TWITTER SENTIMENT ANALYSIS
APPLIED MACHINE LEARNING SYSTEMS II (ELEC0135) 19/20 REPORT

SN: 18154195

ABSTRACT

In this paper, we report our attempt to solve SemEval 2017 Competition Task 4: Twitter Sentiment Analysis. Based on recurrent neural network (RNN), precisely long short-term memory (LSTM), our approach focused on the first two sub-tasks within the competition, namely 3-point scale tweet classification and topic-based tweet sentiment classification. We picked a series of text preprocessing pipeline and pre-trained word embedding that adhere to the use of informal language tone within the Twitter environment, with the possible presence of misspelt and elongated words, as well as the use of emoticons. Although we were not officially participating, our system would have been placed 7th and 15th in the competition for the two sub-tasks, with reported average recall of 0.659 and 0.782 for task A and B, respectively.

Index Terms— Twitter sentiment, LSTM, SemEval 2017, Glove, FastText

1. INTRODUCTION

Sentiment analysis is aimed to extract the polarity of the opinion of a particular topic. There are many promising applications of sentiment analysis in general, one of them is in the area of politics, that is determining the media's point of view towards political candidates to avoid political bias [1]. Another example of sentiment analysis, particularly in the Twitter platform, is audience insight. Companies may use sentiment analysis as a tool to gain customer insights to understand how their consumers interact with products or services and their sentiment associated with the product [2].

Technically, conducting sentiment analysis on Twitter domain has its distinctive challenges. Firstly, the language style within Twitter is varied. However, just like any other social media, the tonality of the text is usually casual, with the possible presence of improper spelling, grammar mistakes, repetitive use of punctuation, and the use of emoticons to convey the message. Nevertheless, unlike other typical social media, Twitter has a strict character limit per *tweet* (post), that is 140 characters.

The source code of the project is available as a GitHub project at https://github.com/donyeun/AMLSII_19-20_SN18154195

In this report, we present our attempt on solving SemEval-2017 Task 4 challenge [3] on conducting sentiment analysis in the Twitter dataset. We focused the discussion on the first two out of four challenges available in this competition. The first task is a 3-point sentiment tweet classification challenge, where a system has to classify the sentiment of a tweet within three possible categories: positive, neutral, or negative. The second task deals with topic-based sentiment classification, where an observation includes not only a tweet, but also a human-annotated topic attached to it, and the system should determine the polarity of the sentiment towards the topic. The class label for the second task can either be positive or negative.

This paper is structured into several sections. In sec. 2, we define several fundamental theories of the system briefly. In sec. 3, we discuss the description of the models that justifies the choice of algorithms. Sec. 4 describes the implementation of the system, followed by sec. 5 listing several interesting findings. Finally, sec. 6 outlines the main conclusions.

2. LITERATURE SURVEY

This section provides a brief description of the sub-tasks given in the SemEval 2017 competition. The summary of the theories that this project based on are also described, such as the selection of pre-trained word embedding vectors, the use of a recurrent neural network (RNN), as well as the explanation of the RNN's extension being used in this project, that is the long short-term memory (LSTM).

2.1. SemEval 2017 Competition Task 4

SemEval competition is an annual event interested in the evaluations of computational semantic analysis systems. In 2017, SemEval conducted a competition on Twitter sentiment analysis [3]. There are four sub-tasks within this challenge, but this project only focused on the attempts to finish task A and B. Task A is a 3-point scale Tweet classification. Given a tweet, the system should define its sentiment on whether it is positive, negative, or neutral where it does not necessarily lean towards the two polarities.

In task B, the dataset is equipped with a topic label annotated by a human. This topic can be of anything, from a named entity such as a person's name, a brand, or an event.

In practice, this topic can be in the form of a word, a phrase, a Twitter username handle, or a Twitter *hashtag*. There are only two degrees of sentiments in this second task: either positive or negative. The implementation of this challenge is that given a pair of a tweet and its tagged topic, we should be able to determine the sentiment label of this datum.

The performance of each task was measured primarily in average recall [3]. This was because unlike accuracy and F1, the average recall is more robust when dealing with the issue of class imbalance [4]. This evaluation metric ranges in [0, 1], where 0.00 depicted all wrong guesses provided by a system, and 1.00 could only be obtained by a perfect classifier that could mimic human raters' intuition. However, another two secondary measurements were also being reported to supplement the main evaluation metric, namely accuracy and macro-average F1 score for both task A and B.

