaneously recognize 128 different diagnoses. How best to evaluate multi-label classifiers remains an open question, especially in the case when each label has a distinct base rate. Macro-, micro-, and per-example-averaged F1-score are popular metrics for evaluating multilabel classifiers. In this chapter, we characterize the optimal thresholds to maximize these multi-label variants of F1 score, showing that setting optimal thresholds for maximizing F1 score does not agree with reasonable behavior for typical real-world tasks.

Predicting Surgery Duration with Neural Heteroscedastic Regression In medicine, we often want to forecast future outcomes from current knowledge. This paper presents a simple technique for predicting all the parameters of a predictive distribution using neural networks. We predict surgery durations accurately, improving significantly on current practice for scheduling surgeries. Moreover, we present useful estimates of uncertainty by predicting the conditional variance.

Improving Factor-Based Quantitative Investing by Forecasting Company Fundamentals While supervised learning systems assume that training data is representative of future data, this assumption rarely holds in the long run as concerns human activities.

As one example, the stock market may behave according to different dynamics now than in previous decades. In this paper we present the first academic paper to consider the application of modern RNNs to fundamental stock market data. Considering the full universe of publicly traded stock data over several decades, we show that owing to nonstationarity, predicting price directly does not perform reliably out of sample. However, by predicting more stable patterns such as the future fundamental reporting data given a trailing window, we apply factor-based quantitative investing strategies to our forecasted fundamentals, showing that the approach significantly outperforms traditional factor models and those which predict price directly, as measured through industry-grade simulations.

Efficient Exploration for Dialogue Policy Learning with BBQ Networks. Ultimately, to build systems that not only make predictions but also take actions in time, we require that they explore their environments efficiently. If a reinforcement learner required millions of dialogues to become a competent interlocutor, it might have no commercial or societal value. In this paper we present to our knowledge, the first paper to apply Bayesian neural networks to achieve efficient exploration in deep reinforcement learners.

Critical Considerations - Safety, Interpretability, and Fairness Applying machine learning in human-interacting systems brings up a number of critical questions. How can we guard a reinforcement learner against causing catastrophic mistakes? How can we explain the decisions of an algorithmic decision-maker as might be required under the law? How can we be reconcile the utility of a machine learning system with certain fairness criteria, as determined by society and the law? Answering these questions requires interdisciplinary investigation and critical consideration of not only technical ideas but of problem formulations themselves. In the final part of this thesis, we present three papers addressing each of these issues in turn. We offer an algorithm to guard a reinforcement learner against repeatedly making catastrophic mistakes while exploring its environment, a critical investigation of the foundations of model interpretability, and a theoretical and empirical study of the irreconcilability of minimizing both disparate treatment and disparate impact, two widely-referenced fairness criteria.

Chapter 2

Background on Recurrent Neural Networks

Countless learning tasks require dealing with sequential data. Image captioning, speech synthesis, music generation, and video game playing all require that a model generate sequences of outputs. In other domains, such as time series prediction, video analysis, and music information retrieval, a model must learn from sequences of inputs. Significantly more interactive tasks, such as natural language translation, engaging in dialogue, and robotic control, often demand both.

Recurrent neural networks (RNNs) are a powerful family of connectionist models that capture time dynamics via cycles in the graph. Unlike feedforward neural networks, recurrent networks can process examples one at a time, retaining a state, or *memory*, that reflects an arbitrarily long context window. While these networks have long been difficult to train and often contain millions of parameters, recent advances in network architectures, optimization techniques, and parallel computation have enabled large-scale learning with recurrent nets.

Over the past several years, systems based on state of the art long short-term memory (LSTM) and bidirectional recurrent neural network (BRNN) architectures have demonstrated record-setting performance on tasks as varied as image captioning, language translation, and handwriting recognition. In this chapter, we synthesize the body of research that over the past three decades has yielded and reduced to practice these powerful models. When appropriate, we reconcile conflicting notation and nomenclature. Our goal is to provide a mostly self-contained explication of state of the art systems, together with a historical perspective and ample references to the primary research.

2.1 Introduction

RNNs are a superset of feedforward neural networks, augmented with the ability to pass information across time steps. They are a rich family of models capable of nearly arbitrary computation. A well-known result by Siegelman and Sontag from 1991 demonstrated that a finite sized recurrent neural network with sigmoidal activation functions can simulate a universal Turing machine [SS91]. In practice, the ability to model temporal dependencies makes recurrent neural networks especially suited to tasks where input and/or output consist of *sequences* of points that are not independent.

2.1.1 Why Recurrent Nets?

In this section, we address the fundamental reasons why recurrent neural networks warrant serious study for modeling sequential input and output. To be clear, we are motivated by a desire to achieve empirical results. This warrants clarification because recurrent nets have roots in both cognitive modeling and supervised machine learning, and owing to this difference of perspectives, many of these papers have different aims and priorities. In the foundational papers, generally published in cognitive science and computational neuroscience journals ([Hop82], [Jor97], [Elm90]), biologically plausible mechanisms are emphasized. In other papers ([SP97], [SKL$^+$14], [KFF14]), biological inspiration is downplayed in favor of achieving empirical results on important tasks and datasets. Given the empirical aim, we now address three significant questions one might reasonably want answered before reading further.

Why Model Time Explicitly?

In light of the practical success and economic value of time-agnostic models, this is a fair question. Support vector machines, logistic regression, and feedforward networks have proved immensely useful without explicitly modeling time. Arguably, it is precisely the assumption of independence that has led to much recent progress in machine learning. Further, many models implicitly capture time by concatenating each input with some number of its immediate predecessors and successors, presenting the machine learning model with a sliding window of context about each point of interest. This approach has been used with deep belief nets for speech modeling in [MLO$^+$12].

Unfortunately, despite the usefulness of the independence assumption, it precludes modeling long-range time-dependencies. For example, a model trained using a finite-length context window of length 5 could never be trained to answer the simple question, *"what was the data point seen* 10 *time steps ago?"* For a practical application such as call center automation, such a limited system might learn to route calls, but could never participate in an extended dialogue. Besides dialogue systems, modern interactive systems of economic importance include self-driving cars and robotic surgery, among others.

Why Neural Networks and Not Markov Models?

Recurrent neural networks are not the first models to capture time dependencies. Markov chains, which model transitions between observed sequences of states ($s^{(1)}$, $s^{(2)}$,

... , $s^{(T)}$), were first described in 1906. Hidden Markov models (HMMs), which model observed data $(o^{(1)}, o^{(2)}, ..., o^{(T)})$ as probabilistically dependent upon unobserved states, were described in the 1950s and have been widely studied since the 1960s. However, traditional approaches to sequence learning with HMMs are limited because their states must be drawn from a modestly sized discrete state space $s_j \in S$. The Viterbi algorithm, which is used to perform efficient inference on hidden Markov models, scales in time $O(|S|^2)$. Further, the transition table capturing the probability of moving between any two adjacent states is of size $|S|^2$. Thus, standard operations are infeasible with an HMM when the set of possible hidden states is larger than roughly 10^6 states. Further, each hidden state $s^{(t)}$ can depend only on the previous state $s^{(t-1)}$. While it is possible to extend any Markov model to account for a larger context window by creating a new state-space equal to the cross product of the possible states at each time in the window, this procedure grows the state space exponentially with the size of the window, rendering Markov models computationally impractical for modeling long-range dependencies.

Given the limitations of Markov models, we ought to explain why it is sensible that artificial neural networks, should fare better. First, RNNs can capture long-range time dependencies, overcoming one chief limitation of Markov models. This point requires a careful explanation. As in Markov models, any state in a traditional RNN depends only on the current input as well as the state of the network at the previous time step [1]. However, the hidden state at any time step can contain information from an arbitrarily long context window. This is possible because the number of distinct states that can be represented in a hidden layer of nodes grows exponentially with the number of nodes in the layer. Even if each node took only binary values, the network could represent 2^N states where N is the number of nodes in the hidden layer. Given real-valued outputs,

[1] Bidirectional recurrent neural networks (BRNNs) [SP97] extend RNNs to model dependence on both past states and future states. Traditional RNNs only model the dependence of each event on the past. This extension is especially useful for sequence to sequence learning with fixed length examples.

even assuming the limited precision of 64 bit numbers, a single hidden layer of nodes can represent 2^{64^N} distinct states. While the potential expressive power grows exponentially with the number of nodes in the hidden representation, the complexity of both inference and training grows only quadratically.

Second, from an empirical standpoint, it is generally reasonable to extend neural networks to tackle any supervised learning problem because at present, they achieve state-of-the-art results on a wide range of supervised learning tasks. Over the past several years, storage has become more affordable, datasets have grown far larger, and the field of parallel computing has advanced considerably. Given such large high-dimensional datasets, linear models tend to under-fit data and under-utilize computing resources. Deep neural networks (DNNs), and convolutional neural networks (CNNs), which exploit the local dependency of visual information, have demonstrated record-setting results on many important applications. Neural networks are especially well-suited for machine perception tasks, where the raw underlying features are not individually informative. This success is attributed to their ability to learn hierarchical representations, unlike traditional algorithms, which rely upon hand-engineered features. However, despite their power, feedforward neural nets have limitations. Most notably, they rely on the assumption of independence among the data points. Additionally, these networks generally rely on input consisting of fixed length vectors. Thus it is sensible to extend these powerful learning tools to model data with temporal structure, especially in the many domains where neural nets are already the state of the art.

Are RNNs Too Expressive?

As described earlier, finite-sized RNNs with sigmoidal activations are Turing complete. The capability of RNNs to run arbitrary computation clearly demonstrates their expressive power, but one could argue that the C programming language is equally capable of expressing arbitrary programs. And yet there are no papers claiming that the invention of C represents a panacea for machine learning. Out of the box, C offers no simple way of efficiently exploring the space of programs. There is no straightforward way to calculate the gradient of an arbitrary C program to minimize a chosen loss function. Further, the biggest problem with treating the set of programs expressible in C as a family of machine learning models is that this set is far too large. Given any finite sized dataset, there exist countless programs which can overfit the data, generating desired output but failing to generalize to test data.

Why then should RNN's not suffer from similar problems? First, given any fixed architecture (set of nodes, edges, and activation functions), the recurrent neural networks described in this chapter are fully differentiable end to end. The derivative of the loss function can always be calculated with respect to each of the parameters (weights) in the model. Second, while the Turing-completeness of finite-length RNNs is an impressive property, given any fixed-size RNN and a specific architecture, it is not actually possible to generate any arbitrary program. Further, unlike an arbitrary program composed in C, a recurrent neural network can be regularized via standard techniques such as weight decay, dropout, and limiting the degrees of freedom.

2.2 Preliminaries

We now introduce some formal notation and provide a brief background on neural networks. RNNs are not limited to sequences which index time. They have been used successfully on non-temporal sequence data, including genetic data [BP03]. However, computation proceeds in time, and many important applications have an explicit or implicit temporal aspect. While we refer to time steps throughout this thesis, the methods described here are applicable to wider family of sequential tasks. Thesis concerns data points $x^{(t)}$ and and desired outputs $y^{(t)}$ that arrive in a discrete *sequence* of *time steps* indexed by t. We use superscripts with parentheses and not subscripts to obviate confusion between time steps and neurons. Our sequences may be of finite length or countably infinite. When they are finite, we call the maximum time index of the sequence T. Thus a sequence of consecutive inputs can be denoted $(x^{(1)}, x^{(2)}, ..., x^{(T)})$ and outputs can be notated $(y^{(1)}, y^{(2)}, ..., y^{(T)})$ These time steps may be equally spaced samples from a continuous real-world process. Examples would include the still images that comprise the frames of videos or the discrete amplitudes sampled at fixed intervals to comprise audio recordings. The time steps may also be ordinal, with no exact correspondence to durations. In fact, these techniques can be extended to domains including genetic sequences, where the sequence has a defined order but no real correspondence to time. This is the case with natural language. In the word sequence "John Coltrane plays the saxophone", $x^{(1)} =$ John, $x^{(2)} =$ Coltrane, etc.

2.2.1 Neural Networks

Neural networks are biologically inspired models of computation. Generally, a neural network consists of a set of *artificial neurons*, commonly referred to as *nodes* or *units*, and a set of directed edges between them, which intuitively represent the *synapses* in a biological neural network. Associated with each neuron j is an activation function l_j, which is sometimes called a link function. We use the notation "l_j" and not "h_j" (unlike some other papers) to distinguish the activation function l_j from the values of the hidden nodes in a network, which is commonly notated h in the RNN literature.

Associated with each edge from node j' to j is a weight $w_{jj'}$. Following the convention adopted in several foundational recurrent net papers ([HS97], [Ger01], [GSC00], [SMH11]), we index neurons with j and j', and by $w_{jj'}$, we denote the weight corresponding to the directed edge from node j' to node j. It is important to note that in many papers, textbooks, and lecture notes, the indices are flipped and $w_{j'j} \neq w_{jj'}$ denotes the weight on the directed edge from the node j' to the node j as in [Elk] and on Wikipedia [Wik15].

The value v_j of each neuron j is calculated by applying its activation function to a weighted sum of its inputs (Figure 2.1):

$$v_j = l_j \left(\sum_{j'} w_{jj'} \cdot v_{j'} \right).$$

For convenience, we term the weighted sum inside the parenthesis the *incoming activation* and notate it as a_j. We represent this entire process in figures by depicting neurons as circles and edges as arrows connecting them. When appropriate, we mark the exact activation function with a symbol, e.g., σ for sigmoid.

Common choices for the activation function include the sigmoid $\sigma(z) = 1/(1+e^{-z})$ and the *tanh* function $\phi(z) = (e^z - e^{-z})/(e^z + e^{-z})$ which has become common in feedforward neural nets and was applied to recurrent nets in [SMH11]. Another activation which has become the state of the art in deep learning research is the rectified linear unit (ReLU) $l_j(z) = \max(0, z)$. These units have been demonstrated to improve the performance of many deep neural networks ([MLO$^+$12], [NH10], [ZRM$^+$13]) on tasks as varied as speech processing and object recognition, and have been used in recurrent neural networks by [BBLP13].

The activation function at the output nodes depends upon the task. For multi-class classification, we apply a softmax nonlinearity to the layer. The softmax function calculates

$$\hat{y}_k = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}}$$

where K is the total number of outputs. The denominator is normalization consisting of a sum of exponentials over all output nodes, ensuring that the output sums to 1. For multilabel classification the activation function is simply a point-wise sigmoid. For regression we may have linear output. Due to the overwhelming number of current applications involving multi-class classification, especially for recurrent nets, in this thesis, unless otherwise specified, we assume that softmax is applied at the output.

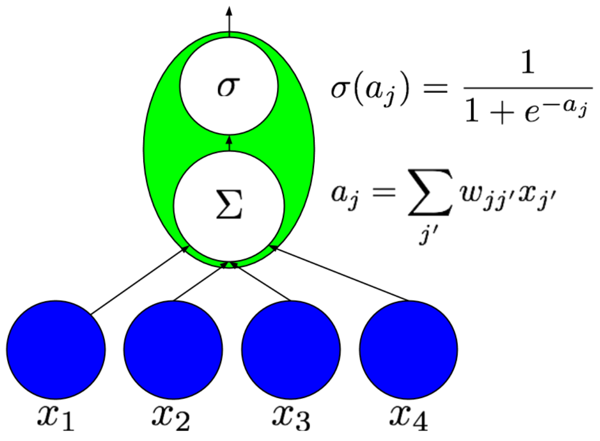

σ
$\sigma(a_j) = \frac{1}{1+e^{-a_j}}$
Σ
$a_j = \sum_{j'} w_{jj'} x_{j'}$
x_1
x_2
x_3
x_4

Figure 2.1: An artificial neuron computes a nonlinear function of a weighted sum of its inputs.

2.2.2 Feedforward Neural Networks

With a neural model of computation, one must determine the order in which computation should proceed. Should nodes be sampled one at a time and updated, or should the value of all nodes be calculated at once and then all updates applied simultaneously? Feedforward neural networks (Figure 2.2) are a restricted class of neural networks which deal with this problem by forbidding cycles in the graph. Thus all nodes can be arranged into layers. The outputs in each layer can be calculated given the outputs from the lower layers.

The input *x* to a feedforward network is presented by setting the values of the lowest layer. Each higher layer is then successively computed until output is generated at

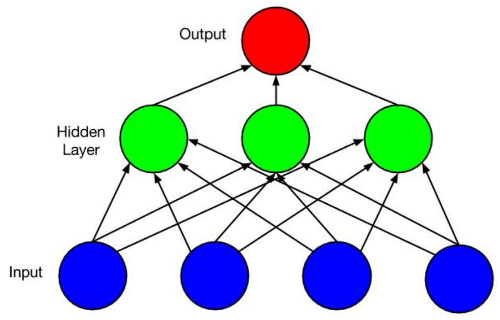


Figure 2.2: A feedforward neural network. An example is presented to the network by setting the values of the blue nodes. Each layer is calculated successively as a function of the prior layers until output is produced at the topmost layer.

the topmost layer $\hat{y}$. These networks are frequently used for supervised learning tasks such as classification and regression. Learning is accomplished by iteratively updating each of the weights to minimize a loss function, $\mathcal{L}(\hat{y}, y)$, which penalizes the distance between the output $\hat{y}$ and the target y. Backpropagation, an algorithm introduced to neural networks in [RHW85], computes the gradient of the loss with respect to each parameter using the chain rule. While the optimization surfaces for neural networks are highly non-convex, and exact optimization is known to be an NP-Hard problem, a large body of work on heuristic pre-training and optimization techniques has led to impressive empirical success on many supervised learning tasks. Convolutional neural networks [LCDH$^+$90] are a variant of feedforward neural network that, since 2012, accomplish state-of-the-art results in many computer vision tasks, such as object recognition [KSH12], detection [RDGF16], and semantic segmentation [LSD15].

Feedforward networks, however, have some limitations. After each example is processed, the entire state of the network is lost. If each data point is independently sampled, this presents no problem. But if data points are related in time, this may not be desirable. Frames from video, snippets of audio, and words pulled from sentences, represent settings where the independence assumption fails.

2.2.3 Training Neural Networks via Backpropagation

The most successful algorithm for training neural networks is backpropagation, introduced to neural networks by Rumelhart et al. in 1985 [RHW85]. Backpropagation uses the chain rule to calculate the derivative of a loss function $\mathcal{L}$ with respect to each parameter in the network. The weights are then adjusted by gradient descent. Because the loss surface is non-convex, there is no assurance that backpropagation will reach a global minimum. However, in practice, networks trained with backpropagation and gradient following techniques have been remarkably successful. In practice, most networks are trained with stochastic gradient descent (SGD) using mini-batches. Here, w.l.o.g. we discuss only the case with batch size equal to 1. The stochastic gradient update equation is given by

$$w \leftarrow w - \eta \nabla_w F_i \tag{2.1}$$

where η is the learning rate and $\nabla_w F_i$ is the gradient of the objective function with respect to the parameters w as calculated on a single example (x_i, y_i).

In practice, many variants of SGD are used to accelerate learning. Some popular heuristics, such as AdaGrad [DHS11], AdaDelta [Zei12], and RMSprop [TH12], adaptively tune the learning rate for each feature. AdaGrad, arguably the most popular, adapts the learning rate by caching the sum of squared gradients with respect to each parameter at each time step. The step size for each feature is scaled to the inverse of this cache. This leads to fast convergence on convex error surfaces, but because the cached sum is monotonically increasing, AdaGrad has a monotonically decreasing learning rate. This may be undesirable on highly non-convex loss surfaces. RMSprop modifies AdaGrad by introducing a decay factor on the cache, transforming the monotonic growing cache into a moving average. Momentum methods are another common SGD variant used to train neural networks. These methods add to each update a decaying sum of the previous updates. When the momentum parameter is well-tuned and the network is initialized well, momentum methods can train deep nets and recurrent nets competitively with more computationally expensive methods like the Hessian Free optimizer [SMDH13].

To calculate the gradient in a feedforward neural network, backpropagation proceeds as follows. First, an example is forward propagated through the network to produce a value v_j at each node and outputs $\hat{y}$ at the topmost layer. Then, a loss function $\mathcal{L}(\hat{y}_k, y_k)$ is assessed at each output node k. Subsequently, for each output node k, we can calculate

$$\delta_k = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial \hat{y}_k} \cdot l'_k(a_k). \tag{2.2}$$

Given these values δ_k, for each node in the level prior we can calculate

$$\delta_j = l'(a_j) \sum_k \delta_k \cdot w_{kj}. \tag{2.3}$$

This calculation is performed successively for each lower layer to calculate δ_j for every node j given the δ values for each node connected by an outgoing edge. Each value δ_j represents the derivative $\partial \mathcal{L}/a_j$ of the total loss function w.r.t. that node's *incoming activation*. Given the values v_j calculated on the forward pass, and the values δ_j calculated on the backward pass, the derivative of the loss $\mathcal{L}$ with respect a given parameter $w_{jj'}$ is given by

$$\frac{\partial \mathcal{L}}{\partial w_{jj'}} = \delta_j v_{j'}.$$

Large scale feedforward neural networks trained via backpropagation have set many large-scale machine learning records, most notably on the computer vision task of object detection ([LXG$^+$14], [KSH12]).

Several other methods have been explored for learning the weights in a neural network. Early work, including Hopfield nets [Hop82], learned via a Hebbian principle but did not produce networks useful for discriminative tasks. A number of papers from the 1990s ([BMS90], [G$^+$94]) championed the idea of learning neural networks with genetic algorithms with some even claiming that achieving success on real-world problems by applying many small changes to a network's weights was impossible. Despite the subsequent success of backpropagation, interest in genetic algorithms persists. Several recent papers explore genetic algorithms for neural networks, especially as means of learning the architecture of neural networks, a problem not addressed by backpropagation ([BWTS09], [HS13]). By *the architecture* we mean the number of layers, the number of nodes in each, the connectivity pattern among the layers, the choice of activation functions, etc.

One open question in neural network research is how to exploit sparsity in training. In a neural network with sigmoidal or tanh activation functions, the nodes in each layer never take value exactly 0. Thus, even if the inputs are sparse, the nodes at each hidden layer are not. However, a rectified linear units (ReLUs) introduce sparsity to hidden layers [GBB11]. In this setting, a promising path may be to store the sparsity pattern when computing each layer's values and use it to speed up computation of the next layer in the network. A growing body of recent work ([Car08], [LLZ09], [SD09], [LE15]), thus read shows that given sparse inputs to a linear model with any standard regularizer, sparsity can be fully exploited even if the gradient is not sparse (owing to regularization). Given sparse hidden layers, these approaches can be extended to the layer-wise computations in neural networks.

2.3 Recurrent Neural Networks

Recurrent neural networks are a strict superset of feedforward neural networks, augmented by the inclusion of recurrent edges that span adjacent time steps, introducing a notion of time to the model. While RNNs may not contain cycles among the conventional edges, recurrent edges may form cycles, including self-connections. At time t, nodes receiving input along recurrent edges receive *input activation* from the current example $x^{(t)}$ and also from hidden nodes $h^{(t-1)}$ in the network's previous state. The output $\hat{y}^{(t)}$ is calculated given the hidden state $h^{(t)}$ at that time step. Thus, input $x^{(t-1)}$ at time $t-1$ can influence the output $\hat{y}^{(t)}$ at time t by way of these recurrent connections.

We can show in two equations, all calculations necessary for computation at each time step on the forward pass in a simple recurrent neural network:

$$h^{(t)} = \sigma(W_{hx}x + W_{hh}h^{(t-1)} + b_h)$$

$$\hat{y}^{(t)} = softmax(W_{yh}h^{(t)} + b_y)$$

Here W_{hx} is the matrix of weights between the input and hidden layers and W_{hh} is the matrix of recurrent weights between the hidden layers at adjacent time steps. The vectors b_h and b_y are biases which allow each node to learn an offset.

Most of the models discussed in this chapter consist of networks with recurrent hidden layers. However, some proposed models, such as Jordan Networks, allow for connections between the outputs in one state and the hidden layer in the next. Others, such as Sutskever et al.'s model for sequence to sequence learning [SVL14], compute the output at each time step and pass a representation of this information as the input at the following time step.

A simple recurrent network is depicted in Figure 2.3. The dynamics of this

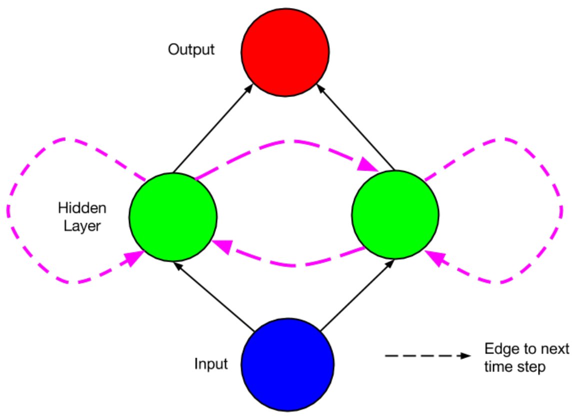


Figure 2.3: A simple recurrent network. At each time step t, activation is passed along solid edges as in a feedforward network. Dashed edges connect the source node j' at time t, i.e., $j'^{(t)}$ to the target node at the following time step $j^{(t+1)}$.

network across time steps can be visualized by *unfolding* the network (Figure 2.4). Given this picture, the model can be interpreted not as cyclic, but rather as a deep network with one layer per time step and shared weights across time steps. It's then clear that the unfolded network can be trained across many time steps using backpropagation. This algorithm is called *backpropagation through time* (BPTT), and was introduced in 1990 [Wer90].

2.3.1 Training Recurrent Networks

Learning with recurrent neural networks has long been considered to be difficult. As with all neural networks, the optimization is NP-Hard. But learning on recurrent networks can be especially hard due to the difficulty of learning long-range dependencies as described by Bengio et al in 1994 [BSF94] and expanded upon in [HBF01]. The well known problems of *vanishing* and *exploding* gradients occur when propagating errors across many time steps. As a trivial example, consider a network with a single input node,

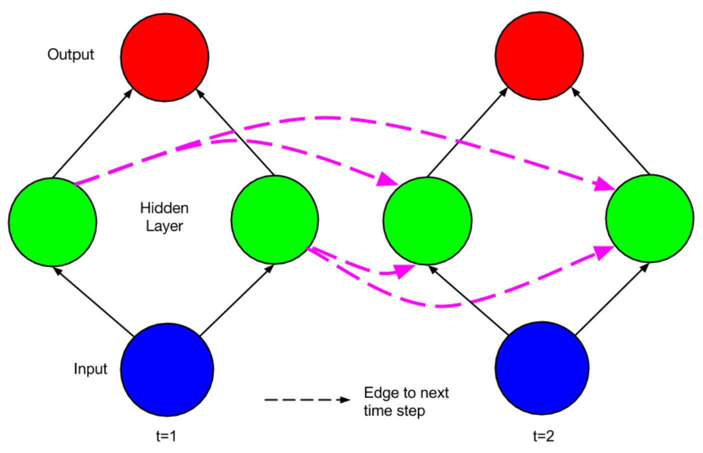


Figure 2.4: Visualizing the network unfolded across time steps.

a single output node, and a single recurrent hidden node (Figure 2.5). Now consider an input passed to the network at time τ and an error calculated at time t, assuming input of 0 in the intervening time steps. Owing to the weight tying across time steps (the recurrent edge at hidden node j always has the same weight), the impact of the input at time τ on the output at time t will either explode exponentially or rapidly approach zero as $t - \tau$ grows large, depending on whether the weight $|w_{jj}| > 1$ or $|w_{jj}| < 1$ and also upon the activation function in the hidden node (Figure 2.6). Given activation function $l_j = \sigma$, the vanishing gradient problem is more pressing, but with a rectified linear unit $max(0,x)$, it's easier to imagine the exploding gradient, even with this trivial example. In [PMB12], Pascanu et al. give a thorough mathematical treatment of the vanishing and exploding gradient problems, characterizing exact conditions under which these problems may occur. Given these conditions under which the gradient may vanish or explode, they suggest an approach to training via a regularization term, which forces the weights to values where the gradient neither vanishes nor explodes.

Truncated backpropagation through time (TBPTT) is one solution to this problem for continuously running networks [WZ89]. With TBPTT, some maximum number of time steps is set along which error can be propagated. While TBPTT with a small cutoff can be used to alleviate the exploding gradient problem, it requires that one sacrifice the ability to learn long-range dependencies.

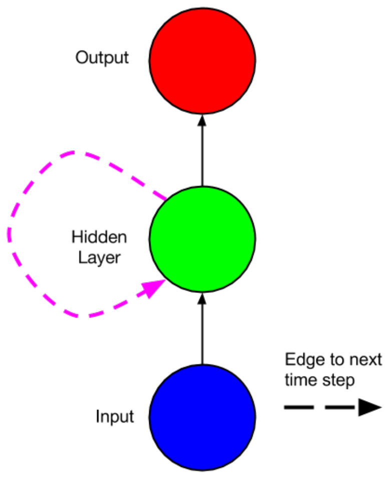


Figure 2.5: A simple recurrent net with one input unit, one output unit, and one recurrent hidden unit.

The optimization problem represents a more fundamental obstacle that cannot as easily be dealt with by modifying network architecture. It has been known since at least 1993 that optimizing even a 3-layer neural network is NP-Complete [BR93]. However, recent empirical and theoretical studies suggest that the problem may not be as hard in practice as once thought. [DPG$^+$14] shows that while many critical points exist on the error surfaces of large neural networks, the ratio of saddle points to true local minima increases exponentially with the size of the network

Fast implementations and improved gradient following heuristics have rendered RNN training feasible. For example, implementations of forward and backward prop-

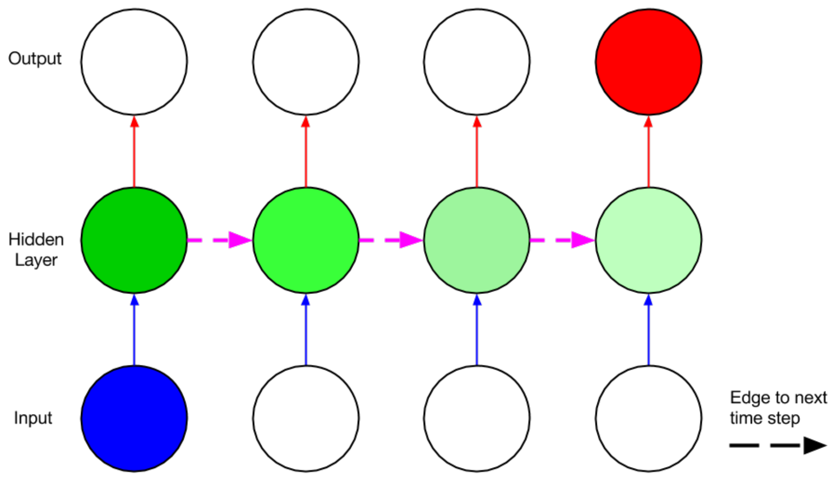


Figure 2.6: A visualization of the vanishing gradient problem. If the weight along the purple edge is less than one, the effect of the input at the first time step on the output at the final time step will rapidly diminish as a function of the size of the interval in between.

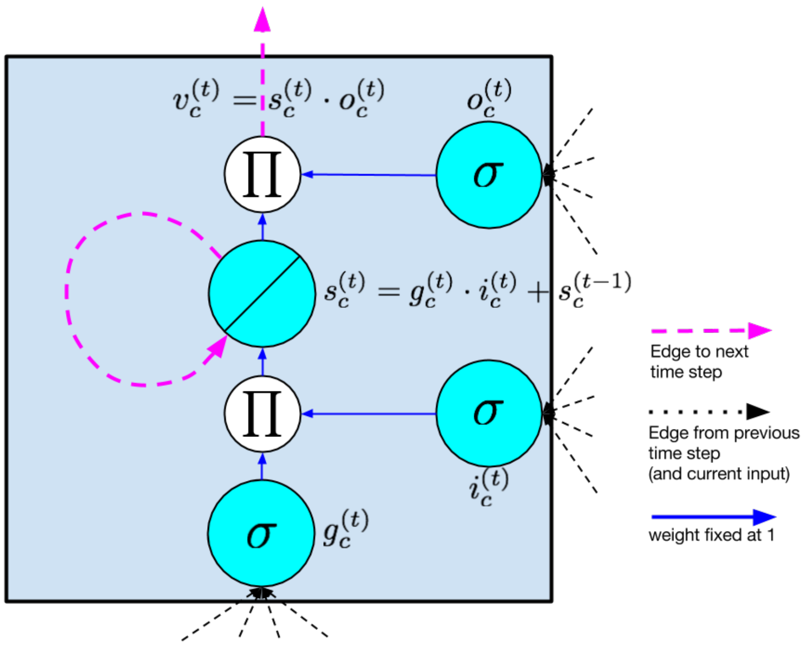
agation using GPUs, such as Theano ([BBB$^+$10]) and Torch ([CKF11]), have made it easy to implement fast training algorithms. In 1996, prior to the introduction of the LSTM, attempts to train recurrent nets to bridge long time gaps were shown to perform no better than random guessing [Hoc]. However, successfully trained RNNs are now relatively common. Sutskever and Martens reported success training recurrent neural networks with a Hessian-Free, i.e., truncated Newton approach [MS11] and applied it to a network which learns to generate text one character at a time in [SMH11]. In the paper that described the abundance of saddle points on the error surfaces of neural networks ([DPG$^+$14]), the authors present a saddle-free version of Newton's method. Unlike Newton's method, which is attracted to critical points, including saddle points, this variant is specially designed to escape from them. Experimental results include a demonstration of improved performance on recurrent networks. Newton's method requires computing the Hessian, which is prohibitively expensive for large networks, scaling quadratically with the number of parameters. While their algorithm only approximates the Hessian, it is still computationally expensive compared to SGD. Thus the authors describe a hybrid approach whereby the saddle-free Newton method is applied in places where SGD appears to be *stuck*.

2.3.2 Modern RNNs

The most successful RNN architectures for sequence learning date to two papers from 1997. The first, *Long Short-Term Memory*, by Hochreiter and Schmidhuber, introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of a network. With these memory cells, networks are able to overcome some difficulties with training encountered in earlier recurrent nets. The second, *Bidirectional Recurrent Neural Networks*, by Schuster and Paliwal, introduces the BRNN architecture in which information from both the future and the past are used to determine the output at any time t. This is in contrast to previous systems, in which only past input can affect the output, and has been used successfully for sequence labeling tasks in natural language processing, among others. Fortunately, the two innovations are not mutually exclusive, and have been successfully combined by Graves et al. for phoneme classification [GS05] and handwriting recognition [GLF$^+$09].

Long Short-Term Memory (LSTM)

In 1997, to overcome the problem of vanishing gradients, Hochreiter and Schmidhuber introduced the LSTM model. This model resembles a standard neural network with a recurrent hidden layer, only each ordinary node (Figure 2.1) in the hidden layer is replaced with a memory cell (Figure 2.7). The memory cell contains a node with a self-connected recurrent edge of weight 1, ensuring that the gradient can pass across many time steps without vanishing or exploding. To distinguish that we are referencing a memory cell and not an ordinary node, we use the index c.



$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
Π
σ
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(t)} + s_c^{(t-1)}$
Π
σ
$i_c^{(t)}$
σ
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

Figure 2.7: LSTM memory cell The self-connected node is the internal state s. The diagonal line indicates that no activation function is applied, Dashed lines indicate recurrent edges and pink edges have fixed weight of 1.

The term "Long Short-Term Memory" comes from the following intuition. Simpler recurrent neural networks have *long term memory* in the form of weights. The weights change very slowly over time encoding general knowledge about the data. They also have *short term memory* in the form of ephemeral activations, which pass from each node's output to successive nodes. The LSTM model introduces an intermediary sort of memory via the memory cell. A memory cell is a composite of simpler units with the novel addition of multiplicative nodes, represented in diagrams with Π. All elements of the LSTM cell are enumerated and described below.

- *Internal State:* At the heart of each memory cell is a node s with linear activation, which is referred to in the original paper as the "internal state" of the cell. We index cells with c and thus the internal state of a cell c is s_c.

- *Constant error carousel:* The internal state s_c has a self-connected (recurrent) edge with weight 1. This edge, called the *constant error carousel*, spans adjacent time steps with constant weight, assuring that error can flow across time steps without vanishing.

- *Input Node:* This node behaves as an ordinary neuron, taking input from the rest of the network (at the previous time step) as well as from the input. In the original paper and most subsequent work the input node is labeled g. We adhere to this convention but note that it may be confusing as g does not stand for *gate*. In the original paper, the gates are called y_{in} and y_{out} but this is confusing because y generally stands for output in the machine learning literature. Seeking comprehensibility, we break with this convention and use i, f, and o to refer to input, forget and output gates respectively as in [SVL14]. When we use vector notation we are referring to the values of the nodes in an entire layer of cells. For example, g is a vector containing the value of g at each memory cell in a layer. When the subscript c is used, it is to refer to an individual memory cell.

- *Multiplicative Gating:* Multiplicative gates are distinctive features of the LSTM model. Here a sigmoidal unit called a *gate* is learned given the input and the incoming recurrent connections from the previous time step. Some value of interest is then multiplied by this output. If the gate outputs 0, flow through the gate is cut off. If the gate outputs 1, all activation is passed through the gate.

  - *Input gate:* The original LSTM contains two gates. The first is an *input gate* i_c, which is multiplied by the *input* node g_c.

  - *Output gate:* The second gate is termed the *output gate*, which we notate as o_c. This gate is multiplied by the value of the internal state s_c to produce the value of v_c output by the memory cell . This then feeds into the LSTM hidden layer at the next time step $h^{(t+1)}$ as well as the output nodes $\hat{y}^{(t)}$ at the current time step.

Since the original LSTM was introduced, several variations have been proposed. *Forget gates*, proposed in 2000 by Gers and Schmidhuber [GSC00], add a gate similar to input and output gates to allow the network to flush information from the constant error carousel. Also in 2000, Gers and Schmidhuber proposed peephole connections [GS00], which pass from the carousel directly to the input and output gates of that same node without first having to pass through the output gate. They report that these connections improve performance on timing tasks where the network must learn to measure precise intervals between events. While peephole connections are not common in modern papers, forget gates have become a mainstay of LSTM work.

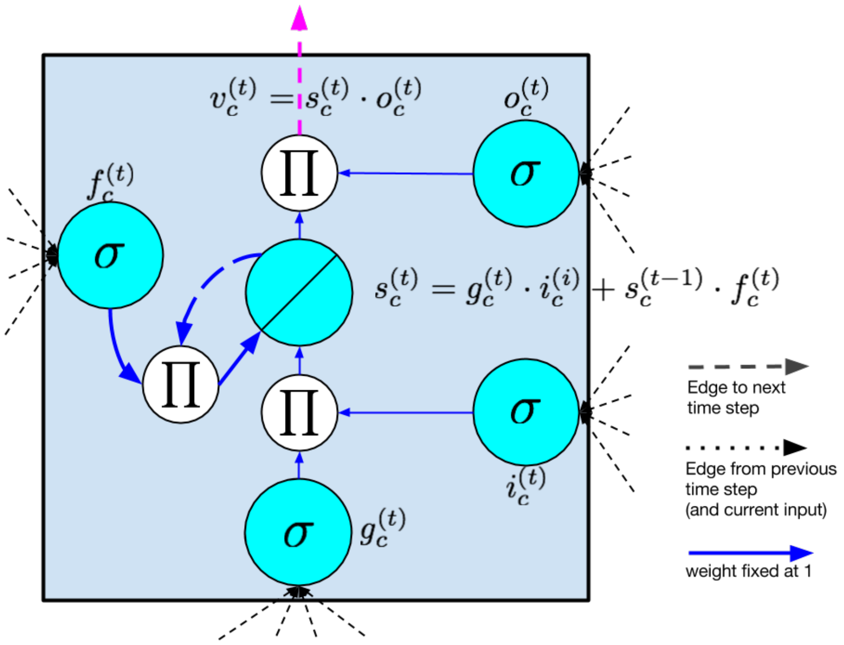

$v_c^{(t)} = s_c^{(t)} \cdot o_c^{(t)}$
$o_c^{(t)}$
$f_c^{(t)}$
$s_c^{(t)} = g_c^{(t)} \cdot i_c^{(i)} + s_c^{(t-1)} \cdot f_c^{(t)}$
$i_c^{(t)}$
$g_c^{(t)}$
Edge to next time step
Edge from previous time step (and current input)
weight fixed at 1

Figure 2.8: LSTM memory cell with forget gate as described by Felix Gers et al. in [GSC00].

Put formally, computation in the LSTM model proceeds according to the following calculations which must be evaluated at each time step. This gives the full algorithm for a modern LSTM with forget gates.

$$g^{(t)} * = * \phi(W_{gx}x^{(t)} + W_{ih}h^{(t-1)} + b_g) \tag{2.4}$$

$$i^{(t)} = \sigma(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i) \tag{2.5}$$

$$f^{(t)} = \sigma(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f) \tag{2.6}$$

$$o^{(t)} = \sigma(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_o) \tag{2.7}$$

$$s^{(t)} = g^{(t)} \odot i^{(i)} + s^{(t-1)} \odot f^{(t)} \tag{2.8}$$

$$h^{(t)} = s^{(t)} \odot o^{(t)} \tag{2.9}$$

where $\odot$ stands for element-wise multiplication. The calculations for the simpler LSTM without forget gates is given by setting $f^{(t)} = 1$ for all t. We use the tanh function ϕ for the input node g following the latest state of the art setup of Zaremba and Sutskever in [ZS14]. However, in the original LSTM paper [HS97], the activation function for g is the sigmoid σ. Again, $h^{(t-1)}$ is a vector containing the values v_c output by each memory cell c in the hidden layer at the previous time step.

Intuitively, in terms of the forward pass, the LSTM can learn when to let activation into the internal state. So long as the input gate takes value 0, no activation can get in. Similarly, the output gate learns when to let the value out. When both gates are *closed*, the activation is trapped in the LSTM, neither growing nor shrinking, nor affecting the output at the intermediary time steps. In terms of the backwards pass, the constant error carousel enables the gradient to propagate back across many time steps, neither exploding nor vanishing. In this sense, the gates are learning when to let *error* in, and when to let it out. In practice, the LSTM has shown a superior ability to learn long-range dependencies as compared to simple RNNs.

2.3.3 Bidirectional Recurrent Neural Networks (BRNNs)

Along with the LSTM, one of the most used RNN setups is the bidirectional recurrent neural network (BRNN) (Figure 2.9) first described in [SP97]. In this architecture, there are two layers of hidden nodes. Both hidden layers are connected to input and output. The two hidden layers are differentiated in that the first has recurrent connections from the past time steps while in the second the direction of recurrent of connections is flipped, passing activation backwards in time. Given a fixed length sequence, the BRNN can be learned with ordinary backpropagation. The following three equations describe a BRNN:

$$h_f^{(t)} = \sigma(W_{h_f x}x + W_{h_f h_f}h_f^{(t-1)} + b_{h_f}) \tag{2.10}$$

$$h_b^{(t)} = \sigma(W_{h_b x}x + W_{h_b h_b}{h_b}^{(t+1)} + b_{h_b}) \tag{2.11}$$

$$\hat{y}^{(t)} = softmax(W_{yh_f}{h_f}^{(t)} + W_{yh_b}{h_b}^{(t)} + b_y) \tag{2.12}$$

Where $h_f^{(t)}$ and $h_b^{(t)}$ correspond to the hidden layers in the forwards and backwards directions respectively.

One limitation of the BRNN is that cannot run continuously, as it requires a fixed endpoint in both the future and in the past. Further, it is not an appropriate machine learning algorithm for the online setting, as it is implausible to receive information from the future, i.e., sequence elements that have not been observed. But for sequence prediction over a sequence of fixed length, it is often sensible to account for both past and future data. Consider the natural language task of *part of speech tagging*. Given a word in a sentence, information about both the words which precede and those which succeed it is useful for predicting that word's part of speech. Karpathy et al. use such a network for generating captions for images [KFF14].

The LSTM and BRNN are in fact compatible ideas. The former introduces a new

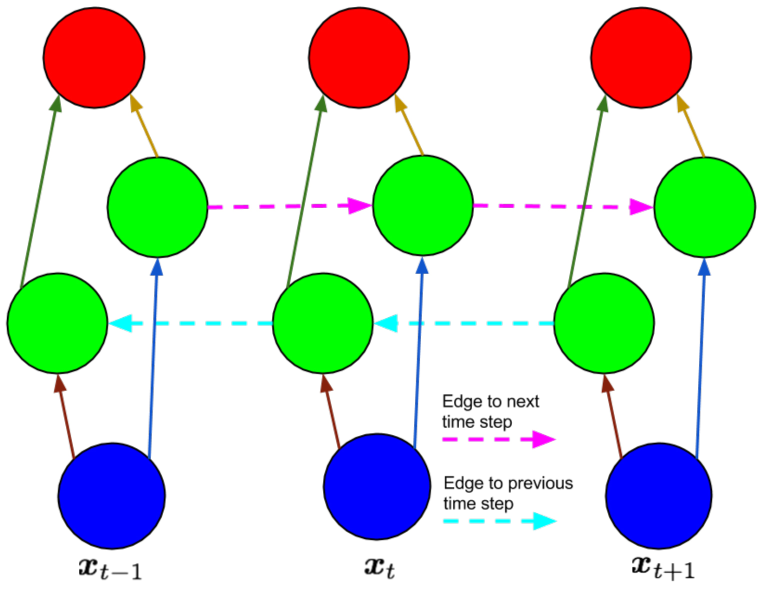


Figure 2.9: Structure of a bidirectional recurrent neural network as described by Schuster and Paliwal in [SP97].

basic unit from which to compose a hidden layer, while the latter concerns the wiring of the hidden layers, regardless of what nodes they contain. Such an approach, termed a BLSTM was used by Graves et al. to achieve state of the art results on handwriting recognition and phoneme classification [GLF$^+$09] [GS05].

2.4 Acknowledgments

Chapter 2, consisting of background on recurrent neural networks was written in collaboration with John Berkowitz and Charles Elkan. This review also benefited from insightful comments from Julian MacAuley, Balakrishnan Narayanaswamy, Stefanos Poulis, and Sharad Vikram. The dissertation author was the primary investigator and author of this paper.

Part I

Mining Temporal Patterns in Medical Data

Chapter 3

Learning to Diagnose with LSTM RNNs

Clinical medical data, especially in the intensive care unit (ICU), consist of multivariate time series of observations. For each patient visit (or *episode*), sensor data and lab test results are recorded in the patient's Electronic Health Record (EHR). While potentially containing a wealth of insights, the data is difficult to mine effectively, owing to varying length, irregular sampling and missing data. Recurrent Neural Networks (RNNs), particularly those using Long Short-Term Memory (LSTM) hidden units, are powerful and increasingly popular models for learning from sequence data. They effectively model varying length sequences and capture long range dependencies. This chapter presents the first study to empirically evaluate the ability of LSTMs to recognize patterns in multivariate time series of clinical measurements.

3.1 Introduction

Time series data comprised of clinical measurements, as recorded by caregivers in the pediatric intensive care unit (PICU), constitute an abundant and largely untapped source of medical insights. Potential uses of such data include classifying diagnoses accurately, predicting length of stay, predicting future illness, and predicting mortality. However, besides the difficulty of acquiring data, several obstacles stymie machine learning research with clinical time series. Episodes vary in length, with stays ranging from just a few hours to multiple months. Observations, which include sensor data, vital signs, lab test results, and subjective assessments, are sampled irregularly and plagued by missing values [MKKW12]. Additionally, long-term time dependencies complicate learning with many algorithms. Lab results that, taken together, might imply a particular diagnosis may be separated by days or weeks. Long delays often separate onset of disease from the appearance of symptoms. For example, symptoms of acute respiratory distress syndrome may not appear until 24-48 hours after lung injury [MBM$^+$10], while symptoms of an asthma attack may present shortly after admission but change or disappear following treatment.

Recurrent Neural Networks (RNNs), in particular those based on Long Short-Term Memory (LSTM) [HS97], model varying-length sequential data, achieving state-of-the-art results for problems spanning natural language processing, image captioning, handwriting recognition, and genomic analysis [AGQZ13, SVL14, VTBE14, KFF14, LGBS07, GLF$^+$09, PPRB02, Voh01, XWIF07]. LSTMs can capture long range dependencies and nonlinear dynamics. Some sequence models, such as Markov models, conditional random fields, and Kalman filters, deal with sequential data but are ill-equipped to learn long-range dependencies. Other models require domain knowledge or feature engineering, offering less chance for serendipitous discovery. In contrast, neural networks learn representations and can discover unforeseen structure.

This chapter presents the first empirical study using LSTMs to classify diagnoses given multivariate PICU time series. Specifically, we formulate the problem as multilabel classification, since diagnoses are not mutually exclusive. Our examples are clinical episodes, each consisting of 13 frequently but irregularly sampled time series of clinical measurements, including body temperature, heart rate, diastolic and systolic blood pressure, and blood glucose, among others. Associated with each patient are a subset of 429 diagnosis codes. As some are rare, we focus on the 128 most common codes, classifying each episode with one or more diagnoses.

Because LSTMs have never been previously used in this setting, we first verify their utility and compare their performance to a set of strong baselines, including both a linear classifier and a MultiLayer Perceptron (MLP). We train the baselines on both a fixed window and hand-engineered features. We then test a straightforward *target replication* strategy for recurrent neural networks, inspired by the *deep supervision* technique of [LXG$^+$14] for training convolutional neural networks. We compose our optimization objective as a convex combination of the loss at the final sequence step and the mean of the losses over *all* sequence steps. Additionally, we evaluate the efficacy of using additional information in the patient's chart as auxiliary outputs, a technique previously used with feedforward nets [CBM$^+$96], showing that it reduces overfitting. Finally, we apply dropout to non-recurrent connections, which improves the performance further. LSTMs with target replication and dropout surpass the performance of the best baseline, namely an MLP trained on hand-engineered features, even though the LSTM has access only to raw time series.

3.2 Data Description

Our experiments use a collection of anonymized clinical time series extracted from the EHR system at Children's Hospital LA [MKKW12, CKL+15] as part of an IRB-approved study. The data consists of $10,401$ PICU episodes, each a multivariate time series of 13 variables: diastolic and systolic blood pressure, peripheral capillary refill rate, end-tidal CO_2, fraction of inspired O_2, Glascow coma scale, blood glucose, heart rate, pH, respiratory rate, blood oxygen saturation, body temperature, and urine output. Episodes vary in length from 12 hours to several months.

Each example consists of irregularly sampled multivariate time series with both missing values and, occasionally, missing variables. We resample all time series to an hourly rate, taking the mean measurement within each one hour window. We use forward- and back-filling to fill gaps created by the window-based resampling. When a single variable's time series is missing entirely, we impute a clinically normal value as defined by domain experts. These procedures make reasonable assumptions about clinical practice: many variables are recorded at rates proportional to how quickly they change, and when a variable is absent, it is often because clinicians believed it to be normal and chose not to measure it. Nonetheless, these procedures are not appropriate in all settings. Back-filling, for example, passes information from the future backwards. This is acceptable for classifying entire episodes (as we do) but not for forecasting. Finally, we rescale all variables to $[0,1]$, using ranges defined by clinical experts. In addition, we use published tables of normal values from large population studies to correct for differences in heart rate, respiratory rate [FTS+11], and blood pressure [Nat04] due to age and gender.

Each episode is associated with zero or more diagnostic codes from an in-house taxonomy used for research and billing, similar to the *Ninth Revision of the International*

Classification of Diseases (ICD-9) codes [Wor04]. The dataset contains 429 distinct labels indicating a variety of conditions, such as acute respiratory distress, congestive heart failure, seizures, renal failure, and sepsis. Because many of the diagnoses are rare, we focus on the most common 128, each of which occurs more than 50 times in the data. These diagnostic codes are recorded by attending physicians during or shortly after each patient episode and subject to limited review afterwards.

Because the diagnostic codes were assigned by clinicians, our experiments represent a comparison of an LSTM-based diagnostic system to human experts. We note that an attending physician has access to much more data about each patient than our LSTM does, including additional tests, medications, and treatments. Additionally, the physician can access a full medical history including free-text notes, can make visual and physical inspections of the patient, and can ask questions. A more fair comparison might require asking additional clinical experts to assign diagnoses given access only to the 13 time series available to our models. However, this would be prohibitively expensive, even for just the 1000 examples, and difficult to justify to our medical collaborators, as this annotation would provide no immediate benefit to patients. Such a study will prove more feasible in the future when this line of research has matured.

3.3 Methods

In this work, we are interested in recognizing diagnoses and, more broadly, the observable physiologic characteristics of patients, a task generally termed *phenotyping* [OCG+15]. We cast the problem of phenotyping clinical time series as multilabel classification. Given a series of observations $x^{(1)},...,x^{(T)}$, we learn a classifier to generate hypotheses $\hat{y}$ of the true labels y. Here, t indexes sequence steps, and for any example, T stands for the length of the sequence. Our proposed LSTM RNN uses memory cells with forget gates [GSC00] but without peephole connections [GSS03]. As output, we use a fully connected layer atop the highest LSTM layer followed by an element-wise sigmoid activation function, because our problem is multilabel. We use *log loss* as the loss function at each output.

