Deep Learning based Movie Review Sentiment Analysis

SUKHADIA KOLINBEN
RAKESHKUMAR

Department of Electrical and
Computer Engineering

Lakehead University

ThunderBay, Canada

***Abstract*— To extract review using natural language processing method, This paper propose methodology in which there is change in activation function, optimizationtechnique and length of vocabulary. Reviews are classified into positive and negative. Long Short Term memory method is used for movie review sentiment analysis. To improve training as well as validation accuracy of it, this paper propose method in which activation function is Exponential linear unit, Optimization technique is RMSprop(Root Mean Squared Propogation). This paper propose method in which by increasing length of vocabulary, there is improvement in sentiment prediction accuracy.**

Keywords—Recurrent Neural Networks, Long Short Term Memory, activation function, movie review dataset,

I. INTRODUCTION

These days, sentiment analysis is a difficult work that isrequired in almost every industry. it's critical to create language models to extract valuable data. Specifically for movie reviews. Sentiment analysis depict people's opinion on any movie to determoine whether they are positive or negative.In preprocessing, there are tokenization, lemmatization, and stop word removal and finally classification method to label test data . Seperating text into words and sentences is called tokenization. Various classification methods are used for movie review sentiment analysis. To convert the word into root word, removing affixes and prefixes is called lemmatization process. This paper aims to improve ttraining as well as validation accuracy of prediction of sentiment of movie review. Various performance evaluation metrics are accuracy training as well as testing accuracy, and AUC. In this research, RNN language model based on Long Short Term Memory is applied on movie review database. In this research, there is improvement in training as well as validation accuracy and AUC. For short sentences, due to lack of data or information, analysis of sentiment is very difficult. In this research, Long Short term memory – Recurrent neural network is used to to prediction for sentiment analysis. In this research, to improve accuracy of prediction of sentiment, we used several hyperparameters namely, length of vocabulary, dropout probability, various activation functions and optimization technique.

II. LITERATURE REVIEW

In [1], author applied various methods namely Naïve Bayes, Random Forest, Logistic Regression, SGD classifier and Knn classifers on feature representations of the review textual information. They used three methodologies namely, bag of words, N-gram modelling and TF-IDF modelling for feature extraction.

In this paper[2], two machine learning algorithms applied such as naïve bayes and SVM(Support Vector Machine) on movie review and sentiment database. They did feature selection, and then preprocessing to remove tab spaces, newlines,numbers or digits. It also helps to remove stopwords as well as punctuations.

In this[3], For sentiment classification, there is comparision of CNN, LSTM and LSTM-CNN architecture done by authors IMDb movie review database. After performing experiments, F1 score of CNN is 91% which is more accurrate than other models namely LSTM and LSTM-CNN.

This paper[4] explains a RNN language model with Long Short Term Memory(LSTM).This compared with traditional RNN model. It performs better than conventional RNN.

In this[5], authors describe about dataset which includes scripts as well as reviews. They used two well-known lexicons, one for sentiment analysis and for emotion analysis, other is used. They looked at how well they worked on their own and in conjuction with vector semantic tools. Results show that CountVectorizer method give more accurate performance than TF-IDF method. Dataset includes 747 movie scripts and 78000 reviews. In this paper, Multinomial Naïve Bayes, Logistic Regression, Support Vector Machine and Multilayer Perception. Highest accuracy is given by Multinomial Naïve Bayes algorithm.

In this paper[6], Neural network model has been made with six layers. Out of six layers, there are four hidden, one input and one output layer is there.validation accuracy of model is 86.67 percent, whereas training accuracy of model is 91.9 percent.

Authors in [7] explain optimization LSTM for sentiment analysis. Evaluation metrics in this research are accuracy, F-measure, precision and recall. In preprocessing stage, there are stop word removal, tokenization, segregation and stemming are applied. To make model more better as well as to depict the word in lower dimension , skip-gram based word2vec architecture is used. In this, author proposed optimized LSTM in which optimal weight parameters are applied to improve performance of LSTM algorithm.
In this paper[8], there is survey of existing methodologies such as Naïve Bayes, Support vector Machine are applied for sentiment analysis. There are various machine learning algorithms are used for predicting sentiment analysis[9]. Authors in this paper[10] prove that sentimental analysis is harder than simple classification task. Adam optimizer, its advantages and disadvantages are discussed by author in this paper[11]. In this paper[12], various hyperparameters namely dropout probability, activation function PreLU(Parametric rectified linear unit) and normalization is used to improve accuracy. In [13], authors prove that Long Short Term Memory(LSTM) is better than other techniques such as convolutional neural network and recurrent neural network.In[14], on IMDb movie review dataset, performance of various methods namely, CNN, LSTM and LSTM-CNN are measured. There is Long Short term memory(LSTM) is applied on movie review dataset[15]. In [16], there are various optimizers namely adam and RMSProp are discussed.

