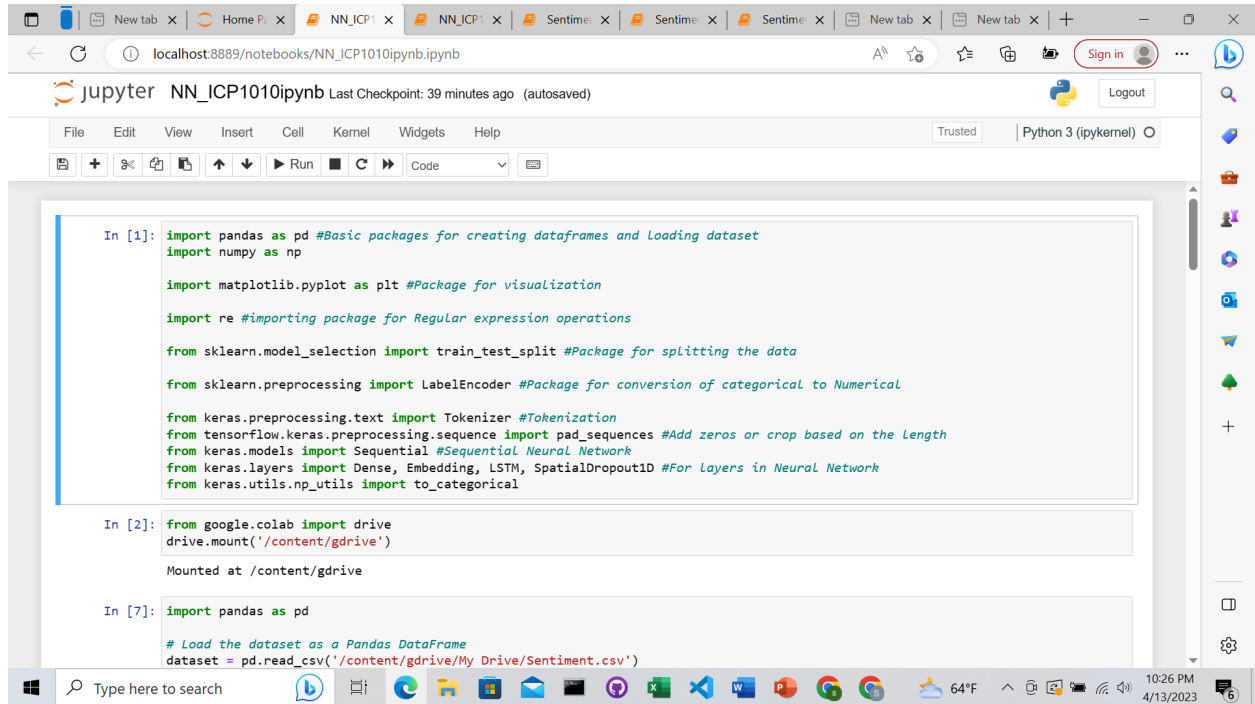


## NN&DeepLearning\_ICP10: LSTM

### In class programming:

1. Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump”)



```
In [1]: import pandas as pd #Basic packages for creating dataframes and Loading dataset
import numpy as np

import matplotlib.pyplot as plt #Package for visualization

import re #importing package for Regular expression operations

from sklearn.model_selection import train_test_split #Package for splitting the data

from sklearn.preprocessing import LabelEncoder #Package for conversion of categorical to Numerical

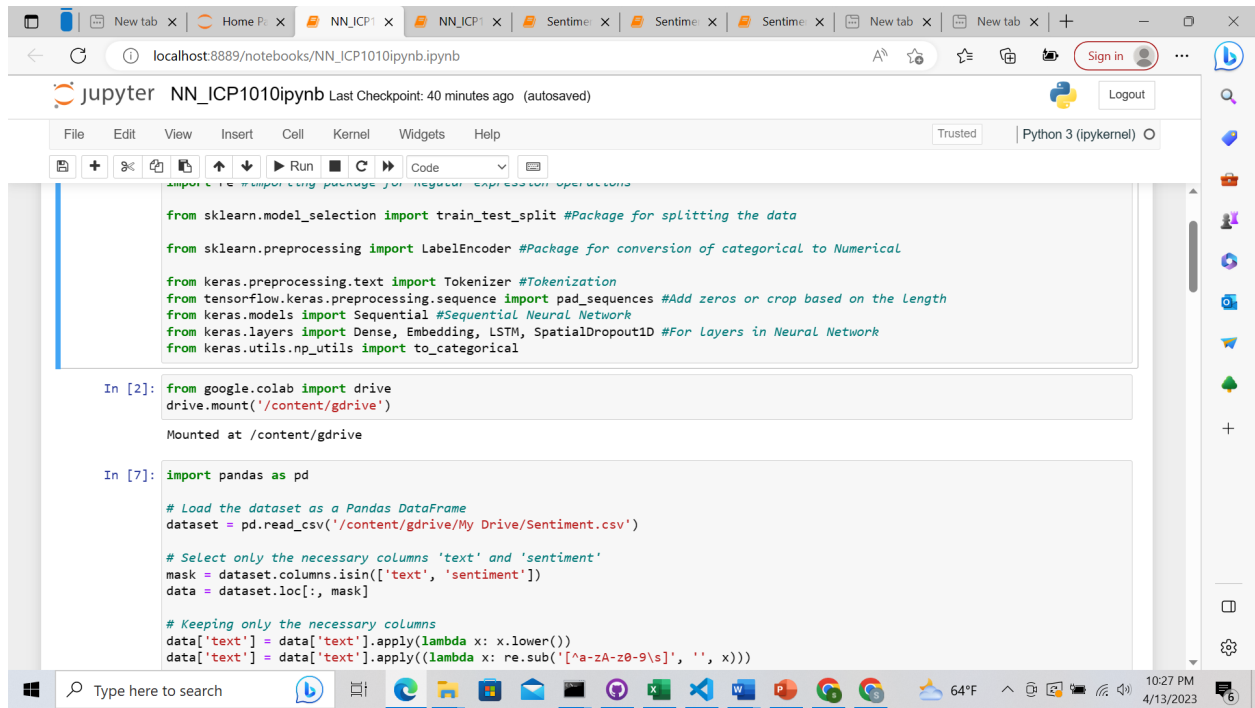
from keras.preprocessing.text import Tokenizer #Tokenization
from tensorflow.keras.preprocessing.sequence import pad_sequences #Add zeros or crop based on the Length
from keras.models import Sequential #Sequential Neural Network
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D #For Layers in Neural Network
from keras.utils.np_utils import to_categorical

In [2]: from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

In [7]: import pandas as pd

# Load the dataset as a Pandas DataFrame
dataset = pd.read_csv('/content/gdrive/My Drive/Sentiment.csv')
```



```
from sklearn.model_selection import train_test_split #Package for splitting the data

from sklearn.preprocessing import LabelEncoder #Package for conversion of categorical to Numerical

from keras.preprocessing.text import Tokenizer #Tokenization
from tensorflow.keras.preprocessing.sequence import pad_sequences #Add zeros or crop based on the Length
from keras.models import Sequential #Sequential Neural Network
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D #For Layers in Neural Network
from keras.utils.np_utils import to_categorical

In [2]: from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

In [7]: import pandas as pd

# Load the dataset as a Pandas DataFrame
dataset = pd.read_csv('/content/gdrive/My Drive/Sentiment.csv')

# Select only the necessary columns 'text' and 'sentiment'
mask = dataset.columns.isin(['text', 'sentiment'])