3.3.1 LSTM Architectures for Multilabel classification

We explore several recurrent neural network architectures for multilabel classification of time series. The first and simplest (Figure 3.1) passes over all inputs in chronological order, generating outputs only at the final sequence step. In this approach, we only have output $\hat{y}$ at the final sequence step, at which our loss function is the average of the losses at each output node. Thus the loss calculated at a single sequence step is the average of *log loss* calculated separately on each label.

$$\text{loss}(\hat{y}, y) = \frac{1}{|L|} \sum_{l=1}^{l=|L|} -(y_l \cdot \log(\hat{y}_l) + (1 - y_l) \cdot \log(1 - \hat{y}_l)). \tag{3.1}$$

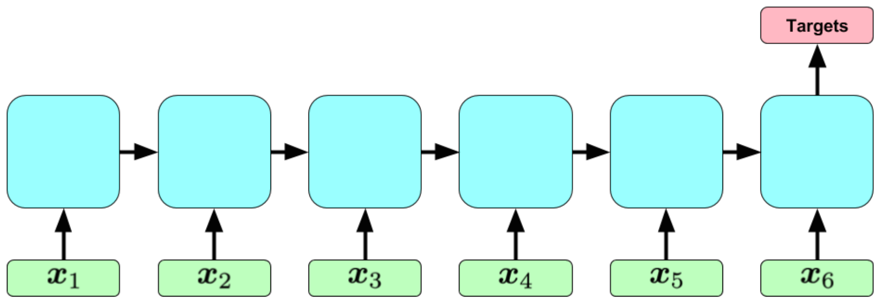


Figure 3.1: A simple RNN model for multilabel classification. Green rectangles represent inputs. The recurrent hidden layers separating input and output are represented with a single blue rectangle. The red rectangle represents targets.

3.3.2 Sequential Target Replication

One problem with the simple approach is that the network must learn to pass information across many sequence steps in order to affect the output. We attack this problem by replicating our static targets at each sequence step (Figure 3.2), providing a local error signal at each step. This approach is inspired by the deep supervision technique that [LXG+14] apply to convolutional nets. This technique is especially sensible in our case because we expect the model to predict accurately even if the sequence were truncated by a small amount. The approach differs from [LXG+14] because we use the same output weights to calculate $\hat{y}^{(t)}$ for all t. Further, we use this target replication to generate output at each sequence step, but not at each hidden layer.

For the model with target replication, we generate an output $\hat{y}^{(t)}$ at every sequence step. Our loss is then a convex combination of the final loss and the average of the losses over all steps:

$$\alpha \cdot \frac{1}{T} \sum_{t=1}^{T} \text{loss}(\hat{y}^{(t)}, y^{(t)}) + (1-\alpha) \cdot \text{loss}(\hat{y}^{(T)}, y^{(T)}) \tag{3.2}$$

where T is the total number of sequence steps and $\alpha \in [0,1]$ is a hyper-parameter which determines the relative importance of hitting these intermediary targets. At prediction time, we take only the output at the final step. In our experiments, networks using target replication outperform those with a loss applied only at the final sequence step.

3.3.3 Auxiliary Output Training

Recall that our initial data contained 429 diagnostic labels but that our task is to predict only 128. Given the well-documented successes of multitask learning with shared representations and feedforward networks, we wish to train a stronger model by using the remaining 301 labels or other information in the patient's chart, such as diagnostic categories, as auxiliary targets [CBM+96]. These additional targets serve

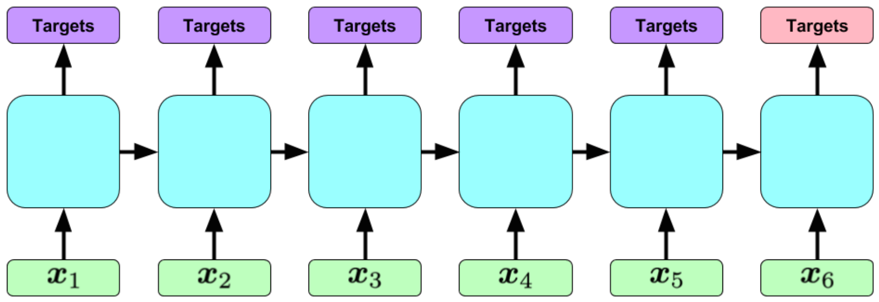


Figure 3.2: An RNN classification model with *target replication*. The primary target (depicted in red) at the final step is used at prediction time, but during training, the model back-propagates errors from the intermediate targets (purple) at every sequence step.

reduce overfitting as the model aims to minimize the loss on the labels of interest while also minimizing loss on the auxiliary targets (Figure 3.3).

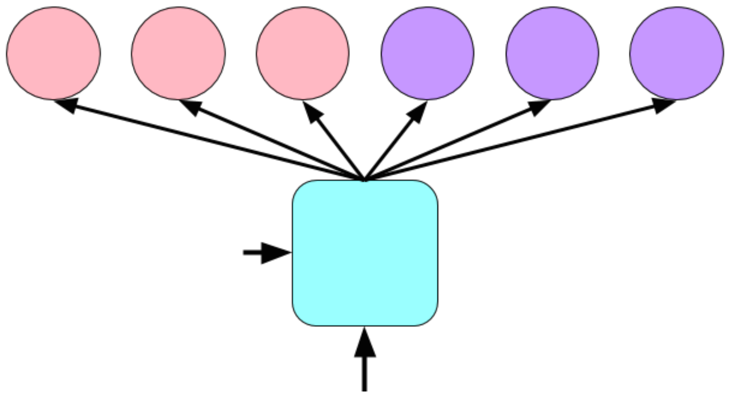


Figure 3.3: Our dataset contains many labels. For our task, a subset of 128 are of interest (depicted in red). Our *Auiliary Output* neural network makes use of extra labels as additional training targets (depicted in purple). At inference time we generate predictions for only the labels of interest.

3.3.4 Regularization

Because we have only $10,401$ examples, overfitting is a considerable obstacle. Our experiments show that both target replication and auxiliary outputs improve performance and reduce overfitting. In addition to these less common techniques we deploy

ℓ_2^2 weight decay and dropout. Following the example of [ZSV14] and [PBKL14], we apply dropout to the non-recurrent connections only. We first compute each hidden layer's sequence of activations in the left-to-right direction and then apply dropout before computing the next layer's activations. In our experiments, we find that dropout decreases overfitting, enabling us to double the size of each hidden layer.

3.4 Experiments

All models are trained on 80% of the data and tested on 10%. The remaining 10% is used as a validation set. We train each LSTM for 100 epochs using stochastic gradient descent (SGD) with momentum. To combat exploding gradients, we scale the norm of the gradient and use ℓ_2^2 weight decay of 10^{-6}, both hyper-parameters chosen using validation data. Our final networks use 2 hidden layers and either 64 memory cells per layer with no dropout or 128 cells per layer with dropout of 0.5. These architectures are also chosen based on validation performance. Throughout training, we save the model and compute three performance metrics (micro AUC, micro F1, and precision at 10) on the validation set for each epoch. We then test the model that scores best on at least two of the three validation metrics. To break ties, we choose the earlier epoch.

We evaluate a number of baselines as well as LSTMs with various combinations of target replication (TR), dropout (DO), and auxiliary outputs (AO), using either the additional 301 diagnostic labels or 12 diagnostic categories. To explore the regularization effects of each strategy, we record and plot both training and validation performance after each epoch. Additionally, we report performance of a target replication model (Linear Gain) that scales the weight of each intermediate target linearly as opposed our proposed approach. Finally, to show that our LSTM learns a model complementary to the baselines, we evaluate an ensemble of the best LSTM with the best baseline.

3.4.1 Multilabel Evaluation Methodology

We report micro- and macro-averaged versions of Area Under the ROC Curve (AUC). By micro AUC, we mean a single AUC computed on flattened $\hat{Y}$ and Y matrices, whereas we calculate macro AUC by averaging each per-label AUC. The blind classifier achieves 0.5 macro AUC but can exceed 0.5 on micro AUC by predicting labels in descending order by base rate. Additionally, we report micro- and macro-averaged F1 score, computed in similar fashion to the respective micro and macro AUCs. F1 metrics require a thresholding strategy, and here we select thresholds based upon validation set performance. We refer to [LEN14] for an analysis of the strengths and weaknesses of each type of multilabel F-score and a characterization of optimal thresholds.

Finally, we report *precision at* 10, which captures the fraction of true diagnoses among the model's top 10 predictions, with a best possible score of 0.2281 on the test split of this data set because there are on average 2.281 diagnoses per patient. While F1 and AUC are both useful for determining the relative quality of a classifier's predictions, neither is tailored to a real-world application. Thus, we consider a medically plausible use case to motivate this more interpretable metric: generating a short list of the 10 most probable diagnoses. If we could create a high recall, moderate precision list of 10 likely diagnoses, it could be a valuable hint-generation tool for differential diagnosis. Testing for only the 10 most probable conditions is much more realistic than testing for all conditions.

3.4.2 Baseline Classifiers

We provide results for a *base rate* model that predicts diagnoses in descending order by incidence to provide a minimum performance baseline for micro-averaged metrics. We also report the performance of logistic regression, which is widely used in clinical research. We train a separate classifier for each diagnosis but choose an overall ℓ_2^2 penalty for all individual classifiers based on validation performance. For a much stronger baseline, we train a multilabel MLP with 3 hidden layers of 300 hidden units each, rectified linear activations, and dropout of 0.5. All MLPs were trained for 1000 epochs, with hyperparameters chosen based on validation set performance. Each baseline is tested with two sets of inputs: raw time series and hand-engineered features. For raw time series, we use the first and last six hours. This provides classifiers with temporal information about changes in patient state from admission to discharge within a fixed-size input, as required by all baselines. We find this works better than providing the first or last 12 hours alone.

Our hand-engineered features are inspired by those used in state-of-the-art severity of illness scores [PPR96]: for each variable, we compute the first and last measurements and their difference scaled by episode length, mean and standard deviation, median and quartiles, minimum and maximum, and slope of a line fit with least squares. These 143 features capture many of the indicators that clinicians look for in critical illness, including admission and discharge state, extremes, central tendencies, variability, and trends. They previously have been shown to be effective for these data [MKKW12, CKL$^+$15]. Our strongest baseline is an MLP using these features.

3.4.3 Results

Our best performing LSTM (LSTM-DO-TR) used two layers of 128 memory cells, dropout of probability 0.5 between layers, and target replication, and outperformed the MLP with hand-engineered features. Moreover simple ensembles of the best LSTM and MLP outperformed both on all metrics. Table 3.1 shows summary results for all models. Table 3.2 shows the LSTM's predictive performance for six diagnoses with the highest F1 scores. Full per-diagnosis results can be found in Section 3.7.

Target replication improves performance on all metrics, accelerating learning and reducing overfitting (Figure 3.6). We also find that the LSTM with target replication learns to output correct diagnoses earlier in the time series, a virtue that we explore qualitatively in Section 3.5. As a comparison, we trained a LSTM-DO-TR variant using the linear gain strategy of [NHV$^+$15, DL15]. In general, this model did not perform as well as our simpler target replication strategy, but it did achieve the highest macro F1 score among the LSTM models.

Table 3.1: Results on performance metrics calculated across all labels. *DO*, *TR*, and *AO* indicate dropout, target replication, and *auxiliary outputs*, respectively. *AO (Diagnoses)* uses the extra diagnosis codes and *AO (Categories)* uses diagnostic categories as additional targets during training.

Classification performance for 128 ICU phenotypes

Model	**Micro AUC**	**Macro AUC**	**Micro F1**	**Macro F1**	**Prec. at 10**
Base Rate	0.7128	0.5	0.1346	0.0343	0.0788
Log. Reg., First 6 + Last 6	0.8122	0.7404	0.2324	0.1081	0.1016
Log. Reg., Expert features	0.8285	0.7644	0.2502	0.1373	0.1087
MLP, First 6 + Last 6	0.8375	0.7770	0.2698	0.1286	0.1096
MLP, Expert features	**0.8551**	**0.8030**	**0.2930**	**0.1475**	**0.1170**
LSTM Models with two 64-cell hidden layers					
LSTM	0.8241	0.7573	0.2450	0.1170	0.1047
LSTM, AuxOut (Diagnoses)	0.8351	0.7746	0.2627	0.1309	0.1110
LSTM-AO (Categories)	0.8382	0.7748	0.2651	0.1351	0.1099
LSTM-TR	0.8429	0.7870	0.2702	0.1348	0.1115
LSTM-TR-AO (Diagnoses)	0.8391	0.7866	0.2599	0.1317	0.1085
LSTM-TR-AO (Categories)	0.8439	0.7860	0.2774	0.1330	0.1138
LSTM Models with Dropout (probability 0.5) and two 128-cell hidden layers					
LSTM-DO	0.8377	0.7741	0.2748	0.1371	0.1110
LSTM-DO-AO (Diagnoses)	0.8365	0.7785	0.2581	0.1366	0.1104
LSTM-DO-AO (Categories)	0.8399	0.7783	0.2804	0.1361	0.1123
LSTM-DO-TR	**0.8560**	**0.8075**	**0.2938**	0.1485	0.**1172**
LSTM-DO-TR-AO (Diagnoses)	0.8470	0.7929	0.2735	0.1488	0.1149
LSTM-DO-TR-AO (Categories)	0.8543	0.8015	0.2887	0.1446	0.1161
LSTM-DO-TR (Linear Gain)	0.8480	0.7986	0.2896	**0.1530**	0.1160
Ensembles of Best MLP and Best LSTM					
Mean of LSTM-DO-TR & MLP	0.8611	0.8143	0.2981	0.1553	0.1201
Max of LSTM-DO-TR & MLP	**0.8643**	**0.8194**	**0.3035**	**0.1571**	**0.1218**

Auxiliary outputs improved performance for most metrics and reduced overfitting.

While the performance improvement is not as dramatic as that conferred by target replication, the regularizing effect is greater. These gains came at the cost of slower training: the auxiliary output models required more epochs (Figure 3.6 and Section 3.6), especially when using the 301 remaining diagnoses. This may be due in part to severe class imbalance in the extra labels. For many of these labels it may take an entire epoch just to learn that they are occasionally nonzero.

Table 3.2: LSTM-DO-TR performance on the 6 diagnoses with highest F1 scores.

Top 6 **diagnoses measured by F1 score**

Label	***F1***	**AUC**	**Precision**	**Recall**
Diabetes mellitus with ketoacidosis	0.8571	0.9966	1.0000	0.7500
Scoliosis, idiopathic	0.6809	0.8543	0.6957	0.6667
Asthma, unspecified with status asthmaticus	0.5641	0.9232	0.7857	0.4400
Neoplasm, brain, unspecified	0.5430	0.8522	0.4317	0.7315
Delayed milestones	0.4751	0.8178	0.4057	0.5733
Acute Respiratory Distress Syndrome (ARDS)	0.4688	0.9595	0.3409	0.7500

The LSTMs appear to learn models complementary to the MLP trained on hand-engineered features. Supporting this claim, simple ensembles of the LSTM-DO-TR and MLP (taking the *mean* or *maximum* of their predictions) outperform the constituent models significantly on all metrics (Table 3.1). Further, there are many diseases for which one model substantially outperforms the other, e.g., intracranial hypertension for the LSTM, septic shock for the MLP (Section 3.7).

3.5 Hourly Diagnostic Predictions

Our LSTM networks predict 128 diagnoses given sequences of clinical measurements. Because each network is connected left-to-right, i.e., in chronological order, we can output predictions at each sequence step. Ultimately, we imagine that this capability could be used to make continuously updated real-time alerts and diagnoses. Below, we explore this capability qualitatively. We choose examples of patients with a correctly classified diagnosis and visualize the probabilities assigned by each LSTM model at each sequence step. In addition to improving the quality of the final output, the LSTMs with target replication (LSTM-TR) arrive at correct diagnoses quickly compared to the simple multilabel LSTM model (LSTM-Simple). When auxiliary outputs are also used (LSTM-TR,AO), the diagnoses appear to be generally more confident.

Our LSTM-TR,AO effectively predicts status asthmaticus and acute respiratory distress syndrome, likely owing to the several measures of pulmonary function among our inputs. Diabetic ketoacidosis also proved easy to diagnose, likely because glucose and pH are included among our clinical measurements. We were surprised to see that the network classified scoliosis reliably, but a deeper look into the medical literature suggests that scoliosis often results in respiratory symptoms. This analysis of step-by-step predictions is preliminary and informal, and we note that for a small number of examples our data preprocessing introduces a target leak by back-filling missing values. In future work, when we explore this capability in greater depth, we will reprocess the data.

3.6 Learning Curves

We present visualizations of the performance of LSTM, LSTM-DO (with dropout probability 0.5), LSTM-AO (using the 301 additional diagnoses), and LSTM-TR (with $\alpha = 0.5$), during training. These charts are useful for examining the effects of dropout, auxiliary outputs, and target replication on both the speed of learning and the regularization they confer. Specifically, for each of the four models, we plot the training and validation micro AUC and F1 score every five epochs in Figure 3.5. Additionally, we plot a scatter of the performance on the training set vs. the performance on the validation set. The LSTM with target replication learns more quickly than a simple LSTM and also suffers less overfitting. With both dropout and auxiliary outputs, the LSTM trains more slowly than a simple LSTM but suffers considerably less overfitting.

3.7 Per Diagnosis Results

While averaged statistics provide an efficient way to check the relative quality of various models, considerable information is lost by reducing performance to a single scalar quantity. For some labels, our classifier makes classifications with surprisingly high accuracy while for others, our features are uninformative and thus the classifier would not be practically useful. To facilitate a more granular investigation of our model's predictive power, we present individual test set F1 and AUC scores for each individual diagnostic label in Table 3.3. We compare the performance our best LSTM, which combines two 128-cell hidden layers with *dropout* of probability 0.5 and *target replication*, against the strongest baseline, an MLP trained on the hand-engineered features, and an ensemble predicts the maximum probability of the two. The results are sorted in descending order using the F1 performance of the LSTM, providing insights into the types of conditions that the LSTM can successfully classify.

3.8 Related Work

The research described in this chapter sits at the intersection of LSTMs, medical informatics, and multilabel classification, three mature fields, each with a long history and rich body of research. While we cannot do justice to all three, we highlight the most relevant works below.

3.8.1 Neural Networks for Medical Data

Neural networks have been applied to medical problems and data for at least 20 years [CBM+96, Bax95], although we know of no work on applying LSTMs to multivariate clinical time series of the type we analyze here. Several papers have applied RNNs to physiologic signals, including electrocardiograms [SM98, AC98, Übe09] and glucose measurements [TB98]. RNNs have also been used for prediction problems in genomics [PPRB02, XWIF07, Voh01]. Multiple recent papers apply modern deep learning techniques (but not RNNs) to modeling psychological conditions [DC15], head injuries [RDL+10], and Parkinson's disease [HFA+15]. Recently, feedforward networks have been applied to medical time series in sliding window fashion to classify cases of gout, leukemia [LDL13], and critical illness [CKL+15].

3.8.2 Neural Networks for Multilabel Classification

Only a few published papers apply LSTMs to multilabel classification tasks, all of which, to our knowledge, are outside of the medical context. [LR+14] formulates music composition as a multilabel classification task, using sigmoidal output units. Most recently, [YRJ+15] uses LSTM networks with multilabel outputs to recognize actions in videos. While we could not locate any published papers using LSTMs for multilabel classification in the medical domain, several papers use feedforward nets for this task. One of the earliest papers to investigate multi-task neural networks modeled risk in pneumonia patients [CBM+96]. More recently, [CKL+15] formulated diagnosis as multilabel classification using a sliding window multilayer perceptron.

3.8.3 Machine Learning for Clinical Time Series

Neural network methodology aside, a growing body of research applies machine learning to temporal clinical data for tasks including artifact removal [ARM$^+$09, QWM$^+$09], early detection and prediction [SWF$^+$14a, HHPS15], and clustering and subtyping [MKKW12, SWS15]. Many recent papers use models with latent factors to capture nonlinear dynamics in clinical time series and to discover meaningful representations of health and illness. Gaussian processes are popular because they can directly handle irregular sampling and encode prior knowledge via choice of covariance functions between time steps and across variables [MKKW12, GPN$^+$15]. [SKP10] combined a hierarchical dirichlet process with autoregressive models to infer latent disease "topics" in the heart rate signals of premature babies. [QWM$^+$09] used linear dynamical systems with latent switching variables to model physiologic events like bradycardias. Seeking *deeper* models, [SWF14b] proposed a second "layer" of latent factors to capture correlations between latent states.

3.8.4 Target Replication

In this work, we make the task of classifying entire sequences easier by replicating targets at every time step, inspired by [LXG$^+$14], who place an optimization objective after each layer in convolutional neural network. While they have a separate set of weights to learn each intermediate objective, our model is simpler owing to the weight tying in recurrent nets, having only one set of output weights. Additionally, unlike [LXG$^+$14], we place targets at each time step, but not following each layer between input and output in the LSTM. After finishing this manuscript, we learned that target replication strategies similar to ours have also been developed by [NHV$^+$15] and [DL15] for the tasks of video classification and character-level document classification respectively. [NHV$^+$15] linearly scale the importance of each intermediate target, emphasizing performance at later sequence steps over those in the beginning of the clip. [DL15] also use a target replication strategy with linearly increasing weight for character-level document classification, showing significant improvements in accuracy. They call this technique *linear gain*.

3.8.5 Regularizing Recurrent Neural Networks

Given the complexity of our models and modest scale of our data, regularization, including judicious use of dropout, is crucial to our performance. Several prior works use dropout to regularize RNNs. [PBKL14], [ZSV14], and [DL15] all describe an application of dropout to only the non-recurrent weights of a network. The former two papers establish the method and apply it to tasks with sequential outputs, including handwriting recognition, image captioning, and machine translation. The setting studied by [DL15] most closely resembles ours as the authors apply it to the task of applying static labels to varying length sequences.

3.8.6 Key Differences

Our experiments show that LSTMs can accurately classify multivariate time series of clinical measurements, a topic not addressed in any prior work. Additionally, while some papers use LSTMs for multilabel classification, this work is the first to address this problem in the medical context. Moreover, for multilabel classification of sequential clinical data with fixed length output vectors, this chapter describes the first work, to our knowledge, to demonstrate the efficacy of a target replication strategy, achieving both faster training and better generalization.

3.9 Discussion

Our results indicate that LSTM RNNs, especially with target replication, can successfully classify diagnoses of critical care patients given clinical time series data. The best LSTM beat a strong MLP baseline using hand-engineered features as input, and an ensemble combining the MLP and LSTM improves upon both. The success of target replication accords with results by both [NHV$^+$15] and [DL15], who observed similar benefits on their respective tasks. However, while they saw improvement using a linearly increasing weight on each target from start to end, this strategy performed worse in our diagnostic classification task than our uniform weighting of intermediate targets. We believe this may owe to the peculiar nature of our data. Linear gain emphasizes evidence from later in the sequence, an assumption which often does not match the progression of symptoms in critical illnesses. Asthma patients, for example, are often admitted to the ICU severely symptomatic, but once treatment begins, patient physiology stabilizes and observable signs of disease may abate or change. Further supporting this idea, we observed that when training fixed-window baselines, using the first 6 and last 6 hours outperformed using the last 12 hours only.

While our data is of large scale by clinical standards, it is small relative to datasets found in deep learning tasks like vision and speech recognition. At this scale, regularization is critical. Our experiments demonstrate that target replication, auxiliary outputs, and dropout all work to reduce the generalization gap. as shown in Figure 3.6 and Section 3.6. However, some of these techniques are complementary while others seem to cancel each other out. For example, our best model combined target replication with dropout. This combination significantly improved upon the performance using target replication alone, and enabled the effective use of larger capacity models. In contrast, the benefits of dropout and auxiliary output training appear to wash each other out. This may be because target replication confers more than regularization, mitigating the difficulty of learning long range dependencies by providing local objectives.

3.10 Conclusion

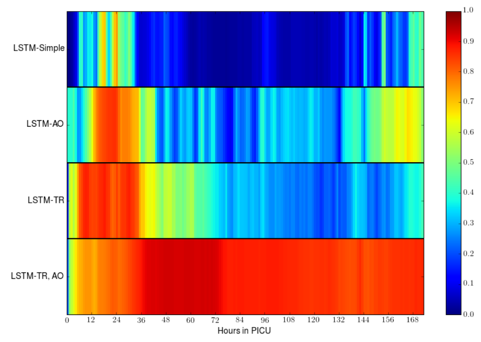
While our LSTMs produce promising results, this is only a first step in this line of research. Recognizing diagnoses given full time series of sensor data demonstrates that LSTMs can capture meaningful signal, but ultimately we would like to predict developing conditions and events, outcomes such as mortality, and treatment responses. In this chapter we used diagnostic labels without timestamps, but we are obtaining timestamped diagnoses, which will enable us to train models to perform early diagnosis by predicting future conditions. In addition, we are extending this work to a larger PICU data set with 50% more patients and hundreds of variables, including treatments and medications.

On the methodological side, we would like to both better exploit and improve the capabilities of LSTMs. Results from speech recognition have shown that LSTMs shine in comparison to other models using raw features, minimizing need for preprocessing and feature engineering. In contrast, our current data preparation pipeline removes valuable structure and information from clinical time series that could be exploited by an LSTM. For example, our forward- and back-filling imputation strategies discard useful information about when each observation is recorded. Imputing normal values for missing time series ignores the meaningful distinction between truly normal and missing measurements. Also, our window-based resampling procedure reduces the variability of more frequently measured vital signs (e.g., heart rate).

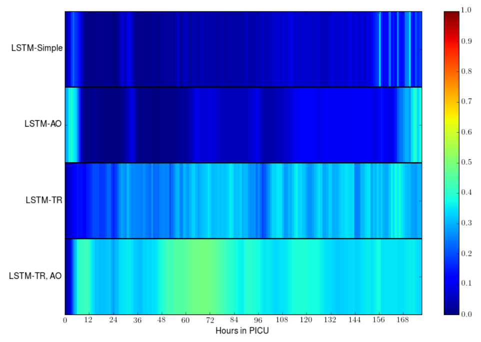
In future work, we plan to introduce indicator variables to allow the LSTM to distinguish actual from missing or imputed measurements. Additionally, the flexibility of the LSTM architecture should enable us to eliminate age-based corrections and to incorporate non-sequential inputs, such as age, weight, and height (or even hand-engineered features), into predictions. Other next steps in this direction include developing LSTM architectures to directly handle missing values and irregular sampling. We also are encouraged by the success of target replication and plan to explore other variants of this technique and to apply it to other domains and tasks. Additionally, we acknowledge that there remains a debate about the interpretability of neural networks when applied to complex medical problems. We are developing methods to interpret the representations learned by LSTMs in order to better expose patterns of health and illness to clinical users. We also hope to make practical use of the distributed representations of patients for tasks such as patient similarity search.

3.11 Acknowledgments

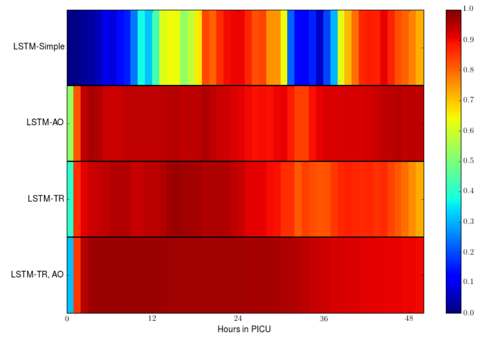
Chapter 3, *Learning to Diagnose with LSTM RNNs*, was written in collaboration with David Kale, Charles Elkan, and Randall Wetzel. We acknowledge the NVIDIA Corporation for Tesla K40 GPU hardware donation and Professors Julian McAuley and Greg Ver Steeg for their support and advice. Finally, we thank the anonymous ICLR reviewers for their feedback, which helped us to make significant improvements to this work and manuscript. The dissertation author was the primary investigator and author of this paper.



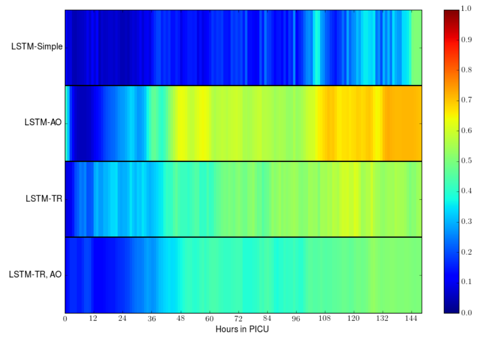
(a) Asthma with Status Asthmaticus



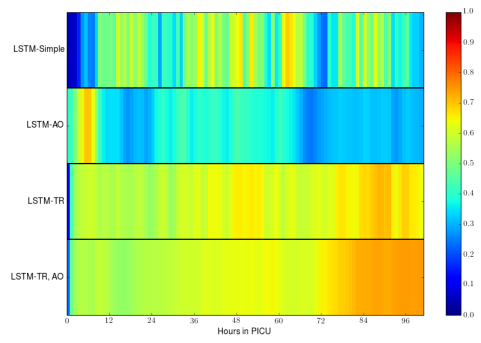
(b) Acute Respiratory Distress Syndrome



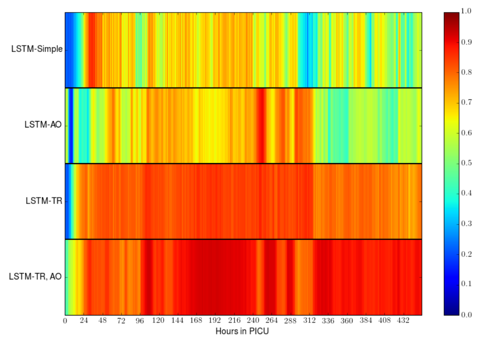
(c) Diabetic Ketoacidosis



(d) Brain Neoplasm, Unspecified Nature

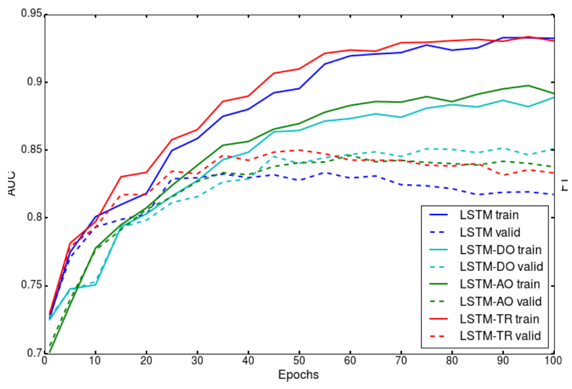


(e) Septic Shock

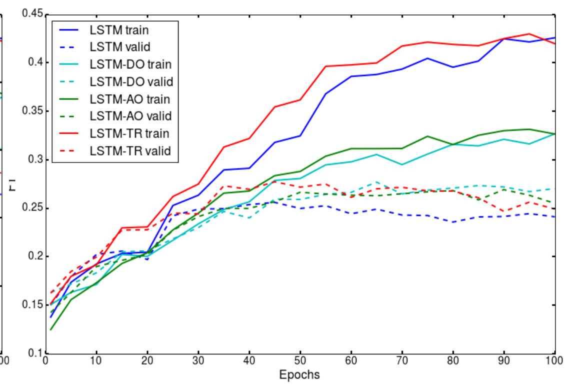


(f) Scoliosis

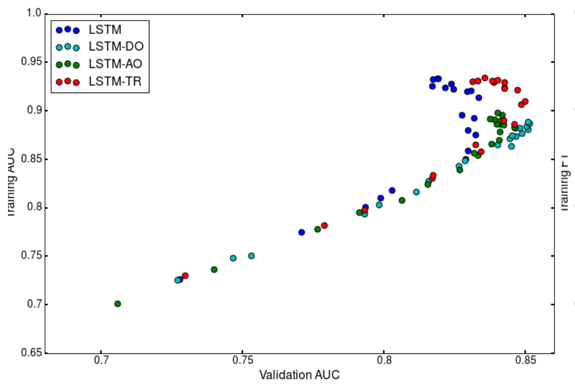
Figure 3.4: Probabilities assigned at each time step. LSTM-Simple uses targets at final step, LSTM-TR uses target replication, LSTM-AO has auxiliary outputs (diagnoses), and LSTM-TR,AO uses both. LSTM-TR tends to make accurate diagnoses earlier.



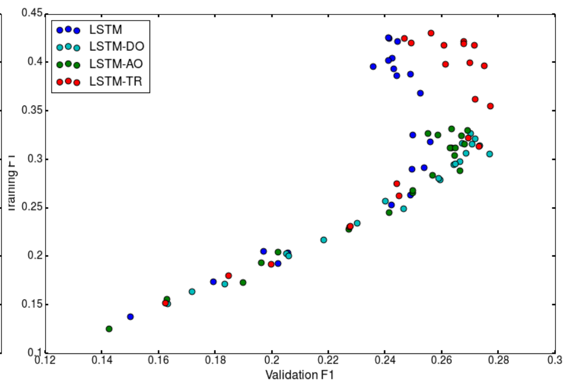
(a) AUC learning curves



(b) F1 learning curves



(c) AUC training *vs.* validation



(d) F1 training *vs.* validation

Figure 3.5: Target replication appears to increase the speed of learning and confers a small regularizing effect. Auxiliary outputs slow down the speed of learning but impart a strong regularizing effect.

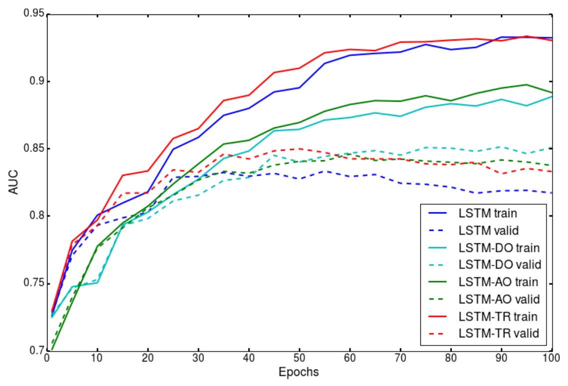


Figure 3.6: Training curves showing the impact of the *DO*, *AO*, and *TR* strategies on overfitting.

Table 3.3: F1 and AUC scores for individual diagnoses.

Classifier Performance on Each Diagnostic Code, Sorted by F1

	LSTM-DO-TR		**MLP, Expert features**		**Max Ensemble**	
Condition	*F1*	AUC	F1	AUC	F1	AUC
Diabetes mellitus with ketoacidosis	0.8571	0.9966	0.8571	0.9966	0.8571	0.9966
Scoliosis, idiopathic	0.6809	0.8543	0.6169	0.8467	0.6689	0.8591
Asthma, unspecified with status asthmaticus	0.5641	0.9232	0.6296	0.9544	0.6667	0.9490
Neoplasm, brain, unspecified nature	0.5430	0.8522	0.5263	0.8463	0.5616	0.8618
Developmental delay	0.4751	0.8178	0.4023	0.8294	0.4434	0.8344
Acute respiratory distress syndrome (ARDS)	0.4688	0.9595	0.3913	0.9645	0.4211	0.9650
Hypertension, unspecified	0.4118	0.8593	0.3704	0.8637	0.3636	0.8652
Arteriovenous malformation of brain	0.4000	0.8620	0.3750	0.8633	0.3600	0.8684
End stage renal disease on dialysis	0.3889	0.8436	0.3810	0.8419	0.3902	0.8464
Acute respiratory failure	0.3864	0.7960	0.4128	0.7990	0.4155	0.8016
Renal transplant status post	0.3846	0.9692	0.4828	0.9693	0.4800	0.9713
Epilepsy, unspecified, not intractable	0.3740	0.7577	0.3145	0.7265	0.3795	0.7477
Septic shock	0.3721	0.8182	0.3210	0.8640	0.3519	0.8546
Other respiratory symptom	0.3690	0.8088	0.3642	0.7898	0.3955	0.8114
Biliary atresia	0.3636	0.9528	0.5000	0.9338	0.4444	0.9541
Acute lymphoid leukemia, without remission	0.3486	0.8601	0.3288	0.8293	0.3175	0.8441
Congenital hereditary muscular dystrophy	0.3478	0.8233	0.0000	0.8337	0.2727	0.8778
Liver transplant status post	0.3448	0.8431	0.3333	0.8104	0.3846	0.8349
Respiratory complications, prodecure status post	0.3143	0.8545	0.2133	0.8614	0.3438	0.8672
Grand mal status	0.3067	0.8003	0.3883	0.7917	0.3529	0.8088
Intracranial injury, closed	0.3048	0.8589	0.3095	0.8621	0.3297	0.8820
Diabetes insipidus	0.2963	0.9455	0.3774	0.9372	0.4068	0.9578
Acute renal failure, unspecified	0.2553	0.8806	0.2472	0.8698	0.2951	0.8821
Other diseases of the respiratory system	0.2529	0.7999	0.1864	0.7920	0.2400	0.8131
Croup syndrome	0.2500	0.9171	0.1538	0.9183	0.0000	0.9263
Bronchiolitis due to other infectious organism	0.2466	0.9386	0.2353	0.9315	0.2712	0.9425
Congestive heart failure	0.2439	0.8857	0.0000	0.8797	0.0000	0.8872
Infantile cerebral palsy, unspecified	0.2400	0.8538	0.1569	0.8492	0.2083	0.8515
Congenital hydrocephalus	0.2393	0.7280	0.2247	0.7337	0.1875	0.7444
Cerebral edema	0.2222	0.8823	0.2105	0.9143	0.2500	0.9190
Craniosynostosis	0.2222	0.8305	0.5333	0.8521	0.6154	0.8658
Anoxic brain damage	0.2222	0.8108	0.1333	0.8134	0.2500	0.8193
Pneumonitis due to inhalation of food or vomitus	0.2222	0.6547	0.0326	0.6776	0.0462	0.6905
Acute and subacute necrosis of the liver	0.2182	0.8674	0.2778	0.9039	0.2381	0.8964
Respiratory syncytial virus	0.2154	0.9118	0.1143	0.8694	0.1622	0.9031
Unspecified disorder of kidney and ureter	0.2069	0.8367	0.1667	0.8496	0.1667	0.8559
Craniofacial malformation	0.2059	0.8688	0.4444	0.8633	0.3158	0.8866
Pulmonary hypertension, secondary	0.2000	0.9377	0.0870	0.8969	0.2105	0.9343
Bronchopulmonary dysplasia	0.1905	0.8427	0.1404	0.8438	0.1333	0.8617
Drowning and non-fatal submersion	0.1905	0.8341	0.1538	0.8905	0.1429	0.8792
Genetic abnormality	0.1828	0.6727	0.1077	0.6343	0.1111	0.6745
Other and unspecified coagulation defects	0.1818	0.7081	0.0000	0.7507	0.1600	0.7328
Vehicular trauma	0.1778	0.8655	0.2642	0.8505	0.2295	0.8723

Table 3.4: F1 and AUC scores for individual diagnoses, continuing

Classifier Performance on Each Diagnostic Code, Sorted by F1

	LSTM-DO-TR		**MLP, Expert features**		**Max Ensemble**	
Condition	*F1*	AUC	F1	AUC	F1	AUC
Other specified cardiac dysrhythmia	0.1667	0.7698	0.1250	0.8411	0.0800	0.8179
Acute pancreatitis	0.1622	0.8286	0.1053	0.8087	0.1379	0.8440
Esophageal reflux	0.1515	0.8236	0.0000	0.7774	0.1739	0.8090
Cardiac arrest, outside hospital	0.1500	0.8562	0.1333	0.9004	0.1765	0.8964
Unspecified pleural effusion	0.1458	0.8777	0.1194	0.8190	0.1250	0.8656
Mycoplasma pneumoniae	0.1429	0.8978	0.1067	0.8852	0.1505	0.8955
Unspecified immunologic disorder	0.1429	0.8481	0.1000	0.8692	0.1111	0.8692
Congenital alveolar hypoventilation syndrome	0.1429	0.6381	0.0000	0.7609	0.0000	0.7246
Septicemia, unspecified	0.1395	0.8595	0.1695	0.8640	0.1905	0.8663
Pneumonia due to adenovirus	0.1379	0.8467	0.0690	0.9121	0.1277	0.8947
Insomnia with sleep apnea	0.1359	0.7892	0.0752	0.7211	0.0899	0.8089
Defibrination syndrome	0.1333	0.9339	0.1935	0.9461	0.2500	0.9460
Unspecified injury, unspecified site	0.1333	0.8749	0.0000	0.7673	0.1250	0.8314
Pneumococcal pneumonia	0.1290	0.8706	0.1149	0.8664	0.1461	0.8727
Genetic or other unspecified anomaly	0.1277	0.7830	0.0870	0.7812	0.1429	0.7905
Other spontaneous pneumothorax	0.1212	0.8029	0.0972	0.8058	0.1156	0.8122
Bone marrow transplant status	0.1176	0.8136	0.0000	0.8854	0.2353	0.8638
Other primary cardiomyopathies	0.1176	0.6862	0.0000	0.6371	0.1212	0.6635
Intracranial hemorrhage	0.1071	0.7498	0.1458	0.7306	0.1587	0.7540
Benign intracranial hypertension	0.1053	0.9118	0.0909	0.7613	0.1379	0.8829
Encephalopathy, unspecified	0.1053	0.8466	0.0909	0.7886	0.0000	0.8300
Ventricular septal defect	0.1053	0.6781	0.0741	0.6534	0.0833	0.6667
Crushing injury, unspecified	0.1017	0.9183	0.0952	0.8742	0.1200	0.9111
Malignant neoplasm, disseminated	0.0984	0.7639	0.0588	0.7635	0.0667	0.7812
Orthopaedic surgery, post status	0.0976	0.7605	0.1290	0.8234	0.0845	0.8106
Thoracic surgery, post status	0.0930	0.9160	0.0432	0.7401	0.0463	0.9137
Ostium secundum type atrial septal defect	0.0923	0.7876	0.1538	0.8068	0.1154	0.7998
Malignant neoplasm, in gastrointestinal organs	0.0853	0.8067	0.1111	0.7226	0.1412	0.7991
Coma	0.0833	0.7255	0.1111	0.6542	0.1250	0.7224
Pneumonia due to inhalation of food or vomitus	0.0800	0.8282	0.0923	0.8090	0.0952	0.8422
Extradural hemorrage from injury, no open wound	0.0769	0.7829	0.0000	0.8339	0.0988	0.8246
Prematurity (less than 37 weeks gestation)	0.0759	0.7542	0.1628	0.7345	0.1316	0.7530
Asthma, unspecified, without status asthmaticus	0.0734	0.6679	0.0784	0.6914	0.0678	0.6867
Gastrointestinal surgery, post status	0.0714	0.7183	0.0984	0.6999	0.0851	0.7069
Nervous disorder, not elsewhere classified	0.0708	0.7127	0.1374	0.7589	0.1404	0.7429
Unspecified gastrointestinal disorder	0.0702	0.6372	0.0348	0.6831	0.0317	0.6713
Pulmonary congestion and hypostasis	0.0678	0.8359	0.0000	0.8633	0.0000	0.8687
Thrombocytopenia, unspecified	0.0660	0.7652	0.0000	0.7185	0.0000	0.7360
Lung contusion, no open wound	0.0639	0.9237	0.0000	0.9129	0.2222	0.9359
Acute pericarditis, unspecified	0.0625	0.8601	0.0000	0.9132	0.0000	0.9089
Nervous system complications from implant	0.0597	0.6727	0.0368	0.7082	0.0419	0.7129
Heart disease, unspecified	0.0588	0.8372	0.0000	0.8020	0.0000	0.8264
Suspected infection in newborn or infant	0.0588	0.6593	0.0000	0.7090	0.0606	0.6954

Table 3.5: F1 and AUC scores for individual diagnoses, continuing

Classifier Performance on Each Diagnostic Code, Sorted by F1

	LSTM-DO-TR		**MLP, Expert features**		**Max Ensemble**	
Condition	*F1*	AUC	F1	AUC	F1	AUC
Anemia, unspecified	0.0541	0.7782	0.0488	0.7019	0.0727	0.7380
Muscular disorder, not elsewhere classified	0.0536	0.6996	0.0000	0.7354	0.1000	0.7276
Malignant neoplasm, adrenal gland	0.0472	0.6960	0.0727	0.6682	0.0548	0.6846
Hematologic disorder, unspecified	0.0465	0.7315	0.1194	0.7404	0.0714	0.7446
Hematemesis	0.0455	0.8116	0.0674	0.7887	0.0588	0.8103
Dehydration	0.0435	0.7317	0.1739	0.7287	0.0870	0.7552
Unspecified disease of spinal cord	0.0432	0.7153	0.0571	0.7481	0.0537	0.7388
Neurofibromatosis, unspecified	0.0403	0.7494	0.0516	0.7458	0.0613	0.7671
Intra-abdominal injury, no open wound	0.0333	0.7682	0.1569	0.8602	0.0690	0.8220
Thyroid disorder, unspecified	0.0293	0.5969	0.0548	0.5653	0.0336	0.6062
Hereditary hemolytic anemia, unspecifed	0.0290	0.7474	0.0000	0.6182	0.0000	0.6962
Subdural hemorrage, no open wound	0.0263	0.7620	0.1132	0.7353	0.0444	0.7731
Unspecified intestinal obstruction	0.0260	0.6210	0.2041	0.7684	0.0606	0.7277
Hyposmolality and/or hyponatremia	0.0234	0.6999	0.0000	0.7565	0.0000	0.7502
Primary malignant neoplasm, thorax	0.0233	0.6154	0.0364	0.6086	0.0323	0.5996
Supraventricular premature beats	0.0185	0.8278	0.0190	0.7577	0.0299	0.8146
Injury to intrathoracic organs, no open wound	0.0115	0.8354	0.0000	0.8681	0.0000	0.8604
Child abuse, unspecified	0.0000	0.9273	0.3158	0.9417	0.1818	0.9406
Acidosis	0.0000	0.9191	0.1176	0.9260	0.0000	0.9306
Infantile spinal muscular atrophy	0.0000	0.9158	0.0000	0.8511	0.0000	0.9641
Fracture, femoral shaft	0.0000	0.9116	0.0000	0.9372	0.0513	0.9233
Cystic fibrosis with pulmonary manifestations	0.0000	0.8927	0.0000	0.8086	0.0571	0.8852
Panhypopituitarism	0.0000	0.8799	0.2222	0.8799	0.0500	0.8872
Blood in stool	0.0000	0.8424	0.0000	0.8443	0.0000	0.8872
Sickle-cell anemia, unspecified	0.0000	0.8268	0.0000	0.7317	0.0000	0.7867
Cardiac dysrhythmia, unspecified	0.0000	0.8202	0.0702	0.8372	0.0000	0.8523
Agranulocytosis	0.0000	0.8157	0.1818	0.8011	0.1667	0.8028
Malignancy of bone, no site specified	0.0000	0.8128	0.0870	0.7763	0.0667	0.8318
Pneumonia, organism unspecified	0.0000	0.8008	0.0952	0.8146	0.0000	0.8171
Unspecified metabolic disorder	0.0000	0.7914	0.0000	0.6719	0.0000	0.7283
Urinary tract infection, no site specified	0.0000	0.7867	0.0840	0.7719	0.2286	0.7890
Obesity, unspecified	0.0000	0.7826	0.0556	0.7550	0.0000	0.7872
Apnea	0.0000	0.7822	0.2703	0.8189	0.0000	0.8083
Respiratory arrest	0.0000	0.7729	0.0000	0.8592	0.0000	0.8346
Hypovolemic shock	0.0000	0.7686	0.0000	0.8293	0.0000	0.8296
Hemophilus meningitis	0.0000	0.7649	0.0000	0.7877	0.0000	0.7721
Diabetes mellitus, type I, stable	0.0000	0.7329	0.0667	0.7435	0.0833	0.7410
Tetralogy of fallot	0.0000	0.7326	0.0000	0.6134	0.0000	0.6738
Congenital heart disease, unspecified	0.0000	0.7270	0.1333	0.7251	0.0000	0.7319
Mechanical complication of V-P shunt	0.0000	0.7173	0.0000	0.7308	0.0000	0.7205
Respiratory complications due to procedure	0.0000	0.7024	0.0000	0.7244	0.0000	0.7323
Teenage cerebral artery occlusion and infarction	0.0000	0.6377	0.0000	0.5982	0.0000	0.6507

Chapter 4

Modeling Missing Data in Clinical Time Series

While the previous chapter deals with missing data by applying imputation heuristics, this ignores an important source of information. Medical data are seldom missing at random, and the patterns of missingness have predictive value. We build on the work in the previous chapter by demonstrating a simple strategy to cope with missing data in sequential inputs. Again, we address the task of multilabel classification of diagnoses given clinical time series using data collected from the pediatric intensive care unit (PICU) at Children's Hospital Los Angeles. The measurements are irregularly spaced, leading to missingness patterns in temporally discretized sequences. While these artifacts are typically handled by imputation, we achieve superior predictive performance by treating the artifacts as features.

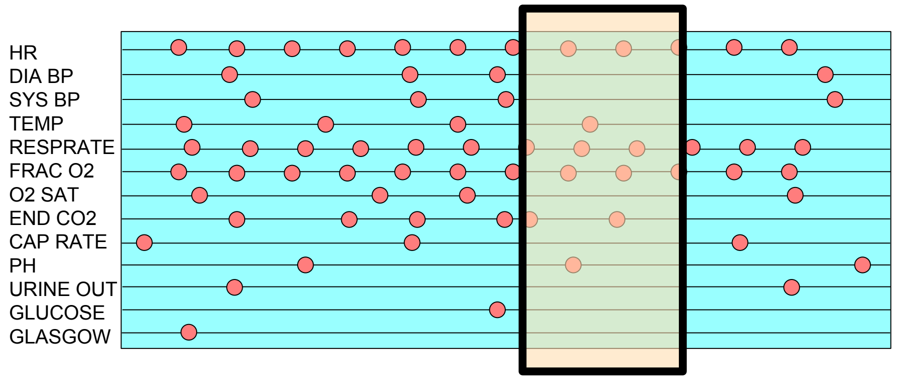


Figure 4.1: Missingness artifacts created by discretization

4.1 Introduction

EHRs record lab test results and medications as they are ordered or delivered by physicians and nurses. As a result, EHRs contain rich sequences of clinical observations depicting both patients' health and care received. We would like to mine these time series to build accurate predictive models for diagnosis and other applications. Recurrent neural networks (RNNs) are well-suited to learning sequential or temporal relationships from such time series. Medical time series data present modeling problems not found in the clean academic datasets on which most RNN research focuses. Clinical observations are recorded irregularly, with measurement frequency varying between patients, across variables, and even over time. In one common modeling strategy, we represent these observations as a sequence with discrete, fixed-width time steps. Problematically, the resulting sequences often contain missing values [MKKW12]. These values are typically not missing at random, but reflect decisions by caregivers. Thus, the pattern of recorded measurements contain potential information about the state of the patient. However, most often, researchers fill missing values using heuristic or unsupervised imputation [LDL13], ignoring the potential predictive value of the missingness itself.

In this chapter we extend the methods described in the previous chapter and orginally published in [LKEW16] for RNN-based multilabel prediction of diagnoses. We focus on data gathered from the Children's Hospital Los Angeles pediatric intensive care unit (PICU). Here, rather than approaching missing data via heuristic imputation, we directly model missingness as a feature, achieving superior predictive performance. RNNs can realize this improvement using only simple binary indicators for missingness. However, linear models are unable to use indicator features as effectively. While RNNs can learn arbitrary functions, capturing the interactions between the missingness indicators the sequence of observation inputs, linear models can only learn substitution values. For linear models, we introduce an alternative strategy to capture this signal, using a small number of simple hand-engineered features.

Our experiments demonstrate the benefit modeling missing data as a first-class feature. Our methods improve the performance of RNNs, multilayer perceptrons (MLPs), and linear models. Additionally we analyze the predictive value of missing data information by training models on the missingness indicators only. We show that for several diseases, *what tests are run* can be as predictive as the actual measurements. While we focus on classifying diagnoses, our methods can be applied to any predictive modeling problem involving sequence data and missing values, such as early prediction of sepsis [HHPS15] or real-time risk modeling [WHG12].

It is worth noting that we may not want our predictive models to rely upon the patterns of treatment, as argued by [CLG$^+$15]. Once deployed, our models may influence the treatment protocols, shifting the distribution of future data, and thus invalidating their predictions. Nonetheless, doctors at present often utilize knowledge of past care, and treatment signal can leak into the actual measurements themselves in ways that sufficiently powerful models can exploit. As a final contribution of this chapter, we present a critical discussion of these practical and philosophical issues.

4.2 Data

As in the previous chapter, our dataset consists of patient records extracted from the EHR system at CHLA [MKKW12, CKL+15] as part of an IRB-approved study. In all, the dataset contains $10,401$ PICU episodes. Each episode describes the stay of one patient in the PICU for a period of at least 12 hours. In addition, each patient record contains a static set of diagnostic codes, annotated by physicians either during or after each PICU visit.

4.2.1 Inputs

In their rawest representation, episodes consist of irregularly spaced measurements of 13 variables: diastolic and systolic blood pressure, peripheral capillary refill rate, end-tidal CO_2 ($ETCO_2$), fraction of inspired O_2 (FIO_2), total Glascow coma scale, blood glucose, heart rate, pH, respiratory rate, blood oxygen saturation, body temperature, and urine output. To render our data suitable for learning with RNNs, we convert to discrete sequences of hourly time steps, where time step t covers the interval between hours t and $t+1$, closed on the left but open on the right. Because actual admission times are not recorded reliably, we use the time of the first recorded observation as time step $t = 0$. We combine multiple measurements of the same variable within the same hour window by taking their mean.

