ECS7001P Neural Networks and Natural Language Processing Project

Matthew Lee
ID:180789269
m.d.lee@se18.qmul.ac.uk

February 20, 2019

1 Parts A-C

The Jupyter Notebooks prefixed with 'NNNLPAssignment1Part' contain both the code and the answers to the assignment questions. Please refer to these notebooks for marking parts A-C. The assignment asks for loss graphs but i had originally created accuracy graphs in the notebooks. Rerunning the notebooks to create loss graphs is unfeasble, as some of the models take an extreme amount of time to run. Therefore I have inserted them here. These are relevant for Part B Q1-4: Figures 1, 2,3, 4 and 5.

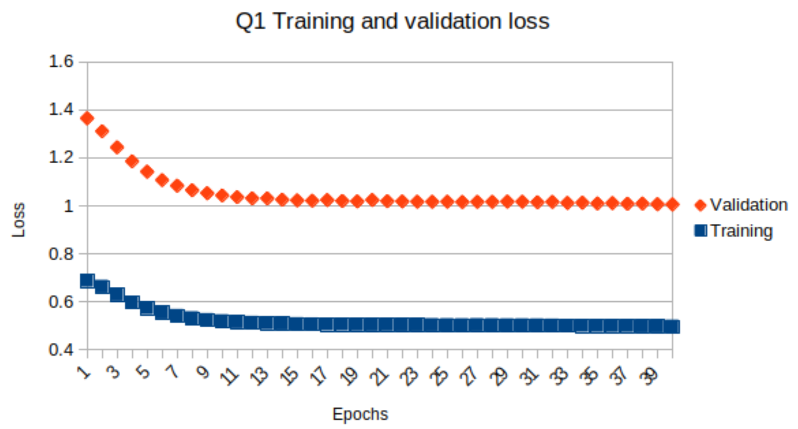

Figure 1: Part B Q1: Loss for Onehot encoded model

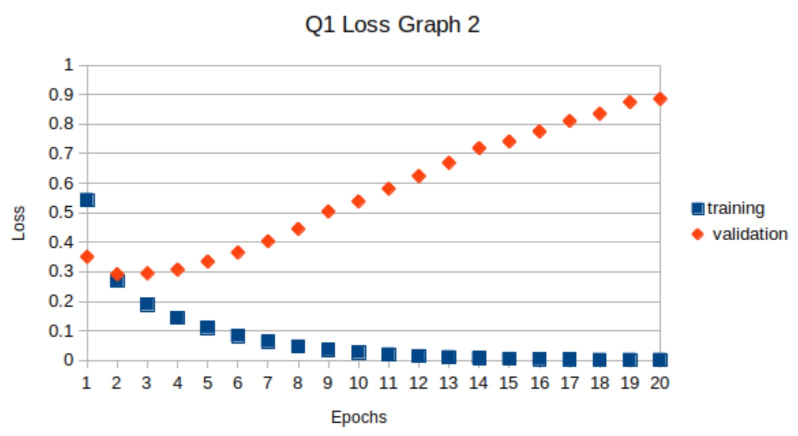

Figure 2: Part B Q1: Loss for Word embeddings model

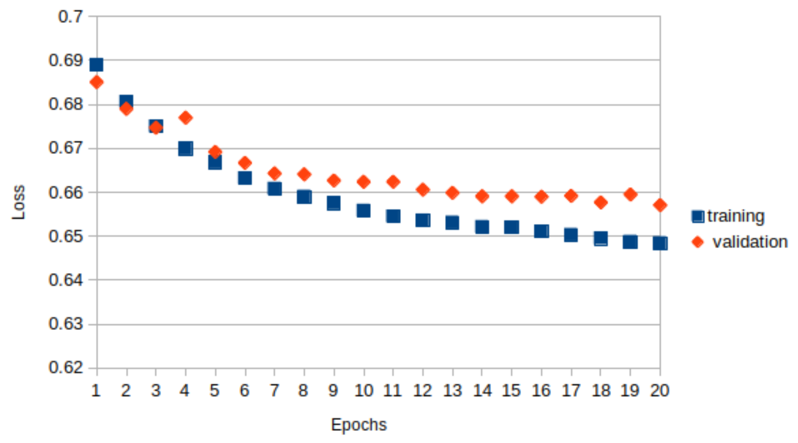

Figure 3: Part B Q2: Loss for Pre-trained Word embeddings model

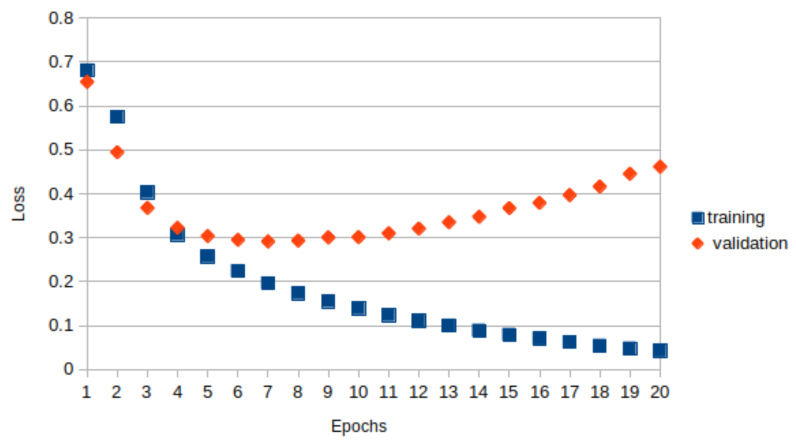

Figure 4: Part B Q3: Loss for Pre-trained Word embeddings model that has unfrozen the embedding layer training

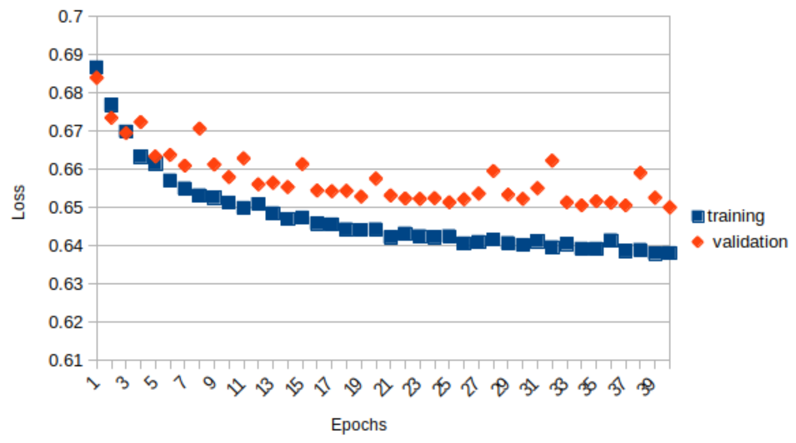

Figure 5: Part B Q4: Loss for for model with additional dense layer

2 Part D

The code for Part D has been split up into steps 1-4. The notebooks prefixed with 'TaskDPt' contain code and comments for these sections. Step 4 extends the models in step 3, therefore both step 3 and 4 can be found in the 'TaskDPt3-4' notebook. Discussion of the experimental process and detailed evaluation of the models is conducted within this document.

2.1 Step 1: Positive, Neutral or Negative Tweet Sentiment Analysis

