VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Natural Language Processing (CO3085)

Assignment: Sentiment Analysis

Lecturer: Assoc. Prof. Quan Thanh Tho
Student: Nguyen Hoang Anh Thu - 2053478

HO CHI MINH CITY, NOVEMBER 2023

Contents

1 Introduction

The field of Natural Language Processing (NLP) acknowledges Sentiment Analysis as a cardinal task, yet the methodologies currently employed are in dire need of enhancement. Predominant challenges encompass a high-dimensional data processing requirement, inefficiency in sentiment classification, and insufficient extraction and interpretation of semantic information from text and intricate abstract contexts.

The **Convolutional Neural Network (CNN)** technique, despite its commendable expediency during the training phase, falls short in its capacity to extract beyond localized information and demonstrate contextual correlations. Conversely, the **Long Short-Term Memory (LSTM)** model, whilst proficient in extracting contextual dependencies and performing effective classification, is burdened by prolonged training periods.

In light of these challenges, the objective of this report is to propose and implement **a composite CNN-LSTM methodology** for Sentiment Analysis. In the proposed method, the analysed words will be vectorized before being incorporated into the combined CNN-LSTM model. Subsequently, the Softmax activation function will be utilized for sentiment classification.

2 Preliminary Literature

2.1 Word Embedding

The representation of words and phrases in a way that can be understood and processed by machines is a critical aspect. A common method for attaining this is known as Word Embedding.

Word Embedding is a technique where words or phrases from the vocabulary are mapped to vectors of real numbers in a predefined vector space. Each word or phrase is represented as a fixed-length vector, where the length is determined by the number of dimensions in the selected vector space. The generated vectors carry semantic meanings, allowing words with similar meanings to have similar vector representations.

A traditional form of representing these vectors is the one-hot encoding method. In this method, each word is depicted as a sequential numerical string, which is then portrayed as a vector with dimensions equal to the size of the vocabulary. One of the limitations of this method is the potential for dimensionality redundancy, where the number of dimensions required to represent the vector is excessively high.

To mitigate the issues presented by one-hot encoding, an alternative approach was developed, which represents words based on the probability distribution. This approach is typically used in the context of Deep Learning and is called Distributed Word Vector, more commonly known as word2vec.

The word2vec model is a three-layered neural network structure that includes two models: the Skip-Gram and Continuous Bag of Words (CBOW). The structure consists of an Input layer, a Projection layer, and an Output layer. The model works by mapping words in the text to vectors, thereby effectively converting the textual information into a format that can be processed by machine learning algorithms.

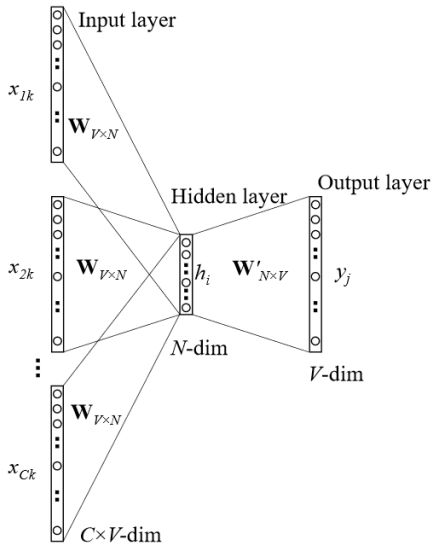

Figure 2: Continuous bag-of-word model

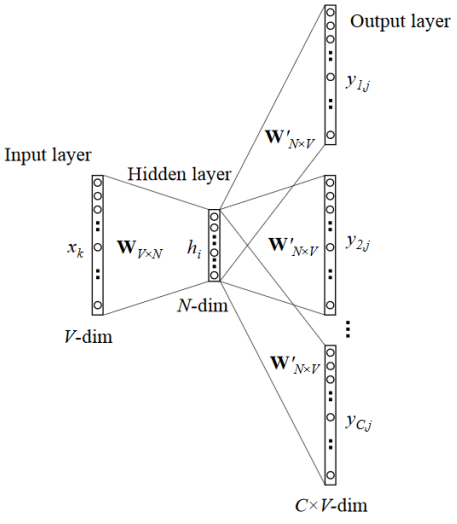

Figure 3: The skip-gram model

2.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) have been widely recognized for their robust performance in image processing tasks. However, their utility extends beyond visual data, and they have proven to be highly effective in the field of Sentiment Analysis as well.