III.METHODOLOGY

Chain of modules of neural network is there in Recurrent neural network. Figure shows block diagram of Recurrent neural network[12].

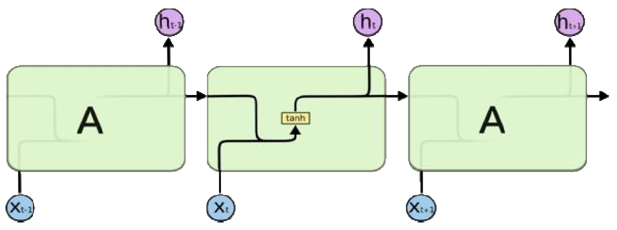


Figure 1.1: block diagram of RNN

artificial neurological system that receives, processes, and transmits information is known as a neural network.
several neurons in neural network.
There are connections between them. A neural network is a collection of neurons with synapses8 that connect them.
There are three layers namely input, hidden and output layers.
Input to neural network is there through this input layer. There are many hidden layers. It take input from input layer. Processed data is produced by output layer.
Input data is handled by input layer. In the input layer, Every neuron depicts an independent variable. It also affect on final output.
In hidden layers, there are also set of neurons and has an activation function. various activation functions namely sigmoid, tanh, ELU and ReLU are applied in hidden layers. This layer is also referred to as the middle layer. The hidden layer's primary function is to extract essential features from data given by preceding layers or layers. Depending on the intricacy of the task, there may be numerous hidden levels in the network. because of the activation function can be implemented directly on the input layer. If there is increase in hidden layers, then only it is not guaranteed to improve in training as well as validation accuracy[13].
Output layer receives input from hidden layers and generate output. Prediction of classes whether it is positive or negative is performed by output layer.
There are several merits of recurrent neural networks. It can process any length input. The other advantage is that for longer input, model size does not increase.Additionally, at every time, same weights applied. Because of problem of explding or vanishing gradients, Recurrent neural network fail to depict long sequence in real time applications. To solve this problem, there is Long short term memory.
LSTM Long Short-Term Memory (LSTM) is a form of RNN architecture implementation that is faster .it is more accurate than RNN. LSTM, in fact, leads to far more successful executions and learns much faster. complex tasks are performed by it and outperforms traditional RNN structures for long-range sequences[13].
Lstm is variantof recurrent neural network . Long term dependencies are learned by LSTM networks.It is ability to remember any information for long amount of period.
To add or remove data from cell state, LSTM is there.
In LSTM, there are four gates such as input , forget, and output gate. When there is removing information from cell state, then forget gate[13]. Less important data and the date or information which is no longer required is removed by filter or forget gate. LSTM(Long Short term memory) also store data. It helps to decide what new information is stored

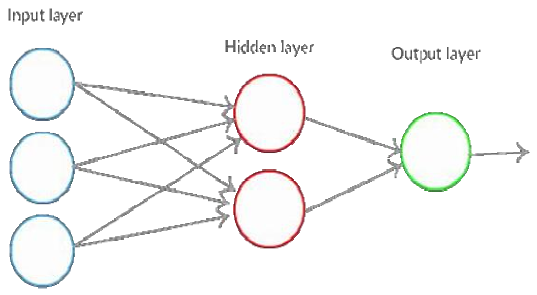
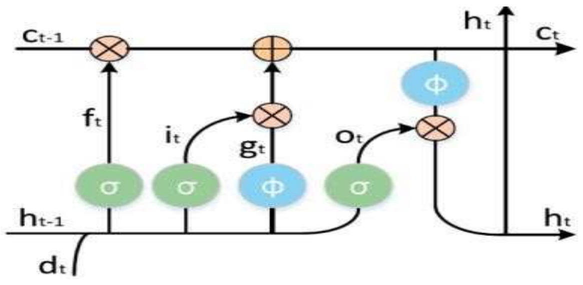


Figure 1.2 : Connection between different layers



.

Figure 1.3 : Illustration of LSTM cell[13]

In Data preprocessing Padding makes it easier to process the data, padding is applied. It is applied to each review for ensuring that they are all same size. Padding is adding some additional bits to the data to make it a specific size. After that, data is trained using machine learning and classification methods. The evaluation metrics are the confusion matrix as well as the model's accuracy.
Tokenization is process of splits a text into symbols, words, and numbers. Other stage in data preprocessing is that removal of stop words and punctuation. Stop words appear so many times in a review.It helps to better predict sentiment whether it is positive or negative. Lemmatization is the process in which it removes words ending part and return only baseform of a word.