Vital signs, such as heart rate, are typically measured about once per hour, while lab tests requiring a blood draw (e.g., glucose) are measured on the order of once per day. In addition, the timing of and time between observations varies across patients and over time. The resulting sequential representation have many missing values, and some variables missing altogether.

Note that our methods can be sensitive to the duration of our discrete time step.

For example, halving the duration would double the length of the sequences, making learning by backpropagation through time more challenging [BSF94]. For our data, such cost would not be justified because the most frequently measured variables (vital signs) are only recorded about once per hour. For higher frequency recordings of variables with faster dynamics, a shorter time step might be warranted.

To better condition our inputs, we scale each variable to the $[0, 1]$ interval, using expert-defined ranges. Additionally, we correct for differences in heart rate, respiratory rate, [FTS+11] and blood pressure [Nat04] due to age and gender using tables of normal values from large population studies.

4.2.2 Diagnostic labels

As in the previous chapter, we formulate *phenotyping* [OCG+15] as multilabel classification of sequences. Our labels include 429 distinct diagnosis codes from an in-house taxonomy at CHLA, similar to ICD-9 codes [Wor04] commonly used in medical informatics research. These labels include a wide range of acute conditions, such as acute respiratory distress, congestive heart failure, and sepsis. A full list is given after the discussion. We focus on the 128 most frequent diagnoses, each having at least 50 positive examples in our dataset. Naturally, the diagnoses are not mutually exclusive. In our data set, the average patient is associated with 2.24 diagnoses. Additionally, the base rates of the diagnoses vary widely.

The loss at a single sequence step is the average *log loss* calculated across all labels:

4.3 Summary of Missing Data Statistics

In this section, we explain our procedures for imputation, missing data indicator sequences, engineering features of missing data patterns.

4.3.1 Imputation

To address the missing data problem, we consider two different imputation strategies (forward-filling and zero imputation), as well as direct modeling via indicator variables. Because imputation and direct modeling are not mutually exclusive, we also evaluate them in combination. Suppose that $x_i^{(t)}$ is "missing." In our *zero-imputation* strategy, we simply set $x_i^{(t)} := 0$ whenever it is missing. In our *forward-filling* strategy, we impute $x_i^{(t)}$ as follows:

- If there is at least one previously recorded measurement of variable i at a time $t' < t$, we perform forward-filling by setting $x_i^{(t)} := x_i^{(t')}$.
- If there is no previous recorded measurement (or if the variable is missing entirely), then we impute the median estimated over all measurements in the training data.

This strategy is motivated by the intuition that clinical staff record measurements at intervals proportional to rate at which they are believed or observed to change. Heart rate, which can change rapidly, is monitored much more frequently than blood pH. Thus it seems reasonable to assume that a value has changed little since the last time it was measured.

4.3.2 Learning with Missing Data Indicators

Our *indicator variable* approach to missing data consists of augmenting our inputs with binary variables $m_i^{(t)}$ for every $x_i^{(t)}$, where $m_i^{(t)} := 1$ if $x_i^{(t)}$ is imputed and 0

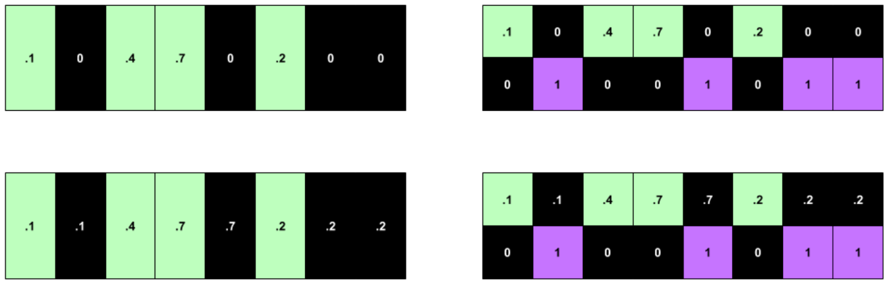


Figure 4.2: (top left) no imputation or indicators, (bottom left) imputation absent indicators, (top right) indicators but no imputation, (bottom right) indicators and imputation. Time flows from left to right.

otherwise. Through their hidden state computations, RNNs can use these indicators to learn arbitrary functions of the past observations and missingness patterns. However, given the same data, linear models can only learn hard substitution rules. To see why, consider a linear model that outputs prediction $f(z)$, where $z = \sum_i w_i \cdot x_i$. With indicator variables, we might say that $z = \sum_i w_i \cdot x_i + \sum_i \theta_i \cdot m_i$ where θ_i are the weights for each m_i. If x_i is set to 0 and m_i to 1, whenever the feature x_i is missing, then the impact on the output $\theta_i \cdot m_i = \theta_i$ is exactly equal to the contribution $w_i \cdot x_i^*$ for some $x_i^* = \theta_i / w_i$. In other words, the linear model can only use the indicator in a way that depends neither on the previously observed values $(x_i^1 ... x_i^{t-1})$, nor any other evidence in the inputs.

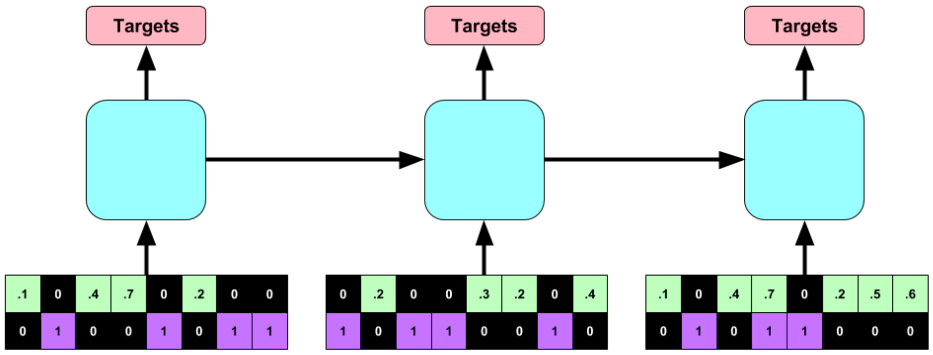


Figure 4.3: Depiction of RNN zero-filled inputs and missing data indicators.

Note that for a linear model, the impact of a missing data indicator on predictions must be monotonic. In contrast, the RNN might infer that for one patient heart rate is missing because they went for a walk, while for another it might signify an emergency. Also note that even without indicators, the RNN might learn to recognize *filled-in* vs *real* values. For example, with forward-filling, the RNN could learn to recognize exact repeats. For zero-filling, the RNN could recognize that values set to exactly 0 were likely missing measurements.

4.3.3 Hand-engineered missing data features

To overcome the limits of the linear model, we also designed features from the indicator sequences. As much as possible, we limited ourselves to features that are simple to calculate, intuitive, and task-agnostic. The first is a binary indicator for whether a variable was measured *at all*. Additionally, we compute the mean and standard deviation of the indicator sequence. The mean captures the frequency with which each variable is measured which carries information about the severity of a patient's condition. The standard deviation, on the other hand, computes a non-monotonic function of frequency that is maximized when a variable is missing exactly 50% of the time. We also compute the frequency with which a variable switches from measured to missing or vice versa across adjacent sequence steps. Finally, we add features that capture the relative timing of the first and last measurements of a variable, computed as the number of hours until the measurement divided by the length of the full sequence.

4.4 Experiments

We now present the training details and empirical findings of our experiments. Our LSTM RNNs each have 2 hidden layers of 128 LSTM cells each, non-recurrent dropout of 0.5, and ℓ_2^2 weight decay of 10^{-6}. We train on 80% of data, setting aside

10% each for validation and testing. We train each RNN for 100 epochs, retaining the parameters corresponding to the epoch with the lowest validation loss.

We compare the performance of RNNs against logistic regression and multilayer perceptrons (MLPs). We apply ℓ_2 regularization to the logistic regression model. The MLP has 3 hidden layers with 500 nodes each, rectified linear unit activations, and dropout (with probability of 0.5), choosing the number of layers and nodes by validation performance. We train the MLP using stochastic gradient descent with momentum.

We evaluate each baseline with two sets of features: raw and hand-engineered. Note that our baselines cannot be applied directly to variable-length inputs. For the raw features, we concatenate three 12-hour subsequences, one each from the beginning, middle, and end of the time series. For shorter time series, these intervals may overlap. Thus raw representations contain $2 \times 3 \times 12 \times 13 = 936$ features. We train each baseline on five different combinations of raw inputs: (1) measurements with zero-filling, (2) measurements with forward-filling, (2) measurements with zero-filling + missing data indicators, (4) forward-filling + missing data indicators, and (5) missing data indicators only.

Our hand-engineered features capture central tendencies, variability, extremes, and trends. These include the first and last measurements and their difference, maximum and minimum values, mean and standard deviation, median and 25th and 75th percentiles, and the slope and intercept of least squares line fit. We also computed the 8 missing data features described in Section 4.3. We improve upon the baselines in [LKEW16] by computing the hand-engineered features over different windows of time, giving them access to greater temporal information and enabling them to better model patterns of missingness. We extract hand-engineered features from the entire time series and from three possibly overlapping intervals: the first and last 12 hours and the interval between (for shorter sequences, we instead use the middle 12 hours). This yields a total of

$4 \times 12 \times 13 = 624$ and $4 \times 8 \times 13 = 416$ hand-engineered measurement and missing data features, respectively. We train baseline models on three different combinations of hand-engineered features: (1) measurement-only, (2) indicator-only, and (3) measurement and indicator.

We evaluate all models on the same training, validation, and test splits. Our evaluation metrics include area under the ROC curve (AUC) and F1 score (with threshold chosen based on validation performance). We report both micro-averaged (calculated across all predictions) and macro-averaged (calculated separately on each label, then averaged) measures to mitigate the weaknesses in each [LEN14]. Finally we also report precision at 10, whose maximum is 0.2238 because we have on average 2.238 diagnoses per patient. This metric seems appropriate because we could imagine this technology would be integrated into a diagnostic assistant. In that case, its role might be to suggest the most likely diagnoses among which a professional doctor would choose. Precision at 10 evaluates the quality of the top 10 suggestions.

4.4.1 Results

The best overall model by all metrics (micro AUC of 0.8730) is an LSTM with zero-imputation and missing data indicators. It outperforms both the strongest MLP baseline and LSTMs absent missing data indicators. For the LSTMs using either imputation strategy, adding the missing data indicators improves performance in all metrics. While all models improve with access to missing data indicators, this information confers less benefit to the raw input linear baselines, consistent with theory discussed in Section 4.3.2.

The results achieved by logistic regression with hand-engineered features indicates that our simple hand-engineered missing data features do a reasonably good job of capturing important information that neural networks are able to mine automatically.

We also find that LSTMs (with or without indicators) appear to perform better with zero-filling than with with imputed values. Interestingly, this is not true for either baseline. It suggests that the LSTM may be learning to recognize missing values implicitly by recognizing a tight range about the value zero and inferring that this is a missing value. If this is true, perhaps imputation interferes with the LSTM's ability to implicitly recognize missing values. Overall, the ability to implicitly infer missingness may have broader implications. It suggests that we might never completely hide this information from a sufficiently powerful model.

4.5 Per Diagnosis Classification Performance

In this section, we provide per-diagnosis AUC and F1 scores for three representative LSTM models trained with imputed measurements, with imputation plus missing indicators, and with indicators only. By comparing performance on individual diagnoses, we can gain some insight into the relationship between missing values and different conditions. Rows are sorted in descending order based on the F1 score of the imputation plus indicators model. It is worth noting that F1 scores are sensitive to threshold, which we chose in order to optimize per-disease validation F1, sometimes based on a very small number of positive cases. Thus, there are cases where one model will have superior AUC but worse F1.

4.6 Missing

In this section, we present information about the sampling rates and missingness characteristics of our 13 variables. The first column lists the average number of measurements per hour in all episodes with at least one measurement (excluding episodes where the variable is missing entirely). The second column lists the fraction of episodes in which the variable is missing completely (there are zero measurements). The third column lists the missing rate in the resulting discretized sequences.

4.7 Related Work

This work builds upon research relating to missing values and machine learning for medical informatics. The basic RNN methodology for phenotyping derives from [LKEW16], addressing a dataset and problem described by [CKL$^+$15]. The methods rely upon LSTM RNNs [HS97, GSC00] trained by backpropagation through time [HOT06, Wer88]. A comprehensive perspective on the history and modern applications of RNNs is provided by [LBE15], while [LKEW16] list many of the previous works that have applied neural networks to digital health data.

While a long and rich literature addresses pattern recognition with missing data [CC75, All01], most of this literature addresses fixed-length feature vectors [GLSGFV10, Pig01]. Indicator variables for missing data were first proposed by [CC75], but we could not find papers that combine missing data indicators with RNNs. Only a handful of papers address missing data in the context of RNNS. [BG96] demonstrate a scheme by which the RNN learns to fill in the missing values such that the filled-in values minimize output error. In 2001, [PG01] built upon this method to improve automatic speech recognition. [BGC01] suggests using a mask of indicators in a scheme for weighting the contribution of reliable vs corrupted data in the final prediction. [TB98] address missing values by combining an RNN with a linear state space model to handle uncertainty. This paper may be one of the first to engineer explicit features of missingness patterns in order to improve discriminative performance. Also, to our knowledge, we are the first to harness patterns of missing data to improve the classification of critical care phenotypes.

4.8 Discussion

Data processing and discriminative learning have often been regarded as separate disciplines. Through this separation of concerns, the complementarity of missing data indicators and training RNNs for classification has been overlooked. In this chapter, we propose that patterns of missing values are an underutilized source of predictive power and that RNNs, unlike linear models, can effectively mine this signal from sequences of indicator values. Our hypotheses are confirmed by empirical evidence. Additionally, we introduce and confirm the utility of a simple set of features, engineered from the sequence of missingness indicators, that can improve performance of linear models. These techniques are simple to implement and broadly applicable and seem likely to confer similar benefits on other sequential prediction tasks, when data is missing not at random. One example might include financial data, where failures to report accounting details could suggest internal problems at a company.

4.8.1 The Perils and Inevitability of Modeling Treatment Patterns

For medical applications, the predictive power of missing data raises important philosophical concerns. We train models with supervised learning, and verify their utility by assessing the accuracy of their classifications on hold-out test data. However, in practice, we hope to make treatment decisions based on these predictions, exposing a fundamental incongruity between the problem on which our models are trained and those for which they are ultimately deployed. As articulated in [Lip16], these supervised models, trained offline, cannot account for changes that their deployment might confer upon the real world, possibly invalidating their predictions. [CLG+15] present a compelling case in which a pneumonia risk model predicted a lower risk of death for patients who also have asthma. The better outcomes of the asthma patients, as it turns out, owed to the more aggressive treatment they received. The model, if deployed, might be used to choose less aggressive treatment for the patients with both pneumonia and asthma, clearly a sub-optimal course of action.

On the other hand, to some degree, learning from treatment signal may be inevitable. Any imputation might leak some information about which values are likely imputed and which are not. Thus any sufficiently powerful supervised model might catch on to some amount of missingness signal, as was the case in our experiments with the LSTM using zero-filled missing values. Even physiologic measurements contain information owing to patterns of treatment, possibly reflecting the medications patients receive and the procedures they undergo.

Sometimes the patterns of treatments may be a reasonable and valuable source of information. Doctors already rely on this kind of signal habitually: they read through charts, noting which other doctors have seen a patient, inferring what their opinions might have been from which tests they ordered. While, in some circumstances, it may be problematic for learning models to rely on this signal, removing it entirely may be both difficult and undesirable.

4.8.2 Complex Models or Complex Features?

Our work also shows that using only simple features, RNNs can achieve state of the art performance classifying clinical time series. The RNNs far outperform linear models. Still, in our experience, there is a strong bias among practitioners toward more familiar models even when they require substantial feature engineering.

In our experiments, we undertook extensive efforts to engineer features to boost the performance of both linear models and MLPs. Ultimately, while RNNs performed best on raw data, we could approach its performance with an MLP and significantly improve the linear model by using hand-engineered features and windowing. A question then emerges: how should we evaluate the trade-off between more complex models and more complex features? To the extent that linear models are believed to be more interpretable than neural networks, most popular notions of interpretability hinge upon the intelligibility of the features [Lip16]. When performance of the linear model comes at the price of this intelligibility, we might ask if this trade-off undermines the linear model's chief advantage. Additionally, such a model, while still inferior to the RNN, relies on application-specific features less likely to be useful on other datasets and tasks. In contrast, RNNs seem better equipped to generalize to different tasks. While the model may be complex, the inputs remain intelligible, opening the possibility to various post-hoc interpretations [Lip16].

4.8.3 Future Work

We see several promising next steps following this work. First, we would like to validate this methodology on tasks with more immediate clinical impact, such as predicting sepsis, mortality, or length of stay. Second, we'd like to extend this work towards predicting clinical decisions. Called policy imitation in the reinforcement literature, such work could pave the way to providing real-time decision support. Finally, we see machine learning as cooperating with a human decision-maker. Thus a machine learning model needn't always make a prediction/classification; it could also abstain. We hope to make use of the latest advances in mining uncertainty information from neural networks to make confidence-rated predictions.

4.9 Acknowledgments

Chapter 4, *Modeling Missing Data in Clinical Time Series with RNNs*, was written in collaboration with David Kale and Randall Wetzell. Thanks to Professors Charles Elkan, Julian McAuley and Greg Ver Steeg for their support and advice. The dissertation author was the primary investigator and author of this paper.

Table 4.1: Performance on aggregate metrics for logistic regression (Log Reg), MLP, and LSTM classifiers with and without imputation and missing data indicators.

Classification performance for 128 ICU phenotypes

Model	Micro AUC	Macro AUC	Micro F1	Macro F1	P@10
Base Rate	0.7128	0.5	0.1346	0.0343	0.0788
Best Possible	1.0	1.0	1.0	1.0	0.2281
Logistic Regression					
Log Reg - Zeros	0.8108	0.7244	0.2149	0.0999	0.1014
Log Reg - Impute	0.8201	0.7455	0.2404	0.1189	0.1038
Log Reg - Zeros & Indicators	0.8143	0.7269	0.2239	0.1082	0.1017
Log Reg - Impute & Indicators	0.8242	0.7442	0.2467	0.1234	0.1045
Log Reg - Indicators Only	0.7929	0.6924	0.1952	0.0889	0.0939
Multilayer Perceptron					
MLP - Zeros	0.8263	0.7502	0.2344	0.1072	0.1048
MLP - Impute	0.8376	0.7708	0.2557	0.1245	0.1031
MLP - Zeros & Indicators	0.8381	0.7705	0.2530	0.1224	0.1067
MLP - Impute & Indicators	0.8419	0.7805	0.2637	0.1296	0.1082
MLP - Indicators Only	0.8112	0.7321	0.1962	0.0949	0.0947
LSTMs					
LSTM - Zeros	0.8662	0.8133	0.2909	0.1557	0.1176
LSTM - Impute	0.8600	0.8062	0.2967	0.1569	0.1159
LSTM - Zeros & Indicators	**0.8730**	**0.8250**	**0.3041**	**0.1656**	**0.1215**
LSTM - Impute & Indicators	0.8689	0.8206	0.3027	0.1609	0.1196
LSTM - Indicators Only	0.8409	0.7834	0.2403	0.1291	0.1074
Models using Hand-Engineered Features					
Log Reg HE	0.8396	0.7714	0.2708	0.1327	0.1118
Log Reg HE Indicators	0.8472	0.7752	0.2841	0.1376	0.1165
Log Reg HE Indicators Only	0.8187	0.7322	0.2287	0.1001	0.1020
MLP HE	0.8599	0.8052	0.2953	0.1556	0.1168
MLP HE Indicators	0.8669	0.8160	0.2954	0.1610	0.1202
MLP HE Indicators Only	0.8371	0.7682	0.2351	0.1179	0.1028

Table 4.2: AUC and F1 scores for individual diagnostic codes.

Classifier Performance on Each Diagnostic Code, Sorted by F1

		Msmt.		**Msmt. + indic.**		**Indic.**	
Condition	Base rate	AUC	F1	AUC	**F1**	AUC	F1
Diabetes mellitus with ketoacidosis	0.0125	1.0000	0.8889	0.9999	**0.9333**	0.9906	0.7059
Asthma with status asthmaticus	0.0202	0.9384	**0.6800**	0.8907	0.6383	0.8652	0.5417
Scoliosis (idiopathic)	0.1419	0.9143	**0.6566**	0.8970	0.6174	0.8435	0.5235
Tumor, cerebral	0.0917	0.8827	**0.5636**	0.8799	0.5560	0.8312	0.4627
Renal transplant, status post	0.0122	0.9667	0.2963	0.9544	0.4762	0.9490	**0.5600**
Liver transplant, status post	0.0106	0.7534	0.3158	0.8283	**0.4762**	0.8271	0.2581
Acute respiratory distress syndrome	0.0193	0.9696	0.3590	0.9705	**0.4557**	0.9361	0.3333
Developmental delay	0.0876	0.8108	**0.4382**	0.8382	0.4331	0.6912	0.2366
Diabetes insipidus	0.0127	0.9220	0.2727	0.9486	**0.4286**	0.9266	0.4000
End stage renal disease (on dialysis)	0.0241	0.8548	0.2778	0.8800	0.4186	0.9043	**0.4255**
Seizure disorder	0.0816	0.7610	0.3694	0.7937	**0.4059**	0.6431	0.1957
Acute respiratory failure	0.0981	0.8414	0.4128	0.8391	0.3835	0.8358	**0.4542**
Cystic fibrosis	0.0076	0.8628	0.2353	0.8740	**0.3810**	0.8189	0.0000
Septic shock	0.0316	0.8296	0.3363	0.8911	**0.3750**	0.8506	0.1429
Respiratory distress, other	0.0716	0.8411	**0.3873**	0.8502	0.3719	0.7857	0.2143
Intracranial injury, closed	0.0525	0.8886	0.2817	0.9002	**0.3711**	0.8442	0.3208
Arteriovenous malformation	0.0223	0.8620	0.3590	0.8716	**0.3704**	0.8494	0.2857
Seizures, status epilepticus	0.0348	0.8381	**0.4158**	0.8505	0.3704	0.8440	0.3226
Pneumonia due to adenovirus	0.0123	0.8604	0.1250	0.9065	**0.3077**	0.8792	0.1818
Leukemia (acute, without remission)	0.0287	0.8585	0.2783	0.8845	**0.3059**	0.8551	0.2703
Dissem. intravascular coagulopathy	0.0099	0.9556	**0.5000**	0.9532	0.2857	0.9555	0.2500
Septicemia, other	0.0240	0.8586	0.2400	0.8870	**0.2812**	0.7593	0.0000
Bronchiolitis	0.0162	0.9513	0.2667	0.9395	**0.2703**	0.8826	0.1778
Congestive heart failure	0.0133	0.8748	0.1429	0.8756	**0.2703**	0.8326	0.1364
Upper airway obstruc. (UAO), other	0.0378	0.8206	**0.2564**	0.8573	0.2542	0.8350	0.1964
Diabetes mellitus type I, stable	0.0064	0.7105	0.0000	0.9625	0.2500	0.9356	**0.3333**
Cerebral palsy (infantile)	0.0262	0.8230	**0.2609**	0.8359	0.2500	0.6773	0.0980
Coagulopathy	0.0131	0.7501	0.1111	0.8098	**0.2449**	0.8548	0.1667
UAO, ENT surgery, post-status	0.0302	0.9059	**0.4058**	0.8733	0.2400	0.8364	0.1975
Hypertension, systemic	0.0169	0.8740	0.2105	0.8831	0.2388	0.8216	**0.2857**
Acute renal failure, unspecified	0.0191	0.9242	0.2381	0.9510	0.2381	0.9507	**0.3291**
Trauma, vehicular	0.0308	0.8673	0.2105	0.8649	**0.2381**	0.8022	0.1395
Hepatic fail. (acute necrosis of liver)	0.0176	0.8489	0.2222	0.9239	**0.2308**	0.8598	0.1935
Craniosynostosis (anomalies of skull)	0.0064	0.7824	0.0000	0.9267	**0.2286**	0.8443	0.0315
Prematurity (<37 weeks gestation)	0.0321	0.7520	0.1548	0.7542	**0.2245**	0.7042	0.1266
Hydrocephalus, other (congenital)	0.0381	0.7118	0.2099	0.7500	**0.2241**	0.7065	0.1961
Pneumothorax	0.0134	0.8220	0.1176	0.7957	0.2188	0.7552	**0.3243**
Congenital muscular dystrophy	0.0121	0.8427	**0.2500**	0.8491	0.2143	0.7460	0.0800
Cardiomyopathy (primary)	0.0191	0.7508	0.1290	0.6057	**0.2143**	0.6372	0.1818
Pulmonary edema	0.0076	0.8839	0.0769	0.8385	**0.2105**	0.8071	0.0870

Table 4.3: AUC and F1 scores for individual diagnostic codes, continuing.

Classifier Performance on Each Diagnostic Code, Sorted by F1

		Msmt.		**Msmt. + indic.**		**Indic.**	
Condition	Base rate	AUC	F1	AUC	**F1**	AUC	F1
(Acute) pancreatitis	0.0106	0.8712	0.0769	0.9512	**0.2000**	0.8182	0.0571
Tumor, disseminated or metastatic	0.0180	0.7178	0.0938	0.7415	**0.1967**	0.6837	0.1062
Hematoma, intracranial	0.0299	0.7724	**0.2278**	0.8249	0.1892	0.7518	0.1474
Neutropenia (agranulocytosis)	0.0108	0.8285	0.0000	0.8114	0.1852	0.8335	**0.2609**
Arrhythmia, other	0.0087	0.8536	0.0000	0.8977	**0.1818**	0.8654	0.0000
Child abuse, suspected	0.0065	0.9544	**0.2222**	0.8642	0.1818	0.8227	0.0870
Encephalopathy, hypoxic/ischemic	0.0116	0.8242	0.1429	0.8571	**0.1818**	0.8009	0.0800
Epidural hematoma	0.0098	0.7389	0.0455	0.8233	**0.1818**	0.7936	0.1000
Tumor, gastrointestinal	0.0100	0.8112	0.1429	0.8636	**0.1778**	0.8732	0.0984
Craniofacial malformation	0.0133	0.8707	**0.2667**	0.8514	0.1778	0.6928	0.2286
Gastroesophageal reflux	0.0182	0.7571	**0.1818**	0.8554	0.1690	0.7739	0.1600
Pneumonia, bacterial (pneumococ.)	0.0186	0.8876	0.1333	0.8806	**0.1600**	0.8616	0.0000
Pneumonia, undetermined	0.0179	0.8323	0.1481	0.8269	**0.1583**	0.7772	0.0947
Cerebral edema	0.0059	0.8275	0.0000	0.9469	**0.1538**	0.9195	0.1500
Pneumonia due to inhalation	0.0078	0.7917	0.1111	0.8602	**0.1538**	0.8268	0.0566
Metabolic or endocrine disorder	0.0095	0.7718	0.0364	0.6929	0.1538	0.6319	**0.2000**
Disorder of kidney and ureter, other	0.0204	0.8486	**0.2857**	0.8650	0.1500	0.8238	0.2500
Urinary tract infection	0.0137	0.7478	0.1154	0.7402	**0.1481**	0.7229	0.0588
Subdural hematoma	0.0147	0.8270	**0.1449**	0.8884	0.1429	0.8190	0.0476
Near drowning	0.0068	0.8296	0.0741	0.7917	**0.1404**	0.6897	0.0397
Cardiac arrest, outside hospital	0.0118	0.8932	0.0976	0.8791	**0.1379**	0.8881	0.0556
Pleural effusion	0.0113	0.8549	0.1081	0.8186	**0.1351**	0.7605	0.1151
Bronchopulmonary dysplasia	0.0252	0.8309	**0.1915**	0.7952	0.1304	0.8503	0.1203
Hyponatremia	0.0056	0.5707	0.0187	0.7398	**0.1176**	0.8775	0.0000
Suspected septicemia, rule out	0.0143	0.7378	0.0923	0.7402	**0.1029**	0.6769	0.0000
Thrombocytopenia	0.0112	0.7381	0.0822	0.7857	**0.1026**	0.8585	0.0800
(Benign) intracranial hypertension	0.0099	0.8494	0.0000	0.9018	0.1020	0.8586	**0.1224**
Pericardial effusion	0.0055	0.8997	0.0870	0.9085	**0.1017**	0.9000	0.0714
Pulmonary contusion	0.0068	0.9029	0.0606	0.8831	**0.0984**	0.8197	0.0225
Surgery, gastrointestinal	0.0104	0.6705	0.0714	0.6666	**0.0976**	0.5545	0.0233
Respiratory Arrest	0.0062	0.8404	0.0000	0.8741	**0.0952**	0.8127	0.0444
Trauma, abdominal	0.0105	0.7426	**0.1667**	0.8623	0.0930	0.6991	0.0426
Atrial septal defect	0.0107	0.7766	0.0727	0.7765	**0.0909**	0.7197	0.0000
Genetic abnormality	0.0557	0.6629	**0.1324**	0.6470	0.0876	0.5705	0.1165
Arrhythmia, ventricular	0.0062	0.8532	0.0303	0.8703	0.0870	0.8182	**0.1250**
Hematologic disorder, other	0.0114	0.6736	0.0800	0.6898	**0.0870**	0.8074	0.0800
Asthma, stable	0.0171	0.7010	**0.0925**	0.6607	0.0870	0.5907	0.0741
Neurofibromatosis	0.0079	0.8022	0.0469	0.7984	**0.0816**	0.7388	0.0160
Tumor, bone	0.0090	0.8830	0.0727	0.8174	**0.0800**	0.7649	0.0417
Shock, hypovolemic	0.0088	0.7703	0.0000	0.8433	**0.0741**	0.8040	0.0000
Gastrointestinal bleed, other	0.0064	0.8325	0.0541	0.7974	0.0741	0.7996	**0.0909**

Table 4.4: AUC and F1 scores for individual diagnostic codes, continuing.

Classifier Performance on Each Diagnostic Code, Sorted by F1

		Msmt.		**Msmt. + indic.**		**Indic.**	
Condition	Base rate	AUC	F1	AUC	**F1**	AUC	F1
Chromosomal abnormality	0.0173	0.8047	0.1034	0.7197	0.0714	0.6300	**0.1600**
Encephalopathy, other	0.0093	0.8265	0.1250	0.8736	0.0688	0.8335	**0.1250**
Respiratory syncytial virus	0.0130	0.8876	**0.2857**	0.9145	0.0645	0.8716	0.0930
(Hereditary) hemolytic anemia, other	0.0088	0.7582	0.0548	0.8544	**0.0645**	0.9125	0.0513
Obstructive sleep apnea	0.0185	0.7564	0.0613	0.8200	0.0645	0.8087	**0.1111**
Apnea, central	0.0142	0.7871	**0.1600**	0.8134	0.0625	0.8051	0.0000
Neuromuscular, other	0.0132	0.7163	0.0452	0.7069	**0.0619**	0.6484	0.0392
Anemia, acquired	0.0056	0.7378	**0.1017**	0.7596	0.0615	0.8129	0.0519
Meningitis, bacterial	0.0070	0.4431	0.0000	0.7676	**0.0606**	0.5480	0.0000
Trauma, long bone injury	0.0096	0.8757	0.0952	0.9085	0.0597	0.7946	**0.1176**
Bowel (intestinal) obstruction	0.0104	0.7512	**0.0984**	0.6559	0.0597	0.6936	0.0424
Neurologic disorder, other	0.0288	0.7628	**0.1481**	0.6978	0.0588	0.5971	0.0769
Panhypopituitarism	0.0057	0.7763	0.0000	0.7724	**0.0571**	0.6415	0.0000
Thyroid dysfunction	0.0072	0.6310	0.0369	0.6420	**0.0541**	0.6661	0.0000
Coma	0.0056	0.6483	**0.1250**	0.6823	0.0513	0.7155	0.0000
Spinal cord lesion	0.0133	0.7298	**0.0585**	0.7052	0.0488	0.8168	0.0414
Pneumonia, other (mycoplasma)	0.0188	0.8589	**0.1613**	0.8792	0.0476	0.8424	0.1164
Trauma, blunt	0.0065	0.9156	**0.0513**	0.8138	0.0469	0.7426	0.0177
Surgery, thoracic	0.0058	0.7405	0.0000	0.6948	0.0469	0.6087	**0.0909**
Neuroblastoma	0.0059	0.6526	0.0306	0.7268	**0.0360**	0.7775	0.0346
Obesity	0.0098	0.7503	0.0365	0.6814	0.0351	0.6647	**0.0667**
Obstructed ventriculoperitoneal shunt	0.0073	0.6824	0.0267	0.7114	0.0331	0.7516	**0.0667**
Ventricular septal defect	0.0119	0.6641	**0.1081**	0.5680	0.0294	0.5593	0.0444
Croup Syndrome, UAO	0.0069	0.9418	0.2222	0.9834	0.0000	0.9682	**0.2222**
Sickle-cell anemia, unspecified	0.0080	0.6262	0.0000	0.9627	0.0000	0.8661	**0.1250**
Biliary atresia	0.0063	0.9383	**0.2667**	0.9164	0.0000	0.7589	0.0714
Metabolic acidosis (¡7.1)	0.0083	0.9475	**0.1818**	0.9046	0.0000	0.9143	0.1538
Immunologic disorder, other	0.0094	0.9539	**0.1500**	0.8868	0.0000	0.8969	0.1212
Pulmonary hypertension, other	0.0112	0.9259	**0.2500**	0.8826	0.0000	0.8098	0.0000
Trauma, chest	0.0051	0.9261	0.0000	0.8818	0.0000	0.7820	0.0000
Spinal muscular atrophy	0.0052	0.9666	0.0000	0.8658	0.0000	0.8362	0.0000
Trauma, unspecified	0.0065	0.7153	**0.1481**	0.8657	0.0000	0.8224	0.0594
Bone marrow transplant, status post	0.0097	0.8161	**0.5217**	0.8562	0.0000	0.8505	0.1695
Surgery, orthopaedic	0.0180	0.7839	**0.1029**	0.8192	0.0000	0.7331	0.0000
Gastrointestinal bleed, upper	0.0063	0.8388	0.0000	0.8078	0.0000	0.7256	0.0000
Arrhythmia, supraventricular tachy.	0.0055	0.8178	**0.0385**	0.7867	0.0000	0.8199	0.0000
Congenital central alveolar hypovent.	0.0057	0.7067	0.0000	0.7716	0.0000	0.7282	0.0000
Tetralogy of fallot	0.0061	0.5759	0.0000	0.7614	0.0000	0.7637	0.0000
Cardiac disorder, other	0.0071	0.7229	**0.0519**	0.7552	0.0000	0.6287	0.0000
Hydrocephalus, shunt failure	0.0083	0.7715	0.0000	0.7542	0.0000	0.7986	**0.0635**
Cerebral infarction (CVA)	0.0058	0.6766	0.0000	0.7495	0.0000	0.7148	**0.1333**
Congenital heart disorder, other	0.0084	0.7590	0.0000	0.7277	0.0000	0.7803	**0.0583**
Gastrointestinal disorder, other	0.0139	0.6755	0.0336	0.6821	0.0000	0.6465	**0.1026**
Aspiration	0.0072	0.6727	**0.0533**	0.6734	0.0000	0.6792	0.0333
Dehydration	0.0105	0.7356	**0.0690**	0.6636	0.0000	0.5899	0.0000
Tumor, thoracic	0.0077	0.6931	**0.0513**	0.6249	0.0000	0.6815	0.0292
UAO, extubation, status post	0.0085	0.8295	**0.0672**	0.6063	0.0000	0.6128	0.0000

Table 4.5: Sampling rates and missingness statistics for all 13 features.

Variable	**Msmt./hour**	**Missing entirely**	**Frac. missing**
Diabstolic blood pressure	0.5162	0.0135	0.1571
Systolic blood pressure	0.5158	0.0135	0.1569
Peripheral capillary refall rate	1.0419	0.0140	0.5250
End-tidal CO_2	0.9318	0.5710	0.5727
Fraction inspired O_2	1.3004	0.1545	0.7873
Total glasgow coma scale	1.0394	0.0149	0.5250
Glucose	1.4359	0.1323	0.9265
Heart rate	0.2477	0.0133	0.0329
pH	1.4580	0.3053	0.9384
Respiratory rate	0.2523	0.0147	0.0465
Pulse oximetry	0.1937	0.0022	0.0326
Temperature	1.0210	0.0137	0.5235
Urine output	1.1160	0.0353	0.5980

Chapter 5

Evaluating Multi-label Classifiers

The previous two chapters address multilabel classification of diagnoses. How to best evaluate multilabel classifiers remains an open question and can depend on the application of interest. This chapter provides new insight into maximizing F1 measures in the context of binary classification and also in the context of multilabel classification. The harmonic mean of precision and recall, the F1 measure is widely used to evaluate the success of a binary classifier when one class is rare. Micro average, macro average, and per instance average F1 measures are used in multilabel classification. For any classifier that produces a real-valued output, we derive the relationship between the best achievable F1 value and the decision-making threshold that achieves this optimum. As a special case, if the classifier outputs are well-calibrated conditional probabilities, then the optimal threshold is half the optimal F1 value. As another special case, if the classifier is completely uninformative, then the optimal behavior is to classify all examples as positive. When the actual prevalence of positive examples is low, this behavior can be undesirable. As a case study, we discuss the results, which can be surprising, of maximizing F1 when predicting 26,853 labels for Medline documents.

5.1 Introduction

Performance measures are useful for comparing the quality of predictions across systems. Some commonly used measures for binary classification are accuracy, precision, recall, F1 score, and Jaccard index [SL09]. Multilabel classification is an extension of binary classification that is currently an area of active research in supervised machine learning [Tso07]. Micro averaging, macro averaging, and per instance averaging are three commonly used variations of F1 measure used in the multilabel setting. In general, macro averaging increases the impact on final score of performance on rare labels, while per instance averaging increases the importance of performing well on each example [Tan05]. In this chapter, we present theoretical and experimental results on the properties of the F1 measure. For concreteness, the results are given specifically for the F1 measure and its multilabel variants. However, the results can be generalized to Fβ measures for $\beta \neq 1$.

Two approaches exist for optimizing performance on the F1 measure. Structured loss minimization incorporates the performance measure into the loss function and then optimizes during training. In contrast, plug-in rules convert the numerical outputs of classifiers into optimal predictions [DKJ+13]. In this chapter, we highlight the latter scenario, and we differentiate between the beliefs of a system and predictions selected to optimize alternative measures. In the multilabel case, we show that the same beliefs can produce markedly dissimilar optimally thresholded predictions depending upon the choice of averaging method.

It is well-known that F1 is asymmetric in the positive and negative class. Given complemented predictions and complemented true labels, the F1 measure is in general different. It also generally known that micro F1 is affected less by performance on rare labels, while macro F1 weighs the F1 achieved on each label equally [MRS08]. In this chapter, we show how these properties are manifest in the optimal threshold for making decisions, and we present results that characterize that threshold. Additionally, we demonstrate that given an uninformative classifier, optimal thresholding to maximize F1 predicts all instances positive regardless of the base rate.

While F1 measures are widely used, some of their properties are not widely recognized. In particular, when choosing predictions to maximize the expected F1 measure for a set of examples, each prediction depends not only on the probability that the label applies to that example, but also on the distribution of probabilities *for all other examples in the set*. We quantify this dependence in Theorem 2, where we derive an expression for optimal thresholds. The dependence makes it difficult to relate predictions that are optimally thresholded for F1 to a system's predicted conditional probabilities.

We show that the difference in F1 measure between perfect predictions and optimally thresholded random guesses depends strongly on the base rate. As a consequence, macro average F1 can be argued not to treat labels equally, but to give greater emphasis to performance on rare labels. In a case study, we consider learning to tag articles in the biomedical literature with MeSH terms, a controlled vocabulary of 26,853 labels. These labels have heterogeneously distributed base rates. Our results imply that if the predictive features for rare labels are lost (because of feature selection or from another cause) then the optimal thresholds to maximize macro F1 lead to predicting these rare labels frequently. For the case study application, and likely for similar ones, this behavior is undesirable.

5.2 Definitions of Performance Measures

Consider binary class prediction in the single or multilabel setting. Given training data of the form $\{\langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle\}$ where each x_i is a feature vector of dimension

	Actual Positive	Actual Negative
Predicted Positive	tp	fp
Predicted Negative	fn	tn

Figure 5.1: Confusion Matrix

d and each y_i is a binary vector of true labels of dimension m, a probabilistic classifier outputs a model that specifies the conditional probability of each label applying to each instance given the feature vector. For a batch of data of dimension $n \times d$, the model outputs an $n \times m$ matrix C of probabilities. In the single-label setting, $m = 1$ and C is an $n \times 1$ matrix, i.e. a column vector.

A decision rule $D(C) : \mathbb{R}^{n\times m} \to \{0,1\}^{n\times m}$ converts a matrix of probabilities C to binary predictions P. The gold standard $G \in \{0,1\}^{n\times m}$ represents the true values of all labels for all instances in a given batch. A performance measure M assigns a score to a prediction given a gold standard:

$$M(P,G) : \{0,1\}^{n\times m} \times \{0,1\}^{n\times m} \to \mathbb{R} \in [0,1].$$

The counts of true positives tp, false positives fp, false negatives fn, and true negatives tn are represented via a confusion matrix (Figure 5.1).

Precision $p = tp/(tp + fp)$ is the fraction of all positive predictions that are actual positives, while recall $r = tp/(tp + fn)$ is the fraction of all actual positives that are predicted to be positive. By definition, the F1 measure is the harmonic mean of precision and recall: $F1 = 2/(1/r + 1/p)$. By substitution, F1 can be expressed as a function of counts of true positives, false positives and false negatives:

$$F1 = \frac{2tp}{2tp + fp + fn}. \tag{5.1}$$

The harmonic mean expression for F1 is undefined when $tp = 0$, but the alternative

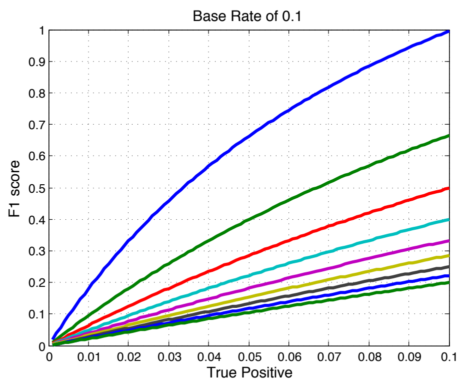


Figure 5.2: Holding base rate and fp constant, F1 is concave in tp. Each line is a different value of fp.

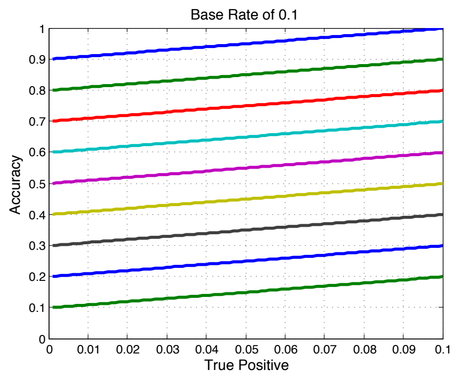


Figure 5.3: Unlike F1, accuracy offers linearly increasing returns. Each line is a fixed value of fp.

expression is undefined only when $tn = n$. This difference does not impact the results below.

Before explaining optimal thresholding to maximize F1, we first discuss some properties of F1. For any fixed number of actual positives in the gold standard, only two of the four entries in the confusion matrix (Figure 5.1) vary independently. This is because the number of actual positives is equal to the sum $tp + fn$ while the number of actual negatives is equal to the sum $tn + fp$. A second basic property of F1 is that it is nonlinear in its inputs. Specifically, fixing the number fp, F1 is concave as a function of tp (Figure 5.2). By contrast, accuracy is a linear function of tp and tn (Figure 5.3).

As mentioned in the introduction, F1 is asymmetric. By this, we mean that the

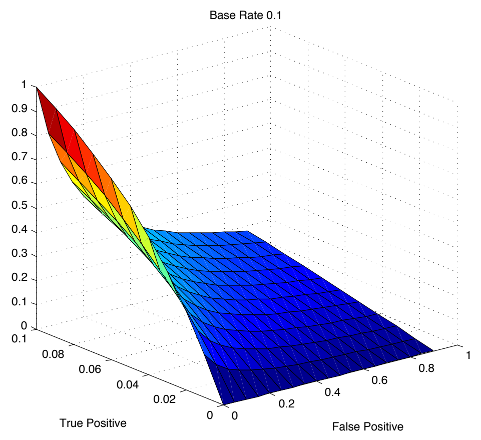


Figure 5.4: Given a fixed base rate, the F1 measure is a nonlinear function with two degrees of freedom.

score assigned to a prediction P given gold standard G can be arbitrarily different from the score assigned to a complementary prediction P^c given complementary gold standard G^c. This can be seen by comparing Figure 5.2 with Figure 5.5. The asymmetry is problematic when both false positives and false negatives are costly. For example, F1 has been used to evaluate the classification of tumors as benign or malignant [Aka09], a domain where both false positives and false negatives have considerable costs.

While F1 was developed for single-label information retrieval, as mentioned there are variants of F1 for the multilabel setting. Micro F1 treats all predictions on all labels as one vector and then calculates the F1 score. Specifically,

$$tp = 2\sum_{i=1}^{n}\sum_{j=1}^{m} \mathbb{1}(P_{ij}=1)\mathbb{1}(G_{ij}=1).$$

We define fp and fn analogously and calculate the final score using (5.1). Macro F1, which can also be called per label F1, calculates the F1 for each of the m labels and averages them:

$$F1_{Macro}(P,G) = \frac{1}{m}\sum_{j=1}^{m} F1(P_{:j}, G_{:j}).$$

The per instance F1 measure is similar, but averages F1 over all n examples:

$$F1_{Instance}(P,G) = \frac{1}{n}\sum_{i=1}^{n} F1(P_{i:}, G_{i:}).$$

Accuracy is the fraction of all instances that are predicted correctly:

$$Acc = \frac{tp+tn}{tp+tn+fp+fn}.$$

Accuracy is adapted to the multilabel setting by summing tp and tn for all labels and then dividing by the total number of predictions:

$$Acc(P,G) = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} \mathbb{1}(P_{ij} = G_{ij}).$$

The Jaccard Index, a monotonically increasing function of F1, is the cardinality of the intersection of the predicted positive set and the actual positive set divided by the cardinality of their union:

$$Jaccard = \frac{tp}{tp+fn+fp}.$$

5.3 Optimal Decision Rule for F1 Maximization

In this section, we provide a characterization of the decision rule that maximizes the F1 measure, and, for a special case, we present a relationship between the optimal threshold and the maximum achievable F1 value.

We assume that the classifier outputs real-valued scores s and that there exist two distributions $p(s|t=1)$ and $p(s|t=0)$ that are the conditional probability of seeing the score s when the true label t is 1 or 0, respectively. We assume that these distributions are known in this section; the next section discusses an empirical version of the result.

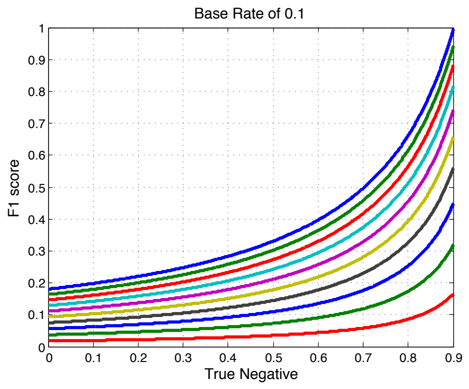


Figure 5.5: F1 score for fixed base rate and number fn of false negatives. F1 offers increasing marginal returns as a function of tn. Each line is a fixed value of fn.

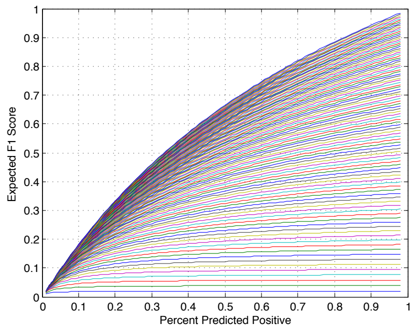


Figure 5.6: The expected F1 score of an optimally thresholded random guess is highly dependent on the base rate.

Note also that in this section tp etc. are fractions that sum to one, not counts.

Given $p(s|t=1)$ and $p(s|t=0)$, we seek a decision rule $D : s \rightarrow \{0,1\}$ mapping scores to class labels such that the resulting classifier maximizes F1. We start with a lemma that is valid for any D.

Lemma 1. The true positive rate $tp = b\int_{s:D(s)=1} p(s|t=1)ds$ where the base rate is $b = p(t=1)$.

Proof. Clearly $tp = \int_{s:D(s)=1} p(t=1|s)p(s)ds$. Bayes rule says that $p(t=1|s) = p(s|t=1)p(t=1)/p(s)$. Hence $tp = b\int_{s:D(s)=1} p(s|t=1)ds$. $\square$

Using three similar lemmas, the entries of the confusion matrix are

$$\begin{aligned}
tp &= b \int_{s:D(s)=1} p(s|t=1)ds \\
fn &= b \int_{s:D(s)=0} p(s|t=1)ds \\
fp &= (1-b) \int_{s:D(s)=1} p(s|t=0)ds \\
tn &= (1-b) \int_{s:D(s)=0} p(s|t=0)ds.
\end{aligned}$$

The following theorem describes the optimal decision rule that maximizes F1.

Theorem 2. *An example with score s is assigned to the positive class, that is $D(s) = 1$, by a classifier that maximizes F1 if and only if*

$$\frac{b \cdot p(s|t=1)}{(1-b) \cdot p(s|t=0)} \geq J \tag{5.2}$$

where $J = tp/(fn+tp+fp)$ is the Jaccard index of the optimal classifier, with ambiguity given equality in (5.2).

Before we provide the proof of this theorem, we note the difference between the rule in (5.2) and conventional cost-sensitive decision making [Elk01] or Neyman-Pearson detection. In both the latter approaches, the right hand side J is replaced by a constant λ that depends only on the costs of $0-1$ and $1-0$ classification errors, and not on the performance of the classifier on the entire batch. We will later describe how the relationship can lead to undesirable thresholding behavior for applications in the multilabel setting.

Proof. Divide the domain of s into regions of fixed size. Suppose that the decision rule $D(\cdot)$ has been fixed for all regions except a particular region denoted Δ around a point s. Write $P_1(\Delta) = \int_\Delta p(s|t=1)ds$ and define $P_0(\Delta)$ similarly.

Suppose that the F1 achieved with decision rule D for all scores besides those in Δ is $F1 = 2tp/(2tp+fn+fp)$. Now, if we add Δ to the positive region of the decision rule, $D(\Delta) = 1$, then the new F1 score is

$$F1' = \frac{2tp + 2bP_1(\Delta)}{2tp + 2bP_1(\Delta) + fn + fp + (1-b)P_0(\Delta)}.$$

On the other hand, if we add Δ to the negative region of the decision rule, $D(\Delta) = 0$, then the new F1 score is

$$F1'' = \frac{2tp}{2tp + fn + bP_1(\Delta) + fp}.$$

We add Δ to the positive region only if $F1' \geq F1''$. With some algebraic simplification, this condition becomes

$$\frac{bP_1(\Delta)}{(1-b)P_0(\Delta)} \geq \frac{tp}{tp + fn + fp}.$$

Taking the limit $|\Delta| \to 0$ gives the claimed result. $\square$

If, as a special case, the model outputs calibrated probabilities, that is $p(t = 1|s) = s$ and $p(t = 0|s) = 1 - s$, then we have the following corollary.

Corollary 3. An instance with predicted probability s is assigned to the positive class by the decision rule that maximizes F1 if and only if $s \geq F/2$ where the F1 score achieved by this optimal decision rule is $F = 2tp/(2tp+fn+fp)$.

Proof. Using the definition of calibration and then Bayes rule, for the optimal decision surface for assigning a score s to the positive class

$$\frac{p(t=1|s)}{p(t=0|s)} = \frac{s}{1-s} = \frac{p(s|t=1)b}{p(s|t=0)(1-b)}. \tag{5.3}$$

Incorporating (5.3) in Theorem 2 gives

$$\frac{s}{1-s} \geq \frac{tp}{fn+tp+fp}$$

and simplifying results in

$$s \geq \frac{tp}{2tp+fn+fp} = F/2.$$

$\square$

Thus, the optimal threshold in the calibrated case is half the maximum F1 value.

Above, we assume that scores have a distribution conditioned on the true class. Using the intuition in the proof of Theorem 2, we can also derive an empirical version of the result. To save space, we provide a more general version of the empirical result in the next section for multilabel problems, noting that a similar non-probabilistic statement holds for the single label setting as well.

5.4 Consequences of the Optimal Decision Rule

We demonstrate two consequences of designing classifiers that maximize the F1 measure, which we call the batch observation and the uninformative classifier observation. We will later show with a case study that these can combine to produce surprising and potentially undesirable predictions when macro F1 is optimized in practice.

