Predictive Representation Learning for Language Modeling

by

Qingfeng Lan

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Language Modeling (LM) is often formulated as a next-word prediction problem over a large vocabulary, which makes it challenging. To effectively perform the task of next-word prediction, Long Short Term Memory networks (LSTMs) must keep track of many types of information. Some information is directly related to the next word's identity, but some is more secondary (e.g. discourse-level features or features of downstream words). Correlates of secondary information appear in LSTM representations, even though they are not part of an *explicitly* supervised prediction task. In contrast, Reinforcement Learning (RL) has found success in techniques that explicitly supervise representations to predict secondary information. Inspired by that success, we propose Predictive Representation Learning (PRL), which explicitly constrains LSTMs to encode specific predictions, like those that might need to be learned implicitly. By dividing the complex next-word prediction task into many simpler prediction tasks of secondary information, we show that PRL 1) significantly improves two strong language modeling methods, 2) converges more quickly, and 3) performs better when data is limited. Our fusion of RL with LSTMs shows that explicitly encoding a simple predictive task facilitates the search for a more effective language model.

To my families

All knowledge is, in final analysis, history.
All sciences are, in the abstract, mathematics.
All judgements are, in their rationale, statistics.

– Calyampudi Radhakrishna Rao

Acknowledgements

First, I would like to thank my great supervisor Alona Fyshe for her strong and consistent support. She edited my papers word by word, corrected my mistakes, and made her suggestions. She always encouraged me when I felt hopeless. I could not imagine how I can finish my master's degree without her advice and support. A special thank you to Martha White, who helped me a lot during my trip to reinforcement learning. Thank Lili Mou for being in my supervisory committee, reading my thesis and providing insightful feedback.

I would also like to thank my collaborators, colleagues, and friends for their help: Yangchen Pan, Luke Kumar, Zichen Zhang, Lei Ding, Yue Wang, Negar Hassanpour, Russell Greiner, Huizhen Yu, Yi Wan, Shangtong Zhang, Vincent Liu, and Chenyang Huang. Thank all students of Alona Fyshe for providing advice to polish my paper and presentation. Finally, I want to thank all AMII and RLAI members for their help during the past two years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Language generation is a complex task. When generating the next word in a sentence, there are multiple things to keep track of (style of writing, sentence topic, current point in a sentence's structure, etc.), and yet it remains a highly unconstrained problem. A common solution to the language generation problem is to train a Recurrent Neural Network (RNN) to predict the next word in a sequence, conditioned on the previous words.

An effective RNN must retain relevant past information, and also compute some information about what is likely to happen beyond the next word. In multiple instances, the hidden states of RNNs have been shown to encode predictions about upcoming words (plurality, grammatical number information, and subject-verb dependency [24]). But RNNs are not given any guidance on what they should remember, nor what predictions they should encode (beyond the identity of the next word). This makes the learning problem much more difficult, and difficult problems tend to require more data.

Predicting future events is a key component of the problem settings in RL. RL uses value functions to encode predictions about future rewards, and predictions secondary to the main reward can be incorporated using General Value Functions (GVFs). RL has shown that models generalize better when state representations are forced to encode secondary predictions [25, 48], a concept known as the *predictive representation hypothesis* [43]. This technique has been successful in robotics and time series prediction [38, 49, 62].

We take inspiration from RL and propose a new approach to representation learning: Predictive Representation Learning (PRL). In general, PRL breaks a complex prediction task into several more tractable prediction tasks. In turn, tackling these simpler tasks would help solve the more difficult task. Specifically, PRL uses Q-learning to train a GVF for a simple sequence labeling task (we use part-of-speech tagging, but any sequence labeling task could be used in principle). The predictions of the GVF are then incorporated into the hidden representations of a language model, allowing the language model to leverage those predictions when predicting the next word.

To summarize, our main contributions are:

- an example of how to solve sequence labeling tasks with an RL algorithm;
- PRL, an auxiliary task learning framework that uses a GVF to force predictive representations;
- evidence that PRL improves language modeling performance perplexity by up to 5%;
- evidence that PRL improves both the convergence rate and sample efficiency.

Chapter 2

Background

In this chapter, we review the background of this thesis. We first introduce Language Modeling (LM) which is the main task we use to test our method Predictive Representation Learning (PRL) in the experiments. Since reinforcement learning is applied in PRL, we then present some basic knowledge of reinforcement learning, including (general) value functions and Q-learning algorithm. Finally, we introduce general value function networks as an important component of PRL.

2.1 Language Modeling

Language modeling is the task of predicting the next word in a document. It can be the foundation of many NLP tasks, such as machine translation, sentiment analysis, text classification, and question answering. It plays a central role in understanding natural language for computers.

A language model assigns a probability to each sentence. Using the chain-rule of probability, we can factorize the probability of a sequence of words $w_1, \cdots, w_T$ into

$$p(w_1, \cdots, w_T) = p(w_1)p(w_2 \mid w_1) \cdots p(w_T \mid w_1, \cdots, w_{T-1}) \tag{2.1}$$

where T is the length of this sentence and w_i is the i-th word in the sentence. Thus, the probability of each word in the sentence is conditioned on the preceding words.

A good language model assigns a high probability $p(w_1, \cdots, w_T)$ to a sentence $w_1, \cdots, w_T$ in the corpus. To train a language model, we optimze the parameters in the model to maximize $p(w_1, \cdots, w_T)$. This is equivalent to maximze $\ln p(w_1, \cdots, w_T) = \sum_{i=1}^{T} \ln p(w_i \mid w_{1:i-1})$. However, $\sum_{i=1}^{T} \ln p(w_i \mid w_{1:i-1})$ is influenced by the length of a sentence. So we normalize it and take the natural exponential to get the definition of *perplexity*:

$$\exp\left(-\frac{1}{T}\sum_{i=1}^{T} \ln p(w_i \mid w_{1:i-1})\right) \tag{2.2}$$

where $w_{1:i-1} = (w_1, \cdots, w_{i-1})$. The perplexity measure is usually used as a quality indicator of a language model. The lower the perplexity is, the better a language model is. It is also a common loss function of neural language models.

In the following sections, we introduce some key components of modern neural language models, including word embeddings and recurrent neural networks. Finally, we introduce some examples of neural language models.

2.1.1 Word Embeddings

The earliest computer ENIAC was slow and heavy. It only knew simple calculations of numbers, far from understanding natural language. To empower a computer to understand natural language, we need to find out an efficient way to store and process language. Since words are the building blocks of a language, constructing the digital representations of words is the first step to train computers to understand natural language. There are mainly two forms of word representations. The most straightforward way is called *local* representation which represents each word with a different discrete symbol or a one-hot vector [17]. Although this form of representation is easy to understand, it suffers from the curse of dimensionality. The length of the one-hot vector is the same as the vocabulary size which usually is a huge number. For example, there are about one million words in English according to Wikipedia. The large size of one-hot vectors greatly increases the computational burden of language models. Moreover, there is no generalization between different words - knowing the information of one word does not help understand the other word because of the independency of features.

To solve these problems, Hinton [18] introduced *distributed* representations to represent a concept in the form of connection weights of neural networks. In the case of word representations, each word is represented as a word embedding which is a feature vector of real numbers. And the meaning of a word is distributed across features in this vector. It reduces computation significantly since the word embedding is usually dense and low-dimensional. The information captured from word A is stored in the word embedding dimensions which can be shared with word B for similar features. This property of word embeddings greatly improves generalization. For example, both "apple" and "orange" are fruit. We can then use one dimension in word embeddings to represent this feature so that what we learn about fruit from "apple" can be shared with "orange" as well as other fruits.

However, since word embeddings are usually learned automatically by optimizing some function (as we will see next), they lack interpretability. Although, there are some methods to interpret them, (e.g. visualizing the projection of word embeddings in 2D or 3D space to show the implicitly learned relationships between words [32], checking the similarity of two words by computing a similarity score given word embeddings with a score function [39], testing word embeddings with semantic and syntactic tasks and showing that the semantic and syntactic regularities are captured as constant vector offsets [34]), interpretation is still a hard and open problem.

The meaning of a word not only exists in itself but also exists in the interactions with other words, especially with the words nearby in sentences. The representation space of word embeddings is essentially a hyperspace and each word embedding is just a point in this space. In this scenario, learning good word embeddings is the same as putting words on the *right* locations in the representation space. Based on these ideas, many self-supervised learning methods have been developed to automatically adjust the locations of words in the representation space. In 2003, Bengio et al. [3] proposed a neural network language model (NNLM) with millions of parameters to learn a distributed representation for a large number of words. This model adjusts the word embeddings by maximizing the predicted probability of the next word given the previous words. Later, Mikolov et al. [33] proposed Word2Vec which implements two novel models, Continuous Bag-of-Words model (CBOW) and Continuous Skip-gram model (Skip-gram), for efficiently computing word embeddings from large text corpora. CBOW is trained to predict the current word based on the surrounding words. In contrast, Skip-gram predicts the surrounding words given the current word. Compared to NNLM, these two models are more efficient for learning high-quality word embeddings and they are better at capturing syntactic and semantic information as shown in a word similarity task.

