NLP classification using Recurrent Neural Networks

Student ID: 230161210

Student name: Qing Zhao

1. Recurrent Neural Networks Review

Recurrent Neural Networks or RNNs are a family of neural networks for processing sequential data. Unlike traditional feedforward neural networks, RNNs can process information based on its context, allowing them to learn from past inputs and use that information to influence the processing of current and future inputs.

RNNs introduce a concept called a "hidden state." This hidden state is a vector that captures information about the past inputs that the network has processed. As the network processes each new element in the sequence, the hidden state is updated to reflect the current input and the context from previous inputs.

Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are two variants of RNNs which are designed to address the vanishing and exploding gradient problems typically in standard RNNs.

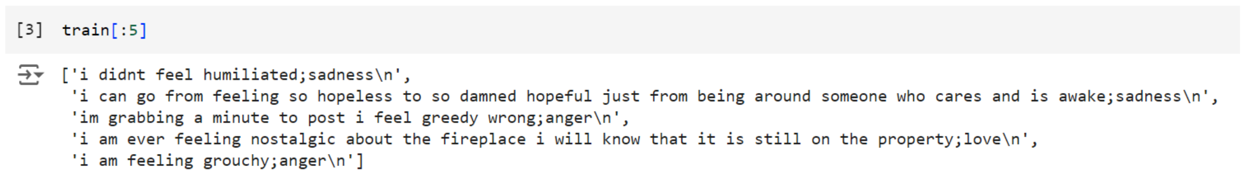
2. Problems Formulation

RNNs excel in tasks like machine translation, sentiment analysis, text generation, and speech recognition where understanding the sequence of words is crucial. In the report I will build a LSTM model to sovle a NLP emotion recognition tasks. Meanwhile, I will build a CNN model to compare with it. Finally, choosing the best model to make predictions on the test dataset.

3. Data Preprocessing

3.1 Exploring Data

The dataset is a collection of 20000 documents with their emotion flag, splitting into train(16000), test(2000) and validation(2000) dataset. The examples of sample is as followed:



```
[3]  train[:5]

['i didnt feel humiliated;sadness\n',
 'i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake;sadness\n',
 'im grabbing a minute to post i feel greedy wrong;anger\n',
 'i am ever feeling nostalgic about the fireplace i will know that it is still on the property;love\n',
 'i am feeling grouchy;anger\n']
```

Figure 1: examples of NLP sample

There are six classes of emotions:joy(33.5%), sadness(29.2%), anger(13.5%), fear(12.1%), love(8.2%) and surprise(3.6%). The figure 2 shows the distribution of classes.

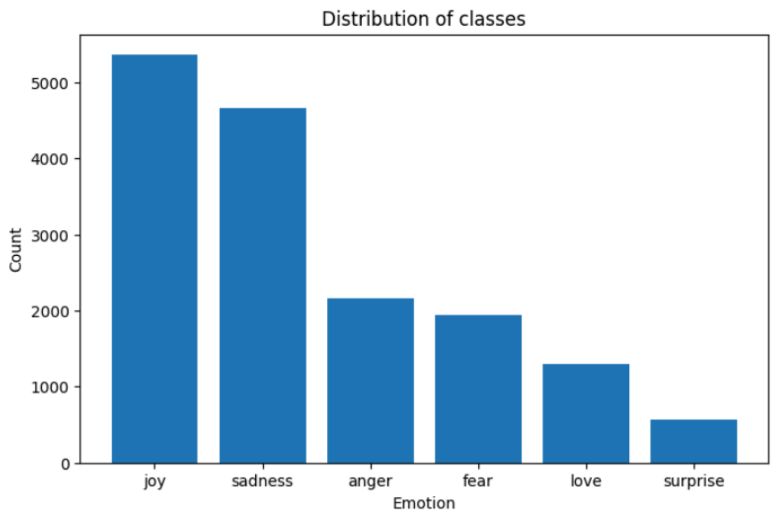
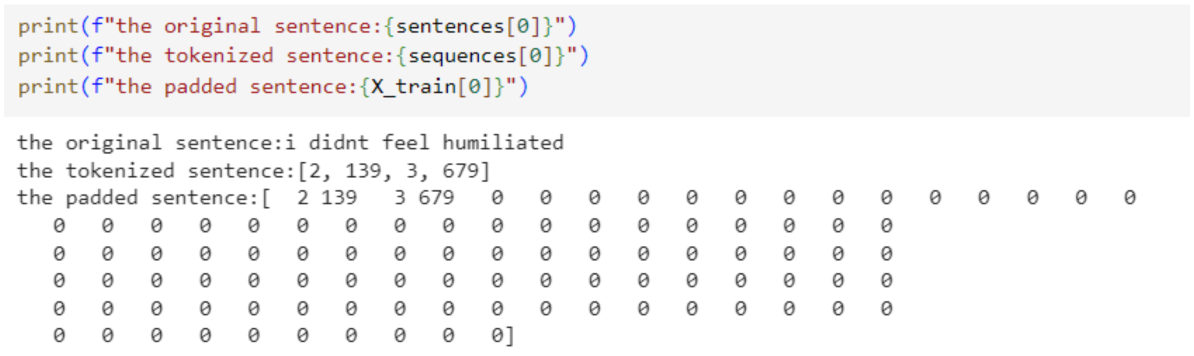


