CSCE 479/879 Homework 2 [IMBD]

Beichen Zhang, Yinglu Jia, Bin Ma

March 13, 2021

Abstract

In this report, we built two different sequential architectures based on the IMDB data set. The model architectures and hyperparameter training are the two most essential components in applying the deep-learning techniques to the industry and research. Therefore, this study employs two models: Long Short-Term Memory (LSTM) and Bidirectional Encoder Representation from Transformers (BERT), to examine the impacts of different model structures and hyperparameters on the natural language classification. For each architecture, we included an attention mechanism and a set of regularization, and tested at least two settings of hyper-parameters. In the first part of the report, we compared the differences between the single LSTM and the stacked LSTM architectures. We also the investigated impacts of the pre-trained word-embedding model (GloVe) and the Global Maximum Pooling layer on the LSTM models. Six architectures were examined and compared in total. The best-performed model achieves an accuracy of 83.34% on the test data set. In the second part, BERT models were trained and tested. We fine-tuned the hyper-parameters of BERT based classifier model with two different regularization methods to overcome its over-fitting problem. As a result, the BERT based model achieved 85.8% accuracy on the test data set.

1 Introduction

Natural language processing (NLP) is a field focusing on enabling the programmed computers to process and analyze large amounts of natural language data. Sentiment analysis is one challenge of NLP techniques, which means extracting emotions from raw texts. This is usually applied to automatically understand whether some users feel positive or negative on social media posts and customer reviews. One dataset to study sensitive analyses is the IMDB data set, which includes more than 5,000 movie reviews labeled "positive" or "negative". IDMB is widely used in theory as a benchmark of the document-level NLP model developed. Therefore, it is beneficial for us to train our proposed sequential architectures on IMDB data set and compare them with other results to evaluate our models' performance.

To analyze reviewers' sentiments based on their content, we consider two methods, LSTM and BERT. LSTM is a method typically applied to text [8]. As a recurrent neural network (RNNs), LSTM takes words following the order of sequence one by one. Thus, the output of prediction is made based on the sequence of words, which enables it to learn the dependencies among the words. BERT is a recent NLP model that has shown groundbreaking results in many tasks such as sentiment classifications [9], Question Answering tasks[10] and Named Entity Recognition [14]. Since it is openly available, it has become popular in the research community.

LSTM and BERT are among the best NLP models in recent research. However, they are both so complex that it is difficult to understand their hyper-parameters' impacts on their performance. This report examined the impacts of LSTM and BERT and their corresponding hyper=-parameters on NLP classification. We pursue LSTM and BERT's best use for our purpose by including an attention mechanism, a regularizer, and pre-training models. We change the models' setting ups and hyper-parameter values and compare their impacts to discover the properties of different architectures and properties.

As a result, our best LSTM model reached a test accuracy of 83.34%. We tested the architecture with a different number of LSTM layers. A more complex stacked LSTM model significantly improved both test accuracy and 5-fold cross-validation (CV) accuracy of prediction. The dropout term and the global maximum pooling layer were also tested as regularization, which improved the LSTM model's performance to some degree. In contrast, the applied pre-trained BERT model was pretty complex itself. So the BERT model showed apparent over-fitting. By overcoming this problem, we applied different regularization methods. However, our performed regularization methods improved the performance slightly. Even though the BERT model was over-fitting, it provided the best test accuracy [84.9%, 86.8%] with 95% confidence among our trials.

2 Problem Description

For this assignment, we apply sequential neural network models to the problem of sentiment analysis on the IMDB dataset. IMDB dataset [1], which consists of 50,000 reviews labeled as positive or negative. The IMDB dataset is a binary sentiment analysis dataset consisting of 50,000 reviews from the Internet Movie Database (IMDB) labeled as positive or negative. The dataset contains an even number of positive and negative reviews. Only highly polarizing reviews are considered. A negative review has a score 4 out of 10, and a positive review has a score 7 out of 10. No more than 30 reviews are included per movie. Models are evaluated based on accuracy.

In this report, We construct several sequential models and train the models to classify whether the review is positive or negative. This problem requires machine to understand the sentiments of natural language, which is not straight-forward to machine. What makes it more difficult is that people use a great deal of phrases to express their feelings rather than simple "yes" or "no". Sometimes a single word have different meanings depending on their position in the sentence. Thus, it is an intriguing problem to study whether a model can learn natural language and further analyze its sentiments.

IMDB dataset is an important benchmark to evaluate whether the new NLP module performs well. And since the movie review usually contains more than 100 words, this dataset represents the complexity of document-level NLP problem. Therefore, it is valuable and instructive for us to play with this dataset and compare the influence of different settings and hyper-parameters of sequential models.

3 Approaches

3.1 Pre-processing

The dataset contains movie reviews written by the viewers. It is written in natural language. In order to process the text into computer recognizable input, we need to pre-process the data. Also, there will always be some noise in the text, and we will also need to remove it. There are different ways to pre-process text, including:

1. Lowercasing
2. Tag Removal
3. Special Character Removal
4. Stopword Removal
5. Effects of Text Normalization
6. Noise Removal
7. ......

The strategies we applied in our models will be show in Section. 4.2

3.2 Feature Engineering

Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques. Working with unstructured text data is hard especially when you are trying to build an intelligent system which interprets and understands free flowing natural language just like humans. We need to be able to process and transform noisy, unstructured textual data into some structured, vectorized formats which can be understood by any machine learning algorithm. All machine learning algorithms are based on principles of statistics, math and optimization, hence, they are not intelligent enough to start processing text in their raw, native form.

1. **Traditional Model** Traditional (count-based) feature engineering strategies for textual data involve models belonging to a family of models popularly known as the Bag of Words model. However, while they are effective methods for extracting features from text, due to the inherent nature of the model being just a

bag of unstructured words, we lose additional information like the semantics, structure, sequence and context around nearby word in each text document.

2. **Word Embedding** when we want to apply deep leaning technique to raw text data, especially count based models like Bag of Words, we are dealing with individual words which may have their own identifiers and do not capture the semantic relationship amongst words. This leads to huge sparse word vectors for textual data and thus if we do not have enough data, we may end up getting poor models or even over-fitting the data due to the curse of dimensionality. Word embedding turns word to the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning. This strategy, in short, treat the word in the text as linked valued not individual operand.

3.2.1 Sequential architecture

In this report, we are going to apply two different sequential architectures for this problem. The first one is LSTM and the second one is BERT.