Activation Functions in neural network are Sigmoid, tanh, and ReLu, ELU[8].
In neural network, there are Nnumber of inputs to nodes, which are then passed through nonlinearity to produce an output. nonlinearities are referred to as activation functions.

A. Sigmoid activation function :

There is range of [0,1] in sigmoid activation function. In feed forward neural network, Sigmoid activation function is used. For predicting probability based output, sigmoid function is utilized. The graph is like S shape. In the output layer of deep learning models, sigmoid activation function applied. Sharp damp gradients from hidden layers to input layer during backpropagation and gradient saturation are demetis of sigmoid activation function.

B..hyperbolic Tangent Function:
It has range of [-1, 1].

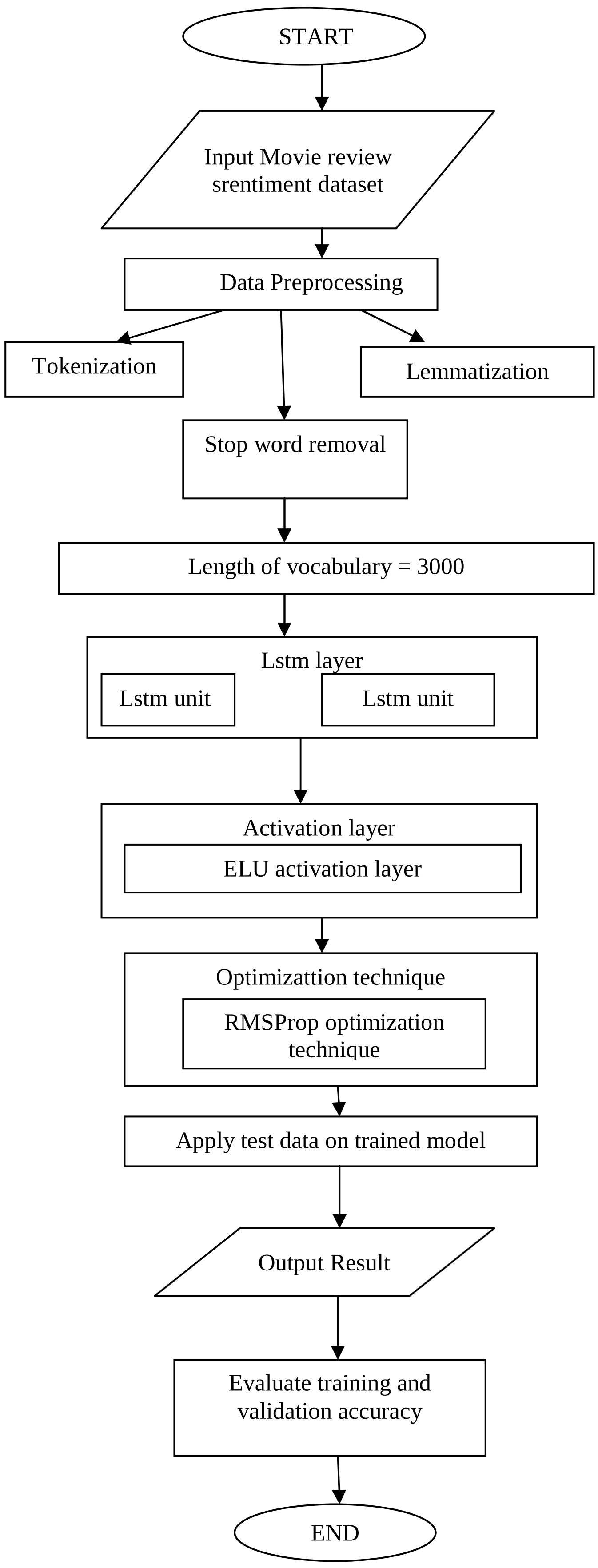
C.Exponential Linear Unit
there is range from -1 to infinity. It is a variant of ReLU nonlinearity. ELU has its merits as well as demerits.ELU is not piece wise linear. There is problem of gradient vanishing and gradient exploding in ELU. There are two parameters namely alpha and inplace. Alpha value is 1.04 and inplace value is boolean.

Optimization Techniques:

RMSProp optimization technique : In neural network training, RMSprop is a gradient-based optimization strategy[15].
As data propagates through very complicated functions like neural networks, gradients have aability to either vanish or explode problem[15].

Adam optimization technique :
In neural network, Adam optimizer is first order gradient based optimization method[15]. It is used in machine learning. It has parameter namely learning rate. Learmning rate in this research is 0.001.

Proposed methodology flowchart :



There is improvement in validation as well as training accuracy when length of vocabulary is increasing from 1000 to 3000. This is one of proposed methodology by which we get better accuracy and less train as well as validation loss.
The other method by whiich I get better accuracy by applying different optimizattion technique namely, RMSProp method. There are several activation functions namely sigmoid, tanh and ELU(Exponential Linear Unit). There is better accuracy and less loss while applying ELU activation method.

