NEURAL NETWORK DEEP LEARNING

ICP 9

700755861

SREEJA MADHAGONI

GitHub: Repository URL for the source code:
https://github.com/SreejaMadhagoni/NNDL/tree/main/ICP%209

```
In [ ]: 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():
```

```
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)
```

Output:

```
print(score)
print(acc)
print(model.metrics_names)

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.
291/291 - 63s - loss: 0.8240 - accuracy: 0.6450 - 63s/epoch - 218ms/step
144/144 - 2s - loss: 0.7616 - accuracy: 0.6658 - 2s/epoch - 11ms/step
0.7615569829940796
0.6657929420471191
['loss', 'accuracy']

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

```
In [ ]: 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.

```
In [ ]: 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.@
        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)
```

```
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
```

```python
In [ ]: 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_))
```

```
97/97 - 1s - loss: 1.7257 - accuracy: 0.5374 - 1s/epoch - 13ms/step
WARNING:tensorflow:Layer lstm_76 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.
[CV] END batch_size=32, dropout=0.1, epochs=1, learning_rate=0.1, units=196; total time=  30.4s
194/194 - 26s - loss: 1.4485 - accuracy: 0.5142 - 26s/epoch - 135ms/step
97/97 - 2s - loss: 1.3439 - accuracy: 0.5670 - 2s/epoch - 19ms/step
WARNING:tensorflow:Layer lstm_77 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.
[CV] END batch_size=32, dropout=0.1, epochs=1, learning_rate=0.1, units=196; total time=  29.4s
194/194 - 28s - loss: 0.8589 - accuracy: 0.6270 - 28s/epoch - 142ms/step
97/97 - 1s - loss: 0.7834 - accuracy: 0.6556 - 1s/epoch - 13ms/step
WARNING:tensorflow:Layer lstm_78 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=64; total time=  29.4s
194/194 - 27s - loss: 0.8535 - accuracy: 0.6320 - 27s/epoch - 141ms/step
97/97 - 1s - loss: 0.7781 - accuracy: 0.6633 - 1s/epoch - 13ms/step
WARNING:tensorflow:Layer lstm_79 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=64; total time=  29.2s
194/194 - 28s - loss: 0.8506 - accuracy: 0.6335 - 28s/epoch - 147ms/step
97/97 - 1s - loss: 0.7717 - accuracy: 0.6748 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_80 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=64; total time=  45.8s
194/194 - 27s - loss: 0.8528 - accuracy: 0.6326 - 27s/epoch - 138ms/step
97/97 - 1s - loss: 0.7893 - accuracy: 0.6511 - 1s/epoch - 13ms/step
WARNING:tensorflow:Layer lstm_81 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=128; total time=  28.2s
194/194 - 29s - loss: 0.8526 - accuracy: 0.6326 - 29s/epoch - 150ms/step
97/97 - 1s - loss: 0.7721 - accuracy: 0.6746 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_82 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=128; total time=  47.1s
194/194 - 28s - loss: 0.8501 - accuracy: 0.6264 - 28s/epoch - 143ms/step
97/97 - 1s - loss: 0.7785 - accuracy: 0.6681 - 1s/epoch - 12ms/step
WARNING:tensorflow:Layer lstm_83 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=128; total time=  29.2s
194/194 - 26s - loss: 0.8481 - accuracy: 0.6337 - 26s/epoch - 135ms/step
97/97 - 1s - loss: 0.7621 - accuracy: 0.6666 - 1s/epoch - 15ms/step
WARNING:tensorflow:Layer lstm_84 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=196; total time=  28.2s
194/194 - 27s - loss: 0.8523 - accuracy: 0.6328 - 27s/epoch - 139ms/step
97/97 - 1s - loss: 0.7815 - accuracy: 0.6740 - 1s/epoch - 13ms/step
WARNING:tensorflow:Layer lstm_85 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.
[CV] END batch_size=32, dropout=0.2, epochs=1, learning_rate=0.001, units=196; total time=  44.7s
```

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