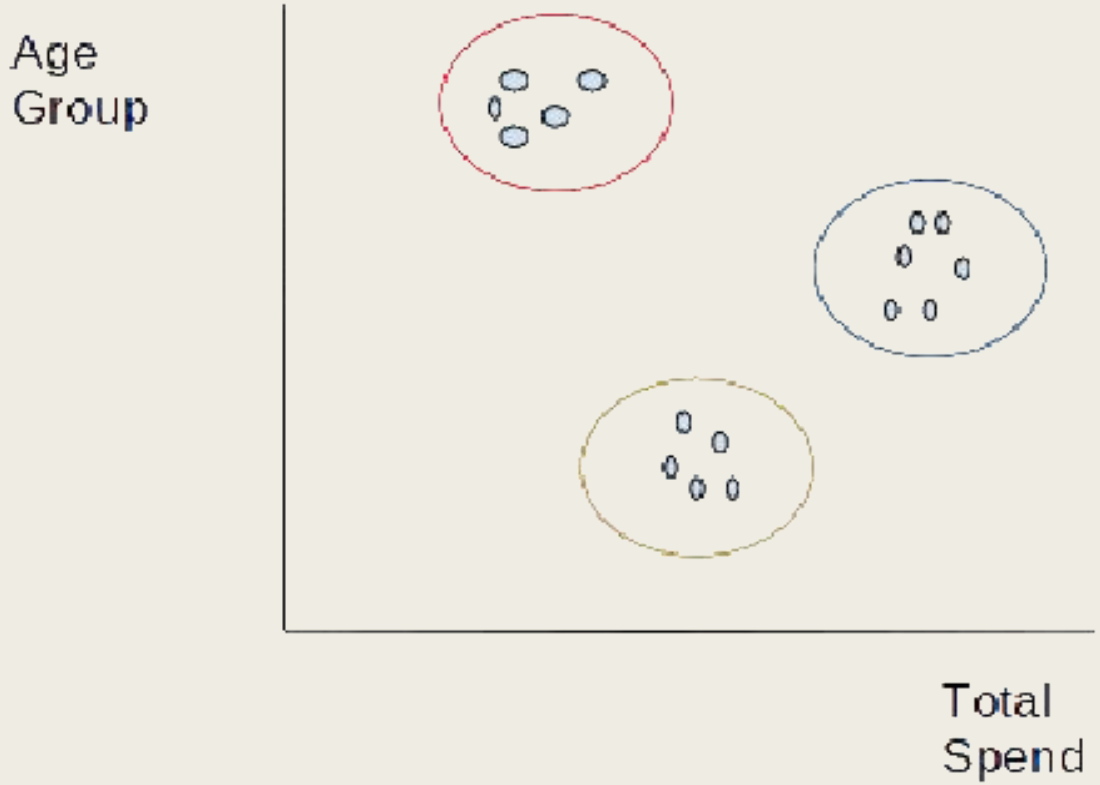
Clustering

- Clustering was introduced in 1932 by H.E. Driver and A.L.Kroeber in their paper on “Quantitative expression of cultural relationship”.
- Since then this technique has taken a big leap and has been used to discover the unknown in a number of application areas eg. Healthcare.
- Clustering is a type of **unsupervised learning where the references need to be drawn from unlabelled datasets.**
- Generally, it is used to capture meaningful structure, underlying processes, and grouping inherent in a dataset.
- In clustering, the task is to **divide the population into several groups in such a way that the data points in the same groups are more similar to each other than the data points in other groups.**
- In short, it is a collection of objects based on their similarities and dissimilarities.
- There are many clustering algorithms grouped into different cluster models. Before choosing any algorithm for a use case, it is important to get familiar with the **cluster models.**
- One more thing which should be considered while choosing any clustering algorithm is the **size of your dataset.**
- Datasets can contain millions of records and not all algorithms scale efficiently. **K-Means is one of the most popular algorithms** and it is also scale-efficient as it has a **complexity of O(n).**

Clustering Example

- Let's take an example, imagine you work in a Walmart Store as a manager and would like to better understand your customers to scale up your business by using new and improved marketing strategies.
- It is difficult to segment your customers manually. You have some data that contains their age and purchase history, here clustering can help to group customers based on their spending.
- Once the customer segmentation will be done, you can define different marketing strategies for each of the groups as per target audiences.