3.2.2 LSTM

LSTM is an artificial recurrent neural network (RNN) architecture. LSTM was proposed in 1997 by Sepp Hochreiter and Jurgen Schmidhuber, and it was gradually improved over the years. LSTM are very widely applied in Robot control, Rhythm learning, Hand writing recognition and Human action recognition et ac. In 2007, LSTM started to revolutionize speech recognition, outperforming traditional models in certain speech applications [8]. The reason for the outstanding performance of LSTM compared with traditional NN is because of its ability in persisting former information with loops in it. This is apparent if you consider how we read an essay. We understand each word based on our understanding of previous words. We don't throw everything away and only focus on a single word.

Compared with other RNN architecture, LSTM is especially capable of learning long-term dependencies, such like speech recognition, translation and sequence classification. Because it is designed to accumulate information about the past, but also decides when to forget information. To illustrate this, the architecture of a basic LSTM cell was shown in Figure. 1. A common LSTM unit use coupled forget and input gates. In thus, LSTM only forget memories from c(t-1) when new input x(t) is added. While it only inputs new values x(t) to the state h(t) when it forgets something older c(t-1). As a result, the output is determined both by long memory (c(t-1)) and short memory h(t). The compact forms of the equations for the forward pass of an LSTM unit with a forget gate are shown below.

3.2.3 BERT

BERT [6] is a new art-of-state model, which arouses a wide heat and interests in application of various NLP tasks, including sentiment classifications [9], Question

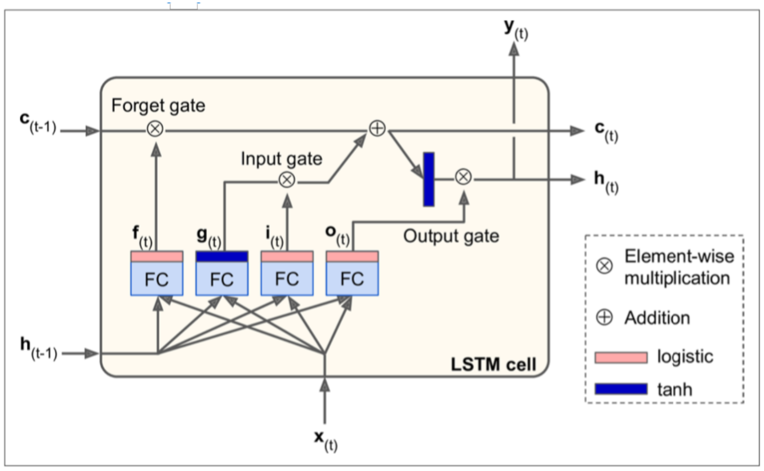


Figure 1: Basic Cell of LSTM

Answering tasks [10] and Named Entity Recognition [14].BERT is a transformer [16] using attention mechanism, which is a different architecture from recurrent networks like LSTM. This transformer learns embedding by training the model to predict next-token and next-sentence probabilities. It can not only output the learned embedding of the sentence, but also provides keywords that are important to the semantics of the sentence.

As shown in Figure. 2, The transformer is combined by two parts: Encoder (left part) and Decoder (right part). Both Encoder and Decoder are composed by stacking multiple (N_x) times of modules, which are consisted mainly of Multi-Head Attention and Feed Forward layers. The scheme of Multi-Head Attention is shown in Figure. 3, in which the left figure is a general description of the attention-mechanism that can be expressed by the following equation:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt(d_k)})V \tag{1}$$

Here the attention can be understood as matching a query (Q, vector representation of one word in the sequence) to a key (K, vector representations of all words in the sequence) and divided by the number of K (d_k) and mapping to a distribution of softmax. And then the distribution is scaled with a set of values (V, vector representations of all the words in the sequence). In Multi-Head Attention (Figure. 3 right), this attention mechanism is repeated multiple times with linear projections of Q, K and V, which is done by multiplying Q, K and V by weight matrix W that are learned during training.

The input and output embedding are n-dimensional embedded vectors from initial Stokes. However, the decoder input will be shifted to the right by one posi-

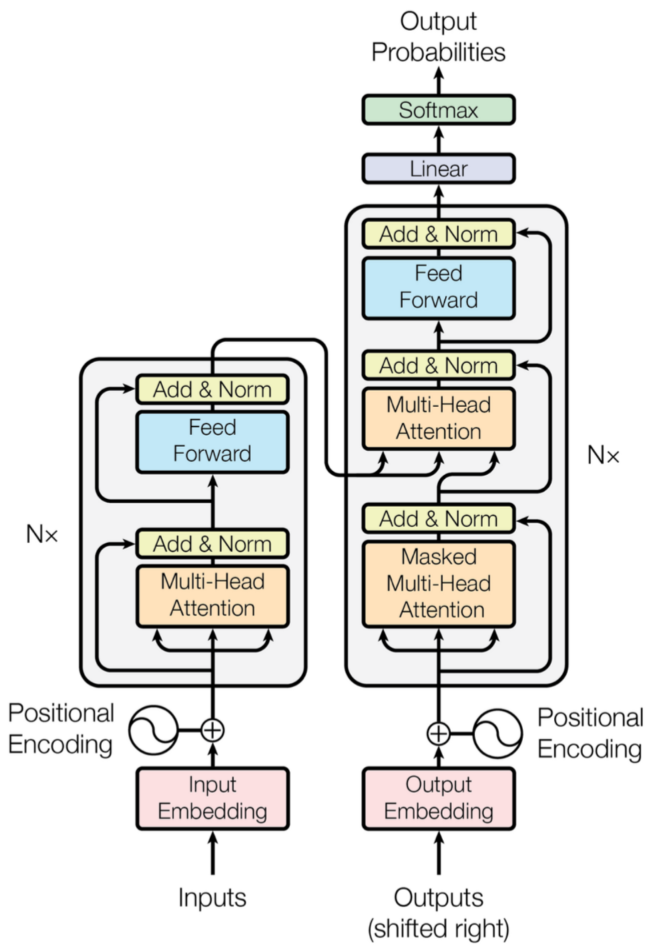


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consist attention layers running in parallel[16]

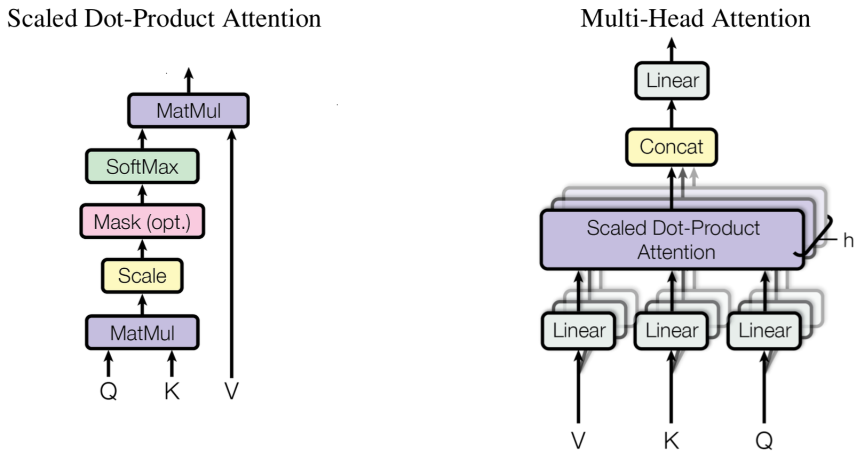


Figure 3: Architecture of a transformer[16]

tion to avoid directly copying decoder input during training. Unlike LSTM which naturally injects the sequence of words with recurrent and convolution, the transformer makes use of positional encoding to remember the relative position of each word/part in the sequence.

