Skip-gram-based Word2vec Testing on Sentiment Classification

1. Word2Vec, Skip-gram and CBOW

1.1. Introduction to Word2Vec

Word2vec is a popular method for learning word embeddings, which are distributed representations of words in a vector space. These embeddings are learned using either the skip-gram or CBOW (continuous bag-of-words) model, which attempt to predict the context words given a target word or vice versa, respectively.

The idea behind Word2Vec is that, we not only want to use vectors to represent the words, but also want the vectors to capture ”meanings” of the words, e.g., we want to use the similarity of the word vectors for word A and B to calculate the probability of A given B (or vice versa). This is boosted by the traditional way to train a machine learning model: let the model learn the representation itself with as little human involvement as possible.

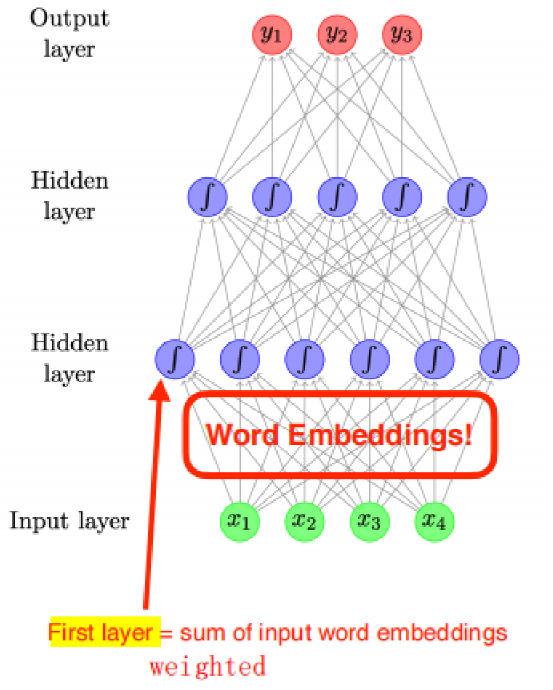


Figure 1. Word Embeddings in Topic Classification

Here’s an example of the application of Word Embeddings in topic classification. Notice that the first weight matrix is exactly the word vector matrix that we want, and then it will become the input to the next layer.

1.2. Skip-gram Model

1.2.1. Ojective Function

The objective function of skip-gram model is to minimize the (average) negative log likelihood:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log P(w_{t+j}|w_t;\theta)$$

which is equal to maximize the predictive accuracy. Here, with no perticular reason except for basic probability properties, we use the following formula to calculate $P(w_{t+j}|w - t;\theta)$:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

where o and c refer to a context word and a center word respectively, v_w the center word vector for w and u_w the context word vector for w.

Word2Vec uses 2 vectors to represent a word, partly because they won’t cause any tidious calculation with respect to the gradient. Another good explanation is that consider the case where both the word dog and the context dog share the same vector v. Words hardly appear in the contexts of themselves, and so the model should assign a low probability top(dog dog), which entails assigning a low value to vv which is impossible.

1.2.2. Gradients

The partial gradient for each parameters are calculated as follow:

$$\frac{\partial J(\theta)}{\partial V_c} = -u_o + \sum_{w=1}^{V} P(w|c)u_w$$

That is, we will use u_o, u_w to update v_c.

$$\frac{\partial J(\theta)}{\partial u_w} = P(w|c)v_c = \frac{\exp(u_w^T v_c)}{\sum_{i=1}^{V}\exp(u_i^T v_c)} v_c$$

That is, we will only use v_c to update u_w.

$$\frac{\partial J(\theta)}{\partial u_o} = P(o|c)v_c - v_c = \left(\frac{\exp(u_o^T v_c)}{\sum_{i=1}^{V} \exp(u_i^T v_c)} - 1 \right) v_c$$

That is, we will only use v_c to update u_o.

1.3. Negative Sampling

As we discussed above, the size of our word vocabulary means that our skip-gram neural network has a tremendous number of weights, which is hard to train. Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them.

To do this, first we choose K many words to be updated and keep any other word vectors unchanged. $K = 5\ 20$ words well for smaller datasets, while $K = 2\ 5$ is better for large models. Then we will select the words according to the probability

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4} \right)}$$

In the implementation, we just use the similar way as what the Word2Vec authors have done. We have a large samplign table, filled with the index of each word in the vocabulary multiple times, and the number of times a word's index appears in the table is given by $P(w_i)*$ table_size. Then to actually select a negative sample, we just generate a random integer between 0 and the size of the sampling table, and use the word at that index in the table.

