Natural Language Processing

Notes taken by Runqiu Ye
Carnegie Mellon University

Spring 2025

Contents

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 propogation efficiently, and implmented 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/infinitely many labels. This is called **structured prediction**. For example, assgining 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 **uncondictioned prediction** and **conditioned prediction**, which predict the distribution of $P(X)$ and $P(Y \mid X)$, 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

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

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:

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

Bigram models and N-gram models

The assumption of bigram models is

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

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

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

We can view $\theta_{i,j} = p(x_j \mid x_i)$ 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_θ to p_{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

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

We can also use the per-word log-likelihood, which is defined as the log-likelihood divided by the length of the test data. The perplexity of the model is $\exp(-\text{WLL})$, 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`[1] toolkit.

[1] `https://github.com/kpu/kenlm`

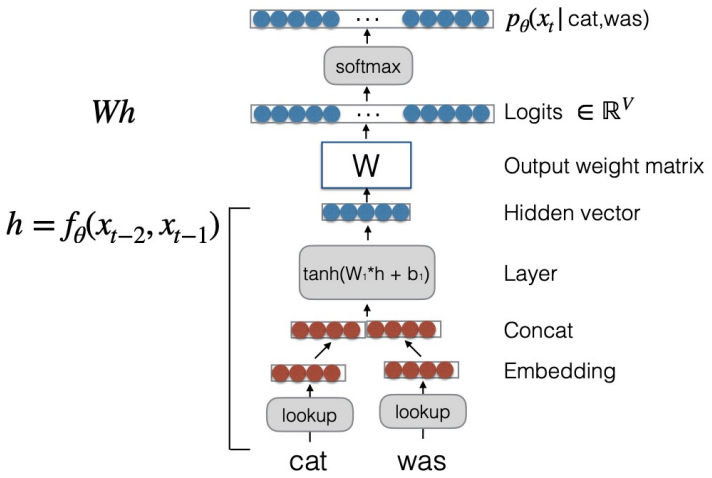

Figure 1: An illustration of feedforward neural language model.

Feedforward neural language model

Use a neural netowrk to model

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

by miniming the loss

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

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.