



**Pimpri-Chinchwad Educational Trusts**  
**Pimpri-Chinchwad College of Engineering And Research,**  
**Ravet, Pune**



**DEPARTMENT OF COMPUTER ENGINEERING**

**LAB MANUAL**

**LAB PRACTICE V**

	<p>Pimpri Chinchwad Education Trust's  <b>Pimpri Chinchwad College of Engineering &amp;  Research Ravet, Pune</b>  <b>IQAC PCCOER</b></p>	 <p><b>Term: II</b></p>
---	---	--

## LABORATORY MANUAL

**Subject: Laboratory Practice-V**

**[Subject Code: 410254]**

**Class: BE Computer Engineering**

**Semester: II**

Prepared by:

Approved by

Dr.Yogeshwari V. Mahajan  
Mrs. Shrinika Mahajan  
**Subject Teachers**

Dr. Archana A. Chaugule  
**H.O.D**

## **Vision – Mission of the Institute**

### **Vision**

To be a Premier institute of technical education & research to serve the need of society and all the stakeholders.

### **Mission**

To establish state-of-the-art facilities to create an environment resulting in individuals who are technically sound having professionalism, research and innovative aptitude with high moral and ethical values.

## **Vision – Mission of the Computer Department**

### **Vision**

To strive for excellence in the field of Computer Engineering and Research through Creative Problem Solving related to societal needs.

### **Mission:**

**M1:** Establish strong fundamentals, domain knowledge and skills among the students with analytical thinking, conceptual knowledge, social awareness and expertise in the latest tools & technologies to serve industrial demands.

**M2:** Establish leadership skills, team spirit and high ethical values among the students to serve industrial demands and societal needs.

**M3:** Guide students towards Research and Development, and a willingness to learn by connecting themselves to the global society.

### **Program Educational Objectives (PEO)**

**PEO1:** To prepare graduates who have strong mathematical, scientific, and Computer engineering fundamentals, to meet technological challenges and be globally competent.

**PEO2:** To prepare committed and motivated graduates with strong communication, ethical values, leadership skills, which augment their professional competency and make them productive team players.

**PEO3:** To prepare graduates with technical proficiency, research outlook and problem-solving abilities to produce innovative solutions in the field of Computer Engineering.

### **Program Specific Outcomes (PSO)**

**PSO1: Problem-Solving Skills-** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality project.

**PSO2: Professional Skills-** The ability to understand, analyze and develop computer programs in the areas related to algorithms, software testing, application software, web design, data analytics, IOT and networking for efficient design of computer-based systems.

**PSO3: Successful Career and Entrepreneurship-** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies, and to generate IPR & Deliver a quality project..

### Course Objectives:

- To understand and implement searching and sorting algorithms.
- To learn the fundamentals of GPU Computing in the CUDA environment.
- To illustrate the concepts of Artificial Intelligence/Machine Learning(AI/ML).
- To understand Hardware acceleration.
- To implement different deep learning models.

### Course Outcomes:

CO	Statements	Cognitive level of learning
C414.1	<b>Analyze and measure</b> performance of sequential and parallel algorithms.	<b>(Analyze)</b>
C414.2	<b>Design and Implement</b> solutions for multicourse/Distributed/parallel environment.	<b>(Design)</b>
C414.3	<b>Identify and apply</b> the suitable algorithms to solve AI/ML problems.	<b>(Apply)</b>
C414.4	<b>Apply</b> the technique of Deep Neural network for implementing Linear regression and classification.	<b>(Apply)</b>
C414.5	<b>Apply</b> the technique of Convolution (CNN) for implementing Deep Learning models.	<b>(Apply)</b>
C414.6	<b>Design and develop</b> Recurrent Neural Network (RNN) for prediction.	<b>(Design)</b>

## Index

Sr. No.	Title of the Experiment	CO	Date of Performance	Page No.		Sign. of teacher	Remarks*
				From	To		
1	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 .	C414.1					
2	.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.	C414.1					
3	Implement Min, Max, Sum and Average operations using Parallel Reduction.	C414.1					
4	Write a CUDA Program for : 1. Addition of two large vectors 2. Matrix Multiplication using CUDA C	C414.1					
5	<b>Mini Project :</b> (( HPC) Students are Implement any one mini project.	C414.1					
6	Linear regression by using Deep Neural network: Implement Boston housing price predictionproblem by Linear regression using Deep Neural network. Use Boston House price predictiondataset	C414.2					
7	Classification using Deep neural network (Any One from the following) 1. Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset <a href="https://archive.ics.uci.edu/ml/datasets/letter+recognition">https://archive.ics.uci.edu/ml/datasets/letter+recognition</a> 2. 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	C414.2					

8	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.	C414.3					
9	Recurrent neural network (RNN) Use the Google stock prices dataset and design a time seriesanalysis and prediction system using RNN.	C414.3					
10	<b>Mini Project :(Deep Learning)</b> Implement any one mini project.	C414.3					

## **CERTIFICATE**

This is to certify that Mr./Miss/Mrs. \_\_\_\_\_

Roll No.: \_\_\_\_\_ Exam. Seat No.: \_\_\_\_\_ of SE/TE/BE Computer has carried out above practical  
/term work within PCCOER as prescribed by SavitribaiPhule Pune University, Pune during the academic year  
20 -20 . His/Her performance is satisfactory and attendance is \_\_\_\_\_%.

**Date:**

**Faculty I/C**

**HOD**

**Principal**



## 410250: High Performance Computing

### Group A

#### Assignment No. 1

**Problem Statement 1: 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.**

**Objective of the Assignment:** Students should be able to perform Parallel Breadth First Search based on existing algorithms using OpenMP.

Prerequisite: 1. Basic of programming language

2. Concept of BFS

3. Concept of Parallelism

#### What is BFS?

BFS stands for Breadth-First Search. It is a graph traversal algorithm used to explore all the nodes of a graph or tree systematically, starting from the root node or a specified starting point, and visiting all the neighboring nodes at the current depth level before moving on to the next depth level.

The algorithm uses a queue data structure to keep track of the nodes that need to be visited, and marks each visited node to avoid processing it again. The basic idea of the BFS algorithm is to visit all the nodes at a given level before moving on to the next level, which ensures that all the nodes are visited in breadth-first order.

BFS is commonly used in many applications, such as finding the shortest path between two nodes, solving puzzles, and searching through a tree or graph.

#### Example of BFS

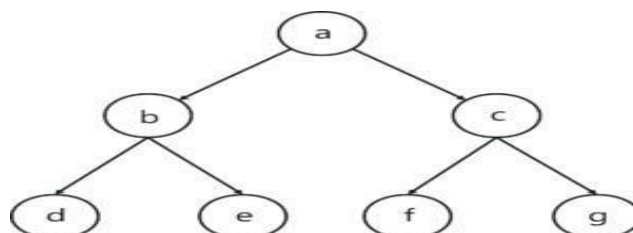
Now let's take a look at the steps involved in traversing a graph by using Breadth-First Search:

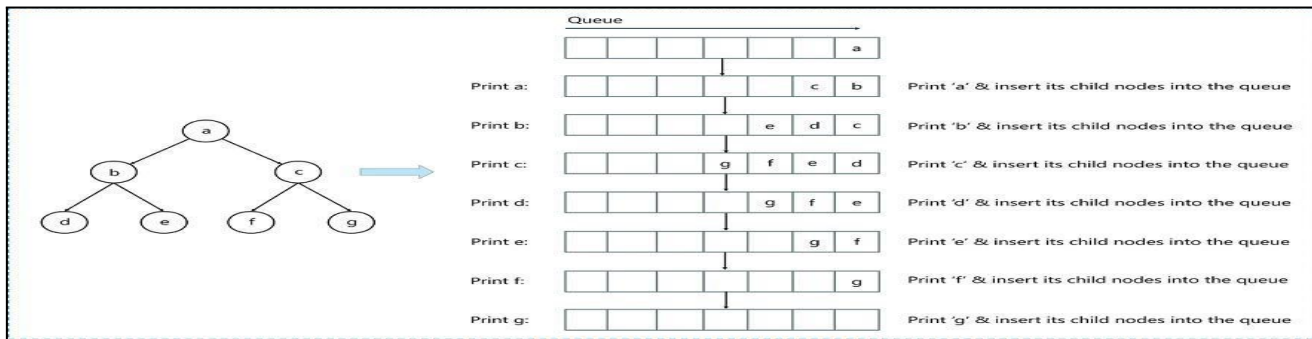
**Step 1:** Take an Empty Queue.

**Step 2:** Select a starting node (visiting a node) and insert it into the Queue.

**Step 3:** Provided that the Queue is not empty, extract the node from the Queue and insert its child nodes(exploring a node) into the Queue.

**Step 4:** Print the extracted node.





## Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.
- OpenMP is widely used in scientific computing, engineering, and other fields that require high-performance computing. It is supported by most modern compilers and is available on a wide range of platforms, including desktops, servers, and supercomputers.

## How Parallel BFS Work

- Parallel BFS (Breadth-First Search) is an algorithm used to explore all the nodes of a graph or tree thematically in parallel. It is a popular parallel algorithm used for graph traversal in distributed computing, shared-memory systems, and parallel clusters.
- The parallel BFS algorithm starts by selecting a root node or a specified starting point, and then assigning it to a thread or processor in the system. Each thread maintains a local queue of nodes to be visited and marks each visited node to avoid processing it again.
- The algorithm then proceeds in levels, where each level represents a set of nodes that are at a certain distance from the root node. Each thread processes the nodes in its local queue at the current level, and then exchanges the nodes that are adjacent to the current level with other threads or processors. This is done to ensure that the nodes at the next level are visited by the next iteration of the algorithm.
- The parallel BFS algorithm uses two phases: the computation phase and the communication phase. In the computation

phase, each thread processes the nodes in its local queue, while in the communication phase, the threads exchange the nodes that are adjacent to the current level with other threads or processors.

- The parallel BFS algorithm terminates when all nodes have been visited or when a specified node has been found. The result of the algorithm is the set of visited nodes or the shortest path from the root node to the target node.
- Parallel BFS can be implemented using different parallel programming models, such as OpenMP, MPI, CUDA, and others. The performance of the algorithm depends on the number of threads or processors used, the size of the graph, and the communication overhead between the threads or processors.

**Conclusion**-In this way we can achieve parallelism while implementing BFS

**Program:**

```
#include<iostream>
#include<stdlib.h>
#include<queue>
using namespace std;

class node
{
public:

    node *left, *right;
    int data;
};

class Breadthfs
{
public:

    node *insert(node *, int);
    void bfs(node *);
};

node *insert(node *root, int data)
// inserts a node in tree
{
    if(!root)
    {
        root=new node;
        root->left=NULL;
        root->right=NULL;
        root->data=data;
        return root;
    }

    queue<node *> q;
    q.push(root);

    while(!q.empty())
    {
        node *temp=q.front();
        q.pop();
```

```
        if(temp->left==NULL)
        {

            temp->left=new node;
            temp->left->left=NULL;
            temp->left->right=NULL;
            temp->left->data=data;
            return root;
        }
        else
        {
            q.push(temp->left);
        }

        if(temp->right==NULL)
        {

            temp->right=new node;
            temp->right->left=NULL;
            temp->right->right=NULL;
            temp->right->data=data;
            return root;
        }
        else
        {
            q.push(temp->right);
        }
    }
}

void bfs(node *head)
{
    queue<node*> q;
    q.push(head);

    int qSize;

    while (!q.empty())
    {
        qSize = q.size();
        #pragma omp parallel for
        //creates parallel threads
        for (int i = 0; i < qSize; i++)
        {
            node* currNode;
            #pragma omp critical
            {
                currNode = q.front();
                q.pop();
                cout<<"\t"<<currNode->data;

                // prints parent node
                #pragma omp critical
```

```

        {
            if(currNode->left)// push parent's left node in queue
                q.push(currNode->left);
            if(currNode->right)
                q.push(currNode->right);
        }// push parent's right node in queue
    }
}
int main(){
    node *root=NULL;
    int data;
    char ans;

    do
    {
        cout<<"\n enter data=>";
        cin>>data;

        root=insert(root,data);

        cout<<"do you want insert one more node?";
        cin>>ans;

    }while(ans=='y'||ans=='Y');

    bfs(root);

    return 0;
}

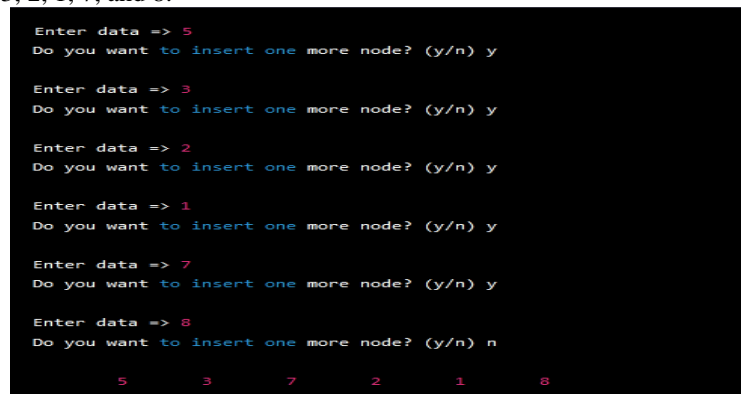
```

Run Commands:

```
g++ -fopenmp bfs.cpp -o bfs
./bfs
```

**Output:**

This code represents a breadth-first search (BFS) algorithm on a binary tree using OpenMP for parallelization. The program asks for user input to insert nodes into the binary tree and then performs the BFS algorithm using multiple threads. Here's an example output for a binary tree with nodes 5, 3, 2, 1, 7, and 8:



```

Enter data => 5
Do you want to insert one more node? (y/n) y

Enter data => 3
Do you want to insert one more node? (y/n) y

Enter data => 2
Do you want to insert one more node? (y/n) y

Enter data => 1
Do you want to insert one more node? (y/n) y

Enter data => 7
Do you want to insert one more node? (y/n) y

Enter data => 8
Do you want to insert one more node? (y/n) n

5      3      7      2      1      8

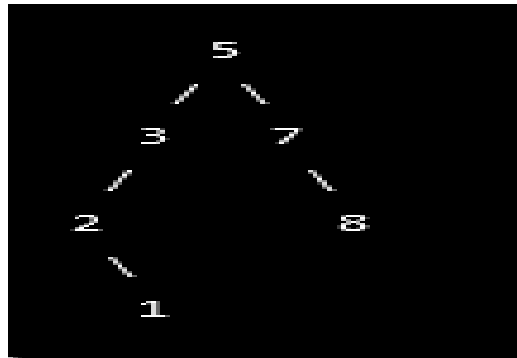
```

The nodes are printed in breadth-first order. The `#pragma omp parallel for` statement is used to parallelize the for loop that processes each level of the binary tree. The `#pragma omp critical` statement is used to synchronize access to shared data structures, such as the

queue that stores the nodes of the binary tree.

Here is an example of the breadth-first traversal for a binary tree with the values 5, 3, 2, 1, 7, and 8:

Starting with the root node containing value 5:



The traversal would be:

```
5, 3, 7, 2, 8, 1
```

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

**Objective of the Assignment:** Students should be able to perform Parallel Depth First Search based on existing algorithms using OpenMP

**Prerequisite:**

1. Basic of programming language
2. Concept of DFS
3. Concept of Parallelism

**Title of the Assignment:** Design and implement Parallel Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for DFS

**Objective of the Assignment:** Students should be able to perform Parallel Depth First Search based on existing algorithms using OpenMP

**Prerequisite:**

4. Basic of programming language
5. Concept of DFS
6. Concept of Parallelism

**Contents for Theory:**

1. What is DFS?
2. Example of DFS
3. Concept of OpenMP
4. How Parallel DFS Work

**Program: 2 Parallel Depth First Search based on existing algorithms using OpenMP**

```
#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>

using namespace std;
```

```
const int MAX = 100000;
vector<int> graph[MAX];
bool visited[MAX];

void dfs(int node) {
    stack<int> s;
    s.push(node);

    while (!s.empty()) {
        int curr_node = s.top();
        s.pop();

        if (!visited[curr_node]) {
            visited[curr_node] = true;

            if (visited[curr_node]) {
                cout << curr_node << " ";
            }

            #pragma omp parallel for
            for (int i = 0; i < graph[curr_node].size(); i++) {
                int adj_node = graph[curr_node][i];
                if (!visited[adj_node]) {
                    s.push(adj_node);
                }
            }
        }
    }

    int main() {
        int n, m, start_node;
        cout << "Enter No of Node,Edges,and start node:" ;
        cin >> n >> m >> start_node;
        //n: node,m:edges

        cout << "Enter Pair of edges:" ;
        for (int i = 0; i < m; i++) {
            int u, v;

            cin >> u >> v;
            //u and v: Pair of edges
            graph[u].push_back(v);
            graph[v].push_back(u);
        }

        #pragma omp parallel for
        for (int i = 0; i < n; i++) {
            visited[i] = false;
        }

        dfs(start_node);

        /*      for (int i = 0; i < n; i++) {
            if (visited[i]) {
                cout << i << " ";
            }
        }
    }
```



```
*/  
  
return 0;  
}
```

**Explanation:**

Let's go through the code step by step:

1. We start by including the necessary headers and declaring some global variables, such as the graph adjacency list, an array to keep track of visited nodes, and a maximum limit for the number of nodes in the graph.
2. Next, we define a function called `dfs()` which takes a starting node as input and performs the depth-first search algorithm. We use a stack to keep track of the nodes to be visited. The algorithm works as follows:
  - We push the starting node onto the stack.
  - While the stack is not empty, we pop the top node from the stack.
  - If the current node has not been visited, we mark it as visited and explore all its neighbors that have not been visited yet by adding them to the stack.
3. In the `main()` function, we read the input values: the number of nodes in the graph, the number of edges, and the starting node.
4. We then read the edge information and build the adjacency list of the graph.
5. We initialize the visited array to false for all nodes in the graph using OpenMP's `parallel for` construct to allow multiple threads to set the array values in parallel.
6. We call the `dfs()` function with the starting node, which performs the depth-first search algorithm in parallel.
7. Finally, we print out the list of visited nodes in the order they were visited.

