Sinhgad Institute of Technology & Science, Narhe
Pune-411041

Department of Computer Engineering

LAB MANUAL

Laboratory Practice V

(410255)

Semester-VIII

Companion Courses:

High Performance Computing (410250)

Deep Learning (410251)

INDEX

Name of the Subject/Course: **410255 - Laboratory Practice V**

Class: **BE**

Sr. No.	Assignment Title
	Group A: High Performance Computing
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.
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.
3	Implement Min, Max, Sum and Average operations using Parallel Reduction.
4	Write a CUDA Program for : 1. Addition of two large vectors 2. Matrix Multiplication using CUDA C
	Group B: Deep Learning
5	**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.
6	**Classification using Deep neural network:** 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.
7	**Convolutional neural network (CNN):** Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.
8	**Recurrent neural network (RNN)** Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.
9	Mini Project on HPC.
10	Mini Project on DL.

Assignment No.1

Aim: Design and implement Parallel Breadth First Search and Depth First Search based on existing algorithms using OpenMP. For

1. Use a Tree
2. An undirected graph for BFS and DFS.

Objective: Student will be able to learn

1. The Basic Concepts of DFS, BFS.
2. Multiple Compiler Directives, library routines, environment variables available for OpenMP

Theory:

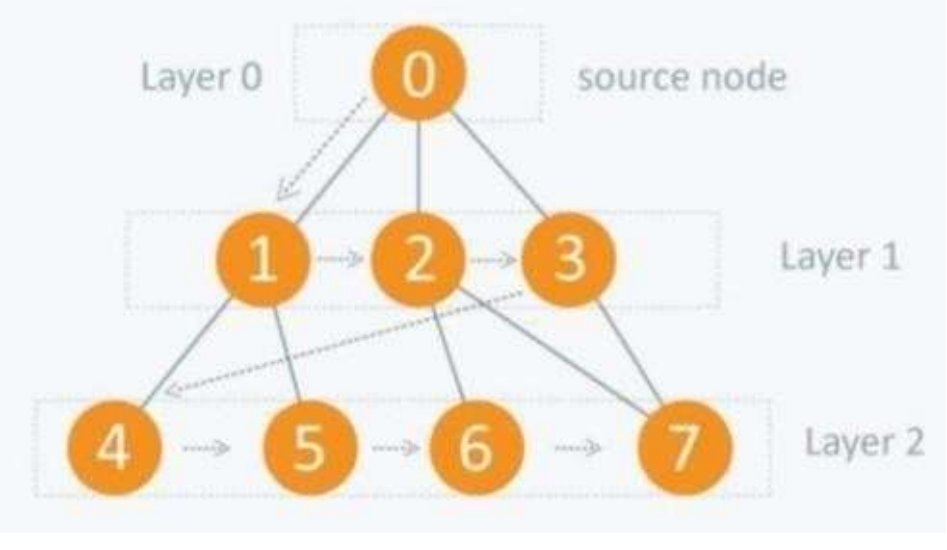
Breadth First Search (BFS): There are many ways to traverse graphs. BFS is the most commonly used approach.

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbor nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbor nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

Consider the following diagram.



Parallel Breadth First Search

1. To design and implement parallel breadth first search, you will need to divide the graph into smaller sub-graphs and assign each sub-graph to a different processor or thread.
2. Each processor or thread will then perform a breadth first search on its assigned sub-graph concurrently with the other processors or threads.
3. Two methods: Vertex by Vertex OR Level By Level

Sample Program:

```
#include <iostream> #include <vector> #include <queue> #include <omp.h>using
namespace std;
void bfs(vector<vector<int>>& graph, int start, vector<bool>& visited) {#pragma
omp task first private(vertex)
{
for (int neighbor : graph[vertex]) {
if (!visited[neighbor]) { q.push(neighbor); visited[neighbor] = true; #pragma omp task bfs
(graph,neighbor,visited);
}}}}
}
void parallel_bfs(vector<vector<int>>& graph, int start) {
 vector<bool> visited(graph.size(), false);
bfs(graph, start, visited);
```

Parallel Depth First Search:

- Different subtrees can be searched concurrently.
- Subtrees can be very different in size.
- Estimate the size of a subtree rooted at a node.
- Dynamic load balancing is required.

Parameters in Parallel DFS: Work Splitting

- Work is split by splitting the stack into two.
- Ideally, we do not want either of the split pieces to be small.
- Select nodes near the bottom of the stack (node splitting), or
- Select some nodes from each level (stack splitting)
- The second strategy generally yields a more even split of the space.

Sample Program

```
#include <iostream>
#include <vector> #include <stack> #include <omp.h>using
namespace std;
void dfs(vector<vector<int>>& graph, int start,
vector<bool>& visited) { stack<int> s; s.push(start); visited[start] = true;#pragma
omp parallel{
#pragma omp single{
while (!s.empty()) { int vertex = s.top(); s.pop();#pragma
```

```
omp task first private(vertex){
for (int neighbor : graph[vertex]) {
if (!visited[neighbor]) { s.push(neighbor); visited[neighbor] = true; #pragma omp task dfs(graph,neighbor,
visited);
}}}}}
}
void parallel_dfs(vector<vector<int>>& graph, int start) { vector<bool> visited(graph.size(),false);
dfs(graph, start, visited);
```

OpenMP Section Compiler Directive:

A parallel loop is an example of independent work units that are numbered. If you have a pre- determined number of independent work units, the sections is more appropriate. In a sections construct can be any number of section constructs. These need to be independent, and they can be execute by any available thread in the current team, including having multiple sections done by the same thread.

The sections construct is a non-iterative work sharing construct that contains a set of structured blocks that are to be distributed among and executed by the threads in a team. Each structured block is executed once by one of the threads in the team in the context of its implicit task.

