Towards Training AI Agents with All Types of Experiences: A Standardized ML Formalism

Zhiting Hu

JANUARY 2021
CMU-ML-21-102

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

Thesis Committee:
Eric P. Xing, Chair
Tom Mitchell
Ruslan Salakhutdinov
Dan Roth (University of Pennsylvania)

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

This research was sponsored by: the National Science Foundation awards IIS1218282, IIS1447676 and IIS1563887; the Defense Advanced Research Projects Agency awards FA8721-05-C-0003 and FA8721-05-C-0003; and a fellowship from IBM Corporation.

Keywords: machine learning, learning with experiences, standard equation, unification, machine learning compositionality

To the peace of the world

Abstract

Machine Learning (ML) is about computational methods that enable machines to learn concepts from experiences. In handling a wide range of experiences ranging from data instances, knowledge, constraints, to rewards, adversaries, and lifelong interplay in an ever-growing spectrum of tasks, contemporary ML/AI research has resulted in a large multitude of learning paradigms (e.g., supervised, unsupervised, active, reinforcement, adversarial learning), models, optimization techniques, not mentioning countless approximation heuristics and tuning tricks, plus combinations of all above. While pushing the field forward rapidly, these results also make mastering existing ML techniques costly, and make it difficult, if possible at all, to build AI agents that are capable of learning from all types of experiences and thus are reusable, repeatable, reliable, and explainable in ML/AI applications and productions.

In this dissertation, I present a standardized mathematical formalism of machine learning that offers a principled framework for understanding, unifying, and generalizing current major paradigms of learning algorithms, and for designing new families of algorithmic solutions and applications for learning with all experiences, in a composable and mechanic manner.

The dissertation consists of four parts, where we study and apply the standardized ML formalism from theoretical, methodological, applicational, and operational aspects. In Part I, we establish the simple yet general formalism, materialized as a standard equation of the objective function which characterizes experience, divergence, and uncertainty in a learning system. The standard equation provides a succinct, structured formulation of a vast design space of learning algorithms, and is justified as we show that a wide range of well-known algorithms with varying losses, constraints, and forms of experiences, all fall under its umbrella. In Part II, we show the formalism is a natural framework for making use of arbitrary available experiences to learn models of interest. On this basis, we develop new mechanisms of learning that go beyond reliance on data instances, and train models (e.g., deep neural networks) by integrating declarative logical rules, as well as rich auxiliary models from relevant tasks. The studies also yield a new set of applications for controllable text generation. In Part III, we show the unified formalism opens up a wide range of opportunities for extending originally specialized algorithms to solve new problems. In particular, we show how a set of seemingly unrelated problems, including training with fuzzy knowledge, automated data augmentation, and stabilizing GAN training, are essentially the same problem within the standardized framework, corresponding to joint model-experience co-learning, and can all be addressed by simply repurposing existing algorithms in the fertile research area of reinforcement learning. In Part IV, we further operationalize the standardized framework by developing a composable ML toolkit, Texar, that allows users to quickly assemble ML solutions to their problems by putting together standard and reusable building blocks.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my amazing PhD advisor, Eric P. Xing, for his continuous support of my graduate study. Eric's immense knowledge, insights, passion, and patience have influenced me a lot. I would also like to thank him for providing me the freedom and resources to pursue various directions.

I am grateful to have had the opportunity to work with Rulsan Salakhutdinov closely on many of the projects. His scientific advice, deep knowledge, and insightful discussions have helped a lot to achieve the fruitful outcomes. I am also extremely grateful to Tom Mitchell and Dan Roth. As my committee members, they have provided valuable guidance to improve my thesis and advice and support to my career. I would also like to thank my undergraduate advisor, Bin Cui, for the guidance and inspirations that led me to a career of research.

I have been fortunate enough to get support and suggestions from many collaborators and friends. I have had lots of collaborations and discussions with Andrew Wilson since he was postdoc at CMU. Le Song has provided me with great advice on my research career. I also deeply appreciate the guidance from my other senior mentors and colleagues, especially Mrinmaya Sachan, Diyi Yang, Qirong Ho, Avinava Dubey, Ming Zhang, Zaiqing Nie, Junjie Yao, Hongzhi Yin, Bishan Yang, Igor Labutov, Lei Li, Chong Wang, and others.

I had great pleasure of collaborating with an amazing set of researchers and colleagues throughout my PhD: Zichao Yang, Xiaodan Liang, Bowen Tan, Maruan Al-Shedivat, Yue Wu, Pan Zhou, Andrew Gordon Wilson, Zhengzhong Liu, Haoran Shi, Shuai Lin, Tiancheng Zhao, Junxian He, Lianhui Qin, Di Wang, Xuezhe Ma, Wangrong Zhu, Devendra Singh Sachan, Jianheng Tang, Chenyan Xiong, Haoye Dong, Christy Y. Li, Hao Zhang, Luchen Liu, Jian Tang, Taylor Berg-Kirkpatrick, Pengtao Xie, Prasoon Goyal, Chenyu Wang, Hai Zhao, Yaoliang Yu, Yuntian Deng, Eduard Hovy, Jinliang Wei, Gang Luo, Yuezhang Li, Zhuo Wang, Poyao Huang, and Yingkai Gao.

My heartfelt thanks to fellow labmates at the CMU Sailing Lab, including Maruan Al-Shedivat, Bryon Aragam, Wei Dai, Avinava Dubey, Han Guo, Jin Kyu Kim, Lisa Lee, Ben Lengerich, Xiaodan Liang, Willie Neiswanger, Aurick Qiao, Mrinmaya Sachan, Bowen Tan, Haohan Wang, Jinliang Wei, Andrew Wilson, Pengtao Xie, Yaoliang Yu, Hao Zhang, Xun Zheng, and Helen Zhou.

Last but not the least, this thesis is dedicated to my parents, my brother, and my love, for their unconditional support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human learning has the hallmark of learning concepts from diverse sources of information. Take the example of learning a language. Humans can benefit from a wide variety of experiences whichever are available—by observing examples through reading and hearing, studying abstract definitions and grammar, making mistakes and getting correction from teacher, interacting with others and observing implicit feedback, etc. Knowledge of prior language can also accelerate the acquisition of new one. How can we build AI agents capable of learning versatilely from such diverse types of experiences?

The past decades of machine learning (ML) research has resulted in a multitude of learning paradigms. Each of these paradigms makes particular assumptions on the form of experiences available. For example, the present most popular supervised learning relies on collections of data instances; active learning (Settles, 2012) manages data instances which, instead of being given all at once, are adaptively selected; reinforcement learning (Sutton and Barto, 2017) makes use of feedbacks obtained via interaction with the environment; knowledge-constrained learning (Chang et al., 2007; Ganchev et al., 2010) learns from structured knowledge expressed as constraints; and generative adversarial learning (Goodfellow et al., 2014) leverages a companion model called discriminator to guide training of the model of interest. The vastly different assumptions on experiences, plus their varying mathematical formulations and the many design choices in implementation, contribute to making ML more like an alchemist's crafting workshop rather than a modern chemist's periodic table. It becomes increasingly difficult to navigate through the bewildering marketplace of ML techniques. Each paradigm of algorithms are best suited for narrow contexts, which, though producing impressive results on specific tasks, fall short of reusability and repeatability to apply on broad new problems, and generality and composability to make use of any useful information signals in different forms, at varying granularities, from diverse sources, and with different intents (e.g., adversaries).

To create intelligent agents that can ingest and improve from arbitrary available experiences, we argue that it is perhaps indispensable to develop a standardized, instead of the current siloed, formalism of machine learning. This could be inspired by taking a look at another scientific discipline, Physics, which has a long history of chasing symmetry and simplicity of its principles.

"You do not need something more in order to get something more." as the physicist Murray Gell-Mann (2007) remarked about how Physics describes the world—exemplified by the famed Maxwell's equations in 1800s which reduced various principles of electricity and magnetism into a single electromagnetic theory, followed by General Relativity in 1910s and the Standard Model in 1970s, physicists describe new phenomena best by reducing and unifying existing theories into a standardized one. Likewise, in the field of machine learning, how can we establish a "Standard Model" that gives a holistic view of the broad learning principles, lays out a blueprint permitting fuller and more systematic exploration in the design and analysis of new algorithms, and eventually serves as a vehicle towards building AI agents that integrate all available sources of experiences?

This dissertation aims to answer the question with multiple steps: (1) we first establish the theoretical foundations by presenting a standardized mathematical formalism of machine learning; (2) we then show how the formalism can derive, in a systematic and mechanic manner, new families of algorithms that can learn from novel forms of experiences and their combinations, and solutions to a broad set of challenging problems involving imperfect and adaptive experiences; (3) we finally deliver an ML toolkit that operationalizes the standardized framework and supports development of ML solutions in a composable, reusable, and repeatable way.

The standardized ML formalism is materialized as a standard equation of the *objective function* that drives the learning of ML models with any given experiences **(Part I)**. A key ingredient of the formalism is an experience function that measures the goodness of a configuration of the target variables in light of the given experiences. The experience function provides a unified language to express arbitrary information about the unknown target model. On this basis, and building upon the maximum entropy and variational principles, the standard equation consists of three principled terms, including the *experience* term (exogenous regularization) for the experience functions to be plugged into the learning system, the *divergence* (fitness) that introduces a teacher-student mechanism of training the target model, and the *uncertainty* (self-regularization) for minimum complexity.

The standard equation formulates a rather broad design space of learning algorithms. We show that a wide range of well-known algorithms that originate in disparate ML paradigms, ranging from the simplest supervised/unsupervised maximum likelihood estimation, to the knowledge-based posterior regularization, reinforcement learning with extrinsic and intrinsic rewards, generative adversarial networks, and their extensions and variants, are all subsumed as special instances due to different choices of the experience and divergence components. The formalism thus shed new lights on fundamental relationships between the diverse algorithms that were each originally designed to deal with a particular type of experience.

The simplicity, modularity and generality of the framework is particularly appealing not only from the theoretical perspective, but also because it offers guiding principles for designing algorithmic solutions to problems in a mechanic way. Specifically, the standard equation is naturally designed to allow combining together all different experiences, such as data, knowledge, constraints, rewards, and auxiliary models, to learn models of interest **(Part II)**. Designing a solution to a problem thus boils down to choosing *what* experiences to use depending on the problem structure and available resources, without worrying too much about *how* to use the experiences in the training. In particular, the framework enables us to develop algorithms that train deep neural networks with first-order logic rules to incorporate domain knowledge and convey human goals (Hu et al., 2016a,b). As another example, we are also able to develop solutions for using rich auxiliary models from related tasks to train models of interest, without need of any direct data supervisions. When applied to the domain of natural language modeling, the algorithms produce a new set of models for controllable text generation, with wide applications in personalized chatbots, article stylization, and offensive language neutralization, etc (Hu et al., 2017a; Yang et al., 2018; Tang et al., 2019).

The standardized ML perspective also highlights that many of the learning problems in seemingly distinct contexts are essentially the same and just correspond to specialized variations of the standard equation components. This opens up a wide range of opportunities for extending the use of existing algorithms on broad sets of new problems, and for exchange between the diverse research areas in aspects of modeling, approximation and optimization tools, and theoretical understanding **(Part III)**. Thus, an earlier successful solution to challenges in one problem can now be readily applied to address challenges in another, and a future progress made in one area could potentially immediately unlock progresses in many others. With the guidance in hand, we consider a set of challenging problems that were previously seen as unrelated and often studied by researchers in different domains. The first problem is about integrating structured knowledge constraints with deep generative models, where some components of the knowledge, and/or the confidence weight of each of the knowledge constraints, cannot be fully specified *a priori*. The second problem is in the supervised learning regime, where only a small set of data instances with imbalanced labels is available, and we want to automatically manipulate the data (e.g., through augmentation and re-weighting) to maximize the training performance. The last problem is to stabilize the notoriously difficult training of generative adversarial networks (GANs) for a wide range of image and text generation tasks. We show that these problems, though concerning with distinctly different experiences, can all be reduced to the same problem within the standardized formalism, where the experience functions in the standard equation are imperfect and need to be adapted jointly with the target models (Hu et al., 2018a, 2019b; Wu et al., 2020). Instead of having to invent new algorithms to address the general problem, we take advantage of the connections between research areas, and import known algorithms from the rich reinforcement learning (RL) literature, such as Inverse RL and Proximal Policy Optimization, as effective solutions.

Drawing on the standardized view of ML, we make a step further and develop ML tools, Texar (Hu et al., 2019a) and ASYML (2019), that operationalize the above theoretical framework and allow practitioners to develop ML (esp., natural language processing) applications handling different experiences in a composable manner **(Part IV)**. Guided by the principled abstraction of ML modules, the toolkit provides a large repository of standardized, reusable, and re-purposable building blocks of learning algorithms and model architectures. Users can quickly compose solutions to their own problems at a high conceptual level without concerning low-level details, and easily switch between different algorithmic paradigms by plugging in or swapping out relevant modules without touching other parts. The toolkit is open-sourced to the community for fast prototyping, experimentation, production, fostering reproducibility, and inspiring technique sharing among different problems and/or research areas.

1.1 Overview of Dissertation

The dissertation consists of four parts as follows.

Theory: A Standardized Formalism of Machine Learning. This part establishes the theoretical foundation of the dissertation by developing the standard equation of objective function.

Chapter 2 presents the details of each of the components in the general mathematical formulation, setting forth a vast and structured design space of learning algorithms dealing with all different types of experiences. A broad range of existing algorithms in different paradigms (e.g., supervised, unsupervised, active, reinforcement, adversarial, knowledge-constrained learning) turn out to involve particular choices of the constituent components. We then give a brief overview of the practical implications of the standardized formalism on designing new algorithms and problem solutions. A subset of this chapter is based on the materials from (Hu et al., 2016a, 2018a; Tan et al., 2018; Hu et al., 2018b).

Application (1): Learning with All Experiences. This part discusses how the standardized formalism can guide us to build algorithmic solutions that learn from diverse experiences and their combinations, highlighting that with the framework one can plug in any experiences in a mechanic way.

In Chapter 3, we develop a principled approach to integrating structured knowledge such as logical rules in training arbitrary deep neural networks, a constant challenge in the ML/AI field. This method was previously published as (Hu et al., 2016a).

In Chapter 4, we consider learning from another useful type of experience, namely rich auxiliary models from related tasks, without any supervised data. The resulting method in turn yields a range of new applications that generate natural language text with control over text attributes, content, and discourse structures. This chapter is primarily based on (Hu et al., 2017a; Yang et al., 2018) and also includes materials from (Tang et al., 2019; Lin et al., 2020).

Application (2): Repurposing Algorithms for New Problems. This part is devoted to the second set of guiding principles by the standardized formalism, on how we can solve new problems with challenging experiences by repurposing existing algorithms based on the connections between research areas, without having to reinvent the wheels.

In Chapter 6, we study an extended problem of Chapter 3, where the structured knowledge itself as the experience is sub-optimal or has uncertain components and thus needs to be adapted along with the target model. By mapping the problem to the standard equation framework, we show the problem can readily be solved by repurposing a well-known inverse reinforcement learning algorithm. This chapter was previously published as (Hu et al., 2018a).

In Chapter 5, we consider the challenging problem of automated data manipulation, such as augmentation and reweighting, for combating low-data and label-imbalance issues in supervised training. We show that the different manipulation schemes can all be reduced to the same problem of learning a parameterized data-reward experience function, leading to an elegant unified solution imported from the reward learning literature in reinforcement learning. This chapter appeared previously as (Hu et al., 2019b).

In Chapter 7, we study the standard equation interpretation of GANs and develop a new stabilized training approach based on its connection with the popular Proximal Policy Optimization for stable policy updates. The chapter was previously published as (Wu et al., 2020).

Tooling: Operationalizing Standardized Composable ML. In the final part, we develop an open-source toolkit that implement the standardized view of ML and allow users to compose ML solutions mechanically.

Chapter 8 describes the core design of the toolkit and introduces a large repository of standardized composable building blocks. The chapter is primarily based on (Hu et al., 2019a) and also includes materials from (Liu et al., 2020).

Part I

Theory: A Standardized Formalism of Machine Learning

Chapter 2

Standard Equation

We begin by establishing the standardized ML formalism that provides the theoretical foundations for the chapters that follow. Building an ML solution typically involves three key ingredients, namely the model of certain architecture whose parameters are the subject to be learned, the objective function that defines the learning problem, and the optimization solver that solves the problem and returns the desired parameter configuration. In dealing with the wide range of experiences in an ever-growing spectrum of tasks, the past research has resulted in countless choices for each of the ingredients and the combinations thereof. To build a unifying standard equation that offers a clean and simple blueprint for ML algorithm and solution building, a good place to begin with is perhaps the objective function as it lies at the core of most learning algorithms and is often the vehicle for people to define new algorithms, understand learning properties, and validate outcomes.

In this chapter, we show it is possible to build a standard equation as a generic expression of the objective function that will emerge in all different learning paradigms handling diverse experiences. At a high level, the standard equation consists of three components, including an experience term that encodes arbitrary information about the unknown target model, a divergence term that measures the fitness of the model, and an uncertainty term that encourages low-complexity solution. Optimizing the objective entails a teacher-student mechanism where an auxiliary variational distribution serves as the “teacher” that absorbs information from the experience and the target model as the “student” learns to match the teacher by minimizing the divergence.

In the following sections, we first review a series of earlier ML frameworks from the first principle perspective, which highlights the central role of the maximum entropy and variational principles (section 2.1). On this basis, we present the standard equation and discuss each of the components in details, showing that the mathematical formulation defines a vast space of learning algorithms, and that a number of known algorithms all fit within this framework (sections 2.2-2.4). We then give a brief overview of how the framework can guide systematic design of new algorithmic solutions to dealing with diverse experiences in different problems. The chapter ends with a discussion of limitations and future directions on the theory of standardized formalism.

2.1 Preliminaries: Maximum Entropy and Maximum Likelihood

2.1.1 MLE

2.1.1.1 Supervised MLE

Without loss of generality, denote the model to be learned as $p_\theta(\boldsymbol{x})$, where $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is the model parameters. The most common method for estimating the parameters $\boldsymbol{\theta}$ is perhaps maximum likelihood estimation (MLE). We are given a set of fully-observed data examples $\mathcal{D} = \{\boldsymbol{x}^*\}$. MLE learns the model by minimizing the negative log-likelihood:

$$\min_{\boldsymbol{\theta}} -\mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}} \left[\log p_\theta(\boldsymbol{x}^*) \right]. \tag{2.1}$$

MLE is known to be intimately related to the principle of *maximum entropy* (Jaynes, 1957). In particular, when the model $p_\theta(\boldsymbol{x})$ is in the *exponential family* of the form:

$$p_\theta(\boldsymbol{x}) = \exp\left\{ \boldsymbol{\theta} \cdot T(\boldsymbol{x}) \right\} / Z(\boldsymbol{\theta}), \tag{2.2}$$

where $T(\boldsymbol{x})$ is the features of data $\boldsymbol{x}$ and $Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \exp\{\boldsymbol{\theta} \cdot T(\boldsymbol{x})\}$ is the normalization factor, it is shown that MLE is the convex dual of maximum entropy estimation.

In the maximum entropy formulation, rather than assuming a specific parametric from of the target model distribution, denoted as $p(\boldsymbol{x})$, we instead impose constraints on the model distribution, that require the expectation of the features $T(\boldsymbol{x})$ to be equal to the empirical expectation:

$$\mathbb{E}_p \left[T(\boldsymbol{x}) \right] = \mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}} \left[T(\boldsymbol{x}^*) \right]. \tag{2.3}$$

Let $\mathcal{P}(\mathcal{X})$ denote the set of all probability distributions on $\mathcal{X}$. In general, there exist many distributions $p \in \mathcal{P}(\mathcal{X})$ that satisfy the constraint. The principle of maximum entropy resolves the ambiguity by choosing the distribution such that its Shannon entropy, $\mathrm{H}(p) := -\mathbb{E}_p[\log p(\boldsymbol{x})]$, is maximized. We thus have the constrained optimization problem:

$$\begin{aligned} \min_{p(\boldsymbol{x})} \ & \mathrm{H}\left(p(\boldsymbol{x})\right) \\ s.t. \ & \mathbb{E}_p \left[T(\boldsymbol{x}) \right] = \mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}} \left[T(\boldsymbol{x}^*) \right] \\ & p(\boldsymbol{x}) \in \mathcal{P}(\mathcal{X}). \end{aligned} \tag{2.4}$$

The problem can be solved with the Lagrangian method. Specifically, we write the Lagrangian:

$$\mathcal{L}(p, \boldsymbol{\theta}, \mu) = \mathrm{H}(p(\boldsymbol{x})) - \boldsymbol{\theta} \left(\mathbb{E}_p \left[T(\boldsymbol{x}) \right] - \mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}} \left[T(\boldsymbol{x}^*) \right] \right) - \mu \left(\sum\nolimits_{\boldsymbol{x}} p(\boldsymbol{x}) - 1 \right), \tag{2.5}$$

where $\boldsymbol{\theta}$ and μ are Lagrangian multipliers. Minimizing w.r.t p leads to the optimal solution which turns out to be in the same form of Eq.(2.6):

$$p(\boldsymbol{x}) = \exp\left\{ \boldsymbol{\theta} \cdot T(\boldsymbol{x}) \right\} / Z(\boldsymbol{\theta}), \tag{2.6}$$

where we see the parameters $\boldsymbol{\theta}$ in the exponential family parameterization are the Lagrangian multipliers that enforce the constraints. Plugging the solution back into the Lagrangian, we obtain:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}} \left[\boldsymbol{\theta} \cdot T(\boldsymbol{x}^*) \right] - \log Z(\boldsymbol{\theta}), \tag{2.7}$$

which is simply the negative of the MLE objective in Eq.(2.1).

Thus maximum entropy is dual to maximum likelihood. It provides an alternative view of the problem of fitting a model into data, where the data instances in the training set are treated as constraints, and the learning problem is treated as a constrained optimization problem.

Iterative Proportional Fitting (IPF) Besides the new points of view of the MLE problem, the maximum entropy formulation is also helpful to derive efficient solvers for the problem, as we discuss in the rest of the section. As an example, consider the MLE problem of undirected graphical models (Jordan, 2003) whose distribution is generally expressed as:

$$p_\theta(\boldsymbol{x}) = \prod\nolimits_{c \in \mathcal{C}} \exp\{\psi_c(\boldsymbol{x}_c)\}/Z, \tag{2.8}$$

where $\mathcal{C}$ is a set of cliques in the graph; $\boldsymbol{x}_c$ is the variables associated with the clique $c \in \mathcal{C}$; $\exp\{\psi_c(\boldsymbol{x}_c)\}$ is a clique potential for the clique c; $\boldsymbol{\theta} = \{\psi_c(\boldsymbol{x}_c) : c \in \mathcal{C}\}$ is the collection of parameters to be learned; and $Z = \sum_{\boldsymbol{x}} \prod_c \exp\{\psi_c(\boldsymbol{x}_c)\}$ is the normalization factor.

To find the maximum likelihood estimates, direct gradient descent on the negative log-likelihood (Eq.2.1) is inefficient due to the cumbersome $\log Z$ term. Iterative Proportional Fitting (IPF, Deming and Stephan, 1940) is a generic solver for the estimation problem, which can readily be derived from the maximum entropy formulation. Specifically, let N be the size of the observed dataset $\mathcal{D}$, $m(\boldsymbol{x})$ the number of times that configuration $\boldsymbol{x}$ is observed in $\mathcal{D}$, and $m(\boldsymbol{x}_c) = \sum_{x_{\setminus c}} m(\boldsymbol{x})$ the marginal count for clique c by summing (or integrating in the continuous case) over all configurations of variables $\boldsymbol{x}_{\setminus c}$ not included in the clique c. Let $\tilde{p}_d(\boldsymbol{x}_c) = m(\boldsymbol{x}_c)/N$ be the empirical marginal. The dual maximum entropy formulation of the MLE is then written as:

$$\begin{aligned} \min_{p(\boldsymbol{x})} \;\; & \mathrm{H}\left(p(\boldsymbol{x})\right) \\ s.t. \;\; & p(\boldsymbol{x}_c) = \tilde{p}_d(\boldsymbol{x}_c), \; \forall c \in \mathcal{C} \\ & p(\boldsymbol{x}) \in \mathcal{P}(\mathcal{X}), \end{aligned} \tag{2.9}$$

where the constraints are that the model marginals $p(\boldsymbol{x}_c)$ must be equal to the empirical marginals $\tilde{p}_d(\boldsymbol{x}_c)$ for each clique c. It is worth noting that the marginal constraint is a special case of the expected feature constraint in Eq.(2.3), by using features that are indicator functions. That is, for a configuration $\boldsymbol{x}_c^*$, the marginal probability can be viewed as the expectation $p(\boldsymbol{x}_c^*) = \mathbb{E}_{p(\boldsymbol{x})}[\mathbb{I}_{x_c^*}(\boldsymbol{x}_c)]$, where the feature $\mathbb{I}_{x_c^*}(\boldsymbol{x}_c)$ is an indicator function that equals 1 when $\boldsymbol{x}_c = \boldsymbol{x}_c^*$ and 0 otherwise.

Again, we solve the problem by introducing the Lagrangian multipliers $\psi_c(\boldsymbol{x}_c)$ to impose the marginal constraints and μ for the normalization constraint, resulting in the Lagrangian:

$$\mathcal{L}(p, \{\psi_c(\boldsymbol{x}_c)\}, \mu) = \mathrm{H}(p(\boldsymbol{x})) - \sum_{c,\boldsymbol{x}_c} \psi_c(\boldsymbol{x}_c)\Big(p(\boldsymbol{x}_c) - \tilde{p}_d(\boldsymbol{x}_c)\Big) - \mu\left(\sum_{\boldsymbol{x}} p(\boldsymbol{x}) - 1\right). \quad (2.10)$$

Solving the Lagrangian gives the following update w.r.t $\psi_c(\boldsymbol{x}_c)$ (see Teh and Welling (2003) for derivations): at each iteration n,

$$\psi_c^{(n+1)}(\boldsymbol{x}_c) = \psi_c^{(n)}(\boldsymbol{x}_c) + \log \frac{\tilde{p}_d(\boldsymbol{x}_c)}{p^{(n)}(\boldsymbol{x}_c)}. \quad (2.11)$$

which is the classical form of IPF updates if we see the correspondence between the Lagrangian multipliers $\psi_c(\boldsymbol{x}_c)$ and the clique potentials. The update is equivalent to updating the primal variables $p(\boldsymbol{x})$ with:

$$p^{(n+1)}(\boldsymbol{x}) = p^{(n)}(\boldsymbol{x}) \frac{\tilde{p}_d(\boldsymbol{x}_c)}{p^{(n)}(\boldsymbol{x}_c)}. \quad (2.12)$$

The IPF procedure can be seen as iterating over the cliques in $\mathcal{C}$ and for each clique c setting the marginal $p(\boldsymbol{x}_c)$ to be $\tilde{p}_d(\boldsymbol{x}_c)$.

2.1.1.2 Unsupervised MLE

We now consider learning with latent variables, where each data instance is partitioned into observed variables $\boldsymbol{x}$ and latent variables $\boldsymbol{y}$. The goal is to learn a model $p_\theta(\boldsymbol{x}, \boldsymbol{y})$ that captures the joint distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$. Since $\boldsymbol{y}$ is unobserved, we minimize the negative log-likelihood with $\boldsymbol{y}$ marginalized out:

$$\min_{\boldsymbol{\theta}} -\mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}} \left[\log \sum_{\boldsymbol{y}} p_\theta(\boldsymbol{x}^*, \boldsymbol{y})\right]. \quad (2.13)$$

Direct optimization of the marginal log-likelihood is typically intractable due to the integration over $\boldsymbol{y}$. Earlier work thus developed different solvers with varying levels of approximations.

Expectation Maximization (EM) The most common approach to the problem is the expectation-maximization (EM) algorithm (Dempster et al., 1977). Neal and Hinton (1998) interpreted the algorithm as minimizing the variational free energy which is an upper bound of the negative marginal log-likelihood. Formally, we introduce an auxiliary distribution $q(\boldsymbol{y}|\boldsymbol{x})$, known as the variational distribution, which is an arbitrary distribution over $\boldsymbol{y}$. Then, for each instance $\boldsymbol{x}^* \in \mathcal{D}$,

$$\begin{aligned} -\log \sum_{\boldsymbol{y}} p_\theta(\boldsymbol{x}^*, \boldsymbol{y}) &= -\mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x}^*)}\left[\log \frac{p_\theta(\boldsymbol{x}^*, \boldsymbol{y})}{q(\boldsymbol{y}|\boldsymbol{x}^*)}\right] - \mathrm{KL}\left(q(\boldsymbol{y}|\boldsymbol{x}^*) \| p_\theta(\boldsymbol{y}|\boldsymbol{x}^*)\right) \\ &\le -\mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x}^*)}\left[\log \frac{p_\theta(\boldsymbol{x}^*, \boldsymbol{y})}{q(\boldsymbol{y}|\boldsymbol{x}^*)}\right] \\ &= -\mathrm{H}\left(q(\boldsymbol{y}|\boldsymbol{x}^*)\right) - \mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x}^*)}\left[\log p_\theta(\boldsymbol{x}^*, \boldsymbol{y})\right] := \mathcal{L}(q, \boldsymbol{\theta}), \end{aligned} \quad (2.14)$$

where the inequality holds because KL divergence is always non-negative. The expression of the variational free energy shows the algorithm entails maximizing the entropy of the variational distribution. By taking into account the empirical distribution $\tilde{p}_d$ from which the instance $\boldsymbol{x}^*$ is drawn, the second term, now written as $-\mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x}^*)\tilde{p}_d(\boldsymbol{x}^*)}\left[\log p_\theta(\boldsymbol{x}^*, \boldsymbol{y})\right]$, is the *cross entropy* between the distributions $q(\boldsymbol{y}|\boldsymbol{x}^*)\tilde{p}_d(\boldsymbol{x}^*)$ and $p_\theta(\boldsymbol{x}^*, \boldsymbol{y})$.

By introducing the auxiliary $q(\boldsymbol{y}|\boldsymbol{x})$, the EM algorithm turns the original problem into an alternating optimization problem, where $\mathcal{L}(q, \boldsymbol{\theta})$ is minimized w.r.t q and $\boldsymbol{\theta}$ in two stages, respectively. At each iteration n, the expectation (E) step maximizes $\mathcal{L}(q, \boldsymbol{\theta}^{(n)})$ w.r.t q. From Eq.(2.14), this is achieved by setting q to the current true posterior:

$$\text{E-step:} \quad q^{(n+1)}(\boldsymbol{y}|\boldsymbol{x}^*) = p_{\theta^{(n)}}(\boldsymbol{y}|\boldsymbol{x}^*), \tag{2.15}$$

so that the KL divergence vanishes and the upper bound is tight. In the subsequent maximization (M) step, $\mathcal{L}(q^{(n+1)}, \boldsymbol{\theta})$ is minimized w.r.t $\boldsymbol{\theta}$:

$$\text{M-step:} \quad \max_{\boldsymbol{\theta}} \mathbb{E}_{q^{(n+1)}(\boldsymbol{y}|\boldsymbol{x}^*)}\left[\log p_\theta(\boldsymbol{x}^*, \boldsymbol{y})\right], \tag{2.16}$$

which is to maximize the expected complete data log-likelihood. The EM algorithm has an appealing property that it monotonically decreases the negative marginal log-likelihood over iterations. To see this, notice that after the E-step the upper bound $\mathcal{L}(q^{(n+1)}, \boldsymbol{\theta}^{(n)})$ is equal to the negative marginal log-likelihood, and the M-step further decreases the upper bound (and thus the negative marginal log-likelihood).

Variational EM When the model $p_\theta(\boldsymbol{x}, \boldsymbol{y})$ is complex (e.g., a neural network or a multi-layer graphical model), directly working with the true posterior in the E-step becomes intractable. Variational EM overcomes the difficulty with approximations. The approach considers a restricted family $\mathcal{Q}'$ of the variational distribution $q(\boldsymbol{y})$ such that optimization w.r.t q within the family is tractable:

$$\text{Variational E-step:} \quad \min_{q \in \mathcal{Q}'} \mathcal{L}(q, \boldsymbol{\theta}^{(t)}). \tag{2.17}$$

A common way to restrict the q family is the *mean-field methods* which partition the components of $\boldsymbol{y}$ into sub-groups $\boldsymbol{y} = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_M)$ and assume that q factorizes w.r.t the groups: $q(\boldsymbol{y}) = \prod_{i=1}^M q_i(\boldsymbol{y}_i)$. The variational principle summarized in (Wainwright and Jordan, 2008) gives a more principled interpretation of the mean-field and other approximation methods. In particular, in the case where $p_\theta(\boldsymbol{x}, \boldsymbol{y})$ is an exponential family distribution with sufficient statistics $T(\boldsymbol{x}, \boldsymbol{y})$, the exact E-step (Eq.2.15) can be interpreted as seeking the optimal valid *mean parameters* (i.e., expected sufficient statistics) for which the free energy is minimized. For discrete latent variables $\boldsymbol{y}$, the set of all valid mean parameters constitutes a marginal polytope $\mathcal{M}$. In this perspective, the mean-field methods (Eq.2.17) correspond to replacing $\mathcal{M}$ with an *inner* approximation $\mathcal{M}' \subseteq \mathcal{M}$. With the restricted set $\mathcal{M}'$ of mean parameters, the E-step generally no longer tightens the bound of the negative marginal log-likelihood, and the algorithm does not necessarily decrease the negative marginal log-likelihood monotonically. However, the algorithm preserves the property that it minimizes the upper bound of the negative marginal log-likelihood.

Besides the mean-field methods, there are other approaches for approximation such as belief propagation. These methods correspond to using an *outer* approximation $\mathcal{M}'' \supseteq \mathcal{M}$ of the marginal polytope, and do not guarantee upper bounds on the negative marginal log-likelihood.

Another approach to restrict the family of q is to assume a parametric distribution $q_\omega(\boldsymbol{y}|\boldsymbol{x})$ and optimize the parameters $\boldsymbol{\omega}$ in the E-step. The approach has been used in black-box variational inference (Ranganath et al., 2014), and variational auto-encoders (VAEs) (Kingma and Welling, 2013) where q is parameterized as a neural network (a.k.a inference network).

Wake-Sleep In some cases when the auxiliary q is assumed to have a certain form (e.g., a deep network), the approximate E-step in Eq.(2.17) may still be too complex to be tractable, or the gradient estimator (w.r.t the parameters of q) can suffer from high variance (Paisley et al., 2012; Mnih and Gregor, 2014). To tackle the challenge, more approximations are introduced. Wake-sleep algorithm (Hinton et al., 1995) is one of such methods. In the E-step w.r.t q, rather than minimizing $\mathrm{KL}(q(\boldsymbol{y})\|p_\theta(\boldsymbol{y}|\boldsymbol{x}^*))$ (Eq.2.14) as in EM and variational EM, the wake-sleep algorithm makes an approximation by minimizing the KL divergence in opposite direction:

$$\text{Approximate E-step (Sleep-phase):} \quad \min_{q \in \mathcal{Q}'} \mathrm{KL}\left(p_\theta(\boldsymbol{y}|\boldsymbol{x}^*)\|q(\boldsymbol{y})\right), \tag{2.18}$$

which can be optimized efficiently with gradient descent when q is parameterized. Besides wake-sleep, one can also use other methods for low-variance gradient estimation in Eq.(2.17), such as reparameterization gradient (Kingma and Welling, 2013) and score gradient (Glynn, 1990; Ranganath et al., 2014; Mnih and Gregor, 2014).

In sum, the above techniques for unsupervised MLE again show the close connection between the principles of maximum likelihood and maximum entropy. Starting from the upper bound of negative marginal log-likelihood (Eq.2.14) with maximum entropy and minimum cross entropy, the original intractable MLE problem gets simplified, and a series of solving algorithms, ranging from (variational) EM to wake-sleep, arise naturally as an approximation to the original solution.

2.1.2 Bayesian Inference

Now we visit another classical learning framework, Bayesian inference, and review its intriguing connections with the maximum entropy principle.

Different from MLE, Bayesian approach for statistical inference treats the hypotheses (parameters $\boldsymbol{\theta}$) to be inferred as random variables. Assuming a prior distribution $\pi(\boldsymbol{\theta})$ over the parameters and considering the model to be a conditional distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$, the inference is based on the Bayes's theorem:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{\pi(\boldsymbol{\theta})\prod_{\boldsymbol{x}^* \in \mathcal{D}} p(\boldsymbol{x}^*|\boldsymbol{\theta})}{p(\mathcal{D})}, \tag{2.19}$$

where $p(\boldsymbol{\theta}|\mathcal{D})$ is the posterior distribution after observing the data $\mathcal{D}$; and $p(\mathcal{D}) = \int_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta}) \prod_{\boldsymbol{x}^*} p(\boldsymbol{x}^*|\boldsymbol{\theta})$ is the marginal likelihood.

Interestingly, the early work by Zellner (1988) showed the relations between Bayesian inference and maximum entropy, by re-formulating the statistical inference problem from the perspective of information processing, and re-discovering the Bayes' theorem as the optimal information processing rule. More specifically, statistical inference can be seen as a procedure of information processing, where the system receives input information in the form of prior knowledge and data, and emits output information in the form of parameter estimates and others. An efficient inference procedure should generate an output distribution such that the system retains all input information and not inject any extraneous information. The learning objective is thus to minimize the difference between the input and output information w.r.t the output distribution:

$$\begin{aligned} \min_{q(\boldsymbol{\theta})} \quad & -\mathrm{H}(q(\boldsymbol{\theta})) + \log p(\mathcal{D}) - \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \pi(\boldsymbol{\theta}) + \sum\nolimits_{\boldsymbol{x}^* \in \mathcal{D}} \log p(\boldsymbol{x}^*|\boldsymbol{\theta})\right] \\ s.t. \quad & q(\boldsymbol{\theta}) \in \mathcal{P}(\Theta), \end{aligned} \tag{2.20}$$

where the first two terms measure the output information in the output distribution $q(\boldsymbol{\theta})$ and marginal $p(\mathcal{D})$, and the third term measures the input information in the prior $\pi(\boldsymbol{\theta})$ and data likelihood $p(\boldsymbol{x}^*|\boldsymbol{\theta})$. Here $\mathcal{P}(\Theta)$ is the space of all probability distributions over $\boldsymbol{\theta}$.

The optimal solution of $q(\boldsymbol{\theta})$ is precisely the the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ due to the Bayes' theorem (Eq.2.19). The proof is straightforward by noticing that the objective can be further rewritten as $\min_q \mathrm{KL}(q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}|\mathcal{D}))$.

Similar to the case of duality between MLE and maximum entropy (Eq.2.4), the entropy point of view casts Bayesian inference into a constrained optimization problem. As Jaynes (1988) commented, this fresh interpretation of Bayes' theorem "could make the use of Bayesian methods more attractive and widespread, and stimulate new developments in the general theory of inference". The next subsection reviews how entropy maximization as a "useful tool in generating probability distributions" (Jaynes, 1988) has related to and resulted in more general learning and inference frameworks, such as posterior regularization.

2.1.3 Posterior Regularization

The maximum entropy perspective offers another important flexibility in learning by allowing imposing rich constraints to regularize the learning. That is, in the optimization-based formulation of Bayesian inference in Eq.(2.20), we have imposed on the posterior q the standard normality constraint of a probability distribution. It is natural to consider other types of constraints that encode richer problem structures and domain knowledge, which guide the model to learn desired behaviors.

The idea has led to posterior regularization (PR, Ganchev et al., 2010) or regularized Bayes (RegBayes, Zhu et al., 2014) which augment the Bayesian inference objective with additional constraints:

$$
\begin{aligned}
\min_{q,\boldsymbol{\xi}} \quad & -\mathrm{H}\left(q(\boldsymbol{\theta})\right) - \mathbb{E}_{q(\boldsymbol{\theta})}\left[\sum\nolimits_{\boldsymbol{x}^* \in \mathcal{D}} \log p(\boldsymbol{x}^*|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\right] + U(\boldsymbol{\xi}) \\
s.t. \quad & q(\boldsymbol{\theta}) \in \mathcal{Q}(\boldsymbol{\xi}) \\
& \boldsymbol{\xi} \geq 0,
\end{aligned}
\tag{2.21}
$$

where we have rearranged the terms and dropped any constant factors in Eq.(2.20), and added constraints with $\boldsymbol{\xi}$ being a vector of slack variables, $U(\boldsymbol{\xi})$ a penalty function (e.g., ℓ_1 norm of $\boldsymbol{\xi}$), and $\mathcal{Q}(\boldsymbol{\xi})$ a subset of valid distributions over $\boldsymbol{\theta}$ that satisfy the constraints determined by $\boldsymbol{\xi}$. The optimization problem is generally easy to solve when the penalty/constraints are convex and defined w.r.t a linear operator (e.g., expectation) of the posterior q. For example, let $T(\boldsymbol{x}^*; \boldsymbol{\theta})$ be a feature vector of data instance $\boldsymbol{x}^* \in \mathcal{D}$, the constraint posterior set Q can be defined as:

$$
Q(\boldsymbol{\xi}) := \left\{ q(\boldsymbol{\theta}) \ : \ \mathbb{E}_q\left[T(\boldsymbol{x}^*; \boldsymbol{\theta})\right] \leq \boldsymbol{\xi} \right\}, \tag{2.22}
$$

which bounds the feature expectations with $\boldsymbol{\xi}$.

Max-margin constraint is another expectation constraint that has shown to be widely effective in classification and regression (Vapnik, 1998). The maximum entropy discrimination (MED) by Jaakkola et al. (2000) regularizes linear regression models with the max-margin constraints, which is latter generalized to more complex models $p(\boldsymbol{x}|\boldsymbol{\theta})$, such as Markov networks (Taskar et al., 2004) and latent variable models (Zhu et al., 2014). Formally, let $\boldsymbol{y}^* \in \mathbb{R}$ be the observed label associated with $\boldsymbol{x}^*$. The margin-based constraint says that a classification/regression function $h(\boldsymbol{x}; \boldsymbol{\theta})$ should make at most ϵ deviation from the true label $\boldsymbol{y}^*$. Specifically, consider the common choice of the function h as a linear function: $h(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{\theta}^\top T(\boldsymbol{x})$, where $T(\boldsymbol{x})$ is, with a slight abuse of notation, the feature of instance $\boldsymbol{x}$. The constraint is written as:

$$
\begin{cases}
\boldsymbol{y}^* - \mathbb{E}_q\left[\boldsymbol{\theta}^\top T(\boldsymbol{x}^*)\right] \leq \epsilon + \xi \\
-\boldsymbol{y}^* + \mathbb{E}_q\left[\boldsymbol{\theta}^\top T(\boldsymbol{x}^*)\right] \leq \epsilon + \xi',
\end{cases}
\tag{2.23}
$$

for all instances $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in \mathcal{D}$.

The posterior regularization technique does not have to be applied in Bayesian inference, but instead can be used to regularize arbitrary random variables in various settings. For example, Hu et al. (2016a) studied the generalized setting of regularizing the output variables of deep neural networks with complex rule constraints. Consider $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ as a neural network of an arbitrary architecture (e.g., a convolutional or recurrent network). Rather than treating $\boldsymbol{\theta}$ as a random variable, we are usually interested in a point estimation of $\boldsymbol{\theta}$. Let f_{rule} be a FOL rule, and $f_{\text{rule}}(\boldsymbol{x}, \boldsymbol{y})$ be the grounding on a configuration $(\boldsymbol{x}, \boldsymbol{y})$ which evaluates the truth value of the logical rule given $(\boldsymbol{x}, \boldsymbol{y})$. By using soft logic, $f_{\text{rule}}(\boldsymbol{x}, \boldsymbol{y})$ can take a continuous truth value $\in [0, 1]$ (see section 2.3.2 for more details). The logical rule serves as a constraint that regularize the output variables $\boldsymbol{y}$ of the neural network during training, encouraging the model to make predictions that satisfy the constraint. Given a data instance $\boldsymbol{x}^* \in \mathcal{D}$, the problem is written as:

$$\begin{aligned} \min_{\boldsymbol{\theta}, q, \xi} \ & - \mathrm{H}\left(q(\boldsymbol{y}|\boldsymbol{x}^*)\right) - \mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x}^*)}\left[\log p_\theta(\boldsymbol{y}|\boldsymbol{x}^*)\right] + C\xi \\ s.t. \ & \mathbb{E}_q\left[1 - f_{\text{rule}}(\boldsymbol{x}^*, \boldsymbol{y})\right] \leq \xi \\ & q \in \mathcal{P}(\mathcal{Y}), \ \ \xi \geq 0, \end{aligned} \tag{2.24}$$

where $q(\boldsymbol{y}|\boldsymbol{x})$ is an auxiliary distribution over $\boldsymbol{y}$, and we want the expected logical truth value to be close to 1; C is a tradeoff hyperparameter.

EM for posterior regularization The posterior regularization framework uses an EM-style algorithm as a general solver. Akin to the standard EM algorithm (section 2.1.1.2), here q and $\boldsymbol{\theta}$ are alternatingly optimized with E- and M-steps, respectively. Taking the setting in Eq.(2.24) for example, at each iteration n, the E-step estimates q in a multiplicative form of the current model $p_{\theta^{(n)}}(\boldsymbol{y}|\boldsymbol{x}^*)$ and the rule $f_{rule}(\boldsymbol{x}^*, \boldsymbol{y})$:

$$\text{E-step:} \quad q^{(n+1)}(\boldsymbol{y}|\boldsymbol{x}^*) = \exp\Big\{ \log p_{\theta^{(n)}}(\boldsymbol{y}|\boldsymbol{x}^*) - C(1 - f_{\text{rule}}(\boldsymbol{x}^*, \boldsymbol{y}))\Big\}/Z, \tag{2.25}$$

where Z is the normalization factor. Intuitively, a configuration of $\boldsymbol{y}$ with a higher truth value $f_{\text{rule}}(\boldsymbol{x}^*, \boldsymbol{y})$ will receive a higher probability under $q^{(n+1)}$. The M-step updates $\boldsymbol{\theta}$ in the same way as the standard EM:

$$\text{M-step:} \quad \max_{\boldsymbol{\theta}} \mathbb{E}_{q^{(n+1)}(\boldsymbol{y}|\boldsymbol{x}^*)}\left[\log p_\theta(\boldsymbol{y}|\boldsymbol{x}^*)\right]. \tag{2.26}$$

2.1.4 Summary

We have seen that the maximum entropy formalism provides alternative insights into the classical learning frameworks of MLE, Bayesian inference, posterior regularization, etc. By treating learning as a constrained optimization problem, the maximum entropy point of view offers a great source of flexibility for applying many powerful tools for efficient approximation and enhanced learning, such as variational approximation and convex duality as above, and kernel methods as used in (Taskar et al., 2004; Zhu and Xing, 2009).

Further, the entropic variational treatment of the learning frameworks highlights the common components shared across them. The (auxiliary) distribution q serves as a vehicle to consolidate the data/constraint information while meeting the entropy criterion (e.g., Eqs.2.14, 2.24). In the cases where the goal is to estimate the parametric model p_θ, the learning is carried out by EM-style coordinate descent, in which the model is often determined by minimizing its cross entropy with q.

It is intriguing that, in the dual point of view on the problem of (supervised) MLE, data instances are encoded as constraints (Eq.2.4), much like the structured constraints in posterior regularization. In the following sections, we present the standardized formalism of machine learning algorithms and show that indeed myriad types of experiences including data instances and constraints can all be encoded in the same generic form and be used in learning.

2.2 The Standard Equation

The preceding review inspires the emergence of a general machine learning formalism, that captures the *standardized* learning problem in the presence of broadly different types of experiences, and that subsumes and generalizes a wealth of major machine learning paradigms. We present the general formalism in this section.

For generality, let $\boldsymbol{t} \in \mathcal{T}$ be the variable of interest, e.g., the input-output pair $\boldsymbol{t} = (\boldsymbol{x}, \boldsymbol{y})$ in a prediction task, or the target variable $\boldsymbol{t} = \boldsymbol{x}$ in generative modeling. The first key ingredient of the standardized formulation is the *experience function* $f(\boldsymbol{t}) \in \mathbb{R}$ that measures the goodness of a configuration $\boldsymbol{t}$ in light of any given experiences. As discussed in section 2.3, all diverse forms of experiences (e.g., data, constraints, reward, adversarial discriminators, etc) can be encoded as an experience function. In essence, the experience function provides a unified language to express all information about the unknown target model, on which basis we are able to formulate a standardized way of using the information to identify the desired model.

Let $p_\theta(\boldsymbol{t})$ be the target model to be learned from K experience functions $\{f_k(\boldsymbol{t})\}_{k=1}^K$. We additionally introduce an auxiliary distribution $q(\boldsymbol{t})$. The standard equation of the learning problem, in a constrained form, is written as:

$$\begin{aligned} \min_{q,\boldsymbol{\theta},\boldsymbol{\xi}} \quad & -\alpha\mathbb{H}\left(q\right) + \beta\mathbb{D}\left(q, p_\theta\right) + U(\boldsymbol{\xi}) \\ s.t. \quad & -\mathbb{E}_q\left[f_k\right] \leq \xi_k, \;\; k = 1, \ldots, K, \end{aligned} \tag{2.27}$$

where $U(\boldsymbol{\xi})$ is a penalty function and $\boldsymbol{\xi} \in \mathbb{R}^K$ the slack variables. The equation contains three essential terms: the uncertainty of q measured with $\mathbb{H}\left(q\right)$, such as the Shannon entropy $\mathrm{H}(q)$; the divergence $\mathbb{D}\left(q, p_\theta\right)$ between the auxiliary q and the model p_θ; and the experiences f_k under the expectation of the auxiliary distribution $\mathbb{E}_q\left[f_k\right]$. Here $\alpha, \beta \geq 0$ are tradeoff hyper-parameters.

Before diving deeper into each of the components, we first take a look at some more specialized forms of the standard equation, which serve as the basis for many of our subsequent discussions. Assuming the common choice of the penalty $U(\boldsymbol{\xi}) = \sum_k \xi_k$, and, with a slight abuse of notations, $f = \sum_k f_k$, the above equation can equivalently be written in an unconstrained form:

$$\min_{q,\boldsymbol{\theta}} \; -\alpha\mathbb{H}\left(q\right) + \beta\mathbb{D}\left(q, p_\theta\right) - \mathbb{E}_q\left[f\right]. \tag{2.28}$$

The optimization is w.r.t both the auxiliary distribution q and the model parameters $\boldsymbol{\theta}$. For a more concrete instantiation of the standard equation, as we shall visit in the following sections, many of the popular algorithms use cross entropy (CE) for the divergence measure $\mathbb{D}$ and Shannon entropy for the uncertainty measure $\mathbb{H}$, leading to:

$$\min_{q,\boldsymbol{\theta}} \; -\alpha\mathrm{H}\left(q\right) - \beta\mathbb{E}_q\left[\log p_\theta\right] - \mathbb{E}_q\left[f\right]. \tag{2.29}$$

An intuitive interpretation of the standard equation can be obtained by taking a closer look at the three constituent parts, each of which plays an indispensable and complementary role in defining the learning problem:

- The experience term $\mathbb{E}_q[f]$ is an exogenous regularization that determines the external information used to drive the learning in the system. The effect of maximizing the expectation is such that q is encouraged to produce samples of high scores as evaluated by the experience function, or in other words, samples of high quality in light of the experiences. We show in section 2.3 how the experience term can flexibly accommodate a broad diversity of experiences, ranging from common types to advanced emerging ones.
- The divergence $\mathbb{D}(q, p_\theta)$ being minimized encourages the auxiliary distribution q and the target distribution p_θ to stay close to each other. Intuitively, this forms a "teacher-student" mechanism where the teacher q absorbs knowledge from the experiences and then, by minimizing the divergence, transfers the knowledge to the student p_θ. That is, the target model parameters $\boldsymbol{\theta}$ are learned by matching the model to the auxiliary q.

  On the other hand, through the minimized divergence, the state of the model p_θ also influences the solution of q. The influence of p on q is particularly interesting when the divergence function takes the form of cross entropy (Eq.2.29), where we can see clear symmetry between the divergence and experience terms, both defined w.r.t the expectations under q. Thus the model can be seen as another "experience" that biases the q solution, as shown more clearly in Eq.(2.30) below. Section 2.4 discusses different choices of the divergence measure $\mathbb{D}$.
- The uncertainty $\mathbb{H}(q)$ is a self-regularization on the complexity of q. It conforms to the idea of maximum entropy principle to generate the distribution q with largest uncertainty (under the "constraint" that q fits the available information including the experiences and the model).

The standard equation formulates a vast design space of learning algorithms with the succinct set of components. We show in the following sections that a number of existing well-known but seemingly distinct learning algorithms are all instances of the standard equation due to different choices of the experience function, divergence measure, and tradeoff hyper-parameters. This provides another justification of the general formalism, and sheds new light on relationships between the existing frameworks. Before moving on, it would be useful to first look at how the standard equation, in particular the special case in Eq.(2.29) is optimized, which helps reveal the connection to existing frameworks in the following.

The problem in Eq.(2.29) can be solved with an EM-style alternating optimization, wherein we optimize w.r.t q and $\boldsymbol{\theta}$ in an iterative manner. At iteration n:

$$\begin{aligned} \text{E-step:} \quad & q^{(n+1)}(\boldsymbol{t}) = \exp\left\{ \frac{\beta \log p_{\theta^{(n)}}(\boldsymbol{t}) + f(\boldsymbol{t})}{\alpha} \right\} \Big/ Z \\ \text{M-step:} \quad & \boldsymbol{\theta}^{(n+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{q^{(n+1)}(\boldsymbol{t})}\big[\log p_\theta(\boldsymbol{t})\big], \end{aligned} \tag{2.30}$$

where Z is the normalization factor. In particular, q has a closed-form solution in the E-step, which is a multiplicative combination between the experience function f and the current state of model $p_{\theta^{(t)}}$. The update rule in the M-step is essentially maximizing the model's log-likelihood of the samples from the auxiliary distribution q.

It is worth noting that, depending on the concrete problems, the model distribution p_θ can have different structures and be decomposed accordingly. For example, if $\boldsymbol{t} = (\boldsymbol{x}, \boldsymbol{y})$, one could specify a generative model $p_\theta(\boldsymbol{x}, \boldsymbol{y})$ decomposed as $p_\theta(\boldsymbol{x}, \boldsymbol{y}) = p_\theta(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})$ with a fixed prior $p(\boldsymbol{y})$. The same applies to the auxiliary distribution q depending on the factors one is interested in learning. We will see concrete examples in the following.

2.3 The Experience Function

The experience function $f(\boldsymbol{t})$ in the standard equation can be instantiated to encode vastly distinct types of experiences. Different choices of $f(\boldsymbol{t})$ result in learning algorithms applied to specific problem settings. Moreover, with particular choices, the standard equation rediscovers a wide array of well-known algorithms. The resulting common treatment of the previously disparate algorithms is appealing as it offers new holistic insights into the commonalities and differences of those algorithms. Table 2.1 shows the extant algorithms that are recovered by the standard equation.

