Lab 7 - Building And Testing Feed Forward Neural Network For The Twitter Sentiment Analysis Dataset

Import Important Libraries

```
import pandas as pd
import string
import os
import numpy as np
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from PIL import Image
import urllib
import requests
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import roc_curve, auc, classification_report, accuracy_score
from sklearn.preprocessing import label_binarize
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
import time
import glob
import seaborn as sns
import matplotlib.pyplot as plt
import operator
import folium
from itertools import cycle, islice
from pandas import options
import warnings
import pickle
import re
import nltk
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch
from matplotlib.pyplot import figure
from nltk.corpus import stopwords
import nltk
```

```
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
True
```

Here we load the dataset (I only imported the necessary columns, i.e. 'target' and 'text') and perform data preprocessing.

```
# Take in the whole dataset and two specific columns
data = pd.read_csv('/content/SentimentTweets.csv', usecols = ['target','text'])
```

```
data.head()
```

	target	text
0	0	#brokenpromises...
1	0	David Carradine so sad. Thai's law not sure i...
2	4	A @ 415 B @ 425. Tell your bro i say congrats!
3	4	@littlefluffycat Indeed.
4	4	Completed Race 4 Life in 58mins with girlies f...

```
data.shape
```

```
(1280000, 2)
```

```
data = data.dropna() # Remove missing values
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1280000 entries, 0 to 1279999
Data columns (total 2 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   target  1280000 non-null  int64
 1   text    1280000 non-null  object
dtypes: int64(1), object(1)
memory usage: 29.3+ MB
```

```
# Replace the positive label of 4 to 1
data["target"] = data["target"].replace(4,1)
display(data["target"])
```

```
0          0
1          0
2          1
3          1
4          1
          ..
1279995    1
1279996    1
1279997    0
1279998    1
1279999    0
Name: target, Length: 1280000, dtype: int64
```

```
data["text"] = data["text"].str.lower() # Lower the text
```

```
data.head()
```

	target	text
0	0	#brokenpromises...
1	0	david carradine so sad. thai's law not sure i...
2	1	a @ 415 b @ 425. tell your bro i say congrats!
3	1	@littlefluffycat indeed.
4	1	completed race 4 life in 58mins with girlies f...

```
# Remove punctuations
data['text'] = data['text'].apply(lambda x:''.join(\
              [i for i in x if i not in string.punctuation]))
```

```
data.head()
```

	target	text
0	0	brokenpromises
1	0	david carradine so sad thais law not sure if ...
2	1	a 415 b 425 tell your bro i say congrats
3	1	littlefluffycat indeed
4	1	completed race 4 life in 58mins with girlies f...

```
# Remove characters that aren't words
data["text"] = data["text"].str.replace(r'[^a-zA-z0-9\s]','', regex=True)
```

```
data
```

	target	text
0	0	brokenpromises
1	0	david carradine so sad thais law not sure if ...
2	1	a 415 b 425 tell your bro i say congrats
3	1	littlefluffycat indeed
4	1	completed race 4 life in 58mins with girlies f...
...	...	...
1279995	1	zawhtutwin watching cartoon and cry oh i do th...
1279996	1	is eating mcdonalds
1279997	0	bestsoylatte so sorry to hear about your carth...
1279998	1	leesherry you have done what you could forgive...
1279999	0	i should b sleepin i hate 8ams

1280000 rows × 2 columns

```
# Remove html tags
data["text"] = data["text"].str.replace(r'<[^>]+>', '', regex=True)
```

```
# Remove html links
data["text"] = data["text"].replace(r'http\S+', '', regex=True).replace(\
             r'www.\S+', '', regex=True).replace(\
             r'http\S+', '', regex=True).replace(r'"', '', regex=True)
```

```
# Remove numbers from text
data["text"] = data["text"].str.replace('\d+', '', regex=True)
```

```
data
```