The parallelization of the DFS algorithm is achieved by using the OpenMP `parallel for` construct inside the `dfs()` function to explore the neighbours of each node in parallel. This allows multiple threads to work on different parts of the graph simultaneously, which can lead to significant performance gains on multi-core CPUs

**step-by-step explanation of how this program works:****dfs function**

This function performs a parallel depth-first search starting from a given node. It takes the node number as a parameter and modifies the global visited array to keep track of which nodes have been visited.

1. It starts by creating an empty stack and pushing the starting node onto it.

A screenshot of a C++ code editor with a dark background. The code shown is:

```
void dfs(int node) {  
    stack<int> s;  
    s.push(node);  
}
```

The editor has a tab labeled "C++" and a "Copy code" button in the top right corner.

2. It enters a loop that continues until the stack is empty.

```
C++ Copy code
while (!s.empty()) {
    int curr_node = s.top();
    s.pop();
```

3. For each node that is popped off the stack, it checks if that node has already been visited. If not, it sets the visited flag for that node to true and adds its unvisited neighbors to the stack in parallel.

```
C++ Copy code
if (!visited[curr_node]) {
    visited[curr_node] = true;

    #pragma omp parallel for
    for (int i = 0; i < graph[curr_node].size(); i++) {
        int adj_node = graph[curr_node][i];
        if (!visited[adj_node]) {
            s.push(adj_node);
        }
    }
}
```

Note that the `#pragma omp parallel for` directive is used to parallelize the loop that adds neighboring nodes to the stack. This can speed up the execution of the program on multi-core machines.

### main function

This function reads in the input, initializes the visited array, calls the dfs function, and prints out the list of visited nodes.

1. It reads in the number of nodes, the number of edges, and the starting node from standard input.

```
C++ Copy code
int main() {
    int n, m, start_node;
    cin >> n >> m >> start_node;
```

2. It reads in the edge list and builds the adjacency list representation of the graph. Note that each edge is added to both the u node's and v node's adjacency list to represent an undirected graph.

```
c++ Copy code
for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    graph[u].push_back(v);
    graph[v].push_back(u);
}
```

3. It initializes the visited array to false in parallel.

```
c++ Copy code
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    visited[i] = false;
}
```

4. It calls the dfs function with the starting node as the argument.

```
c++ Copy code
dfs(start_node);
```

5. It prints out the list of visited nodes in ascending order.

```
c++ Copy code
for (int i = 0; i < n; i++) {
    if (visited[i]) {
        cout << i << " ";
    }
}

return 0;
}
```

### Output:


here's an example input and output for a small graph with 6 nodes and 5 edges:

Input:

```
Copy code
6 7 0
0 1
0 2
1 3
2 4
2 5
4 5
5 3
```

Output:

```
0 1 2 4 5 3
```

 Copy code

**Assignment No: 2**

2. **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.**

**Objective of the Assignment:** Students should be able to Write a program to implement ParallelBubble Sort and can measure the performance of sequential and parallel algorithms.

**Prerequisite:**

1. Basic of programming language
2. Concept of Bubble Sort  
Concept of Parallelism

**What is Bubble Sort?**

Bubble Sort is a simple sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order. It is called "bubble" sort because the algorithm moves the larger elements towards the end of the array in a manner that resembles the rising of bubbles in a liquid.

The basic algorithm of Bubble Sort is as follows:

1. Start at the beginning of the array.
2. Compare the first two elements. If the first element is greater than the second element, swap them.
3. Move to the next pair of elements and repeat step 2.
4. Continue the process until the end of the array is reached.
5. If any swaps were made in step 2-4, repeat the process from step 1.

The time complexity of Bubble Sort is  $O(n^2)$ , which makes it inefficient for large lists. However, it has the advantage of being easy to understand and implement, and it is useful for educational purposes and for sorting small datasets.

Bubble Sort has limited practical use in modern software development due to its inefficient time complexity of  $O(n^2)$  which makes it unsuitable for sorting large datasets. However, Bubble Sort has some advantages and use cases that make it a valuable algorithm to understand, such as:

1. **Simplicity:** Bubble Sort is one of the simplest sorting algorithms, and it is easy to understand and implement. It can be used to introduce the concept of sorting to beginners and as a basis for more complex sorting algorithms.
2. **Educational purposes:** Bubble Sort is often used in academic settings to teach the principles of sorting algorithms and

to help students understand how algorithms work.

3. Small datasets: For very small datasets, Bubble Sort can be an efficient sorting algorithm, as its overhead is relatively low.
4. Partially sorted datasets: If a dataset is already partially sorted, Bubble Sort can be very efficient. Since Bubble Sort only swaps adjacent elements that are in the wrong order, it has a low number of operations for a partially sorted dataset.
5. Performance optimization: Although Bubble Sort itself is not suitable for sorting large datasets, some of its techniques can be used in combination with other sorting algorithms to optimize their performance. For example, Bubble Sort can be used to optimize the performance of Insertion Sort by reducing the number of comparisons needed
6. Example of Bubble sort

Let's say we want to sort a series of numbers 5, 3, 4, 1, and 2 so that they are arranged in ascending order...

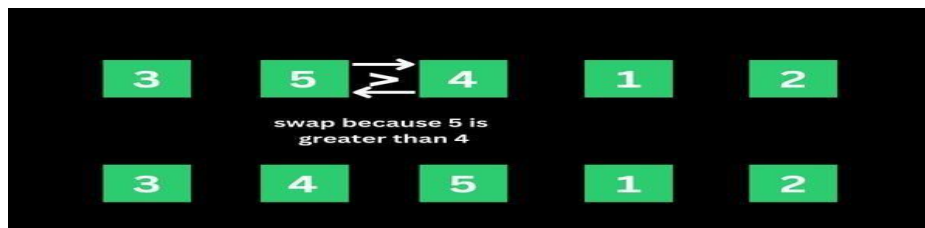
The sorting begins the first iteration by comparing the first two values. If the first value is greater than the second, the algorithm pushes the first value to the index of the second value.

### First Iteration of the Sorting

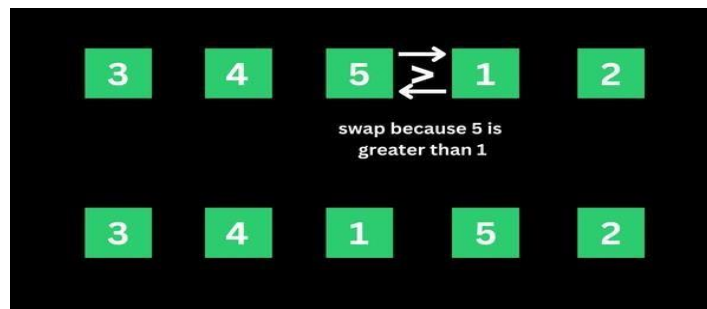
**Step 1:** In the case of 5, 3, 4, 1, and 2, 5 is greater than 3. So 5 takes the position of 3 and the numbers become 3, 5, 4, 1, and 2.



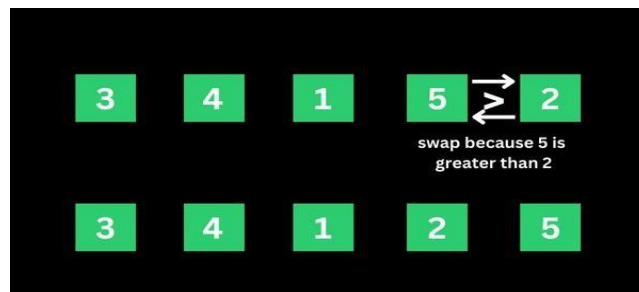
**Step 2:** The algorithm now has 3, 5, 4, 1, and 2 to compare, this time around, it compares the next two values, which are 5 and 4. 5 is greater than 4, so 5 takes the index of 4 and the values now become 3, 4, 5, 1, and 2.



**Step 3:** The algorithm now has 3, 4, 5, 1, and 2 to compare. It compares the next two values, which are 5 and 1. 5 is greater than 1, so 5 takes the index of 1 and the numbers become 3, 4, 1, 5, and 2.



**Step 4:** The algorithm now has 3, 4, 1, 5, and 2 to compare. It compares the next two values, which are 5 and 2. 5 is greater than 2, so 5 takes the index of 2 and the numbers become 3, 4, 1, 2, and 5.

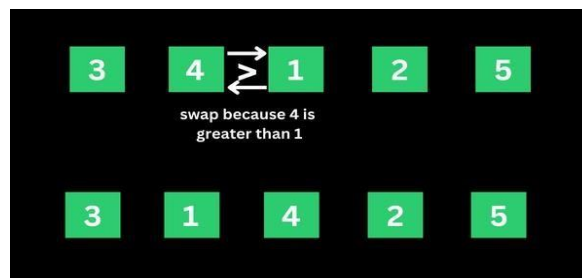


That's the first iteration. And the numbers are now arranged as 3, 4, 1, 2, and 5 – from the initial 5, 3, 4, 1, and 2. As you might realize, 5 should be the last number if the numbers are sorted in ascending order.

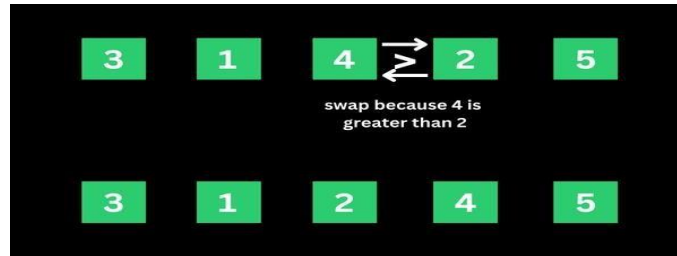
This means the first iteration is really completed.

### Second Iteration of the Sorting and the Rest

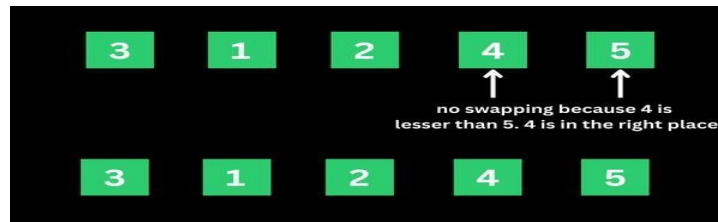
The algorithm starts the second iteration with the last result of 3, 4, 1, 2, and 5. This time around, 3 is smaller than 4, so no swapping happens. This means the numbers will remain the same. The algorithm proceeds to compare 4 and 1. 4 is greater than 1, so 4 is swapped for 1 and the numbers become 3, 1, 4, 2, and 5.



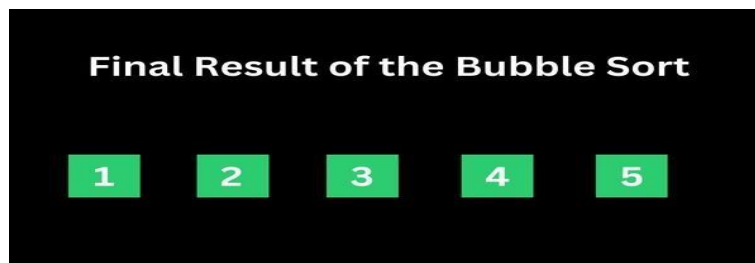
The algorithm now proceeds to compare 4 and 2. 4 is greater than 2, so 4 is swapped for 2 and then numbers become 3, 1, 2, 4, and 5



now in the right place, so no swapping occurs between 4 and 5 because 4 is smaller than 5.



That's how the algorithm continues to compare the numbers until they are arranged in ascending order of 1, 2, 3, 4, and 5.



### Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern

**Program:** Write a program to implement Parallel Bubble Sort using OpenMP



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

void bubble(int *, int);
void swap(int &, int &);

void bubble(int *a, int n)
{
for( int i = 0; i < n; i++ )
{
int first = i % 2;
#pragma omp parallel for shared(a,first)
for( int j = first; j < n-1; j += 2 )
{
if( a[ j ] > a[ j+1 ] )
{
swap( a[ j ], a[ j+1 ] );
}
}
}
}

void swap(int &a, int &b)
{
int test;
test=a;
a=b;
b=test;

}

int main()
{
int *a,n;
cout<<"\n enter total no of elements=>";
cin>>n;
a=new int[n];
cout<<"\n enter elements=>";
```

```
for(int i=0;i<n;i++)
{
cin>>a[i];
}

bubble(a,n);

cout<<"\n sorted array is=";
for(int i=0;i<n;i++)
{
cout<<a[i]<<endl;
}

return 0;
}
```

**Explanation:**

Here, we're using the `#pragma omp parallel` directive to parallelize the code using OpenMP. Within the parallel block, we have a while loop that runs until the array is sorted. Inside the while loop, we have a `#pragma omp for` directive that parallelizes the for loop that performs the bubble sort.

Note that the flag variable is used to keep track of whether any swaps were made during an iteration of the for loop. If no swaps were made, then the array is already sorted and we can exit the loop.

explanation of each line of the code:



```
C++ Copy code

#include <iostream>
#include <omp.h>

using namespace std;
```

These lines are the standard C++ header files used for input/output operations and OpenMP, respectively.

```
c++ Copy code
void parallel_bubble_sort(int arr[], int n) {
    #pragma omp parallel
    {
        int i, temp, flag = 1;
        while (flag) {
            flag = 0;
            #pragma omp for
            for (i = 0; i < n-1; i++) {
                if (arr[i] > arr[i+1]) {
                    temp = arr[i];
                    arr[i] = arr[i+1];
                    arr[i+1] = temp;
                    flag = 1;
                }
            }
        }
    }
}
```

This is the function `parallel_bubble_sort()` that implements the parallel bubble sort algorithm using OpenMP.

- `#pragma omp parallel` is a directive that creates a team of threads to execute the parallel code inside the block. In this case, the block contains the code for bubble sort algorithm.
- `int i, temp, flag = 1;` declares the variables `i`, `temp`, and `flag` that will be used inside the while loop.
- `while (flag)` is a loop that runs until the `flag` variable is 0.
- `flag = 0;` sets the `flag` variable to 0 before starting each iteration of the for loop.
- `#pragma omp for` is a directive that parallelizes the for loop, by dividing the loop iterations among the threads in the team. Each thread performs the sorting operation on a subset of the array, thereby making the sorting process faster.
- `for (i = 0; i < n-1; i++)` is a for loop that iterates over the array, from 0 to `n-1`.
- `if (arr[i] > arr[i+1])` checks if the current element is greater than the next element.
- `temp = arr[i]; arr[i] = arr[i+1]; arr[i+1] = temp;` swaps the current element with the next element, using a temporary variable.
- `flag = 1;` sets the `flag` variable to 1, indicating that a swap has been made.
- Finally, the sorted array is printed using a for loop.

```
c++ Copy code
int main() {
    int arr[] = {5, 3, 1, 9, 8, 2, 4, 7, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    parallel_bubble_sort(arr, n);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

This is the `main()` function, which initializes an array `arr` and its size `n`. The function `parallel_bubble_sort()` is called with these arguments to sort the array. The sorted array is then printed to the console using a for loop.