There were 48 competing teams for SemEval task 4, most of teams' approaches were based on deep learning, such as a convolutional neural network (CNN) and long short-term memory (LSTM) [3].

2.2. Word Embedding

Word embedding is the mapping of words in a dense vector space. This new representation of words captures the semantic and syntactic information of the words and can be useful to be used as the input to the classifier algorithm. Word embedding will be the first layer of the entire neural network layers, that maps the raw input sentence into vectors of word embedding. There are several pre-trained word embeddings made from a various algorithms and from various corpus of text. In the following sub-sections, we describe briefly two word embedding vectors being used in this project, namely FastText and Glove.

2.2.1. FastText

FastText is a pre-trained word embedding developed by Facebook AI Research [5]. FastText represents each word as the combinations of n-gram of characters. This way, FastText able to catch the functionality of prefix and suffix within a word. A skip-gram is then trained to learn this n-grams embedding. The nature of FastText to break a word into smaller units of n-grams makes it better to understand the rare words. Rare words might be problematic in pre-trained word embedding. This is because due to technical and practical limitations, a pre-trained word embedding can only contain a finite amount of vocabularies in it. Any tokens or words that are not known by the pre-trained library will be tagged as an unknown token, and it may cause an issue of losing information, especially if that particular rare token conveys important sentiment signal within the text.

2.2.2. GloVe

GloVe is a pre-trained word embedding developed by researchers from Stanford University [6]. Unlike FastText, this distributed word representation depends on the word-occurrence statistics from the learned corpus. The distance between words in the vector space represents the degree of semantic similarity degree between those words in the language model.

It has various versions of training corpus and different dimensions of vectors. It has versions on Wikipedia, Common Crawl, and Twitter. The Twitter pre-trained word representation by itself has several dimension length selections, ranging from 25-dimension, 50-, 100-, and 200-dimension, made up from 2 billions of tweets. This word embedding has 1.2 million unique words within the vocabulary.

2.3. Recurrent Neural Network

From the perspective of input-output mapping, there are several types of RNN:

1. **One-to-many**. This type of RNN only has one input node and has multiple outputs, which is suitable for many computational creativity applications, such as music generation. Given one musical note, the networks will generate many new notes based on that input.
2. **Many-to-one**. This RNN is the one we are using in this report. Given multiple tokens or words, for instance, the networks will give only one output in the end, such as the sentiment label for this matter.
3. **Many-to-many**. This RNN is suitable for applications in the context of natural language processing such as machine translation. Given multiple words within a sentence, the system will also output multiple words, but in a different language.

Just like any other neural network architectures, RNN can also have multiple hidden layers in between the input and output layer, making it a deep RNN.

2.4. LSTM

The long short-term memory, or commonly known as LSTM, is the extension of RNN that is able to deal with the issue of the vanishing gradient problem that a *vanilla* RNN encountered. The algorithm was invented in 1997 [7].

There are some variations of networks that extend the functionality of a traditional LSTM, one of which is bidirectional LSTM. This type of network has two separated networks, covering both forward and backward to get information from the past and future, respectively. LSTM has one weights tensor that is being updated across the sequence of input.

A Twitter post (tweet) will be broken down into tokens. Each token will be translated into an embedding vector, and

