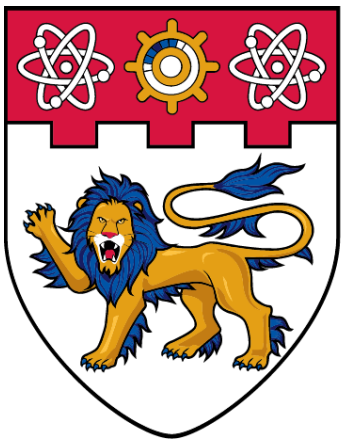

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042 Neural Networks and Deep Learning Project

Name	Matriculation Number
Ong Hui Lee Grace	U1921756L
Luo Wenyu	U1922009G
Wang Anyi	U1922992C

1 Introduction to Project Idea

In this project, we aim to utilise our understanding and knowledge of Neural Networks and Deep Learning on the research issue of Text Emotion Recognition. The project that we decided to work on is to propose a potential deep learning technique which identifies both local emotions expressed by words, and global emotions expressed by sentences. After research and implementation of the existing methods such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), we attempted to improve the models based on the existing methods and analysed the results that we obtained. In the following report, we will be explaining our reviews of current techniques and empirical analysis on the results in detail.

2 Review of Existing Techniques

2.1 CNN

Convolutional Neural Networks (CNN) uses convolution to perform the feature mapping, and then applies a max pooling operation to obtain a fixed-length output. The max pooling operation helps to capture the more important components in the feature map and use them for classification. However, CNN can only capture local information.

2.2 Bidirectional RNN

Bidirectional RNN involves concatenating two RNNs together, one passing information in the forward direction and one backwards. This enables the model to capture both the left context (words from start of sentence to current word) and right context (current word to end of sentence), which is better than the traditional windows method (only context within the window is captured).

Though the recurrence between the hidden units preserves the sequence in sentences, simple RNN models face the problem of vanishing gradient as the gradient is multiplied many times during back propagation. This causes the gradient to be too small when it reaches earlier hidden units and learning becomes inefficient. It also has difficulties in capturing long term dependencies when there is a large distance between information in a sentence. (since later/ nearer words are often more dominant than earlier words in an RNN.)

2.3 Bidirectional LSTM

LSTM models aim to solve the problem of vanishing gradients in RNN. LSTM units include a 'memory cell' that can maintain information in memory for long periods of time using the cell state that runs across the units. A set of gates is used to control when information enters the memory, when it's output, and when it's forgotten.

The structure of bidirectional LSTM is similar to the bidirectional RNN above (except simple RNN units are replaced by LSTM units), and also serves to improve model performance on sequence classification problems by capturing backward and forward information about the sequence at every time step.

2.4 Bidirectional RNN-CNN

We also considered using a bidirectional RNN followed by a convolutional layer in hopes of capturing both local and global information. Here, the bidirectional RNN forward and backward feature learning can help to preserve the sequence data, while the CNN helps make feature extraction accurate.

2.5 BERT (bidirectional encoder representations from transformers)

BERT is a state of art model that uses a Transformer encoder to read the entire sequence of words at once (therefore the sequence of the words does not matter). It enables the model to learn the context of a word based on its neighbours. We expect the BERT model to perform well as the encoder forwards a more complete picture of the sentences than sequential.

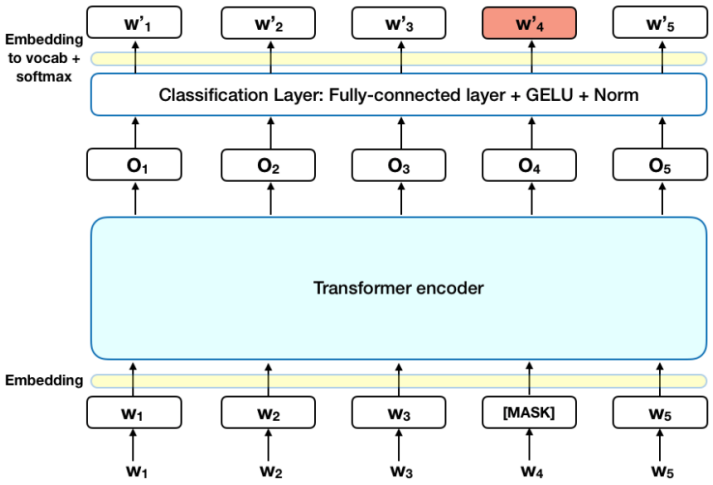

Figure 1. Structure of BERT

3 Description of method

3.1 Data Preprocessing

All of our data sets were preprocessed by converting the sentiments to one hot encoding. The sentences are also tokenized using nltk.tokenize. We also created functions for encoding words in each sentence in our datasets. This is necessary for our glove embedding later on.

