Deep Generative Modeling with Applications
in Semi-Supervised Learning

Zhilin Yang

August 10

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Ruslan Salakhutdinov (Co-Chair), Carnegie Mellon University
William W. Cohen (Co-Chair), Carnegie Mellon University
Graham Neubig, Carnegie Mellon University
Jason Weston, Facebook AI Research

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

Copyright © 2018 Zhilin Yang
Abstract

The ultimate goal of generative modeling is to model the probability of the world, either implicitly or explicitly. In practice, researchers devise models to estimate the probability of data, often unlabeled in its natural form, such as text corpora and images. Generative modeling not only serves as a bridge towards characterizing and understanding the world from a probabilistic perspective, but also has a benefit of learning transferable features from unlabeled data. This thesis proposes novel deep learning architectures for generative modeling, along with semi-supervised learning algorithms that leverage generative modeling on unlabeled data to improve performance on downstream tasks.

Specifically, the thesis consists of two parts—better architectures to improve generative modeling, and applications of generative modeling in semi-supervised learning. In the first part, we identify an expressiveness bottleneck of prior neural language models, and propose a high-rank language model called the *Mixture of Softmaxes* (MoS) to break through such bottleneck. We later propose a faster high-rank language model that trains faster than MoS while maintaining the capacity to break the bottleneck. In the second part, we present four semi-supervised learning algorithms based on generative approaches, including generating low-density adversarial samples, generating natural language questions given the context, generating random walk paths on a graph, and language modeling.
## Contents

1 Introduction .............................................. 1

2 Generative Modeling for Natural Language .................. 3
   2.1 A High-Rank Language Model (Completed Work, ICLR 2018) .......... 3
      2.1.1 Motivation ........................................... 3
      2.1.2 Approach and Analysis ................................ 4
      2.1.3 Experiments ............................................ 8
   2.2 A Faster High-Rank Language Model (Proposed Work) ................. 17
      2.2.1 Motivation ............................................ 17
      2.2.2 Proposed Approach .................................... 17
      2.2.3 Preliminary Results ................................... 18

3 Semi-Supervised Learning by Generative Modeling .......... 19
   3.1 Semi-Supervised Learning by GANs (Completed Work, NIPS 2017) .... 19
      3.1.1 Motivation ............................................ 19
      3.1.2 Theoretical Analysis ................................... 20
      3.1.3 Case Study on Synthetic Data .......................... 22
      3.1.4 Approach ............................................... 24
      3.1.5 Experiments ............................................ 27
   3.2 Semi-Supervised QA by Generating Questions (Completed Work, ACL 2017) 29
      3.2.1 Motivation ............................................ 29
      3.2.2 Problem Definition ..................................... 30
      3.2.3 Approach ............................................... 30
      3.2.4 Experiments ............................................ 35
   3.3 Semi-Supervised Learning on Graphs (Completed Work, ICML 2016) .... 40
      3.3.1 Motivation ............................................ 40
      3.3.2 Background ............................................. 40
      3.3.3 Approach ............................................... 42
      3.3.4 Experiments ............................................ 46
   3.4 Learning Transferable Graph Structures by Language Modeling (Proposed Work) 51
      3.4.1 Motivation ............................................ 51
      3.4.2 Proposed Approach .................................... 52
      3.4.3 Preliminary Results ................................... 56
List of Figures

2.1 Cumulative percentage of normalized singulars given a value in [0, 1]. ........................................ 12

3.1 Labeled and unlabeled data are denoted by cross and point respectively, and different 
colors indicate classes ................................................................. 23

3.2 Left: Classification decision boundary, where the white line indicates true-fake bound- 
ary; Right: True-Fake decision boundary ........................................ 23

3.3 Feature space at convergence .................................................... 23

3.4 Left: Blue points are generated data, and the black shadow indicates unlabeled data. 
Middle and right can be interpreted as above .................................. 23

3.5 Comparing images generated by FM and our model. FM generates collapsed samples, 
while our model generates diverse “bad” samples. ................................. 27

3.6 Model architecture and training. Red boxes denote the modules being updated. “d_true” 
and “d_gen” are two domain tags. D is the discriminative model and G is the generative 
model. The objectives for the three cases are all to minimize the cross entropy loss of 
the answer chunks ................................................................. 33

3.7 Comparison of discriminator training loss $-J(U_G; d_{gen}, D)$ on generated QA pairs. 
The lower the better. MLE refers to questions generated by maximum likelihood train- 
ing, and RL refers to questions generated by reinforcement learning ................. 37

3.8 An example of sampling from context distribution $p(i, c, \gamma)$ when $\gamma = 1$ and 
$d = 2$. In circles, $+1$ denotes positive instances, $-1$ denotes negative instances, 
and ? denotes unlabeled instances. If $\text{random} < r_2$, we first sample a random 
walk $2 \rightarrow 1 \rightarrow 4 \rightarrow 6$, and then sample two nodes in the random walk within 
distance $d$. If $\text{random} \geq r_2$, we sample two instances with the same labels. .... 43

3.9 Network architecture: transductive v.s. inductive. Each dotted arrow represents a 
feed-forward network with an arbitrary number of layers (we use only one layer 
in our experiments). Solid arrows denote direct connections ............................. 45

3.10 t-SNE Visualization of embedding spaces on the Cora dataset. Each color de- 
notes a class ................................................................. 50

3.11 Traditional transfer learning versus our new transfer learning framework. Instead of 
transferring features, we transfer the graphs output by a network. The graphs are multi- 
plied by task-specific features (e.g. embeddings or hidden states) to produce structure- 
aware features ................................................................. 53
3.12 Overview of our approach GLoMo. During the unsupervised learning phase, the feature predictor and the graph predictor are jointly trained to perform context prediction. During the transfer phase, the graph predictor is frozen and used to extract graphs for the downstream tasks. An RNN decoder is applied to all positions in the feature predictor, but we only show the one at position “A” for simplicity. “Select one” means the graphs can be transferred to any layer in the downstream task model. “FF” refers to feed-forward networks. The graphs output by the graph predictor are used as the weights in the “weighted sum” operation (see Eq. 3.13).
List of Tables

2.1 Single model perplexity on validation and test sets on Penn Treebank. Baseline results are obtained from Merity et al. [53] and Krause et al. [44]. † indicates using dynamic evaluation. ................................................................. 9

2.2 Single model perplexity over WikiText-2. Baseline results are obtained from Merity et al. [53] and Krause et al. [44]. † indicates using dynamic evaluation. ................................................................. 10

2.3 Perplexity comparison on 1B word dataset. Train perplexity is the average of the last 4,000 updates. .............................................................................................................. 10

2.4 Evaluation scores on Switchboard. .................................................................................. 11

2.5 Ablation study on Penn Treebank and WikiText-2 without finetuning or dynamical evaluation. .............................................................................................................. 11

2.6 Empirical expected pairwise KLD on PTB. ................................................................. 12

2.7 Rank comparison on PTB. To ensure comparable model sizes, the embedding sizes of Softmax, MoC and MoS are 400, 280, 280 respectively. The vocabulary size, i.e., $M$, is 10,000 for all models. .............................................................................................................. 13

2.8 Empirical rank and test perplexity on PTB with different number of Softmaxes. .... 13

2.9 BPC comparison on text8. For MoS, “-n” indicates using $n$ mixtures. “hid” and “emb” denote the hidden size and embedding size respectively. .................................................. 13

2.10 Training time slowdown compared to Softmax. MoS-$K$ means using $K$ mixture components. “bs” indicates Softmax and MoS use the same batch sizes on one GPU. “best-1” and “best-3” refer to the settings where Softmax and MoS obtain their own best perplexity, with 1 and 3 GPUs respectively. ................................................................. 14

2.11 Comparison of next-token prediction on Penn Treebank test data. N stands for a number as the result of preprocessing [55]. The context shown only includes the previous sentence and the current sentence the prediction step resides in. ................. 16

2.12 Perplexity and training time comparison on Penn Treebank word-level language modeling. $Ppl$ is perplexity. “MoS-$K$” denotes MoS with $K$ mixture components. “Time-B” denotes the training time of the output layer with batch size $B$. ................................................................. 18

3.1 Comparison with state-of-the-art methods on three benchmark datasets. Only methods without data augmentation are included. * indicates using the same (small) discriminator architecture, † indicates using a larger discriminator architecture, and ‡ means self-ensembling. ................................................................................................. 27
3.2 Ablation study. **FM** is feature matching. **LD** is the low-density enforcement term in Eq. (3.3). **VI** and **PT** are two entropy maximization methods described in Section 3.1.4. **Ent** means the conditional entropy term in Eq. (3.5). **Max log-p** is the maximum log probability of generated samples, evaluated by a PixelCNN++ model. **10-quant** shows the 10-quantile of true image log probability. **Error** means the number of misclassified examples on MNIST, and error rate (%) on others.

3.3 Sampled generated questions given the paragraphs and the answers. **P** means paragraphs, **A** means answers, **GQ** means groundtruth questions, and **Q** means questions generated by our models. **MLE** refers to maximum likelihood training, and **RL** refers to reinforcement learning so as to maximize \( J(U_G, d_{true}, D) \). We truncate the paragraphs to only show tokens around the answer spans with a window size of 20.

3.4 Performance with various labeling rates, unlabeled data sizes \(|U|\), and methods. “Dev” denotes the development set, and “test” denotes the test set. **F1** and **EM** are two metrics.

3.5 Dataset statistics.

3.6 Accuracy on text classification. Upper rows are inductive methods and lower rows are transductive methods.

3.7 Recall@\(k\) on DIEL distantly-supervised entity extraction. Upper rows are inductive methods and lower rows are transductive methods. Results marked with \(*\) are taken from the original DIEL paper [7] with the same data splits.

3.8 Accuracy on NELL entity classification with labeling rates of 0.1, 0.01, and 0.001. Upper rows are inductive methods and lower rows are transductive methods.

3.9 Main results on the SQuAD dataset. Self-attention modules are included in all baseline models. All baseline methods are feature-based transfer learning methods, including ELMo and GloVe. Our methods combine graph-based transfer with feature-based transfer. Our graphs operate on various sets of features, including GloVe embeddings, ELMo embeddings, and RNN states.
Chapter 1

Introduction

Given some data sample $x$ such as a text corpus or an image, the approach of generative modeling aims to estimate the probability $p(x)$, either in an explicit form or defined by an implicit transformation from some prior.

By obtaining an accurate estimate of $p(x)$, generative modeling provides a way to characterize and understand the world from a probabilistic perspective. Concretely, a generative model describes how the world is created and developed, and thus has the ability to predict how objects will evolve in the world. For example, a language model enables predicting the future tokens given the history as context, which tests a variety of abilities including understanding natural language and abstracting the regularities underlying natural language. As a result, learning a better generative model is itself an important aspect of intelligence.

To this end, in Chapter 2, we propose our approaches towards building state-of-the-art language models. We identify an expressiveness bottleneck of prior neural language models, and propose a high-rank language model called the Mixture of Softmaxes (MoS) to break through such bottleneck. We later propose a faster high-rank language model that trains faster than MoS while maintaining the capacity to break the bottleneck.

In addition to being an objective itself, generative modeling also yields important benefits for downstream applications. While the generative modeling approaches discussed in Chapter 2 mainly focus on modeling the probability of unlabeled data $x$ in their natural forms, in contrast, many applications involve predicting a label $y$ based on $x$. Traditionally, one line of methods aim to directly model $p(x, y)$ and infer the conditional probability using $p(y|x) = \frac{p(x, y)}{\sum_{y'} p(x, y')}$. In this thesis, we adopt a more popular discriminative approach for downstream applications, which is to directly model the conditional distribution $p(y|x)$. However, even though the model that we eventually care about is a discriminative model, in many cases it is beneficial to train an additional generative model. This generative model can either be jointly trained with the discriminative model to better capture the data distribution, or be pre-trained on large-scale unlabeled data to learn transferable features. In both cases, the generative models are trained on large-scale unlabeled data to capture signals that have not been modeled by the discriminative ones. This results in a semi-supervised learning paradigm.

As concrete examples, in Chapter 3, we demonstrate applications of generative modeling to semi-supervised learning. We consider four scenarios: 1) semi-supervised classification by generating low-density adversarial samples, 2) semi-supervised question answering by generating
natural language questions given context, 3) semi-supervised learning on graphs by modeling the generation of random walk paths, and 4) generative pretraining using a language modeling objective to induce transferable graphical structured representations.

Overall, this thesis proposes state-of-the-art deep learning architectures and algorithms for generative modeling and establishes the importance and effectiveness of generative modeling to downstream tasks.
Chapter 2

Generative Modeling for Natural Language

Neural language models are among the most popular approaches for text generative modeling. Based on autoregressive factorization, the generative process defined by a neural language model is to generate one token at a time given the history as context. In this chapter, we identify an expressiveness bottleneck of prior neural language models, and propose a simple, effective method called the Mixture of Softmaxes to overcome such limitation. Moreover, we also describe another approach to speed up training while maintaining the ability to break the bottleneck.

2.1 A High-Rank Language Model (Completed Work, ICLR 2018)

2.1.1 Motivation

As a fundamental task in natural language processing, statistical language modeling has gone through significant development from traditional Ngram language models to neural language models in the last decade [5, 55, 59]. Despite the huge variety of models, as a density estimation problem, language modeling mostly relies on a universal auto-regressive factorization of the joint probability and then models each conditional factor using different approaches. Specifically, given a corpus of tokens $X = (X_1, \ldots, X_T)$, the joint probability $P(X)$ factorizes as $P(X) = \prod_t P(X_t | X_{<t}) = \prod_t P(X_t | C_t)$, where $C_t = X_{<t}$ is referred to as the context of the conditional probability hereafter.

Based on the factorization, recurrent neural networks (RNN) based language models achieve state-of-the-art results on various benchmarks [44, 51, 53]. A standard approach is to use a recurrent network to encode the context into a fixed size vector, which is then multiplied by the word embeddings [34, 66] using dot product to obtain the logits. The logits are consumed by the Softmax function to give a categorical probability distribution over the next token. In spite of the expressiveness of RNNs as universal approximators [74], an unclear question is whether the combination of dot product and Softmax is capable of modeling the conditional probability, which can vary dramatically with the change of the context.

In this work, we study the expressiveness of the aforementioned Softmax-based recurrent language models from a perspective of matrix factorization. We show that learning a Softmax-
based recurrent language model with the standard formulation is essentially equivalent to solving a matrix factorization problem. More importantly, due to the fact that natural language is highly context-dependent, the matrix to be factorized can be high-rank. This further implies that standard Softmax-based language models with distributed (output) word embeddings do not have enough capacity to model natural language. We call this the Softmax bottleneck.

2.1.2 Approach and Analysis

As discussed in Section 2.1.1, with the autoregressive factorization, language modeling can be reduced to modeling the conditional distribution of the next token $x$ given the context $c$. Though one might argue that a natural language allows an infinite number of contexts due to its compositionality [65], we proceed with our analysis by considering a finite set of possible contexts. The unboundedness of natural language does not affect our conclusions, which will be discussed later.

We consider a natural language as a finite set of pairs of a context and its conditional next-token distribution[1] $\mathcal{L} = \{(c_1, P^*(X|c_1)), \cdots, (c_N, P^*(X|c_N))\}$, where $N$ is the number of possible contexts. We assume $P^* > 0$ everywhere to account for errors and flexibility in natural language. Let $\{x_1, x_2, \cdots, x_M\}$ denote a set of $M$ possible tokens in the language $\mathcal{L}$. The objective of a language model is to learn a model distribution $P_\theta(X|C)$ parameterized by $\theta$ to match the true data distribution $P^*(X|C)$.

In this work, we study the expressiveness of the parametric model class $P_\theta(X|C)$. In other words, we are asking the following question: given a natural language $\mathcal{L}$, does there exist a parameter $\theta$ such that $P_\theta(X|c) = P^*(X|c)$ for all $c$ in $\mathcal{L}$?

We start by looking at a Softmax-based model class since it is widely used.

Softmax

The majority of parametric language models use a Softmax function operating on a context vector (or hidden state) $h_c$ and a word embedding $w_x$ to define the conditional distribution $P_\theta(x|c)$. More specifically, the model distribution is usually written as

$$P_\theta(x|c) = \frac{\exp h_c^\top w_x}{\sum_{x'} \exp h_c^\top w_{x'}}$$

(2.1)

where $h_c$ is a function of $c$, and $w_x$ is a function of $x$. Both functions are parameterized by $\theta$. Both the context vector $h_c$ and the word embedding $w_x$ have the same dimension $d$. The dot product $h_c^\top w_x$ is called a logit.

To help discuss the expressiveness of Softmax, we define three matrices:

$$H_\theta = \begin{bmatrix} h_{c1}^\top \\ h_{c2}^\top \\ \vdots \\ h_{cN}^\top \end{bmatrix}; \quad W_\theta = \begin{bmatrix} w_{x1}^\top \\ w_{x2}^\top \\ \vdots \\ w_{xM}^\top \end{bmatrix}; \quad A = \begin{bmatrix} \log P^*(x_1|c_1), \log P^*(x_2|c_1) & \cdots & \log P^*(x_M|c_1) \\ \log P^*(x_1|c_2), \log P^*(x_2|c_2) & \cdots & \log P^*(x_M|c_2) \\ \vdots & \vdots & \vdots \\ \log P^*(x_1|c_N), \log P^*(x_2|c_N) & \cdots & \log P^*(x_M|c_N) \end{bmatrix}$$

\(^1\)We use capital letters for variables and small letters for constants.
where \( H_\theta \in \mathbb{R}^{N \times d}, W_\theta \in \mathbb{R}^{M \times d}, A \in \mathbb{R}^{N \times M} \), and the rows of \( H_\theta, W_\theta, \) and \( A \) correspond to context vectors, word embeddings, and log probabilities of the true data distribution respectively. We use the subscript \( \theta \) because \((H_\theta, W_\theta)\) is effectively a function indexed by the parameter \( \theta \), from the joint function family \( \mathcal{U} \). Concretely, \( H_\theta \) is implemented as deep neural networks, such as a recurrent network, while \( W_\theta \) is instantiated as an embedding lookup.

We further specify a set of matrices formed by applying row-wise shift to \( A \)

\[
F(A) = \{A + \Lambda J_{N,M}|\Lambda \text{ is diagonal and } \Lambda \in \mathbb{R}^{N \times N}\},
\]

where \( J_{N,M} \) is an all-ones matrix with size \( N \times M \). Essentially, the row-wise shift operation adds an arbitrary real number to each row of \( A \). Thus, \( F(A) \) is an infinite set. Notably, the set \( F(A) \) has two important properties (see the supplementary materials for the proof), which are key to our analysis.

**Property 1.** For any matrix \( A' \), \( A' \in F(A) \) if and only if \( \text{Softmax}(A') = P^* \). In other words, \( F(A) \) defines the set of all possible logits that correspond to the true data distribution.

**Property 2.** For any \( A_1 \neq A_2 \in F(A) \), \( |\text{rank}(A_1) - \text{rank}(A_2)| \leq 1 \). In other words, all matrices in \( F(A) \) have similar ranks, with the maximum rank difference being 1.

Based on the Property 1 of \( F(A) \), we immediately have the following Lemma.

**Lemma 1.** Given a model parameter \( \theta \), \( H_\theta W_\theta^\top \in F(A) \) if and only if \( P_\theta(X|c) = P^*(X|c) \) for all \( c \) in \( \mathcal{L} \).

Now the expressiveness question becomes: does there exist a parameter \( \theta \) and \( A' \in F(A) \) such that

\[
H_\theta W_\theta^\top = A'.
\]

This is essentially a matrix factorization problem. We want the model to learn matrices \( H_\theta \) and \( W_\theta \) that are able to factorize some matrix \( A' \in F(A) \). First, note that for a valid factorization to exist, the rank of \( H_\theta W_\theta^\top \) has to be at least as large as the rank of \( A' \). Further, since \( H_\theta \in \mathbb{R}^{N \times d} \) and \( W_\theta \in \mathbb{R}^{M \times d} \), the rank of \( H_\theta W_\theta^\top \) is strictly upper bounded by the embedding size \( d \). As a result, if \( d \geq \text{rank}(A') \), a universal approximator can theoretically recover \( A' \). However, if \( d < \text{rank}(A') \), no matter how expressive the function family \( \mathcal{U} \) is, no \((H_\theta, W_\theta)\) can even theoretically recover \( A' \). We summarize the reasoning above as follows (see the supplementary materials for the proof).

