ICP9

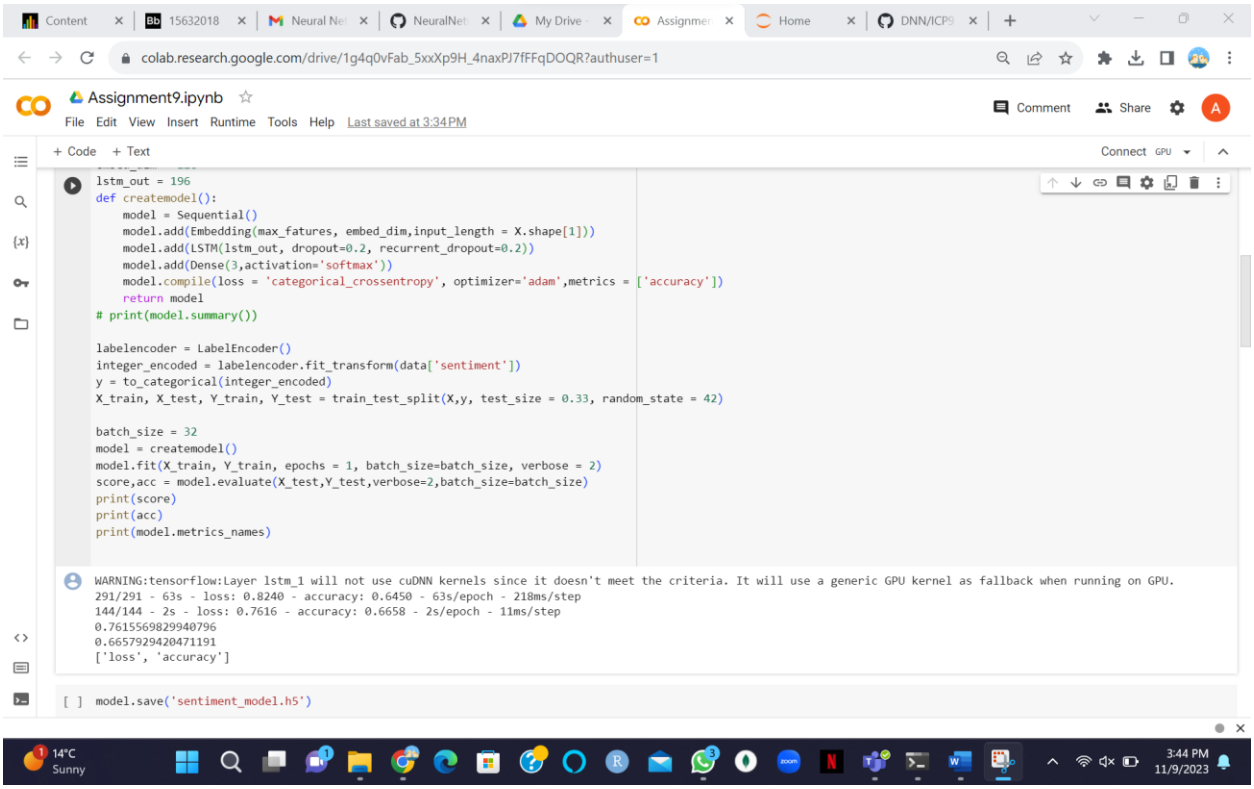
Ushakiran Yadav Avula

700746114

GitHub Link: https://github.com/ushakiranyadav/DNN/upload/main/ICP9

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re

from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('Sentiment.csv')
# Keeping only the neccessary columns
data = data[['text','sentiment']]

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', ' ')

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)

X = pad_sequences(X)

embed_dim = 128
lstm_out = 196
def createmodel():
    model = Sequential()
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3,activation='softmax'))
```

```
    model.compile(loss = 'categorical_crossentropy',
optimizer='adam',metrics = ['accuracy'])
    return model
# print(model.summary())

labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['sentiment'])
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)

batch_size = 32
model = createmodel()
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose =
2)
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)
print(score)
print(acc)
print(model.metrics_names)
```