- Firstly, the create_dico function creates a dictionary of “token : count” key-value pairs from the tokenised data. All words are in lower case.
  Eg. { ‘'i': 24005, 'know': 1351, 'was': 2810, 'listenin': 10, 'to': 14359, … }

- create_mapping function maps each unique token in the dictionary from the create_dico function to a unique index. It returns two dictionaries: one mapping id to token and the other mapping token to id. Both dictionaries are sorted in descending order of token count (number of occurrences of token in data). In case there are unknown words in the test data that did not appear in train data, they will be regarded as <UNK> with index 0.
  Eg. word_to_id dictionary : {'<UNK>': 0, 'i': 1, '!': 2, '@': 3, '.': 4, 'to': 5, 'the': 6, ... }
  Id_to_word dictionary : {0: '<UNK>', 1: 'i', 2: '!', 3: '@', 4: '.', 5: 'to', 6: 'the', … }

- Finally, the prepare_dataset function returns a list of dictionaries. Each dictionary represents a sentence. ‘str_words’ stores the tokens in sentence and ‘words’ stores the indexes of those words.
  Eg. {'str_words': ['wants', 'to', 'hang', 'out', 'with', 'friends', 'SOON', '!'],
  'words': [446, 5, 713, 44, 33, 229, 190, 2]}

3.2 Our model

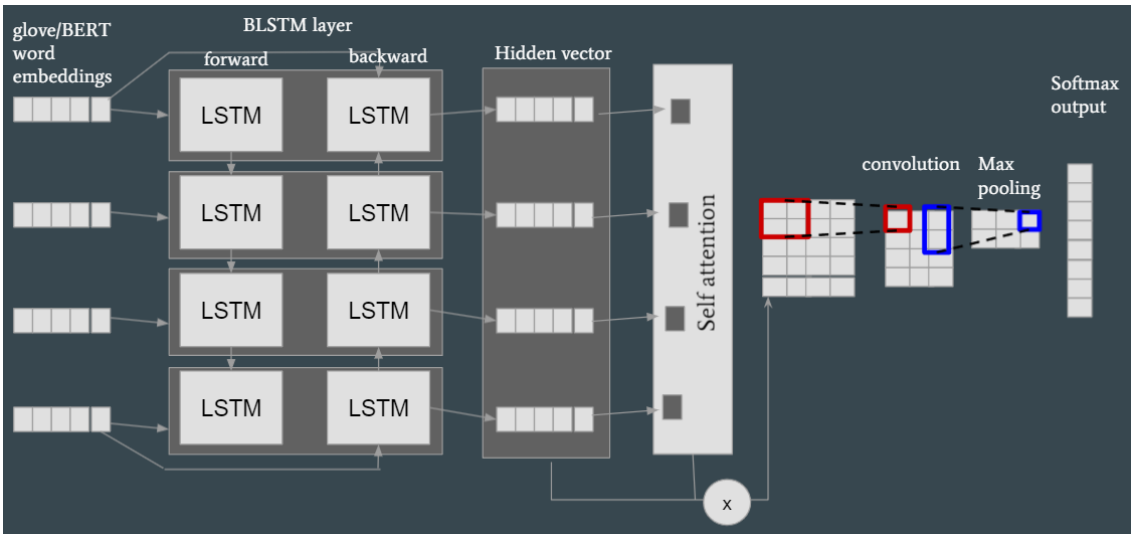

Figure 2. Structure of our model

3.2.1 Bidirectional LSTM-Self_Att-CNN

After referencing the paper (Zhou 2016), we developed our model consisting of an embedding layer, BLSTM with self attention, a convolutional layer, max pooling layer. For the sequence modeling task, BLSTM has access to the past context as well as the future context, hence capturing long-term dependencies. From the paper, it was found that the accuracy decreased with longer sentences, hence multiplicative self-attention was applied to preserve long-term dependencies. Self-attention relates different positions of a single sequence to compute a more vivid representation of the same sequence. Convolution is then applied to extract local features, followed by max pooling to capture the more meaningful information. From the paper, the hyperparameters after fine tuning are as follows:

The dimension of word embeddings is 300, the hidden units of LSTM is 300. We use 100 convolutional filters each for window sizes of (3,3), 2D pooling size of (2,2). We set the mini-batch size as 10 and the learning rate of AdaDelta as the default value 1.0. For regularization, we employ Dropout operation (Hinton et al., 2012) with dropout rate of 0.5 for the word embeddings, 0.2 for the BLSTM layer and 0.4 for the penultimate layer, we also use l2 penalty with coefficient 10−5 over the parameters

3.2.2 glove embedding or BERT embedding

There are 2 different ways of embedding words before feeding them into our model.

3.2.2.1 Glove embedding

We loaded a pre-trained word embedding file, ‘glove.6B.300d.txt’. The embeddings have been pre trained on a 6 billion word corpus (Wikipedia 2014 + Gigaword 5) using word2vec. It consists of 400000 pretrained 300 dimensional vectors (each glove vector representing one unique word). Each one of the 300 dimensions in the vector serves to capture a part of the meaning of the word.

{'the': array([-0.038194, -0.24487 , 0.72812 , -0.39961 , 0.083172, 0.043953,.....

-0.52028 , -0.1459 , 0.8278 , 0.27062]),
'of': array([-0.1529 , -0.24279 , 0.89837 , 0.16996 , 0.53516 , 0.48784 ,.....
-0.56094 , -0.591 , 1.0039 , 0.20664]),
….}

We used this to create our word_embeds embedding lookup matrix, which contains the corresponding 300-dimensional vector for all unique words in our data's dictionary. This way, we can perform a simple lookup in the neural network's embedding layer using the indices of the words we preprocessed earlier. Such matrix operations are considerably faster for doing the lookup.

3.2.2.2 BERT embedding

The BERT embeddings are pre trained on a large corpus of text. BERT applies transformer-based encoding on the text, and masked language modelling, where 15% of the text is hidden and shuffled before inferring the positional embeddings. We then import them and fine tune on our task. We imported the 'small_bert/bert_en_uncased_L-4_H-256_A-4' model, which uses a smaller transformer block. For each of our data, we then used our imported model to transform them into a length 128 array of indices (one index for each token, index 101 for start of sentence and 102 for end of sentence, then padded by zeros). We can then generate contextual embeddings for each token (each token is represented by a vector of length 256). The final input therefore has dimension [batch_size, 128, 256].

4 Experiments and Results

4.1 Description of datasets used

We used 3 datasets in our model: Crowdflower, WASSA and Emotions dataset.

4.1.1 Crowdflower

Contains 40000 rows and 13 unique sentiments.

	tweet_id	sentiment	author	content
0	1956967341	empty	xoshayzers	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	wannamama	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	coolfunky	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	czareaquino	wants to hang out with friends SOON!
4	1956968416	neutral	xkilljoyx	@dannycastillo We want to trade with someone w...

Figure 3. Dataset preview for crowdflower

4.1.2 WASSA

WASSA contains 3960 rows with 4 different emotion labels.

	index	id	tweet	tweettype	score
0	1	10000	How the fu*k! Who the heck! moved my fridge!.....	anger	0.938
1	2	10001	So my Indian Uber driver just called someone t...	anger	0.896
2	3	10002	@DPD_UK I asked for my parcel to be delivered ...	anger	0.896

Figure 4. Dataset preview for WASSA

As each row also has a continuous score for the sentiment and we are conducting a classification problem, we further processed the data by setting a threshold value. From observation, we noticed that tweets with low scores do not really represent the sentiment well. For instance, “i love the word fret so much and im in heaven” had a sadness score of 0.292. We therefore decided to set the sentiment threshold score at 0.7 (drop all scores below 0.7) to ensure the sentiment (“tweettype” column) accurately describes the tweets.