In the context of sentiment analysis, the CNN typically starts with an embedding layer where words are transformed into vector representations. These embeddings can either be pre-trained or learned during training. Following this, multiple convolutional layers and pooling layers are applied. Each convolutional layer uses multiple filters that slide across the input text to compute dot products. This operation enables the model to automatically and adaptively learn spatial hierarchies of features, capturing important local and global semantics of the sentence.

The pooling layers serve to reduce the spatial size of the representation, reducing the computational complexity and helping to prevent overfitting. Max pooling is a commonly used technique, which retains the maximum value from each of the filter's output, providing a fixed-size output regardless of the input length.

Finally, the output from the pooling layers is typically flattened and fed into one or more fully connected layers, with the last layer performing the classification. For sentiment analysis, the final layer would typically use a sigmoid or softmax activation function to output the probabilities of each sentiment class.

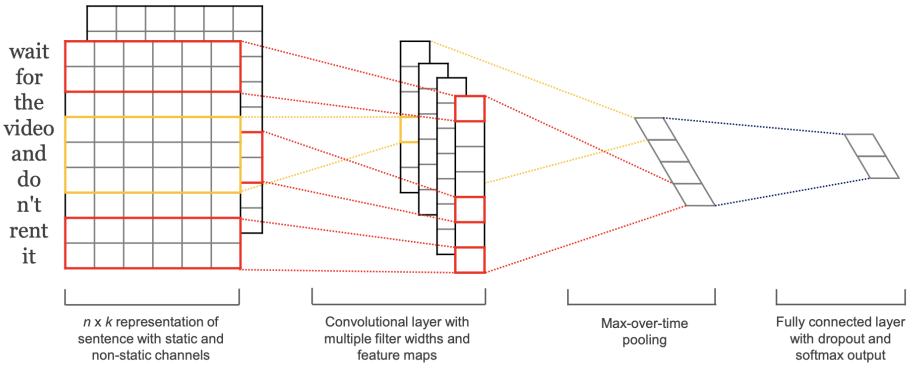

Figure 1: CNN model

2.3 Long-Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) that have gained widespread use in the field of Natural Language Processing (NLP), particularly in tasks such as sentiment analysis.

LSTMs have proven effective in sentiment analysis due to their ability to handle sequence data and their capability to preserve the long-term dependencies in a text sequence. Traditional RNNs, while theoretically capable of handling sequential data and preserving temporal dependencies, in practice suffer from the vanishing gradient problem, which causes them to forget information in longer sequences. LSTMs, on the other hand, use a system of gates to control the flow of infor-

mation and effectively tackle the vanishing gradient problem, enabling them to remember or forget information over longer sequences.

An LSTM network for sentiment analysis typically starts with an embedding layer where words are converted into vector representations. These vector representations can be learned while training the model, or pre-trained word embeddings can be used. The LSTM layer then processes these sequences of vectors, maintaining information about the dependencies between the words in the sequence.

The LSTM layer consists of multiple memory cells that keep track of dependencies in the input sequence. These memory cells include a forget gate, an input gate, and an output gate. The forget gate decides what information to discard from the cell state, the input gate decides what new information to store in the cell state, and the output gate determines what the next hidden state should be.

The output from the LSTM layer(s) is then typically passed to one or more fully connected layers, with the final layer using a softmax activation function for multi-class sentiment classification, or a sigmoid activation function for binary sentiment classification.