In this step the challenge is to create a classifier to identify whether a tweet is positive, neutral or negative. These challenge is based on the SemEval 2017 competition Subtask A and the SemEval 2017 training Subtask A English dataset.

2.1.1 Preprocessing

I started by preprocessing the data. Many of the files had been through some encoding process that had changed the punctuation to the unicode but had not converted it back to text. The pre-processing first converted these back into punctuation. Now the punctuation had been correctly decoded, I was then able to apply contraction expansion. This expands contractions e.g. "they're" $\rightarrow$ "they are". The tweets were then split into a list of words using the NLTK word_tokenize function, which also strips punctuation.

Preprocessing then stripped stop word and non-alphanumeric words. The remaining words were all set to lower case. By setting to lower case I ensure that capitalisation doesn't effect the embedding a word is given.

The preprocessed sequence of words was then converted into a sequence of numbers. Each number representing a word that can then be used by the embedding layer of the model.

The data was then split into a training and validation set using train_test_split from the Sklearn module. The split was left at it's default of 25% validation split. The data is shuffled and stratify is set to true which ensures that there is a consistent split between the training and validation sets. This doesn't mean it address the class imbalance, but it does mean the classes will be imbalanced in the same way in the validation set as in the training set.

The labels were then one hot encoded as this is required for categorical data when using the models I have chosen.

