m7vxdqs4n

April 18, 2025

0.1 Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural Network. Use Boston House Price prediction Dataset.

[1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam
```

[2]:
```
df = pd.read_csv("HousingData.csv")
df.head()
```

[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
```

```
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

[4]:
```
df.fillna(df.mean(), inplace=True)
```

[5]:
```
X = df.drop(columns=['MEDV'])
y = df['MEDV']
```

[6]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

[7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[8]:
```
model = keras.Sequential([
    keras.Input(shape=(X.shape[1],)),  # Explicit Input Layer
    layers.Dense(64, activation='relu'),  # Hidden Layer 1
    layers.Dense(32, activation='relu'),  # Hidden Layer 2
    layers.Dense(1, activation='linear')  # Output layer (Regression)
])
```

[9]:
```
model.compile(optimizer=Adam(learning_rate=0.01), loss='mse', metrics=['mae'])
```

[10]:
```
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_test,␣
 ↪y_test), batch_size=16, verbose=1)
```

```
Epoch 1/100
26/26             1s 8ms/step - loss:
440.8634 - mae: 18.5338 - val_loss: 57.2412 - val_mae: 5.5565
Epoch 2/100
26/26             0s 2ms/step - loss:
45.3372 - mae: 5.1544 - val_loss: 21.6366 - val_mae: 3.2939
Epoch 3/100
26/26             0s 3ms/step - loss:
```

```
5.3956 - mae: 1.7245 - val_loss: 11.9814 - val_mae: 2.3072
Epoch 100/100
26/26             0s 2ms/step - loss:
3.7539 - mae: 1.5173 - val_loss: 11.3920 - val_mae: 2.2617
```

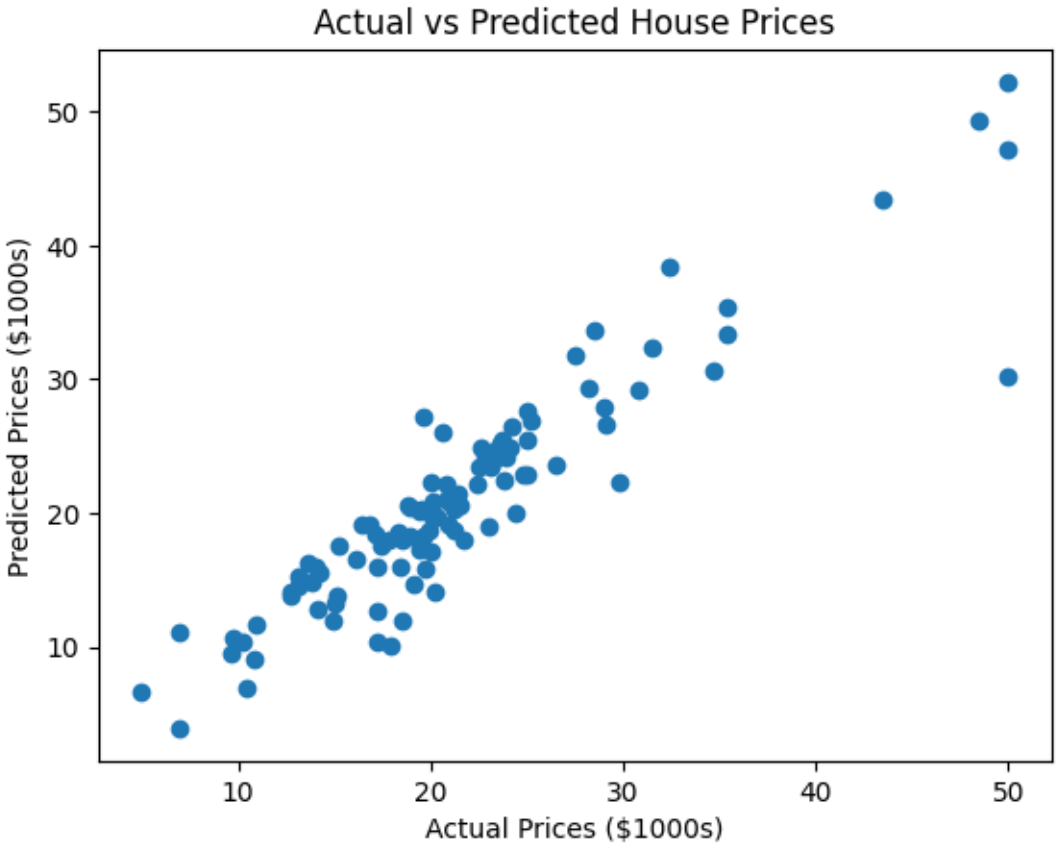
[11]:
```
loss, mae = model.evaluate(X_test, y_test)
print(f"Test Loss (MSE): {loss}")
print(f"Test Mean Absolute Error (MAE): {mae}")
```