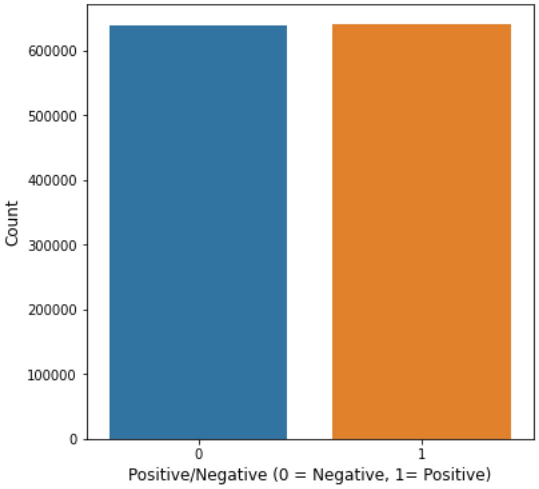
	target	text
0	0	brokenpromises
1	0	david carradine so sad thais law not sure if ...
2	1	a b tell your bro i say congrats
3	1	littlefluffycat indeed
4	1	completed race life in mins with girlies from...
...	...	...
1279995	1	zawhtutwin watching cartoon and cry oh i do th...
1279996	1	is eating mcdonalds
1279997	0	bestsoylatte so sorry to hear about your carth...
1279998	1	leesherry you have done what you could forgive...
1279999	0	i should b sleepin i hate ams

1280000 rows × 2 columns

```
# Check to see how many positive and negative comments we have
data["target"].value_counts()
```

```
1    640609
0    639391
Name: target, dtype: int64
```

```
# Plot the positive and negative values

plt.figure(figsize=(6,6))
sns.countplot(x=data['target'], data=data)
plt.xlabel("Positive/Negative (0 = Negative, 1= Positive)", fontsize=12)
plt.ylabel("Count", fontsize=12)

plt.show()
```



```
# Tokenization and Lemmanization

w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()

def lemmatize_text(text):
    return " ".join([lemmatizer.lemmatize(w, pos="v") for w in w_tokenizer.tokenize(text)])
```

```
data["text"] = data.text.apply(lemmatize_text).copy() # Deep copy

data
```

	target	text
0	0	brokenpromises
1	0	david carradine so sad thais law not sure if i...
2	1	a b tell your bro i say congrats
3	1	littlefluffycat indeed
4	1	complete race life in mins with girlies from w...
...	...	...
1279995	1	zawhtutwin watch cartoon and cry oh i do that ...
1279996	1	be eat mcdonalds
1279997	0	bestsoylatte so sorry to hear about your carth...
1279998	1	leesherry you have do what you could forgivene...
1279999	0	i should b sleepin i hate ams

1280000 rows × 2 columns

Here we split the dataset into train, test, and validation set. I choose to go with 0.02 (98/2) as our test_size.

```
# Split dataset to train and test set.
X_train, X_test, Y_train, Y_test = train_test_split(data['text'], data['target'], test_size=0.02, random_state=42)

# Split train dataset to train and validation set.
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.02, random_state=42)

print("Shape of x_train: ", X_train.shape)
print("Shape of y_train: ", Y_train.shape)
print("Shape of x_test:  ", X_test.shape)
print("Shape of y_test:  ", Y_test.shape)
print("Shape of x_val:  ", X_val.shape)
print("Shape of y_val:  ", Y_val.shape)
```

```
Shape of x_train:  (1229312,)
Shape of y_train:  (1229312,)
Shape of x_test:   (25600,)
Shape of y_test:   (25600,)
Shape of x_val:   (25088,)
Shape of y_val:   (25088,)
```