Execute different code blocks in parallel

```
#pragma omp sections
{ #pragma omp section { ... }
    ...
   #pragma omp section { .. }
}
```

OpenMP Critical Compiler Directive

The omp critical directive identifies a section of code that must be executed by a single thread at a time..

```
#pragma omp critical
{
  code_block
}
```

Conclusion: Thus, we have successfully implemented parallel algorithms for Binary Search and Breadth First Search.

Assignment No.2

Aim: 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: Student will be able to learn

1. The Basic Concepts of Bubble Sort and Merge Sort.
2. Multiple Compiler Directives, library routines, environment variables availablefor OpenMP.

Theory:

Parallel Sorting:

A sequential sorting algorithm may not be efficient enough when we have to sort a huge volume of data. Therefore, parallel algorithms are used in sorting.

Design methodology:

Based on an existing sequential sort algorithm

- Try to utilize all resources available
- Possible to turn a poor sequential algorithm into a reasonable parallel algorithm

Bubble Sort

The idea of bubble sort is to compare two adjacent elements. If they are not in the right order, switch them. Do this comparing and switching (if necessary) until the end of the array is reached.Repeat this process from the beginning of the array n times. Average performance is $O(n^2)$

Bubble Sort Example

Here we want to sort an array containing [8, 5, 1].

8, 5, 1	Switch 8 and 5
5, 8, 1	Switch 8 and 1
5, 1, 8	Reached end start again
5, 1, 8	Switch 5 and 1
1, 5, 8	No Switch for 5 and 8
1, 5, 8	Reached end start again
1, 5, 8	No switch for 1, 5
1, 5, 8	No switch for 5, 8
1, 5, 8	Reached end.

But do not start again since this is the n^{th} iteration of same process

Parallel Bubble Sort

• Implemented as a pipeline.

• Let local_size = n / no_proc. We divide the array in no_proc parts, and each process executes the bubble

sort on its part, including comparing the last element with the first one belonging to the next thread.

• Implement with the loop (instead of j<i) for (j=0; j<n-1; j++)

• For every iteration of i, each thread needs to wait until the previous thread has finished that iteration before starting.

• We'll coordinate using a barrier.

Algorithm for Parallel Bubble Sort

1. For k = 0 to n-2
2. If k is even then
3. for i = 0 to (n/2)-1 do in parallel
4. If A[2i] > A[2i+1] then
5. Exchange A[2i] ↔ A[2i+1]
6. Else
7. for i = 0 to (n/2)-2 do in parallel8. If A[2i+1] > A[2i+2] then
9. Exchange A[2i+1] ↔ A[2i+2]
10. Next k

Parallel Bubble Sort Example

• Compare all pairs in the list in parallel

• Alternate between odd and even phases

• Shared flag, sorted, initialized to true at beginning of each iteration (2 phases), if any processor perform swap, sorted = false

Merge Sort

- Collects sorted list onto one processor
- Merges elements as they come together
- Simple tree structure
- Parallelism is limited when near the root

Steps of Merge Sort:

To sort A[p .. r]:

1. Divide Step

If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split A[p .. r] into two subarrays A[p .. q] and A[q + 1 .. r], each containing about half of the elements of A[p .. r]. That is, q is the halfway point of A[p .. r].

2. Conquer Step

Conquer by recursively sorting the two subarrays A[p .. q] and A[q + 1 .. r].

3. Combine Step

Combine the elements back in A[p .. r] by merging the two sorted subarraysA[p .. q] and A[q + 1 .. r] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

Example:

Parallel Merge Sort

- Parallelize processing of sub-problems
- Max parallelization achived with one processor per node (at each layer/height)

Parallel Merge Sort Example

- Perform Merge Sort on the following list of elements. Given 2 processors, P0 & P1.
- 4,3,2,1

Algorithm for Parallel Merge Sort

1. Procedure parallelMergeSort
2. Begin
3. Create processors Pi where i = 1 to n
4. if i > 0 then recieve size and parent from the root
5. recieve the list, size and parent from the root
6. endif
7. midvalue= listsize/2
8. if both children is present in the tree then
9. send midvalue, first child
10. send listsize-mid,second child
11. send list, midvalue, first child
12. send list from midvalue, listsize-midvalue, second child
13. call mergelist(list,0,midvalue,list, midvalue+1,listsize,temp,0,listsize)
14. store temp in another array list2
15. else
16. call parallelMergeSort(list,0,listsize)
17. endif
18. if i >0 then
19. send list, listsize,parent
20. endif
21. end

ALGORITHM ANALYSIS

1. Time Complexity Of parallel Merge Sort and parallel Bubble sort in best case is(when all datais already in sorted form):**O(n)**

2. Time Complexity Of parallel Merge Sort and parallel Bubble sort in worst case is: **O(n logn)**

3. Time Complexity Of parallel Merge Sort and parallel Bubble sort in average case is: **O(nlogn)**

Conclusion: Thus, we have successfully implemented parallel algorithms for Bubble Sort andMerger Sort.

Assignment No.3

Aim: Implement Min, Max, Sum and Average operations using Parallel Reduction.

Objective: Student will be able to learn to Analyze and measure performance of sequential and parallel algorithms.

Theory:

OpenMP:
OpenMP is a set of C/C++ pragmas which provide the programmer a high-level front-end interface which get translated as calls to threads. The key phrase here is "higher-level"; the goal is to better enable the programmer to "think parallel" alleviating him/her of the burden and distraction of dealing with setting up and coordinating threads. For example, the OpenMP directive.

OpenMP Core Syntax:

Most of the constructs in OpenMP are compiler directives:

#pragma omp construct [clause [clause]...]

Example

#pragma omp parallel num_threads(4)

Function prototypes and types in the file:

#include
<omp.h>
Most OpenMP constructs apply to a “structured block”.

