CODWOE - COmparing Dictionaries and WOrd Embeddings

Connor Horn and Dylan Sain

December 2021

1 Abstract

We attempted the CODWOE - COmparing Dictionaries and WOrd Embeddings SemEval task with the second track, gloss to vec. The ideal input was a definition and the output was a vector of the word corresponding to the definition. We used a very simple translation process and neural net to create these vectors. Our model turned out to be around 25% better than the baseline of 0.15132 given by SemEval with a final average cosine similarity of 0.18828.

2 Introduction

We decided to attempt the CODWOE - COmparing Dictionaries and WOrd Embeddings task as we had already done a few of the sentiment analysis problems throughout the class. We wanted to challenge ourselves and try to do something different this time around. We started with a few word2vec models, mainly the Google one and after experimenting with the models we came up with a solution. Using a function called most_similar in these pretrained models, we would pass in the definition and get out a word that is similar to the words in the definition. This is a good starting point, but needed more refining. We then passed the vector representation of that word into a 256×256 matrix W and compare that to the vector given by the problem. This would prove the basis for our training and our updating as we would slightly adjust the matrix based on the cosine distance from the answer we got and the correct answer. Overall this worked decently well, but we ran into issues with time to complete and processing power.

3 Background

In the reverse dictionary track we were given data in the format of an array of strings, acting as a dictionary, and a final vector that would provide as the ideal answer to the question. We decided to use the full English data, however ran into some problems quickly. The data proved difficult for us to start with as it was in two different formats. The definitions were strings and had yet to be translated into vectors, while the answers were in vector format. However after filtering the dictionaries with stop words, removing punctuation and capitals, and feeding it through a word2vec model we had the data in a format we desired.

4 System Overview

4.1 Most Similar

We started with a basic idea that most word2vec models had functionality to find a word similar to given words. Our idea was to use this functionality to find a word that was similar to the given words. This would provides us with a basic idea of a word or words that might be close to the words of the dictionary. For example, [group of relatives] might return words like, [family, groups, friends, families, siblings]. We would then take these vector embeddings and try to make something that looked like the given embedding. This is where the learning of our model came in.

4.2 Learning

These embeddings, while might be similar to the definition's embeddings, it doesn't necessarily mean that it was close to the given word embeddings. This is where we decided to implement learning in order to get our answer as close to the real one as possible. As these embeddings are vectors, we can do some matrix multiplication to manipulate the embedding to be closer to the answer. Essentially we were building a huge algebra problem and we were learning what to multiply our vectors by to get close to our answer. Using pretty basic machine learning techniques we would be learning the 256×256 matrix W by using the cosine distance between our answer and the given answer as our update. We used a learning rate of 0.1 and a batch size of 3.

4.3 The Math

We first had to create the math to update our learning matrix. Since we were dealing with vector space, and understanding that the difference between vector can be calculated using cosine we decided to start with that. We decided to use the cosine distance as our loss function $L(x, y) = 1 - cossim(x, y)$, where $cossim(x, y) = \frac{x \cdot y}{|x||y|}$, the gradient update simply becomes $\frac{\partial L}{\partial w_{ij}}$. We can say that $Wv = x$, and as such $x_i = \sum_j w_{ij} v_j$. Similarly $x \cdot y = \sum_i x_i y_i = \sum_i y_i \sum_j w_{ij} v_j$ and $|x| = \sqrt{\sum_i x_i^2} = \sqrt{\sum_i (\sum_j w_{ij} v_j)^2}$. Thus we can rewrite our loss function as

$$
\begin{aligned}
L &= 1 - \frac{\sum_i y_i \sum_j w_{ij} v_j}{|y|\sqrt{\sum_i (\sum_j w_{ij} v_j)^2}} \\
\frac{\partial L}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}}(1 - \frac{\sum_i y_i \sum_j w_{ij} v_j}{|y|\sqrt{\sum_i (\sum_j w_{ij} v_j)^2}}) \\
&= -\frac{|x||y| y_i v_j - (x \cdot y)(\frac{|y|}{|x|} x_i v_j)}{|x|^2 |y|^2} \\
&= \frac{x \cdot y}{|x||y|} \frac{x_i v_j}{|x|^2} - \frac{y_i v_j}{|x||y|} \\
\frac{\partial L}{\partial w_{ij}} &= (cossim(x, y) \frac{x_i}{|x|^2} - \frac{y_i}{|x||y|}) v_j
\end{aligned}
$$

is the update to our gradient for an each entry in the matrix W.

5 Experimental Setup

We used the basic train/test split that was provided by the SemEval task. We ran into issues early with the format of the data and the models we were using not recognizing various words. We made it so that the data was parsed with no capitals, punctuation or stop words. This allowed our models to focus only on what mattered and filter out the things that didn't. It also made sure that the words inside the dictionaries were being recognised by the most_similar function. If our corpus didn't recognize any of the words inside the gloss, which occasionally happened, we would simply set the vector v to all 1's. This is a very naive approach and could be immensely improved upon. We used this function on the pretrained gensim model with the nltk brown corpus and a vector size of 256. This model was small but it did provide a decent starting point for our model. For evaluation we decided to test on the same idea that we updated the W matrix with, the cosine difference. Since the data existed in vector space and we needed to see how similar the vectors were, which is evaluated by the cosine function.

6 Results

We tested out code using the devset that was given to us. After averaging all the cosine distances we received an average cosine distance of 0.18828. This wasn't super great, however it was 24.4% better then the SemEval's baseline. Although its not perfect and things could be improved it did get a better score and that was exactly what we were looking for.

7 Conclusion

In conclusion, the simple neural net approach applied to the output of the most_similar word2vec function proved to be somewhat better than the initial baseline given for the English word2vec in the SemEval for the Reverse Dictionary Task. In order to make this model better, there can be improvements made on many different aspects of the approach. First, the model corpus Brown that we used contained many unknowns words and had many problems. A word2vec that was trained in a similar domain to the gloss entries would have performed much better. Similarly, adding more layers to the neural net would help immensely. In this current implementation, only surface level knowledge about the features is being learned. Adding more layers allows the model to gain a deeper understanding about the connections between the vectors produced by the most_similar function and the vectors given by SemEval.

8 Acknowledgments

A big thank you to Professor James Martin for being our teacher throughout Natural Language Processing this year.