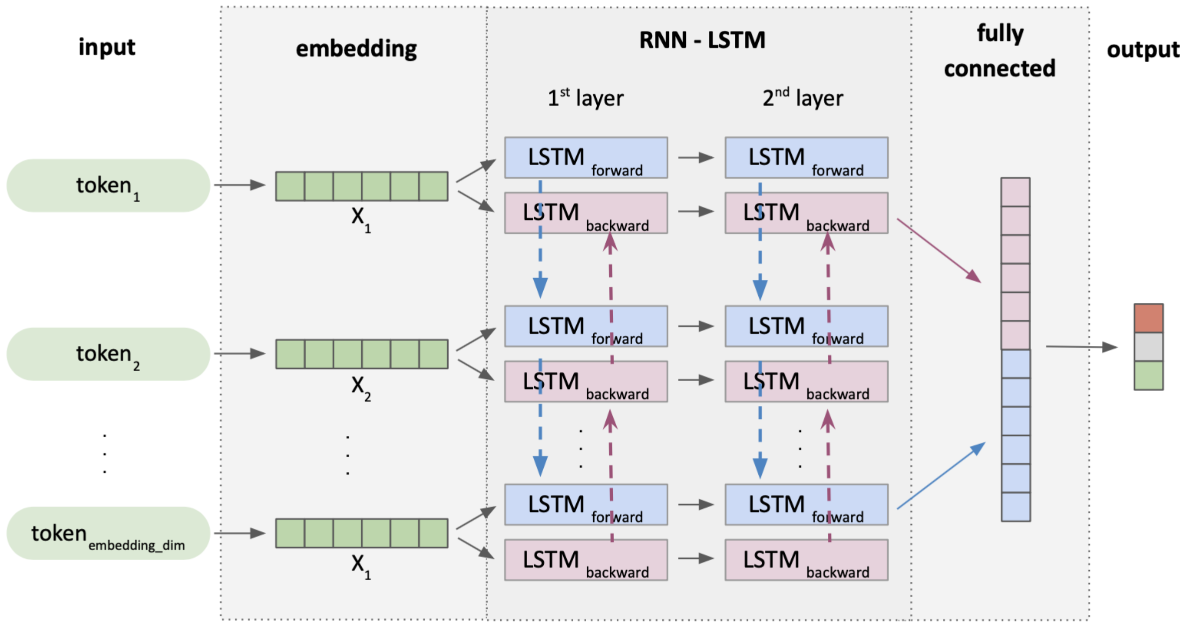

Fig. 1: Network's general architecture

each of them will become an input to the LSTM. For each token, LSTM uses this embedding word vector and the previous hidden state of the network to compute the next hidden state.

3. DESCRIPTION OF MODELS

Figure 1 sums up the architecture of the neural network that we use in this paper. The first layer of this neural network is the embedding layer. It takes each word from a tweet as the input to the system, one token at a time. This first layer is followed by n-layers bidirectional LSTM. After that, a fully connected concatenates the output from both forward and backward LSTM. The output layer serves as the last process within the system and is responsible for determining the sentiment label predicted by the system.

Each layer is described in the following sub-sections below.

3.1. Word Embedding

The initial weights of the embedding layer are equal to the weights of the selected pre-trained word vector embedding. In this case, it can be either the 300-dimension FastText from Crawl corpus or the 200-dimension GloVe trained on the Twitter dataset.

The selection of text preprocessing and the kind of pre-trained embedding to choose becomes a vital part of this project, due to the unique characteristics of the Twitter corpus dataset itself. Any pre-trained word embedding libraries that are trained from general text corpus might not be ideal for this matter, as the tweets' language tone is different from the standard English.

The entire dataset is divided into several batches. The input texts are padded so that they have the same length for each batch. Ideally, each batch should consist of samples that have the same or similar in length. This can be done by sorting the samples within the dataset first before breaking them into smaller batches. This way, the number of padded tokens within each sample of a batch would be minimal as it has similar text length.

This additional padded tokens are added to meet the length criteria. A padded token is marked by a <pad>tag. It is worth noting that the sole purpose of this padding token is so that all samples within a batch have the same length, and padded token should be ignored as it does not give any meaningful information to it. In the implementation, masking is needed for this matter. Masking is the act of telling the networks the actual length of a sample so that the networks will ignore additional padding tokens.

3.2. Bidirectional LSTM

The use of bidirectional LSTM enables the neural network to catch the information from the past by running forward LSTM, as well as from the future by executing the backwards LSTM. We use n-layer LSTM, with additional dropout mechanism to prevent overfitting. During the training process, some hidden or visible nodes will be randomly dropped out. This ensures that the system learns the pattern and does not merely memorise data.

Early stopping mechanism is also being implemented in the system to mitigate overfitting issue. One indication of possible overfitting case is when validation loss as the monitored metric is no longer improving during the training process. By having this information, we immediately stop the learning process once the validation loss worsens.

We use cross-entropy as a choice the loss function. To minimise the training cross-entropy loss score, we use Adam optimiser with an initial learning rate equals to 0.001 [8].