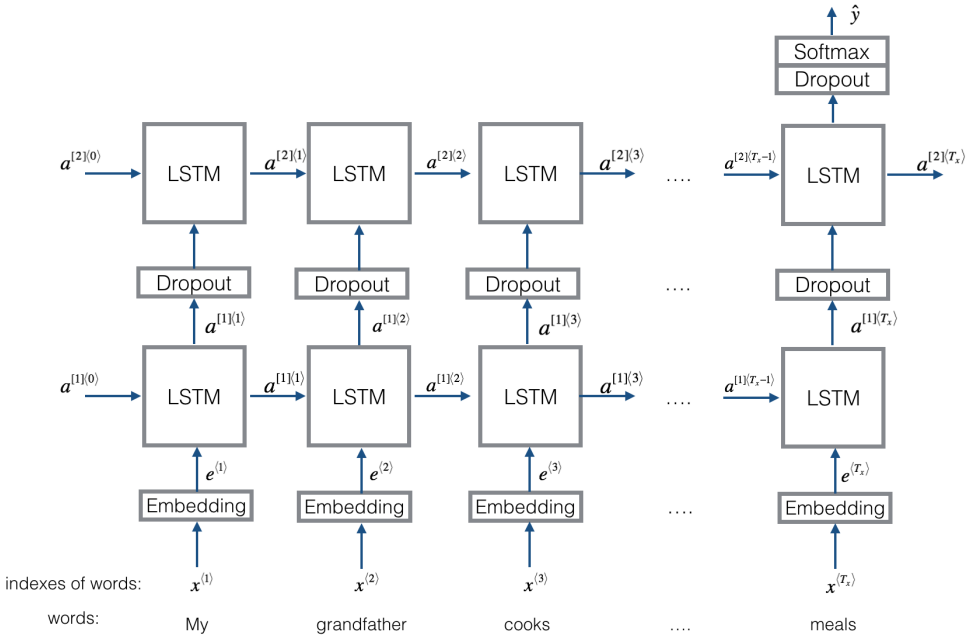

Figure 2: LSTM model

2.4 A combination method of CNN-LSTM

The combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks has emerged as a powerful model for sentiment analysis. This hybrid architecture leverages the strengths of both models, allowing it to extract important features from sequences of words and capture long-term dependencies in text data.

2.4.1 Methodology

CNN-LSTM models typically begin with an embedding layer where each word is mapped to a vector representation. These embeddings can either be learned as part of the model training or can be initialized with pre-trained word embeddings such as Word2Vec or GloVe to incorporate pre-existing linguistic knowledge.

Following the embedding layer, a convolutional layer is applied. The convolutional layer uses multiple filters that slide across the input text, each capturing different local features or patterns in the sequence of word embeddings. This results in a feature map that represents the most salient local features in the input.

A pooling layer is then applied to reduce the dimensionality of the feature map and to extract the most important features. This is usually achieved using Max Pooling, which takes the maximum value of each feature in a window. This not only reduces computational complexity but also provides a form of translation invariance, making the model more robust.

The output from the pooling layer is then fed into an LSTM layer. The LSTM layer is a type of recurrent neural network that can remember and forget information over long sequences, making it ideal for capturing the global context and long-term dependencies between words in a sentence.

The final layer of the CNN-LSTM model is usually a fully connected layer, which takes the high-level features extracted by the LSTM layer and uses them to classify the sentiment. The activation function in the final layer is typically a softmax for multi-class classification or a sigmoid for binary classification.