Structured block: a block of one or more statements surrounded by “{ }”, with one point of entry at the top and one point of exit at the bottom.

Following is the sample code which illustrates max operator usage in OpenMP :

```
#include <stdio.h> #include <omp.h>
 int main()
 {
Doublearr[10];     omp_set_num_threads(4);
  double max_val=0.0;
   int i;
for( i=0; i<10; i++)
arr[i] = 2.0 + i;
   #pragma omp parallel for reduction(max : max_val)
for( i=0;i<10; i++)
   {
printf("thread id = %d and i = %d", omp_get_thread_num(), i);
       if(arr[i] >max_val)
      {
max_val = arr[i];
      }
```

```
    }
  printf("\nmax_val = %f", max_val);
  }
```

Following is the sample code which illustrates min operator usage in OpenMP :

```
#include <stdio.h>
 #include <omp.h>
 int main()
 {
Double          arr[10];
omp_set_num_threads(4);                    double
min_val=0.0;
   int i;
for( i=0; i<10; i++)
arr[i] = 2.0 + i;
   #pragma omp parallel for reduction(min : min_val)
 for( i=0;i<10; i++)
  {
printf("thread id = %d and i = %d", omp_get_thread_num(), i);
    if(arr[i] <min_val)
    {
min_val = arr[i];
    }
  }
printf("\nmin_val = %f", min_val);
}
```

Following is the sample code which illustrates sum operation usage in OpenMP :

```
 #include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
int   i, n;
float a[100], b[100], sum;
 /* Some initializations */
 n = 100;
 for (i=0; i< n; i++)
  a[i] = b[i] = i * 1.0;
 sum = 0.0;
#pragma omp parallel for reduction(+:sum)
  for (i=0; i< n; i++)
  sum = sum + (a[i] * b[i]);
printf("   Sum = %f\n",sum);
 }
```

Following is the sample code which illustrates average operation usage in OpenMP

```
#include<iostream>
#include<omp.h>//header file for openmp
using namespace std;
main()
{
        int arr[5]={1,2,3,4,5},i;
        float avg=0;
        #pragma omp parallel
        {
                int id=omp_get_thread_num();//id will tell us which thread is running the addition
                #pragma omp for //used for running the loop parallelly
                for(i=0;i<5;i++)
                {
                        avg+=arr[i];//summation
                        cout<<"For i= "<<i<<" thread "<<id<<" is executing"<<endl;
                }
        }
        avg/=5;
        cout<<"Output "<<avg<<endl;
}
```

Conclusion: Thus, we have successfully implemented parallel reduction using Min, Max, Sum and Average Operations.

Assignment No.4

Aim:

Write a CUDA Program for:

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

Objective: Students will learn fundamentals of GPU Computing in the CUDA environment.

Theory:

CUDA:

CUDA programming is especially well-suited to address problems that can be expressed as data- parallel computations. Any applications that process large data sets can use a data-parallel model to speed up the computations. Data-parallel processing maps data elements to parallel threads.

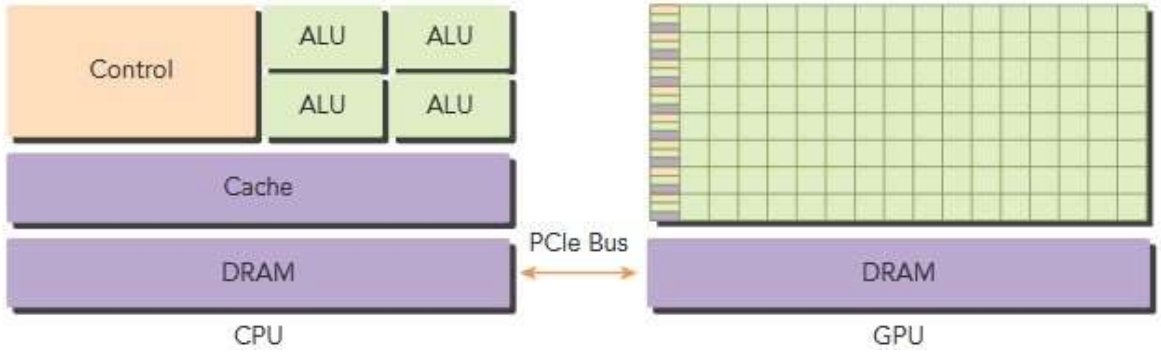
The first step in designing a data parallel program is to partition data across threads, with each thread working on a portion of the data. The first step in designing a data parallel program is to partition data across threads, with each thread working on a portion of the data.

CUDA Architecture:

A heterogeneous application consists of two parts:

- Host code
- Device code

Host code runs on CPUs and device code runs on GPUs. An application executing on a heterogeneous platform is typically initialized by the CPU. The CPU code is responsible for managing the environment, code, and data for the device before loading compute-intensive tasks on the device. With computational intensive applications, program sections often exhibit a rich amount of data parallelism. GPUs are used to accelerate the execution of this portion of data parallelism. When a hardware component that is physically separate from the CPU is used to accelerate computationally intensive sections of an application, it is referred to as a hardware accelerator. GPUs are arguably the most common example of a hardware accelerator. GPUs must operate in conjunction with a CPU-based host through a PCI-Express bus, as shown in Figure.



NVIDIA's CUDA nvcc compiler separates the device code from the host code during the compilation process. The

device code is written using CUDA C extended with keywords for labeling data-parallel functions, called kernels. The device code is further compiled by Nvcc. During the link stage, CUDA runtime libraries are added for kernel procedure calls and explicit GPU device manipulation.

