1. Imdb

from sklearn.preprocessing import LabelEncoder

from keras.preprocessing.text import Tokenizer

from keras import layers

from keras import models

from keras import optimizers

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load the dataset

train\_df = pd.read\_csv("imdbdataset.csv")

# Tokenize the text data

tokenizer = Tokenizer(num\_words=1000)

tokenizer.fit\_on\_texts(train\_df["review"])

x\_train = tokenizer.texts\_to\_matrix(train\_df["review"], mode="binary")

# Convert the sentiment labels to binary format

label\_encoder = LabelEncoder()

y\_train = label\_encoder.fit\_transform(train\_df["sentiment"]).astype("float32")

# Define the model

model = models.Sequential()

model.add(layers.Dense(16, activation="relu", input\_shape=(1000,)))

model.add(layers.Dense(1, activation="sigmoid"))

model.compile(optimizer=optimizers.RMSprop(lr=0.001),

loss="binary\_crossentropy",

metrics=["accuracy"])

history = model.fit(x\_train, y\_train,

epochs=10,

batch\_size=128,

validation\_split=0.2)

# Plot the training and validation accuracy

acc = history.history["accuracy"]

val\_acc = history.history["val\_accuracy"]

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, "bo", label="Training accuracy")

plt.plot(epochs, val\_acc, "b", label="Validation accuracy")

plt.title("Training and validation accuracy")

plt.xlabel("Epochs")

plt.ylabel("Accuracy")

plt.legend()

plt.show()

# Plot the training and validation loss

loss = history.history["loss"]

val\_loss = history.history["val\_loss"]

plt.plot(epochs, loss, "bo", label="Training loss")

plt.plot(epochs, val\_loss, "b", label="Validation loss")

plt.title("Training and validation loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

from tensorflow.keras.utils import plot\_model

plot\_model(model, show\_shapes=True)

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. Boston housing

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Load the data from the CSV file

housing\_df = pd.read\_csv('housing.csv', header=None, delimiter='\s+')

# Split the data into features (X) and target (y)

X = housing\_df.iloc[:, :-1].values

y = housing\_df.iloc[:, -1].values.reshape(-1, 1)

# Scale the data using StandardScaler (scale feature and target variable)

scaler = StandardScaler()

scaled\_X = scaler.fit\_transform(X)

scaled\_y = scaler.fit\_transform(y)

# Define the model

model = tf.keras.models.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=[13]),

tf.keras.layers.Dense(1)

])

# Compile the model

model.compile(loss='mse', optimizer=tf.keras.optimizers.RMSprop(0.001), metrics=['mae'])

# Train the model

history = model.fit(scaled\_X, scaled\_y, epochs=500, validation\_split=0.2, verbose=0)

# Plot the training and validation loss over epochs

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Evaluate the model

loss, mae = model.evaluate(scaled\_X, scaled\_y, verbose=0)

print("Mean absolute error:", mae)

# Predict housing prices using the model

predictions = model.predict(scaled\_X)

predicted\_prices = scaler.inverse\_transform(predictions)

# Print predicted prices and actual prices

print("Predicted prices:", predicted\_prices.flatten())

print("Actual prices:", y.flatten())

# Plot predicted prices and actual prices

plt.scatter(range(len(predicted\_prices)), predicted\_prices.flatten(), label='Predicted prices')

plt.scatter(range(len(y)), y.flatten(), label='Actual prices')

# Set plot title and labels

plt.title('Predicted vs Actual Prices')

plt.xlabel('Data point')

plt.ylabel('Price')

# Add legend

plt.legend()

# Show the plot

plt.show()

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. Fashion-mnist

import tensorflow as tf

from tensorflow import keras

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from tensorflow.keras.utils import plot\_model

# Load the data from CSV files

train\_df = pd.read\_csv("fashion-mnist\_train.csv")

test\_df = pd.read\_csv("fashion-mnist\_test.csv")

# Split the data into features and labels

x\_train = train\_df.drop("label", axis=1).values.reshape(-1, 28, 28, 1) / 255.0

x\_test = test\_df.drop("label", axis=1).values.reshape(-1, 28, 28, 1) / 255.0

y\_train = train\_df["label"].values

y\_test = test\_df["label"].values

# Define the class names

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',

'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Define and train a deep learning model

model = keras.Sequential([

keras.layers.Conv2D(32, (3,3), padding='same', activation='relu', input\_shape=(28,28,1)),

keras.layers.MaxPooling2D((2,2)),

keras.layers.Conv2D(64, (3,3), padding='same', activation='relu'),

keras.layers.MaxPooling2D((2,2)),

keras.layers.Flatten(),

keras.layers.Dense(128, activation='relu'),

keras.layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=10)