```
4/4             0s 0s/step - loss:
8.9360 - mae: 2.1187
Test Loss (MSE): 11.391955375671387
Test Mean Absolute Error (MAE): 2.261707305908203
```

[12]:
```
predictions = model.predict(X_test)
```

```
4/4             0s 6ms/step
```

[13]:
```
plt.scatter(y_test, predictions)
plt.xlabel("Actual Prices ($1000s)")
plt.ylabel("Predicted Prices ($1000s)")
plt.title("Actual vs Predicted House Prices")
plt.show()
```



wxs98airg

April 18, 2025

0.1 Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset.

[1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
```

[2]:
```
vocab_size = 10000
max_length = 200
(x_train, y_train), (x_test, y_test) = keras.datasets.imdb.
 ↪load_data(num_words=vocab_size)
```

[3]:
```
x_train = keras.preprocessing.sequence.pad_sequences(x_train,␣
 ↪maxlen=max_length, padding='post')
x_test = keras.preprocessing.sequence.pad_sequences(x_test, maxlen=max_length,␣
 ↪padding='post')
```

[4]:
```
model = keras.Sequential([
    layers.Embedding(input_dim=vocab_size, output_dim=64),  # Removed␣
 ↪input_length
    layers.Conv1D(32, 5, activation='relu'),  # 1D Convolution for feature␣
 ↪extraction
    layers.GlobalMaxPooling1D(),  # Reduce dimensions
    layers.Dense(64, activation='relu'),  # Fully connected layer
    layers.Dense(1, activation='sigmoid')  # Output layer (Binary␣
 ↪classification)
])
```

[5]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[6]:
```
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test,␣
 ↪y_test), batch_size=64, verbose=1)
```

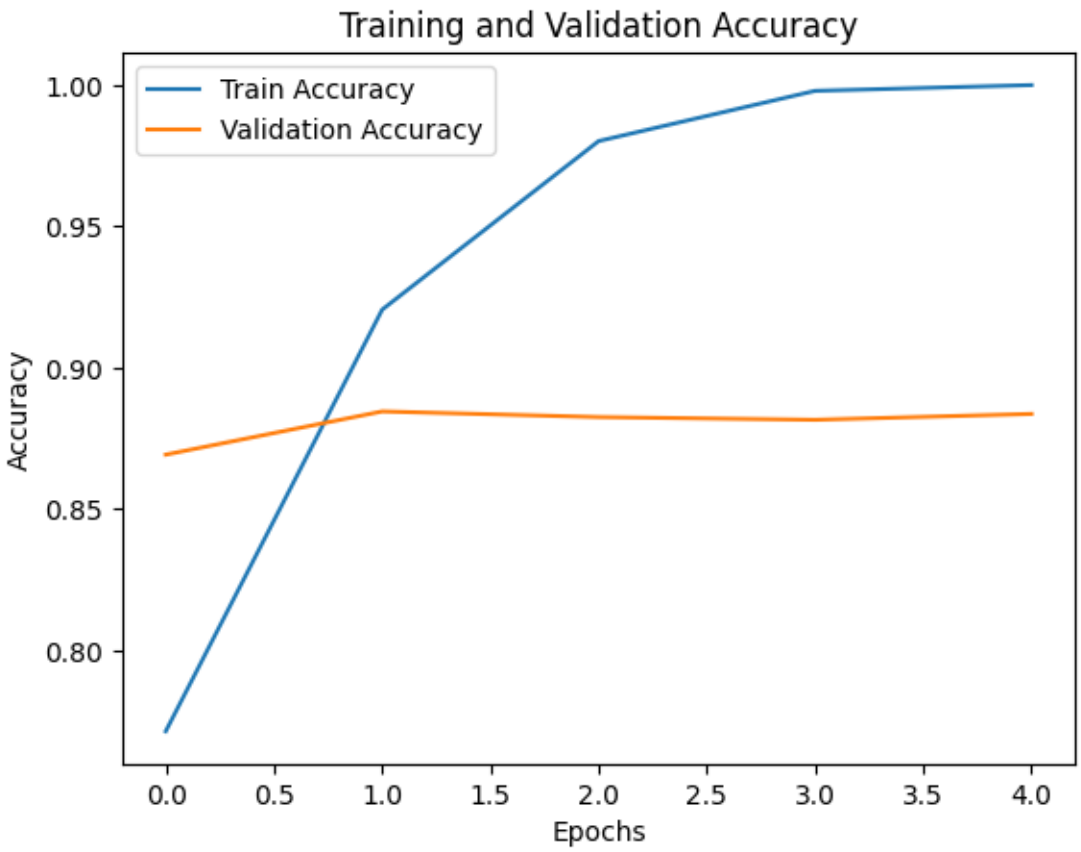
```
Epoch 1/5
391/391            8s 18ms/step -
```

```
accuracy: 0.6687 - loss: 0.5754 - val_accuracy: 0.8692 - val_loss: 0.3026
Epoch 2/5
391/391             7s 18ms/step -
accuracy: 0.9185 - loss: 0.2122 - val_accuracy: 0.8845 - val_loss: 0.2767
Epoch 3/5
391/391             7s 18ms/step -
accuracy: 0.9827 - loss: 0.0740 - val_accuracy: 0.8825 - val_loss: 0.3220
Epoch 4/5
391/391             7s 18ms/step -
accuracy: 0.9979 - loss: 0.0180 - val_accuracy: 0.8816 - val_loss: 0.3672
Epoch 5/5
391/391             7s 17ms/step -
accuracy: 1.0000 - loss: 0.0031 - val_accuracy: 0.8836 - val_loss: 0.4037
```

