Implementation and Analysis of the Skip-Gram Model

Christopher Caballero
Deep Learning, FSU

Abstract

The Skip-Gram model introduced an efficient and intuitive way to create word embeddings with various positive qualities. Among the most important is analog relationships and word sense capturing. However, the naïve implementation of this model carries a large computational burden in learning both input and output word embeddings, with the output embeddings requiring updates across all words in the vocabulary during training. This makes it infeasible to learn on a large corpus with many unique words in the vocabulary. The referenced work expanded upon the original model by developing useful optimizations, Hierarchical Softmax and Negative Sampling, that can efficiently learn word embeddings. In this paper, I cover the naïve implementation to learn word-embeddings on a toy example of 26 vocabulary words, and Hierarchical Softmax to optimize upon it and analyze the increase in efficiency.

Introduction

Learning vector representations of words opened a gateway in natural language processing for understanding how models learn on a corpus, and the various meaningful correlations that can be found with a simple model. It is well known that a simple 2-layer neural network is Turing Equivalent, being able to learn any computable or logical function. Being able to find the correct model architecture and learning rule is not necessarily intuitive, and the extent to which the Skip-Gram model performs is particularly impressive and is what drew my attention to this simple architecture. Indeed, it shows that a streamlined and simple approach can often be better than a more complicated one and can produce impressive results with meaningful word embeddings.

This simple model uses two-layers, an embedding matrix to choose the target hidden vector for the current word in the corpus, and a dense softmax classifier as an output layer. The main issue we arrive at in the naïve implementation of Skip-Gram is the computational burden of performing an O (H x V) operation, where V is the size of the vocabulary and H the size of the hidden layer, every iteration to update the output word embeddings. This provides a strong motivation for finding efficient ways to compute the output word embeddings and reduce the number of operations needed to learn the vector representations as we scale the number of words in the vocabulary.

To increase the efficiency of the full softmax layer I focus on implementing hierarchical softmax, first introduced by Morin and Bengio [3]. Of the two methods presented by Mikolov et al., I focused on implementing hierarchical softmax, since it uses not only data structure optimizations, by using a binary tree as the output layer, but it also optimizes further upon this notion by implementing a Huffman Tree, which uses word frequencies to find the appropriate tree depth for each word in the vocabulary [1]. This reduces the computation significantly by requiring only O (log V) steps on each iteration, and often even less, since the tree is not required to be balanced, and is in fact optimized to reduce operations for more frequent words.

This rest of this work will be divided into two sections and a conclusion. The first covers the implementation of the skip-gram model with a full softmax layer, alongside a hierarchical version of softmax. The second will cover an analysis of the resulting word embeddings and the scalability of the models as vocabulary grows. I will then cover what I learned from this project and how I hope to further this work in the future.

Section 1

For the skip-gram model with a full softmax layer, we define the embedding and weight matrices as orthogonal matrices with random initialized values in the range [0, 1). I extract text from a toy corpus with a vocabulary of 26 words derived from the Orhan Pamuk quote, “Every person had a star, every star had a friend, and for every person carrying a star there was someone else who reflected it, and everyone carried this reflection like a secret confidante in the heart.”

Each word is converted to a unique integer and passed in sequence from left to right though the model as a potential target, w_I. A window size is defined as a hyper-parameter and the words surrounding the target in the range of the window are used as its context. The model then must learn to predict the words in the context, $w_{o,1}, w_{o,2}, \dots, w_{o,C}$, using a softmax classifier. This learning rule can be expressed with the formula

$$E = -log\ p(w_I)$$

where E is the loss function for our model.

Given this learning rule, we can learn various properties about the words through their embeddings, such as analog relationships. However, for the purposes of this paper, we do not study analog relationships. Instead, we simply seek to show that the context predictions can match the ground truth well given a couple hundred epochs of training on the toy corpus. Figure 1.1 shows the loss on the model over time, and Figure 1.2 shows a comparison between the context predictions made by the model before and after training with respect to the ground truth most likely context.

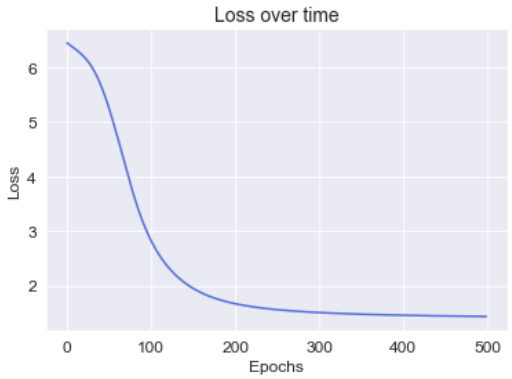

Figure 1.1: The loss on the model over time. The model appears to learn the function well by about 200 epochs.