data = dataset.loc[:, mask]

# Keeping only the necessary columns
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x))
```

```

In [8]: for idx, row in data.iterrows():
        row[0] = row[0].replace('rt', ' ') #Removing Retweets
        max_features = 2000

        tokenizer = Tokenizer(num_words=max_features, split=' ') #Maximum words is 2000 to tokenize sentence
        tokenizer.fit_on_texts(data['text'].values)
        X = tokenizer.texts_to_sequences(data['text'].values) #taking values to feature matrix
        X = pad_sequences(X) #Padding the feature matrix

        embed_dim = 128 #Dimension of the Embedded Layer
        lstm_out = 196 #Long short-term memory (LSTM) Layer neurons
        def createmodel():
            model = Sequential() #Sequential Neural Network
            model.add(Embedding(max_features, embed_dim, input_length = X.shape[1])) #input dimension 2000 Neurons, output dimension 128 Neurons
            model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2)) #Drop out 20%, 196 output Neurons, recurrent dropout 20%
            model.add(Dense(3, activation='softmax')) #3 output neurons[positive, Neutral, Negative], softmax as activation
            model.compile(loss = 'categorical_crossentropy', optimizer='adam', metrics = ['accuracy']) #Compiling the model
            return model

        # print(model.summary())
        labelencoder = LabelEncoder() #Applying Label Encoding on the Label matrix
        integer_encoded = labelencoder.fit_transform(data['sentiment']) #fitting the model
        y = to_categorical(integer_encoded)
        X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.33, random_state = 42) #67% training data, 33% test data split
        batch_size = 32 #Batch size 32
        model = createmodel() #Function call to Sequential Neural Network
        model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2) #verbose the higher, the more messages
        score, acc = model.evaluate(X_test, Y_test, verbose=2, batch_size=batch_size) #evaluating the model
        print(score)
        print(acc)

