CSCE 633 Semester Project Report

Dylan Nguyen
Texas A&M University
dylann39@tamu.edu

Abstract

This report discusses my methodology for sentiment analysis with a proposed transformer model. Sentiment analysis encompasses Natural Language Processing techniques to identify feature spaces, emotions, and textual context in documents. Various methods to accomplish sentiment analysis will be analyzed, and the reasoning for choosing the Bi-directional Encoder Representation from Transformers (BERT) based Convolution Bi-directional Recurrent Neural Network (CBRNN) will be explained. The implementation and results will be provided in detail.

1 Introduction

Sentiment analysis determines the intended emotions of a text corpus through primarily analyzing people's opinions, thoughts, and impressions [1]. Existing lexicon-based methods have poor performance in generalizing over different domains. Therefore, machine (or deep) learning-based methods are typically used for sentiment analysis as they discover patterns from the training data. While there are a plethora of machine learning-based methods, sentiment analysis normally follows a standard protocol: text pre-processing, feature extraction, augmentation, and interpretation of results [2].

Common feature extraction techniques using machine learning are Bag-of-Words, and N-grams [3]. These techniques use one-hot encoding which raises scalability challenges as the feature space becomes high dimensional. Thus, word-embedding models were developed with the purpose of extracting semantic and syntactic features from word representations [4]. One of the most popular word-embedding models is Word2vec which utilizes a combination of predicting words based on context and central or target words. Word2vec and similar method do have a few flaws when applied on sentiment analysis applications, however. They require a very large amount of data for creating word embeddings. Additionally, they don't fully grasp the context of each word as similar words in different contexts have a high possibility of containing similar vector representations [5]. This is partially due to their generation of feature vectors using only words in their vocabularies [6].

Thus, I implemented a transformer similar to the approach in [2]. This approach utilizes an enhance BERT-based CBRNN which addresses the above issues. Implementing a model composed BERT, Convolutional Neural Networks (CNN), and Bi-Directional Long Short Term Memory (Bi-LSTM) removes domain dependency while focusing on semantic and syntactic features, ultimately resulting in the extraction of contextual and sentimental features. The approach in [2] consists of:

- Using a zero-shot algorithm to initally learn the labels.
- Applying BERT to construct contextual and semantic embeddings.
- Feeding embeddings into a dilated CNN to extract local and global sentimental features.
- Using a Bi-LSTM to learn long-term information in two directions of a long sequence of text.
- Apply grid search CV to find the best hyper-parameters.

My method follows a similar architecture, but with different pre-processing steps, layer architecture (number of layers in the Bi-LSTM, number of filters in the CNN, number of dense layers, etc.),

and algorithm for finding the optimal hyper-parameters. In addition, my method uses different loss functions, optimizers, and activation functions.

The proposed method will be discussed in Section 2, with results and experiments analyzed in Section 3, and final remarks in Section 4.

2 Methodology

My proposed method utilizes BERT to generate contextual and semantic feature embeddings from pre-processed and tokenized data, feeds those embeddings into a dilated CNN and is passed through a max pooling layer, applies Bi-LSTM to capture contextual dependencies of sequences of text, and finally fed through a dense layer resulting in logits.

2.1 Pre-processing

First and foremost, I upload the Yelp reviews training data set into a pandas data frame. The dataset is very large, with approximately 175,000 reviews and ratings, so I create a subset of 9% (15,728) with a balanced distribution of class representations (negative, positive, and neutral sentiments) due to fine tuning processes and time constraints. I then remove all of the stop words and punctuation and convert all text samples to lowercase for better processing in my model. From there I split the subset into a training and validation data sets where the training set consists of 80% of the data (12,583) and the validation set consists of 20% of the data (3,145). Both data sets have a balanced representation of sentiments using a stratified function as before. Prior to tokenization, the 95th percentile number of tokens is computed which turns out to be 192. Thus, I initialize a pre-trained WordPiece tokenizer through the BertTokenizer Hugging Face package with a padding length of 192 to keep the best feature representation possible. Both the training and validation data sets are tokenized and are converted into numerical representations (token IDs) and attention masks for denoting which values can be disregarded due to padding. Each data set is generated into data loaders along with each sample's corresponding sentiment label. The train data loaders have batch sizes of 100 whereas the validation data loader has batch sizes of 150 to speed up runtime.