**How to Run code in Ubuntu:**

1. Open a terminal window.

Compile the code using the following command:

```
g++ -fopenmp filename.cpp -o executable
```

Copy code

Run the program by executing the following command:

```
bash
```

```
./executable
```

Copy code

**Output**

```
c
```

```
Sorted array: 1 2 3 4 5 6 7 8 9
```

Copy code

This is because the input array `{5, 3, 1, 9, 8, 2, 4, 7, 6}` is sorted in ascending order using the parallel bubble sort algorithm implemented in the `parallel_bubble_sort()` function. The sorted array is then printed to the console in the `main()` function using a for loop.

**To measure the performance of sequential Bubble sort and parallel Bubble sort algorithms, you can follow these steps:**

1. Implement both the sequential and parallel Bubble sort algorithms.
2. Choose a range of test cases, such as arrays of different sizes and different degrees of sortedness, to test the performance of both algorithms.
3. Use a reliable timer to measure the execution time of each algorithm on each test case.
4. Record the execution times and analyze the results.

When measuring the performance of the parallel Bubble sort algorithm, you will need to specify the number of threads to use. You can experiment with different numbers of threads to find the optimal value for your system.

Here are some additional tips for measuring performance:

- Run each algorithm multiple times on each test case and take the average execution time to reduce the impact of variations in system load and other factors.
- Monitor system resource usage during execution, such as CPU utilization and memory consumption, to detect any performance bottlenecks.
- Visualize the results using charts or graphs to make it easier to compare the performance of the two algorithms.

**How to check CPU utilisation and memory consumption in ubuntu**

In Ubuntu, you can use a variety of tools to check CPU utilization and memory consumption. Here are some common tools:

1. **top:** The top command provides a real-time view of system resource usage, including CPU utilization and memory consumption. To use it, open a terminal window and type top. The output will display a list of processes sorted by resource usage, with the most resource-intensive processes at the top.
2. **htop:** htop is a more advanced version of top that provides additional features, such as interactive process filtering and a color-coded display. To use it, open a terminal window and type htop.
3. **ps:** The ps command provides a snapshot of system resource usage at a particular moment in time. To use it, open a terminal window and type ps aux. This will display a list of all running processes and their resource usage.
4. **free:** The free command provides information about system memory usage, including total, used, and free memory. To use it, open a terminal window and type free -h.
5. **vmstat:** The vmstat command provides a variety of system statistics, including CPU utilization, memory usage, and disk activity. To use it, open a terminal window and type vmstat.

processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

**How Parallel Bubble Sort Work**

- Parallel Bubble Sort is a modification of the classic Bubble Sort algorithm that takes advantage of parallel processing to speed up the sorting process.
- In parallel Bubble Sort, the list of elements is divided into multiple sublists that are sorted concurrently by multiple threads. Each thread sorts its sublist using the regular Bubble Sort algorithm. When all sublists have been sorted, they are merged together to form the final sorted list.
- The parallelization of the algorithm is achieved using OpenMP, a programming API that supports parallel processing in C++, Fortran, and other programming languages. OpenMP provides a set of compiler directives that allow developers to specify which parts of the code can be executed in parallel.
- In the parallel Bubble Sort algorithm, the main loop that iterates over the list of elements is divided into multiple iterations that are executed concurrently by multiple threads. Each thread sorts a subset of the list, and the threads synchronize their work at the end of each iteration to ensure that the elements are properly ordered.
- Parallel Bubble Sort can provide a significant speedup over the regular Bubble Sort algorithm, especially when sorting large datasets on multi-core processors. However, the speedup is limited by the overhead of thread creation and synchronization, and it may not be worth the effort for small datasets or when using a single-core processor.

### How to measure the performance of sequential and parallel algorithms?

To measure the performance of sequential Bubble sort and parallel Bubble sort algorithms, you can follow these steps:

1. Implement both the sequential and parallel Bubble sort algorithms.
2. Choose a range of test cases, such as arrays of different sizes and different degrees of sortedness, to test the performance of both algorithms. Use a reliable timer to measure the execution time of each algorithm on each test case.
3. Record the execution times and analyze the results.

When measuring the performance of the parallel Bubble sort algorithm, you will need to specify the number of threads to use. You can experiment with different numbers of threads to find the optimal value for your system.

Here are some additional tips for measuring performance:

- Run each algorithm multiple times on each test case and take the average execution time to reduce the impact of variations in system load and other factors.
- Monitor system resource usage during execution, such as CPU utilization and memory consumption, to detect any performance bottlenecks.
- Visualize the results using charts or graphs to make it easier to compare the performance of the two algorithms.

### How to check CPU utilization and memory consumption in ubuntu

In Ubuntu, you can use a variety of tools to check CPU utilization and memory consumption. Here are some common tools:

1. **top:** The top command provides a real-time view of system resource usage, including CPU utilization and memory consumption. To use it, open a terminal window and type top. The output will display a list of processes sorted by resource usage, with the most resource-intensive processes at the top.
2. **htop:** htop is a more advanced version of top that provides additional features, such as interactive process filtering and a color-coded display. To use it, open a terminal window and type htop.
3. **ps:** The ps command provides a snapshot of system resource usage at a particular moment in time. To use it, open a terminal window and type ps aux. This will display a list of all running processes and their resource usage.
4. **free:** The free command provides information about system memory usage, including total, used, and free memory. To use it, open a terminal window and type free -h.
5. **vmstat:** The vmstat command provides a variety of system statistics, including CPU utilization, memory usage, and disk activity. To use it, open a terminal window and type vmstat.

**Conclusion-** In this way we can implement Bubble Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm

**Title of the Assignment:** Write a program to implement Parallel Merge Sort. Use existing algorithms and measure the performance of sequential and parallel algorithms.

**Objective of the Assignment:** Students should be able to Write a program to implement Parallel Merge Sort and can measure the performance of sequential and parallel algorithms.

**Prerequisite:**

1. Basic of programming language
2. Concept of Merge Sort
3. Concept of Parallelism

**What is Merge Sort?**

Merge sort is a sorting algorithm that uses a divide-and-conquer approach to sort an array or a list of elements. The algorithm works by recursively dividing the input array into two halves, sorting each half, and then merging the sorted halves to produce a sorted output.

The merge sort algorithm can be broken down into the following steps:

1. Divide the input array into two halves.
  2. Recursively sort the left half of the array.
  3. Recursively sort the right half of the array.
  4. Merge the two sorted halves into a single sorted output array.
- The merging step is where the bulk of the work happens in merge sort. The algorithm compares the first elements of each sorted half, selects the smaller element, and appends it to the output array. This process continues until all elements from both halves have been appended to the output array.
  - The time complexity of merge sort is  $O(n \log n)$ , which makes it an efficient sorting algorithm for large input arrays. However, merge sort also requires additional memory to store the output array, which can make it less suitable for use with limited memory resources.
  - In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.
  - One thing that you might wonder is what is the specialty of this algorithm. We already have a number of sorting algorithms then why do we need this algorithm? One of the main advantages of merge sort is that it has a time complexity of  $O(n \log n)$ , which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.
  - Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is

often used in conjunction with other algorithms, such as quicksort, to improve the overall performance of a sorting routine.

### Example of Merge sort

Now, let's see the working of merge sort Algorithm. To understand the working of the merge sort

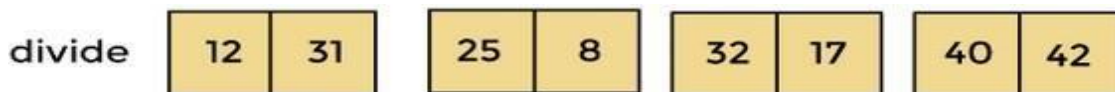
algorithm, let's take an unsorted array. It will be easier to understand the merge sort via an example. Let the elements of array are -

12	31	25	8	32	17	40	42
----	----	----	---	----	----	----	----

- According to the merge sort, first divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.
- As there are eight elements in the given array, so it is divided into two arrays of size 4.



- Now, again divide these two arrays into halves. As they are of size 4, divide them into new arrays of size 2.

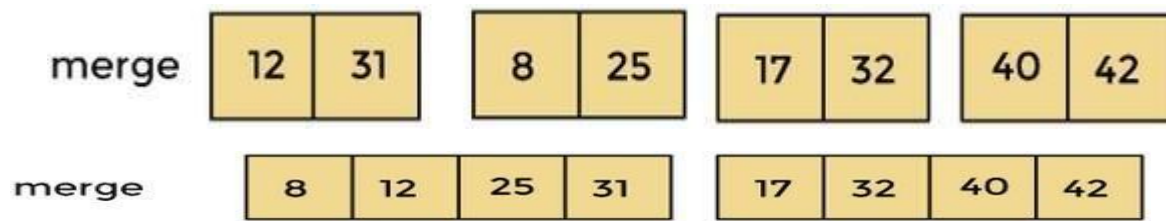


- Now, again divide these arrays to get the atomic value that cannot be further divided.



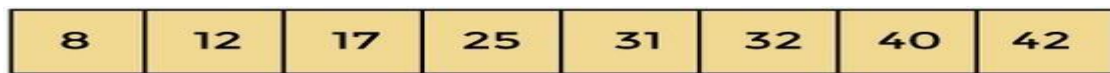
- Now, combine them in the same manner they were broken.
- In combining, first compare the element of each array and then combine them into another array in sorted order.
- So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.





the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.

- Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -



### Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

### How Parallel Merge Sort Work

- Parallel merge sort is a parallelized version of the merge sort algorithm that takes advantage of multiple processors or cores to improve its performance. In parallel merge sort, the input array is divided into smaller subarrays, which are sorted in parallel using multiple processors or cores. The sorted subarrays are then merged together in parallel to produce the final sorted output.
- The parallel merge sort algorithm can be broken down into the following steps:

e the input array into smaller subarrays.

- Assign each subarray to a separate processor or core for sorting.
- Sort each subarray in parallel using the merge sort algorithm.
- Merge the sorted subarrays together in parallel to produce the final sorted output.
- The merging step in parallel merge sort is performed in a similar way to the merging step in the sequential merge sort algorithm. However, because the subarrays are sorted in parallel, the merging step can also be performed in parallel using multiple processors or cores. This can significantly reduce the time required to merge the sorted subarrays and produce the final output.
- Parallel merge sort can provide significant performance benefits for large input arrays with many elements, especially when running on hardware with multiple processors or cores. However, it also requires additional overhead to manage the parallelization, and may not always provide performance improvements for smaller input sizes or when run on hardware with limited parallel processing capabilities.

### **How to measure the performance of sequential and parallel algorithms?**

There are several metrics that can be used to measure the performance of sequential and parallel mergesort algorithms:

1. **Execution time:** Execution time is the amount of time it takes for the algorithm to complete its sorting operation. This metric can be used to compare the speed of sequential and parallel mergesort algorithms.
2. **Speedup:** Speedup is the ratio of the execution time of the sequential merge sort algorithm to the execution time of the parallel merge sort algorithm. A speedup of greater than 1 indicates that the parallel algorithm is faster than the sequential algorithm.
3. **Efficiency:** Efficiency is the ratio of the speedup to the number of processors or cores used in the parallel algorithm. This metric can be used to determine how well the parallel algorithm is utilizing the available resources.
4. **Scalability:** Scalability is the ability of the algorithm to maintain its performance as the input size and number of processors or cores increase. A scalable algorithm will maintain a consistent speedup and efficiency as more resources are added.

To measure the performance of sequential and parallel merge sort algorithms, you can perform experiments on different input sizes and numbers of processors or cores. By measuring the execution time, speedup, efficiency, and scalability of the algorithms under different conditions, you can determine which algorithm is more efficient for different input sizes and hardware configurations. Additionally, you can use profiling tools to analyze the performance of the algorithms and identify areas for optimization.

**Conclusion-** In this way we can implement Merge Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm

**Program: Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.**

**Program:**

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

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

void mergesort(int a[],int i,int j)
{
    int mid;
    if(i<j)
    {
        mid=(i+j)/2;

        #pragma omp parallel sections
        {

            #pragma omp section
            {
                mergesort(a,i,mid);
            }

            #pragma omp section
            {
                mergesort(a,mid+1,j);
            }
        }

        merge(a,i,mid,mid+1,j);
    }
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[1000];
    int i,j,k;
    i=i1;
    j=i2;
    k=0;

    while(i<=j1 && j<=j2)
    {
        if(a[i]<a[j])
        {
            temp[k++]=a[i++];
        }
        else
        {
            temp[k++]=a[j++];
        }
    }

    while(i<=j1)
    {
        temp[k++]=a[i++];
    }
}
```

```

    }

    while(j<=j2)
    {
        temp[k++]=a[j++];
    }

    for(i=i1,j=0;i<=j2;i++,j++)
    {
        a[i]=temp[j];
    }
}

int main()
{
    int *a,n,i;
    cout<<"\n enter total no of elements=>";
    cin>>n;
    a= new int[n];

    cout<<"\n enter elements=>";
    for(i=0;i<n;i++)
    {
        cin>>a[i];
    }
    // start=.....
    //pragma omp....
    mergesort(a, 0, n-1);
    // stop.....
    cout<<"\n sorted array is=>";
    for(i=0;i<n;i++)
    {
        cout<<"\n"<<a[i];
    }
    // Cout<<Stop-Start
    return 0;
}

```

### Second Code:

```

#include <iostream>
#include <omp.h>

void merge(int* arr, int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    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 = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {

```

```

        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int* arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        #pragma omp parallel sections
        {
            #pragma omp section
            {
                mergeSort(arr, l, m);
            }
            #pragma omp section
            {
                mergeSort(arr, m + 1, r);
            }
        }

        merge(arr, l, m, r);
    }
}

int main() {
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);
    double start, stop;
    std::cout << "Given array is: ";
    for (int i = 0; i < n; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;

    start = omp_get_wtime();
    #pragma omp parallel
    {
        mergeSort(arr, 0, n - 1);
    }
    stop = omp_get_wtime();
    std::cout << "Sorted array is: ";
    for (int i = 0; i < n; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
}

```

```
std::cout<<stop-start;
    return 0;
}
```

Explanation

```
C++ Copy code

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

This block of code is a set of header files and a namespace declaration. `iostream` and `stdlib.h` are header files, and `omp.h` is the header file for OpenMP (Open Multi-Processing) library that is used for parallel programming.

```
C++ Copy code

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
```

These are function declarations for mergesort and merge functions.

```
void mergesort(int a[],int i,int j)
{
    int mid;
    if(i<j)
    {
        mid=(i+j)/2;

        #pragma omp parallel sections
        {
            #pragma omp section
            {
                mergesort(a,i,mid);
            }

            #pragma omp section
            {
                mergesort(a,mid+1,j);
            }
        }

        merge(a,i,mid,mid+1,j);
    }
}
```

This is the mergesort function, which implements the merge sort algorithm recursively. `a[]` is the input array that is to be sorted, `i` and `j` are the starting and ending indices of the subarray being sorted.

First, it checks whether `i` is less than `j` because if `i` is greater than or equal to `j`, it means that there is only one element in the subarray, which is already sorted. Then it calculates the midpoint of the subarray using  $(i+j)/2$ .

The `#pragma omp parallel sections` directive creates a parallel region that specifies two parallel sections that can be executed concurrently. The mergesort function is then recursively called on the left and right halves of the subarray in two separate parallel sections.

Finally, the merge function is called to merge the sorted halves of the subarray.

c

Copy code

```
void merge(int a[],int i1,int j1,int i2,int j2)
```

This is the function signature which takes an integer array a and four integer variables i1, j1, i2, and j2. i1 and j1 define the start and end indices of the first sorted subarray, and i2 and j2 define the start and end indices of the second sorted subarray.

c

Copy code

```
int temp[1000];
int i,j,k;
i=i1;
j=i2;
k=0;
```

Here, a temporary array temp is created with a size of 1000. Three integer variables i, j, and k are initialized. i and j are set to the start indices of the two subarrays, while k is set to 0.

c

Copy code

```
while(i<=j1 && j<=j2)
{
    if(a[i]<a[j])
    {
        temp[k++]=a[i++];
    }
    else
    {
        temp[k++]=a[j++];
    }
}
```

This is a while loop that runs as long as i is less than or equal to j1 and j is less than or equal to j2. Inside the loop, if the element at index i of the first subarray is less than the element at index j of the second subarray, then the element at index i is copied to the temp array at index k, and i and k are incremented. Otherwise, the element at index j is copied to the temp array at index k, and j and k are incremented.

```
while(i<=j1)
{
    temp[k++]=a[i++];
}

while(j<=j2)
{
    temp[k++]=a[j++];
}
```

After the above loop terminates, there may be some elements left in one of the subarrays. These loops copy the remaining elements into the temp array.

```
for(i=i1,j=0;i<=j2;i++,j++)  
{  
    a[i]=temp[j];  
}
```

Finally, the sorted temp array is copied back to the original a array. The loop runs from i1 to j2 and copies the elements of temp array to the corresponding indices in the a array. The loop variable j starts from 0 and increments alongside i.