2.1.2 Model Creation

I chose to use an LSTM (Long Short Term Memory) model for this task. I chose it because it had performed well in parts A-C of the assignment and LSTMs are generally strong for text classification tasks. An LSTM is able to use the position of the words as well as their meaning to derive the sentiment of the sentence, which is a key advantage over many other types of model. The tweets are not long sequences, therefore the LSTM will not be required to store information from many timesteps ago, which can be a problem for LSTMs classifying larger documents such as books.

LSTMs require word embeddings rather than the words themselves as inputs. For this task I chose to train my own embeddings, to keep the approach simple. 100 dimension embeddings were trained for a vocabulary size of 60000. 100 LSTM neurons were used, again because of the success of this architecture in previous tasks.

As the validation accuracy varies as the model is trained for additional epochs, a checkpointer is created. This checks the validation accuracy after each epoch and saves the current model weights if the model has had the highest validation accuracy so far. This means the weights from the best epoch can be used rather than the weights from the last epoch.

2.1.3 Model Evaluation

The average f1score and accuracy of the model was calculated as well as precision and recall for each class. Which of these metrics is most important depends on the intended application. The model achieved a maximum validation accuracy of 64.8%, which is a reasonable score for a basic model given many other models achieved performance below 70% accuracy as well [1].

The model achieves a test accuracy of 60.3% and average test fscore of 0.568 This is significantly lower than the validation accuracy (64.8%) and training accuracy (92.9%). The large difference between training accuracies and the other accuracies indicates the model is overfitting. Dropout layers were tried but had no effect on the overfitting. Future work could try reducing the overfitting by trying more aggressive dropout layers, reducing the number of neurons or reducing the size of the embedding vectors.

There is also a notable class imbalance in both the training and test datasets. This was not addressed in this basic model but could also be the focus of future improvements.

2.2 Step 2: Positive, Neutral and Negative Classification of Arabic tweets

In this step the challenge is to create a classifier to identify whether an Arabic tweet is positive, neutral or negative. These challenge is based on the SemEval 2017 competition Subtask A and the SemEval 2017 training Subtask A Arabic dataset.

2.2.1 Preprocessing

The preprocessing for the Arabic data followed a pattern similar to section 2.1.1. One additional step was to handle emojis. This dataset contained emojis in their unicode format. These are a potentially very strong indicators of the sentiment of a tweet. The emoji python module was used to convert the unicode into a word e.g. The emoji for a face with tears of joy is converted to: ":face_with_tears_of_joy". The emojis are therefore converted to words that should give strong indications of the sentiment of the tweets.

Contractions were not expanded as I was not confident of the affect that would have on Arabic tweets. Non-alphanumeric words were not stripped as Arabic uses a different script.

The maximum length of the sequences was also padded to 150 words. This is because the Arabic tweets had more words in its sequences, because the Arabic data included emojis which only count as one character in a tweet, meaning a maximum of 140 words is possible if someone tweets pure emojis.

The vocab size was considerably smaller with only 30000 words instead of 60000, this is probably due to the smaller dataset.

2.2.2 Model Creation

The same model architecture was used as in step 1, see section 2.1.2. The model performed well there so it is reasonable to expect it to have similar performance in step 2. Word embeddings were trained as part of the model and a 100 neuron LSTM layer was used to make the sentiment prediction.

2.2.3 Model Evaluation

The model achieved a validation accuracy of 54.7% and test accuracy of 46.8% with a average test fscore of 0.433. We can see from the confusion matrix that the classifier particularly struggles to classify when the text is neutral, its recall of positive samples is 25.6%. Therefore to help improve this model we could try concentrating on features that help to classify positive text. The fact that the model is worse than guessing when the data is positive is concerning and suggests there is a lot of room for improvement. We would expect that the neutral class is the hardest to classify though as it's most similar to the two other classes. Both the training and test data have a class imbalance. Future work could try addressing this class imbalance. I think large difference between the neutral and the positive recalls shows the model struggles to distinguish between them and has placed its decision boundary to get more of the neutral cases right. The positive class is the smallest in both the test and the training data, which may explain why the model has done this. If we wanted to identify neutral tweets then this model would be very useful but most applications would want to classify the negative or the positive tweets, so the future work may want to incentivise the model to improve recall on negative and positive classes, either by balancing the data or creating a custom loss function.

2.3 Step 3: Five Point Target Sentiment Analysis of English Tweets

In this step the training tweets have been labelled with a 5 point scale (strongly negative = -2, negative = -1, neutral = 0, positive = 1, strongly positive = 2) for sentiment about a particular target. The tweets have also been labelled with the target topic.

