Venkata Krishna Preetham Vayigandla - 1630873

Rank – 11 with F1 Score: 0.8634

Title: Classifying Movie Reviews As Positive or Negative using RNN

Abstract

Given a dataset of movie reviews and the corresponding labels of +1 or -1, we are to train a Recurrent Neural Network, using the data and predict the same on a test set. Here, 1 represents positive and -1 represents negative.

Introduction

We are given a dataset of 25,000 Movie Reviews and each review is a sentence or a paragraph with many words. The possible values for the classification task are +1 or -1. There are a total of two classes in which the reviews could fall in. LSTM architecture was used to train the neural network. And F1 Score was used as the accuracy measure. The techniques used are discussed below.

Approach

Pre-processing

The dataset contains a list of sentences and text should be converted to numeric data. Initially, all the sentences were converted to lower case and the punctuations were removed from all the sentences by iterating over each review. Then we constructed a vocabulary list of all the words in all the reviews.

I used the Counter function from collections library to create a hash table of the vocabulary. It returns a dictionary where the keys are the vocabulary words and the values are its frequency. Once the vocabulary is built, we iterate through each pre-processed review, free of punctuations, and converting the review to a vector based on the frequency of occurrence of each word.

Simultaneously, we convert the labels to an appropriate format, converting the +1 to 1 and -1 to 0. Since, all the reviews are of not the same length, there will be different sized vectors, and an RNN is not capable of handling vectors with different sizes. Hence, we initialize a sequence length which defines the default size of each vector, vector inputs greater than that size are trimmed and smaller than that are padded with zeroes.

I split the training data into validation set and train set, we use TensorDataset to wrap the inputs and outputs into tensors. And we use DataLoader to wrap an iterable around the entire dataset to make the samples accessible.

Next, an RNN class is defined.

Neural Network Architecture

We us Long-Short-Term-Memory to train the model, with 2 layers and 256 nodes in each layer. An embedding layer is used to create word vectors for the incoming words. And finally, we define the LSTM layer using PyTorch with a dropout probability of 0.5. After that we use the dropout layer to add noise to the word vectors so that the model is not too reliant on one specific neuron. Finally, we add the linear layer which acts as a fully connected layer and then comes the sigmoid activation function for transforming the outputs.

Training

Binary cross entropy loss is used to calculate the training loss and backpropagate the error and modify the weights. Initially, I trained a model with 2 layers, a Sequence Length of 500, learning rate of 0.001, for 4 epochs and Batch Size of 50, the F1 score came out to be 86.28.

After training the model for 15 epochs, the F1 score came out to be 86.34, with all the parameters the same. After training the model on different parameters below with their respective scores, the best model was the one trained on 15 epochs above.

1. Epochs – 4, Learning Rate - 0.001, F1 - 86.28, Layers – 2, Sequence Length – 500, Batch Size - 50
2. Epochs – 15, Learning Rate - 0.001, F1 - 86.34, Layers – 2, Sequence Length – 500, Batch Size - 50
3. Epochs – 15, Learning Rate - 0.0005, F1 - 85.43, Layers – 2, Sequence Length – 500, Batch Size - 50
4. Epochs – 3, Learning Rate - 0.001, F1 - 84.65, Layers – 2, Sequence Length – 1200, Batch Size - 50
5. Epochs – 4, Learning Rate - 0.001, F1 – 78, Layers – 2, Sequence Length – 1000, Batch Size - 1000
6. Epochs – 15, Learning Rate - 0.0005, F1 - 86.34, Layers – 2, Sequence Length – 500, Batch Size – 128

Below is a graph of the training and validation loss for 7 epochs, the model is overfitting when trained for more epochs, I tried to apply the above hyper-tuning of the parameters, but none of them worked. (I couldn’t get the graph because while training I received a `CUDA out of memory error’)

Initially, when the model was trained without tuning the hyperparameters, the model had low bias, because the model was overfitting and it has high variance for the same reason. As the hyperparameters were tuned, like using the dropout layer and changing the sequence length of the word vectors, the bias increased and the variance started decreasing, minimizing the overfitting.

Conclusion

By using Random search CV for choosing the hyperparameters, the F1 Score of the model could have been increased.