**Proposition 1.** Given that the function family \( \mathcal{U} \) is a universal approximator, there exists a parameter \( \theta \) such that \( P_\theta(X|c) = P^*(X|c) \) for all \( c \) in \( \mathcal{L} \) if and only if \( d \geq \min_{A' \in F(A)} \text{rank}(A') \).

Combining Proposition 1 with the Property 2 of \( F(A) \), we are now able to state the Softmax Bottleneck problem formally.

**Corollary 1. (Softmax Bottleneck)** If \( d < \text{rank}(A') - 1 \), for any function family \( \mathcal{U} \) and any model parameter \( \theta \), there exists a context \( c \) in \( \mathcal{L} \) such that \( P_\theta(X|c) \neq P^*(X|c) \).

The above corollary indicates that when the dimension \( d \) is too small, Softmax does not have the capacity to express the true data distribution. Clearly, this conclusion is not restricted to a finite language \( \mathcal{L} \). When \( \mathcal{L} \) is infinite, one can always take a finite subset and the Softmax bottleneck still exists. Next, we discuss why the Softmax bottleneck is an issue by presenting our hypothesis that \( A \) is high-rank for natural language.
Hypothesis: Natural Language is High-Rank

We hypothesize that for a natural language \( \mathcal{L} \), the log probability matrix \( A \) is a high-rank matrix. It is difficult (if possible) to rigorously prove this hypothesis since we do not have access to the true data distribution of a natural language. However, it is suggested by the following intuitive reasoning and empirical observations:

- Natural language is highly context-dependent \([54]\). For example, the token “north” is likely to be followed by “korea” or “korean” in a news article on international politics, which however is unlikely in a textbook on U.S. domestic history. We hypothesize that such subtle context dependency should result in a high-rank matrix \( A \).

- If \( A \) is low-rank, it means humans only need a limited number (e.g. a few hundred) of bases, and all semantic meanings can be created by (potentially) negating and (weighted) averaging these bases. However, it is hard to find a natural concept in linguistics and cognitive science that corresponds to such bases, which questions the existence of such bases. For example, semantic meanings might not be those bases since a few hundred meanings may not be enough to cover everyday meanings, not to mention niche meanings in specialized domains.

- Empirically, our high-rank language model outperforms conventional low-rank language models on several benchmarks, as shown in Section \([3.3.4]\). We also provide evidences in Section \([2.1.3]\) to support our hypothesis that learning a high-rank language model is important.

Given the hypothesis that natural language is high-rank, it is clear that the Softmax bottleneck limits the expressiveness of the models. In practice, the embedding dimension \( d \) is usually set at the scale of \( 10^2 \), while the rank of \( A \) can possibly be as high as \( M \) (at the scale of \( 10^5 \)), which is orders of magnitude larger than \( d \). Softmax is effectively learning a low-rank approximation to \( A \), and our experiments suggest that such approximation loses the ability to model context dependency, both qualitatively and quantitatively (Cf. Section \([3.3.4]\)).

Easy Fixes?

Identifying the Softmax bottleneck immediately suggests some possible “easy fixes”. First, as considered by a lot of prior work, one can employ a non-parametric model, namely an Ngram model \([43]\). Ngram models are not constrained by any parametric forms so it can universally approximate any natural language, given enough parameters. Second, it is possible to increase the dimension \( d \) (e.g., to match \( M \)) so that the model can express a high-rank matrix \( A \).

However, these two methods increase the number of parameters dramatically, compared to using a low-dimensional Softmax. More specifically, an Ngram needs \((N \times M)\) parameters in order to express \( A \), where \( N \) is potentially unbounded. Similarly, a high-dimensional Softmax requires \((M \times M)\) parameters for the word embeddings. Increasing the number of model parameters easily leads to overfitting. In past work, \([43]\) used back-off to alleviate overfitting. Moreover, as deep learning models were tuned by extensive hyper-parameter search, increasing the dimension \( d \) beyond several hundred is not helpful\([44, 51, 53]\).\(^2\)

\(^2\)This is also confirmed by our preliminary experiments.
Clearly there is a tradeoff between expressiveness and generalization on language modeling. Naively increasing the expressiveness hurts generalization. Below, we introduce an alternative approach that increases the expressiveness without exploding the parametric space.

**Mixture of Softmaxes: A High-Rank Language Model**

We propose a high-rank language model called Mixture of Softmaxes (MoS) to alleviate the Softmax bottleneck issue. MoS formulates the conditional distribution as

\[
P_\theta(x|c) = \sum_{k=1}^{K} \pi_{c,k} \exp h_{c,k}^T w_x \quad \text{s.t.} \quad \sum_{k=1}^{K} \pi_{c,k} = 1
\]

where \(\pi_{c,k}\) is the prior or mixture weight of the \(k\)-th component, and \(h_{c,k}\) is the \(k\)-th context vector associated with context \(c\). In other words, MoS computes \(K\) Softmax distributions and uses a weighted average of them as the next-token probability distribution. Similar to prior work on recurrent language modeling [44, 51, 53], we first apply a stack of recurrent layers on top of \(X\) to obtain a sequence of hidden states \((g_1, \ldots, g_T)\). The prior and the context vector for context \(c_t\) are parameterized as \(\pi_{c_t,k} = \exp w_{\pi,k}^T g_t \sum_{k'=1}^{K} \exp w_{\pi,k'}^T g_t\) and \(h_{c_t,k} = \tanh(W_{h,k} g_t)\) where \(w_{\pi,k}\) and \(W_{h,k}\) are model parameters.

Our method is simple and easy to implement, and has the following advantages:

- **Improved expressiveness** (compared to Softmax). MoS is theoretically more (or at least equally) expressive compared to Softmax given the same dimension \(d\). This can be seen by the fact that MoS with \(K = 1\) is reduced to Softmax. More importantly, MoS effectively approximates \(A\) by

\[
\hat{A}_{\text{MoS}} = \log \sum_{k=1}^{K} \Pi_k \exp(H_{\theta,k} W_{\theta}^T)
\]

where \(\Pi_k\) is an \((N \times N)\) diagonal matrix with elements being the prior \(\pi_{c,k}\). Because \(\hat{A}_{\text{MoS}}\) is a nonlinear function (log_sum_exp) of the context vectors and the word embeddings, \(\hat{A}_{\text{MoS}}\) can be arbitrarily high-rank. As a result, MoS does not suffer from the rank limitation, compared to Softmax.

- **Improved generalization** (compared to Ngram). Ngram models and high-dimensional Softmax (Cf. Section 2.1.2) improve the expressiveness but do not generalize well. In contrast, MoS does not have a generalization issue due to the following reasons. First, MoS defines the following generative process: a discrete latent variable \(k\) is first sampled from \(\{1, \ldots, K\}\), and then the next token is sampled based on the \(k\)-th Softmax component. By doing so we introduce an inductive bias that the next token is generated based on a latent discrete decision (e.g., a topic), which is often safe in language modeling [8]. Second, since \(\hat{A}_{\text{MoS}}\) is defined by a nonlinear function and not restricted by the rank bottleneck, in practice it is possible to reduce \(d\) to compensate for the increase of model parameters brought by the mixture structure. As a result, MoS has a similar model size compared to Softmax and thus is not prone to overfitting.
Mixture of Contexts: A Low-Rank Baseline

Another possible approach is to directly mix the context vectors (or logits) before taking the Softmax, rather than mixing the probabilities afterwards as in MoS. Specifically, the conditional distribution is parameterized as

$$P_\theta(x|c) = \frac{\exp \left( \sum_{k=1}^{K} \pi_{c,k} h_{c,k}^\top w x \right)}{\sum_{x'} \exp \left( \sum_{k=1}^{K} \pi_{c,k} h_{c,k}^\top w x' \right)} = \frac{\exp \left( \sum_{k=1}^{K} \pi_{c,k} h_{c,k}^\top w x \right)}{\sum_{x'} \exp \left( \sum_{k=1}^{K} \pi_{c,k} h_{c,k}^\top w x' \right)}$$

(2.2)

where $h_{c,k}$ and $\pi_{c,k}$ share the same parameterization as in MoS. Despite its superficial similarity to MoS, this model, which we refer to as mixture of contexts (MoC), actually suffers from the same rank limitation problem as Softmax. This can be easily seen by defining $h'_{c} = \sum_{k=1}^{K} \pi_{c,k} h_{c,k}$, which turns the MoC parameterization (2.2) into $P_\theta(x|c) = \frac{\exp h'^\top c w x}{\sum_{x'} \exp h'^\top c w x'}$. Note that this is equivalent to the Softmax parameterization (2.1). Thus, performing mixture in the feature space can only make the function family $\mathcal{U}$ more expressive, but does not change the fact that the rank of $H_\theta W_\theta^\top$ is upper bounded by the embedding dimension $d$. In our experiments, we implement MoC as a baseline and compare it experimentally to MoS.

2.1.3 Experiments

Main Results

We conduct a series of experiments with the following settings:

- Following previous work [44, 51, 53], we evaluate the proposed MoS model on two widely used language modeling datasets, namely Penn Treebank (PTB) [55] and WikiText-2 (WT2) [52] based on perplexity. For fair comparison, we closely follow the regularization and optimization techniques introduced by Merity et al. [53]. We heuristically and manually search hyper-parameters for MoS based on the validation performance while limiting the model size (see the supplementary materials for our hyper-parameters).

- To investigate whether the effectiveness of MoS can be extended to even larger datasets, we conduct an additional language modeling experiment on the 1B Word dataset [12]. Specifically, we lower-case the text and choose the top 100K tokens as the vocabulary. A standard neural language model with 2 layers of LSTMs followed by a Softmax output layer is used as the baseline. Again, the network size of MoS is adjusted to ensure a comparable number of parameters. Notably, dropout was not used, since we found it not helpful to either model. For training, we use all of the 100 training shards. For validation, we use two shards from the heldout set, namely [heldout-00, heldout-10]. For test, we use another three shards from the heldout set, namely [heldout-20, heldout-30, heldout-40].

- To show that the MoS is a generic structure that can be used to model other context-dependent distributions, we additionally conduct experiments in the dialog domain. We use the Switchboard dataset [25] preprocessed by Zhao et al. [105] to train a Seq2Seq [79] model with

https://github.com/snakeztc/NeuralDialog-CVAE/tree/master/data
MoS added to the decoder RNN. Then, a Seq2Seq model using Softmax and another one augmented by MoC with comparable parameter sizes are used as baselines. For evaluation, we include both the perplexity and the precision/recall of Smoothed Sentence-level BLEU, as suggested by Zhao et al. \cite{105}. When generating responses, we use beam search with beam size 10, restrict the maximum length to 30, and retain the top-5 responses.

The language modeling results on PTB and WT2 are presented in Table 2.1 and Table 2.2 respectively. With a comparable number of parameters, MoS outperforms all baselines with or without dynamic evaluation, and substantially improves over the current state of the art, by up to 3.6 points in perplexity.

The improvement on the large-scale dataset is even more significant. As shown in Table 2.3, MoS outperforms Softmax by over 5.6 points in perplexity. It suggests the effectiveness of MoS is not limited to small datasets where many regularization techniques are used. Note that with limited computational resources, we didn’t tune the hyper-parameters for MoS.

Further, the experimental results on Switchboard are summarized in Table 2.4. Clearly, on all metrics, MoS outperforms MoC and Softmax, showing its general effectiveness.

\*\*The numbers are not directly comparable to \cite{105} since their Seq2Seq implementation and evaluation scripts are not publicly available.
Table 2.2: Single model perplexity over WikiText-2. Baseline results are obtained from Merity et al. [53] and Krause et al. [44]. † indicates using dynamic evaluation.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inan et al. [34] – Variational LSTM + augmented loss</td>
<td>28M</td>
<td>91.5</td>
<td>87.0</td>
</tr>
<tr>
<td>Grave et al. [27] – LSTM + continuous cache pointer†</td>
<td>-</td>
<td>-</td>
<td>68.9</td>
</tr>
<tr>
<td>Melis et al. [51] – 2-layer skip connection LSTM</td>
<td>24M</td>
<td>69.1</td>
<td>65.9</td>
</tr>
<tr>
<td>Merity et al. [53] – AWD-LSTM w/o finetune</td>
<td>33M</td>
<td>69.1</td>
<td>66.0</td>
</tr>
<tr>
<td>Merity et al. [53] – AWD-LSTM</td>
<td>33M</td>
<td>68.6</td>
<td>65.8</td>
</tr>
<tr>
<td>Ours – AWD-LSTM-MoS w/o finetune</td>
<td>35M</td>
<td>66.01</td>
<td>63.33</td>
</tr>
<tr>
<td>Ours – AWD-LSTM-MoS</td>
<td>35M</td>
<td>63.88</td>
<td>61.45</td>
</tr>
<tr>
<td>Merity et al. [53] – AWD-LSTM + continuous cache pointer†</td>
<td>33M</td>
<td>53.8</td>
<td>52.0</td>
</tr>
<tr>
<td>Krause et al. [44] – AWD-LSTM + dynamic evaluation†</td>
<td>33M</td>
<td>46.4</td>
<td>44.3</td>
</tr>
<tr>
<td>Ours – AWD-LSTM-MoS + dynamical evaluation†</td>
<td>35M</td>
<td>42.41</td>
<td>40.68</td>
</tr>
</tbody>
</table>

Table 2.3: Perplexity comparison on 1B word dataset. Train perplexity is the average of the last 4,000 updates.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softmax</td>
<td>119M</td>
<td>41.47</td>
<td>43.86</td>
<td>42.77</td>
</tr>
<tr>
<td>MoS</td>
<td>113M</td>
<td><strong>36.39</strong></td>
<td><strong>38.01</strong></td>
<td><strong>37.10</strong></td>
</tr>
</tbody>
</table>

Ablation Study

To further verify the improvement shown above does come from the MoS structure rather than adding another hidden layer or finding a particular set of hyper-parameters, we conduct an ablation study on both PTB and WT2. Firstly, we compare MoS with an MoC architecture with the same number of layers, hidden sizes, and embedding sizes, which thus has the same number of parameters. In addition, we adopt the hyper-parameters used to obtain the best MoS model (denoted as MoS hyper-parameters), and train a baseline AWD-LSTM. To avoid distractive factors and save computational resources, all ablative experiments excluded the use of finetuning and dynamic evaluation.

The results are shown in Table 2.5. Compared to the vanilla AWD-LSTM, though being more expressive, MoC performs only better on PTB, but worse on WT2. It suggests that simply adding another hidden layer or employing a mixture structure in the feature space does not guarantee a better performance. On the other hand, training AWD-LSTM using MoS hyper-parameters severely hurts the performance, which rules out hyper-parameters as the main source of improvement.

Verify the Role of Rank

While the study above verifies that MoS is the key to achieving the state-of-the-art performance, it is still not clear whether the superiority of MoS comes from its potential high rank, as suggested
### Table 2.4: Evaluation scores on Switchboard.

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
<th>BLEU-1 prec</th>
<th>BLEU-1 recall</th>
<th>BLEU-2 prec</th>
<th>BLEU-2 recall</th>
<th>BLEU-3 prec</th>
<th>BLEU-3 recall</th>
<th>BLEU-4 prec</th>
<th>BLEU-4 recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq2Seq-Softmax</td>
<td>34.657</td>
<td>0.249</td>
<td>0.188</td>
<td>0.193</td>
<td>0.151</td>
<td>0.168</td>
<td>0.133</td>
<td>0.141</td>
<td>0.111</td>
</tr>
<tr>
<td>Seq2Seq-MoC</td>
<td>33.291</td>
<td>0.259</td>
<td>0.198</td>
<td>0.202</td>
<td>0.159</td>
<td>0.176</td>
<td>0.140</td>
<td>0.148</td>
<td>0.117</td>
</tr>
<tr>
<td>Seq2Seq-MoS</td>
<td>32.727</td>
<td>0.272</td>
<td>0.206</td>
<td>0.213</td>
<td>0.166</td>
<td>0.185</td>
<td>0.146</td>
<td>0.157</td>
<td>0.123</td>
</tr>
</tbody>
</table>

Table 2.5: Ablation study on Penn Treebank and WikiText-2 without finetuning or dynamical evaluation.

<table>
<thead>
<tr>
<th>Model</th>
<th>PTB Validation</th>
<th>PTB Test</th>
<th>WT2 Validation</th>
<th>WT2 Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWD-LSTM-MoS</td>
<td>58.08</td>
<td>55.97</td>
<td>66.01</td>
<td>63.33</td>
</tr>
<tr>
<td>AWD-LSTM-MoC</td>
<td>59.82</td>
<td>57.55</td>
<td>68.76</td>
<td>65.98</td>
</tr>
<tr>
<td>AWD-LSTM (Merit et al. [53] hyper-parameters)</td>
<td>61.49</td>
<td>58.95</td>
<td>68.73</td>
<td>65.40</td>
</tr>
<tr>
<td>AWD-LSTM (MoS hyper-parameters)</td>
<td>78.86</td>
<td>74.86</td>
<td>72.73</td>
<td>69.18</td>
</tr>
</tbody>
</table>

by our theoretical analysis in Section 2.1.2. In the sequel, we take steps to verify this hypothesis.

- Firstly, we verify that MoS does induce a high-rank log-probability matrix empirically, while MoC and Softmax fail. On the validation or test set of PTB with tokens $X = \{X_1, \ldots, X_T\}$, we compute the log probabilities $\{\log P(X_i \mid X_{<i}) \in \mathbb{R}^M\}_{i=1}^T$ for each token using all three models. Then, for each model, we stack all $T$ log-probability vectors into a $T \times M$ matrix, resulting in $\hat{A}_{MoS}$, $\hat{A}_{MoC}$ and $\hat{A}_{Softmax}$. Theoretically, the number of non-zero singular values of a matrix is equal to its rank. However, performing singular value decomposition of real valued matrices using numerical approaches often encounter roundoff errors. Hence, we adopt the expected roundoff error suggested by Press [67] when estimating the ranks of $\hat{A}_{MoS}$, $\hat{A}_{MoC}$ and $\hat{A}_{Softmax}$.

The estimated ranks are shown in Table 2.7. As predicted by our theoretical analysis, the matrix ranks induced by Softmax and MoC are both limited by the corresponding embedding sizes. By contrast, the matrix rank obtained from MoS does not suffer from this constraint, almost reaching full rank ($M = 10000$).

Since there can be roundoff mistakes, a less error-prone approach is to directly study the distribution of singular values. Specifically, if more singular values have relatively larger magnitude, the rank of the matrix tends to be higher. Motivated from this intuition, we visualize the distribution of the singular values. To account for the different magnitudes of singular values from different models, we first normalize all singular values to $[0, 1]$. Then, we plot the cumulative percentage of normalized singular values, i.e., percentage of normalized singular values below a threshold, in Figure 2.1. As we can see, most of the singular values of Softmax and MoC concentrate on an area with very low values. In comparison, the concentration area of the MoS singular values is not only several orders larger, but also spans a much wider region. Intuitively, MoS utilizes the corresponding singular vectors to capture a larger and more diverse set of contexts.
What’s more, another indicator of high rank is that the model can precisely capture the nuance of difference contexts. If a model can better capture the distinctions among contexts, we expect the next-step conditional distributions to be less similar to each on average. Based on this intuition, we use the expected pairwise Kullback–Leibler divergence (KLD), i.e., $\mathbb{E}_{c,c' \sim \mathcal{C}} \left[\text{KLD}(P(X \mid c) \| P(X \mid c'))\right]$ where $\mathcal{C}$ denotes all possible contexts, as another metric to evaluate the ranks of the three models (MoS, MoC and Softmax). Practically, we sample $c, c'$ from validation or test data of PTB to get the empirical estimations for the three models, which are shown in the right half of Table 2.6. As we expected, MoS achieves higher expected pairwise KLD, indicating its superiority in covering more contexts of the next-token distribution.