## Assignment No: 3

### 3. Implement Min, Max, Sum and Average operations using Parallel Reduction.

#### Void Min\_reduction()

- `void min_reduction(vector<int>& arr)` declares a void function that takes a reference to an integer vector as its argument.
- `int min_value = INT_MAX;` initializes an integer variable `min_value` to the largest possible integer value using the `INT_MAX` constant from the `<climits>` header file. This is done to ensure that `min_value` is initially greater than any element in `arr`.
- `#pragma omp parallel for reduction(min: min_value)` is an OpenMP directive that specifies that the following loop should be executed in parallel using multiple threads. The `reduction(min: min_value)` clause indicates that each thread should maintain a private copy of `min_value` and update it with the minimum value it finds in its portion of the loop. Once the loop is complete, OpenMP will combine all the private copies of `min_value` into a single shared value that represents the minimum value in `arr`.
- `for (int i = 0; i < arr.size(); i++) {` is a loop that iterates over each element of `arr`.
- `if (arr[i] < min_value) { min_value = arr[i]; }` checks if the current element of `arr` is less than `min_value`. If so, it updates `min_value` to be the current element.
- `cout << "Minimum value: " << min_value << endl;` prints out the minimum value found in `arr`.

#### void max\_reduction()

- `void max_reduction(vector<int>& arr)` declares a void function that takes a reference to an integer vector as its argument.
- `int max_value = INT_MIN;` initializes an integer variable `max_value` to the smallest possible integer value using the `INT_MIN` constant from the `<climits>` header file. This is done to ensure that `max_value` is initially smaller than any element in `arr`.
- `#pragma omp parallel for reduction(max: max_value)` is an OpenMP directive that specifies that the following loop should be executed in parallel using multiple threads. The `reduction(max: max_value)` clause indicates that each thread should maintain a private copy of `max_value` and update it with the maximum value it finds in its portion of the loop. Once the loop is complete, OpenMP will combine all the private copies of `max_value` into a single shared value that represents the maximum value in `arr`.
- `for (int i = 0; i < arr.size(); i++) {` is a loop that iterates over each element of `arr`.
- `if (arr[i] > max_value) { max_value = arr[i]; }` checks if the current element of `arr` is greater than `max_value`. If so, it updates `max_value` to be the current element.
- `cout << "Maximum value: " << max_value << endl;` prints out the maximum value found in `arr`.

**#include <climits>**

<climits> is a header file in C++ that contains constants related to integer types. This header file provides implementation-defined constants for minimum and maximum values of integral types, such as INT\_MAX (maximum value of int) and INT\_MIN (minimum value of int).

Using these constants instead of hardcoding the values of the minimum and maximum integer values is a good practice because it makes the code more readable and avoids the possibility of introducing errors in the code. The use of these constants also ensures that the code will work correctly across different platforms and compilers.

#### **INT\_MIN :**

Minimum value for an object of type int

Value of INT\_MIN is  $-2^{15}+1$  or less\*

#### **INT\_MAX :**

Maximum value for an object of type int

Value of INT\_MAX is  $2^{31}-1$

#### **Program: Implement Min, Max, Sum and Average operations using Parallel Reduction.**

```
#include <iostream>
// #include <vector>
#include <omp.h>
#include <climits>
using namespace std;
void min_reduction(int arr[], int n) {
    int min_value = INT_MAX;
    #pragma omp parallel for reduction(min: min_value)
    for (int i = 0; i < n; i++) {
        if (arr[i] < min_value) {
            min_value = arr[i];
        }
    }
    cout << "Minimum value: " << min_value << endl;
}

void max_reduction(int arr[], int n) {
    int max_value = INT_MIN;
    #pragma omp parallel for reduction(max: max_value)
    for (int i = 0; i < n; i++) {
        if (arr[i] > max_value) {
            max_value = arr[i];
        }
    }
    cout << "Maximum value: " << max_value << endl;
}

void sum_reduction(int arr[], int n) {
    int sum = 0;
    #pragma omp parallel for reduction(+: sum)
    for (int i = 0; i < n; i++) {
```

```

        sum += arr[i];
    }
    cout << "Sum: " << sum << endl;
}

void average_reduction(int arr[], int n) {
    int sum = 0;
    #pragma omp parallel for reduction(+: sum)
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    cout << "Average: " << (double)sum / (n-1) << endl;
}

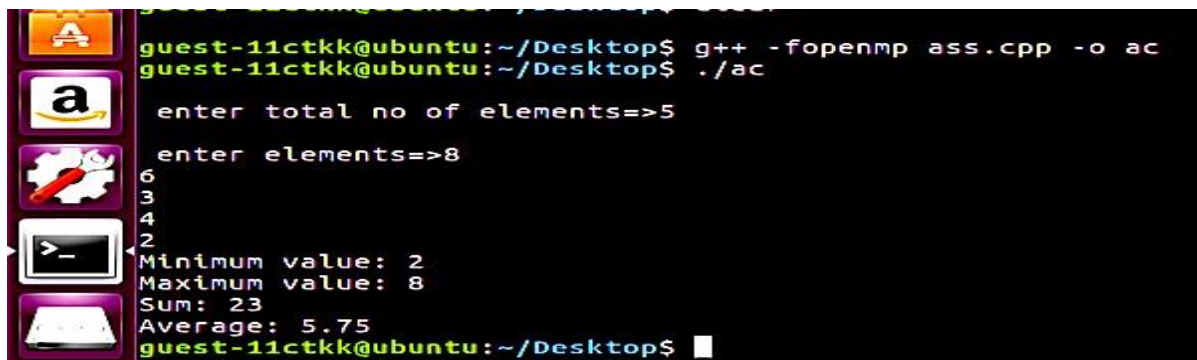
int main() {
    int *arr, n;
    cout << "\n enter total no of elements=>";
    cin >> n;
    arr = new int[n];
    cout << "\n enter elements=>";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }

// int arr[] = {5, 2, 9, 1, 7, 6, 8, 3, 4};
// int n = size(arr);

    min_reduction(arr, n);
    max_reduction(arr, n);
    sum_reduction(arr, n);
    average_reduction(arr, n);
}

```

### Outcome:



```

guest-11ctkk@ubuntu:~/Desktop$ g++ -fopenmp ass.cpp -o ac
guest-11ctkk@ubuntu:~/Desktop$ ./ac

enter total no of elements=>5
enter elements=>8
6
3
4
2
Minimum value: 2
Maximum value: 8
Sum: 23
Average: 5.75
guest-11ctkk@ubuntu:~/Desktop$

```

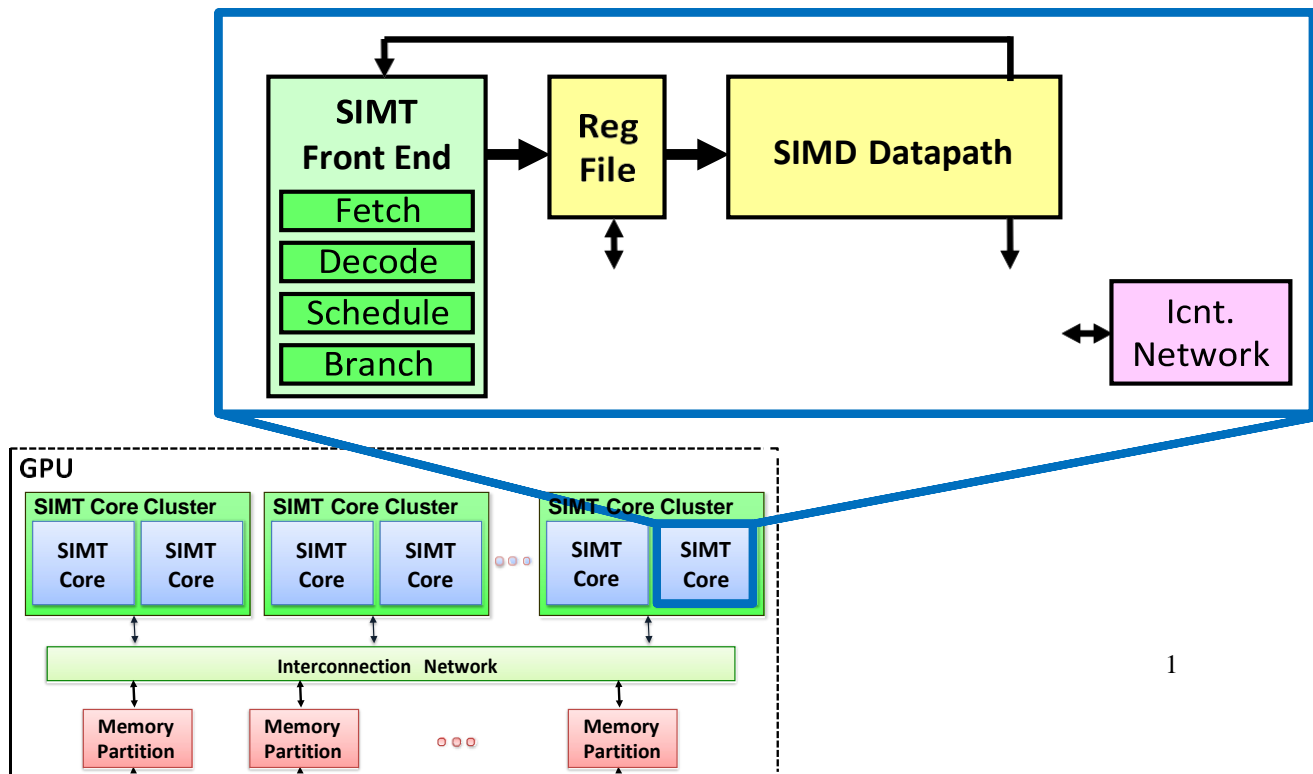
## Assignment No: 4

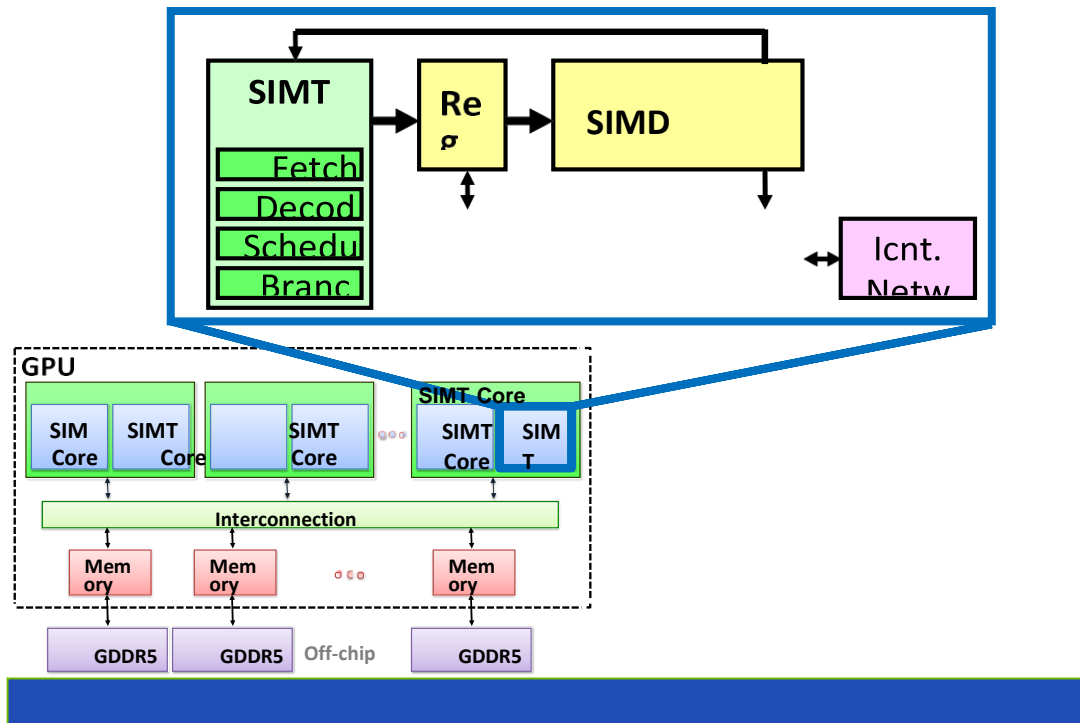
Write a CUDA Program for :

1. Addition of two large vectors
2. Matrix Multiplication using CUDA

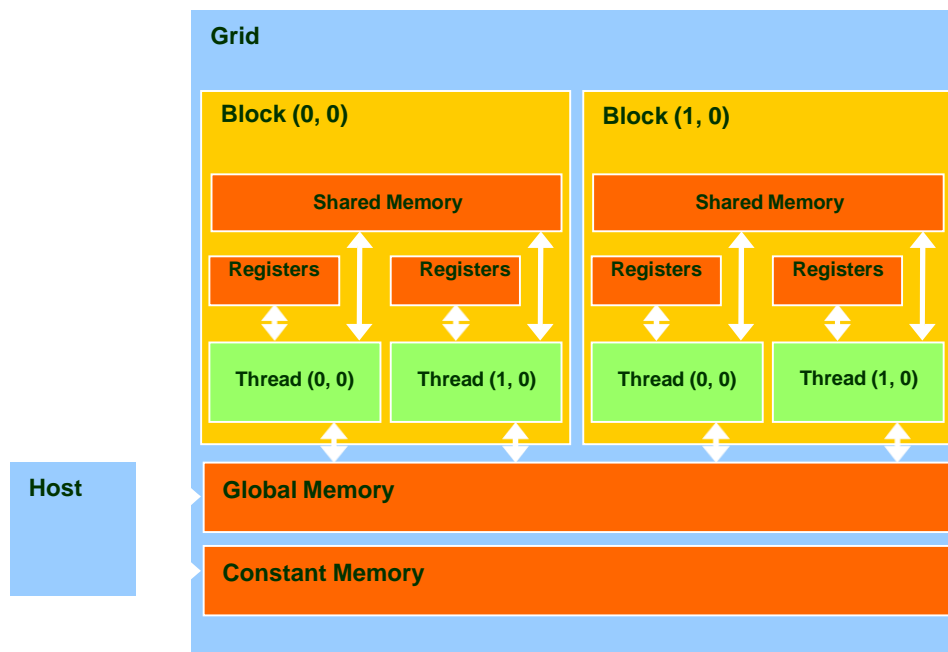
Concept:

	Scal	Vect	Cor	Car
Hardwa				
	ALU	<u>AL</u>	S	GP
Threa				
	Threa	War	Thread Block	Block Grid
Memo	Register		Thread Block	Grid
Addre ss	Local per		Share d	Glob





### Programmer View of CUDA Memories:



## Declaring CUDA Variables:

Variable declaration	Memory	Scope	Lifetime
int LocalVar;	register	thread	thread
__device__ shared _____ int SharedVar;	shared	block	block
__device__ _____ int GlobalVar;	global	grid	application
__device__ constant _____ int ConstantVar;	constant	grid	application

- **device** \_\_\_\_\_ is optional when used with \_\_\_\_\_ **shared** \_\_\_\_\_, or \_\_\_\_\_ **constant** \_\_\_\_\_
- **Automatic variables** reside in a **register**
  - **Except per-thread arrays** that reside in global memory

## Shared Memory in CUDA:

- A special type of memory whose contents are explicitly defined and used in the kernel source code
  - One in each SM
  - Accessed at much higher speed (in both latency and throughput) than global memory
  - Scope of access and sharing - thread blocks
  - Lifetime – thread block, contents will disappear after the corresponding thread finishes terminates execution
  - Accessed by memory load/store instructions
  - A form of scratchpad memory in computer architecture

## Program: 1) Addition of two large vectors

```
#include<iostream>
#include<cstdlib>
using namespace std;
//VectorAdd parallel function
__global__ void vectorAdd(int *a, int *b, int *result, int n)
{
    int tid=threadIdx.x+blockIdx.x*blockDim.x;
    if(tid<n)
    {
```

```

        result[tid]=a[tid]+b[tid];
    }
}
int main()
{
    int *a,*b,*c;
    int *a_dev,*b_dev,*c_dev;
    int n=1<<24;

    a=new int[n];
    b=new int[n];
    c=new int[n];
    int *d=new int[n];
    int size=n*sizeof(int);
    cudaMalloc(&a_dev,size);
    cudaMalloc(&b_dev,size);
    cudaMalloc(&c_dev,size);

    //Array initialization..You can use Random function to assign values
    for(int i=0;i<n;i++)
    {
        a[i]=1;
        b[i]=2;
        d[i]=a[i]+b[i]; //calculating serial addition
    }

    cudaEvent_t start,end;

    cudaEventCreate(&start);
    cudaEventCreate(&end);

    cudaMemcpy(a_dev,a,size,cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev,b,size,cudaMemcpyHostToDevice);
    int threads=1024;
    int blocks=(n+threads-1)/threads;
    cudaEventRecord(start);

    //Parallel addition program
    vectorAdd<<<blocks,threads>>>>(a_dev,b_dev,c_dev,n);

    cudaEventRecord(end);
    cudaEventSynchronize(end);

    float time=0.0;
    cudaEventElapsedTime(&time,start,end);