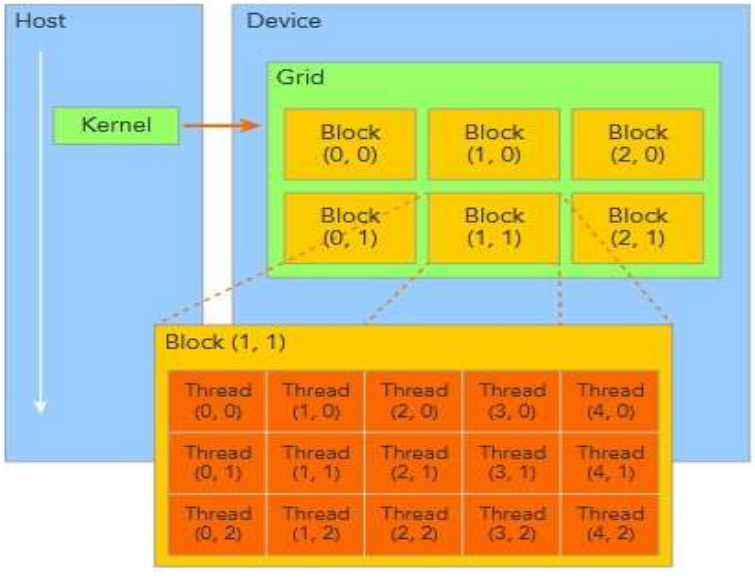
A kernel is executed by an array of threads and all threads run the same code. The parameters within the triple angle brackets are the execution configuration, which specifies how many threads will execute the kernel. In this example, you will run 10 GPU threads. s

A typical processing flow of a CUDA program follows this pattern:
1. Copy data from CPU memory to GPU memory.
2. Invoke kernels to operate on the data stored in GPU memory.
3. Copy data back from GPU memory to CPU memory

Table lists the standard C functions and their corresponding CUDA C functions for memory operations. Host and Device Memory Functions are follows.

STANDARD C FUNCTIONS	CUDA C FUNCTIONS
malloc	cudaMalloc
memcpy	cudaMemcpy
memset	cudaMemset
free	cudaFree

Organizing Threads:
When a kernel function is launched from the host side, execution is moved to a device where a large number of threads are generated and each thread executes the statements specified by the kernel function. The two-level thread hierarchy decomposed into blocks of threads and grids of blocks are shown in following figure:



All threads spawned by a single kernel launch are collectively called a grid. All threads in a grid share the same global memory space. A grid is made up of many thread blocks. A thread block is a group of threads that can

cooperate with each other. Threads from different blocks cannot cooperate.

Threads rely on the following two unique coordinates to distinguish themselves from each other:

- blockIdx (block index within a grid)
- threadIdx (thread index within a block)

These variables appear as built-in, pre-initialized variables that can be accessed within kernel functions. When a kernel function is executed, the coordinate variables blockIdx and
threadIdx are assigned to each thread by the CUDA runtime. Based on the coordinates, you can assign portions of data to different threads. It is a structure containing three unsigned integers, and the 1st, 2nd, and 3rd components are accessible through the fields x, y, and z respectively.

blockIdx.x
blockIdx.y
blockIdx.z

threadIdx.x
threadIdx.y
threadIdx.z

CUDA organizes grids and blocks in three dimensions. The dimensions of a grid and a block are specified by the following two built-in variables:

- blockDim (block dimension, measured in threads)
- gridDim (grid dimension, measured in blocks)

These variables are of type dim3, that is used to specify dimensions. When defining a variable of type dim3, any component left unspecified is initialized to 1. Each component in a variable of type dim3 is accessible through its x, y and z fields, respectively, as shown in the following example:

blockDim.x
blockDim.y
blockDim.

Conclusion: We have successfully implemented CUDA program for calculating Matrix Operations.

Assignment No.5

Aim: 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: Students will learn

1. The concept of linear regression.
2. The concept of deep neural network.

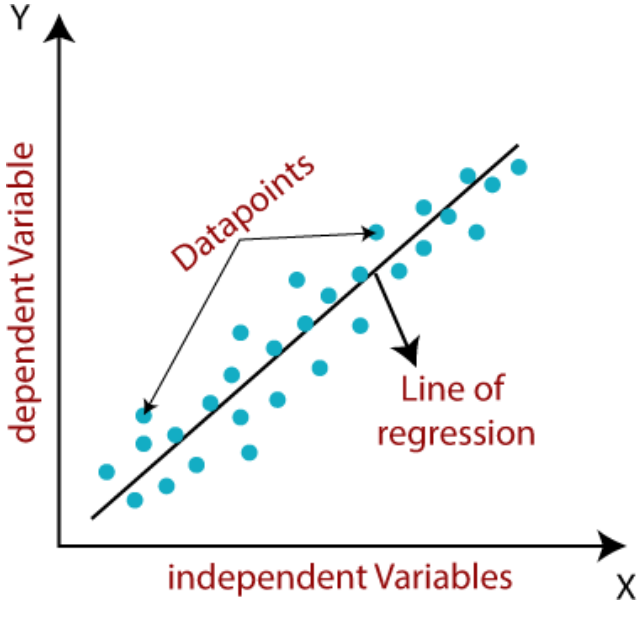
Theory:

Linear Regression:

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$y= a_0+a_1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