[7]:
```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Loss: {loss:.4f}")
```

```
782/782             3s 4ms/step -
accuracy: 0.8828 - loss: 0.4049
Test Accuracy: 0.8836
Test Loss: 0.4037
```

[8]:
```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()
```


Training and Validation Accuracy
Train Accuracy
Validation Accuracy
Accuracy
1.00
0.95
0.90
0.85
0.80
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epochs

rbezg8v4z

April 18, 2025

0.1 Convolutional neural network (CNN) (Any One from the following)

Use any dataset of plant disease and design a plant disease detection system using CNN.
Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

[1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
```

[2]:
```
train_df = pd.read_csv('fashion-mnist_train.csv')
test_df = pd.read_csv('fashion-mnist_test.csv')
```

[3]:
```
train_df.head(2)
```

[3]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      2       0       0       0       0       0       0       0       0
1      9       0       0       0       0       0       0       0       0

   pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0

   pixel781  pixel782  pixel783  pixel784
0         0         0         0         0
1         0         0         0         0

[2 rows x 785 columns]
```

[4]:
```
test_df.head(2)
```

[4]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      0       0       0       0       0       0       0       0       9
1      1       0       0       0       0       0       0       0       0

   pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0       8  …       103        87        56         0         0         0
```

```
1       0  …        34         0         0         0         0         0

   pixel781  pixel782  pixel783  pixel784
0         0         0         0         0
1         0         0         0         0

[2 rows x 785 columns]
```

[5]:
```
# Split features and labels
x_train = train_df.iloc[:, 1:].values
y_train = train_df.iloc[:, 0].values
```

[6]:
```
x_test = test_df.iloc[:, 1:].values
y_test = test_df.iloc[:, 0].values
```

[7]:
```
# Normalize pixel values
x_train = x_train / 255.0
x_test = x_test / 255.0
```

[8]:
```
# Reshape for CNN: (samples, height, width, channels)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

[9]:
```
# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
C:\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[10]:
```
# Compile model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[11]:
```
# Train
model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

```
Epoch 1/5
1688/1688             13s 7ms/step -
accuracy: 0.7610 - loss: 0.6718 - val_accuracy: 0.8638 - val_loss: 0.3871
Epoch 2/5
1688/1688             12s 7ms/step -
accuracy: 0.8786 - loss: 0.3398 - val_accuracy: 0.8882 - val_loss: 0.3101
Epoch 3/5
1688/1688             13s 7ms/step -
accuracy: 0.8976 - loss: 0.2819 - val_accuracy: 0.8890 - val_loss: 0.3041
Epoch 4/5
1688/1688             13s 8ms/step -
accuracy: 0.9079 - loss: 0.2526 - val_accuracy: 0.8978 - val_loss: 0.2800
Epoch 5/5
1688/1688             12s 7ms/step -
accuracy: 0.9167 - loss: 0.2238 - val_accuracy: 0.8975 - val_loss: 0.2802
```

[11]:
```
<keras.src.callbacks.history.History at 0x25f324b7050>
```

[12]:
```
# Evaluate
loss, acc = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {acc}")
```

