Luna McBride

NLP Homework 3 Part 2

- What is the difference between pre-trained word embeddings and word embeddings trained on a specific dataset? How do they compare in terms of performance and generalization to new data?

  Pretrained embeddings, like those of GloVe, use models trained by the giants of the industry based on much more data than the average person's tiny computer can handle. They have been trained on many different sentences to strengthen the connections between the most common words, but faulter when given new or misspelled words. This means it is great for cases where common words are used, but niche industries would find it difficult to use given the likelihood of words it never learned appearing and the different usage of words that may be familiar with (think unionized in the public lexicon versus unionized in a chemistry context, read un-ionized in that case.)

  Word embeddings created from a certain dataset, such as with Word2Vec, are created just for the new data and thus can utilize the unique elements of the data set to better fit a niche topic. The data the common people can handle is much smaller than the big companies can however, so these tend to not have the strong connections were learned in the pretrained model. It can also be more susceptible to outliers, as the pretrained model's usage has it throw out random gibberish words created in preprocessing. This can be a blessing and a curse however, as some of the "gibberish" to the average person could have meaning to the most niche parts of places like Twitter and could be a strong factor that pulls the model to high accuracies. That or it is actual keyboard spamming to show intense surprise like can happen on Tumblr sometimes, and thus means nothing.

- What are the advantages and disadvantages of using pre-trained word embeddings for text classification compared to traditional methods like TF-IDF?

  Traditional methods do not have the hundreds of company hours and strong computing power the big companies have to produce the pretrained word embeddings, and thus connections are weaker for the most common examples. The pretrained version do not have the flexibility to learn new items however, making the traditional methods good for finding connections in underrepresented or niche areas.

- What are the key hyperparameters and design choices when training a Word2Vec model? How do they affect the quality of the word embeddings and the performance of the text classification model?

  The major hyperparameters shown off in this assignment were the number of epochs, feature size, context window, minimum word count, and sample.

  - The epochs hyperparameter was as simple as how many times the model would run through the data to make better vectors. This means if the model ran slowly, it may be better to lower this hyperparameter to finish the training faster, risking worse connections. If the model ran fast, though, that does not mean you should allow it to run for a million epochs. The improvement in loss would either stall or worsen after a certain number of epochs (which depends on which data was selected.) A good number in this case ended up being 10 to avoid the worst of that issue.
  - Feature size represented the size of the vectors. A smaller feature size made it too small to truly learn contexts in such a big dataset. A larger feature size made it take forever to run while not learning contexts as efficiently. The strong middle ground option for this data ended up being 300, as 200 had minimal speedup for a worse loss and 400 had too significant of a slowdown to justify its small decrease in loss.
  - Context window represented the size of the window the model would consider for context, or how many words around any specified word the model would look at to build context. It had a similar relationship as feature size, where small sizes did not consider enough context and larger sizes added too much runtime to justify the small difference in loss. The strong middle ground number here was 5, as 3 just could not meet the same standards and 7 took way too long to run for little benefit.
  - The minimum word count represented how many times a word had to appear in order to be kept. This hyperparameter was interesting since the default value is 5, but a value of 0 actually provided the best results. This is likely due to the odd nature of the dataset with its misspellings and niche concepts being presented. The model just could not do a good job capturing the erratic nature of the test data if low count values were dropped.
  - Sample ended up being the most interesting hyperparameter in this case. It represented how often more common words would be downsampled (per the Gensim Word2Vec documentation.) This means it would knock the high-quantity words down a peg to allow for the niche words to shine. This turned out to be the most interesting parameter in this case, as a 0 sample would keep the loss over 1 million for the first few epochs, the highest recommended value of 0.00001 provided a good jump to a lower loss, but values between made the loss become much better. It was to the point where adding another 0 (that is to say, turning 0.00001 into 0.000001, for example,) brought down loss by factors of hundreds of thousands. There may be some kinks in this process, as at a certain

point, loss became 0.0 or near 0 when fixing this hyperparameter alone. The results in the training process, however, showed that the values created with this hyperparameter set like this were not only great, but comparable to the GloVe representation. Training accuracy, precision, recall, and F1 for the Word2Vec representation came to between 70-75% across the board, while the GloVe representation stayed around 70-80%. The implications of this are interesting, as it is more likely than not that the sample helped the model better distinguish similarities in the dataset. The final sample value to allow for this was 0.00000000000000001, for reference.

- Talk about the different model architectures and hyperparameters you experimented with when training the text classification model. How did they affect the model's performance?

  I did not do as much experimenting on this end as I did with the embeddings side since I got a lot of that curiosity out of my system when I was first playing with Keras. The combination of intrigue I had with the sample in Word2Vec and the difficulty with packages that needed to be set up for GloVe took plenty of that energy out of me. Though, in comparison to Keras, I will note that Pytorch made it a lot easier to comprehend the input and output requirements, as the errors actually displayed shape differences clearly.

  I ended up creating two models, both using basic linear layers, ReLU as the non-linearity function in between all linear layers, and a Sigmoid non-linearity layer at the very end (as otherwise the values would try to go above one or below 0, and the binary crossentropy loss function did not like it when values were not between 0 and 1.) The factor I played with instead was a question I had following the discussion in class on dense versus sparce vectors, being what would happen if I had a hidden layer become denser then reopen up to create a sparse version in the layers. This was achieved by dividing the hidden layer size of 16 by 2 for the output of the first hidden layer, then having the second hidden layer output at 16 once again. The results were unremarkable, with only a small increase in recall occurring in some, but not all, reruns of the script after a re-shuffling of the data.

  The basic model without the dense to sparse difference was used to evaluate both the Word2Vec and GloVe embeddings for comparison purposes. Many of the runs came to similar scores of 75% across the board on average. I would say this is a massive success, given the weird spelling and niche concepts that exist in the model. My interpretation of these results is that the GloVe version had better results on the common words and concepts while the Word2Vec had better results on the niche concepts as expected, which is confirmed by the first five reviews they predicted differently from one another containing a lot of actor names or very in depth niche elements.