Assignment-5

Name:Akhil Reddy Edla
ID:700744709

Programming elements:

LSTM (Long Short-Term Memory):

1. **Basics of LSTM:**

   LSTM is a type of recurrent neural network (RNN) designed to overcome the vanishing gradient problem in traditional RNNs. It introduces memory cells with gating mechanisms to selectively learn and retain information over long sequences. This allows LSTM to capture long-term dependencies in sequential data, making it ideal for tasks like natural language processing and time series analysis.

2. **Types of RNN:**

   RNN can take different forms based on the input-output relationships:

   - One-to-One: Simple RNN processing one input to one output.
   - One-to-Many: Single input generating multiple outputs.
   - Many-to-One: Sequential input producing a single output (e.g., sentiment analysis).
   - Many-to-Many: Both input and output are sequences.

3. **Use case: Sentiment Analysis on Twitter data:**

   Objective:

   Perform sentiment analysis on tweets to determine positive, negative, or neutral sentiments.

   Methodology:

   - Data Preprocessing: Clean and tokenize Twitter text data.
   - Word Embeddings: Represent words as continuous vectors using pre-trained embeddings.
   - LSTM Model: Build an LSTM-based deep learning model to capture sequential dependencies.

- Model Training: Train the LSTM model with sentiment labels as target.
- Evaluation: Evaluate model performance on a test dataset.
- Prediction: Use the model to predict sentiment for new tweets.

Benefits:

- LSTM's ability to handle sequential data makes it suitable for sentiment analysis on Twitter.
- Deep learning-based approaches, like LSTM, offer accurate sentiment predictions by learning complex patterns.
- Sentiment analysis on social media data provides valuable insights into public opinions and trends.

1. **Sentiment Analysis using LSTM:**

This code implements sentiment analysis using LSTM (Long Short-Term Memory), a type of recurrent neural network (RNN). The goal is to build a model that can classify the sentiment of **te**xt data into positive, neutral, or negative categories.

Steps:

1. **Data Preprocessing:**
    a. The code loads the dataset 'Sentiment.csv' as a Pandas DataFrame, containing 'text' and 'sentiment' columns.
    b. It converts all text to lowercase and removes special characters using regular expressions.

```
# Load the dataset as a Pandas DataFrame
dataset = pd.read_csv('Sentiment.csv')

# Select only the necessary columns 'text' and 'sentiment'
mask = dataset.columns.isin(['text', 'sentiment'])
data = dataset.loc[:, mask]

# Preprocess the text data
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', ' '))  # Remove 'rt' (Retweets)
```

2. **Tokenization:**
    a. The text data is tokenized using the Keras Tokenizer, limiting the vocabulary to a maximum of 2000 words.

3. **Label Encoding:**
    a. The sentiment labels ('positive', 'neutral', 'negative') are encoded into numerical form using LabelEncoder.

```
]: # Tokenization
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)

# Label Encoding
label_encoder = LabelEncoder()
integer_encoded = label_encoder.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)
```

4. **Model Architecture:**
    a. The LSTM model is constructed using a Sequential Neural Network.
    b. The model consists of an Embedding layer (input dimension: 2000, output dimension: 128) to convert text sequences into dense vectors.
    c. It is followed by an LSTM layer with 196 neurons, with 20% dropout and 20% recurrent dropout.
    d. The final layer is a Dense layer with 3 output neurons (for the 3 sentiment classes) and a softmax activation function for multi-class classification.
    e. The model is compiled using categorical cross-entropy loss, Adam optimizer, and accuracy metric.

```
: # LSTM Model Architecture
embed_dim = 128
lstm_out = 196

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'])

# Model Summary
print(model.summary())

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose=2)
```

5. **Model Training:**
    a. The data is split into training and testing sets (67% training, 33% testing).
    b. The model is trained on the training set using 1 epoch and a batch size of 32.

