Statistical Language Models

Maneesh Kumar Singh
mksingh4@illinois.edu

November 14, 2020

Abstract

In this technical review of Statistical Language Models, we first provide brief introduction of language models and its two very popular types: n-grams and neural language models. We compare the pros and cons of these models and summarize our findings.

1 Introduction

Statistical language model is a probabilistic model for text data, which defines distributions over sequences of words. Such model is often also called a generative model for text data because it can be used for sampling sequences of words.

More formally, if we have some text $x^{(1)}, ..., x^{(T)}$, then the probability of this text according to the Language Model is

$$P(x^{(1)}, ..., x^{(T)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times ... \times P(x^{(T)}|x^{(T-1)}, .., x^{(1)}) \tag{1}$$

$$= \prod_{t=1}^{T} P(x^{(t)}|x^{(t-1)}, .., x^{(1)}) \tag{2}$$

2 n-gram Language Models

An n-gram is a chunk of n consecutive words. Given text “The water is so transparent that”, its n-grams are

- unigrams: “The”, “water”, “is”, “so”, “transparent”, ”that”
- bigrams: “The water”, “water is”, “is so”, “so transparent”, “transparent that”
- trigrams:“The water is”, “water is so”, “is so transparent”, “so transparent that”
- 4-grams: “The water is so”, “water is so transparent”, “is so transparent that”

and so on.

The idea is to collect statistics about how frequent different n-grams are, and use these to predict next words. In order to do so, we make a Markov assumption: $x^{(t+1)}$ depends only on preceding n-1 words.

$$P(x^{(t+1)}|x^{(t)}, ..., x^{(1)}) = P(x^{(t+1)}|x^{(t)}, ..., x^{(t-n+2)}) \tag{3}$$

$$= \frac{P(x^{(t+1)}, x^{(t)}, ..., x^{(t-n+2)})}{P(x^{(t)}, ..., x^{(t-n+2)})} \tag{4}$$

To compute the probabilities mentioned above, the count of each n-gram can be compared against the frequency of each word. This is called an n-gram Language Model. For example, if the model takes bi-grams, the frequency of each bi-gram, calculated via combining a word with its previous word, would be divided by the frequency of the corresponding uni-gram. Equations 5 and 6 shows this relationship for bigram and trigram models.

$$p(w_2|w_1) = \frac{count(w_1, w_2)}{count(w_1)} \tag{5}$$

$$p(w_3|w_1, w_2) = \frac{count(w_1, w_2, w_3)}{count(w_1, w_2)} \tag{6}$$

2.1 Pros

- Really easy to build
- Can train on billions and billions of words

2.2 Cons

n-gram language model suffers from sparsity problem. Sparsity is used to describe the situation of not observing enough data in corpus to model language accurately.

- Note the numerator of Equation 6. If w_1, w_2, and w_3 never appear in corpus together in the corpus, the probability of w3 is 0. To solve this, a small δ can be added to the count of each word in the vocabulary. This is called *smoothing*.
- Consider the denominator of Equation 6. If w_1 and w_2 never occurred together in the corpus, then no probability can be calculated for w_3. To solve this, we could condition on w_2 alone. This is called *backoff*.
- Increasing n makes sparsity problems worse. Typically, $n \leq 5$.

n-gram language model also suffers from storage problem.

- As n increases (or the corpus size increases), the model size increases as well.

3 Window-based Neural Language Model

Bengio et al introduced a Neural Probabilistic Language Model. It was first large-scale deep learning model for natural language processing. This model learns a *distributed representation of words*, along with the probability function for word sequences expressed in terms of these representations. Figure 1 shows the corresponding neural network architecture. The input word vectors are used by both the hidden layer and output layer.

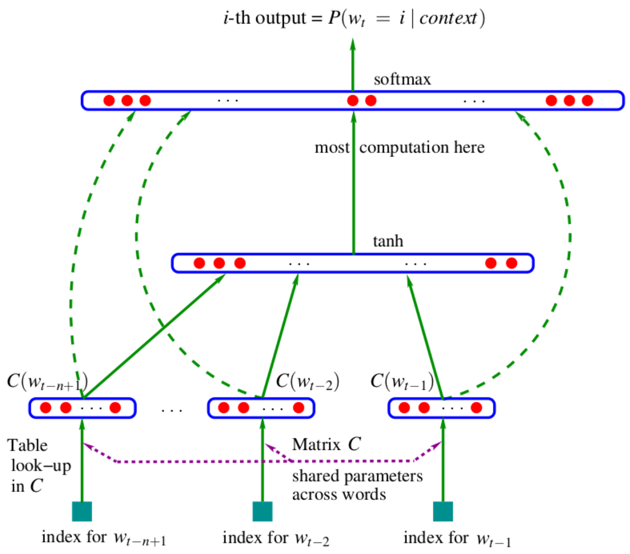

Figure 1: The first deep neural network architecture model for NLP presented by Bengio et al.