```

```

LabelEncoder = LabelEncoder() #Applying Label Encoding on the Label matrix
integer_encoded = labelencoder.fit_transform(data['sentiment']) #fitting the model
y = to_categorical(integer_encoded)
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.33, random_state = 42) #67% training data, 33% test data split
batch_size = 32 #Batch size 32
model = createmodel() #Function call to Sequential Neural Network
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2) #verbose the higher, the more messages
score, acc = model.evaluate(X_test, Y_test, verbose=2, batch_size=batch_size) #evaluating the model
print(score)
print(acc)

291/291 - 42s - loss: 0.8306 - accuracy: 0.6441 - 42s/epoch - 144ms/step
144/144 - 3s - loss: 0.7514 - accuracy: 0.6791 - 3s/epoch - 22ms/step
0.7513718008995056
0.6791175007820129

In [9]: print(model.metrics_names) #metrics of the model
['loss', 'accuracy']

In [10]: #1. Save the model and use the saved model to predict on new text data (ex, "A Lot of good things are happening. We are respected

```

```

In [11]: model.save('sentimentAnalysis.h5') #Saving the model

In [12]: from keras.models import load_model #Importing the package for importing the saved model
         model = load_model('sentimentAnalysis.h5') #Loading the saved model

```

Jupyter NN\_ICP1010ipynb Last Checkpoint: 42 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
['loss', 'accuracy']

In [10]: #1. Save the model and use the saved model to predict on new text data (ex, "A Lot of good things are happening. We are respected

In [11]: model.save('sentimentAnalysis.h5') #Saving the model

In [12]: from keras.models import load_model #Importing the package for importing the saved model
model= load_model('sentimentAnalysis.h5') #Loading the saved model

In [13]: print(integer_encoded)
print(data['sentiment'])

[1 2 1 ... 2 0 2]
0      Neutral
1      Positive
2      Neutral
3      Positive
4      Positive
...
13866   Negative
13867   Positive
13868   Positive
13869   Negative
13870   Positive
Name: sentiment, Length: 13871, dtype: object
```

Thursday, April 13, 2023 10:29 PM 4/13/2023

Jupyter NN\_ICP1010ipynb Last Checkpoint: 43 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
13867   Positive
13868   Positive
13869   Negative
13870   Positive
Name: sentiment, Length: 13871, dtype: object

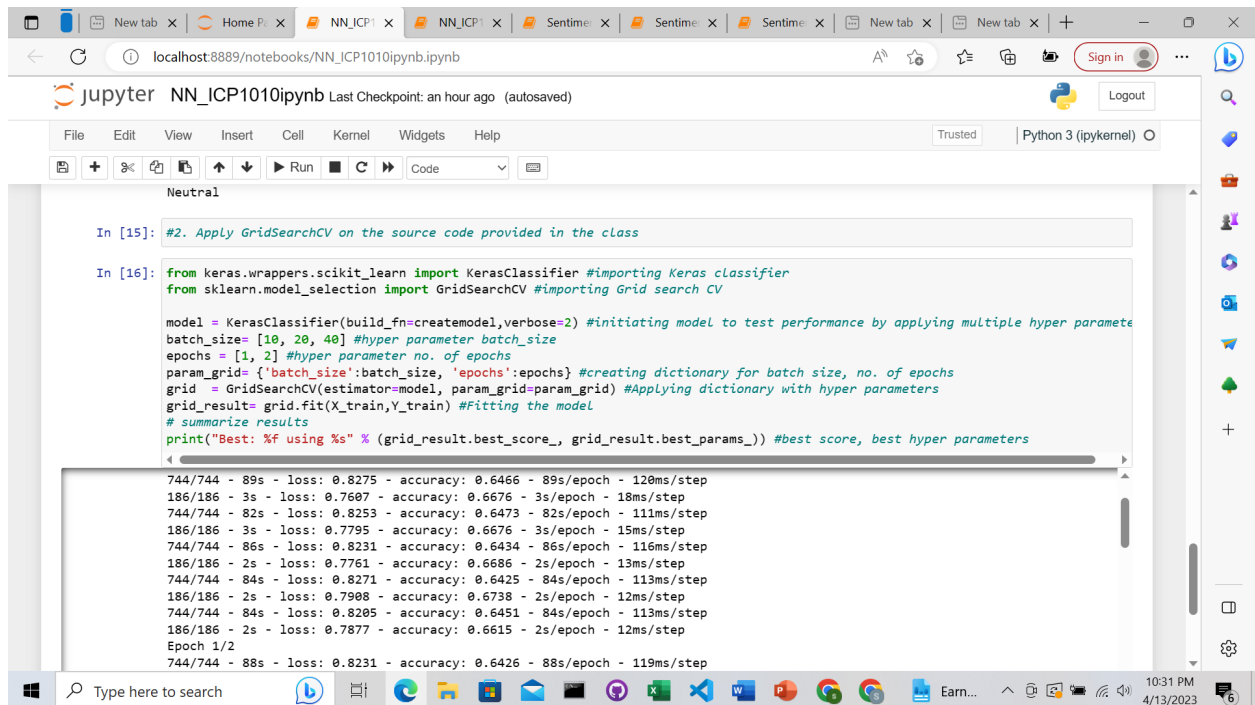
In [14]: # Predicting on the text data
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']
sentence = tokenizer.texts_to_sequences(sentence) # Tokenizing the sentence
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0) # Padding the sentence
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0] # Predicting the sentence text
sentiment = np.argmax(sentiment_probs)

print(sentiment_probs)
if sentiment == 0:
    print("Neutral")
elif sentiment < 0:
    print("Negative")
elif sentiment > 0:
    print("Positive")
else:
    print("Cannot be determined")

1/1 - 0s - 270ms/epoch - 270ms/step
[0.72844136 0.10584743 0.16571125]
Neutral
```