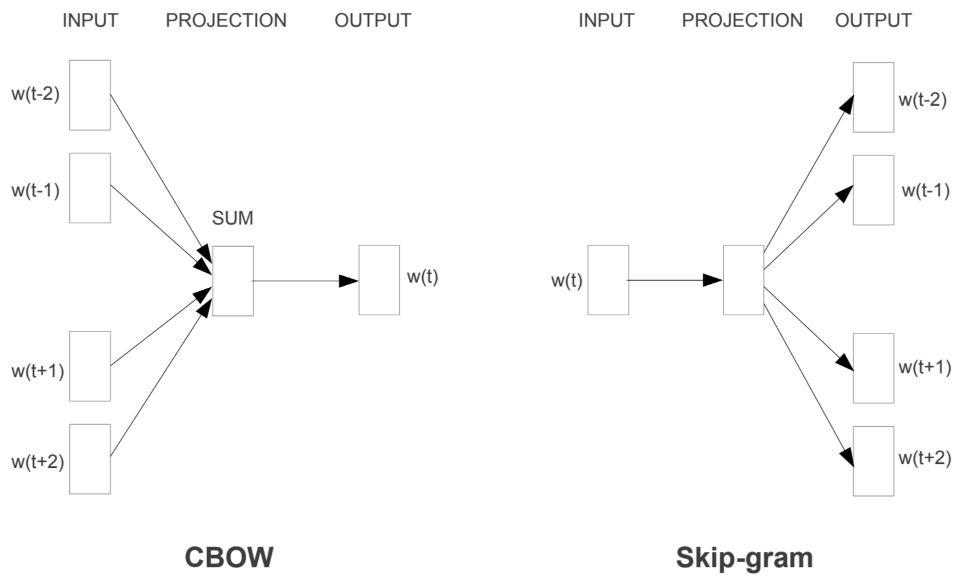

Figure 2.1: The model structures of CBOW and Skip-gram. CBOW is trained to predict the current word based on the surrounding words. In contrast, Skip-gram predicts the surrounding words given the current word. This figure is taken from [33].

Although CBOW and Skip-gram perform well on the word similarity task, they do not utilize global statistics of a text corpus since they are trained only on a small scope around a center word. To cope with this problem, Pennington et al. [41] proposed GloVe – a new weighted least squares regression model, to directly capture global corpus statistics. This model outperforms Word2Vec methods on word analogy, word similarity, and named entity recognition tasks.

2.1.2 Recurrent Neural Networks

Recently, neural networks (NN), especially deep neural networks, have been applied to many areas and achieved great success, such as computer vision [60], speech recognition [2], and reinforcement learning [37]. A traditional neural network only accepts a fixed-sized vector as an input. A Recurrent Neural Network (RNN) [11] is a special NN which accepts inputs of variable sizes. The same parameters are shared across all time steps. It is also able to capture and store long-term information. These properties make it a natural option to handle sequential data. RNNs have been applied in many NLP tasks, such as language modeling [35], machine translation [26], speech recognition [13], and part-of-speech tagging [42].

Specifically, the core of an RNN unit is a state-to-state transition function:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.3}$$

where $\mathbf{x}_t$ is the current input (e.g. the word at time t), $\mathbf{h}_t$ is the current hidden state, $\mathbf{h}_{t-1}$ is the previous hidden state, and f is usually a nonlinear function. The hidden state $\mathbf{h}$ plays the role of memory that stores long-term information and it is updated by the function f given the input and the previous hidden state.

In theory, f can be any reasonable function; in implementation, for a simple RNN, it is usually composed of an element-wise nonlinearity and an affine transformation:

$$\mathbf{h}_t = \phi(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}) \tag{2.4}$$

where $\mathbf{W}$ and $\mathbf{U}$ are two weight matrixes; ϕ is a nonlinearity, such as a sigmoid function or a hyperbolic tangent function.

Compared to simple RNNs, long short-term memory (LSTMs) [19], as an RNN variant, is better at capturing long-term information. In an LSTM unit, a memory cell $\mathbf{c}_t$ is used to store the memory content. Moreover, an input gate $\mathbf{i}_t$ and a forget gate $\mathbf{f}_t$ are employed to control how much new content to remember and how much old content to forget, resepectively. They are computed given the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1}), \tag{2.5}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1}) \tag{2.6}$$

where $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{U}_i$, and $\mathbf{U}_f$ are weight matrixes; σ is a sigmoid function. Similarly, an output gate $\mathbf{o}_t$ controls what memory content to output (i.e. $\mathbf{h}_t$):

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1}), \tag{2.7}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.8}$$

where $\mathbf{W}_o$ and $\mathbf{U}_o$ are weight matrixes for the output gate; $\odot$ is the Hadamard product.

The new memory cell $\mathbf{c}_t$ is a weighted sum of the previous memory cell $\mathbf{c}_{t-1}$ and the new content $\mathbf{g}_t$:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \tag{2.9}$$

where

$$\mathbf{g}_t = \tanh(\mathbf{W}_g \mathbf{x}_t + \mathbf{U}_g \mathbf{h}_{t-1}). \tag{2.10}$$

With the help of these gates and the memory cell, an LSTM unit is able to memorize, forget, and export the memory content adaptively.

Many one-layer RNNs can be stacked together to build a deep RNN [10, 40, 50]. The input of the lowest RNN is the original input and the output of lower RNN becomes the input of higher RNN. In theory, the hidden state of a deep RNN can capture more diverse information at different timescales.

A regular RNN only has one forward direction when processing the coming information. This means that RNNs can only utilize information up to the current frame; the future input is dropped. The power of RNNs is limited since the future information may also be useful to make a decision, improve the prediction performance, or even correct a mistake. To overcome the limitation, Schuster and Paliwal [51] proposed bidirectional recurrent neural networks (BRNNs) which have two directions – the forward direction and the backward direction. A forward pass carries the historical information before the current timestep. Similarly, a backward pass carries future information beyond the current timestep. This advantage allows BRNNs to outperform regular RNNs on many tasks [14].

2.1.3 Neural Language Models: RNNs Applied to Language

A neural language model usually consists of two parts – an encoder and a decoder. The input words are first encoded by the encoder into a list of word embeddings. The decoder then decodes the word embeddings into a probability distribution over the next word. Both the encoder and the decoder are parameterized by neural networks in the neural language model.

Bengio et al. [3] proposed a neural probabilistic language model and proved that training a large model (with millions of parameters) within a reasonable time is possible. Since a multi-layer perceptron is used to process the input words and a multi-layer perceptron only accepts a fixed length of words, they assumed that

$$p(w_i \mid w_{1:i-1}) \approx p(w_i \mid w_{i-n:i-1}) \tag{2.11}$$

where n is the size of input window.

Although this assumption simplifies the language model greatly, it also limits the power of a language model since the input information outside the input window is ignored. To fix this problem, Mikolov et al. [35] replaced the multi-layer perceptron by an RNN so that the new language model accepts variable length of words. We can train an RNN to model the probability distribution over the next word w_{t+1} given a hidden state vector $\mathbf{h}_t$ and a nonlinearity ϕ:

$$\phi(\mathbf{h}_t) = p(w_{t+1} \mid w_{1:t}). \tag{2.12}$$

Implemented with an RNN, this language model significantly outperforms standard n-gram language models (e.g. Kneser-Ney smoothed 5-gram model [20]).

In 2018, Merity et al. [30] introduced AWD-LSTM which is an LSTM based language model improved with many advanced techniques, such as DropConnect, NT-ASGD, randomized-length backpropagation through time, embedding dropout, temporal activation regularization, and activation regularization. Together these techniques greatly decrease the perplexities of AWD-LSTM on two datasets.

In practice, the capacity of softmax-based language models (i.e. models use a softmax function to generate probability distribution, such as AWD-LSTM) is limited by the softmax function, which impedes the ability to model highly context-dependent patterns in natural language. Yang et al. [63] introduce latent variables into an RNN language model and propose the *mixture of softmaxes* (MoS) method to improve the expressiveness of the softmax function. MoS improves the state-of-the-art perplexities on two language modeling benchmarks significantly.

2.2 Reinforcement Learning

In this section, we introduce some basic knowledge of reinforcement learning. In a typical reinforcement learning setting, an agent interacts with an environment by receiving scalar rewards and observing environment states. At each time step t, the agent observes a state S_t from the state space $\mathcal{S}$, and takes an action A_t chosen from the action space $\mathcal{A}$. Then it transitions to the next state $S_{t+1} \in \mathcal{S}$ according to the transition probability function $\mathrm{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and receives a scalar reward $R_{t+1} = r(S_t, A_t, S_{t+1}) \in \mathbb{R}$ where $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function. The return G_t is defined as the sum of discounted rewards,

$$G_t \doteq \sum_{k=t}^{\infty} \gamma^{k-t} R_{k+1} \tag{2.13}$$

with a discount factor $\gamma \in [0, 1]$. A policy $\pi : \mathcal{S} \to \mathcal{A}$ is a projection from a state space to an action space[1].