```
model.save('sentiment_model.h5')
```

This code loads the saved model using the load_model function, and then preprocesses the new text data in the same way as the training data. The predict method is called on the loaded model to get the predicted class probabilities for the new text data. The class with the highest probability is chosen as the predicted sentiment. The predicted sentiment and probabilities are then printed to the console. To apply GridSearchCV on the provided source code, we can use the GridSearchCV class from sklearn to search for the best combination of hyperparameters for the LSTM model. The hyperparameters that can be tuned are the number of LSTM units, the dropout rate, and the learning rate of the optimizer

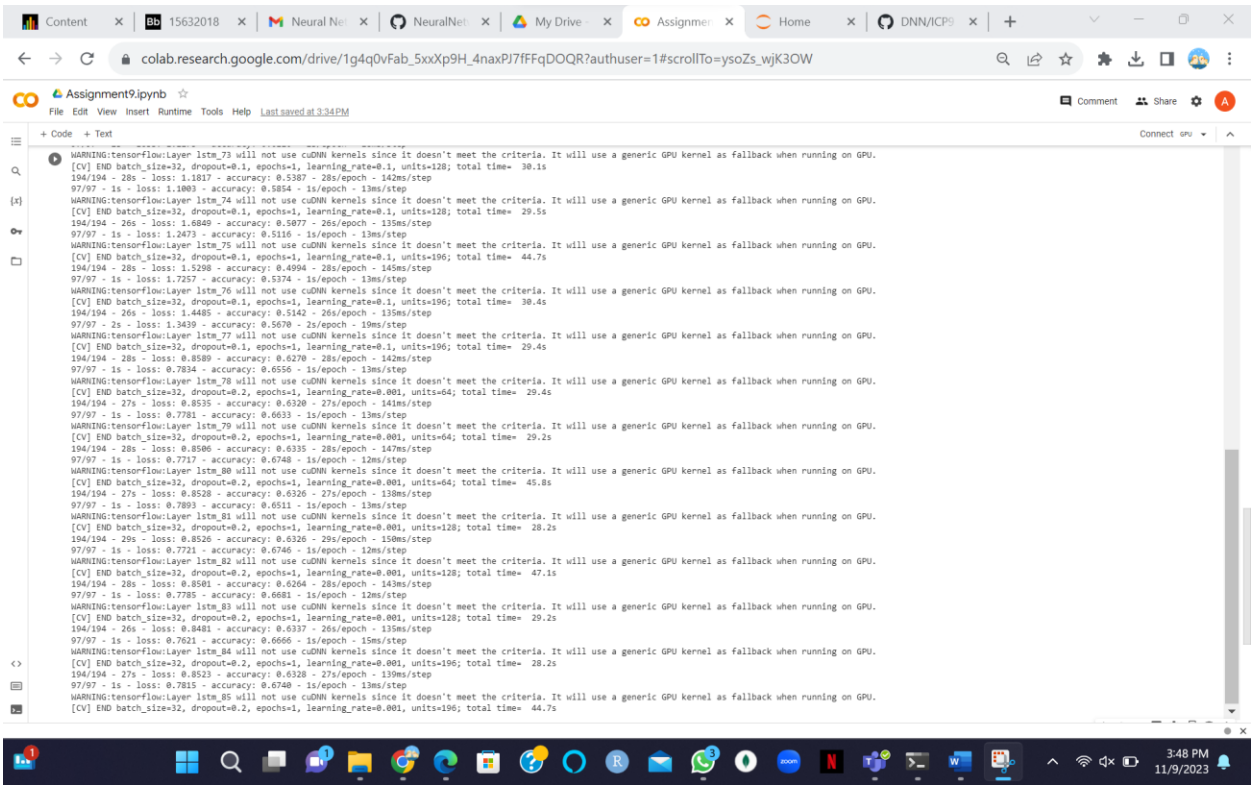
```
from keras.models import load_model
import numpy as np

loaded_model = load_model('sentiment_model.h5')

new_text = ["A lot of good things are happening. We are respected again
throughout the world, and that's a great thing.@realDonaldTrump"]
new_text = tokenizer.texts_to_sequences(new_text)
new_text = pad_sequences(new_text, maxlen=X.shape[1], dtype='int32',
value=0)
sentiment_prob = loaded_model.predict(new_text, batch_size=1,
verbose=2)[0]

sentiment_classes = ['Negative', 'Neutral', 'Positive']
sentiment_pred = sentiment_classes[np.argmax(sentiment_prob)]

print("Predicted sentiment: ", sentiment_pred)
print("Predicted probabilities: ", sentiment_prob)
```

```
from keras.models import load_model
import numpy as np

loaded_model = load_model('sentiment_model.h5')

new_text = ["A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump"]
new_text = tokenizer.texts_to_sequences(new_text)
new_text = pad_sequences(new_text, maxlen=X.shape[1], dtype='int32', value=0)
sentiment_prob = loaded_model.predict(new_text, batch_size=1, verbose=2)[0]

sentiment_classes = ['Negative', 'Neutral', 'Positive']
sentiment_pred = sentiment_classes[np.argmax(sentiment_prob)]

print("Predicted sentiment: ", sentiment_pred)
print("Predicted probabilities: ", sentiment_prob)
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
1/1 - 0s - 428ms/epoch - 428ms/step
Predicted sentiment:  Positive
Predicted probabilities:  [0.39611092 0.19630554 0.4075835 ]
```



```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.optimizers import Adam

def create_model(units=196, dropout=0.2, learning_rate=0.001):
    model = Sequential()
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
    model.add(LSTM(units, dropout=dropout, recurrent_dropout=dropout))
    model.add(Dense(3, activation='softmax'))
    optimizer = Adam(lr=learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model, verbose=2)

units = [64, 128, 196]
dropout = [0.1, 0.2, 0.3]
learning_rate = [0.001, 0.01, 0.1]
epochs = [1]
batch_size = [32]

param_grid = dict(units=units, dropout=dropout,
learning_rate=learning_rate, epochs=epochs, batch_size=batch_size)
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,
verbose=2)
grid_result = grid.fit(X_train, Y_train)

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



This code defines the create_model function that returns a Keras model with the specified hyperparameters. The KerasClassifier class is used to create a wrapper for the create_model function, which can be used as an estimator for GridSearchCV. The hyperparameters to be tuned are defined in the param_grid dictionary. GridSearchCV is then called with the KerasClassifier object, the param_grid dictionary