2.2 BERT for Creating Embeddings

As previously mentioned, BERT has an advantage over standard models like Word2Vec in that it creates word embeddings based on the context of surrounding words. BERT accomplishes that bidirectionally by examing both the left and right context of each term in all layers [7]. Given an input sequence, BERT produces a vector representation for each element in the input sequence. It performs the task through encoders which are neural networks that encode text representations. I used the pre-trained BERT-mini-uncased model from Hugging Face as it has the best computational efficiency. This model has four encoder blocks where each block has two layers composed of multi-head self-attention and feed-forward. BERT is a great choice as each multi-head attention block has several heads running in parallel where each head represents self-attention. The input passes through the multi-head self attention layer first for extracting important linguistic features, then is normalized and fed through the feed forward layer. The BERT-mini-uncased model has four encoder layers and attention heads, as mentioned, and 256 dimensions. The batch of input IDs and attention masks are first passed through BERT. Each encoder sends its output to the encoder above it where the final encoder creates the contextual embeddings for each review in the batch. Thus, the dimensions of the embeddings are (batch size, pad length, embedding dimensions). I reshape the embeddings to (batch size, 1, 192, 256) to pass into the CNN. The following image is the self-attention architecture of BERT mini demonstrated in [2]:

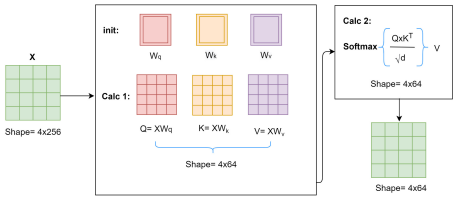

Figure 1: Self-attention architecture

2.3 Dilated CNN

As we've learned, CNNs are great at extracting local features through convolutions, and contain efficient memory usage due to pooling layers. An improvement upon standard CNNs are dilated CNNs. Dilated CNNs use flexible dilation rates which expand the receptive field without requiring more parameters and excessive pooling [8]. This results in each convolution including a broader range of feature information. Dilated CNNs have been widely used in applications that utilize long sequential information like text sequences. The also dilation helps with interpreting global context and increasing the size of outputs. An example of a dilated CNN:

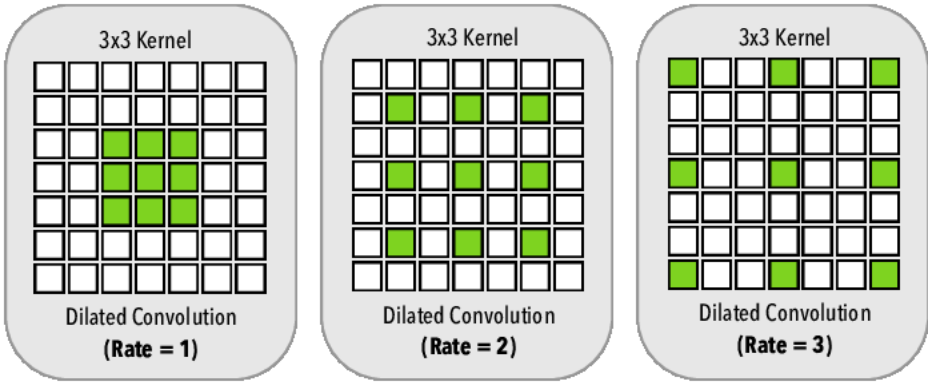

Figure 2: Dilated CNN with dilation rates of 1,2 and 3

My method incorporates three different dilated CNNs with a dilation of 1, 2, and 3 respectively. All three layers have in-channels of 1, 64 filters, and a kernel size of (3,3). The context embeddings from BERT feed forward through each dilated CNN independently, resulting in three different feature maps of different height and width dimensions due to the varying dilation sizes. ReLU activation is applied on each layer to prevent the vanishing gradient issue. There is a difference of 2 between each feature map dimension, therefore adaptive pooling is used to match all feature maps along the dimensions of the third dilated CNN while still preserving the detail of each feature map [9]. The three feature maps are concatenated into one singular feature map. This results in a very detailed feature extraction as all the dimensions from each feature map (namely the kernel sizes) are combined to form large feature representations for each review.

