VIETNAMESE SENTIMENT ANALYSIS

1. Introduction

Sentiment analysis is a natural language processing technique used to determine whether data is positive, negative, or neutral. Sentiment analysis is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback, and understand customer needs.

To perform sentiment analysis on a complex language such as Vietnamese, our model would need to employ a suitable cleaning pipeline as well as a sophisticated machine learning model that can capture the meaning of the dataset.

2. Data preprocessing

The dataset of this experiment contains 2 sets, the training set contains 5100 rows of Vietnamese texts and their labels, and the testing set contains 1050 rows.

Since this experiment concerns Vietnamese texts, a few preprocessing steps will be needed to ensure the highest model performance.

First, we will encode the labels with one-hot encoding to use the softmax activation function in the output layer. Since there are 3 classes, 1 for positive, 0 for neutral, and -1 for negative, we will encode 1 with [1, 0, 0], 0 with [0, 1, 0], and -1 with [0, 0, 1].

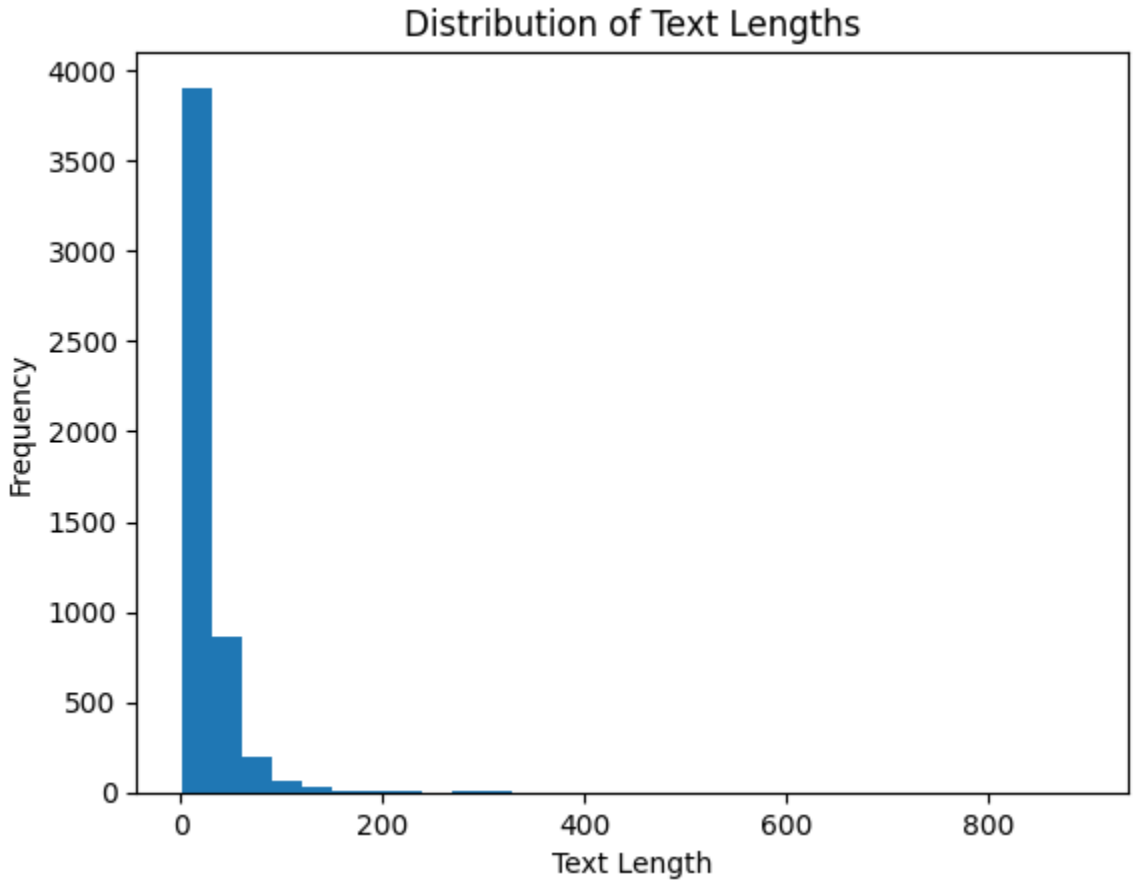
Then, we will put the texts through a cleaning pipeline which consists of:

- Removing HTML elements in the text if available.
- Converting Windows-1252 characters to Unicode UTF-8 characters.
- Normalizing tones such as òa to oà so that words like hoà and hoài have the same oà spelling.
- Removing special characters such as punctuations, numbers, and two or more spaces from the texts.