Figure 2: distribution of emotion classes

3.2 Tokenizing and Padding Sequences

In most NLP tasks, the initial step in preparing data is to extract a vocabulary of words from input texts. We need to define how to represent the texts into numerical representations which can be used to train a neural network. These representations are called tokens. In this task, I use a built-in api in tensorflow.keras, Tokenizer, to generate a word_index dictionary and convert the input sentences into a sequence of tokens.

Then in order to make sure the input numerical sequences have the same length which is required as the input for a neural network, I applied padding technique to these numerical sequences. In this experiment, I set the max_sequences_length is 100, which means that all sentences are converted to a list representation of 100 token lengths, regardless of the original length of the sentence, truncated if the original length exceeds 100, and filled with zeros if the original length is less than 100 .



```
print(f"the original sentence:{sentences[0]}")
print(f"the tokenized sentence:{sequences[0]}")
print(f"the padded sentence:{X_train[0]}")
```

```
the original sentence:i didnt feel humiliated
the tokenized sentence:[2, 139, 3, 679]
the padded sentence:[  2 139   3 679   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

Figure 3 : the transfer process from tokenizer to padding.

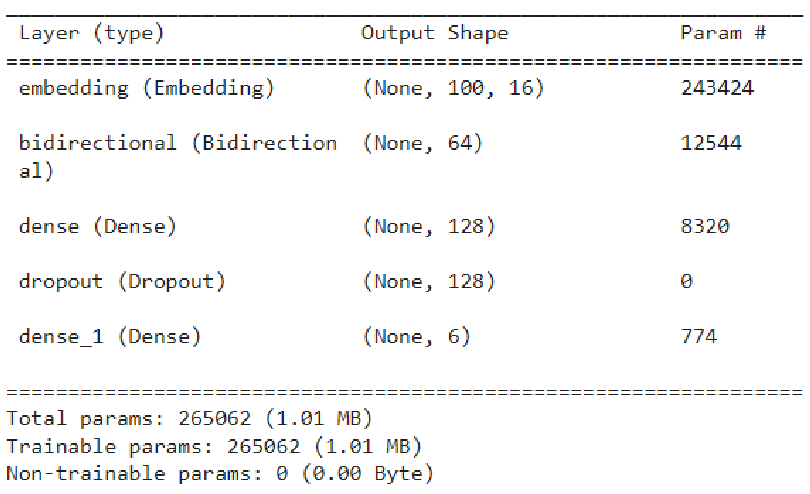
Using the same word_index dictionary generated from the training dataset, samples in the test and validation dataset are tokenized as well.

3.3 Encoding Labels

Since the emotion labels are also text, I use LabelEncoder class from sklearn library to encoder the labels, which is commonly used in converting categorical variables into numerical labels.