Here we perform vectorization with TF-IDF, and then we convert datasets to tensors.

```
from sklearn.feature_extraction.text import TfidfVectorizer

STOPWORDS = set(stopwords.words('english'))

tfidf_vectorizer = TfidfVectorizer(max_features=1000, dtype=np.float32)

tfidfX_train = tfidf_vectorizer.fit_transform(X_train)
tfidfX_train = tfidfX_train.toarray()

tfidfX_val = tfidf_vectorizer.transform(X_val)
tfidfX_val = tfidfX_val.toarray()

tfidfX_test = tfidf_vectorizer.transform(X_test)
tfidfX_test = tfidfX_test.toarray()

print("TF-IDF train shape:", tfidfX_train.shape)
print("TF-IDF test shape:", tfidfX_test.shape)
print("TF-IDF val shape:", tfidfX_val.shape)
```

```
TF-IDF train shape: (1229312, 1000)
TF-IDF test shape: (25600, 1000)
TF-IDF val shape: (25088, 1000)
```

```
# Convert X datasets to tensors.
tfidfX_train = torch.tensor(tfidfX_train)
tfidfX_val = torch.tensor(tfidfX_val)
tfidfX_test = torch.tensor(tfidfX_test)
```

```
# Convert Y labels to tensors with torch.squeeze.
Y_train = torch.squeeze(torch.from_numpy(Y_train.to_numpy()).float())
Y_val = torch.squeeze(torch.from_numpy(Y_val.to_numpy()).float())
Y_test = torch.squeeze(torch.from_numpy(Y_test.to_numpy()).float())
```

Here we make sure to use cuda with our runtime set to GPU. Then we begin by building our Feed Forward Neural Network. After that, we initialize the network and fill in each required parameter.

```
# See if cuda is available
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

```
# Use cuda
cuda_device = torch.device("cuda")
```

```
class FeedforwardNeuralNetModel(nn.Module):
    def __init__(self, input_dim, hidden_dim_1, hidden_dim_2, output_dim):
        super(FeedforwardNeuralNetModel, self).__init__()

        # Linear function 1
        self.layer_1 = nn.Linear(input_dim, hidden_dim_1)

        # Non-linearity 1
        self.relu_1 = nn.ReLU()

        # Linear function 2
        self.layer_2 = nn.Linear(hidden_dim_1, hidden_dim_2)
        # Non-linearity 2
        self.relu_2 = nn.ReLU()

        # Linear function 3
        self.layer_3 = nn.Linear(hidden_dim_2, output_dim)
    def forward(self, x):
        out = self.layer_1(x)
        out = self.relu_1(out)

        out = self.layer_2(out)
        out = self.relu_2(out)

        out = self.layer_3(out)

        return torch.sigmoid(out)
```

```
# Dimensions of each layer and num of epochs.
input_dim = tfidfX_train.shape[1]
hidden_dim_1 = 500
hidden_dim_2 = 100
output_dim = 1
num_epochs = 60
lr = 0.0001
wd = 0.001

# Define feed forward neural network.
ffnn_model = FeedforwardNeuralNetModel(input_dim,hidden_dim_1,hidden_dim_2,output_dim)

# Define loss function.
criterion = nn.BCELoss()
# criterion = nn.CrossEntropyLoss()

# Define as optimizer Adam.
optimizer = optim.Adam(ffnn_model.parameters(),lr=lr,weight_decay=wd)

# Transfer all the computation to GPU (cuda device).
ffnn_model.to(cuda_device)
criterion = criterion.to(cuda_device)
```

Here we train the model.

First we split the dataset for each epoch. Then we performed calculation after each epoch, while keeping tack of the accuracy and loss for each iteration.

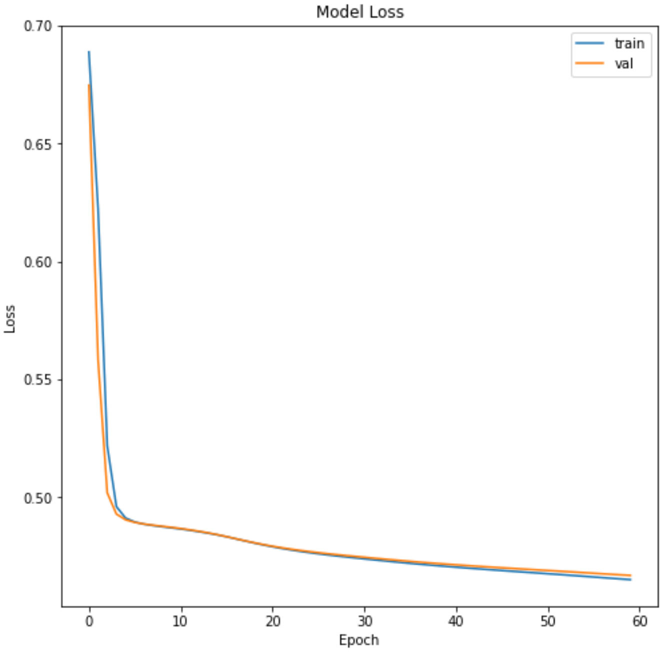
```