IV. Experimental Result and Analysis :

Accuracy :
It checks whether predicted class is same as trueclass. It calculates accuracy of single pair.
The formula for accuracy is
Accuracy = total correct predictions/total predictions.
In below figure, there is training accuracy after applying training part of Imdb dataset for sentiment analysis. When there is increment in Length of vocabulary from 1000 to 3000 then training accuracy increases.

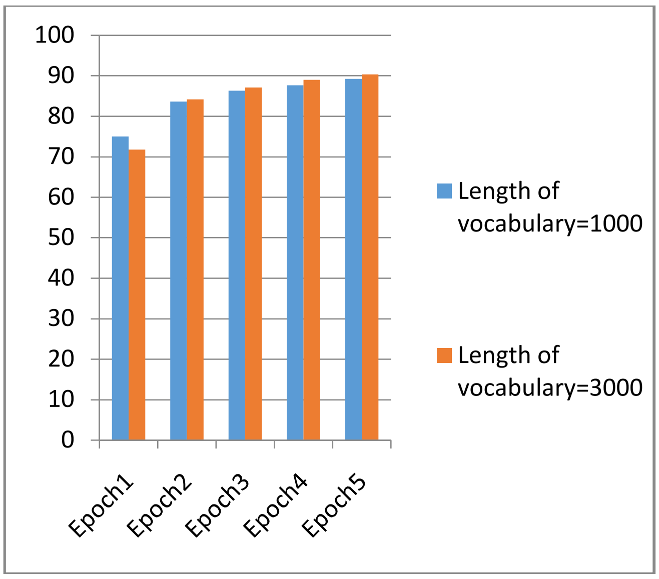


Figure 1.4 : Training Accuracy

So , there is improvement in validation accuracy when dataset is applied on test dataset.

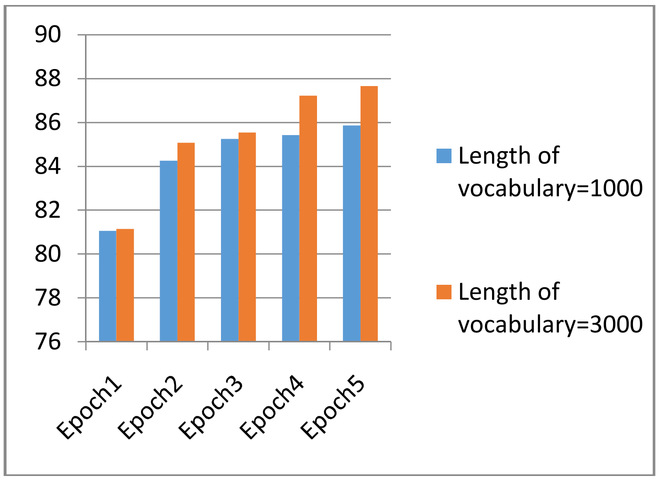


Figure 1.4: Validation Accuracy

AUC (Area under ROC Curve) is also increased when length of vocabulary is 3000.

```
AUC ROC :
0.8767
--------------------------------------------------------
```

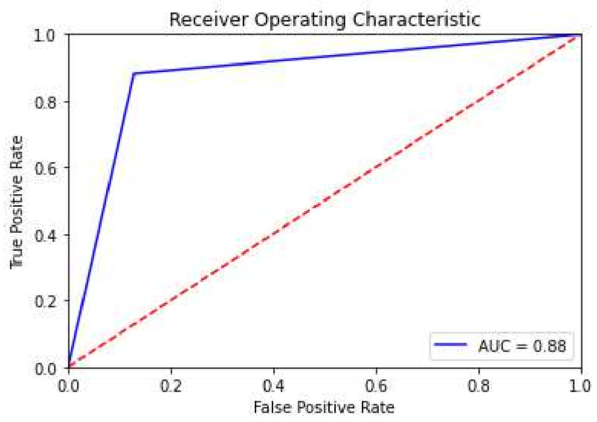


Figure 1.5: improved AUC

AUC value is 0.88 which is improved in this research.