K - means Clustering

- K-means is a **centroid-based clustering algorithm**, where we calculate the **distance between each data point and a centroid to assign it to a cluster.**
- The goal is to identify the K number of groups in the dataset.
- It is an iterative process of assigning each data point to the groups and slowly data points get clustered based on similar features.
- The objective is to minimize the sum of distances between the data points and the cluster centroid, to identify the correct group each data point should belong to.
- Here, we **divide a data space into K clusters and assign a mean value to each.**
- The **data points are placed in the clusters closest to the mean value of that cluster.**
- There are several distance metrics available that can be used to calculate the distance but **euclidean distance is most commonly used.**

K - means Clustering Steps

- **1. Choosing the number of clusters :-** The first step is to define the K number of clusters in which we will group the data.
- **2. Initializing centroids :-** Centroid is the center of a cluster but initially, the exact center of data points will be unknown so, we select random data points and define them as centroids for each cluster.
- **3. Assign data points to the nearest cluster :-** Now that centroids are initialized, the next step is to assign data points X_n to their closest cluster centroid C_k. In this step, we will first calculate the distance between data point X and centroid C using Euclidean Distance metric.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- And then choose the cluster for data points where the distance between the data point and the centroid is minimum.
- **4. Re-initialize centroids :-** Next, we will re-initialize the centroids by calculating the average of all data points of that cluster.

$$Ci = \frac{1}{|Ni|} \sum xi$$

- **5. Repeat steps 3 and 4 :-** We will keep repeating steps 3 and 4 until we have optimal centroids and the assignments of data points to correct clusters are not changing anymore.

K - means Clustering Example Problem

- Use k-means clustering algorithm to divide the following data into two clusters and also compute the the representative data points for the clusters.

x1 1 2 2 3 4 5

x2 1 1 3 2 3 5

- Solution:-

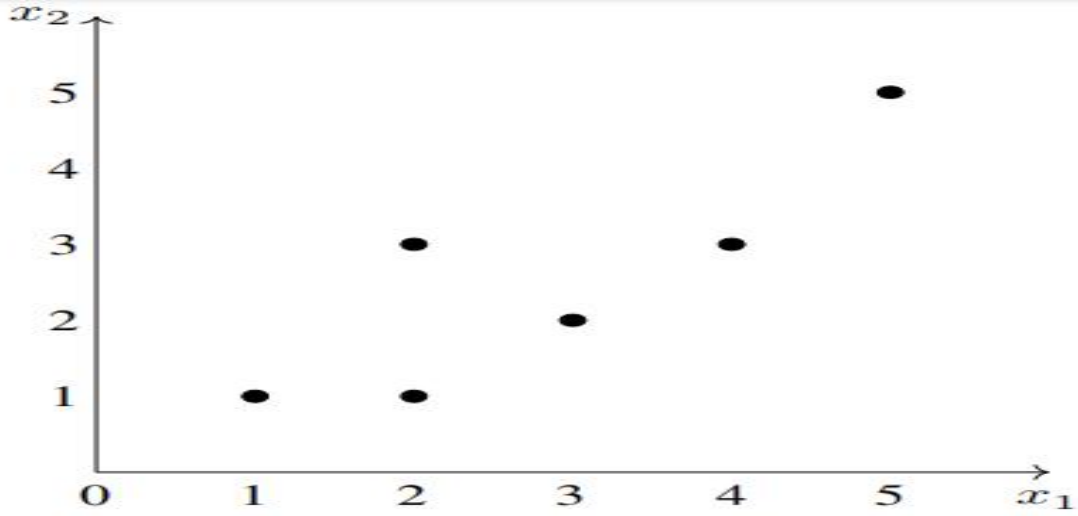


Figure 13.1: Scatter diagram of data in Table 13.1

1. In the problem, the required number of clusters is 2 and we take $k = 2$.
2. We choose two points arbitrarily as the initial cluster centres. Let us choose arbitrarily (see Figure 13.2)

