Danh Nguyen

Statistical Language Models

Introduction:

What are statistical language models, and what can we do with them in a text information context? Currently, various companies on the market are utilizing these statistics language models within their business to build essential features. This document's goal is not about the popularity aspect of these models but, instead, is to focus on the technical aspects of these models and how they are currently being used and developed in the businesses mentioned above. Specifically, after the initial overview of all the statistical language models, this document aims to directly compare the skip-gram language model and the n-gram language model.

Body:

First and foremost, some of the widely used cases of statistical language models are:

- Speech recognition (Siri, Google Assistant)
- On-screen autocompletion (assorted keyboards in various OSs)
- Machine translation (Google Translate)
- Sentiment analysis (HubSpot's Service Hub, Repustate, Lexalytics)
- Parsing tools (spell checking tools)

Usually, these systems do not work well or require several iterations and user inputs to get the correct/desired results. The statistical language models, in these cases, are often restricted due to computational constraints (mobile smartphones). Furthermore, users require quick results, which led to even more demand for a fast algorithm that can provide accurate results in a matter of seconds instead of minutes or hours. For example, imagine that the autocomplete feature on your smartphone took 30 seconds to suggest the next word. Nobody, in their right mind, would use it.

Currently, on the market, there are mainly three popular language models:

- Continuous space: in this model, words are arranged into combinations of weights in a neural network. This model is advantageous in cases where the data set of words is enormous and includes many unique terms.
- N-gram: the most straightforward approach to language modeling. "n" can be any number and defines the length of the sequence of words being assigned a probability. There are variations of n-gram models such as bigram, trigram, unigram, etc.
- Exponential: This model is a combination of n-gram and feature functions. This model is based on the principle of entropy. The best choice among the distribution is the one with the highest entropy. This model is exceptionally effective when accurate results are highly desirable.

The simplest statistical model that will provide us with enough context to discuss the n-gram language models is the unigram language model. In this language model, all words from the vocabulary (V) are equally likely to be generated.

$$p(w_i) = \frac{1}{|V|}$$

The maximum likelihood estimation in a unigram model is provided (Engati):

$$p(w_i) = \frac{\text{count}(w_i)}{\text{count}(w)}$$

The only main advantage of this straightforward statistical model is that it trains exceptionally quickly. However, due to the simplicity of the implementation, the chance that it can provide good suggestions for autocomplete systems and recommendations is doubtful. A much better implementation of this model is called the n-gram language model. This is the main algorithm implemented almost everywhere to satisfy quick, semi-accurate, most-likely recommendations/predictions. Furthermore, the unigram language model has been thoroughly discussed during lectures. Because of that, assuming that we have the basics of the unigram language model, we are moving forward to better understand the variations of this hugely popular algorithm.

The n-gram language model is significantly better than the unigram model because it incorporates contexts when generating predictions. For example, this is the formula for a bigram language model (Engati):

$$p(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

It considered the word that is usually come before the current term. This behavior significantly boosted the likelihood of a suitable outcome. However, as you can see, we can substantially increase the possibility of a positive outcome by looking back even further. The main limitation to this method is that if the word chain is set too high, most advantages to this method (good performance, simple model) are no longer present. Furthermore, suppose the data set contains too many unique words. In that case, this model is highly ineffective since the possible word sequences will increase to the point that predicting the next word becomes a random word generator task. To further enhance this approach, we can also perform interpolation, which means we use all n-gram models (from 1 to n) and then assign a weight to each of these models to predict the outcomes. This method is preferred and extensively used due to its incremental performance nature.

Another way to improve this particular model is to perform smoothing. This is similar to what is already covered in the lecture. Smoothing allows us to deal with words that appear in documents but not in the query or vice versa.

Assuming that, by now, we are already familiar with these methods. We will review and compare the skip-gram language model to the n-gram language models. For the skip-gram language model, specifically the word2vec skip-gram language model, the objective is to predict the context (the nearby words) given a word in a sentence. The way to perform that is to train a neural network with a hidden layer to perform a certain task. However, the interesting thing is that we will not use the neural network to predict the nearby words. The goal here is to learn the weights of the hidden layer. These weights turn out to be the word vectors that we can use to get the words that are most likely to be “nearby” to the current word.