2.3.1 Preprocessing

Preprocessing used the same approach as in section 2.1.1 with one additional step to find the target within the string and replaced it with a "<TARGETTOKEN>" token. This is a straightforward way to ensure the model knows what the target of the sentiment analysis is, however it comes at the cost of losing any information contained within the target word itself. E.g. tweets about Microsoft might generally be negative because it's used as a method to complain to customer service, but the preprocessing would prevent the model from utilising this feature. This replacement was performed before splitting the tweet into a list of words as the topics can be more than one word, which could make detection in the word list harder.

2.3.2 Model Creation

The same model was used in as in steps 1 and 2, a 100 neuron LSTM with an embeddings layer that will learn embeddings as the model is trained, see section 2.1.2. This model was used because of its success in previous steps.

2.3.3 Model Evaluation

The model achieved a validation accuracy of 55.2% and a test accuracy of 51.8% with an average test fscore of 0.220. This is worse than step 1, despite there being considerably overlap between the step 1 and step 3 datasets, however the sentiment analysis for step 3 has been split into 5 rather than 3 classes so we expect the accuracies to be lower as the problem is harder. The model is having to make distinctions between more nuanced sentiment. The model is also being asked to identify the sentiment towards a particular entity rather than a general positive or negative sentiment, which adds additional challenge.

The test fscore is significantly lower than that of step 1 (0.568), this is partially an artefact of their being more classes, making direct comparison difficult, but it also shows that the step 3 model is very poor at classifying some tweets. The recalls and precisions show this, see table 1.

Table 1: Step 3 Recalls and Precisions

Class	Recall	Precision
Strongly Negative	0.00%	0.00%
Negative	2.54%	53.9%
Neutral	89.4%	52.5%
Positive	33.6%	46.8%
Strongly Positive	0.00%	0.00%

It essentially ignores the strongly positive and strongly negative classes and simply tries to classify everything into negative, neutral or positive classes. This is probably because of the large class imbalance; the model is able to ignore the extreme classes because there are so few cases of them in the training, validation and test datasets. It instead concentrates on classifying the majority of tweets which fall under negative, neutral or positive. This explains why the fscore is so much lower than in step 1.

2.4 Step 4

2.4.1 Preprocessing

The only change experimented with for the general preprocessing was stemming which takes conjugated words and strips them back to their stem. E.g. "Running" → "Run". This actually reduced the validation accuracy so was not included in the final model. I believe the reduction in validation accuracy could be because the stemming removes many of the grammatical features from the tweets which help to indicate what the target of the sentiment is. The classic example of this, loss of grammar effect, can be seen with the examples: "Let's eat, Grandma!" vs "Let's eat Grandma!". Future work could investgiat ewhether removing punctuation is also having a negative effect on accuracy. Another reason stemming may reduce accuracy is because the stemmer used, nltk's porter stemmer, is vulnerable to spelling mistakes and names, which are both common in tweets. E.g. The porter stemmer converts the name "Dorothy" to "Dorothi" and may incorrectly stem other misspelt words.

Some model specific preprocessing was required for the new models introduced in step 4. In particular the Neural Bag Of Words (NBOW) model required each tweet to have the word embeddings for every word within it calculated and then an average of them calculated. Spacy provides 300 dimension word embeddings. The average of these embeddings was calculated before padding the tweets, to avoid diluting the average vector with the padding vectors.

2.4.2 Model Creation: Spacy Embedding Neural Bag of Words

For this model I experimented using Spacy embeddings rather than training them myself. Attempts to create a model that use the individual word embeddings and an LSTM or CNN were unsuccessful as the resulting data structures were too large for my laptop's memory. To reduce the memory requirement I used a Neural Bag Of Words (NBOW) model.

This model takes in the average word embedding for each tweet and passes it into a neural network. This essentially turns the problem into a straightforward machine learning problem where the network is simply learning to predict a class given a 300 dimension vector. It should be noted that this model ignores the ordering of the words and also ignores the target token. It is therefore simply classifying based on the vocabulary used in the tweet. We would therefore expect the model to be at a significant disadvantage, but the results were actually the strongest. A better machine might have been able to use the embeddings in a CNN or LSTM without averaging them, thus preserving the sequence and target information, which i expect would yield even better results.