2.3.1 SE with Data Instance Experiences

We first consider the most common type of experience, namely data instances, which can appear in a wide range of contexts including supervised, unsupervised, actively supervised, and other scenarios with data manipulation.

2.3.1.1 Supervised data instances

Without loss of generality, denote the data instance as a pair $\boldsymbol{t} = (\boldsymbol{x}, \boldsymbol{y})$. In the supervised setting, we observe the full data drawn from the data distribution $(\boldsymbol{x}^*, \boldsymbol{y}^*) \sim p_d(\boldsymbol{x}^*, \boldsymbol{y}^*)$. For an arbitrary configuration $(\boldsymbol{x}, \boldsymbol{y})$, its probability $p_d(\boldsymbol{x}, \boldsymbol{y})$ can be seen as measuring the expected *similarity* between $(\boldsymbol{x}, \boldsymbol{y})$ and true data $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, and re-written as $p_d(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}_{p_d(\boldsymbol{x}^*, \boldsymbol{y}^*)}\left[\mathbb{I}_{(\boldsymbol{x}^*, \boldsymbol{y}^*)}(\boldsymbol{x}, \boldsymbol{y})\right]$. Here the similarity measure is $\mathbb{I}_{(\boldsymbol{x}^*, \boldsymbol{y}^*)}(\boldsymbol{x}, \boldsymbol{y})$, an indicator function that takes the value 1 if $(\boldsymbol{x}, \boldsymbol{y})$ equals $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ and 0 otherwise.

In practice, we are given an empirical distribution $\tilde{p}_d(\boldsymbol{x}, \boldsymbol{y})$ by observing a collection of instances $\mathcal{D}$ on which the expectation is evaluated:

$$\mathbb{E}_{(\boldsymbol{x}^*, \boldsymbol{y}^*) \sim \mathcal{D}}\left[\mathbb{I}_{(\boldsymbol{x}^*, \boldsymbol{y}^*)}(\boldsymbol{x}, \boldsymbol{y})\right] = \frac{m(\boldsymbol{x}, \boldsymbol{y})}{N}, \tag{2.31}$$

where N is the size of the dataset $\mathcal{D}$, and $m(\boldsymbol{x}, \boldsymbol{y})$ is the number of occurrences of the configuration $(\boldsymbol{x}, \boldsymbol{y})$ in $\mathcal{D}$. The data instance experience is incorporated in the standard equation for learning. As can be seen below, it is most convenient to set the experience function to the logarithm of the expected similarity:

$$f := f_{\text{data}}(\boldsymbol{x}, \boldsymbol{y}; \mathcal{D}) = \log \mathbb{E}_{(\boldsymbol{x}^*, \boldsymbol{y}^*) \sim \mathcal{D}}\left[\mathbb{I}_{(\boldsymbol{x}^*, \boldsymbol{y}^*)}(\boldsymbol{x}, \boldsymbol{y})\right]. \tag{2.32}$$

With this from of f, we show that the standard equation derives the supervised MLE algorithm.

Supervised MLE In the problem of Eq.(2.29), we set $\alpha = 1$ and β to a very small positive value ϵ, so that the auxiliary distribution $q(\boldsymbol{x}, \boldsymbol{y})$ is determined directly by the full data instances (not the model p_θ through the divergence term). As a result, the solution of q in the E-step (Eq.2.30) reduces to the empirical distribution:

$$q(\boldsymbol{x}, \boldsymbol{y}) = \exp\left\{\frac{\beta \log p_\theta(\boldsymbol{x}, \boldsymbol{y}) + f_{\text{data}}(\boldsymbol{x}, \boldsymbol{y}; \mathcal{D})}{\alpha}\right\} / Z \approx \exp\left\{f_{\text{data}}(\boldsymbol{x}, \boldsymbol{y}; \mathcal{D})\right\} / Z = \tilde{p}_d(\boldsymbol{x}, \boldsymbol{y}). \quad (2.33)$$

The subsequent M-step that maximizes the log-likelihood of samples from q then leads to the supervised MLE updates w.r.t $\boldsymbol{\theta}$.

2.3.1.2 Unsupervised data instances

In the unsupervised setting, we are given the empirical distribution $\tilde{p}_d(\boldsymbol{x})$ by only observing $\mathcal{D} = \{\boldsymbol{x}^*\}$ without the associated $\boldsymbol{y}^*$. The experience can be encoded in the same form as the supervised data (Eq.2.32) but now with only the information of $\boldsymbol{x}^*$:

$$f := f_{\text{data}}(\boldsymbol{x}; \mathcal{D}) = \log \mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}} \left[\mathbb{I}_{\boldsymbol{x}^*}(\boldsymbol{x})\right]. \quad (2.34)$$

Similarly, we apply the standard equation to this setting, and derive the unsupervised MLE algorithm with particular specifications.

Unsupervised MLE The form of Eq.(2.29) is indeed highly reminiscent of the variational free energy objective in the standard EM for unsupervised MLE (Eq.2.14). Exact correspondence is obtained by setting $\alpha = \beta = 1$ and imposing the structure $q(\boldsymbol{x}, \boldsymbol{y}) = \tilde{p}_d(\boldsymbol{x}) q(\boldsymbol{y}|\boldsymbol{x})$. The explanation for $\beta = 1$, which differs from the specification $\beta = \epsilon$ in the supervised setting, is that the auxiliary distribution q cannot be determined fully by the unsupervised, "incomplete" data experience alone, and additionally relies on p_θ through the divergence term. Here q is assumed a specialized structure $q(\boldsymbol{x}, \boldsymbol{y}) = \tilde{p}_d(\boldsymbol{x}) q(\boldsymbol{y}|\boldsymbol{x})$ where $\tilde{p}_d(\boldsymbol{x})$ is fixed and thus not influenced by p_θ. In contrast, if no structure of q is assumed, we could potentially obtain an extended, *instance-weighted* version of EM where each instance $\boldsymbol{x}^*$ is weighted by the marginal likelihood $p_\theta(\boldsymbol{x}^*)$, in line with the previous weighted EM methods for robust clustering (e.g., Gebru et al., 2016; Yu et al., 2011). We defer detailed derivations to the supplementary materials.

2.3.1.3 Manipulated data instances

Data manipulation, such as weighting data instances or augmenting an existing set with new instances, is often a crucial step for efficient learning, especially in low data regime or in presence of low-quality datasets (e.g., imbalanced labels). We show the rich data manipulation schemes can be treated as experiences and be naturally encoded in the experience function (Hu et al., 2019b). This is done by extending the data instance experience function (Eq.2.32), in particular by enriching the similarity metric in different ways.

Data re-weighting Rather than assuming the same importance of all data instances, we can associate each instance $\boldsymbol{t}^*$ with an importance weight $w(\boldsymbol{t}^*) \in \mathbb{R}$, by scaling the 0/1 indicator function in Eq.(2.32) with the weight:

$$f_{\text{data-w}}(\boldsymbol{t}; \mathcal{D}) := \log \mathbb{E}_{\boldsymbol{t}^* \sim \mathcal{D}} \left[w(\boldsymbol{t}^*) \cdot \mathbb{I}_{\boldsymbol{t}^*}(\boldsymbol{t}) \right]. \tag{2.35}$$

Plugging $f_{\text{data-w}}$ into the standard equation (Eq.2.29) with the same other specification of supervised MLE ($\alpha = 1, \beta = \epsilon$), we get the update rule of model parameters $\boldsymbol{\theta}$ in the M-step (Eq.2.30):

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{t}^* \sim \mathcal{D}} \left[w(\boldsymbol{t}^*) \cdot \log p_\theta(\boldsymbol{t}^*) \right], \tag{2.36}$$

which is the familiar weighted supervised MLE. The weights $\boldsymbol{w}$ can be specified *a priori* based on heuristics, e.g., using inverse class frequency. Yet it is often desirable to automatically induce and adapt the weights during the course of model training. In chapter 5, we discuss how the standard equation can easily enable automated data re-weighting by re-using existing algorithms that were designed to solve other seemingly-unrelated problems.

Data augmentation The indicator function $\mathbb{I}$ as the similarity metric restrictively requires exact match between the true $\boldsymbol{t}^*$ and the configuration $\boldsymbol{t}$. Data augmentation arises as an "relaxation" to the similarity metric. Let $a_{\boldsymbol{t}^*}(\boldsymbol{t}) \geq 0$ be a distribution that assigns non-zero probability to not only the configuration of the exact $\boldsymbol{t}^*$ but also other configurations $\boldsymbol{t}$ related to $\boldsymbol{t}^*$ in certain ways (e.g., all rotated images $\boldsymbol{t}$ of the observed image $\boldsymbol{t}^*$). Replacing the indicator function metric in Eq.(2.32) with the new $a_{\boldsymbol{t}^*}(\boldsymbol{t}) \geq 0$ yields the experience function for data augmentation:

$$f_{\text{data-aug}}(\boldsymbol{t}; \mathcal{D}) := \log \mathbb{E}_{\boldsymbol{t}^* \sim \mathcal{D}} \left[a_{\boldsymbol{t}^*}(\boldsymbol{t}) \right]. \tag{2.37}$$

The resulting M-step updates of $\boldsymbol{\theta}$, keeping ($\alpha = 1, \beta = \epsilon$) of supervised MLE, is thus:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{t}^* \sim \mathcal{D},\ \boldsymbol{t} \sim a_{\boldsymbol{t}^*}(\boldsymbol{t})} \left[\log p_\theta(\boldsymbol{t}) \right]. \tag{2.38}$$

The metric $a_{\boldsymbol{t}^*}(\boldsymbol{t})$ can be defined in various ways, leading to different augmentation strategies. For example, setting $a_{\boldsymbol{t}^*}(\boldsymbol{t}) \propto \exp\{R(\boldsymbol{t}, \boldsymbol{t}^*)\}$, where $R(\boldsymbol{t}, \boldsymbol{t}^*)$ is a task-specific evaluation metric such as BLEU for machine translation, results in the reward-augmented maximum likelihood (RAML) algorithm (Norouzi et al., 2016). Besides the manually designed strategies, we can also specify $a_{\boldsymbol{t}^*}(\boldsymbol{t})$ as a parameterized generation process and learn any free parameters automatically. Notice the same form of the augmentation experience $f_{\text{data-aug}}$ and the re-weighting experience $f_{\text{data-w}}$, where the similarity metrics both include learnable components (i.e., $a_{\boldsymbol{t}^*}(\boldsymbol{t})$ and $w(\boldsymbol{t}^*)$, respectively). Thus the same solution to automated data re-weighting can also be applied for automated data augmentation, as discussed more in chapter 5.

2.3.1.4 Actively supervised data instances

Instead of access to data instances $\boldsymbol{x}^*$ with readily available labels $\boldsymbol{y}^*$, in the active supervision setting, we are presented with a large pool of unlabeled instances $\mathcal{D} = \{\boldsymbol{x}^*\}$ as well as a certain budget for querying an oracle (e.g., human annotators) for labeling a limited set of instances. To minimize the need for labeled instances, we strategically select queries from the pool according to an *informativeness* measure $u(\boldsymbol{x}) \in \mathbb{R}$. For example, $u(\boldsymbol{x})$ can be the predictive uncertainty on the instance $\boldsymbol{x}$, quantified by the Shannon entropy of the predictive distribution or the vote entropy based on a committee of predictors (Dagan and Engelson, 1995).

Mapping the standard equation to this setting, we show the informativeness measure $u(\boldsymbol{x})$ is subsumed as part of the experience. (Intuitively, $u(\boldsymbol{x})$ encodes our heuristic belief about sample "informativeness". This heuristic is a form of information we inject into the learning system.) Denote the oracle as o from which we can draw a label $\boldsymbol{y}^* \sim o(\boldsymbol{x}^*)$. The active supervision experience function is then defined as:

$$f_{\text{active}}(\boldsymbol{x}, \boldsymbol{y}; \mathcal{D}) := \log \mathbb{E}_{\boldsymbol{x}^* \sim \mathcal{D}, \boldsymbol{y}^* \sim o(\boldsymbol{x}^*)} \left[\mathbb{I}_{(\boldsymbol{x}^*, \boldsymbol{y}^*)}(\boldsymbol{x}, \boldsymbol{y}) \right] + \lambda \cdot u(\boldsymbol{x}), \tag{2.39}$$

where the first term is essentially the same as the supervised data experience function (Eq.2.32) with the only difference that now the label $\boldsymbol{y}^*$ is from the oracle rather than pre-given in $\mathcal{D}$; $\lambda > 0$ is a trade-off parameter. The formulation of the active supervision is interesting as it is simply a combination of the common supervision experience and the informativeness measure in an *additive* manner.

We plug f_{active} into the standard equation and obtain the algorithm to carry out learning. The result turns out to recover classical active learning algorithms.

Active learning Specifically, in Eq.(2.29), setting $f = f_{\text{active}}$, and $(\alpha = 1, \beta = \epsilon)$ as in supervised MLE, the resulting M-step in Eq.(2.30) for updating $\boldsymbol{\theta}$ is written as

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}^* \sim \tilde{p}_d(\boldsymbol{x}) \cdot \exp\{\lambda u(\boldsymbol{x})\},\ \boldsymbol{y}^* \sim o(\boldsymbol{x}^*)} \left[\log p_\theta(\boldsymbol{x}^*, \boldsymbol{y}^*) \right]. \tag{2.40}$$

If the pool $\mathcal{D}$ is large, the update can be carried out by the following procedure: we first pick a random subset $\mathcal{D}_{\text{sub}}$ from $\mathcal{D}$, and select a sample from $\mathcal{D}_{\text{sub}}$ according to the informativeness distribution proportional to $\exp\{\lambda u(\boldsymbol{x})\}$ over $\mathcal{D}_{\text{sub}}$. The sample is then labeled by the oracle, which is finally used to update the target model. By setting λ to a very large value (i.e., a near-zero "temperature"), we tend to select the *most* informative sample from $\mathcal{D}_{\text{sub}}$. The procedure rediscovers the algorithm proposed in (Ertekin et al., 2007) and more generally the pooling-based active learning algorithms (Settles, 2012).

2.3.2 SE with Knowledge-based Experiences

Many aspects of problem structures and human knowledge are difficult if not impossible to express through individual data instances. Examples include the knowledge of expected feature values, maximum margin structures, and logical rules, as discussed in section 2.1.3. The knowledge generally imposes constraints which we want the target model to satisfy. The experience function in the standard equation is a natural vehicle for encoding and incorporating such knowledge constraints in learning. Specifically, given a configuration $\boldsymbol{t}$, the experience function $f(\boldsymbol{t})$ measures the degree to which the configuration satisfies the respective constraints.

For example, as briefly discussed in section 2.1.3, the experience function can be defined based on *first-order logic* (FOL) rules which provide an expressive declarative language to encode complex symbolic knowledge (Hu et al., 2016a). More concretely, let $f_{\text{rule}}(\boldsymbol{t})$ be a FOL rule w.r.t the variables $\boldsymbol{t}$. For flexible encoding and stable optimization, we use soft logic (Bach et al., 2017) to formulate the rule. Soft logic allows continuous truth values from the interval $[0, 1]$ instead of $\{0, 1\}$, and the Boolean logical operators are redefined as:

$$\begin{aligned} &A\&B = \max\{A + B - 1, 0\}, \quad A \vee B = \min\{A + B, 1\} \\ &A_1 \wedge \cdots \wedge A_N = \sum\nolimits_i A_i/N, \quad \neg A = 1 - A. \end{aligned} \tag{2.41}$$

Here & and $\wedge$ are two different approximations to logical conjunction: & is useful as a selection operator (e.g., $A\&B = B$ when $A = 1$, and $A\&B = 0$ when $A = 0$), while $\wedge$ is an averaging operator. To give a concrete example, consider the problem of sentiment classification, where given a sentence $\boldsymbol{x}$, we want to predict its sentiment $\boldsymbol{y} \in \{\text{negative } 0, \text{positive } 1\}$. A challenge for a sentiment classifier is to understand the contrastive sense within a sentence and capture the dominant sentiment precisely. For example, if a sentence is of structure "A-but-B" with the connective "but", the sentiment of the half sentence after "but" dominates. Let $\boldsymbol{x}_B$ be the half sentence after "but" and $\tilde{\boldsymbol{y}}_B \in [0, 1]$ the (soft) sentiment prediction over $\boldsymbol{x}_B$ by the current model, a possible way to express the knowledge as a logical rule $f_{\text{rule}}(\boldsymbol{x}, \boldsymbol{y})$ is:

$$\text{has-‘A-but-B’-structure}(\boldsymbol{x}) \Rightarrow (\mathbb{I}(\boldsymbol{y} = 1) \Rightarrow \tilde{\boldsymbol{y}}_B \ \& \ \tilde{\boldsymbol{y}}_B \Rightarrow \mathbb{I}(\boldsymbol{y} = 1)), \tag{2.42}$$

where $\mathbb{I}(\cdot)$ is an indicator function that takes 1 when its argument is true, and 0 otherwise. Given an instantiation (i.e., grounding) of $(\boldsymbol{x}, \boldsymbol{y}, \tilde{\boldsymbol{y}}_B)$, the truth value of $f_{\text{rule}}(\boldsymbol{x}, \boldsymbol{y})$ can be evaluated by definitions in Eq.(2.41). Intuitively, the $f_{\text{rule}}(\boldsymbol{x}, \boldsymbol{y})$ truth value gets closer to 1 when $\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}_B$ are more consistent.

We then make use of the knowledge-based experiences such as $f_{\text{rule}}(\boldsymbol{t})$ to drive learning. The standard equation rediscovers classical algorithms for learning with symbolic knowledge.

Posterior regularization and beyond By setting $\alpha = \beta = 1$ and f to a constraint function such as f_{rule}, the cross-entropy based standard equation (Eq.2.29) results in the posterior regularization framework, e.g., as in Eq.(2.24) (in an unconstrained form). The trade-off hyperparameters can also take other values. By allowing arbitrary $\alpha \in \mathbb{R}$, the objective corresponds to the *unified expectation maximization* (UEM) algorithm (Samdani et al., 2012) that extended the posterior regularization for added flexibility.

2.3.3 SE with Reward Experiences

We now consider a different learning setting commonly seen in robotic control and other sequential decision making problems, where the source of experiences is the agent's interaction with the environment. The environment provides feedback in the form of rewards. More formally, consider a Markov decision process (MDP). At time t with state $\boldsymbol{x}_t$ of the environment, the agent draws an action $\boldsymbol{y}_t$ from the policy $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ (i.e., the model of interest). The state subsequently transits to $\boldsymbol{x}_{t+1}$ following certain transition dynamics of the environment, and yields a reward $r_t = r(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathbb{R}$. A discount factor $\gamma \in [0, 1]$ is applied to the future rewards. The goal of the agent is to perform actions that maximize the reward in the long run.

Thus the experience function that measures the goodness of the configuration $(\boldsymbol{x}, \boldsymbol{y})$ can be defined accordingly. Specifically, the base quantity is the expected future reward of taking action $\boldsymbol{y}$ in state $\boldsymbol{x}$ and continuing with the current policy $p_{\theta^{(n)}}$ (at the nth iteration of learning), which is also known as the action value function and can be estimated with various policy evaluation methods (Sutton and Barto, 2017):

$$Q^{(n)}(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \boldsymbol{x}_0 = \boldsymbol{x}, \boldsymbol{y}_0 = \boldsymbol{y}\right], \tag{2.43}$$

where the expectation is taken by following the state dynamics induced by the policy. We next discuss how the experience function can be specified on the basis of $Q^{(n)}(\boldsymbol{x}, \boldsymbol{y})$ in different ways, which derives from the perspective of standard equation different well-known algorithms in the reinforcement learning (RL) area.

Before moving on, we note that in this setting we are primarily interested in the conditional model (policy) $p_\theta(\boldsymbol{y}|\boldsymbol{x})$. The joint distribution $p_\theta(\boldsymbol{x}, \boldsymbol{y})$ in the standard equation is decomposed into $p_\theta(\boldsymbol{x}, \boldsymbol{y}) = p_\theta(\boldsymbol{y}|\boldsymbol{x})\mu^{(n)}(\boldsymbol{x})$, where $\mu^{(n)}(\boldsymbol{x}) = \sum_{t=0}^{\infty} p(\boldsymbol{x}_t = \boldsymbol{x})$ is the state distribution depending on the current policy $p_{\theta^{(n)}}$. The auxiliary distribution has the same structure $q(\boldsymbol{x}, \boldsymbol{y}) = q(\boldsymbol{y}|\boldsymbol{x})\mu^{(n)}(\boldsymbol{x})$.

RL as inference The most straightforward way of defining the experience function is of course to directly set it to the expected future reward:

$$f_{\text{reward},1}(\boldsymbol{x}, \boldsymbol{y}) = Q^{(n)}(\boldsymbol{x}, \boldsymbol{y}). \tag{2.44}$$

By setting $\alpha = \beta := \tau > 0$ in Eq.(2.29), we obtain the RL-as-inference approach which has a long history of research (e.g., Dayan and Hinton, 1997; Deisenroth et al., 2013; Rawlik et al., 2013; Levine, 2018; Abdolmaleki et al., 2018). The approach casts RL as a probabilistic inference problem, by introducing an additional binary random variable o, with $p(o = 1|\boldsymbol{x}, \boldsymbol{y}) \propto \exp\{Q(\boldsymbol{x}, \boldsymbol{y})/\tau\}$. Here τ is the temperature, and $o = 1$ is interpreted as the event that maximum reward is obtained. The goal of learning is to maximize the marginal likelihood of optimality: $\log p(o = 1)$, which however is intractable to solve. Much like how the standard equation applied to unsupervised MLE provides a variational bound objective for the marginal data likelihood (section 2.3.1.2), here the standard equation also derives a variational bound for $\log p(o = 1)$ (up to a constant factor) with the above specification of (f, α, β):

$$\begin{aligned} -\log p(o = 1) &= -\log \mathbb{E}_{\mu^{(n)}(\boldsymbol{x}) p_\theta(\boldsymbol{y}|\boldsymbol{x})}\left[p(o = 1|\boldsymbol{x}, \boldsymbol{y})\right] \\ &\leq -\tau \mathrm{H}(q) - \tau \mathbb{E}_{q(\boldsymbol{x},\boldsymbol{y})}\left[\log p_\theta(\boldsymbol{x}, \boldsymbol{y})\right] - \mathbb{E}_{q(\boldsymbol{x},\boldsymbol{y})}\left[Q^{(n)}(\boldsymbol{x}, \boldsymbol{y})\right]. \end{aligned} \tag{2.45}$$

The learning is then carried out following the EM procedure in Eq.(2.30).

Policy gradient We can also adopt a slightly different definition of the experience function by setting it to the logarithm of the reward:

$$f_{\text{reward},2}(\boldsymbol{x}, \boldsymbol{y}) = \log Q^{(n)}(\boldsymbol{x}, \boldsymbol{y}). \tag{2.46}$$

With $\alpha = \beta = 1$, we arrive at the classic policy gradient algorithm (Sutton et al., 2000). To see this, consider the EM optimization procedure in Eq.(2.30), where the E-step yields the solution $q(\boldsymbol{y}|\boldsymbol{x}) = p_{\theta^{(n)}}(\boldsymbol{y}|\boldsymbol{x}) Q^{(n)}(\boldsymbol{x}, \boldsymbol{y})/Z$, and the M-step updates $\boldsymbol{\theta}$ with the gradient:

$$\begin{aligned} \mathbb{E}_{q(\boldsymbol{x},\boldsymbol{y})} \left[\nabla_\theta \log p_\theta(\boldsymbol{y}|\boldsymbol{x}) \right] &= 1/Z \cdot \mathbb{E}_{\mu^{(n)}(\boldsymbol{x}) p_\theta(\boldsymbol{y}|\boldsymbol{x})} \left[Q^{(n)}(\boldsymbol{x}, \boldsymbol{y}) \nabla_\theta \log p_\theta(\boldsymbol{y}|\boldsymbol{x}) \right] \\ &= 1/Z \cdot \nabla_\theta \mathbb{E}_{\mu^{(n)}(\boldsymbol{x}) p_\theta(\boldsymbol{y}|\boldsymbol{x})} \left[Q^{(n)}(\boldsymbol{x}, \boldsymbol{y}) \right], \end{aligned} \tag{2.47}$$

where the first equation is due to importance sampling estimation with the proposal distribution $p_\theta(\boldsymbol{y}|\boldsymbol{x})$, and the second equation is due to the log-derivative trick $p_\theta \nabla_\theta \log p_\theta = \nabla_\theta p_\theta$. The final form is the policy gradient (up to the constant factor $1/Z$).

RL with intrinsic reward Rewards provided by the extrinsic environment can be sparse in many real-world sequential decision problems. Learning in such problems is made difficult due to the lack of supervision signals. A solution to alleviate the difficulty is to supplement the extrinsic reward with dense *intrinsic* reward that is generated by the agent itself (i.e., the agent is intrinsically motivated). The intrinsic reward can be induced in different ways, such as the "curiosity"-based reward that encourages the agent to explore novel or "supervising" states (Schmidhuber, 2010; Houthooft et al., 2016; Pathak et al., 2017), or the "optimal reward" which is designed with the goal of encouraging maximum extrinsic reward at the end (Singh et al., 2010; Zheng et al., 2018). Formally, let $r_t^{in} = r^{in}(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathbb{R}$ be the intrinsic reward at time t with state $\boldsymbol{x}_t$ and action $\boldsymbol{y}_t$. For example, in (Pathak et al., 2017), r_t^{in} is the prediction error (i.e., the "surprise") of the next state $\boldsymbol{x}_{t+1}$. Let $Q^{in,(n)}(\boldsymbol{x}, \boldsymbol{y})$ denote the action value function for the intrinsic reward defined in the same way as the extrinsic $Q^{(n)}(\boldsymbol{x}, \boldsymbol{y})$:

$$Q^{in,(n)}(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t^{in} \mid \boldsymbol{x}_0 = \boldsymbol{x}, \boldsymbol{y}_0 = \boldsymbol{y} \right]. \tag{2.48}$$

It is straightforward to derive the intrinsically-motivated variants of RL algorithms, by replacing the standard extrinsic-only $Q^{(n)}(\boldsymbol{x}, \boldsymbol{y})$ in the experience functions (Eqs.2.44 and 2.46) with the combined $Q^{(n)}(\boldsymbol{x}, \boldsymbol{y}) + Q^{in,(n)}(\boldsymbol{x}, \boldsymbol{y})$. Denote the resulting experience functions augmented by the intrinsic reward as $f_{\text{reward,in}}(\boldsymbol{x}, \boldsymbol{y})$.

It is interesting to notice the symmetry of $f_{\text{reward,in}}(\boldsymbol{x}, \boldsymbol{y})$ with the actively supervised data experience f_{active} which augments the standard supervised data experience with the additive informativeness measure $u(\boldsymbol{x})$ (section 2.3.1.4). The resemblance could naturally inspire mutual exchange of solutions between the research areas of intrinsic reward and active learning, as was also studied in earlier work (e.g., Schmidhuber, 2010; Li et al., 2011; Pathak et al., 2019). Besides, the RL research also developed rich algorithms of automatically adapting the intrinsic reward jointly with the learning of the target policy (e.g., Pathak et al., 2017; Zheng et al., 2018). Through the framework of standard equation, those algorithms can potentially be generalized to enable joint learning of all other types of experiences f (e.g., data manipulation, logical rules) with the target model p_θ. We discuss more in Part III of the thesis.

2.3.4 SE with Advanced Experiences

Besides the regular types of experiences discussed above, the experience function f can accommodate more advanced forms of experiences, such as those involving complex interactions with the target model through co-training or adversarial dynamics. The experiences may not have an analytic form but instead is defined in a variational way.

A concrete example is the "adversarial" experience emergingly used in certain learning settings. Specifically, the experiences discussed above are mostly defined *a priori* and encoded as a fixed experience function $f(\boldsymbol{t})$. For example, the data instance experience $f_{\text{data}}(\boldsymbol{t}; \mathcal{D})$ measures the closeness between a configuration $\boldsymbol{t}$ with the true data $\mathcal{D}$ based on data instance matching (Eq.2.32). Such manually-defined measures could be subjective, sub-optimal, or demanding expertise or heavy engineering to be properly defined. An alternative way that sidesteps the drawbacks is to automatically induce a measure $f_{\phi}(\boldsymbol{t})$, where $\boldsymbol{\phi}$ denote any free parameters associated with the experiences and to be learned. For example, one can measure the closeness of a configuration $\boldsymbol{t}$ to the dataset $\mathcal{D}$ based on a *discriminator* (or *critic*) that evaluates how easily $\boldsymbol{t}$ can be differentiated from the instances in $\mathcal{D}$. The similar idea of discriminator-based closeness measure was explored in the likelihood-free inference literature (Gutmann et al., 2014). The critic as the experience can be learned or adapted during the course of model training. We show in the next section how the critic-based experience, in combination of certain choices of the divergence measure $\mathbb{D}$, produces the popular generative adversarial learning (GANs, Goodfellow et al., 2014).

Similarly, as briefly mentioned earlier, many of the above conventional experiences can also benefit from the idea of introducing adaptive or learnable components, e.g., data instances with automatically induced data weights or learned augmentation policies. We discuss in Part III several practical algorithms developed based on the standard equation.

2.4 The Divergence Measure

We now turn to the divergence term $\mathbb{D}(q, p_\theta)$ in the standard equation. The discussions in the prior section have focused on the specific cases of $\mathbb{D}$ being the cross entropy (Eq.2.29), yet there is a rather rich set of choices for the divergence measure, such as f-divergence (e.g., KL divergence, Jensen-Shannon divergence), optimal transport distance (e.g., Wasserstein distance), etc.

To see a concrete setting of how the divergence measure may influence the learning, consider the experience to be data instances with the data distribution $p_d(\boldsymbol{t})$, and, following the configurations of supervised MLE (section 2.3.1.1), set $f = f_{\text{data}}$, $\alpha = 1$ and $\beta = \epsilon$. As a result, the solution of q in the standard equation Eq.(2.28) reduces to the data distribution $q(\boldsymbol{t}) = p_d(\boldsymbol{t})$ (assuming the uncertainty measure $\mathbb{H}$ to be the Shannon entropy). The learning of model thus reduces to minimizing the divergence between the model and data distributions:

$$\min_{\boldsymbol{\theta}} \mathbb{D}(p_d, p_\theta), \tag{2.49}$$

which is a commonly seen objective shared by many ML algorithms depending on how the divergence measure is specialized. Thus, in this concrete setting the divergence measure directly determines the learning problem. Below we will see richer influences of $\mathbb{D}$ in other settings in combination with other standard equation components.

By considering certain choices for the divergence measure, we open up the door to recover and generalize some core techniques that are widely practiced in the area of generative adversarial learning. The standard equation offers two different ways of deriving the generative adversarial learning algorithms, where the key concept, discriminator, arises either as an approximation to the optimization procedure, or as part of the experience in the learning objective.

2.4.1 Generative Adversarial Learning

The functional descent view Given a specialized divergence $\mathbb{D}$, the objective Eq.(2.49) can be optimized with different solvers, among which *probability functional descent* (PFD) (Chu et al., 2019) offers an elegant way to derive the optimization which recovers various existing algorithms in GANs (Goodfellow et al., 2014). We give a brief overview here and discuss more details of the optimization techniques in the next section.

Specifically, given p_d, $J(p) := \mathbb{D}(p_d, p)$ is a functional on the distribution space $\mathcal{P}(\mathcal{T})$. The Gâteaux derivative of J at p, if exists, is defined as (Fernholz, 2012):

$$J'_p(h-p) = \lim_{\epsilon \to 0^+} \frac{J(p + \epsilon(h-p)) - J(p)}{\epsilon} \tag{2.50}$$

for any given $h \in \mathcal{P}(\mathcal{T})$. Intuitively, $J'_p(h-p)$ describes the change of the $J(p)$ value with respect to an infinitesimal change in p in the direction of $(h-p)$. The Gâteaux derivative $J'_p(h-p)$ can alternatively be computed with the *influence function* of J at p, denoted as $\psi_p : \mathcal{T} \to \mathbb{R}$, through:

$$J'_p(h-p) = \int_{\boldsymbol{t}} \psi_p(\boldsymbol{t})(h-p) d\boldsymbol{t} = \mathbb{E}_h \left[\psi_p(\boldsymbol{t})\right] - \mathbb{E}_p \left[\psi_p(\boldsymbol{t})\right]. \tag{2.51}$$

The above notions allow us to define gradient descent applied to the functional J. Concretely, we can define a linear approximation to $J(p)$ around a given p_0:

$$\begin{aligned} J(p) &\approx J(p_0) + J'_{p_0}(p - p_0) \\ &= J(p_0) + \mathbb{E}_p \left[\psi_{p_0}(\boldsymbol{t})\right] - \mathbb{E}_{p_0} \left[\psi_{p_0}(\boldsymbol{t})\right] \\ &= \mathbb{E}_p \left[\psi_{p_0}(\boldsymbol{t})\right] + const. \end{aligned} \tag{2.52}$$

Thus, $J(p)$ can approximately be minimized with an iterative descent procedure: at each iteration n, we perform a descent step that decreases $\mathbb{E}_p \left[\psi_{p^{(n)}}(\boldsymbol{t})\right]$ w.r.t p, yielding $p^{(n+1)}$ for the next iteration. Note that, in practice where the model distribution of interest p is often parameterized as p_θ, chain rule can be applied assuming mild regularity conditions to compute the gradient $\nabla_\theta J(p_\theta) \approx \nabla_\theta \mathbb{E}_{p_\theta}[\psi_{p_{\theta^{(n)}}}(\boldsymbol{t})]$.

Once the functional gradient is defined as above, the remaining problem of the optimization is then about how to obtain the influence function ψ_p given the functional $J(p)$. In some cases the influence function as defined in Eq.(2.51) is not directly tractable and approximations are needed. Chu et al. (2019) developed a variational approximation method applied when J is convex. Concretely, with the convex conjugate of J defined as $J^*(\varphi) = \sup_h \mathbb{E}_h [\varphi(\boldsymbol{t})] - J(h)$, it can be shown under mild conditions that the influence function for J at p is:

$$\psi_p = \mathrm{argmax}_{\varphi \in \mathcal{C}(\mathcal{T})} \mathbb{E}_p [\varphi(\boldsymbol{t})] - J^*(\varphi), \tag{2.53}$$

where $\mathcal{C}(\mathcal{T})$ is the space of continuous functions $\mathcal{T} \to \mathbb{R}$. We thus can approximate the influence function by parameterizing it as a neural network and training the network to maximize the objective $\mathbb{E}_p [\varphi(\boldsymbol{t})] - J^*(\varphi)$. Plugging the approximation of influence function into the above functional descent procedure leads to the full PFD optimization:

$$\inf_p \sup_\varphi \mathbb{E}_p [\varphi(\boldsymbol{t})] - J^*(\varphi). \tag{2.54}$$

The saddle-point problem is reminiscent of the generative adversarial learning. In fact, as shown in (Chu et al., 2019), when $J(p) = \mathbb{D}(p_d, p)$ is the Jensen-Shannon divergence and φ is parameterized as $\varphi_\phi = \frac{1}{2}\log(1 - C_\phi) - \frac{1}{2}\log 2$ where C_ϕ is a binary classifier (discriminator), then Eq.(2.54) recovers the original GAN algorithm (Goodfellow et al., 2014):

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \frac{1}{2}\mathbb{E}_{p_d} [\log C_\phi(\boldsymbol{t})] - \frac{1}{2}\mathbb{E}_{p_\theta} [\log(1 - C_\phi(\boldsymbol{t}))] . \tag{2.55}$$

On the other hand, consider the case of Wasserstein GAN algorithm (Arjovsky et al., 2017b) where $\mathbb{D}(p_d, p)$ is set to the first-order Wasserstein distance $W_1(p, p_d)$:

$$\inf_p J(p) := \inf_p W_1(p, p_d) = \inf_p \sup_{\|\varphi\|_L \leq 1} \mathbb{E}_p [\varphi(\boldsymbol{t})] - \mathbb{E}_{p_d} [\varphi(\boldsymbol{t})] , \tag{2.56}$$

where the last equation is due to the Kantorovich duality (Santambrogio, 2015, Section 1.2) and $\|\varphi\|_L \leq 1$ is the constraint of φ being a 1-Lipschitz function. We can also interpret Eq.(2.56) using the PFD procedure. Specifically, due to the Fenchel–Moreau theorem saying that $J(p) = \sup_\varphi \mathbb{E}_p [\varphi(\boldsymbol{t})] - J^*(\varphi)$, we have $J^*(\varphi) = \mathbb{E}_{p_d} [\varphi(\boldsymbol{t})] + \mathbb{I}_\infty(\|\varphi\|_L \leq 1)$, where $\mathbb{I}_\infty(A)$ is an indicator function that equals 1 if A is true and ∞ otherwise, and the influence function (Eq.2.53) corresponds to the Kantorovich potential for the transport from p to p_d (Santambrogio, 2015, Proposition 7.17).

The functional descent view of generative adversarial learning is based on the treatment that the experience is the given data instances, so the various GAN algorithms are due to the different choices of the divergence measure. The standard equation also allows an alternative viewpoint of the learning paradigm, that gives more flexibility in not only choosing the divergence measure but also the experience function, leading to a richer set of GAN extensions.

The variational experience view In the alternative viewpoint, we consider experience that is variationally defined as discussed in section 2.3.4. That is, the experience function f, as a measure of the goodness of a sample $\boldsymbol{t}$, is not specified *a priori* but rather defined through an optimization problem. A concrete example is to define f as a binary classifier f_ϕ with sigmoid activation and parameters $\boldsymbol{\phi}$, where the value $f_\phi(\boldsymbol{t})$ measures the *log* probability of the sample $\boldsymbol{t}$ being a real instance (as opposed to a model generation). Thus the higher $f_\phi(\boldsymbol{t})$ value, the higher quality of sample $\boldsymbol{t}$. The parameters $\boldsymbol{\phi}$ of the experience need to be learned. We can do so by augmenting the standard equation (Eq.2.28) with added optimization of $\boldsymbol{\phi}$ in various ways. The following equation gives one of the approaches:

$$\min_{q,\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \ -\alpha\mathbb{H}(q) + \beta\mathbb{D}(q, p_\theta) - \mathbb{E}_q[f_\phi] + \mathbb{E}_{p_d}[f_\phi], \tag{2.57}$$

where, besides the optimization of q and $\boldsymbol{\theta}$, we additionally maximize over $\boldsymbol{\phi}$ with the extra term $\mathbb{E}_{p_d}[f_\phi]$ to form the classification problem $\max_{\boldsymbol{\phi}} -\mathbb{E}_q[f_\phi] + \mathbb{E}_{p_d}[f_\phi]$. Further assuming a particular configuration of the tradeoff hyper-parameters $\alpha = 0$ and $\beta = 1$, the resulting objective

$$\min_{q,\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \ \mathbb{D}(q, p_\theta) - \mathbb{E}_q[f_\phi] + \mathbb{E}_{p_d}[f_\phi], \tag{2.58}$$

turns out to relate closely to generative adversarial learning.

In particular, with proofs adapted from (Farnia and Tse, 2018), when $\mathbb{D}$ is the Jensen-Shannon divergence and assuming the space of f_ϕ, denoted as $\mathcal{F}$, is convex, then Eq.(2.58) recovers the vanilla GAN algorithm. More specifically, if we denote the probability $C_\phi(\boldsymbol{t}) = \exp f_\phi(\boldsymbol{t})$, then the equation will reduce to the familiar GAN objective in Eq.(2.55). The results can be extended to the more general case of f-GAN (Nowozin et al., 2016): if we set $\mathbb{D}$ to an f-divergence and do not restrict the form (e.g., classifier) of the experience function f_ϕ, then with mild conditions, the equation recovers the f-GAN algorithm. Now consider $\mathbb{D}$ is the first-order Wasserstein distance and suppose the f_ϕ-space $\mathcal{F}$ is a convex subset of 1-Lipschitz functions. It can be shown that Eq.(2.58) reduces to the Wasserstein GAN algorithm as shown in Eq.(2.56) where φ now corresponds to f_ϕ. Note that for the above configurations, if f_ϕ is parameterized as a neural network with a fixed architecture (e.g., ConvNet), its space $\mathcal{F}$ is not necessarily convex (i.e., a linear combination of two neural networks in $\mathcal{F}$ is not necessarily in $\mathcal{F}$). In such cases we formulate the optimization of the experience function over $\mathrm{conv}(\mathcal{F})$, the convex hull of $\mathcal{F}$ containing any convex combination of neural network functions in $\mathcal{F}$ (Farnia and Tse, 2018), and see the various GAN algorithms as approximations by considering only the subset $\mathcal{F} \subseteq \mathrm{conv}(\mathcal{F})$.

The formulation in Eq.(2.58) naturally allows us to consider more options for the divergence measure $\mathbb{D}$, such as the hybrid f-divergence and Wasserstein distance studied in (Farnia and Tse, 2018). Of particular interest is to set $\mathbb{D}$ to the KL divergence $\mathbb{D}(q, p_\theta) = \mathrm{KL}(q \| p_\theta)$, motivated by the simplicity in the sense that the auxiliary distribution q has a closed-form solution:[1] at each iteration n,

$$q^{(n+1)}(\boldsymbol{t}) = p_{\theta^{(n)}}(\boldsymbol{t}) \exp\left\{ f_{\phi^{(n)}}(\boldsymbol{t}) \right\} \ / \ Z, \tag{2.59}$$

where Z is the normalization factor. As studied in Wu et al. (2020), the particular form of solution results in a new variant of GANs that enables more stable optimization of both the experience function (discriminator) f_ϕ and the model (generator) p_θ. Concretely, the discriminator f_ϕ can

[1] We can alternatively derive the objective from Eq.(2.57), by setting $\alpha = \beta = 1, \mathbb{D}$ to the cross entropy, and $\mathbb{H}$ to the Shannon entropy. Note that $\mathrm{KL}(q \| p_\theta) = -\mathrm{H}(q) - \mathbb{E}_q[\log p_\theta]$. Thus the solution of q can be derived as in Eq.(2.30).

now be optimized with importance re-weighting:

$$\begin{aligned}&\max_{\boldsymbol{\phi}} - \mathbb{E}_{\boldsymbol{t}\sim q^{(n+1)}}\left[f_{\phi}(\boldsymbol{t})\right] + \mathbb{E}_{\boldsymbol{t}\sim p_d}\left[f_{\phi}(\boldsymbol{t})\right] \\ &= -\frac{1}{Z}\mathbb{E}_{\boldsymbol{t}\sim p_{\theta}^{(n)}}\Big[\exp\{f_{\phi^{(n)}}(\boldsymbol{t})\}\cdot f_{\phi}(\boldsymbol{t})\Big] + \mathbb{E}_{\boldsymbol{t}\sim p_d}\left[f_{\phi}(\boldsymbol{t})\right],\end{aligned} \tag{2.60}$$

where importance sampling is used to estimate the expectation under $q^{(n+1)}$, using the generator $p_\theta^{(n)}$ as the proposal distribution. Compared to the vanilla and Wasserstein GANs above, the fake samples from the generator are now weighted by the exponentiated discriminator score $\exp\{f_{\phi^{(n)}}(\boldsymbol{t})\}$ when used to update the discriminator. Intuitively, the mechanism assigns higher weights to samples that can fool the discriminator better, while low-quality samples are downplayed to avoid destructing the discriminator performance during training.

As for the generator p_θ, training is normally done by just minimizing $\mathrm{KL}(q^{(n+1)}\|p_\theta)$. However, the generator in GANs is often an *implicit* distribution that does not permit the evaluation of likelihood, rendering the KL objective (which involves evaluating the likelihood of samples from $q^{(n+1)}$) not applicable. Wu et al. (2020) used a simple approximation similar to the wake-sleep algorithm (section 2.1.1.2) by minimizing the KL divergence in opposite direction:

$$\min_{\boldsymbol{\theta}} \mathrm{KL}\left(p_\theta(\boldsymbol{t})\|q^{(n+1)}(\boldsymbol{t})\right) = -\mathbb{E}_{\boldsymbol{t}\sim p_\theta(\boldsymbol{t})}\left[f_\phi(\boldsymbol{t})\right] + \mathrm{KL}\left(p_\theta(\boldsymbol{t})\|p_{\theta^{(n)}}(\boldsymbol{t})\right). \tag{2.61}$$

The first term on the right-hand side is just the conventional update to the generator in GANs. The more interesting part is the second term, a new KL regularizer between the generator p_θ and its "old" state $p_\theta^{(n)}$ from the previous iteration. The regularizer discourages the generator from changing too much between updates, which is shown to be useful to stabilize the stochastic optimization procedure. Besides, the regularization is highly reminiscent of that of Trust-Region Policy Optimization (TRPO, Schulman et al., 2015b) and Proximal Policy Optimization (PPO, Schulman et al., 2017c) in reinforcement learning, where a similar KL regularizer is imposed to prevent uncontrolled policy updates and make policy gradient robust to noises. Based on the connection, Wu et al. (2020) imported from PPO a clipped surrogate objective as an efficient approximation to the regularized updates.

Table 2.1 summarizes example specifications of the components in the standard equation and the corresponding known algorithms.

2.5 Applications

The preceding sections have presented a standardized formalism of machine learning, on the basis of the standard equation of objective function, that provides a succinct, structured formulation of a broad design space of learning algorithms, and subsumes a wide range of known algorithms in a unified manner. The simplicity, modularity and generality of the framework is particularly appealing not only from the theoretical perspective, but also because it offers guiding principles for designing algorithmic solutions to challenging problems in a mechanic way. The PPO-GAN described in section 2.4.1 is one of the instances illustrating how new, more effective algorithms

Experience type	Experience function f	Divergence $\mathbb{D}$	α	β	Algorithm
Data instances	$f_{\text{data}}(\boldsymbol{x}; \mathcal{D})$	CE	1	1	Unsupervised MLE
	$f_{\text{data}}(\boldsymbol{x}, \boldsymbol{y}; \mathcal{D})$	CE	1	ϵ	Supervised MLE
	$f_{\text{data-w}}(\boldsymbol{t}; \mathcal{D})$	CE	1	ϵ	Data re-weighting
	$f_{\text{data-aug}}(\boldsymbol{t}; \mathcal{D})$	CE	1	ϵ	Data Augmentation
	$f_{\text{active}}(\boldsymbol{x}, \boldsymbol{y}; \mathcal{D})$	CE	1	ϵ	Active Learning (Ertekin et al., 2007)
Knowledge	$f_{rule}(\boldsymbol{x}, \boldsymbol{y})$	CE	1	1	Posterior Regularization (Ganchev et al., 2010)
	$f_{rule}(\boldsymbol{x}, \boldsymbol{y})$	CE	$\mathbb{R}$	1	Unified EM (Samdani et al., 2012)
Reward	$\log Q^{(n)}(\boldsymbol{x}, \boldsymbol{y})$	CE	1	1	Policy Gradient
	$\log Q^{(n)}(\boldsymbol{x}, \boldsymbol{y}) + Q^{in,(n)}(\boldsymbol{x}, \boldsymbol{y})$	CE	1	1	+ Intrinsic Reward
	$\log Q^{(n)}(\boldsymbol{x}, \boldsymbol{y})$	CE	$\tau > 0$	$\tau > 0$	RL as Inference
Other advanced	binary classifier	JSD	0	1	Vanilla GAN (Goodfellow et al., 2014)
	discriminator	f-divg.	0	1	f-GAN (Nowozin et al., 2016)
	1-Lipschitz discriminator	W_1 dist.	0	1	WGAN (Arjovsky et al., 2017b)
	1-Lipschitz discriminator	KL	0	1	PPO-GAN (Wu et al., 2020)

Table 2.1: Example configurations of the components in the standard equation (Eq.2.28) which recover existing algorithms. Here, "CE" means Cross Entropy; "JSD" is the Jensen-Shannon divergence; "f-divg" is the f-divergence; "W_1 dist." is the first-order Wasserstein distance; and "KL" is the KL divergence. Refer to sections 2.3 and 2.4 for more details.

can emerge from the standard framework with simple specifications of its components. In this section, we discuss the high-level ideas of using the standard equation to drive systematic design of new learning methods, which in turn yield various algorithmic solutions to problems in different application domains.

The discussion in this section is brief. Parts II and III of the thesis are devoted to in-depth studies of the ideas discussed here.

2.5.1 Formulating and Combining Rich Experiences

As one of the original motivations for the standardization, the framework allows us to combine together all different experiences to learn models of interest. Learning with multiple types of experiences is a necessary capability of AI agents to be deployed in a real dynamic environment and to improve from heterogeneous signals in different forms, at varying granularities, from diverse sources, and with different intents (e.g., adversaries). Such versatile learning capability is best exemplified by human learning, which has the hallmark of making use of any available sources of information. Take the example of learning a language. Humans can benefit from observing examples through reading and hearing, studying abstract definitions and grammar, making mistakes and getting correction from teacher, interacting with others and observing implicit feedback, etc. Knowledge of prior language can also accelerate the acquisition of new one. To build a similar panoramic learning AI agent, having a standardized learning formalism is perhaps an indispensable step.

The standard equation we have presented is naturally designed to fit the needs. Specifically, the experience function provides a straightforward vehicle for experience combination. The simplest approach is perhaps to make a weighted composition of multiple individual experience functions:

$$f(\boldsymbol{t}) = \sum\nolimits_i \lambda_i f_i(\boldsymbol{t}), \tag{2.62}$$

where each f_i characterizes a specific source of experiences, and $\lambda_i > 0$ is the respective weight. Thus one can readily plug in arbitrary available experiences, such as data, logical rules, constraints, and auxiliary models, as components of the experience function.

Designing a solution to a problem thus boils down to choosing and formulating *what* experiences to use depending on the problem structure and available resources, without worrying too much about *how* to use the experiences. Section 2.3 discussed possible formulations of the diverse types of experiences as an experience function to be plugged into Eq.(2.62). Yet it is still an open question how even more types of experiences, such as massive external knowledge graphs, can efficiently be formulated as an experience function that assesses the "goodness" of a configuration $\boldsymbol{t}$. The discussions in the next section offers new opportunities that allow users to not have to manually specify every details of the experience function, but instead only to define parts of it the users are certain of, and leave the remaining parts (plus the experience weights λ_i in Eq.2.62) automatically learned jointly with the target model.

2.5.2 Repurposing Learning Algorithms for New Problems

The standardized formalism sheds new lights on fundamental relationships between a number of algorithms that were each originally designed to deal with a particular type of experience. The unified perspective also shows that many of the learning problems in seemingly distinct contexts are essentially the same and just correspond to specialized variations of the standard equation components. This opens up a wide range of opportunities for extending the use of algorithms on broad sets of new problems, and for exchange between the diverse research areas in aspects of modeling, approximation and optimization tools, and theoretical understanding. Thus an earlier successful solution to challenges in one problem can now be readily applied to address challenges in another, and a future progress made in one area could potentially immediately unlock progresses in many others.

As an illustration, let us consider a set of concrete problems that might seem unrelated at first sight and were often studied by researchers in different domains. The first problem is about integrating structured knowledge constraints with deep generative models, where some components of the knowledge, and/or the confidence weight of each of the knowledge constraints, cannot be fully specified *a priori*. The second problem is in the supervised learning regime, where only a small set of data instances with imbalanced labels is available, and we want to automatically manipulate the data (e.g., through augmentation and re-weighting) to maximize the training performance. The last problem is to stabilize the notoriously difficult training of GANs for a wide range of image and text generation tasks. Under the unified ML perspective, these problems, though concerning with distinctly different experiences, are all really the same problem, corresponding to the contexts where the experience functions in the standard equation are imperfect and need to be adapted jointly with the target models. Instead of having to invent new algorithms to address the general problem, we can take advantage of the revealed connections between research areas, and import known algorithms from areas addressing similar problems as effective solutions. We defer more details of the results to Part III.

2.6 Discussion

We have presented a standardized ML formalism instantiated as the standard equation of the objective function. We showed that the standard equation formulates a large algorithmic space which encompasses a wide range of existing algorithms as special instances due to different choices of the experience function, divergence, trade-off hyperparameters, and other components.

Despite its generality, the standard equation is not meant to be a "universal" formulation that covers all algorithms in machine learning. For example, we have not yet studied its connection with some of the major learning paradigms such as meta learning, online learning, and lifelong learning. For these kinds of learning that happen in a "dynamic" environment, where the tasks or data distributions are changing over time, sometimes with unknown dynamics, we may not be able to describe the learning with a "static" objective function (i.e., assuming a static environment) at the first place. Thus extensions to the standard equation are perhaps necessary in order to bridge the gap, which highlights an very exciting future research venue.

Besides the objective function, the other two core ingredients of creating an ML solution, as mentioned at the beginning of the chapter, are the model architecture and the optimization solver. The current research community has made extensive studies on model architecture, ranging from graphical models to neural networks to compositional architectures. The choice of model architecture often heavily depends on the problem at hand given the input/output formats and the beliefs/heuristics we have regarding the problem structure, though some common building blocks do appear across many different problems. In Chapter 3, we discuss in more details the different choices of baking knowledge into model architectures versus encoding knowledge in experiences (and thus the objective functions) and conveying to model through training. In Chapter 8, we present our efforts of designing and developing a comprehensive catalog of composable building blocks, that allows users to compose arbitrary model architectures in a reusable and repeatable way.

Regarding the optimization solver, our discussions in this chapter have involved several optimization algorithms ranging from the simple stochastic gradient descent, to the alternating projection algorithms including IPF, EM, and variational EM, to PFD for gradient descent in the functional space. Each of those algorithms is applicable to solve the standard equation problem in certain special cases (e.g., the EM procedure in Eq.2.30, in the cases of cross entropy as divergence). However, there seems no general formulation for the optimization solver, on par with the standard equation for the objective function, to unify and generalize the optimization algorithms, and to solve the standard equation problem in the general case. In each of the subsequent chapters, we will describe the optimization procedure specific to the respective problem setting.

Finally, the general standardized formalism poses new open questions about theoretical analysis of the learning problems it formulates. For example, what are the convergence guarantees, complexity, robustness, and other theoretical and statistical properties, when the experience function is instantiated in different forms such as augmented data instances, logic rules, or auxiliary models? What are the effects of configurations of other components in the equation? We discuss more on this topic in the last chapter of the thesis.

Part II

Applications (1): Learning with All Experiences

Chapter 3

Learning Deep Neural Networks with Logic Rules

An indispensable capability of AI agents learning from all experiences is to make sense of abstract structured knowledge. This chapter develops a general algorithm, as a member in the standard equation space, that uses knowledge expressed as first-order logic rules to learn models of interest, in particular deep neural networks of arbitrary architectures.