Additionally, the removal of stopwords from the text had also been tested but the final results showed that such action only lowered the model accuracy so we did not remove stopwords from the texts.

After being cleaned, we will tokenize and convert the texts into sequences of the same length before computing the word vector. To determine the length of each sequence, we will take into account 2 factions.

First is the frequency of the length of each sentence. According to this graph generated from the dataset, most sentences tend to be shorter than 400 characters.



Second is the max text length specified by the pre-trained word embedding model. In the case of SBERT, which is one of the word embedding methods, the max text length is 256.

Combining two factors, we decided that 256 would be the length of each sequence.

3. Word embedding

3.1. Word2Vec

The first word embedding model we experimented with is the Word2Vec pre-trained on Vietnamese texts. This model takes in each word in our vocabulary and returns a vector of size 300, in case the word in our vocabulary is not in the model, we will compute and fill a vector of size 300 with random values. The combination of those vectors forms our embedding matrix.

3.2. Vietnamese SBERT

Alternatively, we also experimented with an SBERT model pre-trained on Vietnamese texts to see how the model performed on another word embedding model. This

Vietnamese SBERT model takes in each word in our vocabulary and returns a vector of size 768. The combination of those vectors forms our embedding matrix.

4. Model architecture

The embedding matrix served as the embedding layer before entering the models. The model architectures of this experiment were Convolutional Neural Networks (CNN), Long Short-term Memory Neural Networks (LSTM), and Convolutional Long Short-term Memory Neural Networks (CNN-LSTM).

4.1. Convolutional Neural Networks

```
Model: "cnn_w2v"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_3 (InputLayer)        [(None, 256)]             0         []

 embedding_2 (Embedding)     (None, 256, 400)          3160400   ['input_3[0][0]']

 conv1d_6 (Conv1D)           (None, 254, 100)          120100    ['embedding_2[0][0]']

 conv1d_7 (Conv1D)           (None, 253, 100)          160100    ['embedding_2[0][0]']

 conv1d_8 (Conv1D)           (None, 252, 100)          200100    ['embedding_2[0][0]']

 max_pooling1d_6 (MaxPoolin  (None, 1, 100)            0         ['conv1d_6[0][0]']
 g1D)

 max_pooling1d_7 (MaxPoolin  (None, 1, 100)            0         ['conv1d_7[0][0]']
 g1D)

 max_pooling1d_8 (MaxPoolin  (None, 1, 100)            0         ['conv1d_8[0][0]']
 g1D)

 concatenate_4 (Concatenate  (None, 3, 100)            0         ['max_pooling1d_6[0][0]',
 )                                                                 'max_pooling1d_7[0][0]',
                                                                   'max_pooling1d_8[0][0]']

 attention_2 (Attention)     (None, 3, 100)            0         ['concatenate_4[0][0]',
                                                                   'concatenate_4[0][0]']

 concatenate_5 (Concatenate  (None, 3, 200)            0         ['concatenate_4[0][0]',
 )                                                                 'attention_2[0][0]']

 flatten (Flatten)           (None, 600)               0         ['concatenate_5[0][0]']

 dropout_2 (Dropout)         (None, 600)               0         ['flatten[0][0]']

 dense_4 (Dense)             (None, 128)               76928     ['dropout_2[0][0]']

 dense_5 (Dense)             (None, 3)                 387       ['dense_4[0][0]']

==================================================================================================
Total params: 3718015 (14.18 MB)
Trainable params: 3718015 (14.18 MB)
Non-trainable params: 0 (0.00 Byte)
```

The overall architecture of the CNN was the same as the CNN in lab 5:

- 3 conv layers, each using the ReLU activation function and containing 100 filters of size 3x3, 4x4, and 5x5 respectively.
- 3 1D max-pooling layers of stride 1 and window sizes of 254, 253, and 252 respectively.
- 1 output layer of 3 classes using the softmax activation function.
- L2 regularization with lambda being 0.01 and a dropout rate of 0.5 to prevent overfitting.
- Adam optimizer with a learning rate of 0.001

Additionally, we also made a few changes that would improve the model performance:

- Adding an attention layer after the max-pooling layers, a key mechanism of the Transformers architecture, to allow the model to dynamically weigh different words or phrases in the input text, giving more attention to those that are likely to be more indicative of the sentiment expressed.
- Adding another dense layer of size 128 with the ReLU activation layer right before the output layer.

4.2. Long Short-term Memory Neural Networks

```
Model: "lstm_w2v"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_4 (InputLayer)        [(None, 256)]             0         []

 embedding_3 (Embedding)     (None, 256, 400)          3160400   ['input_4[0][0]']

 lstm_2 (LSTM)               (None, 128)               270848    ['embedding_3[0][0]']

 attention_3 (Attention)     (None, 128)               0         ['lstm_2[0][0]',
                                                                  'lstm_2[0][0]']

 concatenate_6 (Concatenate  (None, 256)               0         ['lstm_2[0][0]',
 )                                                                'attention_3[0][0]']

 dropout_3 (Dropout)         (None, 256)               0         ['concatenate_6[0][0]']

 dense_6 (Dense)             (None, 128)               32896     ['dropout_3[0][0]']

 dense_7 (Dense)             (None, 3)                 387       ['dense_6[0][0]']

=================================================================================================
Total params: 3464531 (13.22 MB)
Trainable params: 3464531 (13.22 MB)
Non-trainable params: 0 (0.00 Byte)
```

The overall architecture of the LSTM was also the same as the LSTM in lab 6:

- 1 LSTM layer of size 128. In our experiment, we concluded that more LSTM layers or bigger LSTM sizes did little to improve the model accuracy but affected the training time a lot. Therefore we only used 1 LSTM layer of size 128.
- 1 output layer of 3 classes using the softmax activation function.
- L2 regularization with lambda being 0.01 and a dropout rate of 0.5 to prevent overfitting.
- Adam optimizer with a learning rate of 0.001

Additionally, we also add an attention layer after the LSTM layer and another dense layer before the output layer.

4.3. Convolutional Long Short-term Memory Neural Networks

```
Model: "cnn_lstm_w2v"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_5 (InputLayer)        [(None, 256)]             0         []

 embedding_4 (Embedding)     (None, 256, 400)          3160400   ['input_5[0][0]']

 reshape_2 (Reshape)         (None, 256, 400)          0         ['embedding_4[0][0]']

 conv1d_9 (Conv1D)           (None, 254, 100)          120100    ['reshape_2[0][0]']

 conv1d_10 (Conv1D)          (None, 253, 100)          160100    ['reshape_2[0][0]']

 conv1d_11 (Conv1D)          (None, 252, 100)          200100    ['reshape_2[0][0]']

 max_pooling1d_9 (MaxPoolin  (None, 1, 100)            0         ['conv1d_9[0][0]']
 g1D)

 max_pooling1d_10 (MaxPooli  (None, 1, 100)            0         ['conv1d_10[0][0]']
 ng1D)

 max_pooling1d_11 (MaxPooli  (None, 1, 100)            0         ['conv1d_11[0][0]']
 ng1D)

 concatenate_7 (Concatenate  (None, 3, 100)            0         ['max_pooling1d_9[0][0]',
 )                                                                 'max_pooling1d_10[0][0]',
                                                                   'max_pooling1d_11[0][0]']

 lstm_3 (LSTM)               (None, 128)               117248    ['concatenate_7[0][0]']

 attention_4 (Attention)     (None, 128)               0         ['lstm_3[0][0]',
                                                                   'lstm_3[0][0]']

 concatenate_8 (Concatenate  (None, 256)               0         ['lstm_3[0][0]',
 )                                                                 'attention_4[0][0]']

 dropout_4 (Dropout)         (None, 256)               0         ['concatenate_8[0][0]']

 dense_8 (Dense)             (None, 128)               32896     ['dropout_4[0][0]']

 dense_9 (Dense)             (None, 3)                 387       ['dense_8[0][0]']

=================================================================================================
Total params: 3791231 (14.46 MB)
Trainable params: 3791231 (14.46 MB)
Non-trainable params: 0 (0.00 Byte)
```

The overall architecture of the CNN_LSTM was the combination of the above CNN and LSTM, with an addition of the attention layer and another dense layer.