4.1.3 Emotions

Emotions dataset contains 20000 rows including train, test and validation dataset. There are 6 distinct sentiments.

	col	senti
0	i didnt feel humiliated	sadness
1	i can go from feeling so hopeless to so damned...	sadness
2	im grabbing a minute to post i feel greedy wrong	anger
3	i am ever feeling nostalgic about the fireplac...	love
4	i am feeling grouchy	anger
...	...	...

Figure 5. Dataset preview for Emotions

4.1.4 Summary Statistics of Datasets

Dataset	No. of labels	Avg. Sentence Length	Max. Sentence Length	Train Size	Test Size	Vocab Size
Crowdflower	13	73	167	28000	12000	52746
WASSA (threshold 0.7)	4	92	158	500	125	12393
Emotions	6	96	300	16000	4000	15211

Table 1. Summary statistics of the three datasets

4.2 Results

This table shows the validation accuracies of the different models on the 3 datasets. Each dataset was run for 20 epochs with a batch size of 10 (except Bi LSTM-CNN (BERT) and Bi LSTM-CNN (BERT) with attention which ran for 50 epochs with batch size of 32 for faster computation) and the validation accuracy of the last epoch was recorded. We only trained the Bi LSTM-CNN (BERT) model with Emotions dataset due to time constraint.

	Dataset		
	Crowdflower	**WASSA 0.7 threshold**	**Emotions**
CNN	0.2506	0.8240	0.9070

Bi RNN	0.2102	0.4521	0.7875
Bi LSTM	0.2454	0.7040	0.9000
BERT	0.40125	0.8886	0.9393
Bi RNN-CNN	0.2915	0.2960	0.6071
Bi LSTM-CNN (Glove)	0.2124	0.7760	0.9262
Bi LSTM-CNN (Glove) +att	0.2399	0.7760	0.9275
Bi LSTM-CNN (BERT)	-	-	0.7865
Bi LSTM-CNN (BERT) +att	-	-	0.7990

Table 2. Results obtained from different models

4.2.1 BERT Embedding

Bi LSTM-CNN (BERT) without attention

Initially we have set the training epoch to be 20. After obtaining the results, it is observed that although the training loss shows a converging trend, the loss still decreases. Hence, we decided to adopt a higher training epoch number, which is 50 to better monitor the change in loss and validating accuracy. Based on the results collected, the training accuracy increases from 0.2977 to 0.9649, and the validating accuracy increases from 0.2583 to 0.7865. After the 30th epoch, the validating accuracy shows almost no sign of increasing and fluctuates around 0.75. The training loss decreases from 12.3015 to 0.1045 and validating loss decreases from 3.5060 to 1.1039. Both converge at about the 5th epoch. From the 20th epoch onwards, the training and validation accuracies and losses diverged, with the training accuracy higher than the validation accuracy, and the training loss lower than the validation loss. This suggests overfitting was present.

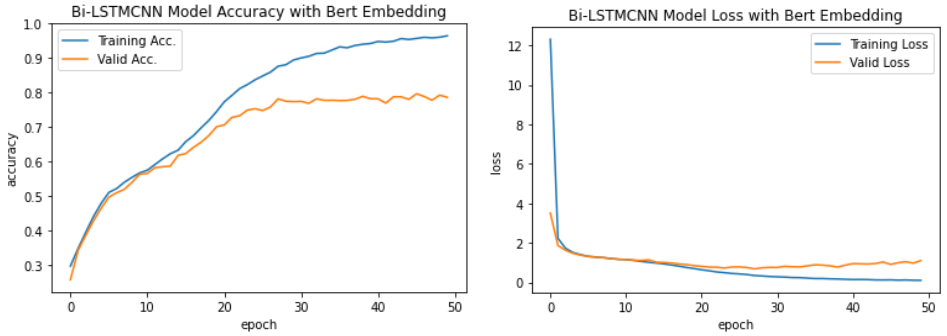

Figure 6. Results for Bi LSTM-CNN model on Emotions dataset

Bi LSTM-CNN (BERT) with attention

We have incorporated self-attention to the model as an attempt to improve the results. After obtaining the results, it can be observed that the increase in the training and validating

accuracy between the 5th epoch to about 15th epoch are larger as compared to the model without self-attention. However, there seems to be larger gap between training and validation accuracy and loss from the 20th epoch onwards, indicating that the model is also overfitting.