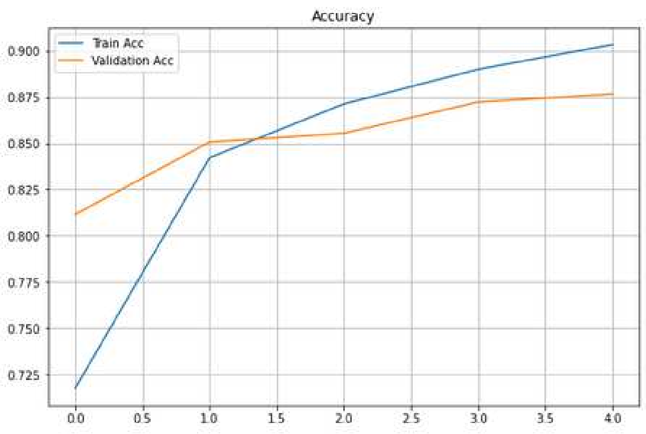


Figure 1.6 : Training accuracy vs validation accuracy

The above graph depicts the information about training accuracy and validation accuracy when length of vocabulary is 3000.

In this research, RMSProp optimizer performs well. It provides better accuracy as well as AUC(Area under ROC curve). RMSProp optimizer provides 87.68 validation accuracy and 91.28 training accuracy , whereas Adam optimizer technique provide highest training accuracy 90.335, whereas validation accuracy 87.66. so RMSProp optimization technique improve validation accuracy by 0.02%.

RMSProp (Root Mean Squared Propagation) is an extension of AdaGrad version of gradient descent
RMSProp has several parameters namely learning rate, alpha, epsilon, momentum and centered. Learning rate is equal to 0.001, alpha value is 1.0.

There is also improvement in accuracy when Exponential Linear Unit(ELU) activation funtion is applied. It has two parameters namely alpha and inplace. Alpha value is 1.04 and inplace value is False.

Accuracy	Project basecode sigmoid activation function	ELU activation function
Training Accuracy	91.28	91.36
Validation Accuracy	87.68	87.72

Table 1.1 : Comparison between basecode sigmoid activation function and Exponential Linear Unit(ELU) acgtivation function

ELU activation function gives better accuracy than basecode sigmoid activation function.

V.CONCLUSION

There are various methodologies applied on movie review sentiment analysis such as naïve bayes, Support Vector Machine, Linear Regression, Logistic Regression. This paper provides more accurate classification training as well as validation accuracy as well as AUC. For better result, there is optimization technique named RMSProp which suits better to the given base code and it provides more accurate training accuracy,validation accuracy and AUC. So it is better optimization algorithm in movie review sentiment analysis.It has parameters namely learning rate, epsilon, weight_decay, momentum and centered.in this paper, by increasing length of vocabulary from 1000 to 3000, There is improvement in training as well as validation accuracy. So proposed method uses most 3000 common words for prediction of sentiment. In this research, proposed methodology has different activation function Exponential Linear Unit(ELU) than base code activation function namely sigmoid. ELU provides better accuracy as well as Area under curve. There is optimization technique RMSprop which provides better accuracy than existing method optimizattion technique adam. So accuracy increases and loss decreased when most common words(length of vocabulary) increases as well as other parameters namely activation function and optimization technique also helps to improve sentiment prediction accuracy.

VI. REFERENCES

1. Ankit Goyal, Amey Parulekar, “Sentiment Analysis for Movie Reviews”,pp. 1-8
2. Prachi Sanghvi, Disha Shah, Bharathi H. N.,”Movie Revieew System using Sentiment Analysis”, International Journal for Research in Applied Science & Engineering Technology,pp.1193-1196.
3. Pallavi Sharma, Nidhi Mishra, “Feature level Sentiment Analysis on Movie Reviews”, 2016 International Conference on Next Generation Computing Technologies,pp.306-311.
4. Dan Li, Jiang Qian, “Text Sentiment Analysis Based on Long Short –Term Memory”, 2016 First IEEE International Conference on Computer Communication and the internet,pp.471-475, 2016.
5. Paschalis Frangidis, Konstantinos Georgiou, Stefanos Papadopoulos, “ Sentiment Analysis on Movie Scripts and Reviews Utilizing Sentiment Scores in Rating Prediction“, International Federation for Information Processing Springer, pp. 430-438, 2020.
6. Zeeshan Shahkat, Abdul Ahad Zulfiqar, Chuangbai Xiao, Muhammad Azeem, “Sentiment analysis on IMDB using lexicon and neural networks”,SN Applied Sciences,pp.,2020.
7. J. Shobana, M.Murali, “An efficient sentiment analysis methodology based on long short term memory networks”
8. Tirath Prasad Sahu, Sanjeev Ahuja, “Sentiment analysis of movie reviews: A study on feature selection & classification algorithms”, IEEE, 2016.
9. J sai, Teja, G Kiran Sai, M Druva Kumar, R. Manikandan, “Sentiment Analysis of Movie Reviews Using Machine Learning Algorithms – A Survey”, International Journal pf pure and Applied Mathematics, 2018.
10. Andrew L. Maas, Raymond E. Daly, PeterPham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)
11. Kingma, D. P., & Ba, J. , "Adam: A method for stochastic optimization.", 2014.”