Training a deep neural network often demands large amounts of high-quality data, and can produce brittle results in the presence of complex concepts or novel circumstances beyond training set. On the other hand, structured knowledge (e.g., rules, laws of physics, relations, functions with explicit semantics) encodes valuable information of the regularities underlying the problems. It is much needed to exploit the rich problem structures and domain knowledge for efficient and robust learning. Previous efforts usually focus on baking inductive bias into model architectures, which however are often dedicated to specific tasks, models, and types of knowledge. It is desirable to have a more general approach, that is agnostic to model architectures and applicable to various problems such as classification, structured prediction, and generation, to seamlessly integrate high-level symbolic knowledge into the low-level distributed representation models.

In this chapter, we develop such an algorithm that incorporates first-order logical knowledge in learning. First-order logic rules provide a flexible declarative language for communicating abstract knowledge and conveying human goals and heuristics. Building upon the standard equation presented in Chapter 2, the algorithm adopts a soft formulation of logic rules as the experience, and iteratively instills the information from the logic rules into the neural parameters through a teacher-student training mechanism. The approach generalizes the previous knowledge constrained methods, like posterior regularization (Ganchev et al., 2010) and constraint-driven learning (Chang et al., 2012), in terms of the applicable forms of both target models and knowledge. We apply the approach on natural language processing tasks, including sentiment classification and named entity recognition, with several linguistic rules, and show superior performance.

We discuss the different general ways of integrating structured knowledge with statistical models at the end of the chapter. Chapter 6 further studies an extended problem where the structured knowledge itself (including its confidence level) needs to be induced/adapted during training the model.

3.1 Introduction

Deep neural networks (DNNs) provide a powerful mechanism for learning patterns from massive data, achieving new levels of performance on image classification (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), machine translation (Bahdanau et al., 2014), playing strategic board games (Silver et al., 2016), and so forth.

Despite the impressive advances, the widely-used DNN methods still have limitations. The high predictive accuracy has heavily relied on large amounts of labeled data; and the purely data-driven learning can lead to uninterpretable and sometimes counter-intuitive results (Szegedy et al., 2014; Nguyen et al., 2015). It is also difficult to encode human intention to guide the models to capture desired patterns, without expensive direct supervision or ad-hoc initialization.

On the other hand, the cognitive process of human beings have indicated that people learn not only from concrete examples (as DNNs do) but also from different forms of general knowledge and rich experiences (Minksy, 1980; Lake et al., 2015). *Logic rules* provide a flexible declarative language for communicating high-level cognition and expressing structured knowledge. It is therefore desirable to integrate logic rules into DNNs, to transfer human intention and domain knowledge to neural models, and regulate the learning process.

In this work, we present a framework capable of enhancing general types of neural networks, such as convolutional networks (CNNs) and recurrent networks (RNNs), on various tasks, with logic rule knowledge. Combining symbolic representations with neural methods have been considered in different contexts. Neural-symbolic systems (Garcez et al., 2012) construct a network from a given rule set to execute reasoning. To exploit *a priori* knowledge in general neural architectures, recent work augments each raw data instance with useful features (Collobert et al., 2011), while network training, however, is still limited to instance-label supervision and suffers from the same issues mentioned above. Besides, a large variety of structural knowledge cannot be naturally encoded in the feature-label form.

Our framework enables a neural network to learn simultaneously from labeled instances as well as logic rules, through an *iterative rule knowledge distillation* procedure that transfers the structured information encoded in the logic rules into the network parameters. Since the general logic rules are complementary to the specific data labels, a natural "side-product" of the integration is the support for semi-supervised learning where unlabeled data is used to better absorb the logical knowledge. Methodologically, our approach can be seen as a combination of the knowledge distillation (Hinton et al., 2015) and the posterior regularization (PR) method (Ganchev et al., 2010). In particular, at each iteration we adapt the posterior constraint principle from PR to construct a rule-regularized *teacher*, and train the *student* network of interest to imitate the predictions of the teacher network. We leverage soft logic to support flexible rule encoding.

We apply the proposed framework on both CNN and RNN, and deploy on the task of sentiment analysis (SA) and named entity recognition (NER), respectively. With only a few (one or two) very intuitive rules, both the distilled networks and the joint teacher networks strongly improve over their basic forms (without rules), and achieve better or comparable performance to state-of-the-art models which typically have more parameters and complicated architectures.

To the best of our knowledge, this is the first work to integrate logic rules with general workhorse types of deep neural networks in a principled framework. The encouraging results indicate our method can be potentially useful for incorporating richer types of human knowledge, and improving other application domains.

3.2 Related Work

Combination of logic rules and neural networks has been considered in different contexts. Neural-symbolic systems (Garcez et al., 2012), such as KBANN (Towell et al., 1990) and CILP++ (França et al., 2014), construct network architectures from given rules to perform reasoning and knowledge acquisition. A related line of research, such as Markov logic networks (Richardson and Domingos, 2006), derives probabilistic graphical models (rather than neural networks) from the rule set.

With the recent success of deep neural networks in a vast variety of application domains, it is increasingly desirable to incorporate structured logic knowledge into general types of networks to harness flexibility and reduce uninterpretability. Recent work that trains on extra features from domain knowledge (Collobert et al., 2011), while producing improved results, does not go beyond the data-label paradigm. Kulkarni et al. (2015) uses a specialized training procedure with careful ordering of training instances to obtain an interpretable neural layer of an image network. Karaletsos et al. (2016) develops a generative model jointly over data-labels and similarity knowledge expressed in triplet format to learn improved disentangled representations.

Though there do exist general frameworks that allow encoding various structured constraints on latent variable models (Ganchev et al., 2010; Zhu et al., 2014; Liang et al., 2009), they either are not directly applicable to the NN case, or could yield inferior performance as in our empirical study. Liang et al. (2008) transfers predictive power of pre-trained structured models to unstructured ones in a pipelined fashion.

Our proposed approach is distinct in that we use an iterative rule distillation process to effectively transfer rich structured knowledge, expressed in the declarative first-order logic language, into parameters of general neural networks. We show that the proposed approach strongly outperforms an extensive array of other either ad-hoc or general integration methods.

3.3 Method

In this section we present our framework which encapsulates the logical structured knowledge into a neural network. This is achieved by forcing the network to emulate the predictions of a rule-regularized teacher, and evolving both models iteratively throughout training (section 3.3.2).

The process is agnostic to the network architecture, and thus applicable to general types of neural models including CNNs and RNNs. We construct the teacher network in each iteration by adapting the posterior regularization principle in our logical constraint setting (section 3.3.3), where our formulation provides a closed-form solution. Figure 3.1 shows an overview of the proposed framework.

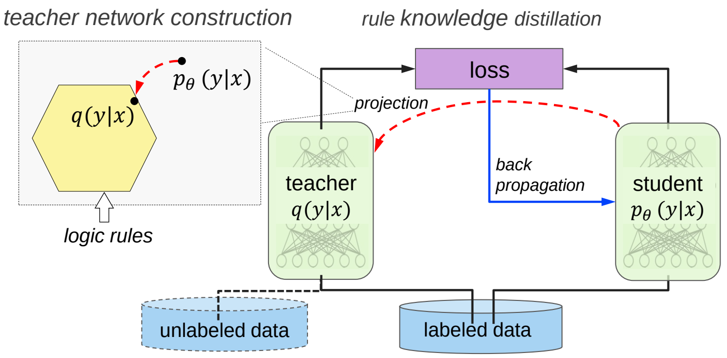


Figure 3.1: Framework Overview. At each iteration, the teacher network is obtained by projecting the student network to a rule-regularized subspace (red dashed arrow); and the student network is updated to balance between emulating the teacher's output and predicting the true labels (black/blue solid arrows).

3.3.1 Experiences: Instances and Rules

Our approach allows neural networks to learn from both specific examples and general rules. Here we give the settings of these resources for learning.

Assume we have input variable $\boldsymbol{x} \in \mathcal{X}$ and target variable $\boldsymbol{y} \in \mathcal{Y}$. For clarity, we focus on K-way classification, where $\mathcal{Y} = \Delta^K$ is the K-dimensional probability simplex and $\boldsymbol{y} \in \{0,1\}^K \subset \mathcal{Y}$ is a one-hot encoding of the class label. However, our method specification can straightforwardly be applied to other contexts such as regression and sequence learning (e.g., NER tagging, which is a sequence of classification decisions). The training data $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$ is a set of instantiations of $(\boldsymbol{x}, \boldsymbol{y})$.

Further consider a set of first-order logic (FOL) rules with confidences, denoted as $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^L$, where R_l is the lth rule over the input-target space $(\mathcal{X}, \mathcal{Y})$, and $\lambda_l \in [0, \infty]$ is the confidence level with $\lambda_l = \infty$ indicating a hard rule, i.e., all groundings are required to be true (=1). Here a grounding is the logic expression with all variables being instantiated. Given a set of examples $(\boldsymbol{X}, \boldsymbol{Y}) \subset (\mathcal{X}, \mathcal{Y})$ (e.g., a minibatch from $\mathcal{D}$), the set of groundings of R_l are denoted as $\{r_{lg}(\boldsymbol{X}, \boldsymbol{Y})\}_{g=1}^{G_l}$. In practice a rule grounding is typically relevant to only a single or subset of examples, though here we give the most general form on the entire set.

We encode the FOL rules using soft logic (Bach et al., 2015) for flexible encoding and stable optimization. Specifically, soft logic allows continuous truth values from the interval $[0, 1]$ instead of $\{0,1\}$, and the Boolean logic operators are reformulated as:

$$\begin{aligned}
&A\&B = \max\{A+B-1, 0\} \\
&A \vee B = \min\{A+B, 1\} \\
&A_1 \wedge \cdots \wedge A_N = \sum\nolimits_i A_i/N \\
&\neg A = 1 - A
\end{aligned} \tag{3.1}$$

Here & and $\wedge$ are two different approximations to logical conjunction (Foulds et al., 2015): & is useful as a selection operator (e.g., $A\&B = B$ when $A = 1$, and $A\&B = 0$ when $A = 0$), while $\wedge$ is an averaging operator.

3.3.2 Rule Knowledge Distillation

A neural network defines a conditional probability $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ by using a softmax output layer that produces a K-dimensional soft prediction vector denoted as $\boldsymbol{\sigma}_\theta(\boldsymbol{x})$. The network is parameterized by weights $\boldsymbol{\theta}$. Standard neural network training has been to iteratively update $\boldsymbol{\theta}$ to produce the correct labels of training instances. To integrate the information encoded in the rules, we propose to train the network to also imitate the outputs of a rule-regularized projection of $p_\theta(\boldsymbol{y}|\boldsymbol{x})$, denoted as $q(\boldsymbol{y}|\boldsymbol{x})$, which explicitly includes rule constraints as regularization terms. In each iteration q is constructed by projecting p_θ into a subspace constrained by the rules, and thus has desirable properties. We present the construction in the next section. The prediction behavior of q reveals the information of the regularized subspace and structured rules. Emulating the q outputs serves to transfer this knowledge into p_θ. The new objective is then formulated as a balancing between imitating the soft predictions of q and predicting the true hard labels:

$$\begin{aligned}
\boldsymbol{\theta}^{(t+1)} = \arg\min_{\theta\in\Theta} \frac{1}{N}\sum_{n=1}^{N} &(1-\pi)\ell(\boldsymbol{y}_n, \boldsymbol{\sigma}_\theta(\boldsymbol{x}_n)) \\
&+ \pi\ell(\boldsymbol{s}_n^{(t)}, \boldsymbol{\sigma}_\theta(\boldsymbol{x}_n)),
\end{aligned} \tag{3.2}$$

where ℓ denotes the loss function selected according to specific applications (e.g., the cross entropy loss for classification); $\boldsymbol{s}_n^{(t)}$ is the soft prediction vector of q on $\boldsymbol{x}_n$ at iteration t; and π is the imitation parameter calibrating the relative importance of the two objectives.

A similar imitation procedure has been used in other settings such as model compression (Hinton et al., 2015) where the process is termed *distillation*. Following them we call $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ the "student" and $q(\boldsymbol{y}|\boldsymbol{x})$ the "teacher", which can be intuitively explained in analogous to human education where a teacher is aware of systematic general rules and she instructs students by providing her solutions to particular questions (i.e., the soft predictions). An important difference from previous distillation work, where the teacher is obtained beforehand and the student is trained thereafter, is that our teacher and student are learned simultaneously during training.

Though it is possible to combine a neural network with rule constraints by projecting the network to the rule-regularized subspace after it is fully trained as before with only data-label instances, or by optimizing projected network directly, we found our iterative teacher-student distillation approach provides a much superior performance, as shown in the experiments. Moreover, since p_θ

distills the rule information into the weights $\boldsymbol{\theta}$ instead of relying on explicit rule representations, we can use p_θ for predicting new examples at test time when the rule assessment is expensive or even unavailable (i.e., the *privileged information* setting (Lopez-Paz et al., 2015)) while still enjoying the benefit of integration. Besides, the second loss term in Eq.(3.2) can be augmented with rich unlabeled data in addition to the labeled examples, which enables *semi-supervised* learning for better absorbing the rule knowledge.

3.3.3 Teacher Network Construction

We now proceed to construct the teacher network $q(\boldsymbol{y}|\boldsymbol{x})$ at each iteration from $p_\theta(\boldsymbol{y}|\boldsymbol{x})$. The iteration index t is omitted for clarity. We adapt the posterior regularization principle in our logic constraint setting. Our formulation ensures a closed-form solution for q and thus avoids any significant increases in computational overhead.

Recall the set of FOL rules $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^{L}$. Our goal is to find the optimal q that fits the rules while at the same time staying close to p_θ. For the first property, we apply a commonly-used strategy that imposes the rule constraints on q through an expectation operator. That is, for each rule (indexed by l) and each of its groundings (indexed by g) on $(\boldsymbol{X}, \boldsymbol{Y})$, we expect $\mathbb{E}_{q(\boldsymbol{Y}|\boldsymbol{X})}[r_{lg}(\boldsymbol{X}, \boldsymbol{Y})] = 1$, with confidence λ_l. The constraints define a rule-regularized space of all valid distributions. For the second property, we measure the closeness between q and p_θ with KL-divergence, and wish to minimize it. Combining the two factors together and further allowing slackness for the constraints, we finally get the following optimization problem:

$$\begin{aligned} \min_{q,\boldsymbol{\xi}\geq 0} \ & \mathrm{KL}(q(\boldsymbol{Y}|\boldsymbol{X}) \| p_\theta(\boldsymbol{Y}|\boldsymbol{X})) + C \textstyle\sum_{l,g_l} \xi_{l,g_l} \\ \text{s.t.} \ & \lambda_l(1 - \mathbb{E}_q[r_{l,g_l}(\boldsymbol{X}, \boldsymbol{Y})]) \leq \xi_{l,g_l} \\ & g_l = 1, \dots, G_l, \ \ l = 1, \dots, L, \end{aligned} \tag{3.3}$$

where $\xi_{l,g_l} \geq 0$ is the slack variable for respective logic constraint; and C is the regularization parameter. The problem can be seen as projecting p_θ into the constrained subspace. The problem is convex and can be efficiently solved in its dual form with closed-form solutions. We provide the detailed derivation in the supplementary materials and directly give the solution here:

$$q^*(\boldsymbol{Y}|\boldsymbol{X}) \propto p_\theta(\boldsymbol{Y}|\boldsymbol{X}) \exp\left\{ - \sum_{l,g_l} C\lambda_l(1 - r_{l,g_l}(\boldsymbol{X}, \boldsymbol{Y})) \right\} \tag{3.4}$$

Intuitively, a strong rule with large λ_l will lead to low probabilities of predictions that fail to meet the constraints. We discuss the computation of the normalization factor in section 3.3.4.

Our framework is related to the posterior regularization (PR) method (Ganchev et al., 2010) which places constraints over model posterior in unsupervised setting. In classification, our optimization procedure is analogous to the modified EM algorithm for PR, by using cross-entropy loss in Eq.(3.2) and evaluating the second loss term on unlabeled data differing from $\mathcal{D}$, so that Eq.(3.4) corresponds to the E-step and Eq.(3.2) is analogous to the M-step. This sheds light from another perspective on why our framework would work. However, we found in our experiments (section 3.5) that to produce strong performance it is crucial to use the same labeled data $\boldsymbol{x}_n$ in the two losses of Eq.(3.2) so as to form a direct trade-off between imitating soft predictions and predicting correct hard labels.

3.3.4 Implementations

The procedure of iterative distilling optimization of our framework is summarized in Algorithm 6.

During training we need to compute the soft predictions of q at each iteration, which is straightforward through direct enumeration if the rule constraints in Eq.(3.4) are factored in the same way as the base neural model p_θ (e.g., the "but"-rule of sentiment classification in section 3.4.1). If the constraints introduce additional dependencies, e.g., bi-gram dependency as the transition rule in the NER task (section 3.4.2), we can use dynamic programming for efficient computation. For higher-order constraints (e.g., the listing rule in NER), we approximate through Gibbs sampling that iteratively samples from $q(\boldsymbol{y}_i|\boldsymbol{y}_{-i}, \boldsymbol{x})$ for each position i. If the constraints span multiple instances, we group the relevant instances in minibatches for joint inference (and randomly break some dependencies when a group is too large). Note that calculating the soft predictions is efficient since only one NN forward pass is required to compute the base distribution $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ (and few more, if needed, for calculating the truth values of relevant rules).

p v.s. q at Test Time At test time we can use either the distilled student network p, or the teacher network q after a final projection. Our empirical results show that both models substantially improve over the base network that is trained with only data-label instances. In general q performs better than p. Particularly, q is more suitable when the logic rules introduce additional dependencies (e.g., spanning over multiple examples), requiring joint inference. In contrast, as mentioned above, p is more lightweight and efficient, and useful when rule evaluation is expensive or impossible at prediction time. Our experiments compare the performance of p and q extensively.

Imitation Strength π The imitation parameter π in Eq.(3.2) balances between emulating the teacher soft predictions and predicting the true hard labels. Since the teacher network is constructed from p_θ, which, at the beginning of training, would produce low-quality predictions, we thus favor predicting the true labels more at initial stage. As training goes on, we gradually bias towards emulating the teacher predictions to effectively distill the structured knowledge. Specifically, we define $\pi^{(t)} = \min\{\pi_0, 1 - \alpha^t\}$ at iteration $t \geq 0$, where $\alpha \leq 1$ specifies the speed of decay and $\pi_0 < 1$ is a lower bound.

3.4 Applications

We have presented our framework that is general enough to improve various types of neural networks with rules, and easy to use in that users are allowed to impose their knowledge and intentions through the declarative first-order logic. In this section we illustrate the versatility of our approach by applying it on two workhorse network architectures, i.e., convolutional network

Algorithm 1 Harnessing NN with Rules

```
Input: The training data $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$,
       The rule set $\mathcal{R} = \{(R_l, \lambda_l)\}_{l=1}^{L}$,
       Parameters: $\pi$ – imitation parameter
                   $C$ – regularization strength
Initialize neural network parameter $\boldsymbol{\theta}$
repeat
    Sample a minibatch $(\boldsymbol{X}, \boldsymbol{Y}) \subset \mathcal{D}$
    Construct teacher network $q$ with Eq.(3.4)
    Transfer knowledge into $p_\theta$ by updating $\boldsymbol{\theta}$ with Eq.(3.2)
until convergence
Output: Distill student network $p_\theta$ and teacher network $q$
```

and recurrent network, on two representative applications, i.e., *sentence-level sentiment analysis* which is a classification problem, and *named entity recognition* which is a sequence learning problem.

For each task, we first briefly describe the base neural network. Since we are not focusing on tuning network architectures, we largely use the same or similar networks to previous successful neural models. We then design the linguistically-motivated rules to be integrated.

3.4.1 Sentiment Classification

Sentence-level sentiment analysis is to identify the sentiment (e.g., positive or negative) underlying an individual sentence. The task is crucial for many opinion mining applications. One challenging point of the task is to capture the contrastive sense (e.g., by conjunction "but") within a sentence.

Base Network We use the single-channel convolutional network proposed in (Kim, 2014). The simple model has achieved compelling performance on various sentiment classification benchmarks. The network contains a convolutional layer on top of word vectors of a given sentence, followed by a max-over-time pooling layer and then a fully-connected layer with softmax output activation. A convolution operation is to apply a filter to word windows. Multiple filters with varying window sizes are used to obtain multiple features. Figure 3.2, left panel, shows the network architecture.

Logic Rules One difficulty for the plain neural network is to identify contrastive sense in order to capture the dominant sentiment precisely. The conjunction word "but" is one of the strong indicators for such sentiment changes in a sentence, where the sentiment of clauses following "but" generally dominates. We thus consider sentences S with an "A-but-B" structure, and expect the sentiment of the whole sentence to be consistent with the sentiment of clause B. The logic rule is written as:

$$\begin{aligned}&\text{has-‘A-but-B’-structure}(S) \Rightarrow \\ &\left(\mathbf{1}(y=+) \Rightarrow \boldsymbol{\sigma}_\theta(B)_+ \ \wedge\ \boldsymbol{\sigma}_\theta(B)_+ \Rightarrow \mathbf{1}(y=+)\right),\end{aligned} \tag{3.5}$$

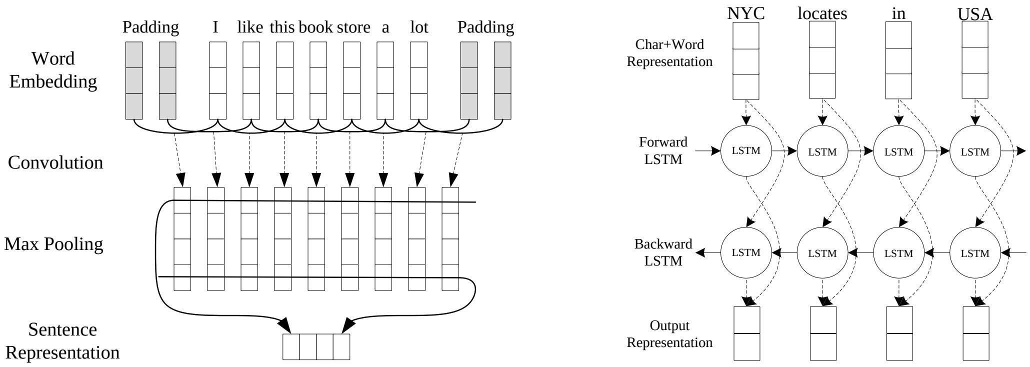


Figure 3.2: Left: The CNN architecture for sentence-level sentiment analysis. The sentence representation vector is followed by a fully-connected layer with softmax output activation, to output sentiment predictions. **Right**: The architecture of the bidirectional LSTM recurrent network for NER. The CNN for extracting character representation is omitted.

where $\mathbf{1}(\cdot)$ is an indicator function that takes 1 when its argument is true, and 0 otherwise; class '+' represents 'positive'; and $\boldsymbol{\sigma}_\theta(B)_+$ is the element of $\boldsymbol{\sigma}_\theta(B)$ for class '+'. By Eq.(3.1), when S has the 'A-but-B' structure, the truth value of the above logic rule equals to $(1+\boldsymbol{\sigma}_\theta(B)_+)/2$ when $y = +$, and $(2 - \boldsymbol{\sigma}_\theta(B)_+)/2$ otherwise [1]. Note that here we assume two-way classification (i.e., positive and negative), though it is straightforward to design rules for finer grained sentiment classification.

3.4.2 Named Entity Recognition

NER is to locate and classify elements in text into entity categories such as "persons" and "organizations". It is an essential first step for downstream language understanding applications. The task assigns to each word a named entity tag in an "X-Y" format where X is one of BIEOS (Beginning, Inside, End, Outside, and Singleton) and Y is the entity category. A valid tag sequence has to follow certain constraints by the definition of the tagging scheme. Besides, text with structures (e.g., lists) within or across sentences can usually expose some consistency patterns.

Base Network The base network has a similar architecture with the bi-directional LSTM recurrent network (called BLSTM-CNN) proposed in (Chiu and Nichols, 2015) for NER which has outperformed most of previous neural models. The model uses a CNN and pre-trained word vectors to capture character- and word-level information, respectively. These features are then fed into a bi-directional RNN with LSTM units for sequence tagging. Compared to (Chiu and Nichols, 2015) we omit the character type and capitalization features, as well as the additive transition matrix in the output layer. Figure 3.2, right panel, shows the network architecture.

[1] Replacing $\wedge$ with $\&$ in Eq.(3.5) leads to a probably more intuitive rule which takes the value $\boldsymbol{\sigma}_\theta(B)_+$ when $y = +$, and $1 - \boldsymbol{\sigma}_\theta(B)_+$ otherwise.

Logic Rules The base network largely makes independent tagging decisions at each position, ignoring the constraints on successive labels for a valid tag sequence (e.g., I-ORG cannot follow B-PER). In contrast to recent work (Lample et al., 2016) which adds a conditional random field (CRF) to capture bi-gram dependencies between outputs, we instead apply logic rules which does not introduce extra parameters to learn. An example rule is:

$$\text{equal}(y_{i-1}, \text{I-ORG}) \Rightarrow \neg\, \text{equal}(y_i, \text{B-PER}) \tag{3.6}$$

The confidence levels are set to ∞ to prevent any violation.

We further leverage the *list* structures within and across sentences of the same documents. Specifically, named entities at corresponding positions in a list are likely to be in the same categories. For instance, in "1. Juventus, 2. Barcelona, 3. ..." we know "Barcelona" must be an organization rather than a location, since its counterpart entity "Juventus" is an organization. We describe our simple procedure for identifying lists and counterparts in the supplementary materials. The logic rule is encoded as:

$$\text{is-counterpart}(X, A) \Rightarrow 1 - \|c(\boldsymbol{e}_y) - c(\boldsymbol{\sigma}_\theta(A))\|_2, \tag{3.7}$$

where $\boldsymbol{e}_y$ is the one-hot encoding of y (the class prediction of X); $c(\cdot)$ collapses the probability mass on the labels with the same categories into a single probability, yielding a vector with length equaling to the number of categories. We use ℓ_2 distance as a measure for the closeness between predictions of X and its counterpart A. Note that the distance takes value in $[0, 1]$ which is a proper soft truth value. The list rule can span multiple sentences (within the same document). We found the teacher network q that enables explicit joint inference provides much better performance over the distilled student network p (section 3.5).

3.5 Experiments

We validate our framework by evaluating its applications of sentiment classification and named entity recognition on a variety of public benchmarks. By integrating the simple yet effective rules with the base networks, we obtain substantial improvements on both tasks and achieve state-of-the-art or comparable results to previous best-performing systems. Comparison with a diverse set of other rule integration methods demonstrates the unique effectiveness of our framework. Our approach also shows promising potentials in the semi-supervised learning and sparse data context.

Throughout the experiments we set the regularization parameter to $C = 6$. In sentiment classification we set the imitation parameter to $\pi^{(t)} = 1 - 0.95^t$, while in NER $\pi^{(t)} = \min\{0.9, 1 - 0.9^t\}$ to downplay the noisy listing rule. The confidence levels of rules are set to $\lambda_l = 1$, except for hard constraints whose confidence is ∞. For neural network configuration, we largely followed the reference work, as specified in the following respective sections. All experiments were performed on a Linux machine with eight 4.0GHz CPU cores, one Tesla K40c GPU, and 32GB RAM. We implemented neural networks using Theano [2], a popular deep learning platform.

[2] http://deeplearning.net/software/theano

	Model	SST2	MR	CR
1	CNN (Kim, 2014)	87.2	81.3±0.1	84.3±0.2
2	CNN-Rule-p	88.8	81.6±0.1	85.0±0.3
3	CNN-Rule-q	89.3	**81.7±0.1**	**85.3±0.3**
4	MGNC-CNN (Zhang et al., 2016b)	88.4	–	–
5	MVCNN (Yin and Schutze, 2015)	**89.4**	–	–
6	CNN-multichannel (Kim, 2014)	88.1	81.1	85.0
7	Paragraph-Vec (Le and Mikolov, 2014)	87.8	–	–
8	CRF-PR (Yang and Cardie, 2014)	–	–	82.7
9	RNTN (Socher et al., 2013)	85.4	–	–
10	G-Dropout (Wang and Manning, 2013)	–	79.0	82.1

Table 3.1: Accuracy (%) of Sentiment Classification. Row 1, CNN (Kim, 2014) is the base network corresponding to the "CNN-non-static" model in (Kim, 2014). Rows 2-3 are the networks enhanced by our framework: CNN-Rule-p is the student network and CNN-Rule-q is the teacher network. For MR and CR, we report the average accuracy±one standard deviation using 10-fold cross validation.

3.5.1 Sentiment Classification

3.5.1.1 Setup

We test our method on a number of commonly used benchmarks, including **1) SST2**, Stanford Sentiment Treebank (Socher et al., 2013) which contains 2 classes (negative and positive), and 6920/872/1821 sentences in the train/dev/test sets respectively. Following (Kim, 2014) we train models on both sentences and phrases since all labels are provided. **2) MR** (Pang and Lee, 2005), a set of 10,662 one-sentence movie reviews with negative or positive sentiment. **3) CR** (Hu and Liu, 2004), customer reviews of various products, containing 2 classes and 3,775 instances. For MR and CR, we use 10-fold cross validation as in previous work. In each of the three datasets, around 15% sentences contains the word "but".

For the base neural network we use the "non-static" version in (Kim, 2014) with the exact same configurations. Specifically, word vectors are initialized using word2vec (Mikolov et al., 2013) and fine-tuned throughout training, and the neural parameters are trained using SGD with the Adadelta update rule (Zeiler, 2012).

3.5.1.2 Results

Table 3.1 shows the sentiment classification performance. Rows 1-3 compare the base neural model with the models enhanced by our framework with the "but"-rule (Eq.(3.5)). We see that our method provides a strong boost on accuracy over all three datasets. The teacher network q further improves over the student network p, though the student network is more widely applicable in certain contexts as discussed in sections 3.3.2 and 3.3.4. Rows 4-10 show the accuracy of recent top-performing methods. On the MR and CR datasets, our model outperforms all the

	Model	Accuracy (%)
1	CNN (Kim, 2014)	87.2
2	-but-clause	87.3
3	-ℓ_2-reg	87.5
4	-project	87.9
5	-opt-project	88.3
6	-pipeline	87.9
7	-Rule-p	88.8
8	-Rule-q	**89.3**

Table 3.2: Performance of different rule integration methods on SST2. 1) CNN is the base network; 2) "-but-clause" takes the clause after "but" as input; 3) "-ℓ_2-reg" imposes a regularization term $\gamma\|\boldsymbol{\sigma}_\theta(S) - \boldsymbol{\sigma}_\theta(Y)\|_2$ to the CNN objective, with the strength γ selected on dev set; 4) "-project" projects the trained base CNN to the rule-regularized subspace with Eq.(3.3); 5) "-opt-project" directly optimizes the projected CNN; 6) "-pipeline" distills the pre-trained "-opt-project" to a plain CNN; 7-8) "-Rule-p" and "-Rule-q" are our models with p being the distilled student network and q the teacher network. Note that "-but-clause" and "-ℓ_2-reg" are ad-hoc methods applicable specifically to the "but"-rule.

baselines. On SST2, MVCNN (Yin and Schutze, 2015) (Row 5) is the only system that shows a slightly better result than ours. Their neural network has combined diverse sets of pre-trained word embeddings (while we use only word2vec) and contained more neural layers and parameters than our model.

To further investigate the effectiveness of our framework in integrating structured rule knowledge, we compare with an extensive array of other possible integration approaches. Table 3.2 lists these methods and their performance on the SST2 task. We see that: 1) Although all methods lead to different degrees of improvement, our framework outperforms all other competitors with a large margin. 2) In particular, compared to the pipelined method in Row 6 which is in analogous to the structure compilation work (Liang et al., 2008), our iterative distillation (section 3.3.2) provides better performance. Another advantage of our method is that we only train one set of neural parameters, as opposed to two separate sets as in the pipelined approach. 3) The distilled student network "-Rule-p" achieves much superior accuracy compared to the base CNN, as well as "-project" and "-opt-project" which explicitly project CNN to the rule-constrained subspace. This validates that our distillation procedure transfers the structured knowledge into the neural parameters effectively. The inferior accuracy of "-opt-project" can be partially attributed to the poor performance of its neural network part which achieves only 85.1% accuracy and leads to inaccurate evaluation of the "but"-rule in Eq.(3.5).

We next explore the performance of our framework with varying numbers of labeled instances as well as the effect of exploiting unlabeled data. Intuitively, with less labeled examples we expect the general rules would contribute more to the performance, and unlabeled data should help better learn from the rules. This can be a useful property especially when data are sparse and labels are expensive to obtain. Table 3.3 shows the results. The subsampling is conducted

	Data size	5%	10%	30%	100%
1	CNN	79.9	81.6	83.6	87.2
2	-Rule-p	81.5	83.2	84.5	88.8
3	-Rule-q	82.5	83.9	85.6	**89.3**
4	-semi-PR	81.5	83.1	84.6	–
5	-semi-Rule-p	81.7	83.3	84.7	–
6	-semi-Rule-q	**82.7**	**84.2**	**85.7**	–

Table 3.3: Accuracy (%) on SST2 with varying sizes of labeled data and semi-supervised learning. The header row is the percentage of labeled examples for training. Rows 1-3 use only the supervised data. Rows 4-6 use semi-supervised learning where the remaining training data are used as unlabeled examples. For "-semi-PR" we only report its projected solution (in analogous to q) which performs better than the non-projected one (in analogous to p).

on the sentence level. That is, for instance, in "5%" we first selected 5% training sentences uniformly at random, then trained the models on these sentences as well as their phrases. The results verify our expectations. 1) Rows 1-3 give the accuracy of using only data-label subsets for training. In every setting our methods consistently outperform the base CNN. 2) "-Rule-q" provides higher improvement on 5% data (with margin 2.6%) than on larger data (e.g., 2.3% on 10% data, and 2.0% on 30% data), showing promising potential in the sparse data context. 3) By adding unlabeled instances for semi-supervised learning as in Rows 5-6, we get further improved accuracy. 4) Row 4, "-semi-PR" is the posterior regularization (Ganchev et al., 2010) which imposes the rule constraint through only unlabeled data during training. Our distillation framework consistently provides substantially better results.

3.5.2 Named Entity Recognition

3.5.2.1 Setup

We evaluate on the well-established CoNLL-2003 NER benchmark (Tjong Kim Sang and De Meulder, 2003), which contains 14,987/3,466/3,684 sentences and 204,567/51,578/46,666 tokens in train/dev/test sets, respectively. The dataset includes 4 categories, i.e., *person, location, organization*, and *misc*. BIOES tagging scheme is used. Around 1.7% named entities occur in lists.

We use the mostly same configurations for the base BLSTM network as in (Chiu and Nichols, 2015), except that, besides the slight architecture difference (section 3.4.2), we apply Adadelta for parameter updating. GloVe (Pennington et al., 2014) word vectors are used to initialize word features.

3.5.2.2 Results

Table 3.4 presents the performance on the NER task. By incorporating the bi-gram transition rules (Row 2), the joint teacher model q achieves 1.56 improvement in F1 score that outperforms most previous neural based methods (Rows 4-7), including the BLSTM-CRF model (Lample

	Model	F1
1	BLSTM	89.55
2	BLSTM-Rule-trans	p: 89.80, q: 91.11
3	BLSTM-Rules	p: 89.93, q: **91.18**
4	NN-lex (Collobert et al., 2011)	89.59
5	S-LSTM (Lample et al., 2016)	90.33
6	BLSTM-lex (Chiu and Nichols, 2015)	90.77
7	BLSTM-CRF$_1$ (Lample et al., 2016)	90.94
8	Joint-NER-EL (Luo et al., 2015)	91.20
9	BLSTM-CRF$_2$ (Ma and Hovy, 2016)	**91.21**

Table 3.4: Performance of NER on CoNLL-2003. Row 2, BLSTM-Rule-trans imposes the transition rules (Eq.(3.6)) on the base BLSTM. Row 3, BLSTM-Rules further incorporates the list rule (Eq.(3.7)). We report the performance of both the student model p and the teacher model q.

et al., 2016) which applies a conditional random field (CRF) on top of a BLSTM in order to capture the transition patterns and encourage valid sequences. In contrast, our method implements the desired constraints in a more straightforward way by using the declarative logic rule language, and at the same time does not introduce extra model parameters to learn. Further integration of the list rule (Row 3) provides a second boost in performance, achieving an F1 score very close to the best-performing systems including Joint-NER-EL (Luo et al., 2015) (Row 8), a probabilistic graphical model optimizing NER and entity linking jointly with massive external resources, and BLSTM-CRF (Ma and Hovy, 2016), a combination of BLSTM and CRF with more parameters than our rule-enhanced neural networks.

From the table we see that the accuracy gap between the joint teacher model q and the distilled student p is relatively larger than in the sentiment classification task (Table 3.1). This is because in the NER task we have used logic rules that introduce extra dependencies between adjacent tag positions as well as multiple instances, making the explicit joint inference of q useful for fulfilling these structured constraints.

3.6 Discussion

We have developed an algorithm which combines deep neural networks with first-order logic rules to allow integrating human knowledge and goals into the neural models. In particular, we proposed an iterative distillation procedure that transfers the structured information of logic rules into the weights of neural networks. The transferring is done via a teacher network constructed using the posterior regularization principle. The algorithm is a member of the algorithm space defined by the standard equation in Chapter 2.

Our approach is general and applicable to various types of neural architectures. With a few intuitive rules, our approach significantly improves base networks on sentiment analysis and named entity recognition, demonstrating the practical significance of our approach. The encouraging empirical results indicate a strong potential of our approach for improving other application domains such as vision tasks, which we plan to explore in the future.

Though we have focused on first-order logic rules, we leveraged soft logic formulation which can be easily extended to general probabilistic models for expressing structured distributions and performing inference and reasoning. The next chapter deals with the to an extent more flexible form of experiences, namely auxiliary models.

Finally, in many real environments, it would be desirable to also automatically learn the confidence of different rules, and even derive new rules from data. The problem is studied in (Hu et al., 2016b, 2018a), where the standard equation again plays a central role for designing the extended solutions.

3.6.1 Incorporating Structured Knowledge: Experiences v.s. Model Architectures

The proposed algorithm, along with the contemporary work (e.g., Hu et al., 2016a,b; Hinton et al., 2015; Lopez-Paz et al., 2015; Hu et al., 2017a), represents a general means of adding (structured) knowledge to black-box neural networks by devising *knowledge-based experiences* that drive the model to learn the desired structures. This differs from the other popular way that embeds domain knowledge into *specifically-designed neural architectures* (e.g., the knowledge of translation-invariance in image classification is hard-coded in the conv-pooling architecture of ConvNet). While the specialized neural architectures can usually be very effective to capture the designated knowledge, incorporating knowledge via specialized losses enjoys the advantage of generality and flexibility:

- **Model-agnostic**. The learning framework is applicable to neural models with any architectures, e.g., ConvNets, RNNs, and other specialized ones (Hu et al., 2016a).
- **Compatibility with rich experiences**. Compared to the conventional end-to-end maximum likelihood learning that usually requires fully-annotated or paired data, the experience-based method allows additional supervisions based on, e.g., auxiliary models (Hinton et al., 2015; Hu et al., 2016b; Yang et al., 2018) and data from other related tasks. For example, as studied in Chapter 4, we leverage datasets of sentence sentiment and phrase tense to learn to control the both attributes (sentiment and tense) when generating sentences.
- **Modular design and learning**. With the rich sources of experiences, design and learning of a model can still be simple and efficient, because each of the experiences can be formulated independently to each other and each be plugged into the objective function. For example, Chapter 4 *separately* learns two classifiers, one for sentiment and the other for tense, on two *separate* datasets, respectively. The two classifiers carry respective semantic knowledge, and are then *jointly* applied to a text generation model for attribute control. In comparison, mixing and hard-coding multiple knowledge in a single neural architecture can be difficult and quickly becoming impossible when the number of knowledge increases.
- **Generation with discrimination knowledge**. In generation tasks, it can sometimes be difficult to incorporate knowledge directly in the generative process (or model architecture), i.e., defining *how to generate*. In contrast, it is often easier to instead specify a evaluation metric that measures the quality of a given sample in terms of the knowledge, i.e., defining *what*

desired generation is. For example, in the human image generation task studied in Chapter 6, evaluating the structured human part consistency could be easier than designing a generator architecture that hard-codes the structured generation process for the human parts.

It is worth noting that the two paradigms are not mutually exclusive. A model with knowledge-inspired specialized architecture can still be learned by optimizing knowledge-inspired losses. Different types of knowledge can be best fit for either architecture hard-coding or loss optimization. It would be interesting to explore the combination of both in the above tasks and others.

3.7 Appendix

Solving Problem Eq.(3.3), Section 3.3.3

We provide the detailed derivation for solving the problem in Eq.(3.3), Section 3.3.3, which we repeat here:

$$\begin{aligned} \min_{q,\boldsymbol{\xi}\geq 0} \ & \mathrm{KL}(q(\boldsymbol{Y}|\boldsymbol{X})\|p_\theta(\boldsymbol{Y}|\boldsymbol{X})) + C\sum_{l,g_l}\xi_{l,g_l} \\ \text{s.t. } \ & \lambda_l(1-\mathbb{E}_q[r_{l,g_l}(\boldsymbol{X},\boldsymbol{Y})]) \leq \xi_{l,g_l} \\ & g_l = 1,\dots,G_l, \ \ l=1,\dots,L, \end{aligned} \tag{3.7.1}$$

The following derivation is largely adapted from (Ganchev et al., 2010) for the logic rule constraint setting, with some reformulation that produces closed-form solution.

The Lagrangian is

$$\max_{\boldsymbol{\mu}\geq 0,\boldsymbol{\eta}\geq 0,\alpha\geq 0} \ \min_{q(\boldsymbol{y}),\boldsymbol{\xi}} L, \tag{3.7.2}$$

where

$$\begin{aligned} L = \ & \mathrm{KL}(q(\boldsymbol{Y}|\boldsymbol{X})\|p_\theta(\boldsymbol{Y}|\boldsymbol{X})) + \sum_{l,g_l}(C-\mu_{l,g_l})\xi_{l,g_l} \\ & + \sum_{l,g_l}\eta_{l,g_l}\left(\mathbb{E}_q[\lambda_l(1-r_{l,g_l}(\boldsymbol{X},\boldsymbol{Y}))]-\xi_{l,g_l}\right) + \alpha(\sum_{\boldsymbol{Y}} q(\boldsymbol{Y}|\boldsymbol{X})-1) \end{aligned} \tag{3.7.3}$$

Solving Eq.(3.7.2), we obtain

$$\begin{aligned} \nabla_q L = \log q(\boldsymbol{Y}|\boldsymbol{X}) + 1 - \log p_\theta(\boldsymbol{Y}|\boldsymbol{X}) + \sum_{l,g_l}\eta_{l,g_l}\left[\lambda_l(1-r_{l,g_l}(\boldsymbol{X},\boldsymbol{Y}))\right] + \alpha = 0 \\ \Longrightarrow \qquad q(\boldsymbol{Y}|\boldsymbol{X}) = \frac{p_\theta(\boldsymbol{Y}|\boldsymbol{X})\exp\left\{-\sum_l \eta_l\lambda_l(1-r_{l,g_l}(\boldsymbol{X},\boldsymbol{Y}))\right\}}{e\exp(\alpha)} \end{aligned} \tag{3.7.4}$$

$$\nabla_{\xi_{l,g_l}} L = C - \mu_{l,g_l} - \eta_{l,g_l} = 0 \qquad \Longrightarrow \qquad \mu_{l,g_l} = C - \eta_{l,g_l} \tag{3.7.5}$$

$$
\begin{aligned}
\nabla_\alpha L = \sum_{\boldsymbol{Y}} \frac{p_\theta(\boldsymbol{Y}|\boldsymbol{X}) \exp\left\{-\sum_{l,g_l} \eta_{l,g_l}\lambda_l(1-r_{l,g_l}(\boldsymbol{X},\boldsymbol{Y}))\right\}}{e\exp(\alpha)} - 1 = 0 \\
\implies \qquad \alpha = \log\left(\frac{\sum_{\boldsymbol{Y}} p(\boldsymbol{Y}|\boldsymbol{X}) \exp\left\{-\sum_{l,g_l} \eta_{l,g_l}\lambda_l(1-r_{l,g_l}(\boldsymbol{X},\boldsymbol{Y}))\right\}}{e}\right)
\end{aligned}
\tag{3.7.6}
$$

Let $Z_\eta = \sum_{\boldsymbol{Y}} p(\boldsymbol{Y}|\boldsymbol{X}) \exp\left\{-\sum_{l,g_l} \eta_{l,g_l}\lambda_l(1-r_{l,g_l}(\boldsymbol{X},\boldsymbol{Y}))\right\}$. Plugging α into L

$$
\begin{aligned}
L &= -\log Z_\eta + \sum_{l,g_l}(C+\mu_{l,g_l})\xi_{l,g_l} - \sum_{l,g_l}\eta_{l,g_l}\xi_{l,g_l} \\
&= -\log Z_\eta
\end{aligned}
\tag{3.7.7}
$$

Since Z_η monotonically decreases as η increases, and from Eq.(3.7.5) we have $\eta_{l,g_l} \le C$, therefore:

$$
\begin{aligned}
&\max_{C\ge\eta\ge0} -\log Z_\eta \\
&\qquad \implies \qquad \eta^*_{l,g_l} = C
\end{aligned}
\tag{3.7.8}
$$

Plugging Eqs.(3.7.6) and (3.7.8) into Eq.(3.7.4) we obtain the solution of q as in Eq.(4).

Identifying Lists for NER

We design a simple pattern-matching based method to identify lists and counterparts in the NER task. We ensure high precision and do not expect high recall. In particular, we only retrieve lists that with the pattern "1. ... 2. ... 3. ..." (i.e., indexed by numbers), and "- ... - ... - ..." (i.e., each item marked with "-"). We require at least 3 items to form a list.

We further require the text of each item follows certain patterns to ensure the text is highly likely to be named entities, and rule out those lists whose item text is largely free text. Specifically, we require 1) all words of the item text all start with capital letters; 2) referring the text between punctuations as "block", each block includes no more than 3 words.

We detect both intra-sentence lists and inter-sentence lists in documents. We found the above patterns are effective to identify true lists. A better list detection method is expected to further improve our NER results.

Chapter 4

Learning with Auxiliary Models

We now consider learning with another general type of experience, namely auxiliary models from related tasks. That is, now the experience itself is models designed for other problems which bear certain relationships with the problem at hand. Those models thus have encoded relevant information of which we hope to make use to train our target model. As discussed in Chapter 2 (section 2.3.4), a prominent example of this paradigm of learning is the Generative Adversarial Networks (GANs, Goodfellow et al., 2014), where, to learn a generative model that can produce realistic images, an auxiliary discriminative model that measures the "realisticness" of input images is used as the experience.

Of particular interest to the discussion in this chapter is a more challenging situation where no single auxiliary model has the full information needed to properly learn the target model, and it is necessary to combine multiple auxiliary models from different related problems or to mix with other types of experiences (e.g., data, logic rules). Here is where the standard equation comes into play, which allows a turnkey interface to plug in all different auxiliary models and other experiences to drive learning. Specifically, as discussed in section 2.5.1, one could set the experience function in the standard equation to be a weighted sum of multiple constituent experience functions:

$$f(\boldsymbol{t}) = \sum\nolimits_i \lambda_i f_i(\boldsymbol{t}),$$

where each f_i encodes a particular source of experiences. The design of the solution thus reduces to formulating each of the experience functions $\{f_i\}$ based on the available auxiliary models. This chapter gives an example of applying this framework on the concrete problem of controllable text generation.

Let us consider a specific setting of controllable text generation, namely text style transfer. Let the text "style" to be controlled be the sentiment (i.e., "positive" or "negative"). Given a sentence $\boldsymbol{x}$ and a target sentiment c, the goal of the problem is *to generate a new sentence* $\boldsymbol{y}$ *that has the target sentiment while preserving all other aspects of the original sentence.* For example, given a sentence "the manager is a horrible person" and the target sentiment "positive", we want to be able to generate new sentence "the manager is a perfect person". One of the key challenges in practice is that we usually do not have direct supervision data (i.e., pairs of sentences that are exact the same except for sentiment), making it necessary to use other forms of experiences. We briefly review the backbone of the solutions here, highlighting how the solution can be built mechanically under the guidance of the standard equation, by identifying what experiences we should use and formulating as experience functions. More details are described in the following sections.

Specifically, from the problem goal we see that the generative model must learn the concept of "sentiment". A natural form of experience that has encoded the particular concept is a pretrained sentiment classifier (SC). The first experience function $f_1(\boldsymbol{x}, c, \boldsymbol{y}) = \mathrm{SC}(c, \boldsymbol{y})$ can then be defined based on the auxiliary classification model, to evaluate the probability of the classifier predicting sentiment c given a configuration $\boldsymbol{y}$. The single experience is not sufficient. As another basic requirement, we want the model to generate fluent and grammatical sentences. The best experience to this end is perhaps again an auxiliary model, namely a pretrained language model (LM), $f_2(\boldsymbol{x}, c, \boldsymbol{y}) = \mathrm{LM}(\boldsymbol{y})$, that estimates the likelihood of a configuration $\boldsymbol{y}$ under the language distribution. Finally, to teach the model about preservation of the input content as part of the problem goal, we combine the third experience f_3 which are simple "reconstruction"-style data instances $\{(\boldsymbol{x}^*, c^* = c_{\boldsymbol{x}^*}, \boldsymbol{y}^* = \boldsymbol{x}^*)\}$, where the target sentiment c^* is set to the sentiment of the original sentence $\boldsymbol{x}^*$, and by the problem definition the ground-truth output $\boldsymbol{y}^*$ is exact the same as the input $\boldsymbol{x}^*$.

The remainder of the chapter elaborate the problems and solutions in more details. Section 4.1 defines the more general problem of controllable text generation, and studies the experiences of auxiliary classifiers, weak data instances, plus other constraints. This section also makes preliminary studies on adapting the auxiliary model experiences jointly with the target model, though we discuss the joint training more comprehensively in the next part of the thesis. Section 4.2 focuses on text style transfer and further incorporates the additional language model for training. As a result, we managed to obtain a set of deep text generation models which are among the earliest deep generative models capable of controlling text attributes.

4.1 Learning with Auxiliary Classifiers for Controllable Text Generation

4.1.1 Introduction

There is a surge of research interest in deep generative models (Hu et al., 2018b), such as Variational Autoencoders (VAEs) (Kingma and Welling, 2013), Generative Adversarial Nets (GANs) (Goodfellow et al., 2014), and auto-regressive models (van den Oord et al., 2016) Despite their impressive advances in visual domain, such as image generation (Radford et al., 2015b), learning interpretable image representations (Chen et al., 2016b), and image editing (Zhu et al., 2016), applications to natural language generation have been relatively less studied. Even generating realistic sentences is challenging as the generative models are required to capture complex semantic structures underlying sentences. Previous work have been mostly limited to task-specific applications in supervised settings, including machine translation (Bahdanau et al.,

2014) and image captioning (Vinyals et al., 2015b). However, autoencoder frameworks (Sutskever et al., 2014) and recurrent neural network language models (Mikolov et al., 2010) do not apply to generic text generation from arbitrary hidden representations due to the unsmoothness of effective hidden codes (Bowman et al., 2015). Very few recent attempts of using VAEs (Bowman et al., 2015; Tang et al., 2016) and GANs (Yu et al., 2017a; Zhang et al., 2016a) have been made to investigate generic text generation, while their generated text is largely randomized and uncontrollable.

In this paper we tackle the problem of *controlled* generation of text. That is, we focus on generating realistic sentences, whose attributes can be controlled by learning disentangled latent representations. To enable the manipulation of generated sentences, a few challenges need to be addressed.

A first challenge comes from the discrete nature of text samples. The resulting non-differentiability hinders the use of global discriminators that assess generated samples and back-propagate gradients to guide the optimization of generators in a holistic manner, as shown to be highly effective in continuous image generation and representation modeling (Chen et al., 2016b; Larsen et al., 2016; Dosovitskiy and Brox, 2016). A number of recent approaches attempt to address the non-differentiability through policy learning (Yu et al., 2017a) which tends to suffer from high variance during training, or continuous approximations (Zhang et al., 2016a; Kusner and Hernández-Lobato, 2016) where only preliminary qualitative results are presented. As an alternative to the discriminator based learning, semi-supervised VAEs (Kingma et al., 2014) minimize element-wise reconstruction error on observed examples and are applicable to discrete visibles. This, however, loses the holistic view of full sentences and can be inferior especially for modeling global abstract attributes (e.g., sentiment).

Another challenge for controllable generation relates to learning disentangled latent representations. Interpretability expects each part of the latent representation to govern and *only* focus on one aspect of the samples. Prior methods (Chen et al., 2016b; Odena et al., 2016) on structured representation learning lack explicit enforcement of the independence property on the full latent representation, and varying individual code may result in unexpected variation of other unspecified attributes besides the desired one.

In this paper, we propose a new text generative model that addresses the above issues, permitting highly disentangled representations with designated semantic structure, and generating sentences with dynamically specified attributes. We base our generator on VAEs in combination with holistic discriminators of attributes for effective imposition of structures on the latent code. End-to-end optimization is enabled with differentiable softmax approximation which anneals smoothly to discrete case and helps fast convergence. The probabilistic encoder of VAE also functions as an additional discriminator to capture variations of implicitly modeled aspects, and guide the generator to avoid entanglement during attribute code manipulation.

Our model can be interpreted as enhancing VAEs with an extended wake-sleep procedure (Hinton et al., 1995), where the sleep phase enables incorporation of generated samples for learning both the generator and discriminators in an alternating manner. The generator and the discriminators effectively provide feedback signals to each other, resulting in an efficient mutual bootstrapping framework. We show a little supervision (e.g., 100s of annotated sentences) is sufficient to learn structured representations.

Besides efficient representation learning and enabled semi-supervised training, another advantage of using discriminators as learning signals for the generator, as compared to conventional conditional reconstruction based methods (Wen et al., 2015; Kingma et al., 2014), is that discriminators of different attributes can be trained independently. That is, for each attribute one can use separate labeled data for training the respective discriminator, and the trained discriminators can be combined arbitrarily to control a set of attributes of interest. In contrast, reconstruction based approaches typically require every instance of the training data to be labeled exhaustively with all target attributes (Wen et al., 2015), or to marginalize out any missing attributes (Kingma et al., 2014) which can be computationally expensive.

As a showing case, we apply our model to generate sentences with controlled sentiment and tenses. Though to our best knowledge there is no text corpus with both sentiment and tense labels, our method enables to use separate datasets, one with annotated sentiment and the other with tense labels. Quantitative experiments demonstrate the efficacy of our method. Our model improves over previous generative models on the accuracy of generating specified attributes as well as performing classification using generated samples. We show our method learns highly disentangled representations from only word-level labels, and produces plausible short sentences.