2.2.1 Value Functions

Given a policy π, we define the value functions which include a state-value function and an action-value function. The *state-value function* of a state s is the expected return starting from state s following the policy π, defined as

$$V^{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t | S_t = s], \forall s \in \mathcal{S} \tag{2.14}$$

where $\mathbb{E}_{\pi}[\cdot]$ is the expected value of a random variable given that the agent follows the policy π. Similarly, the *action-value function* of a state-action pair (s, a) is the expected return starting from s, taking the action a, following the policy π, defined as

$$Q^{\pi}(s, a) \doteq \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a], \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{2.15}$$

The goal of the agent is to find an optimal policy that maximizes the expected return. We first define the optimal state-value function as the largest value among state-values following all possible policies

$$V^{*}(s) \doteq \max_{\pi} V^{\pi}(s) \tag{2.16}$$

for all $s \in \mathcal{S}$. Similarly, we define the optimal action-value function as

$$Q^{*}(s, a) \doteq \max_{\pi} Q^{\pi}(s, a) \tag{2.17}$$

for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. A greedy policy with respect to Q^{*} is known to be an optimal policy. We denote this optimal policy as π^{*} and

$$\pi^{*}(s) = \arg\max_{a \in \mathcal{A}} Q^{*}(s, a) \tag{2.18}$$

[1]We only consider deterministic policies in this thesis. In general, policies can be stochastic which project state-action pairs to probabilities, i.e. $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$

for all $s \in \mathcal{S}$.

Note that in Eq. 2.13 the discount factor γ controls the present value of the future reward. At time step $t + k$, reward R_{t+k+1} is discounted by γ^k. The smaller γ is, the lower the present value of the future reward is. Assume all rewards are 1, $G_t \doteq \sum_{k=t}^{\infty} \gamma^{k-t} R_{k+1} = \frac{1}{1-\gamma}$. We then use $\frac{1}{1-\gamma}$ to represent how far the agent can see in the future. For $\gamma = 0$, $\frac{1}{1-\gamma} = 1$ and the agent is myopic, and only focuses on the immediate reward. When $\gamma \to 1$, $\frac{1}{1-\gamma} \to \infty$, the agent is farsighted and treats all future rewards equally. In other words, value functions encode information about future rewards and the importance of the information is controlled by the discount factor γ.

2.2.2 General Value Functions

General Value Functions (GVFs) are well named, as they generalize value functions. In a GVF, the discount factor is replaced by a termination function $\gamma : \mathcal{S} \to [0, 1]$ so that a different discount can be applied for each state. In a GVF, the reward signal $R_t \in \mathbb{R}$ is replaced with the more general cumulant signal $C_t \in \mathbb{R}$. The cumulant signal can be any scalar signal which allows an agent to predict not only the sum of rewards but the sum of any scalar signal. Formally, we define the target as

$$G_t(\gamma, C) \doteq \sum_{k=t}^{\infty} \left(\prod_{i=t+1}^{k} \gamma(S_i) \right) C_{k+1}. \tag{2.19}$$

Note that when the termination function γ is a constant mapping and $C_t = R_t$, we get the original definition of return (Eq. 2.13). Using the target $G_t(\gamma, C)$, the general state-value function is written as

$$V^{\pi}_{\gamma,C}(s) \doteq \mathbb{E}_{\pi}[G_t(\gamma, C) | S_t = s] \tag{2.20}$$

for all $s \in \mathcal{S}$, given a policy π. Similarly, the action-value function is written as

$$Q^{\pi}_{\gamma,C}(s, a) \doteq \mathbb{E}_{\pi}[G_t(\gamma, C) | S_t = s, A_t = a] \tag{2.21}$$

for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. In the following, for compactness, we use notations $V(s)$ and $Q(s, a)$ as simplifications of $V^{\pi}_{\gamma,C}(s)$ and $Q^{\pi}_{\gamma,C}(s, a)$. For implementation (Chapter 4), we only consider the case that γ is a constant function for simplicity. We denote this constant as γ in an abuse of notation.

2.2.3 Q-learning

Q-learning [61] has succeeded in solving many RL problems. It attempts to learn the optimal action-value function Q^*. It solves the Bellman equation

$$Q^*(s,a) = \mathbb{E}\left[Y_t \middle| S_t = s, A_t = a\right] \tag{2.22}$$

by dynamic programming where

$$Y_t \doteq R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q^*(S_{t+1}, a'). \tag{2.23}$$

Once we get the solution $Q^*(s,a)$, we can get an optimal policy easily by selecting an action from $\arg\max_{a \in \mathcal{A}} Q^*(s,a)$ for each state s. This Q function is usually parameterized with a neural network $Q(s,a;\theta)$ where θ are the parameters and can be optimized with Stochastic Gradient Descent (SGD) by minimizing the square loss

$$\frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} (Y_t(a) - Q(S_t, a; \theta))^2 \tag{2.24}$$

where

$$Y_t(a) = R_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a'; \theta). \tag{2.25}$$

Following [56], we fix $Y_t(a)$ during an update. So instead of updating θ by full gradients, they are updated by semi-gradients, i.e.

$$\theta \leftarrow \theta + \alpha \frac{2}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} [Y_t(a) - Q(S_t, a; \theta)] \nabla Q(S_t, a; \theta) \tag{2.26}$$

where α is the learning rate.

2.2.4 General Value Function Networks

The General Value Function Network (GVFN), a variant of RNNs, was proposed by [49]. It was inspired by the predictive representation hypothesis, which posits that constraining the hidden states of models to be predictions leads to better generalization. In a GVFN, each component of the hidden state is a prediction about the future in the form of a GVF. At each step t, each state component is updated towards the target $G_t(\gamma, C)$. For example, the goal of an air quality agent is to keep the air in a building at a stable state, no matter how the outside environment changes. To do so, the agent may want to predict aspects of the outside environment, such as temperature and humidity. In principle, GVFN can be applied to predict these values. Forcing hidden states to be GVFs also prevents overfitting to current data, and regularizes by reducing the hypothesis state space [49].

GVFNs can be combined with different RNNs. For example, GVFs can be combined with an LSTM to create a GVFN-LSTM. Similarly, we can create a GVFN-GRU or a GVFN-RNN.

2.3 Conclusion

We have explained word embeddings, recurrent neural networks, and how to apply them in neural language models. We also explained some basic concepts in RL, including value functions, general value functions, Q-learning algorithm, and general value function networks.

In the next chapter, we will present our method PRL and show how these concepts are applied together in PRL.

Chapter 3

Predictive Representation Learning

In this chapter, we present our main method which learns a predictive representation with the help of GVFNs. We first show that sequence labeling tasks can be formalized as RL problems and then introduce a key component of our method, called label trace. Finally, we propose a new RNN model called Predictive Representation Learning (PRL) to solve the language modeling task.

3.1 Sequence Labeling as an RL Problem

Many tasks in NLP are sequence labeling tasks, such as Part-Of-Speech (POS) tagging, named entity recognition, and chunking. We use POS tagging as an example to illustrate how a sequence labeling task is transformed into an RL problem.

"I am going to school." is the example sentence we want to label. The correct POS labels are "I (Pronoun) am (Verb) going (Verb) to (Preposition) school (Noun)." We set the state space $\mathcal{S}$ to be the vocabulary set, i.e. $\mathcal{S} = \{I, am, going, to, school, \dots\}$. The action space $\mathcal{A}$ is the set of all possible POS labels, such as $\mathcal{A} = \{Pronoun, Verb, Preposition, Noun, \dots\}$. The example sentence together with the correct labels can be represented as a finite Markov Decision Process (MDP) as shown in Fig. 3.1. When a correct POS label for the current word is chosen, it leads to the next state with cumulant 1;

otherwise, it goes to the terminal state (T) with cumulant 0. For example, in state “am” if action “Verb” is taken, the agent goes to the next state (“going”); otherwise it transitions to the terminal state.

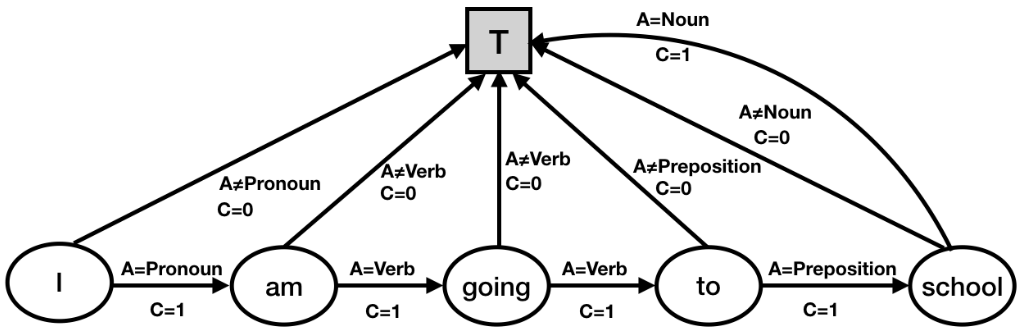

Figure 3.1: An MDP representation of the sentence example for POS tagging. Only when the correct label is selected does the MDP move to the next state, receiving cumulant of 1. All incorrect labels result in a transition to the terminal state (T) with cumulant 0.

Recall that in general, $Q(s, a)$ is the expected return starting from state s, taking the action a in RL. In this setting, $Q(s, a)$ is the expected discounted number of steps that a policy using $Q(s, a)$ will be able to correctly label the next word, from the current state s if it took labeling action a. If this number is large, it suggests that a is the correct label. Additionally, the magnitudes of $Q(s, a)$ for the possible actions provide information about ambiguity: if all the other actions have $Q(s, a) = 0$, then the model is quite sure that the maximal action is the right label; otherwise some of the other actions could be considered. The goal is to find an optimal policy that maximizes the expected target, by choosing the correct label for each word. Following this pattern, other sequence labeling tasks can also be formulated as RL problems.