The batch observation is that a label may or may not be predicted for an instance depending on the distribution of conditional probabilities (or scores) for other instances in the same batch. Earlier, we observed a relationship between the optimal threshold and the maximum achievable F1 value, and demonstrated that this maximum depends on the distribution of conditional probabilities for all instances. Therefore, depending upon the set in which an instance is placed, its conditional probability may or may not exceed the optimal threshold. Note that because an F1 value cannot exceed 1, the optimal threshold cannot exceed 0.5.

Consider for example an instance with conditional probability 0.1. It will be classified as positive if it has the highest probability of all instances in a batch. However, in a different batch, where the probabilities predicted for all other instances are 0.5 and n is large, the maximum achievable F1 score is close to 2/3. According to the results above, we will then classify this last instance as negative because it has a conditional probability less than 1/3.

An uninformative classifier is one that predicts the same score for all examples. If these scores are calibrated conditional probabilities, then the base rate is predicted for every example.

Theorem 4. *Given an uninformative classifier for a label, optimal thresholding to maximize expected F1 results in classifying all examples as positive.*

Proof. Given an uninformative classifier, we seek the threshold that maximizes $\mathbb{E}(F1)$. The only choice is how many labels to predict. By symmetry between the instances, it does not matter which instances are labeled positive.

Let $a = tp + fn$ be the number of actual positives and let $c = tp + fp$ be a fixed number of positive predictions. The denominator of the expression for F1 in Equation (5.1), that is $2tp + fp + fn = a + c$, is then constant. The number of true positives, however, is a random variable. Its expected value is equal to the sum of the probabilities that each example predicted positive actually is positive:

$$\mathbb{E}(F1) = \frac{2\sum_{i=1}^{c} b}{a+c} = \frac{2c \cdot b}{a+c}$$

where $b = a/n$ is the base rate. To maximize this expectation as a function of c, we calculate the partial derivative with respect to c, applying the product rule:

$$\frac{\partial}{\partial c}\mathbb{E}(F1) \quad = \quad \frac{\partial}{\partial c}\frac{2c\cdot b}{a+c} = \frac{2b}{a+c} - \frac{2c\cdot b}{(a+c)^2}.$$

Both terms in the difference are always positive, so we can show that the derivative is always positive by showing that

$$\frac{2b}{a+c} > \frac{2c\cdot b}{(a+c)^2}.$$

Simplification gives the condition $1 > c/(a+c)$. As this condition always holds, the derivative is always positive. Therefore, whenever the frequency of actual positives in the test set is nonzero, and the predictive model is uninformative, then expected F1 is maximized by predicting that all examples are positive. □

Figure 5.6 shows that for small base rates, an optimally thresholded uninformative classifier achieves $\mathbb{E}(F1)$ close to 0, while for high base rates $\mathbb{E}(F1)$ is close to 1. We revisit this point in the next section in the context of maximizing macro F1.

5.5 Multilabel Setting

Different measures are used to measure different aspects of a system's performance. However, changing the measure that is optimized can change the optimal predictions. We relate the batch observation to discrepancies between predictions that are optimal for micro versus macro averaged F1. We show that while performance on rare labels is unimportant for micro F1, macro F1 is dominated by performance on these labels. Additionally, we show that macro averaging F1 can conceal the occurrence of uninformative classifier thresholding.

Consider the equation for micro averaged F1, for m labels with base rates b_i.

Suppose that tp, fp, and fn are fixed for the first $m-1$ labels, and suppose that b_m is small compared to the other b_i. Consider (i) a perfect classifier for label m, (ii) a trivial classifier that never predicts label m, and (iii) a trivial classifier that predicts label m for every example. The perfect classifier increases tp by a small amount $b_m \cdot n$, the number of actual positives for the rare label m, while contributing nothing to the counts fp and fn:

$$F1' = \frac{2(tp + b_m \cdot n)}{2(tp + b_m \cdot n) + fp + fn}.$$

The trivial classifier that never predicts label m increases fn by the same small amount:

$$F1'' = \frac{2tp}{2tp + fp + (fn + b_m \cdot n)}.$$

Finally, the trivial classifier that predicts label m for every example increases fp by a large amount $n(1-b_m)$. Clearly this last classifier leads to micro average F1 that is much worse than that of the perfect classifier for label m. However, $F1'$ and $F1''$ both tend to the same value, namely $2tp/(2tp + fp + fn)$, as b_m tends to zero. Hence, for a label with very small base rate, a perfect classifier does not improve micro F1 noticeably compared to a trivial all-negative classifier. It is fair to say that performance on rare labels is unimportant for micro F1.

Now consider the context of macro F1, where separately calculated F1 scores over all labels are averaged. Consider the two label case where one label has a base rate of 0.5 and the other has a base rate of 0.1. The corresponding F1 measures for trivial all-positive classifiers are 0.67 and 0.18 respectively. Thus the macro F1 for trivial classifiers is 0.42. An improvement to perfect predictions on the rare label increases macro F1 to 0.83, while the same improvement on the common label only increases macro F1 of 0.59. Hence it is fair to say that macro F1 emphasizes performance on rare labels, even though it weights performance on every label equally.

For a rare label with an uninformative predictive model, micro F1 is optimized by classifying all examples as negative, while macro F1 is optimized by classifying all examples as positive. Optimizing micro F1 as compared to macro F1 can be thought of as choosing optimal thresholds given very different batches. If the base rates and distributions of conditional probabilities predicted for instances vary from label to label, so will the optimal binary predictions. Generally, labels with small base rates and less informative classifiers will be over-predicted when maximizing macro F1, and under-predicted when maximizing micro F1. We present empirical evidence of this phenomenon in the following case study.

5.6 Case Study

This section discusses a case study that demonstrates how in practice, thresholding to maximize macro F1 can produce undesirable predictions. To our knowledge, a similar real-world case of pathological behavior has not been previously described in the literature, even though macro averaging F1 is a common approach.

We consider the task of assigning tags from a controlled vocabulary of 26,853 MeSH terms to articles in the biomedical literature based on their titles and abstracts. We represent each abstract as a sparse bag-of-words vector over a vocabulary of 188,923 words. The training data consists of a matrix A with n rows and d columns, where n is the number of abstracts and d is the number of features in the bag of words representation. We apply a tf-idf text preprocessing step to the bag of words representation to account for word burstiness [MKE05] and to elevate the impact of rare words.

Because linear regression models can be trained for multiple labels efficiently, we choose linear regression as a predictive model. Note that square loss is a proper loss function and yields calibrated probabilistic predictions [MJV$^+$12]. Further, to increase

Table 5.1: Selected frequently predicted MeSH terms. When F1 is optimized separately for each term, low thresholds are chosen for rare labels (bold) with uninformative classifiers.

MeSH term	count	maximum F1	threshold
Humans	2346	0.9160	0.458
Male	1472	0.8055	0.403
Female	1439	0.8131	0.407
Phosphinic Acids	**1401**	$1.544 \cdot 10^{-4}$	$7.71 \cdot 10^{-5}$
Penicillanic Acid	**1064**	$8.534 \cdot 10^{-4}$	$4.27 \cdot 10^{-4}$
Adult	1063	0.7004	0.350
Middle Aged	1028	0.7513	0.376
Platypus	**980**	$4.676 \cdot 10^{-4}$	$2.34 \cdot 10^{-4}$

the speed of training and prevent overfitting, we approximate the training matrix A by a rank restricted A_k using singular value decomposition. One potential consequence of this rank restriction is that the signal of extremely rare words can be lost. This can be problematic when rare terms are the only features of predictive value for a label.

Given the probabilistic output of each classifier and the results relating optimal thresholds to maximum attainable F1, we designed three different plug-in rules to maximize micro, macro and per instance average F1. Inspection of the predictions to maximize micro F1 revealed no irregularities. However, inspecting the predictions thresholded to maximize performance on macro F1 showed that several terms with very low base rates were predicted for more than a third of all test documents. Among these terms were "Platypus", "Penicillanic Acids" and "Phosphinic Acids" (Figure 5.1).

In multilabel classification, a label can have low base rate and an uninformative classifier. In this case, optimal thresholding requires the system to predict all examples positive for this label. In the single-label case, such a system would achieve a low F1 and not be used. But in the macro averaging multilabel case, the extreme thresholding behavior can take place on a subset of labels, while the system manages to perform well overall.

5.7 A Winner's Curse

In practice, decision rules that maximize F1 are often set empirically, rather than analytically. That is, given a set of validation examples with predicted scores and true labels, rules for mapping scores to labels are selected that maximize F1 on the validation set. In such situations, the optimal threshold can be subject to a winner's curse [CCC71] where a sub-optimal threshold is chosen because of sampling effects or limited training data. As a result, the future performance of a classifier using this threshold is worse than the anticipated performance. We show that threshold optimization for F1 is particularly susceptible to this phenomenon.

In particular, different thresholds have different rates of convergence of empirical F1 with number of samples n. As a result, for a given n, comparing the empirical performance of low and high thresholds can result in suboptimal performance. This is because, for a fixed number of samples, some thresholds converge to their true error rates fast, while others have higher variance and may be set erroneously. We demonstrate these ideas for a scenario with an uninformative model, though they hold more generally.

Consider an uninformative model, for a label with base rate b. The model is uninformative in the sense that output scores are $s_i = b + n_i$ for examples i, where $n_i = \mathcal{N}(0, \sigma^2)$. Thus, scores are uncorrelated with and independent of the true labels. The empirical accuracy for a threshold t is

$$A^t_{exp} = \frac{1}{n}\sum_{i\in+}\mathbf{1}[s_i \geq t] + \frac{1}{n}\sum_{i\in-}\mathbf{1}[s_i \leq t] \tag{5.4}$$

where $+$ and $-$ index the positive and negative class respectively. Each term in Equation (5.4) is the sum of $O(n)$ i.i.d random variables and has exponential (in n) rate of convergence to the mean irrespective of the base rate b and the threshold t. Thus, for a fixed number T of threshold choices, the probability of choosing the wrong threshold

$P_{err} \leq T2^{-\varepsilon n}$ where ε depends on the distance between the optimal and next nearest threshold. Even if errors occur, the most likely errors are thresholds close to the true optimal threshold, a consequence of Sanov's theorem [CT12].

Consider how to select an F1-maximizing threshold empirically, given a validation set of ground truth labels and scores from an uninformative classifier. The scores s_i can be sorted in decreasing order (w.l.o.g.) since they are independent of the true labels for an uninformative classifier. Based on the sorted scores, we empirically select the threshold that maximizes the F1 measure F on the validation set. The optimal empirical threshold will lie between two scores that include the value $F/2$, when the scores are calibrated, in accordance with Theorem 2.

The threshold s_{min} that classifies all examples positive (and maximizes F1 analytically by Theorem 4) has an empirical F1 close to its expectation of $\frac{2b}{1+b} = \frac{2}{1+1/b}$ since tp, fp and fn are all estimated from the entire data. Consider a threshold s that classifies only the first example positive and all others negative. With probability b, this has F1 value $2/(2+b \cdot n)$, which is worse than that of the optimal threshold only when

$$b \geq \frac{\sqrt{1+\frac{8}{n}}-1}{2}.$$

Despite the threshold s being far from optimal, it has a constant probability of having a better F1 score on validation data, a probability that does not decrease with n, for $n < (1-b)/b^2$. Therefore, optimizing F1 will have a sharp threshold behavior, where for $n < (1-b)/b^2$ the algorithm will incorrectly select large thresholds with constant probability, whereas for larger n it will correctly identify small thresholds. Note that identifying optimal thresholds for F1 is still problematic since it then leads to issues identified in the previous section. While these issues are distinct, they both arise from the nonlinearity of the F1 measure and its asymmetric treatment of positive and negative

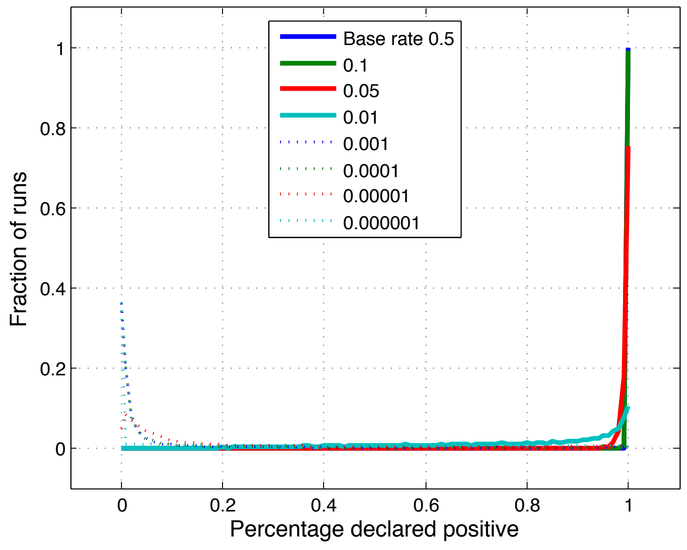


Figure 5.7: The distribution of experimentally chosen thresholds changes with varying base rate b. For small b, a small fraction of examples are predicted positive even though the optimal thresholding is to predict all positive.

labels.

Figure 5.7 shows the result of simulating this phenomenon, executing 10,000 runs for each setting of the base rate, with $n = 10^6$ samples for each run used to set the threshold. Scores are chosen using variance $\sigma^2 = 1$. True labels are assigned at the base rate, independent of the scores. The threshold that maximizes F1 on the validation set is selected. We plot a histogram of the fraction predicted positive as a function of the empirically chosen threshold. There is a shift from predicting almost all positives to almost all negatives as the base rate is decreased. In particular, for low base rate, even with a large number of samples, a small fraction of examples are predicted positive. The analytically derived optimal decision in all cases is to predict all positive, i.e. to use a threshold of 0.

5.8 Related Work

Motivated by the widespread use of the F1 measure in information retrieval and in single and multilabel binary classification, researchers have published extensively on its optimization. The paper [Jan07] proposes an outer-inner maximization technique for F1 maximization, while [dCDB09] studies extensions to the multilabel setting, showing that simple threshold search strategies are sufficient when individual probabilistic classifiers are independent. Finally, [DWCH11] describe how the method of [Jan07] can be extended to efficiently label data points even when classifier outputs are dependent. More recent work in this direction can be found in [YCLC12]. However, none of this work directly identifies the relationship of the optimal threshold to the maximum achievable F1 score over all thresholds, as we do here.

While there has been work on applying general constrained optimization techniques to related measures [MDC$^+$01], research often focuses on specific classification methods. In particular, the paper [SMI06] studies F1 optimization for conditional random fields and [MKO$^+$03] discusses similar optimization for SVMs. In our work, we study the consequences of maximizing F1 for the general case of any classifier that outputs real-valued scores.

A result similar to a special case below, Corollary 1, was recently derived in [ZEPB13]. However, the derivation there is complex and does not prove the more general Theorem 2, which describes the optimal decision-making threshold even when the scores output by a classifier are not probabilities.

The batch observation is related to the note in [Lew95] that given a fixed classifier, a specific example may or may not cross the decision threshold, depending on the other examples present in the test data. However, the previous paper does not characterize what this threshold is, nor does it explain the differences between predictions made to optimize micro and macro average F1.

5.9 Discussion

In this chapter, we present theoretical and empirical results describing properties of the F1 performance measure for binary and multilabel classification. We relate the best achievable F1 score to the optimal decision-making threshold and show that when a classifier is uninformative, classifying all instances as positive maximizes F1. In the multilabel setting, this behavior is problematic when the measure to maximize is macro F1 and for some labels their predictive model is uninformative. In contrast, we demonstrate that given the same scenario, micro F1 is maximized by predicting those labels for all examples to be negative. This knowledge can be useful as such scenarios are likely to occur in settings with a large number of labels. We also demonstrate that micro F1 has the potentially undesirable property of washing out performance on rare labels.

No single performance measure can capture every desirable property. For example, for a single binary label, separately reporting precision and recall is more informative than reporting F1 alone. Sometimes, however, it is practically necessary to define a single performance measure to optimize. Evaluating competing systems and objectively choosing a winner presents such a scenario. In these cases, it is important to understand that a change of performance measure can have the consequence of dramatically altering optimal thresholding behavior.

5.10 Acknowledgments

Chapter 5, *Evaluating Multilabel Classifiers*, was written in collaboration with Charles Elkan and Balakrishnan ”Murali” Narayanaswamy. The dissertation author was

the primary investigator and author of this paper.

Chapter 6

Predicting Surgery Duration with Neural Heteroscedastic Regression

Scheduling surgeries is a challenging task due to the fundamental uncertainty of the clinical environment, as well as the risks and costs associated with under- and over-booking. We investigate neural regression algorithms to estimate the parameters of surgery case durations, focusing on the issue of *heteroscedasticity*. We seek to simultaneously estimate the duration of each surgery, as well as a surgery-specific notion of our *uncertainty* about its duration. Estimating this uncertainty can lead to more nuanced and effective scheduling strategies, as we are able to schedule surgeries more efficiently while allowing an informed and case-specific margin of error. Using surgery records from the UC San Diego Health System, we demonstrate potential improvements on the order of 20% (in terms of minutes overbooked) compared to current scheduling techniques. Moreover, we demonstrate that surgery durations are indeed heteroscedastic. We show that models that estimate case-specific uncertainty better fit the data (log likelihood). Additionally, we show that the heteroscedastic predictions can more optimally trade off between over and under-booking minutes, especially when idle minutes and scheduling collisions confer disparate costs.

6.1 Introduction

In the United States, healthcare is expensive and hospital resources are scarce. Healthcare expenditure now exceeds 17% of US GDP [Wor14], even as surgery wait times appear to have increased over the last decade [BKT$^+$11]. One source of inefficiency (among many) is the inability to fully utilize hospital resources. Because doctors cannot accurately predict the duration of surgeries, operating rooms can become congested (when surgeries run long) or lie vacant (when they run short). Over-booking can lead to long wait times and higher costs of labor (due to over-time pay), while under-booking decreases throughput, increasing the marginal cost per surgery.

At present, doctors book rooms according to a simple formula: The default time reserved is simply the mean duration of that specific procedure. The procedure code does in fact explain a significant amount of the variance in surgery durations. But by ignoring other signals, we hypothesize that the medical system leaves important signals untapped.

We address this issue by developing better and more nuanced strategies for surgery duration prediction. Our work focuses on a collection of surgery logs recorded in Electronic Health Records (EHRs) at a large United States health system. For each patient, we consider a collection of pre-operative features, including patient attributes (age, weight, height, sex, co-morbidities, etc.), as well as attributes of the clinical environment, such as the surgeon, surgery location, and time. For each procedure, we also know how much time was originally scheduled, in addition to the actual *'ground-truth'* surgery duration, recorded after each procedure is performed.

We are particularly interested in developing methods that allow us to better estimate the *uncertainty* associated with the duration of each surgery. Typically, neural network regression objectives assume *homoscedasticity*, i.e., constant levels of target variability for all instances. While mathematically convenient, this assumption is clearly violated in data such as ours, as one might surmise intuitively that operations that typically take a long time tend to exhibit greater variance than shorter ones. For example, among the 30 most common procedures, *epidural injections* are both the shortest procedures and the ones with the least variance (Figure 6.1). Among the same 30 procedures, *exploratory laparotomy* and *major burn surgery* exihibit the greatest variance. All procedures exhibit long (and one-sided) tails.

To model this data, we revisit the idea of heteroscedastic neural regression, combining it with expressive, dropout-regularized neural networks. In our approach, we jointly learn all parameters of a predictive distribution. In particular, we consider Gaussian and Laplace distributions, each of which is parameterized by a mean and standard deviation. We also consider Gamma distributions, which are especially suited to survival analysis. Unlike the Gaussian and Laplace which are long tailed on both ends, the gamma has a long right tail and has only positive support (i.e., it assigns zero probability density to any value less than zero). The restriction to positive values suits the modeling of durations or other survival-type data. While the gamma distribution (and the related Weibull distribution) has been applied to medical data with classical approaches [Ben83, SDAS97], this is, to our knowledge, the first to approximate the a parameters of a gamma distribution using modern neural network approaches.

Our heteroscedastic models better fit the data (as determined by log likelihood) compared to both current practice and neural network baselines that fail to account for heteroscedasticity. Furthermore, our models produce reliable estimates of the variance, which can be used to schedule intelligently, especially when over-booking and under-booking confer disparate costs. These uncertainty estimates come at no cost in performance by traditional measures. The best-performing Gamma MLP model achieves a

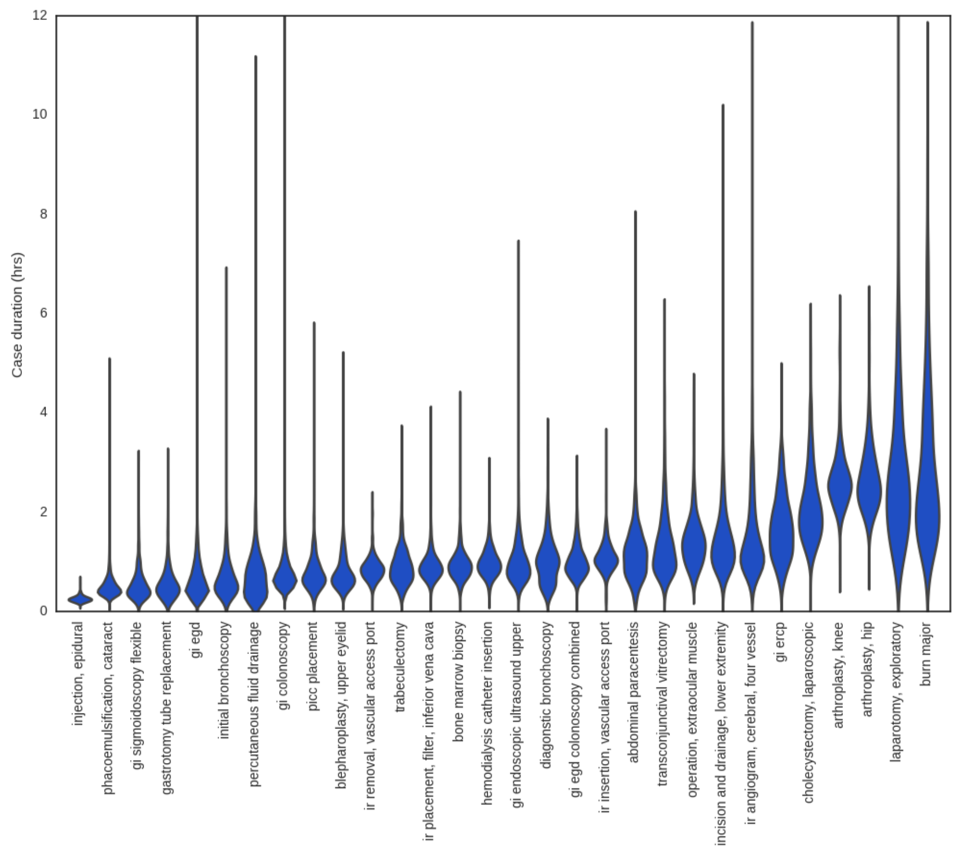


Figure 6.1: Distributions of durations for the 30 most common procedures.

lower mean squared error than a vanilla least squares (Gaussian) MLP, despite optimizing a different objective.

6.2 Dataset

Our dataset consists of patient records extracted from the EHR system of the University of California, San Diego Health System. Specifically, we selected 107,755 records corresponding to surgeries that took place between 2014 and 2016. These surgeries span 995 distinct procedures, and were performed by 368 distinct surgeons. Histograms of both are long-tailed, with over 796 procedures performed fewer than 100 times and 213 doctors performing fewer than 100 surgeries each. Moreover the data contains several clerical mistakes in logging the durations. For example, a number of surgeries in the record were reported as running less than 5 minutes. Discussions with the hospital experts suggest that this may indicate either clerical errors or an inconsistently applied convention for logging canceled surgeries. Additionally, several surgeries were reported to run over 24 hours, suggesting (rare) clerical errors in logging the end times of procedures. We remove all surgeries reported to take less than 5 minutes or more than 24 hours from the dataset. This preprocessing left us with roughly 80% of our original data (86,796 examples). For our experiments, we split this remaining data 80%/8%/12% for training/validation/testing.

6.2.1 Inputs

For each surgery, we extracted a number of pre-operative features from the corresponding EHRs. We restrict attention to features that are available for a majority of patients and (to avoid target leaks) exclude any information that is charted during or following the procedure. Our features fall into several categories: patient, doctor, procedure, and context.

Patient features: For each of our patients, we include the following features:

- **Size:** Patient height and weight are real-valued features. We normalize each to mean 0, variance 1.

- **Age:** A categorical variable, binned according to ten-year wide intervals that are open on the left side $(0-10], (10-20], \ldots$ None of the patients in our cohort are zero years old.

- **ASA score:** an ordinal score that represents the severity of a patient's illness. For example, ASA I denotes a healthy patient, ASA III denotes severe systemic disease, and ASA V denotes that the patient is moribund without surgery. ASA VI refers to a brain-dead patient in preparation for organ transplantation.

- **Anesthesia Type:** This categorical feature represents the type of anesthesia applied to sedate the patient. The values assigned to this variable include *General*, *Monitored anesthesia care (MAC)*—in which a patient undergoes local anesthesia together with sedation, *Neuraxial*, *No Anesthesiologist*, and *other/unknown*.
- **Patient Class:** This categorical feature indicates the patient's current status. The values assigned to this variable include *Emergency Department Encounter*, *Hospital Outpatient Procedure*, *Hospital Outpatient Surgery*, *Hospital Inpatient Surgery*, *Trauma Inpatient Admission*, *Inpatient Admission*, *Trauma Outpatient*.
- **Comorbidities:** We model the following co-morbidities as binary variables: smoker status, atrial fibrillation, chronic kidney disease, chronic obstructive pulmonary disease, congestive heart failure, coronary artery disease, diabetes, hypertension, cirrhosis, obstructive sleep apnea, cardiac device, dialysis, asthma, and dementia.
- **Doctor:** We represent the doctor performing the procedure (categorical) using a one-hot vector. The *doctor* feature exhibits considerable class imbalance, with the most prolific doctor performing 3770 surgeries and the least prolific doctor (in the pruned dataset) performing 100.
- **Procedure:** We represent the procedure performed as a one-hot vector. The most common operations tend to be minor GI procedures: the four most frequent procedures are *colonoscopy*, *upper GI endoscopy*, *cataract removal*, and *abdominal paracentesis*. This distribution is also long-tailed with 11,173 colonoscopies.

- **Context:** We represent the context of the procedure with several categorical variables. First, we represent the hour of the day as a categorical variable with values binned into 8 non-overlapping 3-hour width buckets. Second, we represent the day of the week and month of the year each as one-hot vectors. Finally, we similarly represent the location of the operations as a one-hot vector.

We summarize the number and kind of features in our dataset in Table 6.1. We handle variables with missing values, including height, weight, and hour of the day, by incorporating missing value indicators, following previous work on clinical datasets [LKW16b].

6.3 Methods

This chapter addresses the familiar task of regression. We start off by refreshing some basic preliminaries. Given a set of examples $\{x_i\}$, and corresponding labels $\{y_i\}$, we desire a model f that outputs a prediction $\hat{y} = f(x)$. The task of the machine learning algorithm is to produce the function f given a dataset $\mathcal{D}$ consisting of examples X and labels y. Generally, we seek predictions that are somehow *close* to y, as determined by some computable *loss function* $\mathcal{L}$. Most often we minimize the squared loss $\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2$ for all instances (x_i, y_i).

One popular method for producing such a function is to choose a class of functions f parameterized by some values θ. Linear models are the simplest examples of this approach. To train a linear regression model, we define $f(x) = \theta^T x$. Then we solve the following optimization problem:

$$\theta^* = \text{argmin}_\theta \mathcal{L}(y, \hat{y})$$

Table 6.1: Summary of features.

Feature	Abbreviation	Type	#Categories	Mode
Age	age	Categorical	9	50-60
Sex	sex	Binary	2	Female
Weight	weight	Numerical	-	-
Height	height	Numerical	-	-
Time of Day	hour	Categorical	8	9:00-12:00
Day of Week	day	Categorical	7	Friday
Month	month	Categorical	12	March
Location	location	Categorical	10	-
Patient Class	class	Categorical	7	Hospital Outpatient
ASA Rating	asa	Categorical	6	None
Anesthesia Type	anesthesia	Categorical	5	General
Surgeon	surgeon	Categorical	155	-
Procedure	procedure	Categorical	199	Colonoscopy
Smoker	smoker	Binary	2	No
Heart Arrhytmia	afib	Binary	2	No
Chronic Kidney Disease	ckd	Binary	2	No
Congestive Heart Failure	chf	Binary	2	No
Coronoary Artery Disease	cad	Binary	2	No
Type II Diabetes	diabetes	Binary	2	No
Hypertension	htn	Binary	2	No
Liver Cirrhosis	cirrhosis	Binary	2	No
Sleep Apena	osa	Binary	2	No
Cardiac Device	cardiac_device	Binary	2	No
Dialysis	dialysis	Binary	2	No
Asthma	asthma	Binary	2	No
Dementia	dementia	Binary	2	No
Cognitive Impairment	cognitive	Binary	2	No

over some training data and evaluate the model by its performance on previously unseen data. For linear models, the error-minimizing parameters (on the training data) can be calculated analytically. For all modern deep learning models, no analytic solution exists, so optimization typically proceeds by stochastic gradient descent.

For neural network models, we change only the function f. In multilayer perceptrons (MLP) for example, we transform our input through a series of matrix multiplications, each followed by a nonlinear activation function. Formally, an L-layer MLP for regression has the simple form

$$\hat{y} = W_L \cdot \phi(W_{L-1} \cdot \ldots \cdot \ \phi(W_1 \cdot x + b_1) + \ldots + b_{L-1}) + b_L \,,$$

where ϕ is an activation function such as sigmoid, tanh, or rectified linear unit (ReLU) and θ consists of the full set of parameters W_l and b_l.

We might view the loss function (squared loss) as simply an intuitive measure of distance. Alternatively, it's possible to derive the choice of squared loss by viewing regression from a probabilistic perspective. In the probabilistic view, a parametric model outputs a distribution $P(y|x)$.

In the simplest case, we can assume that the prediction $\hat{y}$ is the mean of a Gaussian predictive distribution with some variance σ. In this view, we can calculate the probability density of any y given x, and thus can choose our parameters according to the maximum likelihood principle:

$$\theta^{\text{MLE}} = \max_{\theta} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\hat{\sigma}^2}} \exp\left(\frac{-(y_i - \hat{y}_i)^2}{2\hat{\sigma}^2}\right) = \min_{\theta} \sum_{i=1}^{n} \left(\log(\hat{\sigma}_i) + \frac{(y_i - \hat{y}_i)^2}{2\hat{\sigma}^2}\right). \quad (6.1)$$

Assuming constant $\hat{\sigma}$, this yields a familiar least-squares objective.

In this work, we relax the assumption of constant variance (homoscedasticity), predicting both $\hat{y}(\theta, x)$ and $\hat{\sigma}(\theta, x)$ simultaneously. While we apply the idea to MLPs, it is easily applied to networks of arbitrary architecture. To predict the standard deviation $\hat{\sigma}$ of the predictive distribution, we modify our MLP to have two outputs: The first output has linear activation and we interpret its output as the conditional mean $\hat{y}$. The second output models the conditional variance $\hat{\sigma}$. To enforce positivity of $\hat{\sigma}$, we run this output through the softplus activation function $\text{softplus}(z) = \log(1 + \exp(z))$.

We extend the same idea to Laplace distributions, which turn out to better describe the target variability for surgery duration, and are also maximum likelihood estimators when optimizing the Mean Absolute Error (MAE). Mean Absolute Error corresponds to the average number of minutes over or underbooked, and is typically the quantity of interest for this type of scheduling task. The Laplace distribution is parameterized by

$b = \sqrt{2}\sigma$:

$$\theta^{\text{MLE}} = \max \prod_{i=1}^{n} \frac{1}{2b} \exp\left(\frac{-|y_i - \hat{y}_i|}{b}\right) = \min_{\theta} \sum_{i=1}^{n} \left(\log b + \frac{|y_i - \hat{y}_i|}{b}\right). \tag{6.2}$$

Finally, we apply the same technique to perform neural regression with gamma predictive distributions. The gamma distribution has strictly positive support and is long-tailed on the right. Since surgeries and other survival-type data have nonnegative lengths, probability distributions with similarly nonnegative support such as the gamma distribution (compared to the real-valued support of the Gaussian and Laplace distributions), might better describe surgery duration. Formally, the expected time between surgeries (or their associated durations) follows a gamma distribution when surgery start times are modeled as a Poisson process.

The gamma distribution is parametrized by a shape parameter k and a scale parameter Φ:

$$\theta^{\text{MLE}} = \max \prod_{i=1}^{n} \frac{1}{\Gamma(k)\Phi^{k}} y_i^{k-1} \exp\left(\frac{-y_i}{\Phi}\right) = \min_{\theta} \sum_{i=1}^{n} \left(\log(\Gamma(k)) + k \log \Phi - (k-1)\log y_i + \frac{y_i}{\Phi}\right). \tag{6.3}$$

In this case, the model now needs to predict two values: $\hat{k}(\theta, x)$ and $\hat{\Phi}(\theta, x)$. As before, our MLP has two outputs, with both passed through a softplus activation to enforce positivity.

6.4 Experiments

We now present the basic experimental setup. For all experiments we use the same 80%/8%/12% training/validation/test set split. Model weights are updated on the training set and we choose all non-differentiable hyper-parameters and architecture details based on validation set performance. In the final tally, we have 441 features, the majority of which are sparse and accounted for by the one-hot representations over procedures and doctors. We express our labels (the surgery durations) as the number of hours that each procedure takes.

Baselines We consider three baselines for comparison. The first is to follow the current heuristic of predicting the average time per procedure. Note that this is equivalent to training an unregularized linear regression model with a single feature per procedure and no others. Although the main technical contribution of this chapter is concerned with modeling heteroscedasticity, we are also generally interested to know how much performance the current approach leaves untapped. This baseline helps us to address this question. We also compare against linear regression. While we tried applying ℓ_2 regularization, choosing the strength of regularization λ on holdout data, this did not lead to improved performance. Finally, we compare against traditional multilayer perceptrons. To calculate NLL for models that assume homoscedasticity, we choose the constant variance that minimizes NLL on the validation set.

Training Details For all neural network experiments, we use MLPs with ReLU activations. We optimize each network's parameters by stochastic gradient descent, halving the learning rate every 50 epochs. For each experiment, we used an initial learning rate of .1. To determine the architecture, we performed a grid search over the number of hidden layers (in the range 1-3) and over the number of hidden nodes, choosing between 128, 256, 384, 512. As determined by our hyper-parameter optimization, for homoscedastic models, all MLPs use 1 hidden layer with 128 nodes. All heteroscedastic models use 1 hidden layer with 256 nodes. All models use dropout regularization.

For our basic quantitative evaluation, we report the root mean squared error (RMSE), mean absolute error (MAE), and negative log-likelihood (NLL). For heteroscedastic models, we evaluate NLL using the predicted parameters of the distribution.

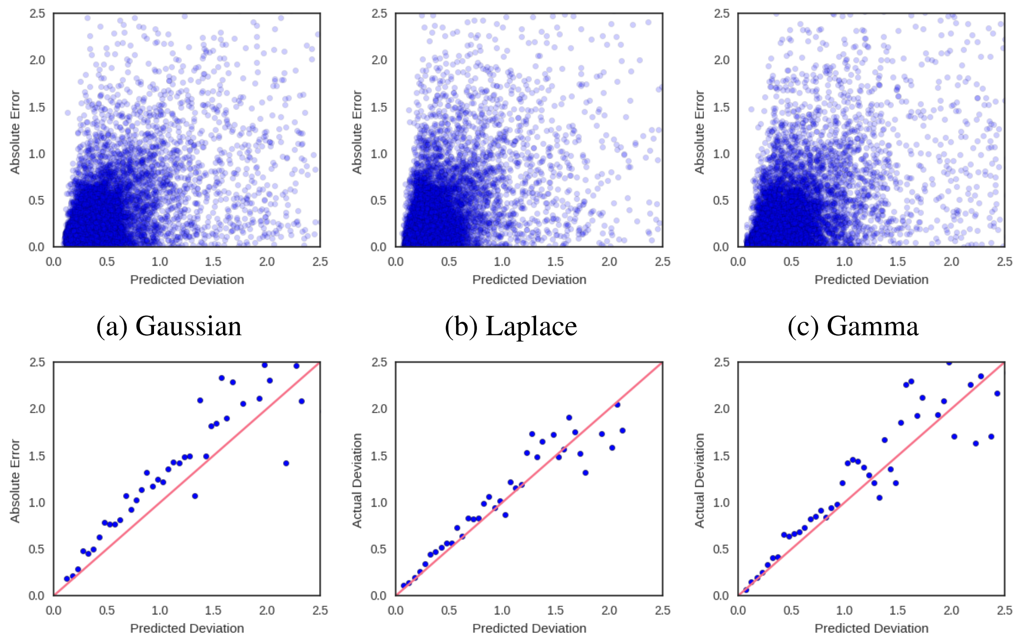
Table 6.2: Performance on test-set data (lower is better). MLP models outperform alternatives at the 1% significance level or better.

Models	**RMSE**	**MAE**	**NLL**	**Change in NLL vs. Current Method**
Current Method	49.80	28.87	1.2385	0.0000
Procedure Means	49.06	27.70	1.2222	0.0164
Linear Regression	45.23	25.07	1.1446	0.0939
MLP Gaussian	43.51	23.90	1.1102	0.1283
MLP Gaussian HS	44.03	24.23	0.7325	0.5060
MLP Laplace	44.24	23.14	1.0621	0.1765
MLP Laplace HS	45.07	23.41	0.5034	0.7351
MLP Gamma HS	43.38	23.23	0.4668	0.7717

For the Gamma distribution, we calculate its mean as $k \cdot \Phi$. We use its mean as the prediction $\hat{y}$ for calculating RMSE. To calculate MAE, we use the median of the Gamma distribution as $\hat{y}$. Although the median of a Gamma distribution has no closed form, it can be efficiently approximated.

We summarize the results in Table 6.2. The heteroscedastic Gamma MLP performs best as measured by both RMSE and NLL, while the Laplace MLP performs best as measured by MAE. All heteroscedastic models outperform all homoscedastic models (as determined by NLL) with the heteroscedastic Gamma MLP achieving an NLL of .4668 as compared to 1.062 by the best performing homoscedastic model (Laplace MLP). The significant quantity when evaluating log likelihood is the *difference* between NLL values, corresponding to the (log of the) likelihood ratio between two models.

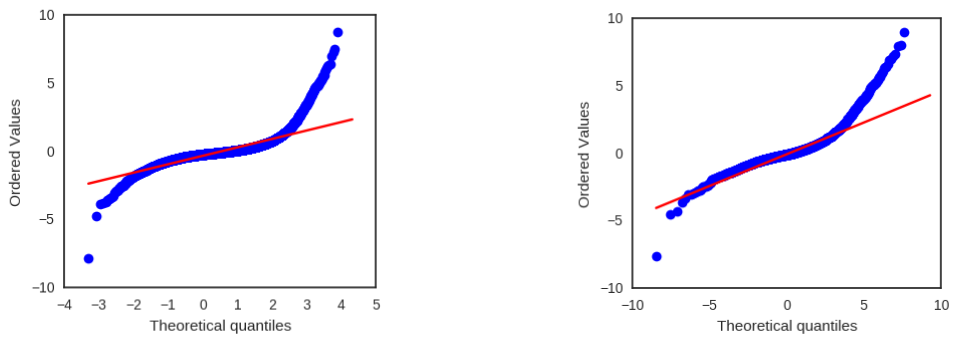
Plots in Figure 6.2 demonstrate that the predicted deviation reliably estimates the observed error, and QQ plots (Figure 6.3) demonstrate that the Laplace distribution appears to fit our targets better than a Gaussian predictive distribution. This gives some (limited) insight into why the Laplace predictive distribution might better fit our data than the Gaussian.



(a) Gaussian (b) Laplace (c) Gamma

(d) Gaussian (e) Laplace (f) Gamma

Figure 6.2: Plots of predicted $\hat{\sigma}$ against absolute error with heteroscedastic Gaussian (a), Laplace (b), and Gamma (c) models. Averaging over bins of width 0.05 (d) (e) (f), shows that $\hat{\sigma}_i$ is a reliable estimator of the observed error.



(a) Gaussian QQ Plot (b) Laplace QQ Plot

Figure 6.3: QQ plots of observed error for Gaussian and Laplace noise models. The Laplace distribution better describes observed error, with shorter tails at both ends.

6.5 Related Work

Previous work in in the medical literature addresses the prediction of surgery duration [EvHN$^+$10, KKSS15, DRS12], accounting for both patient and surgical team characteristics. To our knowledge ours is the first work to address the problem with modern deep learning techniques and the first to model its heteroscedasticity. The idea of neural heteroscedastic regression was first proposed by [NW94], though they do not share hidden layers between the two outputs, and are only concerned with Gaussian predictive distributions. [Wil96] use a shared hidden layer and consider the case of multivariate Gaussian distributions, for which they predict the full covariance matrix via its Cholesky factorization. Heteroscedastic regression has a long history outside of neural networks. [LSC05] address a formulation for Gaussian processes. Most related is [LPB16] which also revisits heteroscedastic neural regression, also using a softplus activation to enforce non-negativity. We show some successful modifications to the above work, such as the use of the Laplace distribution, but our more significant contribution is the application of the idea to clinical medical data.

6.6 Discussion

Our results demonstrate both the efficacy of machine learning (over current approaches) and the heteroscedasticity of surgery duration data. In this section, we explore both results in greater detail. Specifically, we analyze the models to see *which features* are most predictive and examine the uncertainty estimates to see *how they might be used* in decision theory to lower costs.

6.6.1 Feature Importance

First, we consider the importance of the various features. Perhaps the most common way to do this is to see which features corresponded to the largest weights in our linear model. These results are summarized in Figure 6.4. Not surprisingly, the top features are dominated by procedures. In particular pulmonary thromboendarterectomy

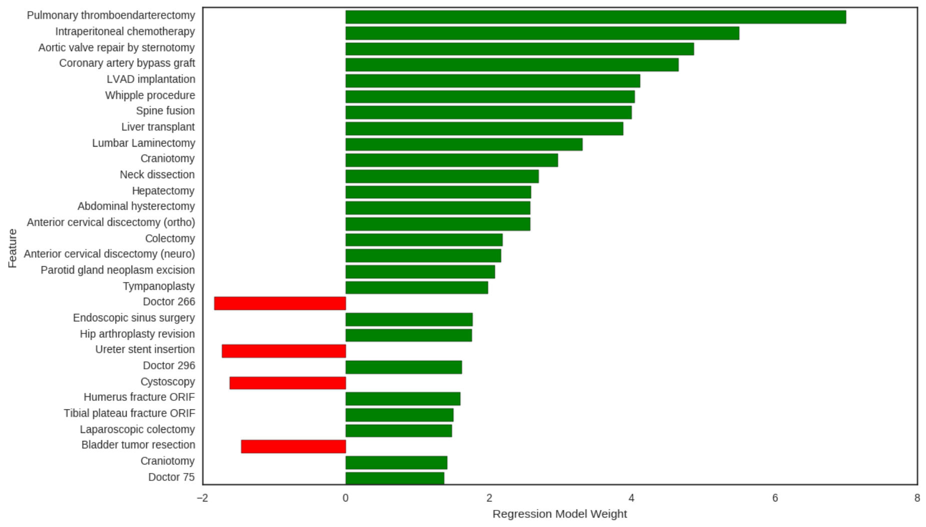


Figure 6.4: Top 30 linear regression features sorted by coefficient magnitude.

receives the highest positive weight. This procedure involves a high risk of mortality, a full cardiopulmonary bypass, hypothermia and full cardiac arrest. Interestingly, even after accounting for procedures, two doctors receive high weight. One (Doctor 266) receives significant negative weight, indicating unusual efficiency and another (Doctor 296) appears to be unusually slow. For ethical reasons, we maintain the anonymity of both the doctors and their specialties.

For neural network models, we evaluate the importance of each feature group by performing an ablation analysis (Figure 6.5). As a group, procedure codes are again the most important features. However, location, patient class, surgeon, anesthesia, and patient sex all contribute significantly. The hour of day appears to influence the performance of our models but the day of the week does not and the month appears to merely introduce noise, leading to a reduction in test set performance. Interestingly, comorbidities also made little difference in performance. However, it is possible that these features only apply to a small subset of patients but are highly predictive for that subset.

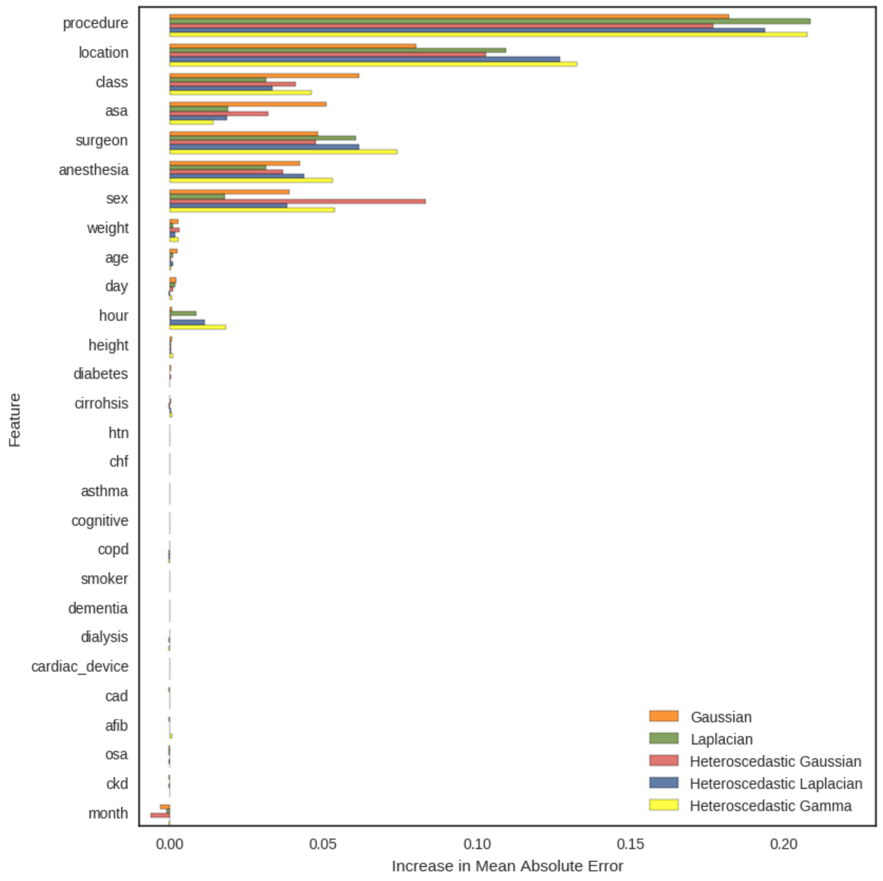


Figure 6.5: Ablation analysis of feature importance for neural models.

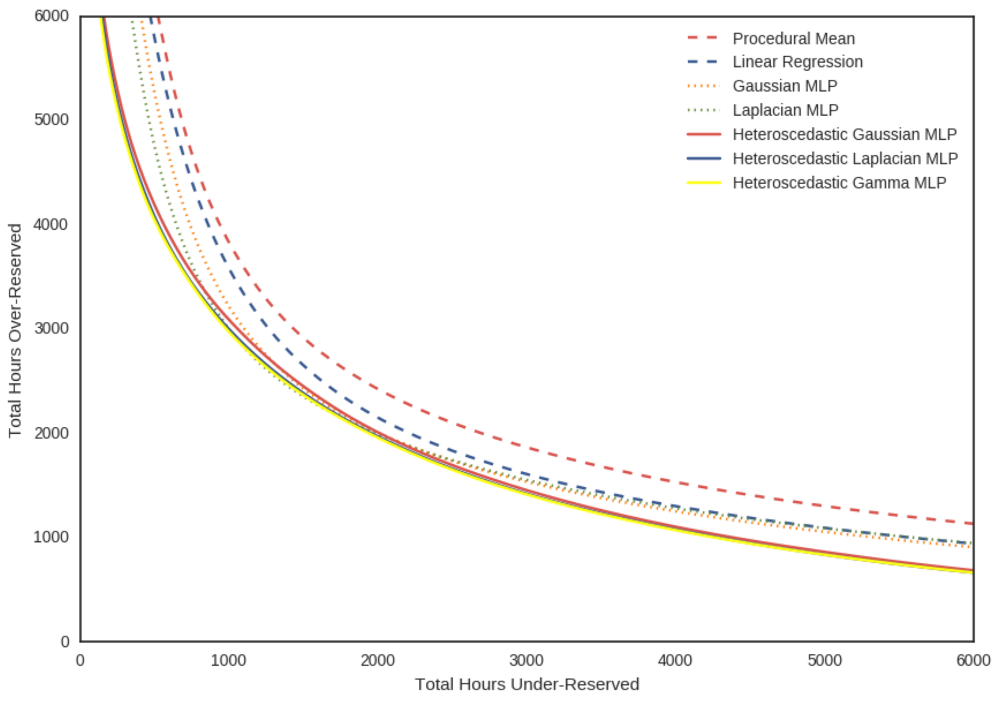


Figure 6.6: Total over-booking and under-booking errors for different models as we adjust predicted values.

6.6.2 Economic Analysis

Our aim in predicting the variance of the error is to provide uncertainty information that could be used to make better scheduling decisions. To compare the various approaches from an economic/decision-theoretic perspective, we might consider the plausible case where the cost to over-reserve the room by one minute (procedure finishes early) differs from the cost to under-reserve the room (procedure runs over). We demonstrate how the two quantities can be traded off in Figure 6.6.

For models that don't output variance, we consider scheduled durations of the form $\hat{y} + k$ and $\hat{y} \cdot k$ where k is a data-independent constant. In either case, by modulating k, one books more or less aggressively. The multiplicative approach performed better, likely because long procedures have higher variance than short ones. This approach is equivalent to selecting a certain percentile of each predicted distribution given a constant sigma.

For heteroscedastic models we make the trade-off by selecting a constant percentile of each predicted distribution. When the cost of over-reserving by one minute and under-reserving by one minute are equal, the problem reduces to minimizing absolute error. Around this point on the curve the homoscedastic Laplace regression outperforms all other models. However, given cost sensitivity, the heteroscedastic Gamma strictly outperforms all other models.

6.6.3 Future Work

We are encouraged by the efficacy of simple machine learning methods both to predict the durations of surgeries and to estimate our uncertainty. We see several promising avenues for future work. Most concretely, we are currently engaged in discussions with the medical institution whose data we used about introducing a trial in which surgeries would be scheduled according to decision theory based on our estimates. Regarding methodology, we look forward to expanding this research in several directions. First, we might extend the approach to modeling covariances and more complex interactions among multiple real-valued predictions. We might also consider problems like bounding box detection, requiring more complex neural architectures.

6.7 Acknowledgments

Chapter 6, *Predicting Surgery Duration with Neural Heteroscedastic Regression* was written in collaboration with Nathan Ng, Rodney Gabriel, Charles Elkan, and Julian McAuley. The dissertation author was the primary investigator and author of this paper.

Part II

Nonstationarity and Decision-Making

Chapter 7

Improving Factor-Based Quantitative Investing by Forecasting Company Fundamentals

On a periodic basis, publicly traded companies are required to report *fundamentals*: financial data such as revenue, operating income, debt, among others. These data points provide some insight into the financial health of a company. Academic research has identified some factors, i.e. computed features of the reported data, that are known through retrospective analysis to outperform the market average. Two popular factors are the book value normalized by market capitalization (book-to-market) and the operating income normalized by the enterprise value (EBIT/EV). In this chapter, we first show through simulation that if we could (clairvoyantly) select stocks using factors calculated on *future* fundamentals (via oracle), then our portfolios would far outperform a standard factor approach. Motivated by this analysis, we train deep neural networks to forecast future fundamentals based on a trailing 5-years window. Quantitative analysis demonstrates a significant improvement in MSE over a naive strategy. Moreover, in retrospective analysis using an industry-grade stock portfolio simulator (backtester), we show an improvement in compounded annual return to 17.1% (MLP) vs 14.4% for a standard factor model.

7.1 Introduction

Public stock markets provide a venue for buying and selling shares, which represent fractional ownership of individual companies. Prices fluctuate frequently, but the myriad drivers of price movements occur on multiple time scales. In the short run, price movements might reflect the dynamics of order execution, and the behavior of high frequency traders. On the scale of days, price fluctuation might be driven by the news cycle. Individual stocks may rise or fall on rumors or reports of sales numbers, product launches, etc. In the long run,we expect a company's market value to reflect its financial performance, as captured in *fundamental data*, i.e., reported financial information such as income, revenue, assets, dividends, and debt. In other words, shares reflect ownership in a company thus share prices should ultimately move towards the company's *intrinsic value*, the cumulative discounted cash flows associated with that ownership. One popular strategy called *value investing* is predicated on the idea that long-run prices reflect this intrinsic value and that the best features for predicting *long-term* intrinsic value are the *currently* available fundamental data.