batch_size = 10000

# Split train dataset to mini batches
X_train_mini_batches = torch.split(tfidfX_train,batch_size)
Y_train_mini_batches = torch.split(Y_train,batch_size)

train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

# Start training
for epoch in range(num_epochs):
  epoch_loss = 0
  epoch_accuracy = 0
  validation_loss=0
  val_accuracy=0

  for X_train_mini_batch,Y_train_mini_batch in zip(X_train_mini_batches,Y_train_mini_batches):

    X_train_mini_batch = X_train_mini_batch.to(cuda_device)
    Y_train_mini_batch = Y_train_mini_batch.to(cuda_device)

    # Forward pass to get output
    train_prediction = ffnn_model.forward(X_train_mini_batch.float())
    train_prediction = torch.squeeze(train_prediction)

    # Calculate Loss
    train_loss = criterion(train_prediction,Y_train_mini_batch)

    # Clearing up accumulated gradients
    optimizer.zero_grad()
    # Getting gradients
    train_loss.backward()

    # Updating parameters
    optimizer.step()

    # Add each mini batch's loss
    epoch_loss += train_loss.item()

    # Add each mini batch's accuracy
    epoch_accuracy += ((train_prediction.round() == Y_train_mini_batch).sum()/float(train_prediction.shape[0]))

  # For some epochs print loss and accucary of train and validation set.
  if epoch % 1 == 0:

    tfidfX_val = tfidfX_val.to(cuda_device)
    Y_val = Y_val.to(cuda_device)

    # Forward pass to get output
    val_prediction = ffnn_model.forward(tfidfX_val.float())
    val_prediction = torch.squeeze(val_prediction)

    # Calculate Loss
    val_loss = criterion(val_prediction,Y_val)

    # Add each mini batch's loss
    validation_loss = val_loss.item()

    # Add each mini batch's accuracy
    val_accuracy = ((val_prediction.round() == Y_val).sum()/float(val_prediction.shape[0]))

    epoch_loss /= len(X_train_mini_batches)
    epoch_accuracy /= len(X_train_mini_batches)
    val_losses.append(validation_loss)
    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_accuracy.cpu().detach().numpy())
    val_accuracies.append(val_accuracy.cpu().detach().numpy())
```

```
print("Epoch:",epoch, "\n"
    "Train_loss:",round(epoch_loss,4), '\n' "Train Accuracy:",round(epoch_accuracy.item(),4), "\n"
    "Validation_loss:  ",round(validation_loss,4), '\n' "Validation Accuracy:  ",round(val_accuracy.item(),4), "\n")
```

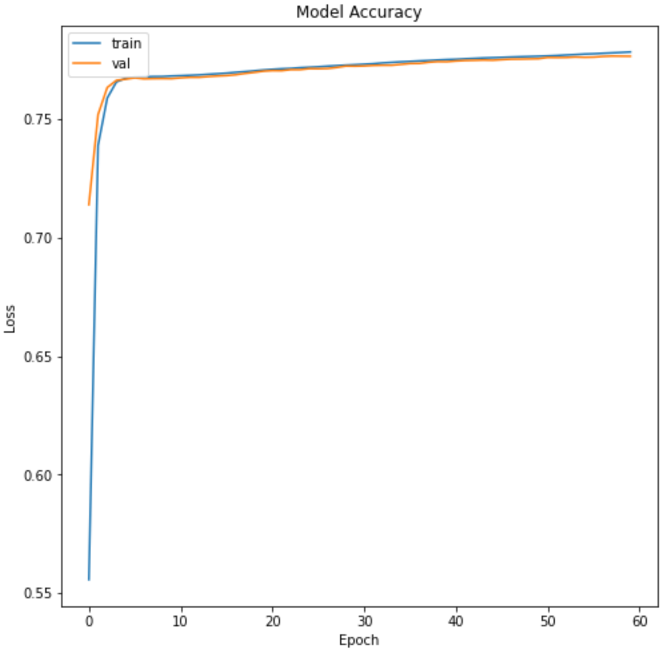
```
Epoch: 59
Train_loss: 0.465
Train Accuracy: 0.7784
Validation_loss:   0.4668
Validation Accuracy:   0.7765
```

Here we plot *loss vs. epochs* and *accuracy vs. epochs* graphs.