- Secondly, we show that, before reaching full rank, increasing the number of mixture components in MoS also increases the rank of the log-probability matrix, which in turn leads to improved performance (lower perplexity). Specifically, on PTB, with other hyper-parameters fixed as used in section 2.1.3, we vary the number of mixtures used in MoS and compare the corresponding empirical rank and test perplexity without finetuning. Table 2.8 summarizes the results. This clear positive correlation between rank and performance strongly supports the our theoretical analysis in section 2.1.2. Moreover, note that after reaching almost full rank (i.e., using 15 mixture components), further increasing the number of components degrades the performance due to overfitting (as we inspected the training and test perplexities).

- In addition, as performance improvement can often come from better regularization, we investigate whether MoS has a better, though unexpected, regularization effect compared to Softmax. We consider the 1B word dataset where overfitting is unlikely and no explicit regularization technique (e.g., dropout) is employed. As we can see from the left part of Table 2.6: Empirical expected pairwise KLD on PTB.

<table>
<thead>
<tr>
<th>Model</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softmax</td>
<td>4.869</td>
<td>4.763</td>
</tr>
<tr>
<td>MoC</td>
<td>4.955</td>
<td>4.864</td>
</tr>
<tr>
<td>MoS</td>
<td><strong>5.400</strong></td>
<td><strong>5.284</strong></td>
</tr>
</tbody>
</table>

Figure 2.1: Cumulative percentage of normalized singulars given a value in $[0, 1]$. 

![Cumulative percentage graph](image-url)
August 19, 2018
DRAFT

Table 2.7: Rank comparison on PTB. To ensure comparable model sizes, the embedding sizes of Softmax, MoC and MoS are 400, 280, 280 respectively. The vocabulary size, i.e., \( M \), is 10,000 for all models.

<table>
<thead>
<tr>
<th></th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softmax</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>MoC</td>
<td>280</td>
<td>280</td>
</tr>
<tr>
<td>MoS</td>
<td>9981</td>
<td>9981</td>
</tr>
</tbody>
</table>

Table 2.8: Empirical rank and test perplexity on PTB with different number of Softmaxes.

<table>
<thead>
<tr>
<th></th>
<th>#Softmax</th>
<th>Rank</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6467</td>
<td></td>
<td>58.62</td>
</tr>
<tr>
<td>5</td>
<td>8930</td>
<td></td>
<td>57.36</td>
</tr>
<tr>
<td>10</td>
<td>9973</td>
<td></td>
<td>56.33</td>
</tr>
<tr>
<td>15</td>
<td>9981</td>
<td></td>
<td>55.97</td>
</tr>
<tr>
<td>20</td>
<td>9981</td>
<td></td>
<td>56.17</td>
</tr>
</tbody>
</table>

2.3 MoS and Softmax achieve a similar generalization gap, i.e., the performance gap between the test set and the training set. It suggests both models have similar regularization effects. Meanwhile, MoS has a lower training perplexity compared to Softmax, indicating that the improvement of MoS results from improved expressiveness.

- The last evidence we provide is based on an inverse experiment. Empirically, we find that when Softmax does not suffer from a rank limitation, e.g., in character-level language modeling, using MoS will not improve the performance. Notice that for character-level language modeling (CharLM), the rank of the log-likelihood matrix is upper bounded by the vocabulary size, and CharLM usually has a very limited vocabulary (tens of characters). In this case, with the embedding size being hundreds in practice, Softmax is no longer a bottleneck in this task. Hence, we expect MoS to yield similar performance to Softmax on CharLM.

We conduct experiments of CharLM using the text8 dataset \[50\], which consists of 100M characters including only alphabetical characters and spaces derived from Wikipedia. We follow Mikolov et al. \[56\] and use the first 90M characters for training, the next 5M for validation and the final 5M for testing. The standard evaluation metric bit-per-character (BPC) is employed. We employ a 1-layer 1024-unit LSTM followed by Softmax as the baseline. For MoS, we consider 7 or 10 mixtures and reduce the hidden and/or embedding size to match the baseline capacity. When decreasing the hidden and/or embedding size, we either keep both the same, or make the hidden size relatively larger. The results are summarized in Table 2.9. Clearly, the Softmax and MoS obtain the same BPC on the test set and comparable BPC on the validation set, which well match our hypothesis. Since the only difference in word-level language mod-

Table 2.9: BPC comparison on text8. For MoS, “-n” indicates using \( n \) mixtures. “hid” and “emb” denote the hidden size and embedding size respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>(#Param</th>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softmax</td>
<td>(hid1024, emb1024)</td>
<td>8.42M</td>
<td>1.35</td>
<td>1.41</td>
</tr>
<tr>
<td>MoS-7</td>
<td>(hid910, emb510)</td>
<td>8.45M</td>
<td>1.35</td>
<td>1.40</td>
</tr>
<tr>
<td>MoS-7</td>
<td>(hid750, emb750)</td>
<td>8.45M</td>
<td>1.38</td>
<td>1.42</td>
</tr>
<tr>
<td>MoS-10</td>
<td>(hid860, emb452)</td>
<td>8.43M</td>
<td>1.35</td>
<td>1.41</td>
</tr>
<tr>
<td>MoS-10</td>
<td>(hid683, emb683)</td>
<td>8.43M</td>
<td>1.38</td>
<td>1.42</td>
</tr>
</tbody>
</table>

eling is the existence of the Softmax bottleneck, the distinct behavior of MoS again supports our hypothesis that it is solving the Softmax bottleneck problem.

**MoS computational time**

<table>
<thead>
<tr>
<th>Model</th>
<th>PTB/bs</th>
<th>PTB/best-1</th>
<th>WT2/bs</th>
<th>WT2/best-1</th>
<th>WT2/best-3</th>
<th>1B/bs</th>
<th>1B/best-1</th>
<th>1B/best-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softmax</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>MoS-5</td>
<td>1.2x</td>
<td>–</td>
<td>1.3x</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MoS-7</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>3.8x</td>
<td>5.7x</td>
<td>2.1x</td>
<td>–</td>
</tr>
<tr>
<td>MoS-10</td>
<td>1.6x</td>
<td>–</td>
<td>1.9x</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MoS-15</td>
<td>1.9x</td>
<td>2.8x</td>
<td>2.5x</td>
<td>6.4x</td>
<td>2.9x</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 2.10: Training time slowdown compared to Softmax. MoS-$K$ means using $K$ mixture components. “bs” indicates Softmax and MoS use the same batch sizes on one GPU. “best-1” and “best-3” refer to the settings where Softmax and MoS obtain their own best perplexity, with 1 and 3 GPUs respectively.

We evaluate the additional computational cost introduced by MoS. We consider two sets of controlled experiments. In the first set, we compare the training time of MoS and Softmax using the same batch sizes. In the second set, we compare the training time of two methods using the hyper-parameter settings that achieve the best performance for each model (i.e., the settings in Tables 2.1, 2.2, and 2.3). In both sets, we control two models to have comparable model sizes.

The results on the three datasets are shown in Table 2.10. Thanks to the efficiency of matrix multiplication on GPU, the computational wall time of MoS is actually sub-linear w.r.t. the number of Softmaxes $K$. In most settings, we observe a two to three times slowdown when using MoS. Specifically, the “bs” setting measures the computational cost introduced by MoS given enough memory, which is 1.9x, 2.5x, and 3.8x slowdown on PTB, WT2, and 1B respectively. The “best-1” setting is usually slower compared to “bs”, because a single batch does not fit into the memory of a single GPU using MoS, in which case we have to split one batch into multiple small ones, resulting in further slowdown. In this sense, the gap between “best-1” and “bs” measures the computational cost introduced due to the increase of memory consumed by MoS. The “best-3” alleviates this issue by using three GPUs, which allows larger-batch training for MoS. In this case, we reduce the computational cost to 2.9x on WT2 and 2.1x on 1B with our best performing model.

Note that the computational cost is closely related to the batch size, which is interleaved with optimization. Though how batch sizes affect optimization remains an open question and might be task dependent, we believe the “best-1” and “best-3” settings well reflect the actual computational cost brought by MoS on language modeling tasks.

**Qualitative analysis**

Since MoC shows a stronger performance than Softmax on PTB, the qualitative study focuses on the comparison between MoC and MoS. Concretely, given the same context (previous tokens), we search for prediction steps where MoS achieves lower negative log loss than MoC by a margin. We show some representative cases in Table 2.11 with the following observations:
• Comparing the first two cases, given the same preceding word “N”, MoS flexibly adjusts its top predictions based on the different topic quantities being discussed in the context. In comparison, MoC emits quite similar top choices regardless of the context, suggesting its inferiority in make context-dependent predictions.

• In the 3rd case, the context is about international politics, where country/region names are likely to appear. MoS captures this nuance well, and yields top choices that can be used to complete a country name given the immediate preceding word “south”. Similarly, in the 4th case, MoS is able to include “ual”, a core entity of discussion in the context, in its top predictions. In contrast, MoC gives rather generic predictions irrelevant to the context in both cases.

• For the 5th and the 6th example, we see MoS is able to exploit less common words accurately according to the context, while MoC fails to yield such choices. This well matches our analysis that MoS has the capacity of modeling context-dependent language.
## Table 2.11: Comparison of next-token prediction on Penn Treebank test data. N stands for a number as the result of preprocessing \[55\]. The context shown only includes the previous sentence and the current sentence the prediction step resides in.
2.2 A Faster High-Rank Language Model (Proposed Work)

2.2.1 Motivation

The previous section shows that the Mixture of Softmaxes (MoS) substantially improves performance on multiple language modeling benchmarks. However, MoS brings additional computational costs and slows down training. In this section, we propose a novel high-rank language model that trains faster than MoS while maintaining the ability to break the Softmax bottleneck.

2.2.2 Proposed Approach

MoS is slower than standard Softmax because MoS has to compute multiple Softmax components while Softmax itself is already an expensive operation. Our proposed approach aims to address this issue by moving the mixture operations from the probability space into the logit space so that only one Softmax needs to be computed. However, as shown in Chapter 2.1.2, naively applying a mixture operation in the logit space cannot break the Softmax bottleneck. In contrast, we propose to perform \textit{gating} operations in the logit space. The difference between gating and mixture is that gating has different weights for different tokens in the vocabulary, which will allow us to effectively break the Softmax bottleneck.

Specifically, following the notation from the previous section, the conditional distribution is formulated as

\[
P_\theta(y|x) = \frac{\exp \left( \sum_{k=1}^{K} \pi_{c,x,k} h_{c,k}^\top w_x \right)}{\sum_{x'} \exp \left( \sum_{k=1}^{K} \pi_{c,x',k} h_{c,k}^\top w_{x'} \right)}; \quad \text{s.t.} \sum_{k=1}^{K} \pi_{c,x,k} = 1 \quad \forall c, x
\]

Note the prior \(\pi\) is dependent not only on the context \(c\) but also on each word \(x\). As a result, \(\sum_{k=1}^{K} \pi_{c,x,k} h_{c,k}^\top\) cannot be clamped into one vector that only depends on \(c\), and thus this is no longer a low-rank approximation. To see it more clearly, this formulation approximates the log probability matrix \(A\) by

\[
\hat{A}_{\text{new}} = \sum_{k=1}^{K} \Pi_k \odot \left( H_{\theta,k} W_{\theta}^\top \right)
\]

The elementwise multiplication introduces nonlinearity into the formulation, and as a result it is not limited by the Softmax bottleneck.

The context vectors \(h_{c,k}\) are formulated similar to MoS. Recall from the previous section that we use \((g_1, \cdots, g_T)\) to denote a sequence of hidden states output by the RNNs. We condition the prior on the hidden states \(g_t\) as well as an additional set of word embeddings \(E\). Let \(e_x\) denote the additional embedding of word \(x\). We have

\[
\pi_{c,x,k} = \frac{\exp \left( e_x^\top \tanh(W_{\pi,k} g_t) \right)}{\sum_{k'} \exp \left( e_x^\top \tanh(W_{\pi,k'} g_t) \right)}
\]

where \(W_{\pi,\cdot}\) are model parameters.

The above formulation is still slow because a Softmax operation is needed for each \(x\) in the vocabulary. To further reduce the computational cost, we propose to use \textit{hierarchical sigmoid}
gating to replace Softmax. The main idea is to replace a Softmax with with multiple sigmoid’s. Given $K$ components, we compute $(K - 1)$ sigmoid gates:

$$\alpha_{c,x,k} = \sigma \left( e_x^T \tanh(W_{\pi,k}g_t) \right)$$

We then construct a binary tree to compose $(K - 1)$ sigmoid gates into a valid probability distribution. Without loss of generality, consider $K = 4$. We compute the prior as:

$$\pi_1 = \alpha_1 \alpha_2, \quad \pi_2 = \alpha_1 (1 - \alpha_2), \quad \pi_3 = (1 - \alpha_1) \alpha_3, \quad \pi_4 = (1 - \alpha_1)(1 - \alpha_3)$$

where we drop the subscript $c$ and $x$ for simplicity. To stabilize training, we normalize the weights $W_{\pi,\cdot}$.

Our approach is faster than MoS because the gating operations take place in the logit space and only one Softmax operation is needed. Moreover, since we use vector gating instead of scalar mixture, our formulation has high capacity so in practice we fix the number of components $K$ to be 4, which is a relatively small value compared to MoS.

### 2.2.3 Preliminary Results

We show preliminary results on Penn Treebank word-level language modeling. We perform random search for hyperparameter tuning on the validation set for both our model and the baselines.

<table>
<thead>
<tr>
<th>Model</th>
<th>Ppl (prior)</th>
<th>Ppl (our impl)</th>
<th>Time-12</th>
<th>Time-24</th>
<th>Time-36</th>
<th>Time-48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softmax</td>
<td>58.8</td>
<td>59.19</td>
<td>5</td>
<td>8</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>MoS-3</td>
<td>58.62</td>
<td>57.62</td>
<td>39</td>
<td>75</td>
<td>112</td>
<td>147</td>
</tr>
<tr>
<td>MoS-5</td>
<td>57.36</td>
<td>57.24</td>
<td>46</td>
<td>88</td>
<td>130</td>
<td>173</td>
</tr>
<tr>
<td>MoS-10</td>
<td>56.33</td>
<td>56.49</td>
<td>63</td>
<td>124</td>
<td>185</td>
<td>242</td>
</tr>
<tr>
<td>MoS-15</td>
<td>55.97</td>
<td>56.14</td>
<td>80</td>
<td>158</td>
<td>236</td>
<td>310</td>
</tr>
<tr>
<td>Ours</td>
<td>-</td>
<td>56.33</td>
<td>27</td>
<td>49</td>
<td>73</td>
<td>95</td>
</tr>
</tbody>
</table>

Table 2.12: Perplexity and training time comparison on Penn Treebank word-level language modeling. Ppl is perplexity. “MoS-K” denotes MoS with $K$ mixture components. “Time-B” denotes the training time of the output layer with batch size $B$.

As shown in Table 2.12, we obtain results competitive to the state-of-the-art MoS model while improving the training speed of the output layer by over three times.

We will perform comprehensive experiments on more datasets to further substantiate our claim, including the 1 Billion Word language modeling benchmark and the WMT machine translation benchmarks.
Chapter 3

Semi-Supervised Learning by Generative Modeling

This chapter focuses on using generative modeling to improve downstream task performance in a semi-supervised learning setting. We consider four scenarios: 1) semi-supervised classification by generating low-density adversarial samples, 2) semi-supervised question answering by generating natural language questions given context, 3) semi-supervised learning on graphs by modeling the generation of random walk paths, and 4) generative pretraining using a language modeling objective to induce transferable graphical structured representations.

3.1 Semi-Supervised Learning by GANs (Completed Work, NIPS 2017)

3.1.1 Motivation

Deep neural networks are usually trained on a large amount of labeled data, and it has been a challenge to apply deep models to datasets with limited labels. Semi-supervised learning (SSL) aims to leverage the large amount of unlabeled data to boost the model performance, particularly focusing on the setting where the amount of available labeled data is limited. Traditional graph-based methods \[4, 108\] were extended to deep neural networks \[41, 92, 98\], which involves applying convolutional neural networks \[47\] and feature learning techniques to graphs so that the underlying manifold structure can be exploited. \[69\] employs a Ladder network to minimize the layerwise reconstruction loss in addition to the standard classification loss. Variational auto-encoders have also been used for semi-supervised learning \[39, 49\] by maximizing the variational lower bound of the unlabeled data log-likelihood.

Recently, generative adversarial networks (GANs) \[26\] were demonstrated to be able to generate visually realistic images. GANs set up an adversarial game between a discriminator and a generator. The goal of the discriminator is to tell whether a sample is drawn from true data or generated by the generator, while the generator is optimized to generate samples that are not distinguishable by the discriminator. Feature matching (FM) GANs \[71\] apply GANs to semi-supervised learning on \(K\)-class classification. The objective of the generator is to match the
first-order feature statistics between the generator distribution and the true distribution. Instead of binary classification, the discriminator employs a \((K+1)\)-class objective, where true samples are classified into the first \(K\) classes and generated samples are classified into the \((K+1)\)-th class. This \((K+1)\)-class discriminator objective leads to strong empirical results, and was later widely used to evaluate the effectiveness of generative models [21, 86].

Though empirically feature matching improves semi-supervised classification performance, the following questions still remain open. First, it is not clear why the formulation of the discriminator can improve the performance when combined with a generator. Second, it seems that good semi-supervised learning and a good generator cannot be obtained at the same time. For example, [71] observed that mini-batch discrimination generates better images than feature matching, but feature matching obtains a much better semi-supervised learning performance. The same phenomenon was also observed in [86], where the model generated better images but failed to improve the performance on semi-supervised learning.

3.1.2 Theoretical Analysis

Given a labeled set \( \mathcal{L} = \{(x, y)\} \), let \( \{1, 2, \cdots, K\} \) be the label space for classification. Let \( D \) and \( G \) denote the discriminator and generator, and \( P_D \) and \( p_G \) denote the corresponding distributions. Consider the discriminator objective function of GAN-based semi-supervised learning [71]:

\[
\max_D \mathbb{E}_{x,y \sim \mathcal{L}} \log P_D(y|x, y \leq K) + \mathbb{E}_{x \sim p} \log P_D(y \leq K|x) + \mathbb{E}_{x \sim p_G} \log P_D(K+1|x), \quad (3.1)
\]

where \( p \) is the true data distribution. The probability distribution \( P_D \) is over \( K+1 \) classes where the first \( K \) classes are true classes and the \((K+1)\)-th class is the fake class. The objective function consists of three terms. The first term is to maximize the log conditional probability for labeled data, which is the standard cost as in supervised learning setting. The second term is to maximize the log probability of the first \( K \) classes for unlabeled data. The third term is to maximize the log probability of the \((K+1)\)-th class for generated data. Note that the above objective function bears a similar merit to the original GAN formulation if we treat \( P(K+1|x) \) to be the probability of fake samples, while the only difference is that we split the probability of true samples into \( K \) sub-classes.

Let \( f(x) \) be a nonlinear vector-valued function, and \( w_k \) be the weight vector for class \( k \). As a standard setting in previous work [21, 71], the discriminator \( D \) is defined as \( P_D(k|x) = \frac{\exp(w_k f(x))}{\sum_{k=1}^{K+1} \exp(w_k f(x))} \). Since this is a form of over-parameterization, \( w_{K+1} \) is fixed as a zero vector [71]. We next discuss the choices of different possible \( G \)'s.

Perfect Generator

Here, by perfect generator we mean that the generator distribution \( p_G \) exactly matches the true data distribution \( p \), i.e., \( p_G = p \). We now show that when the generator is perfect, it does not improve the generalization over the supervised learning setting.
Proposition 2. If \( p_G = p \), and \( D \) has infinite capacity, then for any optimal solution \( D = (w, f) \) of the following supervised objective,

\[
\max_D \mathbb{E}_{x,y \sim \mathcal{L}} \log P_D(y|x, y \leq K),
\]

there exists \( D^* = (w^*, f^*) \) such that \( D^* \) maximizes Eq. (3.14) and that for all \( x \), \( P_D(y|x, y \leq K) = P_{D^*}(y|x, y \leq K) \).