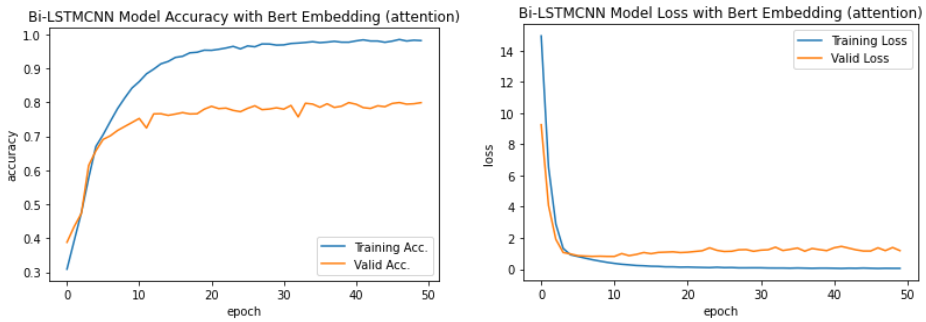

Figure 7. Results for Bi LSTM-CNN (BERT) with attention on Emotions dataset

4.2.2 GloVe Embedding

As shown in the graphs below, accuracy graphs for both Bi LSTM-CNN (GloVe) and Bi LSTM-CNN (GloVe) with attention shows sigmoid shape, indicating that both models learn slowly initially and become faster and more proficient at the later stage.

Bi LSTM-CNN (GloVe) without attention

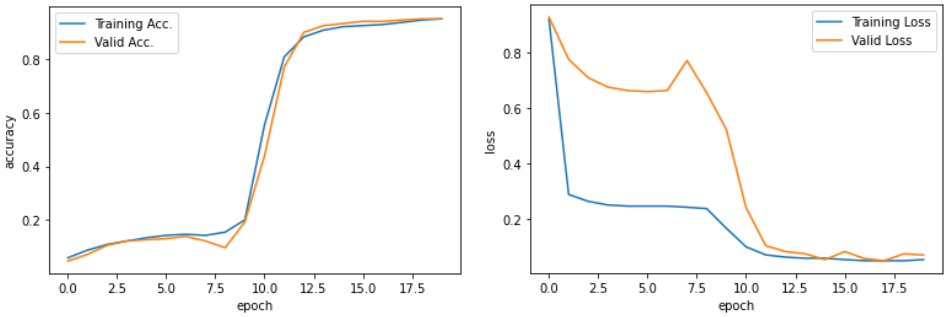

Figure 8. Results for Bi LSTM-CNN model with attention on Emotions dataset

Bi LSTM-CNN (GloVe) with attention

It is evident in the graph that with attention, the graphs for both validating accuracy and loss becomes smoother as compared to the previous graphs without attention. The small spike observed from the previous validating accuracy graph between epoch 6 to 8 is also eliminated after implementing self-attention.

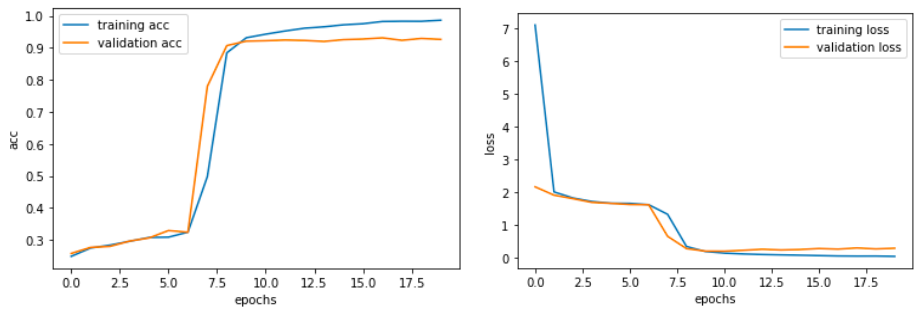

Figure 9 . Results for Bi LSTM-CNN + attention model with attention on Emotions dataset