3.2 Label Trace

In Section 2.2, we showed how value functions and general value functions encode future information which is essential to predict signals in the future. Here, we point out that historical information is also important for doing predictions. In this section, we introduce label traces as a way to encode history.

Label traces encode the labels (e.g. POS tags) of words already seen in the sentence. Let $A_1, \ldots, A_t$ be the label sequence chosen by the optimal policy. The label trace $T_t(a)$ for $a \in \mathcal{A}$ is the exponentially discounted count of a over the already observed words. Formally, we define

$$T_t(a) \doteq \sum_{k=1}^{t} \gamma^{t-k} \mathbb{1}(A_k = a) \tag{3.1}$$

where $\gamma \in [0, 1]$ is a discount factor and $\mathbb{1}$ is an indicator function.

3.3 Predictive Representation Learning

In this section, we develop a new RNN model with the encoder-decoder structure called Predictive Representation Learning (PRL). In general, PRL is a framework designed for auxiliary task learning. Here, we consider the case where Language Modeling (LM) is the main task, and POS tagging is the auxiliary task. The goal of LM is to predict the next word. Though in principle, any sequencing labeling task could be used, we experimented with POS tagging because it is complementary to LM, but has a smaller prediction space. Though POS taggers usually predict the POS label of the current word (w_t), to align with LM, our POS tagger predicts the POS label for the *next* word (w_{t+1}) without observing it. We present our model in Fig. 3.2. It mainly consists of three parts - the shared encoder, the POS decoder, and the LM decoder.

Given an input sentence and the true labels, we transform the sentence into an MDP as shown in Section 3.1. Then we feed the sentence as a list of words into the **shared encoder** (the red rectangle in Fig. 3.2) where each word w is encoded to a word vector e (the green rectangle). The encoder is an embedding layer (i.e. a lookup table that stores word vectors of a vocabulary set).

In our experiments, the **POS decoder** is a 2-layer RNN. We concatenate the label trace vector T (the pink square in Fig. 3.2) with the word vector

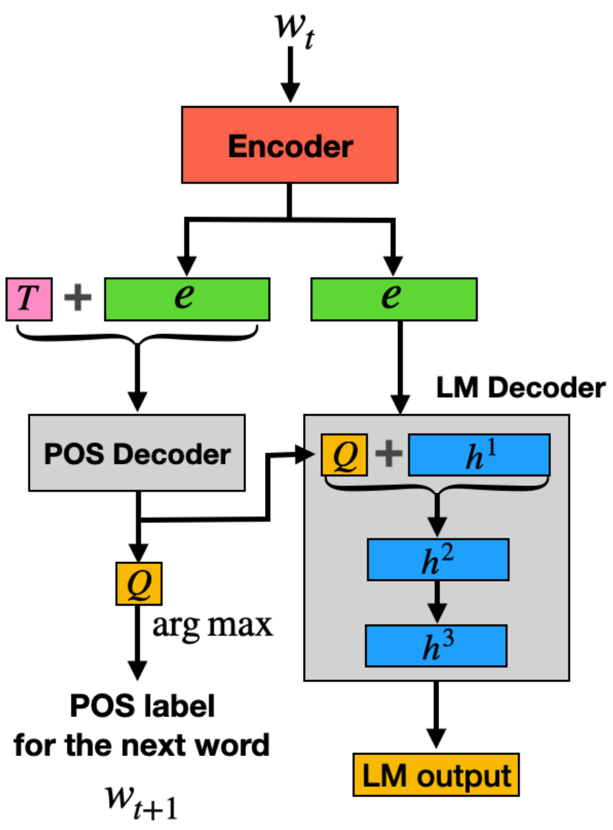

Figure 3.2: The model structure of PRL. The encoder (the red rectangle) is shared by both tasks while each task has its own decoder. Each word w is encoded by the shared encoder into a word vector e. The label trace T (the pink square) is concatenated with the word vector e to form a new word vector $T + e$. The POS decoder decodes $T + e$ into an action-value vector Q (the yellow square). For the LM decoder, the action-value vector Q is concatenated with the first hidden state h^1 inside the LM decoder to form a new hidden state $Q + h^1$. The multi-layer LSTM proceeds as is typical in LM.

(e) to form a new word vector (i.e. $T + e$) which contains historical label information. The POS decoder decodes $T + e$ into an action-value vector Q (the yellow square). Each element in Q is the action-value for taking an action $a \in \mathcal{A}$ where $\mathcal{A}$ is the POS label set. Thus, the length of this vector is the number of actions in $\mathcal{A}$. The action-value for each label represents an estimate of the goodness of that label. To generate the predicted label, we take $\arg\max$ over these action-values.

In this setting, the training of a POS tagger is equivalent to finding an optimal policy to maximize the expected target (as shown in Section 3.1). We use Q-learning [61] to learn an optimal policy, but other RL algorithms could also be applied.

The **LM decoder** is a 3-layer LSTM. After the POS decoder computes the action-value vector Q, it is then concatenated with the first hidden state h^1 inside the LM decoder to form a new hidden state $Q + h^1$. This new hidden state is the input to the next LSTM layer. The rest of the computation proceeds as is typical in a multi-layer LSTM, and produces a prediction for the next word. Following previous work, cross-entropy is the loss function.

LM is strongly related to POS tagging [16]. With access to the predicted POS label of the next word, a language model may be able to reduce the number of probable next words under consideration, thus improving accuracy.

3.4 Conclusion

In this chapter, we showed how to transform a sequence labeling problem to an RL problem and solve it with an RL algorithm. We also introduced label traces to encode history that helps prediction. Finally, we proposed our model PRL. In the next chapter, we will conduct experiments to test PRL and show the superiority of PRL compared to baseline methods.

Chapter 4

Experiments

In this chapter, we perform our experiments on two widely used benchmarks - Penn Treebank (PTB) [28] and WikiText-2 (WT2) [31]. For the following experiments, LM is the main task while POS tagging is the auxiliary task. To align with the goal of LM, we change the goal of POS tagging to predicting the correct POS label for the *next* word instead of the *current* word.

For PTB, we used the version pre-processed by [36]. This dataset contains 929k training words, 73k validation words, and 82k test words. It has 10,000 unique words, and all other words are replaced by $\langle unk \rangle$ tokens. We extracted the corresponding POS labels from the original PTB dataset.

WT2 is created from Wikipedia articles with over 30,000 words. It is about twice the size of the PTB dataset and contains 2088k training words, 217k validation words, 245k test words. Since the original WT2 dataset has no POS labels, we used the English POS tagger in spaCy [1] to label sentences in WT2.

In the following sections, we first combine PRL with two state-of-the-art language models and show that the perplexities are reduced significantly. We also find out that PRL accelerates the learning process and improves language models trained with smaller datasets, by analyzing the convergence rate and the influence of dataset size on PRL. Then we report the performance of GVFN-LSTM on POS tagging. Furthermore, we show the effect of POS tagging accuracy on PRL which hints at a promising future for

[1]https://github.com/explosion/spaCy

PRL. Finally, we do a model ablation analysis to show the importance of the label trace component in PRL.

4.1 PRL's Effect on LM

In this section, we show that the performance of LM can be further improved when predictive representations are incorporated. For the following experiments, LM is the main task while POS tagging is the auxiliary task.

We first integrate a GVF as an auxiliary task into ASGD Weight-Dropped LSTM (AWD-LSTM) [30]. In this thesis, we use AWD as a shorter name for AWD-LSTM.

Our model that combines AWD with PRL is called AWD-PRL-Q in which Q stands for the action-values.

To fairly assess the contributions of AWD-PRL-Q, we needed a baseline that also had access to next-word POS predictions. We call this baseline AWD-PRL-P in which P stands for the probability distribution over the POS label set generated by the POS decoder. AWD-PRL-P is derived from AWD-PRL-Q by replacing the action-values Q with the probability distribution P produced by an LSTM POS tagger, which still has access to the label trace T. We will compare the effect of these two representations – the action-values versus probability distribution – on LM performance.

For the LM decoder, we used the same hyperparameters and optimization settings provided in the official codebases [2]. The POS decoder is a 2-layer LSTM (for AWD-PRL-P) or GVFN-LSTM (for AWD-PRL-Q) with hidden size 380 and a linear layer neural network.

For AWD-PRL-Q, the parameters in the POS decoder were optimized by the square loss $\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}(Y_t(a) - Q(S_t, a; \theta))^2$ defined in Section 2.2.3. The parameters in AWD-PRL-P were optimized by cross-entropy loss. The best learning rate was selected from {30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6} based on validation performance. The discount factor γ in AWD-PRL-

[2] https://github.com/salesforce/awd-lstm-lm

P and AWD-PRL-Q was chosen from {0, 0.5, 0.67, 0.8, 0.9, 0.99}. During training, we select a task from LM and POS tagging in turn. A batch of data samples is used to update the parameters in the corresponding decoder and the shared encoder by gradient descent. The whole process is repeated until convergence. Note that all samples in a batch have a same length by cutting off long sentences or pending short sentences.

We trained all models for 5 different random initializations. We used 500 epochs for PTB and 750 epochs for WT2 following [30]. The runtime and the size of each model are in the appendix. Following [63], we removed finetuning to reduce training time. Models were evaluated using perplexity (PPL, lower is better). We reproduced the original results with AWD's official codebase.