The proof is provided in the supplementary material. Proposition 2 states that for any optimal solution \( D \) of the supervised objective, there exists an optimal solution \( D^* \) of the \((K + 1)\)-class objective such that \( D \) and \( D^* \) share the same generalization error. In other words, using the \((K + 1)\)-class objective does not prevent the model from experiencing any arbitrarily high generalization error that it could suffer from under the supervised objective. Moreover, since all the optimal solutions are equivalent w.r.t. the \((K + 1)\)-class objective, it is the optimization algorithm that really decides which specific solution the model will reach, and thus what generalization performance it will achieve. This implies that when the generator is perfect, the \((K + 1)\)-class objective by itself is not able to improve the generalization performance. In fact, in many applications, an almost infinite amount of unlabeled data is available, so learning a perfect generator for purely sampling purposes should not be useful. In this case, our theory suggests that not only the generator does not help, but also unlabeled data is not effectively utilized when the generator is perfect.

Complement Generator

The function \( f \) maps data points in the input space to the feature space. Let \( p_k(f) \) be the density of the data points of class \( k \) in the feature space. Given a threshold \( \epsilon_k \), let \( F_k \) be a subset of the data support where \( p_k(f) > \epsilon_k \), i.e., \( F_k = \{ f : p_k(f) > \epsilon_k \} \). We assume that given \( \{ \epsilon_k \}_{k=1}^K \), the \( F_k \)’s are disjoint with a margin. More formally, for any \( f_j \in F_j, f_k \in F_k, \) and \( j \neq k \), we assume that there exists a real number \( 0 < \alpha < 1 \) such that \( \alpha f_j + (1 - \alpha) f_k \notin F_j \cup F_k \). As long as the probability densities of different classes do not share any mode, i.e., \( \forall i \neq j, \arg \max f_i \cap \arg \max f_j = \emptyset \), this assumption can always be satisfied by tuning the thresholds \( \epsilon_k \)’s. With the assumption held, we will show that the model performance would be better if the thresholds could be set to smaller values (ideally zero). We also assume that each \( F_k \) contains at least one labeled data point.

Suppose \( \bigcup_{k=1}^K F_k \) is bounded by a convex set \( B \). If the support \( F_G \) of a generator \( G \) in the feature space is a relative complement set in \( B \), i.e., \( F_G = B - \bigcup_{k=1}^K F_k \), we call \( G \) a complement generator. The reason why we utilize a bounded \( B \) to define the complement is presented in the supplementary material. Note that the definition of complement generator implies that \( G \) is a function of \( f \). By treating \( G \) as function of \( f \), theoretically \( D \) can optimize the original objective function in Eq. (3.14).

Now we present the assumption on the convergence conditions of the discriminator. Let \( \mathcal{U} \) and \( \mathcal{G} \) be the sets of unlabeled data and generated data.

Assumption 1. Convergence conditions. When \( D \) converges on a finite training set \( \{ \mathcal{L}, \mathcal{U}, \mathcal{G} \} \), \( D \) learns a (strongly) correct decision boundary for all training data points. More specifically, (1) for any \( (x, y) \in \mathcal{L} \), we have \( w_y^T f(x) > w_k^T f(x) \) for any other class \( k \neq y \); (2) for any \( x \in \mathcal{G} \), we have \( 0 > \max_{k=1}^K w_k^T f(x) \); (3) for any \( x \in \mathcal{U} \), we have \( \max_{k=1}^K w_k^T f(x) > 0 \).
In Assumption 1, conditions (1) and (2) assume classification correctness on labeled data and true-fake correctness on generated data respectively, which is directly induced by the objective function. Likewise, it is also reasonable to assume true-fake correctness on unlabeled data, i.e., \( \log \sum_k \exp w_k^\top f(x) > 0 \) for \( x \in \mathcal{U} \). However, condition (3) goes beyond this and assumes \( \max_k w_k^\top f(x) > 0 \). We discuss this issue in detail in the supplementary material and argue that these assumptions are reasonable. Moreover, in Section 3.1.4, our approach addresses this issue explicitly by adding a conditional entropy term to the discriminator objective to enforce condition (3).

**Lemma 2.** Suppose for all \( k \), the L2-norms of weights \( w_k \) are bounded by \( \|w_k\|_2 \leq C \). Suppose that there exists \( \epsilon > 0 \) such that for any \( f_G \in F_G \), there exists \( f'_G \in \mathcal{G} \) such that \( \|f_G - f'_G\|_2 \leq \epsilon \). With the conditions in Assumption 7 for all \( k \leq K \), we have \( w_k^\top f_G < C\epsilon \).

**Corollary 1.** When unlimited generated data samples are available, with the conditions in Lemma 2 we have \( \lim_{|G| \to \infty} w_k^\top f_G \leq 0 \).

See the supplementary material for the proof.

**Proposition 3.** Given the conditions in Corollary 1 for all class \( k \leq K \), for all feature space points \( f_k \in F_k \), we have \( w_k^\top f_k > w_j^\top f_k \) for any \( j \neq k \).

**Proof.** Without loss of generality, suppose \( j = \arg \max_{j \neq k} w_j^\top f_k \). Now we prove it by contradiction. Suppose \( w_j^\top f_k \leq w_j^\top f_k \). Since \( F_k \)'s are disjoint with a margin, \( B \) is a convex set and \( F_G = B - \cup_k F_k \), there exists \( 0 < \alpha < 1 \) such that \( f_G = \alpha f_k + (1 - \alpha) f_j \) with \( f_G \in F_G \) and \( f_j \) being the feature of a labeled data point in \( F_j \). By Corollary 1, it follows that \( w_j^\top f_G \leq 0 \). Thus, \( w_j^\top f_G = \alpha w_j^\top f_k + (1 - \alpha) w_j^\top f_j \leq 0 \). By Assumption 1, \( w_j^\top f_k > 0 \) and \( w_j^\top f_j > 0 \), leading to contradiction. It follows that \( w_k^\top f_k > w_j^\top f_k \) for any \( j \neq k \).

Proposition 3 guarantees that when \( G \) is a complement generator, under mild assumptions, a near-optimal \( D \) learns correct decision boundaries in each high-density subset \( F_k \) (defined by \( \epsilon_k \)) of the data support in the feature space. Intuitively, the generator generates complement samples so the logits of the true classes are forced to be low in the complement. As a result, the discriminator obtains class boundaries in low-density areas. This builds a connection between our approach with manifold-based methods [4, 108] which also leverage the low-density boundary assumption.

With our theoretical analysis, we can now answer the questions raised in Section 2.1.1. First, the \( (K + 1) \)-class formulation is effective because the generated complement samples encourage the discriminator to place the class boundaries in low-density areas (Proposition 3). Second, good semi-supervised learning indeed requires a bad generator because a perfect generator is not able to improve the generalization performance (Proposition 2).

### 3.1.3 Case Study on Synthetic Data

In the previous section, we have established the fact a complement generator, instead of a perfect generator, is what makes a good semi-supervised learning algorithm. Now, to get a more intuitive understanding, we conduct a case study based on two 2D synthetic datasets, where we can easily verify our theoretical analysis by visualizing the model behaviors. In addition, by analyzing how feature matching (FM) [71] works in 2D space, we identify some potential problems of it, which
Figure 3.1: Labeled and unlabeled data are denoted by cross and point respectively, and different colors indicate classes.

Figure 3.2: Left: Classification decision boundary, where the white line indicates true-fake boundary; Right: True-Fake decision boundary

Figure 3.3: Feature space at convergence

Figure 3.4: Left: Blue points are generated data, and the black shadow indicates unlabeled data. Middle and right can be interpreted as above.

motivates our approach to be introduced in the next section. Specifically, two synthetic datasets are four spins and two circles, as shown in Fig. 3.1

**Soundness of complement generator** Firstly, to verify that the complement generator is a preferred choice, we construct the complement generator by uniformly sampling from the a bounded 2D box that contains all unlabeled data, and removing those on the manifold. Based on the complement generator, the result on four spins is visualized in Fig. 3.2 As expected, both the classification and true-fake decision boundaries are almost perfect. More importantly, the classification decision boundary always lies in the fake data area (left panel), which well matches our theoretical analysis.

**Visualization of feature space** Next, to verify our analysis about the feature space, we choose the feature dimension to be 2, apply the FM to the simpler dataset of two circles, and visualize the feature space in Fig. 3.3 As we can see, most of the generated features (blue points) resides in between the features of two classes (green and orange crosses), although there exists some overlap. As a result, the discriminator can almost perfectly distinguish between true and generated samples as indicated by the black decision boundary, satisfying the our required Assumption. Meanwhile, the model obtains a perfect classification boundary (blue line) as our analysis suggests.

**Pros and cons of feature matching** Finally, to further understand the strength and weakness of FM, we analyze the solution FM reaches on four spins shown in Fig. 3.4 From the left
panel, we can see many of the generated samples actually fall into the data manifold, while the rest scatters around in the nearby surroundings of data manifold. It suggests that by matching the first-order moment by SGD, FM is performing some kind of distribution matching, though in a rather weak manner. Loosely speaking, FM has the effect of generating samples close to the manifold. But due to its weak power in distribution matching, FM will inevitably generate samples outside of the manifold, especially when the data complexity increases. Consequently, the generator density $p_G$ is usually lower than the true data density $p$ within the manifold and higher outside. Hence, an optimal discriminator $P_{D^*}(K + 1 \mid x) = \frac{p(x)}{p(x) + p_G(x)}$ could still distinguish between true and generated samples in many cases. However, there are two types of mistakes the discriminator can still make:

1. Higher density mistake inside manifold: Since the FM generator still assigns a significant amount of probability mass inside the support, wherever $p_G > p > 0$, an optimal discriminator will incorrectly predict samples in that region as “fake”. Actually, this problem has already shown up when we examine the feature space (Fig. 3.3).

2. Collapsing with missing coverage outside manifold: As the feature matching objective for the generator only requires matching the first-order statistics, there exists many trivial solutions the generator can end up with. For example, it can simply collapse to mean of unlabeled features, or a few surrounding modes as along as the feature mean matches. Actually, we do see such collapsing phenomenon in high-dimensional experiments when FM is used (see Fig. 3.5a and Fig. 3.5c). As a result, a collapsed generator will fail to cover some gap areas between manifolds. Since the discriminator is only well-defined on the union of the data supports of $p$ and $p_G$, the prediction result in such missing area is under-determined and fully relies on the smoothness of the parametric model. In this case, significant mistakes can also occur.

### 3.1.4 Approach

As discussed in previous sections, feature matching GANs suffer from the following drawbacks: 1) the first-order moment matching objective does not prevent the generator from collapsing (missing coverage); 2) feature matching can generate high-density samples inside manifold; 3) the discriminator objective does not encourage realization of condition (3) in Assumption 1 as discussed in Section 3.1.2. Our approach aims to explicitly address the above drawbacks.

Following prior work [26, 71], we employ a GAN-like implicit generator. We first sample a latent variable $z$ from a uniform distribution $U(0, 1)$ for each dimension, and then apply a deep convolutional network to transform $z$ to a sample $x$.

**Generator Entropy**

Fundamentally, the first drawback concerns the entropy of the distribution of generated features, $\mathcal{H}(p_G(f))$. This connection is rather intuitive, as the collapsing issue is a clear sign of low entropy. Therefore, to avoid collapsing and increase coverage, we consider explicitly increasing the entropy.

Although the idea sounds simple and straightforward, there are two practical challenges. Firstly, as implicit generative models, GANs only provide samples rather than an analytic density form. As a result, we cannot evaluate the entropy exactly, which rules out the possibility of naive
optimization. More problematically, the entropy is defined in a high-dimensional feature space, which is changing dynamically throughout the training process. Consequently, it is difficult to estimate and optimize the generator entropy in the feature space in a stable and reliable way. Faced with these difficulties, we consider two practical solutions.

The first method is inspired by the fact that input space is essentially static, where estimating and optimizing the counterpart quantities would be much more feasible. Hence, we instead increase the generator entropy in the input space, i.e., $\mathcal{H}(p_G(x))$, using a technique derived from an information theoretical perspective and relies on variational inference (VI). Specially, let $\mathcal{Z}$ be the latent variable space, and $\mathcal{X}$ be the input space. We introduce an additional encoder, $q : \mathcal{X} \mapsto \mathcal{Z}$, to define a variational upper bound of the negative entropy \[ -H(p_G(x)) \leq -\mathbb{E}_{x,z \sim p_G} \log q(z|x) = L_{VI}. \]

Hence, minimizing the upper bound $L_{VI}$ effectively increases the generator entropy. In our implementation, we formulate $q$ as a diagonal Gaussian with bounded variance, i.e., $q(z|x) = \mathcal{N}(\mu(x), \sigma^2(x))$, with $0 < \sigma(x) < \theta$, where $\mu(\cdot)$ and $\sigma(\cdot)$ are neural networks, and $\theta$ is the threshold to prevent arbitrarily large variance.

Alternatively, the second method aims at increasing the generator entropy in the feature space by optimizing an auxiliary objective. Concretely, we adapt the pull-away term (PT) \[ L_{PT} = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j \neq i} \left( \frac{f(x_i)^\top f(x_j)}{\|f(x_i)\| \cdot \|f(x_j)\|} \right)^2, \]

where $N$ is the size of a mini-batch and $x$ are samples. Intuitively, the pull-away term tries to orthogonalize the features in each mini-batch by minimizing the squared cosine similarity. Hence, it has the effect of increasing the diversity of generated features and thus the generator entropy.

### Generating Low-Density Samples

The second drawback of feature matching GANs is that high-density samples can be generated in the feature space, which is not desirable according to our analysis. Similar to the argument in Section 3.1.4, it is infeasible to directly minimize the density of generated features. Instead, we enforce the generation of samples with low density in the input space. Specifically, given a threshold $\epsilon$, we minimize the following term as part of our objective:

\[ \mathbb{E}_{x \sim p_G} \log p(x) \mathbb{I}[p(x) > \epsilon] \]

where $\mathbb{I}[\cdot]$ is an indicator function. Using a threshold $\epsilon$, we ensure that only high-density samples are penalized while low-density samples are unaffected. Intuitively, this objective pushes the generated samples to “move” towards low-density regions defined by $p(x)$. To model the probability distribution over images, we simply adapt the state-of-the-art density estimation model for natural images, namely the PixelCNN++ \[ \text{[72]} \]

model. The PixelCNN++ model is used to estimate the density $p(x)$ in Eq. (3.3). The model is pretrained on the training set, and fixed during semi-supervised training.

### Generator Objective and Interpretation

Combining our solutions to the first two drawbacks of feature matching GANs, we have the following objective function of the generator:

\[ \min_G \mathcal{H}(p_G) + \mathbb{E}_{x \sim p_G} \log p(x) \mathbb{I}[p(x) > \epsilon] + \|\mathbb{E}_{x \sim p_G} f(x) - \mathbb{E}_{x \sim \mathcal{U}} f(x)\|^2. \]

(3.4)
This objective is closely related to the idea of complement generator discussed in Section 3.1.2. To see that, let’s first define a target complement distribution in the input space as follows

\[
p^*(x) = \begin{cases} \frac{1}{Z} \frac{1}{p(x)} & \text{if } p(x) > \epsilon \text{ and } x \in B_x \\ C & \text{if } p(x) \leq \epsilon \text{ and } x \in B_x \end{cases}
\]

where \( Z \) is a normalizer, \( C \) is a constant, and \( B_x \) is the set defined by mapping \( B \) from the feature space to the input space. With the definition, the KL divergence (KLD) between \( p_G(x) \) and \( p^*(x) \) is

\[
KL(p_G \parallel p^*) = -\mathcal{H}(p_G) + \mathbb{E}_{x \sim p_G} \log p(x) \mathbb{1}[p(x) > \epsilon] + \mathbb{E}_{x \sim p_G} \left( \mathbb{1}[p(x) > \epsilon] \log Z - \mathbb{1}[p(x) \leq \epsilon] \log C \right).
\]

The form of the KLD immediately reveals the aforementioned connection. Firstly, the KLD shares two exactly the same terms with the generator objective (3.4). Secondly, while \( p^*(x) \) is only defined in \( B_x \), there is not such a hard constraint on \( p_G(x) \). However, the feature matching term in Eq. (3.4) can be seen as softly enforcing this constraint by bringing generated samples “close” to the true data (Cf. Section 3.1.3). Moreover, because the identity function \( \mathbb{1}[\cdot] \) has zero gradient almost everywhere, the last term in KLD would not contribute any informative gradient to the generator. In summary, optimizing our proposed objective (3.4) can be understood as minimizing the KL divergence between the generator distribution and a desired complement distribution, which connects our practical solution to our theoretical analysis.

**Conditional Entropy**

In order for the complement generator to work, according to condition (3) in Assumption 1, the discriminator needs to have strong true-fake belief on unlabeled data, i.e., \( \max_{k=1}^K w_k^T f(x) > 0 \). However, the objective function of the discriminator in [71] does not enforce a dominant class. Instead, it only needs \( \sum_{k=1}^K P_D(k|x) > P_D(K + 1|x) \) to obtain a correct decision boundary, while the probabilities \( P_D(k|x) \) for \( k \leq K \) can possibly be uniformly distributed. To guarantee the strong true-fake belief in the optimal conditions, we add a conditional entropy term to the discriminator objective and it becomes,

\[
\max_D \mathbb{E}_{x,y \sim D} \log p_D(y|x, y \leq K) + \mathbb{E}_{x \sim \mathcal{U}} \log p_D(y \leq K|x) + \mathbb{E}_{x \sim p_G} \log p_D(K + 1|x) + \mathbb{E}_{x \sim \mathcal{U}} \sum_{k=1}^K p_D(k|x) \log p_D(k|x).
\]

By optimizing Eq. (3.5), the discriminator is encouraged to satisfy condition (3) in Assumption 1. Note that the same conditional entropy term has been used in other semi-supervised learning methods [58, 78] as well, but here we motivate the minimization of conditional entropy based on our theoretical analysis of GAN-based semi-supervised learning.

To train the networks, we alternatively update the generator and the discriminator to optimize Eq. (3.4) and Eq. (3.5) based on mini-batches. If an encoder is used to maximize \( \mathcal{H}(p_G) \), the encoder and the generator are updated at the same time.
<table>
<thead>
<tr>
<th>Methods</th>
<th>MNIST (# errors)</th>
<th>SVHN (% errors)</th>
<th>CIFAR-10 (% errors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CatGAN [78]</td>
<td>191 ± 10</td>
<td>-</td>
<td>19.58 ± 0.46</td>
</tr>
<tr>
<td>SDGM [49]</td>
<td>132 ± 7</td>
<td>16.61 ± 0.24</td>
<td>-</td>
</tr>
<tr>
<td>Ladder network</td>
<td>106 ± 37</td>
<td>-</td>
<td>20.40 ± 0.47</td>
</tr>
<tr>
<td>ADGM [49]</td>
<td>96 ± 2</td>
<td>22.86</td>
<td>-</td>
</tr>
<tr>
<td>FM [71] *</td>
<td>93 ± 6.5</td>
<td>8.11 ± 1.3</td>
<td>18.63 ± 2.32</td>
</tr>
<tr>
<td>ALI [20]</td>
<td>-</td>
<td>7.42 ± 0.65</td>
<td>17.99 ± 1.62</td>
</tr>
<tr>
<td>VAT small [58] *</td>
<td>136</td>
<td>6.83</td>
<td>14.87</td>
</tr>
<tr>
<td>Our best model</td>
<td>79.5 ± 9.8</td>
<td>4.25 ± 0.03</td>
<td>14.41 ± 0.30</td>
</tr>
<tr>
<td>Triple GAN [48] *‡</td>
<td>91 ± 58</td>
<td>5.77 ± 0.17</td>
<td>16.99 ± 0.36</td>
</tr>
<tr>
<td>II model [45] †‡</td>
<td>-</td>
<td>5.43 ± 0.25</td>
<td>16.55 ± 0.29</td>
</tr>
<tr>
<td>VAT+EntMin+Large [58] †</td>
<td>-</td>
<td>4.28</td>
<td>13.15</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison with state-of-the-art methods on three benchmark datasets. Only methods without data augmentation are included. * indicates using the same (small) discriminator architecture, † indicates using a larger discriminator architecture, and ‡ means self-ensembling.

Figure 3.5: Comparing images generated by FM and our model. FM generates collapsed samples, while our model generates diverse “bad” samples.