3.3. The last layers

The last two layers are a fully connected layer and the output layer. The fully connected layer's input dimension is twice the size of the hidden dimension of the LSTM, as it carries out both the outputs from the forward and backward of the n-layer network. Lastly, an output layer should have a number of nodes that equals to the number of classes for each task. This last layer is responsible for deciding the sentiment output from the system.

4. IMPLEMENTATION

In this section, we describe the implementation of the tasks, starting from the description of datasets, followed by a description of text preprocessing pipeline, as well as the hyper-parameter tuning.

4.1. Dataset

The dataset provided by SemEval are the training, validation and testing set from the previous years' competitions. There are two additional text files for task A that can be used; those are the sentiment analysis dataset from LiveJournal social networking website and an SMS dataset. We decided to use these two additional datasets, with speculation that these two supplementary datasets share roughly similar linguistics style as well as has common vocabulary together. These datasets are separated into several CSV files, and we combine them as one pristine bundle of the dataset within one single CSV file for each training and testing set by using Pandas [9].

In task A, that is the 3-point scale tweet classification; we use the list of tokens of a tweet as the input. In task B, however, which is topic-based tweet classification, we have more information within the dataset, that is the tagged topic of the tweet. This annotated topic ranged from name entities, hashtags, Twitter handles, or a phrase. It is worth noting that the topic word or phrase itself may or may not be written in the actual tweet. In task B, we append the topic into the tweet itself.

Fig. 2 describes the tweet distribution for each class. It is apparent that there exists the issue of class imbalance within both training and testing datasets, where the neutral class dominates the proportion of the tweets. While task A has a

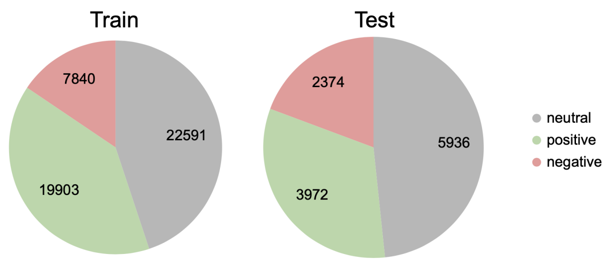

Fig. 2: Task A dataset proportion per label

consistent class share between training and testing dataset, task B does not have this property.

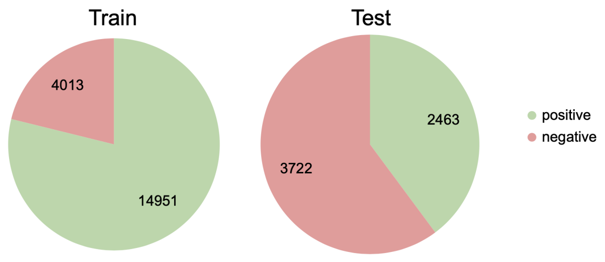

Fig. 3: Task B dataset proportion per label. There is a class proportion difference between training and testing dataset

From Fig. 3, it is apparent that the class domination between train and test set are different. The proportion of tweet with positive sentiments, marked by green area within the pie chart, dominating the samples within the train set. In the test set, on the other hand, the number of negative tweets outnumbers the positive ones.

4.2. Text Preprocessing

We used a text preprocessing library named Ekphrasis [10], made by one of the competing team in this SemEval competition. This text processing library was made for the linguistics style usually written on Twitter. The library also has the capability to understand how to parse some important Twitter jargons such as @username and #hashtag, as well as tagging the presence of a URL link.

Some text cleaning steps that can be used within this library are translating emoticons (e.g. ":)" into "<happy>"), normalising elongated word (e.g. "goooood" into "good <elongated>") and tagging punctuation repetition.

The preprocessing steps started from tokenisation, followed by word normalisation as mentioned previously, and finished by case-folding where all the characters are being set to lowercase for uniformity. Table 1 shows the result comparison between the raw and processed sample texts.