2.4 Pooling Layer

A max pooling layer is applied on the single feature map using a filter of dimensions (2,2). The use of max pooling reduces the feature space while retaining almost all of the important information. Concatenating all feature spaces result in a dimension of [100, 192, 186, 250] where the first term is the number of batches, the second is the combined feature maps (64 filters x 3 feature maps), the third term is the height, and the fourth term is the width. The height and width are reduced from 192 to 186 and 256 to 250, respectively, due to the third dilated CNN having a dilation of 3. Adaptive pooling is applied on all feature maps to match that height and width. Thus, after applying a pooling

layer with a filter dimension of (2,2), the resulting feature map is reduced to: [100, 192, 93, 125] (height/2, width/2).

2.5 Bi-LSTM

Long short-term memory models greatly improve the issues with RNNs. As the distance between two dependent words increase, the gradient begins to either vanish or explode. LSTM's combat that through gating mechanisms (input, forget, and output) that decide what information to keep and discard as the sequence moves on. Bi-LSTM's are even more advantageous, especially in sentiment analysis, as they concurrently traverse the data twice from the right and left, capturing better contextual relationships between words. The reduced feature space from max pooling is reshaped to match the input dimensions of Bi-LSTM: [batch size, sentence length, embbeding dim] = [100, 186, 12000]. In my implementation, I have a hidden size of 128 which denotes the output from the previous layer, 2 layers, a dropout rate of 0.2 to avoid overfitting, and the bidirectional flag set as true. This forms a Bi-LSTM that combines two hidden states, essentially preserving information from past and future. The output of the Bi-LSTM is flattened by only taking the last output timestep which represents the combined results of the forward and backward directions. My Bi-LSTM outputs a matrix of dimensions [100, 186, 256] which I then flatten to [100, 256] to feed through a dense layer. The following is an example of a Bi-LSTM:

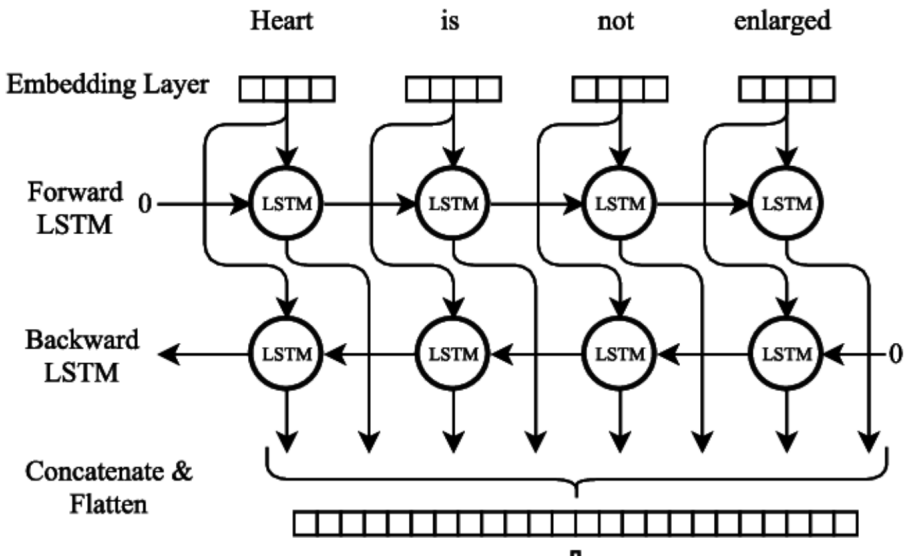

Figure 3: Bi-LSTM Example

2.6 Dense/Fully Connected Layer

This is the last layer in the transformer. It is a linear layer with in features of 256 to match the output from the Bi-LSTM an output size of 3 which represent class logits of each class for every sample in the batch. No other computation is applied on this layer (sigmoid, softmax, etc.) as I use a categorical cross-entropy loss function which computes softmax internally. This is the output of my transformer's forward method and can be used for optimization as well as prediction.