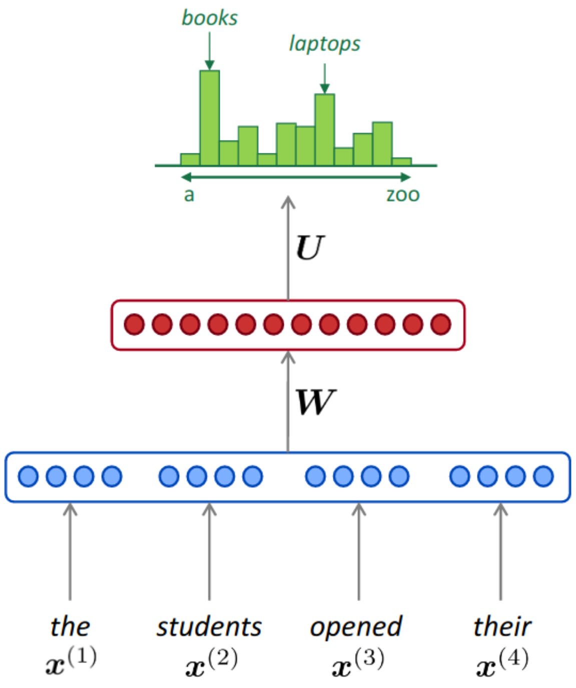
books
laptops
a
zoo
$\boldsymbol{U}$
$\boldsymbol{W}$
the $\boldsymbol{x}^{(1)}$
students $\boldsymbol{x}^{(2)}$
opened $\boldsymbol{x}^{(3)}$
their $\boldsymbol{x}^{(4)}$

Figure 2: A simplified representation of Figure 1

Equation 7 represents Figure 1 and shows the parameters of the $softmax()$ function, consisting of standard $tanh()$ function (i.e. the hidden layer) as well as the linear function, $W^{(3)}x + b^{(3)}$, that captures all the previous n input word vectors.

A simplified version of this model can be seen in Figure 2, where the blue layer represents concatenated word embeddings for the input words: $e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$, the red layer signifies the hidden layer: $h = f(W_e + b_1)$, and the green output distribution is a softmax over the vocabulary: $\hat{y} = softmax(Uh + b_2)$.

$$\hat{y} = softmax(W^{(2)} tanh(W^{(1)}x + b^{(1)}) + W^{(3)}x + b^{(3)}) \tag{7}$$

$$\hat{y} = softmax(Uh + b_2) \in \mathbb{R}^{|V|} \tag{8}$$

3.1 Pros

- No sparsity problem
- Don't need to store all observed n-grams

3.2 Cons

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

4 Recurrent Neural Networks (RNN)

Unlike the conventional translation models, where only a finite window of previous words would be considered for conditioning the language model, Recurrent Neural Networks (RNN) are capable of conditioning the model on all previous words in the corpus.

Figure 3 introduces RNN architecture where each vertical rectangular box is a hidden layer at a time-step t. Each such layer holds a number of neurons, each of which performs a linear matrix operation on its inputs followed by a non-linear operation (e.g. $tanh()$). At each time-step, there are two inputs to the hidden layer: the output of previous layer h_{t-1}, and the input at that time-step x_t. The former input is multiplied by a weight matrix $W^{(hx)}$ to produce output features h_t, which are multiplied with a weight matrix $W^{(S)}$ and run through a softmax over the vocabulary to obtain a prediction output $\hat{y}$ of the next word(Equation 9 and 10).

$$h_t = \alpha(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}) \tag{9}$$

$$\hat{y}_t = softmax(W^{(S)}h_t) \tag{10}$$

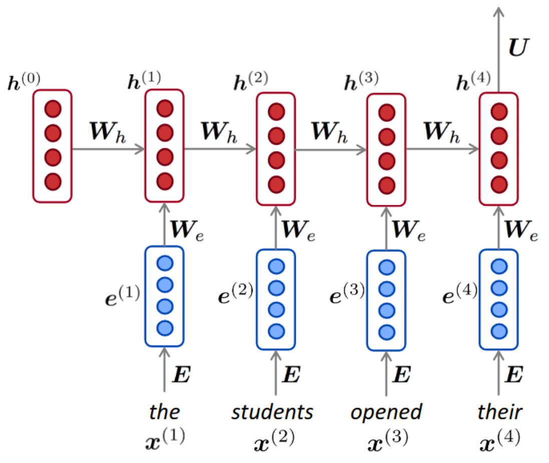

Figure 3: An RNN Language Model

4.1 Pros

- They can process input sequence of any length
- The model size does not increase for longer input sequence lengths
- Computation for step t can (in theory) use information from many steps back
- The same weights are applied to every time-step of the input, so there is symmetry in how inputs are processed

4.2 Cons

- Computation is slow - because it is sequential, it cannot be parallelized
- In practice, it is difficult to access information from many steps back due to problems like vanishing and exploding gradients

5 Summary

In this review we compared n-grams, neural and its improvement RNN language model. We noticed even though n-gram suffers from storage and sparsity problem. It's performance is better than neural language models with comparable accuracy. Neural language model provides better accuracy, but its accuracy is again restricted by window size. RNN overcomes the issue with neural language model, but is slow and cannot access information from many steps. Recently few models are developed over RNN which deals with performance and information access problems e.g. Long short-term memory(LSTM) and Gated recurrent units (GRUs).

References

[1] ChengXiang Zhai and Sean Massung. *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining.*

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin(2003). *A Neural Probabilistic Language Model*. Journal of Machine Learning Research, 3, 1137–1155.

[3] Stanford CS224n Lecture-6 Language Models and RNNS. Lecture Slides. Lecture Notes. Youtube Video.