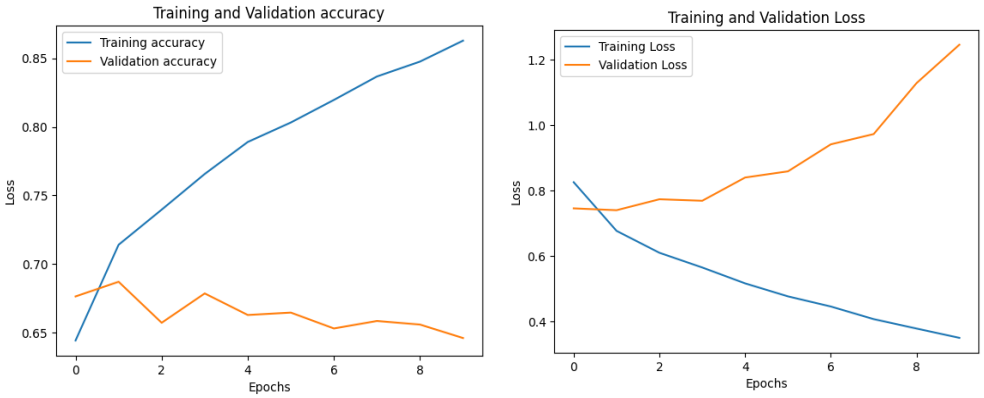
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 28, 128)           256000

 lstm (LSTM)                 (None, 196)               254800

 dense (Dense)               (None, 3)                 591

=================================================================
Total params: 511,391
Trainable params: 511,391
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
2023-07-31 13:29:56.838060: W tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency:
0 Hz
291/291 - 25s - loss: 0.8256 - accuracy: 0.6441 - val_loss: 0.7458 - val_accuracy: 0.6763 - 25s/epoch - 86ms/step
Epoch 2/10
291/291 - 24s - loss: 0.6770 - accuracy: 0.7140 - val_loss: 0.7401 - val_accuracy: 0.6870 - 24s/epoch - 83ms/step
Epoch 3/10
291/291 - 24s - loss: 0.6101 - accuracy: 0.7396 - val_loss: 0.7736 - val_accuracy: 0.6571 - 24s/epoch - 84ms/step
Epoch 4/10
291/291 - 25s - loss: 0.5651 - accuracy: 0.7656 - val_loss: 0.7690 - val_accuracy: 0.6785 - 25s/epoch - 84ms/step
Epoch 5/10
291/291 - 25s - loss: 0.5166 - accuracy: 0.7890 - val_loss: 0.8400 - val_accuracy: 0.6627 - 25s/epoch - 85ms/step
Epoch 6/10
291/291 - 24s - loss: 0.4769 - accuracy: 0.8032 - val_loss: 0.8589 - val_accuracy: 0.6645 - 24s/epoch - 84ms/step
Epoch 7/10
291/291 - 25s - loss: 0.4459 - accuracy: 0.8196 - val_loss: 0.9414 - val_accuracy: 0.6529 - 25s/epoch - 85ms/step
Epoch 8/10
291/291 - 25s - loss: 0.4074 - accuracy: 0.8368 - val_loss: 0.9728 - val_accuracy: 0.6584 - 25s/epoch - 86ms/step
Epoch 9/10
291/291 - 25s - loss: 0.3786 - accuracy: 0.8476 - val_loss: 1.1284 - val_accuracy: 0.6557 - 25s/epoch - 86ms/step
Epoch 10/10
291/291 - 25s - loss: 0.3502 - accuracy: 0.8629 - val_loss: 1.2458 - val_accuracy: 0.6459 - 25s/epoch - 87ms/step
```

6. **Model Evaluation:**
    a. The trained model is evaluated on the test set, and the loss and accuracy metrics are printed.



7. **Save the Model:**
    a. The model is saved as 'sentimentAnalysis.h5' for future use.

 # Save the trained model

model.save('sentimentAnalysis.h5')

8. **Prediction on New Data:**

a. The saved model can be loaded and used to predict the sentiment of new text data.
b. For example, the model can predict the sentiment of the provided text: "A lot of good things are happening. We are respected again throughout the world, and that's a great thing. @realDonaldTrump".

9. **Load the Saved Model**
    a. The code begins by importing the necessary packages and loading the saved sentiment analysis model using load_model from Keras.

    from keras.models import load_model

    model = load_model('sentimentAnalysis.h5')

10. **Preprocess the Text Data**
    a. Next, a new text data 'sentence' is defined for sentiment prediction. The 'sentence' is preprocessed using the same tokenization and padding steps as the training data. The tokenizer used during training is applied to the 'sentence' to convert it into a sequence of integers, and then it is padded to a fixed length.

    # Define the text data to predict sentiment

    sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing. @realDonaldTrump']

11. **Predict Sentiment**
    a. The loaded model predicts the sentiment probabilities for the input 'sentence' using the predict method. The model returns a probability distribution across the three sentiment classes (positive, neutral, and negative). The sentiment label with the highest probability is determined using np.argmax, and the corresponding sentiment ('Positive', 'Neutral', or 'Negative') is printed as the predicted sentiment.
    Note: To ensure accurate predictions, it is essential that the new text data ('sentence') undergoes the same preprocessing steps as the training

data, including tokenization and padding. By following these steps, the model can make meaningful predictions on new text data and provide valuable insights into the sentiment expressed in the text.