The advantage of BERT's is applying the bidirectional training of the transformer. Compared with previous methods which read a text sequence either from left to right or from right to left, it shows bidirectionally trained BERT achieves higher MNLI Dev accuracy than single-direction language models [6]. The model is pre-trained to use a "masked language model" (MLM) objective to predict next-tocken probability.

To use BERT on Classification tasks is pretty straight-forward by adding a classification layer on top of the Transformer output for the [CLS] token shown as Fig. 4 By fine-tuning the hyper-parameters mainly exist on the classifier's layer and keeping the values of pre-trained hyper-parameters almost same in BERT, we can train our model to predict whether the review is positive or negative.

3.3 Text Classification

3.3.1 Attention mechanism

Attention is one of the most influential ideas in the Deep Learning community.The essential idea behind Attention is not to throw away those intermediate state but to utilize all the states in order to construct the context vector required to generate the output sequence.

Intuitively, Attention mechanism will make the model more focused on the features which have high influence on the prediction. For example, if we want to anal-

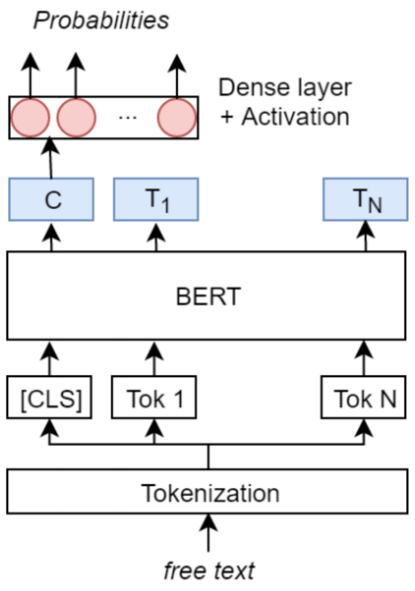


Figure 4: f the BERT model for multi-label classification [13].

yse this sentence "John is a good student", when we want the information about the identity of this person, we need the model more focus on the word "student". However, if we want the information of sentiment of this student, we need the model to be more focused on the "good".

3.3.2 Regularization

Regularization is a common technique we use to prevent over-fitting problems in classification algorithm. Over-fitting is a phenomenon in machine learning where the hypothesis function has high variance and is affected by every small random variance in the data. It is usually caused by lack of enough data or when you have too many features. There are many strategies here, including L1 L2 regularization, dropout, maxpooling, data-augmentation and so on.

4 Experimental Setup

4.1 Data Description

The IMDB dataset consists of 50,000 movie reviews that were grabbed from the IMDB website. It is binary classified into positive and negative labeled as 1 an 0 separately in terms of the sentiment from viewers. There are 25,000 reviews for training and the other 25,000 reviews for testing. The data consist of the text content of the reviews and their binary labels. We loaded full training data set as a tensor with type ‘tf.string‘ in the original text representation. The imported data was then split into training (20,000) and test (5000) data sets. The training data set was further split into 5 folds to perform the k-fold cross validation.

4.2 Data Pre-processing and Feature Engineering

Considering the data is a tensor with type ’tf.string’ in the original text representation, text data pre-processing techniques will be applied here. The applications we used are listed below:

1. **Removing HTML Tags:** Often, the unstructured text contains a lot of noise, especially if you use techniques like web or screen scraping. HTML tags are typically one of these components which don’t add much value towards understanding and analyzing text. The first task of pre-processing is to remove those HTML tags.

2. **Removing Accented Characters:** Usually, in any text corpus, you might be dealing with accented characters/letters, especially if you only want to analyze the English language. Hence, we need to make sure that these characters are converted and standardized into ASCII characters. A simple example- converting é to e.

3. **Removing Special Characters:** Special characters and symbols are usually non-alphanumeric characters or even occasionally numeric characters (depending on the problem), which add to the extra noise in unstructured text. Usually, simple regular expressions (regexes) can be used to remove them.

4. **Removing Stopwords** Words that have little or no significance, especially when constructing meaningful features from the text, are known as stopwords. These are usually words that end up having the maximum frequency if you do a simple term or word frequency in a corpus. Typically, these can be articles, conjunctions, prepositions, and so on. Some examples of stopwords are a, an, the, and the like. Stopwords will be removed in this step.

4.3 Model building

4.3.1 LSTM

1. **LSTM architectures** In this homework, we primarily tested two different LSTM-based architectures: single LSTM and stacked LSTM. The single LSTM was a sequential architecture with a single hidden layer of the LSTM units. And the stacked LSTM contained three stacked LSTM units that expected to learn the sentiment of the reviews better. For the single LSTM, the default hyperbolic tangent (Tanh) was applied with 128 filters. And the dropout rate was set as 0.2 within the units to overcome the over-fitting issues. The stacked LSTM has three hidden layers with consistent LSTM units that used the Tanh activation

function with 100 filters. Instead of adding a dropout term within the units, the stacked LSTM adds a dropout layer onto the model.

2. **Word embedding** Word embedding is an approach to represent the words and their relationships in the text. A word-embedding model will encode the tokens (words) into numerical vectors in the vector space. The closer the distances between two words present a similar meaning in the text. The word-embedding model can be pre-trained in advance and added to a proposed model. Therefore, in this homework, we applied a pre-trained unsupervised learning model: GloVe, and compared it to a self-trained model based on the inputting data [12]. The employed GloVe model was trained by 840B tokens in 2.2M vocabulary in lowercase and represented in 300-dimension vector space [11]. To make a fair comparison, we also set the vector space to 300 dimensions for the self-trained embedding model. Before we input the tokens into the word-embedding model, they were converted to the numerical sequence in the same length (64).

3. **Attention mechanism** A Luong-style attention layer was employed to both LSTM architectures to assign weights on the work vectors and improve the models' capability to understand tokens correctly. The layer was calculated and added after the word embedding and before the LSTM layers.