```
figure(figsize=(8,8))
plt.plot(train_losses)
plt.plot(val_losses)
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()
```



```
figure(figsize=(8,8))
plt.plot(train_accuracies)
plt.plot(val_accuracies)
plt.title('Model Accuracy')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



Here we calculate the accuracy of our test set and show its classification report.

```
tfidfX_test = tfidfX_test.to(cuda_device)
Y_test = Y_test.to(cuda_device)

# Forward pass to get output
```

```
test_prediction = ffnn_model.forward(tfidfX_test.float())
test_prediction = torch.squeeze(test_prediction)

#Calculate accuracy on test set
test_accuracy = ((test_prediction.round() == Y_test).sum()/float(test_prediction.shape[0]))

print("Test Accuracy:",round(test_accuracy.item(),4), "\n")
```

```
Test Accuracy: 0.7792
```

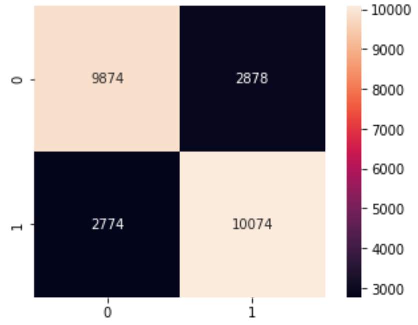
```
#Show the classification report
test_prediction = test_prediction.to(cuda_device)
test_prediction = test_prediction.ge(.5).view(-1).cpu()
Y_test = Y_test.cpu()

print(classification_report(Y_test,test_prediction))
```

```
              precision    recall  f1-score   support

         0.0       0.78      0.77      0.78     12752
         1.0       0.78      0.78      0.78     12848

    accuracy                           0.78     25600
   macro avg       0.78      0.78      0.78     25600
weighted avg       0.78      0.78      0.78     25600
```

```
# Plot/draw confusion matrix (heatmap)

from sklearn.metrics._plot.confusion_matrix import confusion_matrix

conf_matrix = confusion_matrix(Y_test, test_prediction)
```

```
sns.heatmap(conf_matrix,square=True, annot=True, fmt="d")
```

```
<AxesSubplot:>
```



Observations:

Analysis of the results:

- 9874 Negative texts have been correctly classified
- 10074 Positive texts have been correctly classified
- 2878 Negative texts have been classified as Positive texts
- 2774 Positive texts have been classified as Negative texts

The model **accuracy is 0.77** for our large dataset, which is good but can be improved.

We managed to avoid **overfitting** and **underfitting**. I choose the different values for the hidden layers and **epoch of 60**, since higher epoch would cause overfitting for my case.

Side Note: We also needed to take in to the consideration that our dataset was very large (millions), and we had to tweak some data and parameters to achieve our results without running into crashed memory in Colab.

References Used:

https://github.com/alexaapo/Feed-Forward-Neural-Network/blob/main/1st_model_with_tfidf(1).ipynb

https://towardsdatascience.com/feed-forward-neural-networks-how-to-successfully-build-them-in-python-74503409d99a

https://pytorch.org/docs/stable/notes/cuda.html

Colab paid products - Cancel contracts here

✓ 0s completed at 10:42 AM ● ✕