$$\vec{v}_1 = (2, 1), \quad \vec{v}_2 = (2, 3).$$

3. We compute the distances of the given data points from the cluster centers.

K - means Clustering Example Problem

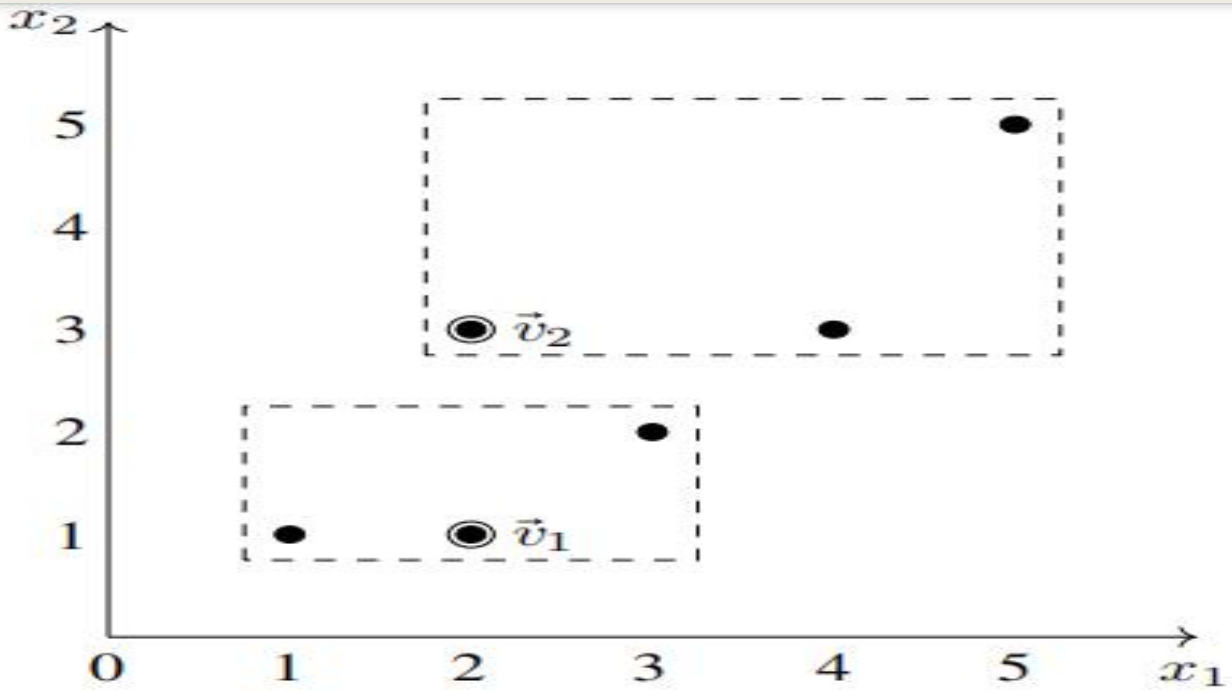


Figure 13.2: Initial choice of cluster centres and the resulting clusters

$\vec{x}_i$	Data point	Distance from $\vec{v}_1 = (2, 1)$	Distance from $\vec{v}_2 = (2, 3)$	Minimum distance	Assigned center
$\vec{x}_1$	$(1, 1)$	1	2.24	1	$\vec{v}_1$
$\vec{x}_2$	$(2, 1)$	0	2	0	$\vec{v}_1$
$\vec{x}_3$	$(2, 3)$	2	0	0	$\vec{v}_2$
$\vec{x}_4$	$(3, 2)$	1.41	1.41	1.41	$\vec{v}_1$
$\vec{x}_5$	$(4, 3)$	2.82	2	2	$\vec{v}_2$
$\vec{x}_6$	$(5, 5)$	5	3.61	3.61	$\vec{v}_2$

K - means Clustering Example Problem

(The distances of $\vec{x}_4$ from $\vec{v}_1$ and $\vec{v}_2$ are equal. We have assigned $\vec{v}_1$ to $\vec{x}_4$ arbitrarily.)

This divides the data into two clusters as follows (see Figure 13.2):

Cluster 1: $\{\vec{x}_1, \vec{x}_2, \vec{x}_4\}$ represented by $\vec{v}_1$

Number of data points in Cluster 1: $c_1 = 3$.

Cluster 2 : $\{\vec{x}_3, \vec{x}_5, \vec{x}_6\}$ represented by $\vec{v}_2$

Number of data points in Cluster 2: $c_2 = 3$.

4. The cluster centres are recalculated as follows:

$$\begin{aligned}
\vec{v}_1 &= \frac{1}{c_1}(\vec{x}_1 + \vec{x}_2 + \vec{x}_4) \\
&= \frac{1}{3}(\vec{x}_1 + \vec{x}_2 + \vec{x}_4) \\
&= (2.00, 1.33) \\
\vec{v}_2 &= \frac{1}{c_2}(\vec{x}_3 + \vec{x}_5 + \vec{x}_6) \\
&= \frac{1}{3}(\vec{x}_3 + \vec{x}_5 + \vec{x}_6) \\
&= (3.67, 3.67)
\end{aligned}$$