Figure 1.2: Context prediction for the target "person" before and after training in comparison to the ground truth on the hierarchical softmax model.

Next, we consider the softmax model with a hierarchical softmax layer. As in the previous model, we define an embedding matrix with orthogonal values in the range $[0, 1)$. We use the same corpus as before. The process differs when considering the output layer, which uses a binary tree that has the vocabulary words in the leaves. We use the $V - 1$ inner units as computational nodes, each with an output vector which must be learned to maximize the probabilities which result from traversing the tree, where V is the size of the vocabulary. To get our predicted output probability for a given context word w_O, we use the formula

$$p(w = w_O) = \prod_{i=1}^{L(w)-1} \sigma\left(\left[n(w,j) = ch\left(n(w,j)\right)\right] v_{n(w,j)}^{T} \cdot h\right)$$

where $[n(w,j) = ch(n(w,j))] = 1$ if the next node along the path to the leaf node is the left child of the current node, and -1 otherwise [3]. The inner product is between the current computational node's output vector and the hidden layer. We take the resulting value and apply the sigmoid activation function to map it into a value in the range $(0, 1)$. We carry on this multiplication for each node along the path to the leaf, which is of length $L(w) - 1$. The learning rule is expressed by the formula

$$E = -\log p(w_I) = -\sum_{i=1}^{L(w)-1} \log \sigma\left([n(w,j) = ch(n(w,j))]\, v_{n(w,j)}^T \cdot h\right)$$

repeated for each word in the context [3].

To implement this, I created a Node class, which carries the important information for each computational node, such as the path from the root and the output vector associated with it. Additionally, I made the HuffmanTree class, which builds the tree for the given vocabulary using a Huffman Tree. I chose to implement the binary tree in this way because a Huffman tree has some nice properties, such as every node which is not a leaf has both children, and the depth to the bottom is proportional to some conditional measurement [1]. In our case, we chose to make words which appear more frequently to have a shorter depth in the tree, ideally saving time on computation in the long run. This optimization is the same as is used in the paper by Mikolov et. al., and is a common implementation feature when developing hierarchical softmax for the skip-gram model [1,2,3]. Lastly, we create a model class which puts together the layers defined above and trains the model in a similar fashion to the full softmax model.

Figures 1.3 and 1.4 are equivalent to those above, allowing us to compare how well the model appears to do after training with the full softmax version:

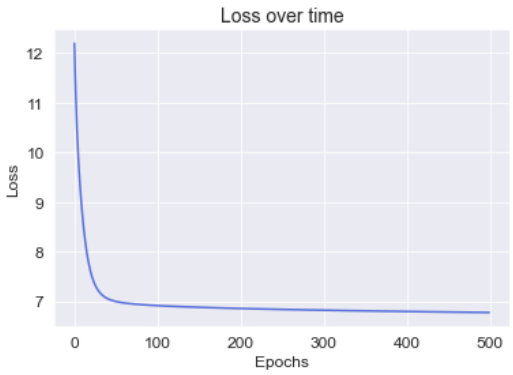

Figure 1.3: The loss on the hierarchical softmax model over time. The model appears to learn faster, converging at around 50 epochs.