In a typical quantitative (systematic) investing strategy, we sort the set of available stocks according to some *factor* and construct investment portfolios comprised of those stocks which score highest. Many quantitative investors engineer *value factors* by taking fundamental data in a ratio to stocks price, such as EBIT/EV or book-to-market. Stocks with high value factor ratios are called *value* stocks and those with low ratios are called *growth* stocks. Academic researchers have demonstrated empirically that portfolios of

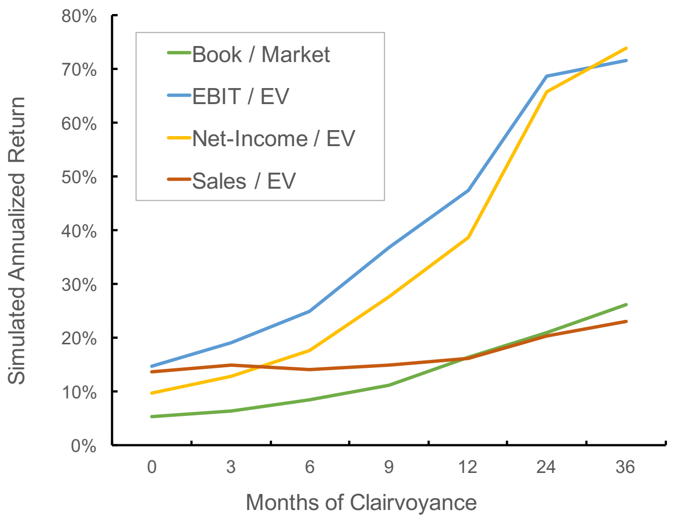


Figure 7.1: Annualized return for various factor models for different degrees of clairvoyance.

stocks which overweight value stocks have significantly outperformed portfolios that overweight growth stocks over the long run [PL17, FF92].

In this chapter, we propose an investment strategy that constructs portfolios of stocks today based on *predicted future fundamentals*. Recall that value factors should identify companies that are inexpensively priced with respect to current company fundamentals such as earnings or book-value. We suggest that the long-term success of an investment should depend on the how well-priced the stock *currently is* with respect to its *future fundamentals*. We run simulations with a *clairvoyant model* that can access future financial reports (by oracle). In Figure 7.1, we demonstrate that for the 2000-2014 time period, a clairvoyant model applying the EBIT/EV factor with 12-month clairvoyant fundamentals, if possible, would achieve a 44% compound annualized return.

Motivated by the performance of factors applied to *clairvoyant* future data, we propose to *predict* future fundamental data based on trailing time series of 5 years of fundamental data. We denote these algorithms as Lookahead Factor Models (LFMs). Both multilayer perceptrons (MLPs) and recurrent neural networks (RNNs) can make informative predictions, achieving out-of-sample MSE of .47, vs .53 for linear regression and .62 for a naive predictor. Simulations demonstrate that investing with LFMs based on the predicted factors yields a compound annualized return (CAR) of 17.1%, vs 14.4% for a normal factor model and a Sharpe ratio .68 vs .55.

7.2 Related Work

Deep neural networks models have proven powerful for tasks as diverse as language translations [SVL14, BCB14], video captioning [MXY$^+$15, VTBE15], video recognition [DAHG$^+$15, TLBN16], and time series modeling [LKEW16, LKW16a, CPC$^+$16]. A number of recent papers consider deep learning approaches to predicting stock market performance. [BE15] evaluates MLPs for stock market prediction. [DZLD] uses recursive tensor nets to extract events from CNN news reports and uses convolutional neural nets to predict future performance from a sequence of extracted events. Several preprinted drafts consider deep learning for stock market prediction [CZD15, WM14, Jia16] however, in all cases, the empirical studies are limited to few stocks and short time periods.

7.3 Deep Learning for Forecasting Fundamentals

7.3.1 Data

In this research, we consider all stocks that were publicly traded on the NYSE, NASDAQ or AMEX exchanges for at least 12 consecutive months between between January, 1970 and September, 2017. From this list, we exclude non-US-based companies, financial sector companies, and any company with an inflation-adjusted market capitalization value below 100 million dollars. The final list contains $11,815$ stocks. Our features consist of reported financial information as archived by the *Compustat North America* and *Compustat Snapshot* databases. Because reported information arrive intermittently throughout a financial period, we discretize the raw data to a monthly time step. Because we are interested in long-term predictions and to smooth out seasonality in the data, at every month, we feed in inputs with a 1-year lag between time frames and predict the fundamentals 12 months into the future.

For each stock and at each time step t, we consider a total of 20 input features. We engineer 16 features from the fundamentals as inputs to our models. Income statement features are cumulative *trailing twelve months*, denoted TTM, and balance sheet features are most recent quarter, denoted MRQ. First we consider These items include *revenue* (TTM); cost of goods sold (TTM); selling, general & and admin expense (TTM); earnings before interest and taxes or EBIT (TTM); net income (TTM); cash and cash equivalents (MRQ); receivables (MRQ); inventories (MRQ); other current assets (MRQ); property plant and equipment (MRQ); other assets (MRQ); debt in current liabilities (MRQ); accounts payable (MRQ); taxes payable (MRQ); other current liabilities (MRQ); total liabilities (MRQ). For all features, we deal with missing values by filling forward previously observed values, following the methods of [LKEW16]. Additionally we incorporate 4 *momentum features*, which indicate the price movement of the stock over the previous 1, 3, 6, and 9 months respectively. So that our model picks up on relative changes and doesn't focus overly on trends in specific time periods, we use the percentile among all stocks as a feature (vs absolute numbers).

7.3.2 Preprocessing

Each of the fundamental features exhibits a wide dynamic range over the universe of considered stocks. For example, Apple's 52-week revenue as of September 2016 was \$215 billion (USD). By contrast, National Presto, which manufactures pressure cookers, had a revenue \$340 million. Intuitively, these statistics are more meaningful when scaled by some measure of a company's size. In preprocessing, we scale all fundamental features in given time series by the market capitalization in the last input time-step of the series. We scale all time steps by the same value so that the neural network can assess the relative change in fundamental values between time steps. While other notions of size are used, such as enterprise value and book equity, we choose to avoid these measure because they can, although rarely, take negative values. We then further scale the features so that they each individually have zero mean and unit standard deviation.

7.3.3 Modeling

In our experiments, we divide the timeline in to an *in-sample* and *out-of-sample* period. Then, even within the in-sample period, we need to partition some of the data as a validation set. In forecasting problems, we face distinct challenges in guarding against overfitting. First, we're concerned with the traditional form of overfitting. Within the in-sample period, we do not want to over-fit to the finite observed training sample. To protect against and quantify this form of overfitting, we randomly hold out a validation set consisting of 30% of all stocks. On this *in-sample* validation set, we determine all hyperparameters, such as learning rate, model architecture, objective function weighting. We also use the in-sample validation set to determine early stopping criteria. When training, we record the validation set accuracy after each training epoch, saving the model for each best score achieved. When 25 epochs have passed without improving on the best validation set performance, we halt training and selecting the model with the best validation performance. In addition to generalizing well to the in-sample holdout set, we evaluate whether the model can predict the future *out-of-sample* stock performance. Since this research is focused on long-term investing, we chose large in-sample and out-of-sample periods of the years 1970-1999 and 2000-2017, respectively.

In previous experiments, we tried predicting price movements directly with RNNs and while the RNN outperformed other approaches on the in-sample period, it failed to meaningfully out-perform a linear model (See results in Table 7.1).

Given only price data, RNN's easily overfit the training data while failing to improve performance on in-sample validation. **One key benefit of our approach** is that by doing *multi-task learning*, predicting all 16 future fundamentals, we provide the model with considerable training signal and may thus be less susceptible to overfitting.

The price movement of stocks is extremely noisy [Shi80] and so, suspecting that the relationships among fundamental data may have a larger signal to noise ratio than the relationship between fundamentals and price, we set up the problem thusly: For MLPs, at each month t, given features for 5 months spaced 1 year apart ($t-48$, $t-36$, $t-24$, $t-12$), predict the fundamental data at time $t+12$. For RNNs, the setup is identical but with the small modification that for each input in the sequence, we predict the corresponding 12 month lookahead data.

We evaluated two classes of deep neural networks: MLPs and RNNs. For each of these, we tune hyperparameters on the in-sample period. We then evaluated the resulting model on the out-of-sample period. For both MLPs and RNNs, we consider architectures evaluated with 1, 2, and 4 layers with 64, 128, 256, 512 or 1024 nodes. We also evaluate the use of dropout both on the inputs and between hidden layers. For MLPs we use ReLU activations and apply batch normalization between layers. For RNNs we test both GRU and LSTM cells with layer normalization. We also searched over various optimizers (SGD, AdaGrad, AdaDelta), settling on AdaDelta. We also applied L2-norm clipping on RNNs to prevent exploding gradients. Our optimization objective is to minimize square loss.

To account for the fact that we care more about our prediction of EBIT over the other fundamental values, we up-weight it in the loss (introducing a hyperparameter α_1).

Table 7.1: Out-of-sample performance for the 2000-2014 time period. All factor models use EBIT/EV. QFM uses current EBIT while our proposed LFMs use predicted EBIT. Price-LSTM is trained to predict price directly.

Strategy	MSE	CAR	Sharpe Ratio
S&P 500	n/a	4.5%	0.19
Market Avg.	n/a	7.7%	0.29
Price-LSTM	n/a	11.3%	0.60
QFM	0.62	14.4%	0.55
Linear-LFM	0.53	15.9%	0.63
MLP-LFM	0.47	17.1%	0.68
RNN-LFM	0.47	16.7%	0.67

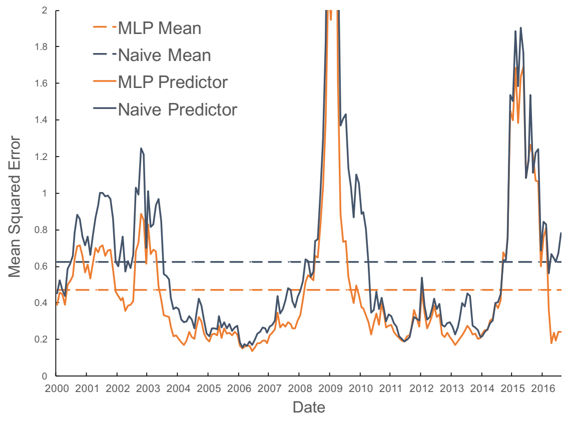


Figure 7.2: MSE over out-of-sample period for MLP (orange) and naive predictor (black).

For RNNs, because we care primarily about the accuracy of the prediction at the final time step (of 5), we upweight the loss at the final time step by hyperparameter α_2 (as in [LKEW16]). Some results from our hyperparameter search on in-sample data are displayed in Table 1. These hyperparameters resulted in MSE on in-sample validation data of 0.6141 for and 0.6109 for the MLP and RNN, respectively.

7.3.4 Evaluation

As a first step in evaluating the forecast produced by the neural networks, we compare the MSE of the predicted fundamental on out-of-sample data with a naive

Table 7.2: Final hyperparameters for MLP and RNN

Hyperparameter	MLP	RNN
Hidden Units	1024	64
Hidden Layers	2	2
Input Dropout Keep Prob.	1.0	1.0
Hidden Dropout Keep Prob.	0.5	1.0
Recurrent Dropout Keep Prob.	n/a	0.7
Max Gradient Norm	1.0	1.0
α_1	0.75	0.5
α_2	n/a	0.7

prediction where predicted fundamentals at time t is assumed to be the same as the fundamentals at $t-12$. To compare the practical utility of traditional factor models vs lookahead factor models we employ an industry grade investment simulator. The simulator evaluates hypothetical stock portfolios constructed on out-of-sample data. Simulated investment returns reflect how an investor might have performed had they invested in the past according to given strategy.

The simulation results reflect assets-under-management at the start of each month that, when adjusted by the S&P 500 Index Price to January 2010, are equal to \$100 million. We construct portfolios by ranking all stocks according to the factor EBIT/EV in each month and investing equal amounts of capital into the top 50 stocks holding each stock for one-year. When a stock falls out of the top 50 after one year, it is sold with proceeds reinvested in another highly ranked stock that is not currently in the simulated portfolio. We limit the number of shares of a security bought or sold in a month to no more than 10% of the monthly volume for a security. Simulated prices for stock purchases and sales are based on the volume-weighted daily closing price of the security during the first 10 trading days of each month. If a stock paid a dividend during the period it was held, the dividend was credited to the simulated fund in proportion to the shares held. Transaction costs are factored in as \$0.01 per share, plus an additional slippage factor that increases as a square of the simulations volume participation in a security. Specifically, if participating at the maximum 10% of monthly volume, the simulation buys at 1% more than the average market price and sells at 1% less than the average market price. Slippage accounts for transaction friction, such as bid/ask spreads, that exists in real life trading.

Our results demonstrate a clear advantage for the lookahead factor model. In nearly all months, however turbulent the market, neural networks outperform the naive predictor (that fundamentals remains unchanged) (Figure 7.2). Simulated portfolios lookahead factor strategies with MLP and RNN perform similarly, both beating traditional factor models (Table 7.1).

7.4 Discussion

In this chapter, we demonstrated a new approach for automated stock market prediction based on time series analysis. Rather than predicting price directly, predict future fundamental data from a trailing window of values. Retrospective analysis with an oracle motivates the approach, demonstrating the superiority of LFM over standard factor approaches. In future work we will thoroughly investigate the relative advantages of LFMs vs directly predicting price. We also plan to investigate the effects of the sampling window, input length, and lookahead distance.

7.5 Acknowledgments

Chapter 7, *Improving Factor-Based Quantitative Investing by Forecasting Company Fundamentals*, was written in collaboration with John Alberg. The dissertation author was the primary investigator and author of this paper.

Chapter 8

Efficient Dialogue Policy Learning with BBQ Networks

We present a new algorithm that significantly improves the efficiency of exploration for deep Q-learning agents in dialogue systems. Our agents explore via Thompson sampling, drawing Monte Carlo samples from a *Bayes-by-Backprop* neural network. Our algorithm learns much faster than common exploration strategies such as ε-greedy, Boltzmann, bootstrapping, and intrinsic-reward-based ones. Additionally, we show that spiking the replay buffer with experiences from just a few successful episodes can make Q-learning feasible when it might otherwise fail.

8.1 Introduction

Increasingly, we interact with computers via natural-language dialogue interfaces. Simple question answering (QA) bots already serve millions of users through Amazon's Alexa, Apple's Siri, Google's Now, and Microsoft's Cortana. These bots typically carry out single-exchange conversations, but we aspire to develop more general dialogue agents, approaching the breadth of capabilities exhibited by human interlocutors. In this work, we consider task-oriented bots [WY04], agents charged with conducting a multi-turn dialogue to achieve some task-specific goal. In our case, we attempt to assist a user to book movie tickets.

For complex dialogue systems, it is often impossible to specify a good policy *a priori* and the dynamics of an environment may change over time. Thus, learning policies online and interactively via reinforcement learning (RL) has emerged as a popular approach [SKLW00, GJK$^+$10, FEAS$^+$16]. Inspired by RL breakthroughs on Atari and board games [MKS$^+$15, SHM$^+$16], we employ deep reinforcement learning (DRL) to learn policies for dialogue systems. Deep Q-network (DQN) agents typically explore via the ε-greedy heuristic, but when rewards are sparse and action spaces are large (as in dialogue systems), this strategy tends to fail. In our experiments, a randomly exploring Q-learner never experiences success in thousands of episodes.

We offer a new, efficient solution to improve the exploration of Q-learners. We propose a Bayesian exploration strategy that encourages a dialogue agent to explore state-action regions in which the agent is relatively uncertain in action selection. Our algorithm, the *Bayes-by-Backprop Q-network* (BBQN), explores via Thompson sampling, drawing Monte Carlo samples from a Bayesian neural network [BCKW15]. In order to produce the temporal difference targets for Q-learning, we must generate predictions from a frozen target network [MKS$^+$15]. We show that using the maximum a posteriori (MAP) assignments to generate targets results in better performance (in addition to being computationally efficient). We also demonstrate the effectiveness of *replay buffer spiking* (RBS), a simple technique in which we pre-fill the experience replay buffer with a small set of transitions harvested from a naïve, but occasionally successful, rule-based agent. This technique proves essential for both BBQNs and standard DQNs.

We evaluate our dialogue agents on two variants of a movie-booking task. Our agent interacts with a user to book a movie. Success is determined at the end of the dialogue if a movie has been booked that satisfies the user. We benchmark our algorithm and baselines using an agenda-based user simulator similar to [STY07]. To make the task plausibly challenging, our simulator introduces random mistakes to account for the effects of speech recognition and language understanding errors. In the first variant, our environment remains fixed for all rounds of training. In the second variant, we consider a non-stationary, domain-extension environment. In this setting, new attributes of films become available over time, increasing the diversity of dialogue actions available to both the user and the agent. Our experiments on both the stationary and domain-extension environments demonstrate that BBQNs outperform DQNs using either ε-greedy exploration, Boltzmann exploration, or the bootstrap approach introduced by [OBPVR16]. Furthermore, the real user evaluation results consolidate the effectiveness of our approach that BBQNs are more effective than DQNs in exploration. Besides, we also show that all agents only work given replay buffer spiking, although the number of pre-filled dialogues can be small.

8.2 Task-completion dialogue systems

In this chapter, we consider goal-oriented dialogue agents, specifically one that aims to help users to book movie tickets. Over the course of several exchanges, the agent gathers information such as movie name, theater and number of tickets, and ultimately completes a booking. A typical dialogue pipeline is shown in Figure 8.1. In every turn of a conversation, the *language understanding* module converts raw text into structured semantic representations known as *dialog-acts*, which pass through the *state-tracker* to maintain a record of information accumulated from previous utterances. The *dialogue policy* then selects an action (to be defined later) which is transformed to a natural

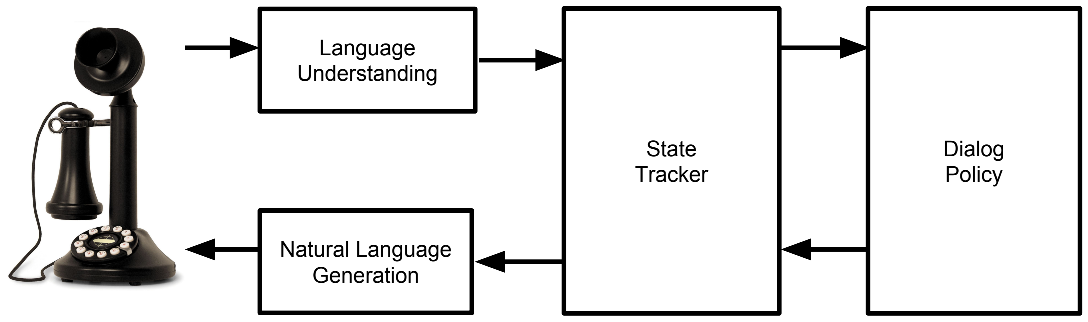


Figure 8.1: Components of a dialogue system

language form by a *generation* module. The conversation continues until the dialogue terminates. A numerical reward signal is used to measure the utility of the conversation. Details of this process are given below.

8.2.1 Dialog-acts

Following [STY07], we represent utterances as dialog-acts, consisting of a single *act* and a (possibly empty) collection of *(slot=value)* pairs, some of which are *informed* while others are *requested* (value omitted). For example, the utterance, "I'd like to see *Our Kind of Traitor* tonight in Seattle" maps to the structured semantic representation *request(ticket, moviename=Our Kind of Traitor, starttime=tonight, city=Seattle).*

8.2.2 State tracker

Other than information inferred from previous utterances, the state-tracker may also interact with a database, providing the policy with information such as how many movies match the current constraints. It then *de-lexicalizes* the dialog-act, allowing the dialogue policy to act upon more generic states. The tracked state of the dialogue, consisting of a representation of the conversation history and several database features, is passed on to the policy to select actions.

8.2.3 Actions

Each action is a de-lexicalized *dialog-act*. In the movie-booking task, we consider a set of 39 actions. These include basic actions such as *greeting(), thanks(), deny(), confirm_question(), confirm_answer(), closing()*. Additionally, we add two actions for each slot: one to inform its value and the other to request it. The pipeline then flows back to the user. Any slots informed by the policy are then filled in by the state tracker. This yields a structured representation such as *inform(theater=Cinemark Lincoln Square)*, which is then mapped by a natural language generation module to a textual utterance, such as "This movie is playing tonight at Cinemark Lincoln Square."

The conversation process above can be naturally mapped to the reinforcement learning (RL) framework, as follows [LPE97]. The RL agent navigates a Markov decision process (MDP), interacting with its environment over a sequence of discrete steps [SB98]. At step $t \in \{1, 2, \ldots\}$, the agent observes the current state s_t, and chooses some action a_t according to a policy π. The agent then receives reward r_t and observes new state s_{t+1}, continuing the cycle until the episode terminates. In this work, we assume that the set of actions, denoted $\mathcal{A}$, is finite. In our dialogue scenario, the state-tracker produces states, actions are the de-lexicalized dialog-acts described earlier, state transitions are governed by the dynamics of the conversation, and a properly defined reward function is used to measure the degree of success of a dialogue. In our experiment, for example, success corresponds to a reward of 40, failure to a reward of -10, and we apply a per-turn penalty of -1 to encourage pithy exchanges.

The goal of RL is to find an optimal policy to maximize long-term reward. The Q-function measures, for every state-action pair (s, a), the maximum expected cumulative discounted reward achieved by choosing a in s and then following an optimal policy thereafter: $Q^*(s, a) = \max_\pi \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s, a_t = a\right]$, where $\gamma \in (0, 1)$ is a discount factor. Owing to large state spaces, most practical reinforcement learners approximate the Q-function by some parameterized model $Q(s,a;\theta)$. An example, as we used in this chapter, is a neural network, where θ represents the set of weights to be learned. Once a good estimate of θ is found so that $Q(\cdot,\cdot;\theta)$ is a good approximation of $Q(\cdot,\cdot)$, the greedy policy, $\pi(s;\theta) = \arg\max_a Q(s,a;\theta)$, is a near-optimal policy [SB98]. A popular way to learn a neural-network-based Q-function is known as DQN [MKS$^+$15].

8.3 Deep Q-Learning

An RL agent navigates a Markov decision process (MDP), interacting with its environment over a sequence of discrete steps. At each step t, the agent observes the current state $s_t \in \mathcal{S}$, and chooses some action $a_t \in \mathcal{A}$ according to a policy π. The agent then receives reward r_t and observes new state s_{t+1}, continuing the cycle until the episode terminates. Here, $\mathcal{S}$ represents the set of all possible states, $\mathcal{A}$ defines the space of possible actions and the policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ maps states onto actions. In this work, we assume actions to be discrete and $|\mathcal{A}|$ to be finite. Under a policy π and in state s the *value* of action a is the expected cumulative discounted reward (also known as *return*):

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{i=0}^{T} \gamma^{i} r_{t+i} | s_t = s, a_t = a\right] \tag{8.1}$$

where γ is a discount factor. An optimal policy is one whose Q-function uniformly dominates others. Its value function, called the *optimal value function*, is denoted Q^* [SB98]. Owing to large state spaces, most practical reinforcement learners approximate the Q function by some parameterized model $Q(s,a;\theta)$ among which deep neural networks have become especially popular.

Given the optimal value function Q^*, at any time-step t, the optimal move is for the agent to choose action $a^* = \text{argmax}_a Q^*(s,a)$. Thus, learning an optimal policy can be reduced to learning the optimal value function. For toy problems, where an environment can be fully explored, we can maintain an estimate of the Q function as a table of values, with rows indexing each state and columns for each action. In practice, the number of states may be intractably large, and the sample complexity of exploration can grow at least linearly with the number of states $|S|$ and the size of the action space $|\mathcal{A}|$. Thus, most practical reinforcement learners approximate the Q function by some parameterized model $Q(s,a;\theta)$, among which deep neural networks have become especially popular.

The definition of return specifies a recursion: the value of the current state, action pair (s,a), depends upon the expected value of the successor state s_{t+1} and the action chosen in that state:

$$Q(s_t,a_t) = r_t + \gamma \max_{a'} Q(s_{t+1},a')\,. \tag{8.2}$$

For a fixed policy, the value function can be iteratively improved by approximate value iteration. We represent experiences as tuples (s_t,a_t,r_t,s_{t+1}). In Q-learning, we aim to improve the value function (and, in turn, the greedy policy) by minimizing the squared error between the current prediction and the one step look-ahead prediction

$$\mathcal{L}(\theta_t) = \mathbb{E}\left[(y_t - Q(s_t,a_t;\theta_t))^2\right] \tag{8.3}$$

for $y_t = r_t + \gamma \max_{a'} Q(s_{t+1},a';\theta_t)$. Traditionally, the Q-function is trained by stochastic approximation, estimating the loss on each experience as it is encountered, yielding the update:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(y_t - Q(s_t,a_t;\theta_t))\nabla Q(s_t,a_t;\theta_t). \tag{8.4}$$

A few tricks improve the effectiveness of DQNs. First, *experience replay* maintains a buffer of experiences, training off-policy on randomly selected mini-batches [Lin92, MKS$^+$15]. Second, it's common to periodically cache DQN parameters parameters, using the stale parameters to compute the training targets y_t.

Other techniques such as double deep Q-learning [VHGS15] and prioritized experience replay [SQAS16] appear effective for learning the Q-function. For simplicity and because these techniques are straightforward to combine with ours, we build on the basic DQN model and focus on the issue of exploration.

In order to expose the agent to a rich set of experiences, one must employ a strategy for exploration. Most commonly in the DQN literature, researchers use the ϵ-greedy exploration heuristic. In this work, we improve upon greedy exploration strategies by using uncertainty information (in the predicted Q values) to make more intelligent exploration choices.

8.4 Bayes-by-Backprop

We now introduce Bayes-by-Backprop [BCKW15], a method for extracting uncertainty information from neural networks by maintaining a probability distribution over the weights in the network. We confine the present discussion to multilayer perceptrons (MLPs), i.e., feedforward neural networks composed entirely of fully connected layers, without recurrent connections. A standard MLP for regression models $P(y|x,w)$, parameterized by weights $w = \{W_l, b_l\}_{l=1}^{L}$. MLPs have the simple architecture:

$$\hat{y} = W_L \cdot \phi(W_{L-1} \cdot \ldots \cdot \ \phi(W_1 \cdot x + b_1) + \ldots + b_{L-1}) + b_L \tag{8.5}$$

for a network with L layers ($L-1$ hidden) and activation function ϕ (commonly sigmoid, tanh, or rectified linear unit (ReLU).

In standard neural network training, we learn the weights w given a dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$ by maximum likelihood estimation (MLE), using some variant of stochastic gradient descent: $w^{MLE} = \text{argmax}_w \log p(\mathcal{D}|w)$. Frequently, we regularize models by placing priors on the parameters w. The resulting optimization seeks the maximum a posteriori (MAP) assignment $w^{MAP} = \operatorname{argmax}_w \log p(w|\mathcal{D})$. This yields ℓ_2^2 regularization for Gaussian prior or ℓ_1 regularization for Laplace prior:

$$\begin{aligned} w^{MAP} &= \operatorname*{argmax}_w \log p(w|\mathcal{D}) \\ &= \operatorname*{argmax}_w \ln p(\mathcal{D}|w) + \ln p(w). \end{aligned} \tag{8.6}$$

Both MLE and MAP assignments produce point estimates of w, and thus capture only the mode of the predictive distribution, But to enable efficient exploration, we prefer a model that can quantify uncertainty. Thus, we consider a Bayesian treatment of neural networks, learning a full posterior distribution $p(w|\mathcal{D})$.

Problematically, $p(w|\mathcal{D})$ may be intractable. The weights may be arbitrarily correlated and the joint distribution might be of arbitrary complexity, and thus difficult both to learn and to sample from. So we approximate the potentially intractable posterior by a variational distribution $q(w|\theta)$. In this work, we choose q to be Gaussian with diagonal covariance. Thus, we sample each weight w_i from a univariate Gaussian distribution parameterized by mean μ_i and standard deviation σ_i. To ensure that all σ_i remain strictly positive, we parameterize σ_i by the softplus function $\sigma_i = \log(1+\exp(\rho_i))$, giving variational parameters $\theta = \{(\mu_i, \rho_i)\}_{i=1}^{D}$ for D-dimensional weight vector w.

Note that the true posterior is both multi-modal (owing to symmetry among the nodes) and intractable. There is no reason to believe that the true posterior exhibits conditional independence between every pair of two weights regardless of the values taken by the others. So this is only an approximation in a very narrow sense. Nonetheless, it proves useful in practice.

We learn these parameters by minimizing the Kullback-Liebler (KL) divergence between the variational approximation $q(w|\theta)$ and the posterior $p(w|\mathcal{D})$

$$\begin{aligned}\theta^* &= \mathrm{argmin}_\theta \mathrm{KL}[q(w|\theta)||p(w|\mathcal{D})] \\ &= \mathrm{argmin}_\theta \int q(w|\theta)\log\frac{q(w|\theta)}{p(w)p(\mathcal{D}|w)}\mathrm{d}w \\ &= \mathrm{argmin}_\theta \mathrm{KL}[q(w|\theta)||p(w)] - \mathbb{E}_{q(w|\theta)}[\log p(\mathcal{D}|w)].\end{aligned} \tag{8.7}$$

The expression minimized is termed by [HVC93] the *variational free energy*

$$\mathcal{F} = \mathrm{KL}[q(w|\theta)||p(w)] - \mathbb{E}_{q(w|\theta)}[\ln p(\mathcal{D}|w)]. \tag{8.8}$$

The term on the left term penalizes distance between the variational posterior $q(w|\theta)$ and the prior $p(w)$. Specifically, the KL divergence measures the amount of information (measured in nats) that are lost when $p(w)$ is used to approximate $q(w|\theta)$. Thus [HVC93] describe this term as a penalty on the description length of weights. Assuming a Gaussian predictive distribution, the rightmost term is simply the expected square loss. Sampling from q, our cost function is $f(\mathcal{D},\theta) = \log q(w|\theta) - \log p(w) - \log p(\mathcal{D}|w)$. When $q(w|\theta)$ and $p(w)$ are both parameterized as univariate Gaussian distributions, their distance is minimized by setting $\mu_q = \mu_p$ for every setting of the variances, and by setting the standard deviations $\sigma_q = \sigma_p$ for any setting of the means μ_q and μ_p.

We can learn the variational parameters θ by gradient descent, using the reparametrization trick popularized by [KW13]. In short, we want to differentiate the loss with respect to the variational parameters θ, but the loss depends upon the random vector $w \sim q(w|\theta)$. We can overcome this problem by expressing w as a deterministic function of θ, $g(\eta,\theta)$, where η is a random vector. When we choose g and noise distribution $p(\eta)$ such that $p(\eta)d\eta = q(w|\theta)dw$, we can express our optimization objective equivalently as an expectation over η. In our case, we take η to be a noise vector drawn from D-dimensional standard normal $\mathcal{N}(0,I)^D$. Thus $w = g(\eta,\theta) = \mu + \log(1+\exp(\rho)) \odot \eta$, where $\odot$ is the element-wise product.

For any given value of η our loss is differentiable with respect to the variational parameters. We can then proceed with backpropagation, treating η as a noise input sampled for each batch. Thus, we minimize the loss by stochastic gradient descent, using a single Monte Carlo sample $\eta \sim p(\eta)$ at each iteration. In our case, we take η to be a noise vector drawn from isotropic standard normal $\mathcal{N}(0,I)^D$.

8.5 BBQ-networks

We are now ready to introduce *BBQN*, our algorithm for learning dialogue policies with deep learning models. BBQN builds upon the deep Q-network, or DQN [MKS$^+$15], and uses a Bayesian neural network to approximate the Q-function and the uncertainty in its approximation. Since we work with fixed-length representations of dialogues, we use an MLP, but extending our methodology to recurrent or convolutional neural networks is straightforward.

8.5.1 Action selection

A distinct feature of BBQN is that it explicitly quantifies uncertainty in the Q-function estimate, which can be used to guide exploration. In DQN, the Q-function is represented by a network with parameter w. BBQN, in contrast, maintains a distribution q over w. As described in the previous section, q is a multivariante Gaussian with diagonal covariance, parameterized by $\theta = \{(\mu_i, \rho_i)\}_{i=1}^D$. In other words, a weight w_i has a posterior distribution q that is $\mathcal{N}(\mu_i, \sigma_i^2)$ where $\sigma_i = \log(1+\exp(\rho_i))$.

Given a posterior distribution q over w, a natural and effective approach to exploration is posterior sampling, or Thompson Sampling [Tho33, CL11, ORR13], in which actions are sampled according to the posterior probability that they are optimal in the current state. Formally, given a state s_t and network parameter θ_t in step t, an action a is selected to be a_t with the probability $\Pr(a_t = a|s_t, \theta_t) =$

$$\int_w \mathbf{1}\{\ Q(s_t, a; w) > Q(s, a'; w), \forall a' \neq a\} \cdot dq(w|\theta_t) \tag{8.9}$$

Computing these probabilities is usually difficult, but fortunately all we need is a *sample* of an action from the corresponding multinomial distribution. To do so, we first draw $w_t \sim q(\cdot|\theta_t)$, then set $a_t = \operatorname{argmax}_a Q(s_t, a; w_t)$. It can be verified that this process samples actions with the same probabilities given in the Equation 8.9. We have also considered integrating the ε-greedy approach, exploring by Thompson sampling with probability $1 - \varepsilon$ and uniformly at random with probability ε. But empirically, uniform random exploration confers no supplementary benefit for our task.

8.5.2 BBQN

The BBQN is initialized by a prior distribution p over w. It consists of an isotropic Gaussian whose variance σ_p^2 is a single hyper-parameter introduced by our model. We initialize the variational parameters to match the prior. So μ is initialized to the zero vector 0 and the variational standard deviation σ matches the prior σ_p for each weight. Note that unlike conventional neural networks, we need not assign the weights randomly because sampling breaks symmetry. As a consequence of this initialization, from the outset, the agent explores uniformly at random. Over the course of training, as the experience buffer fills, the mean squared error starts to dominate the objective function and the variational distribution moves further from the prior.

Given experiences of the form $\mathcal{T} = \{(s, a, r, s')\}$ consisting of transitions collected so far, we apply a Q-learning approach to optimize the network parameter, in a way similar to DQN [MKS+15]. To do so, we maintain a frozen, but periodically updated, copy of the same BBQN, whose parameter is denoted by $\tilde{\theta} = \{(\tilde{\mu}_i, \tilde{\rho}_i)\}_{i=1}^{D}$. For any transition $(s,a,r,s') \in \mathcal{T}$, this network is used to compute a target value y for $Q(s,a;\theta)$, resulting in a regression data set $\mathcal{D} = \{(x,y)\}$, for $x = (s,a)$. We then apply the Bayes-by-backprop method described in the previous section to optimize θ, until it converges when $\tilde{\theta}$ is replaced by θ. There are two ways to generate the target value y.

The first uses a Monte Carlo sample from the frozen network, $\tilde{w} \sim q(\cdot|\tilde{\theta})$, to compute the target y: $y = r + \gamma \max_{a'} Q(s',a';\tilde{w})$. To speed up training, for each mini-batch, we draw one sample of $\tilde{w}$ for target generation, and one sample of w for sample-based variational inference (see previous section). With this implementation, the training speeds of BBQN and DQN are roughly equivalent.

The second uses maximum a posterior (MAP) estimate to compute y: $y = r + \gamma \max_{a'} Q(s',a';\tilde{\mu})$. This computationally more efficient choice is motivated by the observation that, since we only require the uncertainty estimates for exploration, it may not be necessary to sample from the frozen network for synthesizing targets. Furthermore, early in training, the predictive distribution of the networks has high variance, resulting in a large amount of noise in target values that can slow down training.

8.5.3 BBQN with intrinsic reward

Variational Information Maximizing Exploration (VIME) [HCD$^+$16] introduces an exploration strategy based on maximizing the information gain about the agent's belief of environment dynamics. It adds an *intrinsic reward* bonus to the reward function, which quantifies the agent's *surprise*: $r'(s_t,a_t,s_{t+1}) = r(s_t,a_t) + \eta \mathbb{D}_{KL}[p(\theta|\xi_t,a_t,s_{t+1})||p(\theta|\xi_t)]$, and has demonstrated strong empirical performance. We explore a version of BBQNs that incorporates the intrinsic reward from VIME, terming the approach BBQN-VIME-MC/MAP. The BBQN-VIME variations encourage the agents to explore the state-action regions that are relatively unexplored and in which BBQN is relatively uncertain in action selection. In our full-domain experiment, both BBQN and BBQN-VIME variations achieve similar performance with no significant difference, but in domain-extension experiments, we observe that BBQN-VIME-MC slightly outperforms BBQN-MAP.

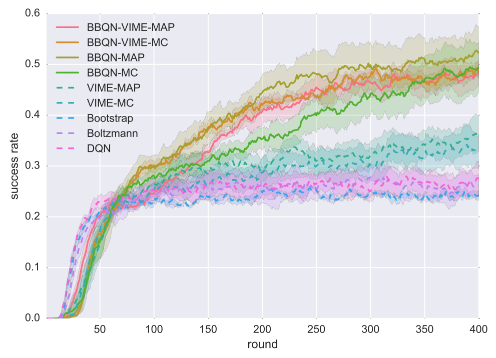
8.5.4 Replay buffer spiking

In reinforcement learning, there are multiple sources of uncertainty. These include uncertainty over the parameters of our model and uncertainty over unseen parts of the environment. BBQN addresses parameter uncertainty but it can struggle given extreme reward sparsity. Researchers use various techniques to accelerate learning in these settings. One approach is to leverage prior knowledge, as by reward shaping or imitation learning. Our approach falls into this category. Fortunately, in our setting, it's easy to produce a few successful dialogues manually. Even though the manual dialogues do not follow an optimal policy, they contain some successful movie bookings, so they indicate the existence of the large ($+40$) reward signal. Pre-filling the replay buffer with these experiences dramatically improves performance (Figure 8.3). For these experiments, we construct a simple rule-based agent that, while sub-optimal (18.3% success rate), achieves success sometimes. In each experiment, we harvest 100 dialogues of experiences from the rule-based agent, adding them to the replay buffer. We find that, in on our task, RBS is essential for both BBQN and DQN approaches. Interestingly, performance does not strictly improve with the number pre-filled dialogues (Figure 8.3). Note that replay buffer spiking is different from imitation learning. RBS works well with even a small number of warm-start dialogues, suggesting that it is helpful to communicate even the very *existence* of a big reward. We find that even one example of a successful dialogue in the replay buffer could successfully jump-start a Q-learner.

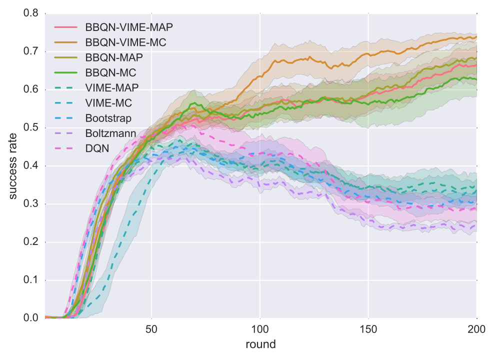
8.6 Experiments

We evaluate our methods on two variants of the movie-booking task. In the first, the agent interacts with the user simulator over 400 rounds. Each round consists of 50 simulated dialogues, followed by 2 epochs of training. All slots are available starting from the very first episode. In the second, we test each model's ability to adapt to domain extension by periodically introducing new slots. Each time we add a new slot, we augment both the state space and action space. We start out with only the essential slots: [*date, ticket, city, theater, starttime, moviename, numberofpeople, taskcomplete*] and train for 40 training rounds up front. Then, every 10 rounds, we introduce a new slot in a fixed order. For each added slot, the state space and action space grow accordingly. This experiment terminates after 200 rounds. In both experiments, quantifying uncertainty in the network weights is important to guide effective exploration.

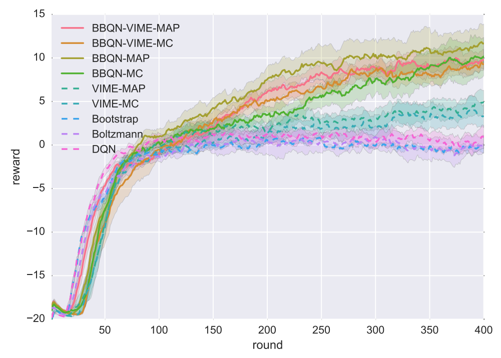
To represent the state of the dialogue at each turn, we construct a 268 dimensional feature vector, consisting of the following: (i) one-hot representations of the *act* and *slot* corresponding to the current user action, with separate components for requested and informed slots; (ii) corresponding representations of the *act* and *slot* corresponding to the last agent action; (iii) a bag of *slots* corresponding to all previously filled slots over the course of the dialog history; (iv) both a scalar and one-hot representation of the current turn count; and (v) counts representing the number of results from the knowledge base that match each presently filled-in constraint (informed slot) as well as the intersection of all filled-in constraints. For domain-extension experiments, features corresponding to unseen slots take value 0 until they are seen. When domain is extended, we add features and corresponding weights to input layer, initializing the new weights to 0 (or $\mu_i = 0$, $\sigma_i = \sigma_{prior}$ for BBQN), a trick due to [LVM15].



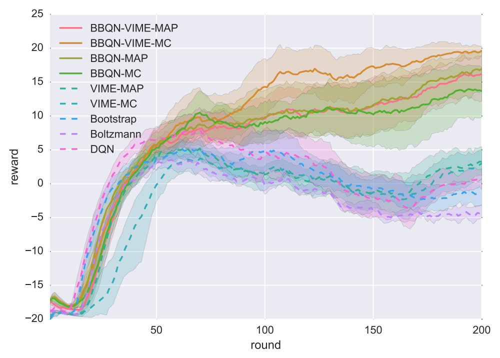
(a) Full domain (success rate)



(b) Domain extension (success rate)



(c) Full domain (reward)



(d) Domain extension (reward)

Figure 8.2: Training plots with confidence intervals for the full domain (all slots available from start) and domain extension problems (slots added every 10 rounds).

8.6.1 Training details

For training, we first use a naive but occasionally successful rule-based agent for RBS. All experiments use 100 dialogues to spike the replay buffer. We note that experiments showed models to be insensitive to the precise number. After each round of 50 simulated dialogues, the agent freezes the target network parameters θ^{-}, and then updates the Q- function, training for 2 epochs, then re-freezes and trains for another 2 epochs. There are two reasons for proceeding in 50-dialog spurts, rather than updating one mini-batch per turn. First, in a deployed system, real-time updates might not be realistic. Second, we train for more batches per new turn than is customary in DQN literatures owing to the economic considerations: computational costs are negligible,

Table 8.1: Final performance of trained agents on 10k simulated dialogues, averaged over 5 runs.

Agents	Full Domain		Domain Extension	
	Success Rate	Reward	Success Rate	Reward
BBQN-VIME-MAP	0.4856	9.8623	0.6813	15.8223
BBQN-VIME-MC	0.4941	10.4268	**0.7120**	**17.6261**
BBQN-MAP	**0.5031**	**10.7093**	0.6852	17.3230
BBQN-MC	0.4877	9.9840	0.6722	16.1320
VIME-MAP	0.3893	5.8616	0.3751	4.9223
VIME-MC	0.3700	4.9990	0.3675	4.8270
Bootstrap	0.2516	-0.1300	0.3170	-0.6820
Boltzmann	0.2658	0.4180	0.2435	-3.4640
DQN	0.2693	0.8660	0.3503	4.7560

while failed dialogues either consume human labor (in testing) or confer opportunity costs (in the wild).

8.6.2 Baseline methods

To demonstrate the efficacy of BBQN, we compare against ε-greedy in a standard DQN. Additionally, we compare against Boltzmann exploration, an approach in which the probability of selecting any action in a given state is determined by a softmax function applied to the predicted Q-values. Here, affinity for exploration is parameterized by the Boltzmann *temperature*. We also compare to the bootstrap method of [OBPVR16]. For the bootstrap experiments, we use 10 bootstrap heads, and assign each data point to each head with probability 0.5. We evaluate all four methods on both the full domain (static) learning problem and on the domain extension problem.

We also tried comparing against Gaussian processes (GP) based approaches. However, in our setting, due to the high-dimensional inputs and large number of time steps, we were unable to get good results. In our experiments, the computation and memory requirement grow quadratically over time, and memory starts to explode at the 10th (simulation) round. Limiting data size for GP was not helpful. Furthermore, in contrast to [GJK$^+$10] where the state is 3-dimensional, our experiments have 268-dimensional states, making scalability an even bigger challenge. A recent paper [FEAS$^+$16] compares deep RL (both policy gradient and Q-learning) to GP-SARSA [EMM05] on a simpler dialogue policy learning problem. In order to make Gaussian processes computationally tractable, they rely on sparsification methods [EMM05], gaining computation efficiency at the expense of accuracy. Despite this undertaking to make GPs feasible and competitive, they found that deep RL approaches outperform GP-SARSA with respect to final performance, regret, and computational expense (by wall-clock). While we consider Gaussian processes to be an evolving area, it is worthwhile to try the Gaussian processes with sparsification methods to compare with deep RL approaches as future work.

8.6.3 Architecture details

All models are MLPs with ReLU activations. Each network has 2 hidden layers with 256 hidden nodes each. We optimize over parameters using Adam [KB15] with a batch size of 32 and initial learning rate of 0.001, determined by a grid search. To avoid biasing the experiments towards our methods, we determine common hyper-parameters using standard DQN. Because BBQN confers regularization, we equip DQN models with dropout regularization of 0.5, shown by [BCKW15] to confer comparable predictive performance on holdout data.

Each model has additional hyper-parameters. For example, ε-greedy exploration requires an initial value of ε and an attenuation schedule. Boltzmann exploration requires a temperature. The bootstrapping-based method of [OBPVR16] requires both a number of bootstrap heads and the probability that each data point is assigned to each head. Our BBQN requires that we determine the variance of the Gaussian prior distribution and the variance of the Gaussian error distribution.

8.6.4 Simulation results

As shown in Figure 8.2, BBQN variants perform better than the baselines. In particular, BBQN-MAP performs the best on the full domain setting, BBQN-VIME-MC achieves the best performance on the domain extension setting, with respect to cumulative successes during training and final performance of the trained models (Table 8.1). Note that the domain extension problem becomes more difficult every 10 epochs, so sustained performance corresponds to getting better, while declining performance does not imply the policy becomes worse. On both problems, no method achieves a single success absent RBS. Evaluating our best algorithm (BBQN-MAP) using 0, 100, and 1000 RBS dialogues (Figure 8.3), we find that using 1000 (as compared to 100) dialogues, our agents learn quickly but that their long-term performance is worse. One heuristic to try in the future may be to discard pre-filled experiences after meeting some performance threshold.

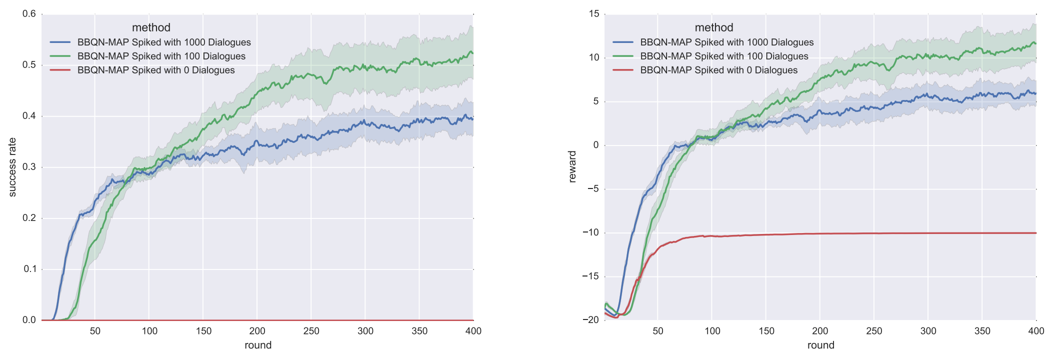
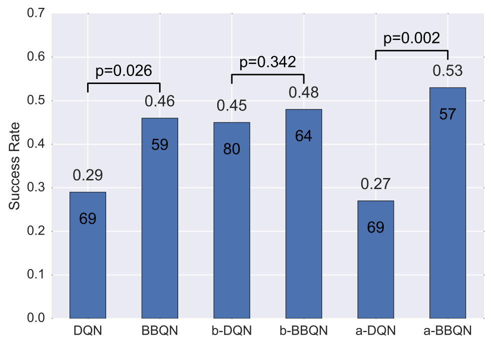


Figure 8.3: RBS with 100 dialogues improves both success rate (left) and reward (right).

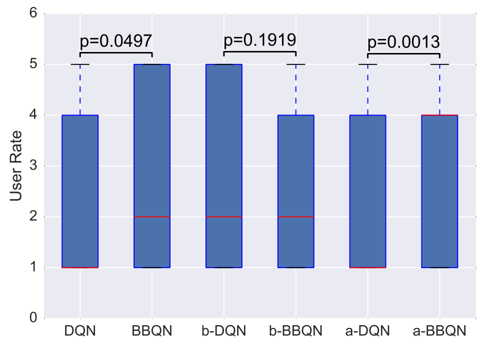
We also considered that perhaps some promising trajectories might never be sampled by the BBQN. Thus, we constructed an experiment exploring via a hybridization of the BBQN's Thompson sampling with the ε-greedy approach. With probability $1-\varepsilon$, the agent selects an action by Thompson sampling given one Monte Carlo sample from the BBQN and with probability ε the agent selects an action uniformly at random. However, the uniformly random exploration confers no additional benefit.

8.6.5 Human evaluation

We evaluate the agents trained using simulated users against real users, recruited from the authors' affiliation. We conducted the study using the DQN and BBQN-MAP agents. In the full-domain setting, the agents were trained with all the slots. In the domain-extension setting, we first picked DQN (b-DQN) and BBQN (b-BBQN) agents before the domain extension at training epoch 40 and the performance of these two agents is tied, nearly 45% success rate. From training epoch 40, we started to introduce new slots, and we selected another two agents (a-DQN and a-BBQN) at training epoch 200. In total, we compare three agent pairs: {DQN, BBQN} for full domain, {b-DQN, b-BBQN} from before domain extension, and {a-DQN, a-BBQN} from after domain extension. In the real user study, for each dialogue session, we select one of six agents randomly to converse with a user. We present the user with a user goal sampled from our corpus. At the end of each dialogue session, the user was asked to give a rating on a scale from 1 to 5 based on the naturalness, coherence, and task-completion capability of the agent (1 is the worst rating, 5 is the best). In total, we collected 398 dialogue sessions. Figure 8.4a presents the performance of these agents against real users in terms of success rate. Figure 8.4b shows the comparison in user ratings. In the full-domain setting, the BBQN agent is significantly better than the DQN agent in terms of success rate and user rating. In the domain-extension setting, before domain extension, the performance of both agents (b-DQN and b-BBQN) is tied; after domain extension, the BBQN (a-BBQN) agent significantly outperforms the DQN (a-DQN) in terms of success rate and user rating.



(a) Distribution of Success Rate



(b) Distribution of User Ratings

Figure 8.4: Performance of BBQN agent versus DQN agent tested with real users, number of tested dialogues and p-values are indicated on each bar (difference in mean is significant with $p < 0.05$).

8.7 Related work

This work touches on several areas of research, namely Bayesian neural networks, reinforcement learning with deep Q-networks, Thompson Sampling, and dialogue systems. This work employs Q-learning [WD92a], a popular method for model-free RL. For a broad resource on RL, we point to [SB98]. Recently, [MKS$^+$15] achieved super-human performance on Atari games using deep Q-learning and incorporating techniques such as experience replay [Lin92].

Efficient exploration remains one of the defining challenges in RL. While provably efficient exploration strategies are known for problems with finite states/actions or problems with *nice* structures [Kak03, ALL$^+$09, JOA10, LLWS11, ORR13], less is known for the general case, especially when general nonlinear function approximation is used. The first DQN papers relied upon the ε-greedy exploration heuristic [MKS$^+$15]. More recently, [SLA15] and [HCD$^+$16] introduced approaches to encourage exploration by perturbing the reward function. [OBPVR16] attempts to mine uncertainty information by training a neural network with multiple output *heads*. Each head is associated with a distinct subset of the data. This works for some Atari games, but does not confer a benefit for us. [CL11] empirically examine Thompson sampling, one of the oldest exploration heuristics [Tho33], for contextual bandits, which is later shown to be effective for solving finite-state MDPs [Str00, ORR13].

We build on the Bayes-by-backprop method of [BCKW15], employing the reparameterization trick popularized by [KW13], and following a long history of variational treatments of neural networks [HVC93, Gra11]. After we completed this work, [KPR$^+$17] independently investigated parameter uncertainty for deep Q-networks to mitigate catastrophic forgetting issues. [BCKW15] consider Thompson sampling for contextual bandits, but do not consider the more challenging case of MDPs. This work also builds on prior work in task-oriented dialogue systems [WY04, GJK$^+$10, WGM$^+$16] and RL for learning dialogue policies [LPE97, SKLW00, WY07, GJK$^+$10, FEAS$^+$16]. Our domain-extension experiments take inspiration from [GKT$^+$14] and our user simulator is modeled on [STY07].

