EE4483 Mini Project Report

Option 1: Sentiment Analysis

Name: Ryan Tan Zhiyang (U1921205A)

GP05

**a) State your choice of feature format. Describe the data pre-processing procedures to convert the raw data into appropriate feature format for input to the model. (10% marks)**

The feature format used for this mini project was word embeddings. In my model, I did not use pre-trained word embeddings. Instead, I tried to learn word embeddings from scratch by starting with random word vectors and progressively learning more meaningful ones. This is don’t by using tensorflow.keras by implementing an embedding layer in my Neural Network stack. Starting with raw data from train.json file, the data had to be cleaned first. (The test data was also cleaned in the same way to ensure consistency) To do this, punctuations, stop words (e.g I, are etc.), symbols (remove emojis) were removed. Letters were also converted to all lowercase letters. This was done by the following codes below.

Text

Description automatically generated

Code 1: Removing unnecessary items

Graphical user interface, text

Description automatically generated

Code 2: changing the data into a list

Graphical user interface, text, application

Description automatically generated

Code 3: removing punctuations and changing uppercase to lowercase

Graphical user interface, text, application

Description automatically generated

Code 4: Detokenizing data and putting it back into sentences

The purpose of detokenizing was to combine the individual words back into sentences as I was not going to use the bag of words pretrained word embeddings method.

The data was then transformed from text data into numeric tensors. This was done as the deep-learning model do not take in raw text but only work with numeric tensors. Therefore, since I was using a deep-learning model, the sequences of words were transformed into low-dimensional floating-point vectors. This was done through the codes below.

A picture containing application

Description automatically generated

Code 5: transforming words and labels

As can be seen above, the data was transformed into numeric tensors and padded with zeros to fit the max length to ensure that all the sequences are the same length. The labels of the dataset (1 , 0) were transformed into a one hot vector in order to enter the same vector dimensions.

Also, the training data was split into training and validation data through the code below. This was done so as to be able to fine tune some parameters and to see if the model is overfitting which is common in Deep Neural Networks.

Graphical user interface, Word

Description automatically generated with medium confidence

Code 6: partitioning data

As can be seen, the data was partitioned into training and “test” (validation) data. The validation data was 5% of the whole training data. This was done by fine tuning. I started with 0.33 or 1/3 of the data. However, the predictions were not optimal. Thus, I reduced the validation data and increased the training data in hopes that the model will be exposed to more data for training and better accuracy in prediction.

**b) Select at least one appropriate model (e.g., RNN + linear layer, etc.) to build your classifier. Clearly describe the model you use, including model architecture figure, the input and output dimensions, structure of the model, loss function(s), training strategy, etc. Include your code and instructions on how to run the code if you are solving the problem by programming. If non-deterministic method is used, ensure reproducibility of your results by averaging the result over multiple runs, cross-validation, fixing random seed, etc. (20% marks)**

The model used was RNN with bidirectional LTSM layer. The model can be seen in the code below.

The model I deployed was a Recurrent Neural Network with Bidirectional LSTM layers. LSTM layer was added as RNNs have a downside that they cannot process long sequences. Thus, LSTM (Long Short-Term Memory) layer is added to hold the memory. The model consisted of

1. Word embedding layer
2. Bidirectional LSTM layers
3. Dense output layers

This model was used as I tested other models such as simple RNN, single LSTM layer and CNN models and this model had the best accuracy out of all the models tested. This may be due to the bidirectional model’s functions. The model can catch more complex patterns as one of the layers process the sequences in chronological order while the other processes the sequence in an anti-chronological order. As for the CNN, the model was able to train significantly faster, however the accuracy of the model was not optimal.

Scatter chart

Description automatically generated

Code 7: Model architecture

Since the model was to predict two labels 1 and 0, I decided to use the sigmoid activation function as well as binary cross entropy as these parameters is optimal for predicting binary labels.

The strategy of training was by comparing results of validation graphs. The model is first to be run, then the code to plot graphs were run right after to observe the trends. The code for the graph plots is seen below.

Text, letter

Description automatically generated

Code 8: graph plot

The first graph was Loss to Epoch and the second graph was Accuracy to Epoch. From the first graph, overfitting can be observed by the separation of the validation data and training data. 100 epochs were tried first and was found that at around the 55-epoch mark, the model started to severely overfit quickly. Thus, epochs were set to 55 and other hyper parameters were tuned. Another parameter added was learning rate and regularization method of dropout so as to further simplify the model and to limit model expressiveness to fit my dataset which might not be rich enough for a complex model.

**c) Discuss how you consider and determine the parameters (e.g., learning rate, etc.) / settings of your model as well as your reasons of doing so. (10% marks)**

The parameters were determining by firstly setting the epochs at a high value. From there, we can see where the model overfits and then adjust the epochs. Once the epochs are set, the learning rate was then altered to produce a better loss to epoch graph. The graphs below are derived from code 7 above.

Chart, line chart, histogram

Description automatically generated

Graph plot 1: validation graphs

At first, I started the model at 20 input dimensions and 20 LSTM layers. However, the graphs showed overfitting and thus I had to reduce the complexity of the model by first reducing the number of layers.