ε = random error

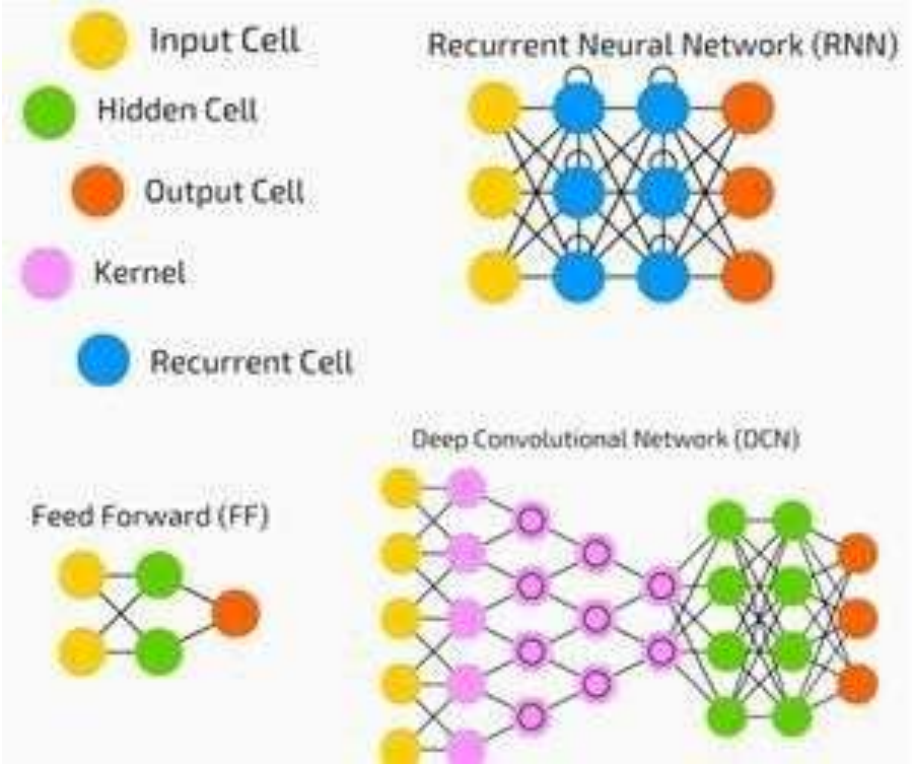
The values for x and y variables are training datasets for Linear Regression model representation.

Deep Neural Network:

A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs can model complex non-linear relationships.

The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification. We restrict ourselves to feed forward neural networks.

We have an input, an output, and a flow of sequential data in a deep network.



Neural networks are widely used in supervised learning and reinforcement learning problems. These networks are based on a set of layers connected to each other.

In deep learning, the number of hidden layers, mostly non-linear, can be large; say about 1000 layers.

DL models produce much better results than normal ML networks.

We mostly use the gradient descent method for optimizing the network and minimising the loss function.

We can use the **Imagenet**, a repository of millions of digital images to classify a dataset into categories like cats and dogs. DL nets are increasingly used for dynamic images apart from static ones and for time series and text analysis.

Training the data sets forms an important part of Deep Learning models. In addition, Backpropagation is the main algorithm in training DL models.

DL deals with training large neural networks with complex input output transformations.

One example of DL is the mapping of a photo to the name of the person(s) in photo as they do on social networks and describing a picture with a phrase is another recent application of DL.

Conclusion: We have studied and implemented Boston housing price prediction problem by linear regression using deep neural network.

Assignment No.6

Aim: Classification using Deep neural network: 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: Student will learn:

1. The classification using deep neural network.
2. The concept of binary classification.

Theory:

Binary Classification:

Binary classification is one of the most common and frequently tackled problems in the machine learning domain. In its simplest form the user tries to classify an entity into one of the two possible categories. For example, give the attributes of the fruits like weight, color, peel texture, etc. that classify the fruits as either peach or apple. Through the effective use of Neural Networks (Deep Learning Models), binary classification problems can be solved to a fairly high degree.

Binary classification refers to those classification tasks that have two class labels.

Examples include:

1. Email spam detection (spam or not).
2. Churn prediction (churn or not).
3. Conversion prediction (buy or not).

Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.

For example "*not spam*" is the normal state and "*spam*" is the abnormal state. Another example is "*cancer not detected*" is the normal state of a task that involves a medical test and "*cancer detected*" is the abnormal state.

The class for the normal state is assigned the class label 0 and the class with the abnormal state is assigned the class label 1.

Popular algorithms that can be used for binary classification include:

1. Logistic Regression
2. k-Nearest Neighbors
3. Decision Trees
4. Support Vector Machine
5. Naive Bayes

If the model successfully predicts the patients as positive, this case is called *True Positive (TP)*. If the model successfully predicts patients as negative, this is called *True Negative (TN)*. The binary classifier may misdiagnose some patients as well. If a diseased patient is classified as healthy by a negative test

result, this error is called *False Negative (FN)*. Similarly, If a healthy patient is classified as diseased by a positive test result, this error is called *False Positive(FP)*.

We can evaluate a binary classifier based on the following parameters:

i. True Positive (TP): The patient is diseased and the model predicts "diseased"
ii. False Positive (FP): The patient is healthy but the model predicts "diseased"
iii. True Negative (TN): The patient is healthy and the model predicts "healthy"
iv. False Negative (FN): The patient is diseased and the model predicts "healthy"

Conclusion: We have studied and implemented the binary classification using deep neural network.

Assignment No.7

Aim: Convolutional neural network (CNN): Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Objective: Student will learn:

1. The concept of convolutional neural network.
2. The working of convolutional neural network.

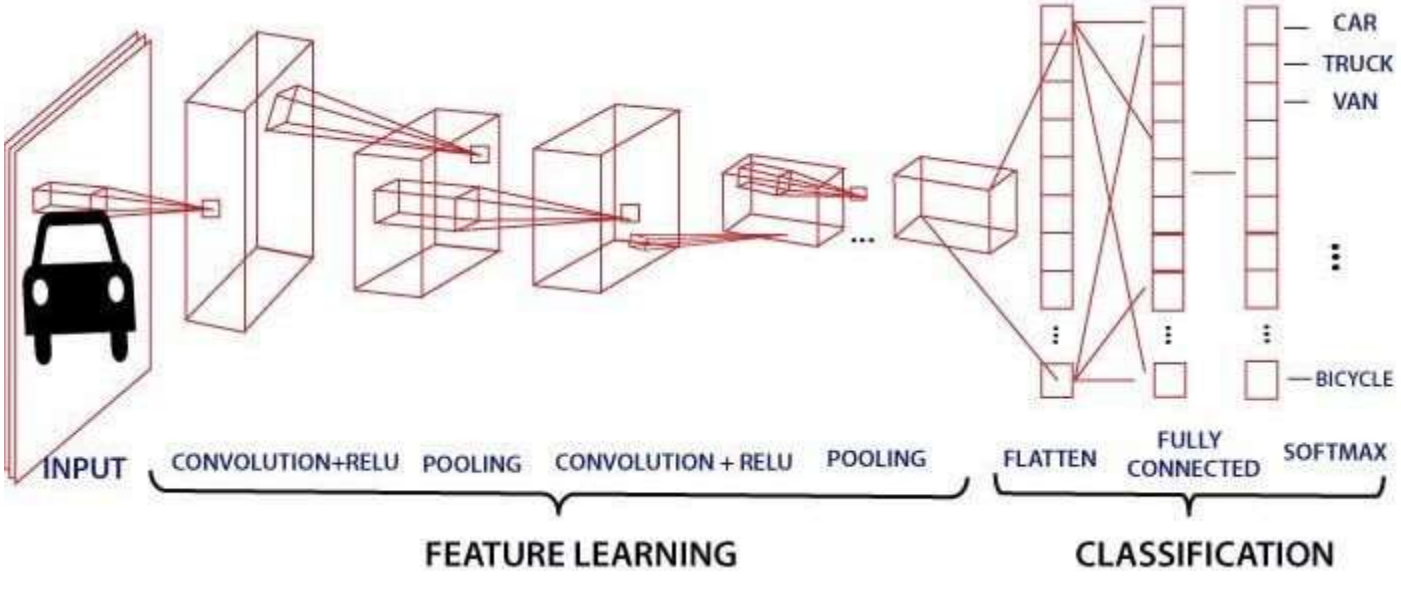
Theory:

Convolutional neural network (CNN):

Convolutional Neural Network is one of the main categories to do image classification and image recognition in neural networks. Scene labeling, objects detections, and face recognition, etc., are some of the areas where convolutional neural networks are widely used.

CNN takes an image as input, which is classified and process under a certain category such as dog, cat, lion, tiger, etc. The computer sees an image as an array of pixels and depends on the resolution of the image. Based on image resolution, it will see as **h * w * d**, where h= height w= width and d= dimension. For example, An RGB image is **6 * 6 * 3** array of the matrix, and the grayscale image is **4 * 4 * 1** array of the matrix.

In CNN, each input image will pass through a sequence of convolution layers along with pooling, fully connected layers, filters (Also known as kernels). After that, we will apply the Soft-max function to classify an object with probabilistic values 0 and 1.



Convolution Layer:

Convolution layer is the first layer to extract features from an input image. By learning image features using a small square of input data, the convolutional layer preserves the relationship between pixels. It is a mathematical operation which takes two inputs such as image matrix and a kernel or filter.

- The dimension of the image matrix is **h×w×d**.
- The dimension of the filter is $\mathbf{f_h \times f_w \times d}$.
- The dimension of the output is $\mathbf{(h-f_h+1)\times(w-f_w+1)\times 1}$.

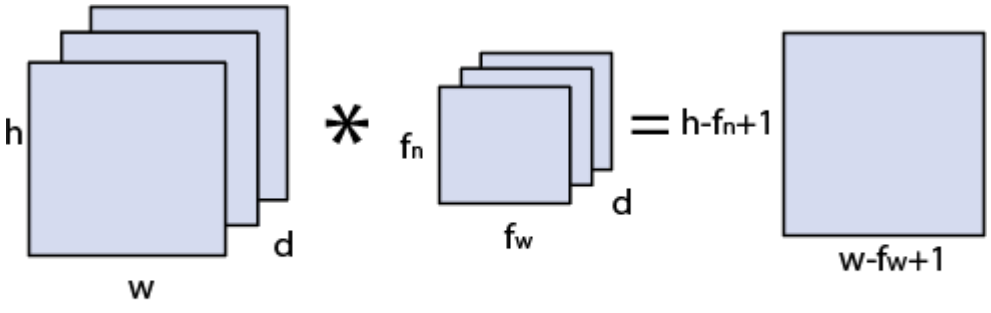


Image matrix multiplies kernl or filter matrix

Let's start with consideration a 5*5 image whose pixel values are 0, 1, and filter matrix 3*3 as:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

5×5 – Image Matrix 3×3 – Filter Matrix

The convolution of 5*5 image matrix multiplies with 3*3 filter matrix is called "**Features Map**" and show as an output.

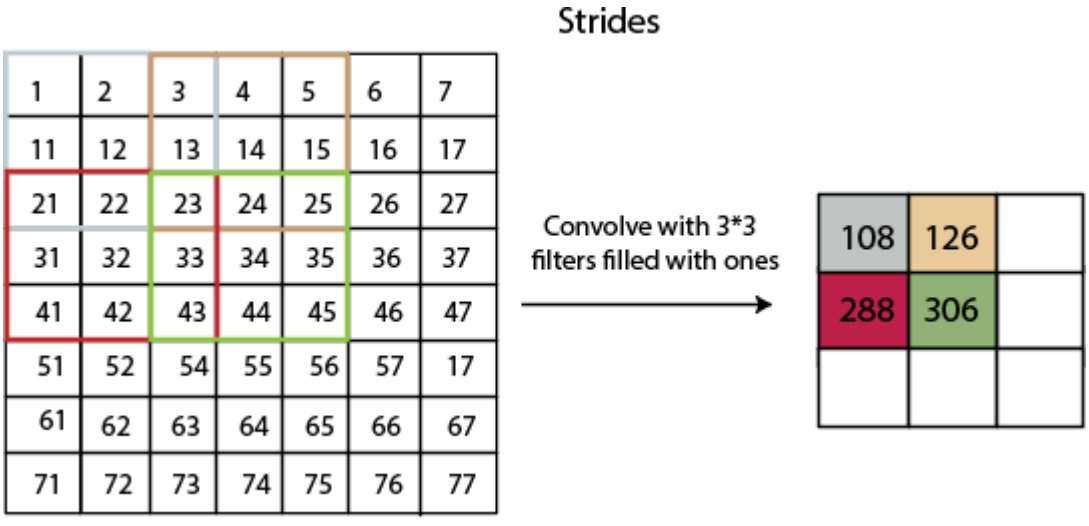
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