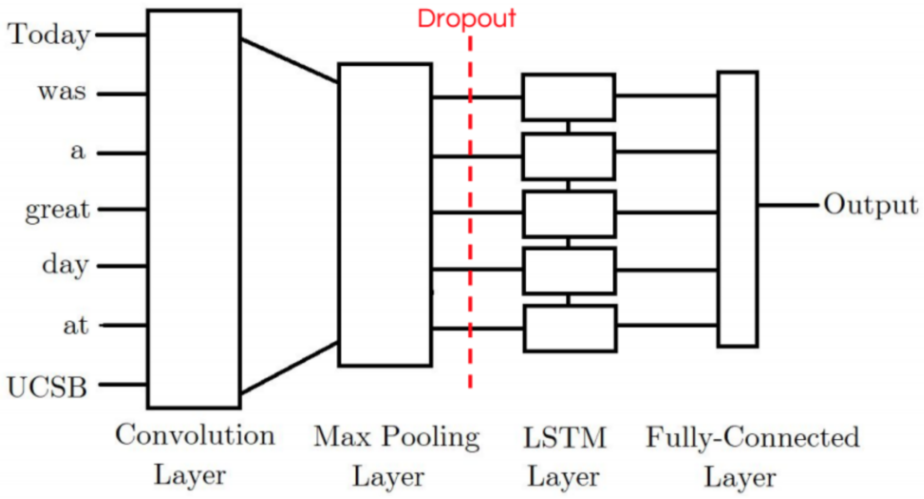

Figure 3: Combination of CNN-LSTM model

2.4.2 Comparison

Features	CNN	LSTM	CNN-LSTM
Feature extraction	Excellent at extracting local and global features and identifying important patterns in the data	Not specifically designed for feature extraction, but can capture patterns over time	Combines the feature extraction capabilities of CNNs with the sequence modeling of LSTMs
Sequence modeling	Not inherently designed for sequence data, although 1D convolutions can be used on sequences	Excellent at handling sequence data and can capture long-term dependencies in the data	Effectively handles sequence data
Handling long-term dependencies	Struggles with capturing long-term dependencies due to lack of memory of past inputs	Excels at capturing long-term dependencies due to its gating mechanisms (forget, input, output gates)	Combines CNN's feature extraction with LSTM's ability to capture long-term dependencies
Robustness to noise and invariance	Robust to noise due to convolution and pooling operations, and provides translation invariance	Less robust to noise and does not inherently provide translation invariance	Inherits the robustness to noise and invariance of CNNs
Overfitting	Can potentially overfit, especially with small datasets, but pooling layers help reduce this risk	Can overfit due to large number of parameters, but this can be mitigated using dropout or other regularization techniques	Generally less prone to overfitting due to combination of feature-dimensionality reduction (CNN) and complex sequence modeling without many parameters (LSTM)
Implementation complexity	Relatively easy to implement and train	More complex to implement and train due to the sequential nature of the model	More complex than individual CNN or LSTM due to the hybrid nature, but most modern deep learning libraries provide support
Performance	Good performance, but can struggle with long sentences and context understanding	Good performance, especially in tasks that require understanding of long-term dependencies	Often outperforms individual CNN or LSTM models in sentiment analysis tasks

Table 1: Comparison between three methods

3 Implementation

3.1 Support packages

3.1.1 Pyvi

Pyvi constitutes a crucial package specifically designed for processing Vietnamese text data. The library is equipped with multiple features that render it pertinent to the project at hand.

The Pyvi library offers functionalities such as word segmentation, Part-of-Speech (POS) tagging, and Named Entity Recognition (NER) that are customized for the Vietnamese language. Word segmentation pertains to the division of a piece of continuous text into separate words. POS tagging involves the marking up of a word in a text (corpus) corresponding to a particular part of speech, based on both its definition and its context. NER is a process where essential entities in the text are identified and classified into predetermined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.

```
pip3 install pyvi
```

3.1.2 Keras

Keras is a high-level neural networks Application Programming Interface (API) which designed to operate on multiple backend engines such as TensorFlow, CNTK, or Theano.

Keras is primarily employed for the development and training of deep learning models. It offers a streamlined, user-friendly interface for defining and configuring the architecture of these models. By providing multiple levels of abstraction, Keras allows users to build and train models with ease, without getting bogged down by the complexities of underlying algorithms.

A significant attribute of Keras is its highly modular and flexible nature. It enables the creation of a wide range of neural network architectures by allowing layers, loss functions, optimizers, initialization schemes, activation functions, and regularization schemes to be combined in various ways.

```
pip3 install keras
pip3 install --upgrade keras
```

3.1.3 TensorFlow

Developed by Google, TensorFlow is renowned for its flexibility and extensive capabilities in the domain of machine learning and artificial intelligence.

The primary functionalities of TensorFlow encompass the building, training, and deployment of machine learning models. It provides a comprehensive ecosystem of tools, libraries, and community resources that equip researchers and developers with the requisite infrastructure to advance machine learning and drive its practical applications.

TensorFlow supports a wide array of machine learning and deep learning algorithms, thereby offering flexibility in the choice of methodology based on the specific requirements of the project. Its robustness and versatility enable the implementation of complex computational models with relative ease, while also ensuring scalability and efficiency.