Table 4.1 shows the averaged results of all models over 5 runs as well as the standard errors. Note that our reproduced results are slightly worse than the results reported in [30] since we report the averaged perplexities over 5 runs without further finetuning the models. Results with and without neural cache [12] are both reported.

On both datasets, AWD-PRL-P and AWD-PRL-Q outperform AWD, with or without applying neural cache. This supports our hypothesis that constraining the learned representation to be simple predictions improves model performance. It also confirms that predicted POS tags improve the performance of LM, whether the information itself is represented by a probability distribution or action-values. Furthermore, the perplexities of AWD-PRL-Q are consistently lower than the perplexities of AWD-PRL-P, and so GVFN-LSTMs outperform LSTMs in this setting. The Q values' representation of future information (as a form of predictive representations) is more beneficial to LM than simple probabilities (P).

To test the generalization of PRL, we incorporated it into another language model, Mixture of Softmaxes (AWD-LSTM-MoS) [63]. In this thesis, we use MoS as a shorter name for AWD-LSTM-MoS.

Our model which combines PRL with MoS is called MoS-PRL-Q.

We used a similar training process and hyperparameter tuning as in the

Dataset	Model	No Neural Cache		Neural Cache	
		Test PPL	Δ	Test PPL	Δ
PTB	AWD	58.48 ± 0.06	/	54.36 ± 0.07	/
PTB	AWD-PRL-P	56.31 ± 0.06	2.17	52.01 ± 0.05	2.35
PTB	AWD-PRL-Q	55.90 ± 0.06	2.58	51.90 ± 0.06	2.46
WT2	AWD	68.85 ± 0.11	/	54.24 ± 0.07	/
WT2	AWD-PRL-P	67.52 ± 0.06	1.33	52.86 ± 0.05	1.38
WT2	AWD-PRL-Q	66.75 ± 0.07	2.10	52.58 ± 0.05	1.66

Table 4.1: The comparison of test perplexities on PTB and WT2 for LM based on AWD. The Δ columns show the improvements of AWD-PRL-P/Q in terms of perplexity, compared with AWD. We report results with and without neural cache. All results were averaged over 5 runs with the standard errors reported. PRL improves the performance of AWD, with and without neural cache.

Dataset	Model	No Dyneval		Dyneval	
		Test PPL	Δ	Test PPL	Δ
PTB	MoS	56.44 ± 0.06	/	49.99 ± 0.05	/
PTB	MoS-PRL-Q	53.88 ± 0.06	2.56	48.21 ± 0.04	1.78
WT2	MoS	64.92 ± 0.18	/	44.08 ± 0.07	/
WT2	MoS-PRL-Q	64.39 ± 0.14	0.53	44.13 ± 0.08	-0.05

Table 4.2: The comparison of test perplexities on PTB and WT2 for LM based on MoS. The Δ columns show the improvements of MoS-PRL-Q in terms of perplexity, compared with MoS. We reported results with dynamic evaluation (Dyneval) and without (No Dyneval). All results were averaged over 5 runs with the standard errors reported. Overall, PRL improves the performance of MoS.

previous experiment for AWD. Models were trained to convergence for 800 epochs on both datasets. The runtime and the size of each model are in the appendix. We reproduced the original results with hyperparameters provided in the official codebase [3]. For similar reasons to the case of AWD, our reproduced results are slightly worse than the results reported [63].

Table 4.2 shows the averaged results of the two models over 5 runs as well as the standard errors. Results with dynamic evaluation [21] (column Dyneval) and without dynamic evaluation (column No Dyneval) are reported. MoS-PRL-Q outperforms MoS on PTB and WT2 consistently without dynamic evaluation. With dynamic evaluation, the performance gaps of two models narrow. Overall, the results show that PRL improves the performance of LM.

In the experiments, it is found that a small discount factor (<0.9) usually leads to a better performance. This indicates that a good agent may not need to look ahead for a large number of steps. The relation between the label of the current word and the label of a word far away is just too weak to take it into account.

4.2 Analysis of Convergence Rate and Influence of Dataset Size

We have shown that PRL improves the performance of the final trained language models. Because PRL helps to guide representation learning, we hypothesized it would also improve the convergence rate and sample efficiency. Fig. 4.1 shows the validation PPL while training AWD and AWD-PRL-Q on PTB. Notice that the PPL of AWD after 500 epochs is achieved by AWD-PRL-Q after 100 epochs. The results for WT2 are similar, as shown in Figure 4.2.

PRL helps language models converge faster, but what is convergence like with less data? To study the influence of dataset size on LM performance, we trained AWD and AWD-PRL-Q with a subset of the training data. We

[3]https://github.com/zihangdai/mos

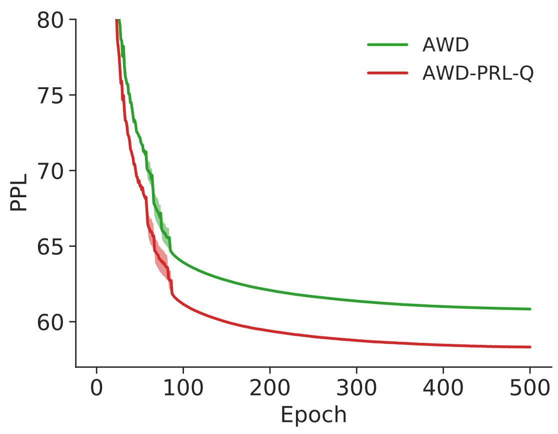

Figure 4.1: The validation perplexity for AWD and AWD-PRL-Q on Penn Treebank during training. Average over 5 runs is shown, shaded regions represent standard errors. AWD-PRL-Q converges faster than AWD.

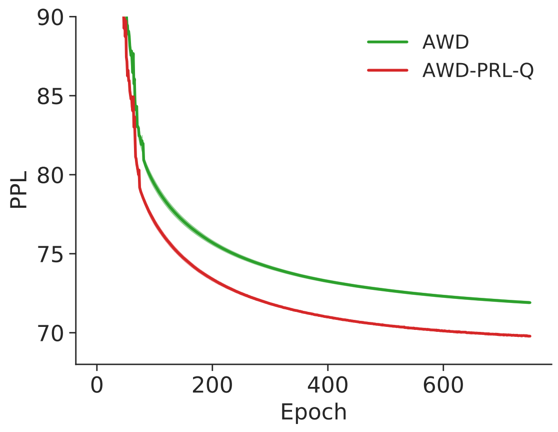

Figure 4.2: The validation perplexity for AWD and AWD-PRL-Q on WikiText2 during training. Average over 5 runs is shown, shaded regions represent standard errors. AWD-PRL-Q converges faster than AWD.

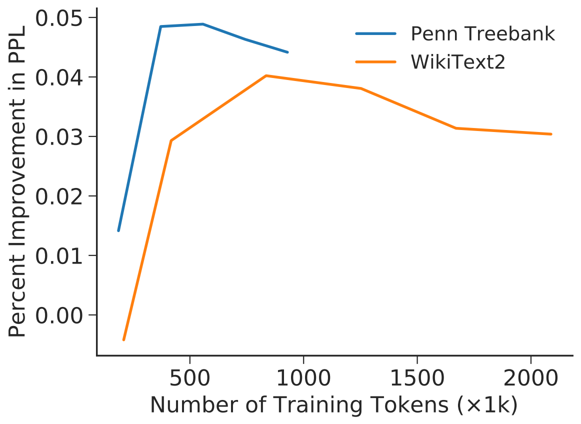

Figure 4.3: The percent change in perplexity of AWD-PRL-Q over AWD on the test set of Penn Treebank and WikiText2, as a function of dataset size (measured by the number of training tokens). Average over 5 runs is reported. PRL improves language models trained with datasets at different scales.

created several smaller PTB and WT2 datasets and trained models on them. In Fig. 4.3, we see improvements on PTB, even when we include only 250k tokens. The performance of AWD-PRL-Q on WT2 with 250k tokens is slightly worse than AWD, but improvements are seen with just slightly more tokens. The percent improvement in PPL can approach 5%, and appears to stabilize at 3% even as the dataset size surpasses 2M tokens. Thus, PRL can improve language models trained with smaller datasets, and even when the datasets are quite large, there still exists a significant improvement.

4.3 Performance of the GVFN-LSTM

We were also interested in the performance of a GVFN-LSTM POS tagger, and the impact of the label trace on POS tagging accuracy. To study this, we ignored the LM decoder part from AWD-PRL-P and AWD-PRL-Q after we trained them as shown in Section 4.1, and had two models – POS-LSTM (the baseline model, derived from AWD-PRL-P) and POS-PRL (our method, derived from AWD-PRL-Q).

To test the impact of the label trace on POS tagging, we first removed the label trace components from AWD-PRL-P and AWD-PRL-Q and got two models, denoted as AWD-PRL-P (no T) and AWD-PRL-Q (no T) respectively. We trained two models the same as we did in Section 4.1, and then ignored the LM decoder part. Thus we had two baseline models from AWD-PRL-P (no T) and AWD-PRL-Q (no T), denoted as POS-LSTM (no T) and POS-PRL (no T) respectively).

Table 4.3 shows the test performance of POS-LSTM and POS-PRL on PTB and WT2, with and without the label trace. All results were averaged over 5 runs with the standard errors reported.