```
]: # Make predictions using the loaded model
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0]
                                                                text

# Convert sentiment probabilities to sentiment label
sentiment = np.argmax(sentiment_probs)

# Print the sentiment label
if sentiment == 0:
    print("Neutral")
elif sentiment < 0:
    print("Negative")
elif sentiment > 0:
    print("Positive")
else:
    print("Cannot be determined")
```

```
1/1 - 0s - 173ms/epoch - 173ms/step
Positive
```

2. **GridSearchCV on LSTM Model for Sentiment Analysis:**

GridSearchCV: An Overview

GridSearchCV is a powerful technique used for hyperparameter tuning in machine learning and deep learning models. Hyperparameters are configuration settings that are not learned during training and need to be set before training the model. The choice of hyperparameters can significantly impact the model's performance. GridSearchCV automates the process of finding the best combination of hyperparameters by exhaustively searching through a predefined grid of hyperparameter values.

In the context of deep learning models, such as LSTM for sentiment analysis, some common hyperparameters include the number of hidden units, the learning rate, batch size, and the number of epochs. GridSearchCV systematically evaluates all possible combinations of hyperparameter values specified in the grid and identifies the combination that yields the best performance according to a given evaluation metric (e.g., accuracy).

Applying GridSearchCV to the Source Code

In the provided source code, the following steps are taken to apply GridSearchCV to the LSTM sentiment analysis model:

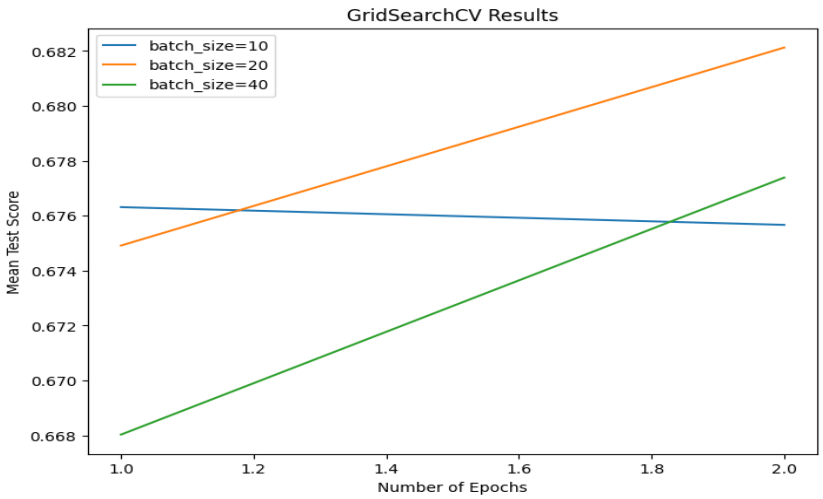
1. Define the LSTM Model Function: The function createmodel () is defined to create the LSTM model architecture. This function includes an Embedding layer, LSTM layer, and a Dense layer with softmax activation for multiclass classification.
2. Load and Preprocess the Dataset: The dataset containing text data and corresponding sentiment labels is loaded. The text data is preprocessed by converting it to lowercase and removing special characters using regular expressions. The labels are encoded categorically to represent positive, neutral, and negative sentiments.
3. Split the Dataset: The dataset is split into training and test sets using train_test_split from scikit-learn.
4. Perform GridSearchCV: The KerasClassifier is used to wrap the LSTM model function. A grid of hyperparameter values is defined,

including batch size and the number of epochs. GridSearchCV is applied to the model with the defined grid, and it evaluates the model's performance on the training data using cross-validation.

5. Retrieve the Best Model: GridSearchCV identifies the best combination of hyperparameters based on the specified evaluation metric (accuracy in this case). The best model is retrieved, which is the model with the hyperparameters that resulted in the highest accuracy on the training data.
6. Evaluate the Best Model: The best model is evaluated on the test set to obtain its accuracy on unseen data. This accuracy provides an estimate of how well the model is likely to perform on new, unseen text data.