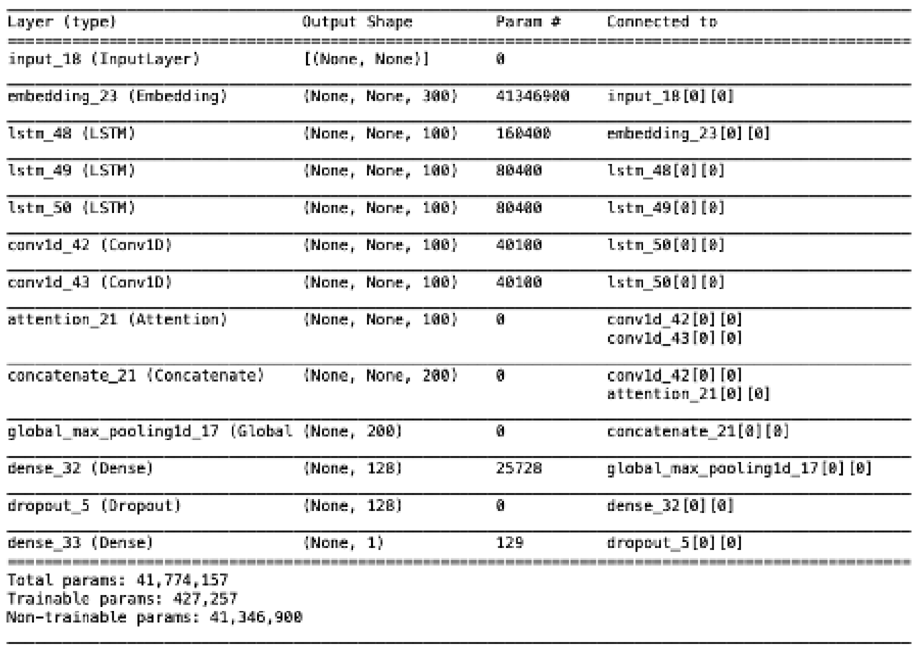
4. **Regularization** Except for the dropout term and layer added onto the model, a one-dimensional global maximum pooling layer was added to the models as an optional. This layer will gather information from all LSTM units and output the maximum value. A comparison between models with and without the maximum pooling layer was applied to investigate its impacts on addressing the over-fitting and improve models' performances.

5. **Activation functions** The default activation function (Tanh) was applied to all LSTM units to make the training process more stable and robust. For all dense layers, ReLu was used as the activation function to improve the training speed and provide sparse representations of values.

6. **Output** In the end, the sigmoid activation function was used as the output layer because the classification is binary. We used 0.5 as the threshold to transform the probabilities to labels, where the probabilities smaller than 0.5 were labeled as negative (0), and the rest were positive reviews (1). In Figure 5, the architecture of best-performed LSTM model was presented as an example.

4.3.2 BERT

Sequential architecture The other 2 sequential architectures for this problem we used is BERT, which is a self-attention model. In thus, we do not need to add extra attention layer as implemented in LSMT methods. With the limited time of model training, we used a pre-trained BERT model[15] obtained from Tensorflow-hub (TF



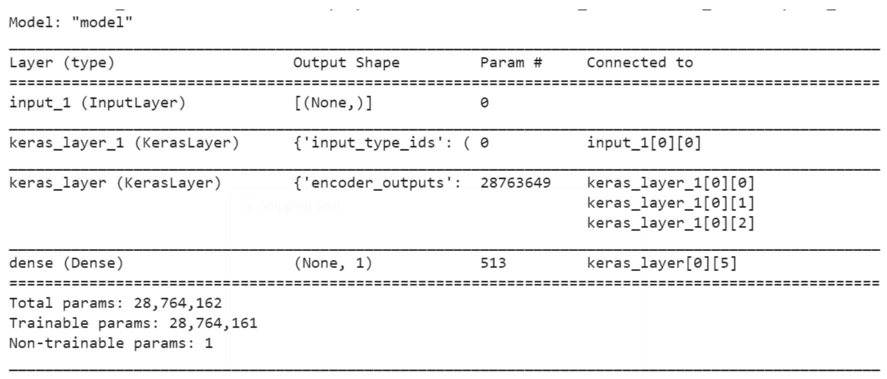
```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_18 (InputLayer)           [(None, None)]       0

__________________________________________________________________________________________________
embedding_23 (Embedding)        (None, None, 300)    41346900    input_18[0][0]
__________________________________________________________________________________________________
lstm_48 (LSTM)                  (None, None, 100)    160400      embedding_23[0][0]
__________________________________________________________________________________________________
lstm_49 (LSTM)                  (None, None, 100)    80400       lstm_48[0][0]
__________________________________________________________________________________________________
lstm_50 (LSTM)                  (None, None, 100)    80400       lstm_49[0][0]
__________________________________________________________________________________________________
conv1d_42 (Conv1D)              (None, None, 100)    40100       lstm_50[0][0]
__________________________________________________________________________________________________
conv1d_43 (Conv1D)              (None, None, 100)    40100       lstm_50[0][0]
__________________________________________________________________________________________________
attention_21 (Attention)        (None, None, 100)    0           conv1d_42[0][0]
                                                                 conv1d_43[0][0]
__________________________________________________________________________________________________
concatenate_21 (Concatenate)    (None, None, 200)    0           conv1d_42[0][0]
                                                                 attention_21[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_17 (Global (None, 200)          0           concatenate_21[0][0]
__________________________________________________________________________________________________
dense_32 (Dense)                (None, 128)          25728       global_max_pooling1d_17[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 128)          0           dense_32[0][0]
__________________________________________________________________________________________________
dense_33 (Dense)                (None, 1)            129         dropout_5[0][0]
==================================================================================================
Total params: 41,774,157
Trainable params: 427,257
Non-trainable params: 41,346,900
__________________________________________________________________________________________________
```

Figure 5: The architecture of the best-performed LSTM model.

Hub)[2]. This TF Hub model uses the implementation of BERT from the TensorFlow Models repository on GitHub [5]. It uses L=4 Transformer blocks. Each block has a size of H=512, and A=8 attention heads. This model has been pre-trained for English on the Wikipedia and BooksCorpus. Text inputs have been normalized the "uncased" way, meaning that the text has been lower-cased before tokenization into word pieces, and any accent markers have been stripped. For training, random input masking has been applied independently to word pieces.