### 3.1.5 Experiments

We mainly consider three widely used benchmark datasets, namely MNIST, SVHN, and CIFAR-10. As in previous work, we randomly sample 100, 1,000, and 4,000 labeled samples for MNIST, SVHN, and CIFAR-10 respectively during training, and use the standard data split for testing. We use the 10-quantile log probability to define the threshold $\epsilon$ in Eq. (3.4). We add instance noise to the input of the discriminator [1, 77], and use spatial dropout [83] to obtain faster convergence.

**Main Results**

We compare the the results of our best model with state-of-the-art methods on the benchmarks in Table 3.1. Our proposed methods consistently improve the performance upon feature matching. We achieve new state-of-the-art results on all the datasets when only small discriminator architecture is considered. Our results are also state-of-the-art on MNIST and SVHN among all single-model results, even when compared with methods using self-ensembling and large discriminator architectures. Finally, note that because our method is actually orthogonal to VAT [58], combin-
Table 3.2: Ablation study. *FM* is feature matching. *LD* is the low-density enforcement term in Eq. (3.3). *VI* and *PT* are two entropy maximization methods described in Section 3.1.4. *Ent* means the conditional entropy term in Eq. (3.5). *Max log-p* is the maximum log probability of generated samples, evaluated by a PixelCNN++ model. *10-quant* shows the 10-quantile of true image log probability. *Error* means the number of misclassified examples on MNIST, and error rate (%) on others.

Ablation Study

We report the results of ablation study in Table 3.2. In the following, we analyze the effects of several components in our model, subject to the intrinsic features of different datasets.

First, the generator entropy terms (VI and PT) (Section 3.1.4) improve the performance on SVHN and CIFAR by up to 2.2 points in terms of error rate. Moreover, as shown in Fig. 3.7, our model significantly reduces the collapsing effects present in the samples generated by FM, which also indicates that maximizing the generator entropy is beneficial. On MNIST, probably due to its simplicity, no collapsing phenomenon was observed with vanilla FM training [71] or in our setting. Under such circumstances, maximizing the generator entropy seems to be unnecessary, and the estimation bias introduced by approximation techniques can even hurt the performance.

Second, the low-density (LD) term is useful when FM indeed generates samples in high-density areas. MNIST is a typical example in this case. When trained with FM, most of the generated handwritten digits are highly realistic and have high log probabilities according to the density model (Cf. max log-p in Table 3.2). Hence, when applied to MNIST, LD improves the performance by a clear margin. By contrast, few of the generated SVHN images are realistic (Cf. Fig. 3.5a). Quantitatively, SVHN samples are assigned very low log probabilities (Cf. Table 3.2). As expected, LD has a negligible effect on the performance for SVHN. Moreover, the “max log-p” column in Table 3.2 shows that while LD can reduce the maximum log probability of the generated MNIST samples by a large margin, it does not yield noticeable difference on SVHN. This further justifies our analysis. Based on the above conclusion, we conjecture LD would not
help on CIFAR where sample quality is even lower. Thus, we did not train a density model on CIFAR due to the limit of computational resources.

Third, adding the conditional entropy term has mixed effects on different datasets. While the conditional entropy (Ent) is an important factor of achieving the best performance on SVHN, it hurts the performance on MNIST and CIFAR. One possible explanation relates to the classic exploitation-exploration tradeoff, where minimizing Ent favors exploitation and minimizing the classification loss favors exploration. During the initial phase of training, the discriminator is relatively uncertain and thus the gradient of the Ent term might dominate. As a result, the discriminator learns to be more confident even on incorrect predictions, and thus gets trapped in local minima.

Lastly, we vary the values of the hyper-parameter $\epsilon$ in Eq. (3.4). As shown at the bottom of Table 3.2, reducing $\epsilon$ clearly leads to better performance, which further justifies our analysis in Sections 3.1.3 and 3.1.2 that off-manifold samples are favorable.

### Generated Samples

We compare the generated samples of FM and our approach in Fig. 3.7. The FM images in Fig. 3.5c are extracted from previous work [71]. While collapsing is widely observed in FM samples, our model generates diverse “bad” images, which is consistent with our analysis.

### 3.2 Semi-Supervised QA by Generating Questions (Completed Work, ACL 2017)

#### 3.2.1 Motivation

Recently, various neural network models were proposed and successfully applied to the tasks of questions answering (QA) and/or reading comprehension [19, 96, 101]. While achieving state-of-the-art performance, these models rely on a large amount of labeled data. However, it is extremely difficult to collect large-scale question answering datasets. Historically, many of the question answering datasets have only thousands of question answering pairs, such as WebQuestions [6], MCTest [70], WikiQA [97], and TREC-QA [90]. Although larger question answering datasets with hundreds of thousands of question-answer pairs have been collected, including SQuAD [68], MSMARCO [60], and NewsQA [85], the data collection process is expensive and time-consuming in practice. This hinders real-world applications for domain-specific question answering.

Compared to obtaining labeled question answer pairs, it is trivial to obtain unlabeled text data. In this work, we study the following problem of semi-supervised question answering: is it possible to leverage unlabeled text to boost the performance of question answering models, especially when only a small amount of labeled data is available? The problem is challenging because conventional manifold-based semi-supervised learning algorithms [98, 107] cannot be straightforwardly applied. Moreover, since the main foci of most question answering tasks are extraction rather than generation, it is also not sensible to use unlabeled text to improve language modeling as in machine translation [29].
3.2.2 Problem Definition

Let us first introduce the problem of semi-supervised question answering. Let \( L = \{q^{(i)}, a^{(i)}, p^{(i)}\}_{i=1}^{N} \) denote a question answering dataset of \( N \) instances, where \( q^{(i)} \), \( a^{(i)} \), and \( p^{(i)} \) are the question, answer, and paragraph of the \( i \)-th instance respectively. The goal of question answering is to produce the answer \( a^{(i)} \) given the question \( q^{(i)} \) along with the paragraph \( p^{(i)} \). We will drop the superscript \( .^{(i)} \) when the context is unambiguous. In our formulation, following the setting in SQuAD [68], we specifically focus on extractive question answering, where \( a \) is always a consecutive chunk of text in \( p \). More formally, let \( p = (p_1, p_2, \cdots, p_T) \) be a sequence of word tokens with \( T \) being the length, then \( a \) can always be represented as \( a = (p_j, p_{j+1}, \cdots, p_{k-1}, p_k) \), where \( j \) and \( k \) are the start and end token indices respectively. The questions can also be represented as a sequence of word tokens \( q = (q_1, q_2, \cdots, q_{T'}) \) with length \( T' \).

In addition to the labeled dataset \( L \), in the semi-supervised setting, we are also given a set of unlabeled data, denoted as \( U = \{a^{(i)}, p^{(i)}\}_{i=1}^{M} \), where \( M \) is the number of unlabeled instances. Note that it is usually trivial to have access to an almost infinite number of paragraphs \( p \) from sources such as Wikipedia articles and other web pages. And since the answer \( a \) is always a consecutive chunk in \( p \), we argue that it is also sensible to extract possible answer chunks from the unlabeled text using linguistic tags. We will discuss the technical details of answer chunk extraction in Section 3.2.4, and in the formulation of our framework, we assume that the answer chunks \( a \) are available.

Given both the labeled data \( L \) and the unlabeled data \( U \), the goal of semi-supervised question answering is to learn a question answering model \( D \) that captures the probability distribution \( P(a|p,q) \). We refer to this question answering model \( D \) as the discriminative model, in contrast to the generative model that we will present in Section 3.2.3.

A Simple Baseline

We now present a simple baseline for semi-supervised question answering. Given a paragraph \( p = (p_1, p_2, \cdots, p_T) \) and the answer \( a = (p_j, p_{j+1}, \cdots, p_{k-1}, p_k) \), we extract \( (p_{j-W}, p_{j-W+1}, \cdots, p_{j-1}, p_{j+1}, p_{j+2}, \cdots, p_{k+W}) \) from the paragraph and treat it as the question. Here \( W \) is the window size and is set at 5 in our experiments so that the lengths of the questions are similar to human-generated questions. The context-based question-answer pairs on \( U \) are combined with human-generated pairs on \( L \) for training the discriminative model. Intuitively, this method extracts the contexts around the answer chunks to serve as hints for the question answering model. Surprisingly, this simple baseline method leads to substantial improvements when labeled data is limited.

3.2.3 Approach

Though the simple method described in Section 3.2.2 can lead to substantial improvement, we aim to design a learning-based model to move even further. In this section, we will describe the model architecture and the training algorithms for the GDANs. We will use a notation in the context of question answering following Section 3.2.2, but one should be able to extend the notion of GDANs to other applications as well.
The GDAN framework consists of two models, a discriminative model and a generative model. We will first discuss the two models in detail in the context of question answering, and then present an algorithm based on reinforcement learning to combine the two models.

**Discriminative Model**

The discriminative model learns the conditional probability of an answer chunk given the paragraph and the question, i.e., \( P(a|p, q) \). We employ a gated-attention (GA) reader [19] as our base model in this work, but our framework does not make any assumptions about the base models being used. The discriminative model is referred to as \( D \).

The GA model consists of \( K \) layers with \( K \) being a hyper-parameter. Let \( H_k^p \) be the intermediate paragraph representation at layer \( k \), and \( H_q \) be the question representation. The paragraph representation \( H_k^p \) is a \( T \times d \) matrix, and the question representation \( H_q \) is a \( T' \times d \) matrix, where \( d \) is the dimensionality of the representations. Given the paragraph \( p \), we apply a bidirectional Gated Recurrent Unit (GRU) network [14] on top of the embeddings of the sequence \((p_1, p_2, \cdots, p_T)\), and obtain the initial paragraph representation \( H_0^p \). Given the question \( q \), we also apply another bidirectional GRU to obtain the question representation \( H_q \).

The question and paragraph representations are combined with the gated-attention (GA) mechanism [19]. More specifically, for each paragraph token \( p_i \), we compute

\[
\alpha_j = \frac{\exp h_{q,j}^T h_{p,i}^{k-1}}{\sum_{j'=1}^{T'} \exp h_{q,j'}^T h_{p,i}^{k-1}}
\]

\[
h_{p,i}^k = \sum_{j=1}^{T'} \alpha_j h_{q,j} \odot h_{p,i}^{k-1}
\]

where \( h_{p,i}^k \) is the \( i \)-th row of \( H_k^p \) and \( h_{q,j} \) is the \( j \)-th row of \( H_q \).

Since the answer \( a \) is a sequence of consecutive word tokens in the paragraph \( p \), we apply two softmax layers on top of \( H_K^p \) to predict the start and end indices of \( a \), following Yang et al. [101].

**Domain Adaptation with Tags**

We will train our discriminative model on both model-generated question-answer pairs and human-generated pairs. However, even a well-trained generative model will produce questions somewhat different from human-generated ones. Learning from both human-generated data and model-generated data can thus lead to a biased model. To alleviate this issue, we propose to view the model-generated data distribution and the human-generated data distribution as two different data domains and explicitly incorporate domain adaptation into the discriminative model.

More specifically, we use a domain tag as an additional input to the discriminative model. We use the tag “d_true” to represent the domain of human-generated data (i.e., the true data), and “d_gen” for the domain of model-generated data. Following a practice in domain adaptation [13, 37], we append the domain tag to the end of both the questions and the paragraphs. By introducing the domain tags, we expect the discriminative model to factor out domain-specific and domain-invariant representations. At test time, the tag “d_true” is appended.
Algorithm 1 Training Generative Domain-Adaptive Nets

Input: labeled data $L$, unlabeled data $U$, #iterations $T_G$ and $T_D$

Initialize $G$ by MLE training on $L$
Randomly initialize $D$

while not stopping do
  for $t ← 1$ to $T_D$ do
    Update $D$ to maximize $J(L, d_{true}, D) + J(U_G, d_{gen}, D)$ with SGD
  end for
  for $t ← 1$ to $T_G$ do
    Update $G$ to maximize $J(U_G, d_{true}, D)$ with Reinforce and SGD
  end for
end while
return model $D$

Generative Model

The generative model learns the conditional probability of generating a question given the paragraph and the answer, i.e., $P(q|p, a)$. We implement the generative model as a sequence-to-sequence model [79] with a copy mechanism [28, 30].

The generative model consists of an encoder and a decoder. An encoder is a GRU that encodes the input paragraph into a sequence of hidden states $H$. We inject the answer information by appending an additional zero/one feature to the word embeddings of the paragraph tokens; i.e., if a word token appears in the answer, the feature is set at one, otherwise zero.

The decoder is another GRU with an attention mechanism over the encoder hidden states $H$. At each time step, the generation probabilities over all word types are defined with a copy mechanism:

$$P_{overall} = g_t P_{vocab} + (1 - g_t) P_{copy}$$

where $g_t$ is the probability of generating the token from the vocabulary, while $(1 - g_t)$ is the probability of copying a token from the paragraph. The probability $g_t$ is computed based on the current hidden state $h_t$:

$$g_t = \sigma(w_g^T h_t)$$

where $\sigma$ denotes the logistic function and $w_g$ is a vector of model parameters. The generation probabilities $P_{vocab}$ are defined as a softmax function over the word types in the vocabulary, and the copying probabilities $P_{copy}$ are defined as a softmax function over the word types in the paragraph. Both $P_{vocab}$ and $P_{copy}$ are defined as a function of the current hidden state $h_t$ and the attention results [28].

Training Algorithm

We first define the objective function of the GDANs, and then present an algorithm to optimize the given objective function. Similar to the Generative Adversarial Nets (GANs) [26] and adversarial domain adaptation [24], the discriminative model and the generative model have
(a) Training the discriminative model on labeled data.  
(b) Training the discriminative model on unlabeled data.  
(c) Training the generative model on unlabeled data.

Figure 3.6: Model architecture and training. Red boxes denote the modules being updated. “d_true” and “d_gen” are two domain tags. D is the discriminative model and G is the generative model. The objectives for the three cases are all to minimize the cross entropy loss of the answer chunks.

different objectives in our framework. However, rather than formulating the objective as an adversarial game between the two models [24, 26], in our framework, the discriminative model relies on the data generated by the generative model, while the generative model aims to match the model-generated data distribution with the human-generated data distribution using the signals from the discriminative model.

Given a labeled dataset \( L = \{p^{(i)}, q^{(i)}, a^{(i)}\}_{i=1}^N \), the objective function of a discriminative model \( D \) for a supervised learning setting can be written as
\[
\sum_{p^{(i)}, q^{(i)}, a^{(i)} \in L} \log P_D(a^{(i)} | p^{(i)}, q^{(i)})
\]
where \( P_D \) is a probability distribution defined by the model \( D \). Since we also incorporate domain tags into the model \( D \), we denote the objective function as
\[
J(L, \text{tag}, D) = \frac{1}{|L|} \sum_{p^{(i)}, q^{(i)}, a^{(i)} \in L} \log P_{D, \text{tag}}(a^{(i)} | p^{(i)}, q^{(i)})
\]
meaning that the domain tag, “tag”, is appended to the dataset \( L \). We use \( |L| = N \) to denote the number of the instances in the dataset \( L \). The objective function is averaged over all instances such that we can balance labeled and unlabeled data.

Let \( U_G \) denote the dataset obtained by generating questions on the unlabeled dataset \( U \) with the generative model \( G \). The objective of the discriminative model is then to maximize \( J \) for both labeled and unlabeled data under the domain adaptation notions, i.e.,
\[
J(L, d_{\text{true}}, D) + J(U_G, d_{\text{gen}}, D).
\]

Now we discuss the objective of the generative model. Similar to the dual learning [95] framework, one can define an auto-encoder objective. In this case, the generative model aims to generate questions that can be reconstructed by the discriminative model, i.e., maximizing \( J(U_G, d_{\text{gen}}, D) \). However, this objective function can lead to degenerate solutions because the questions can be thought of as an overcomplete representation of the answers [89]. For example, given \( p \) and \( a \), the generative model might learn to generate trivial questions such as copying the answers, which does not contributed to learning a better \( D \).
Instead, we leverage the discriminative model to better match the model-generated data distribution with the human-generated data distribution. We propose to define an adversarial training objective $J(U_G, d_{true}, D)$. We append the tag “d_true” instead of “d_gen” for the model-generated data to “fool” the discriminative model. Intuitively, the goal of $G$ is to generate "useful" questions where the usefulness is measured by the probability that the generated questions can be answered correctly by $D$.

The overall objective function now can be written as

$$\max_D J(L, d_{true}, D) + J(U_G, d_{gen}, D)$$

$$\max_G J(U_G, d_{true}, D)$$

With the above objective function in mind, we present a training algorithm in Algorithm 1 to train a GDAN. We first pretrain the generative model on the labeled data $L$ with maximum likelihood estimation (MLE):

$$\max_G \sum_{i=1}^{N} \sum_{t=1}^{T'} \log P_G(q^{(i)}_{<t} | p^{(i)}, a^{(i)})$$

where $P_G$ is the probability defined by Eq. 3.6.

We then alternatively update $D$ and $G$ based on their objectives. To update $D$, we sample one batch from the labeled data $L$ and one batch from the unlabeled data $U_G$, and combine the two batches to perform a gradient update step. Since the output of $G$ is discrete and non-differentiable, we use the Reinforce algorithm [94] to update $G$. The action space is all possible questions with length $T'$ (possibly with padding) and the reward is the objective function $J(U_G, d_{true}, D)$. Let $\theta_G$ be the parameters of $G$. The gradient can be written as

$$\frac{\partial J(U_G, d_{true}, D)}{\partial \theta_G} = E_{P_G(q|p,a)}(\log P_{D, d_{true}}(a|p, q) - b) \frac{\partial \log P_G(q|p, a)}{\partial \theta_G}$$

where we use an average reward from samples as the baseline $b$. We approximate the expectation $E_{P_G(q|p,a)}$ by sampling one instance at a time from $P_G(q|p,a)$ and then do an update step. This training algorithm is referred to as reinforcement learning (RL) training in the following sections. The overall architecture and training algorithm are illustrated in Figure 3.9.

**MLE vs RL.** The generator $G$ has two training phases–MLE training and RL training, which are different in that: 1) RL training does not require labels, so $G$ can explore a broader data domain of $p$ using unlabeled data, while MLE training requires labels; 2) MLE maximizes log $P(q|p,a)$, while RL maximizes log $P_D(a|q,p)$. Since log $P(q|a,p)$ is the sum of log $P(q|p)$ and log $P(a|q,p)$ (plus a constant), maximizing log $P(a|q,p)$ does not require modeling log $P(q|p)$ that is irrelevant to QA, which makes optimization easier. Moreover, maximizing log $P(a|q,p)$ is consistent with the goal of QA.
3.2.4 Experiments

Answer Extraction

As discussed in Section 3.2.2, our model assumes that answers are available for unlabeled data. In this section, we introduce how we use linguistic tags and rules to extract answer chunks from unlabeled text.

To extract answers from massive unlabelled Wikipedia articles, we first sample 205,511 Wikipedia articles that are not used in the training, development and test sets in the SQuAD dataset. We extract the paragraphs from each article, and limit the length of each paragraph at the word level to be less than 850. In total, we obtain 950,612 paragraphs from unlabelled articles.

Answers in the SQuAD dataset can be categorized into ten types, i.e., “Date”, “Other Numeric”, “Person”, “Location”, “Other Entity”, “Common Noun Phrase”, “Adjective Phrase”, “Verb Phrase”, “Clause” and “Other” [68]. For each paragraph from the unlabeled articles, we utilize Stanford Part-Of-Speech (POS) tagger [84] to label each word with the corresponding POS tag, and implement a simple constituency parser to extract the noun phrase, verb phrase, adjective and clause based on a small set of constituency grammars. Next, we use Stanford Named Entity Recognizer (NER) [22] to assign each word with one of the seven labels, i.e., “Date”, “Money”, “Percent”, “location”, “Organization” and “Time”. We then categorize a span of consecutive words with the same NER tags of either “Money” or “Percent” as the answer of the type “Other Numeric”. Similarly, we categorize a span of consecutive words with the same NER tags of “Organization” as the answer of the type “Other Entity”. Finally, we subsample five answers from all the extracted answers for each paragraph according to the percentage of answer types in the SQuAD dataset. We obtain 4,753,060 answers in total, which is about 50 times larger than the number of answers in the SQuAD dataset.