5. We compute the distances of the given data points from the new cluster centers.

K - means Clustering Example Problem

$\vec{x}_i$	Data point	Distance from $\vec{v}_1 = (2,1)$	Distance from $\vec{v}_2 = (2,3)$	Minimum distance	Assigned center
$\vec{x}_1$	$(1,1)$	1.05	3.77	1.05	$\vec{v}_1$
$\vec{x}_2$	$(2,1)$	0.33	3.14	0.33	$\vec{v}_1$
$\vec{x}_3$	$(2,3)$	1.67	1.80	1.67	$\vec{v}_1$
$\vec{x}_4$	$(3,2)$	1.20	1.80	1.20	$\vec{v}_1$
$\vec{x}_5$	$(4,3)$	2.60	0.75	0.75	$\vec{v}_2$
$\vec{x}_6$	$(5,5)$	4.74	1.89	1.89	$\vec{v}_2$

This divides the data into two clusters as follows (see Figure 13.4):

Cluster 1 : $\{\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4\}$ represented by $\vec{v}_1$

Number of data points in Cluster 1: $c_1 = 4$.

Cluster 2 : $\{\vec{x}_5, \vec{x}_6\}$ represented by $\vec{v}_2$

Number of data points in Cluster 1: $c_2 = 2$.

6. The cluster centres are recalculated as follows:

$$\vec{v}_1 == \frac{1}{c_1}(\vec{x}_1 + \vec{x}_2 + \vec{x}_3 + \vec{x}_4)$$
$$= \frac{1}{4}(\vec{x}_1 + \vec{x}_2 + \vec{x}_3 + \vec{x}_4)$$
$$= (2.00, 1.33)$$
$$\vec{v}_2 = \frac{1}{2}(\vec{x}_5 + \vec{x}_6) = (3.67, 3.67)$$