4.1.2 Related Work

Remarkable progress has been made in deep generative modeling. Hu et al. (2018b) provide a unified view of a diverse set of deep generative methods. Variational Autoencoders (VAEs) (Kingma and Welling, 2013) consist of encoder and generator networks which encode a data example to a latent representation and generate samples from the latent space, respectively. The model is trained by maximizing a variational lower bound on the data log-likelihood under the generative model. A KL divergence loss is minimized to match the posterior of the latent code with a prior, which enables every latent code from the prior to decode into a plausible sentence. Without the KL regularization, VAEs degenerate to autoencoders and become inapplicable for the generic generation. The vanilla VAEs are incompatible with discrete latents as they hinder differentiable parameterization for learning the encoder. Wake-sleep algorithm (Hinton et al., 1995) introduced for learning deep directed graphical models shares similarity with VAEs by also combining an inference network with the generator. The wake phase updates the generator with samples generated from the inference network on training data, while the sleep phase updates the inference network based on samples from the generator. Our method combines VAEs with an extended wake-sleep in which the sleep procedure updates both the generator and inference network (discriminators), enabling collaborative semi-supervised learning.

Besides reconstruction in raw data space, discriminator-based metric provides a different way for generator learning, i.e., the discriminator assesses generated samples and feedbacks learning signals. For instance, GANs (Goodfellow et al., 2014) use a discriminator to feedback the probability of a sample being recognized as a real example. Larsen et al. (2016) combine VAEs with GANs for enhanced image generation. Dosovitskiy and Brox (2016); Taigman et al. (2017)

use discriminators to measure high-level perceptual similarity. Applying discriminators to text generation is hard due to the non-differentiability of discrete samples (Yu et al., 2017a; Zhang et al., 2016a; Kusner and Hernández-Lobato, 2016). Bowman et al. (2015); Tang et al. (2016); Yang et al. (2017b) instead use VAEs without discriminators. All these text generation methods do not learn disentangled latent representations, resulting in randomized and uncontrollable samples. In contrast, disentangled generation in visual domain has made impressive progress. E.g., InfoGAN (Chen et al., 2016b), which resembles the extended sleep procedure of our joint VAE/wake-sleep algorithm, disentangles latent representation in an unsupervised manner. The semantic of each dimension is observed after training rather than designated by users in a controlled way. Siddharth et al. (2017); Kingma et al. (2014) base on VAEs and obtain disentangled image representations with semi-supervised learning. Zhou and Neubig (2017) extend semi-supervised VAEs for text transduction. In contrast, our model combines VAEs with discriminators which provide a better, holistic metric compared to element-wise reconstruction. Moreover, most of these approaches have only focused on the disentanglement of the structured part of latent representations, while ignoring potential dependence of the structured code with attributes not explicitly encoded. We address this by introducing an independency constraint, and show its effectiveness for improved interpretability.

4.1.3 Methodology

Our model aims to generate plausible sentences conditioned on representation vectors which are endowed with designated semantic structures. For instance, to control sentence sentiment, our model allocates one dimension of the latent representation to encode "positive" and "negative" semantics, and generates samples with desired sentiment by simply specifying a particular code. Benefiting from the disentangled structure, each such code is able to capture a salient attribute and is independent with other features. Our deep text generative model possesses several merits compared to prior work, as it 1) facilitates effective imposition of latent code semantics by enabling global discriminators to guide the discrete text generator learning; 2) improves model interpretability by explicitly enforcing the constraints on independent attribute controls; 3) permits efficient semi-supervised learning and bootstrapping by synthesizing variational auto-encoders with a tailored wake-sleep approach. We first present the overview of our framework, then describe the model in detail.

4.1.3.1 Model Overview

We build our framework starting from variational auto-encoders (§4.1.2) which have been used for text generation (Bowman et al., 2015), where sentence $\hat{\boldsymbol{x}}$ is generated conditioned on latent code $\boldsymbol{z}$. The vanilla VAE employs an unstructured vector $\boldsymbol{z}$ in which the dimensions are entangled. To model and control the attributes of interest in an interpretable way, we augment the unstructured variables $\boldsymbol{z}$ with a set of structured variables $\boldsymbol{c}$ each of which targets a salient and independent semantic feature of sentences.

We want our sentence generator to condition on the combined vector $(\boldsymbol{z}, \boldsymbol{c})$, and generate samples that fulfill the attributes as specified in the structured code $\boldsymbol{c}$. Conditional generation in the context of VAEs (e.g., semi-supervised VAEs (Kingma et al., 2014)) is often learned by reconstructing observed examples given their feature code. However, as demonstrated in visual domain, compared to computing element-wise distances in the data space, computing distances in the feature space allows invariance to distracting transformations and provides a better, holistic metric. Thus, for each attribute code in $\boldsymbol{c}$, we set up an individual discriminator to measure how well the generated samples match the desired attributes, and drive the generator to produce improved results. The difficulty of applying discriminators in our context is that text samples are discrete and non-differentiable, which breaks down gradient propagation from the discriminators to the generator. We use a continuous approximation based on softmax with a decreasing temperature, which anneals to the discrete case as training proceeds. This simple yet effective approach enjoys low variance and fast convergence.

Intuitively, having an interpretable representation would imply that each structured code in $\boldsymbol{c}$ can independently control its target feature, without entangling with other attributes, especially those not explicitly modeled. We encourage the independency by enforcing those irrelevant attributes to be completely captured in the unstructured code $\boldsymbol{z}$ and thus be separated from $\boldsymbol{c}$ that we will manipulate. To this end, we reuse the VAE encoder as an additional discriminator for recognizing the attributes modeled in $\boldsymbol{z}$, and train the generator so that these unstructured attributes can be recovered from the generated samples. As a result, varying different attribute codes will keep the unstructured attributes invariant as long as $\boldsymbol{z}$ is unchanged.

Figure 4.1 shows the overall model structure. Our complete model incorporates VAEs and attribute discriminators, in which the VAE component trains the generator to reconstruct real sentences for generating plausible text, while the discriminators enforce the generator to produce attributes coherent with the conditioned code. The attribute discriminators are learned to fit labeled examples to entail designated semantics, as well as trained to explain samples from the generator. That is, the generator and the discriminators form a pair of collaborative learners and provide feedback signals to each other. The collaborative optimization resembles wake-sleep algorithm. We show the combined VAE/wake-sleep learning enables a highly efficient semi-supervised framework, which requires only a little supervision to obtain interpretable representation and generation.

4.1.3.2 Model Structure

We now describe our model in detail, by presenting the learning of generator and discriminators, respectively.

Generator Learning. The generator G is an LSTM-RNN for generating token sequence $\hat{\boldsymbol{x}} = \{\hat{x}_1, \dots, \hat{x}_T\}$ conditioned on the latent code $(\boldsymbol{z}, \boldsymbol{c})$, which depicts a generative distribution:

$$\begin{aligned} \hat{\boldsymbol{x}} \sim G(\boldsymbol{z}, \boldsymbol{c}) &= p_G(\hat{\boldsymbol{x}}|\boldsymbol{z}, \boldsymbol{c}) \\ &= \prod\nolimits_t p(\hat{x}_t|\hat{\boldsymbol{x}}^{<t}, \boldsymbol{z}, \boldsymbol{c}), \end{aligned} \tag{4.1.1}$$

where $\hat{\boldsymbol{x}}^{<t}$ indicates the tokens preceding $\hat{x}_t$. The generation thus involves a sequence of discrete decision making which samples a token from a multinomial distribution parametrized using soft-

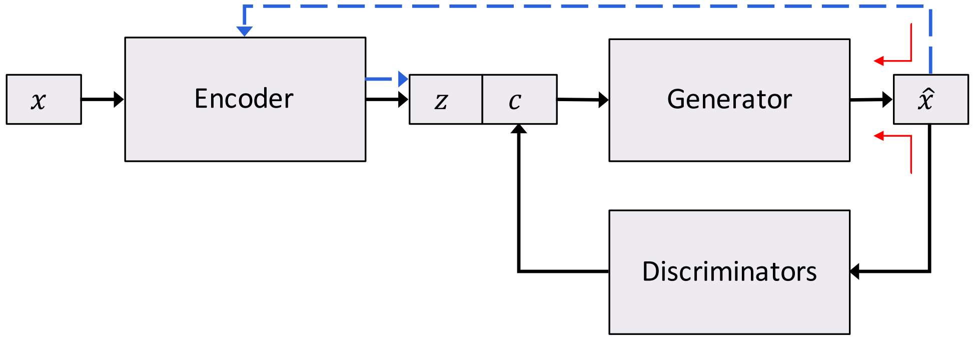


Figure 4.1: The generative model, where $\boldsymbol{z}$ is unstructured latent code and $\boldsymbol{c}$ is structured code targeting sentence attributes to control. Blue dashed arrows denote the proposed independency constraint, and red arrows denote gradient propagation enabled by the differentiable approximation.

max function at each time step t:

$$\hat{x}_t \sim \text{softmax}(\boldsymbol{o}_t/\tau), \tag{4.1.2}$$

where $\boldsymbol{o}_t$ is the logit vector as the inputs to the softmax function, and $\tau > 0$ is the temperature normally set to 1.

The unstructured part $\boldsymbol{z}$ of the representation is modeled as continuous variables with standard Gaussian prior $p(\boldsymbol{z})$, while the structured code $\boldsymbol{c}$ can contain both continuous and discrete variables to encode different attributes (e.g., sentiment categories, formality) with appropriate prior $p(\boldsymbol{c})$. Given observation $\boldsymbol{x}$, the base VAE includes a conditional probabilistic encoder E to infer the latents $\boldsymbol{z}$:

$$\boldsymbol{z} \sim E(\boldsymbol{x}) = q_E(\boldsymbol{z}|\boldsymbol{x}). \tag{4.1.3}$$

Let $\boldsymbol{\theta}_G$ and $\boldsymbol{\theta}_E$ denote the parameters of the generator G and the encoder E, respectively. The VAE is then optimized to minimize the reconstruction error of observed real sentences, and at the same time regularize the encoder to be close to the prior $p(\boldsymbol{z})$:

$$\begin{aligned}\mathcal{L}_{\text{VAE}}(\boldsymbol{\theta}_G, \boldsymbol{\theta}_E; \boldsymbol{x}) = \text{KL}(q_E(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})) \\ - \mathbb{E}_{q_E(\boldsymbol{z}|\boldsymbol{x}) q_D(\boldsymbol{c}|\boldsymbol{x})} \left[\log p_G(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{c})\right],\end{aligned} \tag{4.1.4}$$

where $\text{KL}(\cdot\|\cdot)$ is the KL-divergence; and $q_D(\boldsymbol{c}|\boldsymbol{x})$ is the conditional distribution defined by the discriminator D for each structured variable in $\boldsymbol{c}$:

$$D(\boldsymbol{x}) = q_D(\boldsymbol{c}|\boldsymbol{x}). \tag{4.1.5}$$

Here, for simplicity of notations, we assume only one structured variable and thus one discriminator, though our model specification can straightforwardly be applied to many attributes. The distribution over $(\boldsymbol{z}, \boldsymbol{c})$ factors into q_E and q_D as we are learning disentangled representations. Note that here the discriminator D and code $\boldsymbol{c}$ are not learned with the VAE loss, but instead optimized with the objectives described shortly. Besides the reconstruction loss which drives the generator to produce realistic sentences, the discriminator provides extra learning signals which enforce the generator to produce coherent attribute that matches the structured code in $\boldsymbol{c}$. However, as it is impossible to propagate gradients from the discriminator through the discrete samples, we resort to a deterministic continuous approximation. The approximation replaces the sampled token $\hat{x}_t$ (represented as a one-hot vector) at each step with the probability vector in Eq.(4.1.2) which is differentiable w.r.t the generator's parameters. The probability vector is used as the output at the current step and the input to the next step along the sequence of decision making. The resulting "soft" generated sentence, denoted as $\widetilde{G}_\tau(\boldsymbol{z}, \boldsymbol{c})$, is fed into the discriminator[1] to measure the fitness to the target attribute, leading to the following loss for improving G:

$$\mathcal{L}_{\text{Attr},c}(\boldsymbol{\theta}_G) = -\mathbb{E}_{p(\boldsymbol{z})p(\boldsymbol{c})}\left[\log q_D(\boldsymbol{c}|\widetilde{G}_\tau(\boldsymbol{z}, \boldsymbol{c}))\right]. \tag{4.1.6}$$

The temperature τ (Eq.4.1.2) is set to $\tau \to 0$ as training proceeds, yielding increasingly peaked distributions that finally emulate discrete case. The simple deterministic approximation effectively leads to reduced variance and fast convergence during training, which enables efficient learning of the conditional generator. The diversity of generation results is guaranteed since we use the approximation only for attribute modeling and the base sentence generation is learned through VAEs.

With the objective in Eq.(4.1.6), each structured attribute of generated sentences is controlled through the corresponding code in $\boldsymbol{c}$ and is independent with other variables in the latent representation. However, it is still possible that other attributes not explicitly modeled may also entangle with the code in $\boldsymbol{c}$, and thus varying a dimension of $\boldsymbol{c}$ can yield unexpected variation of these attributes we are not interested in. To address this, we introduce the independency constraint which separates these attributes with $\boldsymbol{c}$ by enforcing them to be fully captured by the unstructured part $\boldsymbol{z}$. Therefore, besides the attributes explicitly encoded in $\boldsymbol{c}$, we also train the generator so that other non-explicit attributes can be correctly recognized from the generated samples and match the unstructured code $\boldsymbol{z}$. Instead of building a new discriminator, we reuse the variational encoder E which serves precisely to infer the latents $\boldsymbol{z}$ in the base VAE. The loss is in the same form as with Eq.(4.1.6) except replacing the discriminator conditional q_D with the encoder conditional q_E:

$$\mathcal{L}_{\text{Attr},z}(\boldsymbol{\theta}_G) = -\mathbb{E}_{p(\boldsymbol{z})p(\boldsymbol{c})}\left[\log q_E(\boldsymbol{z}|\widetilde{G}_\tau(\boldsymbol{z}, \boldsymbol{c}))\right]. \tag{4.1.7}$$

Note that, as the discriminator in Eq.(4.1.6), the encoder now performs inference over generated samples from the prior, as opposed to observed examples as in VAEs.

Combining Eqs.(4.1.4)-(4.1.7) we obtain the generator objective:

$$\min\nolimits_{\boldsymbol{\theta}_G} \mathcal{L}_G = \mathcal{L}_{\text{VAE}} + \lambda_c \mathcal{L}_{\text{Attr},c} + \lambda_z \mathcal{L}_{\text{Attr},z}, \tag{4.1.8}$$

where λ_c and λ_z are balancing parameters. The variational encoder is trained by minimizing the VAE loss, i.e., $\min_{\boldsymbol{\theta}_E} \mathcal{L}_{\text{VAE}}$.

[1] The probability vector thus functions to average over the word embedding matrix to obtain a "soft" word embedding at each step.

Algorithm 2 Controlled Generation of Text

```
Input: A large corpus of unlabeled sentences $\mathcal{X} = \{\boldsymbol{x}\}$
       A few sentence attribute labels $\mathcal{X}_L = \{(\boldsymbol{x}_L, \boldsymbol{c}_L)\}$
       Parameters: $\lambda_c, \lambda_z, \lambda_u, \beta$ – balancing parameters
Initialize the base VAE by minimizing Eq.(4.1.4) on $\mathcal{X}$ with $\boldsymbol{c}$ sampled from prior $p(\boldsymbol{c})$
repeat
    Train the discriminator $D$ by Eq.(4.1.11)
    Train the generator $G$ and the encoder $E$ by Eq.(4.1.8) and minimizing Eq.(4.1.4), respectively.
until convergence
Output: Sentence generator $G$ conditioned on disentangled representation $(\boldsymbol{z}, \boldsymbol{c})$
```

Discriminator Learning. The discriminator D is trained to accurately infer the sentence attribute and evaluate the error of recovering the desired feature as specified in the latent code. For instance, for categorical attribute, the discriminator can be formulated as a sentence classifier; while for continuous target a probabilistic regressor can be used. The discriminator is learned in a different way compared to the VAE encoder, since the target attributes can be discrete which are not supported in the VAE framework. Moreover, in contrast to the unstructured code $\boldsymbol{z}$ which is learned in an unsupervised manner, the structured variable $\boldsymbol{c}$ uses labeled examples to entail designated semantics. We derive an efficient semi-supervised learning method for the discriminator.

Formally, let $\boldsymbol{\theta}_D$ denote the parameters of the discriminator. To learn specified semantic meaning, we use a set of labeled examples $\mathcal{X}_L = \{(\boldsymbol{x}_L, \boldsymbol{c}_L)\}$ to train the discriminator D with the following objective:

$$\mathcal{L}_s(\boldsymbol{\theta}_D) = -\mathbb{E}_{\mathcal{X}_L}\left[\log q_D(\boldsymbol{c}_L|\boldsymbol{x}_L)\right]. \tag{4.1.9}$$

Besides, the conditional generator G is also capable of synthesizing (noisy) sentence-attribute pairs $(\hat{\boldsymbol{x}}, \boldsymbol{c})$ which can be used to augment training data for semi-supervised learning. To alleviate the issue of noisy data and ensure robustness of model optimization, we incorporate a minimum entropy regularization term (Grandvalet et al., 2004; Reed et al., 2014). The resulting objective is thus:

$$\mathcal{L}_u(\boldsymbol{\theta}_D) = -\mathbb{E}_{p_G(\hat{\boldsymbol{x}}|\boldsymbol{z},\boldsymbol{c})p(\boldsymbol{z})p(\boldsymbol{c})}\big[\log q_D(\boldsymbol{c}|\hat{\boldsymbol{x}}) + \beta\mathcal{H}(q_D(\boldsymbol{c}'|\hat{\boldsymbol{x}}))\big], \tag{4.1.10}$$

where $\mathcal{H}(q_D(\boldsymbol{c}'|\hat{\boldsymbol{x}}))$ is the empirical Shannon entropy of distribution q_D evaluated on the generated sentence $\hat{\boldsymbol{x}}$; and β is the balancing parameter. Intuitively, the minimum entropy regularization encourages the model to have high confidence in predicting labels.

The joint training objective of the discriminator using both labeled examples and synthesized samples is then given as:

$$\min_{\boldsymbol{\theta}_D} \mathcal{L}_D = \mathcal{L}_s + \lambda_u\mathcal{L}_u, \tag{4.1.11}$$

where λ_u is the balancing parameter.

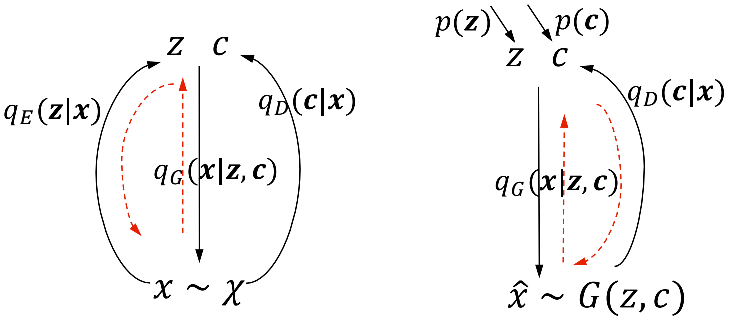

z
c
$q_E(\boldsymbol{z}|\boldsymbol{x})$
$q_D(\boldsymbol{c}|\boldsymbol{x})$
$q_G(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{c})$
$x \sim \chi$
$p(\boldsymbol{z})$
$p(\boldsymbol{c})$
z
c
$q_D(\boldsymbol{c}|\boldsymbol{x})$
$q_G(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{c})$
$\hat{x} \sim G(z,c)$

Figure 4.2: Left: The VAE and wake procedure, corresponding to Eq.(4.1.4). **Right:** The sleep procedure, corresponding to Eqs.(4.1.6)-(4.1.7) and (4.1.10). Black arrows denote inference and generation; red dashed arrows denote gradient propagation. The two steps in the sleep procedure, i.e., optimizing the discriminator and the generator, respectively, are performed in an alternating manner.

4.1.3.3 Summarization and Discussion

We have derived our model and its learning procedure. The generator is first initialized by training the base VAE on a large corpus of unlabeled sentences, through the objective of minimizing Eq.(4.1.4) with the latent code $\boldsymbol{c}$ at this time sampled from the prior distribution $p(\boldsymbol{c})$. The full model is then trained by alternating the optimization of the generator and the discriminator, as summarized in Algorithm 2.

Our model can be viewed as combining the VAE framework with an extended wake-sleep method, as illustrated in Figure 4.2. Specifically, in Eq.(4.1.10), samples are produced by the generator and used as targets for maximum likelihood training of the discriminator. This resembles the sleep phase of wake-sleep. Eqs.(4.1.6)-(4.1.7) further leverage the generated samples to improve the generator. We can see the above together as an extended sleep procedure based on "dream" samples obtained by ancestral sampling from the generative network. On the other hand, Eq.(4.1.4) samples $\boldsymbol{c}$ from the discriminator distribution $q_D(\boldsymbol{c}|\boldsymbol{x})$ on observation $\boldsymbol{x}$, to form a target for training the generator, which corresponds to the wake phase. The effective combination enables discrete latent code, holistic discriminator metrics, and efficient mutual bootstrapping.

Training of the discriminators need supervised data to impose designated semantics. Discriminators for different attributes can be trained independently on separate labeled sets. That is, the model does not require a sentence to be annotated with all attributes, but instead needs only independent labeled data for each individual attribute. Moreover, as the labeled data are used only for learning attribute semantics instead of direct sentence generation, we are allowed to extend the data scope beyond labeled sentences to, e.g., labeled words or phrases. As shown in the experiments (section 4.1.4), our method is able to effectively lift the word level knowledge to sentence level and generate convincing sentences. Finally, with the augmented unsupervised training in the sleep phrase, we show a little supervision is sufficient for learning structured representations.

4.1.4 Experiments

We apply our model to generate short sentences (length $\leq$ 15) with controlled sentiment and tense. Quantitative experiments using trained classifiers as evaluators show our model gives improved generation accuracy. Disentangled representation is learned with a few labels or only word annotations. We also validate the effect of the proposed independency constraint for interpretable generation.

Datasets

Sentence corpus. We use a large IMDB text corpus (Diao et al., 2014) for training the generative models. This is a collection of 350K movie reviews. We select sentences containing at most 15 words, and replace infrequent words with the token "<unk>". The resulting dataset contains around 1.4M sentences with the vocabulary size of 16K.

Sentiment. To control the sentiment ("positive" or "negative") of generated sentences, we test on the following labeled sentiment data: (1) Stanford Sentiment Treebank-2 (**SST-full**) (Socher et al., 2013) consists of 6920/872/1821 movie review sentences with binary sentiment annotations in the train/dev/test sets, respectively. We use the 2837 training examples with sentence length ≤ 15, and evaluate classification accuracy on the original test set. (2) **SST-small.** To study the size of labeled data required in the semi-supervised learning for accurate attribute control, we sample a small subset from SST-full, containing only 250 labeled sentences for training. (3) **Lexicon.** We also investigate the effectiveness of our model in terms of using word-level labels for sentence-level control. The lexicon from (Wilson et al., 2005) contains 2700 words with sentiment labels. We use the lexicon for training by treating the words as sentences, and evaluate on the SST-full test set. (4) **IMDB.** We collect a dataset from the IMDB corpus by randomly selecting positive and negative movie reviews. The dataset has 5K/1K/10K sentences in train/dev/test.

Tense. The second attribute is the tense of the main verb in a sentence. Though no corpus with sentence tense annotations is readily available, our method is able to learn from only labeled words and generate desired sentences. We compile from the TimeBank (timeml.org) dataset and obtain a lexicon of 5250 words and phrases labeled with one of {"past", "present", "future"}. The lexicon mainly consists of verbs in different tenses (e.g., "was", "will be") as well as time expressions (e.g., "in the future").

Note that our method requires only separate labeled copora for each attribute. And for the tense attribute only annotated words/phrases are used.

Parameter Setting

The generator and encoder are set as single-layer LSTM RNNs with input/hidden dimension of 300 and max sample length of 15. Discriminators are set as ConvNets. Detailed configurations are in the supplements. To avoid vanishingly small KL term in the VAE module (Eq.4.1.4) (Bowman et al., 2015), we use a KL term weight linearly annealing from 0 to 1 during training. Balancing parameters are set to $\lambda_c = \lambda_z = \lambda_u = 0.1$, and β is selected on the dev sets. At test time sentences are generated with Eq.(4.1.1).

Model	Dataset		
	SST-full	SST-small	Lexicon
S-VAE	0.822	0.679	0.660
Ours	**0.851**	**0.707**	**0.701**

Table 4.1: Sentiment accuracy of generated sentences. S-VAE (Kingma et al., 2014) and our model are trained on the three sentiment datasets and generate 30K sentences, respectively.

4.1.4.1 Accuracy of Generated Attributes

We quantitatively measure sentence attribute control by evaluating the accuracy of generating designated sentiment, and the effect of using samples for training classifiers. We compare with semi-supervised VAE (S-VAE) (Kingma et al., 2014), one of the few existing deep models capable of conditional text generation. S-VAE learns to reconstruct observed sentences given attribute code, and no discriminators are used.

We use a state-of-the-art sentiment classifier (Hu et al., 2016a) which achieves 90% accuracy on the SST test set, to automatically evaluate the sentiment generation accuracy. Specifically, we generate sentences given sentiment code $\boldsymbol{c}$, and use the pre-trained sentiment classifier to assign sentiment labels to the generated sentences. The accuracy is calculated as the percentage of the predictions that match the sentiment code $\boldsymbol{c}$. Table 4.1 shows the results on 30K sentences by the two models which are trained with SST-full, SST-small, and Lexicon, respectively. We see that our method consistently outperforms S-VAE on all datasets. In particular, trained with only 250 labeled examples in SST-small, our model achieves reasonable generation accuracy, demonstrating the ability of learning disentangled representations with very little supervision. More importantly, given only word-level annotations in Lexicon, our model successfully transfers the knowledge to sentence level and generates desired sentiments reasonably well. Compared to our method that drives learning by directly assessing generated sentences, S-VAE attempts to capture sentiment semantics only by reconstructing labeled words, which is less efficient and gives inferior performance.

We next use the generated samples to augment the sentiment datasets and train sentiment classifiers. While not aiming to build best-performing classifiers on these datasets, the classification accuracy serves as an auxiliary measure of the sentence generation quality. That is, higher-quality sentences with more accurate sentiment attribute can predictably help yield stronger sentiment classifiers. Figure 4.3 shows the accuracy of classifiers trained on the four datasets with different augmentations. "Std" is a ConvNet trained on the standard original datasets, with the same network structure as with the sentiment discriminator in our model. "H-reg" additionally imposes the minimum entropy regularization on the generated sentences. "Ours" incorporates the minimum entropy regularization and the sentiment attribute code $\boldsymbol{c}$ of the generated sentences, as in Eq.(4.1.10). S-VAE uses the same protocol as our method to augment with the data generated by the S-VAE model. Comparison in Figure 4.3 shows that our method consistently gives the best performance on four datasets. For instance, on Lexicon, our approach achieves 0.733 accuracy, compared to 0.701 of "Std". The improvement of "H-Reg" over "Std" shows positive effect of

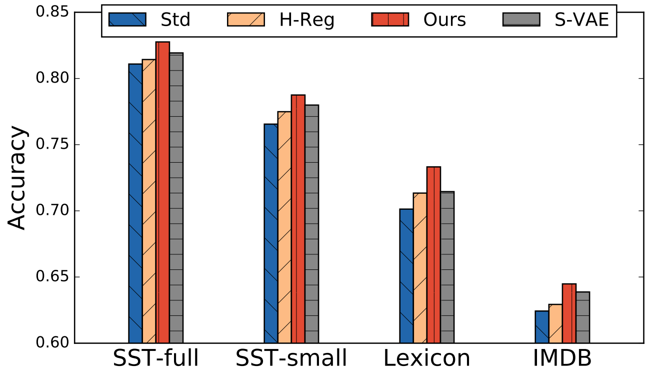


Figure 4.3: Test-set accuracy of classifiers trained on four sentiment datasets augmented with different methods (see text for details). The first three datasets use the SST-full test set for evaluation.

the minimum entropy regularization on generated sentences. Further incorporating the conditioned sentiment code of the generated samples, as in “Ours” and “S-VAE”, provides additional performance gains, indicating the advantages of conditional generation for automatic creation of labeled data. Consistent with the above experiment, our model outperforms S-VAE.

4.1.4.2 Disentangled Representation

We study the interpretability of generation and the explicit independency constraint (Eq.4.1.7) for disentangled control.

Table 4.2 compares the samples generated by models with and without the constraint term, respectively. In the left column where the constraint applies, each pair of sentences, conditioned on different sentiment codes, are highly relevant in terms of, e.g., subject, tone, and wording which are not explicitly modeled in the structured code $\boldsymbol{c}$ while instead implicitly encoded in the unstructured code $\boldsymbol{z}$. Varying the sentiment code precisely changes the sentiment of the sentences (and paraphrases slightly to ensure fluency), while keeping other aspects unchanged. In contrast, the results in the right column, where the independency constraint is unactivated, show that varying the sentiment code not only changes the polarity of samples, but can also change other aspects unexpected to control, making the generation results less interpretable and predictable.

We demonstrate the power of learned disentangled representation by varying one attribute variable at a time. Table 4.3 shows the generation results. We see that each attribute variable in our model successfully controls its corresponding attribute, and is disentangled with other attribute code. The right column of the table shows meaningful variation of sentence tense as the tense code varies. Note that the semantic of tense is learned only from a lexicon without complete sentence examples. Our model successfully captures the key ingredients (e.g., verb “was” for past tense and “will be” for future tense) and combines with the knowledge of well-formed sentences to generate realistic samples with specified tense attributes. Table 4.4 further shows generated sentences with varying code $\boldsymbol{z}$ in different settings of structured attribute factors. We obtain samples that are diverse in content while consistent in sentiment and tense.

w/ independency constraint	w/o independency constraint
the film is strictly routine !	the acting is bad .
the film is full of imagination .	the movie is so much fun .
after watching this movie , i felt that disappointed .	none of this is very original .
after seeing this film , i 'm a fan .	highly recommended viewing for its courage , and ideas .
the acting is uniformly bad either .	too bland
the performances are uniformly good .	highly watchable
this is just awful .	i can analyze this movie without more than three words .
this is pure genius .	i highly recommend this film to anyone who appreciates music .

Table 4.2: Samples from models with or without independency constraint on attribute control (i.e., Eq.4.1.7). Each pair of sentences are generated with sentiment code set to "negative" and "positive", respectively, while fixing the unstructured code $\boldsymbol{z}$. The SST-full dataset is used for learning the sentiment representation.

Varying the code of tense	
i thought the movie was too bland and too much	this was one of the outstanding thrillers of the last decade
i guess the movie is too bland and too much	this is one of the outstanding thrillers of the all time
i guess the film will have been too bland	this will be one of the great thrillers of the all time

Table 4.3: Each triple of sentences is generated by varying the tense code while fixing the sentiment code and $\boldsymbol{z}$.

Varying the unstructured code $\boldsymbol{z}$	
("negative", "past")	*("positive", "past")*
the acting was also kind of hit or miss .	his acting was impeccable
i wish i 'd never seen it	this was spectacular , i saw it in theaters twice
by the end i was so lost i just did n't care anymore	it was a lot of fun
("negative", "present")	*("positive", "present")*
the movie is very close to the show in plot and characters	this is one of the better dance films
the era seems impossibly distant	i 've always been a big fan of the smart dialogue .
i think by the end of the film , it has confused itself	i recommend you go see this, especially if you hurt
("negative", "future")	*("positive", "future")*
i wo n't watch the movie	i hope he 'll make more movies in the future
and that would be devastating !	i will definitely be buying this on dvd
i wo n't get into the story because there really is n't one	you will be thinking about it afterwards, i promise you

Table 4.4: Samples by varying the unstructured code $\boldsymbol{z}$ given sentiment ("positive"/"negative") and tense ("past"/"present"/"future") code.

Failure cases	
the plot is not so original	it does n’t get any better the other dance movies
the plot weaves us into <unk>	it does n’t reach them , but the stories look
he is a horrible actor ’s most part	i just think so
he ’s a better actor than a standup	i just think !

Table 4.5: Failure cases when varying sentiment code with other codes fixed.

We also occasionally observed failure cases as in Table 4.5, such as implausible sentences, unexpected variations of irrelevant attributes, and inaccurate attribute generations. Improved modeling is expected such as using dilated convolutions as decoder, and decoding with beam search, etc. Better systematic quantitative evaluations are also desired.

4.1.5 Discussion

We have proposed a deep generative model that learns interpretable latent representations and generates sentences with specified attributes. We obtained meaningful generation with restricted sentence length, and improved accuracy on sentiment and tense attributes. Our approach combines VAEs with auxiliary attribute discriminators and imposes explicit independency constraints on attribute controls, enabling disentangled latent code. Semi-supervised learning within the joint VAE/wake-sleep framework is effective with little or incomplete data supervision. Hu et al. (2017c) develop a unified view of a diverse set of deep generative paradigms, including GANs, VAEs, and wake-sleep algorithm. Our model can be alternatively motivated under the view as enhancing VAEs with the extended sleep phase and by leveraging generated samples.

Interpretability of the latent representations not only allows dynamic control of generated attributes, but also provides an interface that connects the end-to-end neural model with conventional structured methods. For instance, we can encode structured constraints (e.g., logic rules or probabilistic structured models) on the interpretable latent code, to incorporate prior knowledge or human goals (Hu et al., 2016a,b); or plug the disentangled generation model into dialog systems to generate natural language responses from structured dialog states (Young et al., 2013).

Though we have focused on the generation capacity of our model, the proposed collaborative semi-supervised learning framework also helps improve the discriminators by generating labeled samples for data augmentation (e.g., see Figure 4.3). More generally, for any discriminative task, we can build a conditional generative model to synthesize additional labeled data. The accurate attribute generation of our approach can offer larger performance gains compared to previous generative methods.

4.2 Learning with Auxiliary Language Models for Text Style Transfer

4.2.1 Introduction

Recently there has been growing interest in designing natural language generation (NLG) systems that allow for control over various attributes of generated text – for example, sentiment and other stylistic properties. Such controllable NLG models have wide applications in dialogues systems (Wen et al., 2016) and other natural language interfaces. Recent successes for neural text generation models in machine translation (Bahdanau et al., 2014), image captioning (Vinyals et al., 2015b) and dialogue (Vinyals and Le, 2015; Wen et al., 2016) have relied on massive parallel data. However, for many other domains, only non-parallel data – which includes collections of sentences from each domain without explicit correspondence – is available. Many text style transfer problems fall into this category. The goal for these tasks is to transfer a sentence with one attribute to a sentence with an another attribute, but with the same style-independent content, trained using only non-parallel data.

Unsupervised text style transfer requires learning disentangled representations of attributes (e.g., negative/positive sentiment, plaintext/ciphertext orthography) and underlying content. This is challenging because the two interact in subtle ways in natural language and it can even be hard to disentangle them with parallel data. The recent development of deep generative models like variational auto-encoders (VAEs) (Kingma and Welling, 2013) and generative adversarial networks(GANs) (Goodfellow et al., 2014) have made learning disentangled representations from non-parallel data possible. However, despite their rapid progress in computer vision—for example, generating photo-realistic images (Radford et al., 2015b), learning interpretable representations (Chen et al., 2016b), and translating images (Zhu et al., 2017)—their progress on text has been more limited. For VAEs, the problem of training collapse can severely limit effectiveness (Bowman et al., 2015; Yang et al., 2017b), and when applying adversarial training to natural language, the non-differentiability of discrete word tokens makes generator optimization difficult. Hence, most attempts use REINFORCE (Sutton et al., 2000) to finetune trained models (Yu et al., 2017b; Li et al., 2017) or uses professor forcing (Lamb et al., 2016) to match hidden states of decoders.

Previous work on unsupervised text style transfer (Hu et al., 2017b; Shen et al., 2017) adopts an encoder-decoder architecture with style discriminators to learn disentangled representations. The encoder takes a sentence as an input and outputs a style-independent content representation. The style-dependent decoder takes the content representation and a style representation and generates the transferred sentence. (Hu et al., 2017b) use a style classifier to directly enforce the desired style in the generated text. (Shen et al., 2017) leverage an adversarial training scheme where a binary CNN-based discriminator is used to evaluate whether a transferred sentence is real or fake, ensuring that transferred sentences match real sentences in terms of target style. However, in practice, the error signal from a binary classifier is sometimes insufficient to train the generator to produce fluent language, and optimization can be unstable as a result of the adversarial training step.

We propose to use an implicitly trained language model as a new type of discriminator, replacing the more conventional binary classifier. The language model calculates a sentence's likelihood, which decomposes into a product of token-level conditional probabilities. In our approach, rather than training a binary classifier to distinguish real and fake sentences, we train the language model to assign a high probability to real sentences and train the generator to produce sentences with high probability under the language model. Because the language model scores sentences directly using a product of locally normalized probabilities, it may offer more stable and more useful training signal to the generator. Further, by using a continuous approximation of discrete sampling under the generator, our model can be trained using back-propagation in an end-to-end fashion.

We find empirically that when using the language model as a structured discriminator, it is possible to eliminate adversarial training steps that use negative samples—a critical part of traditional adversarial training. Language models are *implicitly* trained to assign a low probability to negative samples because of its normalization constant. By eliminating the adversarial training step, we found the training becomes more stable in practice.

To demonstrate the effectiveness of our new approach, we conduct experiments on three tasks: word substitution decipherment, sentiment modification, and related language translation. We show that our approach, which uses only a language model as the discriminator, outperforms a broad set of state-of-the-art approaches on the three tasks.

4.2.2 Unsupervised Text Style Transfer

We start by reviewing the current approaches for unsupervised text style transfer (Hu et al., 2017b; Shen et al., 2017), and then go on to describe our approach in Section 4.2.3. Assume we have two text datasets $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}\}$ and $\mathbf{Y} = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(n)}\}$ with two different styles $\mathbf{v}_x$ and $\mathbf{v}_y$, respectively. For example, $\mathbf{v}_x$ can be the positive sentiment style and $\mathbf{v}_y$ can be the negative sentiment style. The datasets are non-parallel such that the data does not contain pairs of $(\mathbf{x}^{(i)}, \mathbf{y}^{(j)})$ that describe the same content. The goal of style transfer is to transfer data $\mathbf{x}$ with style $\mathbf{v_x}$ to style $\mathbf{v_y}$ and vice versa, i.e., to estimate the conditional distribution $p(\mathbf{y}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{y})$. Since text data is discrete, it is hard to learn the transfer function directly via back-propagation as in computer vision (Zhu et al., 2017). Instead, we assume the data is generated conditioned on two disentangled parts, the style $\mathbf{v}$ and the content $\mathbf{z}$[2] (Hu et al., 2017b).

Consider the following generative process for each style: 1) the style representation $\mathbf{v}$ is sampled from a prior $p(\mathbf{v})$; 2) the content vector $\mathbf{z}$ is sampled from $p(\mathbf{z})$; 3) the sentence $\mathbf{x}$ is generated from the conditional distribution $p(\mathbf{x}|\mathbf{z}, \mathbf{v})$. This model suggests the following parametric form for style transfer where q represents a posterior:

$$p(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{z_x}} p(\mathbf{y}|\mathbf{z_x}, \mathbf{v_y}) q(\mathbf{z_x}|\mathbf{x}, \mathbf{v_x}) d\mathbf{z_x}.$$

The above equation suggests the use of an encoder-decoder framework for style transfer problems. We can first encode the sentence $\mathbf{x}$ to get its content vector $\mathbf{z_x}$, then we switch the style

[2] We drop the subscript in notations wherever the meaning is clear.

label from $\mathbf{v_x}$ to $\mathbf{v_y}$. Combining the content vector $\mathbf{z_x}$ and the style label $\mathbf{v_y}$, we can generate a new sentence $\tilde{\mathrm{x}}$ (the transferred sentences are denotes as $\tilde{\mathrm{x}}$ and $\tilde{\mathrm{y}}$).

One unsupervised approach is to use the auto-encoder model. We first use an encoder model $\mathbf{E}$ to encode $\mathbf{x}$ and $\mathbf{y}$ to get the content vectors $\mathbf{z_x} = \mathbf{E}(\mathbf{x}, \mathbf{v_x})$ and $\mathbf{z_y} = \mathbf{E}(\mathbf{y}, \mathbf{v_y})$. Then we use a decoder $\mathbf{G}$ to generate sentences conditioned on $\mathbf{z}$ and $\mathbf{v}$. The $\mathbf{E}$ and $\mathbf{G}$ together form an auto-encoder and the reconstruction loss is:

$$\mathcal{L}_{\text{rec}}(\theta_{\mathbf{E}}, \theta_{\mathbf{G}}) = \mathbb{E}_{\mathbf{x}\sim\mathbf{X}}[-\log p_{\mathbf{G}}(\mathbf{x}|\mathbf{z_x}, \mathbf{v_x})] + \mathbb{E}_{\mathbf{y}\sim\mathbf{Y}}[-\log p_{\mathbf{G}}(\mathbf{y}|\mathbf{z_y}, \mathbf{v_y})],$$

where $\mathbf{v_x}$ and $\mathbf{v_y}$ can be two learnable vectors to represent the label embedding. In order to make sure that the $\mathbf{z_x}$ and $\mathbf{z_y}$ capture the content and we can deliver accurate transfer between the style by switching the labels, we need to guarantee that $\mathbf{z_x}$ and $\mathbf{z_y}$ follow the same distribution. We can assume $p(\mathbf{z})$ follows a prior distribution and add a KL-divergence regularization on $\mathbf{z_x}$, $\mathbf{z_y}$. The model then becomes a VAE. However, previous works (Bowman et al., 2015; Yang et al., 2017b) found that there is a training collapse problem with the VAE for text modeling and the posterior distribution of $\mathbf{z}$ fails to capture the content of a sentence.

To better capture the desired styles in the generated sentences, (Hu et al., 2017b) additionally impose a style classifier on the generated samples, and the decoder $\mathbf{G}$ is trained to generate sentences that maximize the accuracy of the style classifier. Such additional supervision with a *discriminative* model is also adopted in (Shen et al., 2017), though in that work a binary real/fake classifier is instead used within a conventional adversarial scheme.

Adversarial Training (Shen et al., 2017) use adversarial training to align the $\mathbf{z}$ distributions. Not only do we want to align the distribution of $\mathbf{z_x}$ and $\mathbf{z_y}$, but also we hope that the transferred sentence $\tilde{\mathrm{x}}$ from $\mathbf{x}$ to resemble $\mathbf{y}$ and vice versa. Several adversarial discriminators are introduced to align these distributions. Each of the discriminators is a binary classifier distinguishing between real and fake. Specifically, the discriminator $D_{\mathbf{z}}$ aims to distinguish between $\mathbf{z_x}$ and $\mathbf{z_y}$:

$$\mathcal{L}^{\mathbf{z}}_{\text{adv}}(\theta_{\mathbf{E}}, \theta_{\mathbf{D_z}}) = \mathbb{E}_{\mathbf{x}\sim\mathbf{X}}[-\log D_{\mathbf{z}}(\mathbf{z_x})] + \mathbb{E}_{\mathbf{y}\sim\mathbf{Y}}[-\log(1 - D_{\mathbf{z}}(\mathbf{z_y}))].$$

Similarly, $D_{\mathbf{x}}$ distinguish between $\mathbf{x}$ and $\tilde{\mathrm{y}}$, yielding an objective $\mathcal{L}^{\mathbf{x}}_{\text{adv}}$ as above; and $D_{\mathbf{y}}$ distinguish between $\mathbf{y}$ and $\tilde{\mathrm{x}}$, yielding $\mathcal{L}^{\mathbf{y}}_{\text{adv}}$. Since the samples of $\tilde{\mathrm{x}}$ and $\tilde{\mathrm{y}}$ are discrete and it is hard to train the generator in an end-to-end way, professor forcing (Lamb et al., 2016) is used to match the distributions of the hidden states of decoders. The overall training objective is a min-max game played among the encoder $\mathbf{E}$/decoder $\mathbf{G}$ and the discriminators $D_{\mathbf{z}}, D_{\mathbf{x}}, D_{\mathbf{y}}$ (Goodfellow et al., 2014):

$$\min_{E,G} \max_{D_{\mathbf{z}}, D_{\mathbf{x}}, D_{\mathbf{y}}} \mathcal{L}_{\text{rec}} - \lambda(\mathcal{L}^{\mathbf{z}}_{\text{adv}} + \mathcal{L}^{\mathbf{x}}_{\text{adv}} + \mathcal{L}^{\mathbf{y}}_{\text{adv}})$$

The model is trained in an alternating manner. In the first step, the loss of the discriminators are minimize to distinguish between the $\mathbf{z_x}, \mathbf{x}, \mathbf{y}$ and $\mathbf{z_y}, \tilde{\mathrm{x}}, \tilde{\mathrm{y}}$, respectively; and in the second step the encoder and decoder are trained to minimize the reconstruction loss while maximizing loss of the discriminators.

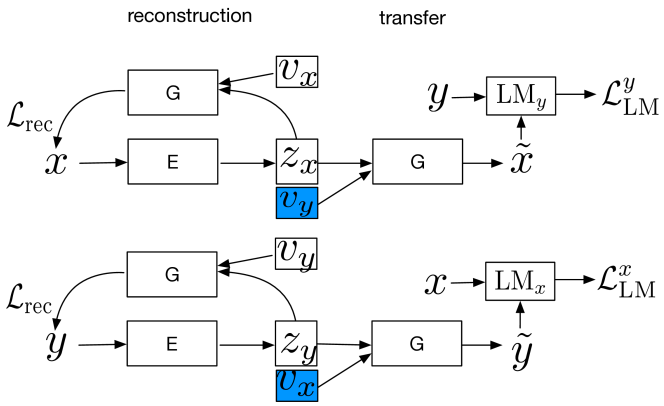


Figure 4.4: The overall model architecture consists of two parts: reconstruction and transfer. For transfer, we switch the style label and sample an output sentence from the generator that is evaluated by a language model.

4.2.3 Language Models as Discriminators

In most past work, a classifier is used as the discriminator to distinguish whether a sentence is real or fake. We propose instead to use locally-normalized language models as discriminators. We argue that using an explicit language model with token-level locally normalized probabilities offers a more direct training signal to the generator. If a transfered sentence does not match the target style, it will have high perplexity when evaluated by a language model that was trained on target domain data. Not only does it provide an overall evaluation score for the whole sentence, but a language model can also assign a probability to each token, thus providing more information on which word is to blame if the overall perplexity is very high.

The overall model architecture is shown in Figure 4.4. Suppose $\tilde{\mathbf{x}}$ is the output sentence from applying style transfer to input sentence $\mathbf{x}$, i.e., $\tilde{\mathbf{x}}$ is sampled from $p_G(\tilde{\mathbf{x}}|\mathbf{z}_{\mathbf{x}}, \mathbf{v}_{\mathbf{y}})$ (and similary for $\tilde{\mathbf{y}}$ and $\mathbf{y}$). Let $p_{\text{LM}}(\mathbf{x})$ be the probability of a sentence $\mathbf{x}$ evaluate against a language model, then the discriminator loss becomes:

$$\mathcal{L}^{\mathbf{x}}_{\text{LM}}(\theta_{\mathbf{E}}, \theta_{\mathbf{G}}, \theta_{\text{LM}_{\mathbf{x}}}) = \mathbb{E}_{\mathbf{x}\sim\mathbf{X}}[-\log p_{\text{LM}_{\mathbf{x}}}(\mathbf{x}))] + \gamma\mathbb{E}_{\mathbf{y}\sim\mathbf{Y},\tilde{\mathbf{y}}\sim p_G(\tilde{\mathbf{y}}|\mathbf{z}_{\mathbf{y}},\mathbf{v}_{\mathbf{x}})}[\log p_{\text{LM}_{\mathbf{x}}}(\tilde{\mathbf{y}})], \tag{4.2.1}$$

$$\mathcal{L}^{\mathbf{y}}_{\text{LM}}(\theta_{\mathbf{E}}, \theta_{\mathbf{G}}, \theta_{\text{LM}_{\mathbf{y}}}) = \mathbb{E}_{\mathbf{y}\sim\mathbf{Y}}[-\log p_{\text{LM}_{\mathbf{y}}}(\mathbf{y}))] + \gamma\mathbb{E}_{\mathbf{x}\sim\mathbf{X},\tilde{\mathbf{x}}\sim p_G(\tilde{\mathbf{x}}|\mathbf{z}_{\mathbf{x}},\mathbf{v}_{\mathbf{y}})}[\log p_{\text{LM}_{\mathbf{y}}}(\tilde{\mathbf{x}})]. \tag{4.2.2}$$

Our overall objective becomes:

$$\min_{E,G} \max_{\text{LM}_{\mathbf{x}},\text{LM}_{\mathbf{y}}} \mathcal{L}_{\text{rec}} - \lambda(\mathcal{L}^{\mathbf{x}}_{\text{LM}} + \mathcal{L}^{\mathbf{y}}_{\text{LM}}) \tag{4.2.3}$$

Negative samples: Note that Equation 4.2.1 and 4.2.2 differs from traditional ways of training language models in that we have a term including the negative samples. We train the LM in an adversarial way by minimizing the loss of LM of real sentences and maximizing the loss of transferred sentences. However, since the LM is a structured discriminator, we would hope that a language model trained on the real sentences will automatically assign high perplexity to sentences not in the target domain, hence negative samples from the generator may not be

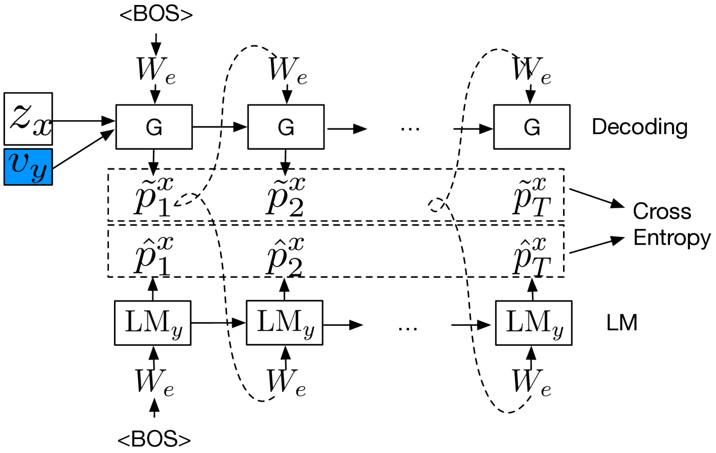


Figure 4.5: Continuous approximation of language model loss. The input is a sequence of probability distributions $\{\tilde{\mathbf{p}}_t^{\mathbf{x}}\}_{t=1}^T$ sampled from the generator. At each timestep, we compute a weighted embedding as input to the language model and get the sequence of output distributions from the LM as $\{\hat{\mathbf{p}}_t^{\mathbf{x}}\}_{t=1}^T$. The loss is the sum of cross entropies between each pair of $\tilde{\mathbf{p}}_t^{\mathbf{x}}$ and $\hat{\mathbf{p}}_t^{\mathbf{x}}$.

necessary. To investigate the necessity of negative samples, we add a weight γ to the loss of negative samples. The weight γ adjusts the negative sample loss in training the language models. If $\gamma = 0$, we simply train the language model on real sentences and fix its parameters, avoiding potentially unstable adversarial training steps. We investigate the necessity of using negative samples in the experiment section.

Training consists of two steps alternatively. In the first step, we train the language models according to Equation 4.2.1 and 4.2.2. In the second step, we minimize the reconstruction loss as well as the perplexity of generated samples evaluated by the language model. Since $\tilde{\mathbf{x}}$ is discrete, one can use the REINFORCE (Sutton et al., 2000) algorithm to train the generator:

$$\nabla_{\theta_G} \mathcal{L}_{\mathrm{LM}}^{\mathbf{y}} = \mathbb{E}_{\mathbf{x}\sim\mathbf{X},\tilde{\mathbf{x}}\sim p_G(\tilde{\mathbf{x}}|\mathbf{z_x},\mathbf{v_y})}[\log p_{\mathrm{LM}}(\tilde{\mathbf{x}}) \nabla_{\theta_G} \log p_G(\tilde{\mathbf{x}}|\mathbf{z_x}, \mathbf{v_y})]. \tag{4.2.4}$$

However, using a single sample to approximate the expected gradient leads to high variance in gradient estimates and thus unstable learning.

Continuous approximation: Instead, we propose to use a continuous approximation to the sampling process in training the generator, as demonstrated in Figure 4.5. Instead of feeding a single sampled word as input to the next timestep of the generator, we use a Gumbel-softmax (Jang et al., 2016) distribution as a continuous approximation to sample instead. Let u be a categorical distribution with probabilities $\pi_1, \pi_2, \ldots, \pi_c$. Samples from u can be approximated using:

$$p_i = \frac{\exp((\log \pi_i) + g_i)/\tau}{\sum_{j=1}^c \exp((\log \pi_j + g_j)/\tau)},$$

where the g_i's are independent samples from $\mathrm{Gumbel}(0, 1)$.

Let the tokens of the transferred sentence be $\tilde{\mathbf{x}} = \{\tilde{x}_t\}_{t=1}^T$. Suppose the output of the logit at timestep t is $\mathbf{v}_t^{\mathbf{x}}$, then $\tilde{\mathbf{p}}_t^{\mathbf{x}} = \text{Gumbel-softmax}(\mathbf{v}_t^{\mathbf{x}}, \tau)$, where τ is the temperature. When $\tau \to 0$,

$\tilde{\mathbf{p}}_t^{\mathbf{x}}$ becomes the one hot representation of token $\tilde{x}_t$. Using the continuous approximation, then the output of the decoder becomes a sequence of probability vectors $\tilde{\mathbf{p}}^{\mathbf{x}} = \{\tilde{\mathbf{p}}_t^{\mathbf{x}}\}_{t=1}^T$.

With the continuous approximation of $\tilde{\mathbf{x}}$, we can calculate the loss evaluated using a language model easily, as shown in Figure 4.5. For every step, we feed $\tilde{\mathbf{p}}_t^{\mathbf{x}}$ to the language model of $\mathbf{y}$ (denoted as $\text{LM}_{\mathbf{y}}$) using the weighted average of the embedding $W_e\tilde{\mathbf{p}}_t^{\mathbf{x}}$, then we get the output from the $\text{LM}_{\mathbf{y}}$ which is a probability distribution over the vocabulary of the next word $\hat{\mathbf{p}}_{t+1}^{\mathbf{x}}$. The loss of the current step is the cross entropy loss between $\tilde{\mathbf{p}}_{t+1}^{\mathbf{x}}$ and $\hat{\mathbf{p}}_{t+1}^{\mathbf{x}}$: $(\tilde{\mathbf{p}}_{t+1}^{\mathbf{x}})^\intercal \log \hat{\mathbf{p}}_{t+1}^{\mathbf{x}}$. Note that when the decoder output distribution $\tilde{\mathbf{p}}_{t+1}^{\mathbf{x}}$ aligns with the language model output distribution $\hat{\mathbf{p}}_{t+1}^{\mathbf{x}}$, the above loss achieves minimum. By summing the loss over all steps and taking the gradient, we can use standard back-propagation to train the generator:

$$\nabla_{\theta_G}\mathcal{L}_{\text{LM}}^{\mathbf{y}} \approx \mathbb{E}_{\mathbf{x}\sim\mathbf{X},\tilde{\mathbf{p}}^{\mathbf{x}}\sim p_G(\tilde{\mathbf{x}}|\mathbf{z}_{\mathbf{x}},\mathbf{v}_{\mathbf{y}})}[\nabla_{\theta_G}\sum_{t=1}^{T}(\tilde{\mathbf{p}}_t^{\mathbf{x}})^\intercal \log \hat{\mathbf{p}}_t^{\mathbf{x}}]. \tag{4.2.5}$$

The above Equation is a continuous approximation of Equation 4.2.4 with Gumbel softmax distribution. In experiments, we use a single sample of $\tilde{\mathbf{p}}^{\mathbf{x}}$ to approximate the expectation.