Thursday, April 13, 2023 10:29 PM 4/13/2023

## 2. Apply GridSearchCV on the source code provided in the class.

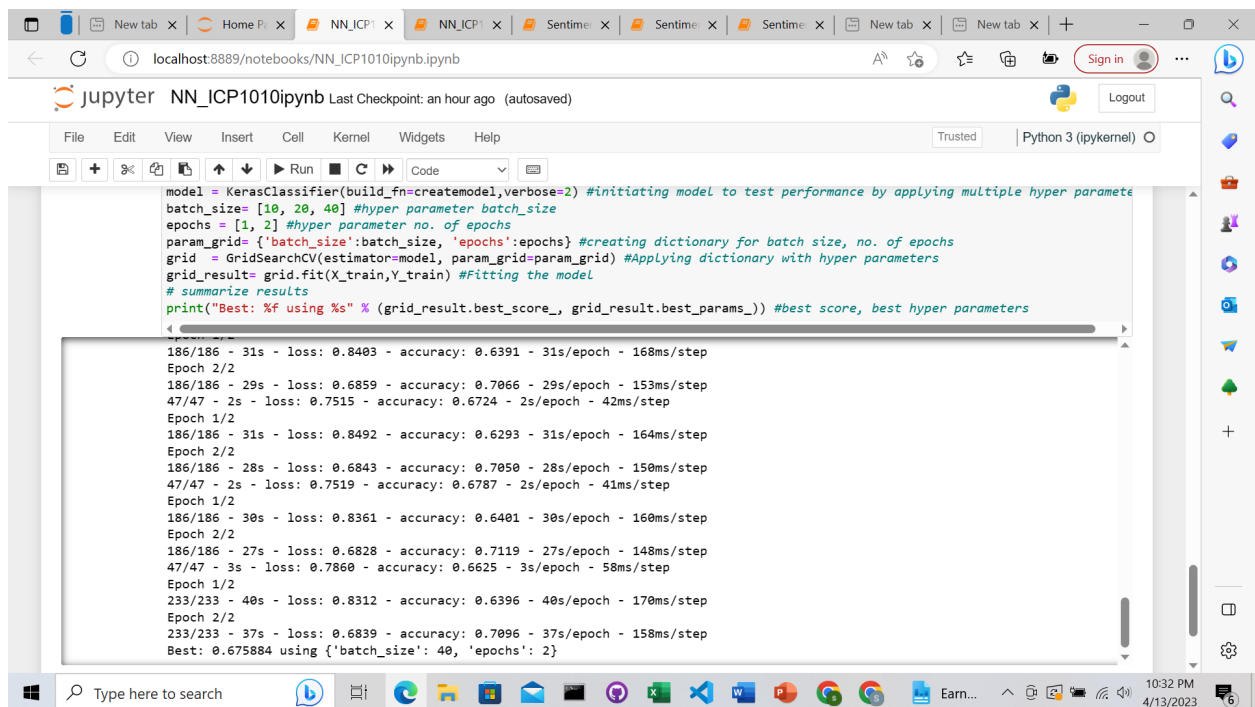


```
In [15]: #2. Apply GridSearchCV on the source code provided in the class

In [16]: from keras.wrappers.scikit_learn import KerasClassifier #importing Keras classifier
from sklearn.model_selection import GridSearchCV #importing Grid search CV

model = KerasClassifier(build_fn=createmodel,verbose=2) #initiating model to test performance by applying multiple hyper parameters
batch_size=[10, 20, 40] #hyper parameter batch_size
epochs=[1, 2] #hyper parameter no. of epochs
param_grid={'batch_size':batch_size, 'epochs':epochs} #creating dictionary for batch size, no. of epochs
grid = GridSearchCV(estimator=model, param_grid=param_grid) #Applying dictionary with hyper parameters
grid_result= grid.fit(X_train,Y_train) #Fitting the model
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)) #best score, best hyper parameters

4
744/744 - 89s - loss: 0.8275 - accuracy: 0.6466 - 89s/epoch - 120ms/step
186/186 - 3s - loss: 0.7607 - accuracy: 0.6676 - 3s/epoch - 18ms/step
744/744 - 82s - loss: 0.8253 - accuracy: 0.6473 - 82s/epoch - 111ms/step
186/186 - 3s - loss: 0.7795 - accuracy: 0.6676 - 3s/epoch - 15ms/step
744/744 - 86s - loss: 0.8231 - accuracy: 0.6434 - 86s/epoch - 116ms/step
186/186 - 2s - loss: 0.7761 - accuracy: 0.6686 - 2s/epoch - 13ms/step
744/744 - 84s - loss: 0.8271 - accuracy: 0.6425 - 84s/epoch - 113ms/step
186/186 - 2s - loss: 0.7908 - accuracy: 0.6738 - 2s/epoch - 12ms/step
744/744 - 84s - loss: 0.8205 - accuracy: 0.6451 - 84s/epoch - 113ms/step
