Neural network and deep learning programming quiz

Pushkara Naga Sai Sri Vyshnavi Chakka

700752861

Github link: <https://github.com/PushkaraChakka/programming-quiz-submission>

Step 1: Load the dataset You can start by loading your time series dataset. For example, if you're working with stock prices, you can use libraries like Pandas to read CSV files containing historical stock price data.

Program:

import pandas as pd

# Load the dataset

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

# Optionally, you may need to preprocess the data (e.g., handle missing values, scale the data).

Step 2: Preprocess the data Preprocessing may involve steps like handling missing values, scaling the data, and splitting it into training and testing sets.

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

# Preprocess the data

scaler = MinMaxScaler()

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

# Split the data into training and testing sets

train\_data, test\_data = train\_test\_split(scaled\_data, test\_size=0.2, shuffle=False)

Step 3: Prepare the data for LSTM LSTMs require input data in a specific format, typically in the form of sequences. You'll need to create sequences of input-output pairs from the time series 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)

seq\_length = 10 # Define the sequence length

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

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

Step 4: Build the LSTM model Now, you can build the LSTM model using Keras.

from keras.models import Sequential

from keras.layers import LSTM, Dense

model = Sequential()

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

model.add(Dense(units=1)) # Output layer with 1 neuron for single-step forecasting

model.compile(optimizer='adam', loss='mean\_squared\_error')

Step 5: Train the model Train the LSTM model using the training data.

model.fit(X\_train, y\_train, epochs=50, batch\_size=32)

Step 6: Evaluate the model Evaluate the performance of the model using appropriate metrics.

from sklearn.metrics import mean\_squared\_error

# Predictions

predictions = model.predict(X\_test)

# Inverse scaling

predictions = scaler.inverse\_transform(predictions)

y\_test = scaler.inverse\_transform(y\_test)

# Evaluate

mse = mean\_squared\_error(y\_test, predictions)

rmse = np.sqrt(mse)

print("Root Mean Squared Error:", rmse)

**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: Load the MNIST dataset

from keras.datasets import mnist

# Load the MNIST dataset

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

Step 2: Preprocess the data Preprocess the data by normalizing pixel values and reshaping the input images.

import numpy as np

# Normalize pixel values to be between 0 and 1

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

# Reshape input images to 3D tensors (height, width, channels)

X\_train = X\_train.reshape(-1, 28, 28, 1)

X\_test = X\_test.reshape(-1, 28, 28, 1)

Step 3: Build the CNN model

from keras.models import Sequential

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

model = Sequential([

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

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(64, activation='relu'),

Dense(10, activation='softmax')

])

Step 4: Compile the model Compile the model specifying the optimizer, loss function, and metrics.

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

Step 5: Train the model Train the CNN model on the MNIST training dataset.

history = model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_data=(X\_test, y\_test))

Step 6: Evaluate the model Evaluate the performance of the model on the test set and report accuracy.

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

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

Step 7: Use Callback functions Implement callback functions such as ReduceLROnPlateau to adjust learning rate and ModelCheckpoint to save the best model during training.

from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint

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

checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_accuracy', save\_best\_only=True, mode='max')

callbacks = [reduce\_lr, checkpoint]

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=64, validation\_data=(X\_test, y\_test), callbacks=callbacks)

from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint

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

checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_accuracy', save\_best\_only=True, mode='max')

callbacks = [reduce\_lr, checkpoint]

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=64, validation\_data=(X\_test, y\_test), callbacks=callbacks)

Step 8: Visualize training history Visualize the training and validation accuracy and loss over epochs.

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()