Natural Language Processing Word Sense Disambiguation Report

CS 7322

Wes Anderson and Kassi Nzalasse

**Problem –**

For our final project we chose to take a look under the hood of BERT to see how the model actually understood different senses of the word. In program 3 we trained BERT as a classification network. In this problem we used the same fine tuned models we had trained for program 3.

**Data Generation –**

These models were trained on a dataset of roughly 10,000 sentences per word split 50/50 for sense 1 and sense 2 for a total dataset size of 30,000 sentences. Since it is rather tough to generate 30,000 from scratch we used ChatGPT to generate them in batches of 100. The prompts we used was formatted as follows “Hello, I am trying to generate sentences using the word "rubbish" and the definition "nonsense". Could you generate 100?”

**Training BERT –**

After the sentences were generated they were processed into a csv so that we could utilize the pandas library for quick read in capability. We then started the training of the neural networks. This consisted of splitting the data into a training set, validation set, and a testing set which was randomly decided and given a split of 80/20/20. After the data had been partitioned we started 3 rounds of training which sent a sentence through and updated the neural network weights. Once all the training data had been sent through the validation data was used and predicted on. This was for us to see if the network was getting better as it was running which all 3 did get more accurate as the training went on. Then the training was restarted and this process happened 2 more times. Once all 3 rounds of training and validation completed the final testing data was sent through to test the results. The reason we withheld this data from the validation data is because we wanted to see how the networks performed on data that it had never seen before. With this training the models were saved so they could be loaded for quick use in other programs.

**Results –**

Overall all three models performed very well with two of them predicting 95% correct and the last model predicting 67.5% correct on the actual program 3 sentences. With these networks we wanted to see how well the word sense was grasped by the model. So as a modification to our original program we accessed the hidden states of the neural network. This was a multi dimension vector that was massive. Each token for a sentence had a corresponding vector that was 768 elements. Since we wanted to know how the computer handled only the word we were training for we only grabbed the 768 element vectors corresponding to “thorn,” “conviction,” and “rubbish” from each sentence. Then using PCA from scikit-learn we did dimension reduction to try and graph the data. We got PCA to work when reducing in two different directions, but what we wanted to see was not the numbers having a certain value for sense 1 and sense 2, but instead having 2 distinct clusters.

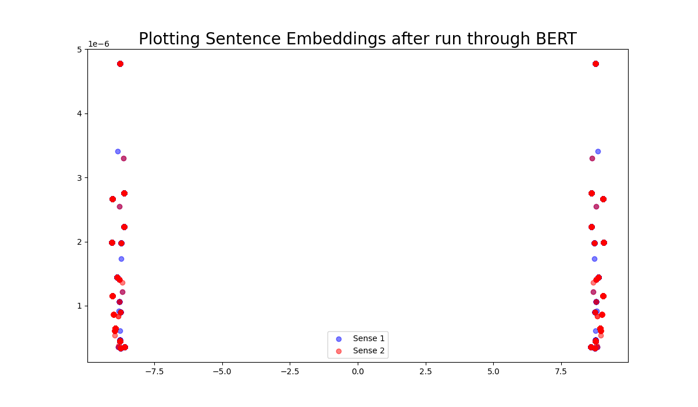
A red and blue dot diagram

Description automatically generatedA graph with numbers and lines

Description automatically generated

Conviction Embeddings Flipped PCA

A red and blue dot

Description automatically generated

Thorn Embeddings Flipped PCA

A red and blue dot

Description automatically generatedA graph with numbers and symbols

Description automatically generated with medium confidence

Rubbish Embeddings Flipped PCA

As you can see with both PCA version one and the flipped version of PCA we did not see the clustering that we had hoped to see form. We had hoped for a blue sense 1 cluster and a red sense 2 cluster in at least 1 of the graphs. Instead we see 1 distinct cluster in the left graphs and 2 clusters near the bottom in the right graphs. With this we thought that since the vectors were extremely high in dimensionality that maybe PCA was not a good measure of clustering. So we next tried to take the cosine similarities of vectors. We hoped to see that sense 1 vectors would have higher similarity values when compared with other sense 1 vectors and lower values when compared to sense 2 vectors, and the same thing for sense 2 vectors. So we took the cosine similarities and took 1 million datapoints for each graph which is not the entire dataset, but with 1 million it ran in a reasonable timeframe and it at least approximates the distribution of the similarity values.

If you would like to run any of our code please download and follow our guide found in our repository - <https://github.com/wesoa012/NLP-Program3>

A graph of a number of points

Description automatically generated with medium confidenceA graph of a number of points

Description automatically generated with medium confidenceA graph of a normality distribution

Description automatically generated

Thorn distributions

A graph of a number of points

Description automatically generated with medium confidenceA graph of a number of points

Description automatically generated with medium confidenceA graph of a line

Description automatically generated with medium confidence

Rubbish Distributions

A graph of a number of points

Description automatically generated with medium confidenceA graph of a number of points

Description automatically generated with medium confidenceA graph of a normality distribution

Description automatically generated

Conviction Distributions

With all of these distributions we see that when compared against each other the sense vectors have a lower similarity value. Granted that the similarity value is still above .9 for the cosine similarities when both senses are compared. This lines up with the results from the PCA graphs from before. Although since the dimensionality is so high it also may mean that comparing cosine similarities is also not the best option to determine the grouping of word sense.

**Follow Up –**

Something that we did not try is to compare the entire sentence embeddings for each sense. This may yield better results since even humans would have a hard time determining word sense without the context of a sentence. Something else to try would be beefing up the dataset even more and diversifying the sources. Of course human generated would be the best source for the sentences. It may be important to note that for classifying word sense the models that had more variation performed better, and again with the cosine similarity when comparing across the senses they had more variance. For conviction and thorn, ChatGPT created sentences that did not always contain the exact word “thorn” or “conviction,” but instead contained “thorny” and “convicted.” This may have led to a very slight difference in BERT’s sense of those words. The result of this has been better performance, so with a larger more varied dataset this may be exemplified even more.