8.8 Conclusions

For learning dialogue policies, BBQNs explore with greater efficiency than traditional approaches. The results are similarly strong for both static and domain extension experiments in simulation and real human evaluation. Additionally, we showed that we can benefit from combining BBQ-learning with other, orthogonal approaches to exploration, such as those that perturb the reward function to add a bonus for uncovering surprising transitions, i.e., state transitions given low probability by a dynamics model, or previously rarely seen states [SLA15, HCD$^+$16, BSO$^+$16]. Our BBQN addresses uncertainty in the Q-value given the current policy, whereas curiosity addresses uncertainty of the dynamics of under-explored parts of the environment. Thus there is a synergistic effect of combining the approaches. On the domain extension task, BBQN-VIME proved especially promising, outperforming all other methods. We see several promising paths for future work. Notably, given the substantial improvements of BBQNs over other exploration strategies, we would like to extend this work to popular deep reinforcement learning benchmark tasks (Atari, etc.) and other domains, like robotics, where the cost of exploration is high, to see if it confers a comparably dramatic improvement.

8.9 Acknowledgments

Chapter 8, *Efficient Dialogue Policy Learning with BBQ-Networks* was written in collaboration with Xiujun Li, Lihong Li, Jianfeng Gao, Faisal Ahmed, and Li Deng. The dissertation author was the primary investigator and author of this paper.

Part III

Critical Considerations - Safety, Interpretability, and Fairness

Chapter 9

Combating Deep Learning's Sisyphean Curse with Intrinsic Fear

Many practical reinforcement learning problems contain catastrophic states that the optimal policy visits infrequently or never. Even on toy problems, deep reinforcement learners periodically revisit these states, once they are forgotten under a new policy. In this chapter, we introduce *intrinsic fear*, a learned *reward shaping* that accelerates deep reinforcement learning and guards oscillating policies against periodic catastrophes. Our approach incorporates a second model trained via supervised learning to predict the probability of imminent catastrophe. This score acts as a penalty on the Q-learning objective. Our theoretical analysis demonstrates that the perturbed objective yields the same average return under strong assumptions and an ϵ-close average return under weaker assumptions. Our analysis also shows robustness to classification errors. Equipped with intrinsic fear, our DQNs solve the toy environments and improve on the Atari games Seaquest, Asteroids, and Freeway.

9.1 Introduction

Following success on Atari games [MKS$^+$15] and the board game Go [SHM$^+$16], many researchers have begun exploring practical applications of deep reinforcement learning (DRL). Some investigated applications include robotics [Lea16a], dialogue systems [FEAS$^+$16, Lea16b], energy management [Nig16], and self-driving cars [SSSS16]. Amid this push to apply DRL, we might ask, *can we trust these agents in the wild?* Agents acting in real-world environments might possess the ability to cause catastrophic outcomes. Consider a self-driving car that might hit pedestrians or a domestic robot that might injure a child. We might hope to prevent DRL agents from ever making catastrophic mistakes. But doing so requires extensive prior knowledge of the environment in order to constrain the exploration of policy space [GF15].

Many conflicting definitions of safety and catastrophe exist, a problem that invites further philosophical consideration. In this chapter, we introduce a specific but plausible notion of *avoidable catastrophes*. These are states that prior knowledge dictates an optimal policy should never visit. For example, we might believe that an optimal self-driving algorithm would never hit a pedestrian. Moreover, we assume that an optimal policy never even comes *near* an avoidable catastrophe state. We define proximity in trajectory space, and not by the geometry of feature space. We denote states proximal to avoidable catastrophes as *danger states*. While we don't assume prior knowledge of which states are dangerous, we do assume the existence of a *catastrophe detector*. After encountering a catastrophic state, an agent can realize this and take action to avoid dangerous states in the future.

Given this definition, we address two challenges: First, can we expect DRL agents, after experiencing some number of catastrophic failures, to avoid perpetually making the same mistakes? Second, can we use our prior knowledge that catastrophes should be kept at a distance to accelerate learning of a DRL agent? Our experiments show that even on toy problems, the deep Q-network (DQN), a basic algorithm behind many of today's state-of-the-art DRL systems, struggles on both counts. Even in toy environments, DQNs may encounter thousands of catastrophes before learning to avoid them and are susceptible to repeating old errors. We call this latter problem *the Sisyphean curse*.

This poses a formidable obstacle to using DQNs in the real world. How can we hand over responsibility for consequential actions (control of a car, say) to a DRL agent if it may be doomed to periodically remake every kind of mistake, however grave, so long as it continues to learn? Imagine a self-driving car that had to periodically hit a few pedestrians in order to remember that is undesirable. In the tabular setting, an RL agent never forgets the learned dynamics of its environment, even as its policy evolves. Moreover, if the Markovian assumption holds, eventual convergence to a globally optimal policy is guaranteed. Unfortunately, the tabular approach becomes infeasible in high-dimensional, continuous state spaces.

The trouble for DQNs owes to the use of function approximation [MO05]. When training a DQN, we successively update a neural network based on experiences. These experiences might be sampled in an online fashion, from a trailing window (*experience replay buffer*), or uniformly from all past experiences. Regardless of which mode we use to train the network, eventually, states that a learned policy never encounters will come to form an infinitesimally small region of the training distribution. At such times, our networks are subject to the classic problem of catastrophic interference [MC89, MMO95]. Nothing prevents the DQN's policy from drifting back towards a policy that revisits long-forgotten catastrophic mistakes.

More formally, we characterize the problem as unfolding in the following steps: (i) Training under distribution $\mathcal{D}$, our agent produces a safe policy π_s that avoids catastrophes

(ii) Collecting data generated under π_s yields a new distribution of transitions $\mathcal{D}'$ (iii) Training under $\mathcal{D}'$, the agent produces π_d, a policy that once again experiences avoidable catastrophes. To illustrate the brittleness of modern DRL algorithms, we introduce a simple pathological problem called *Adventure Seeker*. This problem consists of a one-dimensional continuous state, two actions, simple dynamics, and a clear analytic solution. Nevertheless, the DQN fails. We then show that similar dynamics exist in the classic RL environment Cart-Pole.

In this chapter, to combat these problems, we propose *intrinsic fear*. In this approach, we train a supervised *fear model* that predicts which states are likely to lead to a catastrophe within k_r steps. The output of the fear model (a probability), scaled by a *fear factor* penalizes the Q-learning target. Our approach draws inspiration from intrinsic motivation [CBS04]. However, instead of perturbing the reward function to encourage the discovery of novel states, we perturb it to discourage revisiting catastrophic states.

We validate the approach both empirically and theoretically. Our experiments address both our *Adventure Seeker* problem and Cartpole as well as the Atari games Seaquest, Asteroids, and Freeway. For these environments, we label each loss of a *life* as a catastrophic state. On the toy environments, the intrinsic fear agent learns to avoid death indefinitely, achieving unbounded reward per episode. On Seaquest and Asteroids, the intrinsic fear agent improves markedly and on Freeway the improvement is dramatic. Theoretically, we demonstrate the following: First, we prove that when the reward is bounded and the optimal policy rarely visits the catastrophic states, the policy learned on the altered value function has return similar to the optimal policy on the original value function. Second we prove that the method is robust to noise in the danger model.

9.2 Intrinsic fear

Over a series of turns, an agent interacts with its environment via a Markov decision process, or MDP, $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. At each step t, an agent observes a state $s \in \mathcal{S}$. The agent then chooses an action $a \in \mathcal{A}$ according to some policy π. In turn, the environment transitions to a new state $s_{t+1} \in \mathcal{S}$ according to transition dynamics $\mathcal{T}(s_{t+1}|s_t, a_t)$ and generates a reward r_t with expectation $\mathcal{R}(s,a)$. This cycle continues until each episode terminates.

The goal of an agent is to maximize the cumulative discounted return $\sum_{t=0}^{T} \gamma^t r_t$. Temporal-differences (TD) methods [Sut88] such as Q-learning [WD92b] model the Q-function, which gives the *optimal* discounted total reward of a state-action pair; the greedy policy w.r.t. the Q-function is optimal [SB98]. Problems of practical interest tend to have large state spaces, thus the Q-function is typically approximated by parametric models such as neural networks.

In Q-learning with function approximation, an agent alternately collects experiences by acting greedily with respect to $Q(s,a;\theta_Q)$ and updates its parameters θ_Q. Updates proceed as follows. For a given experiences (s_t, a_t, r_t, s_{t+1}), we minimize the squared Bellman error:

$$\mathcal{L} = (Q(s_t, a_t; \theta_Q) - y_t)^2 \tag{9.1}$$

for $y_t = r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'; \theta_Q)$. Traditionally, the parameterised $Q(s,a;\theta)$ is trained by stochastic approximation, estimating the loss on each experience as it is encountered, yielding the update:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(y_t - Q(s_t, a_t; \theta_t))\nabla Q(s_t, a_t; \theta_t)\,. \tag{9.2}$$

Q-learning methods also require an exploration strategy for action selection. For simplicity, we consider only the ε-greedy heuristic.

A few tricks help to stabilize Q-learning with function approximation. Of particular relevance to this work is experience replay [Lin92]: the RL agent maintains a buffer of past experiences, applying TD-learning on randomly selected mini-batches of experience to update the Q-function.

In this chapter, we propose a new formulation of the safety problem. We suppose there exists a subset $\mathcal{C} \subset \mathcal{S}$ of states that an optimal policy encounters them very rarely or never and denote them *catastrophic states*. Moreover, we assume that for some environments, optimal policies are rarely within a short distance of a catastrophic state. As a measure of distance, we consider steps in trajectory space. We define the distance $d(s_i, s_j)$ to be length N of the smallest sequence of transitions $\{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^{N}$ that traverses state space from s_i to s_j.[1]

Definition 9.2.1. Suppose that we are given a priori knowledge that acting according to the optimal policy π^*, an agent never encounters states $s \in S$ for which lie within distance $d(s,c) < k_\tau$ for any catastrophe state $c \in \mathcal{C}$. Then each state s for which $\exists c \in \mathcal{C}$ s.t. $d(s,c) < k_\tau$ is a *danger state*.

We also suppose that the agent can recognize the catastrophe states as they are encountered.

Definition 9.2.2. A *catastrophe detector* is a function $f : \mathcal{S} \mapsto \{0,1\}$ that returns 1 if and only if a state is a catastrophe state.

We propose Intrinsic Fear (IF) (Algorithm 1), a novel algorithm for avoiding catastrophes when learning online with function approximation. In our approach, we maintain both a DQN and a separate, supervised *fear model* $F : \mathcal{S} \mapsto [0,1]$. Our fear

[1]In the stochastic dynamics setting, the distance is the minimum mean passing time between the states.

model F provides an auxiliary source of reward, penalizing the Q-learner for entering possibly dangerous states.

The goal in modeling danger states is twofold. First, by shaping rewards away from suboptimal states, we encode prior knowledge about the environment and can thus accelerates learning. Second, when catastrophic states correspond to especially undesirable outcomes, the learned reward shaping can protect DQNs, which are susceptible to catastrophic forgetting, from drifting close to catastrophic states. Owing to this self-assigned reward, once the fear model is trained, a Q-learner might update to avoid catastrophes without having to actually repeat them, so long as the fear model is not itself susceptible to catastrophic forgetting. We draw some inspiration from the idea of a parent scolding a child for running around with a knife. The child can learn to adjust its behavior without actually having to stab someone. We also draw inspiration from the way humans appear to process traumatic experience, remembering especially bad events vividly even as most other memories from the same time period fade. Perhaps this selective memorization of bad events confers a benefit for avoiding similar outcomes in the future.

Our instantiation of intrinsic fear works as follows: In addition to the DQN, we maintain a binary classifier that we term a *fear model*. In our case, we use a neural network of the same architecture as the DQN (but for the output layer). The fear model's purpose is to predict the probability that any state will lead to catastrophe within k moves. Over the course of training, our agent adds each experience (s, a, r, s') to its experience replay buffer. As each catastrophe is reached at the n_{th} turn of an episode, we add the k_r (*fear radius*) states leading up to the catastrophe to a list of *danger states*. We add the preceding $n - k_r$ states to a list of *safe states*. When $n < k_r$, all states for that episode are added to the list of danger states. Then after each turn, in addition to making one update to the Q-network, we make one mini-batch update to the fear model. To make

Algorithm 1 Training DQN with Intrinsic Fear

```
1: Input: Two models: $Q$ (DQN) and $F$ (fear model), fear factor $\lambda$, fear phase-in length $k_\lambda$, fear radius $k_r$
2: Output: Learned parameters $\theta_Q$ and $\theta_F$
3: Initialize parameters $\theta_Q$ and $\theta_F$ randomly
4: Initialize replay buffer $\mathcal{D}$, danger state buffer $\mathcal{D}_D$, and safe state buffer $\mathcal{D}_S$
5: Start per-episode turn counter $n_e$
6: for $t$ in 1:$T$ do
7:     With probability $\varepsilon$ select random action $a_t$
8:     Otherwise, select action $a_t = argmax_{a'} Q(s_t, a'; \theta_Q)$
9:     Execute action $a_t$ in environment, observing reward $r_t$ and successor state $s_{t+1}$
10:    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
11:    if $s_{t+1}$ is a catastrophe state then
12:        Add states $s_{t-k_r}$ through $s_t$ to $\mathcal{D}_D$
13:    else
14:        Add states $s_{t-n_e}$ through $s_{t-k_r-1}$ to $\mathcal{D}_S$
15:    end if
16:    Sample random minibatch of transitions ($s_\tau$ , $a_\tau$, $r_\tau$, $s_{\tau+1}$) from $\mathcal{D}$
17:    $\lambda_\tau \leftarrow \min(\lambda, \frac{\lambda \cdot t}{k_\lambda})$
18:    $y_\tau \leftarrow \left\{ \begin{array}{ll} r_\tau - \lambda_\tau, & \text{for terminal } s_{\tau+1} \\ r_\tau + \max_{a'} Q(s_{\tau+1}, a'; \theta_Q) - \lambda \cdot F(s_{\tau+1}; \theta_F) & \text{for non-terminal } s_{\tau+1} \end{array} \right\}$
19:    $\theta_Q \leftarrow \theta_Q - \eta \cdot \nabla_{\theta_Q} (y_\tau - Q(s_\tau, a_\tau; \theta_Q))^2$
20:    Sample random mini-batch $s_j$ with 50% of examples from $\mathcal{D}_D$ and 50% from $\mathcal{D}_S$
21:    $y_j \leftarrow \left\{ \begin{array}{ll} 1, & \text{for } s_j \in \mathcal{D}_D \\ 0, & \text{for } s_j \in \mathcal{D}_S \end{array} \right\}$
22:    $\theta_F \leftarrow \theta_F - \eta \cdot \nabla_{\theta_F} \text{loss}_F (y_j, F(s_j; \theta_F))$
23: end for
```

this update, we sample 50% of samples in the batch from the *danger states*, assigning them label 1 and the remaining 50% from the *safe states*, assigning them label 0.

For each update to the DQN, we perturb the TD target y_t. Instead of updating $Q(s_t, a_t; \theta_Q)$ towards $r_t + \max_{a'} Q(s_{t+1}, a'; \theta_Q)$, we introduce the *intrinsic fear* to the model via the target:

$$y_t^{IF} = r_t + \max_{a'} Q(s_{t+1}, a'; \theta_Q) - \lambda \cdot F(s_{t+1}; \theta_F) \tag{9.3}$$

where $F(s; \theta_F)$ is the fear model and λ is a *fear factor* determining the scale of the impact of intrinsic fear on the Q-function update.

Note that IF perturbs the objective function. Thus, one might be concerned that the perturbed reward might indicate a different optimal policy. Fortunately, if the labeled catastrophe states and danger zone do not violate our assumptions, and if the fear model reaches arbitrarily high accuracy, then this will not happen.

For an MDP, $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, with $0 \leq \gamma \leq 1$, the average reward return is as follows:

$$\eta_M(\pi) := \begin{cases} \lim_{T \to \infty} \frac{1}{T} \mathbb{E}_M \left[\sum_t^T r_t | \pi \right] & \textit{if} \quad \gamma = 1 \\ (1-\gamma) \mathbb{E}_M \left[\sum_t^\infty r_t | \pi \right] & \textit{if} \quad 0 \leq \gamma < 1 \end{cases} \tag{9.4}$$

The optimal policy π^* of the model M is the policy which maximizes the average reward return, $\pi^* = \max_{\pi \in \mathcal{P}} \eta(\pi)$ where $\mathcal{P}$ is a set of stationary polices.

Theorem 5. *For a given MDP, M, with $\gamma \in [0, 1]$ and a catastrophe detector f, let π^* denote the optimal policy of model M and $\tilde{\pi}$ denote the optimal policy of model M equipped with fear model F. If the cost of λ prevents $\tilde{\pi}$ from visiting danger zone and the* *probability* π^* *visits the states in the danger zone is less than* ε, *and* $\mathcal{R}_{\min} \leq \mathcal{R}(s,a) \leq \mathcal{R}_{\max}$, *then*

$$\eta^*_M - \varepsilon(\mathcal{R}_{\max} - \mathcal{R}_{\min}) \leq \eta_M(\tilde{\pi}) \leq \eta^*_M. \tag{9.5}$$

At the same time, the average return of the optimal policy π^* *on the environment with intrinsic reward* $\eta_{M,F}(\pi^*)$ *is bounded as*

$$\eta_{M,F}(\tilde{\pi}) - \lambda\varepsilon(\mathcal{R}_{\max} - \mathcal{R}_{\min}) \leq \eta_{M,F}(\pi^*) \leq \eta_{M,F}(\tilde{\pi}).$$

Proof. Appendix 9.5

Since we learn the catastrophe detector f and fear model F empirically using the collected data, our RL agent has access to an imperfect detector $\hat{f}$ and imperfect fear model $\hat{F}$, and therefore assumes the fear model is $\hat{F}$. In this case, the RL agent trains with intrinsic fear generated by $\hat{f}$, learning a different value function than the RL agent with perfect f. To show robustness against modeling errors, we are interested in the average deviation in the value functions of the two agents.

In general, in practical RL problems, we use discount factors $\gamma < 1$ [KS06] in order to reduce the planing horizon, and computation cost. Moreover, [JKSL15] suggests that when we have estimation (up to the confidence intervals) of our MDP model, it is better to use smaller discount factors in order to the estimated model from overfitting. We show that under modeling errors, if the actual objective function to optimize for Eq. 9.4 has discount factor γ_{eval}, it's better to use some $\gamma \leq \gamma_{eval}$ because it reduces the average deviation in the value functions.

For a given environment, with fear model F_1 and discount factor γ_1, let $V^{\pi^*_{F_2,\gamma_2}}_{F_1,\gamma_1}(s)$, $s \in$

$\mathcal{S}$, denote the state value function under the optimal policy of a environment with fear model F_2 and the discount factor γ_2. On the same environment, let $\omega_{F_1}^{\pi^*_{F_2,\gamma_2}}(s)$ denote the stationary distribution over states. Therefore we are interested in the average deviation on value functions caused by an imperfect classifier:

$$\mathcal{L}(F,\widehat{F},\gamma_{eval},\gamma) := (1-\gamma_{eval})\int_{s\in\mathcal{S}} \omega_F^{\pi^*_{\widehat{F},\gamma}}(s)\left|V_{F,\gamma_{eval}}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma_{eval}}^{\pi^*_{\widehat{F},\gamma}}(s)\right| ds$$

Theorem 6. *For a given MDP model, the average deviation on the value functions, $\mathcal{L}(F,\widehat{F},\gamma_{eval},\gamma)$, $F,\hat{F}\in\mathcal{F}$, vanishes as the number of samples N increases*

$$\mathcal{L} = O\left((\lambda+\mathcal{R}_{\max}-\mathcal{R}_{\min})\frac{1-\gamma_{eval}}{1-\gamma}\frac{\mathcal{VC}(\mathcal{F})+\log\frac{1}{\delta}}{N}+\frac{\gamma_{eval}-\gamma}{1-\gamma}\right) \tag{9.6}$$

with probability at least $1-\delta$ *where* $\mathcal{VC}(\mathcal{F})$ *is the* $\mathcal{VC}$ *dimension of the hypothesis class* $\mathcal{F}$.

Proof. Appendix 9.6

Thm. 6 holds for both tabular MDPs and continuous state-action MDPs. In addition to proofs of these results, we provide a deeper theoretical analysis on deterministic and stochastic fear models in the tabular setting in Appendix 9.6

Over the course of our experiments, we discovered the following pattern: Intrinsic fear models are more effective when the fear radius k_r is large enough that the model can experience danger states at a safe distance and correct the policy, without experiencing many catastrophes. When the fear radius is too small, the danger probability is only nonzero at states from which catastrophes are inevitable anyway and intrinsic fear seems not to help. We also found that wider fear factors train more stably when phased in over the course of many episodes. So, in all of our experiments we gradually phase in the *fear factor* λ from 0 to λ reaching full strength at predetermined time step k_λ. In our

Cart-Pole experiments, we phase λ in over $1M$ steps.

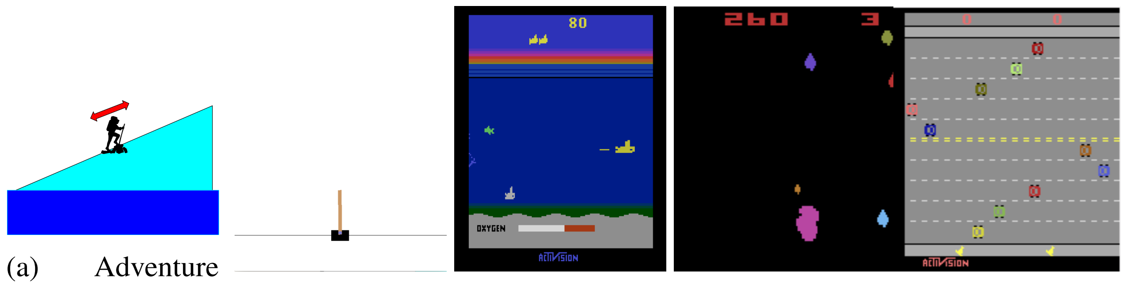
9.3 Environments

We demonstrate our algorithms on three environments. These include *Adventure Seeker*, a toy pathological environment which we designed to demonstrate the Sisyphean curse; *Cartpole*, a classic reinforcement learning environment; and three Atari games, *Seaquest*, *Asteroids*, and *Freeway*, simulated in the Arcade Learning Environment [BNVB13].

9.3.1 Adventure Seeker

We imagine a player placed on a hill, sloping upward to the right (Figure 9.1a). At each turn, the player can move to the right (up the hill) or left (down the hill). The environment adjusts the player's position accordingly, adding some random noise. Between the left and right edges of the hill, the player gets more reward for spending time higher on the hill. But if the player goes too far to the right, he/she will fall off (a *catrastrophic state*), terminating the episode and receiving a return of 0. Formally, the state consists of a single continuous variable $s \in [0, 1.0]$, denoting the player's position. The starting position for each episode is chosen uniformly at random in the interval $[.25, .75]$. The available actions consist only of $\{-1, +1\}$ (*left* and *right*). Given an action a_t in state s_t, $\mathcal{T}(s_{t+1}|s_t, a_t)$ gives successor state $s_{t+1} \leftarrow s_t + .01 \cdot a_t + \eta$ where $\eta \sim \mathcal{N}(0, .01^2)$. The reward at each turn is equal to s_t (proportional to height). The player falls off the hill, entering the catastrophic terminating state, whenever $s_{t+1} > 1.0$ or $s_{t+1} < 0.0$.

This game admits an analytic solution: there exists some threshold above which the agent should always choose to go left, and below which it should always go right. And



(a) Adventure Seeker (b) Cart-Pole (c) Seaquest (d) Asteroids (e) Freeway

Figure 9.1: In experiments, we consider two toy environments (a,b) and the Atari games Seaquest (c), Asteroids (d), and Freeway (e)

yet a state-of-the-art DQN model learning online or with experience replay successively plunges to its death. To be clear, the DQN does learn a near-optimal thresholding policy quickly. But over the course of continued training, the agent oscillates between a reasonable thresholding policy and one which always moves right, regardless of the state. The pace of this oscillation evens out and all networks (over multiple runs) quickly reach a constant catastrophe per turn rate that does not attenuate with continued training. How could we trust a system that can't solve *Adventure Seeker* to make consequential decisions?

9.3.2 Cart-Pole

In this classic RL environment, an agent balances a pole atop a cart (Figure 9.1b). Qualitatively, the game exhibits four distinct catastrophe modes. The pole could fall down to the right or fall down to the left. Additionally, the cart could run off the right boundary of the screen or run off the left. Formally, at each time, the agent observes a four-dimensional state vector (x, v, θ, ω) consisting respectively of the cart position, cart velocity, pole angle, and the pole's angular velocity. At each time step, the agent chooses an action, applying a force of either -1 or $+1$. For every time step that the pole remains upright and the cart remains on the screen, the agent receives a reward of 1. If the pole falls, the episode terminates, giving a return of 0 from the penultimate state. In experiments, we use the implementation *CartPole-v0* contained in the openAI gym [Bea16]. Like Adventure Seeker, this problem admits an analytic solution. A perfect policy should never drop the pole. But, as with Adventure Seeker, a DQN converges to a constant rate of catastrophes per turn.

9.3.3 Atari games

In addition to these pathological cases, we address Freeway, Asteroids, and Seaquest, games from the Atari Learning Environment. In Freeway, the agent controls a chicken with a goal of crossing the road while dodging traffic. The chicken loses a life and starts from the original location if hit by a car. Points are only rewarded for successfully crossing the road. In Asteroids, the agent pilots a ship and gains points from shooting the asteroids. She must avoid colliding with asteroids which cost it lives. In Seaquest, a player swims under water. Periodically, as the oxygen gets low, she must rise to the surface for oxygen. Additionally, fishes swim across the screen. The player gains points each time she shoots a fish. Colliding with a fish or running out of oxygen result in death. In all three games, the agent has 3 lives, and the final death is a terminal state. We label each loss of a life as a catastrophe state.

9.4 Experiments

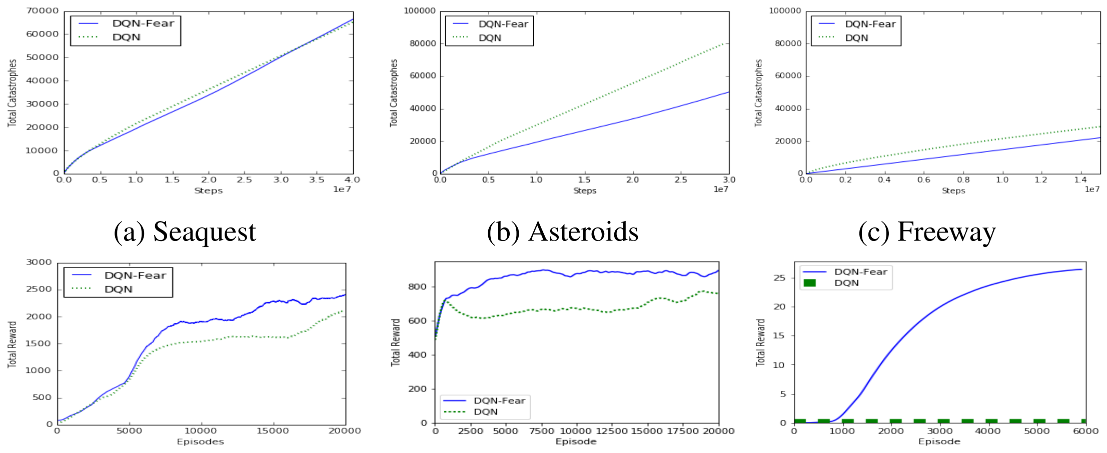
To assess the effectiveness of the intrinsic fear model, we evaluate both a standard DQN (DQN-NoFear) and one enhanced by *intrinsic fear* (DQN-Fear). In both cases, we use multilayer perceptrons (MLPs) with a single hidden layer and 128 hidden nodes. We train all MLPs by stochastic gradient descent using the Adam optimizer [KB15] to adaptively tune the learning rate.

Because, for the *Adventure Seeker* problem, an agent can escape from danger with only a few time steps of notice, we set the fear radius k_r to 5. We phase in the fear factor quickly, reaching full strength in just 1000 moves. On this problem we set the fear factor λ to 40.

For *Cart-Pole*, we set a wider fear radius of $k_r = 20$. We initially tried training this model with a shorter fear radius but made the following observation. Some models would learn well surviving for millions of experiences, with just a few hundred catastrophes. This compared to a DQN (Figure 9.2) which would typically suffer 4000-5000 catastrophes. When examining the output from the fear models on successful vs unsuccessful runs, we noticed that the unsuccessful models would output danger of probability greater than .5 for precisely the 5 moves before a catastrophe. But by that time it would be too late for an agent to correct course. In contrast, on the more successful runs, the fear model typically outputs predictions in the range $.1 - .5$. We suspect that the gradation between mildly dangerous states and those with imminent danger provides a richer reward signal to the DQN.

On both the Adventure Seeker and Cart-Pole environments, the DQNs augmented by intrinsic fear far outperform their otherwise identical counterparts (Figure 9.2). We cannot plot the reward per episode for the intrinsic fear models on these environments because after the first several deaths, the episodes never terminate. In contrast, both the DQN and related approaches like expected SARSA continue to visit the catastrophic states regularly. We compared our approach against some traditional approaches for mitigating catastrophic forgetting. For example, we tried a memory-based method in which we preferentially sample the catastrophic states for updating the model, but they did not improve over the DQN. It seems that the notion of a danger zone is necessary here.

For Seaquest, Asteroids, and Freeway, we use a fear radius of 5 and a fear factor of .5. For all Atari games, the IF models outperform their DQN counterparts. Interestingly



(a) Seaquest (b) Asteroids (c) Freeway

(d) Seaquest (reward) (e) Asteroids (reward) (f) Freeway (reward)

Figure 9.2: On Seaquest, the IF model achieves a similar catastrophe rate but significantly higher total reward. On Asteroids, the IF model outperforms DQN. For Freeway, a randomly exploring DQN (under our time limit) never gets reward but IF model learns successfully.

while for all games, the IF models achieve higher reward, on Seaquest, models trained with Intrinsic Fear have similar catastrophe rates. More precisely, they appear to have fewer catastrophes early on but eventually enter a different reward regime, exchanging more catastrophes for higher reward. This result suggests an interplay between the various reward signals that warrants further exploration. For Asteroids and Freeway, the improvements are more dramatic. Over just a few thousand episodes of Freeway, a randomly exploring DQN achieves zero reward. However, the reward shaping of intrinsic fear leads to rapid improvement.

9.5 Loss in Optimal Value

The average return of the reward under a policy π is as follows:

$$\eta_M(\pi) = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_t^T r_t|\pi\right] \tag{9.7}$$

Let's assume that any stationary policy π, induces a stationary distribution $\omega(s),\ s \in \mathcal{S}$. Therefore we can rewrite Eq. 9.7 in terms of stationary distribution [Put14].

$$\eta_M(\pi) = \lim_{T\to\infty} \mathbb{E}\left[\sum_t r_t|\pi\right] = \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \omega(s)\pi(a|s)\mathcal{R}(s,a)$$

In RL, we are interested in a policy π^* which maximize the the expected average reward.

$$\pi^* := arg \max_{\pi} \eta_M(\pi)$$

where $\eta^*_M = \eta_M(\pi^*)$. In a first place, the optimization in Eq. 9.7 looks linear in π but actually the policy π derives the stationary distribution $\omega(\cdot)$, which makes the optimization problem a bit harder.

Given the policy π, let's define the joint distribution in (s,a) as follows:

$$\mu_\pi(s,a) := \mathbb{P}(s,a|\pi) = \omega(s)\pi(a|s),\ \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Then we can rewrite the optimization problem in terms of the joint probability distribution μ_π.

$$\eta_m(\mu_\pi) :== \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \mu_\pi(s,a)\mathcal{R}(s,a) \tag{9.8}$$

We can see that this new formalization, makes our optimization problem as linear function of μ_π. Since μ_π is join distribution of (s,a) under the model dynamics $\mathcal{T}$ it can not take any arbitrary value. Let Δ denote the set of feasible value for μ_π,

$$\Delta := \{\mu_\pi(s,a) : \sum_{a'} \mu(s',a') = \sum_{s,a} \mathcal{T}(s'|s,a)\mu_\pi(s',a')\} \tag{9.9}$$

Since $\sum_{s,a}\mu_\pi(s,a) = 1$ therefore Δ is a polytope on the simplex in $\mathbb{R}^{S\times A}$

Now, we can rewrite the optimization problem Eq. 9.7 as an constraint linear programing on μ_π

$$\eta_M^* = \max_{\mu_\pi \in \Delta} \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mu_\pi(s,a)\mathcal{R}(s,a)$$

This change of variable allows us to analyze the introduction of intrinsic fear in different situations.

$(i)-$ If the probability that at any time step the optimal agent happens to be in the danger zone, $\sum_{s \in \mathcal{C},a} \mu_{\pi^*}(s,a)$, is negligible, and the intrinsic fear reward assigned to the states in this zone $\mathcal{C}$ is negative, then optimal policy in the original environment (without intrinsic reward) is the same as the optimal policy in the model with intrinsic reward. Moreover, the long term average reward provided under these models are same. (The intrinsic fear helps to learn the optimal behavior faster in the RL framework).

$(ii)-$ Now, we consider the situation where the $\sum_{s \in \mathcal{C},a} \mu_{\pi^*}(s,a)$ is not negligible, but less than ε. In this situation, let's assume that the negative reward assigned to the states in the danger zone is λ and the optimal policy $\tilde{\pi}$ of the environment with the intrinsic fear has return of $\eta_{M,F}(\tilde{\pi})$.

If there exists a cost λ such that the $\sum_{s \in \mathcal{C},a} \mu_{\tilde{\pi}}(s,a)$ under $\tilde{\pi}$ in either environments is negligible, since the set Δ is a convex polytope and $\mathcal{R}_{\min} \leq \mathcal{R}(s,a) \leq \mathcal{R}_{\max}$, then

$$\eta_M^* \geq \eta_M(\tilde{\pi}) = \eta_M(\tilde{\pi}) \geq \eta_M^* - \varepsilon(\mathcal{R}_{\max} - \mathcal{R}_{\min}). \tag{9.10}$$

At the same time, the average return of the optimal policy π^* on the environment with intrinsic fear, $\eta_{M,F}(\pi^*)$ is bounded as

$$\eta_{M,F}(\tilde{\pi}) \geq \eta_{M,F}(\pi^*) \geq \eta_{M,F}(\tilde{\pi}) - \lambda\varepsilon(\mathcal{R}_{\max} - \mathcal{R}_{\min}). \tag{9.11}$$

9.5.1 Discounted Cumulative Reward

For the γ-discounted setting, we are interested in

$$\eta(\pi) = (1-\gamma)\lim_{T\to\infty}\mathbb{E}\left[\sum_{t=0}\gamma^t r_t\right] \tag{9.12}$$

Both of the above mentioned equations hold in this setting, *i.e.*, $\eta^*_M \geq \eta_{M,F}(\tilde{\pi}) = \eta_M(\tilde{\pi}) \geq \eta^*_M - \varepsilon(\mathcal{R}_{\max} - \mathcal{R}_{\min})$, and $\eta_{M,F}(\tilde{\pi}) \geq \eta_{M,F}(\pi^*) \geq \eta_{M,F}(\tilde{\pi}) - \lambda\varepsilon(\mathcal{R}_{\max} - \mathcal{R}_{\min})$.

9.6 Imperfect Classifier

In the previous section, we assumed that we have access to the perfect classifier F which can exactly label the danger zone. This assumption does not hold in real world where we train the classifier. In this section we derive an analysis in order to show that imperfect classifier $\widehat{F}$ can not change the overall performance by much.

In general, in practical RL problems, we use discount factors $\gamma_{eval} < 1$ [KS06] in order to reduce the planing horizon, and computation cost. Moreover, [JKSL15] suggest that when we have an estimation (up to the confidence intervals) of our MDP model, it is better to use $\gamma \leq \gamma_{eval}$. They show that since larger discount factor enriches the class of optimal policies for a given set of plausible models, large discount factors enrich models and end up over fitting to the noisy estimate of the environment.

In this section, we show how to choose the discount factor $\gamma \leq \gamma_{eval}$ such that the learned value function stays close to the value function under the perfect classifier F. Let, $V^{\pi^*_{F_1,\gamma_1}}_{F_2,\gamma_2}(s)$, $s \in \mathcal{S}$, denote the state value under the optimal policy of model with classifier F_1 under the discount factor γ_1 on the environment equipped with classifier F and discount factor γ_2. On the same environment, $\omega^{\pi^*_{F_1,\gamma_1}}_{F_2}(s)$ denotes the stationary distribution over states. We are interested in the average deviation on value functions caused by the imperfect classifier:

$$\mathcal{L} := (1-\gamma_{eval}) \sum_{s\in\mathcal{S}} \omega_F^{\pi^*_{\widehat{F},\gamma}}(s) \left| V_{F,\gamma_{eval}}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma_{eval}}^{\pi^*_{\widehat{F},\gamma}}(s) \right|$$

This quantity can be upper bounded by

$$\mathcal{L} \leq (1-\gamma_{eval}) \| V_{F,\gamma_{eval}}^{\pi^*_{F,\gamma_{eval}}} - V_{F,\gamma_{eval}}^{\pi^*_{\widehat{F},\gamma}} \|_\infty \tag{9.13}$$

The goal is to find an γ^* which minimizes this loss, i.e. $\gamma^* = argmin_{\gamma\leq\gamma_{eval}}\mathcal{L}$ with high probability. For simplicity and without loss of generality, let's assume that all the rewards adding intrinsic fears are in $[0,1]$ and call λ', the transformed version of λ [2]. One can decompose the upper bound in Eq. 9.13 as follows:

$$V_{F,\gamma_{eval}}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma_{eval}}^{\pi^*_{\widehat{F},\gamma}}(s) = \left(V_{F,\gamma_{eval}}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma}^{\pi^*_{F,\gamma_{eval}}}(s) \right) + \left(V_{F,\gamma}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma_{eval}}^{\pi^*_{\widehat{F},\gamma}}(s) \right) \tag{9.14}$$

The first term is the deviation on value function when applying same policy on the same environment but with different discount factors. Since $\gamma \leq \gamma_{eval}$ we have $V_{F,\gamma}^{\pi^*_{F,\gamma_{eval}}}(s) \leq$

[2]Shifting and then rescaling the reward is equivalent to shifting and rescaling the Q and value function, and does not change the optimal policy. Moreover the mentioned transformation is $r \rightarrow (r - (\mathcal{R}_{min} - \lambda))/(\mathcal{R}_{\max} - \mathcal{R}_{\min} + \lambda)$, therefore, $\lambda' = (-\mathcal{R}_{min})/(\mathcal{R}_{\max} - \mathcal{R}_{\min} + \lambda)$

$V_{F,\gamma_{eval}}^{\pi^*_{F,\gamma_{eval}}}(s)$.

$$V_{F,\gamma_{eval}}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma}^{\pi^*_{F,\gamma_{eval}}}(s) = \mathbb{E}_F\left[\sum_{t=0}^{\infty}\gamma_{eval}^t r_t | s_0 = s, \pi^*_{F,\gamma_{eval}}\right] - \mathbb{E}_F\left[\sum_{t=0}^{\infty}\gamma^t r_t | s_0 = s, \pi^*_{F,\gamma_{eval}}\right] \tag{9.15}$$

$$= \mathbb{E}_F\left[\sum_{t=0}^{\infty}(\gamma_{eval}^t - \gamma^t) r_t | s_0 = s, \pi^*_{F,\gamma_{eval}}\right] \leq \left(\frac{1}{1-\gamma_{eval}} - \frac{1}{1-\gamma}\right) \tag{9.16}$$

The second part of Eq. 9.14 is the deviation in value function under different policies and different classifiers. Again, since $\gamma \leq \gamma_{eval}$, we have $V_{F,\gamma_{eval}}^{\pi^*_{\widehat{F},\gamma}}(s) \geq V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}(s)$

$$V_{F,\gamma}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma_{eval}}^{\pi^*_{\widehat{F},\gamma}}(s) \leq V_{F,\gamma}^{\pi^*_{F,\gamma_{eval}}}(s) - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}(s) \leq V_{F,\gamma}^{\pi^*_{F,\gamma}}(s) - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}(s) \tag{9.17}$$

where the last inequality is due to the optimality of $\pi^*_{F,\gamma}$ on the environment of F, γ. To bound this part we exploit the proof trick used in [JKSL15].

$$V_{F,\gamma}^{\pi^*_{F,\gamma}}(s) - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}(s) = \left(V_{F,\gamma}^{\pi^*_{F,\gamma}}(s) - V_{\widehat{F},\gamma}^{\pi^*_{F,\gamma}}(s)\right) + \left(V_{\widehat{F},\gamma}^{\pi^*_{F,\gamma}}(s) - V_{\widehat{F},\gamma}^{\pi^*_{\widehat{F},\gamma}}(s)\right) + \left(V_{\widehat{F},\gamma}^{\pi^*_{\widehat{F},\gamma}}(s) - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}(s)\right) \tag{9.18}$$

since the middle term is negative we have

$$\begin{aligned} V_{F,\gamma}^{\pi^*_{F,\gamma}}(s) - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}(s) &\leq \left(V_{F,\gamma}^{\pi^*_{F,\gamma}}(s) - V_{\widehat{F},\gamma}^{\pi^*_{F,\gamma}}(s)\right) + \left(V_{\widehat{F},\gamma}^{\pi^*_{\widehat{F},\gamma}}(s) - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}(s)\right) \\ &\leq 2 \max_{\{\pi^*_{\widehat{F},\gamma}, \pi^*_{\widehat{F},\gamma}\}} \left|V_{\widehat{F},\gamma}^{\pi}(s) - V_{F,\gamma}^{\pi}(s)\right| \end{aligned} \tag{9.19}$$

This quantity $V_{\widehat{F},\gamma}^{\pi}(s) - V_{F,\gamma}^{\pi}(s)$ is the difference between the performance of the same policy on two different environments. These two value functions should satisfy the following bellman equations:

$$V^{\pi}_{F,\gamma}(s) = \mathcal{R}(s,\pi(s)) + \lambda' F(s) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,\pi(s)) V^{\pi}_{F,\gamma}(s')$$
$$V^{\pi}_{\widehat{F},\gamma}(s) = \mathcal{R}(s,\pi(s)) + \lambda' \widehat{F}(s) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,\pi(s)) V^{\pi}_{\widehat{F},\gamma}(s')$$

To compute the solution two this equation, we use dynamic programing. Let initialize $V_0, \widehat{V} = V$(an arbitrary value) and construct the following updates.

$$\textit{for } i \in \{1, \dots \infty\}$$
$$V^{\pi}_{i}(s) = \mathcal{R}(s,\pi(s)) + \lambda' F(s) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,\pi(s)) V^{\pi}_{i-1}(s)$$
$$\widehat{V}^{\pi}_{i}(s) = \mathcal{R}(s,\pi(s)) + \lambda' \widehat{F}(s) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,\pi(s)) \widehat{V}^{\pi}_{i-1}(s')$$

As i tends to infinity, these two dynamics updates converge to $V^{\pi}_{F,\gamma}(s)$, and $V^{\pi}_{\widehat{F},\gamma}(s)$ respectively. To bound the right hand side of Eq. 9.19 we have

$$\begin{aligned} V^{\pi}_{i}(s) - \widehat{V}^{\pi}_{i}(s) &= \lambda' F(s) - \lambda' \widehat{F}(s) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s'|s,\pi(s)) \left(V_{i-1}(s') - \widehat{V}_{i-1}(s')\right) \\ &\leq \lambda' \sum_{i'=0}^{i} \gamma^{i'} \max_{s} \left|F(s) - \widehat{F}(s)\right| \end{aligned} \tag{9.20}$$

As i tends to infinity, we have

$$\begin{aligned} \left|V^{\pi^*_{F,\gamma}}_{F,\gamma}(s) - V^{\pi^*_{\widehat{F},\gamma}}_{F,\gamma}(s)\right| &= \max_{\pi} \left|\lim_{i\to\infty} \left(V^{\pi}_{i}(s) - \widehat{V}^{\pi}_{i}(s)\right)\right| \\ &\leq \lambda' \sum_{i'=0}^{i} \gamma^{i'} \max_{s,\pi} \left|F(s) - \widehat{F}(s)\right| \leq \lambda' \max_{s,\pi} \frac{\left|F(s) - \widehat{F}(s)\right|}{(1-\gamma)} \end{aligned}$$

9.6.1 Lookup Table Classifier

If we consider the fear model as a lookup table, and deterministic, then observing each state once is enough to exactly recover the classifier.

For the stochastic F, at time step N

$$\left|F(s)-\widehat{F}(s)\right| \leq \sqrt{\frac{\log\frac{N}{\delta}}{N(s)}} \tag{9.21}$$

with probability δ where $N(s)$ is the number visits to a state s at time step N. The trajectory produced by algorithm does not produce $i.i.d.$ samples of state. Therefore, for Eq. 9.21 we use Hoeffding's inequality accompanied with union bound over time N. In order to have this bound to hold for all the states at once, we need another union bounds over states and all possibly optimal policies Π_γ under noisy classifier , which requires to replace $\delta \rightarrow \delta/SA\Pi_\gamma$. Let's assume a minimum number of visit $\overline{N}$ to each state,

$$\|V_{F,\gamma}^{\pi^*_{F,\gamma}} - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}}\|_\infty \leq \frac{\lambda'}{1-\gamma}\sqrt{\frac{\log\frac{NSA\Pi_\gamma}{\delta}}{\overline{N}}} \tag{9.22}$$

Finally, adding Eq. 9.15 and Eq. 9.22, the upper bound on $\mathbb{L}$ is as follows:

$$\mathcal{L} \leq \lambda'\frac{1-\gamma_{eval}}{1-\gamma}\sqrt{\frac{\log\frac{NSA\Pi_\gamma}{1-\delta}}{\overline{N}}} + \frac{\gamma_{eval}-\gamma}{1-\gamma}$$

9.6.2 Classifier from set of functions

Let $\mathcal{F}$ denote a set of given binary classifiers and $F \in \mathcal{F}$. In this case, let's assume that we are given a set of N $i.i.d$ samples from the stationary distribution $\omega_F^{\pi^*_{\widehat{F},\gamma}}$. Given a policy π, the MDP transition process reduces to a Markov chain with transition probability $\mathcal{T}^\pi$. Now we rewrite the Eq. 9.20 in a matrix format where $V_i^\pi, F \in \mathbb{R}^S$ are vectors of concatenation of $V_i^\pi(s)$ and $F(s)$, $\forall s \in \mathcal{S}$ respectively.

$$V_i^\pi - \widehat{V}_i^\pi = \lambda' F - \lambda' \widehat{F} + \gamma \mathcal{T}^\pi \left(V_{i-1}^\pi - \widehat{V}_{i-1}^\pi \right) \leq \lambda' \sum_{i'=0}^{i} \left(\gamma \mathcal{T}^\pi \right)^{i'} \left(F - \widehat{F} \right) \tag{9.23}$$

as i goes to infinity we have

$$V_{F,\gamma}^{\pi^*_{F,\gamma}} - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}} \leq \lambda' \left(\mathbb{1} - \gamma \mathcal{T}^\pi \right)^{-1} \left(F - \widehat{F} \right)$$

Using PAC analysis of binary classification in [Han16] a follow up to [Vap13], we have

$$\left| F - \widehat{F} \right|^\top \omega_F^{\pi^*_{\widehat{F},\gamma}} \leq 3200 \frac{\mathcal{VC}(\mathcal{F}) + \log \frac{1}{\delta}}{N}$$

with probability at least $1 - \delta$ where $\mathcal{VC}(\mathcal{F})$ is the $\mathcal{VC}$ dimension of the hypothesis class and $|\cdot|$ is entry-wise absolute value. Since $\gamma < 1$, then $\alpha_{\max}$, the maximum eigenvalue of $\left(\mathbb{1} - \gamma \mathcal{T}^\pi \right)^{-1}$ is bounded above and we have

$$\| V_{F,\gamma}^{\pi^*_{F,\gamma}} - V_{F,\gamma}^{\pi^*_{\widehat{F},\gamma}} \|_1 \leq 3200 \lambda' \alpha_{\max} \frac{\mathcal{VC}(\mathcal{F}) + \log \frac{1}{\delta}}{N}$$

and therefore,

$$\mathcal{L} \leq 3200 \lambda' \alpha_{\max} (1 - \gamma_{eval}) \frac{\mathcal{VC}(\mathcal{F}) + \log \frac{1}{\delta}}{N} + \frac{\gamma_{eval} - \gamma}{1 - \gamma}$$

The remaining part is to solve

$$\gamma^* = \operatorname{argmin}_{\gamma \leq \gamma_{eval}} \mathcal{L}$$

to find the optimal γ.

The same analysis, up to a slight modification[3], holds for the continuous state and action spaces.

9.7 Related work

The chapter addresses safety in RL, intrinsically motivated RL, and the stability of Q-learning with function approximation under distributional shift. Our work also has some connection to reward shaping. We attempt to highlight the most relevant papers here. Several papers address safety in RL. [GF15] provide a thorough review on the topic, identifying two main classes of methods: those that perturb the objective function and those that use external knowledge to improve the safety of exploration.

While a typical reinforcement learner optimizes expected return, some papers suggest that a safely acting agent should also minimize risk. [HSSU08] defines a *fatality* as any return below some threshold τ. They propose a solution comprised of a *safety function*, which identifies unsafe states, and a *backup model*, which navigates away from those states. Their work, which only addresses the tabular setting, suggests that an agent should minimize the probability of fatality instead of maximizing the expected return. [Heg94] suggests an alternative Q-learning objective concerned with the minimum (vs expected) return.Other papers suggest modifying the objective to penalize policies with high-variance returns [GF15]. Maximizing expected returns while minimizing their variance is a classic problem in finance, where a common objective is the ratio of expected return to its standard deviation [Sha66]. [MA12] gives a definition of safety based on ergodicity. They consider a fatality to be a state from which one cannot return

[3]Instead of having V as a vector of state values indexed by states, it is a continuous function of states. Furthermore, the transition kernel is over continuous distribution therefore the same bellman update in Eq. 9.23 holds.

to the start state. [SSSS16] theoretically analyzes how strong a penalty should be to discourage accidents. They also consider hard constraints to ensure safety. None of the above works address the case where distributional shift dooms an agent to perpetually revisit known catastrophic failure modes. Other papers incorporate external knowledge into the exploration process. Typically, this requires access to an oracle or extensive prior knowledge of the environment. In the extreme case, some papers suggest confining the policy search to the subset of policies known to be *safe*. For reasonably complex environments or classes of policies this seems infeasible.

The potential oscillatory or divergent behavior of Q-learners with function approximation has been previously identified [BM95, B$^+$95, Gor96]. Outside of RL, the problem of covariate shift has been extensively studied [SK12]. [MO05] addresses the problem of catastrophic forgetting owing to distributional shift in RL with function approximation, proposing a memory-based solution. Many papers address intrinsic rewards, which are internally assigned, vs the standard (extrinsic) reward. Typically, intrinsic rewards are used to encourage exploration [Sch91, BSO$^+$16] and to acquire a modular set of skills [CBS04]. Some papers refer to the intrinsic reward for discovery as *curiosity*. Like classic work on intrinsic motivation, our methods perturb the reward function. But instead of assigning bonuses to encourage discovery of novel transitions, we assign penalties to discourage catastrophic transitions.

Key differences In this chapter, we undertake a novel treatment of safe reinforcement learning, While the literature offers several notions of safety in reinforcement learning, we see the following problem: Existing safety research that perturbs the reward function requires little foreknowledge, but fundamentally changes the objective globally. On the other hand, processes relying on expert knowledge may presume an unreasonable level of foreknowledge. Moreover, little of the prior work on safe reinforcement learning, to our knowledge, specifically addresses the problem of catastrophic forgetting. This chapter proposes a new class of algorithms for avoiding catastrophic states and a theoretical analysis supporting its robustness.

9.8 Conclusions

Our experiments demonstrate that DQNs are susceptible to periodically repeating mistakes, however bad, raising questions about their real-world utility when harm can come of actions. While it's easy to visualize these problems on toy examples, similar dynamics are embedded in more complex domains. Consider a domestic robot acting as a barber. The robot might receive positive feedback for giving a closer shave. This reward encourages closer contact at a steeper angle. Of course, the shape of this reward function belies the catastrophe lurking just past the optimal shave. Similar dynamics might be imagines in a vehicle that is rewarded for traveling faster but could risk an accident with excessive speed. Our results with the intrinsic fear model suggest that with only a small amount of prior knowledge (the ability to recognize catastrophe states after the fact), we can simultaneously accelerate learning and avoid catastrophic states. This work represents a first step towards combating some issues relating to safety in RL stemming from catastrophic forgetting.

9.9 Acknowledgments

Chapter 9, *Combating Deep Reinforcement Learning's Sisyphean Curse with Intrinsic Fear* was written in collaboration with Kamyar Azizzadenesheli, Abhishek Kumar, Lihong Li, Jianfeng Gao, and Li Deng. The dissertation author was the primary investigator and author of this paper.