Multiple architectures were tried for the model, starting with a simple model, which I found underfit. I therefore deepened and widened the model until overfitting became more of a problem than underfitting. At this point I added dropout layers to reduce the overfitting. Once I had done this further changes were not able to improve the validation accuracy so I settled on the final architecture of: 64-dropout-128-dropout-128-dropout-64-5. Each dropout set to

20% probability and each layer using a relu activation, except for the last which used softmax for multiclass classification. This approach is essentially a crude neural architecture search.

I tried using a linear activation function in the final layer, this was to capture the fact that the strongly negative class is closer to the negative than to the strongly positive. This was combined with the mean squared error loss function. Doing this did not improve accuracy and so I returned to using a classifier. Future work may want to re-investigate changing the loss calculation to capture the relationships between classes.

2.4.3 Model Evaluation: Spacy Embedding Neural Bag of Words

This model achieved a validation accuracy of 58.4% which is the better than the original model (55.2%). It achieves an accuracy of 59.0% on the test data which is significantly better than the original model (51.8%). The test average fscore is also significantly better than the original model at 0.33 to the original's 0.22. This is the strongest of all models I have tried for this task.

It is notable that the model performs similarly on the test data to the validation accuracy, where we would normally expect test accuracy to be significantly lower. I believe this is because the Spacy vectors have been trained on a larger corpus, which will allow it to generalise better to unseen data.

This model also outperformed the LSTM in training time, taken seconds per epoch rather than minutes. This might be a significant advantage if deploying to real world applications.

2.4.4 Model Creation: Convolutional Neural Network with Embeddings from Step 3

For this model I tried using a Convolutional Neural Network (CNN). I would have liked to use the spacy embeddings for this but as mentioned in section 2.4.2, memory errors prevented me. Instead I applied the CNN to embeddings that had been learnt by the original model in step 3, to see if the limitation of the model was the structure of the model. I also hoped this would accelerate the learning of this model. I experimented with "freezing" this embedding layer by making it untrainable. I wanted to see if this would prevent overfitting by limiting the number of trainable parameters. By freezing the embedding layer I reduce the number of trainable parameters by 6 million to just 160 000.

The CNN is able to use the structure of the sentence to assist in its classification, so had potential to outperform the NBOW.

Again, multiple architectures were tried and in some architectures the embedding layer was unfrozen. Most architectures performed similarly so I chose simplest of the better performing models.

2.4.5 Model Evaluation: Convolutional Neural Network with Embeddings from Step 3

This model achieved a validation accuracy of 55.6% which is not a significant improvement over the original model and worse that the NBOW model. As it did not perform better than the NBOW on validation I did not apply it to the test data. All the CNN architectures I tried seemed to suffer from overfitting, despite adding dropout layers, reducing width and depth of the network , freezing the embedding layer or even training the embeddings from a clean initialisation.

2.4.6 Model Creation: Bi-Directional Long Short Term Memory

For this model I used a Bi-Directional Long Short Term Memory (BiLSTM). This applies an LSTM to the sequence and also to the sequence in reverse. This is an extension of the LSTM and so I hoped it would improve on the original model.

2.4.7 Model Evaluation: Bi-Directional Long Short Term Memory

The model had a validation accuracy of 56.5% which is only a small improvement over the original model. It achieved a test accuracy of 51.7% which is very similar to the original model. The similar validation accuracies between the original, CNN and BiLSTM models may indicate that there is a limited amount that can be done by training 100 dimension word embeddings on this dataset. It may be the case that to make significant improvements to the model requires additional data or embeddings that have been trained on a larger corpus. Future work could investigate this by trying other word embeddings and other datasets.

2.5 Step 4: Conclusion

Several models types were tried for step 4 and several architectures of those model types were also tried. The Spacy Embeddings combined with NBOW model had the best performance on the test data with an average fscore of 0.33 and an accuracy of 59.0%. The model could be adapted slightly and then be applied to general sentiment analysis from step 1 as it doesn't need the target label to identify the sentiment. The NBOW model's performance is surprising given it ignores the ordering of the words and I expect that the main advantage is not from the NBOW structure but from the Spacy embeddings. Future work should examine Spacy embeddings combined with a model that uses sequence rather than averaging the word embeddings.

References

[1] David Zimbra, Ahmed Abbasi, Daniel Zeng, and Hsinchun Chen. The state-of-the-art in twitter sentiment analysis: A review and benchmark evaluation.

ACM Transactions on Management Information Systems (TMIS), 9(2):5, 2018.