Video Link:

https://drive.google.com/file/d/1MkVIPjCfHb0l_JiHcqFcQQ1GD-g0yvMb/view?usp=drive_link

Git-hub Link: Sabitha-C/Neural-networks (github.com)

Code:

```
import pandas as pd
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical  # Updated import
from sklearn.preprocessing import LabelEncoder
import re

# Load and preprocess data
data = pd.read_csv('Sentiment.csv')
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))
data['text'] = data['text'].apply(lambda x: x.replace('rt', ' '))

# Tokenization and padding
max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)

# Encode labels
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(data['sentiment'])
y = to_categorical(integer_encoded)

# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Define LSTM model
embed_dim = 128
lstm_out = 196

def create_model():
    model = Sequential()
    model.add(Embedding(max_features, 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
```

```python
# Create and train the model
batch_size = 32
epochs = 5
model = create_model()
model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, verbose=2)

# Evaluate the model
score, acc = model.evaluate(X_test, Y_test, verbose=2, batch_size=batch_size)
print("Test score:", score)
print("Test accuracy:", acc)

# Save the model
model.save("sentiment_analysis_model.h5")
print("Model saved to disk.")

# Load the model for prediction
loaded_model = load_model("sentiment_analysis_model.h5")

# Example of predicting new text data
new_texts = [
    "A lot of good things are happening. We are respected again throughout the world, and that's a great thing. @realDonaldTrump"
]

# Preprocess new text data
new_texts = [text.lower() for text in new_texts]
new_texts = [re.sub('[^a-zA-z0-9\s]', '', text) for text in new_texts]
sequences = tokenizer.texts_to_sequences(new_texts)
padded_sequences = pad_sequences(sequences, maxlen=X.shape[1])

# Predict sentiment
predictions = loaded_model.predict(padded_sequences)
sentiments = label_encoder.inverse_transform([predictions.argmax(axis=-1)])

# Print predictions
for text, sentiment in zip(new_texts, sentiments):
    print(f'Text: {text}')
    print(f'Predicted Sentiment: {sentiment}')
    print()
```

Output:

```
Epoch 1/5
291/291 - 51s - loss: 0.8285 - accuracy: 0.6490 - 51s/epoch - 174ms/step
Epoch 2/5
291/291 - 47s - loss: 0.6817 - accuracy: 0.7096 - 47s/epoch - 162ms/step
Epoch 3/5
291/291 - 47s - loss: 0.6195 - accuracy: 0.7381 - 47s/epoch - 162ms/step
Epoch 4/5
291/291 - 45s - loss: 0.5676 - accuracy: 0.7651 - 45s/epoch - 156ms/step
Epoch 5/5
291/291 - 47s - loss: 0.5224 - accuracy: 0.7830 - 47s/epoch - 162ms/step
144/144 - 4s - loss: 0.8423 - accuracy: 0.6706 - 4s/epoch - 30ms/step
Test score: 0.8423432111740112
Test accuracy: 0.6705985069274902
Model saved to disk.
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native
  saving_api.save_model(
1/1 [==============================] - 1s 829ms/step
Text: a lot of good things are happening we are respected again throughout the world and thats a great thing realdonaldtrump
Predicted Sentiment: Positive

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:155: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ra
  y = column_or_1d(y, warn=True)
```

Description:

I have read the file 'Sentiment.csv', trained the LSTM model for sentiment analysis and I saved the model to the disc and given a new input data to get the sentiment analysis. I have done the analysis and got the output as positive as the given data is positive.

```
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_))
```