Convolved Feature

Convolution of an image with different filters can perform an operation such as blur, sharpen, and edge detection by applying filters.

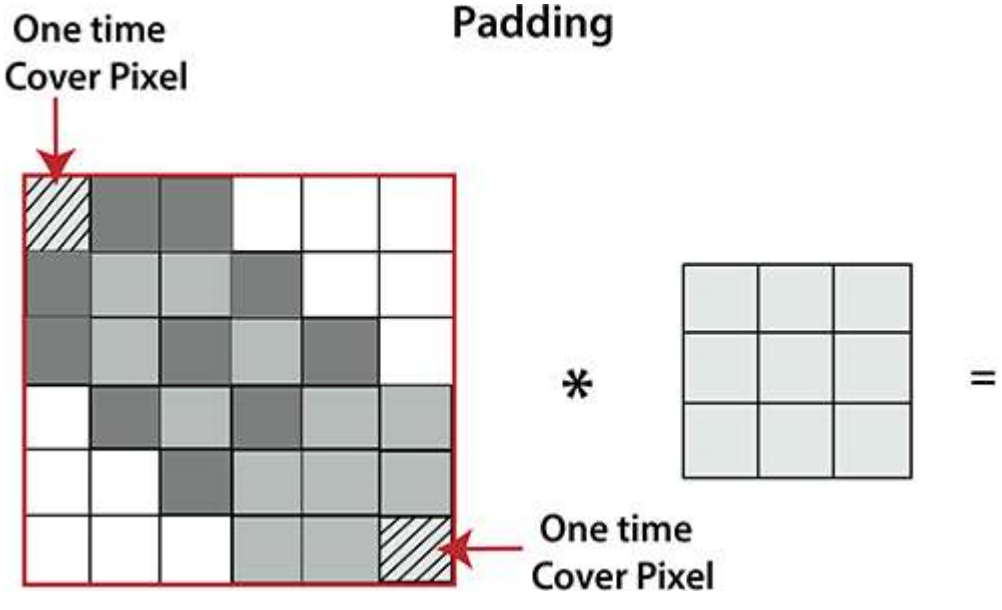
Strides:

Stride is the number of pixels which are shift over the input matrix. When the stride is equaled to 1, then we move the filters to 1 pixel at a time and similarly, if the stride is equaled to 2, then we move the filters to 2 pixels at a time. The following figure shows that the convolution would work with a stride of 2.



Padding:

Padding plays a crucial role in building the convolutional neural network. If the image will get shrink and if we will take a neural network with 100's of layers on it, it will give us a small image after filtered in the end. If we take a three by three filter on top of a grayscale image and do the convolving then what will happen?



It is clear from the above picture that the pixel in the corner will only get covers one time, but the middle pixel will get covered more than once. It means that we have more information on that middle pixel, so there are two downsides:

- Shrinking outputs
- Losing information on the corner of the image.

To overcome this, we have introduced padding to an image. **"Padding is an additional layer which can add to the border of an image."**

Pooling Layer:

Pooling layer plays an important role in pre-processing of an image. Pooling layer reduces the number of parameters when the images are too large. Pooling is "**downscaling**" of the image obtained from the previous

layers. It can be compared to shrinking an image to reduce its pixel density. Spatial pooling is also called downsampling or subsampling, which reduces the dimensionality of each map but retains the important information. There are the following types of spatial pooling:

Max Pooling:

Max pooling is a **sample-based discretization process**. Its main objective is to downscale an input representation, reducing its dimensionality and allowing for the assumption to be made about features contained in the sub-region binned.

Max pooling is done by applying a max filter to non-overlapping sub-regions of the initial representation.

Average Pooling:

Down-scaling will perform through average pooling by dividing the input into rectangular pooling regions and computing the average values of each region.

Syntax

layer = averagePooling2dLayer(poolSize)