```
Epoch 1/2
186/186 - 15s - loss: 0.8379 - accuracy: 0.6361 - 15s/epoch - 79ms/step
Epoch 2/2
186/186 - 13s - loss: 0.6870 - accuracy: 0.7084 - 13s/epoch - 70ms/step
47/47 - 1s - loss: 0.7423 - accuracy: 0.6912 - 690ms/epoch - 15ms/step
Epoch 1/2
186/186 - 14s - loss: 0.8537 - accuracy: 0.6311 - 14s/epoch - 76ms/step
Epoch 2/2
186/186 - 14s - loss: 0.6856 - accuracy: 0.7106 - 14s/epoch - 73ms/step
47/47 - 1s - loss: 0.7555 - accuracy: 0.6636 - 731ms/epoch - 16ms/step
Epoch 1/2
186/186 - 14s - loss: 0.8453 - accuracy: 0.6355 - 14s/epoch - 76ms/step
Epoch 2/2
186/186 - 13s - loss: 0.6752 - accuracy: 0.7093 - 13s/epoch - 71ms/step
47/47 - 1s - loss: 0.7921 - accuracy: 0.6755 - 716ms/epoch - 15ms/step
Epoch 1/2
465/465 - 30s - loss: 0.8117 - accuracy: 0.6530 - 30s/epoch - 64ms/step
Epoch 2/2
465/465 - 29s - loss: 0.6741 - accuracy: 0.7206 - 29s/epoch - 63ms/step
Best Score: 0.682126 using {'batch_size': 20, 'epochs': 2}
```

7. Plot GridSearchCV Results: The results of GridSearchCV, including the mean test scores for each combination of hyperparameters, are plotted using matplotlib. This visualization helps to understand how different hyperparameter combinations affect the model's performance.



By applying GridSearchCV, the code identifies the best hyperparameters for the LSTM sentiment analysis model, which can lead to improved accuracy and generalization on unseen data. The plot of GridSearchCV results provides insights into the impact of different hyperparameters on the model's performance, aiding in making informed decisions for hyperparameter tuning.

GITBUH LINK:https://github.com/edlaakhilreddy12/NNDL5

VIDEO LINK:https://drive.google.com/file/d/1mztwXGuCG3M8pnfLNMF5gNuQjFOy5ztI/view?usp=sharing