K - means Clustering Example Problem

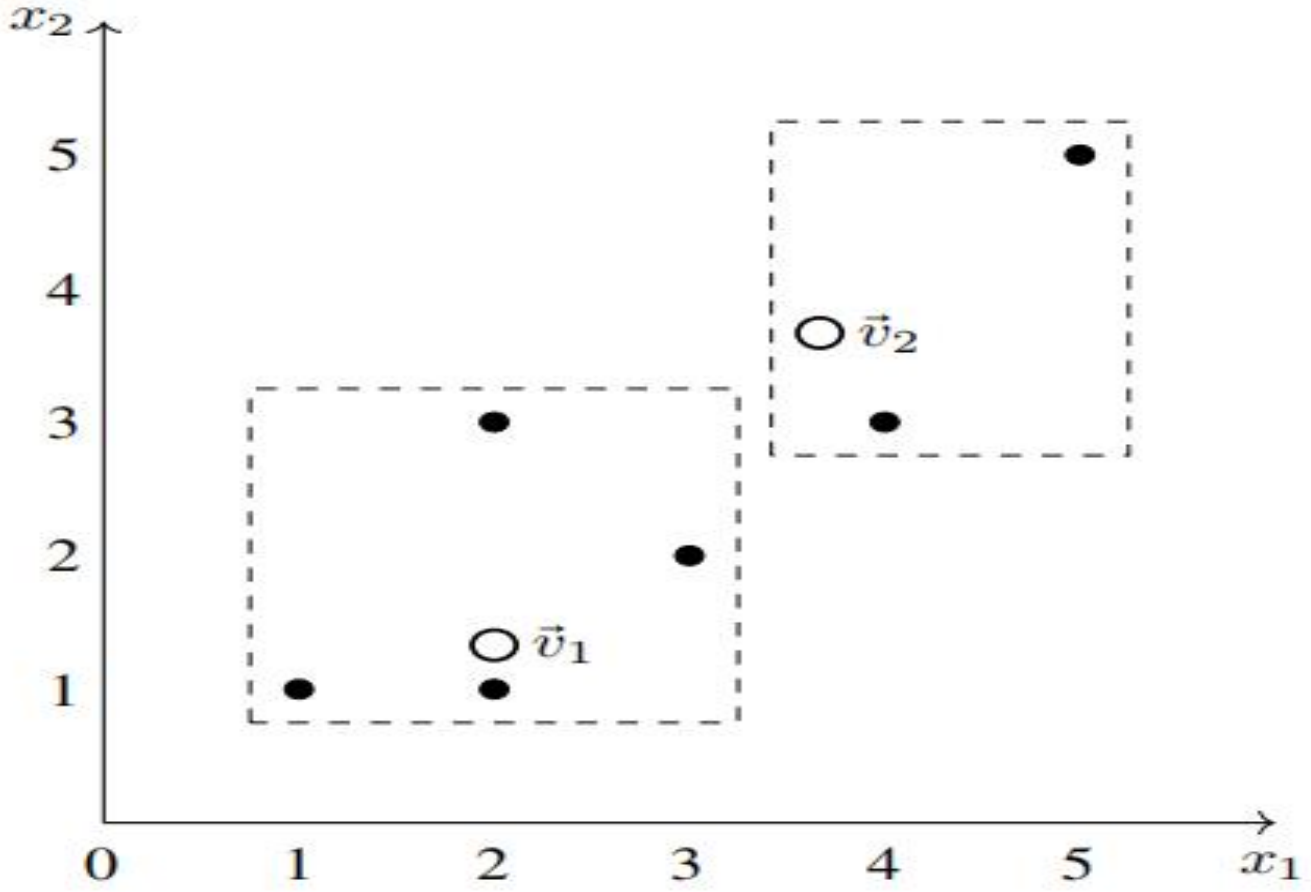


Figure 13.3: Cluster centres after first iteration and the corresponding clusters

7. We compute the distances of the given data points from the new cluster centers.
   4.609772 3.905125 2.692582 2.500000 1.118034 1.118034

K - means Clustering Example Problem

$\vec{x}_i$	Data point	Distance from $\vec{v}_1 = (2,1)$	Distance from $\vec{v}_2 = (2,3)$	Minimum distance	Assigned center
$\vec{x}_1$	$(1,1)$	1.25	4.61	1.25	$\vec{v}_1$
$\vec{x}_2$	$(2,1)$	0.75	3.91	0.75	$\vec{v}_1$
$\vec{x}_3$	$(2,3)$	1.25	2.69	1.25	$\vec{v}_1$
$\vec{x}_4$	$(3,2)$	1.03	2.50	1.03	$\vec{v}_1$
$\vec{x}_5$	$(4,3)$	2.36	1.12	1.12	$\vec{v}_2$
$\vec{x}_6$	$(5,5)$	4.42	1.12	1.12	$\vec{v}_2$

This divides the data into two clusters as follows (see Figure **??**):

Cluster 1 : $\{\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4\}$ represented by $\vec{v}_1$

Number of data points in Cluster 1: $c_1 = 4$.

Cluster 2 : $\{\vec{x}_5, \vec{x}_6\}$ represented by $\vec{v}_2$

Number of data points in Cluster 1: $c_1 = 2$.

8. The cluster centres are recalculated as follows:

$$\begin{aligned}
\vec{v}_1 &= \frac{1}{c_1}(\vec{x}_1 + \vec{x}_2 + \vec{x}_3 + \vec{x}_4) \\
&= \frac{1}{4}(\vec{x}_1 + \vec{x}_2 + \vec{x}_3 + \vec{x}_4) \\
&= (2.00, 1.75) \\
\vec{v}_2 &= \frac{1}{c_2}(\vec{x}_5 + \vec{x}_6) \\
&= \frac{1}{2}(\vec{x}_5 + \vec{x}_6) \\
&= (4.00, 4.50)
\end{aligned}$$

