A Comparison of N-gram Model and RNNLM

Snehal Somalraju

Introduction

In natural language processing, one of the open challenges is to create an accurate language model, which is a probability distribution of word sequences. A language model can be used to predict the next word based on a sequence. For instance, take the phrase: "*Please turn your homework…*" An accurate language model would predict that the next word would be "in" or "over," but not "dog" or "eat."

One real-world application of language models is speech recognition, which involves tasks where words need to be identified from noisy, ambiguous input. Suppose a speech recognition system needs to determine whether an audio signal represents "*I will be back soonish*" and "*I will be bassoon dish."* It would be helpful to know that the former phrase is much more likely than the latter phrase.

Several techniques have been devised to create a language model. We will compare two widely used approaches: n-gram models and RNNLM.

N-gram Model

The n-gram model is a conventional technique used to create language models. N-gram models are built by counting how often word sequences occur in a corpus text and then estimating the probabilities.

Suppose there are two sentences: "*There was heavy rain*" and "*There was heavy flood.*" Suppose we used two word sequences to calculate the probabilities, which would be called a bigram. "*Heavy rain*" would be more likely to occur than "*heavy flood*," so "*There was heavy rain*" would more likely be selected.

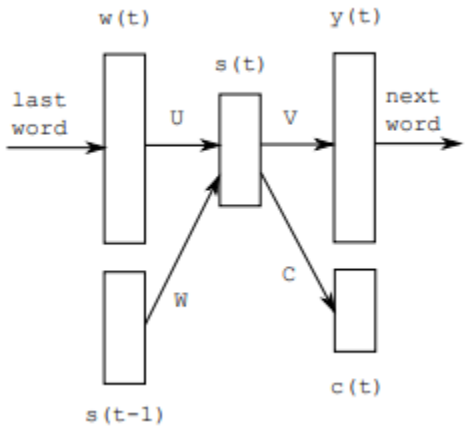
However, n-grams come with their disadvantages. N-grams model may have a sparsity issues, as certain word sequences may not appear in a corpus of text, and smoothing is required to assign probability to it. For instance, suppose we were to calculate P (w | students opened their). If "*students opened their*" never appeared in the corpus of text, then P (w | students opened their) would be 0. This would reduce the accuracy of n-gram models.

Recurrent Neural Networks

An alternate approach is to use recurrent neural networks to create a language model. These language models are called **recurrent neural net language models** (RNLLMs). Recurrent neural networks are neural networks that allow previous outputs to be used as inputs. This property can be referred to as memory.

We can see how this would be applicable to creating language models. Since the previous sequence of words is stored as memory in the RNN, this makes it convenient to calculate the probability of word given the previous sequence of words.

To explain how recurrent neural networks based language models work, we can examine a toolkit built by Microsoft. Here is the following diagram:



The input layer uses the previous word (w(t)) mapped from a one-hot vector and the previous state of the hidden layer (s(t-1)). A one hot vector is a vector that contains binary representation of all the words in the corpus; the word being considered is labeled 1, while all other words are labeled 0. The neurons in the hidden layer uses a sigmoid activation functions (s(t)) and outputs the probability distribution of the next word given the previous word (y(t)). The class layers can be optionally used in the toolkit to reduce the computational complexity of the model, though accuracy may be slightly compromised.

Studies have shown that language models built with recurrent neural networks have higher performance than those built with n-grams. Another advantage of using an RNN is that it is possible to effectively sequences and patterns with arbitrary length, as all these words would simply be stored in the memory of the RNN.

However, a disadvantage of using RNNs is that they can be computationally expensive, especially if the total vocabulary of the corpus text is large. For instance, if we use the Web corpora, the vocabulary will contain tens of millions of unique words. The matrices used in RNN, used to represent the input and hidden layers, would contain tens of billions of elements, as the size of the one-hot vector in the input layer would be the size of the vocabulary and the matrix used in the hidden layer contains probability distributions of all words in the vocabulary. This would make the RNN model too big to fit into the memory of GPU devices.

Conclusion

Due to the increased performance of using RNNLM over n-gram models, they may soon become the “conventional technique”. However, RNNLMs do have their challenges that need to

be overcome, such as the fact they can be computationally expensive. Fortunately, several improvements have been proposed, such as CUED-RNLLM, which contains GPU-based training for RNNs. Thus, we will reach closer to our dreams of creating machines that can emulate sentences from human languages.

References

Arvindpdmn. (2021, March 28). *N-Gram model*. Devopedia. https://devopedia.org/n-gram-model

Chen, X., Liu, X., Qian, Y., Gales, M., & Woodland, P. C. (n.d.). *CUED-RNNLM — An open-source toolkit for efficient training and evaluation of recurrent neural network language models*. IEEE Xplore. https://ieeexplore.ieee.org/document/7472829

Jurafsky, D., & Martin, J. R. (2021, September 21). *N-gram Language Models*. Stanford University. https://web.stanford.edu/~jurafsky/slp3/3.pdf

Li, X., Qin, T., Yang, J., & Liu, T. (2016). *LightRNN: Memory and Computation-Efficient Recurrent Neural Netwoks*. List of Proceedings. https://proceedings.neurips.cc/paper/2016/file/c3e4035af2a1cde9f21e1ae1951ac80b-Paper.pdf

Mikolov, T., Kombrink, S., Deoras, A., Burget, L., & Cernocky, J. (n.d.). *RNNLM - Recurrent Neural Network Language Modeling Toolkit*. Microsoft. https://www.microsoft.com/en-us/research/wp-content/uploads/2011/12/ASRU-Demo-2011.pdf

See, A. (n.d.). *Natural Language Processing with Deep Learning*. Stanford University. https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture06-rnnlm.pdf