Dataset	Model	Test accuracy
PTB	POS-LSTM	50.23% ± 0.03%
PTB	POS-LSTM (no T)	48.45% ± 0.02%
PTB	POS-PRL	44.56% ± 0.02%
PTB	POS-PRL (no T)	42.40% ± 0.06%
WT2	POS-LSTM	49.96% ± 0.01%
WT2	POS-LSTM (no T)	48.38% ± 0.04%
WT2	POS-PRL	43.97% ± 0.02%
WT2	POS-PRL (no T)	42.28% ± 0.03%

Table 4.3: The comparison of POS tagging accuracy on PTB and WT2 using POS-LSTM and POS-PRL. The incomplete models without the label trace components are denoted as LSTM (no T) and PRL (no T). All results were averaged over 5 runs with the standard errors reported.

Notice that predicting the POS label for the next word is much harder than predicting the POS label for the current word. The accuracy of POS tagging for the next word is barely 50% while the accuracy of POS tagging for the current word is close to 100% [5].

The ablation experiments also show that the label trace component improves the POS prediction significantly. This suggests that forcing the representation to explicitly encode historical information improves future prediction. Although both POS-LSTM and POS-PRL can *learn to* extract and retain historical information, the explicit encoding of history still helps.

Note that POS-LSTM outperforms POS-PRL on both datasets. In Section 4.1, we showed that AWD-PRL-Q (a model that uses predictions from POS-

PRL) is better than AWD-PRL-P (a model that uses predictions from POS-LSTM). Thus, AWD performs best when supported by a less accurate POS tagger. As we will see, this also contradicts the results in Section 4.4, which show that higher POS tagging accuracy leads to better performance of LM. Recall that action-values from AWD-PRL-Q encode not only the prediction of the immediate next POS label, but all labels following the current word. In comparison, the probability distribution in AWD-PRL-P only tells us about the immediate *next* POS label. Perhaps it is AWD-PRL-Q's additional future predictions that provide the performance boost.

4.4 Effect of POS Tagging Accuracy on PRL

We have shown that the performance of LM is improved by PRL. Note that this is true even though the accuracy of the underlying POS tagging is as low as 50%. This inspired us to ask: what would the impact of PRL be if the POS tagging was more accurate? How does the LM improve as POS tagging improves?

Current state-of-the-art methods for predicting a POS label given the representation of the current word (i.e. the word to be tagged) are extremely accurate [5]. Recall that our model predicts the POS label of the next word without access to that word's embedding. We leveraged these two facts in order to study the effect of POS tagging accuracy on MoS-PRL-Q. We developed a "cheating" model, wherein the POS tagger, trained to predict the POS label of the next word, actually has access to the embedding of that next word (rather than access only to the current word). This is clearly cheating because some information of the next word is leaked to the LM decoder through the POS action-values. But, because the upper limit of these cheating predictions can approach 90%, we can more fully experiment with the impact of POS tagging accuracy on MoS-PRL-Q. To control the accuracy of POS tagging, we changed the size of the hidden layers in the POS decoder (smaller layers leads to lower performance). Note that we trained the cheating model with only *10* epochs.

Fig. 4.4 shows how PPL changes as the accuracy of POS tagging changes. The PPL decreases almost linearly as the accuracy of POS tagging increases to 80%. This again strongly supports our hypothesis that constraining representations to be predictive can improve the performance of LM. Moreover, it hints at a promising future for PRL. As we improve the accuracy of POS tagging or incorporate other sequence labeling tasks with higher accuracy, we can expect the performance of LM to improve too.

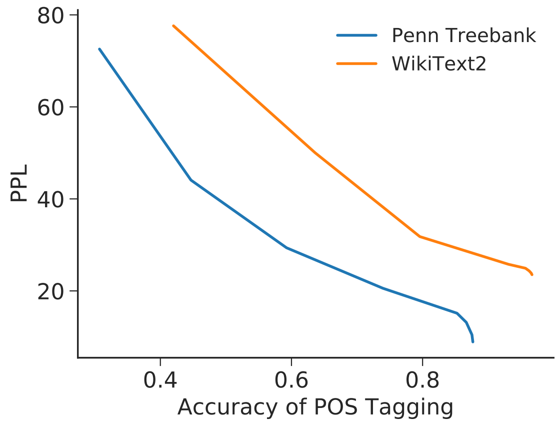

Figure 4.4: The relation between the accuracy of POS tagging and perplexity of LM on Penn Treebank and WikiText2. The perplexity drops almost linearly as the POS tagging accuracy increases.

4.5 Ablation Analysis of Label Trace

We have shown that the label trace component improves the POS prediction significantly. We further showed that higher POS tagging accuracy leads to a better language model. Based on these two statements, we draw the corollary that the label trace component should improve the performance of language modeling. In this section, we conducted an ablation study on both PTB and WT2 to check the corollary. We remove the label trace component from AWD-PRL-Q and the new model is denoted as AWD-PRL-Q (no T).

Dataset	Model	No Neural Cache Test PPL	No Neural Cache Δ	Neural Cache Test PPL	Neural Cache Δ
PTB	AWD	58.48 $\pm$ 0.06	/	54.36 $\pm$ 0.07	/
PTB	AWD-PRL-Q	55.90 $\pm$ 0.06	2.58	51.90 $\pm$ 0.06	2.46
PTB	AWD-PRL-Q (no T)	58.74 $\pm$ 0.04	-0.26	54.54 $\pm$ 0.03	-0.18
WT2	AWD	68.85 $\pm$ 0.11	/	54.24 $\pm$ 0.07	/
WT2	AWD-PRL-Q	66.75 $\pm$ 0.07	2.10	52.58 $\pm$ 0.05	1.66
WT2	AWD-PRL-Q (no T)	69.29 $\pm$ 0.07	-0.44	54.51 $\pm$ 0.07	-0.27

Table 4.4: The ablation study of the label trace for AWD-PRL-Q on PTB and WT2. AWD-PRL-Q (no T) is derived from AWD-PRL-Q by removing the label trace component. The Δ columns show the improvements of AWD-PRL-Q and AWD-PRL-Q (no T) in terms of perplexity, compared with AWD. We report results with and without neural cache (column Neural Cache and column No Neural Cache). All results were averaged over 5 runs with the standard errors reported. Without the label trace, the performance of AWD-PRL-Q is significantly worse.

We report the results in Table 4.4. The performance of AWD-PRL-Q (no T) is significantly worse than AWD-PRL-Q, even worse than AWD. This supports our corollary that the label trace component can improve the performance of language modeling.

As shown in Section 4.3, after we remove the label trace component from AWD-PRL-Q, the accuracy of POS tagging drops significantly. A POS decoder with poor performance generates bad predictive representations which contain too much noise information. The representations with low quality hurt the performance of LM itself. The label trace doesn't improve the performance of LM directly; it improves the quality of the predictive representations which finally lead to a better language model.

4.6 Conclusion

In this chapter, we performed our experiments to show that PRL improves the performance of LM when it is combined with language models. We also analyzed the convergence rate and sample efficiency of PRL. Then, we surprisingly found that the performance of GVFN-LSTM for POS tagging is actually worse than LSTM which pointed out a future research direction. Moreover, we studied the effect of POS tagging accuracy on PRL and concluded that higher POS tagging accuracy would lead to better performance of LM. Finally, we did an ablation study to show the importance of label traces.

Chapter 5

Related Work

In this chapter, we present the related work of our method. Overall, our work is closely related to three areas – predictive representations, multi-task learning, and learning from hints.

5.1 Predictive Representations

The predictive representations hypothesis, first proposed in the area of RL, posits that representations which predict future observations are better for generalization [43].

Littman and Sutton [25] developed the Predictive State Representation (PSR) which updates the state representation recursively. In particular, a PSR is a vector of predictions for several action-observation sequences. These sequences are also known as *tests*. Among all tests, *core tests* are those tests that are sufficient to make predictions about the future. For each core test, an agent records the test success probability and this probability then becomes a feature in the state representation of the agent. The core tests are usually specially selected with some methods [6, 22, 29, 44, 52]. Sun et al. [55] developed the Predictive State Inference Machine (PSIM), which directly learns predictors for inference in predictive state space. In the context of dynamic systems, a PSIM was shown to converge faster than simple auto-regressive models. Rafols et al. [43] showed that predictive representations help to improve sample efficiency, allowing agents to learn faster in a grid-world navigation task.

Value functions and general value functions can also be viewed as a special form of tests. Sutton and Tanner [57, 59] extended Temporal-Difference (TD) learning methods and proposed TD networks as an effective alternative learning algorithm for PSR. TD networks include two kinds of networks: question networks and answer networks. A question network suggests a question about possible future observations, analogous to a test in PSR. An answer network learns to answer the question; it is analogous to the function used to compute the success probability of the test. The predictions of TD networks are then incorporated into state representations. Sutton and Tanner showed that TD networks can learn state representations that make it possible to find an exact solution of a non-Markov problem.

Later, Sutton et al. [58] proposed the Horde architecture to learn general knowledge about the environment in the form of a large amount of GVFs. It is shown that GVFs have the capacity to capture and encode both short- and long-term information, which improves performance and generalization [48]. Based on this work, Schlegel et al. [49] proposed a new RNN architecture, the General Value Function Network (GVFN), which combines GVFs in an RNN. In a GVFN, a state consists of predictions about the future. Schlegel et al. went on to show the robustness of GVFNs for several time series prediction tasks and RL prediction problems.