```

```

        cudaMemcpy(c,c_dev,size,cudaMemcpyDeviceToHost);

        //Calculate the error term.
        int error=0;
        for(int i=0;i<n;i++){
            error+=d[i]-c[i];
            //cout<<" gpu "<<c[i]<<" CPU "<<d[i];
        }

        cout<<"Error : "<<error;
        cout<<"\nTime Elapsed: "<<time;

        return 0;
    }

```

### Program:

## 2) Matrix Multiplication using CUDA C

```

#include<iostream.h>
#include<cstdlib>
#include<cmath>
using namespace std;
//Matrix multiplication Cuda
__global__ void matrixMultiplication(int *a, int *b, int *c, int n)
{
    int row=threadIdx.y+blockDim.y*blockIdx.y;
    int col=threadIdx.x+blockDim.x*blockIdx.x;
    int sum=0;

    if(row<n && col<n)
        for(int j=0;j<n;j++)
        {
            sum=sum+a[row*n+j]*b[j*n+col];
        }

    c[n*row+col]=sum;
}

int main()
{
    int *a,*b,*c;
    int *a_dev,*b_dev,*c_dev;

```



```

int n=3;

a=new int[n*n];
b=new int[n*n];
c=new int[n*n];
int *d=new int[n*n];
int size=n*n*sizeof(int);
cudaMalloc(&a_dev,size);
cudaMalloc(&b_dev,size);
cudaMalloc(&c_dev,size);

//Array initialization
for(int i=0;i<n*n;i++)
{
    a[i]=2; //rand()%n;
    b[i]=1;//rand()%n;
    // d[i]=a[i]+b[i];
}

cudaEvent_t start,end;

cudaEventCreate(&start);
cudaEventCreate(&end);

cudaMemcpy(a_dev,a,size,cudaMemcpyHostToDevice);
cudaMemcpy(b_dev,b,size,cudaMemcpyHostToDevice);


dim3 threadsPerBlock(n, n);
dim3 blocksPerGrid(1, 1);

if(n*n>512){
    threadsPerBlock.x=512;
    threadsPerBlock.y=512;
    blocksPerGrid.x=ceil((double)n/(double)threadsPerBlock.x);
    blocksPerGrid.y=ceil((double)n/(double)threadsPerBlock.y);
}

//GPU Multiplication
cudaEventRecord(start);
matrixMultiplication<<<blocksPerGrid,threadsPerBlock>>>>(a_dev,b_dev,c_dev,n);

```

```

    cudaEventRecord(end);
    cudaEventSynchronize(end);

    float time=0.0;
    cudaEventElapsedTime(&time,start,end);

    cudaMemcpy(c,c_dev,size,cudaMemcpyDeviceToHost);

    //CPU matrix multiplication
    int sum=0;
    for(int row=0;row<n;row++)
    {
        for(int col=0;col<n;col++)
        {
            sum=0;
            for(int k=0;k<n;k++)
                sum=sum+a[row*n+k]*b[k*n+col];
            d[row*n+col]=sum;
        }
    }

    int error=0;
    for(int i=0;i<n*n;i++){
        error+=d[i]-c[i];
        //cout<<" gpu "<<c[i]<<" CPU "<<d[i]<<endl;
    }

    cout<<"Error : "<<error;
    cout<<"\nTime Elapsed: "<<time;

    return 0;
}

```

# Group B: 410251: Deep Learning

## Assignment No: 1

**Title of the Assignment:** 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.

**Objective of the Assignment:** Students should be able to perform Linear regression by using Deep Neural network on Boston House Dataset.

### Prerequisite:

1. Basic of programming language
  2. Concept of Linear Regression
  3. Concept of Deep Neural Network
- 

### Contents for Theory:

1. What is Linear Regression
  2. Example of Linear Regression
  3. Concept of Deep Neural Network
  4. How Deep Neural Network Work
  5. Code Explanation with Output
- 

#### What is Linear Regression?

Linear regression is a statistical approach that is commonly used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables and uses mathematical methods to estimate the coefficients that best fit the data.

Deep neural networks are a type of machine learning algorithm that are modeled after the structure and function of the human brain. They consist of multiple layers of interconnected neurons that process data and learn from it to make predictions or classifications.

Linear regression using deep neural networks combines the principles of linear regression with the power of deep learning algorithms. In this approach, the input features are passed through one or more layers of neurons to extract features and then a linear regression model is applied to the output of the last layer to make predictions. The weights and biases of the neural network are adjusted during training to optimize the performance of the model.

This approach can be used for a variety of tasks, including predicting numerical values, such as stock prices or housing prices, and classifying data into categories, such as detecting whether an image contains a particular object or not. It is often used in fields such as finance, healthcare, and image recognition.

## Example Of Linear Regression

A suitable example of linear regression using deep neural network would be predicting the price of a house based on various features such as the size of the house, the number of bedrooms, the location, and the age of the house.

In this example, the input features would be fed into a deep neural network, consisting of multiple layers of interconnected neurons. The first few layers of the network would learn to extract features from the input data, such as identifying patterns and correlations between the input features.

The output of the last layer would then be passed through a linear regression model, which would use the learned features to predict the price of the house.

During training, the weights and biases of the neural network would be adjusted to minimize the difference between the predicted price and the actual price of the house. This process is known as gradient descent, and it involves iteratively adjusting the model's parameters until the optimal values are reached.

Once the model is trained, it can be used to predict the price of a new house based on its features. This approach can be used in the real estate industry to provide accurate and reliable estimates of house prices, which can help both buyers and sellers make informed decisions.

## Concept of Deep Neural Network

**deep neural network is a type of machine learning algorithm that is modeled after the structure and function of the human brain. It consists of multiple layers of interconnected nodes, or artificial neurons, that process data and learn from it to make predictions or classifications.**

Each layer of the network performs a specific type of processing on the data, such as identifying patterns or correlations between features, and passes the results to the next layer. The layers closest to the input are known as the "input layer", while the layers closest to the output are known as the "output layer".

The intermediate layers between the input and output layers are known as "hidden layers". These layers are responsible for extracting increasingly complex features from the input data, and can be deep (i.e., containing many hidden layers) or shallow (i.e., containing only a few hidden layers).

Deep neural networks are trained using a process known as backpropagation, which involves adjusting the weights and biases of the nodes based on the error between the predicted output and the actual output. This process is repeated for multiple iterations until the model reaches an optimal level of accuracy.

Deep neural networks are used in a variety of applications, such as image and speech recognition, natural language processing, and recommendation systems. They are capable of learning from vast amounts of data and can automatically extract features from raw data, making them a powerful tool for solving complex problems in a wide range of domains.

## How Deep Neural Network Work-

Boston House Price Prediction is a common example used to illustrate how a deep neural network can work for regression tasks. The goal of this task is to predict the price of a house in Boston based on various features such as the number of rooms, crime rate, and accessibility to public transportation.

Here's how a deep neural network can work for Boston House Price Prediction:

1. **Data preprocessing:** The first step is to preprocess the data. This involves normalizing the input features to have a

mean of 0 and a standard deviation of 1, which helps the network learn more efficiently. The dataset is then split into training and testing sets.

2. **Model architecture:** A deep neural network is then defined with multiple layers. The first layer is the input layer, which takes in the normalized features. This is followed by several hidden layers, which can be deep or shallow. The last layer is the output layer, which predicts the house price.
3. **Model training:** The model is then trained using the training set. During training, the weights and biases of the nodes are adjusted based on the error between the predicted output and the actual output. This is done using an optimization algorithm such as stochastic gradient descent.
4. **Model evaluation:** Once the model is trained, it is evaluated using the testing set. The performance of the model is measured using metrics such as mean squared error or mean absolute error.
5. **Model prediction:** Finally, the trained model can be used to make predictions on new data, such as predicting the price of a new house in Boston based on its features.
6. By using a deep neural network for Boston House Price Prediction, we can obtain accurate predictions based on a large set of input features. This approach is scalable and can be used for other regression tasks as well.

### **Boston House Price Prediction Dataset-**

Boston House Price Prediction is a well-known dataset in machine learning and is often used to demonstrate regression analysis techniques. The dataset contains information about 506 houses in Boston, Massachusetts, USA. The goal is to predict the median value of owner-occupied homes in thousands of dollars.

#### **The dataset includes 13 input features, which are:**

**CRIM:** per capita crime rate by town

**ZN:** proportion of residential land zoned for lots over 25,000 sq.ft.

**INDUS:** proportion of non-retail business acres per town

**CHAS:** Charles River dummy variable (1 if tract bounds river; 0 otherwise)

**NOX:** nitric oxides concentration (parts per 10 million)

**RM:** average number of rooms per dwelling

**AGE:** proportion of owner-occupied units built prior to 1940 **DIS:**

weighted distances to five Boston employment centers **RAD:** index of accessibility to radial highways

**TAX:** full-value property-tax rate per \$10,000

**PTRATIO:** pupil-teacher ratio by town

**B:**  $1000(B_k - 0.63)^2$  where  $B_k$  is the proportion of black people by town

**LSTAT:** % lower status of the population

The output variable is the median value of owner-occupied homes in thousands of dollars (MEDV).

To predict the median value of owner-occupied homes, a regression model is trained on the dataset. The model can be a simple linear regression model or a more complex model, such as a deep neural network.

After the model is trained, it can be used to predict the median value of owner-occupied homes based on the input features. The model's accuracy can be evaluated using metrics such as mean squared error or mean absolute error.

Boston House Price Prediction is an example of regression analysis and is often used to teach machine learning concepts. The dataset is also used in research to compare the performance of different regression models.

#### Source Code with Explanation-

```
#Importing the pandas for data processing and numpy for numerical computing
```

```
import numpy as np
```

```
import pandas as pd
```

```
# Importing the Boston Housing dataset from the sklearn from sklearn.datasets import
```

```
load_boston
```

```
boston = load_boston()
```

```
#Converting the data into pandas dataframe data =
```

```
pd.DataFrame(boston.data)
```

```
#First look at the data data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```
#Adding the feature names to the dataframe data.columns =
```

```
boston.feature_names #Adding the target variable to the dataset
```

```
data['PRICE'] = boston.target
```

```
#Looking at the data with names and target variable data.head(n=10)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	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	2.0	242.0	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	2.0	242.0	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	3.0	222.0	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	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10	18.9

#Shape of the data

```
print(data.shape)
```

#Checking the null values in the dataset

```
data.isnull().sum()
```

CRIM 0

ZN 0

INDUS 0

CHAS 0

NOX 0

RM 0

AGE 0

DIS 0

RAD 0

TAX 0

PTRATIO 0

B 0

LSTAT 0

PRICE 0

dtype: int64

#Checking the statistics of the data

```
data.describe()
```

# This is sometimes very useful, for example if you look at the CRIM the max is

88.97 and 75% of the value is below 3.677083 and

# mean is 3.613524 so it means the max values is actually an outlier or there are

outliers present in the column

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000

data.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex:

506 entries, 0 to 505 Data columns (total 14 columns):

#	Column	Non-Null Count		Dtype
0	CRIM	506	non-null	float64
1	ZN	506	non-null	float64
2	INDUS	506	non-null	float64
3	CHAS	506	non-null	float64
4	NOX	506	non-null	float64
5	RM	506	non-null	float64
6	AGE	506	non-null	float64
7	DIS	506	non-null	float64
8	RAD	506	non-null	float64
9	TAX	506	non-null	float64
10	PTRATIO	506	non-null	float64
11	B	506	non-null	float64
12	LSTAT	506	non-null	float64
13	PRICE	506	non-null	float64



```

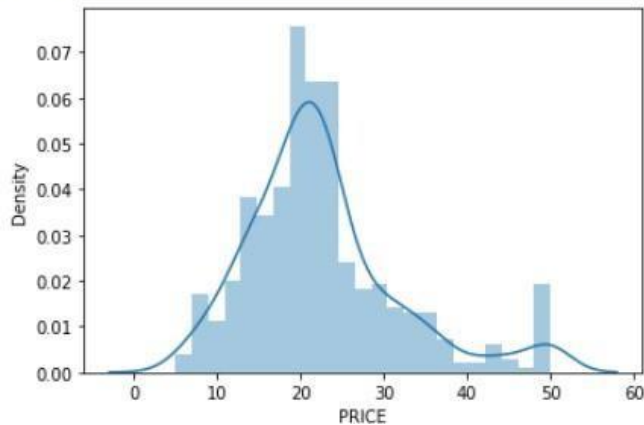
dtypes: float64(14) memory
usage: 55.5 checking the
distribution of the target
variable import seaborn as
sns
sns.distplot(data.PRICE)

```

#The distribution seems normal, has not be the data normal we would have performlog transformation or took to square root of the data to make the data normal.

# Normal distribution is need for the machine learning for better predictiblityof the model

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44d082c670>
```

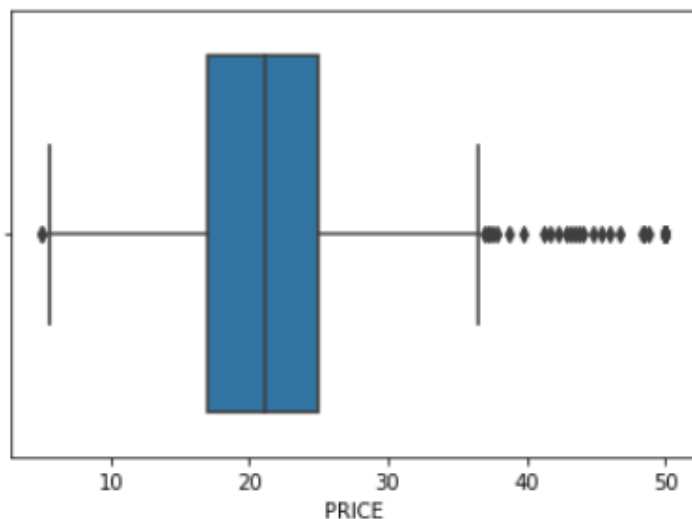


```

#Distribution using box plot
sns.boxplot(data.PRICE)

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44d077ed60>
```



#Checking the correlation of the independent feature with the dependent feature # Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.An intelligent correlation analysis can lead to a greater understanding of your data

#checking Correlation of the data correlation =

```
data.corr().loc['PRICE']
```

```
CRIM          -0.388305
```

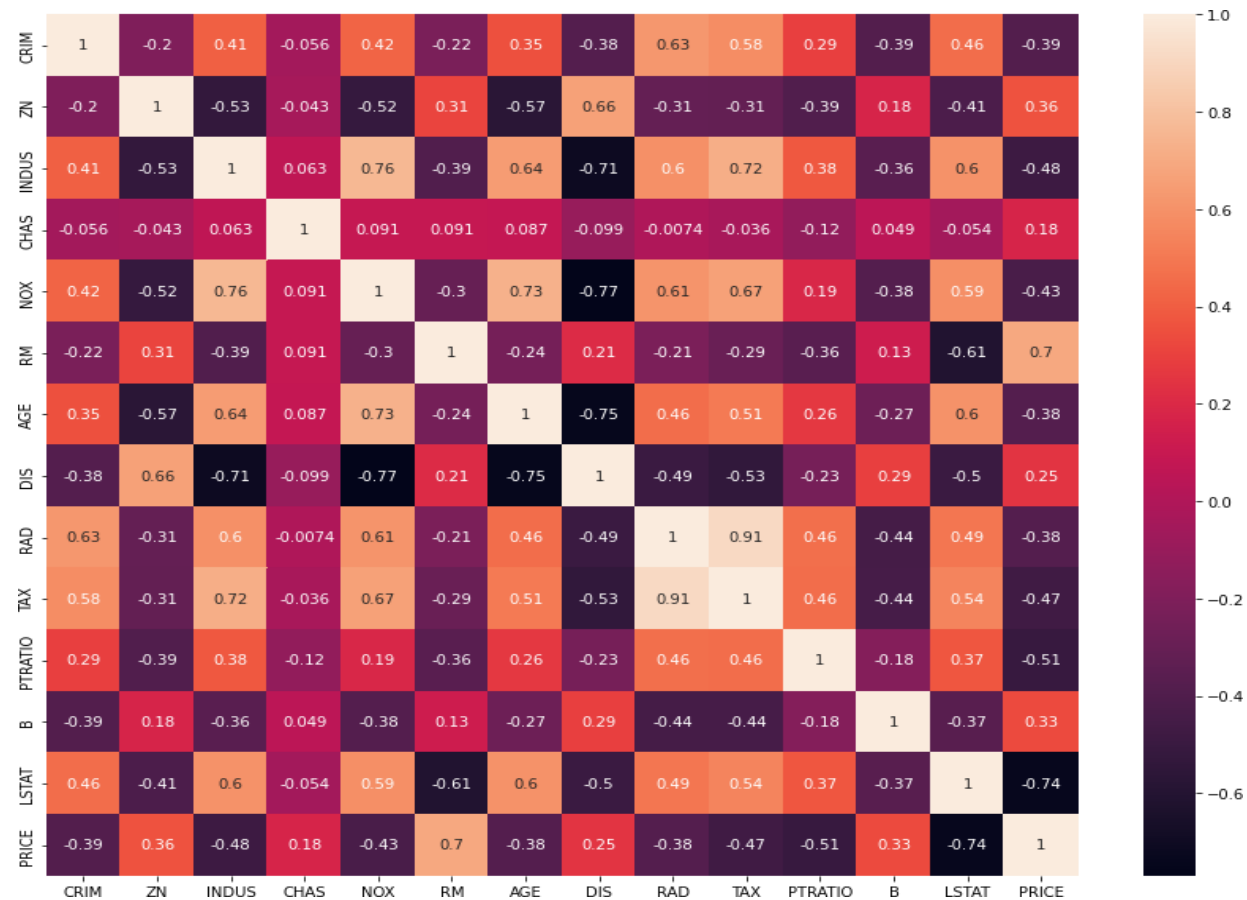
```
Name: PRICE, dtype: float64# plotting the heatmap
```

```
import matplotlib.pyplot as plt
```

```
fig,axes = plt.subplots(figsize=(15,12)) sns.heatmap(correlation,square =
```

```
True,annot = True)
```

```
# By looking at the correlation plot LSTAT is negatively correlated with -0.75 andRM is positively correlated to the price  
and PTRATIO is correlated negatively with -0.51# Checking the scatter plot with the most correlated features
```



```
plt.figure(figsize = (20,5))
```

```
features = ['LSTAT','RM','PTRATIO']
```

```
for i, col in enumerate(features): plt.subplot(1,
```