Table 1: Pairs of raw and processed text

Raw Text	Processed Text
"You know it's been a great weekend when you spend $50 in dunkin donuts on fri, sat, and sunday."	['"', 'you', 'know', 'it', "'", 's', 'been', 'a', 'great', 'weekend', 'when', 'you', 'spend', '<money>', 'in', 'dunkin', 'donuts', 'on', 'fri', ',', 'sat', ',', 'and', 'sunday', '.', '"']
Now playing the debut album by @FlagshipOffcl - it's out on Monday and the band play The Horn on Tuesday! :-) http://t.co/n0zcm3ML4p	['now', 'playing', 'the', 'debut', 'album', 'by', '<user>', '-', 'it', "'", 's', 'out', 'on', 'monday', 'and', 'the', 'band', 'play', 'the', 'horn', 'on', 'tuesday', '!', '<happy>', '<url>']

4.3. Hyperparameters Tuning

We did not use the entire training dataset. Instead, we set aside some portion of it and made it as a validation set. The 80% of the training data were used as training set, while the rest 20% of it were used as validation data. The validation set is useful to evaluate the performance during the training process and act as a signal indicating when to stop learning. During the hyperparameter tuning, we determine the best model to be the one that has the smallest loss value when evaluated using this validation set.

During system tuning, we sought the best hyperparameter for the neural network, as well as the best combination of word embedding settings that resulted in the lowest score returned from the loss function. For the hyperparameter selections, we can tune the system with several variations of dropout, batch size, the number of hidden layers for the LSTM, as well as the hidden nodes per layers.

During training, we tried to minimise the training loss function. We did early stopping the training process if the validation loss during the training process did not decrease. Early stopping acted as regularisation, and it reduces the chance that the system suffers from overfitting.

The architecture of the system, including its optimiser, definition of loss function and all the things related to the neural network were being made with PyTorch [11]. Other contributing libraries were (but not limited to) NumPy [12] and Scikit-learn [13]. We leveraged the service provided by Google Colaboratory in order to gain more computing powers needed to run the network architecture [14].

5. EXPERIMENTAL RESULTS AND ANALYSIS

The reported average recall as the primary evaluation metric is listed in Table 2. This table also provides secondary measurements being used in the official competition, namely macro-averaged F1 score as well as accuracy. As seen from the table, our system has an average recall of 0.659, which put us in the 7th place had we officially register in the competition. Task B reported a 0.782 average recall, which put us in 15th position within the list.

Table 2: The performance result from the test set

	Task A	Task B
Avg Recall	0.659 (7th)	0.782 (15th)
F1 Score	0.633 (8th)	0.743 (18th)
Accuracy	0.658 (4th)	0.767 (15th)

Some key findings of each task are described in the following sub-sections.

5.1. Task A: 3-point scale Tweet Classification

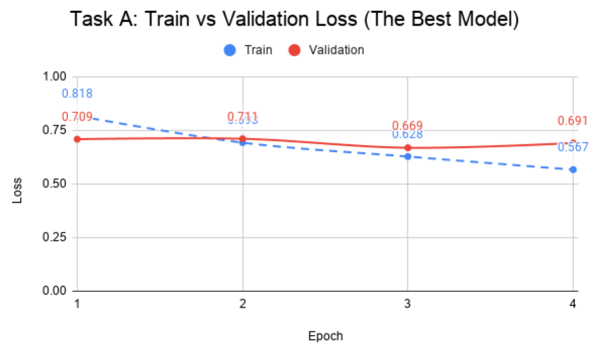

Fig. 4: Loss function score for task A

Fig. 4 shows the training loss minimisation process during the training phase. The dotted blue line depicted training loss that the model tried to minimise, whereas the red line corresponds to validation loss. As mentioned, validation loss is a tool we used in order to prematurely stop the learning process before the designated number of epochs ended.

As can be seen from the plot, the X-axis that corresponds to the number of epochs stops at the fourth epoch. In fact, we stopped the learning process after the third epoch as the validation loss was rather increasing. As we did not want to suffer the model from being overfitting, we immediately halted the training phase and saved the model from the third epoch as the best model.

It is worth noting that the graph depicted a training phase with a combination of neural network's hyperparameters as well as word embedding settings that resulted in the best performance during hyperparameter tuning. This highest performance corresponds to the model that had the lowest final validation loss. The best hyperparameters selected from the tuning process are listed in table 3.

Table 3: The best parameter of Task A's model

Parameter	**Value**
Pre-trained embedding	Glove's Twitter Vectors
Vocab Size	100,000
Embedding dimension	200
N hidden layers	3
LSTM direction	Bidirectional
N output	3 [positive, neutral, negative]
Dropout	0.5

When plotting the confusion matrix of each class label, as depicted in Fig. 5, we can determine what makes the model to be confused when classifying the sentiment. The matrix is normalised based on the frequency of the actual labels in the testing dataset. This is in order to deal with the issue of imbalance class within the testing set provided, where each label does not have the same amount of samples. A cell with a darker colour represents a larger frequency relative to the lighter ones. The matrices are constructed during the testing phase by comparing the ground truth label versus the prediction from the system.