5 Discussion

From the results, our model seems to outperform most existing models, except for the BERT model. BERT capitalises on a transformer is an encoder architecture model which uses attention mechanisms to forward a more complete picture of the whole sequence to the decoder at once rather than sequentially (e.g. RNN, LSTM etc.) Using a transformer architecture may contribute to a more effective modeling of long term dependencies among tokens in a temporal sequence, and the more efficient training of the model in general by eliminating the sequential dependency on previous tokens. Perhaps the BERT model was able to generate more contextualized word embeddings and we accept that our model is not comparable to such state-of-the-art models.

Since applying attention mechanisms helps better model long term dependencies of sequential data, it could be used to improve upon our model. From our results,our proposed model Bi LSTM-CNN (GloVe) incorporates a self-attention mechanism which transformers capitalise on and perform better than the majority of existing models, with the exception of the BERT model.

Based on the results, Bi LSTM-CNN with GloVe embedding yields better results as to Bi LSTM-CNN with BERT embedding. This is probably because the BERT embedding was trained from scratch on a smaller corpus compared to GloVe, which may be the cause of the compared overfitting as the contextual embeddings learned may not apply to larger general data contexts. The high training accuracy suggests GloVe embedding was able to learn the context of the training data well. Hence perhaps with the pre-trained embeddings on a large corpus, this could help reduce overfitting.The best model obtained from Bi LSTM-CNN (GloVe) was saved and loaded to predict the emotion label for the input phrase or sentence, demonstrating its ability to recognise the local and global emotions expressed by word and sentences. We will receive two outputs, one is an array of floats and the other is the predicted label. The array contains six floats, each indicating the probability of the sentence being classified to each label. The predicted label is taken from the one with the highest probability. Below are a few examples of our emotion recognition test results:

Test 1:
Input: ‘A wonderful day’
Output: [0.004831992089748383, 0.002158567076548934, 0.8076752424240112, 0.15726424753665924, 0.010409458540380001, 0.017660509794950485]
→ joy

Test 2:
Input: 'I am not like everyone around me who are leading great lives and doing well at work.'
Output:
[0.000549889518879354, 0.0001760266604833305, 0.8754567503929138, 0.01664547063410282, 0.107157863676548, 1.3987215425004251e-05]
→ joy

Test 3:
Input: 'i went to work in the morning and worked overtime and cried when i got home'
output:

[0.005073187872767448, 0.00014828183338977396, 0.12221956998109818, 0.0001271497312700376, 0.8724318146705627, 1.2256062831283998e-08]
→ sadness

For all three tests above, the input can be either a sentence or a phrase. We input a few strings of different lengths to show the ability of the model to capture both local and global information.

Test 1 and Test 3 seem to yield reasonable outputs. The model was able to capture local emotion in the word ‘wonderful’ in Test 1 and the dependency between ‘I’ and ‘cried’ in Test 3. However, test 2 failed to recognise the negation in the sentence which expresses a completely different emotion. Hence, further improving the long-term negation dependency could be a potential improvement for the model to better recognise the emotions expressed, especially in longer sentences.

In addition, the crowdflower dataset generally yielded bad results for all models. This was possibly due to the ambiguity of the labels in the dataset itself. Even with manual human labelling, it was difficult to label texts correctly especially since there are many unique emotions like emptiness and boredom, hate and anger, happiness, surprise, enthusiasm, fun, which were too fine grained and difficult to differentiate.

6. Reference

1. Kim, Y. Convolutional neural networks for sentence classification, Conference on Empirical Methods in Natural Language Processing, pp. 1746–1751, 2014.

2. Lai, S., Xu, L., Liu, K., & Zhao, J. Recurrent convolutional neural networks for text classification. Proceedings of the National Conference on Artificial Intelligence, vol. 3, pp. 2267–2273, 2015.

3. Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., & Xu, B. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. 26th International Conference on Computational Linguistics, vol. 2, no.1, 3485–3495, 2016.

4. ManuelVs, NNForTextClassification, (2019), GitHub repository, https://github.com/ManuelVs/NNForTextClassification