Note that the use of the language model discriminator is a somewhat different in each of the two types of training update steps because of the continuous approximation. We use discrete samples from the generators as negative samples in training the language model discriminator step, while we use a continuous approximation in updating the generator step according to Equation 4.2.5.

Overcoming mode collapse: It is known that in adversarial training, the generator can suffer from mode collapse (Arjovsky and Bottou, 2017; Hu et al., 2017c) where the samples from the generator only cover part of the data distribution. In preliminary experimentation, we found that the language model prefers short sentences. To overcome this length bias, we use two tricks in our experiments: 1) we normalize the loss of Equation 4.2.5 by length and 2) we fix the length of $\tilde{\mathbf{x}}$ to be the same of $\mathbf{x}$. We find these two tricks stabilize the training and avoid generating collapsed overly short outputs.

4.2.4 Experiments

In order to verify the effectiveness of our model, we experiment on three tasks: word substitution decipherment, sentiment modification, and related language translation. We mainly compare with the most comparable approach of (Shen et al., 2017) that uses CNN classifiers as discriminators[3]. Note that (Shen et al., 2017) use three discriminators to align both $\mathbf{z}$ and decoder hidden states, while our model only uses a single language model as a discriminator directly on the output sentences $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$. Moreover, we also compare with a broader set of related work (Hu et al., 2017b; Fu et al., 2017b; Li et al., 2018b) for the tasks when appropriate. Our proposed model provides substantiate improvements in most of the cases. We implement our model with the Texar (**?**) toolbox based on Tensorflow (Abadi et al., 2016).

[3] We use the code from `https://github.com/shentianxiao/language-style-transfer`.

Model	20%	40%	60%	80%	100%
Copy	64.3	39.1	14.4	2.5	0
(Shen et al., 2017)*	86.6	77.1	70.1	61.2	**50.8**
`Our results:`					
LM	89.0	**80.0**	**74.1**	62.9	49.3
LM + adv	**89.1**	79.6	71.8	**63.8**	44.2

Table 4.6: Decipherment results measured in BLEU. Copy is directly measuring $\mathbf{y}$ against $\mathbf{x}$. LM + adv denotes we use negative samples to train the language model.*We run the code open-sourced by the authors to get the results.

Model	Accu	BLEU	$\text{PPL}_{\mathbf{X}}$	$\text{PPL}_{\mathbf{Y}}$
(Shen et al., 2017)	79.5	12.4	50.4	52.7
(Hu et al., 2017b)	87.7	**65.6**	115.6	239.8
`Our results:`				
LM	83.3	38.6	**30.3**	**42.1**
LM + Classifier	**91.2**	57.8	47.0	60.9

Table 4.7: Results for sentiment modification. $\mathbf{X}$ = negative, $\mathbf{Y}$ = positive. $\text{PPL}_{\mathbf{x}}$ denotes the perplexity of sentences transferred from positive sentences evaluated by a language model trained with negative sentences and vice versa.

4.2.4.1 Word substitution decipherment

As the first task, we consider the word substitution decipherment task previous explored in the NLP literature (Dou and Knight, 2012). We can control the amount of change to the original sentences in word substitution decipherment so as to systematically investigate how well the language model performs in a task that requires various amount of changes. In word substitution cipher, every token in the vocabulary is mapped to a cipher token and the tokens in sentences are replaced with cipher tokens according to the cipher dictionary. The task of decipherment is to recover the original text without any knowledge of the dictionary.

Data: Following (Shen et al., 2017), we sample 200K sentences from the Yelp review dataset as plain text $\mathbf{X}$ and sample other 200K sentences and apply word substitution cipher on these sentences to get $\mathbf{Y}$. We use another 100k *parallel* sentences as the development and test set respectively. Sentences of length more than 15 are filtered out. We keep all words that appear more than 5 times in the training set and get a vocabulary size of about 10k. All words appearing less than 5 times are replaced with a "¡unk¿" token. We random sample words from the vocabulary and replace them with cipher tokens. The amount of ciphered words ranges from 20% to 100%. As we have ground truth plain text, we can directly measure the BLEU [4] score to evaluate the model. Our model configurations are included in Appendix 4.2.7.2.

[4] BLEU score is measured with `multi-bleu.perl`.

Results: The results are shown in Table 4.6. We first investigate the effect of using negative samples in training the language model, as denotes by LM + adv in Table 4.6. We can see that using adversarial training sometimes improves the results. However, we found empirically that using negative samples makes the training very unstable and the model diverges easily. This is the main reason why we did not get consistently better results by incorporating adversarial training.

Comparing with (Shen et al., 2017), we can see that the language model without adversarial training is already very effective and performs much better when the amount of change is less than 100%. This is intuitive because when the change is less than 100%, a language model can use context information to predict and correct enciphered tokens. It's surprising that even with 100% token change, our model is only 1.5 BLEU score worse than (Shen et al., 2017), when all tokens are replaced and no context information can be used by the language model. We guess our model can gradually decipher tokens from the beginning of a sentence and then use them as a bootstrap to decipher the whole sentence. We can also combine language models with the CNNs as discriminators. For example, for the 100% case, we get BLEU score of 52.1 when combing them. Given unstableness of adversarial training and effectiveness of language models, we set $\gamma = 0$ in Equation 4.2.1 and 4.2.2 in the rest of the experiments.

4.2.4.2 Sentiment Manipulation

We have demonstrated that the language model can successfully crack word substitution cipher. However, the change of substitution cipher is limited to a one-to-one mapping. As the second task, we would like to investigate whether a language model can distinguish sentences with positive and negative sentiments, thus help to transfer the sentiments of sentences while preserving the content. We compare to the model of (Hu et al., 2017b) as an additional baseline, which uses a pre-trained classifier as guidance.

Data: We use the same data set as in (Shen et al., 2017). The data set contains 250K negative sentences (denoted as $\mathbf{X}$) and 380K positive sentences (denoted as $\mathbf{Y}$), of which 70% are used for training, 10% are used for development and the remaining 20% are used as test set. The pre-processing steps are the same as the previous experiment. We also use similar experiment configurations.

Evaluation: Evaluating the quality of transferred sentences is a challenging problem as there are no ground truth sentences. We follow previous papers in using model-based evaluation. We measure whether transferred sentences have the correct sentiment according to a pre-trained sentiment classifier. We follow both (Hu et al., 2017b) and (Shen et al., 2017) in using a CNN-based classifier. However, simply evaluating the sentiment of sentences is not enough since the model can output collapsed output such as a single word "good" for all negative transfer and "bad" for all positive transfer. We not only would like transferred sentences to preserve the content of original sentences, but also to be smooth in terms of language quality. For these two aspects, we propose to measure the BLEU score of transferred sentences against original sentences and measure the perplexity of transferred sentences to evaluate the fluency. A good model should perform well on all three metrics.

Results: We report the results in Table. 4.7. As a baseline, the original corpus has perplexity of 35.8 and 38.8 for the negative and positive sentences respectively. Comparing LM with (Shen et al., 2017), we can see that LM outperforms it in all three aspects: getting higher accuracy, preserving the content better while being more fluent. This demonstrates the effectiveness of using LM as the discriminator. (Hu et al., 2017b) has the highest accuracy and BLEU score among the three models while the perplexity is very high. It is not surprising that the classifier will only modify the features of the sentences that are related to the sentiment and there is no mechanism to ensure that the modified sentence being fluent. Hence the corresponding perplexity is very high. We can manifest the best of both models by combing the loss of LM and the classifier in (Hu et al., 2017b): a classifier is good at modifying the sentiment and an LM can smooth the modification to get a fluent sentence. We find improvement of accuracy and perplexity as denoted by LM + classifier compared to classifier only (Hu et al., 2017b).

Comparing with other models: Recently there are other models that are proposed specifically targeting the sentiment modification task such as (Li et al., 2018b). Their method is feature based and consists of the following steps: (`Delete`) first, they use the statistics of word frequency to delete the attribute words such as "good, bad" from original sentences, (`Retrieve`) then they retrieve the most similar sentences from the other corpus based on nearest neighbor search, (`Generate`) the attribute words from retrieved sentences are combined with the content words of original sentences to generate transferred sentences. The authors provide 500 human annotated sentences as the ground truth of transferred sentences so we measure the BLEU score against those sentences. The results are shown in Table 4.8. We can see our model has similar accuracy compared with DeleteAndRetrieve, but has much better BLEU scores and slightly better perplexity.

We list some examples of transferred sentences in Table 4.10 in the appendix. We can see that (Shen et al., 2017) does not keep the content of the original sentences well and changes the meaning of the original sentences. (Hu et al., 2017b) changes the sentiment but uses improper words, e.g. "maintenance is equally `hilarious`". Our LM can change the change the sentiment of sentences. But sometimes there is an over-smoothing problem, changing the less frequent words to more frequent words, e.g. changing "my goodness it was so gross" to "my `food` it was so good.". In general LM + classifier has the best results, it changes the sentiment, while keeps the content and the sentences are fluent.

4.2.4.3 Related language translation

In the final experiment, we consider a more challenging task: unsupervised related language translation (Pourdamghani and Knight, 2017). Related language translation is easier than normal pair language translation since there is a close relationship between the two languages. Note here we don't compare with other sophisticated unsupervised neural machine translation systems such as (Lample et al., 2017; Artetxe et al., 2017), whose models are much more complicated and use other techniques such as back-translation, but simply compare the different type of discriminators in the context of a simple model.

Data: We choose Bosnian (bs) vs Serbian (sr) and simplified Chinese (zh-CN) vs traditional

Model	ACCU	BLEU	$\text{PPL}_{\mathbf{X}}$	$\text{PPL}_{\mathbf{Y}}$
(Shen et al., 2017)	76.2	6.8	49.4	45.6
(Fu et al., 2017b):				
StyleEmbedding	9.2	16.65	97.51	142.6
MultiDecoder	50.9	11.24	111.1	119.1
(Li et al., 2018b):				
Delete	87.2	11.5	75.2	68.7
Template	86.7	18.0	192.5	148.4
Retrieval	**95.1**	1.3	**31.5**	**37.0**
DeleteAndRetrieval	90.9	12.6	104.6	43.8
Our results:				
LM	85.4	13.4	32.8	40.5
LM + Classifier	90.0	**22.3**	48.4	61.6

Table 4.8: Results for sentiment modification based on the 500 human annotated sentences as ground truth from (Li et al., 2018b).

Chinese (zh-TW) pair as our experiment languages. Due to the lack of parallel data for these data, we build the data ourselves. For bs and sr pair, we use the monolingual data from Leipzig Corpora Collections[5]. We use the news data and sample about 200k sentences of length less than 20 for each language, of which 80% are used for training, 10% are used for validation and remaining 10% are used for test. For validation and test, we obtain the parallel corpus by using the Google Translation API. The vocabulary size is 25k for the sr vs bs language pair. For zh-CN and zh-TW pair, we use the monolingual data from the Chinese Gigaword corpus. We use the news headlines as our training data. 300k sentences are sampled for each language. The data is partitioned and parallel data is obtained in a similar way to that of sr vs bs pair. We directly use a character-based model and the total vocabulary size is about 5k. For evaluation, we directly measure the BLEU score using the references for both language pairs.

Note that the relationship between zh-CN and zh-TW is simple and mostly like a decipherment problem in which some simplified Chinese characters have the corresponding traditional character mapping. The relation between bs vs sr is more complicated.

Results: The results are shown in Table. 4.9. For sr–bos and bos–sr, since the vocabulary of two languages does not overlap at all, it is a very challenging task. We report the BLEU1 metric since the BLEU4 is close to 0. We can see that our language model discriminator still outperforms (Shen et al., 2017) slightly. The case for zh–tw and tw–zh is much easier. Simple copying already has a reasonable score of 32.3. Using our model, we can improve it to 81.6 for cn–tw and 85.5 for tw–cn, outperforming (Shen et al., 2017) by a large margin.

[5] `http://wortschatz.uni-leipzig.de/en`

Model	sr–bs	bs–sr	cn–tw	tw–cn
Copy	0	0	32.3	32.3
(Shen et al., 2017)	29.1	30.3	60.1	60.7
`Our results:`				
LM	**31.0**	**31.7**	**81.6**	**85.5**

Table 4.9: Related language translation results measured in BLEU. The results for sr vs bs in measured in BLEU1 while cn vs tw is measure in BLEU.

4.2.5 Related Work

Non-parallel transfer in natural language: (Hu et al., 2017b; Shen et al., 2017; Prabhumoye et al., 2018; Gomez et al., 2018) are most relevant to our work. (Hu et al., 2017b) aim to generate sentences with controllable attributes by learning disentangled representations. (Shen et al., 2017) introduce adversarial training to unsupervised text style transfer. They apply discriminators both on the encoder representation and on the hidden states of the decoders to ensure that they have the same distribution. These are the two models that we mainly compare with. (Prabhumoye et al., 2018) use the back-translation technique in their model, which is complementary to our method and can be integrated into our model to further improve performance. (Gomez et al., 2018) use GAN-based approach to decipher shift ciphers. (Lample et al., 2017; Artetxe et al., 2017) propose unsupervised machine translation and use adversarial training to match the encoder representation of the sentences from different languages. They also use back-translation to refine their model in an iterative way.

GANs: GANs have been widely explored recently, especially in computer vision (Zhu et al., 2017; Chen et al., 2016b; Radford et al., 2015b; Sutton et al., 2000; Salimans et al., 2016; Denton et al., 2015; Isola et al., 2017). The progress of GANs on text is relatively limited due to the non-differentiable discrete tokens. Lots of papers (Yu et al., 2017b; Che et al., 2017b; Li et al., 2017; Yang et al., 2017a) use REINFORCE (Sutton et al., 2000) to finetune a trained model to improve the quality of samples. There is also prior work that attempts to introduce more structured discriminators, for instance, the energy-based GAN (EBGAN) (Zhao et al., 2016) and RankGAN (Lin et al., 2017). Our language model can be seen as a special energy function, but it is more complicated than the auto-encoder used in (Zhao et al., 2016) since it has a recurrent structure. (Hu et al., 2018a) also proposes to use structured discriminators in generative models and establishes its the connection with posterior regularization.

Computer vision style transfer: Our work is also related to unsupervised style transfer in computer vision (Gatys et al., 2016; Huang and Belongie, 2017). (Gatys et al., 2016) directly uses the covariance matrix of the CNN features and tries to align the covariance matrix to transfer the style. (Huang and Belongie, 2017) proposes adaptive instance normalization for an arbitrary style of images. (Zhu et al., 2017) uses a cycle-consistency loss to ensure the content of the images is preserved and can be translated back to original images.

Language model for reranking: Previously, language models are used to incorporate the knowledge of monolingual data mainly by reranking the sentences generated from a base model such as (Brants et al., 2007; Gulcehre et al., 2015; He et al., 2016a). (Liu et al., 2017; Chen et al., 2016a) use a language model as training supervision for unsupervised OCR. Our model is more advanced in using language models as discriminators in distilling the knowledge of monolingual data to a base model in an end-to-end way.

4.2.6 Conclusion

We showed that by using language models as discriminators and we could outperform traditional binary classifier discriminators in three unsupervised text style transfer tasks including word substitution decipherment, sentiment modification and related language translation. In comparison with a binary classifier discriminator, a language model can provide a more stable and more informative training signal for training generators. Moreover, we empirically found that it is possible to eliminate adversarial training with negative samples if a structured model is used as the discriminator, thus pointing one possible direction to solve the training difficulty of GANs. In the future, we plan to explore and extend our model to semi-supervised learning.

4.2.7 Appendix

4.2.7.1 Training Algorithms

The training algorithm is summarized in Algorithm 3.

Algorithm 3 Unsupervised text style transfer.

```
Input: Data set of two different styles $\mathbf{X}$, $\mathbf{Y}$.
       Parameters: weight $\lambda$ and $\gamma$, temperature $\tau$.
  Initialized model parameters $\theta_{\mathbf{E}}, \theta_{\mathbf{G}}, \theta_{\mathrm{LM}_{\mathbf{x}}}, \theta_{\mathrm{LM}_{\mathbf{y}}}$.
  repeat
    Update $\theta_{\mathrm{LM}_{\mathbf{x}}}$ and $\theta_{\mathrm{LM}_{\mathbf{y}}}$ by minimizing $\mathcal{L}^{\mathbf{x}}_{\mathrm{LM}}(\theta_{\mathrm{LM}_{\mathbf{x}}})$ and $\mathcal{L}^{\mathbf{y}}_{\mathrm{LM}}(\theta_{\mathrm{LM}_{\mathbf{y}}})$ respectively.
    Update $\theta_{\mathbf{E}}, \theta_{\mathbf{G}}$ by minimizing: $\mathcal{L}_{\text{rec}} - \lambda(\mathcal{L}^{\mathbf{x}}_{\mathrm{LM}} + \mathcal{L}^{\mathbf{y}}_{\mathrm{LM}})$ using Eq.(4.2.5).
  until convergence
Output: A text style transfer model with parameters $\theta_{\mathbf{E}}$, $\theta_{\mathbf{G}}$.
```

4.2.7.2 Model Configurations

Similar model configuration to that of (Shen et al., 2017) is used for a fair comparison. We use one-layer GRU (Chung et al., 2014) as the encoder and decoder (generator). We set the word embedding size to be 100 and GRU hidden size to be 700. $\mathbf{v}$ is a vector of size 200. For the language model, we use the same architecture as the decoder. The parameters of the language model are not shared with parameters of other parts and are trained from scratch. We use a batch size of 128, which contains 64 samples from $\mathbf{X}$ and $\mathbf{Y}$ respectively. We use Adam (Kingma and Ba, 2014) optimization algorithm to train both the language model and the auto-encoder and the learning rate is set to be the same. Hyper-parameters are selected based on the validation set. We use grid search to pick the best parameters. The learning rate is selected from $[1e-3, 5e-4, 2e-$

$4, 1e-4]$ and λ, the weight of language model loss, is selected from $[1.0, 0.5, 0.1]$. Models are trained for a total of 20 epochs. We use an annealing strategy to set the temperature of τ of the Gumbel-softmax approximation. The initial value of τ is set to 1.0 and it decays by half every epoch until reaching the minimum value of 0.001.

4.2.7.3 Sentiment Transfer Examples

Table 4.10 shows more generated samples by various models for the text style transfer.

Model	Negative to Positive
Original	it was super dry and had a weird taste to the entire slice .
(Shen et al., 2017)	it was super friendly and had a nice touch to the same .
(Hu et al., 2017b)	it was super well-made and had a weird taste to the entire slice .
LM	it was very good , had a good taste to the food service .
LM + classifier	it was super fresh and had a delicious taste to the entire slice .
Original	my goodness it was so gross .
(Shen et al., 2017)	my server it was so .
(Hu et al., 2017b)	my goodness it was so refreshing .
LM	my food it was so good .
LM + classifier	my goodness it was so great .
Original	maintenance is equally incompetent .
(Shen et al., 2017)	everything is terrific professional .
(Hu et al., 2017b)	maintenance is equally hilarious .
LM	maintenance is very great .
LM + classifier	maintenance is equally great .
Original	if i could give them a zero star review i would !
(Shen et al., 2017)	if i will give them a breakfast star here ever !
(Hu et al., 2017b)	if i lite give them a sweetheart star review i would !
LM	if i could give them a _num_ star place i would .
LM + classifier	if i can give them a great star review i would !

Model	Positive to Negative
Original	did n't know this type cuisine could be this great !
(Shen et al., 2017)	did n't know this old food you make this same horrible !
(Hu et al., 2017b)	did n't know this type cuisine could be this great !
LM	did n't know this type , could be this bad .
LM + classifier	did n't know this type cuisine could be this horrible .
Original	besides that , the wine selection they have is pretty awesome as well .
(Shen et al., 2017)	after that , the quality prices that does n't pretty much well as .
(Hu et al., 2017b)	besides that , the wine selection they have is pretty borderline as atrocious .
LM	besides that , the food selection they have is pretty awful as well .
LM + classifier	besides that , the wine selection they have is pretty horrible as well .
Original	uncle george is very friendly to each guest .
(Shen et al., 2017)	if there is very rude to our cab .
(Hu et al., 2017b)	uncle george is very lackluster to each guest .
LM	uncle george is very rude to each guest .
LM + classifier	uncle george is very rude to each guest .

Table 4.10: Sentiment transfer examples.

Part III

Applications (2): Repurposing Algorithms for New Problems

Chapter 5

Learning Data Manipulation for Augmentation and Reweighting

The third part of the thesis discusses how the standardized ML formalism allows us to reduce many seemingly different problems that deal with distinct types of experiences to the same general problem, through which an originally specialized solution to a particular problem becomes reusable and repeatable to also solve other similar problems.

In this chapter, we use this guiding principle and study the long challenging problem of learning data manipulation for low-data and label-imbalance tasks. Manipulating data, such as weighting data examples or augmenting with new instances, has been increasingly used to improve model training. Previous research has studied various rule- or learning-based approaches designed for specific types of data manipulation. In this work, we develop a new method that supports learning different manipulation schemes with the same single algorithm. Remarkably, the method builds upon the fundamental relationships between supervised learning (SL) and reinforcement learning (RL) as revealed by the standard equation in Chapter 2. That is, data instances and their manipulation in supervised learning are equivalent to reward in RL in the sense that both are formulated as the experience function in the standard equation. The symmetry indicates that any existing reward learning algorithms in the fertile RL literature can potentially be repurposed to enable learning of data manipulation in SL.

Specifically, in this work, we import a recent intrinsic reward learning algorithm to form our manipulation framework for joint data manipulation learning and model training. Different parameterization of the "data reward" function instantiates different manipulation schemes. We showcase data augmentation that learns a text transformation network, and data weighting that dynamically adapts the data sample importance. Experiments show the resulting algorithms significantly improve the image and text classification performance in low data regime and class-imbalance tasks.

In a more general sense, the problem of joint model-experience training is indeed a very common challenge in practice, especially when the experiences used in learning are under-specified or not of good enough quality (e.g., low-quality data as considered in this chapter). The next chapter will consider another typical situation where we have complex knowledge constraints whose constituent modules are sub-optimal and require automated adaptation. The same guiding principle of algorithmic repurposing still applies and yields effective solutions.

5.1 Introduction

The performance of machines often crucially depend on the amount and quality of the data used for training. It has become increasingly ubiquitous to *manipulate* data to improve learning, especially in low data regime or in presence of low-quality datasets (e.g., imbalanced labels). For example, *data augmentation* applies label-preserving transformations on original data points to expand the data size; *data weighting* assigns an importance weight to each instance to adapt its effect on learning; and *data synthesis* generates entire artificial examples. Different types of manipulation can be suitable for different application settings.

Common data manipulation methods are usually designed manually, e.g., augmenting by flipping an image or replacing a word with synonyms, and weighting with inverse class frequency or loss values (Freund and Schapire, 1997; Malisiewicz et al., 2011). Recent work has studied automated approaches, such as learning the composition of augmentation operators with reinforcement learning (Ratner et al., 2017; Cubuk et al., 2019), deriving sample weights adaptively from a validation set via meta learning (Ren et al., 2018), or learning a weighting network by inducing a curriculum (Jiang et al., 2018). These learning-based approaches have alleviated the engineering burden and produced impressive results. However, the algorithms are usually designed specifically for certain types of manipulation (e.g., either augmentation or weighting) and thus have limited application scope in practice.

In this work, we propose a new approach that enables learning for different manipulation schemes with the same single algorithm. Our approach draws inspiration from the recent work (Tan et al., 2018) that shows equivalence between the data in supervised learning and the reward function in reinforcement learning. We thus adapt an off-the-shelf *reward learning* algorithm (Zheng et al., 2018) to the supervised setting for automated data manipulation. The marriage of the two paradigms results in a simple yet general algorithm, where various manipulation schemes are reduced to different parameterization of the *data reward*. Free parameters of manipulation are learned jointly with the target model through efficient gradient descent on validation examples. We demonstrate instantiations of the approach for automatically fine-tuning an augmentation network and learning data weights, respectively.

We conduct extensive experiments on text and image classification in challenging situations of very limited data and imbalanced labels. Both augmentation and weighting by our approach significantly improve over strong base models, even though the models are initialized with large-scale pretrained networks such as BERT (Devlin et al., 2019) for text and ResNet (He et al., 2016b) for images. Our approach, besides its generality, also outperforms a variety of dedicated rule- and learning-based methods for either augmentation or weighting, respectively. Lastly, we observe that the two types of manipulation tend to excel in different contexts: augmentation shows superiority over weighting with a small amount of data available, while weighting is better at addressing class imbalance problems.

5.2 Related Work

Rich types of data manipulation have been increasingly used in modern machine learning pipelines. Previous work each has typically focused on a particular manipulation type. Data augmentation that perturbs examples without changing the labels is widely used especially in vision (Simard et al., 1998; Krizhevsky et al., 2012) and speech (Ko et al., 2015; Park et al.) domains. Common heuristic-based methods on images include cropping, mirroring, rotation (Krizhevsky et al., 2012), and so forth. Recent work has developed automated augmentation approaches (Cubuk et al., 2019; Ratner et al., 2017; Lemley et al., 2017; Peng et al., 2018; Tran et al., 2017). Xie et al. (2019) additionally use large-scale unlabeled data. Cubuk et al. (2019); Ratner et al. (2017) learn to induce the composition of data transformation operators. Instead of treating data augmentation as a policy in reinforcement learning (Cubuk et al., 2019), we formulate manipulation as a reward function and use efficient stochastic gradient descent to learn the manipulation parameters. Text data augmentation has also achieved impressive success, such as contextual augmentation (Kobayashi, 2018; Wu et al., 2018), back-translation (Sennrich et al., 2016), and manual approaches (Wei and Zou, 2019; Andreas et al., 2016). In addition to perturbing the input text as in classification tasks, text generation problems expose opportunities to adding noise also in the output text, such as (Norouzi et al., 2016; Xie et al., 2017). Recent work (Tan et al., 2018) shows output nosing in sequence generation can be treated as an intermediate approach in between supervised learning and reinforcement learning, and developed a new sequence learning algorithm that interpolates between the spectrum of existing algorithms. We instantiate our approach for text contextual augmentation as in (Kobayashi, 2018; Wu et al., 2018), but enhance the previous work by additionally fine-tuning the augmentation network jointly with the target model.

Data weighting has been used in various algorithms, such as AdaBoost (Freund and Schapire, 1997), self-paced learning (Kumar et al., 2010), hard-example mining (Shrivastava et al., 2016), and others (Chang et al., 2017; Katharopoulos and Fleuret, 2018). These algorithms largely define sample weights based on training loss. Recent work (Jiang et al., 2018; Fan et al., 2018) learns a separate network to predict sample weights. Of particular relevance to our work is (Ren et al., 2018) which induces sample weights using a validation set. The data weighting mechanism instantiated by our framework has a key difference in that samples weights are treated as parameters that are updated iteratively, instead of re-estimated from scratch at each step. We show improved performance of our approach. Besides, our data manipulation approach is derived based on a different perspective of reward learning, instead of meta-learning as in (Ren et al., 2018).

Another popular type of data manipulation involves data synthesis, which creates entire artificial samples from scratch. GAN-based approaches have achieved impressive results for synthesizing conditional image data (Baluja and Fischer, 2017; Mirza and Osindero, 2014). In the text domain, controllable text generation (Hu et al., 2017a) presents a way of co-training the data generator and classifier in a cyclic manner within a joint VAE (Kingma and Welling, 2013) and wake-sleep (Hinton et al., 1995) framework. It is interesting to explore the instantiation of the present approach for adaptive data synthesis in the future.

5.3 Background

Chapter 2 has presented the connections between relevant learning paradigms in a principled way. Here we replicate some of the discussions. This section also establishes the notations used throughout the work, which, for ease of presentation, may slightly be different than those in Chapter 2.

Let $\boldsymbol{x}$ denote the input and y the output. For example, in text classification, $\boldsymbol{x}$ can be a sentence and y is the sentence label. Denote the model of interest as $p_\theta(y|\boldsymbol{x})$, where $\boldsymbol{\theta}$ is the model parameters to be learned. In supervised setting, given a set of training examples $\mathcal{D} = \{(\boldsymbol{x}^*, y^*)\}$, we learn the model by maximizing the data log-likelihood.

Equivalence between Data and Reward Chapter 2 has introduced a unifying perspective of reformulating both maximum likelihood supervised learning and policy optimization as special instances of the standardized framework. In this perspective, data examples providing supervision signals can equivalently be seen as a specialized reward function.

To connect the maximum likelihood supervised learning with policy optimization, consider the model $p_\theta(y|\boldsymbol{x})$ as a policy that takes "action" y given the "state" $\boldsymbol{x}$. Let $R(\boldsymbol{x}, y|\mathcal{D}) \in \mathbb{R}$ denote a reward function (corresponding to the experience function f in the standard equation), and $p(\boldsymbol{x})$ be the empirical data distribution which is known given $\mathcal{D}$. Further assume a variational distribution $q(\boldsymbol{x}, y)$ that factorizes as $q(\boldsymbol{x}, y) = p(\boldsymbol{x})q(y|\boldsymbol{x})$. A variational policy optimization objective is then written as:

$$\mathcal{L}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{x},y)}\left[R(\boldsymbol{x}, y|\mathcal{D})\right] - \beta\mathbb{E}_{q(\boldsymbol{x},y)}\big[\log p(\boldsymbol{x})p_\theta(y|\boldsymbol{x})\big] + \alpha \mathrm{H}(q), \tag{5.3.1}$$

where $\mathrm{H}(\cdot)$ is the Shannon entropy; and $\alpha, \beta > 0$ are balancing weights. The objective is in the same form with the RL-as-inference formalism of policy optimization (e.g., Dayan and Hinton, 1997; Levine, 2018; Abdolmaleki et al., 2018). Intuitively, the objective maximizes the expected reward under q, and enforces the model p_θ to stay close to q, with a maximum entropy regularization over q. The problem is solved with an EM procedure that optimizes q and $\boldsymbol{\theta}$ alternatingly:

$$\begin{aligned} \text{E-step:} \quad & q'(\boldsymbol{x}, y) = \exp\left\{\frac{\beta \log p(\boldsymbol{x})p_\theta(y|\boldsymbol{x}) + R(\boldsymbol{x}, y|\mathcal{D})}{\alpha}\right\} \Big/ Z, \\ \text{M-step:} \quad & \boldsymbol{\theta}' = \operatorname{argmax}_\theta \mathbb{E}_{q'(\boldsymbol{x},y)}\big[\log p_\theta(y|\boldsymbol{x})\big], \end{aligned} \tag{5.3.2}$$

where Z is the normalization term. With the established framework, it is easy to show that the above optimization procedure reduces to maximum likelihood learning by taking $\alpha = \beta = 1$, and the reward function:

$$R_\delta(\boldsymbol{x}, y|\mathcal{D}) = \begin{cases} 1 & \text{if } (\boldsymbol{x}, y) \in \mathcal{D} \\ -\infty & \text{otherwise.} \end{cases} \tag{5.3.3}$$

That is, a sample $(\boldsymbol{x}, y)$ receives a unit reward only when it matches a training example in the dataset, while the reward is negative infinite in all other cases. To make the equivalence to maximum likelihood learning clearer, note that the above M-step now reduces to

$$\boldsymbol{\theta}' = \mathrm{argmax}_{\theta}\, \mathbb{E}_{p(\boldsymbol{x}) \exp\{R_\delta\}/Z} \big[\log p_\theta(y|\boldsymbol{x}) \big], \tag{5.3.4}$$

where the joint distribution $p(\boldsymbol{x}) \exp\{R_\delta\}/Z$ equals the empirical data distribution, which means the M-step is in fact maximizing the data log-likelihood of the model p_θ.

Gradient-based Reward Learning There is a rich line of research on learning the reward in reinforcement learning. Of particular interest to this work is (Zheng et al., 2018) which learns a parametric *intrinsic* reward that additively transforms the original task reward (a.k.a *extrinsic* reward) to improve the policy optimization. For consistency of notations with above, formally, let $p_\theta(y|\boldsymbol{x})$ be a policy where y is an action and $\boldsymbol{x}$ is a state. Let R^{in}_{ϕ} be the intrinsic reward with parameters $\boldsymbol{\phi}$. In each iteration, the policy parameter $\boldsymbol{\theta}$ is updated to maximize the joint rewards, through:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \gamma \nabla_\theta \mathcal{L}^{ex+in}(\boldsymbol{\theta}, \boldsymbol{\phi}), \tag{5.3.5}$$

where $\mathcal{L}^{ex+in}$ is the expectation of the sum of extrinsic and intrinsic rewards; and γ is the step size. The equation shows $\boldsymbol{\theta}'$ depends on $\boldsymbol{\phi}$, thus we can write as $\boldsymbol{\theta}' = \boldsymbol{\theta}'(\boldsymbol{\phi})$.

The next step is to optimize the intrinsic reward parameters $\boldsymbol{\phi}$. Recall that the *ultimate measure* of the performance of a policy is the value of extrinsic reward it achieves. Therefore, a *good* intrinsic reward is supposed to, when the policy is trained with it, increase the eventual extrinsic reward. The update to $\boldsymbol{\phi}$ is then written as:

$$\boldsymbol{\phi}' = \boldsymbol{\phi} + \gamma \nabla_\phi \mathcal{L}^{ex}(\boldsymbol{\theta}'(\boldsymbol{\phi})). \tag{5.3.6}$$

That is, we want the expected extrinsic reward $\mathcal{L}^{ex}(\boldsymbol{\theta}')$ of the new policy $\boldsymbol{\theta}'$ to be maximized. Since $\boldsymbol{\theta}'$ is a function of $\boldsymbol{\phi}$, we can directly backpropagate the gradient through $\boldsymbol{\theta}'$ to $\boldsymbol{\phi}$.

5.4 Learning Data Manipulation

5.4.1 Method

Parameterizing Data Manipulation We now develop our approach of learning data manipulation, through a novel marriage of supervised learning and the above reward learning. Specifically, from the policy optimization perspective, due to the δ-function reward (Eq.5.3.3), the standard maximum likelihood learning is restricted to use only the exact training examples $\mathcal{D}$ in a uniform way. A natural idea of enabling data manipulation is to relax the strong restrictions of the δ-function reward and instead use a relaxed reward $R_\phi(\boldsymbol{x}, y|\mathcal{D})$ with parameters $\boldsymbol{\phi}$. The relaxed reward can be parameterized in various ways, resulting in different types of manipulation. For example, when a sample $(\boldsymbol{x}, y)$ matches a data instance, instead of returning constant 1 by R_δ, the new R_ϕ can return varying reward values depending on the matched instance, resulting in

Algorithm 4 Joint Learning of Model and Data Manipulation

```
Input: The target model $p_\theta(y|\boldsymbol{x})$
       The data manipulation function $R_\phi(\boldsymbol{x}, y|\mathcal{D})$
       Training set $\mathcal{D}$, validation set $\mathcal{D}^v$
Initialize model parameter $\boldsymbol{\theta}$ and manipulation parameter $\boldsymbol{\phi}$
repeat
    Optimize $\boldsymbol{\theta}$ on $\mathcal{D}$ enriched with data manipulation
    through Eq.(5.4.1)
    Optimize $\boldsymbol{\phi}$ by maximizing data log-likelihood on $\mathcal{D}^v$
    through Eq.(5.4.2)
until convergence
Output: Learned model $p_{\theta^*}(y|\boldsymbol{x})$ and manipulation
    $R_{\phi^*}(y, \boldsymbol{x}|\mathcal{D})$
```

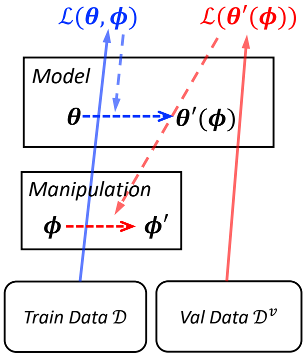

$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi})$
$\mathcal{L}(\boldsymbol{\theta}'(\boldsymbol{\phi}))$
Model
$\boldsymbol{\theta}$
$\boldsymbol{\theta}'(\boldsymbol{\phi})$
Manipulation
$\boldsymbol{\phi}$
$\boldsymbol{\phi}'$
Train Data $\mathcal{D}$
Val Data $\mathcal{D}^v$

Figure 5.1: Algorithm Computation. Blue arrows denote learning model $\boldsymbol{\theta}$. Red arrows denote learning manipulation $\boldsymbol{\phi}$. Solid arrows denote forward pass. Dashed arrows denote backward pass and parameter updates.

a data weighting scheme. Alternatively, R_ϕ can return a valid reward even when $\boldsymbol{x}$ matches a data example only in part, or $(\boldsymbol{x}, y)$ is an entire new sample not in $\mathcal{D}$, which in effect makes data augmentation and data synthesis, respectively, in which cases $\boldsymbol{\phi}$ is either a data transformer or a generator. In the next section, we demonstrate two particular parameterizations for data augmentation and weighting, respectively.

We thus have shown that the diverse types of manipulation all boil down to a parameterized data reward R_ϕ. Such an concise, uniform formulation of data manipulation has the advantage that, once we devise a method of learning the manipulation parameters $\boldsymbol{\phi}$, the resulting algorithm can directly be applied to automate any manipulation type. We present a learning algorithm next.

Learning Manipulation Parameters To learn the parameters $\boldsymbol{\phi}$ in the manipulation reward $R_\phi(\boldsymbol{x}, y|\mathcal{D})$, we could in principle adopt any off-the-shelf reward learning algorithm in the literature. In this work, we draw inspiration from the above gradient-based reward learning (section 5.3) due to its simplicity and efficiency. Briefly, the objective of $\boldsymbol{\phi}$ is to maximize the *ultimate measure* of the performance of model $p_\theta(\boldsymbol{y}|\boldsymbol{x})$, which, in the context of supervised learning, is the model performance on a held-out validation set.

The algorithm optimizes $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ alternatingly, corresponding to Eq.(5.3.5) and Eq.(5.3.6), respectively. More concretely, in each iteration, we first update the model parameters $\boldsymbol{\theta}$ in analogue to Eq.(5.3.5) which optimizes intrinsic reward-enriched objective. Here, we optimize the log-likelihood of the training set enriched with data manipulation. That is, we replace R_δ with R_ϕ in Eq.(5.3.4), and obtain the augmented M-step:

$$\boldsymbol{\theta}' = \operatorname{argmax}_\theta \mathbb{E}_{p(\boldsymbol{x}) \exp\{R_\phi(\boldsymbol{x}, y|\mathcal{D})\}/Z} \left[\log p_\theta(y|\boldsymbol{x}) \right]. \tag{5.4.1}$$

By noticing that the new $\boldsymbol{\theta}'$ depends on $\boldsymbol{\phi}$, we can write $\boldsymbol{\theta}'$ as a function of $\boldsymbol{\phi}$, namely, $\boldsymbol{\theta}' = \boldsymbol{\theta}'(\boldsymbol{\phi})$.

The practical implementation of the above update depends on the actual parameterization of manipulation R_ϕ, which we discuss in more details in the next section.

The next step is to optimize $\boldsymbol{\phi}$ in terms of the model validation performance, in analogue to Eq.(5.3.6). Formally, let $\mathcal{D}^v$ be the validation set of data examples. The update is then:

$$\begin{aligned} \boldsymbol{\phi}' &= \operatorname{argmax}_{\phi} \mathbb{E}_{p(\boldsymbol{x}) \exp\{R_\delta(\boldsymbol{x},y|\mathcal{D}^v)\}/Z} \big[\log p_{\theta'}(y|\boldsymbol{x}) \big] \\ &= \operatorname{argmax}_{\phi} \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}^v} \big[\log p_{\theta'}(y|\boldsymbol{x}) \big], \end{aligned} \tag{5.4.2}$$

where, since $\boldsymbol{\theta}'$ is a function of $\boldsymbol{\phi}$, the gradient is backpropagated to $\boldsymbol{\phi}$ through $\boldsymbol{\theta}'(\boldsymbol{\phi})$. Taking data weighting for example where $\boldsymbol{\phi}$ is the training sample weights (more details in section 5.4.2), the update is to optimize the weights of training samples so that the model performs best on the validation set.

The resulting algorithm is summarized in Algorithm 6. Figure 5.1 illustrates the computation flow. Learning the manipulation parameters effectively uses a held-out validation set. We show in our experiments that a very small set of validation examples (e.g., 2 labels per class) is enough to significantly improve the model performance in low data regime.

It is worth noting that some previous work has also leveraged validation examples, such as learning data augmentation with policy gradient (Cubuk et al., 2019) or inducing data weights with meta-learning (Ren et al., 2018). Our approach is inspired from a distinct paradigm of (intrinsic) reward learning. In contrast to (Cubuk et al., 2019) that treats data augmentation as a policy, we instead formulate manipulation as a reward function and enable efficient stochastic gradient updates. Our approach is also more broadly applicable to diverse data manipulation types than (Ren et al., 2018; Cubuk et al., 2019).

5.4.2 Instantiations: Augmentation & Weighting

As a case study, we show two parameterizations of R_ϕ which instantiate distinct data manipulation schemes. The first example learns augmentation for text data, a domain that has been less studied in the literature compared to vision and speech (Kobayashi, 2018; Giridhara et al., 2019). The second instance focuses on automated data weighting, which is applicable to any data domains.

Fine-tuning Text Augmentation

The recent work (Kobayashi, 2018; Wu et al., 2018) developed a novel contextual augmentation approach for text data, in which a powerful pretrained language model (LM), such as BERT (Devlin et al., 2019), is used to generate substitutions of words in a sentence. Specifically, given an observed sentence $\boldsymbol{x}^*$, the method first randomly masks out a few words. The masked sentence is then fed to BERT which fills the masked positions with new words. To preserve the original sentence class, the BERT LM is retrofitted as a label-conditional model, and trained on the task training examples. The resulting model is then fixed and used to augment data during the training of target model. We denote the augmentation distribution as $g_{\phi_0}(\boldsymbol{x}|\boldsymbol{x}^*, \boldsymbol{y}^*)$, where $\boldsymbol{\phi}_0$ is the fixed BERT LM parameters.

The above process has two drawbacks. First, the LM is fixed after fitting to the task data. In the subsequent phase of training the target model, the LM augments data without knowing the state of the target model, which can lead to sub-optimal results. Second, in the cases where the task dataset is small, the LM can be insufficiently trained for preserving the labels faithfully, resulting in noisy augmented samples.

To address the difficulties, it is beneficial to apply the proposed learning data manipulation algorithm to additionally fine-tune the LM *jointly* with target model training. As discussed in section 5.4, this reduces to properly parameterizing the data reward function:

$$R_\phi^{aug}(\boldsymbol{x}, y|\mathcal{D}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \sim g_\phi(\boldsymbol{x}|\boldsymbol{x}^*, y), \;\; (\boldsymbol{x}^*, y) \in \mathcal{D} \\ -\infty & \text{otherwise.} \end{cases} \tag{5.4.3}$$

That is, a sample $(\boldsymbol{x}, y)$ receives a unit reward when y is the true label and $\boldsymbol{x}$ is the augmented sample by the LM (instead of the exact original data $\boldsymbol{x}^*$). Plugging the reward into Eq.(5.4.1), we obtain the data-augmented update for the model parameters:

$$\boldsymbol{\theta}' = \text{argmax}_\theta \, \mathbb{E}_{\boldsymbol{x} \sim g_\phi(\boldsymbol{x}|\boldsymbol{x}^*, y), \; (\boldsymbol{x}^*, y) \sim \mathcal{D}} \big[\log p_\theta(y|\boldsymbol{x}) \big]. \tag{5.4.4}$$

That is, we pick an example from the training set, and use the LM to create augmented samples, which are then used to update the target model. Regarding the update of augmentation parameters $\boldsymbol{\phi}$ (Eq.5.4.2), since text samples are discrete, to enable efficient gradient propagation through $\boldsymbol{\theta}'$ to $\boldsymbol{\phi}$, we use a gumbel-softmax approximation (Jang et al., 2016) to $\boldsymbol{x}$ when sampling substitution words from the LM.

Learning Data Weights

We now demonstrate the instantiation of data weighting. We aim to assign an importance weight to each training example to adapt its effect on model training. We automate the process by learning the data weights. This is achieved by parameterizing R_ϕ as:

$$R_\phi^{w}(\boldsymbol{x}, y|\mathcal{D}) = \begin{cases} \phi_i & \text{if } (\boldsymbol{x}, y) = (\boldsymbol{x}_i^*, y_i^*), \;\; (\boldsymbol{x}_i^*, y_i^*) \in \mathcal{D} \\ -\infty & \text{otherwise,} \end{cases} \tag{5.4.5}$$

where $\phi_i \in \mathbb{R}$ is the weight associated with the ith example. Plugging R_ϕ^w into Eq.(5.4.1), we obtain the weighted update for the model $\boldsymbol{\theta}$:

$$\begin{aligned} \boldsymbol{\theta}' &= \text{argmax}_\theta \, \mathbb{E}_{(\boldsymbol{x}_i^*, y_i^*) \in \mathcal{D}, \; i \sim \text{softmax}(\phi_i)} \big[\log p_\theta(y_i^*|\boldsymbol{x}_i^*) \big] \\ &= \text{argmax}_\theta \, \mathbb{E}_{(\boldsymbol{x}_i^*, y_i^*) \sim \mathcal{D}} \big[\text{softmax}(\phi_i) \log p_\theta(y_i^*|\boldsymbol{x}_i^*) \big] \end{aligned} \tag{5.4.6}$$

In practice, when minibatch stochastic optimization is used, we approximate the weighted sampling by taking the softmax over the weights of only the minibatch examples. The data weights $\boldsymbol{\phi}$ are updated with Eq.(5.4.2). It is worth noting that the previous work (Ren et al., 2018) similarly derives data weights based on their gradient directions on a validation set. Our algorithm differs in that the data weights are parameters maintained and updated throughout the training, instead of re-estimated from scratch in each iteration. Experiments show the parametric treatment achieves superior performance in various settings. There are alternative parameterizations of R_ϕ other than Eq.(5.4.5). For example, replacing ϕ_i in Eq.(5.4.5) with $\log \phi_i$ in effect changes the softmax normalization in Eq.(5.4.6) to linear normalization, which is used in (Ren et al., 2018).

5.5 Experiments

We empirically validate the proposed data manipulation approach through extensive experiments on learning augmentation and weighting. We study both text and image classification, in two difficult settings of low data regime and imbalanced labels[1].

5.5.1 Experimental Setup

Base Models. We choose strong pretrained networks as our base models for both text and image classification. Specifically, on text data, we use the BERT (base, uncased) model (Devlin et al., 2019); while on image data, we use ResNet-34 (He et al., 2016b) pretrained on ImageNet. We show that, even with the large-scale pretraining, data manipulation can still be very helpful to boost the model performance on downstream tasks. Since our approach uses validation sets for manipulation parameter learning, for a fair comparison with the base model, we train the base model in two ways. The first is to train the model on the training sets as usual and select the best step using the validation sets; the second is to train on the merged training and validation sets for a fixed number of steps. The step number is set to the average number of steps selected in the first method. We report the results of both methods.

Comparison Methods. We compare our approach with a variety of previous methods that were designed for specific manipulation schemes: (1) For text data augmentation, we compare with the latest model-based augmentation (Wu et al., 2018) which uses a **fixed conditional BERT** language model for word substitution (section 5.4.2). As with base models, we also tried fitting the augmentatin model to both the training data and the joint training-validation data, and did not observe significant difference. Following (Wu et al., 2018), we also study a conventional approach that replaces words with their **synonyms** using WordNet (Miller, 1995). (2) For data weighting, we compare with the state-of-the-art approach (Ren et al., 2018) that dynamically re-estimates sample weights in each iteration based on the validation set gradient directions. We follow (Ren et al., 2018) and also evaluate the commonly-used **proportion** method that weights data by inverse class frequency.

Training. For both the BERT classifier and the augmentation model (which is also based on BERT), we use Adam optimization with an initial learning rate of 4e-5. For ResNets, we use SGD optimization with a learning rate of 1e-3. For text data augmentation, we augment each minibatch by generating two or three samples for each data points (each with 1, 2 or 3 substitutions), and use both the samples and the original data to train the model. For data weighting, to avoid exploding value, we update the weight of each data point in a minibatch by decaying the previous weight value with a factor of 0.1 and then adding the gradient. All experiments were implemented with PyTorch (pytorch.org) and were performed on a Linux machine with 4 GTX 1080Ti GPUs and 64GB RAM. All reported results are averaged over 15 runs ± one standard deviation.

[1] Code available at https://github.com/tanyuqian/learning-data-manipulation

	Model	**SST-5** (40+2)	**IMDB** (40+5)	**TREC** (40+5)
	Base model: BERT (Devlin et al., 2019)	33.32 ± 4.04	63.55 ± 5.35	88.25 ± 2.81
	Base model + val-data	35.86 ± 3.03	63.65 ± 3.32	88.42 ± 4.90
Augment	Synonym	32.45 ± 4.59	62.68 ± 3.94	88.26 ± 2.76
	Fixed augmentation (Wu et al., 2018)	34.84 ± 2.76	63.65 ± 3.21	88.28 ± 4.50
	Ours: Fine-tuned augmentation	$\mathbf{37.03 \pm 2.05}$	$\mathbf{65.62 \pm 3.32}$	$\mathbf{89.15 \pm 2.41}$
Weight	Ren et al. (2018)	36.09 ± 2.26	63.01 ± 3.33	88.60 ± 2.85
	Ours	$\mathbf{36.51 \pm 2.54}$	$\mathbf{64.78 \pm 2.72}$	$\mathbf{89.01 \pm 2.39}$

Table 5.1: Accuracy of Data Manipulation on Text Classification. All results are averaged over 15 runs ± one standard deviation. The numbers in parentheses next to the dataset names indicate the size of the datasets. For example, (40+2) denotes 40 training instances and 2 validation instances *per class*.

5.5.2 Low Data Regime

We study the problem where only very few labeled examples for each class are available. Both of our augmentation and weighting boost base model performance, and are superior to respective comparison methods. We also observe that augmentation performs better than weighting in the low-data setting.

Setup For text classification, we use the popular benchmark datasets, including SST-5 for 5-class sentence sentiment (Socher et al., 2013), IMDB for binary movie review sentiment (Maas et al., 2011), and TREC for 6-class question types (Li and Roth, 2002). We subsample a small training set on each task by randomly picking 40 instances for each class. We further create small validation sets, i.e., 2 instances per class for SST-5, and 5 instances per class for IMDB and TREC, respectively. The reason we use slightly more validation examples on IMDB and TREC is that the model can easily achieve 100% validation accuracy if the validation sets are too small. Thus, the SST-5 task has 210 labeled examples in total, while IMDB has 90 labels and TREC has 270. Such extremely small datasets pose significant challenges for learning deep neural networks. Since the manipulation parameters are trained using the small validation sets, to avoid possible overfitting we restrict the training to small number (e.g., 5 or 10) of epochs. For image classification, we similarly create a small subset of the CIFAR10 data, which includes 40 instances per class for training, and 2 instances per class for validation.

Results Table 5.1 shows the manipulation results on text classification. For data augmentation, our approach significantly improves over the base model on all the three datasets. Besides, compared to both the conventional synonym substitution and the approach that keeps the augmentation network fixed, our adaptive method that fine-tunes the augmentation network jointly with model training achieves superior results. Indeed, the heuristic-based synonym approach can sometimes harm the model performance (e.g., SST-5 and IMDB), as also observed in previous work (Wu et al., 2018; Kobayashi, 2018). This can be because the heuristic rules do not fit the

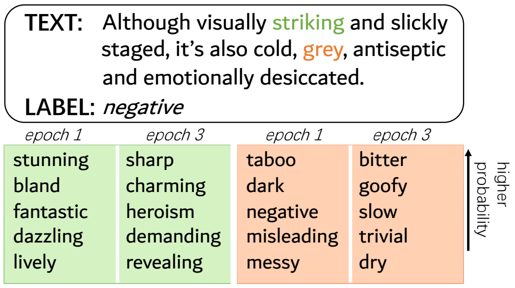


Figure 5.2: Words predicted with the highest probabilities by the augmentation LM. Two tokens "striking" and "grey" are masked for substitution. The boxes in respective colors list the predicted words after training epoch 1 and 3, respectively. E.g., "stunning" is the most probable substitution for "striking" in epoch 1.

Model	Pretrained	Not-Pretrained
Base model: ResNet-34	37.69 ± 3.03	22.98 ± 2.81
Base model + val-data	38.09 ± 1.87	23.42 ± 1.47
Ren et al. (2018)	38.02 ± 2.14	23.44 ± 1.63
Ours	$\mathbf{38.95 \pm 2.03}$	$\mathbf{24.92 \pm 1.57}$

Table 5.2: Accuracy of Data Weighting on Image Classification. The small subset of CIFAR10 used here has 40 training instances and 2 validation instances for each class. The "pretrained" column is the results by initializing the ResNet-34 (He et al., 2016b) base model with ImageNet-pretrained weights. In contrast, "Not-Pretrained" denotes the base model is randomly initialized. Since every class has the same number of examples, the proportion-based weighting degenerates to base model training and thus is omitted here.

task or datasets well. In contrast, learning-based augmentation has the advantage of adaptively generating useful samples to improve model training.

Table 5.1 also shows the data weighting results. Our weight learning consistently improves over the base model and the latest weighting method (Ren et al., 2018). In particular, instead of re-estimating sample weights from scratch in each iteration (Ren et al., 2018), our approach treats the weights as manipulation parameters maintained throughout the training. We speculate that the parametric treatment can adapt weights more smoothly and provide historical information, which is beneficial in the small-data context.

It is interesting to see from Table 5.1 that our augmentation method consistently outperforms the weighting method, showing that data augmentation can be a more suitable technique than data weighting for manipulating small-size data. Our approach provides the generality to instantiate diverse manipulation types and learn with the same single procedure.

To investigate the augmentation model and how the fine-tuning affects the augmentation results, we show in Figure 5.2 the top-5 most probable word substitutions predicted by the augmentation model for two masked tokens, respectively. Comparing the results of epoch 1 and epoch 3, we can see the augmentation model evolves and dynamically adjusts the augmentation behavior as the training proceeds. Through fine-tuning, the model seems to make substitutions that are more coherent with the conditioning label and relevant to the original words (e.g., replacing the word "striking" with "bland" in epoch 1 v.s. "charming" in epoch 3).