The framework's capabilities extend beyond just numerical computing. It also supports graph computations, symbolic computations, and even generalized probabilistic computations, which

makes it a powerful tool for researchers and developers working on a variety of complex tasks in machine learning.

```
pip3 install tensorflow
pip3 install --upgrade tensorflow
pip3 install "tensorflow-text==2.13.*" # avoid errors on Colab
```

3.1.4 Gensim

Gensim is a freely available Python library that is specifically designed for the extraction of semantic topics from documents with a focus on efficiency. Gensim's primary functionality lies in its ability to analyze large text corpora, identify semantic structure, and extract relevant topics.

The primary application of Gensim within the project is hypothesized to be in the creation of word embeddings or other forms of text representation. Word embeddings are essentially vector representations of words that capture their semantic content. This is achieved by placing semantically similar words closer in the vector space, thereby allowing algorithms to understand and leverage these semantic relationships.

```
pip3 install gensim
```

3.2 CNN implementation

The provided Python code outlines the construction, compilation, and training of a Convolutional Neural Network (CNN) for sentiment analysis. This model is built using the Keras library.

```
from keras.models import Model
from keras.layers import *
from keras import regularizers
from tensorflow.keras.optimizers import Adam

sequence_length = data.shape[1]
filter_sizes = [3,4,5]
num_filters = 100
drop = 0.5

inputs = Input(shape=(sequence_length,))
embedding = Embedding(input_dim=vocabulary_size, output_dim=EMBEDDING_DIM,
    input_length=sequence_length)(inputs)

conv_0 = Conv1D(num_filters, filter_sizes[0], activation='relu', kernel_regularizer=
    regularizers.l2(0.01))(embedding)
maxpool_0 = GlobalMaxPooling1D()(conv_0)

conv_1 = Conv1D(num_filters, filter_sizes[1], activation='relu', kernel_regularizer=
    regularizers.l2(0.01))(embedding)
maxpool_1 = GlobalMaxPooling1D()(conv_1)

conv_2 = Conv1D(num_filters, filter_sizes[2], activation='relu', kernel_regularizer=
    regularizers.l2(0.01))(embedding)
maxpool_2 = GlobalMaxPooling1D()(conv_2)

merged_tensor = concatenate([maxpool_0, maxpool_1, maxpool_2])
dropout = Dropout(drop)(merged_tensor)
output = Dense(units=3, activation='softmax',kernel_regularizer=regularizers.l2(0.01)
     (dropout)
```

```
26
27 model = Model(inputs, output)
28
29 adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
30 model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
31 model.summary()
```

3.2.1 Parameters

- sequence_length: the length of sequences that the model will process, which is equal to the number of columns in the input data.
- filter_sizes: defines the sizes of the filters in the convolutional layers.
- num_filters: the number of filters to be used in the convolutional layers. There will be 100 filters for each filter size.
- drop: the dropout rate used in the Dropout layer of the model, which helps prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.

3.2.2 Architecture

The architecture of the model can be broken down into 6 layers:

1. Input layer: accepts sequences of a defined length.
2. Embedding layer: after input layer, embedding layer transforms the input sequences into dense vectors of fixed size. This process helps in reducing the dimensionality of the input data and interprets the semantics behind words.
3. Conv1D and GlobalMaxPooling1D layers: the model applies 3 Conv1D layers with different filter sizes to the embedded sequences. Each Conv1D layer applies 100 filters to the sequences, using a Rectified Linear Unit (ReLU) activation function and L2 regularization. After each Conv1D layer, a GlobalMaxPooling1D layer is used which reduces the output of Conv1D to a single vector, capturing the most important information from the outputs.
4. Concatenation layer: the outputs of those max pooling layers are concatenated into a single tensor.
5. Dropout layer: The concatenated tensor is passed through a dropout layer. This layer randomly sets a proportion of input units to 0 at each update during training, which helps prevent overfitting.
6. Output layer: the output from the dropout layer is fed into a dense layer with 3 units and a softmax activation function. This is the output layer of the model and it outputs the probabilities of each of the three classes (positive, negative, neutral).