```
313/313             1s 4ms/step -
accuracy: 0.9019 - loss: 0.2680

Test Accuracy: 0.9064000248908997
```

[13]:
```
import matplotlib.pyplot as plt
class_names=['T-shirt/
 ↪top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle␣
 ↪boot']
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_train[i],cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()
```


Pullover
Ankle boot
Shirt
T-shirt/top
Dress
Coat
Coat
Sandal
Coat
Bag
T-shirt/top
Bag
Ankle boot
T-shirt/top
Pullover
Pullover
Ankle boot
Dress
Dress
Dress
Bag
Sneaker
Coat
Coat
T-shirt/top

[ ]:

```
//Design and implement Parallel Breadth First Search and Depth First Search
based on existing algorithms using OpenMP. Use a Tree or an undirected graph for
BFS and DFS .

#include <iostream>
#include <vector>
#include <queue>
#include <stack>
#include <omp.h>
using namespace std;

const int N = 6;  // Number of nodes
vector<int> graph[N];
bool visited_bfs[N], visited_dfs[N];

// Add edge to undirected graph
void addEdge(int u, int v) {
    graph[u].push_back(v);
    graph[v].push_back(u);
}

// Parallel BFS using OpenMP
void parallelBFS(int start) {
    queue<int> q;
    q.push(start);
    visited_bfs[start] = true;

    while (!q.empty()) {
        int size = q.size();

        #pragma omp parallel for
        for (int i = 0; i < size; i++) {
            int node;
            #pragma omp critical
            {
                node = q.front(); q.pop();
                cout << "BFS visited: " << node << endl;
            }

            for (int neighbor : graph[node]) {
                #pragma omp critical
                {
                    if (!visited_bfs[neighbor]) {
                        visited_bfs[neighbor] = true;
                        q.push(neighbor);
                    }
                }
            }
        }
    }
}

// Parallel DFS using OpenMP
void parallelDFS(int start) {
```

```
    stack<int> s;
    s.push(start);
    visited_dfs[start] = true;

    while (!s.empty()) {
        int node;
        #pragma omp critical
        {
            node = s.top(); s.pop();
            cout << "DFS visited: " << node << endl;
        }

        #pragma omp parallel for
        for (int i = 0; i < graph[node].size(); i++) {
            int neighbor = graph[node][i];
            #pragma omp critical
            {
                if (!visited_dfs[neighbor]) {
                    visited_dfs[neighbor] = true;
                    s.push(neighbor);
                }
            }
        }
    }
}

int main() {
    addEdge(0, 1);
    addEdge(0, 2);
    addEdge(1, 3);
    addEdge(1, 4);
    addEdge(2, 5);

    cout << "Parallel BFS:\n";
    parallelBFS(0);

    cout << "\nParallel DFS:\n";
    parallelDFS(0);

    return 0;
}

//run= g++ -fopenmp HPC_Practical_1.cpp -o HPC_Practical_1
//Windows:- HPC_Practical_1.exe
//Linux:- ./HPC_Practical_1
```

```
//Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP.
Use existing algorithms and measure the performance of sequential and parallel
algorithms.

#include <iostream>
#include <vector>
#include <omp.h>
using namespace std;

// Sequential Bubble Sort
void bubbleSortSeq(vector<int>& arr) {
    int n = arr.size();
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}

// Parallel Bubble Sort
void bubbleSortPar(vector<int>& arr) {
    int n = arr.size();
    for (int i = 0; i < n; i++) {
        #pragma omp parallel for
        for (int j = i % 2; j < n - 1; j += 2)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
    }
}

// Merge function
void merge(vector<int>& arr, int l, int m, int r) {
    vector<int> left(arr.begin() + l, arr.begin() + m + 1);
    vector<int> right(arr.begin() + m + 1, arr.begin() + r + 1);
    int i = 0, j = 0, k = l;
    while (i < left.size() && j < right.size())
        arr[k++] = (left[i] < right[j]) ? left[i++] : right[j++];
    while (i < left.size()) arr[k++] = left[i++];
    while (j < right.size()) arr[k++] = right[j++];
}

// Sequential Merge Sort
void mergeSortSeq(vector<int>& arr, int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSortSeq(arr, l, m);
        mergeSortSeq(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

// Parallel Merge Sort
void mergeSortPar(vector<int>& arr, int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
```