By looking at the diagonal cells with the darkest colour amongst other cells as depicted in Fig. 5, it is apparent that the systems perform well at predicting each of the three labels. The lighter shades of blue translate to the fact that the system returned the wrong predictions at some samples at differentiating between neutral-positive (and vice versa) and neutral-negative. The white colour cells represent that the system to be able to differentiate relatively well at differentiating tweets with positive sentiment vs the negative ones.

Observing the confusion matrix is one feasible solution to provide us with an insight as to what to improve in the future. The interpretation of our plot indicates that the system suffers from differentiating a neutral sentiment tweet with another sentiment labels. Although it needs to be investigated further, this initial finding might suggest that these false predictions were due to imbalance class samples within the training set. This was true to our condition, as nearly half of the samples within the training set were neutral. The suggested improvement solution to this class imbalance proportion within training set might be to perform resampling, reweighting by taking into consideration the number of samples proportion for each class, or by using a class-balanced loss based [15].

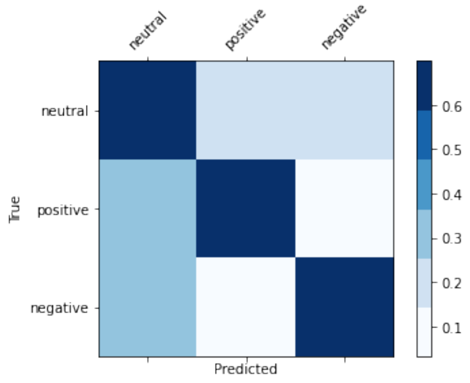

Fig. 5: Confusion matrix for task A

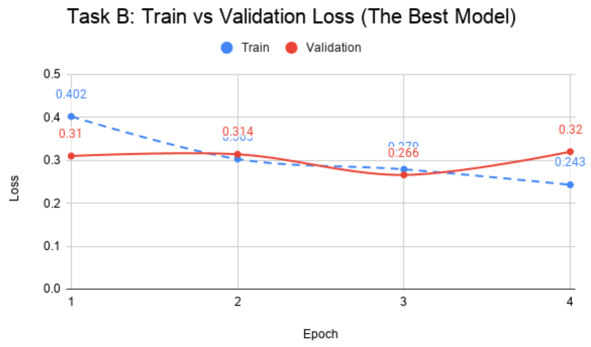

Fig. 6: Loss function score for task B

5.2. Task B: Topic-based Tweet Classification with two classes

The history of training and validation loss during training phase for task B is shown in Fig. 6. Just like the first one, the training phase of task B also stopped prematurely at a low iteration of epochs. The use of pre-trained word embedding to initialise the first weight as well as the use of Adam optimiser to minimise the training loss might contribute to this early optimisation. Had we not used this, the training loss would require more epoch iteration to minimise. The training phase stopped after a third iteration, as the validation loss did not improve during the next iteration, causing the model to overfit the data if we did not halt the process. From the hyperparameter tuning process, the system determined the best selection of parameters that gained minimum validation loss. These hyperparameters and vocabulary settings are listed in table 4.

Table 4: The best parameter of Task B's model

Parameter	**Value**
Pre-trained embedding	FastText
The use of topic	Append topic into the tweet text
Vocab Size	50,000
Embedding dimension	300
N hidden layers	2
LSTM direction	Bidirectional
N output	2 [positive, negative]
Dropout	0.8

As for the evaluation for the result, Fig. 7 indicates that the system performed well at the true positive element of the confusion matrix. The rest of the elements, however, could be improved in order to ramp up the reported average recall.