Predictive representations have also proved their value in the area of NLP. Kuncoro et al. [23] proposed structure-distilled BERT models by injecting explicit syntactic inductive biases into BERT models. The syntactic biases are expressed as syntactically informative predictions, in the form of probability distributions. The structure-distilled BERT models were found to outperform the baseline method on six diverse structured prediction tasks. These results demonstrated that predictive representations of syntactic biases are beneficial for large models that exploit large amounts of data.

5.2 Multi-task Learning

Many multi-task learning frameworks have been proposed for NLP tasks. During training, parameters for each task can be shared among tasks, improving the optimization of those shared parameters. Bingel and Søgaard [4] found that multi-task learning can help neural networks training get out of local minima. Multi-task learning also improves model generalization, accelerates the training process, and allows for knowledge sharing across domains [7, 45].

Specifically, Luong et al. [27] developed a multi-task sequence to sequence model and found that training this model on syntactic parsing and image caption data improves translation quality between English and German. Søgaard and Goldberg [53] presented a multi-task learning RNN architecture for sequence tagging. When comparing low-level task supervision (e.g. POS tagging) at both the innermost and outmost layers of the RNN, Søgaard and Goldberg found that supervision at the innermost layer gives the greatest increase in performance. Furthermore, Hashimoto et al. [15] proposed a joint many-task model that reflects linguistic hierarchies, and achieves state-of-the-art or competitive results on five different NLP tasks. Similarly, Sanh et al. [47] introduced a hierarchical model trained on a set of selected semantic tasks by supervising low-level tasks at lower layers and more complex tasks at higher layers. They also found that the learned sentence representations encode more diverse semantic information. Subramanian et al. [54] applied large scale multi-task learning to learn a general purpose distributed sentence representations. They demonstrated that these learned representations are competitive to or even better than general-purpose sentence representations produced by previous methods. Such representations also greatly speed up low-resource learning. More recently, Ruder et al. [46] proposed the Sluice Network, a multi-task learning framework, where the *amount* of sharing is learnable. In particular, sharing of layers, subspaces, and skip connections are all learnable; the amount of sharing is updated automatically during training. Sluice Networks showed gains on named entity recognition and semantic role labeling tasks. In practice, they were also robust across domains.

Unlike these approaches, our work focuses on encouraging the predictive nature of the underlying representations and incorporating those predictions into states. Additionally, the multi-task model structure we propose is different from them in terms of using information from auxiliary tasks.

5.3 Learning from Hints

Our work is also closely related to learning from hints [1]. Hints are pieces of information about the function we want to learn, such as features of the function. Useful hints can be used to reduce the hypothesis function space. The predicted features can also be incorporated as auxiliary tasks to benefit the learning of the main task, as supported by our work. Specifically, hints are equivalent to the output of GVFNs in our work.

Yu and Jiang [64] built a neural network architecture for cross-domain sentiment classification, and included two auxiliary tasks to predict whether the input text contained domain-*independent* positive or negative sentiment words. They found their model could identify more domain-*specific* sentiment words, compared to several highly competitive baseline methods. In a similar vein, Cheng et al. [8] proposed a multi-task RNN language model for sentence-level name detection. In this model, the auxiliary task is to predict whether a sentence has a name in it as an auxiliary task. Compared to a system using n-gram lexical features, their sentence-level model improves the name-error detection F-score by 26%.

Chapter 6

Conclusion and Future Work

Representation learning is an important topic in machine learning. Good representations are crucial for models that must represent states or otherwise compress input information. However, the criteria of what makes a good representation are not always clear.

In this work, we explored a concept known as predictive representations hypothesis which has shown great success in the area of RL, in the scenario of language modeling. To be specific, we introduced PRL as an addition to RNN models that learns predictive representations with the help of a GVF. It supports auxiliary task learning and can improve the learned solutions for sequence labeling tasks. We incorporated PRL into two strong language models and tested them on two standard benchmarks. Our main results demonstrated that PRL outperforms the baselines significantly in terms of perplexities. Furthermore, we analyzed the convergence rate and sample efficiency of different methods. The results showed that PRL converged much faster than the baselines, and it improved the performance of LM even when training datasets are small. As a by-product, we also reported the performance of the auxiliary task (i.e. POS tagging) in our experiments. We discovered that AWD-PRL-Q is worse than AWD-PRL-P in terms of the POS tagging accuracy, although AWD-PRL-Q is better than AWD-PRL-P for LM. Note that the action-values from AWD-PRL-Q encode not only the prediction of the immediate next POS label, but all future labels following the current word. This fact might explain why AWD-PRL-Q is better at predicting the next word even though its POS tagging accuracy is lower. However, future work is still needed to figure this out. Then we studied the effect of POS tagging accuracy on PRL. The results showed that the performance of LM can be further improved if the POS tagging accuracy was higher, which points out a future improvement of PRL. Finally, we did an ablation experiment to verify the importance of label trace.

Our results point to several options for future work, too. We would like to test our method on more datasets, more sequence labeling tasks, and measure the influence of the number of auxiliary tasks in the auxiliary task learning setting. Solving sequence labeling tasks with other RL algorithms (e.g. SARSA and Actor-Critic in [56]) is also an interesting direction. The transformer [9], which has shown great improvement for LM, is another ripe area for GVFs.

We also note that our work requires additional labeled data to learn predictive representations. In the future, it may be possible for algorithms to define their own supervision tasks (self-supervision), and incorporate those predictions into the learned representations. As mentioned previously, this occurs naturally in RNNs for LM, but encouraging it more explicitly could leverage the advances we presented here without requiring new labeled datasets. However, the NLP community's strong tradition of creating benchmarks (and associated labeled data) means there will remain plenty to gain by using supervised learning in the PRL framework.

References

[1] Yaser S Abu-Mostafa. "Learning from hints in neural networks." In: *Journal of Complexity* 6 (1990), pp. 192–198. DOI: 10.1016/0885-064X(90)90006-Y. 35

[2] Dario Amodei et al. "Deep speech 2: End-to-end speech recognition in english and mandarin." In: *International Conference on Machine Learning*. 2016, pp. 173–182. 7

[3] Yoshua Bengio et al. "A neural probabilistic language model." In: *Journal of Machine Learning Research* 3.Feb (2003), pp. 1137–1155. 6, 9

[4] Joachim Bingel and Anders Søgaard. "Identifying beneficial task relations for multi-task learning in deep neural networks." In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. 2017, pp. 164–169. 34

[5] Bernd Bohnet et al. "Morphosyntactic tagging with a meta-BiLSTM model over context sensitive token encodings." In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. Vol. 1. 2018, pp. 2642–2652. DOI: 10.18653/v1/P18-1246. URL: https://www.aclweb.org/anthology/P18-1246. 27, 28

[6] Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. "Closing the learning-planning loop with predictive state representations." In: *The International Journal of Robotics Research* 30.7 (2011), pp. 954–966. 32

[7] Rich Caruana. "Multitask learning." In: *Machine Learning* 28.1 (1997), pp. 41–75. 34

[8] Hao Cheng, Hao Fang, and Mari Ostendorf. "Open-domain name error detection using a multi-task RNN." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 737–746. DOI: 10 . 18653 / v1 / D15 - 1085. URL: https://www.aclweb.org/anthology/D15-1085. 35

[9] Zihang Dai et al. "Transformer-XL: attentive language models beyond a fixed-length context." In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 2978–2988. DOI: 10.18653/v1/P19-1285. URL: https://www.aclweb.org/anthology/P19-1285. 37

[10] Salah El Hihi and Yoshua Bengio. "Hierarchical recurrent neural networks for long-term dependencies." In: *Advances in Neural Information Processing Systems*. 1996, pp. 493–499. 8

[11] Jeffrey L Elman. "Finding structure in time." In: *Cognitive Science* 14.2 (1990), pp. 179–211. 7

[12] Edouard Grave, Armand Joulin, and Nicolas Usunier. "Improving neural language models with a continuous cache." In: *International Conference on Learning Representations*. 2016. URL: `https://openreview.net/forum?id=B184E5qee`. 22

[13] Alex Graves and Navdeep Jaitly. "Towards end-to-end speech recognition with recurrent neural networks." In: *International Conference on Machine Learning*. 2014, pp. 1764–1772. 7

[14] Alex Graves and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." In: *Neural Networks* 18.5-6 (2005), pp. 602–610. 9

[15] Kazuma Hashimoto et al. "A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks." In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 1923–1933. 34

[16] Peter A. Heeman. "POS tags and decision trees for language modeling." In: *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. 1999. URL: `https://www.aclweb.org/anthology/W99-0617`. 19

[17] Geoffrey E Hinton. "Distributed representations." In: *Parallel Distributed Processing* (1986), pp. 77–109. 4

[18] Geoffrey E Hinton et al. "Learning distributed representations of concepts." In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Vol. 1. Amherst, MA. 1986, p. 12. 5

[19] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780. 8

[20] Frankie James. "Modified Kneser-Ney smoothing of n-gram models." In: *Research Institute for Advanced Computer Science, Tech. Rep. 00.07* (2000). 10

[21] Ben Krause et al. "Dynamic evaluation of neural sequence models." In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. 2018, pp. 2766–2775. URL: `http://proceedings.mlr.press/v80/krause18a.html`. 24