12. SUMESH KUMAR NAIR, RAVINDRA SONI, “Sentiment Analysis on Movie Reviews using Recurrent Neural Network”, IRE ICONIC RESEARCH AND ENGINEERING JOURNALS, Volume 1 Issue 10, ISSN : 2456-8880, PP.242-251.
13. Ashok Tholusuri, Manish Anumala, Bhagyaraj Malapolu, G. Jaya Lakshmi, “Sentiment Analysis using LSTM”, International Journal of Engineering and Advanced Technology(IJEAT), ISSN:2249-8958, Volume-8, Issue-6S3, pp.1338-1340, September 2019.
14. Md. Rakibul Haque, Salma Aktar Lima, Sadia Zaman Mishu, “Performance Analysis of different Neural Networks for Sentiment Analysis on IMDb Movie reviews”, ResearchGate, 18July 2020.
15. Jyotsna Devi Bodapati, N. Veeranjaneyulu, Shareef Shaik, “Sentiment Analysis from Movie Reviews Using LSTMS”.
16. https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be

VII. APPENDIX

```
#CODE
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.optim import SGD
from torch.optim import rmsprop
import sklearn.metrics as metrics
from tensorflow.keras import activations
from tensorflow.keras import layers
from sklearn.metrics import roc_auc_score
#from mxnet import autograd, np
#from d2l import mxnet as d2l
import torch.nn.functional as F
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
from collections import Counter
import string
import re
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import train_test_split
from google.colab import drive
drive.mount('/content/drive')
is_cuda = torch.cuda.is_available()
if is_cuda:
  device = torch.device("cuda")
  print("GPU is available")
else:
      device = torch.device("cpu")
      print("GPU not available, CPU used")
base_csv = '/content/drive/MyDrive/IMDB Dataset.csv'
df = pd.read_csv(base_csv)
df.head()
X,y = df['review'].values,df['sentiment'].values
#x_train,x_test,y_train,y_test = train_test_split(X,y,stratify=y)
#This line split data in 80% 20% ratio
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.20,stratify=y)
print(f'shape of train data is  {x_train.shape}')
print(f'shape of test data is {x_test.shape}')
dd = pd.Series(y_train).value_counts()
sns.barplot(x=np.array(['negative','positive']),y=dd.values)
plt.show()
def preprocess_string(s):
  #remove all non word characters
  s = re.sub(r"[^\w\s]",'',s)
  #replace all runs of whitespaces with no space
  s = re.sub(r"\s+",'', s)
  #replace digits wiyh no space
  s = re.sub(r"\d",'',s)
  return s
#end of the def block
def tockenize(x_train,y_train,x_val,y_val):
  word_list = []
  stop_words = set(stopwords.words('english'))
  for sent in x_train:
    for word in sent.lower().split():
      word = preprocess_string(word)
      if word not in stop_words and word !='':
        word_list.append(word)
  corpus = Counter(word_list)
  #sorting on the basis of most common words
  corpus_ = sorted(corpus,key=corpus.get,reverse=True)[:3000]
  #creating dictionary
  onehot_dict = {w:i+1 for i,w in enumerate(corpus_)}
  #tockenize
  final_list_train,final_list_test =  [],[]
  for sent in x_train:
    final_list_train.append([onehot_dict[preprocess_string(word)]
     for word in sent.lower().split()
      if preprocess_string(word) in onehot_dict.keys()])
  for sent in x_val:
    final_list_test.append([onehot_dict[preprocess_string(word)]
      for word in sent.lower().split()
        if preprocess_string(word) in onehot_dict.keys()])
  encoded_train = [1 if label == 'positive' else 0 for label in y_train]
  encoded_test = [1 if label =='positive' else 0 for label in y_val]
 # (pd.Series(nltk.ngrams(word, 3)).value_counts())
 # bigrams_series.sort_values().plot.barh(color='blue', width=0.9, figsize(10,8))
  return np.array(final_list_train), np.array(encoded_train),np.array(final_list_test),np.array(encoded_test),onehot_dict

#end of the def block
x_train,y_train,x_test,y_test,vocab = tockenize(x_train,y_train,x_test,y_test)
print(f'Length of vocabulary is {len(vocab)}')
rev_len = [len(i) for i in x_train]
pd.Series(rev_len).hist()
#pd.Series(nltk.ngrams(word, 3)).value_counts()
plt.show()
pd.Series(rev_len).describe()
def padding_(sentences, seq_len):
  features = np.zeros((len(sentences), seq_len),dtype=int)
  for ii, review in enumerate(sentences):
    if len(review) != 0:
      features[ii, -len(review):] = np.array(review)[:seq_len]
  return features
  #end of the def block
x_train_pad = padding_(x_train,500)
x_test_pad = padding_(x_test,500)
# create tensor datasets
train_data = TensorDataset(torch.from_numpy(x_train_pad), torch.from_numpy(y_train))