The SavedModel provides a trainable .mlm subobject with predictions for the Masked Language Model task it was originally trained with. Based on that, we added a simple classifier as the output layer, which is dense layer with a sigmoid activation function to do the binary classification. To fine-tune the model, the original text has to be encoded firstly with uncased tokenizer[3], which returns a dictionary with three int32 Tensors as input: input_word_ids, input_mask, and input_type_ids. Here, the input_word_ids have been randomly masked or altered by the encoder to fine tune the ML training model. Then this input was used to represent the index of each words together with its sequence in the sentence. The input was fed into the new practical model which is a sequential combination of pre-trained BERT model and classifier. The fine-tuned model, as shown by the Figure.6, includes 28,764,161 trainable parameters.

After training the model for 14 episodes, the validation accuracy tend to con-



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None,)]            0
__________________________________________________________________________________________________
keras_layer_1 (KerasLayer)      {'input_type_ids': ( 0           input_1[0][0]
__________________________________________________________________________________________________
keras_layer (KerasLayer)        {'encoder_outputs':  28763649    keras_layer_1[0][0]
                                                                 keras_layer_1[0][1]
                                                                 keras_layer_1[0][2]
__________________________________________________________________________________________________
dense (Dense)                   (None, 1)            513         keras_layer[0][5]
==================================================================================================
Total params: 28,764,162
Trainable params: 28,764,161
Non-trainable params: 1
__________________________________________________________________________________________________
```

Figure 6: The architecture of starting BERT model.

verge as shown by the learning curve of the last 4 episodes in Figure. 7. The training accuracy (0.97) is a little higher than validation accuracy (0.86), which means that this model is over-fitting. As a result, the test accuracy of this model is [84.9%, 86.8%] with 95% confidence.

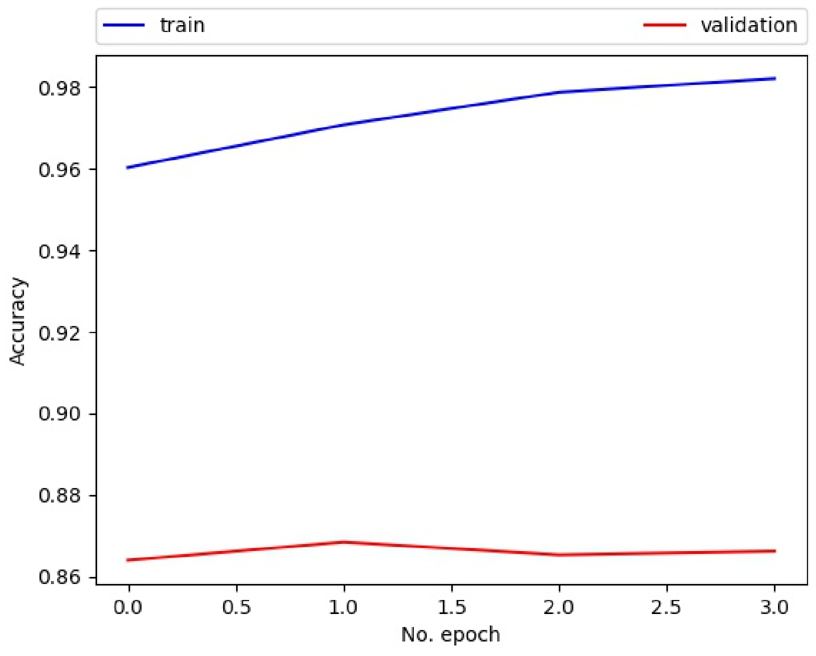
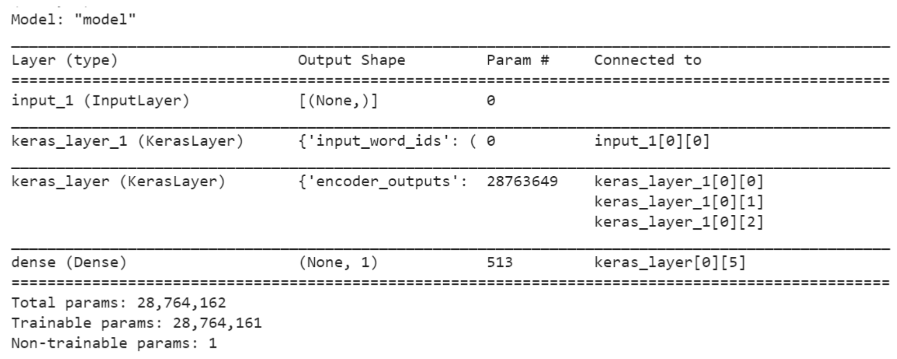


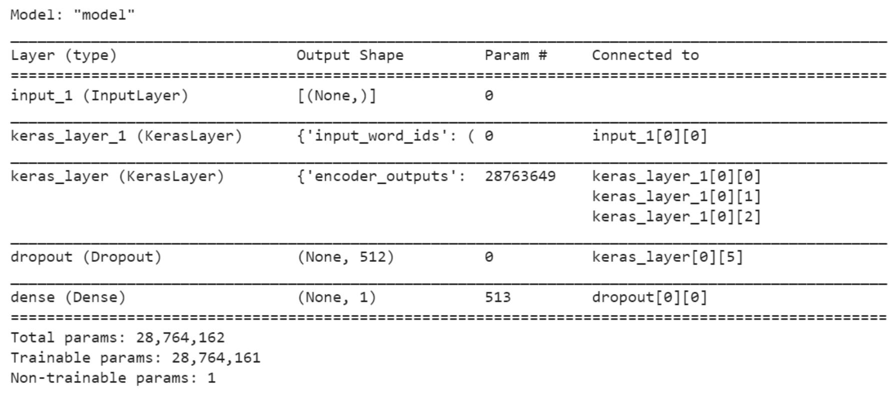
Figure 7: Learning curve of starting BERT model covering the last 4 episodes.

To solve this over-fitting problem, we proposed another 2 model with regularization. In the first architecture we applied L2 regularization in the dense output layer. While for the other architecture, in addition to use L2 regularization in the dense layer, we also added a dropout layer with rate of 0.2 before the dense output layer. Their detailed architecture are shown in Figure.8 and Figure.9 respectively.



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None,)]            0
__________________________________________________________________________________________________
keras_layer_1 (KerasLayer)      {'input_word_ids': ( 0           input_1[0][0]
__________________________________________________________________________________________________
keras_layer (KerasLayer)        {'encoder_outputs':  28763649    keras_layer_1[0][0]
                                                                 keras_layer_1[0][1]
                                                                 keras_layer_1[0][2]
__________________________________________________________________________________________________
dense (Dense)                   (None, 1)            513         keras_layer[0][5]
==================================================================================================
Total params: 28,764,162
Trainable params: 28,764,161
Non-trainable params: 1
```

Figure 8: The architecture of regularized BERT model 1.