The model is compiled with Adam optimizer with a learning rate of 0.001. The loss function used is "categorical_crossentropy" for multi-class classification.

```
Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)        [(None, 300)]             0         []

embedding_1 (Embedding)     (None, 300, 400)          3167600   ['input_1[0][0]']

conv1d (Conv1D)             (None, 298, 100)          120100    ['embedding_1[0][0]']

conv1d_1 (Conv1D)           (None, 297, 100)          160100    ['embedding_1[0][0]']

conv1d_2 (Conv1D)           (None, 296, 100)          200100    ['embedding_1[0][0]']

global_max_pooling1d (Glob  (None, 100)               0         ['conv1d[0][0]']
alMaxPooling1D)

global_max_pooling1d_1 (Gl  (None, 100)               0         ['conv1d_1[0][0]']
obalMaxPooling1D)

global_max_pooling1d_2 (Gl  (None, 100)               0         ['conv1d_2[0][0]']
obalMaxPooling1D)

concatenate (Concatenate)   (None, 300)               0         ['global_max_pooling1d[0][0]',
                                                                 'global_max_pooling1d_1[0][0]
...
Total params: 3648803 (13.92 MB)
Trainable params: 3648803 (13.92 MB)
Non-trainable params: 0 (0.00 Byte)
```

3.2.3 Training

The model is trained for 10 epochs with a batch size of 256. The validation_split of 0.2 signifies that 20% of the data will be used for validation.

The model employs an EarlyStopping callback, which halts training when the validation loss has stopped improving for a certain number of epochs (patience is set to 4 epochs in this case).

3.3 LSTM implementation

The provided Python code outlines the construction, compilation, and training of a Long-Short Term Memory (LSTM) for sentiment analysis. This model is built using the Keras library.

```
from keras.layers import Bidirectional, LSTM
from keras import regularizers
from tensorflow.keras.optimizers import Adam

sequence_length = data.shape[1]
drop = 0.5

inputs = Input(shape=(sequence_length,))
embedding = embedding_layer(inputs)

reshape = Reshape((sequence_length,EMBEDDING_DIM))(embedding)

# Use bidirectional LSTM layers
lstm_2 = Bidirectional(LSTM(1024, return_sequences=True))(reshape)
lstm_1 = Bidirectional(LSTM(512, return_sequences=True))(lstm_2)
lstm_0 = Bidirectional(LSTM(256))(lstm_1)

```

```
dropout = Dropout(drop)(lstm_0)
output = Dense(units=3, activation='softmax',kernel_regularizer=regularizers.l2(0.01)
    )(dropout)

model = Model(inputs, output)

adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
model.summary()

# early_stopping = EarlyStopping(monitor='val_loss', min_delta=0.01, patience=4,
    verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
lr_reduction = ReduceLROnPlateau(monitor='val_loss', patience=2, verbose=1, factor
    =0.5, min_lr=0.00001)
callbacks_list = [early_stopping, lr_reduction]

model.fit(data, labels, validation_split=0.2,
          epochs=10, batch_size=256, callbacks=callbacks_list, shuffle=True)
```

3.3.1 Parameters

- sequence_length: the length of sequences that the model will process, which is equal to the number of columns in the input data.
- drop: the dropout rate used in the Dropout layer of the model, which helps prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.

3.3.2 Architecture

The architecture of the model can be broken down into 6 layers:

1. Input layer: accepts sequences of a defined length.
2. Embedding layer: transform the input sequences into dense vectors of fixed size.
3. Reshaped layer: the output of the embedding layer is reshaped to match the input requirements of the LSTM layers.
4. Bidirectional LSTM layers: Bidirectional LSTM layers are used to capture both the past (backward) and future (forward) context of a sequence at every time step. This is particularly useful in the context of text, where the understanding of a word can often depend on the words that follow it. Three such layers are used in the model with 1024, 512, and 256 units. The first two LSTM layers are set to return sequences which they will provide outputs at each time step. The last LSTM layer returns only the last output.
5. Dropout layer: prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.
6. Output layer: the output from the dropout layer is fed into a dense layer with 3 units and a softmax activation function. This is the output layer of the model and it outputs the probabilities of each of the three classes (positive, negative, neutral).