# Evaluate the model on the testing data

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print('\nTest accuracy:', test\_acc)

# Make predictions on new data

predictions = model.predict(x\_test)

# Plot some images with their predictions

plt.figure(figsize=(10, 10))

for i in range(25):

plt.subplot(5, 5, i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(x\_test[i].reshape(28, 28), cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions[i])

true\_label = y\_test[i]

if predicted\_label == true\_label:

color = 'green'

else:

color = 'red'

plt.xlabel("{} ({})".format(class\_names[predicted\_label],

class\_names[true\_label]),

color=color)

plt.show()

plot\_model(model, show\_shapes=True)

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Google stocks price train

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

dataset\_train = pd.read\_csv('Google\_Stock\_Price\_Train.csv')

training\_set = dataset\_train.iloc[:, 1:2].values

from sklearn.preprocessing import MinMaxScaler

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

training\_set\_scaled = sc.fit\_transform(training\_set)

X\_train = []

y\_train = []

for i in range(60, 1258):

X\_train.append(training\_set\_scaled[i-60:i, 0])

y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

# LSTML requires data in 3d tensonr with shape (batch\_size, timesteps, input\_dim)

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

regressor = Sequential()

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))

regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

# configure learning process

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

regressor.fit(X\_train, y\_train, epochs = 10, batch\_size = 32)

dataset\_test = pd.read\_csv('Google\_Stock\_Price\_Test.csv')

real\_stock\_price = dataset\_test.iloc[:, 1:2].values

dataset\_total = pd.concat((dataset\_train['Open'], dataset\_test['Open']), axis = 0)

inputs = dataset\_total[len(dataset\_total) - len(dataset\_test) - 60:].values

inputs = inputs.reshape(-1,1)

inputs = sc.transform(inputs)

X\_test = []

for i in range(60, 80):

X\_test.append(inputs[i-60:i, 0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

predicted\_stock\_price = regressor.predict(X\_test)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

plt.plot(real\_stock\_price, color = 'red', label = 'Real Google Stock Price')

plt.plot(predicted\_stock\_price, color = 'blue', label = 'Predicted Google Stock Price')

plt.title('Google Stock Price Prediction')

plt.xlabel('Time')

plt.ylabel('Google Stock Price')

plt.legend()

plt.show()

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH

1. BFS/ DFS

#include <iostream>

#include <vector>

#include <queue>

#include <omp.h>

#include <stack>

using namespace std;

void bfs(vector<vector<int>>& graph, int start, vector<bool>& visited) {

queue<int> q;

q.push(start);

visited[start] = true;

#pragma omp parallel

{

#pragma omp single

{

while (!q.empty()) {

int vertex = q.front();

cout << vertex << " "<<endl;

q.pop();

#pragma omp task firstprivate(vertex)

{

for (int neighbor : graph[vertex]) {

if (!visited[neighbor]) {

q.push(neighbor);

visited[neighbor] = true;

#pragma omp task

bfs(graph, neighbor, visited);

}

}

}

}

}

}

}

void parallel\_bfs(vector<vector<int>>& graph, int start) {

vector<bool> visited(graph.size(), false);

bfs(graph, start, visited);

}

void dfs(vector<vector<int>>& graph, int start, vector<bool>& visited) {

stack<int> s;

s.push(start);

visited[start] = true;

#pragma omp parallel

{

#pragma omp single

{

while (!s.empty()) {

int vertex = s.top();

cout << vertex << " "<<endl;

s.pop();

#pragma omp task firstprivate(vertex)

{

for (int neighbor : graph[vertex]) {

if (!visited[neighbor]) {

s.push(neighbor);

visited[neighbor] = true;

#pragma omp task

dfs(graph, neighbor, visited);

}

}

}

}

}

}

}

void parallel\_dfs(vector<vector<int>>& graph, int start) {

vector<bool> visited(graph.size(), false);

dfs(graph, start, visited);

}

int main() {

vector<vector<int>> graph(7);

graph[0] = {1, 2};

graph[1] = {0, 2, 3, 4};

graph[2] = {0, 1, 5, 6};

graph[3] = {1, 4};

graph[4] = {1, 3};

graph[5] = {2};

graph[6] = {2};

cout << "BFS"<<endl;

parallel\_bfs(graph, 0);

cout << "DFS"<<endl;

parallel\_dfs(graph, 0);

return 0;

}

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#include <iostream>

#include <cuda\_runtime.h>

\_\_global\_\_ void addVectors(int\* A, int\* B, int\* C, int n) {

int i = blockIdx.x \* blockDim.x + threadIdx.x;

if (i < n) {

C[i] = A[i] + B[i];

