Neural Network ICP 07

Bhavani Bhimavaram
700752330

Github link:https://github.com/Bhavani2023ICC/Neural_network_ICP7

In [1]:
```
!pip uninstall -y tensorflow keras pandas matplotlib scikit-learn
!pip install tensorflow==2.15.0 keras==2.15.0 pandas==2.0.3 matplotlib==3.9.1 scikit-learn==1.5.1
```

```
Found existing installation: tensorflow 2.15.0
Uninstalling tensorflow-2.15.0:
  Successfully uninstalled tensorflow-2.15.0
Found existing installation: keras 2.15.0
Uninstalling keras-2.15.0:
  Successfully uninstalled keras-2.15.0
Found existing installation: pandas 2.0.3
Uninstalling pandas-2.0.3:
  Successfully uninstalled pandas-2.0.3
Found existing installation: matplotlib 3.9.1
Uninstalling matplotlib-3.9.1:
  Successfully uninstalled matplotlib-3.9.1
Found existing installation: scikit-learn 1.5.1
Uninstalling scikit-learn-1.5.1:
  Successfully uninstalled scikit-learn-1.5.1
Collecting tensorflow==2.15.0
  Using cached tensorflow-2.15.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (475.2 MB)
Collecting keras==2.15.0
  Using cached keras-2.15.0-py3-none-any.whl (1.7 MB)
```

In [10]:
```
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 tensorflow.keras.preprocessing.text import Tokenizer # Tokenization
from tensorflow.keras.preprocessing.sequence import pad_sequences # Add zeros or crop based on the length
from tensorflow.keras.models import Sequential # Sequential Neural Network
```

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

# Load the dataset as a Pandas DataFrame
path_to_csv = '/content/sample_data/Sentiment.csv'
dataset = pd.read_csv(path_to_csv, header=0)

# 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))

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
```

```
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
    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

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 spl

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)
print(model.metrics_names) # metrics of the model
print(integer_encoded)
print(data['sentiment'])

# 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.@realDona
sentence = tokenizer.texts_to_sequences(sentence) # Tokenizing the sentence
sentence = pad_sequences(sentence, maxlen=X.shape[1], 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 == 1:
    print("Negative")
else:
    print("Positive")

# Custom wrapper for Keras model
from sklearn.base import BaseEstimator, ClassifierMixin

class CustomKerasClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, build_fn=None, epochs=1, batch_size=32, verbose=1, **sk_params):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.verbose = verbose
        self.sk_params = sk_params
        self.model = None

    def fit(self, X, y, **kwargs):
        self.model = self.build_fn()
        return self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size, verbose=self.verbose, **kwargs)

    def predict(self, X, **kwargs):
        return self.model.predict(X, **kwargs)

    def predict_proba(self, X, **kwargs):
        return self.model.predict(X, **kwargs)

    def score(self, X, y, **kwargs):
        _, accuracy = self.model.evaluate(X, y, verbose=0)
        return accuracy

# Use the custom Keras classifier
model = CustomKerasClassifier(build_fn=createmodel, verbose=2)
batch_size = [10, 20, 40]
epochs = [1, 2]
param_grid = {'batch_size': batch_size, 'epochs': epochs}
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)
```

```
        _, accuracy = self.model.evaluate(X, y, verbose=0)
        return accuracy

# Use the custom Keras classifier
model = CustomKerasClassifier(build_fn=createmodel, verbose=2)
batch_size = [10, 20, 40]
epochs = [1, 2]
param_grid = {'batch_size': batch_size, 'epochs': epochs}
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)

# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
291/291 - 46s - loss: 0.8224 - accuracy: 0.6441 - 46s/epoch - 159ms/step
144/144 - 3s - loss: 0.7604 - accuracy: 0.6654 - 3s/epoch - 20ms/step
0.7604355216026306
0.6653560400009155
['loss', 'accuracy']
[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
```