CNN and LSTM for Sentiment Analysis

CSE 582 - Assignment 2

Report by:

Name: SWATI RAGHAV

Email: sfr5677@psu.edu

CNN:

A Convolutional Neural Network (CNN) is a type of deep neural network widely used for image and video analysis, but also for natural language processing tasks such as sentiment classification.
In sentiment classification, the task is to predict the sentiment of a piece of text as either positive, negative or neutral.
CNNs are able to extract meaningful features from text data by using convolutional filters that slide over the text data and detect patterns such as n-grams or semantic features. These features are then fed into a fully connected layer that performs the final classification. CNNs have been shown to be effective for sentiment classification tasks due to their ability to capture both local and global features in text data, leading to improved accuracy compared to traditional machine learning models.

CNN for Sentiment Analysis (tanh):

Steps taken for sentiment analysis using CNN using activation layer: tanh

1. Get the data
    1. Downloaded the Yelp Restaurant reviews in a JSON file.
    2. Preprocessed the JSON File to give it a valid JSON format.
    3. Converted the JSON file into an Excel file with multiple columns.
        1. Extracted only the useful columns - text, stars
    4. Mapped stars to the sentiment using the following logic -
    ```
    def map_sentiment(stars_received):
        if stars_received <= 2:
            return -1
        elif stars_received == 3:
            return 0
        else:
            return 1
    ```
    5. Considered the top 10k records for analysis for each sentiment.
    6. Data Preprocessing
        1. Did not remove the stop words: the, a, an, etc.
            1. This would remove the necessary words that would be needed to get the sentiments.
        2. Tokenized the text using Gensim simple_preprocess function
        3. Performed Stemming using Porter
    7. Split the data into train set and test set (70:30)
    8. Define model
        1. Generate input and output tensors
            1. Word2Vec vectors trained with the embedding size 500
            2. Mapping from label to positive values
    9. Train the Model
        1. Loss: Entropy Loss
        2. Epochs: 1 (Asked by Professor)
        3. Learning Rate: 0.001
        4. Activation Function: tanh
    10. Save the model
    11. Test the model

CNN for Sentiment Analysis (relu):

1. All Steps stay the same as above. In the model, we have to mention relu in place of tanh as the activation function.

Parameters in the Layer of CNN for sentiment analysis:

1. CNN involves two operations: convolution and pooling.
2. The output of these operations is connected to the multi-layer perceptron to get the final output.
3. Convolution layers:
    a. Slide a small kernel window over the input
    b. Based on the window size, sliding happens through embedding vectors of a few words
    c. To look at sequences of word embeddings, the window has to consider multiple word embeddings in a sequence, resulting in rectangular shapes with a size of window_size * embedding_size.
    d. If the window size is 3, the kernel will be 3*500, representing n-grams in the model.
    e. The kernel weights (filter) are multiplied to word embeddings in pairs and summed up to get output values.
    f. The network learns these kernel weights while learning.
4. Input and output channels for convolutional:
    a. nn.Conv2d is used to create a convolution layer.
    b. Number of input channels: 1 for word embedding
    c. Output channels: total number of features
5. Padding
    a. This is needed when kernel size does not overlay perfectly
    b. Padding used = window size - 1
6. Max pooling:
    a. This is used to get the feature from a feature vector of a sentence.
    b. For example: if a sentence has “great ambience”, max pooling focused only on this phrase and not where it appears. Only this phrase is extracted.
7. Feedforward neural network
    a. Feedfoward neural network is added after the max-pooling layer

Input and Output Tensor:

1. Input: Word2Vec vectors
    a. Embedding size: 500
    b. A padding token is used to fill extra remaining words, it is needed for sentences that are shorter than the longest sentence in the corpus. This is done to keep the length of sentences the same
2. Output:
    a. Mapping from label to positive values is done

LSTM:

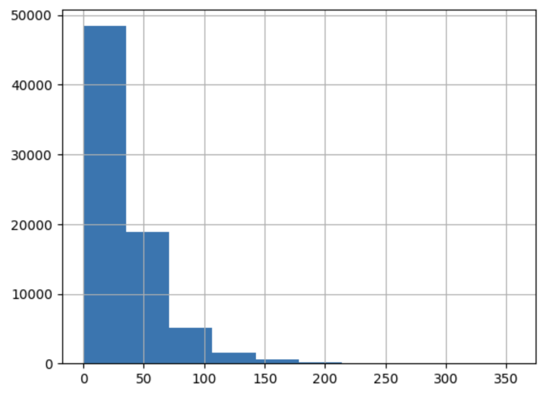
LSTM is a type of Recurrent Neural Network architecture that overcomes the vanishing gradient problem in traditional RNNs, allowing for the capture of long-term dependencies in sequential data. LSTM for sentiment analysis involves training an LSTM model on labelled text data and their corresponding sentiment labels, allowing the model to learn to accurately predict the sentiment of a given text based on the complex relationships between words and their context.

LSTM for Sentiment Analysis (Sigmoid):

Steps taken for sentiment analysis using LSTM using activation layer: Sigmoid

1. Get the data
- Import necessary libraries
- Load the dataset
- Visualize the dataset
- Extract required columns

2. Split the train (70:30)
   a. Initial data split helps avoid
3. Data Processing
   a. Remove all non-word characters (everything except numbers and letters)
   b. Replace all runs of whitespaces with no space
   c. Replace digits with no space.
   d. Convert data to lowercase
   e. Sorting on the basis of most common words
   f. Tokenize
4. Now we will pad each of the sequences to the same length
5. Splitting to train and test data (70:30)
6. Pad train and test data according to review length
   a. Getting rid of extremely long and short reviews
   b. Padding / Truncating the remaining data



7. Batching and loading as tensor
   a. Data loaders and Batching
   b. Batch size = 50
8. Define the Model
   a. Define the LSTM Network Architecture
   b. Define the Model Class
   c. Embedding dimension = 64
   d. Learning rate = 0.001
   e. Loss = BCELoss
   f. Epoch = 1 (Mentioned by Prof)
9. Test the model

LSTM for Sentiment Analysis (Relu):

1. All Steps stay the same as above. In the model, we have to mention relu in place of Sigmoid as the activation function.