5. Model training loop

To set up the training loop, first, we will define the early stopping mechanism monitoring the val_loss with the min_delta of 0.01 and patience of 10. We also set True to restore_best_weights so that the final model would be the one that has the lowest val_loss in all epochs.

Then we randomly split the training data into training and validation data with a ratio of 8:2. The model would be trained on the training set and be cross-validated with the validation set.

Finally, changing between 2 word embedding models and 3 architectures, we trained 6 of our models with 50 epochs and a batch size of 256.

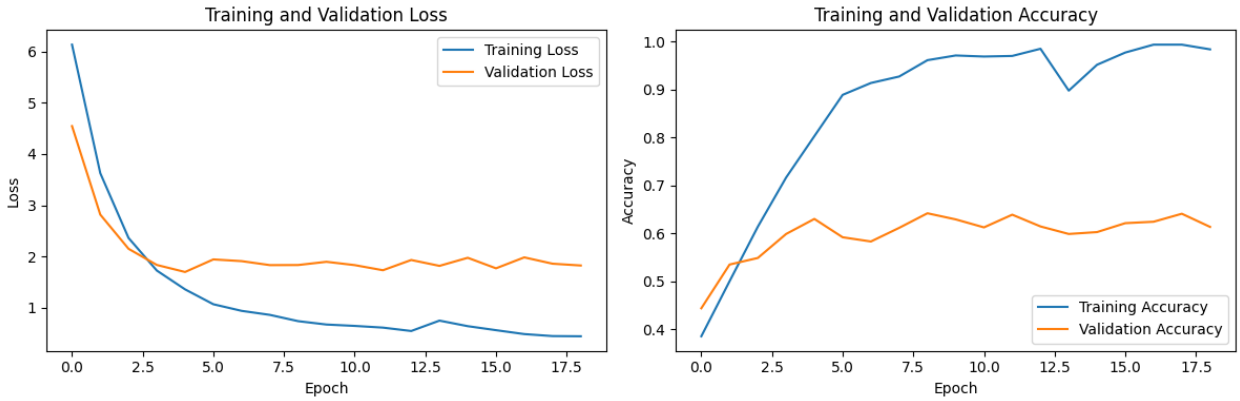
6. Evaluation

	Embedding	Testing loss	Testing acc	Params	Training time (with ES)
CNN	W2V	1.01	70.01	3718015	82s
LSTM	W2V	1.02	67.43	3464531	32s
CNN_LSTM	W2V	1.35	72.38	3791231	46s
CNN	SBERT	1.16	67.05	7067183	101s
LSTM	SBERT	1.04	66.19	6560515	40s
CNN_LSTM	SBERT	1.60	60.95	7140399	56s

All the training and testing were performed on Google Colab with GPU T4.

As we can see, by combining the architecture of CNN and LSTM, using the Word2Vec embedding, our model can achieve an accuracy of 72.38% on the testing set, making it the best-performing model in the list.

Additionally, the additional SBERT models did not seem to perform as well as the Word2Vec models in both accuracy in training time. This may be because the BERT model requires a large training dataset to be effective.



For the CNN_LSTM Word2Vec model, it seemed that the val_loss reached the minimum value at around the 4th, or at most the 10th epoch, without much change afterward, therefore the early stopping mechanism kicked in and made the training stopped at around the 20th epoch, saving the training time and resources.

7. Conclusion

After the experiment, we concluded that the combination of CNN and LSTM did indeed yield a better result in the text sentiment analysis task. Further testing using other optimizing techniques such as batch normalization or Bi-LSTM layers or a better data preprocessing pipeline could improve the results further. Additionally, another word embedding model such as GloVe or ELMo could perform better than Word2Vec.