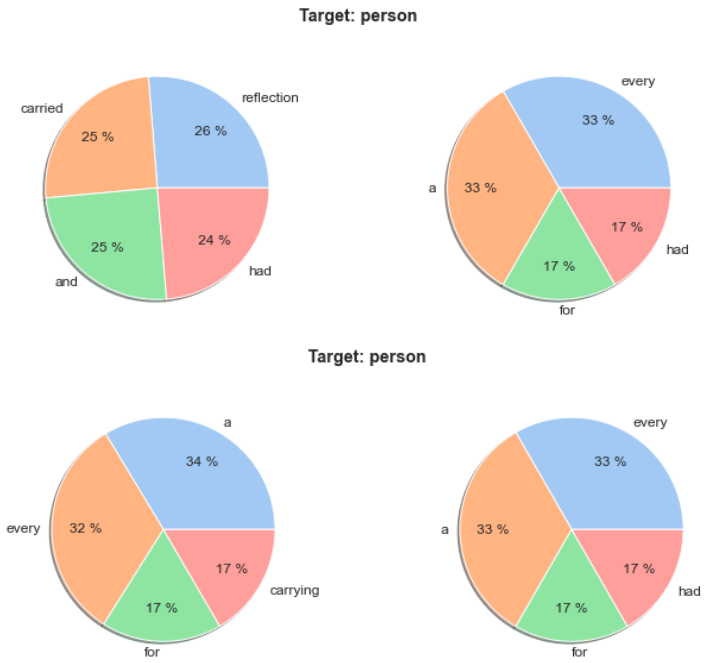

Figure 1.4: Context prediction for the target “person” before and after training in comparison to the ground truth on the full softmax model.

In the interest of space, I will not expand upon the backpropagation steps implemented in my work, but the full notebook will be made available alongside this paper for those interested in seeing the algorithms and functions developed to bring this together.

Section 2

In this section we will cover a brief overview of the time-complexity of the models with respect to V. It was expected that the hierarchical softmax would perform faster than the full softmax model, since the whole point is that it is an optimization on the number of computations in the output layer. What I did not consider was that for the size of vocabulary I was using, these results were not yet apparent. Given the range of vocabulary sizes tested, it is likely that the performance does not outweigh the overhead of using the various classes I developed for this purpose. I believe that the performance increase would likely be noticeable if the size of the vocabulary grew into the thousands. Previous work processed over 100 billion words in a day, largely due to the full parallelism employed [1]. This is the scale at which the increased performance of the optimizations model would be most clear. In our case, we process Pride and Prejudice by Jane Austen, which has over 5000 unique words.

Figure 2.1 shows the amount of time needed to train the model for one epoch using the hierarchical softmax model versus the full softmax model. We can see that both increase in time approximately linearly as the vocabulary size scales.

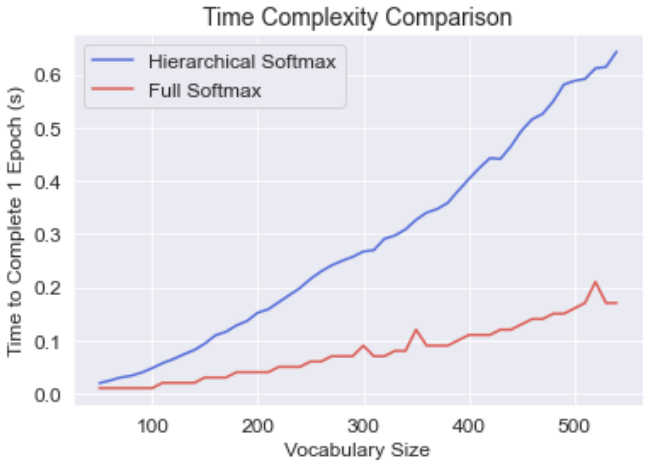

Figure 2.1: Amount of time to complete 1 epoch of training for vocabulary size in the range [50, 550]. Hierarchical Softmax not only appears to be slower, but the gap grows over time.

Since we know that the hierarchical model should scale at an $O(log\ V)$ rate, it is apparent that we have a narrow view of what this model can do. These results were limited because I could not find a plain text corpus with enough words to test this out, and further study is needed to understand why it consistently takes much longer than the full softmax model. The largest corpus I could find was a full novel and it still only had about 5000 unique words. This is the limit that is reached without a proper mechanism for sifting through and preprocessing various online corpora, such as Wikipedia documents.

Conclusion

As shown, both models can learn their context and even perform relatively well on much larger corpuses than previously anticipated. Although it was expected that the performance increase would be more noticeable in the softmax model, I believe this is partly limited by the data that was able to be collected. Given more time I would like to further clean the code and avoid redundancies in memory and computation, to get the most accurate results on the time-complexity comparisons between both models. Implementing the full softmax model as a proper class may increase the overhead enough to see if that made an actual difference in the performance, since the average times to complete the computations are very short to begin with. Also, I would like to expand the data collection capabilities to be able to collect trainable data from Wikipedia. This could also be reconciled by using open-source libraries to read in corpora and encode them. For the purposes of this project, I went ahead and did all the utilities by hand.

References

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems 26, pages 3111–3119.

[2] Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In Proceedings of the international workshop on artificial intelligence and statistics, pages 246–252.

[3] Rong, Xin. 2014. word2vec Parameter Learning Explained.