Improving Language Model Performance with Smarter Vocabularies

Brad Jascob *bjascob@msn.com*

Abstract

In the field of Language Modeling, neural-network models have become popular due to their ability to reach low Perplexity [1] scores. A common approach to training these models is to use a large corpus, such as the Billion Word Corpus [2], and restrict the vocabulary to the top-N most common words (aka tokens). The less common words are then replaced with an "unknown" token. These unknown tokens then become a single representation for all low occurrence words which may not be closely related semantically. In addition, some closely related tokens, such as numbers, may be common enough to be given a unique integer ID when we might prefer that they be combined under a single ID.

In the following article, we'll explore using part-of-speech (POS) tagging to identify word types and then use this information to create a "smarter" vocabulary. Using this smarter vocabulary, we'll show that it achieves a lower perplexity score, for a given epoch, than a similar model using a top-N type vocabulary.

Keywords: Language Model, Neural Network, Perplexity, Artificial Intelligence

1. Introduction

When creating a language model, we are trying to create a system that can recognize patterns of words and predict the probability of the next word in the sequence. The issue arises that we generally work with a vocabulary of very specific words, rather than grouping them into categories. This leads to situations where predicting the next word in a sequence is completely ambiguous. For instance, if you were given the phrase "I left for the movies at" you would probably predict the next word to be a time or possibly a location. Given just this small piece of text, it's impossible to be more specific and guess the exact time the person left. However, the typical neural-net language model is asked to do just that. Depending on how the vocabulary was derived, it may have multiple different tags to choose from, or they may have all been lumped under an "unknown" symbol.

In another example you might be given the phrase, "The capital of Arkansas is". Here the obvious next words are "Little Rock", however this is a very specific piece of knowledge that likely does not show up verbatim in the training corpus. Such specific answers are better suited for a Question/Answer task where a lookup table of facts is used. For modeling sentence patterns we may be better off generalizing this sentence to "The capital of <LOCATION> is <LOCATION>". This generalization takes the knowledge portion out of the pattern and has the added bonus of recognizing that some locations, such as "Little Rock" contain multiple word tokens. Again, typical vocabularies today may include many locations in their vocabulary and/or lump them under an “unknown” token, neither of which is optimal for pattern learning.

While today we typically don’t directly convert words into categories, it is common practice to train word embeddings in the input stage of the network. When properly trained, words that show up in similar contexts are expected to show up close to one another, based on their cosine similarity. In concept this provides a generalization similar to what is discussed above.

There are some potential deficiencies to this however. The first being that for low frequency words there may not be sufficient information to fully train the vector. For instance, if you only have the two phrases, “I left the house at 12:35.” and “We went to the 6 PM movie.” to train the vectors for the word tokens “12:35” and “6”, it’s very possible there would not be enough information for the network to properly locate these close to one another. In addition, the token “6” may show up in significantly different contexts such as “There were 6 of us.” in which case we may not want it to be encoded closely to “12:35”.

The second deficiency with word vectors is at the output stage of the network. Regardless of how words were encoded in the input, networks today often use a one-hot encoded output. In this case the net must choose which token is the most likely. For the phrase “I left for the movies at”, there are an infinite number of possibilities. One would expect a well trained network to rank all tokens representing a time very high, but because it’s likely there are many of these in the vocabulary, each would be expected to have a low confidence score and thus drive the perplexity higher.

2. Proposed Improvement

One potential solution to these issues is to use a Natural Language Processor to preprocess the training corpus and group words that represent, names, times, etc.. into more general categories while leaving dictionary words the same. By doing this preprocessing step, we are injecting outside knowledge of the word category and not relying on our network to learn this from the training corpus, which may not be well suited for such distinctions.

3. Experiments

In the following experiments, the Stanford CoreNLP [3] software was used to perform a processing task. The software was fed sentences from the Billion Word Corpus and asked to tokenize words, identify the Part-of-Speech (POS), and tag applicable words with Named Entity (NE) tags. Using this information the words were given unique integer IDs based on a simple grouping scheme. Several different schemes were used, including the typical method employed today of keeping the top-N tokens and lumping the rest under an “unknown” tag. The following details the different methods tested.

Simple : This follows the standard method of keeping the top-N tokens and lumping the remaining low occurrence ones under an “unknown” tag. The number of tokens kept is the same as the resulting number of tokens in the *Smart A* vocabulary. This was done to facilitate comparisons by minimizing the possibility that the overall number of tokens in the vocabulary was a factor influencing the final perplexity.