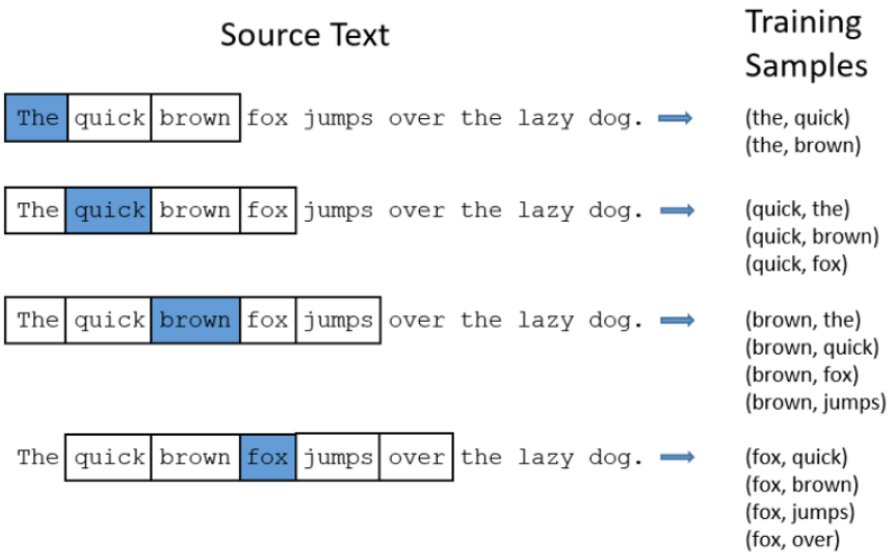
First, the mentioned neural network is to do the following:

- The input is a word that is in the middle of a sentence.
- The output is a vector that contains all the probabilities of every word in our vocabulary that can be randomly selected nearby the current word.

The probabilities are going to show how likely it is to find each word in the vocabulary near the input word. For example, if you give a trained network the input “king”, the output probabilities for words like “castle” and “knight” are going to be much higher than unrelated words like “banana” or “noodle”.

To obtain this probability matrix to predict the most likely word given a word, these are the steps that we need to follow.

First, we will give the neural network word pairs found in our training documents. This illustration below clearly shows how that is going to be performed (Chablani, 2017).



Next, we will assume that there are currently 10,000 words in our vocabulary. Then, we will represent an input word like “monster” as a one-hot vector. A single vector representing the probabilities that a randomly selected nearby word is the vocabulary word (Chablani, 2017).

Furthermore, assuming that we are learning word vectors with 300 features. The hidden layer is a weight matrix with 10,000 rows and 300 columns. This is what we are looking for! By using this hidden layer weight matrix, with a 1 x 300 word vector for any word, you can get back the probabilities of all the words in the vocabulary as if they are being picked nearby the current word. With this piece of information, you can clearly use it in various mentioned applications above.

Now is the most important section, between n-gram language models and skip-gram language models, which one is better? The answer to this question is highly dependent on the use cases. Below, I will try to list each method's most glaring pros and cons.

N-gram statistical language models:

- Pros:

  - Very easy to set up and run
  - Reliable result
  - Good performance in most instances
  - It can be used in other models (even skip-gram language models)
- Cons:
  - For data sets that have a high amount of unique words, this method's performance cannot be guaranteed.

Skip-gram statistical language models:

- Pros:
  - Better results than the n-gram model.
  - Can be used instead of other methods when appropriate
  - Can work with data sets that have many unique words
- Cons:
  - Performance is bad compared to n-gram
  - More set up steps

Summary:

Overall, what I learned from my analysis of these two methods is that one should select a statistical language model that can work well with the current data set that one is working with. Both of these statistical language models are used very frequently in many applications across the industry. In the end, it is the responsibility of the data scientist/machine learning engineers to select the most efficient model(s) for their particular use cases.

References:

Dehdari, J. (n.d.). A Short Overview of Statistical Language Models. Retrieved October 5, 2022, from https://jon.dehdari.org/tutorials/lm_overview.pdf

Statistical language modeling. Engati. (n.d.). Retrieved October 5, 2022, from https://www.engati.com/glossary/statistical-language-modeling

Chablani, M. (2017, August 21). *Word2Vec (skip-gram model): Part 1 - intuition.* Medium. Retrieved October 5, 2022, from https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b