Settings and Comparison Methods

The original SQuAD dataset consists of 87,636 training instances and 10,600 development instances. Since the test set is not published, we split 10% of the training set as the test set, and the remaining 90% serves as the actual training set. Instances are split based on articles; i.e., paragraphs in one article always appear in only one set. We tune the hyper-parameters and perform early stopping on the development set using the F1 scores, and the performance is evaluated on the test set using both F1 scores and exact matching (EM) scores [68].

We compare the following methods. SL is the supervised learning setting where we train the model $D$ solely on the labeled data $L$. Context is the simple context-based method described in Section 3.2.2. Context + domain is the “Context” method with domain tags as described in Section 3.2.3. Gen is to train a generative model and use the generated questions as additional training data. Gen + GAN refers to the domain adaptation method using GANs [24]; in contrast to the original work, the generative model is updated using Reinforce. Gen + dual refers to the dual learning method [95]. Gen + domain is “Gen” with domain tags, while the generative model is trained with MLE and fixed. Gen + domain + adv is the approach we propose (Cf. Figure 3.9 and Algorithm 1), with “adv” meaning adversarial training based on Reinforce. We use our own implementation of “Gen + GAN” and “Gen + dual”, since the GAN model [24]
does not handle discrete features and the dual learning model [95] cannot be directly applied to question answering. When implementing these two baselines, we adopt the learning schedule introduced by Ganin and Lempitsky [24], i.e., gradually increasing the weights of the gradients for the generative model $G$.

Results and Analysis

We study the performance of different models with varying labeling rates and unlabeled dataset sizes. Labeling rates are the percentage of training instances that are used to train $D$. The results are reported in Table 3.4. Though the unlabeled dataset we collect consists of around 5 million instances, we also sample a subset of around 50,000 instances to evaluate the effects of the size of unlabeled data. The highest labeling rate in Table 3.4 is 0.9 because 10% of the training instances are used for testing. Since we do early stopping on the development set using the F1 scores, we also report the development F1. We report two metrics, the F1 scores and the exact matching (EM) scores [68], on the test set. All metrics are computed using the official evaluation scripts.

**SL v.s. SSL.** We observe that semi-supervised learning leads to consistent improvements over supervised learning in all cases. Such improvements are substantial when labeled data is limited. For example, the GDANs improve over supervised learning by 9.87 points in F1 and 7.26 points in EM when the labeling rate is 0.1. With our semi-supervised learning approach, we can use only 0.1 training instances to obtain even better performance than a supervised learning approach with 0.2 training instances, saving more than half of the labeling costs.

**Comparison with Baselines.** By comparing “Gen + domain + adv” with “Gen + GAN” and “Gen + Dual”, it is clear that the GDANs perform substantially better than GANs and dual learning. With labeling rate 0.1, GDANs outperform dual learning and GANs by 2.47 and 4.29 points respectively in terms of F1.

**Ablation Study.** We also perform an ablation study by examining the effects of “domain” and “adv” when added to “gen”. It can be seen that both the domain tags and the adversarial training contribute to the performance of the GDANs when the labeling rate is equal to or less than 0.5. With labeling rate 0.9, adding domain tags still leads to better performance but adversarial training does not seem to improve the performance by much.

**Unlabeled Data Size.** Moreover, we observe that the performance can be further improved when a larger unlabeled dataset is used, though the gain is relatively less significant compared to changing the model architectures. For example, increasing the unlabeled dataset size from 50K to 5M, the performance of GDANs increases by 0.38 points in F1 and 0.52 points in EM.

**Context-Based Method.** Surprisingly, the simple context-based method, though performing worse than GDANs, still leads to substantial gains; e.g., 7.00 points in F1 with labeling rate 0.1. Adding domain tags can improve the performance of the context-based method as well.

**MLE vs RL.** We plot the loss curve of $-J(U_G, d_{gen}, D)$ for both the MLE-trained generator (“Gen + domain”) and the RL-trained generator (“Gen + domain + adv”) in Figure 3.7. We observe that the training loss for $D$ on RL-generated questions is lower than MLE-generated questions, which confirms that RL training maximizes $\log P(a|p, q)$.

**Samples of Generated Questions.** We present some questions generated by our model in Table 3.3. The generated questions are post-processed by removing repeated subs-sequences. Compared to MLE-generated questions, RL-generated questions are more informative (Cf., P1,
Figure 3.7: Comparison of discriminator training loss $-J(U_G, d_{\text{gen}}, D)$ on generated QA pairs. The lower the better. MLE refers to questions generated by maximum likelihood training, and RL refers to questions generated by reinforcement learning.

P2, and P4), and contain less “UNK” (unknown) tokens (Cf., P1). Moreover, both semantically and syntactically, RL-generated questions are more accurate (Cf., P3 and P5).
Table 3.3: Sampled generated questions given the paragraphs and the answers. P means paragraphs, A means answers, GQ means groundtruth questions, and Q means questions generated by our models. MLE refers to maximum likelihood training, and RL refers to reinforcement learning so as to maximize $J(U_G, d_{true}, D)$. We truncate the paragraphs to only show tokens around the answer spans with a window size of 20.

| P1: | is mediated by ige, which triggers degranulation of mast cells and basophils when cross-linked by antigen. Type ii hypersensitivity occurs when antibodies bind to antigens on the patient’s own cells, marking them for destruction. This
| A: | type ii hypersensitivity
| GQ: | antibody-dependent hypersensitivity belongs to what class of hypersensitivity?
| Q (MLE): | what was the UNK of the patient’s own cells?
| Q (RL): | what occurs when antibodies bind to antigens on the patient’s own cells by antigen when cross
| P2: | An additional warming of the earth’s surface. They calculate with confidence that CO2 has been responsible for over half the enhanced greenhouse effect. They predict that under a “business as usual” (bau) scenario,
| A: | over half
| GQ: | how much of the greenhouse effect is due to carbon dioxide?
| Q (MLE): | what is the enhanced greenhouse effect?
| Q (RL): | what the enhanced greenhouse effect that CO2 been responsible for
| P3: | Narrow gauge lines, which are the remnants of five formerly government-owned lines which were built in mountainous areas.
| A: | mountainous areas
| GQ: | where were the narrow gauge rail lines built in victoria?
| Q (MLE): | what is the government government government-owned lines built?
| Q (RL): | what were the remnants of government-owned lines built in
| P4: | But not both). In 0000, bankamericard was renamed and spun off into a separate company known today as visa inc.
| A: | visa inc.
| GQ: | what present-day company did bankamericard turn into?
| Q (MLE): | what was the separate company bankamericard?
| Q (RL): | what today as bankamericard off into a separate company known today as spun off into a separate company known today
| P5: | Legrande writes that "the formulation of a single all-encompassing definition of the term is extremely difficult", if
| A: | legrande
| GQ: | who wrote that it is difficult to produce an all inclusive definition of civil disobedience?
| Q (MLE): | what is the term of a single all all all encompassing definition of a single all
| Q (RL): | what writes "the formulation of a single all-all-encompassing definition of the term all encompassing encompassing encompassing encompassing encompasses
Table 3.4: Performance with various labeling rates, unlabeled data sizes \([U]\), and methods. “Dev” denotes the development set, and “test” denotes the test set. F1 and EM are two metrics.

<table>
<thead>
<tr>
<th>Labeling rate</th>
<th>([U])</th>
<th>Method</th>
<th>Dev F1</th>
<th>Test F1</th>
<th>Test EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>50K</td>
<td>SL</td>
<td>0.4262</td>
<td>0.3815</td>
<td>0.2492</td>
</tr>
<tr>
<td>0.1</td>
<td>50K</td>
<td>Context</td>
<td>0.5046</td>
<td>0.4515</td>
<td>0.2966</td>
</tr>
<tr>
<td>0.1</td>
<td>50K</td>
<td>Context + domain</td>
<td>0.5139</td>
<td>0.4575</td>
<td>0.3036</td>
</tr>
<tr>
<td>0.1</td>
<td>50K</td>
<td>Gen</td>
<td>0.5049</td>
<td>0.4553</td>
<td>0.3018</td>
</tr>
<tr>
<td>0.1</td>
<td>50K</td>
<td>Gen + GAN</td>
<td>0.4897</td>
<td>0.4373</td>
<td>0.2885</td>
</tr>
<tr>
<td>0.1</td>
<td>50K</td>
<td>Gen + dual</td>
<td>0.5036</td>
<td>0.4555</td>
<td>0.3005</td>
</tr>
<tr>
<td>0.1</td>
<td>50K</td>
<td>Gen + domain</td>
<td>0.5234</td>
<td>0.4703</td>
<td>0.3145</td>
</tr>
<tr>
<td>0.1</td>
<td>50K</td>
<td>Gen + domain + adv</td>
<td><strong>0.5313</strong></td>
<td><strong>0.4802</strong></td>
<td><strong>0.3218</strong></td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>SL</td>
<td>0.5134</td>
<td>0.4674</td>
<td>0.3163</td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>Context</td>
<td>0.5652</td>
<td>0.5132</td>
<td>0.3573</td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>Context + domain</td>
<td>0.5672</td>
<td>0.5200</td>
<td>0.3581</td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>Gen</td>
<td>0.5643</td>
<td>0.5159</td>
<td>0.3618</td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>Gen + GAN</td>
<td>0.5525</td>
<td>0.5037</td>
<td>0.3470</td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>Gen + dual</td>
<td>0.5720</td>
<td>0.5192</td>
<td>0.3612</td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>Gen + domain</td>
<td>0.5749</td>
<td>0.5216</td>
<td>0.3658</td>
</tr>
<tr>
<td>0.2</td>
<td>50K</td>
<td>Gen + domain + adv</td>
<td><strong>0.5867</strong></td>
<td><strong>0.5394</strong></td>
<td><strong>0.3781</strong></td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>SL</td>
<td>0.6280</td>
<td>0.5722</td>
<td>0.4187</td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>Context</td>
<td>0.6300</td>
<td>0.5740</td>
<td>0.4195</td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>Context + domain</td>
<td>0.6307</td>
<td>0.5791</td>
<td>0.4237</td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>Gen</td>
<td>0.6237</td>
<td>0.5717</td>
<td>0.4155</td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>Gen + GAN</td>
<td>0.6110</td>
<td>0.5590</td>
<td>0.4044</td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>Gen + dual</td>
<td>0.6368</td>
<td>0.5746</td>
<td>0.4163</td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>Gen + domain</td>
<td><strong>0.6378</strong></td>
<td>0.5826</td>
<td>0.4261</td>
</tr>
<tr>
<td>0.5</td>
<td>50K</td>
<td>Gen + domain + adv</td>
<td>0.6375</td>
<td>0.5831</td>
<td><strong>0.4267</strong></td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>SL</td>
<td><strong>0.6611</strong></td>
<td>0.6070</td>
<td>0.4534</td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>Context</td>
<td>0.6560</td>
<td>0.6028</td>
<td>0.4507</td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>Context + domain</td>
<td>0.6553</td>
<td><strong>0.6105</strong></td>
<td>0.4557</td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>Gen</td>
<td>0.6464</td>
<td>0.5970</td>
<td>0.4445</td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>Gen + GAN</td>
<td>0.6396</td>
<td>0.5874</td>
<td>0.4317</td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>Gen + dual</td>
<td>0.6511</td>
<td>0.5892</td>
<td>0.4340</td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>Gen + domain</td>
<td><strong>0.6611</strong></td>
<td>0.6102</td>
<td><strong>0.4573</strong></td>
</tr>
<tr>
<td>0.9</td>
<td>50K</td>
<td>Gen + domain + adv</td>
<td>0.6585</td>
<td>0.6043</td>
<td>0.4497</td>
</tr>
<tr>
<td>0.1</td>
<td>5M</td>
<td>SL</td>
<td>0.4262</td>
<td>0.3815</td>
<td>0.2492</td>
</tr>
<tr>
<td>0.1</td>
<td>5M</td>
<td>Context</td>
<td>0.5140</td>
<td>0.4641</td>
<td>0.3014</td>
</tr>
<tr>
<td>0.1</td>
<td>5M</td>
<td>Context + domain</td>
<td>0.5166</td>
<td>0.4599</td>
<td>0.3083</td>
</tr>
<tr>
<td>0.1</td>
<td>5M</td>
<td>Gen</td>
<td>0.5099</td>
<td>0.4619</td>
<td>0.3103</td>
</tr>
<tr>
<td>0.1</td>
<td>5M</td>
<td>Gen + domain</td>
<td>0.5301</td>
<td>0.4703</td>
<td>0.3227</td>
</tr>
<tr>
<td>0.1</td>
<td>5M</td>
<td>Gen + domain + adv</td>
<td><strong>0.5442</strong></td>
<td><strong>0.4840</strong></td>
<td><strong>0.3270</strong></td>
</tr>
<tr>
<td>0.9</td>
<td>5M</td>
<td>SL</td>
<td>0.6611</td>
<td>0.6070</td>
<td>0.4534</td>
</tr>
<tr>
<td>0.9</td>
<td>5M</td>
<td>Context</td>
<td>0.6605</td>
<td>0.6026</td>
<td>0.4473</td>
</tr>
<tr>
<td>0.9</td>
<td>5M</td>
<td>Context + domain</td>
<td>0.6642</td>
<td>0.6066</td>
<td>0.4548</td>
</tr>
<tr>
<td>0.9</td>
<td>5M</td>
<td>Gen</td>
<td>0.6647</td>
<td>0.6065</td>
<td><strong>0.4600</strong></td>
</tr>
<tr>
<td>0.9</td>
<td>5M</td>
<td>Gen + domain</td>
<td><strong>0.6726</strong></td>
<td>0.6092</td>
<td>0.4599</td>
</tr>
<tr>
<td>0.9</td>
<td>5M</td>
<td>Gen + domain + adv</td>
<td>0.6670</td>
<td><strong>0.6102</strong></td>
<td>0.4531</td>
</tr>
</tbody>
</table>
3.3 Semi-Supervised Learning on Graphs (Completed Work, ICML 2016)

In this section, we consider semi-supervised learning on graphs. We define a generative process to generate the random walk paths on a graph, and this generative modeling objective is jointly trained with the supervised learning objective for classification on graphs.

3.3.1 Motivation

A large number of semi-supervised learning algorithms jointly optimize two training objective functions: the supervised loss over labeled data and the unsupervised loss over both labeled and unlabeled data. Graph-based semi-supervised learning defines the loss function as a weighted sum of the supervised loss over labeled instances and a graph Laplacian regularization term \[4, 92, 106, 108\]. The graph Laplacian regularization is based on the assumption that nearby nodes in a graph are likely to have the same labels. Graph Laplacian regularization is effective because it constrains the labels to be consistent with the graph structure.

Recently developed unsupervised representation learning methods learn embeddings that predict a distributional context, e.g. a word embedding might predict nearby context words \[57, 62\], or a node embedding might predict nearby nodes in a graph \[63, 81\]. Embeddings trained with distributional context can be used to boost the performance of related tasks. For example, word embeddings trained from a language model can be applied to part-of-speech tagging, chunking and named entity recognition \[16, 99\].

In this paper we consider not word embeddings but graph embeddings. Existing results show that graph embeddings are effective at classifying the nodes in a graph, such as user behavior prediction in a social network \[63, 81\]. However, the graph embeddings are usually learned separately from the supervised task, and hence do not leverage the label information in a specific task. Hence graph embeddings are in some sense complementary to graph Laplacian regularization that does not produce useful features itself and might not be able to fully leverage the distributional information encoded in the graph structure.

The main highlight of our work is to incorporate embedding techniques into the graph-based semi-supervised learning setting. We propose a novel graph-based semi-supervised learning framework, **Planetoid** (Predicting Labels And Neighbors with Embeddings Transductively Or Inductively from Data).

3.3.2 Background

Graph-Based Semi-Supervised Learning

Let \( L \) and \( U \) be the number of labeled and unlabeled instances. Let \( x_{1:L} \) and \( x_{L+1:L+U} \) denote the feature vectors of labeled and unlabeled instances respectively. The labels \( y_{1:L} \) are also given. Based on both labeled and unlabeled instances, the problem of semi-supervised learning is defined as learning a classifier \( f : x \rightarrow y \). There are two learning paradigms, transductive learning and inductive learning. Transductive learning \[106, 108\] only aims to apply the classifier \( f \) on
the unlabeled instances observed at training time, and the classifier does not generalize to unobserved instances. For instance, transductive support vector machine (TSVM) \[36\] maximizes the “unlabeled data margin” based on the low-density separation assumption that a good decision hyperplane lies on a sparse area of the feature space. Inductive learning \[4, 92\], on the other hand, aims to learn a parameterized classifier $f$ that is generalizable to unobserved instances.

In addition to labeled and unlabeled instances, a graph, denoted as a $(L+U) \times (L+U)$ matrix $A$, is also given to graph-based semi-supervised learning methods. Each entry $a_{ij}$ indicates the similarity between instance $i$ and $j$, which can be either labeled or unlabeled. The graph $A$ can either be derived from distances between instances \[108\], or be explicitly derived from external data, such as a knowledge graph \[93\] or a citation network between documents \[35\]. In this paper, we mainly focus on the setting that a graph is explicitly given and represents additional information not present in the feature vectors (e.g., the graph edges correspond to hyperlinks between documents, rather than distances between the bag-of-words representation of a document).

Graph-based semi-supervised learning is based on the assumption that nearby nodes tend to have the same labels. Generally, the loss function of graph-based semi-supervised learning in the binary case can be written as

$$L \sum_{i=1}^{L} l(y_i, f(x_i)) + \lambda \sum_{i,j} a_{ij} \|f(x_i) - f(x_j)\|^2$$

(3.7)

In Eq. (3.7), the first term is the standard supervised loss function, where $l(\cdot, \cdot)$ can be log loss, squared loss or hinge loss. The second term is the graph Laplacian regularization, which incurs a large penalty when similar nodes with a large $w_{ij}$ are predicted to have different labels $f(x_i) \neq f(x_j)$. The graph Laplacian matrix $\Delta$ is defined as $\Delta = A - D$, where $D$ is a diagonal matrix with each entry defined as $d_{ii} = \sum_j a_{ij}$. $\lambda$ is a constant weighting factor. (Note that we omit the parameter regularization terms for simplicity.) Various graph-based semi-supervised learning algorithms define the loss functions as variants of Eq. (3.7). Label propagation \[108\] forces $f$ to agree with labeled instances $y_{1:L}$; $f$ is a label lookup table for unlabeled instances in the graph, and can be obtained with a closed-form solution. Learning with local and global consistency \[106\] defines $l$ as squared loss and $f$ as a label lookup table; it does not force $f$ to agree with labeled instances. Modified Adsorption (MAD) \[80\] is a variant of label propagation that allows prediction on labeled instances to vary and incorporates node uncertainty. Manifold regularization \[4\] parameterizes $f$ in the Reproducing Kernel Hilbert Space (RKHS) with $l$ being squared loss or hinge loss. Since $f$ is a parameterized classifier, manifold regularization is inductive and can naturally handle unobserved instances.

Semi-supervised embedding \[92\] extends the regularization term in Eq. (3.7) to be $\sum_{i,j} a_{ij} \|g(x_i) - g(x_j)\|^2$, where $g$ represents embeddings of instances, which can be the output labels, hidden layers or auxiliary embeddings in a neural network. By extending the regularization from $f$ to $g$, this method imposes stronger constraints on a neural network. Iterative classification algorithm (ICA) \[75\] uses a local classifier that takes the labels of neighbor nodes as input, and employs an
iterative process between estimating the local classifier and assigning new labels.

**Learning Embeddings**

Extensive research was done on learning graph embeddings. A probabilistic generative model was proposed to learn node embeddings that generate the edges in a graph [76]. A clustering method [31] was proposed to learn latent social states in a social network to predict social ties.

