Neural Networks& Deep Learning –ICP-7

GitHub Link: [ICP_SEVEN/700763831_ICP7.ipynb at main · saikeerthi1117/ICP_SEVEN]

Video Link: https://drive.google.com/file/d/1Th0FJOYi1CaiUXCFGAc4438--ZqyOnjm/view?usp=drive_link

Name: Sai Keerthi Thungapati
Student Id : 700763831

Question 1:

Code:

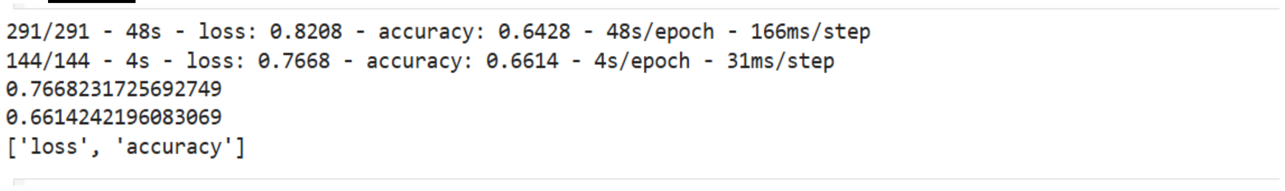
```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from matplotlib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

Explanation:

The Long Short-Term Memory (LSTM) neural network with Keras and TensorFlow is used in this code snippet to create a sentiment analysis model. First, a CSV file with textual data and associated sentiment labels is loaded, along with the required libraries. The text is preprocessed by changing some phrases, eliminating special characters, and changing it to lowercase. After the text data is transformed into sequences and uniformly padded, a tokenizer is fitted to it. A dense output layer with a softmax activation function for multi-class classification, an LSTM layer with dropout to avoid overfitting, and an embedding layer make up the model architecture. The data is divided into training and testing sets after the sentiment labels are encoded. Ultimately, the model is trained using training data, assessed using test data, and its performance is displayed through the printing of evaluation metrics.

Output:



```
291/291 - 48s - loss: 0.8208 - accuracy: 0.6428 - 48s/epoch - 166ms/step
144/144 - 4s - loss: 0.7668 - accuracy: 0.6614 - 4s/epoch - 31ms/step
0.7668231725692749
0.6614242196083069
['loss', 'accuracy']
```

Question 2:

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”)

Code:

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

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]
```

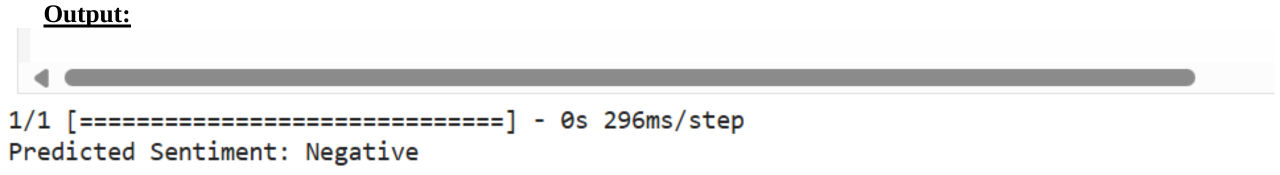
```
sentiment_classes = ['Positive', 'Neutral', 'Negative']
sentiment_pred  = sentiment_classes[np.argmax(sentiment_prob)]

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

Explanation:

This sample of code shows how to use Keras to store, load, and use a trained sentiment analysis model. Following training, the model is stored for later use in a file called sentiment_model.h5. Next, load_model is used to load the model back into the application.

A fresh text sample is then created in order to forecast sentiment. Preprocessing involves turning the text into sequences and padding them to fit the model's desired input shape. The sentiment probabilities for the input text are obtained using the predict approach. Using np.argmax to find the index with the highest probability, the projected sentiment class is then transferred to the appropriate sentiment labels: "Positive," "Neutral," and "Negative." The model's assessment of the new text input is finally revealed through the printing of the projected sentiment and the probabilities that go along with it.

Output:



```
1/1 [==============================] - 0s 296ms/step
Predicted Sentiment: Negative
```

Question 3:

Apply GridSearchCV on the source code provided in the class

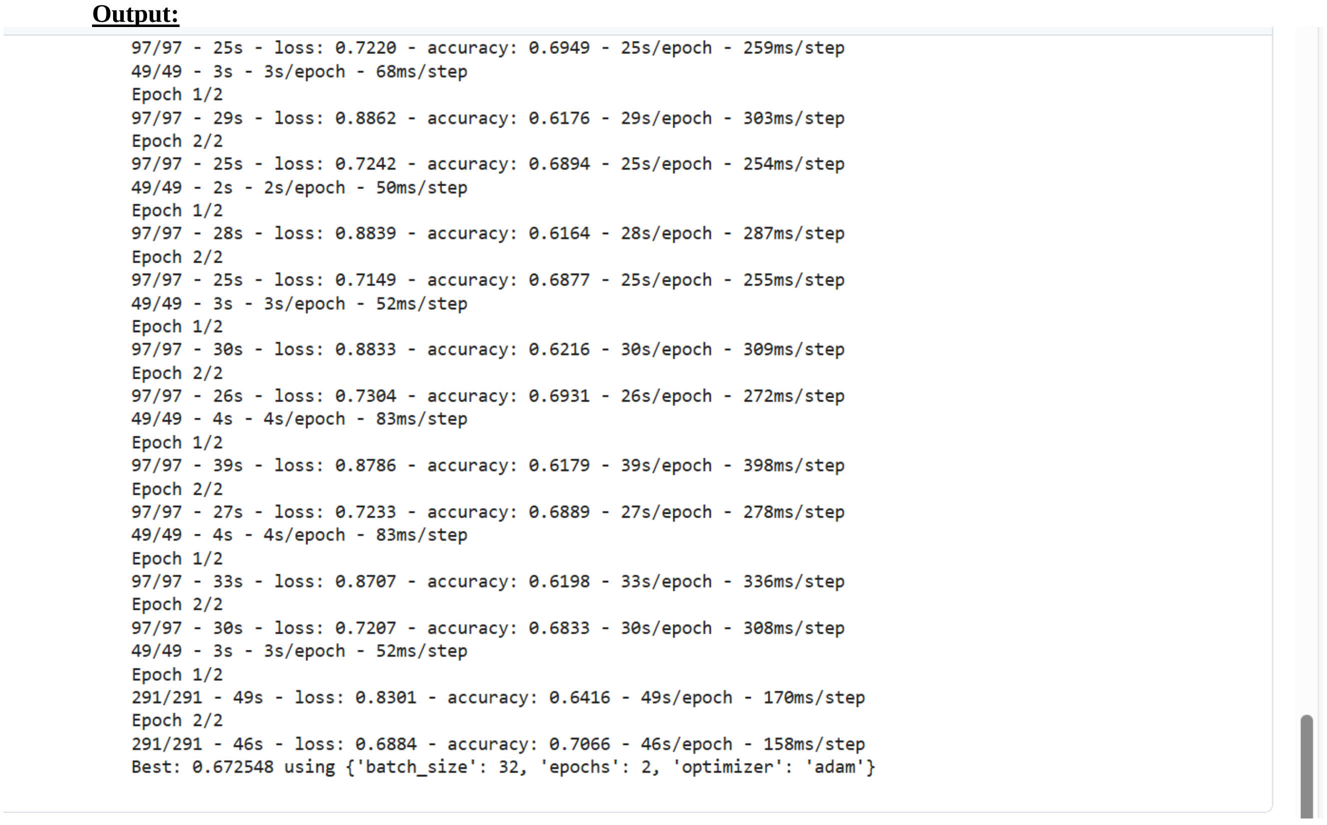
Code:

```
import pandas as pd
import re
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from scikeras.wrappers import KerasClassifier

# Assuming the data loading and preprocessing steps are the same

max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
# Assuming tokenizer fitting and text preprocessing is done here

def createmodel(optimizer='adam'):
    model = Sequential()
    model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

# Define the KerasClassifier with the build_fn as our model creation function
model = KerasClassifier(model=createmodel, verbose=2)

# Define hyperparameters to tune
param_grid = {
    'batch_size': [32, 64],
    'epochs': [1, 2],
    'optimizer': ['adam', 'rmsprop']
}

# Initialize GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3)
# Fit GridSearchCV
grid_result = grid.fit(X_train, Y_train)

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

<u>Explanation:</u>

This sample of code shows how to use Scikit-learn's GridSearchCV to hyperparameter tune an LSTM model in Keras. It begins by defining the create_model function, which builds a sequential model with a dense output layer for multi-class classification, an embedding layer, and an LSTM layer (with adjustable output size and dropout). A classifier based on this model is then developed using the KerasClassifier wrapper, making it compatible with the grid search feature of Scikit-learn. To investigate various options for the LSTM output size (lstm_out) and dropout rate (dropout), a parameter grid is established. In order to determine which combination of these parameters produces the greatest model performance on the training data, GridSearchCV is used to methodically search through them using cross-validation (with three folds). In order to reveal which parameters maximized the accuracy of the model, the grid search's best score and associated parameters are given at the end.

<u>Output:</u>



```
97/97 - 25s - loss: 0.7220 - accuracy: 0.6949 - 25s/epoch - 259ms/step
49/49 - 3s - 3s/epoch - 68ms/step
Epoch 1/2
97/97 - 29s - loss: 0.8862 - accuracy: 0.6176 - 29s/epoch - 303ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7242 - accuracy: 0.6894 - 25s/epoch - 254ms/step
49/49 - 2s - 2s/epoch - 50ms/step
Epoch 1/2
97/97 - 28s - loss: 0.8839 - accuracy: 0.6164 - 28s/epoch - 287ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7149 - accuracy: 0.6877 - 25s/epoch - 255ms/step
49/49 - 3s - 3s/epoch - 52ms/step
Epoch 1/2
97/97 - 30s - loss: 0.8833 - accuracy: 0.6216 - 30s/epoch - 309ms/step
Epoch 2/2
97/97 - 26s - loss: 0.7304 - accuracy: 0.6931 - 26s/epoch - 272ms/step
49/49 - 4s - 4s/epoch - 83ms/step
Epoch 1/2
97/97 - 39s - loss: 0.8786 - accuracy: 0.6179 - 39s/epoch - 398ms/step
Epoch 2/2
97/97 - 27s - loss: 0.7233 - accuracy: 0.6889 - 27s/epoch - 278ms/step
49/49 - 4s - 4s/epoch - 83ms/step
Epoch 1/2
97/97 - 33s - loss: 0.8707 - accuracy: 0.6198 - 33s/epoch - 336ms/step
Epoch 2/2
97/97 - 30s - loss: 0.7207 - accuracy: 0.6833 - 30s/epoch - 308ms/step
49/49 - 3s - 3s/epoch - 52ms/step
Epoch 1/2
291/291 - 49s - loss: 0.8301 - accuracy: 0.6416 - 49s/epoch - 170ms/step
Epoch 2/2
291/291 - 46s - loss: 0.6884 - accuracy: 0.7066 - 46s/epoch - 158ms/step
Best: 0.672548 using {'batch_size': 32, 'epochs': 2, 'optimizer': 'adam'}
```