Output:

```
194/194 - 37s - loss: 0.8596 - accuracy: 0.6328 - 37s/epoch - 192ms/step
97/97 - 2s - 2s/epoch - 23ms/step
194/194 - 41s - loss: 0.8563 - accuracy: 0.6297 - 41s/epoch - 210ms/step
97/97 - 3s - 3s/epoch - 34ms/step
194/194 - 36s - loss: 0.8773 - accuracy: 0.6278 - 36s/epoch - 186ms/step
97/97 - 2s - 2s/epoch - 23ms/step
194/194 - 32s - loss: 0.8712 - accuracy: 0.6326 - 32s/epoch - 167ms/step
97/97 - 3s - 3s/epoch - 28ms/step
194/194 - 33s - loss: 0.8588 - accuracy: 0.6292 - 33s/epoch - 171ms/step
97/97 - 3s - 3s/epoch - 27ms/step
194/194 - 34s - loss: 0.8675 - accuracy: 0.6252 - 34s/epoch - 173ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 33s - loss: 0.8632 - accuracy: 0.6300 - 33s/epoch - 171ms/step
Epoch 2/2
194/194 - 29s - loss: 0.7171 - accuracy: 0.6888 - 29s/epoch - 151ms/step
97/97 - 3s - 3s/epoch - 32ms/step
Epoch 1/2
194/194 - 33s - loss: 0.8599 - accuracy: 0.6271 - 33s/epoch - 170ms/step
Epoch 2/2
194/194 - 30s - loss: 0.6978 - accuracy: 0.6991 - 30s/epoch - 157ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 35s - loss: 0.8553 - accuracy: 0.6285 - 35s/epoch - 179ms/step
Epoch 2/2
194/194 - 29s - loss: 0.6883 - accuracy: 0.7022 - 29s/epoch - 151ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 35s - loss: 0.8565 - accuracy: 0.6320 - 35s/epoch - 178ms/step
Epoch 2/2
194/194 - 29s - loss: 0.7122 - accuracy: 0.6949 - 29s/epoch - 150ms/step
97/97 - 3s - 3s/epoch - 34ms/step
Epoch 1/2
194/194 - 33s - loss: 0.8660 - accuracy: 0.6295 - 33s/epoch - 168ms/step
Epoch 2/2
194/194 - 30s - loss: 0.7025 - accuracy: 0.6999 - 30s/epoch - 157ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 35s - loss: 0.8494 - accuracy: 0.6320 - 35s/epoch - 181ms/step
Epoch 2/2
194/194 - 30s - loss: 0.6845 - accuracy: 0.7093 - 30s/epoch - 156ms/step
97/97 - 3s - 3s/epoch - 30ms/step
```

```
97/97 - 30s - loss: 0.8820 - accuracy: 0.6182 - 30s/epoch - 309ms/step
49/49 - 3s - 3s/epoch - 51ms/step
97/97 - 28s - loss: 0.8731 - accuracy: 0.6228 - 28s/epoch - 290ms/step
49/49 - 3s - 3s/epoch - 52ms/step
97/97 - 30s - loss: 0.8955 - accuracy: 0.6165 - 30s/epoch - 307ms/step
49/49 - 2s - 2s/epoch - 51ms/step
97/97 - 29s - loss: 0.8696 - accuracy: 0.6263 - 29s/epoch - 298ms/step
49/49 - 2s - 2s/epoch - 50ms/step
97/97 - 29s - loss: 0.8740 - accuracy: 0.6218 - 29s/epoch - 304ms/step
49/49 - 3s - 3s/epoch - 65ms/step
97/97 - 28s - loss: 0.8783 - accuracy: 0.6241 - 28s/epoch - 289ms/step
49/49 - 3s - 3s/epoch - 67ms/step
Epoch 1/2
97/97 - 29s - loss: 0.8779 - accuracy: 0.6242 - 29s/epoch - 302ms/step
Epoch 2/2
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'}
```

Description:

I have used Gridsearchcv from scikit-learn to optimize hyperparameters for an LSTM-based sentiment analysis model. By specifying parameters such as batch size, epochs, and

optimizers, searched for the best to maximize accuracy on the training data. The final output shows the best accuracy achieved and the corresponding optimal hyperparameters discovered through the grid search process.