The model is compiled with Adam optimizer with a learning rate of 0.001. The loss function used is "categorical_crossentropy" for multi-class classification.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 300)]             0

 embedding (Embedding)       (None, 300, 400)          3167600

 reshape (Reshape)           (None, 300, 400)          0

 bidirectional (Bidirection  (None, 300, 2048)         11673600
 al)

 bidirectional_1 (Bidirecti  (None, 300, 1024)         10489856
 onal)

 bidirectional_2 (Bidirecti  (None, 512)               2623488
 onal)

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 3)                 1539

=================================================================
Total params: 27956083 (106.64 MB)
Trainable params: 27956083 (106.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

3.4 Combination method

The provided Python code outlines the construction, compilation, and training of a CNN-LSTM combination for sentiment analysis.

```
from keras.models import Model
from keras.layers import Input, Embedding, Reshape, Conv1D, MaxPooling1D, Dropout,
    LSTM, Dense, concatenate, Attention, Concatenate
from keras.optimizers import Adam
from keras import regularizers

sequence_length = data.shape[1]
filter_sizes = [3, 4, 5]
num_filters = 100
drop = 0.5

```

```
input = Input(shape=(sequence_length,))
embedding = embedding_layer(input)
reshape = Reshape((sequence_length, EMBEDDING_DIM))(embedding)

conv_0 = Conv1D(num_filters, filter_sizes[0], activation='relu', kernel_regularizer=
    regularizers.l2(0.01))(embedding)
maxpool_0 = MaxPooling1D(sequence_length-filter_sizes[0]+1, strides=1)(conv_0)

conv_1 = Conv1D(num_filters, filter_sizes[1], activation='relu', kernel_regularizer=
    regularizers.l2(0.01))(embedding)
maxpool_1 = MaxPooling1D(sequence_length-filter_sizes[1]+1, strides=1)(conv_1)

conv_2 = Conv1D(num_filters, filter_sizes[2], activation='relu', kernel_regularizer=
    regularizers.l2(0.01))(embedding)
maxpool_2 = MaxPooling1D(sequence_length-filter_sizes[2]+1, strides=1)(conv_2)

merged_tensor = concatenate([maxpool_0, maxpool_1, maxpool_2], axis=1)
# merged_tensor = Reshape((-1, 1))(merged_tensor)
lstm_2 = LSTM(units=128, return_sequences=False)(merged_tensor)

# attention = SeqSelfAttention(attention_type=SeqSelfAttention.ATTENTION_TYPE_MUL,
                            #  kernel_regularizer=regularizers.l2(0.01))(lstm_2)

attention = Attention()([lstm_2, lstm_2])
attended_output = Concatenate(axis=-1)([lstm_2, attention])

dropout = Dropout(rate=drop)(attention)
output = Dense(units=3, activation='softmax', kernel_regularizer=regularizers.l2
    (0.01))(dropout)

model = Model(inputs=input, outputs=output)
model.compile(loss='categorical_crossentropy',
            optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08),
            metrics=['accuracy'])
model.summary()
```

1. Input layer: defines the input shape for the model.
2. Embedding layer: maps the input sequences to a dense vector representation. It takes the input sequences as input and outputs the embedded representation of the sequences.
3. Reshape layer: reshapes the output of the embedding layer to the desired shape.
4. Conv1D layers and MaxPooling1D layers: perform 1D convolution on the input sequences. Each Conv1D layer has a specified number of filters, kernel size, activation function, and kernel regularizer.
5. Concatenate layer: concatenates the output tensors of the max pooling layers along the specified axis.
6. LSTM layer: processes the concatenated tensor using LSTM units.
7. Attention layer: performs self-attention on the LSTM output.
8. Dropout layer: applies dropout regularization to the attention output to prevent overfitting.