4. Building a LSTM model for NLP classification

Firstly, I built a bidirectional single LSTM model. The first layer is an Embedding layer which acts as representing all the words in the word_index dictionary with vectors. These vectors have trainable weights so as the neural network learns, words that are most likely to appear in a positive tweet will converge towards similar weights. Similarly, words in negative tweets will be clustered more closely together. In this experiment, I set the dimensions of vectors to be 100.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           243424

 bidirectional (Bidirection  (None, 64)                12544
 al)

 dense (Dense)               (None, 128)               8320

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 265062 (1.01 MB)
Trainable params: 265062 (1.01 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 4: the summary architecture of the RNNs model

After the embedding layer, I added a LSTM layer, nesting it inside a bidirectional layer so the passing of the sequence information goes both forwards and backwards. Although I tried to stack more LSTM layers to make a deeper model, the additional computations naturally make the training go slower. After comprehensively evaluating the operational efficiency and model effect, I finally chose the single layer of LSTM model. Figure 4 and 5 demonstrates the model architecture and its training results.

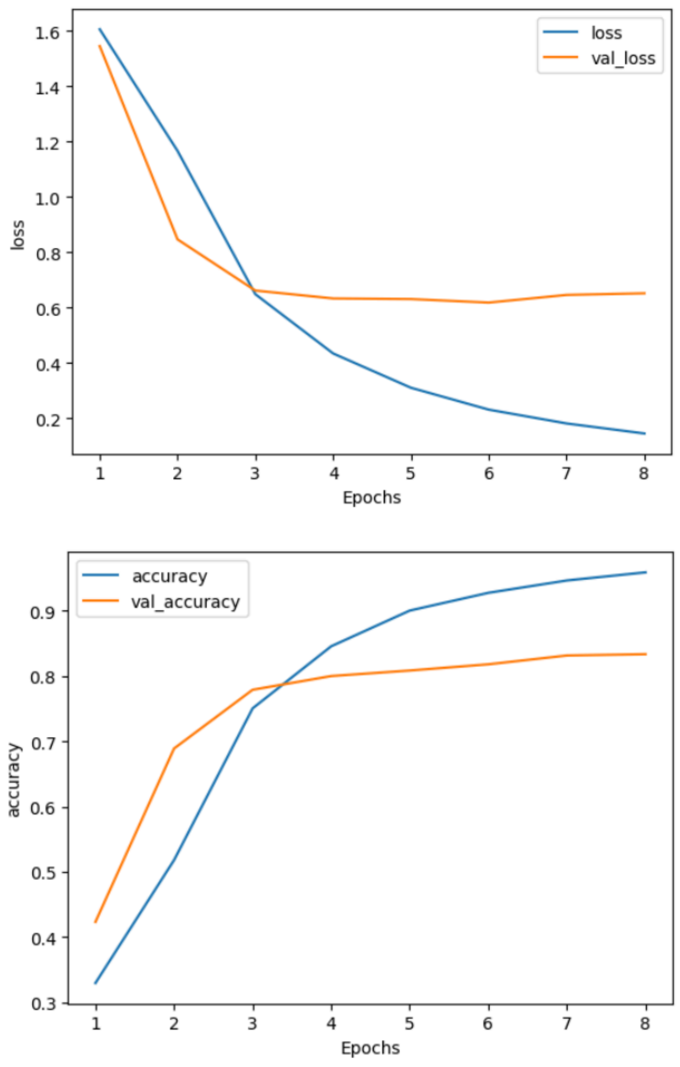


Figure 5: the loss and accuracy of the LSTM model