More recently, a number of embedding learning methods are based on the Skipgram model, which is a variant of the softmax model. Given an instance and its context, the objective of Skipgram is usually formulated as minimizing the log loss of predicting the context using the embedding of an instance as input features. Formally, let \( \{(i, c)\} \) be a set of pairs of instance \( i \) and context \( c \), the loss function can be written as

\[
- \sum_{(i,c)} \log p(c|i) = - \sum_{(i,c)} \left( w_c^T e_i - \log \sum_{c' \in C} \exp(w_{c'}^T e_i) \right)
\]

where \( C \) is the set of all possible context, \( w \)'s are parameters of the Skipgram model, and \( e_i \) is the embedding of instance \( i \). Skipgram was first introduced to learn representations of words, known as word2vec [57]. In word2vec, for each training pair \( (i, c) \), the instance \( i \) is the current word whose embedding is under estimation; the context \( c \) is each of the surrounding words of \( i \) within a fixed window size in a sentence; the context space \( C \) is the vocabulary of the corpus. Skipgram was later extended to learn graph embeddings. Deepwalk [63] uses the embedding of a node to predict the context in the graph, where the context is generated by random walk. More specifically, for each training pair \( (i, c) \), the instance \( i \) is the current node whose embedding is under estimation; the context \( c \) is each of the neighbor nodes within a fixed window size in a generated random walk sequence; the context space \( C \) is all the nodes in the graph. LINE [81] extends the model to have multiple context spaces \( C \) for modeling both first and second order proximity.

Although Skipgram-like models for graphs have received much recent attention, many other models exist. TransE [9] learns the embeddings of entities in a knowledge graph jointly with their relations. Autoencoders were used to learn graph embeddings for clustering on graphs [82].

**3.3.3 Approach**

Following the notations in the previous section, the input to our method includes labeled instances \( x_{1:L}, y_{1:L} \), unlabeled instances \( x_{L+1:L+U} \) and a graph denoted as a matrix \( A \). Each instance \( i \) has an embedding denoted as \( e_i \).

We formulate our framework based on feed-forward neural networks. Given the input feature vector \( x \), the \( k \)-th hidden layer of the network is denoted as \( h^k \), which is a nonlinear function of the previous hidden layer \( h^{k-1} \) defined as: \( h^k(x) = \text{ReLU}(W^k h^{k-1}(x) + b^k) \), where \( W^k \) and \( b^k \) are parameters of the \( k \)-th layer, and \( h^0(x) = x \). We adopt rectified linear unit \( \text{ReLU}(x) = \max(0, x) \) as the nonlinear function in this work.

The loss function of our framework can be expressed as

\[
\mathcal{L}_s + \lambda \mathcal{L}_u,
\]
Figure 3.8: An example of sampling from context distribution \( p(i, c, \gamma) \) when \( \gamma = 1 \) and \( d = 2 \). In circles, +1 denotes positive instances, \(-1\) denotes negative instances, and \(?\) denotes unlabeled instances. If \( \text{random} < r_2 \), we first sample a random walk \( 2 \rightarrow 1 \rightarrow 4 \rightarrow 6 \), and then sample two nodes in the random walk within distance \( d \). If \( \text{random} \geq r_2 \), we sample two instances with the same labels.

where \( \mathcal{L}_s \) is a supervised loss of predicting the labels, and \( \mathcal{L}_u \) is an unsupervised loss of predicting the graph context. In the following sections, we first formulate \( \mathcal{L}_u \) by introducing how to sample context from the graph, and then formulate \( \mathcal{L}_s \) to form our semi-supervised learning framework.

**Sampling Context**

We formulate the unsupervised loss \( \mathcal{L}_u \) as a variant of Eq. (3.8). Given a graph \( A \), the basic idea of our approach is to sample pairs of instance \( i \) and context \( c \), and then formulate the loss \( \mathcal{L}_u \) using the log loss \(-\log p(c|i)\) as in Eq. (3.8). We first present the formulation of \( \mathcal{L}_u \) by introducing negative sampling, and then discuss how to sample pairs of instance and context.

It is usually intractable to directly optimize Eq. (3.8) due to normalization over the whole context space \( \mathcal{C} \). Negative sampling was introduced to address this issue [57], which samples negative examples to approximate the normalization term. In our case, we are sampling \((i, c, \gamma)\) from a distribution, where \( i \) and \( c \) denote instance and context respectively, \( \gamma = +1 \) means \((i, c)\) is a positive pair and \( \gamma = -1 \) means negative. Given \((i, c, \gamma)\), we minimize the cross entropy loss of classifying the pair \((i, c)\) to a binary label \( \gamma \):

\[
-\mathbb{I}(\gamma = 1) \log \sigma(w_c^T e_i) - \mathbb{I}(\gamma = -1) \log \sigma(-w_c^T e_i),
\]

where \( \sigma \) is the sigmoid function defined as \( \sigma(x) = 1/(1 + e^{-x}) \), and \( \mathbb{I}(\cdot) \) is an indicator function that outputs 1 when the argument is true, otherwise 0. Therefore, the unsupervised loss with negative sampling can be written as

\[
\mathcal{L}_u = -\mathbb{E}_{(i,c,\gamma)} \log \sigma(\gamma w_c^T e_i)
\]  
(3.9)
Algorithm 2 Sampling Context Distribution $p(i, c, \gamma)$

**Input:** graph $A$, labels $y_{1:L}$, parameters $r_1, r_2, q, d$

Initialize triplet $(i, c, \gamma)$

\[
\text{if } \text{random} < r_1 \text{ then } \gamma \leftarrow +1 \text{ else } \gamma \leftarrow -1
\]

\[
\text{if } \text{random} < r_2 \text{ then }
    \text{Uniformly sample a random walk } S \text{ of length } q
    \text{Uniformly sample } (S_j, S_k) \text{ with } |j - k| < d
    i \leftarrow S_j, c \leftarrow S_k
    \text{if } \gamma = -1 \text{ then uniformly sample } c \text{ from } 1 : L + U
\]

\[
\text{else }
    \text{if } \gamma = +1 \text{ then }
        \text{Uniformly sample } (i, c) \text{ with } y_i = y_c
    \text{else}
        \text{Uniformly sample } (i, c) \text{ with } y_i \neq y_c
\]

end if

end if

return $(i, c, \gamma)$

The distribution $p(i, c, \gamma)$ is conditioned on labels $y_{1:L}$ and the graph $A$. However, since they are the input to our algorithm and kept fixed, we drop the conditioning in our notation.

We now define the distribution $p(i, c, \gamma)$ directly using a sampling process, which is illustrated in Algorithm 2. There are two types of context that are sampled in this algorithm. The first type of context is based on the graph $A$, which encodes the structure (distributional) information, and the second type of context is based on the labels, which we use to inject label information into the embeddings. We use a parameter $r_1 \in (0, 1)$ to control the ratio of positive and negative samples, and use $r_2 \in (0, 1)$ to control the ratio of two types of context.

With probability $r_2$, we sample the context based on the graph $A$. We first uniformly sample a random walk sequence $S$. More specifically, we uniformly sample the first instance $S_1$ from the set $1 : L + U$. Given the previous instance $S_{k-1} = i$, the next instance $S_k = j$ is sampled with probability $a_{ij} / \sum_{j'=1}^{L+U} a_{ij'}$. With probability $r_1$, we sample a positive pair $(i, c)$ from the set $\{(S_j, S_k) : |j - k| < d\}$, where $d$ is another parameter determining the window size. With probability $(1 - r_1)$, we uniformly corrupt the context $c$ to sample a negative pair.

With probability $(1 - r_2)$, we sample the context based on the class labels. Positive pairs have the same labels and negative pairs have different labels. Only labeled instances $1 : L$ are sampled.

Our random walk based sampling method is built upon Deepwalk [63]. In contrast to their method, our method handles real-valued $A$, incorporates negative sampling, and explicitly samples from labels with probability $(1 - r_2)$ to inject supervised information.

An example of sampling when $\gamma = 1$ is shown in Figure 3.8.
Transductive Formulation

In this section, we present a method that infers the labels of unlabeled instances $y_{L+1:L+U}$ without generalizing to unobserved instances. Transductive learning usually performs better than inductive learning because transductive learning can leverage the unlabeled test data when training the model [36].

We apply $k$ layers on the input feature vector $x$ to obtain $h^k(x)$, and $l$ layers on the embedding $e$ to obtain $h^l(e)$, as illustrated in Figure 3.9a. The two hidden layers are concatenated, and fed to a softmax layer to predict the class label of the instance. More specifically, the probability of predicting the label $y$ is written as:

$$p(y|x, e) = \frac{\exp[h^k(x)^T, h^l(e)^T]w_y}{\sum_{y'}\exp[h^k(x)^T, h^l(e)^T]w_{y'}}$$

(3.10)

where $[\cdot, \cdot]$ denotes concatenation of two row vectors, the superscript $h^T$ denotes the transpose of vector $h$, and $w$ represents the model parameter.

Combined with Eq. (3.9), the loss function of transductive learning is defined as:

$$- \frac{1}{L} \sum_{i=1}^{L} \log p(y_i|x_i, e_i) - \lambda E_{(i,c,\gamma)} \log \sigma(\gamma w_c^T e_i),$$

where the first term is defined by Eq. (3.10), and $\lambda$ is a constant weighting factor. The first term is the loss function of class label prediction and the second term is the loss function of context prediction. This formulation is transductive because the prediction of label $y$ depends on the embedding $e$, which can only be learned for instances observed in the graph $A$ during training time.

Inductive Formulation

While we consider transductive learning in the above formulation, in many cases, it is desirable to learn a classifier that can generalize to unobserved instances, especially for large-scale tasks.
For example, machine reading systems [11] very frequently encounter novel entities on the Web and it is not practical to train a semi-supervised learning system on the entire Web. However, since learning graph embeddings is transductive in nature, it is not straightforward to do it in an inductive setting. Perozzi et al. [63] addressed this issue by retraining the embeddings incrementally, which is time consuming and does not scale (and not inductive essentially).

To make the method inductive, the prediction of label $y$ should only depend on the input feature vector $x$. Therefore, we define the embedding $e$ as a parameterized function of feature $x$, as shown in Figure 3.10d. Similar to the transductive formulation, we apply $k$ layers on the input feature vector $x$ to obtain $h^k(x)$. However, rather than using a “free” embedding, we apply $l_1$ layers on the input feature vector $x$ and define it as the embedding $e = h^{l_1}(x)$. Then another $l_2$ layers are applied on the embedding $h^{l_2}(e) = h^{l_2}(h^{l_1}(x))$, denoted as $h^l(x)$ where $l = l_1 + l_2$. The embedding $e$ in this formulation can be viewed as a hidden layer that is a parameterized function of the feature $x$.

With the above formulation, the label $y$ only depends on the feature $x$. More specifically,

$$p(y|x) = \frac{\exp[h^k(x)^T, h^{l_1}(x)^T]w_y}{\sum_{y'} \exp[h^k(x)^T, h^{l_1}(x)^T]w_{y'}}$$  \hspace{1cm} (3.11)

Replacing $e_i$ in Eq. (3.9) with $h^{l_1}(x_i)$, the loss function of inductive learning is

$$-\frac{1}{L} \sum_{i=1}^L \log p(y_i|x_i) - \lambda \mathbb{E}_{(i,c,\gamma)} \log \sigma(\gamma w^T h^{l_1}(x_i))$$

where the first term is defined by Eq. (3.11).

**Training**

We adopt stochastic gradient descent (SGD) [10] to train our model in the mini-batch mode. We first sample a batch of labeled instances and take a gradient step to optimize the loss function of class label prediction. We then sample a batch of context $(i, c, \gamma)$ and take another gradient step to optimize the loss function of context prediction. We repeat the above procedures for $T_1$ and $T_2$ iterations respectively to approximate the weighting factor $\lambda$. Algorithm 3 illustrates the SGD-based training algorithm for the transductive formulation. Similarly, we can replace $p(y_i|x_i, e_i)$ with $p(y_i|x_i)$ in $L_s$ to obtain the training algorithm for the inductive formulation. Let $\theta$ denote all model parameters. We update both embeddings $e$ and parameters $\theta$ in transductive learning, and update only parameters $\theta$ in inductive learning. Before the joint training procedure, we apply a number of training iterations that optimize the unsupervised loss $L_u$ alone and use the learned embeddings $e$ as initialization for joint training.

**3.3.4 Experiments**

In our experiments, Planetoid-T and Planetoid-I denote the transductive and inductive formulation of our approach. We compare our approach with label propagation (LP) [108], semi-supervised embedding (SemiEmb) [92], manifold regularization (ManiReg) [4], TSVM [36],
Algorithm 3 Model Training (Transductive)

Input: $A, \mathbf{x}_{1:L+U}, y_{1:L}, \lambda$, batch iterations $T_1, T_2$ and sizes $N_1, N_2$

repeat
  for $t \leftarrow 1$ to $T_1$ do
    Sample a batch of labeled instances $i$ of size $N_1$
    $\mathcal{L}_s = -\frac{1}{N_1} \sum_i p(y_i | \mathbf{x}_i, e_i)$
    Take a gradient step for $\mathcal{L}_s$
  end for

for $t \leftarrow 1$ to $T_2$ do
  Sample a batch of context from $p(i, c, \gamma)$ of size $N_2$
  $\mathcal{L}_u = -\frac{1}{N_2} \sum_{(i,c,\gamma)} \log \sigma(\gamma \mathbf{w}_{i}^T \mathbf{e}_i)$
  Take a gradient step for $\mathcal{L}_u$
end for

until stopping

Table 3.5: Dataset statistics.

<table>
<thead>
<tr>
<th>DATASET</th>
<th>#CLASSES</th>
<th>#NODES</th>
<th>#EDGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITESEER</td>
<td>6</td>
<td>3,327</td>
<td>4,732</td>
</tr>
<tr>
<td>CORA</td>
<td>7</td>
<td>2,708</td>
<td>5,429</td>
</tr>
<tr>
<td>PUBMED</td>
<td>3</td>
<td>19,717</td>
<td>44,338</td>
</tr>
<tr>
<td>DIEL</td>
<td>4</td>
<td>4,373,008</td>
<td>4,464,261</td>
</tr>
<tr>
<td>NELL</td>
<td>210</td>
<td>65,755</td>
<td>266,144</td>
</tr>
</tbody>
</table>

and graph embeddings (GraphEmb) [63]. Another baseline method, denoted as $\text{Feat}$, is a linear softmax model that takes only the feature vectors $\mathbf{x}$ as input. We also derive a variant $\text{Planetoid-G}$ that learns embeddings to jointly predict class labels and graph context without use of feature vectors. The architecture of Planetoid-G is similar to Figure 3.9a except that the input feature and the corresponding hidden layers are removed. Among the above methods, LP, GraphEmb and Planetoid-G do not use the features $\mathbf{x}$, while TSVM and $\text{Feat}$ do not use the graph $A$. We include these methods into our experimental settings to better evaluate our approach. Our preliminary experiments on the text classification datasets show that the performance of our model is not very sensitive to specific choices of the network architecture\footnote{We note that it is possible to develop other architectures for different applications, such as using a shared hidden layer for feature vectors and embeddings.}. We adapt the implementation of GraphEmb\footnote{https://github.com/phanein/deepwalk} to our Skipgram implementation. We use the Junto library [80] for label propagation, and SVMLight\footnote{http://svmlight.joachims.org/} for TSVM. We also use our own implementation of ManiReg and SemiEmb by modifying the symbolic objective function in Planetoid. In all of our experiments, we set the model hyper-parameters to $r_1 = 5/6$, $q = 10$, $d = 3$, $N_1 = 200$ and $N_2 = 200$ for Planetoid. We use the same $r_1$, $q$ and $d$ for GraphEmb, and the same $N_1$ and $N_2$ for ManiReg and SemiEmb. We tune $r_2, T_1, T_2$, the learning rate and hyper-parameters in other models based.
Table 3.6: Accuracy on text classification. Upper rows are inductive methods and lower rows are transductive methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Citeseer</th>
<th>Cora</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEAT</td>
<td>0.572</td>
<td>0.574</td>
<td>0.698</td>
</tr>
<tr>
<td>ManReg</td>
<td>0.601</td>
<td>0.595</td>
<td>0.707</td>
</tr>
<tr>
<td>SEMIEmb</td>
<td>0.596</td>
<td>0.590</td>
<td>0.711</td>
</tr>
<tr>
<td>PLANETOID-I</td>
<td><strong>0.647</strong></td>
<td>0.612</td>
<td><strong>0.772</strong></td>
</tr>
<tr>
<td>TSVM</td>
<td>0.640</td>
<td>0.575</td>
<td>0.622</td>
</tr>
<tr>
<td>LP</td>
<td>0.453</td>
<td>0.680</td>
<td>0.630</td>
</tr>
<tr>
<td>GRAPHEmb</td>
<td>0.432</td>
<td>0.672</td>
<td>0.653</td>
</tr>
<tr>
<td>PLANETOID-G</td>
<td>0.493</td>
<td>0.691</td>
<td>0.664</td>
</tr>
<tr>
<td>PLANETOID-T</td>
<td>0.629</td>
<td><strong>0.757</strong></td>
<td>0.757</td>
</tr>
</tbody>
</table>

The statistics for five of our benchmark datasets are reported in Table 3.5. For each dataset, we split all instances into three parts, labeled data, unlabeled data, and test data. Inductive methods are trained on the labeled and unlabeled data, and tested on the test data. Transductive methods, on the other hand, are trained on the labeled, unlabeled data, and test data without labels.

Text Classification

We first considered three text classification datasets\(^4\) Citeseer, Cora and Pubmed [75]. Each dataset contains bag-of-words representation of documents and citation links between the documents. We treat the bag-of-words as feature vectors \(x\). We construct the graph \(A\) based on the citation links; if document \(i\) cites \(j\), then we set \(a_{ij} = a_{ji} = 1\). The goal is to classify each document into one class. We randomly sample 20 instances for each class as labeled data, 1,000 instances as test data, and the rest are used as unlabeled data. The same data splits are used for different methods, and we compute the average accuracy for comparison.

The experimental results are reported in Table 3.6. Among the inductive methods, Planetoid-I achieves the best performance on all the three datasets with the improvement of up to 6.1% on Pubmed, which indicates that our embedding techniques are more effective than graph Laplacian regularization. Among the transductive methods, Planetoid-T achieves the best performance on Cora and Pubmed, while TSVM performs the best on Citeseer. However, TSVM does not perform well on Cora and Pubmed. Planetoid-I slightly outperforms Planetoid-T on Citeseer and Pubmed, while Planetoid-T gets up to 14.5% improvement over Planetoid-I on Cora. We conjecture that in Planetoid-I, the feature vectors impose constraints on the learned embeddings, since they are represented by a parameterized function of the input feature vectors. If such constraints are appropriate, as is the case on Citeseer and Pubmed, it improves the non-convex optimization of embedding learning and leads to better performance. However, if such constraints rule out the

Table 3.7: Recall@$k$ on DIEL distantly-supervised entity extraction. Upper rows are inductive methods and lower rows are transductive methods. Results marked with * are taken from the original DIEL paper [7] with the same data splits.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>RECALL@$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>*FEAT</td>
<td>0.349</td>
</tr>
<tr>
<td>MANIREG</td>
<td>0.477</td>
</tr>
<tr>
<td>SEMIEmb</td>
<td>0.486</td>
</tr>
<tr>
<td>PLANETOID-I</td>
<td>0.501</td>
</tr>
<tr>
<td>*DIEL</td>
<td>0.405</td>
</tr>
<tr>
<td>*LP</td>
<td>0.162</td>
</tr>
<tr>
<td>GRAPHEmb</td>
<td>0.258</td>
</tr>
<tr>
<td>PLANETOID-G</td>
<td>0.394</td>
</tr>
<tr>
<td>PLANETOID-T</td>
<td>0.500</td>
</tr>
<tr>
<td>*UPPER BOUND</td>
<td>0.617</td>
</tr>
</tbody>
</table>

Table 3.8: Accuracy on NELL entity classification with labeling rates of 0.1, 0.01, and 0.001. Upper rows are inductive methods and lower rows are transductive methods.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>0.1</th>
<th>0.01</th>
<th>0.001</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEAT</td>
<td>0.621</td>
<td>0.404</td>
<td>0.217</td>
</tr>
<tr>
<td>MANIREG</td>
<td>0.634</td>
<td>0.413</td>
<td>0.218</td>
</tr>
<tr>
<td>SEMIEmb</td>
<td>0.654</td>
<td>0.438</td>
<td>0.267</td>
</tr>
<tr>
<td>PLANETOID-I</td>
<td>0.702</td>
<td>0.598</td>
<td>0.454</td>
</tr>
<tr>
<td>LP</td>
<td>0.714</td>
<td>0.448</td>
<td>0.265</td>
</tr>
<tr>
<td>GRAPHEmb</td>
<td>0.795</td>
<td>0.725</td>
<td>0.581</td>
</tr>
<tr>
<td>PLANETOID-G/T</td>
<td><strong>0.845</strong></td>
<td><strong>0.757</strong></td>
<td><strong>0.619</strong></td>
</tr>
</tbody>
</table>

optimal embeddings, the inductive model will suffer.

