CMPSC 448 FINAL PROJECT

Nicholas Czwartacki, Section 002

Individual Project

Task & dataset & preprocessing:

There were many different classification tasks I could have possibly done here, but in the end, I chose to do a simple binary classification involving classifying whether a sentence/review is positive or negative overall. Not only was this simple to do, but there were datasets that were publically available which could easily be used for solving this problem. I chose to use a dataset available here: IMDB Amazon and Yelp Classification Dataset (kaggle.com) The problem is to classify a review as 1 if it is positive or 0 if it is negative. I chose this as it was a simple set perfect for my simple problem! I also installed keras for this as I knew it would be very useful for deep learning problems.

As for preprocessing, this was a somewhat extensive process. I first simply extracted the data from the txt files provided and separated them into training and testing parts with train_test_split, with a test size of 0.2. I then ran the samples through the keras tokenizer with it set for 5000 words. I also saved a vocabulary size variable to help with the training of the models. I then used kera's padding feature to pad up the data I had, which would be especially useful for things like my CNN algorithm. After that, I just made some small adjustments to my train/test data in order to get it to fully fit with my algorithms.

The implementation & architectures of two deep learning systems:

I chose to solve this problem with both a CNN and an RNN algorithm. I thought these 2 would be the simplest and likely most optimal and computationally inexpensive for the job. I implemented both thanks to the keras library for python as well as tensorflow for some of the machine learning specifics that keras could not cover on its own. I chose to use the Sequential() model provided by keras for both of my algorithms.

For both CNN and RNN, I chose to represent the 1st layer with an Embedding Layer, and the last 2 with ReLU and Sigmoid to get everything to converge to a binary result. The main difference between my 2 algorithms is the middle layers. For CNN, I made a Conv1D CNN layer, which is basically a convolutional NN optimized to work with things like arrays or similar 1D related stuff. While I then put a Pooling function as well right

after that layer to further optimize everything. For the RNN I used kera's simpleRNN feature to create 1 RNN layer and send the results straight to the later sigmoid/relu layers for a result to be processed. These were relatively simple and easy algorithms that could get the job done and hopefully converge to an accurate result. I then finished by compiling both neural networks before training them on my data. Overall, my architecture was straightforward, and helped make for decent results.

The training details of two deep learning systems:

To train these algorithms, I used a common and simple trick of the train_test_split function, which divided the data into a training and testing section, as well as input and output values. I converted these train_test_split sets to numpy arrays after the preprocessing and plugged them into the keras sequential fit function to train my CNN and RNN algorithms on. It took quite a while but I got a result in the end. I also had to make sure for my y datasets they were integers of 0 or 1 instead of strings of '0' and '1' finding that bug was annoying but it was an easy fix.

The results & observations & conclusions:

As for the results, it was clear my CNN algorithm did significantly better than my RNN algorithm in getting results. As visible in code files, my CNN was able to generate something which performed far better both on the training and validation data compared to my RNN, having far higher accuracies. I attribute this primarily to certain advantages of CNN networks when coded correctly to work with these sorts of problems, as well as my own issues in making a proper RNN algorithm. I also noticed that for my RNN that it seemed to struggle with getting more accurate as compared to my CNN network when looking at the graphs generated within my code. I felt if I coded my RNN algorithm a lot better, and fixed a few more bugs, the RNN would be doing just as well if not better than my CNN algorithm. In conclusion, CNN isn't just for image classification, and using either CNN or RNN correctly even in unusual cases can still make for great results!

The challenges & obstacles you met and your solutions

The challenges I met were primarily working with keras and fixing/working out all of the bugs in my code that made my algorithms weaker. I still feel I could have done a lot more to make my results better, but with enough grit and determination I was able to work through most of these issues and deliver a decent result. Debugging was really my largest obstacle other than learning everything about keras and how to correctly preprocess the data I had. Other challenges included also modifying certain parts of the

keras sequential() function in order to get an RNN to work, which I had limited success in unfortunately. Other than that. That would be all for problems and solutions,