From this diagram, we can see that although the training accuracy is close to 95%, the validation accuracy is much lower, only 83%, and the accuracy growth range in the validation dataset is much lower than in the training dataset.

4. Building a CNN model for the same task

Secondly, I tried to build a CNN model to tackle this task, just simply swap the bidirectional LSTM layer with a convolutional layer and a globalmaxpooling layer. In the convolutional layer, I selected 64 4-sized filter to extract features from input. From the below diagram, it is clear that the CNN model is better than the LSTM model mentioned above, with 96% training accuracy and 91% validation accuracy. Meanwhile this model did better to avoid overfitting than the LSTM model.

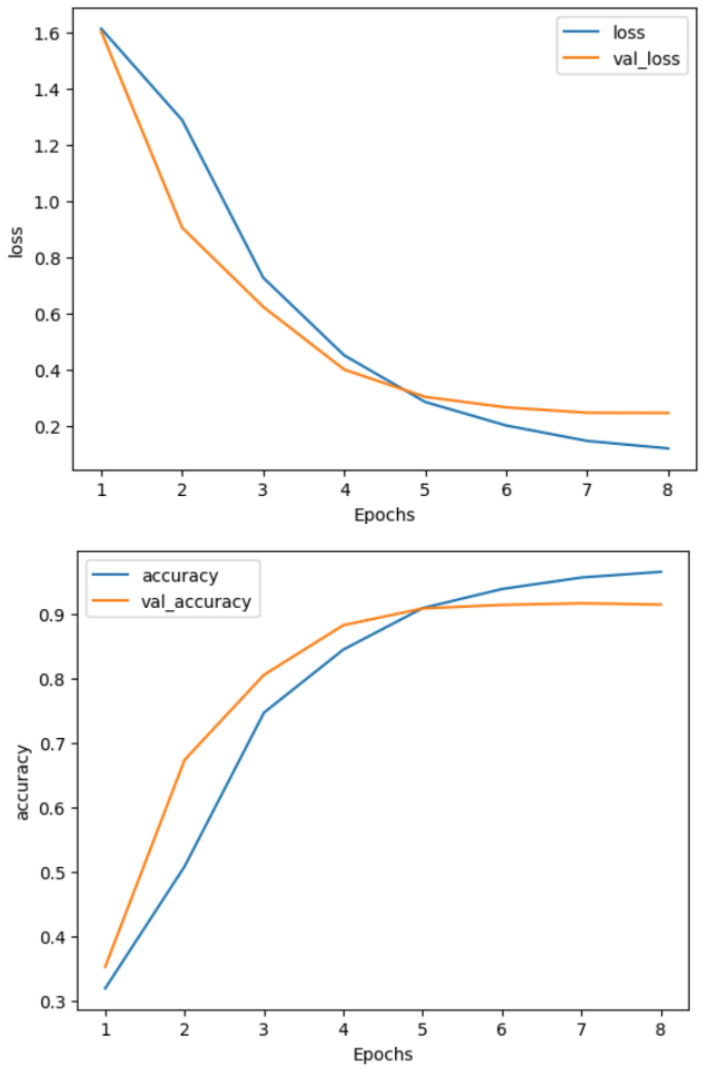


Figure 6: the loss and accuracy of the CNN model

5. Model Prediction

Using the CNN model to predict the testing data. The accuracy is 91%, which remains a high and stable performance. And from the confusion matrix, we can tell that misclassifying “love” as” joy” and “surprise” as “fear” is a common mistake this model made. Considerig the proportions of these two classes of emotion samples are the lowest, especially the proportion of “surprise” accounted for only 3.6%, It is explicable that the model does not learn enough about the features of these two classes.

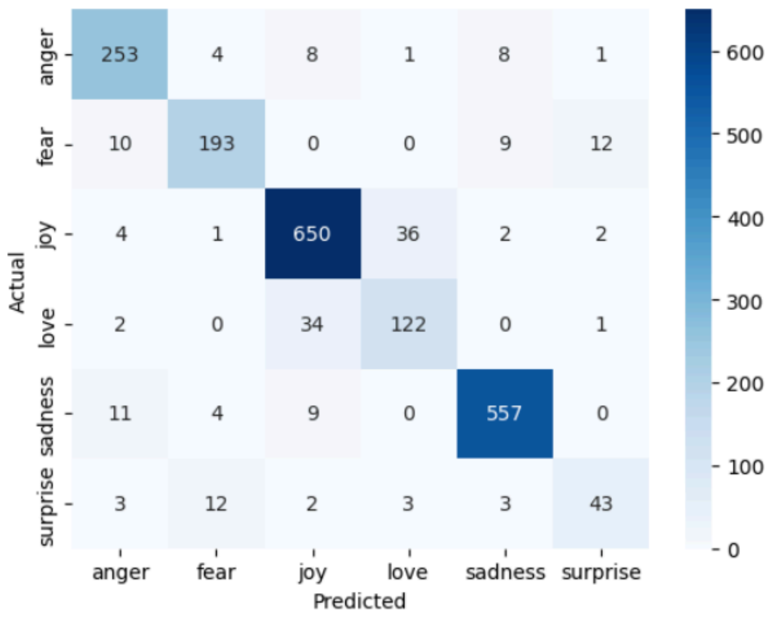


Figure 7: the confusion matrix of the testing dataset

References

1. course video: MIT S191: Recurrent Neural Networks
2. course video: Natural Language Processing in TensorFlow
3. textbook: Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
4. website: https://en.wikipedia.org/wiki/Recurrent_neural_network
5. dataset: https://www.kaggle.com/datasets/praveengovi/emotions-dataset-for-nlp