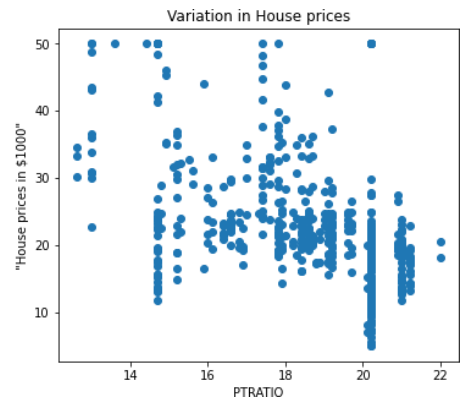
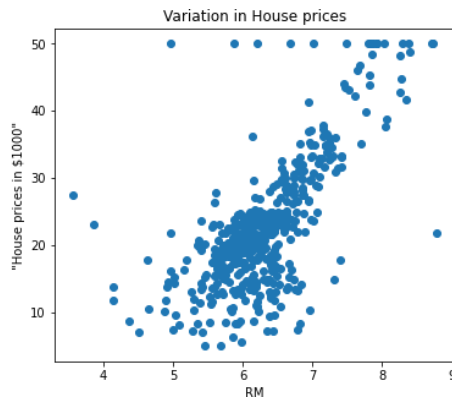
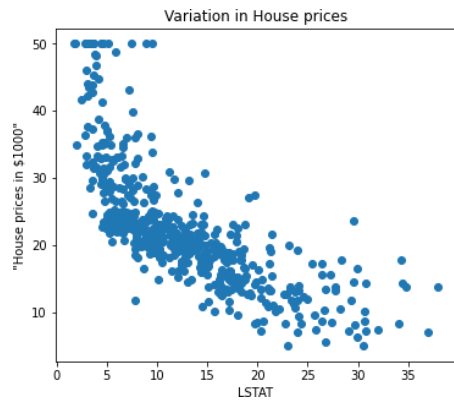
```
len(features) , i+1)x = data[col]
```

```
y = data.PRICE
```

```
plt.scatter(x, y, marker='o') plt.title("Variation in House
```

```
prices")plt.xlabel(col)
```

```
plt.ylabel("House prices in $1000")
```



```
# Splitting the dependent feature and independent feature #X =
data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:, :-1] y=
data.PRICE
```

# In order to provide a standardized input to our neural network, we need to perform the normalization of our dataset.

# This can be seen as a step to reduce the differences in scale that may arise from the existent features.

# We perform this normalization by subtracting the mean from our data and dividing it by the standard deviation.

# One more time, this normalization should only be performed by using the mean and standard deviation from the training set,

# in order to avoid any information leak from the test set.

```
mean = X_train.mean(axis=0) std =
X_train.std(axis=0)
```

```
X_train = (X_train - mean) / std X_test =
(X_test - mean) / std #Linear Regression
```

```
from sklearn.linear_model import LinearRegression regressor = LinearRegression() #Fitting the model
regressor.fit(X_train,y_train)# Model Evaluation
```

```
#Prediction on the test dataset y_pred =
regressor.predict(X_test)
```

# Predicting RMSE the Test set results

```
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred))) print(rmse)
```

```
from sklearn.metrics import r2_score r2 =
r2_score(y_test, y_pred) print(r2)
```

```

# Neural Networks
#Scaling the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Due to the small amount of presented data in this dataset, we must be careful to not create an overly complex model,
# which could lead to overfitting our data. For this, we are going to adopt an architecture based on two Dense layers,
# the first with 128 and the second with 64 neurons, both using a ReLU activation function.
# A dense layer with a linear activation will be used as output layer.

# In order to allow us to know if our model is properly learning, we will use a mean squared error loss function and to
# report the performance of it we will adopt the mean average error metric.
# By using the summary method from Keras, we can see that we have a total of 10,113 parameters, which is
# acceptable for us.

#Creating the neural network model
import keras
from keras.layers import Dense, Activation, Dropout
from keras.models import Sequential

model = Sequential()

model.add(Dense(128, activation='relu', input_dim=13))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))

#model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

!pip install ann_visualizer

!pip install graphviz

from ann_visualizer.visualize import ann_viz; #Build your model here
ann_viz(model, title="DEMO ANN");

history = model.fit(X_train, y_train, epochs=100, validation_split=0.05)

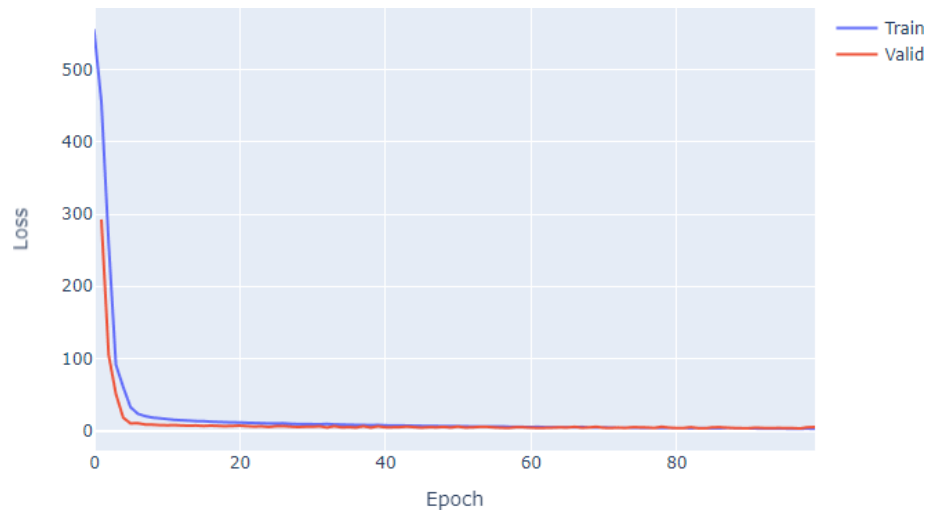
# By plotting both loss and mean average error, we can see that our model was capable of learning patterns in our data
# without overfitting taking place (as shown by the validation set curves)

from plotly.subplots import make_subplots
import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['loss'],
                           name='Train'))

```

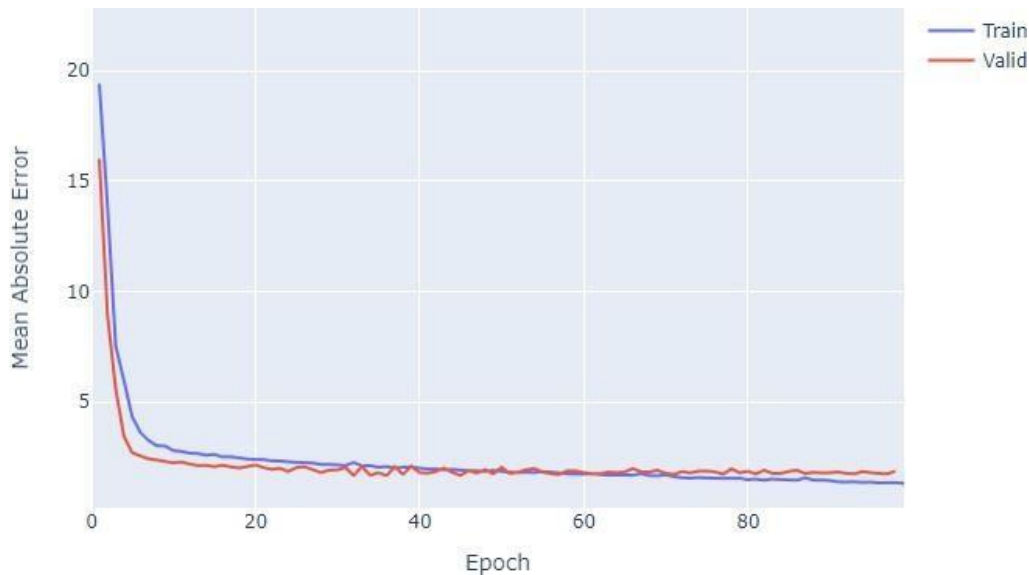
```
fig.add_trace(go.Scattergl(y=history.history['val_loss'],
                           name='Valid'))
fig.update_layout(height=500, width=700,
                  xaxis_title='Epoch',
                  yaxis_title='Loss')
fig.show()
```



```
fig = go.Figure()fig.add_trace(go.Scattergl(y=history.history['mae'],
                                              name='Train'))

fig.add_trace(go.Scattergl(y=history.history['val_mae'],
                           name='Valid'))

fig.update_layout(height=500, width=700,
                  xaxis_title='Epoch', yaxis_title='Mean Absolute
                  Error')
fig.show()
```



```
#Evaluation of the model y_pred =
```

```
model.predict(X_test)
```

```
mse_nn, mae_nn = model.evaluate(X_test, y_test)
```

```
print('Mean squared error on test data: ', mse_nn) print('Mean absolute error  
on test data: ', mae_nn)
```

```
4/4 [=====] - 0s 4ms/step - loss: 10.5717 - mae: 2.2670
```

```
Mean squared error on test data: 10.571733474731445
```

```
Mean absolute error on test data: 2.2669904232025146
```

```
#Comparison with traditional approaches
```

```
#First let's try with a simple algorithm, the Linear Regression:
```

```
from sklearn.metrics import mean_absolute_error
```

```
lr_model = LinearRegression()
```

```
lr_model.fit(X_train, y_train)
```

```
y_pred_lr = lr_model.predict(X_test)
```

```
mse_lr = mean_squared_error(y_test, y_pred_lr) mae_lr =  
mean_absolute_error(y_test, y_pred_lr)
```

```
print('Mean squared error on test data: ', mse_lr) print('Mean absolute error  
on test data: ', mae_lr) from sklearn.metrics import r2_score
```

```
r2 = r2_score(y_test, y_pred) print(r2)
```

```
0.8812832788381159
```

```
# Predicting RMSE the Test set results
```

```

from sklearn.metrics import mean_squared_error

rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))print(rmse)
3.320768607496587

# Make predictions on new data import
sklearn
new_data = sklearn.preprocessing.StandardScaler().fit_transform([[0.1, 10.0,5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300,
10]])
prediction = model.predict(new_data) print("Predicted house
price:", prediction)
1/1 [=====] - 0s 70ms/step

Predicted house price: [[11.104753]]

#new_data =
sklearn.preprocessing.StandardScaler().fit_transform([[0.1, 10.0,
5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]]) is a line of code

```

that standardizes the input features of a new data point.

In this specific case, we have a new data point represented as a list of 13 numeric values ([0.1, 10.0, 5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]) that represents the values for the 13 features of the Boston House Price dataset.

The StandardScaler() function from the sklearn.preprocessing module is used to standardize the data. Standardization scales each feature to have zero mean and unit variance, which is a common preprocessing step in machine learning to ensure that all features contribute equally to the model.

The fit\_transform() method is used to fit the scaler to the data and apply the standardization transformation. The result is a new data point with standardized feature values.

**Conclusion-** In this way we can Predict the Boston House Price using Deep Neural Network.

### Assignment Question

1. What is Linear Regression?
2. What is a Deep Neural Network?
3. What is the concept of standardization?
4. Why split data into train and test?
5. Write Down Application of Deep Neural Network?

## Group B Deep Learning

### Assignment No: 2A

**Title of the Assignment:** 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

**Objective of the Assignment:** Students should be able to Classify movie reviews into positive reviews and "negative reviews on IMDB Dataset.

#### Prerequisite:

1. Basic of programming language
  2. Concept of Classification
  3. Concept of Deep Neural Network
- 

#### Contents for Theory:

1. What is Classification
  2. Example of Classification
  3. How Deep Neural Network Work on Classification
  4. Code Explanation with Output
- 

#### what is Classification?

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the class of new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be used to classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease based on their medical records and symptoms, or to classify images into different categories based on their visual features.

Classification algorithms can vary in complexity, ranging from simple models such as decision trees and k-nearest neighbors to more complex models such as support vector machines and neural networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level of accuracy and interpretability.

Classification is a common task in deep neural networks, where the goal is to predict the class of an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.

The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10



digits.

We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of deep neural network that is commonly used for image classification tasks.

### **How Deep Neural Network Work on Classification-**

Deep neural networks are commonly used for classification tasks because they can automatically learn to extract relevant features from raw input data and map them to the correct output class.

The basic architecture of a deep neural network for classification consists of three main parts: an input layer, one or more hidden layers, and an output layer. The input layer receives the raw input data, which is usually preprocessed to a fixed size and format. The hidden layers are composed of neurons that apply linear transformations and nonlinear activations to the input features to extract relevant patterns and representations. Finally, the output layer produces the predicted class labels, usually as a probability distribution over the possible classes.

During training, the deep neural network learns to adjust its weights and biases in each layer to minimize the difference between the predicted output and the true labels. This is typically done by optimizing a loss function that measures the discrepancy between the predicted and true labels, using techniques such as gradient descent or stochastic gradient descent.

One of the key advantages of deep neural networks for classification is their ability to learn hierarchical representations of the input data. In a deep neural network with multiple hidden layers, each layer learns to capture more complex and abstract features than the previous layer, by building on the representations learned by the earlier layers. This hierarchical structure allows deep neural networks to learn highly discriminative features that can separate different classes of input data, even when the data is highly complex or noisy.

Overall, the effectiveness of deep neural networks for classification depends on the choice of architecture, hyperparameters, and training procedure, as well as the quality and quantity of the training data. When trained properly, deep neural networks can achieve state-of-the-art performance on a wide range of classification tasks, from image recognition to natural language processing.

**IMDB Dataset-**The IMDB dataset is a large collection of movie reviews collected from the IMDB website, which is a popular source of user-generated movie ratings and reviews. The dataset consists of 50,000 movie reviews, split into 25,000 reviews for training and 25,000 reviews for testing.

Each review is represented as a sequence of words, where each word is represented by an integer index based on its frequency in the dataset. The labels for each review are binary, with 0 indicating a negative review and 1 indicating a positive review.

The IMDB dataset is commonly used as a benchmark for sentiment analysis and text classification tasks, where the goal is to classify the movie reviews as either positive or negative based on their text content. The dataset is challenging because the reviews are often highly subjective and can contain complex language and nuances of meaning, making it difficult for traditional machine learning approaches to accurately classify them.

Deep learning approaches, such as deep neural networks, have achieved state-of-the-art performance on the IMDB dataset by automatically learning to extract relevant features from the raw text data and map them to the correct output class. The IMDB dataset is widely used in research and education for natural language processing and machine learning, as it provides a rich source of labeled text data for training and testing deep learning models.