```
        #pragma omp parallel sections
        {
            #pragma omp section
            mergeSortPar(arr, l, m);
            #pragma omp section
            mergeSortPar(arr, m + 1, r);
        }
        merge(arr, l, m, r);
    }
}

int main() {
    vector<int> data = {8, 5, 2, 9, 1, 4};
    vector<int> arr1 = data, arr2 = data;
    vector<int> arr3 = data, arr4 = data;

    bubbleSortSeq(arr1);
    bubbleSortPar(arr2);
    mergeSortSeq(arr3, 0, arr3.size() - 1);
    mergeSortPar(arr4, 0, arr4.size() - 1);

    cout << "Sorted (Seq Bubble): ";
    for (int x : arr1) cout << x << " ";
    cout << "\nSorted (Par Bubble): ";
    for (int x : arr2) cout << x << " ";
    cout << "\nSorted (Seq Merge):  ";
    for (int x : arr3) cout << x << " ";
    cout << "\nSorted (Par Merge):  ";
    for (int x : arr4) cout << x << " ";
}


//Complie Windows:- g++ -fopenmp HPC_Practical_2.cpp -o HPC_Practical_2
//Run:- HPC_Practical_2.exe

//Complie linux:- sudo apt update
//sudo apt install g++ libomp-dev
//g++ -fopenmp HPC_Practical_2.cpp -o HPC_Practical_2
//./HPC_Practical_2
```

bqcofxpye

April 18, 2025

0.1 Implement Min, Max, Sum and Average oprations using Parallel Reduction.

[1]:
```
import multiprocessing
import random

def parallel_reduction(operation, arr):
    with multiprocessing.Pool() as pool:
        if operation == "min":
            return min(pool.map(min, arr))
        elif operation == "max":
            return max(pool.map(max, arr))
        elif operation == "sum":
            return sum(pool.map(sum, arr))
        elif operation == "avg":
            return sum(pool.map(sum, arr)) / len(arr)

if __name__ == "__main__":
    arr = [random.randint(0, 10000) for _ in range(10000)]
    chunked_arr = [arr[i::multiprocessing.cpu_count()] for i in␣
 ↪range(multiprocessing.cpu_count())]

    print(f"Min: {parallel_reduction('min', chunked_arr)}")
    print(f"Max: {parallel_reduction('max', chunked_arr)}")
    print(f"Sum: {parallel_reduction('sum', chunked_arr)}")
    print(f"Average: {parallel_reduction('avg', chunked_arr)}")
```

```
Min: 0
Max: 10000
Sum: 49891056
Average: 6236382.0
```

[ ]:
```

```

Write a cuda program for
1. Addition of two large vectors
2. Matrics Multiplication using CUDA C

```
from numba import cuda
import numpy as np

@cuda.jit
def add_vectors(a, b, c):
    i = cuda.grid(1)
    if i < a.size:
        c[i] = a[i] + b[i]

# Host code
N = 1000000
a = np.arange(N, dtype=np.float32)
b = np.arange(N, dtype=np.float32)
c = np.zeros(N, dtype=np.float32)

# Copy to device
d_a = cuda.to_device(a)
d_b = cuda.to_device(b)
d_c = cuda.device_array_like(c)

# Launch kernel
threadsperblock = 256
blockspergrid = (N + (threadsperblock - 1)) // threadsperblock
add_vectors[blockspergrid, threadsperblock] (d_a, d_b, d_c)

# Copy back
c = d_c.copy_to_host()

print(c[:5])
```

```
from numba import cuda
import numpy as np

@cuda.jit
def matmul(A, B, C):
   row, col = cuda.grid(2)
   if row < C.shape[0] and col < C.shape [1]:
      tmp = 0
      for k in range(A.shape[1]):
         tmp += A[row, k] * B[k, col]
      C[row, col] = tmp


N = 512
A = np.ones((N, N), dtype=np.float32)
B = np.ones((N, N), dtype=np.float32)
C = np.zeros((N, N), dtype=np.float32)

d_A = cuda.to_device(A)
d_B = cuda.to_device(B)
d_C = cuda.device_array_like(C)

threadsperblock = (16, 16)
blockspergrid_x = (A.shape[0] + threadsperblock[0] - 1) // threadsperblock[0]
blockspergrid_y = (B.shape[1] + threadsperblock[1] - 1) // threadsperblock[1]
matmul[(blockspergrid_x, blockspergrid_y), threadsperblock](d_A, d_B, d_C)

C = d_C.copy_to_host()

print(C[0, 0])
```