3 Experimentation and Results

3.1 Training and Optimization

Prior to training my transformer model, I first define important pieces for optimization: loss function, optimizer, and learning rate scheduler. I used PyTorch's cross entropy loss which incorporates multi-categorical cross entropy loss computation and softmax on each sample's class logits. The optimizer

is stochastic gradient descent with an initial learning rate of 0.01. The learning rate scheduler is used to reduce the learning rate in later iterations when the model begins to converge. I start off training with 6 epochs to examine how the transformer iteratively performs on the validation data set. Additionally, since BERT is pretrained, many epochs are not needed to optimize its parameters, the epochs are mainly used for the additional layers. The chosen batch size of train set is 100. The best model is saved, which can be loaded later to continue training. Below is my training loop algorithm:

Algorithm 1 Training for Sentiment Transformer

Initialize: model, cross entropy, SGD, StepLR, epochs, B_t, B_v

for $t = 1...$epochs **do**
  for batch $\in \mathcal{B}_t$ **do**
    get logits from model.forward(batch)
    compute loss through cross entropy
    reset SGD
    loss back propagation
    update parameters through SGD
    update classification accuracy
  end for
  compute current epoch train accuracy
  for batch $\in \mathcal{B}_v$ **do**
    get logits from model.forward(batch)
    update validation accuracy
  end for
  if current validation accuracy is best **then**
    save current model
  end if
  update learning rate with StepLR
end for

3.2 Training Results

After 6 epochs, the validation and training accuracy showed signs of convergence. However, I still wanted to train my transformer for a few more epochs to determine if there was any room left for learning. I ran another training loop with early stopping implemented and found that my model did indeed converge previously as the training accuracy began to increase while the validation accuracy and losses plateaued. Early stopping terminated the iterations before the model could overfit. The final training and validation accuracies each converged to 80%.

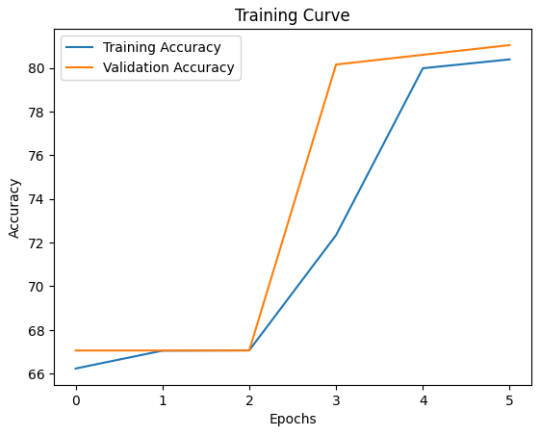

Figure 4: Accuracy vs Epochs

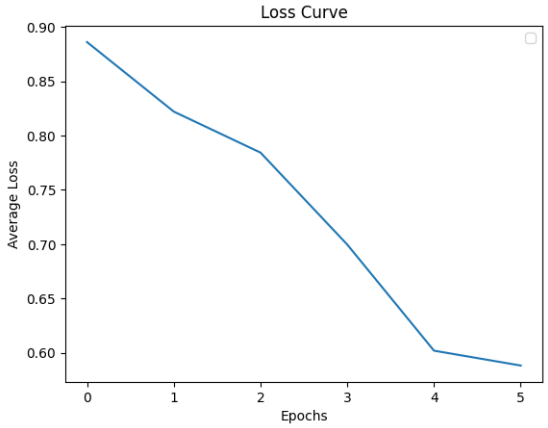

Figure 5: Loss vs Epochs

3.3 Evaluation

I preprocessed all of the test data in preparation for evaluation. The test results were very decent with results shown below.

