

# Natural Language Processing

Notes taken by Runqiu Ye  
Carnegie Mellon University

Spring 2025

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Language Modeling Fundamentals</b>	<b>4</b>
2.1	Neural Networks and Auto Differentiation . . . . .	4
2.2	Language and Sequence Modeling . . . . .	4
2.2.1	Bigram models and N-gram models . . . . .	4
2.2.2	Feedforward neural language model . . . . .	5
2.2.3	Practice tips for neural language models . . . . .	5

# 1 Introduction

Below is a list of topics that will be covered in this note. The topics include fundamental theoretical knowledge in language processing and language modeling in the first few sections, as well as frontier research results in later sections.

1. Language modeling fundamentals
  - (a) Representing words
  - (b) Language modeling
  - (c) Sequence modeling architectures
2. Training and inference models
  - (a) Decoding and Generation Algorithms
  - (b) In-context learning
  - (c) Pre-training
  - (d) Fine-tuning
  - (e) Reinforcement Learning
3. Evaluation and Experimental Design
  - (a) Evaluating Language Generators
  - (b) Experimental Design
  - (c) Human Annotation
  - (d) Debugging/Interpretation Techniques
4. Advanced Algorithms and Architectures
  - (a) Advanced Pretraining, Post-Training, and Inference
  - (b) Retrieval and Retrieval-augmented Generation
  - (c) Long Sequence Models
  - (d) Distillation and Quantization
  - (e) Ensembling and Mixture of Experts
5. NLP Applications and Society
  - (a) Complex Reasoning Tasks
  - (b) Language Agents
  - (c) Multimodal NLP
  - (d) Multilingual NLP
  - (e) Bias and Fairness

## 2 Language Modeling Fundamentals

### 2.1 Neural Networks and Auto Differentiation

This process calculates the gradient of the loss with respect to different parameters. Use computational graph and chain rule to compute back propagation efficiently, and implemented in PyTorch. With the calculated gradient, we can use Stochastic Gradient Descent (SGD) to update the model parameters. Other than PyTorch, TensorFlow and JAX are also popular neural network frameworks with different advantages.

### 2.2 Language and Sequence Modeling

In language modeling, other than binary or multi-class classification, sometimes there are exponential/ininitely many labels. This is called **structured prediction**. For example, assigning a part-of-speech tag for each word in the sentence. It is easy to note that in this case the number of labels grows exponentially with respect to the length of the sequence. There are also distinctions between **unconditioned prediction** and **conditioned prediction**, which predict the distribution of  $P(X)$  and  $P(Y | X)$ , respectively.

A language model is a probability distribution over all sequences. It can be used to **score** sequences and **generate** sequences. For example, in conditional generation, we condition on an input text and tries to continue it

$$\hat{x}_{t+1:T} \sim p(X_{t+1:T} | x_{1:t}).$$

For another example, it can be used to translate between languages (machine translation). In all of these scenarios, the key is to model the following distribution:

$$p(x_{t+1} | x_1, x_2, \dots, x_n).$$

#### 2.2.1 Bigram models and N-gram models

The assumption of bigram models is

$$p(X) \approx \prod_{t=1}^T p_{\theta}(x_t | x_{t-1}).$$

This is to say all tokens only depend on the token right before it. To train the bigram models, we simply count

$$p(x_t | x_{t-1}) = \frac{\text{count}(x_{t-1}, x_t)}{\sum_{x'} \text{count}(x_{t-1}, x')}.$$

We can view  $\theta_{i,j} = p(x_j | x_i)$  the parameters of the model. The reason behind the counting procedure is to produce a **maximum likelihood estimation**:

$$\max_{\theta} \sum_{x \in D_{\text{train}}} \log p_{\theta}(x).$$

This essentially tries to match  $p_{\theta}$  to  $p_{\text{data}}$ , which can be more formally derived with KL divergence.

To evaluate the model, we can use the **log-likelihood** and the **perplexity** of the model. The log-likelihood is simply defined as

$$\text{LL}(\mathcal{X}_{\text{test}}) = \sum_{X \in \mathcal{X}_{\text{test}}} \log p(X).$$

We can also use the per-word log-likelihood  $\text{WLL}(\mathcal{X}_{\text{test}})$ , which is defined as the log-likelihood divided by the length of the test data. The perplexity of the model is  $\exp(-\text{WLL}(\mathcal{X}_{\text{test}}))$ , which can be used to simulate the number of tries it need to take before getting the correct prediction.

The bigram models or N-gram models in general have a lot of problems. When  $N$  gets large, the number of parameters will grow exponential with respect to  $N$ , but the occurrence of these parameters will be extremely rare. Additionally, it cannot share strength among similar words, it cannot condition on context with intervening words, and it cannot handle long-distance dependencies. To try it out, use the `kenlm`<sup>1</sup> toolkit.

<sup>1</sup> <https://github.com/kpu/kenlm>

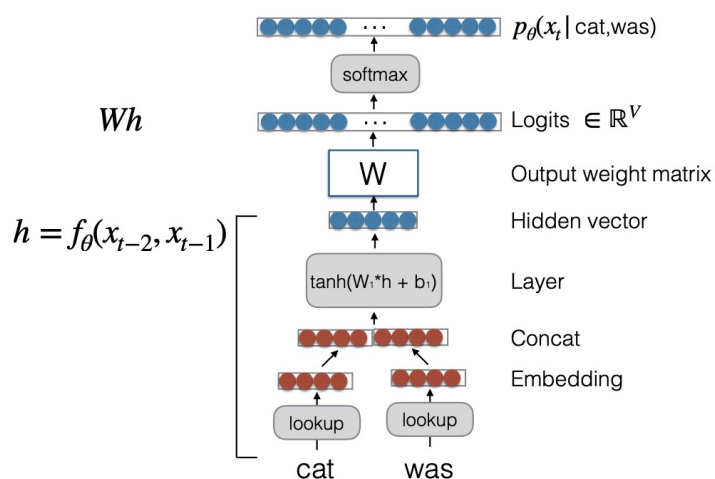


Figure 1: An illustration of feedforward neural language model.

### 2.2.2 Feedforward neural language model

Use a neural network to model

$$p(X) = \prod_{t=1}^T p(x_t | x_{t-1}, \dots, x_{t-n+1})$$

by minimizing the loss

$$L = -\log p_\theta(x_t | x_{1:t-1}).$$

This actually correspond to cross entropy loss. An example structure of this is shown in Figure 1.

Note that since we can make the embeddings of word closed to each other if the words have similar meaning, it solves the problem of N-gram models that they cannot share strength between similar words. Neural models can also condition on context with intervening words, which is another advantage compare to N-gram models.

### 2.2.3 Practice tips for neural language models

It is good to initialize weight matrices using the **Xavier initialization** technique, which is implmented in function `nn.init.xavier_uniform_` in PyTorch. Weight initialization is important in deep learning models, since it has a huge impact on the training results. For optimizer, instead of SGD, the most standard optimization method is **Adam**, which is also implmented in PyTorch. For the choice of learning rate, we also have consine learning rate schedule and warmup techniques.

Check out paper to learn more about these methods.