<u>*Smart A*</u> : In this method, if a word is identified as an NE, that tag is applied. If not an NE, then the POS tag is used for proper-nouns, cardinal-numbers and punctuation. If none of these apply, the word is looked up in a dictionary. If it exists there, the word itself is used as the tag. Finally, if the word doesn't fit any of the above criteria, the POS tag is used. For all instances of NEs or POS tags, if identical adjacent tokens are detected, only one is used. For example, "Little Rock" translates to LOCATION.

The dictionary used is a pre-processed version of the American English [4] dictionary. Preprocessing converts words to lower-case and strips suffixes such as "'s". The final dictionary contains 72,330 words, including proper nouns.

<u>*Smart B*</u> *:* Same as SmartA but NE tags are not considered. This has the effect that NEs such as LOCATION or PERSON are grouped under their proper noun POS tags.

<u>*Smart C*</u> *:* In this vocabulary, punctuation is first grouped and then words looked up in the dictionary. If a word does not appear in the dictionary, the POS tag is used. NE tags are not considered.

It should be noted that because all of the S*mart* vocabularies default to using the POS tag if none of the earlier criteria are met, there are no words that get grouped under an "unknown" tag.

After tokenization and grouping, the training data was then fed into a single layer network consisting of 2048 LSTM cells with 512 projected units, using a 0.1 dropout rate. The Adam optimizer was used for minimization. Due to the amount of time it takes to train these networks on the available hardware, the Billion word corpus data was distributed across 800 epochs and a single pass through the data was made. Performance was then tested at epochs 200 and 800.

It should be noted that other researchers have shown better results by using different optimizers and making multiple passes over the entire corpus. In the paper *Estimation of gap between current language models and human performance* [5] the researchers achieved a 45.3 perplexity on a similar model.

The concession to take one pass over the corpus allowed the experiments to be completed in reasonable time-frame while still showing the improvements that can be made with these techniques.

Table 1: Results of Language Model training

Model	**Vocab**	**Num Tokens**	**Perplex @200**	**Perplex @800**
L1-2048-512-0.1	Simple	64,831	72.1	62.8
L1-2048-512-0.1	SmartA	64,831	52.0	46.2
L1-2048-512-0.1	SmartB	66,043	49.1	43.6
L1-2048-512-0.1	SmartC	69,222	62.6	55.1

4. Analysis

From the data in *table 1*, we can see that the *Smart A* grouping improves the overall perplexity score by more than 15 points from a simple top-N vocabulary, under these test conditions.

The *Smart* B vocab generically groups entities under proper noun POS tags. These POS tags are differentiated as either singular or plural (NNP or NNPS). This vocab yields a small additional improvement over *Smart A.*

The scores for both of these are close to or slightly better than the previously mentioned paper which achieves a 45.3 perplexity. The score here is achieved using significantly less processing time. It remains to be demonstrated how much these scores can be further reduced by the use of a multi-pass optimization process.

The *Smart C* vocabulary does not use NEs or group proper nouns together. It gives unique IDs to words that appear in the dictionary and assigns the POS tag if they don't. In this case, the perplexity increases (gets worse) noticeably from SmartA and B. However, the perplexity is still significantly better than the simple top-N vocabulary.

5. Conclusions and Future Work

The results shown here indicate that grouping words into categories will improve the overall perplexity score of a language model significantly when training under similar conditions.

While the improvement is noticeable when training over a single pass of the data, it remains to be demonstrated how much these methods improve the final perplexity of a net trained using more extensive training setups.

The code used in the above experiments can be found on GitHub at https://github.com/bjascob/SmartLMVocabs.

6. References

[1] Perplexity: See https://en.wikipedia.org/wiki/Perplexity.

[2] Google Billion Word Benchmark for Language Modeling: See https://opensource.google.com/projects/lm-benchmark

[3] Stanford Core NLP software: see https://stanfordnlp.github.io/CoreNLP

[4] american-english dictionary: From Ubuntu 18.04 wamerican package, which is created from SCOWL. See https://packages.ubuntu.com/bionic/wamerican

[5] Xiaoyu Shen, Youssef Oualil, Clayton Greenberg, Mittul Singh, Dietrich Klakow *“Estimation of gap between current language models and human performance”*