Table 5.2 shows the data weighting results on image classification. We evaluate two settings with the ResNet-34 base model being initialized randomly or with pretrained weights, respectively. Our data weighting consistently improves over the base model and (Ren et al., 2018) regardless

Model	20 : 1000	50 : 1000	100 : 1000
Base model: BERT (Devlin et al., 2019)	54.91 ± 5.98	67.73 ± 9.20	75.04 ± 4.51
Base model + val-data	52.58 ± 4.58	55.90 ± 4.18	68.21 ± 5.28
Proportion	57.42 ± 7.91	71.14 ± 6.71	76.14 ± 5.8
Ren et al. (2018)	74.61 ± 3.54	76.89 ± 5.07	80.73 ± 2.19
Ours	$\mathbf{75.08 \pm 4.98}$	$\mathbf{79.35 \pm 2.59}$	$\mathbf{81.82 \pm 1.88}$

Table 5.3: Accuracy of Data Weighting on Imbalanced SST-2. The first row shows the number of training examples in each of the two classes.

Model	20 : 1000	50 : 1000	100 : 1000
Base model: ResNet (He et al., 2016b)	72.20 ± 4.70	81.65 ± 2.93	86.42 ± 3.15
Base model + val-data	64.66 ± 4.81	69.51 ± 2.90	79.38 ± 2.92
Proportion	72.29 ± 5.67	81.49 ± 3.83	84.26 ± 4.58
Ren et al. (2018)	74.35 ± 6.37	82.25 ± 2.08	86.54 ± 2.69
Ours	$\mathbf{75.32 \pm 6.36}$	$\mathbf{83.11 \pm 2.08}$	$\mathbf{86.99 \pm 3.47}$

Table 5.4: Accuracy of Data Weighting on Imbalanced CIFAR10. The first row shows the number of training examples in each of the two classes.

of the initialization.

5.5.3 Imbalanced Labels

We next study a different problem setting where the training data of different classes are imbalanced. We show the data weighting approach greatly improves the classification performance. It is also observed that, the LM data augmentation approach, which performs well in the low-data setting, fails on the class-imbalance problems.

Setup Though the methods are broadly applicable to multi-way classification problems, here we only study binary classification tasks for simplicity. For text classification, we use the SST-2 sentiment analysis benchmark (Socher et al., 2013); while for image, we select class 1 and 2 from CIFAR10 for binary classification. We use the same processing on both datasets to build the class-imbalance setting. Specifically, we randomly select 1,000 training instances of class 2, and vary the number of class-1 instances in $\{20, 50, 100\}$. For each dataset, we use 10 validation examples in each class. Trained models are evaluated on the full binary-class test set.

Results Table 5.3 shows the classification results on SST-2 with varying imbalance ratios. We can see our data weighting performs best across all settings. In particular, the improvement over the base model increases as the data gets more imbalanced, ranging from around 6 accuracy points on 100:1000 to over 20 accuracy points on 20:1000. Our method is again consistently better than (Ren et al., 2018), validating that the parametric treatment is beneficial. The proportion-based data weighting provides only limited improvement, showing the advantage of adaptive data weighting. The base model trained on the joint training-validation data for fixed steps fails to perform well, partly due to the lack of a proper mechanism for selecting steps.

Table 5.4 shows the results on imbalanced CIFAR10 classification. Similarly, our method outperforms other comparison approaches. In contrast, the fixed proportion-based method sometimes harms the performance as in the 50:1000 and 100:1000 settings.

We also tested the text augmentation LM on the SST-2 imbalanced data. Interestingly, the augmentation tends to hinder model training and yields accuracy of around 50% (random guess). This is because the augmentation LM is first fit to the imbalanced data, which makes label preservation inaccurate and introduces lots of noise during augmentation. Though a more carefully designed augmentation mechanism can potentially help with imbalanced classification (e.g., augmenting only the rare classes), the above observation further shows that the varying data manipulation schemes have different applicable scopes. Our approach is thus favorable as the single algorithm can be instantiated to learn different schemes.

5.6 Conclusion

We have developed a new method of learning different data manipulation schemes with the same single algorithm. Different manipulation schemes reduce to just different parameterization of the data reward function (i.e., the experience function in the standard equation). The manipulation parameters are trained jointly with the target model parameters. We instantiate the algorithm for data augmentation and weighting, and show improved performance over strong base models and previous manipulation methods. We are excited to explore more types of manipulations such as data synthesis, and in particular study the combination of different manipulation schemes.

Chapter 6

Learning Knowledge Constraints for Enhanced Deep Generative Modeling

In Chapter 3, we have studied the problem of integrating logic rules as experience to learn a target model. The logic rules can usually be specified *a priori* and fixed during learning. Yet, it is often the case in practice that one has only partial or fuzzy domain knowledge which is either impractical (e.g., requiring expensive engineering efforts) or sub-optimal to be fully specified and fixed beforehand. (Figure 6.1 illustrates concrete scenarios as examples.) These situations necessitate more flexible approaches that allow users to express only the modules of knowledge they are certain of, and automatically adapt or learn the remaining modules.

In this chapter, we continue to take advantage of the standardized formalism and derive solutions to the above challenging problem. In particular, with the similar idea as in the previous chapter, here we base on the equivalence between knowledge constraints and reward (both corresponding to the experience function in the standard equation), and enable updates of the knowledge constraints by seamlessly repurposing a well-known inverse reinforcement learning algorithm that was originally designed to learn the reward function in the reinforcement learning context. The resulting approach is model-agnostic to apply to any model, in particular deep generative models as we study here, and is flexible to adapt arbitrary constraints with the model jointly. Experiments on human image generation and templated sentence generation show models with learned knowledge constraints by our approach greatly improve over base generative models.

Besides, we show the resulting joint model-constraint updates rediscover the generative adversarial learning, with crucial difference from the conventional formulation, which leads to more stable training of GANs. The next chapter deepens the discussion.

6.1 Introduction

Generative models provide a powerful mechanism for learning data distributions and simulating samples. Recent years have seen remarkable advances especially on the deep approaches (Goodfellow et al., 2016; Hu et al., 2018b) such as Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), Variational Autoencoders (VAEs) (Kingma and Welling, 2013), auto-regressive networks (Larochelle and Murray, 2011; Oord et al., 2016), and so forth. However, it is usually difficult to exploit in these various deep generative models rich problem structures and domain knowledge (e.g., the human body structure in image generation, Figure 6.1). Many times we have to hope the deep networks can discover the structures from massive data by themselves, leaving much valuable domain knowledge unused. Recent efforts of designing specialized network architectures or learning disentangled representations (Chen et al., 2016b; Hu et al., 2017a) are usually only applicable to specific knowledge, models, or tasks. It is therefore highly desirable to have a *general* means of incorporating arbitrary structured knowledge with any types of deep generative models in a principled way.

On the other hand, posterior regularization (PR) (Ganchev et al., 2010) is a principled framework to impose knowledge constraints on posterior distributions of probabilistic models, and has shown effectiveness in regulating the learning of models in different context. For example, Hu et al. (2016a) extends PR to incorporate structured logic rules with neural classifiers. However, the previous approaches are not directly applicable to the general case of deep generative models, as many of the models (e.g., GANs, many auto-regressive networks) are not straightforwardly formulated with the probabilistic Bayesian framework and do not possess a posterior distribution or even meaningful latent variables. Moreover, PR has required *a priori* fixed constraints. That means users have to fully specify the constraints beforehand, which can be impractical due to heavy engineering, or suboptimal without adaptivity to the data and models. To extend the scope of applicable knowledge and reduce engineering burden, it is necessary to allow users to specify only *partial* or *fuzzy* structures, while learning remaining parts of the constraints jointly with the regulated model.

To this end, we establish formal connections between the PR framework with a broad set of algorithms in the control and reinforcement learning (RL) domains, and, based on the connections, transfer well-developed RL techniques for constraint learning in PR. In particular, though the PR framework and the RL are apparently distinct paradigms applied in different context, we show mathematical correspondence between the model and constraints in PR with the policy and reward in entropy-regularized policy optimization (Peters et al., 2010; Schulman et al., 2015b; Abdolmaleki et al., 2018), respectively. This thus naturally inspires to leverage relevant approach from the RL domain (specifically, the maximum entropy inverse RL (Ziebart et al., 2008; Finn et al., 2016b)) to learn the PR constraints from data (i.e., demonstrations in RL).

Based on the unified perspective, we drive a practical algorithm with efficient estimations and moderate approximations. The algorithm is efficient to regularize large target space with arbitrary constraints, flexible to couple adapting the constraints with learning the model, and model-agnostic to apply to diverse deep generative models, including implicit models where generative density cannot be evaluated (Mohamed and Lakshminarayanan, 2016; Goodfellow et al., 2014). We demonstrate the effectiveness of the proposed approach in both image and text generation (Figure 6.1). Leveraging domain knowledge of structure-preserving constraints, the resulting models improve over base generative models.

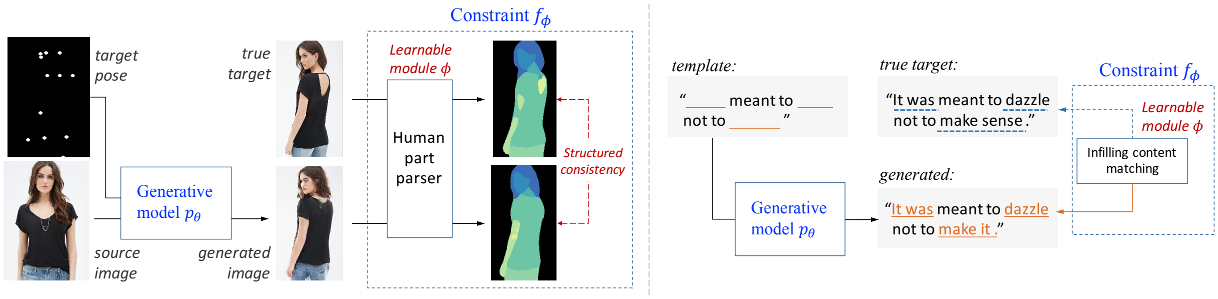


Figure 6.1: Two example applications of imposing learnable knowledge constraints on generative models. **Left:** Given a person image and a target pose (defined by key points), the goal is to generate an image of the person under the new pose. The constraint is to force the human parts (e.g., head) of the generated image to match those of the true target image. **Right:** Given a text template, the goal is to generate a complete sentence following the template. The constraint is to force the match between the infilling content of the generated sentence with the true content. (See sec 6.5 for more details.)

6.2 Related Work

It is of increasing interest to incorporate problem structures and domain knowledge in machine learning approaches (Taskar et al., 2004; Ganchev et al., 2010; Hu et al., 2016a). The added structure helps to facilitate learning, enhance generalization, and improve interpretability. For deep neural models, one of the common ways is to design specialized network architectures or features for specific tasks (e.g., Andreas et al. (2016); Liang et al. (2018); Kusner et al. (2017); Liang et al. (2017)). Such a method typically has a limited scope of applicable tasks, models, or knowledge. On the other hand, for structured probabilistic models, posterior regularization (PR) and related frameworks (Ganchev et al., 2010; Liang et al., 2009; Bellare et al., 2009) provide a general means to impose knowledge constraints during model estimation. Hu et al. (2016a) develops *iterative knowledge distillation* based on PR to regularize neural networks with any logic rules. However, the application of PR to the broad class of deep generative models has been hindered, as many of the models do not even possess meaningful latent variables or explicit density evaluation (i.e., implicit models). Previous attempts thus are limited to applying simple max-margin constraints (Li et al., 2015). The requirement of *a priori* fixed constraints has also made PR impractical for complex, uncertain knowledge. Previous efforts to alleviate the issue either require additional manual supervision (Mei et al., 2014) or is limited to regularizing small label space (Hu et al., 2016b). This paper develops a *practical* algorithm that is generally applicable to any deep generative models and any learnable constraints on arbitrary (large) target space.

Our work builds connections between the Bayesian PR framework and reinforcement learning. A relevant, broad research topic of formalizing RL as a probabilistic inference problem has been explored in the RL literature (Dayan and Hinton, 1997; Deisenroth et al., 2013; Neumann et al., 2011; Levine, 2018; Abdolmaleki et al., 2018; Tan et al., 2018), where rich approximate inference tools are used to improve the modeling and reasoning for various RL algorithms. The link

Components	PR	Entropy-Reg RL	MaxEnt IRL	(Energy) GANs
$\boldsymbol{x}$	data/generations	action-state samples	demonstrations	data/generations
$p(\boldsymbol{x})$	generative model p_θ	(old) policy p_π	—	generator
$f(\boldsymbol{x})/R(\boldsymbol{x})$	constraint f_ϕ	reward R	reward R_ϕ	discriminator
$q(\boldsymbol{x})$	variational distr. q, Eq.6.3.3	(new) policy q_π	policy q_ϕ	—

Table 6.1: Unified perspective of the different approaches, showing mathematical correspondence of PR with the entropy-regularized RL (sec 6.3.2.1) and maximum entropy IRL (sec 6.3.2.2), and its (conceptual) relations to (energy-based) GANs (sec 6.4).

between RL and PR has not been previously studied. We establish the mathematical correspondence, and, differing from the RL literature, we in turn transfer the tools from RL to expand the probabilistic PR framework. Inverse reinforcement learning (IRL) seeks to learn a reward function from expert demonstrations. Recent approaches based on maximum-entropy IRL (Ziebart et al., 2008) are developed to learn both the reward and policy (Finn et al., 2016b,a; Fu et al., 2017a). We adopt the maximum-entropy IRL formulation to derive the constraint learning objective in our algorithm, and leverage the unique structure of PR for efficient importance sampling estimation, which differs from these previous approaches.

6.3 Connecting PR and RL

In this section, we provide the background of posterior regularization (PR) and discuss its connection with reinforcement learning techniques. Some of the discussions have been covered in Chapter 2, while we replicate here for completeness and readability. Note that we use slightly different notations from Chapter 2 for ease of presentation.

6.3.1 PR for Deep Generative Models

PR (Ganchev et al., 2010) was originally proposed to provide a principled framework for incorporating constraints on posterior distributions of probabilistic models with latent variables. The formulation is not generally applicable to deep generative models as many of them (e.g., GANs and autoregressive models) are not formulated within the Bayesian framework and do not possess a valid posterior distribution or even semantically meaningful latent variables. Here we adopt a slightly adapted formulation that makes minimal assumptions on the specifications of the model to regularize. It is worth noting that though we present in the generative model context, the formulations, including the algorithm developed later (sec 6.4), can straightforwardly be extended to other settings such as discriminative models.

Consider a generative model $\boldsymbol{x} \sim p_\theta(\boldsymbol{x})$ with parameters $\boldsymbol{\theta}$. Note that generation of $\boldsymbol{x}$ can condition on arbitrary other elements (e.g., the source image for image transformation) which are omitted for simplicity of notations. Denote the original objective of $p_\theta(\boldsymbol{x})$ with $\mathcal{L}(\boldsymbol{\theta})$. PR augments the objective by adding a constraint term encoding the domain knowledge. Without loss of generality, consider constraint function $f(\boldsymbol{x}) \in \mathbb{R}$, such that a higher $f(\boldsymbol{x})$ value indicates a better $\boldsymbol{x}$ in terms of the particular knowledge. Note that f can also involve other factors such as latent variables and extra supervisions, and can include a set of multiple constraints.

A straightforward way to impose the constraint on the model is to maximize $\mathbb{E}_{p_\theta}\left[f(\boldsymbol{x})\right]$. Such method is efficient only when p_θ is a GAN-like implicit generative model or an explicit distribution that can be efficiently reparameterized (e.g., Gaussian Kingma and Welling (2013)). For other models such as the large set of non-reparameterizable explicit distributions, the gradient $\nabla_\theta \mathbb{E}_{p_\theta}\left[f(\boldsymbol{x})\right]$ is usually computed with the *log-derivative* trick and can suffer from high variance. For broad applicability and efficient optimization, PR instead imposes the constraint on an auxiliary variational distribution q, which is encouraged to stay close to p_θ through a KL divergence term:

$$\mathcal{L}(\boldsymbol{\theta}, q) = (q(\boldsymbol{x}) \| p_\theta(\boldsymbol{x})) - \alpha \mathbb{E}_q\left[f(\boldsymbol{x})\right], \tag{6.3.1}$$

where α is the weight of the constraint term. The PR objective for learning the model is written as:

$$\min_{\theta, q} \mathcal{L}(\boldsymbol{\theta}) + \lambda \mathcal{L}(\boldsymbol{\theta}, q), \tag{6.3.2}$$

where λ is the balancing hyperparameter. As optimizing the original model objective $\mathcal{L}(\boldsymbol{\theta})$ is straightforward and depends on the specific generative model of choice, in the following we omit the discussion of $\mathcal{L}(\boldsymbol{\theta})$ and focus on $\mathcal{L}(\boldsymbol{\theta}, q)$ introduced by the framework.

The problem is solved using an EM-style algorithm (Ganchev et al., 2010; Hu et al., 2016a). Specifically, the E-step optimizes Eq.(6.3.1) w.r.t q, which is convex and has a closed-form solution at each iteration given $\boldsymbol{\theta}$:

$$q^*(\boldsymbol{x}) = p_\theta(\boldsymbol{x}) \exp\left\{\alpha f(\boldsymbol{x})\right\}/Z, \tag{6.3.3}$$

where Z is the normalization term. We can see q^* as an energy-based distribution with the negative energy defined by $\alpha f(\boldsymbol{x}) + \log p_\theta(\boldsymbol{x})$. With q from the E-step fixed, the M-step optimizes Eq.(6.3.1) w.r.t $\boldsymbol{\theta}$ with:

$$\min_\theta (q(\boldsymbol{x}) \| p_\theta(\boldsymbol{x})) = \min_\theta -\mathbb{E}_q\left[\log p_\theta(\boldsymbol{x})\right] + const. \tag{6.3.4}$$

Constraint f in PR has to be fully-specified *a priori* and is fixed throughout the learning. It would be desirable or even necessary to enable learnable constraints so that practitioners are allowed to specify only the known components of f while leaving any unknown or uncertain components automatically learned. For example, for human image generation in Figure 6.1, left panel, users are able to specify structures on the parsed human parts, while it is impractical to also manually engineer the human part parser that involves recognizing parts from raw image pixels. It is favorable to instead cast the parser as a learnable module in the constraint. Though it is possible to pre-train the module and simply fix in PR, the lack of adaptivity to the data and model can lead to suboptimal results, as shown in the empirical study (Table 6.2). This necessitates to expand the PR framework to enable joint learning of constraints with the model.

Denote the constraint function with learnable components as $f_\phi(\boldsymbol{x})$, where $\boldsymbol{\phi}$ can be of various forms that are optimizable, such as the free parameters of a structural model, or a graph structure to optimize.

Simple way of learning the constraint. A straightforward way to learn the constraint is to directly optimize Eq.(6.3.1) w.r.t $\boldsymbol{\phi}$ in the M-step, yielding

$$\max_\phi \mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x})}[f_\phi(\boldsymbol{x})]. \tag{6.3.5}$$

That is, the constraint is trained to fit to the samples from the current regularized model q. However, such objective can be problematic as the generated samples can be of low quality, e.g., due to poor state of the generative parameter $\boldsymbol{\theta}$ at initial stages, or insufficient capability of the generative model per se.

In this paper, we propose to treat the learning of constraint as an *extrinsic reward*, as motivated by the connections between PR with the reinforcement learning domain presented below.

6.3.2 PR and RL

RL or optimal control has been studied primarily for determining optimal action sequences or strategies, which is significantly different from the context of PR that aims at regulating generative models. However, formulations very similar to PR (e.g., Eqs.6.3.1 and 6.3.3) have been developed and widely used, in both the (forward) RL for policy optimization and the inverse RL for reward learning.

To make the mathematical correspondence clearer, we intentionally re-use most of the notations from PR. Table 6.1 lists the correspondence. Specifically, consider a stationary Markov decision process (MDP). An agent in state s draws an action a following the policy $p_\pi(a|s)$. The state subsequently transfers to s' (with some transition probability of the MDP), and a reward is obtained $R(s,a)\in\mathbb{R}$. Let $\boldsymbol{x}=(s,a)$ denote the state-action pair, and $p_\pi(\boldsymbol{x})=\mu^\pi(s)p_\pi(a|s)$ where $\mu^\pi(s)$ is the stationary state distribution (Sutton and Barto, 2017).

6.3.2.1 Entropy regularized policy optimization

The goal of policy optimization is to find the optimal policy that maximizes the expected reward. The rich research line of entropy regularized policy optimization has augmented the objective with information theoretic regularizers such as KL divergence between the new policy and the old policy for stabilized learning. With a slight abuse of notations, let $q_\pi(\boldsymbol{x})$ denote the new policy and $p_\pi(\boldsymbol{x})$ the old one. A prominent algorithm for example is the relative entropy policy search (REPS) (Peters et al., 2010) which follows the objective:

$$\min_{q_\pi} \mathcal{L}(q_\pi) = (q_\pi(\boldsymbol{x})\|p_\pi(\boldsymbol{x})) - \alpha\mathbb{E}_{q_\pi}\left[R(\boldsymbol{x})\right], \tag{6.3.6}$$

where the KL divergence prevents the policy from changing too rapidly. Similar objectives have also been widely used in other workhorse algorithms such as trust-region policy optimization (TRPO) (Schulman et al., 2015b), soft Q-learning (Haarnoja et al., 2017; Schulman et al., 2017a), and others.

We can see the close resemblance between Eq.(6.3.6) with the PR objective in Eq.(6.3.1), where the generative model $p_\theta(\boldsymbol{x})$ in PR corresponds to the reference policy $p_\pi(\boldsymbol{x})$, while the constraint $f(\boldsymbol{x})$ corresponds to the reward $R(\boldsymbol{x})$. The new policy q_π can be either a parametric distribution (Schulman et al., 2015b) or a non-parametric distribution (Peters et al., 2010; Abdolmaleki et al., 2018). For the latter, the optimization of Eq.(6.3.6) precisely corresponds to the E-step of PR, yielding the optimal policy $q^*_\pi(\boldsymbol{x})$ that takes the same form of $q^*(\boldsymbol{x})$ in Eq.(6.3.3), with p_θ and f replaced with the respective counterparts p_π and R, respectively. The parametric policy p_π is subsequently updated with samples from q^*_π, which is exactly equivalent to the M-step in PR (Eq.6.3.4).

While the above policy optimization algorithms have assumed a reward function given by the external environment, just as the pre-defined constraint function in PR, the strong connections above inspire us to treat the PR constraint as an extrinsic reward, and utilize the rich tools in RL (especially the inverse RL) for learning the constraint.

6.3.2.2 *Maximum entropy inverse reinforcement learning*

Maximum entropy (MaxEnt) IRL (Ziebart et al., 2008) is among the most widely-used methods that induce the reward function from expert demonstrations $\boldsymbol{x} \sim p_d(\boldsymbol{x})$, where p_d is the empirical demonstration (data) distribution. MaxEnt IRL adopts the same principle as the above entropy regularized RL (Eq.6.3.6) that maximizes the expected reward regularized by the relative entropy (i.e., the KL), except that, in MaxEnt IRL, p_π is replaced with a *uniform* distribution and the regularization reduces to the entropy of q_π. Therefore, same as above, the optimal policy takes the form $\exp\{\alpha R(\boldsymbol{x})\}/Z$. MaxEnt IRL assumes the demonstrations are drawn from the optimal policy. Learning the reward function $R_\phi(\boldsymbol{x})$ with unknown parameters $\boldsymbol{\phi}$ is then cast as maximizing the likelihood of the distribution $q_\phi(\boldsymbol{x}) := \exp\{\alpha R_\phi(\boldsymbol{x})\}/Z_\phi$:

$$\boldsymbol{\phi}^* = \arg\max\nolimits_\phi \mathbb{E}_{\boldsymbol{x}\sim p_d}\left[\log q_\phi(\boldsymbol{x})\right]. \tag{6.3.7}$$

Given the direct correspondence between the policy q_{ϕ^*} in MaxEnt IRL and the policy optimization solution q^*_π of Eq.(6.3.6), plus the connection between the regularized distribution q^* of PR (Eq.6.3.3) and q^*_π as built in sec 6.3.2.1, we can readily link q^* and q_{ϕ^*}. This motivates to plug q^* in the above maximum likelihood objective to learn the constraint $f_\phi(\boldsymbol{x})$ which is parallel to the reward function $R_\phi(\boldsymbol{x})$. We present the resulting full algorithm in the next section. Table 6.1 summarizes the correspondence between PR, entropy regularized policy gradient, and maximum entropy IRL.

6.4 Algorithm

We have formally related PR to the RL methods. With the unified view of these approaches, we derive a practical algorithm for arbitrary learnable constraints on any deep generative models. The algorithm alternates the optimization of the constraint f_ϕ and the generative model p_θ.

6.4.1 Learning the Constraint f_ϕ

As motivated in section 6.3.2, instead of directly optimizing f_ϕ in the original PR objectives (Eq.6.3.5) which can be problematic, we treat f_ϕ as the reward function to be induced with the MaxEnt IRL framework. That is, we maximize the data likelihood of $q(\boldsymbol{x})$ (Eq.6.3.3) w.r.t $\boldsymbol{\phi}$, yielding the gradient:

$$\begin{aligned}\nabla_\phi \mathbb{E}_{\boldsymbol{x}\sim p_d}\left[\log q(\boldsymbol{x})\right] &= \nabla_\phi \big[\mathbb{E}_{\boldsymbol{x}\sim p_d}\left[\alpha f_\phi(\boldsymbol{x})\right] - \log Z_\phi\big] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_d}\left[\alpha \nabla_\phi f_\phi(\boldsymbol{x})\right] - \mathbb{E}_{q(\boldsymbol{x})}\left[\alpha \nabla_\phi f_\phi(\boldsymbol{x})\right].\end{aligned} \tag{6.4.1}$$

The second term involves estimating the expectation w.r.t an energy-based distribution $\mathbb{E}_{q(\boldsymbol{x})}[\cdot]$, which is in general very challenging. However, we can exploit the special structure of $q \propto p_\theta \exp\{\alpha f_\phi\}$ for efficient approximation. Specifically, we use p_θ as the proposal distribution, and obtain the importance sampling estimate of the second term as following:

$$\begin{aligned}\mathbb{E}_{q(\boldsymbol{x})}\left[\alpha \nabla_\phi f_\phi(\boldsymbol{x})\right] &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\frac{q(\boldsymbol{x})}{p_\theta(\boldsymbol{x})} \cdot \alpha \nabla_\phi f_\phi(\boldsymbol{x})\right] \\ &= 1/Z_\phi \cdot \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}\left[\exp\{\alpha f_\phi(\boldsymbol{x})\} \cdot \alpha \nabla_\phi f_\phi(\boldsymbol{x})\right].\end{aligned} \tag{6.4.2}$$

Note that the normalization $Z_\phi = \int p_\theta(\boldsymbol{x}) \exp\{\alpha f_\phi(\boldsymbol{x})\}$ can also be estimated efficiently with MC sampling: $\hat{Z}_\phi = 1/N \sum_{x_i} \exp\{\alpha f_\phi(\boldsymbol{x}_i)\}$, where $\boldsymbol{x}_i \sim p_\theta$. The base generative distribution p_θ is a natural choice for the proposal as it is in general amenable to efficient sampling, and is close to q as forced by the KL divergence in Eq.(6.3.1). Our empirical study shows low variance of the learning process (sec 6.5). Moreover, using p_θ as the proposal distribution allows p_θ to be an implicit generative model (as no likelihood evaluation of p_θ is needed). Note that the importance sampling estimation is consistent yet biased.

6.4.2 Learning the Generative Model p_θ

Given the current parameter state ($\boldsymbol{\theta} = \boldsymbol{\theta}^t, \boldsymbol{\phi} = \boldsymbol{\phi}^t$), and $q(\boldsymbol{x})$ evaluated at the parameters, we continue to update the generative model. Recall that optimization of the generative parameter $\boldsymbol{\theta}$ is performed by minimizing the KL divergence in Eq.(6.3.4), which we replicate here:

$$\min\nolimits_\theta (q(\boldsymbol{x}) \| p_\theta(\boldsymbol{x})) = \min\nolimits_\theta -\mathbb{E}_{q(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] + const. \tag{6.4.3}$$

The expectation w.r.t $q(\boldsymbol{x})$ can be estimated as above (Eq.6.4.2). A drawback of the objective is the requirement of evaluating the generative density $p_\theta(\boldsymbol{x})$, which is incompatible to the emerging *implicit* generative models (Mohamed and Lakshminarayanan, 2016) that only permit simulating samples but not evaluating density.

To address the restriction, when it comes to regularizing implicit models, we propose to instead minimize the *reverse* KL divergence:

$$\begin{aligned}\min\nolimits_\theta \left(p_\theta(\boldsymbol{x}) \| q(\boldsymbol{x})\right) &= \min\nolimits_\theta \mathbb{E}_{p_\theta}\left[\log \frac{p_\theta \cdot Z_{\phi^t}}{p_{\theta^t} \exp\{\alpha f_{\phi^t}\}}\right] \\ &= \min\nolimits_\theta -\mathbb{E}_{p_\theta}\left[\alpha f_{\phi^t}(\boldsymbol{x})\right] + (p_\theta \| p_{\theta^t}) + const.\end{aligned} \tag{6.4.4}$$

By noting that $\nabla_\theta \left(p_\theta \| p_{\theta^t}\right)|_{\theta=\theta^t} = 0$, we obtain the gradient w.r.t $\boldsymbol{\theta}$:

$$\nabla_\theta \left(p_\theta(\boldsymbol{x}) \| q(\boldsymbol{x})\right)|_{\theta=\theta^t} = -\nabla_\theta \mathbb{E}_{p_\theta}\left[\alpha f_{\phi^t}(\boldsymbol{x})\right]|_{\theta=\theta^t}. \tag{6.4.5}$$

That is, the gradient of minimizing the reversed KL divergence equals the gradient of maximizing $\mathbb{E}_{p_\theta}\left[\alpha f_{\phi^t}(\boldsymbol{x})\right]$. Intuitively, the objective encourages the generative model p_θ to generate samples that the constraint function assigns high scores. Though the objective for implicit model deviates the original PR framework, reversing KL for computationality was also used previously such as in the classic wake-sleep method (Hinton et al., 1995). The resulting algorithm also resembles the adversarial learning in GANs, as we discuss in the next section. Empirical results on implicit models show the effectiveness of the objective.

The resulting algorithm is summarized in Alg.6.

Algorithm 5 Joint Learning of Deep Generative Model and Constraints

```
Input: The base generative model $p_\theta(\boldsymbol{x})$
       The (set of) constraints $f_\phi(\boldsymbol{x})$
Initialize generative parameter $\boldsymbol{\theta}$ and constraint parameter $\boldsymbol{\phi}$
repeat
    Optimize constraints $\boldsymbol{\phi}$ with Eq.(6.4.1)
    if $p_\theta$ is an implicit model then
        Optimize model $\boldsymbol{\theta}$ with Eq.(6.4.5) along with minimizing original model objective $\mathcal{L}(\boldsymbol{\theta})$
    else
        Optimize model $\boldsymbol{\theta}$ with Eq.(6.4.3) along with minimizing $\mathcal{L}(\boldsymbol{\theta})$
    end if
until convergence
Output: Jointly learned generative model $p_{\theta^*}(\boldsymbol{x})$ and constraints $f_{\phi^*}(\boldsymbol{x})$
```

Connections to adversarial learning For implicit generative models, the two objectives w.r.t $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ (Eq.6.4.1 and Eq.6.4.5) are conceptually similar to the adversarial learning in GANs (Goodfellow et al., 2014) and the variants such as energy-based GANs (Kim and Bengio, 2016; Zhao et al., 2016; Zhai et al., 2016; Wang and Liu, 2016). Specifically, the constraint $f_\phi(\boldsymbol{x})$ can be seen as being optimized to assign lower energy (with the energy-based distribution $q(\boldsymbol{x})$) to real examples from $p_d(\boldsymbol{x})$, and higher energy to fake samples from $q(\boldsymbol{x})$ which is the regularized model of the generator $p_\theta(\boldsymbol{x})$. In contrast, the generator $p_\theta(\boldsymbol{x})$ is optimized to generate samples that confuse f_ϕ and obtain lower energy. Such adversarial relation links the PR constraint $f_\phi(\boldsymbol{x})$ to the discriminator in GANs (Table 6.1). Note that here fake samples are generated from $q(\boldsymbol{x})$ and $p_\theta(\boldsymbol{x})$ in the two learning phases, respectively, which differs from previous adversarial methods for energy-based model estimation that simulate only from a generator. Besides, distinct from the discriminator-centric view of the previous work (Kim and Bengio, 2016; Zhai et al., 2016; Wang and Liu, 2016), we primarily aim at improving the generative model by incorporating learned constraints. Last but not the least, as discussed in sec 6.3.1, the proposed framework and algorithm are more generally and efficiently applicable to not only implicit generative models as in GANs, but also (non-)reparameterizable explicit generative models.

6.5 Experiments

We demonstrate the applications and effectiveness of the algorithm in two tasks related to image and text generation, respectively.

	Method	SSIM	Human
1	Ma et al. (2018)	0.614	—
2	Pumarola et al. (2018)	0.747	—
3	Ma et al. (2017)	0.762	—
4	Base model	0.676	0.03
5	With fixed constraint	0.679	0.12
6	With learned constraint	**0.727**	**0.77**

Table 6.2: Results of image generation on Structural Similarity (SSIM) (Wang et al., 2004) between generated and true images, and human survey where the full model yields better generations than the base models (Rows 5-6) on 77% test cases. See the text for more results and discussion.

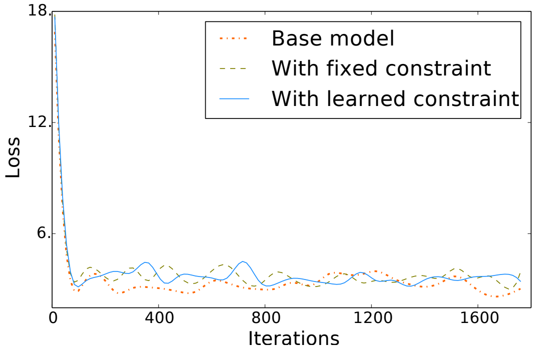


Figure 6.2: Training losses of the three models. The model with learned constraint converges smoothly as base models.

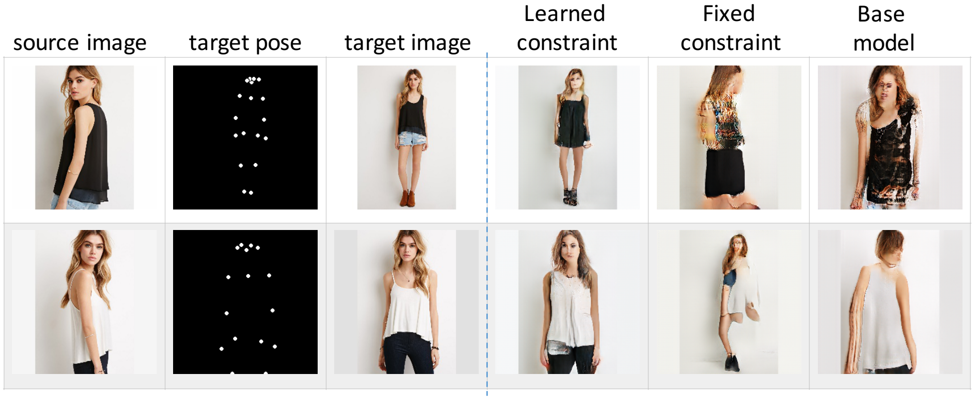


Figure 6.3: Samples generated by the models in Table 6.2. The model with learned human part constraint generates correct poses and preserves human body structure much better.

6.5.1 Pose Conditional Person Image Generation

Given a person image and a new body pose, the goal is to generate an image of the same person under the new pose (Figure 6.1, left). The task is challenging due to body self-occlusions and many cloth and shape ambiguities. Complete end-to-end generative networks have previously failed (Ma et al., 2017) and existing work designed specialized generative processes or network architectures (Ma et al., 2017; Pumarola et al., 2018; Ma et al., 2018). We show that with an added body part consistency constraint, a plain end-to-end generative model can also be trained to produce highly competitive results, significantly improving over base models that do not incorporate the problem structure.

Setup. We follow the previous work (Ma et al., 2017) and obtain from DeepFashion (Liu et al., 2016) a set of triples (source image, pose keypoints, target image) as supervision data. The base generative model p_ϕ is an implicit model that transforms the input source and pose directly to the pixels of generated image (and hence defines a Dirac-delta distribution). We use the residual block architecture (Wang et al., 2017) widely-used in image generation for the generative model. The base model is trained to minimize the L1 distance loss between the real and generated pixel values, as well as to confuse a binary discriminator that distinguishes between the generation and the true target image.

Knowledge constraint. Neither the pixel-wise distance nor the binary discriminator loss encode any task structures. We introduce a structured consistency constraint f_ϕ that encourages each of the body parts (e.g., head, legs) of the generated image to match the respective part of the true image. Specifically, the constraint f_ϕ includes a human parsing module that classifies each pixel of a person image into possible body parts. The constraint then evaluates cross entropies of the per-pixel part distributions between the generated and true images. The average negative cross entropy serves as the constraint score. The parsing module is parameterized as a neural network with parameters $\boldsymbol{\phi}$, pre-trained on an external parsing dataset (Gong et al., 2017), and subsequently adapted within our algorithm jointly with the generative model.

Results. Table 6.2 compares the full model (with the learned constraint, Row 6) with the base model (Row 4) and the one regularized with the constraint that is fixed after pre-training (Row 5). Human survey is performed by asking annotators to rank the quality of images generated by the three models on each of 200 test cases, and the percentages of ranked as the best are reported (Tied ranking is treated as negative result). We can see great improvement by the proposed algorithm. The model with fixed constraint fails, partially because pre-training on external data does not necessarily fit to the current problem domain. This highlights the necessity of the constraint learning. Figure 6.3 shows examples further validating the effectiveness of the algorithm.

In sec 6.4, we have discussed the close connection between the proposed algorithm and (energy-based) GANs. The conventional discriminator in GANs can be seen as a special type of constraint. With this connection and given that the generator in the task is an implicit generative model, here we can also apply and learn the structured consistency constraint using GANs, which is equivalent to replacing $q(\boldsymbol{x})$ in Eq.(6.4.1) with $p_\theta(\boldsymbol{x})$. Such a variant produces a SSIM score of 0.716, slightly inferior to the result of the full algorithm (Row 6). We suspect this is because fake samples by q (instead of p) can help with better constraint learning. It would be interesting to explore this in more applications.

To give a sense of the state of the task, Table 6.2 also lists the performance of previous work. It is worth noting that these results are not directly comparable, as discussed in (Pumarola et al., 2018), due to different settings (e.g., the test splits) between each of them. We follow (Ma et al., 2017, 2018) mostly, while our generative model is much simpler than these work with specialized, multi-stage architectures. The proposed algorithm learns constraints with moderate approximations. Figure 6.2 validates that the training is stable and converges smoothly as the

	Model	Perplexity	Human
1	Base model	30.30	0.19
2	With binary D	30.01	0.20
3	With constraint updated in M-step (Eq.6.3.5)	31.27	0.15
4	With learned constraint	**28.69**	**0.24**

Table 6.3: Sentence generation results on test set perplexity and human survey. Samples by the full model are considered as of higher quality in 24% cases.

______	acting	______
the	acting	is the acting .
the	acting	is also very good .
______	out of 10 .	
10	out of 10 .	
I will give the movie 7	out of 10 .	

Table 6.4: Two test examples, including the template, the sample by the base model, and the sample by the constrained model.

base models.

6.5.2 Template Guided Sentence Generation

The task is to generate a text sentence $\boldsymbol{x}$ that follows a given template $\boldsymbol{t}$ (Figure 6.1, right). Each missing part in the template can contain arbitrary number of words. This differs from previous sentence completion tasks (Fedus et al., 2018; Zweig and Burges, 2011) which designate each masked position to have a single word. Thus directly applying these approaches to the task can be problematic.

Setup. We use an attentional sequence-to-sequence (seq2seq) (Bahdanau et al., 2014) model $p_\theta(\boldsymbol{x}|\boldsymbol{t})$ as the base generative model for the task. Paired (template, sentence) data is obtained by randomly masking out different parts of sentences from the IMDB corpus (Diao et al., 2014). The base model is trained in an end-to-end supervised manner, which allows it to memorize the words in the input template and repeat them almost precisely in the generation. However, the main challenge is to generate meaningful and coherent content to fill in the missing parts.

Knowledge constraint. To tackle the issue, we add a constraint that enforces matching between the generated sentence and the ground-truth text in the missing parts. Specifically, let $\boldsymbol{t}_-$ be the masked-out true text. That is, plugging $\boldsymbol{t}_-$ into the template $\boldsymbol{t}$ recovers the true complete sentence. The constraint is defined as $f_\phi(\boldsymbol{x}, \boldsymbol{t}_-)$ which returns a high score if the sentence $\boldsymbol{x}$ matches $\boldsymbol{t}_-$ well. The actual implementation of the matching strategy can vary. Here we simply specify f_ϕ as another seq2seq network that takes as input a sentence $\boldsymbol{x}$ and evaluates the likelihood of recovering $\boldsymbol{t}_-$—This is all we have to specify, while the unknown parameters $\boldsymbol{\phi}$ are learned jointly with the generative model. Despite the simplicity, the empirical results show the usefulness of the constraint.

Results. Table 6.3 shows the results. Row 2 is the base model with an additional binary discriminator that adversarial distinguishes between the generated sentence and the ground truth (i.e., a GAN model). Row 3 is the base model with the constraint learned in the direct way through Eq.(6.3.5). We see that the improper learning method for the constraint harms the model performance, partially because of the relatively low-quality model samples the constraint is trained to fit. In contrast, the proposed algorithm effectively improves the model results. Its superiority over the binary discriminator (Row 2) shows the usefulness of incorporating problem structures. Table 6.4 demonstrates samples by the base and constrained models. Without the explicit constraint forcing in-filling content matching, the base model tends to generate less meaningful content (e.g., duplications, short and general expressions).

6.6 Conclusion

We leveraged the connections between posterior regularization and reinforcement learning, treated the problem of learning the knowledge constraints in PR as reward learning in RL. The resulting approach is generally applicable to any deep generative models, and flexible to learn the constraints and model jointly. Experiments on image and text generation showed the effectiveness of the algorithm.

Chapter 7

Improving GAN Training with Probability Ratio Clipping and Sample Reweighting

Despite success on a wide range of problems related to vision, generative adversarial networks (GANs) often suffer from inferior performance due to unstable training, especially for text generation. In this chapter, we consider the problem of stabilizing the notoriously difficult GAN training—a problem that is important in the field, but seemingly unrelated with those addressed in the previous chapter. However, based on the standard equation, we show that the adversarial learning mechanism can be derived in the same way as how we derived the automated knowledge constraint adaptation in Chapter 6, and its stabilization can similarly be achieved by repurposing successful policy optimization algorithms in reinforcement learning.

Indeed, in Chapter 2 (section 2.4.1), we have discussed how the conventional GAN and many of its variants are subsumed as special instances of the standard equation by treating the GAN discriminator as a type of "variational experience". By setting the divergence $\mathbb{D}$ to the KL divergence, we obtained a new variant of GANs that bears close connections to the regularized policy optimization algorithms including Trust-Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO). The work in this chapter continues the study of the connections and develops a practical algorithm that imports efficient regularization and approximation techniques from those most successful policy optimization approaches to the GAN context for stabilized training. The work also provides alternative derivations that treat the updates of the discriminator-based experience as an "inverse RL" procedure (same as in Chapter 6) and arrive at the same variational experience formulation of GANs.

As a result, the new GAN training approach consists of two key components: (1) probability ratio clipping that regularizes generator training to prevent excessively large updates, and (2) a sample re-weighting mechanism that improves discriminator training by downplaying bad-quality fake samples. By plugging the training approach in diverse state-of-the-art GAN architectures, we obtain significantly improved performance over a range of tasks, including text generation, text style transfer, and image generation.

7.1 Introduction

Generative adversarial networks (GANs) (Goodfellow et al., 2014) have achieved remarkable success in image and video synthesis (Radford et al., 2015a; Brock et al., 2018; Mathieu et al., 2015). However, it is usually hard to train a GAN well, because the training process is commonly unstable, subject to disturbances and even collapses. To alleviate this issue, substantial efforts have been paid to improve the training stability from different perspectives, e.g., divergence minimization (Nowozin et al., 2016; Nock et al., 2017), Wasserstein distance with Lipschitz continuity of the discriminator (Arjovsky et al., 2017a; Gulrajani et al., 2017; Wei et al., 2018), energy-based models (Zhao et al., 2016; Berthelot et al., 2017), to name a few.

In spite of the above progresses, the instability in training has not been well resolved (Chu et al., 2020), since it is difficult to well balance the strength of the generator and the discriminator. What is worse, such an instability issue is exacerbated in text generation due to the sequential and discrete nature of text (Fedus et al., 2018; Caccia et al., 2020; Hu et al., 2017b). Specifically, the high sensitivity of text generation to noise and the underlying errors caused by sparse discriminator signals in the generated text can often result in destructive updates to both generator and discriminator, enlarging the instability in GANs.

In this work, we develop a novel variational GAN training framework to improve the training stability, which is broadly applicable to GANs of a variety of architectures for image and text generation. This training framework is derived from a variational perspective of GANs (Hu et al., 2018a) and the resulting connections to reinforcement learning (in particular, RL-as-inference) (Abdolmaleki et al., 2018; Schulman et al., 2017b) and other rich literature (Grover et al., 2019; Hu et al., 2017c; Burda et al., 2015). Our approach consists of two stabilization techniques, namely, probability ratio clipping and sample re-weighting, for stabilizing the generator and discriminator respectively. **(1)** Under the variational perspective, the generator update is subject to a KL penalty on the change of the generator distribution. This KL penalty closely resembles that in the popular Trust-Region Policy Optimization (TRPO) (Schulman et al., 2015a) and its variant, i.e., Proximal Policy Optimization (PPO) (Schulman et al., 2017b). This connection motivates a simple surrogate objective with a clipped probability ratio between the new generator and the old one. The probability ratio clipping discourages excessively large generator updates, and has shown to be effective in the context of stabilizing policy optimization (Schulman et al., 2017b). Figure 7.1 (left) shows the intuition about the surrogate objective, where we can observe the objective value decreases with an overly large generator change and thus imposes regularization on the updates.

(2) When updating the discriminator, the new perspective induces an importance sampling mechanism, which effectively re-weights fake samples by their discriminator scores. Since low-quality samples tend to receive smaller weights, the discriminator trained on the re-weighted samples is more likely to maintain stable performance, and in turn provide informative gradients for subsequent generator updates. Figure 7.1 (middle/right) demonstrates the effect of the re-weighting in reducing the variance of both discriminator and generator losses.

Besides, our variational GAN training framework can provably overcome the training issue (Zhou

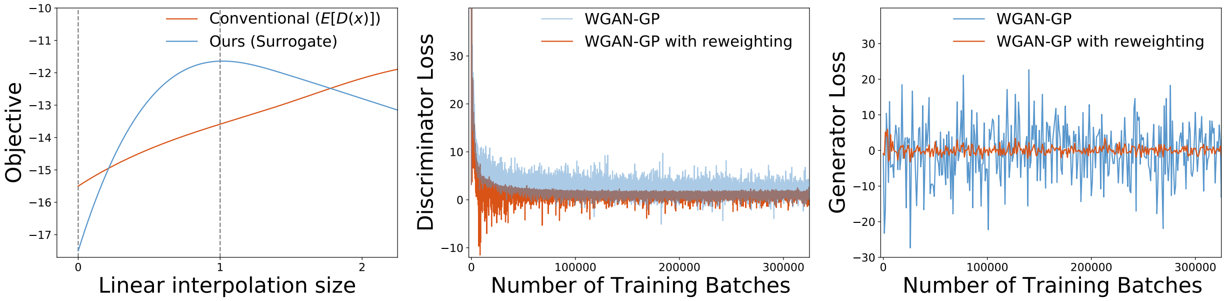


Figure 7.1: Illustration of the proposed approach for stabilizing GAN training. Results are from the CIFAR-10 experiment in Sec.7.4.1. **Left:** The conventional and surrogate objectives for generator training, as we interpolate between the initial generator parameters $\boldsymbol{\theta}_{old}$ and the updated generator parameters $\boldsymbol{\theta}_{new}$ which we compute after one iteration of training. The $\boldsymbol{\theta}_{new}$ obtains maximal surrogate objective. The surrogate objective curve starts decreasing after $x = 1$, showing the objective imposes a penalty for having too large of a generator update. In contrast, the conventional objective (for WGAN-GP) keeps increasing with larger generator updates. **Middle and right:** Discriminator and generator losses w/ and w/o sample re-weighting. WGAN-GP with our re-weighting plugged in shows lower variance in both discriminator and generator losses throughout training.

et al., 2019) that an optimal discriminator cannot provide any informative gradient to training generator. This issue usually occurs in GAN training (Zhou et al., 2019), since the discriminator often converges much faster than the generator. Empirically, we conduct extensive experiments on a wide range of tasks, including text generation, text style transfer, and image generation. Our approach shows significant improvement over state-of-the-art methods, demonstrating its broad applicability and efficacy.[1]

7.2 Related Work

Wasserstein distance, WGAN, and Lipschitz continuity. The GAN framework (Goodfellow et al., 2014) features two components: a generator G_θ that synthesizes samples $\boldsymbol{x}$ given some noise source $\boldsymbol{z}$, namely $\boldsymbol{x} = G_\theta(\boldsymbol{z})$ with $\boldsymbol{z} \sim p_z(\boldsymbol{z})$, and a discriminator that distinguishes generator's output and real data, which provides gradient feedback to improve the generator's performance. WGAN (Arjovsky et al., 2017a) improves the training stability of GANs by minimizing the Wasserstein distance $W(p_r, p_\theta)$ between the generation distribution p_θ (induced from G_θ) and the real data distribution p_r. Its training loss is formulated as:

$$\min_{\boldsymbol{\theta}} \max_{f \in \mathcal{D}} \mathbb{E}_{\boldsymbol{x} \sim p_r}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x} \sim p_\theta}[f(\boldsymbol{x})], \tag{7.2.1}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions; f acts as the discriminator and is usually implemented by a neural network f_ϕ. The original resort to enforce the Lipschitz constraint is through weight clipping (Arjovsky et al., 2017a). WGAN-GP (Gulrajani et al., 2017) later improves it by

[1] Code available at: github.com/Holmeswww/PPOGAN

replacing it with a gradient penalty on the discriminator. CT-GAN (Wei et al., 2018) further imposes the Lipschitz continuity constraint on the manifold of the real data $\boldsymbol{x} \sim p_r$. Our approach is orthogonal to these prior works and can serve as a drop-in replacement to stabilize generator and discriminator in various kinds of GANs, such as WGAN-GP and CT-GAN.

Research on the Lipschitz continuity of GAN discriminators have resulted in the theory of "informative gradients" (Zhou et al., 2019, 2018). Under certain mild conditions, a Lipschitz discriminator can provide informative gradient to the generator in a GAN framework: when p_θ and p_r are disjoint, the gradient $\nabla f^*(\boldsymbol{x})$ of optimal discriminator f^* w.r.t each sample $\boldsymbol{x} \sim p_\theta$ points to a sample $\boldsymbol{x}^* \sim p_r$, which guarantees that the generation distribution p_θ is moving towards p_r. We extend the informative gradient theory to our new case and show theoretical guarantees of our approach.

Reinforcement learning as inference. Casting RL as probabilistic inference has a long history of research (Dayan and Hinton, 1997; Deisenroth et al., 2013; Rawlik et al., 2013; Levine, 2018; Abdolmaleki et al., 2018). For example, Abdolmaleki et al. (2018) introduced maximum a-posteriori policy optimization from a variational perspective. Tan et al. (2018) connected the formulation with other paradigms of learning such as maximum likelihood estimation and data augmentation (Hu et al., 2019b). TRPO (Schulman et al., 2015a) is closely related to this line by using a KL divergence regularizer to stabilize standard RL objectives. PPO (Schulman et al., 2017b) further proposed a practical clipped surrogate objective that emulates the regularization. Our approach draws on the connections to the research, particularly the variational perspective and PPO, to improve GAN training.

Other related work. Importance re-weighting has been adopted in different problems, such as learning knowledge constraints (Hu et al., 2018a), and improving VAEs (Burda et al., 2015) and GANs (Hu et al., 2017c; Song and Ermon, 2020). We derive from the variational perspective which leads to re-weighting and clipping in the new context of GAN training stabilization. Our approach is orthogonal to and can be combined with other stabilization techniques such as large-batch training (Brock et al., 2018) and parameter averaging (EMA) (Yaz et al., 2018; Brock et al., 2018).

7.3 Improving GAN Training

7.3.1 Motivations

Our approach is motivated by connecting GAN training with the well-established RL-as-inference methods (Abdolmaleki et al., 2018; Levine, 2018; Tan et al., 2018) under a variational perspective. The connections enable us to augment GAN training with existing powerful probabilistic inference tools as well as draw inspirations from the rich RL literature for stable training. In particular, the connection to the popular TRPO (Schulman et al., 2015a) and PPO (Schulman et al., 2017b) yields the probability ratio clipping in generator training that avoids destructive updates (Sec.7.3.2), and the application of importance sampling estimation gives rise to sample re-weighting for adaptive discriminator updates (Sec.7.3.3). The full training procedure is summarized in Alg.6.

Specifically, as described in Sec.7.2, the conventional WGAN formulation for updating the generator $p_\theta(\boldsymbol{x})$ maximizes the expected discriminator score $\mathbb{E}_{p_\theta}[f_\phi(\boldsymbol{x})]$, where f_ϕ is the Lipschitz-continuous discriminator parameterized with $\boldsymbol{\phi}$. The objective straightforwardly relates to policy optimization in RL by seeing p_θ as a policy and f_ϕ as a reward function. Thus, inspired by the probabilistic inference formulations of policy optimization (Abdolmaleki et al., 2018; Hu et al., 2018a; Tan et al., 2018), here we transform the conventional objective by introducing a non-parameterized auxiliary distribution $q(\boldsymbol{x})$ and defining a new variational objective:

$$\mathcal{L}(\boldsymbol{\theta}, q) = \mathbb{E}_q[f_\phi(\boldsymbol{x})] - \mathrm{KL}\left(q(\boldsymbol{x}) \| p_\theta(\boldsymbol{x})\right), \tag{7.3.1}$$

where KL is the KL divergence. Intuitively, we are maximizing the expected discriminator score of the auxiliary q (instead of generator p_θ), and meanwhile encouraging the generator to stay close to q. We note that Hu et al. (2018a) have also related the above objective to GANs, with the different goal of integrating structured knowledge with deep generative modeling.

