Vinny Pilone

CMPSC 497 Midterm Project

For the CMPSC 497 midterm project I choose to pursue named entity recognition, a task that attempts to both identify and classify named entities within a given sentence. Specifically for this assignment, the dataset that I used had a total of 14 different classifications, and further differentiated between the start and continuation of different named entities. Although I faced many challenges throughout this project, I do think this was a very valuable learning experience.

The problem I defined for myself is a form of named entity recognition, in which I sought to identify and categorize named entities within a given sentence. To complete this task, I went to "Hugging Face" datasets and found a dataset called MultiCoNER. MultiCoNER is a dataset uploaded by tomaarsen which contains 13 languages of examples, each with over 100k token sequences and named entity tags. For simplicity, I only used the english portion of this dataset, which contains just over 234k rows total. Of these, just over 15k were used for training, and testing was done on 2k examples for the sake of time. While this is a very robust and complete dataset, having now gone through the process of programming this NLP program it is probably a bit too complex to be used with just a RNN process.

My word embeddings were learned from the training dataset using a library called gensim. This allowed me to learn embeddings using the word2vec ngram method. I choose to use ngrams as I figured it would give me higher performance on a task such as named entity recognition (where a lot of the context comes from surrounding words in the sentence). I ended up using 50 parameters to encode my vectors, with a window size of 4. These parameters were discovered largely by experimentation, and seemed to give me the best results. After getting a library of word embeddings to pull information from, I did some basic processing of the data to turn the given words into word embeddings, and then pad the data to give me a uniform size to pass to my neural network.

My overall algorithm is rather basic. As I mentioned before, I first do general preprocessing, turning the given words into word embedding vectors and padding everything to be a consistent size. After that, I use the given tokens and named entity tags to train my model. The model itself is made up of a Recurrent Neural Network layer as well as a Dense layer. The recurrent neural network layer takes in a tensor of form {batch, timestamp, data} as per Tensorflow API, which in my case is the form {sentence, word number, word embeddings}. This layer then using a basic recurrent neural network to output a tensor of form {sentance, unit} where units are specified by the output of the RNN. I then send that tensor through a simple Dense layer which condenses the information down to 50 units (equivalent to the padding used for the named entity recognition tags). I then compare the output of the model with the given tag, using Cosine Similarity to calculate loss. Finally, the model updates itself using RMSprop, the default optimizer that Tensorflow provides and the one that gave me the best performance.

While this is a very basic RNN implementation, there are a few key optimizations that have been included. First, the window and density of the word embeddings were played with to get the best performance possible. Additionally, the inclusion of a Dense vector layer made the model much more efficient.

To put it bluntly, the results of this programming exercise were not the best. As I mentioned before, the task I choose appears to be a bit too complex for a simple RNN. Additionally, my algorithm is effectively trying to predict the entire given sentence at the same time, rather than one word at a time (which would likely product better results). However, when training I was able to get up to almost 20% accuracy, and testing was around 5% accurate. Although this may not seem like a lot, given that this was often attempting to predict a sentence of up to 41 or more words with 14 possible categories per word, it is not a terrible accuracy rating. You can see the training results as exported information during each training epoch, and you can see the testing results at the very end using the evaluate function.

I did two major experiments during the process of creating this program. First, I spent a lot of time attempting to find the most efficient and effective word embeddings. I tried a few different embedding methods, but in particular I played around with size of embeddings and whether pretrained embeddings would perform better overall. It surprised me to find that embeddings of size 50 seemed to be best for this particular dataset, despite seeming a bit small for the sheer amount of data present. Additionally, using 3 negative samples seemed to help my overall performance, especially when evaluating at the end. I also tried using pre trained embeddings, but that actually decreased my overall performance. Secondly, I spent a lot of time experimenting with different loss functions and optimizations. Although Tensorflow has a decent variety of options to choose from, only a few worked at all with my data. I settled on Cosine Similarity for my loss function (again, realizing around this point that the program was predicting entire sentences at once instead of word by word). I experimented with a few of the optimizers, but a lack of knowledge combined with a lack of great documentation meant it was difficult for me to find a good one, so I kept using the given version.

There were a lot of lessons learned during this project, so I'm going to divide this into two sections: personal lessons and technical lessons. For the personal lessons, first I simply underestimated how long it would take to create this program. Although I knew it would be difficult, I always forget just how time consuming creating a complex AI is. For the final project I will certainly be starting much earlier and spending a lot more time to ensure I am not rushing during the last minute. Another personal lesson is learning just how important it is to plan the AI and dataflow beforehand, rather than rushing into programming. There were many moments in this process where I realized that I had made a logical error earlier in the process, and had to redo large sections of code to get everything working. Finally, I personally learned that I need to stop relying so much on libraries in python. Within this project I occasionally got to the point where I knew what I wanted and how to get there, but was unable to change critical internal facets of the AI system. I think that I honestly know more than I think I do, and need to make sure I give myself the ability to showcase that.

In terms of technical lessons, there were two major lessons learned. First was the importance of creating a clear and concise data flow. One of the major slowdowns of this project was that I had trouble tracking the form of the data and how to access it at any given time. Although this was eventually cleaned up for the most part, it probably cost me 4-5 hours all

things considered and could have been solved early on. The second technical lesson was simply to ensure that the program I plan to make can handle the task required. While named entity recognition is not the most difficult task, a basic RNN like this simply can't handle the complex relationships available.

Overall, as a program I honestly do not feel as though I can call this experience a success. However, as a learning opportunity and a chance to practice my skills, I do think this was a great chance to learn to create AI from scratch and gave me a lot of insight into how I would do other tasks in the future. I hope that for the final I will have the opportunity to really push my limits and successfully create a more complex AI program.