Parameters in the Layer:

1. The model is the subclass of nn.Module
2. The model has an
    a. embedding layer,
    b. an LSTM layer,
    c. a dropout layer,
    d. a linear layer, and
    e. A sigmoid activation function Relu Function.
3. Embedding layer
    a. Converts input data to lower-dimensional space (embedding)
    b. Each token in input data is represented by a dense vector
    c. Initialized with random values and learns useful representation for downstream tasks
    d. Defined using nn.Embedding class in PyTorch
4. LSTM layer
    a. Takes in a sequence of inputs and produces a sequence of hidden states
    b. Solves vanishing gradients problem that arises in traditional RNNs when training on long sequences
    c. Defined using nn.LSTM class in PyTorch
    d. Takes in embedded input data and the initial hidden state of the LSTM, and produces a sequence of hidden states as output
5. Dropout layer
    a. Used to prevent overfitting during training
    b. Randomly drops out some units in the previous layer with a probability drop_prob
    c. Defined using nn.Dropout class in PyTorch
6. Linear layer
    a. Takes in the output of the LSTM layer and produces a single output value for each time step in input sequence
    b. Defined using nn.Linear class in PyTorch
7. Sigmoid activation function
    a. Maps output of linear layer to a value between 0 and 1
    b. Interpreted as the probability of input sequence being positive or negative
    c. Defined using nn.Sigmoid class in PyTorch

3. Comparison of two models

Training time and learning rate:

Model	**Activation Function**	**Learning rate**	**Training Time**
CNN	tanh	0.001	5 minutes 56 seconds
CNN	relu	0.001	5 minutes 38 seconds
LSTM		0.001	12mins, 57 seconds

Model	Activation Function	Learning rate	Training Time
CNN	tanh	0.0001	494 minutes 32 seconds
CNN	relu	0.0001	400 minutes 30 seconds
LSTM		0.0001	621 minutes

Learning rate impacts the training time for each model a lot.

Training Time VS Activation Function:

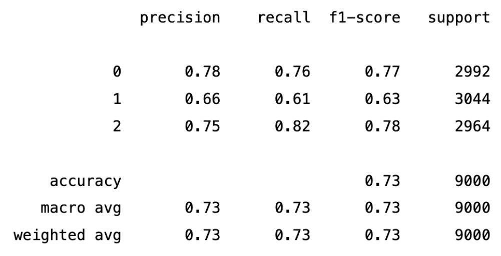
Model + Activation Layer	Training Time
CNN + tanh	5 minutes 56 seconds
CNN + relu	5 minutes 38 seconds

Training time is same for both the activation functions for CNN model.

Accuracy:

Model + Activation Layer	Accuracy
CNN + tanh	73% for one epoch
CNN + relu	71% for one epoch
LSTM	90% for one epoch

Precision, recall, F1 score for CNN (tanh)

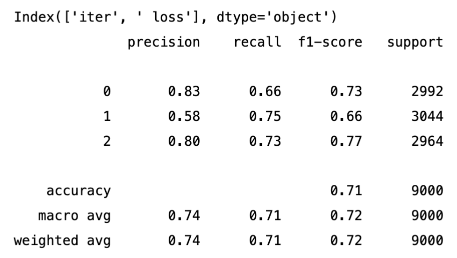


```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77      2992
           1       0.66      0.61      0.63      3044
           2       0.75      0.82      0.78      2964

    accuracy                           0.73      9000
   macro avg       0.73      0.73      0.73      9000
weighted avg       0.73      0.73      0.73      9000
```

Precision, recall, F1 score for CNN (relu)



```
Index(['iter', ' loss'], dtype='object')
              precision    recall  f1-score   support

           0       0.83      0.66      0.73      2992
           1       0.58      0.75      0.66      3044
           2       0.80      0.73      0.77      2964

    accuracy                           0.71      9000
   macro avg       0.74      0.71      0.72      9000
weighted avg       0.74      0.71      0.72      9000
```

- Based on the tables and data, we can see that LSTM is better than CNN models in terms of accuracy.
- Based on training time, CNN is faster than LSTM.

Learning from the assignment:

1. Refreshed NLP concepts:
    a. To work with the Yelp restaurant reviews dataset and build and train the CNN and LSTM models for sentiment analysis, I refreshed the NLP concepts such as
        i. tokenization,
        ii. word embeddings, and
        iii. sequence modelling
2. Refreshed data preprocessing techniques:
    a. Preprocessing the Yelp restaurant reviews dataset involved
        i. cleaning the text,
        ii. transforming the text into a format that the models can work with.
3. Learnt more about deep learning architectures: CNN and LSTM
    a. How these two architectures are used for sentiment analysis.
4. Hyperparameter tuning for performance optimization:
    a. Changed the below parameters and observed the behaviour of the two models:
        i. learning rate,
        ii. batch size,
        iii. Activation function
5. Model evaluation:
    a. Evaluation helped understand the strengths and weaknesses of each architecture.

References

1. https://www.kaggle.com/code/arunmohan003/sentiment-analysis-using-lstm-pytorch#kln-14
2. https://towardsdatascience.com/sentiment-classification-using-cnn-in-pytorch-fba3c6840430