As we shall see in more details shortly, the new formulation allows us to take advantage of off-the-shelf inference methods, which naturally leads to new components to improve the GAN training. Maximization of the above objective is solved by the expectation maximization (EM) algorithm (Neal and Hinton, 1998) which alternatingly optimizes q at E-step and optimizes $\boldsymbol{\theta}$ at M-step. More specifically, at each iteration t, given the current status of generator parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t)}$, the E-step that maximizes $\mathcal{L}(\boldsymbol{\theta}^{(t)}, q)$ w.r.t q has a closed-form solution:

$$q^{(t)}(\boldsymbol{x}) = \frac{p_{\theta^{(t)}}(\boldsymbol{x}) \exp\{f_\phi(\boldsymbol{x})\}}{Z_\phi}, \tag{7.3.2}$$

where $Z_\phi = \int_x p_{\theta^{(t)}}(\boldsymbol{x}) \exp\{f_\phi(\boldsymbol{x})\}$ is a normalization term that depends on the discriminator parameters $\boldsymbol{\phi}$. We elaborate on the M-step in the following subsections, where we continue to develop the practical procedures for updating the generator and the discriminator, respectively.

7.3.2 Generator Training with Probability Ratio Clipping

The M-step optimizes $\mathcal{L}(\boldsymbol{\theta}, q^{(t)})$ w.r.t $\boldsymbol{\theta}$, which is equivalent to minimizing the KL divergence term in Eq.(7.3.1). However, since the generator p_θ in GANs is often an *implicit* distribution that does not permit evaluating likelihood, the above KL term (which involves evaluating the likelihood of samples from q) is not applicable. We adopt an approximation, which has also been used in the classical wake-sleep algorithm (Hinton et al., 1995) and recent work (Hu et al., 2018a), by minimizing the *reverse* KL divergence as below. With Eq.(7.3.2) plugged in, we have:

$$\min_\theta \mathrm{KL}\left(p_\theta(\boldsymbol{x}) \| q^{(t)}(\boldsymbol{x})\right) = \min_\theta -\mathbb{E}_{p_\theta}\left[f_\phi(\boldsymbol{x})\right] + \mathrm{KL}\left(p_\theta(\boldsymbol{x}) \| p_{\theta^{(t)}}(\boldsymbol{x})\right). \tag{7.3.3}$$

As proven in the appendix, this reverse KL approximation does not change the optimization problem in Eq.(7.3.1). The first term on the right-hand side of Eq.(7.3.3) recovers the conventional objective of updating the generator. Of particular interest is the second term, which is a new KL regularizer between the generator p_θ and its "old" state $p_{\theta^{(t)}}$ from the previous iteration. The regularizer discourages the generator from changing too much between updates, which is useful to stabilize the stochastic optimization procedure. The regularization closely resembles to that of TRPO/PPO, where a similar KL regularizer is imposed to prevent uncontrolled policy updates and make policy gradient robust to noises. Sec.7.3.4 gives analysis on the KL-regularized generator updates.

In practice, directly optimizing with the KL regularizer can be infeasible due to the same difficulty with the implicit distribution as above. Fortunately, PPO (Schulman et al., 2017b) has presented a simplified solution that emulates the regularized updates using a clipped surrogate objective, which is widely-used in RL. We import the solution to our context, leading to the following practical procedure of generator updates.

Probability Ratio Clipping. Let r_t denote the probability ratio $r_t(\boldsymbol{\theta}) = \frac{p_\theta(\boldsymbol{x})}{p_{\theta^{(t)}}(\boldsymbol{x})}$ which measures the difference between the new and old generator distributions. For instance, $r_t(\boldsymbol{\theta}^{(t)}) = 1$. The clipped surrogate objective for updating the generator, as adapted from PPO, is:

$$\mathcal{L}^{CLIP}(\boldsymbol{\theta}) = \mathbb{E}_{p_\theta}\left[\min\left(r_t(\boldsymbol{\theta})f_\phi(\boldsymbol{x}),\ r_t^{clip}(\boldsymbol{\theta})f_\phi(\boldsymbol{x})\right)\right], \tag{7.3.4}$$

where $r_t^{clip}(\boldsymbol{\theta}) = \text{clip}\left(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon\right)$ clips the probability ratio, so that moving $r_t(\boldsymbol{\theta})$ outside of the interval $[1-\epsilon, 1+\epsilon]$ is discouraged. Taking the minimum puts a ceiling on the increase of the objective. Thus the generator does not benefit by going far away from the old generator.

Finally, to estimate the probability ratio $r_t(\boldsymbol{\theta})$ when p_θ is implicit, we use an efficient approximation similar to (Che et al., 2017a; Grover et al., 2019) by introducing a binary classifier C trained to distinguish real and generated samples. Assuming an optimal classifier C which has $p_\theta(\boldsymbol{x}) = \frac{1-C(\boldsymbol{x})}{C(\boldsymbol{x})}p_r(\boldsymbol{x})$ (Goodfellow et al., 2014; Che et al., 2017a), we approximate r_t by:

$$r_t(\boldsymbol{\theta}) = \frac{p_\theta(\boldsymbol{x})}{p_{\theta^{(t)}}(\boldsymbol{x})} \approx \frac{(1-C(\boldsymbol{x}))\cdot C^{(t)}(\boldsymbol{x})}{(1-C^{(t)}(\boldsymbol{x}))\cdot C(\boldsymbol{x})}, \tag{7.3.5}$$

where $C^{(t)}(\boldsymbol{x})$ denotes the classifier at the t-th iteration. Note that the rightmost expression depends on $\boldsymbol{\theta}$ because $\boldsymbol{x}$ is the output of the generator, i.e., $\boldsymbol{x} = G_\theta(\boldsymbol{z})$. In practice, during the phase of generator training, we maintain C by fine-tuning it for only one iteration every time after $\boldsymbol{\theta}$ is updated (Alg.6). Thus the maintenance of C is cheap. We give more details of the configuration of C in the appendix. In the cases where an explicit generative model is used (e.g., a language model for text generation), the probability ratio r_t can directly be evaluated by definition without the need of C, though in our text generation experiments (Sec.7.4.2) we still used C for approximating r_t.

7.3.3 Discriminator Training with Sample Re-weighting

We next discuss the training of the discriminator f_ϕ, where we augment the conventional training with an importance weighting mechanism for adaptive updates. Concretely, given the form of the auxiliary distribution solution $q^{(t)}$ in Eq.(7.3.2), we first draw from the recent energy-based modeling work (Kim and Bengio, 2016; Hu et al., 2018a) and propose to optimize $\boldsymbol{\phi}$ by maximizing the data log-likelihood of $q^{(t)}$, $\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{p_r}[\log q^{(t)}(\boldsymbol{x})]$. By taking the gradient, we have:

$$\nabla_\phi \mathcal{L}(\boldsymbol{\phi}) = \nabla_\phi\Big(\mathbb{E}_{p_r}\left[f_\phi(\boldsymbol{x})\right] - \log Z_\phi\Big) = \mathbb{E}_{p_r}\left[\nabla_\phi f_\phi(\boldsymbol{x})\right] - \mathbb{E}_{q^{(t)}}\left[\nabla_\phi f_\phi(\boldsymbol{x})\right]. \tag{7.3.6}$$

We can observe that the resulting form resembles the conventional one (Eq.7.2.1) as we are essentially maximizing f_ϕ on real data while minimizing f_ϕ on fake samples. An important difference is that here fake samples are drawn from the auxiliary distribution $q^{(t)}$ instead of the generator p_θ. This difference leads to the new sample re-weighting component as below. Note that, as in WGAN (Sec.7.2), we maintain f_ϕ to be from the class of 1-Lipschitz functions, which is necessary for the convergence analysis in Sec.7.3.4. In practice, we can use gradient penalty (Gulrajani et al., 2017; Wei et al., 2018) for the Lipschitz continuity.

Sample Re-weighting. We use the tool of importance sampling to estimate the expectation under $q^{(t)}$ in Eq.(7.3.6). Given the multiplicative form of $q^{(t)}$ in Eq.(7.3.2), similar to (Abdolmaleki et al., 2018; Hu et al., 2018a; Deng et al., 2020), we use the generator $p_{\theta^{(t)}}$ as the proposal distribution. This leads to

$$\mathbb{E}_{q^{(t)}}\left[\nabla_\phi f_\phi(\boldsymbol{x})\right] = \mathbb{E}_{p_{\theta^{(t)}}}\left[\exp\{f_\phi(\boldsymbol{x})\} \cdot \nabla_\phi f_\phi(\boldsymbol{x})\right] / Z_\phi. \tag{7.3.7}$$

Note that Z_ϕ is the normalization factor defined in Eq.(7.3.2). Thus, fake samples from the generator are weighted by the exponentiated discriminator score when used to update the discriminator. Intuitively, the mechanism assigns higher weights to samples that can fool the discriminator better, while low-quality samples are downplayed to avoid destructing the discriminator performance. It is worth mentioning that similar importance weighting scheme has been used in (Hu et al., 2017c; Che et al., 2017a) for generator training in GANs, and (Burda et al., 2015) for improving variational auto-encoders. Our work instead results in a re-weighting scheme in the new context of discriminator training.

The algorithm below summarizes the proposed training procedure for the generator and discriminator.

Algorithm 6 GAN Training with Probability Ratio Clipping and Sampling Re-weighting

```
Initialize the generator $p_\theta$, the discriminator $f_\phi$, and the auxiliary binary classifier $C$
for $t \leftarrow 1$ to $T$ do
    for certain number of steps do
        Update the discriminator $f_\phi$ with sample re-weighting through Eqs.(7.3.6)-(7.3.7),
        and maintain $f_\phi$ to have upper-bounded Lipschitz constant through, e.g., gradient
        penalty (Gulrajani et al., 2017).
    end for
    for certain number of steps do
        Finetune the real/fake binary classifier $C$ (for 1 step)
        Estimate probability ratio $r_t(\boldsymbol{\theta})$ using $C$ through Eq.(7.3.5)
        Update the generator $p_\theta$ with probability ratio clipping through Eq.(7.3.4)
    end for
end for
```

7.3.4 Theoretical Analysis

To provide theoretical insight on the performance of our method, we prove that our framework holds the same guarantees as WGAN-GP (Gulrajani et al., 2017) and Lipschitz GANs (Zhou et al., 2019). Formally, we show that the method is fully compatible with *Proposition 1* in (Gulrajani et al., 2017) and *Theorem 2* in (Zhou et al., 2019), which provides rigorous analysis on GANs with Lipschitz discriminators and concludes 1) informative gradient pushes the generator distribution to the real data distribution and 2) the only Nash-equilibrium is $p_\theta = p_r$. Note that the theorems do not guarantee distributional convergence of p_θ to p_r, same as in (Gulrajani et al., 2017; Zhou et al., 2019).

Our analysis is based on the reverse KL updates for the generator (Eq.7.3.3), while the probability ratio clipping serves as a practical emulation for the updates. We begin by adapting *Proposition 1* in Gulrajani et al. (2017) to our problem:

Proposition 7.3.1 *Let p_r and q be two distributions in X, a compact metric space. Then, there is a 1-Lipschitz function f^* which is the optimal solution of*

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{x}\sim p_r}\left[f(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim q}\left[f(\boldsymbol{x})\right]$$

Let π^* be the optimal coupling between p_r and q, defined as the minimizer of: $W(p_r, q) = \inf_{\pi\in\Pi(p_r,q)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\pi}\left[\|\boldsymbol{x}-\boldsymbol{y}\|\right]$ where $\Pi(p_r, q)$ is the set of joint distributions $\pi(\boldsymbol{x}, \boldsymbol{y})$ whose marginals are p_r and q, respectively. Then, if f^* is differentiable, $\pi^*(\boldsymbol{x} = \boldsymbol{y}) = 0$, and $\boldsymbol{x}_\tau = \tau\boldsymbol{x} + (1-\tau)\boldsymbol{y}$ with $0 \leq \tau \leq 1$, it holds that $\mathbb{P}_{(\boldsymbol{x},\boldsymbol{y})\sim\pi^*}\left[\nabla f^*(\boldsymbol{x}_\tau) = \frac{\boldsymbol{y}-\boldsymbol{x}_\tau}{\|\boldsymbol{y}-\boldsymbol{x}_\tau\|}\right] = 1$.

Proposition 7.3.1 indicates that in presence of an optimal discriminator f^*, given any sample $\boldsymbol{y}$ drawn from the variational distribution q, there exists a sample $\boldsymbol{x}$ drawn from real data distribution p_r such that $\nabla_{\boldsymbol{x}} f^*(\boldsymbol{x}_\tau) = \frac{\boldsymbol{y}-\boldsymbol{x}_\tau}{\|\boldsymbol{y}-\boldsymbol{x}_t\|}$ for all linear interpolations $\boldsymbol{x}_\tau = \tau\boldsymbol{x} + (1-\tau)\boldsymbol{y}$ with $0 \leq \tau \leq 1$. Therefore, an optimal discriminator f^* can provide informative gradient to update q and push q towards to the real distribution p_r.

By the definition of q with respect to p_θ in Eq.(7.3.2), the support of p_θ and q are the same; namely, given any $\boldsymbol{x} \sim p_\theta, \boldsymbol{y} \sim p_r$, we also have $q(\boldsymbol{x}) \neq 0$. Therefore, for all $\boldsymbol{x} \sim p_\theta$, $\boldsymbol{x}$ is also a valid sample from q, the f^* in Proposition 7.3.1 provides informative gradient with respect to $\boldsymbol{x}_\tau = \tau\boldsymbol{x} + (1-\tau)\boldsymbol{y}, \forall\tau \in [0, 1]$: $\mathbb{P}_{(\boldsymbol{x},\boldsymbol{y})\sim\pi^*}\left[\nabla f^*(\boldsymbol{x}_\tau) = \frac{\boldsymbol{y}-\boldsymbol{x}_\tau}{\|\boldsymbol{y}-\boldsymbol{x}_\tau\|}\right] = 1$ Therefore, assuming f^* is the optimal discriminator to (7.3.6), optimizing Eq.(7.3.3) can provide informative gradient that points the generator p_θ toward p_r.

7.4 Experiments

We conduct extensive experiments on three unsupervised generation tasks, including image generation, text generation, and text style transfer. The three tasks apply GANs to model different data modalities, namely, image, text, and neural hidden representations, respectively. Our approach consistently offers improvement over the state-of-the-arts on all tasks. See appendix for all experimental details.

Method	**IS** (↑)	**FID** (↓)
Real data	11.24±.12	7.8
WGAN-GP (2017)	7.86±.08	-
CT-GAN (2018)	8.12±.12	-
SN-GANs (2018)	8.22±.05	21.7±.21
WGAN-ALP (2020)	8.34±.06	12.96±.35
SRNGAN (2020)	8.53 ±.04	19.83
Ours (re-weighting only)	8.45±.14	13.21±.60
Ours (full)	**8.69±.13**	**10.70±.10**

Table 7.1: CIFAR-10 results. Our method is run 3 times for average and standard deviation.

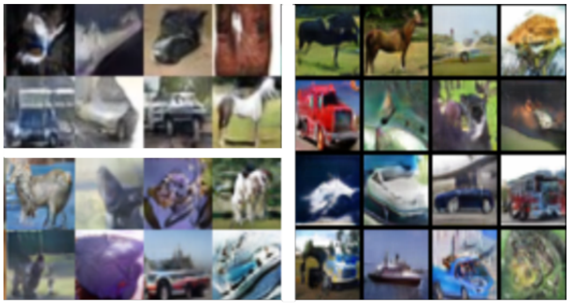


Figure 7.2: Generated samples by WGAN-GP (top-left), CT-GAN (bottom-left), and ours (right).

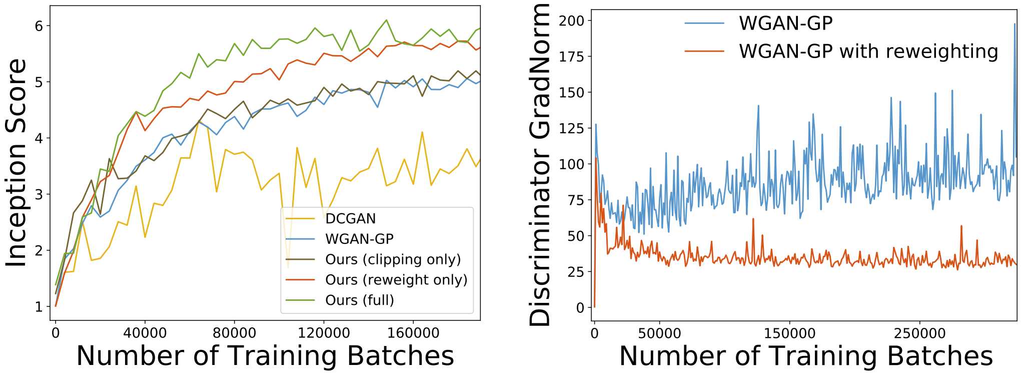


Figure 7.3: Left: Inception score on CIFAR-10 v.s. training batches (including both generator and discriminator batches). The DCGAN (Radford et al., 2015a) architecture is used. **Right:** The gradient norms of discriminators on fake samples.

7.4.1 Image Generation

We first use the popular CIFAR-10 benchmark for evaluation and in-depth analysis of our approach.

Setup. CIFAR-10 (Krizhevsky and Hinton, 2010) contains 50K images of sizes 32×32. Following the setup in CT-GAN (Wei et al., 2018), we use a residual architecture to implement both generator and discriminator, and also impose a Lipschitz constraint on the discriminator. For each iteration, we update both generator and discriminator for 5 times. We use Inception Score (IS) (Salimans et al., 2016) for evaluating generation quality and diversity, and Frechet Inception Distance (FID) (Heusel et al., 2017) for capturing model issues, e.g., mode collapse (Xu et al., 2018).

Results. Table 7.1 reports the results on CIFAR-10. For the three latest methods, SN-GANs (Miyato et al., 2018) introduced spectral normalization to stabilize the discriminator training; WGAN-ALP (Terjék, 2020) developed an explicit Lipschitz penalty; and SRNGAN (Sanyal et al., 2020) introduced a weight-normalization scheme for generalization. Table 7.1 shows that our full approach (CT-GAN + discriminator sample re-weighting + generator probability ratio clipping)

Length	MLE	SeqGAN	LeakGAN	RelGAN	WGAN-GP	Ours	Real
20	9.038	8.736	7.038	6.680	6.89	**5.67**	5.750
40	10.411	10.310	7.191	6.765	6.78	**6.14**	4.071

Table 7.2: Oracle negative log-likelihood scores (↓) on synthetic data, in comparison with SeqGAN (Yu et al., 2017b), LeakGAN (Guo et al., 2018), RelGAN (Nie et al., 2018), and WGAN-GP (Gulrajani et al., 2017).

achieves the best, with both IS and FID significantly surpassing the baselines. These results accord with the visual results in Figure 7.2 where our generated samples show higher visual quality than those of the baselines. Comparison between CT-GAN and our approach with only re-weighting shows significant improvement. By further adding the probability ratio clipping to arrive our full approach, the performance (both IS and FID) is further improved with a large margin. The results demonstrate the effectiveness of the two components in our approach.

Figure 7.1 in Sec.7.1 has shown the effects of the proposed approach in stabilizing the generator and discriminator training. Here we further analyze these two components. Figure 7.3 (left) shows the convergence curves of different GAN methods. For a fair comparison, all models use the same DCGAN architecture (Radford et al., 2015a), and both our approach and WGAN-GP (Gulrajani et al., 2017) enforce the same discriminator Lipschitz constraint. Following the optimal setup in (Gulrajani et al., 2017), the update ratio of both WGAN-GP and our "re-weighting only" is 5:1 (i.e., each iteration updates the discriminator for 5 times and the generator for one time). Our full approach and "clipping only" use an update ratio of 5:5, because the probability ratio clipping that discourages large generator updates allows us to update the generator more frequently, which is desirable. Note that the x-axis in Figure 7.3 accounts for both generator and discriminator batches (i.e., an 5:5 iteration is counted as 10 training batches). Thus, for any given point on the x-axis, all comparison methods used roughly the same amount of computation. From the curves, one can observe that our full approach surpasses our approach with only sample re-weighting, and they both converge faster and achieve a higher IS score than "clipping only", WGAN-GP, and DCGAN. It is interesting to note that "clipping only" does not offer a performance improvement over WGAN-GP, though its combination with sample re-weighting (i.e., the full approach) does improve over "re-weighting only". This is indeed not unexpected, because clipping and re-weighting are derived from the variational framework (Eq.7.3.1) in a principled way. Discarding either of the two could lead to improper handling of the variational distribution q and fails to conform to the framework.

Figure 7.3 (right) investigates how the fake sample re-weighting can affect the discriminator training. By injecting re-weighting into WGAN-GP, the gradients on fake samples become more stable with lower variance, which partially explains the better training stability of discriminator in Figure 7.1.

Method	**BLEU-2** (↑)	**BLEU-3** (↑)	**BLEU-4** (↑)	**BLEU-5** (↑)	**NLL**$_{gen}$ (↓)	**Human** (↑)
MLE	0.768	0.473	0.240	0.126	2.382	-
LeakGAN	0.826	0.645	0.437	0.272	2.356	-
RelGAN 100	0.881	**0.705**	**0.501**	0.319	2.482	-
RelGAN 1000	0.837	0.654	0.435	0.265	2.285	3.42±1.23
WGAN-GP	0.872	0.636	0.379	0.220	**2.209**	-
Ours	**0.905**	0.692	0.470	**0.322**	2.265	**3.59 ± 1.12**

Table 7.3: Results on EMNLP2017 WMT News. BLEU measures text quality and NLL$_{gen}$ evaluates sample diversity. Results of previous text GAN models are from (Nie et al., 2018), where RelGAN (100) and RelGAN (1000) use different hyper-parameter for gumbel-softmax. Our approach uses the same gumbel-softmax hyper-parameter as RelGAN (1000).

7.4.2 Text Generation

In this section, we evaluate our approach on text generation, a task that is known to be notoriously difficult for GANs due to the discrete and sequential nature of text.

Setup. We implement our approach based on the RelGAN (Nie et al., 2018) architecture, a state-of-the-art GAN model for text generation. Specifically, we replace the generator and discriminator objectives in RelGAN with ours. We follow WGAN-GP (Gulrajani et al., 2017) and impose discriminator Lipschitz constraint with gradient penalty. Same as (Nie et al., 2018), we use Gumbel-softmax approximation (Jang et al., 2016; Maddison et al., 2017) on the discrete text to enable gradient backpropagation, and the generator is initialized with maximum likelihood (MLE) pre-training. Same as previous studies, we evaluate on both synthetic and real text datasets.

Results on Synthetic Data. The synthetic data consists of 10K discrete sequences generated by an oracle-LSTM with fixed parameters (Yu et al., 2017b). This setup facilitates evaluation, as the quality of generated samples can be directly measured by the negative log-likelihood (NLL) of the oracle on the samples. We use synthetic data with sequence lengths 20 and 40, respectively. Table 7.2 reports the results. MLE is the baseline with maximum likelihood training, whose output model is used to initialize the generators of GANs. Besides the previous text generation GANs (Yu et al., 2017b; Guo et al., 2018; Nie et al., 2018), we also compare with WGAN-GP which uses the same neural architecture as RelGAN and ours. From Table 7.2, one can observe that our approach significantly outperforms all other approaches on both synthetic sets. Our improvement over RelGAN and WGAN-GP demonstrates that our proposed generator and discriminator objectives are more effective than the previous ones.

Results on Real Data. We then evaluate our method on the EMNLP2017 WMT News, a large real text data used for text GAN studies (Guo et al., 2018; Nie et al., 2018). The dataset consists of 270K/10K training/test sentences with a maximum length of 51 and a vocabulary size of 5,255. To measure the generation quality, we use the popular BLEU-n metric which measures n-gram overlap between generated and real text ($n \in \{2, 3, 4, 5\}$). To evaluate the diversity of generation, we use the negative log-likelihood of the generator on the real test set (NLL$_{gen}$) (Guo

Method	BLEU
Zhang et al. (2018)	24.48
Tian et al. (2018)	24.90
Subramanian et al. (2018)	31.20
Tikhonov et al. (2019)	32.82
Ours	**33.45±.95**

Table 7.4: BLEU scores between model generations and human-written text on the Yelp data. We run our method for 5 times and report the average and standard deviation.

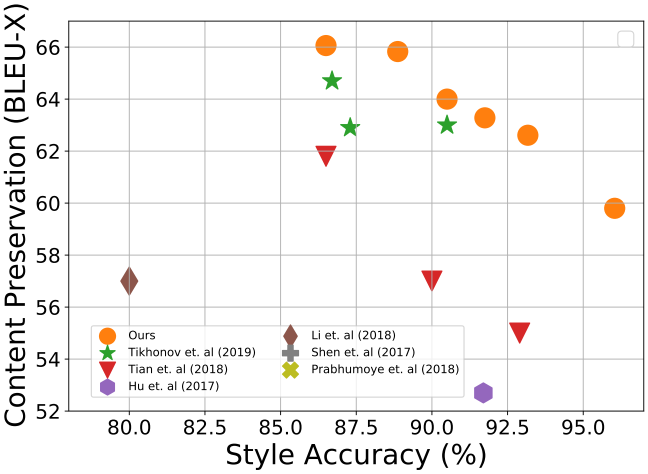


Figure 7.4: Trade-off between style accuracy and content preservation. The orange circles denote our results using varying values for an objective weight (Tikhonov et al., 2019) which manages the trade-off.

et al., 2018; Nie et al., 2018). From the results in Table 7.3, one can see that our approach shows comparable performance with the previous best model RelGAN (100) in terms of text quality (BLEU), but has better sample diversity. Our model also achieves much higher BLEU scores than WGAN-GP. We perform *human* evaluation, with randomly sampled 50 sentences for RelGAN (1000) against ours and asked 5 annotators to score each sentence on a scale of 1-5. We use the same questions as designed by (Nie et al., 2018). Ours obtained an average human score of 3.59 ± 1.12, higher than 3.42 ± 1.23 by RelGAN (Fleiss' Kappa score 0.61 showing substantial inter-rater agreement).

7.4.3 Text Style Transfer

Text style transfer task is gaining increasing attention in NLP (Hu et al., 2017b; Shen et al., 2017; Yang et al., 2018). The task aims at rewriting a sentence to modify its style (e.g., sentiment) while preserving the content. Previous work applies GANs on neural hidden states to learn disentangled representations (Shen et al., 2017; Tikhonov et al., 2019). The task thus can serve as a good benchmark for GANs, as hidden state modeling provides a new modality that differs from image and text modeling as studied above.

Setup. We follow the same experimental setting and use the same model architecture in the latest work (Tikhonov et al., 2019). In particular, the VAE-based model (Hu et al., 2017b; Kingma and Welling, 2013) is extended by adding a latent code discriminator which eliminates stylistic information in the latent code. We replace their adversarial objectives with our proposed ones, and impose discriminator Lipschitz constraint with gradient penalty (Gulrajani et al., 2017). We test on sentiment transfer, in which the sentiment (positive/negative) is treated as the text style. We use the standard Yelp review dataset, and the ground truth output text provided by (Li et al., 2018b).

Results. Following the previous work (Tikhonov et al., 2019), we first report the BLEU score that measures the similarity of the generated samples against the human written text. Table 7.4 shows that our approach achieves best performance, improving the state-of-the-art result (Tikhonov et al., 2019) from BLEU 32.82 to 33.45.

The second widely used evaluation method is to measure (1) the style accuracy by applying a pre-trained style classifier on generated text, and (2) the content preservation by computing the BLEU score between the generated text and the original input text (BLEU-X). There is often a trade-off between the two metrics. Figure 7.4 displays the trade-off by different models. Our results locate on the top-right corner, indicating that our approach achieves the best overall style-content trade-off.

7.5 Conclusion

We have presented a new training framework of GANs derived from a variational perspective based on the standardized formalism, and draws on rich connections with RL-as-inference and TRPO/PPO. This results in probability ratio clipping for generator updates to discourage overly large changes, and fake sample re-weighting for stabilized discriminator updates. Experiments show our approach demonstrates superior training stability and improves over previous best methods on image generation, text generation, and text style transfer.

Part IV

Tooling: Operationalizing Standardized Composable ML

Chapter 8

Texar: A Modularized, Versatile, and Extensible Tool for Text Generation and Machine Learning

This chapter introduces Texar, an open-source toolkit aiming to support fast and mechanic composition of ML solutions from first principle, by implementing and operationalizing the standardized ML viewpoint developed in the previous chapters of the thesis.

With the design goals of modularity, versatility, and extensibility in mind, Texar extracts common patterns underlying the diverse tasks and methodologies, creates a library of standardized and highly reusable modules, and allows arbitrary model architectures and algorithmic paradigms. In Texar, model architecture, inference, and learning processes are properly decomposed. Modules at a high concept level can be freely assembled and plugged in/swapped out. The toolkit also supports a rich set of large-scale pretrained models. Texar is thus particularly suitable for researchers and practitioners to do fast prototyping and experimentation. The versatile toolkit also fosters technique exchange across different machine learning tasks, as exemplified in Chapters 5-7 in the algorithmic perspective.

The toolkit originally focused on the particular broad set of text generation tasks that transform any inputs into natural language (e.g., machine translation, summarization, dialog, content manipulation, etc; see also Chapter 4). Since released, it has continuously been expanded with more building blocks in support of a wider range of machine learning problems. In the remainder of the chapter, we describe Texar in aspects of core design principles, architecture, and main features, with a proper focus on the text generation area. Texar supports both TensorFlow and PyTorch, and is released under Apache License 2.0 at `https://github.com/asyml`.

8.1 Introduction

Text generation spans a broad set of natural language processing tasks that aim to generate natural language from input data or machine representations. Such tasks include machine translation (Brown et al., 1990; Vaswani et al., 2017), dialog systems (Williams and Young, 2007; Serban et al., 2016; Tang et al., 2019), text summarization (Hovy and Lin, 1998; See et al., 2017), data description (Wiseman et al., 2017; Li et al., 2018a), text paraphrasing and manipulation (Hu et al., 2017a; Madnani and Dorr, 2010; Lin et al., 2019), image captioning (Vinyals et al., 2015b; Karpathy and Fei-Fei, 2015), and more. Recent years have seen rapid progress of this active area, in part due to the integration of modern deep learning approaches in many of the tasks. On the other hand, considerable research efforts are still needed in order to improve techniques and enable real-world applications.

A few remarkable open-source toolkits have been developed in support of text generation applications (sec 8.4). Those toolkits, however, are largely designed for one or a small number of specific tasks, particularly machine translation (e.g., Britz et al., 2017; Klein et al., 2017) and conversation systems (e.g., Miller et al., 2017), and usually support a narrow set of machine learning algorithms such as supervised learning. Emerging new applications and approaches instead are often developed by individual teams in a more ad-hoc manner, which can easily result in hard-to-maintain custom code and duplicated efforts across the disjoint projects.

The variety of text generation tasks indeed have many common properties. For instance, two central goals are shared across most of them, namely 1) generating well-formed, grammatical, and readable text, and 2) realizing in the generated text all desired information inferred from the inputs. To this end, a set of key techniques are increasingly widely-used, such as neural encoder-decoders (Sutskever et al., 2014), attention (Bahdanau et al., 2014; Luong et al., 2015; Vaswani et al., 2017), memory networks (Sukhbaatar et al., 2015), adversarial methods (Goodfellow et al., 2014; Lamb et al., 2016), reinforcement learning (Ranzato et al., 2015; Bahdanau et al., 2016; Tan et al., 2018), structured supervision (Hu et al., 2018a; Yang et al., 2018), as well as optimization techniques, data pre-processing and result post-processing procedures, evaluations, etc. These techniques are often combined together in various ways to tackle different problems. Figures 8.1 and 8.2 summarize examples of various model architectures and learning paradigms, respectively.

It is therefore highly desirable to have an open-source platform that unifies the development of the diverse yet closely-related applications, backed with clean and consistent implementations of the core algorithms. Such a platform would enable reuse of common components and functionalities; standardize design, implementation, and experimentation; foster reproducibility; and, importantly, encourage technique sharing among different text generation tasks so that an algorithmic advance developed for a specific task can quickly be evaluated and generalized to many other tasks.

We introduce *Texar*, a general-purpose text generation and machine learning toolkit aiming to support popular and emerging applications in the field, by providing researchers and practitioners a unified and flexible framework for building their models. Texar has two versions, building upon TensorFlow (`tensorflow.org`) and PyTorch (`pytorch.org`), respectively, with the same uniform design.

Underlying the core of Texar's design is principled anatomy of extensive machine learning models and algorithms (Hu et al., 2017c; Tan et al., 2018), which subsumes the diverse cases in Fig-

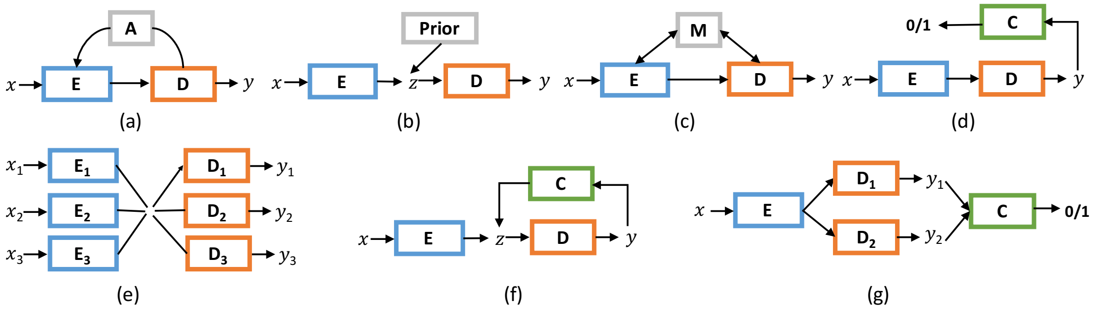


Figure 8.1: An example of various model architectures in recent text generation literatures. E denotes encoder, D denotes decoder, C denotes classifier (i.e., binary discriminator). **(a):** The canonical encoder-decoder, sometimes with attention A (Sutskever et al., 2014; Bahdanau et al., 2014; Luong et al., 2015; Vaswani et al., 2017) or copy mechanisms (Gu et al., 2016; Vinyals et al., 2015a; Gulcehre et al., 2016). **(b):** Variational encoder-decoder (Bowman et al., 2015; Yang et al., 2017b). **(c):** Encoder-decoder augmented with external memory (Sukhbaatar et al., 2015; Bordes et al., 2016). **(d):** Adversarial model using a binary discriminator C, with or without reinforcement learning (Liang et al., 2017; Zhang et al., 2017; Yu et al., 2017b). **(e):** Multi-task learning with multiple encoders and/or decoders (Luong et al., 2016; Firat et al., 2016). **(f):** Augmenting with cyclic loss (Hu et al., 2017a). **(g):** Learning to align with adversary, either on samples y or hidden states (Lamb et al., 2016; Lample et al., 2017; Shen et al., 2017).

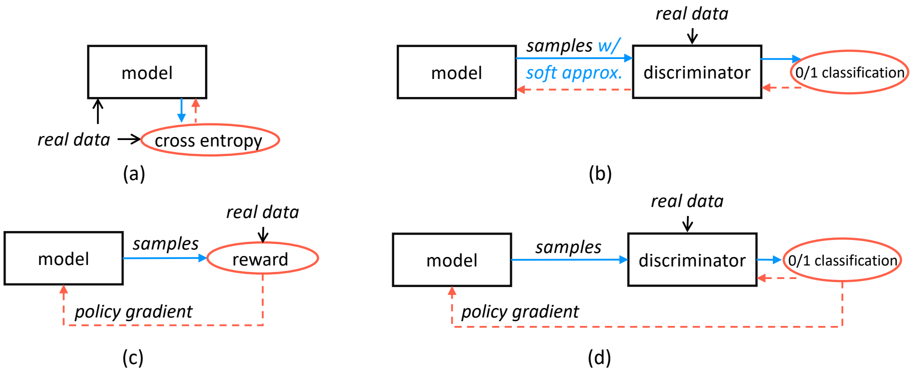


Figure 8.2: An example of various learning paradigms for text generation models. Inference processes are denoted as blue solid arrows; learning signals and gradient propagation are denoted as red dashed arrows. **(a):** Maximum likelihood learning with a cross-entropy loss (Mikolov et al., 2010). **(b):** Adversarial learning that propagates gradient through samples with continuous approximation (*soft approx.*) (Hu et al., 2017a; Yang et al., 2018). A discriminator is trained jointly with the target model. **(c):** Reinforcement learning with a specified reward function (e.g., BLEU score) and policy gradient (Ranzato et al., 2015; Rennie et al., 2017). **(d):** Combination of adversarial learning and reinforcement learning by using a learnable discriminator as the reward function and updating the target model with policy gradient (Fedus et al., 2018; Yu et al., 2017b). Other learning algorithms can include reward-augmented maximum likelihood (Norouzi et al., 2016), learning-to-search (Hal Daumé et al., 2009; Wiseman and Rush, 2016), interpolation algorithm (Tan et al., 2018), etc.

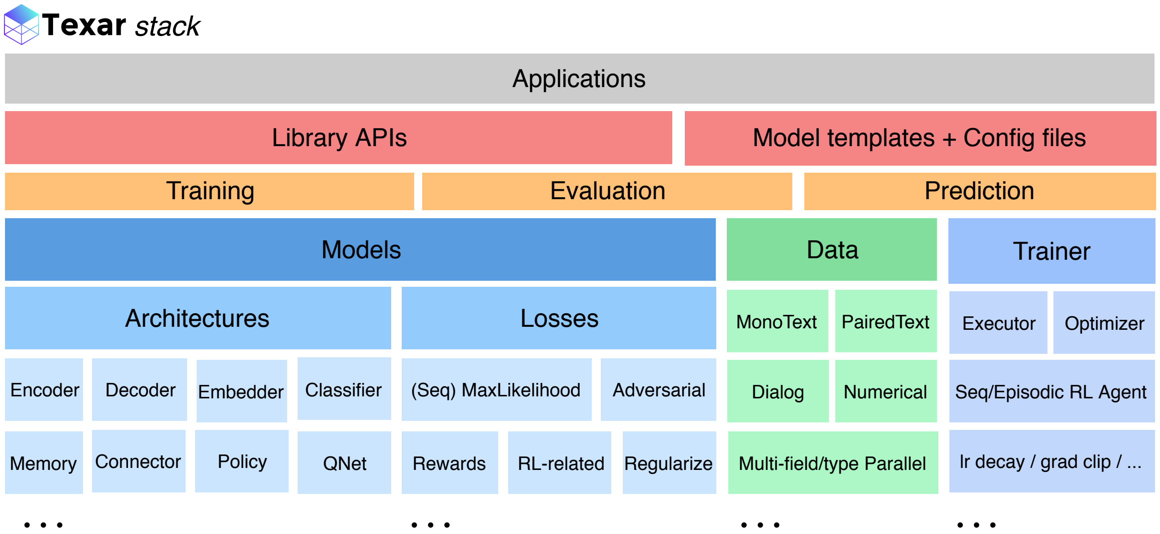


Figure 8.3: The stack of main modules and functionalities in Texar. Many of the low-level components are omitted.

ures 8.1, 8.2 and beyond, enabling a unified formulation and consistent implementation. Texar emphasizes three key properties:

- **Versatility**: Texar contains a wide range of features and functionalities for 1) arbitrary model architectures as a combination of encoders, decoders, embedders, discriminators, memories, and many other modules; and 2) different modeling and learning paradigms such as sequence-to-sequence, probabilistic models, adversarial methods, and reinforcement learning. Based on these, both workhorse and cutting-edge solutions to the broad spectrum of text generation tasks are either already included or can be easily constructed.

- **Modularity**: Texar is designed to be highly modularized, by decoupling solutions of diverse tasks into a set of highly reusable modules. Users can construct their model at a high conceptual level just like assembling building blocks. It is convenient to plug in or swap out modules, configure rich options of each module, or even switch between distinct modeling paradigms. For example, switching between maximum likelihood learning and reinforcement learning involves only minimal code changes (e.g., Figure 8.7). Modularity makes Texar particularly suitable for fast prototyping and experimentation.

- **Extensibility**: The toolkit provides interfaces of multiple functionality levels, ranging from simple configuration files to full library APIs. Users of different needs and expertise are free to choose different interfaces for appropriate programmability and internal accessibility. The library APIs are fully compatible with the native TensorFlow/PyTorch interfaces, which allows seamless integration of user-customized modules, and enables the toolkit to take advantage of the vibrant open-source community by effortlessly importing any external components as needed.

Furthermore, Texar puts much emphasis on well-structured high-quality code of uniform design patterns and consistent styles, along with clean documentations and rich tutorial examples. Other useful features such as distributed GPU training are also supported.

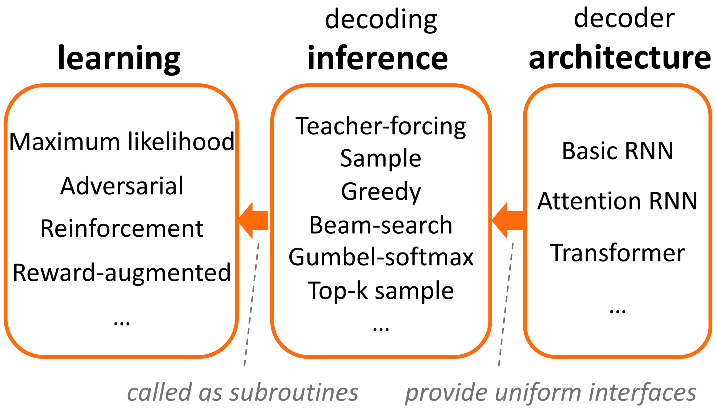


Figure 8.4: The learning–inference–architecture anatomy in Texar, taking decoder for example. Specifically, a sequence decoder in a model can have an arbitrary architecture (e.g., basic RNN decoder); all architectures expose uniform interfaces for specifying one of the tens of decoding strategies (e.g., teacher-forcing decoding) to generate samples or infer probabilities; a learning procedure repeated calls arbitrary specified inference procedure during training. Learning can be totally agnostic to the model architecture.

8.2 Structure and Design

Figure 8.3 shows the stack of main modules and functionalities in Texar. Building upon the lower level deep learning platforms, Texar provides a comprehensive set of building blocks for model construction, training, evaluation, and prediction. In the following, we first present the design principles that lead to the attainment of these goals (sec 8.2.1), and then describe the detailed structure of Texar with running examples to demonstrate the properties of the toolkit (sec 8.2.2-8.2.4).

8.2.1 The Design of Texar

The broad variation of the many text generation tasks and the fast-growing new models and algorithms have posed unique challenges to designing a versatile toolkit. We tackle the challenges by principally decomposing the modeling and experimentation pipeline, developing an extensive set of ready-to-assemble modules, and providing user interfaces of varying abstract levels.

Unified Anatomy. We break down the complexity of the rich text generation tasks into three dimensions of variations, namely, the varying data types and formats for different use cases, the arbitrary combinational model architectures and associated inference procedures (e.g., Figure 8.1), and the distinct learning algorithms such as maximum likelihood learning, reinforcement learning, and combinations thereof (e.g., Figure 8.2). Within the unified abstraction, all learning paradigms are each specifying one or multiple loss functions (e.g., cross-entropy loss, policy gradient loss), along with an optimization procedure that improves the losses:

$$\min_\theta \mathcal{L}(p_\theta, D) \tag{8.2.1}$$

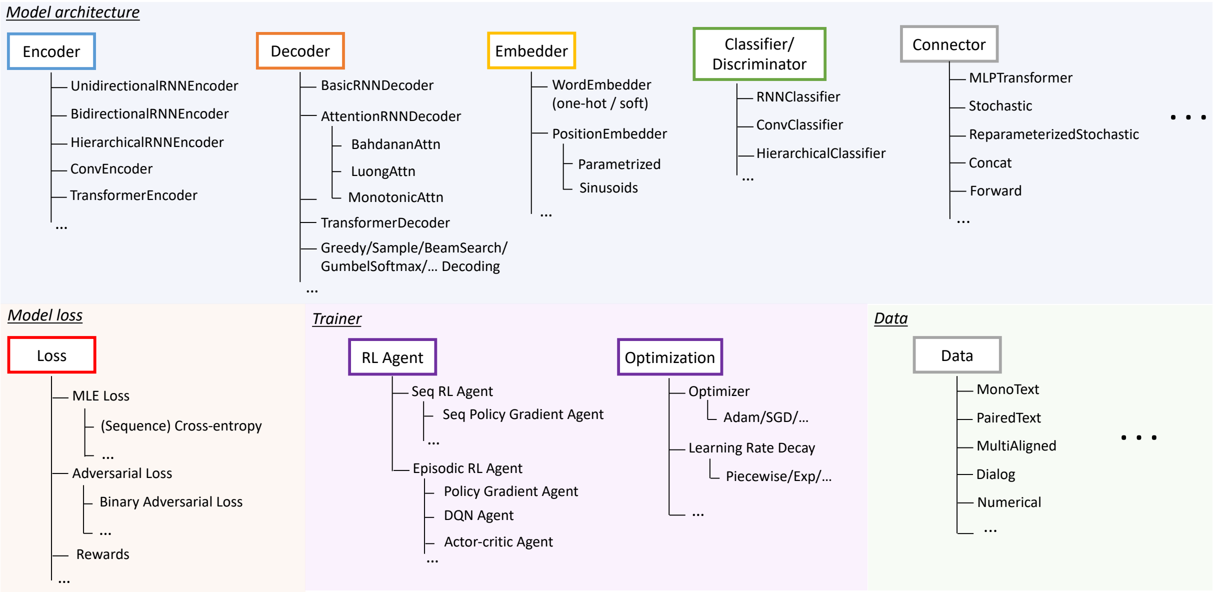


Figure 8.5: The catalog of a subset of modules for model construction and learning. Other modules, such as memory network modules, and those for evaluation and prediction, are omitted due to space limitations.

where $\boldsymbol{p}_\theta$ is the model that defines the model architecture and the inference procedure; $\boldsymbol{D}$ is the data; $\boldsymbol{\mathcal{L}}$ is the learning objectives (losses); and $\mathbf{min}$ denotes the optimization procedure. Note that the above can have multiple losses imposed on different model parts (e.g., adversarial learning).

Further, as illustrated in Figure 8.4, we decouple learning, inference, and model architecture to the maximum extent, forming abstraction layers of **learning – inference – architecture**, as illustrated in Figure 8.4. That is, different architectures implement the same set of inference procedures and provide the same interfaces, so that learning algorithms can call proper inference procedures as subroutines while staying agnostic to the underlying architecture and implementation details. For example, maximum likelihood learning uses teacher-forcing decoding (Mikolov et al., 2010); a policy gradient algorithm can invoke stochastic or greedy decoding (Ranzato et al., 2015); and adversarial learning can use either stochastic decoding for policy gradient-based updates (Yu et al., 2017b) or Gumbel-softmax reparameterized decoding (Jang et al., 2016) for direct gradient back-propagation. Users can effortlessly switch between different learning algorithms for the same model, by simply specifying the corresponding inference strategy and plugging into the new learning module, without adapting the model architecture (see section 8.2.3 for a running example).

The unified anatomy has underlay the strong modularity of the toolkit. It helps maximize the opportunities for reuse, enable free combinations of different parts, and greatly improve the cleanness of code structure.

Module Assembly. The fast evolution of modeling and learning methodologies has led to sophisticated models that go beyond the canonical (attentional) sequence-to-sequence alike forms

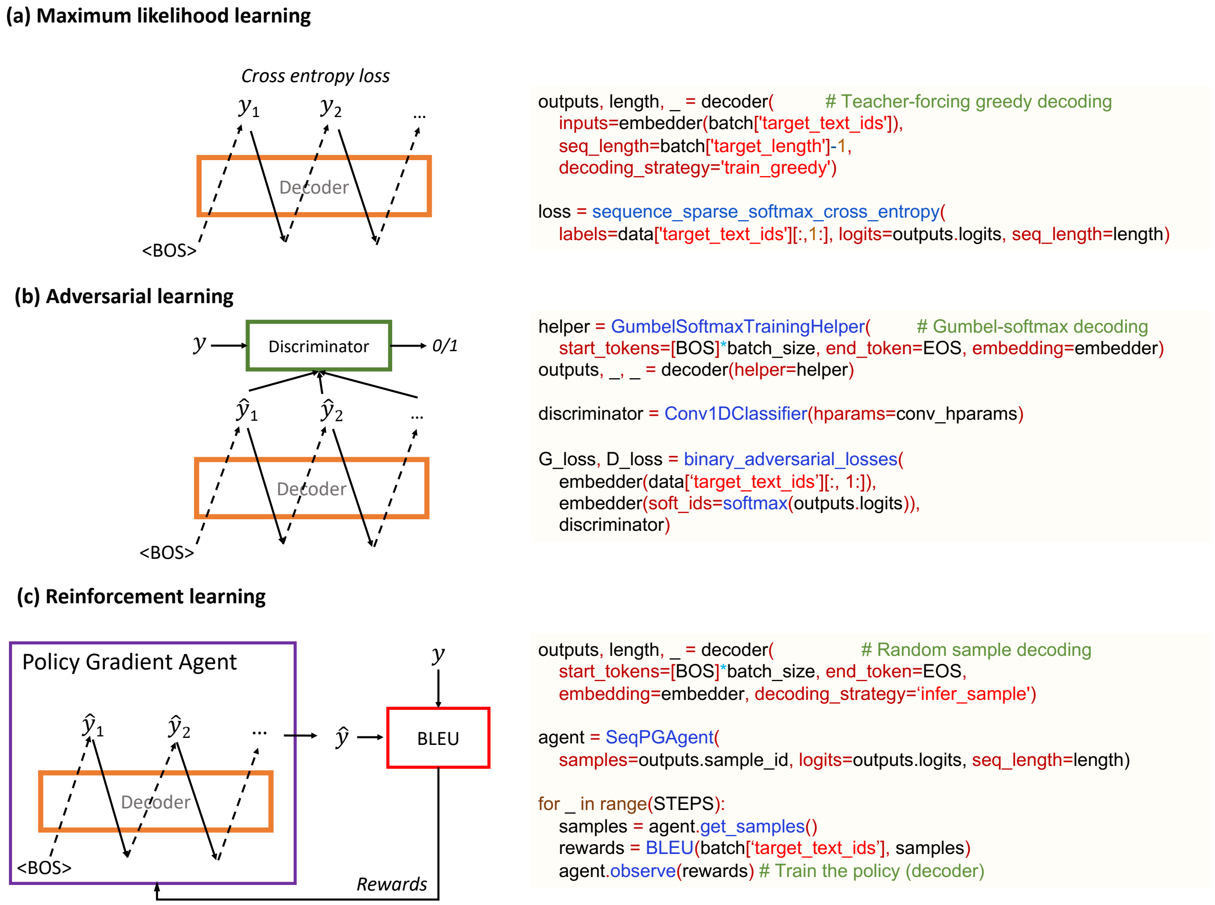
```yaml
source_embedder: WordEmbedder
source_embedder_hparams:
  dim: 300
encoder: UnidirectionalRNNEncoder
encoder_hparams:
  rnn_cell:
    type: BasicLSTMCell
    kwargs:
      num_units: 300
    num_layers: 1
    dropout:
      output_dropout: 0.5
      variational_recurrent: True
embedder_share: True
decoder: AttentionRNNDecoder
decoder_hparams:
  attention:
    type: LuongAttention
beam_search_width: 5
optimization: ...
```

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()

# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])

# Decode
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
outputs, length, _ = decoder(inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)

# Loss
loss = sequence_sparse_softmax_cross_entropy(
  labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)

```

Figure 8.6: Two ways of specifying an attentional sequence-to-sequence model. **Left:** Snippet of an example YAML configuration file of the sequence-to-sequence model template. Only those hyperparameters that the user concerns are specified explicitly in the particular file, while the remaining many hyperparameters can be omitted and will take default values. **Right:** Python code assembling the sequence-to-sequence model using the Texar library APIs. Modules are created as Python objects, and then called as functions to add TensorFlow ops to the computation graph and return output tensors. Other code such as optimization is omitted.

and introduce many new composite architectures. (Figure 8.1 summarizes several model architectures developed in recent literature for different tasks.) To versatilely support all these diverse approaches, we break down the complex models and extract a set of frequently-used modules (e.g., `encoders`, `decoders`, `embedders`, `classifiers`, etc). Figure 8.5 shows the catelog of a subset of modules. Crucially, Texar allows free concatenation between these modules in order to assemble arbitrary model architectures. Such concatenation can be done by directly interfacing two modules, or through an intermediate `connector` module that provides general functionalities of shape transformation, reparameterization (e.g., Kingma and Welling, 2013; Jang et al., 2016), sampling, and others.

User Interfaces. It is crucial for the toolkit to be flexible enough to allow construction of simple and advanced models, while at the same time providing proper abstractions to relieve users from overly concerning about low-level implementations. In particular, Texar provides two major types of user interfaces: 1) YAML configuration files that instantiate pre-defined model templates, and 2) full Python library APIs. The former is simple, clean, straightforwardly understandable for non-expert users, and is also adopted by other toolkits (Britz et al., 2017; Klein et al., 2017), while the latter allows maximal flexibility, full access to internal states, and essentially unlimited customizability. The libray APIs also provide interfaces at different abstract levels for key functionalities, allowing users to select and trade off between readily usability and



(a) Maximum likelihood learning

```
outputs, length, _ = decoder(        # Teacher-forcing greedy decoding
    inputs=embedder(batch['target_text_ids']),
    seq_length=batch['target_length']-1,
    decoding_strategy='train_greedy')

loss = sequence_sparse_softmax_cross_entropy(
    labels=data['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

(b) Adversarial learning

```
helper = GumbelSoftmaxTrainingHelper(        # Gumbel-softmax decoding
    start_tokens=[BOS]*batch_size, end_token=EOS, embedding=embedder)
outputs, _, _ = decoder(helper=helper)

discriminator = Conv1DClassifier(hparams=conv_hparams)

G_loss, D_loss = binary_adversarial_losses(
    embedder(data['target_text_ids'][:, 1:]),
    embedder(soft_ids=softmax(outputs.logits)),
    discriminator)
```

(c) Reinforcement learning

```
outputs, length, _ = decoder(            # Random sample decoding
    start_tokens=[BOS]*batch_size, end_token=EOS,
    embedding=embedder, decoding_strategy='infer_sample')

agent = SeqPGAgent(
    samples=outputs.sample_id, logits=outputs.logits, seq_length=length)

for _ in range(STEPS):
    samples = agent.get_samples()
    rewards = BLEU(batch['target_text_ids'], samples)
    agent.observe(rewards) # Train the policy (decoder)
```

Figure 8.7: Switching between different learning paradigms of a decoder involves only modification of Line.14-19 in the right panel of Figure 8.6. In particular, the same decoder is called with different decoding strategies, and discriminator or reinforcement learning agent is added as needed with simple API calls. **(Left):** Module structure of each paradigm; **(Right):** The respective code snippets. For adversarial learning in (b), continuous Gumbel-softmax approximation (Jang et al., 2016) to the generated samples (with `GumbelSoftmaxTrainingHelper`) is used to enable gradient propagation from the discriminator to the decoder.

customizability. Examples are provided in the following sections.