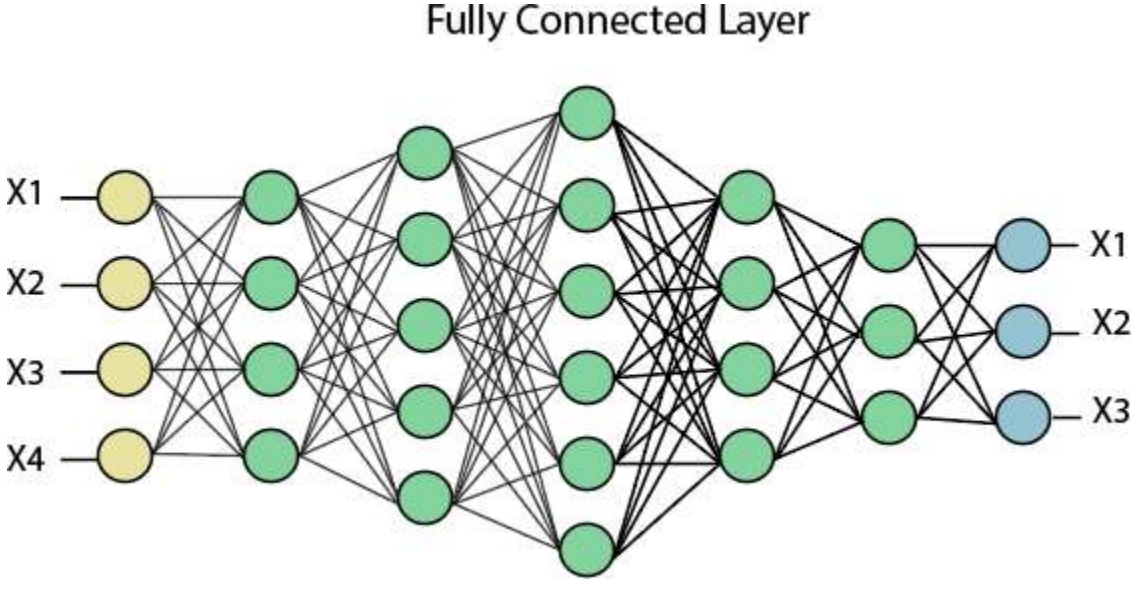
layer = averagePooling2dLayer(poolSize,Name,Value)

Sum Pooling:

The sub-region for **sum pooling** or **mean pooling** are set exactly the same as for **max-pooling** but instead of using the max function we use sum or mean.

Fully Connected Layer:

The fully connected layer is a layer in which the input from the other layers will be flattened into a vector and sent. It will transform the output into the desired number of classes by the network.



In the above diagram, the feature map matrix will be converted into the vector such as **x1, x2, x3... xn** with the help of fully connected layers. We will combine features to create a model and apply the activation function such as **softmax** or **sigmoid** to classify the outputs as a car, dog, truck, etc.

Conclusion: We have studied the concept of Convolutional Neural Network (CNN) and created a classifier to classify fashion clothing into categories.

Assignment No.8

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

Objective: Student will learn:

1. The concept of Recurrent Neural Network (RNN)

Theory:

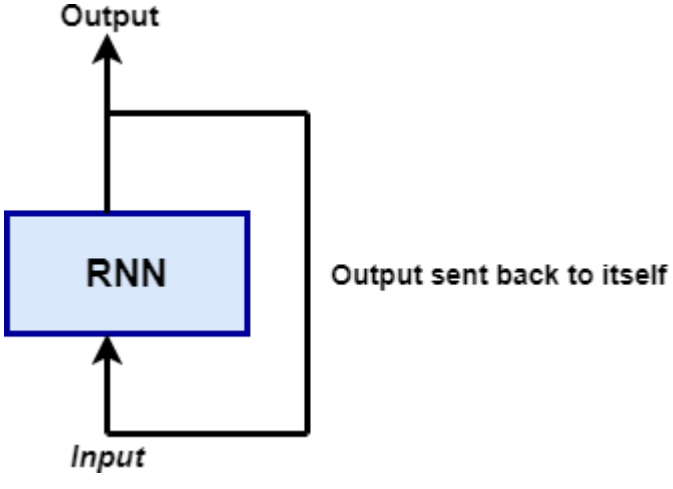
Recurrent neural network (RNN):

A recurrent neural network (**RNN**) is a kind of artificial neural network mainly used in **speech recognition** and **natural language processing** (NLP). RNN is used in deep learning and in the development of models that imitate the activity of neurons in the human **brain**.

Recurrent Networks are designed to **recognize patterns** in sequences of data, such as **text, genomes, handwriting, the spoken word,** and **numerical** time series data emanating from sensors, stock markets, and government agencies.

A recurrent neural network looks similar to a traditional neural network except that a memory-state is added to the neurons. The computation is to include a simple memory.

The recurrent neural network is a type of deep learning-oriented algorithm, which follows a sequential approach. In neural networks, we always assume that each input and output is dependent on all other layers. These types of neural networks are called recurrent because they sequentially perform mathematical computations.



Application of RNN:

RNN has multiple uses when it comes to predicting the future. In the financial industry, RNN can help predict stock prices or the sign of the stock market direction (i.e., **positive** or **negative**).

RNN is used for an autonomous car as it can avoid a car accident by anticipating the route of the vehicle.

RNN is widely used in **image captioning, text analysis, machine translation,** and **sentiment analysis**. **For example,** one should use a movie review to understanding the feeling the spectator perceived after **watching**

the movie. Automating this task is very useful when the movie company can not have more time to review, consolidate, label, and analyze the reviews. The machine can do the job with a higher level of accuracy.

Limitations of RNN:

RNN is supposed to carry the information in time. However, it is quite challenging to propagate all this information when the time step is too long. When a network has too many deep layers, it becomes untrainable. This problem is called: vanishing gradient problem.

If we remember, the neural network updates the weight use of the gradient descent algorithm. The gradient grows smaller when the network progress down to lower layers.

The gradient stays constant, meaning there is no space for improvement. The model learns from a change in its gradient; this change affects the network's output. If the difference in the gradient is too small (i.e., the weight change a little), the system can't learn anything and so the output. Therefore, a system facing a vanishing gradient problem cannot converge towards the right solution.

Training through RNN:

- The network takes a single time-step of the input.
- We can calculate the current state through the current input and the previous state.
- Now, the current state through h_{t-1} for the next state.
- There is n number of steps, and in the end, all the information can be joined.
- After completion of all the steps, the final step is for calculating the output.
- At last, we compute the error by calculating the difference between actual output and the predicted output.
- The error is backpropagated to the network to adjust the weights and produce a better outcome.

Conclusion: We have studied the concept of Recurrent Neural Network (RNN) and designed a time series analysis and prediction system.

Assignment No.9

Aim: Mini-project on High Performance Computing.

Assignment No.10

Aim: Mini-project on Deep Learning.