[22] Alex Kulesza, Nan Jiang, and Satinder Singh. "Spectral learning of predictive state representations with insufficient statistics." In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015. 32

[23] Adhiguna Kuncoro et al. “Syntactic Structure Distillation Pretraining For Bidirectional Encoders.” In: *arXiv preprint arXiv:2005.13482* (2020). URL: https://arxiv.org/abs/2005.13482. 33

[24] Yair Lakretz et al. “The emergence of number and syntax units in LSTM language models.” In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Vol. 1. 2019, pp. 11–20. URL: https://www.aclweb.org/anthology/N19-1002. 1

[25] Michael L Littman and Richard S Sutton. “Predictive representations of state.” In: *Advances in Neural Information Processing Systems*. MIT Press, 2002, pp. 1555–1561. URL: https://papers.nips.cc/paper/1983-predictive-representations-of-state. 1, 32

[26] Shujie Liu et al. “A Recursive Recurrent Neural Network for Statistical Machine Translation.” In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2014, pp. 1491–1500. 7

[27] Minh-Thang Luong et al. “Multi-task sequence to sequence learning.” In: *International Conference on Learning Representations*. 2016. 34

[28] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. “Building a large annotated corpus of English: the Penn Treebank.” In: *Computational Linguistics* 19.2 (1993), pp. 313–330. URL: https://www.aclweb.org/anthology/J93-2004. 20

[29] Peter McCracken and Michael Bowling. “Online discovery and learning of predictive state representations.” In: *Advances in Neural Information Processing Systems*. 2006, pp. 875–882. 32

[30] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. “Regularizing and optimizing LSTM language models.” In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=SyyGPP0TZ. 10, 21, 22

[31] Stephen Merity et al. “Pointer sentinel mixture models.” In: *International Conference on Learning Representations*. 2016. URL: https://openreview.net/forum?id=Byj72udxe. 20

[32] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality.” In: *Advances in Neural Information Processing Systems*. 2013, pp. 3111–3119. 5

[33] Tomas Mikolov et al. “Efficient estimation of word representations in vector space.” In: *arXiv preprint arXiv:1301.3781* (2013). 6

[34] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." In: *Proceedings of the 2013 Conference of the North American Chapter of the Association For Computational Linguistics: Human Language Technologies*. 2013, pp. 746–751. 5

[35] Tomáš Mikolov et al. "Recurrent neural network based language model." In: *Eleventh Annual Conference of the International Speech Communication Association*. 2010. 7, 9

[36] Tomáš Mikolov et al. "Empirical evaluation and combination of advanced language modeling techniques." In: *Twelfth Annual Conference of the International Speech Communication Association*. 2011. URL: https://www.isca-speech.org/archive/interspeech_2011/i11_0605.html. 20

[37] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), pp. 529–533. 7

[38] Joseph Modayil, Adam White, and Richard S Sutton. "Multi-timescale nexting in a reinforcement learning robot." In: *International Conference on Simulation of Adaptive Behavior*. Springer. 2012, pp. 299–309. DOI: 10.1177/1059712313511648. 1

[39] Malvina Nissim, Rik van Noord, and Rob van der Goot. "Fair is better than sensational: Man is to doctor as woman is to doctor." In: *Computational Linguistics* 46.2 (2020), pp. 487–497. URL: https://doi.org/10.1162/coli_a_00379. 5

[40] Razvan Pascanu et al. "How to construct deep recurrent neural networks." In: *arXiv preprint arXiv:1312.6026* (2013). 8

[41] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: http://www.aclweb.org/anthology/D14-1162. 7

[42] Barbara Plank, Anders Søgaard, and Yoav Goldberg. "Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2016, pp. 412–418. 7

[43] Eddie J Rafols et al. "Using predictive representations to improve generalization in reinforcement learning." In: *International Joint Conference on Artificial Intelligence*. 2005, pp. 835–840. URL: https://www.ijcai.org/Proceedings/05/Papers/1650.pdf. 1, 32

[44] Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. "Learning low dimensional predictive representations." In: *Proceedings of the Twenty-first International Conference on Machine Learning*. 2004, p. 88. 32

[45] Sebastian Ruder. "An overview of multi-task learning in deep neural networks." In: *arXiv preprint arXiv:1706.05098* (2017). 34

[46] Sebastian Ruder et al. "Sluice networks: Learning what to share between loosely related tasks." In: *ArXiv* abs/1705.08142 (2017). 34

[47] Victor Sanh, Thomas Wolf, and Sebastian Ruder. "A hierarchical multi-task approach for learning embeddings from semantic tasks." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 6949–6956. 34

[48] Tom Schaul and Mark Ring. "Better generalization with forecasts." In: *International Joint Conference on Artificial Intelligence*. 2013. URL: `https://www.ijcai.org/Proceedings/13/Papers/246.pdf`. 1, 33

[49] Matthew Schlegel et al. "General value function networks." In: *arXiv preprint arXiv:1807.06763* (2018). URL: `http://arxiv.org/abs/1807.06763`. 1, 13, 14, 33

[50] Jürgen Schmidhuber. "Learning complex, extended sequences using the principle of history compression." In: *Neural Computation* 4.2 (1992), pp. 234–242. 8

[51] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks." In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681. 9

[52] Satinder Singh, Michael R James, and Matthew R Rudary. "Predictive state representations: a new theory for modeling dynamical systems." In: *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. 2004, pp. 512–519. 32

[53] Anders Søgaard and Yoav Goldberg. "Deep multi-task learning with low level tasks supervised at lower layers." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2016, pp. 231–235. 34

[54] Sandeep Subramanian et al. "Learning General Purpose Distributed Sentence Representations via Large Scale Multi-task Learning." In: *International Conference on Learning Representations*. 2018. 34

[55] Wen Sun et al. "Learning to filter with predictive state inference machines." In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016, pp. 1197–1205. URL: `http://proceedings.mlr.press/v48/sun16.html`. 32

[56] Richard S Sutton and Andrew G Barto. *Reinforcement learning: an introduction*. Second. MIT Press, 2018. 13, 37

[57] Richard S Sutton and Brian Tanner. "Temporal-Difference networks." In: *Advances in Neural Information Processing Systems*. 2005, pp. 1377–1384. 33

[58] Richard S Sutton et al. "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction." In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 2011, pp. 761–768. 33

[59] Brian Tanner and Richard S Sutton. "Temporal-Difference Networks with History." In: *International Joint Conference on Artificial Intelligence*. 2005, pp. 865–870. 33

[60] Athanasios Voulodimos et al. "Deep learning for computer vision: A brief review." In: *Computational Intelligence and Neuroscience* 2018 (2018). 7

[61] Chris Watkins. "Learning from delayed rewards." PhD thesis. 1989. URL: `http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf`. 13, 19

[62] Adam White. "Developing a predictive approach to knowledge." PhD thesis. 2015. DOI: `10.7939/R3FF3M75H`. 1

[63] Zhilin Yang et al. "Breaking the softmax bottleneck: a high-rank RNN language model." In: *International Conference on Learning Representations*. 2018. 10, 22, 24

[64] Jianfei Yu and Jing Jiang. "Learning sentence embeddings with auxiliary tasks for cross-domain sentiment classification." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 236–246. DOI: `10.18653/v1/D16-1023`. URL: `https://www.aclweb.org/anthology/D16-1023`. 35

Appendix A

Runtime and Model Size

All models were trained on an NVIDIA Tesla V100-32GB GPU. We report the training speed and the number of parameters of each model in Table A.1. All models were trained on an NVIDIA Tesla V100-32GB GPU.

Dataset	Model	Speed	# Param
PTB	AWD	38 s/epoch	35.44 M
PTB	AWD-PRL-P/Q	55 s/epoch	39.20 M
WT2	AWD	60 s/epoch	44.76 M
WT2	AWD-PRL-P/Q	76 s/epoch	48.59 M
PTB	MoS	81 s/epoch	30.41 M
PTB	MoS-PRL-Q	110 s/epoch	33.95 M
WT2	MoS	412 s/epoch	47.17 M
WT2	MoS-PRL-Q	476 s/epoch	50.84 M

Table A.1: The training speed and the number of parameters of each model.

Appendix B

Experiment with Named Entity Recognition

To test our method further, we introduced named entity recognition (NER) as another auxiliary task. We tested model AWD-PRL-Q on Penn Treebank using NER. The NER labels were generated using spaCy. Results were presented in Table B.1. It is shown that NER also helps improve the performance of LM, although the improvement is not as much as POS tagging.

Auxiliary Task	Model	No Neural Cache		Neural Cache	
		Test PPL	Δ	Test PPL	Δ
/	AWD	58.48 ± 0.06	/	54.36 ± 0.07	/
POS	AWD-PRL-Q	55.90 ± 0.06	2.58	51.90 ± 0.06	2.46
NER	AWD-PRL-Q	57.06 ± 0.05	1.42	52.97 ± 0.05	1.39

Table B.1: The comparison of test perplexities on PTB for LM based on AWD, with and without an auxiliary task. The Δ columns show the improvements of AWD-PRL-Q in terms of perplexity, compared with AWD. We report results with and without neural cache. All results were averaged over 5 runs with the standard errors reported. For both POS tagging and NER, PRL improves the performance of AWD, with and without neural cache. In terms of improvement for LM, POS tagging is a better auxiliary task compared with NER.