valid_data = TensorDataset(torch.from_numpy(x_test_pad), torch.from_numpy(y_test))
```

```
#dataloaders
batch_size = 50

train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size)

dataiter = iter(train_loader)
sample_x, sample_y = dataiter.next()

print('sample input size:', sample_x.size())
print('sample input: \n', sample_x)
print('sample input: \n', sample_y)




from google.colab import drive
drive.mount('/content/drive')

class SentimentRNN(nn.Module):
   def __init__(self,no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.3):
        super(SentimentRNN,self).__init__()

        self.output_dim = output_dim
        self.hidden_dim = hidden_dim

        self.no_layers = no_layers
        self.vocab_size = vocab_size

        # embedding and LSTM layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        #lstm
        self.lstm = nn.LSTM(input_size=embedding_dim,hidden_size=self.hidden_dim,
                           num_layers=no_layers, batch_first=True)


        # dropout layer
        self.dropout = nn.Dropout(0.3)
     #   self.fc = nn.Linear(self.hidden_dim, output_dim)


         # linear and sigmoid layer
        self.fc = nn.Linear(self.hidden_dim, output_dim)

       # Exponential Linear Unit which helps to improve accuracy better than base code activation function sigmoid. its
parameters are alpha and inplace.
        self.ELU = nn.ELU(alpha=1.04, inplace=False)







   def forward(self,x,hidden):
        batch_size = x.size(0)
        # embeddings and lstm_out
        embeds = self.embedding(x)  # shape: B x S x Feature   since batch = True
        #print(embeds.shape)  #[50, 500, 1000]
        lstm_out, hidden = self.lstm(embeds, hidden)

        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully connected layer
        out = self.dropout(lstm_out)
        out = self.fc(out)



        # ELU activation function instead function

        elu_out = self.ELU(out)

        # reshape to be batch_size first
        #sig_out = sig_out.view(batch_size, -1)
        #sig_out = sig_out[:, -1]
        #relu_out = relu_out.view(batch_size, -1)
        #relu_out = relu_out[:, -1]
        # this change is for ELU
        elu_out = elu_out.view(batch_size, -1)
        elu_out = elu_out[:, -1]

        # return last sigmoid output and hidden state
        #return sig_out, hidden
        #return relu_out, hidden
        return elu_out, hidden



    def init_hidden(self, batch_size):
        ''' Initializes hidden state '''
        # Create two new tensors with sizes n_layers x batch_size x hidden_dim,
        # initialized to zero, for hidden state and cell state of LSTM
        h0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
        c0 = torch.zeros((self.no_layers,batch_size,self.hidden_dim)).to(device)
        hidden = (h0,c0)
        return hidden

#class NgramModel(nn.Module):
 # def __init__(self, vocab_size, context)

no_layers = 2
vocab_size = len(vocab) + 1 #extra 1 for padding
embedding_dim = 64
output_dim = 1
hidden_dim = 256


model = SentimentRNN(no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.3)

#moving to gpu
model.to(device)

print(model)

#loss and optimization functions
lr=0.001

#criterion = nn.BCELoss()
#criterion = nn.L1Loss()
```

```
criterion = nn.MSELoss()
#optimizer = torch.optim.Adam(model.parameters(), lr=lr)
#optimizer = torch.optim.Adadelta(model.parameters(), lr=lr, rho=0.9, eps=1e-06, weight_decay=0)
#optimizer = torch.optim.Adam(model.parameters(), lr=lr)
#optimizer = torch.optim.RMSprop(model.parameters(),)
#optimizer = torch.optim.Adadelta(model.parameters(), lr=lr)
#optimizer = torch.optim.Adamax(model.parameters(), lr=lr, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)
#optimizer = torch.optim.SGD(model.parameters(), lr=lr)
#optimizer = torch.optim.AdamW(model.parameters(), lr=lr, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)
#RMSProp optimization function gives better accuracy than Adam optimizer.
optimizer = torch.optim.RMSprop(model.parameters(), lr=lr, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0,
centered=False)
#optimizer = torch.optim.Rprop(model.parameters(), lr=lr, etas=(0.5, 1.2), step_sizes=(1e-06, 50))
#low output
#optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9)
#optimizer = torch.optim.Adadelta(model.parameters(), lr=lr, rho=0.9, eps=1e-06)
#optimizer = torch.optim.RMSprop(model.parameters(), lr=lr, alpha=0.99)


def acc(pred,label):
    pred = torch.round(pred.squeeze())
    return torch.sum(pred == label.squeeze()).item()