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None,)]            0
__________________________________________________________________________________________________
keras_layer_1 (KerasLayer)      {'input_word_ids': ( 0           input_1[0][0]
__________________________________________________________________________________________________
keras_layer (KerasLayer)        {'encoder_outputs':  28763649    keras_layer_1[0][0]
                                                                 keras_layer_1[0][1]
                                                                 keras_layer_1[0][2]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 512)          0           keras_layer[0][5]
__________________________________________________________________________________________________
dense (Dense)                   (None, 1)            513         dropout[0][0]
==================================================================================================
Total params: 28,764,162
Trainable params: 28,764,161
Non-trainable params: 1
```

Figure 9: The architecture of regularized BERT model 2.

4.4 Training model

A shuffled five-fold cross-validation (CV) was employed to train the models. The mean accuracy from the five folds was output as a metric to ensure the robustness of the trained models. The number of training epoch is set as 30 for each fold of cross-validation. Early stopping is used by monitoring the validation accuracy. When the validation accuracy has no more improvement than 0.0001, the model will stop training to avoid over-fitting.

4.5 Performance measurement

After training the model, we calculated the binary categorical accuracy together with 95% confidential interval (CI) on test set to measure the performance since the class labels will be one-hot vectors. The CI was applied by Equation (1) with an assumption of the error following the Gaussian distribution, where p is the test accuracy. Accuracy is a percent value of right classifying instants in total predictions. Moreover, the confusion matrix was calculated in a classification problem, providing more comprehensive understandings of whether the model mis-classified the sentiment or not. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class.

5 Experimental Results

5.1 LSTM

Two primary architectures with different hyperparameter settings were trained and tested in the homework. Table 1 shows a summary of the models' performances, where the pre-trained means whether the model used the pre-trained GloVe embedding data, and the pooling stands for the global maximum pooling layer. The accuracy was calculated based on the test data set. Overall, LSTM-6 employed the pre-trained word-embedding model, and the global maximum pooling layer had the best performance. With 95% confidence, the mean accuracy from the five-fold CV was from 82.57% to 88.67%, and the accuracy on the test data set was between 82.30% and 84.4%.

Table 1: A summary of LSTM-based models' performances

Label	Architecture	Pre-trained	Regularization	5-fold CV	Accuracy
LSTM-1	Single	False	Dropout	73.89%	76.02%
LSTM-2	Single	False	Dropout Pooling	74.81%	69.98%
LSTM-3	Single	True	Dropout Pooling	74.71%	77.40%
LSTM-4	Stacked	True	Dropout	84.58%	83.12%
LSTM-5	Stacked	False	Dropout Pooling	72.72%	72.98%
LSTM-6	Stacked	True	Dropout Pooling	85.67%	83.34%

As compared to the self-trained word embedding model, the pre-trained GloVe model had significant impacts on both single and stacked-LSTM models. It improved the test accuracy from 72.98% to 83.34% for the stacked-LSTM models (LSTM5, LSTM6), and from 69.98% to 77.40% for the single-LSTM models (LSTM-2, LSTM-3). Moreover, the global maximum pooling layer also slightly improved the model's performances (LSTM-1&LSTM-2, LSTM-4&LSTM-6), however, not as significantly as the pre-trained embedding model.

Figure 10 shows the learning curve of the last six steps in the training process for the best-performed LSTM model based on the accuracy. The training accuracy consistently increased to about 0.975, whereas the validation accuracy fluctuated between 0.8 to 0.825. The figure also indicates the model had a minor over-fitting issue. The curve of cross-entropy loss was plotted in Figure 11, which indicates a similar trend as the learning curve. However, in the loss curve, the loss on validation data set even slightly rose from 0.5 to 0.8 in the last six epochs, which also reveals that model was over-fitting. Therefore, in the future, we could add layer and batch normalization to address the issues. Figure 12 is the confusion matrix of the best model. Overall, the model performed slightly better on identifying positive reviews with accuracy at 84.55% as compared to negative reviews (82.09%).