## Source Code and Output-

```
# The IMDB sentiment classification dataset consists of 50,000 movie reviews from IMDB users that are labeled as either
positive (1) or negative (0).
# The reviews are preprocessed and each one is encoded as a sequence of word indexes in the form of integers.
# The words within the reviews are indexed by their overall frequency within the dataset. For example, the integer "2"
encodes the second most frequent word in the data.
# The 50,000 reviews are split into 25,000 for training and 25,000 for testing.

# Text Process word by word at different timestamp ( You may use RNN LSTM GRU )# convert input
text to vector represent input text
# DOMAIN: Digital content and entertainment industry

# CONTEXT: The objective of this project is to build a text classification model that analyses the customer's sentiments based
on their reviews in the IMDB database. The model uses a complex deep learning model to build an embedding layer followed
by a classification algorithm to analyse the sentiment of the customers.
# DATA DESCRIPTION: The Dataset of 50,000 movie reviews from IMDB, labelled by sentiment (positive/negative).
# Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers).
# For convenience, the words are indexed by their frequency in the dataset, meaning the word that has index 1 is the most
frequent word.
# Use the first 20 words from each review to speed up training, using a max vocabulary size of 10,000.
# As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.
# PROJECT OBJECTIVE: Build a sequential NLP classifier which can use input text parameters to determine the customer
sentiments.

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split #loading
imdb data with most frequent 10000 words

from keras.datasets import imdb

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000) # you may take top 10,000 word frequently used
review of movies other are discarded
#consolidating data for EDA Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and
summarize their main characteristics

data = np.concatenate((X_train, X_test), axis=0) # axis 0 is first running vertically downwards across rows (axis 0), axis 1 is
second running horizontally across columns (axis 1),
label = np.concatenate((y_train, y_test), axis=0)

X_train.shape
(25000,)
X_test.shape
(25000,)
y_train.shape
(25000,)
```

```

y_test.shape
(25000,)

print("Review is ",X_train[0]) # series of no converted word to vocabulary associated with index print("Review is
",y_train[0])
Review is [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14,
394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114,
9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5,
89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4,
1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165,
4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255,
5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64,
1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]

Review is 0

vocab=imdb.get_word_index() # Retrieve the word index file mapping words to indicesprint(vocab)
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders':
16115,

y_train
array([1, 0, 0, ..., 0, 1, 0])

y_test
array([0, 1, 1, ..., 0, 0, 0])

# Function to perform relevant sequence adding on the data

# Now it is time to prepare our data. We will vectorize every review and fill it with zeros so that it
contains exactly 10000 numbers.

# That means we fill every review that is shorter than 500 with zeros.

# We do this because the biggest review is nearly that long and every input for our neural network needsto have the same size.
# We also transform the targets into floats.

# sequences is name of method the review less than 10000 we perform padding overthere # binary
vectorization code:

# VECTORIZE as one cannot feed integers into a NN

# Encoding the integer sequences into a binary matrix - one hot encoder basically

# From integers representing words, at various lengths - to a normalized one hot encoded tensor (matrix)of 10k columns
def vectorize(sequences, dimension = 10000): # We will vectorize every review and fill it with zerosso that it
contains exactly 10,000 numbers.

    # Create an all-zero matrix of shape (len(sequences), dimension)results =

```

```

np.zeros((len(sequences), dimension))
for i, sequence in enumerate(sequences):
    results[i, sequence] = 1
return results

```

# Now we split our data into a training and a testing set. # The training set will contain reviews and the testing set # # Set a VALIDATION set

```

test_x = data[:10000] test_y
= label[:10000] train_x =
data[10000:] train_y =
label[10000:]test_x.shape
(10000,)

```

```

test_y.shape
(10000,)

```

```

train_x.shape
(40000,)

```

```

train_y.shape
(40000,)

```

```

print("Categories:", np.unique(label))

```

```

print("Number of unique words:", len(np.unique(np.hstack(data))))

```

# The hstack() function is used to stack arrays in sequence horizontally (column wise).Categories: [0 1]

Number of unique words: 9998 length

```

= [len(i) for i in data]

```

```

print("Average Review length:", np.mean(length))

```

```

print("Standard Deviation:", round(np.std(length)))

```

# The whole dataset contains 9998 unique words and the average review length is 234 words, with a standard deviation of 173 words.

Average Review length: 234.75892

Standard Deviation: 173

# If you look at the data you will realize it has been already pre-processed.

# All words have been mapped to integers and the integers represent the words sorted by their frequency. # This is very common in text analysis to represent a dataset like this.

# So 4 represents the 4th most used word, # 5 the

5th most used word and so on...

# The integer 1 is reserved for the start marker,

# the integer 2 for an unknown word and 0 for padding. # Let's look at a single training example:

```

print("Label:", label[0])

```

Label: 1

```
print("Label:", label[1])
```

Label: 0

```
print(data[0])
```

```
# Retrieves a dict mapping words to their index in the IMDB dataset. index =
```

```
imdb.get_word_index() # word to index
```

```
# Create inverted index from a dictionary with document ids as keys and a list of terms as values foreach document
```

```
reverse_index = dict([(value, key) for (key, value) in index.items()]) # id to word
```

```
decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] )
```

```
# The indices are offset by 3 because 0, 1 and 2 are reserved indices for "padding", "start of sequence" and "unknown".
```

```
print(decoded)
```

```
# this film was just brilliant casting location scenery story direction everyone's really suited the part they
```

```
played and you could just imagine being there robert # is an amazing actor and now the same being director # father came from  
the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the  
film
```

```
#Adding sequence to data
```

```
# Vectorization is the process of converting textual data into numerical vectors and is a process that is usually applied once the  
text is cleaned.
```

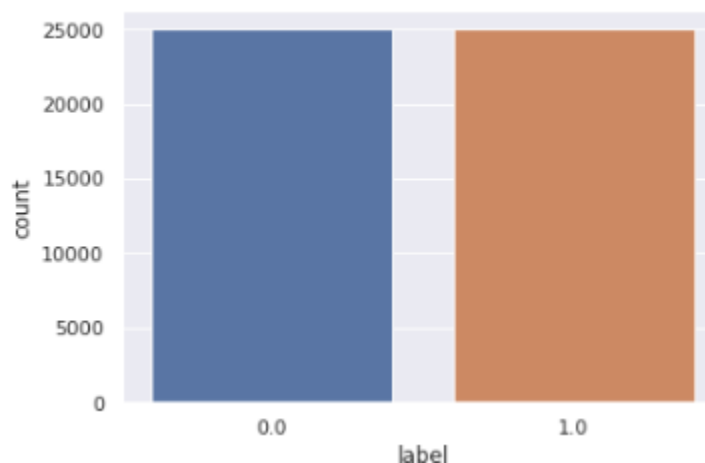
```
data = vectorize(data)
```

```
label = np.array(label).astype("float32")
```

```
labelDF=pd.DataFrame({'label':label})
```

```
sns.countplot(x='label', data=labelDF)
```

<AxesSubplot:xlabel='label', ylabel='count'>



```
# Creating train and test data set
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.20, random_state=1)
```

```
X_train.shape (40000,
```

```

10000)
X_test.shape      (10000,
10000)
# Let's create sequential model

from keras.utils import to_categorical
from keras import models
from keras import layers
model = models.Sequential()# Input - Layer
# Note that we set the input-shape to 10,000 at the input-layer because our reviews are 10,000 integers long.
# The input-layer takes 10,000 as input and outputs it with a shape of 50.
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))

# Hidden - Layers

# Please note you should always use a dropout rate between 20% and 50%. # here in our case 0.3 means 30% dropout we are using dropout to prevent overfitting.
# By the way, if you want you can build a sentiment analysis without LSTMs, then you simply need to replace it by a flatten layer:
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))

# Output- Layer

model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()
Model: "sequential"

```

Layer (type)	Output Shape	Param #
=====		
dense (Dense)	(None, 50)	500050
dropout (Dropout)	(None, 50)	0
dense_1 (Dense)	(None, 50)	2550
dropout_1 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 50)	2550
dense_3 (Dense)	(None, 1)	51

```
=====
```

Total params: 505,201

Trainable params: 505,201

Non-trainable params: 0 #For early

stopping

# Stop training when a monitored metric has stopped improving. # monitor: Quantity to be monitored.

# patience: Number of epochs with no improvement after which training will be stopped.

```
import tensorflow as tf
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
```

# We use the “adam” optimizer, an algorithm that changes the weights and biases during training.

# We also choose binary-crossentropy as loss (because we deal with binary classification) and accuracy as our evaluation metric.

```
model.compile( optimizer =  
    "adam",  
    loss = "binary_crossentropy", metrics =  
    ["accuracy"]  
)
```

```
from sklearn.model_selection import train_test_split
```

```
results = model.fit( X_train,  
    y_train, epochs= 2,  
    batch_size = 500,  
    validation_data = (X_test, y_test), callbacks=[callback]  
)
```

# Let's check mean accuracy of our model

```
print(np.mean(results.history["val_accuracy"])) # Evaluate the model
```

```
score = model.evaluate(X_test, y_test, batch_size=500) print("Test loss:", score[0])
```

```
print("Test accuracy:", score[1])
```

20/20 [=====] - 1s 24ms/step - loss: 0.2511 - accuracy:

0.8986

Test loss: 0.25108325481414795

Test accuracy: 0.8985999822616577

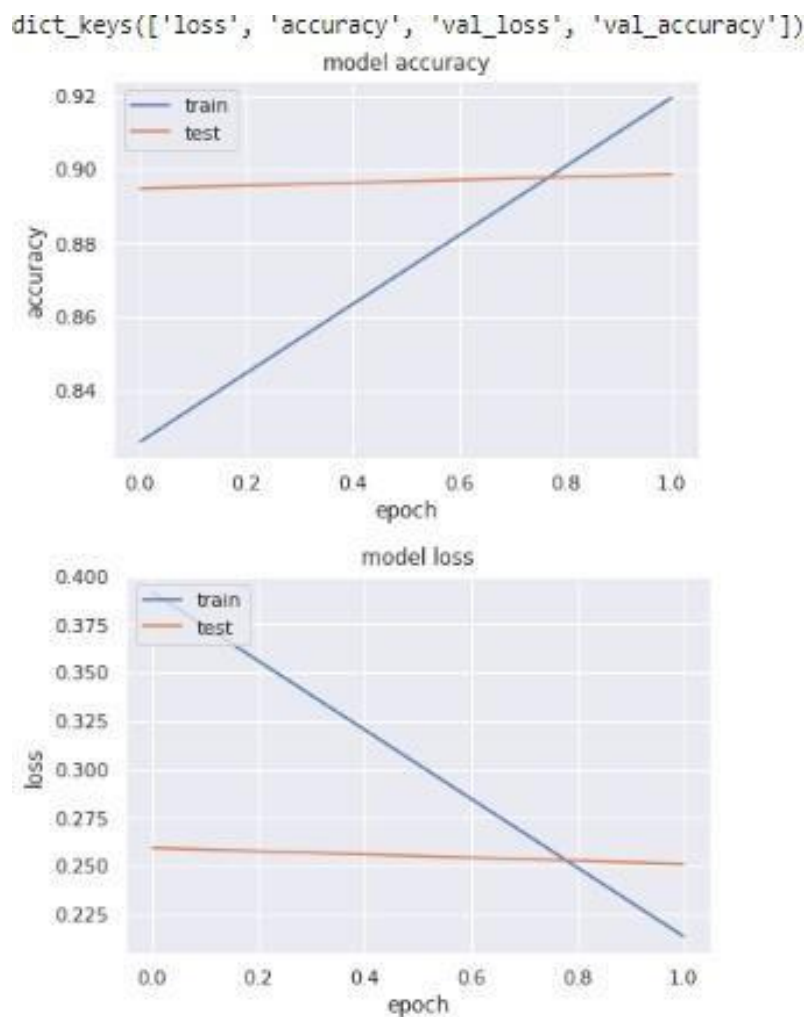
#Let's plot training history of our model.

# list all data in history

```

print(results.history.keys())
# summarize history for accuracy
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])plt.title('model accuracy')
plt.ylabel('accuracy') plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')plt.show()
# summarize history for loss
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss']) plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')plt.show()

```



**Conclusion-** In this way we can Classify the Movie Reviews by using DNN.

### Assignment Question

1. What is Binary Classification?



2. What is binary Cross Entropy?
3. What is Validation Split?
4. What is the Epoch Cycle?
5. What is Adam Optimizer?

## Group B Deep Learning

### Assignment No: 3B

**Title of the Assignment:** Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

**Objective of the Assignment:** Students should be able to Classify movie reviews into positive reviews and "negative reviews on IMDB Dataset.

#### Prerequisite:

1. Basic of programming language
  2. Concept of Classification
  3. Concept of Deep Neural Network
- 

#### Contents for Theory:

1. What is Classification
2. Example of Classification
3. What is CNN?
4. How Deep Neural Network Work on Classification
5. Code Explanation with Output

#### What is Classification?

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the class of new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be used to classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease based on their medical records and symptoms, or to classify images into different categories based on their visual features.

Classification algorithms can vary in complexity, ranging from simple models such as decision trees and k-nearest neighbors to more complex models such as support vector machines and neural networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level of accuracy and interpretability.

**Example-** Classification is a common task in deep neural networks, where the goal is to predict the class of an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.

The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image

is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10 digits.

We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of deep neural network that is commonly used for image classification tasks.

### **What is CNN-**

Convolutional Neural Networks (CNNs) are commonly used for image classification tasks, and they are designed to automatically learn and extract features from input images. Let's consider an example of using a CNN to classify images of handwritten digits. A typical CNN architecture for image classification, there are several layers, including convolutional layers, pooling layers, and fully connected layers. Here's a diagram of a simple CNN architecture for the digit classification task:

The input to the network is an image of size 28x28 pixels, and the output is a probability distribution over the 10 possible digits (0 to 9).

The convolutional layers in the CNN apply filters to the input image, looking for specific patterns and features. Each filter produces a feature map that highlights areas of the image that match the filter. The filters are learned during training, so the network can automatically learn which features are most relevant for the classification task.

The pooling layers in the CNN downsample the feature maps, reducing the spatial dimensions of the data. This helps to reduce the number of parameters in the network, while also making the features more robust to small variations in the input image.

The fully connected layers in the CNN take the flattened output from the last pooling layer and perform a classification task by outputting a probability distribution over the 10 possible digits.

During training, the network learns the optimal values of the filters and parameters by minimizing a loss function. This is typically done using stochastic gradient descent or a similar optimization algorithm.

Once trained, the network can be used to classify new images by passing them through the network and computing the output probability distribution.

Overall, CNNs are powerful tools for image recognition tasks and have been used successfully in many applications, including object detection, face recognition, and medical image analysis.

CNNs have a wide range of applications in various fields, some of which are:

**Image classification:** CNNs are commonly used for image classification tasks, such as identifying objects in images and recognizing faces.

**Object detection:** CNNs can be used for object detection in images and videos, which involves identifying the location of objects in an image and drawing bounding boxes around them.

**Semantic segmentation:** CNNs can be used for semantic segmentation, which involves partitioning an image into segments and assigning each segment a semantic label (e.g., "road", "sky", "building").

**Natural language processing:** CNNs can be used for natural language processing tasks, such as sentiment analysis and text classification.

**Medical imaging:** CNNs are used in medical imaging for tasks such as diagnosing diseases from X-rays and identifying tumors from MRI scans.

**Autonomous vehicles:** CNNs are used in autonomous vehicles for tasks such as object detection and lane detection.

**Video analysis:** CNNs can be used for tasks such as video classification, action recognition, and videocaptioning.

Overall, CNNs are a powerful tool for a wide range of applications, and they have been used successfully in many areas of research and industry.

### **How Deep Neural Network Work on Classification using CNN-**

Deep neural networks using CNNs work on classification tasks by learning to automatically extract features from input images and using those features to make predictions. Here's how it works:

**Input layer:** The input layer of the network takes in the image data as input.

**Convolutional layers:** The convolutional layers apply filters to the input images to extract relevant features. Each filter produces a feature map that highlights areas of the image that match the filter.

**Activation functions:** An activation function is applied to the output of each convolutional layer to introduce non-linearity into the network.

**Pooling layers:** The pooling layers downsample the feature maps to reduce the spatial dimensions of the data.

**Dropout layer:** Dropout is used to prevent overfitting by randomly dropping out a percentage of the neurons in the network during training.

**Fully connected layers:** The fully connected layers take the flattened output from the last pooling layer and perform a classification task by outputting a probability distribution over the possible classes.

**Softmax activation function:** The softmax activation function is applied to the output of the last fully connected layer to produce a probability distribution over the possible classes.

**Loss function:** A loss function is used to compute the difference between the predicted probabilities and the actual labels.

**Optimization:** An optimization algorithm, such as stochastic gradient descent, is used to minimize the loss function by adjusting the values of the network parameters.