8.2.2 Assemble Arbitrary Model Architectures

Figure 8.6 shows an example of specifying an attentional sequence-to-sequence model through either a YAML configuration file (left panel), or concise Python code (right panel), respectively.

- The configuration file passes hyperparameters to the model template which instantiates the model for subsequent training and evaluation (which are also configured through YAML). Text highlighted in blue in the figure specifies the names of modules to use. Module hyperparameters are specified under `*_hparams` in the configuration hierarchy (for example, `source_embedder_hparams` for the `source_embedder` module). Note that most of the hyperparameters have sensible default values, and users only have to specify a small subset of them. Hyperparameters taking default values can be omitted in the configuration file.

- The library APIs enable users to efficiently build any desired pipelines at a high conceptual level, without worrying too much about the low-level implementations. Power users are also given the option to access the full internal states for native programming and low-level manipulations.

  Texar modules have multiple features for ease of use, including 1) *Convenient variable re-use*: Each module instance creates its own sets of variables, and automatically re-uses its variables on subsequent calls. Hence TensorFlow *variable scope* is transparent to users; 2) *Configurable through hyperparameters*: Each module defines allowed hyperparameters and default values. Hyperparameter values are configured by passing the `hparams` argument to the module constructor, which precisely corresponds to the `*_hparams` sections in YAML configuration files; 3) *PyTorch-alike function calls*: As in Figure 8.6, after a module is created as an object, it can be called as a function which performs the module logic on input tensors and returns output tensors. Both the Texar TensorFlow and PyTorch versions have the same interfaces.

8.2.3 Plug-in and Swap-out Modules

Texar builds a shared abstraction of the broad set of text generation tasks. It is convenient to switch between different application contexts, or change from one modeling paradigm to another, by simply plugging in/swapping out a single or few modules, or even merely changing a configuration parameter, while keeping all other parts of the modeling pipeline unchanged.

For example, given the base code of the sequence-to-sequence model in Figure 8.6 (right panel), Figure 8.7 illustrates how Texar can easily support switching between different learning algorithms, by changing only the relevant code snippet (i.e., Line.14–19 in Figure 8.6 right panel). In particular, Figure 8.7 shows three major learning paradigms, including maximum-likelihood based supervised learning, adversarial learning, and reinforcement learning, each of which invokes different decoding (inference) methods of the decoder, namely teacher-forcing decoding, Gumbel-softmax decoding, and random-sample decoding, respectively.

The convenient module switch can be useful for fast exploration of different algorithms for a specific task, or quick experimentation of an algorithm's generalization on different tasks.

8.2.4 Customize with Extensible Interfaces

Texar emphasizes heavily on extensibility, and allows easy addition of customized or external modules through various interfaces, without editing the Texar codebase.

With the YAML configuration file, users can directly insert their own modules by providing the Python importing path to the module. For example, to use an externally implemented RNN cell in the sequence-to-sequence model encoder, one can simply change Lines.6-9 in the left panel of

Task: VAE language modeling			
Dataset	**Metrics**	**VAE-LSTM**	**VAE-Transformer**
Yahoo (Yang et al., 2017b)	Test PPL	68.31	**61.26**
	Test NLL	337.36	**328.67**
PTB (Bowman et al., 2015)	Test PPL	105.27	**102.46**
	Test NLL	102.06	**101.46**

Table 8.1: Deployment of Transformer on the task of VAE language modeling (Bowman et al., 2015). Both test set perplexity (PPL) and sentence-level negative log likelihood (NLL) are evaluated (The lower the better). The model with a Transformer decoder consistently outperforms the one with a conventional LSTM decoder. For fair comparison, both models have the same parameter size.

Figure 8.6 to the following:

```
rnn_cell:
  type: path.to.MyCell
  kwargs:
    my_kwarg_1: 123
    my_kwarg_2: 'xyz'
    ...
```

as long as the `MyCell` class is accessible by Python, and its interface is compatible to other parts of the model.

Incorporating customized modules with Texar library APIs is even more flexible and straightforward. As the library APIs are designed to be coherent with the native TensorFlow/PyTorch programming interfaces, any externally-defined modules can directly be combined with Texar components as needed.

8.3 Case Studies

In this section, we conduct several case studies to show that Texar can greatly reduce implementation efforts and easily enable technique sharing among different tasks.

8.3.1 One Technique on Many Tasks: Transformer

Transformer, as first introduced in (Vaswani et al., 2017), has greatly improved the machine translation results and created other successful models, such as BERT for text embedding (Devlin et al., 2019), GPT-2 for language modeling (Radford et al., 2018), text infilling (Zhu et al., 2019), etc. Texar supports easy construction of these models and fine-tuning pretrained weights. On Texar, we can easily deploy the Transformer components to various other tasks and get improved results.

Task: Conversation generation		
Metrics	**HERD-GRU**	**HERD-Transformer**
BLEU-3 prec	0.281	**0.289**
BLEU-3 recall	0.256	**0.273**
BLEU-4 prec	0.228	**0.232**
BLEU-4 recall	0.205	**0.214**

Table 8.2: Comparison of Transformer decoder and GRU RNN decoder within the conversation model HERD (Serban et al., 2016) for response generation, on the Switchboard dataset (Zhao et al., 2017).

The first task we explore is the variational autoencoder (VAE) language modeling (Bowman et al., 2015). LSTM RNN has been widely-used in VAE for decoding sentences. We follow the experimental setting in previous work (Bowman et al., 2015; Yang et al., 2017b), and test two models, one with the traditional LSTM RNN decoder, and the other with the Transformer decoder. All other model configurations are the same in the two models. Notably, with Texar, changing the decoder from an LSTM to a Transformer is achieved by modifying only 3 lines of code. Table 8.1 shows the results. We see that the Transformer VAE consistently improves over the conventional LSTM VAE, showing that the Transformer architecture can benefit tasks beyond machine translation.

It is worth noting that, building the VAE language model (including data reading, model construction and optimization specification) on Texar uses only 70 lines of code (with the length of each line < 80 chars). As a (rough) reference, a popular public TensorFlow code (Li, 2017) of the same model has used around 400 lines of code for the same part (without line length limit).

The second task is to generate a response given conversation history. Following (Serban et al., 2016), the conversation history is encoded with a Texar module `HierarchicalRNNEncoder` which is followed with a decoder to generate the response. Similar as above, we study the performance of a Transformer decoder in comparison with a more conventional GRU RNN decoder. Table 8.2 shows the results. Again, we see that the Transformer model generalizes well to the conversation generation setting, and consistently outperforms the GRU RNN counterpart. Regarding implementation efforts, our implementation based on Texar has around 100 lines of code, while the reference code (Zhao et al., 2017) based on TensorFlow involves over 600 lines for constructing the same part.

8.3.2 One Task with Many Techniques: Language Modeling

We next showcase how Texar can support investigation of diverse techniques on a single task. This can be valuable for research community to standardize experimental configurations and foster fair, reproducible comparisons. As a case study, we choose the standard language modeling task (Zaremba et al., 2014). Note that this is different from the VAE language modeling task above, due to different data partition strategies conventionally adopted in respective research

Models	Test PPL	Lines of Model Code
LSTM RNN with MLE (Zaremba et al., 2014)	74.23	42
LSTM RNN with seqGAN (Yu et al., 2017b)	74.12	115
Memory Network LM (Sukhbaatar et al., 2015)	94.82	39

Table 8.3: Comparison of the three models on the task of language modeling, using the PTB dataset (Zaremba et al., 2014). The lower the test perplexity (PPL), the better. We also report the lines of code for implementing the respective model (including model construction, and loss/optimization specification, excluding data reading). In comparison, the official implementation of SeqGAN on TensorFlow involves 480 lines of model code (Yu et al., 2017b). Texar implementations have limited the maximum length of each line to 80 chars.

Models	Accuracy	BLEU	Lines of Model Code
Shen et al. (2017)	79.5	12.4	485
Shen et al. (2017) on Texar	82.5	13.0	136
Hu et al. (2017a) on Texar	**88.6**	**38.0**	105

Table 8.4: Text style transfer on the Yelp data (Shen et al., 2017). The first row is the original open-source implementation by the authors of (Shen et al., 2017). The subsequent two rows are Texar implementations of the two work. Texar implementations have limited the maximum length of each line to 80 chars.

lines.

We compare three models as shown in Table 8.3. The LSTM RNN trained with maximum likelihood estimation (MLE) (Zaremba et al., 2014) is the most widely used model for language modeling, due to its simplicity and prominent performance. We use the exact same architecture as generator and setup a (seq)GAN (Yu et al., 2017b) framework to train the language model with adversarial learning. (The generator is pre-trained with MLE.) From Table 8.3 we see that adversarial learning (almost) does not improve in terms of perplexity (which can be partly because of the high variance of the policy gradient in seqGAN learning). We further evaluate a memory network-based language model (Sukhbaatar et al., 2015) which has the same number of parameters with the LSTM RNN model. The test set perplexity is significantly higher than the LSTM RNNs in our experiments, which is not unexpected because LSTM RNN models are well studied for language modeling and a number of optimal modeling and optimization choices are already known.

Besides the benchmark performance of the various models, Table 8.3 also reports the amount of code for implementing each of the models. Texar makes the development very efficient.

8.3.3 Composite Model Architectures: Text Style Transfer

To further demonstrate the versatility of Texar for composing complex model architectures, we next choose the newly-emerging task of text style transfer (Hu et al., 2017a; Shen et al., 2017).

The task aims to manipulate the text of an input sentence to change from one attribute to another (e.g., from positive sentiment to negative), given only non-parallel training data of each style. The criterion is that the output sentence accurately entails the target style, while preserving the original content and other properties well.

We use Texar to implement the models from both (Hu et al., 2017a) and (Shen et al., 2017), whose model architectures fall in the category (f) and (g) in Figure 8.1, respectively. Experimental settings mostly follow those in (Shen et al., 2017). Following previous setting, we use a pre-trained sentiment classifier to evaluate the transferred style accuracy. For evaluating how well the generated sentence preserves the original content, we measure the BLEU score between the generated sentence and the original one (the higher the better) (Yang et al., 2018). Table 8.4 shows the results. Our re-implementation of (Shen et al., 2017) replicates and slightly surpasses the original results, while the implementation of (Hu et al., 2017a) provides better performance in terms of the two metrics. Implementations on Texar use fewer lines of code for the composite model construction.

8.4 Related Work

Text generation is a broad research area with rapid advancement. There exist several toolkits that focus on one or a few specific tasks. For example, for neural machine translation and alike, there are Google Seq2seq (Britz et al., 2017) and Tensor2Tensor (Vaswani et al., 2018) on TensorFlow, OpenNMT (Klein et al., 2017) on (Py)Torch, XNMT (Neubig et al., 2018) on DyNet, Nematus (Sennrich et al., 2017) on Theano, MarianNMT (Junczys-Dowmunt et al., 2018) on C++, and others. For dialogue, ParlAI (Miller et al., 2017) is a software platform specialized for research in the field. Differing from these highly task-focusing toolkits, Texar aims to cover as many text generation tasks as possible. The goal of versatility poses unique challenges to the design, requiring high modularity and extensibility.

On the other end of spectrum, there are libraries and tools for more general natural language processing (NLP) or deep learning applications. For example, AllenNLP (AllenAI, 2018), QickNLP (Pytorch, 2018), GluonNLP (DMLC, 2018) and others are designed for the broad NLP tasks in general, while Keras (Chollet et al., 2017) is for high conceptual-level programming without specific task focuses. In comparison, though extensible to broader techniques and applications, Texar has a proper focus on the text generation sub-area, and provide a comprehensive set of modules and functionalities that are well-tailored and readily-usable for relevant tasks. For example, Texar provides rich `text docoder` modules with optimized interfaces to support over ten decoding methods (see section 8.2.3 for an example), all of which can be invoked and interact with other modules conveniently.

It is also notable that some platforms have been developed for specific types of algorithms, such as OpenAI Gym (Brockman et al., 2016), DeepMind Control Suite (Tassa et al., 2018), and ELF (Tian et al., 2017) for reinforcement learning in game environments. Texar has drawn inspirations from these toolkits when designing relevant specific algorithm supports.

8.5 Conclusion

This chapter has introduced Texar, an open-source, general-purpose toolkit based on the standardized ML formulation and design, that supports a broad set of machine learning, especially text generation, applications and algorithms. The toolkit is modularized to enable easy replacement of components, and extensible to allow seamless integration of any external or customized modules. With the toolkit, a wide range of existing and future methods to come can be composed and solved mechanically with standardized modules.

Chapter 9

Conclusion

In this dissertation, we studied the problem of learning with all types of experiences, aiming to find principled methods of building AI agents that can improve from diverse information sources and can solve problems in a reusable, repeatable, and composable manner. To this end, we established a standardized formalism of machine learning, materialized as a standard equation of objective function, that gives a holistic view of the broad landscape of learning algorithms, unifies a number of well-known algorithms originally dealing with disparate types of experiences, and formulates a vast algorithmic design space governed by a few components regarding the experiences, model fitness, and uncertainty. The standardized formalism allows mechanic ways to design learning solutions that integrate all available experiences in training a model. On this basis, we developed effective approaches of learning with logic rules and auxiliary models, in combination with all other forms of experiences. The standardized formalism also bridges together vastly different learning problems and the respective research areas, fostering unified and repeatable solutions. With the guidance in hand, we were able to create solutions systematically to a wide range of long challenging problems that were previously seen as distinct, including automated data manipulation, learning and refining knowledge constraints, and stabilizing GAN training, etc. Finally, we operationalized the standardized and composable view of ML by delivering an open-source toolkit in support of fast and repeatable solution creation for broad set of text generation and machine learning problems.

The research presented here shapes exciting open questions and opportunities for future study. Below we discuss a few of these directions.

Broader landscape. Our discussion on the standard equation and related learning paradigms (e.g., supervised, unsupervised, active, reinforcement, adversarial, knowledge-constrained learning) has largely focused on learning in a static environment (e.g., fixed data or reward distributions). A natural next step is to study and connect to the other broad set of learning contexts, such as meta-learning and lifelong learning, to gain a new dimension of understanding about learning in changing environment. To this end, it would be necessary to extend the definition of experience function to be time dependent. Further, in the cases of unknown dynamics of the environment, an objective function based standard equation may no longer be flexible enough to describe the changing learning processes, making it necessary to move towards finer-grained formulations of the learning procedures. Similar to the strong utilities of the current standard equation for solution design, a standardized formalism of the broader landscape is expected to unleash even more power by enabling principled design of learning systems that continuously improve by interacting with and collecting diverse signals from the outer world.

Unified theoretical analysis. As discussed in Chapter 2, the general standardized formalism poses new questions about theoretical analysis of learning from all experiences. A question of particular importance in practice is about how we can guarantee better performance after integrating more experiences? The analysis is challenging because the different types of experiences can each encode different information, sometimes noisy and even conflicting with each other (e.g., not all data instances comply with a logic rule), and thus plugging in an additional source of experience does not always lead to positive effects. But before that, a more basic question to ask is perhaps about how we can characterize learning with some special or novel forms of experiences, such as logic rules and auxiliary models? What statistical tools we may use for characterization, and what would the convergence guarantees, complexity, robustness, and other theoretical and statistical properties be? Inspired by how we generalized the specialized algorithms to new problems, a promising way of the theoretical analysis would be to again leverage the standard equation and repurpose the existing analyses originally dealing with supervised learning, online learning, and reinforcement learning, to now analyze the learning with all other experiences.

From standardization to automation. As in other mature engineering disciplines such as mechanical engineering, standardization is followed by automation. The standardized formalism opens up the possibility of automating the process of creating and improving ML algorithms and solutions. The current "AutoML" practice has largely focused on automatic search of model architectures and hyperparameters, thanks to the well-defined architectural and hyperparameter search spaces. A standard equation that defines a structured algorithmic space would similarly suggest opportunities for automatic search or optimization of learning algorithms, which is expected to be more efficient than direct search on the programming code space (Real et al., 2020). We have demonstrated in Parts II and III how new algorithms can mechanically be created by composing experiences and/or other algorithmic components together. It is of great practical significance to have an automated engine that further streamlines the process. For example, once a new advance is made to reward learning, the engine would automatically amplify the progress and deliver enhanced functionalities of learning data manipulation and adapting knowledge constraints. Similarly, domain experts can simply input a variety of experiences available in their own problem, and expect an algorithm to be automatically composed to learn the target model they want. The sophisticated algorithm manipulation and creation would greatly simplify machine learning workflow in practice and boost the accessibility of ML to much broader users.

From Maxwell's equations to general relativity, and to quantum mechanics and standard model, "Physics is the study of symmetry." remarked physicist Phil Anderson. The end goal of physics research seems to be clear—a "theory of everything" that fully explains and links together all physical aspects. The "end goal" of ML/AI is surely much elusive. Yet the unifying way of thinking would be incredibly valuable to continuously unleash the extensive more power of the current vibrant research, to produce more principled understanding, and to build more versatile AI agents.

Bibliography

ASYML Open-source Machine Learning Libraries. 2019. `https://asyml.io`. 1

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 4.2.4

A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a posteriori policy optimisation. In *ICLR*, 2018. 2.3.3, 5.3, 6.1, 6.2, 6.3.2.1, 7.1, 7.2, 7.3.1, 7.3.3

AllenAI. AllenNLP. 2018. URL `http://allennlp.org`. 8.4

J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*, 2016. 5.2, 6.2

M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *ICLR*, 2017. 4.2.3

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017a. 7.1, 7.2, 7.2

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017b. 2.4.1, **??**

M. Artetxe, G. Labaka, E. Agirre, and K. Cho. Unsupervised neural machine translation. *arXiv preprint arXiv:1710.11041*, 2017. 4.2.4.3, 4.2.5

S. H. Bach, M. Broecheler, B. Huang, and L. Getoor. Hinge-loss Markov random fields and probabilistic soft logic. *arXiv preprint arXiv:1505.04406*, 2015. 3.3.1

S. H. Bach, M. Broecheler, B. Huang, and L. Getoor. Hinge-loss Markov random fields and probabilistic soft logic. *The Journal of Machine Learning Research*, 18(1):3846–3912, 2017. 2.3.2

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. (document), 3.1, 4.1.1, 4.2.1, 6.5.2, 8.1

D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016. 8.1

S. Baluja and I. Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017. 5.2

K. Bellare, G. Druck, and A. McCallum. Alternating projections for learning with expectation constraints. In *UAI*, 2009. 6.2

D. Berthelot, T. Schumm, and L. Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017. 7.1

A. Bordes, Y.-L. Boureau, and J. Weston. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*, 2016. (document), 8.1

S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015. (document), 4.1.1, 4.1.2, 4.1.3.1, 4.1.4, 4.2.1, 4.2.2, 8.1, **??**, 8.1, 8.3.1

T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007. 4.2.5

D. Britz, A. Goldie, T. Luong, and Q. Le. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*, 2017. 8.1, 8.2.1, 8.4

A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 7.1, 7.2

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016. 8.4

P. F. Brown, J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2): 79–85, 1990. 8.1

Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015. 7.1, 7.2, 7.3.3

M. Caccia, L. Caccia, W. Fedus, H. Larochelle, J. Pineau, and L. Charlin. Language gans falling short. In *ICLR*, 2020. 7.1

H.-S. Chang, E. Learned-Miller, and A. McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *NeurIPS*, pages 1002–1012, 2017. 5.2

M.-W. Chang, L. Ratinov, and D. Roth. Guiding semi-supervision with constraint-driven learning. In *ACL*, 2007. 1

M.-W. Chang, L. Ratinov, and D. Roth. Structured learning with constrained conditional models. *Machine learning*, 2012. 3

T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017a. 7.3.2, 7.3.3

T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint:1702.07983*, 2017b. 4.2.5

J. Chen, P.-S. Huang, X. He, J. Gao, and L. Deng. Unsupervised learning of predictors from unpaired input-output samples. *arXiv preprint arXiv:1606.04646*, 2016a. 4.2.5

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In

NeurIPS, 2016b. 4.1.1, 4.1.2, 4.2.1, 4.2.5, 6.1

J. P. Chiu and E. Nichols. Named entity recognition with bidirectional LSTM-CNNs. *arXiv preprint arXiv:1511.08308*, 2015. 3.4.2, 3.5.2.1, **??**

F. Chollet et al. Keras (2015), 2017. 8.4

C. Chu, J. Blanchet, and P. Glynn. Probability functional descent: A unifying perspective on gans, variational inference, and reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, 2019. 2.4.1, 2.4.1, 2.4.1

C. Chu, K. Minami, and K. Fukumizu. Smoothness and stability in gans. In *ICLR*, 2020. 7.1

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 4.2.7.2

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537, 2011. 3.1, 3.2, **??**

E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. In *CVPR*, 2019. 5.1, 5.2, 5.4.1

I. Dagan and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *Machine Learning Proceedings*, pages 150–157. Elsevier, 1995. 2.3.1.4

P. Dayan and G. E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997. 2.3.3, 5.3, 6.2, 7.2

M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013. 2.3.3, 6.2, 7.2

W. E. Deming and F. F. Stephan. On a least squares adjustment of a sampled frequency table when the expected marginal totals are known. *The Annals of Mathematical Statistics*, 11(4): 427–444, 1940. 2.1.1.1

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977. 2.1.1.2

Y. Deng, A. Bakhtin, M. Ott, A. Szlam, and M. Ranzato. Residual energy-based models for text generation. In *ICLR*, 2020. 7.3.3

E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015. 4.2.5

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. 5.1, 5.4.2, 5.5.1, **??**, **??**, 8.3.1

Q. Diao, M. Qiu, C.-Y. Wu, A. J. Smola, J. Jiang, and C. Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (JMARS). In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 193–202. ACM, 2014. 4.1.4, 6.5.2

DMLC. GluonNLP. 2018. URL `https://github.com/dmlc/gluon-nlp`. 8.4

A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. *arXiv preprint arXiv:1602.02644*, 2016. 4.1.1, 4.1.2

Q. Dou and K. Knight. Large scale decipherment for out-of-domain machine translation. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 266–275. Association for Computational Linguistics, 2012. 4.2.4.1

S. Ertekin, J. Huang, L. Bottou, and L. Giles. Learning on the border: active learning in imbalanced data classification. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 127–136, 2007. 2.3.1.4, **??**

Y. Fan, F. Tian, T. Qin, X.-Y. Li, and T.-Y. Liu. Learning to teach. In *ICLR*, 2018. 5.2

F. Farnia and D. Tse. A convex duality framework for gans. *Advances in Neural Information Processing Systems*, 31:5248–5258, 2018. 2.4.1

W. Fedus, I. Goodfellow, and A. M. Dai. Maskgan: Better text generation via filling in the _. *arXiv preprint arXiv:1801.07736*, 2018. (document), 6.5.2, 7.1, 8.2

L. T. Fernholz. *Von Mises calculus for statistical functionals*, volume 19. Springer Science & Business Media, 2012. 2.4.1

C. Finn, P. Christiano, P. Abbeel, and S. Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016a. 6.2

C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016b. 6.1, 6.2

O. Firat, K. Cho, and Y. Bengio. Multi-way, multilingual neural machine translation with a shared attention mechanism. *arXiv preprint arXiv:1601.01073*, 2016. (document), 8.1

J. Foulds, S. Kumar, and L. Getoor. Latent topic networks: A versatile probabilistic programming framework for topic models. In *Proc. of ICML*, pages 777–786, 2015. 3.3.1

M. V. França, G. Zaverucha, and A. S. d. Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104, 2014. 3.2

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. 5.1, 5.2

J. Fu, K. Luo, and S. Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017a. 6.2

Z. Fu, X. Tan, N. Peng, D. Zhao, and R. Yan. Style transfer in text: Exploration and evaluation. *arXiv preprint arXiv:1711.06861*, 2017b. 4.2.4, **??**

K. Ganchev, J. Gillenwater, B. Taskar, et al. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11(Jul):2001–2049, 2010. 1, 2.1.3, **??**, 3, 3.1, 3.2, 3.3.3, 3.5.1.2, 3.7, 6.1, 6.2, 6.3.1, 6.3.1

A. S. d. Garcez, K. Broda, and D. M. Gabbay. *Neural-symbolic learning systems: foundations* *and applications*. Springer Science & Business Media, 2012. 3.1, 3.2

L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2414–2423. IEEE, 2016. 4.2.5

I. D. Gebru, X. Alameda-Pineda, F. Forbes, and R. Horaud. EM algorithms for weighted-data clustering with application to audio-visual scene analysis. *IEEE transactions on pattern analysis and machine intelligence*, 38(12):2402–2415, 2016. 2.3.1.2

M. Gell-Mann. Beauty and truth in physics: Murray Gell-Mann on TED.com, 2007. URL `https://www.ted.com/talks/murray_gell_mann_beauty_truth_and_physics?language=en`. 1

P. K. B. Giridhara, M. Chinmaya, R. K. M. Venkataramana, S. S. Bukhari, and A. Dengel. A study of various text augmentation techniques for relation classification in free text. In *ICPRAM*, 2019. 5.4.2

P. W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990. 2.1.1.2

A. N. Gomez, S. Huang, I. Zhang, B. M. Li, M. Osama, and L. Kaiser. Unsupervised cipher cracking using discrete gans. *arXiv preprint arXiv:1801.04883*, 2018. 4.2.5

K. Gong, X. Liang, X. Shen, and L. Lin. Look into person: Self-supervised structure-sensitive learning and a new benchmark for human parsing. In *CVPR*, pages 6757–6765, 2017. 6.5.1

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014. 1, 2.3.4, 2.4.1, 2.4.1, **??**, 4, 4.1.1, 4.1.2, 4.2.1, 4.2.2, 6.1, 6.4.2, 7.1, 7.2, 7.3.2, 8.1

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 6.1

Y. Grandvalet, Y. Bengio, et al. Semi-supervised learning by entropy minimization. In *NeurIPS*, volume 17, pages 529–536, 2004. 4.1.3.2

A. Grover, J. Song, A. Kapoor, K. Tran, A. Agarwal, E. J. Horvitz, and S. Ermon. Bias correction of learned generative models using likelihood-free importance weighting. In *NeurIPS*, 2019. 7.1, 7.3.2

J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016. (document), 8.1

C. Gulcehre, O. Firat, K. Xu, K. Cho, L. Barrault, H.-C. Lin, F. Bougares, H. Schwenk, and Y. Bengio. On using monolingual corpora in neural machine translation. *arXiv preprint arXiv:1503.03535*, 2015. 4.2.5

C. Gulcehre, S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*, 2016. (document), 8.1

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017. (document), 7.1, 7.2, 7.3.3, 4, 7.3.4, **??**, 7.2, 7.4.1, 7.4.2, 7.4.3

J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang. Long text generation via adversarial training with leaked information. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. (document), 7.2, 7.4.2

M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander. Statistical inference of intractable generative models via classification. *arXiv preprint arXiv:1407.4981*, 2014. 2.3.4

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017. 6.3.2.1

I. Hal Daumé, J. Langford, and D. Marcu. Search-based structured prediction as classification. 2009. (document), 8.2

D. He, Y. Xia, T. Qin, L. Wang, N. Yu, T. Liu, and W.-Y. Ma. Dual learning for machine translation. In *Advances in Neural Information Processing Systems*, pages 820–828, 2016a. 4.2.5

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016b. (document), 5.1, 5.5.1, 5.2, **??**

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017. 7.4.1

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6): 82–97, 2012. 3.1

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 3.1, 3.3.2, 3.6.1

G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158, 1995. 2.1.1.2, 4.1.1, 4.1.2, 5.2, 6.4.2, 7.3.2

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. VIME: Variational information maximizing exploration. In *NeurIPS*, 2016. 2.3.3

E. Hovy and C.-Y. Lin. Automated text summarization and the summarist system. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998*, pages 197–214. Association for Computational Linguistics, 1998. 8.1

M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proc. of KDD*, pages 168–177. ACM, 2004. 3.5.1.1

Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing. Harnessing deep neural networks with logic rules. In *ACL*, 2016a. 1, 1.1, 2.1.3, 2.3.2, 3.6.1, 4.1.4.1, 4.1.5, 6.1, 6.2, 6.3.1

Z. Hu, Z. Yang, R. Salakhutdinov, and E. Xing. Deep neural networks with massive learned knowledge. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016b. 1, 3.6, 3.6.1, 4.1.5, 6.2

Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing. Toward controlled generation of text. In *ICML*, 2017a. (document), 1, 1.1, 3.6.1, 5.2, 6.1, 8.1, 8.1, 8.2, **??**, 8.3.3

Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing. Toward controlled generation of text. In *International Conference on Machine Learning*, pages 1587–1596, 2017b. 4.2.1, 4.2.2, 4.2.4, **??**, 4.2.4.2, 4.2.5, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 7.1, 7.4.3

Z. Hu, Z. Yang, R. Salakhutdinov, and E. P. Xing. On unifying deep generative models. In *ICLR*, 2017c. 4.1.5, 4.2.3, 7.1, 7.2, 7.3.3, 8.1

Z. Hu, Z. Yang, R. Salakhutdinov, X. Liang, L. Qin, H. Dong, and E. Xing. Deep generative models with learnable knowledge constraints. In *NeurIPS*, 2018a. 1, 1.1, 3.6, 4.2.5, 7.1, 7.2, 7.3.1, 7.3.1, 7.3.2, 7.3.3, 7.3.3, 8.1

Z. Hu, Z. Yang, R. Salakhutdinov, and E. P. Xing. On unifying deep generative models. In *ICLR*, 2018b. 1.1, 4.1.1, 4.1.2, 6.1

Z. Hu, H. Shi, B. Tan, W. Wang, Z. Yang, T. Zhao, J. He, L. Qin, D. Wang, et al. Texar: A modularized, versatile, and extensible toolkit for text generation. In *ACL 2019, System Demonstrations*, 2019a. `https://github.com/asyml/texar`. 1, 1.1

Z. Hu, B. Tan, R. Salakhutdinov, T. Mitchell, and E. Xing. Learning data manipulation for augmentation and weighting. In *NeurIPS*, 2019b. 1, 1.1, 2.3.1.3, 7.2

X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR, abs/1703.06868*, 2017. 4.2.5

P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017. 4.2.5

T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *Advances in neural information processing systems*, pages 470–476, 2000. 2.1.3

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. (document), 4.2.3, 5.4.2, 7.4.2, 8.2.1, 8.2.1, 8.7

E. Jaynes. Comment. *The American Statistician*, 42(4):280–281, 1988. 2.1.2

E. T. Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957. 2.1.1.1

L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *ICML*, 2018. 5.1, 5.2

M. I. Jordan. An introduction to probabilistic graphical models, 2003. 2.1.1.1

M. Junczys-Dowmunt, R. Grundkiewicz, T. Dwojak, H. Hoang, K. Heafield, T. Neckermann, F. Seide, U. Germann, A. F. Aji, N. Bogoychev, A. F. T. Martins, and A. Birch. Marian: Fast neural machine translation in C++. *arXiv preprint arXiv:1804.00344*, 2018. 8.4

T. Karaletsos, S. Belongie, C. Tech, and G. Rätsch. Bayesian representation learning with oracle constraints. In *Proc. of ICLR*, 2016. 3.2

A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015. 8.1

A. Katharopoulos and F. Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *ICML*, 2018. 5.2

T. Kim and Y. Bengio. Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*, 2016. 6.4.2, 7.3.3

Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014. (document), 3.4.1, **??**, **??**, 3.1, 3.5.1.1, **??**

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4.2.7.2

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013. 2.1.1.2, 2.1.1.2, 4.1.1, 4.1.2, 4.2.1, 5.2, 6.1, 6.3.1, 7.4.3, 8.2.1

D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *NeurIPS*, pages 3581–3589, 2014. (document), 4.1.1, 4.1.2, 4.1.3.1, 4.1, 4.1.4.1

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*, 2017. 8.1, 8.2.1, 8.4

T. Ko, V. Peddinti, D. Povey, and S. Khudanpur. Audio augmentation for speech recognition. In *INTERSPEECH*, 2015. 5.2

S. Kobayashi. Contextual augmentation: Data augmentation by words with paradigmatic relations. In *NAACL*, 2018. 5.2, 5.4.2, 5.5.2

A. Krizhevsky and G. Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010. 7.4.1

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3.1, 5.2

T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *NeurIPS*, pages 2539–2547, 2015. 3.2

M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *NeurIPS*, pages 1189–1197, 2010. 5.2

M. Kusner and J. Hernández-Lobato. GANs for sequences of discrete elements with the Gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016. 4.1.1, 4.1.2

M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017. 6.2

B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 3.1

A. M. Lamb, A. G. A. P. GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609, 2016. (document), 4.2.1, 4.2.2, 8.1

G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. In *Proc. of NAACL*, 2016. 3.4.2, 3.5.2.2, **??**, **??**

G. Lample, L. Denoyer, and M. Ranzato. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*, 2017. (document), 4.2.4.3, 4.2.5, 8.1

H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *AISTATS*, 2011. 6.1

A. B. L. Larsen, S. K. Sønderby, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. In *ICML*, 2016. 4.1.1, 4.1.2

Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *Proc. of ICML*, 2014. **??**

J. Lemley, S. Bazrafkan, and P. Corcoran. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:5858–5869, 2017. 5.2

S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018. 2.3.3, 5.3, 6.2, 7.2, 7.3.1

C. Li, J. Zhu, T. Shi, and B. Zhang. Max-margin deep generative models. In *Advances in neural information processing systems*, pages 1837–1845, 2015. 6.2

C. Y. Li, X. Liang, Z. Hu, and E. P. Xing. Hybrid retrieval-generation reinforced agent for medical image report generation. *arXiv preprint arXiv:1805.08298*, 2018a. 8.1

J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017. 4.2.1, 4.2.5

J. Li, R. Jia, H. He, and P. Liang. Delete, retrieve, generate: A simple approach to sentiment and style transfer. *arXiv preprint arXiv:1804.06437*, 2018b. (document), 4.2.4, 4.2.4.2, **??**, 4.8, 7.4.3

L. Li, M. L. Littman, T. J. Walsh, and A. L. Strehl. Knows what it knows: a framework for self-aware learning. *Machine learning*, 2011. 2.3.3

X. Li and D. Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, 2002. 5.5.2

Z.-Y. Li. A Tensorflow implementation of VAE. 2017. URL `https://github.com/Chung-I/Variational-Recurrent-Autoencoder-Tensorflow`. 8.3.1

P. Liang, H. Daumé III, and D. Klein. Structure compilation: trading structure for features. In *Proc. of ICML*, pages 592–599. ACM, 2008. 3.2, 3.5.1.2

P. Liang, M. I. Jordan, and D. Klein. Learning from measurements in exponential families. In *Proc. of ICML*, pages 641–648. ACM, 2009. 3.2, 6.2

X. Liang, Z. Hu, H. Zhang, C. Gan, and E. P. Xing. Recurrent topic-transition GAN for visual paragraph generation. In *ICCV*, 2017. (document), 6.2, 8.1

X. Liang, Z. Hu, and E. Xing. Symbolic graph reasoning meets convolutions. In *NeurIPS*, 2018. 6.2

K. Lin, D. Li, X. He, Z. Zhang, and M.-T. Sun. Adversarial ranking for language generation. In *Advances in Neural Information Processing Systems*, pages 3155–3165, 2017. 4.2.5

S. Lin, W. Wang, Z. Yang, H. Shi, F. Xu, X. Liang, E. Xing, and Z. Hu. Towards unsupervised text content manipulation. *arXiv preprint arXiv:1901.09501*, 2019. 8.1

S. Lin, W. Wang, Z. Yang, X. Liang, F. F. Xu, E. Xing, and Z. Hu. Data-to-text generation with style imitation. In *EMNLP Findings*, 2020. 1.1

Y. Liu, J. Chen, and L. Deng. Unsupervised sequence classification using sequential output statistics. In *Advances in Neural Information Processing Systems*, pages 3550–3559, 2017. 4.2.5

Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1096–1104, 2016. 6.5.1

Z. Liu, G. Ding, A. Bukkittu, M. Gupta, P. Gao, A. Ahmed, S. Zhang, X. Gao, S. Singhavi, L. Li, et al. A data-centric framework for composable nlp workflows. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 197–204, 2020. 1.1

D. Lopez-Paz, L. Bottou, B. Schölkopf, and V. Vapnik. Unifying distillation and privileged information. *arXiv preprint arXiv:1511.03643*, 2015. 3.3.2, 3.6.1

G. Luo, X. Huang, C.-Y. Lin, and Z. Nie. Joint named entity recognition and disambiguation. In *Proc. of EMNLP*, 2015. **??**, 3.5.2.2

M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015. (document), 8.1

M.-T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. Multi-task sequence to sequence learning. In *ICLR*, 2016. (document), 8.1

L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool. Pose guided person image generation. In *Advances in Neural Information Processing Systems*, pages 405–415, 2017. 6.5, 6.5.1

L. Ma, Q. Sun, S. Georgoulis, L. Van Gool, B. Schiele, and M. Fritz. Disentangled person image generation. In *CVPR*, 2018. 6.5, 6.5.1

X. Ma and E. Hovy. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proc. of ACL*, 2016. **??**, 3.5.2.2

A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011. 5.5.2

C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. 2017. 7.4.2

N. Madnani and B. J. Dorr. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387, 2010. 8.1

T. Malisiewicz, A. Gupta, A. A. Efros, et al. Ensemble of exemplar-SVMs for object detection and beyond. In *ICCV*, volume 1, page 6. Citeseer, 2011. 5.1

M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015. 7.1

S. Mei, J. Zhu, and J. Zhu. Robust regbayes: Selectively incorporating first-order logic domain knowledge into bayesian models. In *International Conference on Machine Learning*, pages 253–261, 2014. 6.2

T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010. (document), 4.1.1, 8.2, 8.2.1

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS*, pages 3111–3119, 2013. 3.5.1.1

A. H. Miller, W. Feng, A. Fisch, J. Lu, D. Batra, A. Bordes, D. Parikh, and J. Weston. ParlAI: A dialog research software platform. *arXiv preprint arXiv:1705.06476*, 2017. 8.1, 8.4

G. A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11): 39–41, 1995. 5.5.1

M. Minksy. Learning meaning. *Technical Report AI Lab Memo. Project MAC. MIT*, 1980. 3.1

M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 5.2

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018. **??**, 7.4.1

A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014. 2.1.1.2, 2.1.1.2

S. Mohamed and B. Lakshminarayanan. Learning in implicit generative models. *arXiv preprint arXiv:1610.03483*, 2016. 6.1, 6.4.2

R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998. 2.1.1.2, 7.3.1

G. Neubig, M. Sperber, X. Wang, M. Felix, A. Matthews, S. Padmanabhan, Y. Qi, D. S. Sachan, P. Arthur, P. Godard, et al. XNMT: The eXtensible neural machine translation toolkit. *arXiv preprint arXiv:1803.00188*, 2018. 8.4

G. Neumann et al. Variational inference for policy search in changing situations. In *ICML*, pages 817–824, 2011. 6.2

A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proc. of CVPR*, pages 427–436. IEEE, 2015. 3.1

W. Nie, N. Narodytska, and A. Patel. Relgan: Relational generative adversarial networks for text generation. 2018. (document), 7.2, 7.3, 7.4.2

R. Nock, Z. Cranko, A. K. Menon, L. Qu, and R. C. Williamson. f-GANs in an information geometric nutshell. In *Advances in Neural Information Processing Systems*, pages 456–464, 2017. 7.1

M. Norouzi, S. Bengio, N. Jaitly, M. Schuster, Y. Wu, D. Schuurmans, et al. Reward augmented maximum likelihood for neural structured prediction. In *Advances In Neural Information Processing Systems*, pages 1723–1731, 2016. (document), 2.3.1.3, 5.2, 8.2

S. Nowozin, B. Cseke, and R. Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In *NeurIPS*, pages 271–279, 2016. 2.4.1, **??**, 7.1

A. Odena, C. Olah, and J. Shlens. Conditional image synthesis with auxiliary classifier GANs. *arXiv preprint arXiv:1610.09585*, 2016. 4.1.1

A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016. 6.1

J. Paisley, D. Blei, and M. Jordan. Variational Bayesian inference with stochastic search. In *ICML*, 2012. 2.1.1.2

B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proc. of ACL*, pages 115–124, 2005. 3.5.1.1

D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv 1904.08779*. 5.2

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 2.3.3, 2.3.3

D. Pathak, D. Gandhi, and A. Gupta. Self-supervised exploration via disagreement. In *ICML*, 2019. 2.3.3

X. Peng, Z. Tang, F. Yang, R. S. Feris, and D. Metaxas. Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation. In *CVPR*, 2018. 5.2

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proc. of EMNLP*, volume 14, pages 1532–1543, 2014. 3.5.2.1

J. Peters, K. Mülling, and Y. Altun. Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta, 2010. 6.1, 6.3.2.1, 6.3.2.1

N. Pourdamghani and K. Knight. Deciphering related languages. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2513–2518, 2017. 4.2.4.3

S. Prabhumoye, Y. Tsvetkov, R. Salakhutdinov, and A. W. Black. Style transfer through back-translation. *arXiv preprint arXiv:1804.09000*, 2018. 4.2.5

A. Pumarola, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer. Unsupervised person image synthesis in arbitrary poses. In *CVPR*, 2018. 6.5, 6.5.1

Pytorch. QuickNLP. 2018. URL `https://github.com/outcastofmusic/quick-nlp`. 8.4

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015a. (document), 7.1, 7.3, 7.4.1

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015b. 4.1.1, 4.2.1, 4.2.5

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. Technical report, Technical report, OpenAI, 2018. 8.3.1

R. Ranganath, S. Gerrish, and D. Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014. 2.1.1.2, 2.1.1.2

M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015. (document), 8.1, 8.2, 8.2.1

A. J. Ratner, H. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré. Learning to compose domain-specific transformations for data augmentation. In *NeurIPS*, 2017. 5.1, 5.2

K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *IJCAI*, 2013. 2.3.3, 7.2

E. Real, C. Liang, D. So, and Q. Le. Automl-zero: evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, pages 8007–8019. PMLR, 2020. 9

S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014. 4.1.3.2

M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to reweight examples for robust deep learning. In *ICML*, 2018. 5.1, 5.2, 5.4.1, 5.4.2, 5.5.1, **??**, **??**, 5.5.2, **??**, **??**, 5.5.3

S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel. Self-critical sequence training for image captioning. In *CVPR*, pages 7008–7024, 2017. (document), 8.2

M. Richardson and P. Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006. 3.2

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In *NeurIPS*, pages 2226–2234, 2016. 4.2.5, 7.4.1

R. Samdani, M.-W. Chang, and D. Roth. Unified expectation maximization. In *ACL*, 2012. 2.3.2, **??**

F. Santambrogio. Optimal transport for applied mathematicians. *Birkäuser, NY*, 55(58-63):94, 2015. 2.4.1

A. Sanyal, P. H. Torr, and P. K. Dokania. Stable rank normalization for improved generalization in neural networks and gans. In *ICLR*, 2020. **??**, 7.4.1

J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2010. 2.3.3, 2.3.3

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015a. 7.1, 7.2, 7.3.1

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015b. 2.4.1, 6.1, 6.3.2.1

J. Schulman, X. Chen, and P. Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017a. 6.3.2.1

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017b. 7.1, 7.2, 7.3.1, 7.3.2

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017c. 2.4.1

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017. 8.1

R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data. In *ACL*, 2016. 5.2

R. Sennrich, O. Firat, K. Cho, A. Birch, B. Haddow, J. Hitschler, M. Junczys-Dowmunt, S. Läubli, A. V. M. Barone, J. Mokry, et al. Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*, 2017. 8.4

I. V. Serban, A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. 2016. (document), 8.1, 8.2, 8.3.1

B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012. 1, 2.3.1.4

T. Shen, T. Lei, R. Barzilay, and T. Jaakkola. Style transfer from non-parallel text by cross-alignment. In *NeurIPS*, 2017. (document), 4.2.1, 4.2.2, 4.2.4, **??**, **??**, 4.2.4.1, 4.2.4.1, 4.2.4.2, **??**, 4.2.4.3, **??**, 4.2.5, 4.2.7.2, **??**, **??**, **??**, **??**, **??**, **??**, **??**, 7.4.3, 8.1, **??**, **??**, 8.4, 8.3.3

A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, pages 761–769, 2016. 5.2

N. Siddharth, B. Paige, A. Desmaison, J.-W. v. d. Meent, F. Wood, N. D. Goodman, P. Kohli, and P. H. Torr. Learning disentangled representations in deep generative models. 2017. 4.1.2

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 3.1

P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition—tangent distance and tangent propagation. In *Neural networks: tricks of the trade*, pages 239–274. Springer, 1998. 5.2

S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2010. 2.3.3

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, pages 1631–1642, 2013. **??**, 3.5.1.1, 4.1.4, 5.5.2, 5.5.3

J. Song and S. Ermon. Bridging the gap between f-gans and wasserstein gans. In *ICML*, 2020. 7.2

S. Subramanian, G. Lample, E. M. Smith, L. Denoyer, M. Ranzato, and Y.-L. Boureau. Multiple-attribute text style transfer. *arXiv preprint arXiv:1811.00552*, 2018. **??**

S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *NeurIPS*, pages 2440–2448, 2015. (document), 8.1, **??**, 8.3.2

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks.

In *Advances in neural information processing systems*, pages 3104–3112, 2014. (document), 4.1.1, 8.1

R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction (2nd edition). 2017. 1, 2.3.3, 6.3.2

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 2.3.3, 4.2.1, 4.2.3, 4.2.5

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *Proc. of ICLR*, 2014. 3.1

Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017. 4.1.2

B. Tan, Z. Hu, Z. Yang, R. Salakhutdinov, and E. Xing. Connecting the dots between MLE and RL for sequence generation. *arXiv preprint arXiv:1811.09740*, 2018. (document), 1.1, 5.1, 5.2, 6.2, 7.2, 7.3.1, 8.1, 8.1, 8.2

J. Tang, T. Zhao, C. Xiong, X. Liang, E. P. Xing, and Z. Hu. Target-guided open-domain conversation. In *ACL*, 2019. 1, 1.1, 8.1

S. Tang, H. Jin, C. Fang, and Z. Wang. Unsupervised sentence representation learning with adversarial auto-encoder. 2016. 4.1.1, 4.1.2

B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Advances in neural information processing systems*, pages 25–32, 2004. 2.1.3, 2.1.4, 6.2

Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 8.4

Y. W. Teh and M. Welling. On improving the efficiency of the iterative proportional fitting procedure. In *AIStats*, 2003. 2.1.1.1

D. Terjék. Adversarial lipschitz regularization. In *ICLR*, 2020. **??**, 7.4.1

Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick. ELF: An extensive, lightweight and flexible research platform for real-time strategy games. In *NeurIPS*, pages 2656–2666, 2017. 8.4

Y. Tian, Z. Hu, and Z. Yu. Structured content preservation for unsupervised text style transfer. *arXiv preprint arXiv:1810.06526*, 2018. **??**

A. Tikhonov, V. Shibaev, A. Nagaev, A. Nugmanova, and I. P. Yamshchikov. Style transfer for texts: Retrain, report errors, compare with rewrites. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3927–3936, 2019. (document), **??**, 7.4, 7.4.3

E. F. Tjong Kim Sang and F. De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proc. of CoNLL*, pages 142–147. Association for Computational Linguistics, 2003. 3.5.2.1

G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the eighth National conference on Artificial intelligence*, pages 861–866. Boston, MA, 1990. 3.2

T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid. A Bayesian data augmentation approach for learning deep models. In *NeurIPS*, pages 2797–2806, 2017. 5.2

A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016. 4.1.1

V. N. Vapnik. Statistical learning theory. *John Wiley and Sons*, 1998. 2.1.3

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. (document), 8.1, 8.1, 8.3.1

A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, et al. Tensor2Tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*, 2018. 8.4

O. Vinyals and Q. Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015. 4.2.1

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *NeurIPS*, pages 2692–2700, 2015a. (document), 8.1

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015b. 4.1.1, 4.2.1, 8.1

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, pages 1–305, 2008. 2.1.1.2

D. Wang and Q. Liu. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016. 6.4.2

S. Wang and C. Manning. Fast dropout training. In *Proc. of ICML*, pages 118–126, 2013. **??**

T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. *arXiv preprint arXiv:1711.11585*, 2017. 6.5.1

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. (document), 6.2

J. W. Wei and K. Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019. 5.2

X. Wei, B. Gong, Z. Liu, W. Lu, and L. Wang. Improving the improved training of wasserstein GANs: A consistency term and its dual effect. *arXiv preprint arXiv:1803.01541*, 2018. 7.1, 7.2, 7.3.3, **??**, 7.4.1

T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *EMNLP*, 2015. 4.1.1

T.-H. Wen, D. Vandyke, N. Mrksic, M. Gasic, L. M. Rojas-Barahona, P.-H. Su, S. Ultes, and S. Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016. 4.2.1

J. D. Williams and S. Young. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007. 8.1

T. Wilson, J. Wiebe, and P. Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005. 4.1.4

S. Wiseman and A. M. Rush. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*, 2016. (document), 8.2

S. Wiseman, S. M. Shieber, and A. M. Rush. Challenges in data-to-document generation. *arXiv preprint arXiv:1707.08052*, 2017. 8.1

X. Wu, S. Lv, L. Zang, J. Han, and S. Hu. Conditional BERT contextual augmentation. *arXiv preprint arXiv:1812.06705*, 2018. 5.2, 5.4.2, 5.5.1, **??**, 5.5.2

Y. Wu, P. Zhou, A. G. Wilson, E. P. Xing, and Z. Hu. Improving GAN training with probability ratio clipping and sample reweighting. *arXiv preprint arXiv:2006.06900*, 2020. 1, 1.1, 2.4.1, 2.4.1, 2.4.1, **??**

Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and L. Q. V. Unsupervised data augmentation. *arXiv preprint arXiv 1904.12848*, 2019. 5.2

Z. Xie, S. I. Wang, J. Li, D. Lévy, A. Nie, D. Jurafsky, and A. Y. Ng. Data noising as smoothing in neural network language models. In *ICLR*, 2017. 5.2

Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *arXiv preprint arXiv:1806.07755*, 2018. 7.4.1

B. Yang and C. Cardie. Context-aware learning for sentence-level sentiment analysis with posterior regularization. In *Proc. of ACL*, pages 325–335, 2014. **??**

Z. Yang, W. Chen, F. Wang, and B. Xu. Improving neural machine translation with conditional sequence generative adversarial nets. *arXiv preprint arXiv:1703.04887*, 2017a. 4.2.5

Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. In *ICML*, 2017b. (document), 4.1.2, 4.2.1, 4.2.2, 8.1, **??**, 8.3.1

Z. Yang, Z. Hu, C. Dyer, E. Xing, and T. Berg-Kirkpatrick. Unsupervised text style transfer using language models as discriminators. In *NeurIPS*, 2018. (document), 1, 1.1, 3.6.1, 7.4.3, 8.1, 8.2, 8.3.3

Y. Yaz, C.-S. Foo, S. Winkler, K.-H. Yap, G. Piliouras, V. Chandrasekhar, et al. The unusual effectiveness of averaging in gan training. In *International Conference on Learning Representations*, 2018. 7.2

W. Yin and H. Schutze. Multichannel variable-size convolution for sentence classification. *Proc.*

of CONLL, 2015. **??**, 3.5.1.2

S. Young, M. Gašić, B. Thomson, and J. D. Williams. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013. 4.1.5

J. Yu, M.-S. Yang, and E. S. Lee. Sample-weighted clustering methods. *Computers & mathematics with applications*, 62(5):2200–2208, 2011. 2.3.1.2

L. Yu, W. Zhang, J. Wang, and Y. Yu. SeqGAN: sequence generative adversarial nets with policy gradient. In *AAAI*, 2017a. 4.1.1, 4.1.2

L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017b. (document), 4.2.1, 4.2.5, 7.2, 7.4.2, 8.1, 8.2, 8.2.1, **??**, 8.3, 8.3.2

W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014. (document), 8.3.2, **??**, 8.3

M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 3.5.1.1

A. Zellner. Optimal information processing and bayes's theorem. *The American Statistician*, 42 (4):278–280, 1988. 2.1.2

S. Zhai, Y. Cheng, R. Feris, and Z. Zhang. Generative adversarial networks as variational training of energy based models. *arXiv preprint arXiv:1611.01799*, 2016. 6.4.2

Y. Zhang, Z. Gan, and L. Carin. Generating text via adversarial training. In *NeurIPS Workshop on Adversarial Training*, 2016a. 4.1.1, 4.1.2

Y. Zhang, S. Roller, and B. Wallace. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. *Proc. of NAACL*, 2016b. **??**

Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin. Adversarial feature matching for text generation. *arXiv preprint arXiv:1706.03850*, 2017. (document), 8.1

Z. Zhang, S. Ren, S. Liu, J. Wang, P. Chen, M. Li, M. Zhou, and E. Chen. Style transfer as unsupervised machine translation. *arXiv preprint arXiv:1808.07894*, 2018. **??**

J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016. 4.2.5, 6.4.2, 7.1

T. Zhao, R. Zhao, and M. Eskenazi. Learning discourse-level diversity for neural dialog models using conditional variational autoencoders. In *ACL*, 2017. URL `https://github.com/snakeztc/NeuralDialog-CVAE`. (document), 8.2, 8.3.1

Z. Zheng, J. Oh, and S. Singh. On learning intrinsic rewards for policy gradient methods. In *NeurIPS*, 2018. 2.3.3, 2.3.3, 5.1, 5.3

C. Zhou and G. Neubig. Multi-space variational encoder-decoders for semi-supervised labeled sequence transduction. In *ACL*, 2017. 4.1.2

Z. Zhou, Y. Song, L. Yu, H. Wang, J. Liang, W. Zhang, Z. Zhang, and Y. Yu. Understanding the effectiveness of lipschitz-continuity in generative adversarial nets. *arXiv preprint arXiv:1807.00751*, 2018. 7.2

Z. Zhou, J. Liang, Y. Song, L. Yu, H. Wang, W. Zhang, Y. Yu, and Z. Zhang. Lipschitz generative adversarial nets. *arXiv preprint arXiv:1902.05687*, 2019. 7.1, 7.2, 7.3.4

J. Zhu and E. P. Xing. Maximum entropy discrimination markov networks. *Journal of Machine Learning Research*, 10(11), 2009. 2.1.4

J. Zhu, N. Chen, and E. P. Xing. Bayesian inference with posterior regularization and applications to infinite latent svms. *JMLR*, 15(1):1799–1847, 2014. 2.1.3, 2.1.3, 3.2

J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pages 597–613. Springer, 2016. 4.1.1

J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017. 4.2.1, 4.2.2, 4.2.5

W. Zhu, Z. Hu, and E. Xing. Text infilling. *arXiv preprint arXiv:1901.00158*, 2019. 8.3.1

B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 6.1, 6.2, 6.3.2.2

G. Zweig and C. J. Burges. The microsoft research sentence completion challenge. Technical report, Citeseer, 2011. 6.5.2