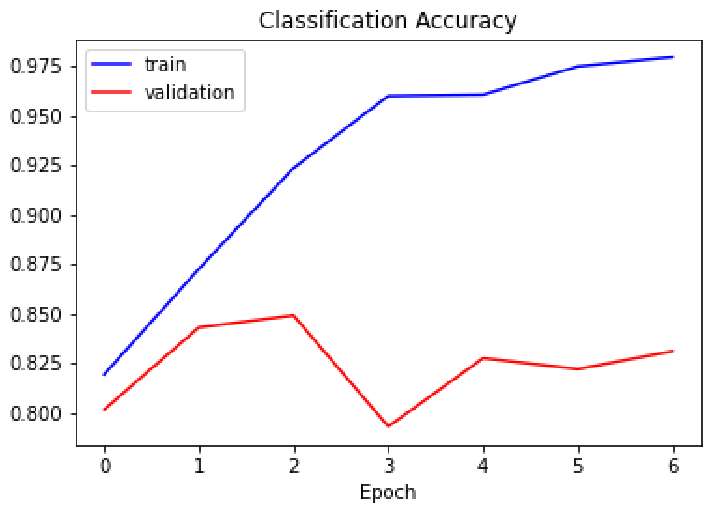


Figure 10: The learning curve of the stacked LSTM model with pre-trained word embedding.

5.2 BERT

The test and validation accuracy with 95% confidence of different BERT architectures are shown in Table. 2. Both architectures reached decent test accuracy about 85% and 4-fold cross-validation accuracy around 90%. The architecture BERT-2 presents a little better test accuracy (85.9%) than that of BERT-1 (85.0%), resulting from the regularization of the dropout layer. However, its effect is not apparent. Besides, we can also get the evidence that both BERT architectures are still greatly over-fitting from the learning curves shown in Figure. 13 and Figure. 14. In both learning

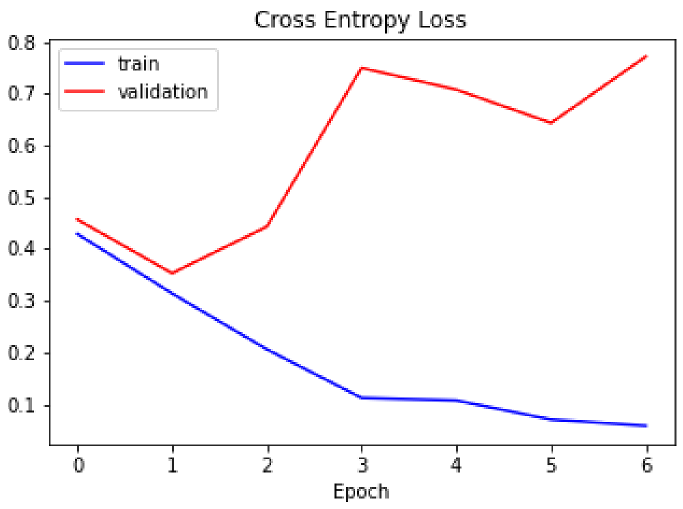


Figure 11: The loss curve of the stacked LSTM model with pre-trained word embedding.

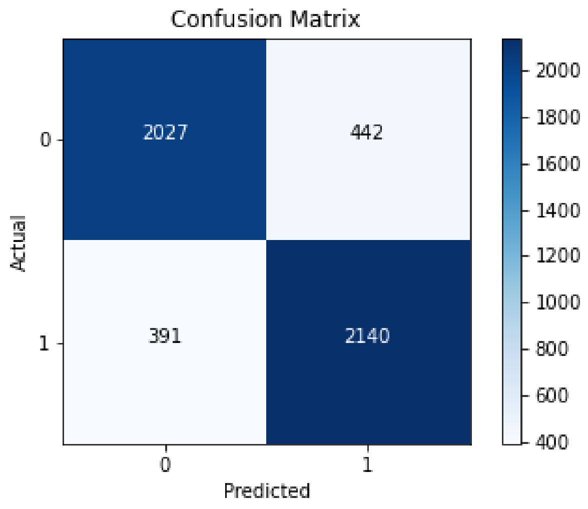


Figure 12: The confusion matrix of the stacked LSTM model with pre-trained word embedding.

curves, the validation accuracy is much higher than training accuracy. Therefore, we need to add more processes of regularization to improve the BERT models. Figure 15 and 16 are the confusion matrices for BERT-1 and BERT-2 respectively. BERT-1 had a better performance in classifying negative reviews, and BERT-2 performed better on the positive reviews.

Table 2: The performances of different BERT architectures

Label	Architecture	Regularization	Attention	Accuracy	5-fold CV
BERT-1	BERT	L2	self-attention	[84.0%, 86.0%]	[87.58%, 96.36%]
BERT-2	BERT	L2 Dropout	self-attention	[84.9%, 86.8%]	[87.74% , 96.36%]

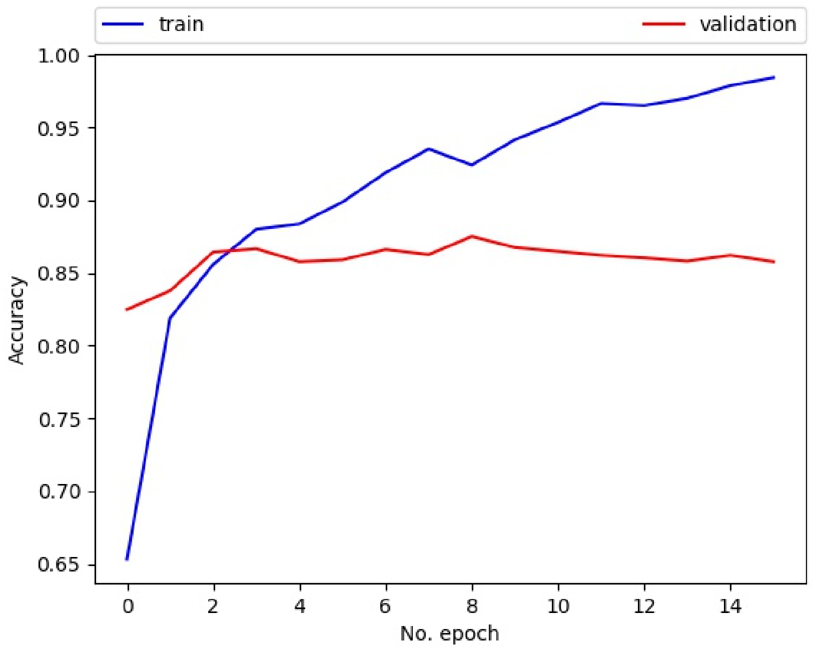


Figure 13: The learning curve of regularized BERT-1 model.

6 Discussion

6.1 Pre-trained vs. self-trained embedding model

In this experiment, we built two models with pre-trained embedding models or without. As we mentioned in the introduction, the pre-trained embedding model

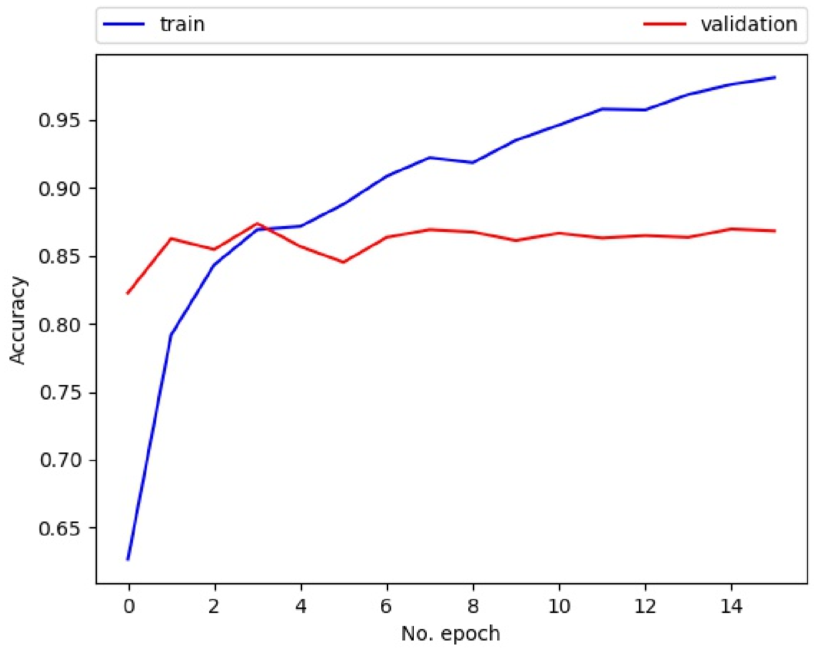


Figure 14: The learning curve of regularized BERT-2 model.

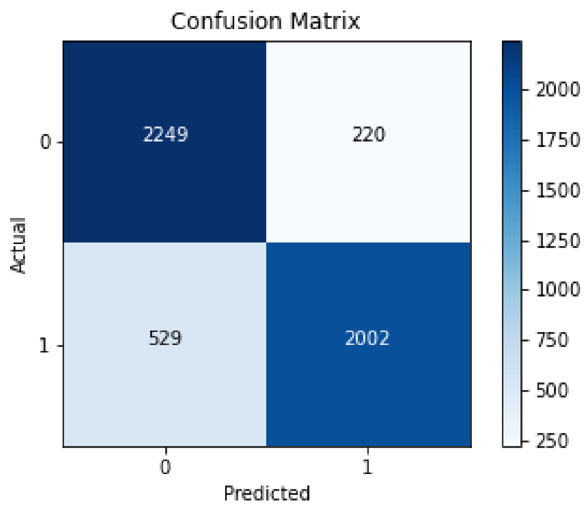


Figure 15: The confusion matrix of regularized BERT-2 model.

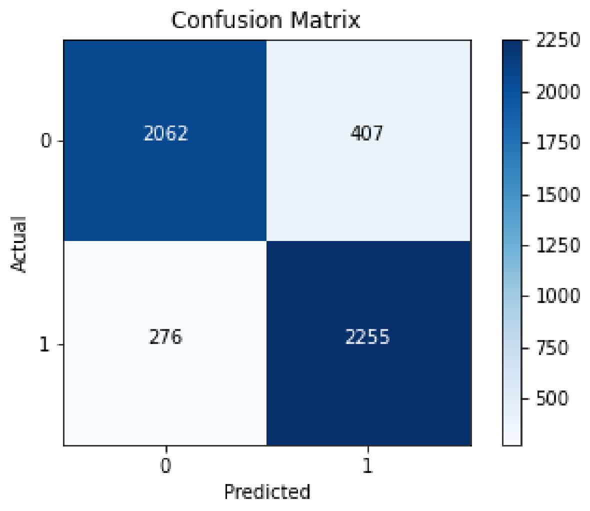


Figure 16: The confusion matrix of regularized BERT-2 model.

turns words into the form of a real-valued vector that encodes the meaning of the word. The words that are closer in the vector space are expected to be similar in meaning. This is a more human-like way of learning, and the result also showed great improvement when we applied a pre-trained embedding model.

6.2 Regularization

In order to avoid the over-fitting problem, we applied different regularization strategies. And we even compared the results with or without some regularization. The results showed that those regularizations improved the model performance.

1. **Dropout** In our case, we also fine-tuned different dropout ratios (not shown in this report), where we found the 0.3 is the best. The dropout strategy is the most common regularization.

2. **1D GlobalMaxpooling** Here 1D GlobalMaxpooling improved the LSTM results. This strategy also reduce the shape of the data which will decrease the over-fitting problem and thus improve the model performance. GlobalMaxpooling is widely used in text classification problems.

However, We can still find the over-fitting problems more or less in all the models we built. Many reasons are leading to this problem. The model may contain more parameters than can be justified by the data, or the over-training can also lead to the over-fitting. In order to get even better accuracy, we can further test the different regularization strategies or variable parameters in the existing regularizations.

6.3 Single LSTM vs stacked LSTM

In our experiment, we tested the single LSTM Model and the stacked LSTM model. The result shows that the stacked LSTM improves the model performance. Stacking LSTM hidden layers makes the model deeper, more accurately earning the description as a deep learning technique. The additional hidden layers are understood to recombine the learned representation from prior layers and create new representation at high levels of abstraction. Increasing the depth of the network also provides an alternative solution for fewer neurons and fast training. This is one type of representational optimization.