Objective Function

$$J(\theta) = -\log(\sigma(u_o^T v_c)) - \sum_{w=1}^{K} \log(\sigma(-u_w^T v_c))$$

Gradient The partial gradient for each parameters are calculated as follow:

$$\frac{\partial J(\theta)}{\partial u_w} = \left[1 - \sigma(-u_w^T v_c)\right] v_c$$

$$\frac{\partial J(\theta)}{\partial u_o} = \left[\sigma(u_o^T v_c) - 1\right] v_c$$

$$\frac{\partial J(\theta)}{\partial v_c} = \left[\sigma(u_o^T v_c) - 1\right] u_o + \sum_{w=1}^{K} \left[1 - \sigma(-u_w^T v_c)\right] u_w$$

1.4. CBOW

In Continuous Bag of Words, the algorithm is really similar, but doing the opposite operation. From the context words, we want our model to predict the main word.

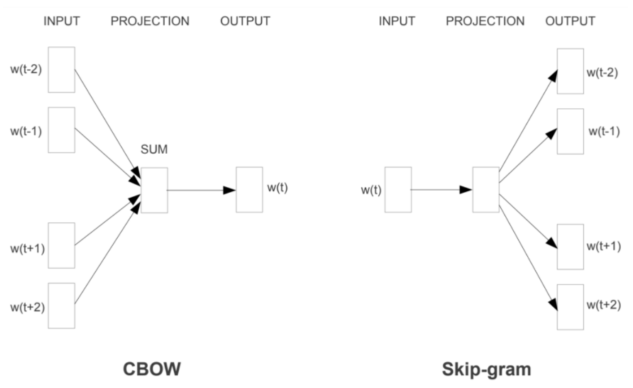


Figure 2. Network Visualization of CBOW and skip-gram

According to the original paper, it is found that Skip-Gram works well with small datasets, and can better represent less frequent words. However, CBOW is found to train faster than Skip-Gram, and can better represent more frequent words.

2. Sentiment classification

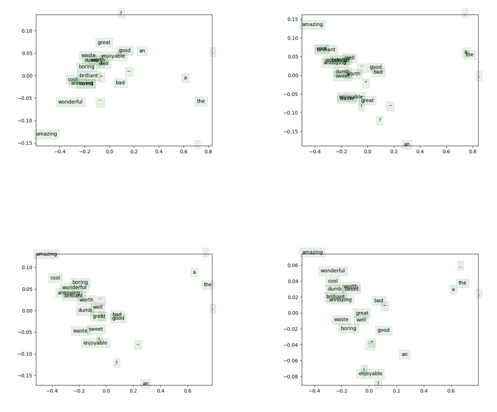
After calculating the word vectors, we can easily use it on other downstream tasks, like sentiment classification.

Note that here we simply construct the cross-entropy loss function and the gradient is the same formula in the last project.

3. Experiment Result

3.1. Word2Vec Visualization

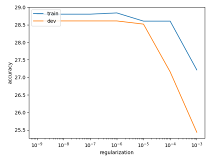
We have tried different combinations of the context window size C and vector dimension $dimVectors$. The figures below, clockwise from the top left corner, visualize the word vectors in the 2-D space of $C5dim10$, $C6dim10$, $C9dim10$, $C10dim20$ respectively.

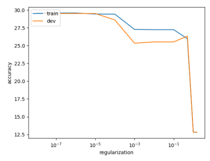


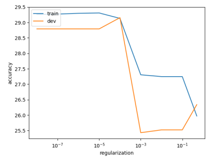
We can see the result did not as good as we assumed. But since we are training only a small little tiny toy-dataset with just 10 dimension as our feature size, the results already can discover part of the meanings of the words.

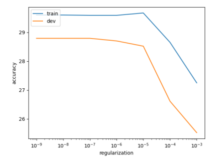
3.2. Sentiment Classification

We have tried different combinations of the context window size C and vector dimension $dimVectors$. The figures below, clockwise from the top left corner, represent the Accuracy score of sentiment classification for of $C5dim10$, $C6dim10$, $C9dim10$, $C10dim20$ respectively.









Overall, as the regularization coefficient increases, the accuracy decreases.

4. Summary and Thinking

During the experiment, we found that our Word2Vec did not performance well, since even Naive Bayes can achieve about 40% Accuracy, almost 33% better than our Word2Vec. From the original papers and some explanation docs, we assume that this is due to the relatively small dataset and vector dimension. In the original paper we can find that, most models that Word2Vec outperformed other models are with high vector dimension equal to or bigger than 300. Our vector dimension is really small compared to this baseline.

What's more, the computation power and computation time are also worth being taken into consideration. It is true that the given dataset is small, but even if we have larger dataset, we still can not train a great Word2Vec model limited by our time and computation resource. One possible solution may be turn to pre-trained models. In the future, we can have more comparison experiments on the performance difference between pre-trianed models and other typical machine learning models.

References

Goldberg, Y. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *Neural Information Processing Systems*, 0(0):0, 2014.

Mikolov, T. Distributed representations of words and phrases and their compositionality. *Neural Information Processing Systems*, 0(0):0, 2013a.

Mikolov, T. Efficient estimation of word representations in vector space. *Neural Information Processing Systems*, 0 (0):0, 2013b.