Chapter 10

The Mythos of Model Interpretability

Supervised machine learning models boast remarkable predictive capabilities. But can you trust your model? Will it work in deployment? What else can it tell you about the world? We want models to be not only good, but interpretable. And yet the task of *interpretation* appears underspecified. Papers provide diverse and sometimes non-overlapping motivations for interpretability, and offer myriad notions of what attributes render models interpretable. Despite this ambiguity, many papers proclaim interpretability axiomatically, absent further explanation. In this chapter, we seek to refine the discourse on interpretability. First, we examine the desiderata sought in papers addressing interpretability, finding them to be diverse and occasionally discordant. Then, we explore model properties and techniques thought to confer interpretability, identifying transparency to humans and post-hoc explanations as competing concepts. Throughout, we discuss the feasibility and desirability of different notions of interpretability, and question the oft-made assertions that linear models are interpretable and that deep neural networks are not.

10.1 Introduction

Until recently, humans had a monopoly on agency in society. If you applied for a job, a loan, or bail, a human decided your fate. And if you went to the hospital, a human would attempt to classify what was wrong with you and might recommend whether to undergo surgery. For consequential decisions like these, you might demand an explanation from the decision-making agent.

If your loan application was denied, you might want to understand the agent's reasoning in a bid to strengthen your next application. Or if the decision was based on a flawed premise, you might contest this premise in the hope of overturning the decision. From a doctor, an explanation might serve to educate you about your condition. In societal contexts, the *reasons* for a decision often matter. For example, intentionally causing death (murder) vs unintentionally (manslaughter) are distinct crimes. Whether a hiring decision is based (directly or indirectly) on a protected characteristic like race has bearing on its legality.

Over the past 20 years, rapid progress in machine learning (ML) has enabled the deployment of automatic decision processes. Most ML-based decision-making in practical use works in the following way: the ML algorithm is *trained* to take some input and predict the corresponding output. Given a set of attributes characterizing a financial transaction, predict the long-term return on investment. Given images from a CT scan, assign a probability that it depicts a cancerous tumor. The ML algorithm takes in a large corpus of (input, output) pairs, and outputs a *model* which can predict the output corresponding to a previously unseen input. To fully automate decisions, one feeds the model's output into some decision rule. For example, a spam-filter programatically discards emails predicted to be spam with confidence exceeding some threshold.

As ML penetrates critical areas like medicine, the criminal justice system, and financial markets, the inability of humans to understand these models seems problematic. Some suggest *model interpretability* as a remedy, but the academic literature, few authors articulate precisely *what* interpretability means or *why* the offered solution is useful.

Despite the lack of a definition, a growing body of literature proposes purportedly interpretable algorithms. From this, we might conclude that either: (i) the definition of interpretability is universally agreed upon, but no one has bothered to set it in writing, or (ii) the term interpretability is ill-defined, and thus claims regarding interpretability of various models exhibit a quasi-scientific character.

An investigation of the literature suggests the latter. Both the desiderata and methods suggested in papers investigating interpretability are diverse, suggesting that interpretability is not a monolithic concept, but several distinct ideas that ought to be disentangled before we can make progress. We hope, through this critical analysis, to bring focus to the dialogue.

In this chapter, we mainly consider supervised learning and not other machine learning paradigms, such as reinforcement learning and interactive learning. This scope derives from (1) the primacy of supervised learning in the real world, and (2) our interest in the common claim that linear models interpretable while deep neural networks are not [LCG12]. To gain conceptual clarity, we ask the refining questions: *what is interpretability?* and *why is it important?*

To ground our discussion, we address the second question first (expanded in §10.2). Many papers propose interpretability as a means to engender trust [Kim15, RMRO98]. But this leaves us with a similarly vexing epistemological question: *what is trust?* Does it refer to faith that a model will perform well? Or does interpretability simply mean a low-level mechanistic understanding of our models? Is trust defined subjectively?

Other papers suggest that an interpretable model is desirable because it might help to uncover causal structure in observational data [AI15]. The legal notion of a *right to explanation* offers yet another lens on interpretability. Another goal of interpretability might simply be to get more useful information from the model.

While the discussed desiderata of interpretability are diverse, they typically speak to situations where an ML problem formulation is imperfectly matched to the complex real-life task it is meant to solve. Consider medical research with longitudinal data. Our real goal may be to discover potentially causal associations, as with smoking and cancer [WFF$^+$99]. But the optimization objective for most supervised learning models is simply to minimize error, a feat that might be achieved in a purely correlative fashion.

Another example of such a mismatch is that available training data imperfectly representats the likely deployment environment. For example, real environments often have changing dynamics. Imagine training a product recommender for an online store, where new products are periodically introduced and customer preferences can change over time. In more extreme cases, actions from an ML-based system may alter the environment, invalidating future predictions.

After addressing the desiderata of interpretability, we consider what properties of models might render models interpretable (expanded in §10.3). Some papers equate interpretability with *understandability* or *intelligibility* [LCGH13], i.e., that we can grasp *how the models work*. In these papers, understandable models are sometimes called *transparent*, while incomprehensible models are called *black boxes*. But what constitutes transparency? We might look to the algorithm itself. Will it converge? Does it produce a unique solution? Or we might look to its parameters: do we understand what each represents? Alternatively, we could consider the model's complexity. Is it simple enough to be examined all at once by a human?

Other papers investigate so-called post-hoc interpretations. These interpretations might *explain* predictions without elucidating the mechanisms by which models work.

Examples of post-hoc interpretations include the verbal explanations produced by people or the saliency maps used to analyze deep neural networks. Thus, human decisions might admit post-hoc interpretability despite the *black box* nature of human brains, revealing a contradiction between two popular notions of interpretability.

10.2 Desiderata of Interpretability Research

In this section we spell out the various desiderata of interpretability research through the lens of the literature. The demand for interpretability arises when there is a mismatch between the formal objectives of supervised learning (test set predictive performance) and the real world costs in a deployment setting.

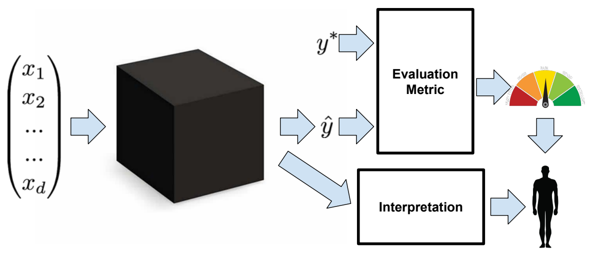


Figure 10.1: Typically, evaluation metrics require only predictions and *ground truth* labels. When stakeholders additionally demand *interpretability*, we might infer the existence of desiderata that cannot be captured in this fashion.

Consider that most common evaluation metrics for supervised learning require only predictions, together with ground truth, to produce a score. So the very desire for an *interpretation* suggests that sometimes, predictions alone and metrics calculated on them do not suffice to characterize the model (Figure 10.1). We should then ask, what are these other desiderata and under what circumstances are they sought?

Often, real-world objectives are difficult to encode as simple mathematical functions. Otherwise, we might just incorporate them into the objective function and consider the problem solved. For example, an algorithm for making hiring decisions should simultaneously optimize productivity, ethics, and legality. But how would you go about writing a function that measure ethics or legality? The problem can also arise when we desire robustness to changes between the dynamics of the training and deployment environments.

Trust Some papers motivate interpretability by suggesting it to be prerequisite for *trust* [Kim15, RSG16]. But what is trust? Is it simply confidence that a model will perform well? If so, a sufficiently accurate model should be demonstrably trustworthy and interpretability would serve no purpose. Trust might also be defined subjectively. For example, a person might feel more at ease with a well-understood model, even if this understanding served no obvious purpose. Alternatively, when the training and deployment objectives diverge, trust might denote confidence that the model will perform well with respect to the real objectives and scenarios.

For example, consider the growing use of machine learning models to forecast crime rates for purposes of allocating police officers. We may trust the model to make accurate predictions but not to account for racial biases in the training data for the model's own effect in perpetuating a cycle of incarceration by over-policing some neighborhoods. Another sense in which we might trust a machine learning model might be that we feel comfortable relinquishing control to it. In this sense, we might care not only about *how often a model is right* but also *for which examples it is right*. If the model tends to make mistakes in regions of input space where humans also make mistakes, and is typically accurate when humans are accurate, then it may be considered trustworthy in the sense that there is no expected cost of relinquishing control. But if a model tends to make mistakes for inputs that humans classify accurately, then there may always be an advantage to maintaining human supervision of the algorithms.

Causality Although supervised learning models are only optimized directly to make associations, researchers often use them in the hope of inferring properties of the natural world. For example, a simple regression model might reveal a strong association between thalidomide use and birth defects or smoking and lung cancer [WFF$^+$99].

The associations learned by supervised learning algorithms are not guaranteed to reflect causal relationships. There could always exist unobserved causes responsible for both associated variables. One might hope, however, that by interpreting supervised learning models, we could generate hypotheses that scientists could then test experimentally. [LRS05], for example, emphasizes regression trees and Bayesian neural networks, suggesting that these models are interpretable and thus better able to provide clues about the causal relationships between physiologic signals and affective states. The task of inferring causal relationships from observational data has been extensively studied [Pea09]. But causal inference methods tends to rely on strong assumptions and are not widely used by practitioners, especially on large, complex datasets.

10.2.1 Transferability

Typically we choose training and test data by randomly partitioning examples from the same distribution. We then judge a model's generalization error by the gap between its performance on training and test data. However, humans exhibit a far richer capacity to generalize, transferring learned skills to unfamiliar situations. We already use machine learning algorithms in situations where such abilities are required, such as when the environment is non-stationary. We also deploy models in settings where their use might alter the environment, invalidating their future predictions. Along these lines, [CLG$^+$15] describe a model trained to predict probability of death from pneumonia that assigned less risk to patients if they also had asthma. In fact, asthma was predictive of lower risk of death. This owed to the more aggressive treatment these patients received.

But if the model were deployed to aid in triage, these patients would then receive less aggressive treatment, invalidating the model.

Even worse, we could imagine situations, like machine learning for security, where the environment might be actively adversarial. Consider the recently discovered susceptibility of convolutional neural networks (CNNs) to adversarial examples. The CNNs were made to misclassify images that were imperceptibly (to a human) perturbed [SZS+13]. Of course, this isn't overfitting in the classical sense. The results achieved on training data generalize well to i.i.d. test data. But these are mistakes a human wouldn't make and we would prefer models not to make these mistakes either.

Already, supervised learning models are regularly subject to such adversarial manipulation. Consider the models used to generate credit ratings, scores that when higher should signify a higher probability that an individual repays a loan. According to their own technical report, FICO trains credit models using logistic regression [Fai11], specifically citing interpretability as a motivation for the choice of model. Features include dummy variables representing binned values for average age of accounts, debt ratio, and the number of late payments, and the number of accounts in good standing.

Several of these factors can be manipulated at will by credit-seekers. For example, one's debt ratio can be improved simply by requesting periodic increases to credit lines while keeping spending patterns constant. Similarly, the total number of accounts can be increased by simply applying for new accounts, when the probability of acceptance is reasonably high. Indeed, FICO and Experian both acknowledge that credit ratings can be manipulated, even suggesting guides for improving one's credit rating. These rating improvement strategies may fundamentally change one's underlying ability to pay a debt. The fact that individuals actively and successfully game the rating system may invalidate its predictive power.

10.2.2 Informativeness

Sometimes we apply decision theory to the outputs of supervised models to take actions in the real world. However, in another common use paradigm, the supervised model is used instead to provide information to human decision makers, a setting considered by [KGJS15, HDM$^+$11]. While the machine learning objective might be to reduce error, the real-world purpose is to provide useful information. The most obvious way that a model conveys information is via its outputs. However, it may be possible via some procedure to convey additional information to the human decision-maker.

An interpretation may prove informative even without shedding light on a model's inner workings. For example, a diagnosis model might provide intuition to a human decision-maker by pointing to similar cases in support of a diagnostic decision. In some cases, we train a supervised learning model, but our real task more closely resembles unsupervised learning. Here, our real goal is to explore the data and the objective serves only as *weak supervision*.

Fair and Ethical Decision-Making At present, politicians, journalists and researchers have expressed concern that we must produce *interpretations* for the purpose of assessing whether decisions produced automatically by algorithms conform to ethical standards [GF16]. Recidivism predictions are already used to determine who to release and who to detain, raising ethical concerns. How can we be sure that predictions do not discriminate on the basis of race? Conventional evaluation metrics such as accuracy or AUC offer little assurance that ML-based decisions will behave acceptably. Thus demands for fairness often lead to demands for *interpretable* models.

10.3 Properties of Interpretable Models

We turn now to consider the techniques and model properties that are proposed to confer *interpretability*. These broadly fall into two categories. The first relate to *transparency*, i.e., *how does the model work?* The second consists of *post-hoc explanations*, i.e., *what else can the model tell me?*

10.3.1 Transparency

Informally, *transparency* is the opposite of *opacity* or *blackbox-ness*. It connotes some sense of understanding the mechanism by which the model works. We consider transparency at the level of the entire model (*simulatability*), at the level of individual components (e.g. parameters) (*decomposability*), and at the level of the training algorithm (*algorithmic transparency*).

Simulatability

In the strictest sense, we might call a model transparent if a person can contemplate the entire model at once. This definition suggests that an interpretable model is a simple model. We might think, for example that for a model to be fully understood, a human should be able to take the input data together with the parameters of the model and in *reasonable* time step through every calculation required to produce a prediction. This accords with the common claim that sparse linear models, as produced by lasso regression [Tib96], are more interpretable than dense linear models learned on the same inputs. [RSG16] also adopt this notion of interpretability, suggesting that an interpretable model is one that "can be readily presented to the user with visual or textual artifacts."

For some models, such as decision trees, the size of the model (total number of nodes) may grow much faster than the time to perform inference (length of pass from root to leaf). This suggests that simulatability may admit two subtypes, one based on the total size of the model and another based on the computation required to perform inference.

Fixing a notion of simulatability, the quantity denoted by *reasonable* is subjective. But clearly, given the limited capacity of human cognition, this ambiguity might only span several orders of magnitude. In this light, we suggest that neither linear models, rule-based systems, nor decision trees are intrinsically interpretable. Sufficiently high-dimensional models, unwieldy rule lists, and deep decision trees could all be considered less transparent than comparatively compact neural networks.

Decomposability

A second notion of transparency might be that each part of the model - each input, parameter, and calculation - admits an intuitive explanation. This accords with the property of *intelligibility* as described by [LCG12]. For example, each node in a decision tree might correspond to a plain text description (e.g. *all patients with diastolic blood pressure over 150*). Similarly, the parameters of a linear model could be described as representing strengths of association between each feature and the label.

Note that this notion of interpretability requires that inputs themselves be individually interpretable, disqualifying some models with highly engineered or anonymous features. While this notion is popular, we shouldn't accept it blindly. The weights of a linear model might seem intuitive, but they can be fragile with respect to feature selection and pre-processing. For example, associations between flu risk and vaccination might be positive or negative depending on whether the feature set includes indicators of old age, infancy, or immunodeficiency.

Algorithmic Transparency

A final notion of transparency might apply at the level of the learning algorithm itself. For example, in the case of linear models, we understand the shape of the error surface. We can prove that training will converge to a unique solution, even for previously unseen datasets. This may give some confidence that the model might behave in an online setting requiring programmatic retraining on previously unseen data. On the other hand, modern deep learning methods lack this sort of algorithmic transparency. While the heuristic optimization procedures for neural networks are demonstrably powerful, we don't understand how they work, and at present cannot guarantee a priori that they will work on new problems. Note, however, that humans exhibit none of these forms of transparency.

10.3.2 Post-hoc Interpretability

Post-hoc interpretability presents a distinct approach to extracting information from learned models. While post-hoc interpretations often do not elucidate precisely how a model works, they may nonetheless confer useful information for practitioners and end users of machine learning. Some common approaches to post-hoc interpretations include natural language explanations, visualizations of learned representations or models, and explanations by example (e.g. *this tumor is classified as malignant because to the model it looks a lot like these other tumors*).

To the extent that we might consider humans to be interpretable, it is this sort of interpretability that applies. For all we know, the processes by which we humans make decisions and those by which we explain them may be distinct. One advantage of this concept of interpretability is that we can interpret opaque models after-the-fact, without sacrificing predictive performance.

Text Explanations

Humans often justify decisions verbally. Similarly, we might train one model to generate predictions and a separate model, such as a recurrent neural network language model, to generate an explanation. Such an approach is taken in a line of work by [KHF$^+$16]. They propose a system in which one model (a reinforcement learner) chooses actions to optimize cumulative discounted return. They train another model to map a model's state representation onto verbal explanations of strategy. These explanations are trained to maximize the likelihood of previously observed ground truth explanations from human players, and may not faithfully describe the agent's decisions, however plausible they appear. We note a connection between this approach and recent work on neural image captioning in which the representations learned by a discriminative convolutional neural network (trained for image classification) are co-opted by a second model to generate captions. These captions might be regarded as interpretations that accompany classifications.

In work on recommender systems, [ML13] use text to explain the decisions of a latent factor model. Their method consists of simultaneously training a latent factor model for rating prediction and a topic model for product reviews. During training they alternate between decreasing the squared error on rating prediction and increasing the likelihood of review text. The models are connected because they use normalized latent factors as topic distributions. In other words, latent factors are regularized such that they are also good at explaining the topic distributions in review text. The authors then explain user-item compatibility by examining the top words in the topics corresponding to matching components of their latent factors. Note that the practice of interpreting topic models by presenting the top words is itself a post-hoc interpretation technique that has invited scrutiny [CGW$^+$09].

Visualization Another common approach to generating post-hoc interpretations is to render visualizations in the hope of determining qualitatively what a model has learned. One popular approach is to visualize high-dimensional distributed representations with t-SNE [VdMH08], a technique that renders 2D visualizations in which nearby data points are likely to appear close together.

[MOT15] attempt to explain what an image classification network has learned by altering the input through gradient descent to enhance the activations of certain nodes selected from the hidden layers. An inspection of the perturbed inputs can give clues to what the model has learned. Likely because the model was trained on a large corpus of animal images, they observed that enhancing some nodes caused the dog faces to appear throughout the input image.

In the computer vision community, similar approaches have been explored to investigate what information is retained at various layers of a neural network. [MV15] pass an image through a discriminative convolutional neural network to generate a representation. They then demonstrate that the original image can be recovered with high fidelity even from reasonably high-level representations (level 6 of an AlexNet) by performing gradient descent on randomly initialized pixels.

Local Explanations

While it may be difficult to succinctly describe the full mapping learned by a neural network, some papers focus instead on explaining what a neural network depends on locally. One popular approach for deep neural nets is to compute a saliency map. Typically, they take the gradient of the output corresponding to the correct class with respect to a given input vector. For images, this gradient can be applied as a mask (Figure 10.2), highlighting regions of the input that, if changed, would most influence the output [SVZ13, WdFL16].

Note that these explanations of what a model is *focusing on* may be misleading. The saliency map is a local explanation only. Once you move a single pixel, you may get a very different saliency map. This contrasts with linear models, which model global relationships between inputs and outputs.

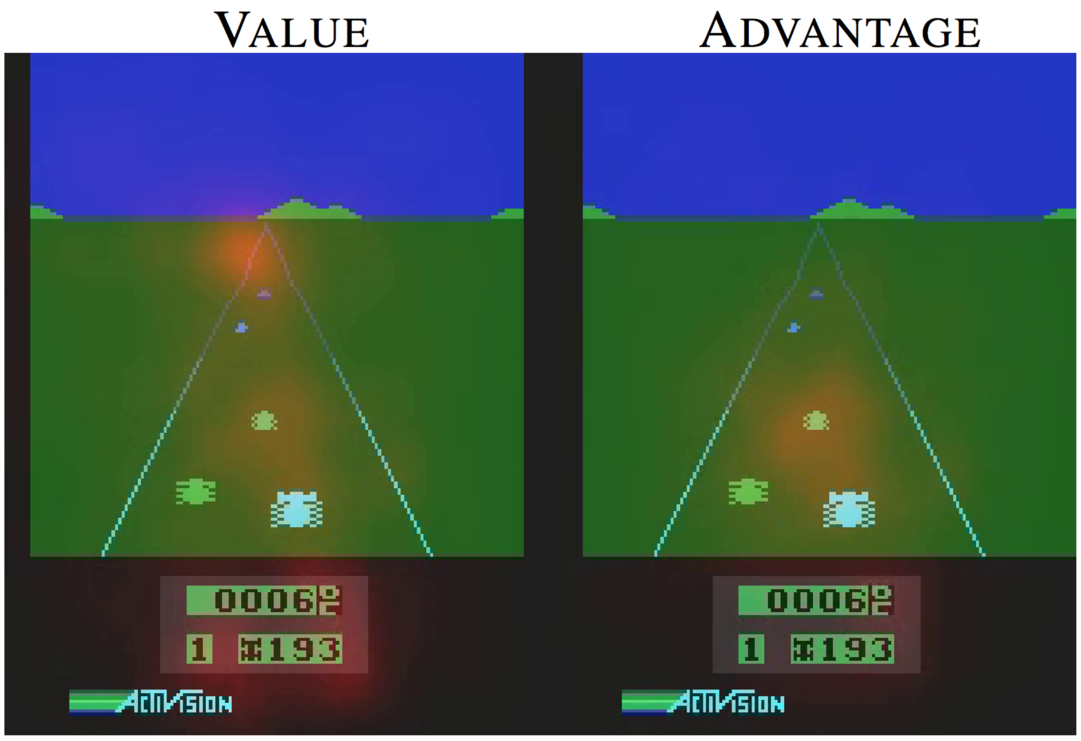


Figure 10.2: Saliency map by [WdFL16] to convey intuition over what the value function and advantage function portions of their deep Q-network are *focusing* on.

Another attempt at local explanations is made by [RSG16]. In this work, the authors explain the decisions of any model in a local region near a particular point, by learning a separate sparse linear model to explain the decisions of the first.

Explanation by Example

One post-hoc mechanism for explaining the decisions of a model might be to report (in addition to predictions) which other examples the model considers to be most similar, a method suggested by [CKD+99]. After training a deep neural network or latent variable model for a discriminative task, we then have access not only to predictions but also to the learned representations. Then, for any example, in addition to generating a prediction, we can use the activations of the hidden layers to identify the k-nearest neighbors based on the proximity in the space learned by the model. This sort of explanation by example has precedent in how humans sometimes justify actions by analogy. For example, doctors often refer to case studies to support a planned treatment protocol.

In the neural network literature, [MSC+13] use such an approach to examine the learned representations of words after word2vec training. While their model is trained for discriminative skip-gram prediction, to examine what relationships the model has learned, they enumerate nearest neighbors of words based on distances calculated in the latent space. We also point to related work in Bayesian methods: [KRS14] and [DVWA15] investigate cased-base reasoning approaches for interpreting generative models.

10.4 Discussion

The concept of interpretability appears simultaneously important and slippery. Earlier, we analyzed both the desiderata for which interpretability is purportedly an answer and some attempts by the research community to produce models that confer it. In this discussion, we consider the implications of our analysis and offer several takeaways to the reader.

10.4.1 Linear models are not strictly more interpretable than deep neural networks

Despite this claim's enduring popularity, its truth content varies depending on what notion of interpretability we employ. With respect to *algorithmic transparency*, this claim seems uncontroversial, but given high dimensional or heavily engineered features, linear models lose *simulatability* or *decomposability*, respectively.

When choosing between linear and deep models, we must often make a trade-off between *algorithic transparency* and *decomposability*. This is because deep neural networks tend to operate on raw or lightly processed features. So if nothing else, the features are intuitively meaningful, and post-hoc reasoning is sensible. However, in order to get comparable performance, linear models often must operate on heavily hand-engineered features. [LKW16b] demonstrates such a case where linear models can only approach the performance of RNNs at the cost of decomposability.

For some kinds of post-hoc interpretation, deep neural networks exhibit a clear advantage. They learn rich representations that can be visualized, verbalized, or used for clustering. Considering the desiderata for interpretability, linear models appear to have a better track record for studying the natural world but we do not know of a theoretical reason why this must be so. Conceivably, post-hoc interpretations could prove useful in similar scenarios.

10.4.2 Claims about interpretability must be qualified

As demonstrated in this chapter, the term does not reference a monolithic concept. To be meaningful, any assertion regarding interpretability should fix a specific definition. If the model satisfies a form of transparency, this can be shown directly. For post-hoc interpretability, papers ought to fix a clear objective and demonstrate evidence that the offered form of interpretation achieves it.

10.4.3 In some cases, transparency may be at odds with the broader objectives of AI

Some arguments against *black-box* algorithms appear to preclude any model that could match or surpass our abilities on complex tasks. As a concrete example, the short-term goal of building trust with doctors by developing transparent models might clash with the longer-term goal of improving health care. We should be careful when giving up predictive power, that the desire for transparency is justified and isn't simply a concession to institutional biases against new methods.

10.4.4 Post-hoc interpretations can potentially mislead

We caution against blindly embracing post-hoc notions of interpretability, especially when optimized to placate subjective demands. In such cases, one might - deliberately or not - optimize an algorithm to present misleading but plausible explanations. As humans, we are known to engage in this behavior, as evidenced in hiring practices and college admissions. Several journalists and social scientists have demonstrated that acceptance decisions attributed to virtues like *leadership* or *originality* often disguise racial or gender discrimination [Mou14]. In the rush to gain acceptance for machine learning and to emulate human intelligence, we should be careful not to reproduce pathological behavior at scale.

10.4.5 Future Work

We see several promising directions for future work. First, for some problems, the discrepancy between real-life and machine learning objectives could be mitigated by developing richer loss functions and performance metrics. Exemplars of this direction include research on sparsity-inducing regularizers and cost-sensitive learning. Second, we can expand this analysis to other ML paradigms such as reinforcement learning. Reinforcement learners can address some (but not all) of the objectives of interpretability research by directly modeling interaction between models and environments. However, this capability may come at the cost of allowing models to experiment in the world, incurring real consequences. Notably, reinforcement learners are able to learn causal relationships between their actions and real world impacts. However, like supervised learning, reinforcement learning relies on a well-defined scalar objective. For problems like fairness, where we struggle to verbalize precise definitions of success, a shift of ML paradigm is unlikely to eliminate the problems we face.

10.5 Acknowledgments

Chapter 10, *The Mythos of Model Interpretability*, was written solely by the dissertation author. Thanks to Charles Elkan, Julian McAuley, Maggie Makar, David Kale, Been Kim, Lihong Li, Rich Caruana, Sepp Hochreiter, Daniel Fried, and Jack Berkowitz, for helpful conversations and critical feedback. The dissertation author was the primary investigator and author of this paper.

Chapter 11

Can We Reduce ML's Disparate Impact without Disparate Treatment?

Following related work in law and policy, two notions of prejudice have come to shape the study of fairness in algorithmic decision-making. Algorithms exhibit *disparate treatment* if they formally treat people differently according to a protected characteristic, like race, or if they *intentionally* discriminate (even if via proxy variables). Algorithms exhibit *disparate impact* if they affect subgroups differently. Disparate impact can arise unintentionally and absent disparate treatment. The natural way to reduce disparate impact would be to apply disparate treatment in favor of the disadvantaged group, i.e. to apply *affirmative action*. However, owing to the practice's contested legal status, several papers have proposed trying to eliminate both forms of unfairness simultaneously, introducing a family of algorithms that we denote *disparate learning processes* (DLPs). These processes incorporate the protected characteristic as an input to the learning algorithm (e.g. via a regularizer) but produce a model that cannot directly access the protected characteristic as an input. In this chapter, we make the following arguments: (i) DLPs can be functionally equivalent to disparate treatment, and thus should carry the same legal status; (ii) when the protected characteristic is redundantly encoded in the nonsensitive features, DLPs can exactly apply any disparate treatment protocol; (iii) when the characteristic is only partially encoded, DLPs may induce within-class discrimination. Finally, we argue the normative point that rather than masking efforts towards proportional representation, it is preferable to undertake them transparently.

11.1 Introduction

Effective decision-making relies on the ability of the decision maker to distinguish between options on the basis of available information. Selection processes, such as hiring and admissions, are typically driven by human assessments of applicants' qualifications. This much is unavoidable, unless we opt to make trivial decisions and either select everyone, no one, or perform selection entirely at random. Yet some kinds of selection criteria violate ethical and legal principles. In many domains, the law explicitly prohibits adverse decisions made on the basis of an applicant's irrelevant or protected characteristics. For example, in the United States, in Title VII of the Civil Rights Act of 1964 [Civ64], the law forbids employment decisions that discriminate on the basis of the following *protected characteristics*: *race, color, religion, sex,* and *national origin*. The interpretation of this law has led to two widely-referenced notions of unfairness: *disparate treatment* and *disparate impact*.

Disparate treatment refers to intentional discrimination. This can include: (i) making decisions explicitly on the basis of a protected characteristic or (ii) making intentionally prejudiced decisions against members of a protected class via proxy variables. For example, in the 1900s, literacy tests were used to determine voting eligibility in order to disenfranchise racial minorities. Even absent disparate treatment, a facially neutral decision-making policy might exhibit *disparate impact*; i.e., unequal outcomes, for people in different classes with respect to some protected characteristic(s). This again may occur due to correlations between protected and unprotected characteristics.

In some cases, observed disparities are evidence of unfair treatment. For example, black defendants are sentenced to death more frequently than white defendants for the same crimes [For14]. This might owe in part to the racial biases of judges and juries; it also might owe to the correlation between race and wealth, and by extension, access to legal services. But disparate impact can also stem from more benign sources. For example, the over-representation of Asian students in prestigious US colleges appears not to stem from pro-Asian discrimination; on the contrary, investigative reports suggest that the over-representation actually arises despite admissions policies that set a higher bar for Asian applicants [HS17].

Disparate treatment and disparate impact are concepts rooted in United States labor law. In some texts, the terms have a more technical meaning (as defined above), and in others the meaning is tied up in the legal doctrine associated with a specific set of decisions, such as hiring. Throughout this chapter, we will apply the technical definitions, following the convention in the existing Machine Learning (ML) literature that interprets these notions as widely applicable fairness criteria. Our discussion of disparate treatment and disparate impact in the context of algorithmic decision making may not at every stage adhere to prevailing legal doctrine. We also note that because current anti-discrimination laws were developed with human decision making in mind, there is considerable debate over their applicability to governing algorithmic decision-making systems [BS16, GW17, Kim17]. However, this topic exceeds the scope of the current work.

11.1.1 Algorithmic Decision Making

Automated decision-making systems, based on ML models, are now trusted to make decisions of legal consequence, such as extending lines of credit, matching employees with employers, etc. These systems are typically built atop supervised ML models. In practice, the ML models do not take any actions themselves: they simply estimate the conditional probability of a label given some features. As a concrete example, a label might be a binary indicator $\{0,1\}$ of whether a loan defaults, and the features might be attributes related to a loan applicant's financial history. Decisions are typically made by feeding the conditional probabilities into some decision rule.

In the simplest systems, the decision rule consists of a threshold applied to the prediction. For example, if $\hat{P}(\mathit{default}|\mathbf{x}) > .2 \Rightarrow \mathit{reject\ loan}$. In many cases, while model outputs are generated programatically, decisions are made by humans. For example recidivism scores – predicted probabilities that an individual will be rearrested after being released – are taken into account as one among many criteria by judges when making decisions about bail, parole and sentencing.

A major concern is that a decision-making system based on an ML model might exhibit behavior that is unlawful with respect to protected characteristics. If the model explicitly incorporates a protected characteristic as input, this amounts to disparate treatment [Civ64, BS16, ZVGRG17]. An automated ML-based system can also exhibit disparate impact, which could arise via several possible mechanisms: (i) The choice of the target may be arbitrarily chosen among several that are all loose surrogates for the real quantity of interest (e.g. credit-worthiness). Any one particular choice might then benefit or disadvantage a given group; (ii) The targets themselves might reflect patterns of historical prejudice. For example, if members of the dominant group are more likely to keep their jobs during a layoff, then they may be more likely to repay their loans; (iii) If a group is grossly under-represented in the dataset, that could potentially lead to a model that is not as accurate when evaluating members of that group. In some of the literature on fairness in algorithmic decision-making, use of the protected characteristic is called *direct discrimination*, while points (i-iii) are sometime described as *indirect discrimination* [PRT08, KAS11].

Note that the behavior of an ML model depends on the dataset used for training, and thus these mechanisms could induce disparate impact absent a data scientist's knowledge. As Barocas and Selbst note, "honest attempts to certify the absence of prejudice on the part of those involved in the data mining process may wrongly confer the imprimatur of impartiality on the resulting decisions" [BS16]. These issues have come to light starting with the ground-breaking paper [FN96], which recognized the ability of computer systems to disparately affect protected groups. Work specifically focused on discrimination owing to data-mining became more common following [PRT08]. Recently, several prominent cases of disparate impact garnered widespread media attention. For example, a report by ProPublica suggested that automated recidivism risk scores, used in sentencing and bail decisions, show racial bias [ALMK16]. Following the organization of workshops and conferences addressing related topics, research into algorithmic methods for mitigating disparate impact has accelerated.

11.1.2 An Overview of Disparate Learning Processes

To combat prejudice in ML-based decisions, there has been significant research interest in developing algorithms that constrain the level of disparate impact. The proposed approaches vary both in the working definition of fairness, and the algorithmic mechanisms used to ensure it. Most problem setups assume a binary classification setting in which the positive class is preferable. For example, the positive class might correspond to the designation "credit-worthy" or "employable", which would lead a decision system to issue a loan, or recommend a job candidate, respectively. A common approach is then to cast the fairness problem as that of choosing a model that minimizes the disparate impact with minimal reduction in accuracy. A number of these papers [PRT08, KAS11, ZVGRG17, BL17] propose to remove disparate impact without resorting to disparate treatment.

On the surface, this is a desirable goal. Disparate treatment and disparate impact may be seen as complementary definitions, both describing discriminatory mechanisms. In another sense, however, these definitions are in opposition. Given a decision-making system that exhibits disparate impact, the most obvious fix is to apply disparate treatment in favor of the disadvantaged group. Disparate treatment in the service of equality or diversity is sometimes described as *affirmative action* and *reverse-discrimination* [ZVGRG17]. In some cases, the courts have upheld the legality of applying disparate treatment to improve diversity, but there is also popular resentment of *affirmative action* and its future legality remains contested [BS16].

Many of the algorithmic proposals to reduce disparate impact while avoiding disparate treatment—which we denote *disparate learning processes* (DLPs)—operate according to the following principle: The protected characteristic may be used during training, but is not available to the model at prediction time. In the earliest such approach [PRT08], the protected characteristic was used to winnow the set of acceptable rules from an expert system. In other papers, the protected characteristic is incorporated into the learning objective (as a regularizer or constraint) or is used in pre-processing the training data [KC09, KCP10, ZVGRG17]. These approaches are grounded in the premise that DLPs are acceptable in cases where using a protected characteristic as a direct input to a model would constitute disparate treatment and thus be impermissible.

In this paper, we call this premise into question on the following grounds:

1. Disparate treatment in the service of improving diversity has been upheld as legal.

2. As we will show, the optimal way to trade off accuracy for proportional representa-

tion in the positive class is to apply disparate treatment directly.

3. When the protected characteristic is redundantly encoded in the other features, any disparate treatment can be equally implemented through a DLP.

4. When the protected characteristic is partially encoded in the other features, disparate treatment induces within-class discrimination applying the benefit of the affirmative action unevenly, and can even harm some members of the protected class.

The fact that DLPs and disparate treatment are functionally equivalent when the protected attributes are redundantly encoded in non-protected features should cast doubt on the legality of these algorithms in contexts where disparate treatment is prohibited. A legal opinion by Grimmelmann and Westreich supports this view [GW17]:

> In our view, Title VII does not permit an employer to do indirectly what it could not do directly. An employer that explicitly selects applicants on the basis of [group membership] violates Title VII under a disparate treatment theory [...] regardless of whether it bears animus against particular [groups]. It is the selection on the basis of [group membership] that is the problem. An employer that uses home address to infer applicants [group membership] and then selects applicants from particular [groups] does exactly the same, only in two steps rather than one. This too is a form of disparate treatment.

While to our knowledge none of the existing approaches have considered the task of maximizing disparate impact, in principle similar mechanisms could be used to do so. Applying a DLP to improve the fortunes of the dominant class might be judged by a court to constitute intentional discrimination and thus be a form of disparate treatment. Such a judgment might apply to the mechanism irrespective of the outcome it was used to effect. This would undermine the chief argument for DLPs, since the same ends can be achieved more effectively by proportional representation-promoting disparate treatment. It is worth noting that an algorithm that disproportionately fortunes the dominant class is more likely to raise red flags (under disparate impact) than one that effects representative outcomes. So the extent to which DLPs can mask intentional discrimination is limited to levels that do not subject the practice to scrutiny on the basis of significant disparate impact. Problematically, the law is often ambiguous, and the legal status of these algorithms has not yet, to our knowledge, been tested in the courts, leaving practitioners and researchers without clear guidelines.

One potential source of ambiguity lies in whether we consider the algorithms to be correcting for biases in the dataset, or if the disparate learning process is deemed to be an explicit diversity-promoting *positive discrimination*. Problematically, when a dataset with discriminatory labeling (such as historical hiring decisions) arrives, it is seldom accompanied by meta-data precisely quantifying the prejudice. This leaves researchers to make extreme assumptions, e.g. that any correlation between the protected class and the label is attributable to discrimination. That assumption would entail numerous erroneous conclusions, such as that the high academic performance of Asian Americans is due to systemic pro-Asian discrimination, despite the abundant evidence to the contrary [HS17].

11.1.3 A Note on Organization

The rest of this chapter is organized as follows: In Section 11.2, we give a more technical description of DLPs. In Section 11.3, we demonstrate some simple theoretical problems with disparate learning processes. In Section 11.4, we demonstrate the tendency of DLPs to perpetrate within-class discrimination, applying them both to a clean synthetic dataset and a real dataset of University admissions data. These sections aim to be objective and aim to make no value judgments. Then, in Section 11.6, we express the position that the policy and technical communities might benefit from accepting the efficacy and transparency of explicit pro-diversity disparate treatment. We conclude with a discussion of the challenges that remain such as the issues posed by data bias, and the difficulty in navigating the terrain between estimation and decision-making.

Throughout this paper we strive to maintain a distinction between the objective question of whether an algorithm discriminates along a protected characteristic, and the normative question of whether that discrimination should be permitted because it serves a socially desirable goal, such as ensuring proportional representation. In an attempt to distinguish between the descriptive and the normative, we use an organization scheme that separates the discussion of these different questions. Sections 11.2, 11.3, and 11.4 are descriptive while Section 11.6 is normative.

11.2 Disparate Learning Processes

To begin our formal description of the prior work, we'll introduce some formal notation. A dataset X, Y, consists of *n examples*, or *data points* $\{\mathbf{x}_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$, each consisting of a feature vector $\mathbf{x}_i$ and a label y_i. A supervised learning algorithm produces a model $\hat{y} : \mathcal{X} \to \mathcal{Y}$, which given a feature vector $\mathbf{x_i}$, predicts the corresponding output $\mathbf{y}_i$. In this discussion, we'll focus on binary classification, the setting in which the label y takes values from the set $\{0,1\}$. We'll also focus on probabilistic classifiers, which produce estimates $\hat{p}(\mathbf{x})$ of the conditional probability $P(y = 1 \mid \mathbf{x})$ of the label given a feature vector $\mathbf{x}$. To make a prediction $\hat{y}(\mathbf{x}) \in \mathcal{Y}$ given an estimated probability $\hat{p}(\mathbf{x})$, a thresholding strategy is applied such that $\hat{y}_i = 1$ *if* $\hat{p} > t$. The optimal choice of the threshold t may depend on the performance metric being optimized. For instance, under $0-1$ loss, the optimal decision rule thresholds $P(y = 1 \mid \mathbf{x})$ at $t = 0.5$. To optimize F1 score, the optimal threshold depends on the classifier's confidence [LEN14]. Following prior work, we focus most of our analysis on the accuracy metric or $0-1$ loss.

Note that a supervised learning algorithm is itself a function, mapping from datasets to models $f : (\mathcal{X}^n, \mathcal{Y}^n) \to (\mathcal{X} \to [0,1])$. Additionally, some datasets possess a sensitive attribute Z, making each example a three-tuple $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i$. The protected characteristic may be real-valued, like age, or categorical, like race or gender. Following the related work, we focus on categorical characteristics, and look specifically at the binary case where the protected characteristic divides the population into groups a and b. As shorthand, we will refer to the number of members of classes a and b as $n_a = \sum_i^n \mathbb{1}(z_i = a)$ and $n_b = \sum_i^n \mathbb{1}(z_i = b)$, respectively.

The papers which propose DLPs generally consider training a classifier on the protected characteristic, i.e. with feature vector $\tilde{\mathbf{x}} = [\mathbf{x}; \mathbf{z}]$ to be impermissible, amounting to *disparate treatment*. However, even if the protected features $\mathbf{z}_i$ are discarded, the model may still produce probabilities of belonging to the the positive class $\hat{p}(\mathbf{x})$ that are correlated with $\mathbf{z}$. Applying thresholds to make decisions, the ML-based decisions might correlate with the protected characteristic: $P(\hat{y} = 1 \mid \mathbf{z}) \neq P(\hat{y} = 1)$. Empirically, we could estimate the proportions assigned to the positive class by evaluating the quantities $q_a = (\sum_{i:z_i=a} \mathbb{1}(\hat{y}(\mathbf{x_i}) > .5))/n_a$ and $q_b = (\sum_{i:z_i=b} \mathbb{1}(\hat{y}(\mathbf{x_i} > .5))/n_b$.

When our goal in learning is simply to maximize accuracy, the estimation of $P(y = 1|\mathbf{x})$ simplifies to the standard problem of binary classification. However, some papers propose trading off the accuracy of the classifier for reduction in disparate impact [PRT08, KC09, ZVGRG17]. Different papers address different measures of disparate impact, but a few formulations are common. One approach addresses the Calders-Verwer (CV) gap, which quantifies disparate impact as $q_b - q_a$, the difference between the proportions assigned to the positive class in the disadvantaged group (a) and the advantaged group (b) [KAS11]. This definition is asymmetric in that it assumes a disadvantaged class, but could be made more generic by taking the absolute value of the difference. In [ZVGRG17], the proposed measure of disparate impact is q_a/q_b, following a book by Biddle on fair employment practices [Bid06], assuming that a is the disadvantaged group. They propose constraining a model to satisfy a *p-% rule*:

$$q_a/q_b > p/100. \tag{11.1}$$

This generalizes the heuristic in [Bid06] that disparate impact can be diagnosed when the ratio of proportions assigned to the positive class is less than .8 (a p-% rule of 80). For simplicity, we focus on these two scores, but our results can be trivially extended to some (but possibly not all) measures of disparate impact. For example, [BL17], proposes another regularization-based scheme with two penalty terms. The first minimizes the difference in the false negative rate between the two groups, and the other minimizes the difference in the false positive rate. A similar analysis can be applied there.

Many papers propose to minimize their chosen measure of disparate impact to within some acceptable range with minimal reduction of accuracy. But even calculating these measures requires access to the sensitive feature. These papers state that incorporating the feature directly in determining the class assignments $\tilde{y}_i$ constitutes disparate treatment. So they propose instead to incorporate the sensitive feature in the learning algorithm, but not the model. The formal function signature for a disparate learning process follows:

$$DLP : (\mathcal{X}^n, \mathcal{Y}^n, \mathcal{Z}^n) \to (\mathcal{X} \to \mathcal{Y}). \tag{11.2}$$

Since **z** is not a direct input of the resulting model, it is often asserted that such a model has a better legal standing than a model that uses **z** directly. We argue qualitatively that this claim is unreasonable, both with respect to the law [GW17] and because under some circumstances the DLP is technically equivalent to disparate treatment.

Before addressing any qualitative arguments, we first demonstrate some theoretical properties of disparate treatment, demonstrating among other things that it optimally addresses the constrained optimization problem posed by several authors of DLP papers.

11.3 Theoretical Issues

In this section we introduce theoretical arguments in support of counterarguments (2-4) outlined in Section 11.1.2. We present a set of simple theoretical results that demonstrate the optimality of disparate treatment, and highlight properties of DLPs. Our optimality results are all derived in the population or "infinite data" setting where we assume knowledge of the true conditional probability function $p_{Y|X,Z}(x,z) \equiv \mathbb{P}(Y = 1 \mid X = x, Z = z)$. The main results can be summarized as follows.

1. Direct disparate treatment on the basis of z is the optimal strategy for minimizing the expected $0-1$ loss subject to CV and p-% constraints.
2. When X fully encodes Z, a sufficiently powerful DLP is equivalent to disparate treatment.
3. When X only partially encodes Z, a DLP may be suboptimal and induce intra-group disparity.

11.3.1 Disparate treatment is optimal

Absent disparate impact constraints, the Bayes-optimal decision rule for minimizing expected $0-1$ loss (i.e., maximizing accuracy) is given by

$$d^*_{\text{uncon}}(x,z) = \begin{cases} 1 & p_{Y|X,Z}(x,z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}.$$

In this section we show that the optimal decision rule in the CV and p-% constrained problems has a similar form. The optimal decision rule will again be based on thresholding $p_{Y|X,Z}(x,z)$, but at *group-specific thresholds*. These rules can be be thought of as operationalizing the following disparate treatment mechanism. Suppose that we start with the classifications of the unconstrained rule $d^*_{\text{uncon}}(x,z)$, and this allocated results in a CV gap of $q_b - q_a > \gamma$. To reduce the CV gap to γ we have two mechanisms. We can flip predictions of cases in group a from 0 to 1, and we can also flip predictions of cases in group b from 1 to 0. The optimal strategy is to perform these flips on group a cases that have the highest value of $p_{Y|X,Z}(x,z)$ and group b cases that have the lowest value of $p_{Y|X,Z}(x,z)$.

The results in this section adapt the work of [CDPF$^+$17], who establish optimal decision rules d under different kinds of fairness constraints. In this work the authors characterize the optimal decision rule $d = d(x,z)$ that maximizes the *immediate utility* $u(d,c) = \mathbb{E}[Yd(X,Z) - cd(X,Z)]$ $(0 < c < 1)$ under different parity criteria. We begin with a lemma showing that expected classification accuracy has the functional form of an immediate utility function.

Lemma 7. Optimizing classification accuracy is equivalent to optimizing immediate utility with $c = 0.5$.

Proof. The expected accuracy of a binary decision rule $d(X)$ can be written as $\mathbb{E}[Yd(X) + (1-Y)(1-d(X))]$. Expanding and rearranging this expression gives

$$\begin{aligned}\mathbb{E}[Yd(X) + (1-Y)(1-d(X))] &= \mathbb{E}(2Yd(X) - d(X)) + \mathbb{E}(Y) + 1 \\ &= 2u(d,0.5) + \mathbb{E}(Y) + 1\end{aligned}$$

The only term in this expression that depends on d is the immediate utility, u. Thus the decision rule that maximizes u also maximizes accuracy. □

For the next set of results, we follow [CDPF$^+$17] and assume that $p_{Y|X,Z}(X,Z)$ viewed as a random variable has positive density on $[0,1]$. This ensures that the optimal rules are unique and deterministic by disallowing point-masses of probability that would necessitate tie breaking among observations with equal probability. The first result that we state is a direct corollary of two results in [CDPF$^+$17]. It considers the case where we desire exact parity; i.e., the constraint $q_a = q_b$.

Corollary 8. The optimal decision rules d^* under various fairness constraints have the following form, and are unique up to a set of probability zero.

1. Among rules satisfying statistical parity (the 100% rule), the optimum is

$$d^*(x,z) = \begin{cases} 1 & p_{Y|X,Z}(x,z) \geq t_z \\ 0 & \text{otherwise} \end{cases}$$

   where $t_z \in [0,1]$ are constants that depend only on group membership z.

2. Among rules that have equal false positive rates across groups, the optimum is

$$d^*(x,z) = \begin{cases} 1 & p_{Y|X,Z}(x,z) \geq s_z \\ 0 & \text{otherwise} \end{cases}$$

   where s_z are constants that depend only on group membership z (but are different from t_z).

3. (1) and (2) continue to hold even in the resource-constrained setting where the overall proportion of cases classified as positive is constrained.

Proof. (1) and (2) are direct corollaries of Lemma 7 combined with Theorem 3.2 and Prop 3.3 of [CDPF$^+$17]. □

The next set of results establish optimality under general p-% and CV rules.

Proposition 9. Under the same assumptions as above, the optimum among rules that satisfy the CV constraint $0 \leq q_b - q_a < \gamma$ or the p-% rule also has the form

$$d^*(x,z) = \begin{cases} 1 & p_{Y|X,Z}(x,z) \geq t_z \\ 0 & \text{otherwise} \end{cases},$$

where $t_z \in [0,1]$ are constants that depend on the group membership z, and on the choice of constraint parameter γ or p. The thresholds t_z are different for the CV constraint and p-% rule.

Proof. Suppose that the optimal solution under the CV or p-% rule constraint classifies as positive q_a proportion of disadvantaged cases and q_b proportion of advantaged cases. As shown in [CDPF$^+$17], we can rewrite the immediate utility as

$$u(d, 0.5) = \mathbb{E}[d(X,Z)(p_{Y|X,Z} - 0.5)].$$

From this expression it is clear that the utility will be maximized precisely when $d^*(X,Z) = 1$ for the q_z proportion of individuals in each group that have the highest values of $p_{Y|X,Z}$. Since the optimal values of q_z may differ between the CV constrained solution and the p-% solution, the optimal thresholds may differ as well. $\square$

The final result in this section shows that a decision rule that does not directly use z as an input variable or for determining the thresholds will have lower accuracy than the optimal rule that uses this information. That is, we show that DLPs are suboptimal for trading off between accuracy and disparate impact.

Theorem 10. *Let $d^*(X,Z)$ be the optimal decision rule under a the CV-γ or p-% constraint. Let $d_{DLP}(X)$ be the optimal solution to a DLP. If $d(X,Z)$ and $d_{DLP}(X)$ satisfy CV* *or p-% constraints with the same q_a and q_b, the DLP solution results in lower or equal accuracy. (Equal only if the solutions are the same.)*

Proof. From Proposition 9 we know that the unique accuracy optimizing solution is given by

$$d^*(x,z) = \begin{cases} 1 & p_{Y|X,Z}(x,z) \geq t_z \\ 0 & \text{otherwise} \end{cases}$$

where t_z is the 1 - q_z quantile of $p_{Y|X,Z}$. The difference in immediate utility between the two decision rules can be expressed as follows.

$$\begin{aligned}
&\mathbb{E}[d^*(X,Z)(p_{Y|X,Z} - 0.5)] - \mathbb{E}[d_{DLP}(X)(p_{Y|X,Z} - 0.5)] \\
&= \mathbb{E}[(d^*(X,Z) - d_{DLP}(X))(p_{Y|X,Z} - 0.5)] \\
&= \mathbb{E}[p_{Y|X,Z} - 0.5 \mid d^* = 1, d_{DLP} = 0]\mathbb{P}(d^* = 1, d_{DLP} = 0) \\
&\qquad - \mathbb{E}[p_{Y|X,Z} - 0.5 \mid d^* = 0, d_{DLP} = 1]\mathbb{P}(d^* = 0, d_{DLP} = 1) \\
&= \big(\mathbb{E}[p_{Y|X,Z} - 0.5 \mid d^* = 1, d_{DLP} = 0] \\
&\qquad - \mathbb{E}[p_{Y|X,Z} - 0.5 \mid d^* = 0, d_{DLP} = 1]\big)\mathbb{P}(d^* = 1, d_{DLP} = 0) \\
&\geq 0
\end{aligned}$$

The final inequality follows from the observation that $d^*(X,Z) = 1$ for the highest values of $p_{Y|X,Z}$, so $p_{Y|X,Z}$ is stochastically greater on the event $\{d^* = 1, d_{DLP} = 0\}$ than on $\{d^* = 0, d_{DLP} = 1\}$. Note that equality holds only if $\mathbb{P}(d^* = 1, d_{DLP} = 0) = 0$; that is, if the two rules are equivalent with probability 1. □

All of the results in this section continue to hold under "do no harm" constraints where the proportion of cases in the disadvantaged group classified as positive is constrained to be no lower than the proportion under the unconstrained rule $d_{\text{uncons}}(x,z)$ (or no lower than some fixed value q_a^{min}). This constraint imposes an upper bound on the optimal thresholds t_a, but does not change the structure of the optimal rules.

11.3.2 Functional equivalence when protected characteristic is redundantly encoded

Consider the case where the protected characteristic z is redundantly encoded in the permissible data features x. More precisely, suppose that there exists a known subcomputation g such that $z_i = g(x_i)\,\forall i$. This allows for any function of the data $f(x,z)$ to be represented as a function of x alone via $\tilde{f}(x) = f(x, g(x))$. While it remains the case that $\tilde{f}(x)$ does not directly use z as an input variable, $\tilde{f}$ should be no less suspect from a disparate impact perspective than the original function f that uses z directly. The main difference for the purpose of our discussion is that $\tilde{f}$ is a valid possible result from a DLP whereas f is not. Yet it is clear that f and $\tilde{f}$ are functionally equivalent in two senses: (i) as mathematical objects; and more importantly, (ii) as mechanisms for classifying cases.