186/186 - 2s - loss: 0.7877 - accuracy: 0.6615 - 2s/epoch - 12ms/step
Epoch 1/2
744/744 - 88s - loss: 0.8231 - accuracy: 0.6426 - 88s/epoch - 119ms/step
```



```
model = KerasClassifier(build_fn=createmodel,verbose=2) #initiating model to test performance by applying multiple hyper parameters
batch_size=[10, 20, 40] #hyper parameter batch_size
epochs=[1, 2] #hyper parameter no. of epochs
param_grid={'batch_size':batch_size, 'epochs':epochs} #creating dictionary for batch size, no. of epochs
grid = GridSearchCV(estimator=model, param_grid=param_grid) #Applying dictionary with hyper parameters
grid_result= grid.fit(X_train,Y_train) #Fitting the model
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)) #best score, best hyper parameters

186/186 - 31s - loss: 0.8403 - accuracy: 0.6391 - 31s/epoch - 168ms/step
Epoch 2/2
186/186 - 29s - loss: 0.6859 - accuracy: 0.7066 - 29s/epoch - 153ms/step
47/47 - 2s - loss: 0.7515 - accuracy: 0.6724 - 2s/epoch - 42ms/step
Epoch 1/2
186/186 - 31s - loss: 0.8492 - accuracy: 0.6293 - 31s/epoch - 164ms/step
Epoch 2/2
186/186 - 28s - loss: 0.6843 - accuracy: 0.7050 - 28s/epoch - 150ms/step
47/47 - 2s - loss: 0.7519 - accuracy: 0.6787 - 2s/epoch - 41ms/step
Epoch 1/2
186/186 - 30s - loss: 0.8361 - accuracy: 0.6401 - 30s/epoch - 160ms/step
Epoch 2/2
186/186 - 27s - loss: 0.6828 - accuracy: 0.7119 - 27s/epoch - 148ms/step
47/47 - 3s - loss: 0.7860 - accuracy: 0.6625 - 3s/epoch - 58ms/step
Epoch 1/2
233/233 - 40s - loss: 0.8312 - accuracy: 0.6396 - 40s/epoch - 170ms/step
Epoch 2/2
233/233 - 37s - loss: 0.6839 - accuracy: 0.7096 - 37s/epoch - 158ms/step
Best: 0.675884 using {'batch_size': 40, 'epochs': 2}
```