Model	Accuracy	Precision	Recall	F1 Score	AUC
BERT-Based CBRNN	80.8	0.726	0.808	0.764	0.710

Table 1: Test Results

The accuracy indicates that the model correctly predicts the sentiment 80% of the time, it shows good performance but does not tell me about performance on individual classes. Precision is lower than accuracy which means there are a number of false positives. The equal recall and accuracy demonstrate my model is good at predicting positive cases. The F1 score shows that there is a good balance between recall and precision, although there can be improvement in precision. The AUC is decent but has room for improvement.

Many training epochs were not needed due to the trained parameters of BERT mini. However, this model is very robust and can perform much better with a lot more hyper-parameter tuning. First, hyper-parameters such as batch size, learning rate, and epochs can be further experimented and optimized to ensure optimal convergence. Next, the experimentation of more encoder and attention layers can even better contextual representations. It was not feasible to tune this layers as BERT is computationally expensive which means that I could not explore with other BERT variations with more layers. The additional encoder and multi-head layers can further learn the contexts and semantics around each piece of text, ultimately resulting in better representations of the context. Regardless, this model did perform well given the computational hardships and complexity.

4 Conclusion

In conclusion, my transformer model performed decently well with room for improvement. The architecture is very robust with BERT generating detailed semantic and contextual embeddings, dilated CNNs extracting a local feature space, and Bi-LSTM further expanding upon the contexts over long dependencies. The results were not poor, however with proper hyper-parameter tuning such as more encoder and multi-head layers, optimized learning rate and batch size, etc., this model can definitely perform well for any general corpus of text.

References

[1] M. Wankhade, A. C. S. Rao, and C. Kulkarni, "A survey on sentiment analysis methods, applications, and challenges," Artificial Intelligence Review, vol. 55, no. 55, Feb. 2022, doi: https://doi.org/10.1007/s10462-022-10144-1.

[2] S. Tabinda Kokab, S. Asghar, and S. Naz, "Transformer-based deep learning models for the sentiment analysis of social media data," Array, p. 100157, Apr. 2022, doi: https://doi.org/10.1016/j.array.2022.100157.

[3] Z. Jianqiang, G. Xiaolin, and Z. Xuejun, "Deep Convolution Neural Networks for Twitter Sentiment Analysis," IEEE Access, vol. 6, pp. 23253–23260, 2018, doi: https://doi.org/10.1109/access.2017.2776930.

[4] D.-H. Pham and A.-C. Le, "Exploiting multiple word embeddings and one-hot character vectors for aspect-based sentiment analysis," International Journal of Approximate Reasoning, vol. 103, pp. 1–10, Dec. 2018, doi: https://doi.org/10.1016/j.ijar.2018.08.003.

[5] C. Colón-Ruiz and I. Segura-Bedmar, "Comparing deep learning architectures for sentiment analysis on drug reviews," Journal of Biomedical Informatics, vol. 110, p. 103539, Oct. 2020, doi: https://doi.org/10.1016/j.jbi.2020.103539.

[6] S. Wang, W. Zhou, and C. Jiang, "A survey of word embeddings based on deep learning," Computing, vol. 102, no. 3, pp. 717–740, Nov. 2019, doi: https://doi.org/10.1007/s00607-019-00768-7.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv.org, Oct. 11, 2018. https://arxiv.org/abs/1810.04805

[8] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," arXiv:1511.07122 [cs], Apr. 2016, Available: https://arxiv.org/abs/1511.07122

[9] A. Stergiou and R. Poppe, "AdaPool: Exponential Adaptive Pooling for Information-Retaining Downsampling," arXiv.org, Dec. 02, 2022. https://arxiv.org/abs/2111.00772 (accessed Dec. 12, 2023).

[10] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "The Performance of LSTM and BiLSTM in Forecasting Time Series," 2019 IEEE International Conference on Big Data (Big Data), Dec. 2019, doi: https://doi.org/10.1109/bigdata47090.2019.9005997.