}

}

int main() {

int n;

cout << "Enter number of elements "<<endl;

cin>>n;

int\* A, \* B, \* C;

int size = n \* sizeof(int);

cudaMallocHost(&A, size);

cudaMallocHost(&B, size);

cudaMallocHost(&C, size);

for (int i = 0; i < n; i++) {

cout << "Enter value in A" << endl;

cin >> A[i];

cout << "Enter value in B "<< endl;

cin >> B[i];

}

int\* dev\_A, \* dev\_B, \* dev\_C;

cudaMalloc(&dev\_A, size);

cudaMalloc(&dev\_B, size);

cudaMalloc(&dev\_C, size);

cudaMemcpy(dev\_A, A, size, cudaMemcpyHostToDevice);

cudaMemcpy(dev\_B, B, size, cudaMemcpyHostToDevice);

int blockSize = 256;

int numBlocks = (n + blockSize - 1) / blockSize;

addVectors<<<numBlocks, blockSize>>>(dev\_A, dev\_B, dev\_C, n);

cudaMemcpy(C, dev\_C, size, cudaMemcpyDeviceToHost);

for (int i = 0; i < 100; i++) {

std::cout << C[i] << " - ";

}

std::cout << std::endl;

cudaFree(dev\_A);

cudaFree(dev\_B);

cudaFree(dev\_C);

cudaFreeHost(A);

cudaFreeHost(B);

cudaFreeHost(C);

return 0;

}

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#include <iostream>

#include <vector>

#include <omp.h>

using namespace std;

void min\_reduction(vector<int>& arr) {

int min\_value = 10000;

#pragma omp parallel for reduction(min: min\_value)

for (int i = 0; i < arr.size(); i++) {

if (arr[i] < min\_value) {

min\_value = arr[i];

}

}

cout << "Minimum value: " << min\_value << endl;

}

void max\_reduction(vector<int>& arr) {

int max\_value = -1;

#pragma omp parallel for reduction(max: max\_value)

for (int i = 0; i < arr.size(); i++) {

if (arr[i] > max\_value) {

max\_value = arr[i];

}

}

cout << "Maximum value: " << max\_value << endl;

}

void sum\_reduction(vector<int>& arr) {

int sum = 0;

#pragma omp parallel for reduction(+: sum)

for (int i = 0; i < arr.size(); i++) {

sum += arr[i];

}

cout << "Sum: " << sum << endl;

}

void average\_reduction(vector<int>& arr) {

int sum = 0;

#pragma omp parallel for reduction(+: sum)

for (int i = 0; i < arr.size(); i++) {

sum += arr[i];

}

cout << "Average: " << (double)sum / arr.size() << endl;

}

int main() {

vector<int> arr = {5, 2, 9, 1, 7, 6, 8, 3, 4};

min\_reduction(arr);

max\_reduction(arr);

sum\_reduction(arr);

average\_reduction(arr);

}

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#include <iostream>

#include <vector>

#include <omp.h>

using namespace std;

void bubble\_sort\_odd\_even(vector<int>& arr) {

bool isSorted = false;

while (!isSorted) {

isSorted = true;

#pragma omp parallel for

for (int i = 0; i < arr.size() - 1; i += 2) {

if (arr[i] > arr[i + 1]) {

swap(arr[i], arr[i + 1]);

isSorted = false;

}

}

#pragma omp parallel for

for (int i = 1; i < arr.size() - 1; i += 2) {

if (arr[i] > arr[i + 1]) {

swap(arr[i], arr[i + 1]);

isSorted = false;

}

}

}

}

void merge(vector<int>& arr, int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

vector<int> L(n1), R(n2);

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j])

arr[k++] = L[i++];

else

arr[k++] = R[j++];

}

while (i < n1)

arr[k++] = L[i++];

while (j < n2)

arr[k++] = R[j++];

}

void merge\_sort(vector<int>& arr, int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

merge\_sort(arr, l, m);

merge\_sort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void parallel\_merge\_sort(vector<int>& arr) {

#pragma omp parallel

{

#pragma omp single

merge\_sort(arr, 0, arr.size() - 1);

}

}

int main() {

vector<int> arr1 = {5, 2, 9, 1, 7, 6, 8, 3, 4};

vector<int> arr2 = {6, 1, 19, 11, 17, 61, 18, 13, 41};

double bstart, bend, mstart, mend;

bstart = omp\_get\_wtime();

bubble\_sort\_odd\_even(arr1);

bend = omp\_get\_wtime();

cout << "Parallel bubble sort using odd-even transposition time: " << bend - bstart << endl;

mstart = omp\_get\_wtime();

parallel\_merge\_sort(arr2);

mend = omp\_get\_wtime();

cout << "Parallel merge sort time: " << mend - mstart << endl;

return 0;

}