9. Output layer: applies a dense transformation to produce the final output of the model.

```
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_21 (InputLayer)       [(None, 300)]             0         []

 embedding_14 (Embedding)    (None, 300, 400)          3167600   ['input_21[0][0]']

 conv1d_52 (Conv1D)          (None, 298, 100)          120100    ['embedding_14[5][0]']

 conv1d_53 (Conv1D)          (None, 297, 100)          160100    ['embedding_14[5][0]']

 conv1d_54 (Conv1D)          (None, 296, 100)          200100    ['embedding_14[5][0]']

 max_pooling1d_25 (MaxPooli  (None, 1, 100)            0         ['conv1d_52[0][0]']
 ng1D)

 max_pooling1d_26 (MaxPooli  (None, 1, 100)            0         ['conv1d_53[0][0]']
 ng1D)

 max_pooling1d_27 (MaxPooli  (None, 1, 100)            0         ['conv1d_54[0][0]']
 ng1D)

 concatenate_24 (Concatenat  (None, 3, 100)            0         ['max_pooling1d_25[0][0]',
 e)                                                               'max_pooling1d_26[0][0]',
...
Total params: 3765535 (14.36 MB)
Trainable params: 3765535 (14.36 MB)
Non-trainable params: 0 (0.00 Byte)
```

4 Experiments

4.1 Evaluation metrics

To evaluate the performance of the models, the chosen evaluation criteria are accuracy and loss.

Accuracy is one of the most common evaluation metrics in the field of Natural Language Processing, especially in this project. The formula for calculating classification accuracy reflects the percentage of total words in the text that are correctly classified:

$$\eta = \frac{TP + TN}{TP + FP + FN + TN} \times 100\%$$

Loss quantifies the discrepancy between the actual and predicted values produced by the machine learning model. Lower values of the loss function indicate a better fit of the model to the data, implying a more accurate representation of the underlying relationships.

4.2 Results

Model	Loss (%)	Accuracy (%)
CNN	96.19%	66.67%
LSTM	84.02%	68.00%
CNN-LSTM	97.44%	69.71%

Table 2: Results of three models

4.3 Proposal

In the CNN-LSTM model, I add the **Attention layer** to potentially increase the accuracy due to the its specification:

- Enhanced focus: Attention mechanisms allow the model to focus on relevant parts of the input sequence. By assigning different weights to different input elements, attention helps the model selectively attend to important information and ignore irrelevant or noisy information.
- Contextual information: Attention mechanisms provide the model with contextual information about the input sequence. By considering the relationship between different elements in the sequence, attention helps capture long-range dependencies and understand the global context.
- Handling sequence variability: Attention mechanisms are particularly effective in handling variable-length input sequences. They allow the model to adaptively attend to different parts of the sequence, regardless of the sequence length.

Therefore, in order to enhance the performance potentially, I recommend including an Attention layer (you can try in both CNN and LSTM model).

Another proposal is the preprocessing data. As we've dealt with the problem of Vietnamese words, there are various stop words, punctuation, ... included that could be blurred the results. Hence, we need to improve this step by analyzing and clearly clean the datasets. The number of epochs and batch_sizes can be adjusted in training process to leverage the accuracy.

5 Conclusion

The combined CNN and LSTM model for text classification capitalizes on the LSTM's ability to retain historical information and contextual understanding of words within lengthy texts. It mitigates the issue of vanishing gradients and leverages the strengths of the CNN model to extract local textual features. Consequently, this fusion model amalgamates the advantages of both CNN and LSTM architectures.

However, empirical findings suggest that the model's performance improvement is not significant, and in certain cases, it even achieves inferior accuracy compared to the basic LSTM model. As the length and complexity of the text escalate, text classification becomes notably more challenging. Consequently, for extended text sequences, the application of Attention mechanisms in Deep Learning models can enhance the efficacy of text classification.

References

[1] Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR abs/1810.04805* (2018).

[2] Kim, Y. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Doha, Qatar, Oct. 2014), A. Moschitti, B. Pang, and W. Daelemans, Eds., Association for Computational Linguistics, pp. 1746–1751.

[3] trituenhantao.io. Vietnamese bert: Pretrained on news and wiki. *GitHub repository* (2020).

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023.

[5] Wang, J., Yu, L.-C., Lai, K. R., and Zhang, X. Dimensional sentiment analysis using a regional cnn-lstm model. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)* (2016), pp. 225–230.