Planetoid-G consistently outperforms GraphEmb on all three datasets, which indicates that joint training with label information can improve the performance over training the supervised and unsupervised objectives separately. Figure 3.10 displays the 2-D embedding spaces on the Cora dataset using t-SNE [87]. Note that different classes are better separated in the embedding space of Planetoid-T than that of GraphEmb and SemiEmb, which is consistent with our empirical findings. We also observe similar results for the other two datasets.

**Distantly-Supervised Entity Extraction**

We next considered the DIEL (Distant Information Extraction using coordinate-term Lists) dataset [7]. The DIEL dataset contains pre-extracted features for each entity mention in text, and a graph that connects entity mentions to coordinate lists. The goal is to extract medical entities from text
Figure 3.10: t-SNE Visualization of embedding spaces on the Cora dataset. Each color denotes a class.

given feature vectors and the graph.

We follow the exact experimental setup as in the original DIEL paper [7], including data splits of different runs, preprocessing of entity mentions and coordinate lists, and evaluation. We treat the top-$k$ entities given by a model as positive instances, and compute recall@$k$ for evaluation ($k$ is set to 240,000 following the DIEL paper). We report the average result of 10 runs in Table 3.7, whereFeat refers to a result obtained by SVM (referred to as DS-Baseline in the DIEL paper). The result of LP was also taken from [7]. DIEL in Table 3.7 refers to the method proposed by the original paper, which is an improved version of label propagation that trains classifiers on feature vectors based on the output of label propagation. We did not include TSVM into the comparison since it does not scale. Since we use Freebase as ground truth and some entities are not present in text, the upper bound of recall as shown in Table 3.7 is 0.617.

Both Planetoid-I and Planetoid-T significantly outperform all other methods. Each of Planetoid-I and Planetoid-T achieves the best performance in 5 out of 10 runs, and they give a similar recall on average, which indicates that there is no significant difference between these two methods on this dataset. Planetoid-G clearly outperforms GraphEmb, which again shows the benefit of joint training.
Entity Classification

We sorted out an entity classification dataset from the knowledge base of Never Ending Language Learning (NELL) [11] and a hierarchical entity classification dataset [18] that links NELL entities to text in ClueWeb09. We extracted the entities and the relations between entities from the NELL knowledge base, and then obtained text description by linking the entities to ClueWeb09. We use text bag-of-words representation as feature vectors of the entities.

We next describe how to construct the graph based on the knowledge base. We first remove relations that are not populated in NELL, including “generalizations”, “haswikipediaurl”, and “atdate”. In the knowledge base, each relation is denoted as a triplet \((e_1, r, e_2)\), where \(e_1\), \(r\), \(e_2\) denote head entity, relation, and tail entity respectively. We treat each entity \(e\) as a node in the graph, and each relation \(r\) is split as two nodes \(r_1\) and \(r_2\) in the graph. For each \((e_1, r, e_2)\), we add two edges in the graph, \((e_1, r_1)\) and \((e_2, r_2)\).

We removed all classes with less than 10 entities. The goal is to classify the entities in the knowledge base into one of the 210 classes given the feature vectors and the graph. Let \(\beta\) be the labeling rate. We set \(\beta\) to 0.1, 0.01, and 0.001. \(\max(\beta N, 1)\) instances are labeled for a class with \(N\) entities, so each class has at least one entity in the labeled data.

We report the results in Table 3.8. We did not include TSVM since it does not scale to such a large number of classes with the one-vs-rest scheme. Adding feature vectors does not improve the performance of Planetoid-T, so we set the feature vectors for Planetoid-T to be all empty, and therefore Planetoid-T is equivalent to Planetoid-G in this case.

Planetoid-I significantly outperforms the best of the other compared inductive methods—i.e., SemiEmb—by 4.8%, 16.0%, and 18.7% respectively with three labeling rates. As the labeling rate decreases, the improvement of Planetoid-I over SemiEmb becomes more significant.

Graph structure is more informative than features in this dataset, so inductive methods perform worse than transductive methods. Planetoid-G outperforms GraphEmb by 5.0%, 3.2% and 3.8%.

3.4 Learning Transferable Graph Structures by Language Modeling (Proposed Work)

In this section, we study learning latent graphs structure underlying natural language. Compared to the previous section, this setting has two major differences: 1) the graph structure is underlying a data point, e.g., a paragraph of text, rather than a set of data points; and 2) the graph structure is automatically learned from the data rather than given as the input. We employ language modeling as our objective, which is to generate the next token while learning a graph structure as latent variables.

3.4.1 Motivation

Recent advances in deep learning have largely relied on building blocks such as convolutional networks (CNNs) [46] and recurrent networks (RNNs) [33] augmented with attention mechanisms [2]. While possessing high representational capacity, these architectures primarily operate
on grid-like or sequential structures due to their built-in “innate priors”. As a result, CNNs and RNNs largely rely on high expressiveness to model complex structural phenomena, compensating the fact that they do not explicitly leverage structural, graphical representations.

This paradigm has led to a standardized norm in transfer learning and pretraining—fitting an expressive function on a large dataset with or without supervision, and then applying the function to downstream task data for feature extraction. Notable examples include pretrained ImageNet features [32] and pretrained word embeddings [57, 62].

In contrast, a variety of real-world data exhibit much richer relational graph structures than the simple grid-like or sequential structures. This is also emphasized by a parallel work [3]. For example in the language domain, linguists use parse trees to represent syntactic dependency between words; information retrieval systems exploit knowledge graphs to reflect entity relations; and coreference resolution is devised to connect different expressions of the same entity. As such, these exemplified structures are universally present in almost any natural language data regardless of the target tasks, which suggests the possibility of transfer across tasks. These observations also generalize to other domains such as vision, where modeling the relations between pixels is proven useful [61, 91, 103]. One obstacle remaining, however, is that many of the universal structures are essentially human-curated and expensive to acquire on a large scale, while automatically-induced structures are mostly limited to one task [40, 88, 91].

In this work, we attempt to address two challenges: 1) to break away from the standardized norm of feature-based deep transfer learning and 2) to learn versatile structures in the data with a data-driven approach. In particular, we are interested in learning transferable latent relational graphs, where the nodes of a latent graph are the input units, e.g., all the words in a sentence. The task of latent relational graph learning is to learn an affinity matrix where the weights (possibly zero) capture the dependencies between any pair of input units.

3.4.2 Proposed Approach

We propose a framework for unsupervised latent graph learning. Given a one-dimensional input \(x = (x_1, \ldots, x_T)\), where each \(x_i\) denotes an input unit at position \(t\) and \(T\) is the length of the sequence, the goal of latent graph learning is to learn a \((T \times T)\) affinity matrix \(G\) such that each entry \(G_{ij}\) captures the dependency between the unit \(x_i\) and the unit \(x_j\). The affinity matrix is asymmetric, representing a directed weighted graph. In particular, in this work we consider the case where each column of the affinity matrix sums to one, for computational convenience. In the following text, with a little abuse of notation, we use \(G\) to denote a set of affinity matrices. We use the terms “affinity matrices” and “graphs” interchangeably.

During the unsupervised learning phase, our framework trains two networks, a graph predictor network \(g\) and a feature predictor network \(f\). Given the input \(x\), our graph predictor \(g\) produces a set of graphs \(G = g(x)\). The graphs \(G\) are represented as a 3-d tensor in \(\mathbb{R}^{L \times T \times T}\), where \(L\) is the number of layers that produce graphs. For each layer \(l\), the last two dimensions \(G^l\) define a \((T \times T)\) affinity matrix that captures the dependencies between any pair of input units. The feature predictor network \(f\) then takes the graphs \(G\) and the original input \(x\) to perform a

\[\text{Throughout the paper, we use “feature” to refer to unary feature representations, and use “graph” to refer to structural, graphical representations.}\]
During the transfer phase, given an input $x'$ from a downstream task, we use the graph predictor $g$ to extract graphs $G = g(x')$. The extracted graphs $G$ are then fed as the input to the downstream task network to augment training. Specifically, we multiply $G$ with task-specific features such as input embeddings and hidden states (see Figure 3.11). The network $f$ is discarded during the transfer phase.

Next, we will introduce the network architectures and objective functions for unsupervised learning, followed by the transfer procedure. An overview of our framework is illustrated in Figure 3.12.

Unsupervised Learning

Graph Predictor The graph predictor $g$ is instantiated as two multi-layer CNNs, a key CNN, and a query CNN. Given the input $x$, the key CNN outputs a sequence of convolutional features $(k_1, \cdots, k_T)$ and the query CNN similarly outputs $(q_1, \cdots, q_T)$. At layer $l$, based on these convolutional features, we compute the graphs as

$$G_{ij}^l = \frac{\left(\text{ReLU}(k_i^l q_j^l + b)\right)^2}{\sum_{i'} \left(\text{ReLU}(k_{i'}^l q_j^l + b)\right)^2} \quad (3.12)$$

where $k_i^l = W_k^l k_i$ and $q_j^l = W_q^l q_j$. The matrices $W_k^l$ and $W_q^l$ are model parameters at layer $l$, and the bias $b$ is a scalar parameter. This resembles computing the attention weights [2] from position $j$ to position $i$ except that the exponential activation in the softmax function is replaced with a squared ReLU operation—we use ReLUs to enforce sparsity and the square operations to stabilize training. Moreover, we employ convolutional networks to let the graphs $G$ be aware of the local order of the input and context, up to the size of each unit’s receptive field.

Feature Predictor Now we introduce the feature predictor $f$. At each layer $l$, the input to the feature predictor $f$ is a sequence of features $F^{l-1} = (f_1^{l-1}, \cdots, f_t^{l-1})$ and an affinity matrix

Figure 3.11: Traditional transfer learning versus our new transfer learning framework. Instead of transferring features, we transfer the graphs output by a network. The graphs are multiplied by task-specific features (e.g. embeddings or hidden states) to produce structure-aware features.
Figure 3.12: Overview of our approach GLoMo. During the unsupervised learning phase, the feature predictor and the graph predictor are jointly trained to perform context prediction. During the transfer phase, the graph predictor is frozen and used to extract graphs for the downstream tasks. An RNN decoder is applied to all positions in the feature predictor, but we only show the one at position “A” for simplicity. “Select one” means the graphs can be transferred to any layer in the downstream task model. “FF” refers to feed-forward networks. The graphs output by the graph predictor are used as the weights in the “weighted sum” operation (see Eq. 3.13).

GLoMo extracted by the graph predictor g. The zero-th layer features F_0 are initialized to be the embeddings of x. The affinity matrix G_l is then combined with the current features to compute the next-layer features at each position t,

\[ f_l^t = v\left(\sum_j G_{jt}^l f_{l-1}^t, f_{l-1}^t\right) \]  

(3.13)

where v is a compositional function such as a GRU cell [14] or a linear layer with residual connections. In other words, the feature at each position is computed as a weighted sum of other features, where the weights are determined by the graph G^l, followed by transformation function v.

Objective Function At the top layer, we obtain the features F^L. At each position t, we use the feature f^L_t to initialize the hidden states of an RNN decoder, and employ the decoder to predict the units following x_t. Specifically, the RNN decoder maximizes the conditional log probability \( \log P(x_{t+1}, \cdots, x_{t+D}|x_t, f_t^L) \) using an auto-regressive factorization as in standard language modeling [100] (also see Figure 3.12). Here D is a hyper-parameter called the context length. The overall objective is written as the sum of the objectives at all positions t,

\[ \max \sum_t \log P(x_{t+1}, \cdots, x_{t+D}|x_t, f_t^L) \]  

(3.14)

Because our objective is context prediction, we mask the convolutional filters and the graph G (see Eq. 3.12) in the network g to prevent the network from accessing the future, following [73].
Desiderata

There are several key desiderata of the above unsupervised learning framework, which also highlight the essential differences between our framework and previous work on self-attention and predictive unsupervised learning:

**Decoupling graphs and features** Unlike self-attention [88] that fuses the computation of graphs and features into one network, we employ separate networks $g$ and $f$ for learning graphs and features respectively. The features represent the semantic meaning of each unit while the graph reflects how the units may interact. This increases the transferability of the graphs $G$ because (1) the graph predictor $g$ is freed from encoding task-specific non-structural information, and (2) the decoupled setting is closer to our transfer setting, where the graphs and features are also separated.

**Sparsity** Instead of using Softmax for attention [2], we employ a squared ReLU activation in Eq. (3.12) to enforce sparse connections in the graphs. In fact, most of the linguistically meaningful structures are sparse, such as parse trees and coreference links. We believe sparse structures reduce noise and are more transferable.

**Hierarchical graph representations** We learn multiple layers of graphs, which allows us to model hierarchical structures in the data.

**Unit-level objectives** In Eq. (3.14), we impose a context prediction objective on each unit $x_t$. An alternative is to employ a sequence-level objective such as predicting the next sentence [42] or translating the input into another language [88]. However, since the weighted sum operation in Eq. (3.13) is permutation invariant, the features in each layer can be randomly shuffled without affecting the objective, which we observed in our preliminary experiments. As a result, the induced graph bears no relation to the structures underlying the input $x$ when a sequence-level objective is employed.

**Sequence prediction** As opposed to predicting just the immediate next unit [61] [64], we predict the context of length up to $D$. This gives stronger training signals to the unsupervised learner.

Later in the experimental section, we will demonstrate that all these factors contribute to successful training of our framework.

Latent Graph Transfer

In this section, we discuss how to transfer the graph predictor $g$ to downstream tasks.

Suppose for a downstream task the model is a deep multi-layer network. Specifically, each layer is denoted as a function $h$ that takes in features $H = (h_1, \cdots, h_T)$ and possibly additional inputs, and outputs features $(h'_1, \cdots, h'_T)$. The function $h$ can be instantiated as any neural network component, such as CNNs, RNNs, attention, and feed-forward networks. This setting is general and subsumes the majority of modern neural architectures.

Given an input example $x'$ from the downstream task, we apply the graph predictor to obtain the graphs $G = g(x')$. Let $\Lambda^l = \prod_{i=1}^{l} G^i \in \mathbb{R}^{T \times T}$ denote the product of all affinity matrices from the first layers to the $l$-th layer. This can be viewed as propagating the connections among multiple layers of graphs, which allows us to model hierarchical structures. We then take a
mixture of all the graphs in $\{G^l\}_{l=1}^L \cup \{A^l\}_{l=1}^L$,

$$M = \sum_{l=1}^L m^l_G G^l + \sum_{l=1}^L m^l_A A^l, \text{ s.t. } \sum_{l=1}^L (m^l_G + m^l_A) = 1$$

The mixture weights $m^l_G$ and $m^l_A$ can be instantiated as Softmax-normalized parameters \[64\] or can be conditioned on the features $H$. To transfer the mixed latent graph, we again adopt the weighted sum operation as in Eq. (3.13). Specifically, we use the weighted sum $HM$ (see Figures 3.11 and 3.12), in addition to $H$, as the input to the function $h$. This can be viewed as performing attention with weights given by the mixed latent graph $M$. This setup of latent graph transfer is general and easy to be plugged in, as the graphs can be applied to any layer in the network architecture, with either learned or pretrained features $H$, at variable length.

**Extensions and Implementation**

So far we have introduced a general framework of unsupervised latent graph learning. This framework can be extended and implemented in various ways.

In our implementation, at position $t$, in addition to predicting the forward context $(x_{t+1}, \cdots, x_{t+D})$, we also use a separate network to predict the backward context $(x_{t-D}, \cdots, x_{t-1})$, similar to \[64\]. This allows the graphs $G$ to capture both forward and backward dependencies, as graphs learned from one direction are masked on future context. Accordingly, during transfer, we mix the graphs from two directions separately.

In the transfer phase, there are different ways of effectively fusing $H$ and $HM$. In practice, we feed the concatenation of $H$ and a gated output, $W_1[H; HM] \odot \sigma(W_2[H; HM])$, to the function $h$. Here $W_1$ and $W_2$ are parameter matrices, $\sigma$ denotes the sigmoid function, and $\odot$ denotes element-wise multiplication. We also adopt the multi-head attention \[88\] to produce multiple graphs per layer. We use a mixture of the graphs from different heads for transfer.

It is also possible to extend our framework to 2-d or 3-d data such as images and videos. The adaptations needed are to adopt high-dimensional attention \[61, 91\], and to predict a high-dimensional context (e.g., predicting a grid of future pixels). As an example, in our experiments, we use these adaptations on the task of image classification.

### 3.4.3 Preliminary Results

We preprocessed the Wikipedia dump and obtained a corpus of over 700 million tokens after cleaning html tags and removing short paragraphs. We trained the networks $g$ and $f$ on this corpus as discussed in Section 3.4.2. We used randomly initialized embeddings to train both $g$ and $f$, while the graphs are tested on other embeddings during transfer. We transfer the graph predictor $g$ to a downstream task to extract graphs, which are then used for supervised training, as introduced in Section 3.4.2.

On SQuAD, we follow the open-sourced implementation \[15\] except that we dropped weight averaging to rule out ensembling effects. This model employs a self-attention layer following the bi-attention layer, along with multiple layers of RNNs.
Table 3.9: Main results on the SQuAD dataset. Self-attention modules are included in all baseline models. All baseline methods are feature-based transfer learning methods, including ELMo and GloVe. Our methods combine graph-based transfer with feature-based transfer. Our graphs operate on various sets of features, including GloVe embeddings, ELMo embeddings, and RNN states.

<table>
<thead>
<tr>
<th>Transfer method</th>
<th>SQuAD GloVe</th>
<th></th>
<th>SQuAD ELMo</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EM</td>
<td>F1</td>
<td>EM</td>
<td>F1</td>
</tr>
<tr>
<td>transfer feature only (baseline)</td>
<td>69.33</td>
<td>78.73</td>
<td>74.75</td>
<td>82.95</td>
</tr>
<tr>
<td>GLoMo on embeddings</td>
<td>70.84</td>
<td>79.90</td>
<td><strong>76.00</strong></td>
<td><strong>84.13</strong></td>
</tr>
<tr>
<td>GLoMo on RNN states</td>
<td><strong>70.95</strong></td>
<td><strong>79.95</strong></td>
<td>75.59</td>
<td>83.62</td>
</tr>
</tbody>
</table>

The main results are reported in Table 3.9. There are three important messages. First, we have purposely incorporated the self-attention module into all of our baselines models—indeed having self-attention in the architecture could potentially induce a supervisedly-trained graph, because of which one may argue that this graph could replace its unsupervised counterpart. However, as is shown in Table 3.9, augmenting training with unsupervisedly-learned graphs has further improved performance. Second, as we adopt pretrained embeddings in all the models, the baselines establish the performance of feature-based transfer. Our results in Table 3.9 indicate that when combined with feature-based transfer, our graph transfer methods are able to yield further improvement. Third, the learned graphs are generic enough to work with various sets of features, including GloVe embeddings, ELMo embeddings, and RNN output.
Chapter 4

Timeline

The timeline of this thesis is outlined as follows:

- 2018/09 - 2018/10: finish the work *Learning Transferable Graph Structures by Language Modeling*
- 2018/11 - 2018/12: finish the work *A Faster High-Rank Language Model*
- 2019/01: thesis writing
- 2019/02: thesis defense
Bibliography


[12] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn,


[35] Ming Ji, Yizhou Sun, Marina Danilevsky, Jiawei Han, and Jing Gao. Graph regularized transductive classification on heterogeneous information networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 570–586. Springer, 2010. 41


[42] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Anto-


[74] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In International Conference on Artificial Neural Networks,


[89] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine


[106] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard

67

