Evaluating the Effects of Different Optimization Techniques on Word Embedding Performance

Joseph Bongo

10/22/2019

Introduction

Semantic Word Embeddings

Semantic word embeddings have revolutionized the field of natural language processing, allowing us to far more accurately represent text data in a quantitative way that can be used for a variety of modeling applications. Earlier methods for embedding text into numerical representations involved simply one hot encoding each word in a given corpus. For example, suppose we have a corpus that is a single document reading, "Hello, my friend". In this case, the embedding corresponding to the word "friend" would be the vector $[0, 0, 1]$. While this is can be somewhat useful, it does not consider the meaning of the actual word, but only the position where it occurs in a given corpus. The dimension of this vector is also completely dependent on the size of the vocabulary used in the corpus. Then in 2013, Google published a paper about algorithms that their research scientists had developed called Word2Vec, which used transfer learning to produce embeddings related to the meaning of the word that perform much better in modeling applications than the traditional one hot encoded word vectors, as a form of dimensionality reduction [Mikolov, Tomas, et al.].

Word2Vec CBOW model

Word2Vec's Continuous Bag of Words model assumes that looking at the words that are used within the same context as a given word gives an idea about the word's semantics (or meaning) [Mikolov, Tomas, et al.]. Therefore, this allowed the meaning of a word to be represented in a quantitative way more easily. Word2Vec is able to generate these embeddings by feeding the one-hot encoded vectors for the context words around a target word that we are trying to encode [Mikolov, Tomas, et al.]. We will call this word w_t. This context, which is typically the vectors for the two previous words and two words that occur next $(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$, is used as the input of a shallow neural network to classify what the target word is [Mikolov, Tomas, et al.]. While this is not particularly useful on its own, this is a form of transfer learning as the values of the hidden layer nodes of the neural network are now a representation of the different contexts of the output word w_t. The paper does not go into too much detail about how the model is trained, but the weights of neural networks are often optimized using gradient descent with a fixed learning rate. I think that some of the algorithms that we have learned in S&DS 630 can do better.

Project, Algorithms, and Evaluation

I would like to try different optimization techniques such as gradient descent with exact line search, gradient descent with backtracking line search, and Newton's method along with the fixed learning rate gradient descent. The goal of this is analysis to see if different optimization techniques of the neural network weights can produce better word embeddings, or if it perhaps does not matter which method is used and the weights converge to the optimal values regardless. Evaluating the performance of word embeddings can be slightly subjective, but I plan to do this in at least two ways. The first way is to observe the cosine similarity of the vectors for given set of synonyms, as this is the standard measure used to evaluate semantic similarity. The second is to evaluate the accuracy of a simple classification task where the input is the embedding

only (i.e. using word embeddings to classify whether a word is "positive" or "negative"). The assumptions are that better embedding techniques will produce more similar synonyms as well as offer more insight when used as input for modeling. The algorithms will be evaluated not only on the performance of their corresponding embeddings, but also on factors such as speed of convergence and total run time. Each optimization method will be used in methods with different epochs to give us a better idea of convergence as well. If certain algorithms, such as Newton's Method, or gradient descent with exact line search, prove to be too computationally expensive to run in practice, then I can instead evaluate other algorithms frequently used for optimization in neural networks such as stochastic gradient descent or Adam.

Optimization Problem

Set-Up

For this optimization problem, our objective function is the cross-entropy of the softmax layer of the neural network in question. Therefore, if we have a vocabulary V that is the set of all words in the corpus, such that $|V| = v$, we produce a $4v \times 1$ vector ι of the one hot encoded input values of words $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$ and another vector h of hidden layer values for a hidden layer of n neurons (i.e. h is a $n \times 1$ vector). Let ω_1 be the $4v \times n$ dimensional matrix for the weights from the input layer to the hidden layer and let ω_2 be the $n \times v$ dimensional matrix for the weights from the hidden layer to the softmax layer θ. Therefore, for some element-wise activation functions a_1, a_2, we have $\theta^\intercal = a_2(h^\intercal \omega_2 + b_2)$ and we also have $h^\intercal = a_1(\iota^\intercal \omega_1 + b_1)$, where b_1, b_2 are bias terms. Typically a_1 is simpliy the identity function, but in this case it may make more sense to use the ReLU function. a_2 must be the softmax function.

Problem Statment

We know that since the neural network will be a classifier, the loss function that we would like to minimize is the cross-entropy loss. Therefore, this is our objective function and is listed below. Remember that w_t is the one hot encoded vector of the target word, meaning that it is the indicator of correct classification. Thus, if we let w_t be the $v \times 1$ target word indicator and θ be the $v \times 1$ vector of the output layer of our classification network, we have:

$$f_0 = -\sum_{i=1}^{v} w_{t,i} log(\theta_i) = -(w_t^\intercal log(\theta))$$

We know that θ is a function of ω_1, ω_2 and therefore we will rewrite it as $\theta(\omega_1, \omega_2)$. This gives us

$$f_0(\omega_1, \omega_2) = -(w_t^\intercal log(\theta(\omega_1, \omega_2)))$$

a_2 will be a softmax activation function and intuitively θ represents a vector of probabilities. Therefore, it will by default have the following constraints.

$$\theta_i \leq 1, \theta_i \geq 0, ||\theta||_1 = 1$$

That being said, this does not mean that there is any contraint placed on the actual weights. Therefore, our optimization problem is the unconstrained problem below:

minimize: $f_0(\omega_1, \omega_2) = -(w_t^\intercal log(\theta(\omega_1, \omega_2)))$

In order to configure the loss function and perform the actual optimization, I plan to write the code for the neural networks, (and thus for the code for the optimization) in Python. Packages will be used for smaller indivdual tasks within this, but the greater architecture and optimization methods will be written by me directly. That said, packages will be used freely for the evaluation of the embeddings after optimization is conducted. It is also important that we are given a large corpus so that the words in question occur in as

many contexts as possible. For this, I will likely turn to publicly available data sources, or sample corpora including within Python modules related to natural laguage processing such as NLTK. Input data of this size will lead to very computatoinally expensive calculations, but also produce higher performing embeddings. This will likely be something that I have to balance, but I have a developer PC and free cloud space from my student Amazon Web Services account that can likely be leveraged if I run into these problems. The corpus also need not be extremely large as all of the embeddings will be generated from the same corpus, meaning that all of the performace metrics will be relative.

Expected Deliverables

The final expected deliverables for this project are listed below:

- Word2Vec style neural networks coded in Python for the optimization techniques explored (gradient descent with a fixed learning rate, gradient descent with exact line search, gradient descent with backtracking line search, Newton's Method). The performance at a several numbers of total epochs will also be tested to get an idea if one algorithm converges faster.
- An analysis of how the embeddings produced by each network perform in an identical simple classification task.
- An analysis of which network produces embeddings with a greater cosine similarity for a given set of synonyms.
- A detailed report of the advantages and disadvantages of each algorithm (including run time, steps to convergence, perforamnce, etc.) and an opinion of which technique is best for this task.
- Any other Python code used over the course of the analysis.

References

[Mikolov, Tomas, et al.] "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.