11.3.3 Within-class differentiation when protected characteristic is partially redundantly encoded

When the protected characteristic is partially encoded in the other features, disparate treatment may induce within-class discrimination by applying the benefit of the affirmative action unevenly, and can even harm some members of the protected class. This claim asserts a possibility, so it is sufficient to produce one example to support the claim. In the following section, we establish the claim empirically using both synthetic data and real university admissions data. The ease of producing such examples might convince the reader that the highly varied effects of intervention with a DLP on members of the disadvantaged group raise serious questions about the usefulness of DLPs.

11.4 Empirical Analysis

This simple analysis that precedes makes plain several advantages to mitigating disparate impact by applying disparate treatment:

- **Optimality:** As demonstrated for CV score and for p-% rule, the disparate treatment intervention maximizes accuracy subject to a constraint on disparate impact.
- **Rational ordering:** Within each group, individuals with higher probability of belonging to the positive class are always assigned to the positive class ahead of those with lower probabilities.
- **Does no harm to the protected group:** The disparate treatment intervention can only benefit members of the disadvantaged class.

But DLPs do not directly apply disparate treatment. Instead, they must recover a classifier that satisfies the disparate impact constraints, by relying upon the proxy features to minimize the disparate impact measure. In many of these papers, this is accomplished either by introducing constraints to a convex optimization problem, or by adding a regularization term and tuning the corresponding hyper-parameter. Because the CV score and p-% rule are non-convex in model parameters (scores only change when a point crosses the decision boundary), [KAS11, ZVGRG17] introduce convex surrogates aimed at reducing the correlation between the sensitive feature and the prediction.

All of these approaches assume that the proxy variables contain information about the sensitive attribute. Otherwise, the model could only achieve fairness by arriving at a trivial solution (e.g., assign everyone or no one to the positive class). So we must consider two scenarios: (i) the proxy variables $\mathbf{x}$ fully redundantly encode z. In this case, an sufficiently powerful DLP will implicitly reconstruct z, because this gives the optimal solution to the impact-constrained objective. However, when $\mathbf{x}$ doesn't fully capture z, then the DLP may (i) be sub-optimal, (ii) violate rational ordering within groups, and (iii) harm members of the disadvantaged group.

11.4.1 Synthetic data example: work experience and hair length in hiring

To begin, we confirm these arguments empirically with a simple synthetic data experiment. To construct the data we sample $n_{all} = 2000$ total observations from the data generating process described below. 70% of the observations are used for training, and the remaining 30% are reserved for model testing.

$$\begin{aligned}
z_i &\sim \text{Bernoulli}(0.5)\\
\text{hair\_length}_i \mid z_i = 1 &\sim 35 \cdot \text{Beta}(2,2)\\
\text{hair\_length}_i \mid z_i = 0 &\sim 35 \cdot \text{Beta}(2,7)\\
\text{work\_exp}_i \mid z_i &\sim \text{Poisson}(25+6z_i) - \text{Normal}(20, \sigma = 0.2)\\
y_i \mid \text{work\_exp} &\sim 2 \cdot \text{Bernoulli}(p_i) - 1, \text{ where}\\
p_i &= 1/\left(1+\exp[-(-25.5+2.5\text{work\_exp})]\right)
\end{aligned}$$

This data generating process has the following key properties: (i) The historical hiring process was based solely on the number of years of work experience; (ii) Because women on on average have fewer years of work experience than men (5 years vs. 11), men have been hired at a much higher rate than women; (iii) Women have longer hair than men, a fact that was irrelevant to historical hiring practice.

Figure 11.1 shows the test set results of applying a DLP to the available historical data to equalize hiring rates between men and women. We apply the DLP proposed by

[ZVGRG17], using code available from the authors.[1] While the DLP is successful in equalizing hiring rates (satisfying a 100-% rule), it does so through a problematic within-class discrimination mechanism. The DLP rule advantages individuals with very long hair length over those with short hair length and considerably longer work experience. We find that several women who would have been hired under historical practices owing to their 11+ years of work experience would not be hired under the DLP due to their short hair length (i.e., their male-like characteristics in the data). Similarly, several men who would not have been hired based on work experience alone are advantaged by the DLP on account of their longer hair length (i.e., their female-like characteristics in the data). The DLP mechanism violates rational ordering, and also has the effect of harming some of the most qualified individuals in the protected group. Group parity is achieved at the cost of significant individual unfairness.

Granted, factors such as hair length could not knowingly and defensibly be used as an input to a typical hiring algorithm. This example was constructed to illustrate a more general point. Since DLPs do not have direct access to the protected attribute, they must infer from the data cases that are most likely to be members of each subgroup. Using the protected attribute directly yields more reasonable policies: ones that hire the most qualified individuals in each group, rather than those that are most qualified among those that *appear*, from their un-protected characteristics, to be the most feminine.

11.4.2 Case Study: Gender Bias in CS Graduate Admissions

For our next example, we considered data from the Master's admissions process of a large public university, considering a sample of $\sim$9,000 students considered for admission over an 11-year period spanning 2006-2016. Half are withheld for testing. The available attributes include basic demographic information, such as country of origin,

[1]`https://github.com/mbilalzafar/fair-classification/`

interest areas, and gender, as well as quantitative information such as GRE scores. Finally, it includes a label in the form of a decision provided by an admissions committee.[2]

Based on a superficial analysis, the data does not appear to exhibit gender bias (the admissions rates for male and female applicants are within 1% of each other). So, for the purposes of our experiments, we corrupt the data with *synthetic discrimination*. Of all women who were admitted, i.e., $z_i = a, y_i = 1$, we flip 25% of those labels to 0: giving noisy labels $\bar{y}_i = y_i \cdot \eta$, for $\eta \sim \mathit{Bernoulli}(.25)$. This simulates a setting in which the training data exhibits a historical bias.

We then train three logistic regressors: (1) To predict the (prejudice-corrupted) labels from the non-sensitive features $\{x_i, \bar{y}_i\}$; (2) The same model, applying the fairness constraint of [ZVGRG17]; and (3) A logistic regressor that predicts the sensitive feature from the non-sensitive features $\{x_i, z_i\}$. The data contains limited information that can predict gender, though such predictions can be made better than random (AUC=0.59) due to different rates of gender imbalance across (e.g.) countries and interest areas.

Figure 11.2 (left) shapes our basic intuition for what is happening here: Considering the probability of admission for the unconstrained classifier (y-axis), students whose decisions are 'flipped' (after applying the fairness constraint) tend to be those close to the decision boundary. Furthermore, students predicted to be male (x-axis) tend to be flipped to the negative class (left half of plot) while students predicted to be female tend to be flipped to the positive class (right half of plot). This is shown in detail in Figure 11.2 (center and right). However, of the 19 students whose decisions are flipped to 'admit,' the majority (10) are males, each of whom has 'female-like' characteristics according to their other features. Demonstrated here with real-world data, the DLP both disrupts the within-group ordering, and violates the *do no harm* principle by disadvantaging some

[2]These decisions do not precisely determine whether a student is made an offer, but rather represent an 'above-the-bar' assessment that is used to guide admissions decisions, and can be considered as a binary label.

women who, but for the DLP, would have been admitted.

Comparison with Disparate Treatment

To demonstrate the better performance of disparate treatment, we implement a simple thresholding scheme. Assuming that our model gives us calibrated probabilities, and that this is all the information available to the decision maker, it's easy to derive the optimal thresholds for maximizing accuracy under linear constraints on the proportions of predicted positives, like the CV-gap or p-% rule.

We now present a simple thresholding scheme for maximizing accuracy subject to a p-% rule. Recall that the p-% rule requires that $q_a/q_b > p/100$. We can rewrite this as:

$$\frac{p}{100}q_b - q_a < 0 \tag{11.3}$$

Like the CV-gap, the p-% rule imposes a linear constraint. We denote the quantity $\frac{p}{100}q_b - q_a$ the p-gap. To maximize accuracy subject to satisfying the p-% rule, we construct a score, that quantifies reduction in p-gap per reduction in accuracy. Starting from the accuracy-maximizing predictions (thresholded at .5), we then flip those predictions which close the gap fastest:

1. Assign each example with $\{\tilde{y}_i = 0, z_i = a\}$ or $\{\tilde{y}_i = 1, z_i = b\}$, a score c_i equal to the reduction in the CV-gap divided by the reduction in accuracy:

   (a) For each example in group a with initial $\tilde{y}_i = 0$,

   $c_i = \frac{n}{n_a(1-2\hat{p}_i)}$.

   (b) For each example in group b with initial $\hat{y}_i = 1$,

   $c_i = \frac{np}{100 n_b(2\hat{p}_i-1)}$.

2. Flip examples in descending order according to this score until the desired CV-score is reached.

These scores do not change after each iteration. So the greedy policy is optimal.

Overall, the fairness constraint of [ZVGRG17] achieves a p-% rule of 77.59%, compared to a p-% rule of 71.44% by naïve classification (on unseen test data). Both have similar accuracy: given that both positive labels and female applicants are a minority, assigning negative labels to males close to the boundary impacts accuracy very little. Both methods had accuracy of around 78% on this data. Critically though, by applying an optimal thresholding strategy, we were able to obtain the same accuracy as the method of [ZVGRG17], but with a higher p-% rule of 78.34%. Similarly, we could achieve a modest improvement in accuracy (¡0.1%) while maintaining the same p-% rule as the method of [ZVGRG17].

11.5 Related Work

In this section we provide a brief overview of some of the other approaches that have been put forth for trading off between classification performance and disparate impact. One common approach consists of preprocessing or "massaging" the training data to reduce the dependence between the resulting model predictions and the sensitive attribute [KC09, KC12, FFM$^+$15, AFF$^+$16, JL17]. These methods differ both in terms of what variables are affected by the data processing, and the degree of independence that is achieved. For instance, [KC09] propose flipping the negative labels of some observations in the disadvantaged class. [ZWS$^+$13] proposes learning representations - in this case, cluster assignments - of each example such that each example maps to a cluster with some probability. They seek statistical parity in the percentage withing each group assigned to each cluster. [FFM$^+$15] also investigates transformations of the features X into a new set of features that are constructed to be marginally independent from Z. [JL17] demonstrate how to construct transformations to ensure that the derived features are jointly independent of Z, and show that this produces distributional parity of the resulting fitted model.

A second widely adopted approach is to modify existing classification methods either through post-hoc corrections or in the training stage to constrain the level of disparate impact in the resulting model. [KAS11, GCGF16, CV10, KCP10] consider modifications to methods such as SVM, logistic regression, Naive Bayes and decision trees. [ABDL17] show how disparate impact constraints can be framed as a cost-sensitive classification problem.

11.6 Discussion

Following our description of the problem, theoretical analysis, and empirical findings, we now offer a more normative take on these findings.

11.6.1 Coming to Terms with Disparate Treatment

At present, most legal scholarship and technical machine learning scholarship take place in a disjoint set of journals. These communities occasionally intersect when some paper, such as the widely influential California Law Review article by Barocas and Selbst reaches a cross-disciplinary audience [BS16]. However, the subsequent interdisciplinary technical work tends to be published in technical conferences, where the peer-reviewers may be ill-equipped to identify shortcomings in problem formulation.

For instance, the interpretations of anti-discrimination law that motivate DLPs appear not to consider that (i) present-day law might rule DLPs to be equivalent to disparate treatment if tested under the law (see e.g., arguments in [GW17]); and, (ii) disparate treatment may already be tolerated under the law in order to ensure more fair outcomes. This latter view is supported by Pauline Kim in her paper *Data-driven* *discrimination at work* [Kim17]:

> A formalist reading of Title VII might appear to prohibit any use of variables capturing sensitive characteristics in a data model. Certainly, a simple model that relied on race or other protected characteristics as the basis for adverse decisions would run afoul of Title VIIs prohibitions. However, when dealing with a complex statistical model involving multiple variables, the appropriate treatment of these sensitive variables is more complicated. If the goal is to reduce biased outcomes, then a simple prohibition on using data about race or sex could be either wholly ineffective or actually counterproductive due to the existence of class proxies and the risk of omitted variable bias. Instead, avoiding classification bias may sometimes call for excluding sensitive demographic variables and at other times call for including them. Any response to biased data models must be sensitive to these nuances.

On the balance of these considerations, there are several compelling reasons for practitioners to promote equality more transparently through direct disparate treatment, rather than through hidden changes to the learning algorithm. As articulated earlier, a disparate treatment based approaches have three principal advantages over DLPs: they (i) optimally trade accuracy for representativeness; (ii) preserve rankings among members of each group (as compared to the unconstrained scores); and (iii) do no harm to members of the disadvantaged group.

In addition to these three properties, disparate treatment has another advantage. By setting class-dependent thresholds, it's much easier to quantify how disparate treatment impacts individuals. Having an intuitive quantity to reason about might help policy-makers to decide what magnitude of disparate treatment best trades off group equality and individual fairness. For more indirect methods to satisfy disparate impact constraints, it might be hard to reason about the the intervention. As an example, it seems doubtful that policy-makers would similarly intuitive meaning in the setting of a specific regularization coefficient.

Several key challenges still remain. The theoretical arguments in this paper demonstrate that disparate treatment approaches are optimal in the setting where we assume complete knowledge of the data generating distribution. It is not always clear how best to realize these gains in practice, where imbalanced or unrepresentative data sets can pose a significant obstacle to accurate estimation. Furthermore, some of our results are tailored to the CV or the *p*-% rule notions of fairness. As shown in [HPS$^+$16, WGOS17] and [DIKL17], the situation can much more complicated for other fairness criteria.

11.6.2 Separating Estimation and Decision-Making

In many algorithm-supported decision making contexts, it is desirable to obtain not just a classification, but also an accurate probability estimate. These estimates could then be incorporated into the decision-theoretic part of the pipeline and appropriate measures could be taken at that stage to ensure the desired outcome properties. By intervening at the modeling phase, DLPs risk distorting the probabilities themselves. It is not clear what the probabilities that come out of the resulting classifiers actually signify. In unconstrained learning approaches, even if the label itself may reflect historical prejudice, one at least knows what is being estimated. This leaves open the possibility of intervening at decision time to promote more equitable outcomes.

While the distinction between building a model and making decisions is stated clearly in the first modern work on fairness in discrimination-aware classification [PRT08], this distinction is frequently muddled in discussions of algorithmic fairness. For example, [KC09] state that “a learned model may exhibit unlawfully prejudiced behavior”. The conflation of modeling and decision-making may lead to counterproductive corrections to the models that do not adequately account for how the models are actually used. For example, it is commonly assumed that decision makers desire to optimize accuracy and hence that decisions will be made by thresholding probability estimates at .5. This is often not the case. First, due to differences in the cost of false positives and false negatives, accuracy is seldom a task-relevant metric. Furthermore, decision-makers are often faced with a multi-objective problem that entails considerations beyond what the algorithm is designed to predict.

11.6.3 Fairness beyond disparate impact

How best to quantify discrimination and unfairness remains an important open question. The CV scores and $p-\%$ rules addressed in this paper offer one set of definitions (often technically termed ‘disparate impact’), but there are many other notions of fairness to which our results do not directly apply. For example, equality of opportunity as introduced in [HPS$^+$16]—requiring equality of true positive rates across groups—has received considerable attention. Other conditional notions of fairness and trade-offs between them have been studied by [JKM$^+$16, KMR16, Cho17, BHJ$^+$17, RSZ17]. The work of [ZVR$^+$17] departs from parity-based definitions and proposes instead a preference-based notion of fairness. [DIKL17] address the problem of how best to incorporate information about protected attributes for several of these other fairness criteria.

Problematically, research into fair algorithms is often motivated by the case in which our ground-truth data is itself biased. It is not clear how to assess many of these other fairness criteria in the presence of biased data. Characterizing different forms of data bias and their impacts on fairness assessment remains an important outstanding challenge.

Even if we accept that the solution for many proportional representation problems will take the form of disparate treatment in favor of the disadvantaged class, a question remains of “how much?”. At what point is the disparate treatment simply correcting for biased labels? At what point does it more explicitly amount to affirmative action? Recent work on identifying proxy discrimination [DFK$^+$17] and causal formulations of fairness [NS17, KRCP$^+$17, KLRS17] offer approaches to framing such problems. To answer these questions, it would help to have a better understanding of by what mechanisms and to what degree the data has been influenced by prejudice. Perhaps data mining and machine learning have some role to play in asking these questions? The answers could guide decisions about where and how strongly to intervene.

11.7 Acknowledgments

TChapter 11, *Does Mitigating ML's Disparate Impact Require Disparate Treatment?*, was written with Alexandra Chouldechova and Julian McAuley. Thanks to Charles Elkan and Yu-Xiang Wang for critical feedback. The dissertation author was the primary investigator and author of this paper.

If there's any appendix material, it will go here.

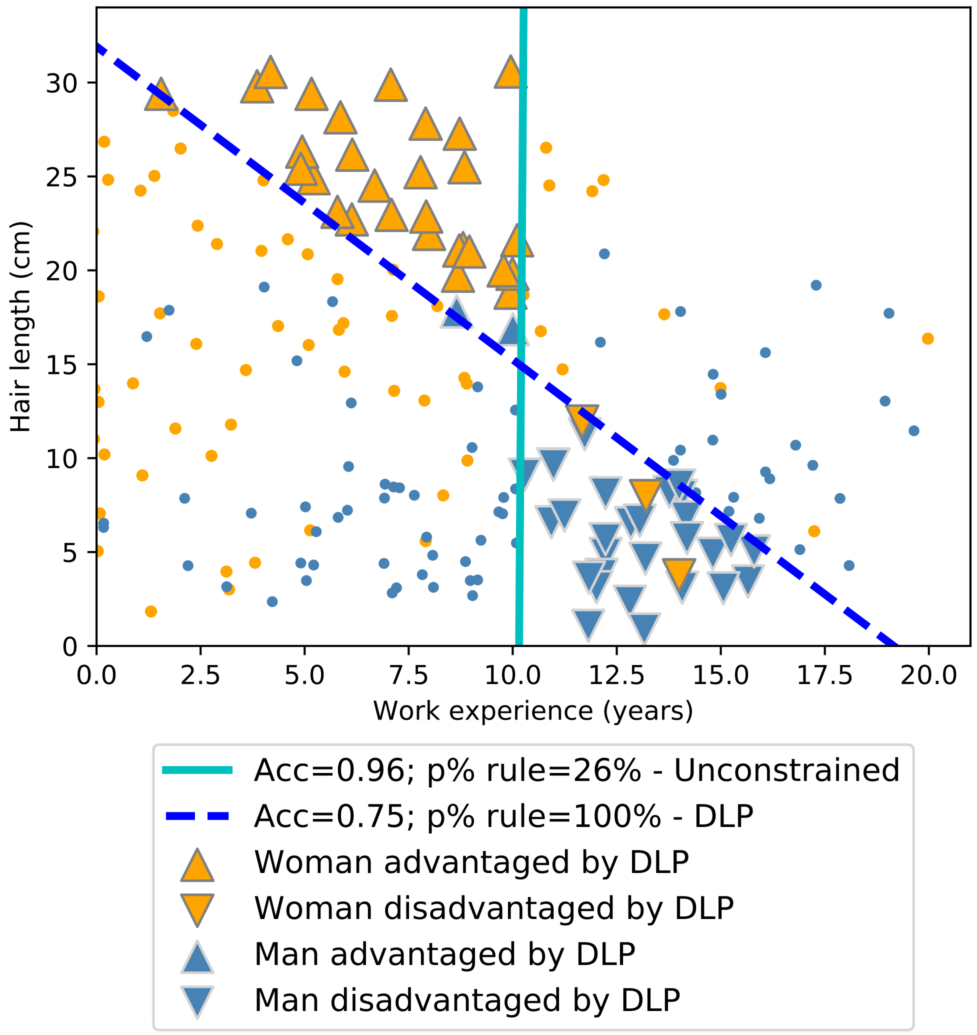


Figure 11.1: Unconstrained classifier (solid vertical) hires candidates based on work experience, giving higher hiring rates for men than women. DLP classifier (dashed diagonal) achieves parity at the cost of within-class differentiation based on an irrelevant attribute (hair length).

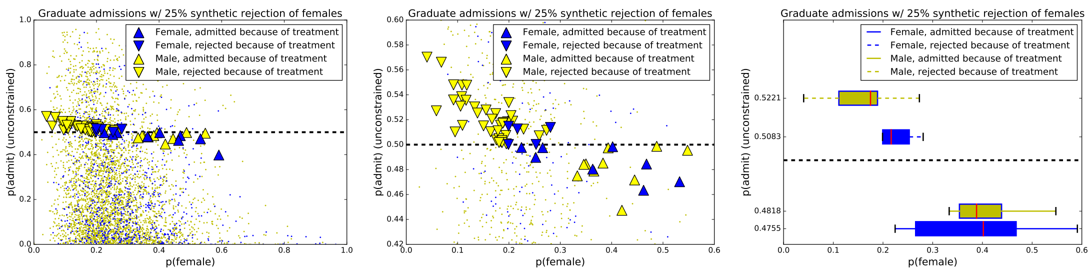


Figure 11.2: Left: probability of the sensitive variable versus (unconstrained) admission probability, on unseen test data. Points above 0.5 are individuals who are classified as ‘reject’ *only after applying the fairness constraint*; points below 0.5 are individuals who are classified as ‘admit’ only after applying the fairness constraint; the remaining ∼4,000 individuals (whose labels were not altered by the fairness constraint) are shown as blue/yellow dots. Note that most students admitted due to the fairness approach are actually males who ‘look like’ females on the basis of their other features, whereas females who reflect male characteristics are more likely to be rejected. Center: detail view of the same plot. Right: summary statistics of the same plot.

Bibliography

[ABDL17] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, and John Langford. A reductions approach to fair classification. 2017.

[AC98] Shun-ichi Amari and Andrzej Cichocki. Adaptive blind signal processing-neural network approaches. *Proceedings of the IEEE*, 1998.

[AFF$^+$16] Philip Adler, Casey Falk, Sorelle A Friedler, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian. Auditing black-box models by obscuring features. *arXiv:1602.07043*, 2016.

[AGQZ13] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. Joint language and translation modeling with recurrent neural networks. In *EMNLP*, pages 1044–1054, 2013.

[AI15] Susan Athey and Guido W Imbens. Machine learning methods for estimating heterogeneous causal effects. *Stat*, 2015.

[Aka09] Mehmet Fatih Akay. Support vector machines combined with feature selection for breast cancer diagnosis. *Expert Systems with Applications*, 2009.

[All01] Paul D Allison. *Missing data*. 2001.

[ALL$^+$09] John Asmuth, Lihong Li, Michael L. Littman, Ali Nouri, and David Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *UAI*, 2009.

[ALMK16] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. 2016.

[ARM$^+$09] Norm Aleks, Stuart J Russell, Michael G Madden, Diane Morabito, Kristan Staudenmayer, Mitchell Cohen, and Geoffrey T Manley. Probabilistic detection of short events, with application to critical care monitoring. In *NIPS*, 2009.

[B$^+$95] Leemon Baird et al. Residual algorithms: Reinforcement learning with function approximation. 1995.

[Bax95] W.G Baxt. Application of artificial neural networks to clinical medicine. *The Lancet*, 1995.

[BBB$^+$10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX, 2010.

[BBLP13] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *ICASSP*. IEEE, 2013.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

[BCKW15] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *ICML*, 2015.

[BE15] Bilberto Batres-Estrada. Deep learning for multivariate financial time series. 2015.

[Bea16] Greg Brockman et al. OpenAI gym. *arXiv:1606.03152*, 2016.

[Ben83] Steve Bennett. Log-logistic regression models for survival data. *Applied Statistics*, 1983.

[BG96] Yoshua Bengio and Francois Gingras. Recurrent neural networks for missing or asynchronous data. 1996.

[BGC01] Jon Barker, Phil Green, and Martin Cooke. Linking auditory scene analysis and robust asr by missing data techniques. 2001.

[BHJ$^+$17] Richard Berk, Hoda Heidari, Shahin Jabbari, Michael Kearns, and Aaron Roth. Fairness in criminal justice risk assessments: The state of the art. *arXiv:1703.09207*, 2017.

[Bid06] Dan Biddle. *Adverse impact and test validation: A practitioner's guide to valid and defensible employment testing*. Gower Publishing, Ltd., 2006.

[BKT$^+$11] Karl Y Bilimoria, Clifford Y Ko, James S Tomlinson, Andrew K Stewart, Mark S Talamonti, Denise L Hynes, David P Winchester, and David J Bentrem. Wait times for cancer surgery in the united states: trends and predictors of delays. *Annals of surgery*, 2011.

[BL17] Yahav Bechavod and Katrina Ligett. Learning fair classifiers: A regularization-inspired approach. *arXiv:1707.00044*, 2017.

[BM95] Justin Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *NIPS*, 1995.

[BMS90] Richard K Belew, John McInerney, and Nicol N Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In *CSE Technical Report CS90-174*, 1990.

[BNVB13] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 2013.

[BP03] Pierre Baldi and Gianluca Pollastri. The principled design of large-scale recursive neural network architectures–DAG-RNNs and the protein structure prediction problem. *JMLR*, 2003.

[BR93] Avrim L Blum and Ronald L Rivest. Training a 3-node neural network is NP-complete. In *Machine learning: From theory to applications*, pages 9–28. 1993.

[BS16] Solon Barocas and Andrew D Selbst. Big data's disparate impact. *California Law Review*, 2016.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 1994.

[BSO$^+$16] Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016.

[BWTS09] Justin Bayer, Daan Wierstra, Julian Togelius, and Jürgen Schmidhuber. Evolving memory cell structures for sequence learning. In *Artificial Neural Networks–ICANN 2009*. 2009.

[Car08] Bob Carpenter. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. *Alias-i, Inc., Tech. Rep*, 2008.

[CBM$^+$96] Rich Caruana, Shumeet Baluja, Tom Mitchell, et al. Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation. In *NIPS*, 1996.

[CBS04] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, 2004.

[CC75] Jacob Cohen and Patricia Cohen. Applied multiple regression/correlation analysis for the behavioral sciences. 1975.

[CCC71] Edward C. Capen, Robert V. Clapp, and William M. Campbell. Competitive bidding in high-risk situations. *Journal of Petroleum Technology*, 23(6):641–653, 1971.

[CDPF$^+$17] Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. Algorithmic decision making and the cost of fairness. *arXiv:1701.08230*, 2017.

[CGW$^+$09] Jonathan Chang, Sean Gerrish, Chong Wang, Jordan L Boyd-Graber, and David M Blei. Reading tea leaves: How humans interpret topic models. In *NIPS*, 2009.

[Cho17] Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, 2017.

[Civ64] Civil rights act of 1964, 1964. Accessed on September 11th, 2017.

[CKD$^+$99] Rich Caruana, Hooshang Kangarloo, JD Dionisio, Usha Sinha, and David Johnson. Case-based explanation of non-case-based learning methods. In *Proceedings of the AMIA Symposium*, 1999.

[CKF11] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[CKL$^+$15] Zhengping Che, David C. Kale, Wenzhe Li, Mohammad Taha Bahadori, and Yan Liu. Deep computational phenotyping. In *KDD*. ACM, 2015.

[CL11] Olivier Chapelle and Lihong Li. An empirical evaluation of Thompson sampling. In *NIPS*, 2011.

[CLG$^+$15] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noémie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *KDD*, 2015.

[CPC$^+$16] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *arXiv:1606.01865*, 2016.

[CT12] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[CV10] Toon Calders and Sicco Verwer. Three naive bayes approaches for discrimination-free classification. *Data Mining and Knowledge Discovery*, 2010.

[CZD15] Kai Chen, Yi Zhou, and Fangyan Dai. A lstm-based method for stock returns prediction: A case study of china stock market. In *IEEE International Conference on Big Data*, 2015.

[DAHG$^+$15] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[DC15] Filip Dabek and Jesus J. Caban. A neural network based model for predicting psychological conditions. In *Brain Informatics and Health*. 2015.

[dCDB09] Juan José del Coz, Jorge Diez, and Antonio Bahamonde. Learning nondeterministic classifiers. *JMLR*, 2009.

[DFK$^+$17] Anupam Datta, Matt Fredrikson, Gihyuk Ko, Piotr Mardziel, and Shayak Sen. Proxy non-discrimination in data-driven systems. *arXiv:1707.08120*, 2017.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011.

[DIKL17] Cynthia Dwork, Nicole Immorlica, Adam Tauman Kalai, and Max Leiserson. Decoupled classifiers for fair and efficient machine learning. *arXiv:1707.06613*, 2017.

[DKJ$^+$13] Krzysztof Dembczynski, Wojciech Kotłowski, Arkadiusz Jachnik, Willem Waegeman, and Eyke Hüllermeier. Optimizing the F-measure in multi-label classification: Plug-in rule approach versus structured loss minimization. In *ICML*, 2013.

[DL15] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[DPG$^+$14] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, 2014.

[DRS12] S Prasanna Devi, K Suryaprakasa Rao, and S Sai Sangeetha. Prediction of surgery times and scheduling of operation theaters in optholmology department. *Journal of medical systems*, 2012.

[DVWA15] Finale Doshi-Velez, Byron Wallace, and Ryan Adams. Graph-sparse lda: a topic model with structured sparsity. *AAAI*, 2015.

[DWCH11] Krzysztof Dembczyński, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. An exact algorithm for F-measure maximization. In *NIPS*, 2011.

[DZLD] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction.

[Elk] Charles Elkan. Learning meanings for sentences. `http://cseweb.ucsd.edu/~elkan/250B/learningmeaning.pdf`. Accessed: 2015-05-18.

[Elk01] Charles Elkan. The foundations of cost-sensitive learning. In *International joint conference on artificial intelligence*, pages 973–978, 2001.

[Elm90] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 1990.

[EMM05] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with Gaussian processes. In *ICML*, 2005.

[EvHN$^+$10] Marinus JC Eijkemans, Mark van Houdenhoven, Tien Nguyen, Eric Boersma, Ewout W Steyerberg, and Geert Kazemier. Predicting the unpredictable: A new prediction model for operating room times using individual characteristics and the surgeon's estimate. *The Journal of the American Society of Anesthesiologists*, 2010.

[Fai11] Fair Isaac Corporation. Introduction to scorecard for fico model builder, 2011. Accessed: 2017-02-22.

[FEAS$^+$16] Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. Policy networks with two-stage training for dialogue systems. In *SIGDIAL*, 2016.

[FF92] Eugene F. Fama and Kenneth R. French. The cross-section of expected stock returns. *Journal of Finance 47, 427-465.*, 1992.

[FFM$^+$15] Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015.

[FN96] Batya Friedman and Helen Nissenbaum. Bias in computer systems. *ACM Transactions on Information Systems (TOIS)*, 1996.

[For14] Matt Ford. Racism and the execution chamber. 2014.

[FTS$^+$11] Susannah Fleming, Matthew Thompson, Richard Stevens, Carl Heneghan, Annette Plddemann, Ian Maconochie, Lionel Tarassenko, and David Mant. Normal ranges of heart rate and respiratory rate in children from birth to 18 years: A systematic review of observational studies. *The Lancet*, 2011.

[G$^+$94] Frederic Gruau et al. Neural network synthesis using cellular encoding and the genetic algorithm. 1994.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, volume 15, pages 315–323, 2011.

[GCGF16] Gabriel Goh, Andrew Cotter, Maya Gupta, and Michael P Friedlander. Satisfying real-world goals with dataset constraints. In *NIPS*, 2016.

[Ger01] Felix Gers. Long short-term memory in recurrent neural networks. *Unpublished PhD dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland*, 2001.

[GF15] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *JMLR*, 2015.

[GF16] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a" right to explanation". *arXiv:1606.08813*, 2016.

[GJK$^+$10] M Gašić, F Jurčíček, Simon Keizer, François Mairesse, Blaise Thomson, Kai Yu, and Steve Young. Gaussian processes for fast policy optimisation of pomdp-based dialogue managers. In *SIGDial*, 2010.

[GKT$^+$14] Milica Gašic, Dongho Kim, Pirros Tsiakoulis, Catherine Breslin, Matthew Henderson, Martin Szummer, Blaise Thomson, and Steve Young. Incremental on-line adaptation of pomdp-based dialogue managers to extended domains. *Interspeech*, 2014.

[GLF$^+$09] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[GLSGFV10] Pedro J García-Laencina, José-Luis Sancho-Gómez, and Aníbal R Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 2010.

[Gor96] Geoffrey J Gordon. Chattering in SARSA(λ) - a CMU learning lab internal report. 1996.

[GPN$^+$15] Marzyeh Ghassemi, Marco AF Pimentel, Tristan Naumann, Thomas Brennan, David A Clifton, Peter Szolovits, and Mengling Feng. A multivariate timeseries modeling approach to severity of illness assessment and forecasting in ICU with sparse, heterogeneous clinical data. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Gra11] Alex Graves. Practical variational inference for neural networks. In *NIPS*, 2011.

[GS00] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*. IEEE, 2000.

[GS05] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 2005.

[GSC00] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural computation*, 2000.

[GSS03] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with LSTM recurrent networks. *JMLR*, 2003.

[GW17] James Grimmelmann and Daniel Westreich. Incomprehensible discrimination. *California Law Review*, 2017.

[Han16] Steve Hanneke. The optimal sample complexity of pac learning. *JMLR*, 2016.

[HBF01] Sepp Hochreiter, Yoshua Bengio, and Paolo Frasconi. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[HCD$^+$16] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Curiosity-driven exploration in deep reinforcement learning via Bayesian neural networks. *arXiv:1605.09674*, 2016.

[HDM$^+$11] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 2011.

[Heg94] Matthias Heger. Consideration of risk in reinforcement learning. In *Machine Learning*, 1994.

[HFA$^+$15] Nils Y Hammerla, James M Fisher, Peter Andras, Lynn Rochester, Richard Walker, and Thomas Plötz. PD disease state assessment in naturalistic environments using deep learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[HHPS15] Katharine E Henry, David N Hager, Peter J Pronovost, and Suchi Saria. A targeted real-time early warning score (trewscore) for septic shock. *Science Translational Medicine*, 2015.

[Hoc] Sepp Hochreiter. Bridging long time lags by weight guessing and long short-term memory.

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 1982.

[HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[HPS$^+$16] Moritz Hardt, Eric Price, Nati Srebro, et al. Equality of opportunity in supervised learning. In *NIPS*, 2016.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[HS13] Steven A Harp and Tariq Samad. Optimizing neural networks with genetic algorithms. In *Proceedings of the 54th American Power Conference, Chicago*, volume 2, 2013.

[HS17] Anemona Hartocollis and Stepanie Saul. Affirmative action battle has a new focus: Asian americans. 2017.

[HSSU08] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN*, 2008.

[HVC93] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Conference on Learning Theory*, 1993.

[Jan07] Martin Jansche. A maximum expected utility framework for binary sequence labeling. In *ACL*, 2007.

[Jia16] Hengjian Jia. Investigation into the effectiveness of long short term memory networks for stock price prediction. *arXiv:1603.07893*, 2016.

[JKM$^+$16] Matthew Joseph, Michael Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth. Rawlsian fairness for machine learning. *arXiv:1610.09559*, 2016.

[JKSL15] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 2015.

[JL17] James E Johndrow and Kristian Lum. An algorithm for removing sensitive information: application to race-independent recidivism prediction. *arXiv:1703.04957*, 2017.

[JOA10] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *JMLR*, 2010.

[Jor97] Michael I Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1997.

[Kak03] Sham Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, UCL, UK, 2003.

[KAS11] Toshihiro Kamishima, Shotaro Akaho, and Jun Sakuma. Fairness-aware learning through regularization approach. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*. IEEE, 2011.

[KB15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[KC09] Faisal Kamiran and Toon Calders. Classifying without discriminating. In *Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on*. IEEE, 2009.

[KC12] Faisal Kamiran and Toon Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 2012.

[KCP10] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. Discrimination aware decision tree learning. In *ICDM*. IEEE, 2010.

[KFF14] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *CVPR*, 2014.

[KGJS15] Been Kim, Elena Glassman, Brittney Johnson, and Julie Shah. ibcm: Interactive bayesian case model empowering humans via intuitive interaction. 2015.

[KHF$^+$16] Samantha Krening, Brent Harrison, Karen Feigh, Charles Isbell, Mark Riedl, and Andrea Thomaz. Learning from explanations using sentiment and advice in rl. *IEEE Transactions on Cognitive and Developmental Systems*, 2016.

[Kim15] Been Kim. *Interactive and interpretable machine learning models for human machine collaboration*. PhD thesis, Massachusetts Institute of Technology, 2015.

[Kim17] Pauline T Kim. Data-driven discrimination at work. *William & Mary Law Review*, 2017.

[KKSS15] Enis Kayış, Taghi T Khaniyev, Jaap Suermondt, and Karl Sylvester. A robust estimation model for surgery durations with temporal, operational, and surgery team effects. *Health care management science*, 2015.

[KLRS17] Matt J Kusner, Joshua R Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. *arXiv:1703.06856*, 2017.

[KMR16] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*, 2016.

[KPR$^+$17] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

[KRCP$^+$17] Niki Kilbertus, Mateo Rojas-Carulla, Giambattista Parascandolo, Moritz Hardt, Dominik Janzing, and Bernhard Schölkopf. Avoiding discrimination through causal reasoning. *arXiv:1706.02744*, 2017.

[KRS14] Been Kim, Cynthia Rudin, and Julie A Shah. The Bayesian Case Model: A generative approach for case-based reasoning and prototype classification. In *NIPS*, 2014.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, 2006.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2013.

[LBE15] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv:1506.00019*, 2015.

[LCDH$^+$90] B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1990.

[LCG12] Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *KDD*, 2012.

[LCGH13] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *KDD*, 2013.

[LDL13] Thomas A. Lasko, Joshua C. Denny, and Mia A. Levy. Computational phenotype discovery using unsupervised feature learning over noisy, sparse, and irregular clinical data. *PLoS ONE*, 2013.

[LE15] Zachary C Lipton and Charles Elkan. Efficient elastic net regularization for sparse linear models. 2015.

[Lea16a] Sergey Levine et al. End-to-end training of deep visuomotor policies. *JMLR*, 2016.

[Lea16b] Zachary C Lipton et al. Efficient exploration for dialogue policy learning with BBQ networks & replay buffer spiking. In *NIPS Workshop on Deep Reinforcement Learning*, 2016.

[LEN14] Zachary C Lipton, Charles Elkan, and Balakrishnan Naryanaswamy. Optimal thresholding of classifiers to maximize F1 measure. In *ECML*. 2014.

[Lew95] David D. Lewis. Evaluating and optimizing autonomous text classification systems. In *SIGIR*. ACM, 1995.

[LGBS07] Marcus Liwicki, Alex Graves, Horst Bunke, and Jürgen Schmidhuber. A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks. In *Proc. 9th Int. Conf. on Document Analysis and Recognition*, 2007.

[Lin92] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 1992.

[Lip16] Zachary C Lipton. The mythos of model interpretability. *ICML Workshop on Human Interpretability in Machine Learning*, 2016.

[LKEW16] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *ICLR*, 2016.

[LKW16a] Zachary C Lipton, David C Kale, and Randall Wetzel. Directly modeling missing data in sequences with rnns: Improved classification of clinical time series. *Machine Learning for Healthcare*, 2016.

[LKW16b] Zachary C Lipton, David C Kale, and Randall Wetzel. Modeling missing data in clinical time series with rnns. In *Machine Learning for Healthcare*, 2016.

[LLWS11] Lihong Li, Michael L. Littman, Thomas J. Walsh, and Alexander L. Strehl. Knows what it knows: A framework for self-aware learning. *Machine Learning*, 2011.

[LLZ09] John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. In *NIPS*, 2009.

[LPB16] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv:1612.01474*, 2016.

[LPE97] Esther Levin, Roberto Pieraccini, and Wieland Eckert. Learning dialogue strategies within the Markov decision process framework. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, 1997.

[LR$^+$14] I Liu, Bhiksha Ramakrishnan, et al. Bach in 2014: Music composition with recurrent neural network. *arXiv:1412.3191*, 2014.

[LRS05] Changchun Liu, Pramila Rani, and Nilanjan Sarkar. An empirical study of machine learning techniques for affect recognition in human-robot interaction. In *International Conference on Intelligent Robots and Systems*. IEEE, 2005.

[LSC05] Quoc V Le, Alex J Smola, and Stéphane Canu. Heteroscedastic gaussian process regression. In *ICML*, 2005.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[LVM15] Zachary C Lipton, Sharad Vikram, and Julian McAuley. Capturing meaning in product reviews with character-level generative text models. *arXiv:1511.03683*, 2015.

[LXG$^+$14] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *AISTATS*, 2014.

[MA12] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *ICML*, 2012.

[MBM$^+$10] Robert J. Mason, V. Courtney Broaddus, Thomas Martin, Talmadge E. King Jr., Dean Schraufnagel, John F. Murray, and Jay A. Nadel. *Murray and Nadel's textbook of respiratory medicine: 2-volume set*. Elsevier Health Sciences, 2010.

[MC89] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 1989.

[MDC$^+$01] Michael C. Mozer, Robert H Dodier, Michael D. Colagrosso, César Guerra-Salcedo, and Richard H Wolniewicz. Prodding the ROC curve: Constrained optimization of classifier performance. In *NIPS*, 2001.

[MJV$^+$12] A. Menon, X. Jiang, S. Vembu, C. Elkan, and L. Ohno-Machado. Predicting accurate probabilities with a ranking loss. 2012.

[MKE05] Rasmus Madsen, David Kauchak, and Charles Elkan. Modeling word burstiness using the Dirichlet distribution. 2005.

[MKKW12] Ben M. Marlin, David C. Kale, Robinder G. Khemani, and Randall C. Wetzel. Unsupervised pattern discovery in electronic health care data using probabilistic clustering models. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium (IHI)*, 2012.

[MKO$^+$03] David R. Musicant, Vipin Kumar, Aysel Ozgur, et al. Optimizing F-measure with support vector machines. In *FLAIRS Conference*, 2003.

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[ML13] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *RecSys*. ACM, 2013.

[MLO$^+$12] Andrew L Maas, Quoc V Le, Tyler M O'Neil, Oriol Vinyals, Patrick Nguyen, and Andrew Y Ng. Recurrent neural networks for noise reduction in robust ASR. In *INTERSPEECH*. Citeseer, 2012.

[MMO95] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 1995.

[MO05] Makoto Murata and Seiichi Ozawa. A memory-based reinforcement learning model utilizing macro-actions. In *Adaptive and Natural Computing Algorithms*. 2005.

[MOT15] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 2015.

[Mou14] Yascha Mounk. Is Harvard unfair to asian-americans?, 2014.

[MRS08] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[MS11] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *ICML*, 2011.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[MV15] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

[MXY$^+$15] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *ICLR*, 2015.

[Nat04] National High Blood Pressure Education Program Working Group on Children and Adolescents. The fourth report on the diagnosis, evaluation, and treatment of high blood pressure in children and adolescents. *Pediatrics*, 2004.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[NHV$^+$15] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *arXiv:1503.08909*, 2015.

[Nig16] Will Night. The AI that cut googles energy bill could soon help you. *MIT Tech Review*, 2016.

[NS17] Razieh Nabi and Ilya Shpitser. Fair inference on outcomes. *arXiv:1705.10378*, 2017.

[NW94] David A Nix and Andreas S Weigend. Estimating the mean and variance of the target probability distribution. In *International Conference on Neural Networks*. IEEE, 1994.

[OBPVR16] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. *arXiv:1602.04621*, 2016.

[OCG$^+$15] Anika Oellrich, Nigel Collier, Tudor Groza, Dietrich Rebholz-Schuhmann, Nigam Shah, Olivier Bodenreider, Mary Regina Boland, Ivo Georgiev, Hongfang Liu, Kevin Livingston, Augustin Luna, Ann-Marie Mallon, Prashanti Manda, Peter N. Robinson, Gabriella Rustici, Michelle Simon, Liqin Wang, Rainer Winnenburg, and Michel Dumontier. The digital revolution in phenotyping. *Briefings in Bioinformatics*, 2015.

[ORR13] Ian Osband, Dan Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. In *NIPS*, 2013.

[PBKL14] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, 2014.

[Pea09] Judea Pearl. *Causality*. Cambridge university press, 2009.

[PG01] Shahla Parveen and P Green. Speech recognition with missing data using recurrent neural nets. In *NIPS*, 2001.

[Pig01] Therese D Pigott. A review of methods for missing data. *Educational research and evaluation*, 2001.

[PL17] Eero Ptri and Timo Leivo. A closer look at the value premium. *Journal of Economic Surveys, Vol. 31, Issue 1, pp. 79-168, 2017*, 2017.

[PMB12] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv:1211.5063*, 2012.

[PPR96] M. M. Pollack, K. M. Patel, and U. E. Ruttimann. *PRISM III: an updated Pediatric Risk of Mortality score. Critical Care Medicine*, 1996.

[PPRB02] Gianluca Pollastri, Darisz Przybylski, Burkhard Rost, and Pierre Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 2002.

[PRT08] Dino Pedreshi, Salvatore Ruggieri, and Franco Turini. Discrimination-aware data mining. In *KDD*. ACM, 2008.

[Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[QWM$^+$09] John Quinn, Christopher KI Williams, Neil McIntosh, et al. Factorial switching linear dynamical systems applied to physiological condition monitoring. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[RDL+10] Anand I. Rughani, Travis M. Dumont, Zhenyu Lu, Josh Bongard, Michael A. Horgan, Paul L. Penar, and Bruce I Tranmer. Use of an artificial neural network to predict head injury outcome: clinical article. *Journal of Neurosurgery*, 2010.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[RMRO98] Greg Ridgeway, David Madigan, Thomas Richardson, and John O'Kane. Interpretable boosted naïve bayes classification. In *KDD*, 1998.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. *KDD*, 2016.

[RSZ17] Ya'acov Ritov, Yuekai Sun, and Ruofei Zhao. On conditional parity as a notion of non-discrimination in machine learning. *arXiv:1706.08519*, 2017.

[SB98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[Sch91] Jurgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *From animals to animats: proceedings of the first international conference on simulation of adaptive behavior (SAB90)*, 1991.

[SD09] Yoram Singer and John C Duchi. Efficient learning using forward-backward splitting. In *NIPS*, 2009.

[SDAS97] Sujit K Sahu, Dipak K Dey, Helen Aslanidou, and Debajyoti Sinha. A weibull regression model with gamma frailties for multivariate survival data. *Lifetime data analysis*, 1997.

[Sha66] William F Sharpe. Mutual fund performance. *The Journal of Business*, 1966.

[Shi80] Robert J Shiller. Do stock prices move too much to be justified by subsequent changes in dividends?, 1980.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine

Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[SK12] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT Press, 2012.

[SKL$^+$14] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2014.

[SKLW00] Satinder P Singh, Michael J Kearns, Diane J Litman, and Marilyn A Walker. Reinforcement learning for spoken dialogue systems. In *NIPS*, 2000.

[SKP10] Suchi Saria, Daphne Koller, and Anna Penn. Learning individual and population level traits from clinical temporal data. In *Proc. Neural Information Processing Systems (NIPS), Predictive Models in Personalized Medicine Workshop*, 2010.

[SL09] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 2009.

[SLA15] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv:1507.00814*, 2015.

[SM98] Rosaria Silipo and Carlo Marchesi. Artificial neural networks for automatic ecg analysis. *IEEE Transactions on Signal Processing*, 1998.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[SMH11] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[SMI06] Jun Suzuki, Erik McDermott, and Hideki Isozaki. Training conditional random fields with multivariate evaluation measures. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. ACL, 2006.

[SP97] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 1997.

[SQAS16] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICML*, 2016. arXiv:1511.05952.

[SS91] Hava T Siegelmann and Eduardo D Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 1991.

[SSSS16] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv:1610.03295*, 2016.

[Str00] Malcolm J. A. Strens. A Bayesian framework for reinforcement learning. In *ICML*, 2000.

[STY07] Jost Schatzmann, Blaise Thomson, and Steve Young. Statistical user simulation with a hidden agenda. *SIGDial*, 2007.

[Sut88] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 1988.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2013.

[SWF$^+$14a] Ioan Stanculescu, Christopher K Williams, Yvonne Freer, et al. Autoregressive hidden markov models for the early detection of neonatal sepsis. *Biomedical and Health Informatics, IEEE Journal of*, 2014.

[SWF14b] Ioan Stanculescu, Christopher KI Williams, and Yvonne Freer. A hierarchical switching linear dynamical system applied to the detection of sepsis in neonatal condition monitoring. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

[SWS15] Peter Schulam, Fredrick Wigley, and Suchi Saria. Clustering longitudinal clinical marker trajectories from electronic health data: Applications to phenotyping and endotype discovery. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[SZS$^+$13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CVPR*, 2013.

[Tan05] Songbo Tan. Neighbor-weighted k-nearest neighbor for unbalanced text corpus. *Expert Systems with Applications*, 2005.

[TB98] Volker Tresp and Thomas Briegel. A solution for missing data in recurrent neural networks with an application to blood glucose prediction. In *NIPS*. 1998.

[TH12] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5- rmsprop: Divide the gradient by a running average of its recent magnitude. `https://www.youtube.com/watch?v=LGA-gRkLEsI`, 2012.

[Tho33] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 1933.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1996.

[TLBN16] Subarna Tripathi, Zachary C Lipton, Serge Belongie, and Truong Nguyen. Context matters: Refining object detection in video with recurrent neural networks. *BMVC*, 2016.

[Tso07] Ioannis Tsoumakas, Grigorios & Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 2007.

[Tur50] Alan M Turing. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

[Übe09] Elif Derya Übeyli. Combining recurrent neural networks with eigenvector methods for classification of ecg beats. *Digital Signal Processing*, 2009.

[Vap13] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[VdMH08] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 2008.

[VHGS15] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2015.

[Voh01] Jiří Vohradský. Neural network model of gene expression. *The FASEB Journal*, 2001.

[VTBE14] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *CVPR*, 2014.

[VTBE15] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *CVPR*, 2015.

[WD92a] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 1992.

[WD92b] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 1992.

[WdFL16] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. In *ICML*, 2016.

[Wer88] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1988.

[Wer90] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990.

[WFF$^+$99] Hui-Xin Wang, Laura Fratiglioni, Giovanni B Frisoni, Matti Viitanen, and Bengt Winblad. Smoking and the occurence of alzheimer's disease: Cross-sectional and longitudinal data in a population-based study. *American journal of epidemiology*, 1999.

[WGM$^+$16] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv:1604.04562*, 2016.

[WGOS17] Blake Woodworth, Suriya Gunasekar, Mesrob I Ohannessian, and Nathan Srebro. Learning non-discriminatory predictors. *arXiv:1702.06081*, 2017.

[WHG12] Jenna Wiens, Eric Horvitz, and John V Guttag. Patient risk stratification for hospital-associated c. diff as a time-series classification task. In *Advances in neural information processing systems*, 2012.

[Wik15] Wikipedia. Backpropagation — Wikipedia, the free encyclopedia, 2015. [Online; accessed 18-May-2015].

[Wil96] Peter M Williams. Using neural networks to model conditional multivariate densities. *Neural Computation*, 1996.

[WM14] Barack Wamkaya Wanjawa and Lawrence Muchemi. Ann model to predict stock prices at stock exchange markets. *arXiv:1502.06434*, 2014.

[Wor04] World Health Organization. *International statistical classification of diseases and related health problems*. World Health Organization, 2004.

[Wor14] World Bank. Health expenditure, total (% of gdp), 2014.

[WY04] Jason D Williams and Steve J Young. Characterizing task-oriented dialog using a simulated ASR channel. In *Interspeech*, 2004.

[WY07] Jason D. Williams and Steve J. Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, 2007.

[WZ89] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1989.

[XWIF07] Rui Xu, Donald Wunsch II, and Ronald Frank. Inference of genetic regulatory networks with recurrent neural network models using particle swarm optimization. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 2007.

[YCLC12] Nan Ye, Kian Ming Chai, Wee Sun Lee, and Hai Leong Chieu. Optimizing F-measures: A tale of two approaches. In *ICML*, 2012.

[YRJ$^+$15] Serena Yeung, Olga Russakovsky, Ning Jin, Mykhaylo Andriluka, Greg Mori, and Li Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. *arXiv:1507.05738*, 2015.

[Zei12] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv:1212.5701*, 2012.

[ZEPB13] Ming-Jie Zhao, Narayanan Edakunni, Adam Pocock, and Gavin Brown. Beyond Fano's inequality: Bounds on the optimal F-score, BER, and cost-sensitive risk and their implications. *JMLR*, 2013.

[ZRM$^+$13] Matthew D Zeiler, M Ranzato, Rajat Monga, M Mao, K Yang, Quoc Viet Le, Patrick Nguyen, A Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3517–3521. IEEE, 2013.

[ZS14] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv:1410.4615*, 2014.

[ZSV14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

[ZVGRG17] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. Fairness constraints: Mechanisms for fair classification. In *AISTATS*, 2017.

[ZVR$^+$17] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, Krishna P Gummadi, and Adrian Weller. From parity to preference-based notions of fairness in classification. *NIPS*, 2017.

[ZWS$^+$13] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *ICML*, 2013.