clip = 5
epochs = 5
valid_loss_min = np.Inf
# train for some number of epochs
epoch_tr_loss,epoch_vl_loss = [],[]
epoch_tr_acc,epoch_vl_acc = [],[]

for epoch in range(epochs):
    train_losses = []
    train_acc = 0.0
    model.train()
    # initialize hidden state
    h = model.init_hidden(batch_size)
    for inputs, labels in train_loader:

        inputs, labels = inputs.to(device), labels.to(device)
        # Creating new variables for the hidden state, otherwise
        # we'd backprop through the entire training history
        h = tuple([each.data for each in h])

        model.zero_grad()

        if(len(inputs)==batch_size):
          output,h = model(inputs,h)
         # calculate the loss and perform backprop
          loss = criterion(output.squeeze(), labels.float())
          loss.backward()
          train_losses.append(loss.item())
        # calculating accuracy
          accuracy = acc(output,labels)
          train_acc += accuracy
        #`clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
        nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()
    val_h = model.init_hidden(batch_size)
    val_losses = []
    val_acc = 0.0
    model.eval()
    for inputs, labels in valid_loader:
            val_h = tuple([each.data for each in val_h])
            inputs, labels = inputs.to(device), labels.to(device)

            output, val_h = model(inputs, val_h)
            val_loss = criterion(output.squeeze(), labels.float())

            val_losses.append(val_loss.item())

            accuracy = acc(output,labels)
            val_acc += accuracy

    epoch_train_loss = np.mean(train_losses)
    epoch_val_loss = np.mean(val_losses)
    epoch_train_acc = train_acc/len(train_loader.dataset)
    epoch_val_acc = val_acc/len(valid_loader.dataset)
    epoch_tr_loss.append(epoch_train_loss)
    epoch_vl_loss.append(epoch_val_loss)
    epoch_tr_acc.append(epoch_train_acc)
    epoch_vl_acc.append(epoch_val_acc)
    print(f'Epoch {epoch+1}')
    print(f'train_loss : {epoch_train_loss} val_loss : {epoch_val_loss}')
    print(f'train_accuracy : {epoch_train_acc*100} val_accuracy : {epoch_val_acc*100}')
    if epoch_val_loss <= valid_loss_min:
      torch.save(model.state_dict(), '/content/drive/MyDrive/Colab Notebooks/torch_save.pt')
      print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(valid_loss_min,epoch_val_loss))
      valid_loss_min = epoch_val_loss
    print(25*'==')




fig = plt.figure(figsize = (20, 6))
plt.subplot(1, 2, 1)
plt.plot(epoch_tr_acc, label='Train Acc')
plt.plot(epoch_vl_acc, label='Validation Acc')
plt.title("Accuracy")
plt.legend()
plt.grid()
plt.subplot(1, 2, 2)
plt.plot(epoch_tr_loss, label='Train loss')
plt.plot(epoch_vl_loss, label='Validation loss')
plt.title("Loss")
plt.legend()
plt.grid()

plt.show()


import sklearn.metrics as metrics
from sklearn.metrics import roc_auc_score
def plot_auc_roc(model,valid_loader, version='title', threshold=0.5):
    y_pred = []
    y_true = []
    val_h = model.init_hidden(batch_size)
    model.eval()
    with torch.no_grad():
        for inputs, labels in valid_loader :

            val_h = tuple([each.data for each in val_h])

            inputs, labels = inputs.to(device), labels.to(device)

            output, val_h = model(inputs, val_h)

            output = (output > threshold).int()
            y_pred.extend(output.tolist())
            y_true.extend(labels.tolist())

    print('AUC ROC :')
    fpr, tpr, threshold = metrics.roc_curve(y_true, y_pred)
    roc_auc = metrics.auc(fpr, tpr)

    print(roc_auc)
    print('----------------------------------------------------------')

    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
plot_auc_roc(model, valid_loader)

def predict_text(text):
    word_seq = np.array([vocab[preprocess_string(word)] for word in text.split()]
      if preprocess_string(word) in vocab.keys()])
    word_seq = np.expand_dims(word_seq,axis=0)
    pad = torch.from_numpy(padding_(word_seq,500))
    inputs = pad.to(device)
    batch_size = 1
    h = model.init_hidden(batch_size)
    h = tuple([each.data for each in h])
    output, h = model(inputs, h)
    return(output.item())



index = 30
print(df['review'][index])
print('='*70)
print(f'Actual sentiment is : {df["sentiment"][index]}')
print('='*70)
pro = predict_text(df['review'][index])
status = "positive" if pro>0.5 else "negative"
pro = (1 - pro) if status == "negative" else pro
print(f'predicted sentiment is {status} with a probability of {pro}')
```