As seen from the validation graphs above, the model seems to be overfitting from the loss to epoch graph. Thus, finetuning of the hyper-parameters was done.

A picture containing text

Description automatically generated

Code 9: new model parameters

As seen from the above code comparing to code 7 in the previous question, I tuned the hyper parameters of learning rate and dropout. Since the model was too complex, I decreased the learning rate from 0.0006 to 0.0004. and the dropout from 0.8 to 0.9. This tuning had a slight improvement of the results.

Chart, line chart, histogram

Description automatically generated

Graph plot 2: validation graphs

As can be seen above, the graphs are smoother. However, improvements could have been made. A decaying learning rate could be added ad from the plot, it starts to plateau at a point. From that point, it may have been stuck at a local minimum and learning rate could be decayed to solve the problem. The reason why I stayed with this model tuning was that it produced the best results so far as seen from the heatmap below.

A picture containing application

Description automatically generated

Code 10: Heatmap

From this heatmap the model predicted true positives (positive reviews as positive) for 97% while the negative reviews were only 68%. I tried many tunings of the model such as further simplifying the model, partitioning the training data even more. However, I realised that the reason for this suboptimal prediction of negative reviews might be due to the model not being trained with enough negative reviews. Thus, I took a deeper look into the dataset.

Graphical user interface, text, application

Description automatically generated

Code 11: training data by labels

From the above figures, it could be seen that the number of positive sentiments in the dataset was nearly 6 times the number of negative reviews. This causes a problem as I am using self-learning word embeddings. Thus, I theorize that there is simply too little negative data for the model to learn, thus only able to predict negative data 68% accuracy.

**e) Analyse some correctly and incorrectly classified (if any) samples in the test set. Select 1-2 cases to discuss the strength and weakness of the model. (10% marks)**

As mentioned above, due to data limitations of negative reviews, the model deployed was only able to predict negative reviews with an accuracy of 68%. I will be giving an example of a correctly predicted and a incorrectly predicted negative review.

Firstly, this sentence was correctly classified as a negative review.

*“Be warned that the business card side is deep enough to hold 50 cards. I bought this wallet only to hold a few credit cards, my ID and some cash. Hence, there is a lot of wasted space in my wallet. From the picture, it looks as though the business card side is just a slot. Whereas, in reality, it is actually a pouch (like most of the other card holders). Other drawbacks are that there is some filler underneath the leather to make it look smoother (which takes up quite a bit of space), the leather itself is very low quality, and the wallet does not close well even without anything in it. Furthermore, on the inside of the wallet there is a large white tag stating that it was Made in China. I mean, China is great and all but if you're on a date and you say \"Hey, let me pay for this,\" and you pull out your credit card and a large white tag protrudes with it saying Made in China, well, you are pretty much a schmuck (I cut mine out with a pair of scissors, but there are still some remnants). So if you like being low class, by all means, buy this wallet. I personally don't care about my wallet very much so I am keeping it and going to use it, but I almost returned it. It is most definitely a solid 2 stars. Hope this helps”*

I assume that this review was classified as negative due to multiple instances of negative tensors.

Secondly this sentence was wrongly classified.

*"I wore this shirt just once. It's almost transparent. Not worth it even if you get it for free"*

In my opinion, this sentence was wrongly classified as the model might have only identified “worth it” and “free” as positive tensors.

In conclusion, I feel that with lesser negative review data, the model is not able to associate some words with others. For example, the wrongly classified negative review. The model might only identify “worth it” and not associating it with “not” in front of the phrase.

**f) Discuss how different choice of feature format may affect the project in terms of resource consumption and accuracy. (10% marks)**

In my opinion, my method of learning word embeddings from scratch was not enough to classify negative sentiments. This might be due to the small amount of negative review data as compared to positive data. However, by using pretrained word embeddings would generate a better prediction. This is due to the model analysing input data with the “knowledge” of how words put together might have a whole different meaning. For example, for TF-IDF can rank the documents relevance to the user’s query. Also, using pretrained word embeddings, might be less computationally expensive as the model would not need to use processing power to learn word’s meanings.

**g) Assuming that in a separate project, you need to perform sentiment classification on hotel reviews. However, the hotel reviews consist only of reviews in raw text and does not come with rating scores. Discuss how your classification algorithm would perform in this new project and how can it be modified to perform well in this new problem. [Hint: the challenge can be address from beyond the model architecture.] (20% marks)**

If given this task, my model will not be able to perform. This is due to the model needing some form of label to learn word embeddings as I did not use pretrained feature formats. I feel that if the dataset has no labels, it would be quite hard to self-learn and classify labels. This is due to complexity of language. For example, sarcasm would be hard to predict as it could be thought of a false positive (sounding like a compliment but is an insult). However, I feel using pre-trained word embeddings would make the job much easier. By using pre-trained word embeddings, the model will be able to associate words together and generate a label.

To continue using self-learning word embeddings, rules must be defined first. I could take some “easy” data from the dataset and label it myself as a positive or negative review. The labelled data will then be used to train the model. From there, if the model encounters any “more complicated” data, it can then be fine-tuned.