However, it is not the case that increasing the depth will improve the performance. The more hidden layers there is the further distance between the representation and the original data it is. This will lead to misrepresentation. So fine-tuning the number of hidden layers is also very essential. And here, we applied the three-stacked LSTM model, which gives us the best model.

6.4 Pre-trained LSTM vs BERT

Overall, the BERT-based models had a slightly better performance than the LSTM-based models, and the pre-trained models were significantly better than self-trained models. These results fit our expectations. The sizes of the given data sets were relatively small in the field of NLP, which made it challenging for models to learn the meaning of texts and perform well on test data. And as compared to the BERT models, the stacked LSTM architectures applied in this homework only preserved information of the past because only inputs to the unit were from the previous unit. Besides, the BERT-based models employed transformers rather than recurrent neurons. Several past studies have shown that BERT had some outstanding performance on many NLP tasks [7, 17]. In this homework, although we applied the GloVe embedding model to perform the transfer learning on the stacked LSTM architectures, the GloVe model did not consider the context when the word embedding was developed. It means the word vectors cannot be adapted to the context in the applied text or document. However, the BERT models can address this limitation by accounting for the contexts of both before and after the words. Therefore, it is reasonable that BERT-based models had better performances. But, during the training process, we found it cost more time and resources to train BERT models than the stacked LSTM models with the pre-trained embedding.

7 Conclusions

In this homework, we trained, tested, and compared LSTM and BERT models with various sets of hyperparameters. We found the pre-trained models (LSTM with GloVe embedding, BERT) were significantly better than the self-trained models based on the given data because of the size of training sets. Besides, the BERT models performed slightly better than the pre-trained stacked LSTM models. BERT's primary properties determine that it can learn the meaning of words from the context in the text or document, which is impossible for the LSTM models with the word-embedding model. However, both the best BERT and LSTM models faced overfitting issues on the test data. And the accuracy of the best BERT model on the test data set is still low compared to other models based on BERT [4]. Therefore, in future studies, we would train and test BERT models with more complex architectures with a larger set of efficient regularization.

References

[1] URL: `https://https://www.imdb.com/interfaces`.

[2] URL: `https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1`.

[3] URL: `https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3`.

[4] Papers with code - imdb benchmark (sentiment analysis). URL: `https://paperswithcode.com/sota/sentiment-analysis-on-imdb`.

[5] Chen Chen, Xianzhi Du, Le Hou, Jaeyoun Kim, Pengchong Jin, Jing Li, Yeqing Li, Abdullah Rashwan, and Hongkun Yu. Tensorflow official model garden, 2020. URL: `https://github.com/tensorflow/models/tree/master/official`.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Aysu Ezen-Can. A comparison of lstm and bert for small corpus. *arXiv preprint arXiv:2009.05451*, 2020.

[8] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *Proceedings of the 17th International Conference on Artificial Neural Networks*, ICANN'07, page 220–229, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] Zhengjie Gao, Ao Feng, Xinyu Song, and Xi Wu. Target-dependent sentiment classification with bert. *IEEE Access*, 7:154290–154299, 2019.

[10] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[11] Jeffrey Pennington. URL: `https://nlp.stanford.edu/projects/glove/`.

[12] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[13] Heereen Shim, Stijn Luca, Dietwig Lowet, and Bart Vanrumste. Data augmentation and semi-supervised learning for deep neural networks-based text classifier. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 1119–1126, 2020.

[14] Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. Portuguese named entity recognition using bert-crf. *arXiv preprint arXiv:1909.10649*, 2019.

[15] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*, 2019.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[17] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.