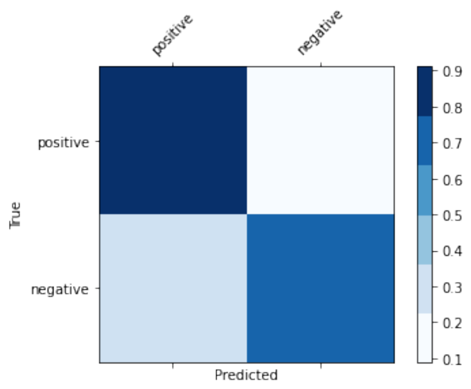

Fig. 7: Confusion matrix for task B

6. CONCLUSION

In this report, we present our attempt to solving the SemEval 2017 Task 4 challenge on Twitter sentiment analysis. There were four tasks introduced, but we focused on only the first two tasks of the challenge. The first task is detecting a sentiment of a tweet into three possible values: positive, neutral, or negative. The second task attaches additional human-annotated topic to a tweet, and the system should automatically classify the polarity of that post into either positive or negative.

By conducting sentiment analysis in the Twitter domain, we learned that the domain dataset itself provides us with a unique challenge. This is due to Twitter's distinctive use of language being written by its community, making it necessary for our system to be able to handle informal writing styles. These include the possibility of improper spelling and the presence of emoticons that might be helpful to indicate non-neutrality sentiment of a tweet. Using a proper text preprocessing tool is considered to be essential to clean the dataset.

After a series of text preprocessing, the cleaned dataset was then being exposed to the machine learning system. We used bidirectional deep LSTM that covered both forward and backward directions of the streams of information. With the help of pre-trained word embedding, we could initialise the weight of the neural network so that it had the knowledge of semantic and syntactic information of the words. The process of hyperparameter tuning was then being run in order to get the best model that could capture the pattern within the data. During this process, we also introduced early stopping to the system to prevent the model from being overfitting.

In the end, our system's reported average recall were 0.659 and 0.782 respectively for task A and B. This made us being in 7th position for task A, and 15th position for task B had we formally competed in this competition back in 2017. It is worth noting that the participants during that time were perhaps not being exposed to some new theories and tools being used in our project.

In retrospect, some future improvements can potentially be achieved for this project. For instance, implementing resampling and using weighted loss might be useful in order to adjust to the issue of imbalance class samples on the training set.

7. REFERENCES

[1] Enric Junqué De Fortuny, Tom De Smedt, David Martens, and Walter Daelemans, "Media coverage in times of political crisis: A text mining approach," *Expert Systems with Applications*, vol. 39, no. 14, pp. 11616–11622, 2012.

[2] Martin Harrysson, Estelle Metayer, and Hugo Sarrazin, "How 'social intelligence' can guide decisions," *McKinsey Quarterly*, , no. 4, pp. 81–89, 2012.

[3] Preslav Nakov, Alan Ritter, Sara Rosenthal, Fabrizio Sebastiani, and Veselin Stoyanov, "SemEval-2016 task 4: Sentiment analysis in twitter," *SemEval 2016 - 10th International Workshop on Semantic Evaluation, Proceedings*, pp. 1–18, 2016.

[4] Fabrizio Sebastiani, "An axiomatically derived measure for the evaluation of classification algorithms," in *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, 2015, pp. 11–20.

[5] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov, "Bag of tricks for efficient text classification," *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of Conference*, vol. 2, pp. 427–431, 2017.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, "GloVe: Global Vectors for Word Representation," *Empirical Methods in Natural Language Processing (EMNLP)*, vol. 31, pp. 1532–1543, 2014.

[7] M Schuster and K K Paliwal, "Bidirectional Recurrent Neural Networks," *Trans. Sig. Proc.*, vol. 45, no. 11, pp. 2673–2681, 11 1997.

[8] Diederik P. Kingma and Jimmy Lei Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[9] Wes McKinney and others, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*. Austin, TX, 2010, vol. 445, pp. 51–56.

[10] Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis, "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis," *SemEval-2017*, pp. 747–754, 2017.

[11] Adam Paszke, Sam Gross, Francisco Massa, et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H Wallach, H Larochelle, A Beygelzimer, et al., Eds., pp. 8024–8035. Curran Associates, Inc., 2019.

[12] Travis E Oliphant, *A guide to NumPy*, vol. 1, Trelgol Publishing USA, 2006.

[13] F Pedregosa, F. Pedregosa, G. Varoquaux, et al., "Scikit-learn: Machine Learning in {P}ython," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] Google, "Welcome to Colaboratory," https://colab.research.google.com/.

[15] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie, "Class-Balanced Loss Based on Effective Number of Samples," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2019.