PROGRAMMING QUIZ

NAME:SAI PREETHAM NAGASWARAM

ID:700743350

Github Link: <https://github.com/Saipreethamnagaswaram/PROGRAMMING-QUIZ>

1. **Time Series Forecasting Task**:

• Load a time series dataset (e.g., stock prices, weather data).

• Build a recurrent neural network (RNN) or LSTM model using Keras.

• Train the model to forecast future values based on historical data.

• Evaluate the model's performance using appropriate metrics (e.g., MAE, RMSE).

Step 1: Import necessary libraries

Step 2: Load and preprocess the dataset

Step 3: Prepare the data for LSTM

Step 4: Build and train the LSTM model

Step 5: Evaluate the model

PROGRAM:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# Load your time series dataset, for example, let's assume you have a CSV file named 'data.csv'

data = pd.read\_csv('data.csv')

# Convert the dataframe to a numpy array

dataset = data.iloc[:, 1].values.reshape(-1, 1)

# Normalize the dataset

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(dataset)

# Define a function to create sequences of data

def create\_sequences(data, seq\_length):

X, y = [], []

for i in range(len(data) - seq\_length):

X.append(data[i:i+seq\_length])

y.append(data[i+seq\_length])

return np.array(X), np.array(y)

# Define sequence length (number of time steps to look back)

seq\_length = 10

# Create sequences

X, y = create\_sequences(scaled\_data, seq\_length)

# Split the data into training and testing sets

train\_size = int(len(X) \* 0.8)

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Build the LSTM model

model = Sequential()

model.add(LSTM(50, activation='relu', input\_shape=(seq\_length, 1)))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=32, verbose=1)

# Make predictions

train\_preds = model.predict(X\_train)

test\_preds = model.predict(X\_test)

# Inverse transform the predictions

train\_preds\_inv = scaler.inverse\_transform(train\_preds)

test\_preds\_inv = scaler.inverse\_transform(test\_preds)

y\_train\_inv = scaler.inverse\_transform(y\_train.reshape(-1, 1))

y\_test\_inv = scaler.inverse\_transform(y\_test.reshape(-1, 1))

# Calculate RMSE and MAE

train\_rmse = np.sqrt(mean\_squared\_error(y\_train\_inv, train\_preds\_inv))

test\_rmse = np.sqrt(mean\_squared\_error(y\_test\_inv, test\_preds\_inv))

train\_mae = mean\_absolute\_error(y\_train\_inv, train\_preds\_inv)

test\_mae = mean\_absolute\_error(y\_test\_inv, test\_preds\_inv)

print('Train RMSE:', train\_rmse)

print('Test RMSE:', test\_rmse)

print('Train MAE:', train\_mae)

print('Test MAE:', test\_mae)

2.**Image Classification Task**:

• Load the MNIST dataset.

• Build a simple convolutional neural network (CNN) using Keras Sequential model.

• Train the CNN model on the MNIST dataset.

• Evaluate the model's performance on a test set and report accuracy.

• Use grid search to optimize hyperparameters such as learning rate, batch size, and

optimizer choice.

• Use Callback functions to automate training process like “ReduceLROnPlateau” and keep

check on validation loss. Also use history object for result visualization.

Step 1: Import necessary libraries

Step 2: Load and preprocess the MNIST dataset

Step 3: Build a CNN model

Step 4: Define callback functions

Step 5: Train the model

Step 6: Evaluate the model

Step 7: Hyperparameter optimization using GridSearchCV

Step 8: Visualize training history

PROGRAM:

import numpy as np

import matplotlib.pyplot as plt

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from keras.optimizers import Adam

from keras.callbacks import ReduceLROnPlateau, History

from sklearn.model\_selection import GridSearchCV

# Load MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# Reshape and normalize the dataset

X\_train = X\_train.reshape(X\_train.shape[0], 28, 28, 1).astype('float32') / 255

X\_test = X\_test.reshape(X\_test.shape[0], 28, 28, 1).astype('float32') / 255

# Convert class vectors to binary class matrices

num\_classes = 10

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

def create\_model(optimizer='adam'):

model = Sequential([

Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(num\_classes, activation='softmax')

])

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

return model

# Instantiate the model

model = create\_model()

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=3, min\_lr=0.0001)

history = History()

callbacks = [reduce\_lr, history]

history = model.fit(X\_train, y\_train, batch\_size=128, epochs=10, verbose=1, validation\_split=0.2, callbacks=callbacks)

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print('Test Loss:', test\_loss)

print('Test Accuracy:', test\_accuracy)

param\_grid = {'optimizer': ['adam', 'rmsprop', 'sgd'],

'batch\_size': [32, 64, 128]}

grid = GridSearchCV(estimator=create\_model(), param\_grid=param\_grid, cv=3)

grid\_result = grid.fit(X\_train, y\_train)

print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()