**Training:** The network is trained on a large dataset of labeled images, adjusting the values of the parameters to minimize the loss function.

**Prediction:** Once trained, the network can be used to classify new images by passing them through the network and computing the output probability distribution.

### **MNIST Dataset-**

The MNIST Fashion dataset is a collection of 70,000 grayscale images of 28x28 pixels, representing 10 different categories of clothing and accessories. The categories include T-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots.

The dataset is often used as a benchmark for testing image classification algorithms, and it is considered a more challenging version of the original MNIST dataset which contains handwritten digits. The MNIST Fashion dataset was released by Zalando Research in 2017 and has since become a popular dataset in the machine learning community.

The MNIST Fashion dataset is a collection of 70,000 grayscale images of 28x28 pixels each. These images represent 10 different categories of clothing and accessories, with each category containing 7,000 images. The categories are as follows:

T-shirt/tops

Trousers

Pullovers

Dresses

Coats

Sandals

Shirts

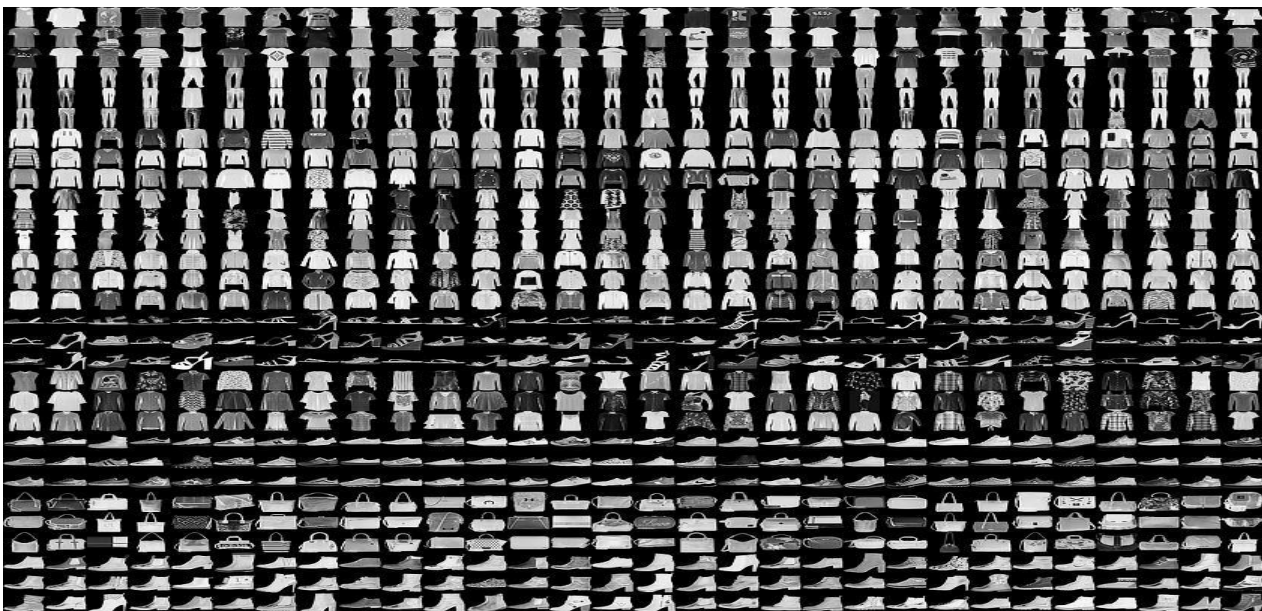
Sneakers

Bags

Ankle boots

The images were obtained from Zalando's online store and are preprocessed to be normalized and centered. The training set contains 60,000 images, while the test set contains 10,000 images. The goal of the dataset is to accurately classify the images into their respective categories.

The MNIST Fashion dataset is often used as a benchmark for testing image classification algorithms, and it is considered a more challenging version of the original MNIST dataset which contains handwritten digits. The dataset is widely used in the machine learning community for research and educational purposes. Here are the general steps to perform Convolutional Neural Network (CNN) on the MNIST Fashion dataset:



- Import the necessary libraries, including TensorFlow, Keras, NumPy, and Matplotlib.
- Load the dataset using Keras' built-in function, `keras.datasets.fashion_mnist.load_data()`. This will provide the training and testing sets, which will be used to train and evaluate the CNN.
- Preprocess the data by normalizing the pixel values between 0 and 1, and reshaping the images to be of size (28, 28, 1) for compatibility with the CNN.
- Define the CNN architecture, including the number and size of filters, activation functions, and pooling layers. This can vary based on the specific problem being addressed.
- Compile the model by specifying the loss function, optimizer, and evaluation metrics. Common choices include categorical cross-entropy, Adam optimizer, and accuracy metric.
- Train the CNN on the training set using the `fit()` function, specifying the number of epochs and batch size.
- Evaluate the performance of the model on the testing set using the `evaluate()` function. This will provide metrics such as accuracy and loss on the test set.
- Use the trained model to make predictions on new images, if desired, using the `predict()` function.

## Source Code with Output-

```
import tensorflow as tf

import matplotlib.pyplot as plt
from tensorflow import keras
import numpy as np
```

```
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
```

# There are 10 image classes in this dataset and each class has a mapping corresponding to the following labels:

#0 T-shirt/top

#1 Trouser

#2 pullover

#3 Dress

#4 Coat

#5 sandals

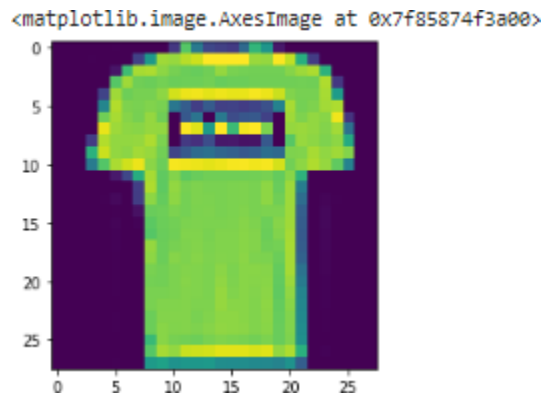
#6 shirt

#7 sneaker

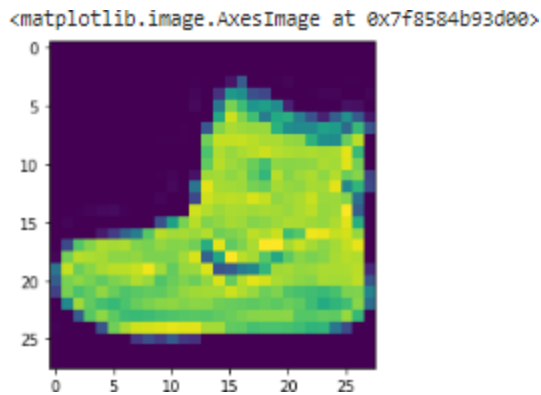
#8 bag

#9 ankle boot

```
plt.imshow(x_train[1])
```



```
plt.imshow(x_train[0])
```



# Next, we will preprocess the data by scaling the pixel values to be between 0 and 1, and then reshaping the images to be 28x28 pixels.

```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
x_train = x_train.reshape(-1, 28, 28, 1)
```

```
x_test = x_test.reshape(-1, 28, 28, 1)
```

# 28, 28 comes from width, height, 1 comes from the number of channels# -1 means that the length in that dimension is inferred.

# This is done based on the constraint that the number of elements in an ndarray or Tensor when reshaped must remain the same.

# each image is a row vector (784 elements) and there are lots of such rows (let it be n, so there are 784n elements). So TensorFlow can infer that -1 is n.

# converting the training\_images array to 4 dimensional array with sizes 60000, 28, 28, 1 for 0th to 3rd dimension.

```
x_train.shape
```

```
(60000, 28, 28)
```

```
x_test.shape (10000,
```

```
28, 28, 1)
```

```
y_train.shape
```

```
(60000,)
```

```
y_test.shape
```

```
(10000,)
```

# We will use a convolutional neural network (CNN) to classify the fashion items.# The CNN will consist of multiple convolutional layers followed by max pooling, # dropout, and dense layers.

Here is the code for the model:

```
model = keras.Sequential([
```

```
    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)), # 32 filters
```

```

(default), randomly initialized
# 3*3 is Size of Filter

# 28,28,1 size of Input Image

# No zero-padding: every output 2 pixels less in every dimension

# in Parameter shown 320 is value of weights: (3x3 filter weights + 32 bias) * 32 filters #
 $32 * 3 * 3 = 288$  (Total) + 32 (bias) = 320
keras.layers.MaxPooling2D((2,2)),

# It shown 13 * 13 size image with 32 channel or filter or depth.
keras.layers.Dropout(0.25),

# Reduce Overfitting of Training sample drop out 25% Neuron
keras.layers.Conv2D(64, (3,3), activation='relu'),
# Deeper layers use 64 filters #
3*3 is Size of Filter
# Observe how the input image on 28x28x1 is transformed to a 3x3x64 feature map

#  $13(\text{Size}) - 3(\text{Filter Size}) + 1(\text{bias}) = 11$  Size for Width and Height with 64 Depth or filter or channel # in Parameter
shown 18496 is value of weights: (3x3 filter weights + 64 bias) * 64 filters
#  $64 * 3 * 3 = 576 + 1 = 577 * 32 + 32(\text{bias}) = 18496$ 

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

# It shown 5 * 5 size image with 64 channel or filter or depth.
keras.layers.Dropout(0.25),

keras.layers.Conv2D(128, (3,3), activation='relu'), # Deeper
layers use 128 filters
# 3*3 is Size of Filter

# Observe how the input image on 28x28x1 is transformed to a 3x3x128 feature map

# It show  $5(\text{Size}) - 3(\text{Filter Size}) + 1(\text{bias}) = 3$  Size for Width and Height with 64 Depth or filter or channel
#  $128 * 3 * 3 = 1152 + 1 = 1153 * 64 + 64(\text{bias}) = 73856$ 

# To classify the images, we still need a Dense and Softmax layer.

# We need to flatten the 3x3x128 feature map to a vector of size 1152
keras.layers.Flatten(),
keras.layers.Dense(128, activation='relu'), # 128 Size
of Node in Dense Layer
#  $1152 * 128 = 147584$ 

keras.layers.Dropout(0.25), keras.layers.Dense(10,
activation='softmax') # 10 Size of Node another Dense
Layer
#  $128 * 10 + 10 \text{ bias} = 1290$ 

])

```



```
model.summary()
```

```
Model: "sequential"
```

---

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147584
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
=====		

```
Total params: 241,546
```

```
Trainable params: 241,546
```

```
Non-trainable params: 0
```

```
# Compile and Train the Model
```

```
# After defining the model, we will compile it and train it on the training data. model.compile(optimizer='adam',
```

```
loss='sparse_categorical_crossentropy', metrics=['accuracy']) history = model.fit(x_train, y_train, epochs=10,
validation_data=(x_test, y_test))
# 1875 is a number of batches. By default batches contain 32 samples. 60000 / 32 = 1875 # Finally, we
will evaluate the performance of the model on the test data.
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
313/313 [=====] - 3s 10ms/step - loss: 0.2606 - accuracy: 0.9031
Test accuracy: 0.9031000137329102
```

**Conclusion-** In this way we can Classify fashion clothing into categories using CNN.

### Assignment Question

1. What is Binary Classification?
2. What is binary Cross Entropy?
3. What is Validation Split?
4. What is the Epoch Cycle
5. What is Adam Optimizer?

## **Experiment No 9**

**TITLE:** Recurrent neural network (RNN) Use the Google stock prices dataset and design a time series analysis and prediction system using RNN

**PROBLEM STATEMENT /DEFINITION:** Implement time series analysis and stock prediction system using RNN by using google stock price dataset.

**OBJECTIVE:** To build a RNN model to predict the stock price and time analysis .

**OUTCOME:** Able to understand the exploratory data analysis,split the training and testing data, Model Evaluation and Prediction by the RNN on the google stock price dataset.

**S/W PACKAGES AND HARDWARE/ APPARATUS USED:** upyter notebook IDE, python3  
PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15"Color Monitor, Keyboard, Mouse.

**REFERENCES:** <https://www.kaggle.com/code/kandij/google-stock-prediction-using-lstm-keras>  
<https://medium.com/analytics-vidhya/share-price-prediction-using-rnn-and-lstm-8776456dea6f>

**STEPS:** Installing Jupyter notebook with python3  
import the required libraries-numpy,matplotlib.pyplot,pandas,seaborn

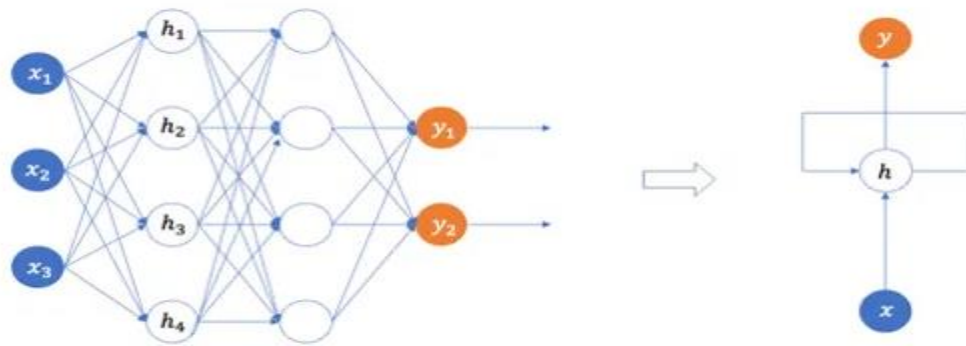
importing Data (kaggle and scikit-learn library) and Checking out

Exploratory Data Analysis for stock price Prediction and time series analysis

*Split Data into Train, Test*

**Concepts related Theory:**

Recursive Neural Network (RNN) should be specially designed. RNN translates the provided inputs to machine readable vectors. Then the system processes each of this sequence of vectors one by one, moving from very first vector to the next one in a sequential order. While processing, the system passes the information through the hidden state (memory) to the next step of the sequence. Once the hidden state has collected all the existing information in the system until time period t, it is ready to move towards the next step and in this newer step the hidden step is classified as the previous hidden state defined. The outputs of the previous periods should somewhat become the inputs of the current periods. And the hidden layers will recursively take the inputs of previous periods. The hidden layer receives the inputs from the input layer, and there is a line to connect a hidden layer back to itself to represent the recursive nature.



### Training through RNN

1. A single time step of the input is provided to the network.
2. Then calculate its current state using set of current input and the previous state.
3. The current  $h_t$  becomes  $h_{t-1}$  for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

### Advantages of Recurrent Neural Network

1. An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
2. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.

### Disadvantages of Recurrent Neural Network

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

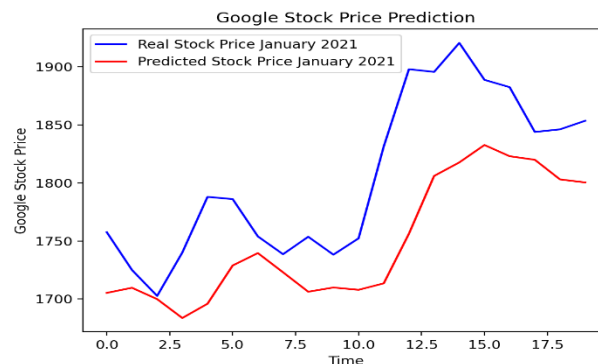
For example , we have considered the amazon stock data for prediction and explained in detail in the similar way Google stock prediction should be implemented and plot the graph

### Algorithm:

1. Import Libraries: Install the required libraries and setup for the environment. importing SciKit-Learn, Pandas, Seaborn, Matplotlib ,Tensorflow and Numpy.
2. Importing Data and Checking out: As data is in the CSV file, we will read the CSV using pandas read\_csv function and check the first 5 rows of the data frame using head().
3. Visualize the data using matplotlib
4. Training the data.
5. Creating the Training data
6. Building the Long Short-Term Model
7. Compiling the model
8. Training the model
9. Creating the test data
10. To get predicted price values
11. Evaluation and Visualizing the predicted values of the Google stock prices

#### Prediction Results:

Comparing the real Google stock values and predicted Google stock values that are generated using RNN model, for the test period, the first month of 2021. RNN based on 5 LSTMs was able to properly predict all upward and downward trends as we see that the red line corresponding to the predicted stock prices follows the same pattern as the blue line which corresponds to the real stock prices.



**Conclusion:** We have analyzed a RNN Model on amazon stock prediction and in similar way Google stock prediction can be calculated whose prediction model will be as below graph.

#### Review Questions:

1. What's the difference between Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN)

and in which cases would use each one?

2. How many dimensions must the inputs of an RNN layer have? What does each dimension
3. represent? What about its outputs?
4. What are the main difficulties when training RNNs?