K - means Clustering Example Problem

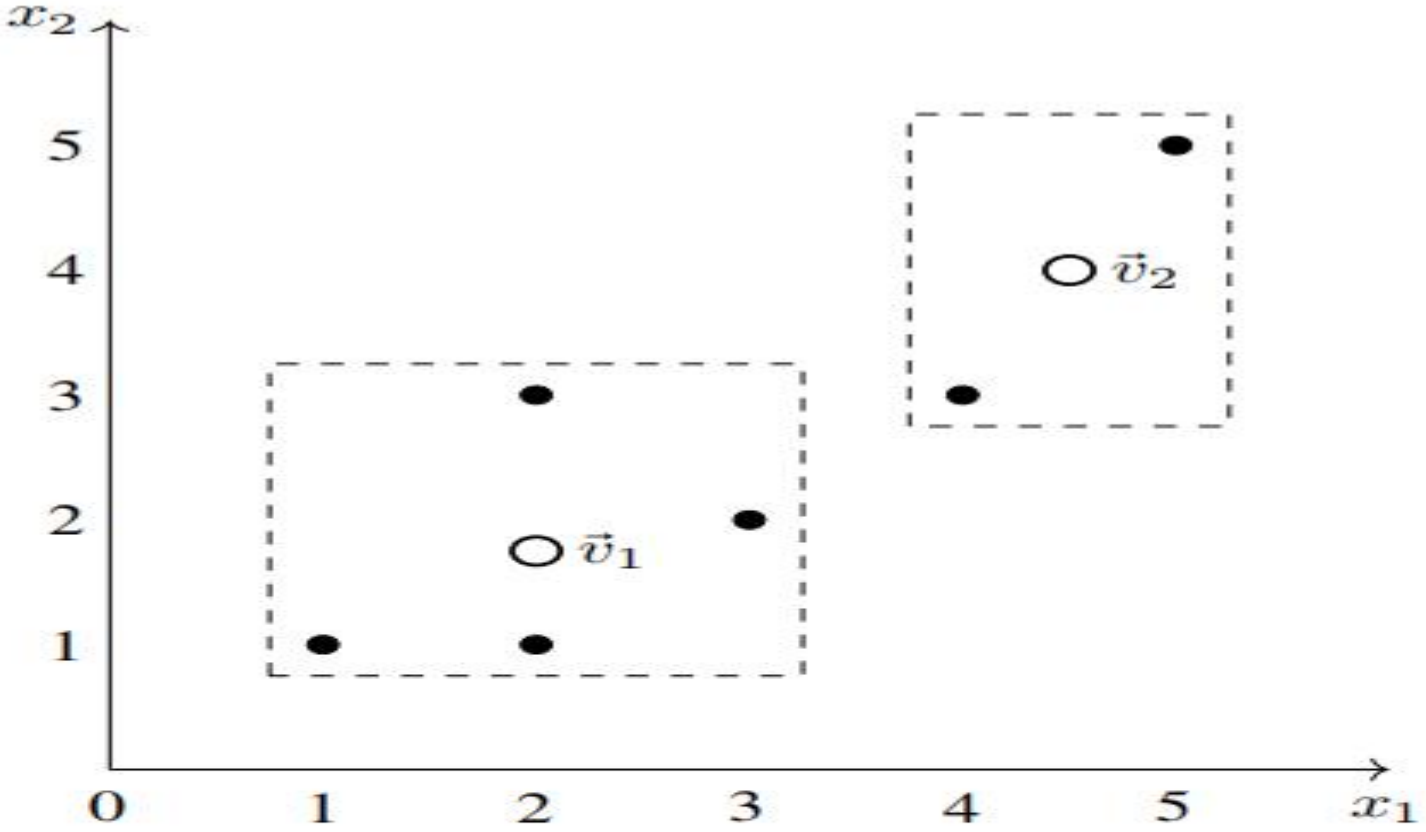


Figure 13.4: New cluster centres and the corresponding clusters

9. This divides the data into two clusters as follows (see Figure **??**):
   Cluster 1 : $\{\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4\}$ represented by $\vec{v}_1$
   Cluster 2 : $\{\vec{x}_5, \vec{x}_6\}$ represented by $\vec{v}_2$

K - means Clustering Example Problem

10. The cluster centres are recalculated as follows:

$$\vec{v}_1 = \frac{1}{4}(\vec{x}_1 + \vec{x}_2 + \vec{x}_3 + \vec{x}_4) = (2.00, 1.75)$$

$$\vec{v}_2 = \frac{1}{2}(\vec{x}_5 + \vec{x}_6) = (4.00, 4.50)$$

We note that these are identical to the cluster centres calculated in Step 8. So there will be no reassignment of data points to different clusters and hence the computations are stopped here.

11. Conclusion: The k means clustering algorithm with $k = 2$ applied to the dataset in Table 13.1 yields the following clusters and the associated cluster centres:

Cluster 1 : $\{\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4\}$ represented by $\vec{v}_1 = (2.00, 1.75)$

Cluster 2 : $\{\vec{x}_5, \vec{x}_6\}$ represented by $\vec{v}_2 = (2.00, 4.75)$

Disadvantages of K - means Clustering

- Even though the k-means algorithm is fast, robust and easy to understand, there are several disadvantages to the algorithm.
- The learning algorithm requires apriori specification of the number of cluster centers.
- The final cluster centres depend on the initial vi's.
- With different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).
- Euclidean distance measures can unequally weight underlying factors.
- Randomly choosing of the initial cluster centres may not lead to a fruitful result.

K - medians Clustering

- K-median clustering is similar to K-means. Only three differences are there:-
- 1. K-median clustering uses actual distance (which is also called the **L1 norm or the Manhattan or Taxicab distance**) to the center, instead of the square of the distance. So the distance formula is:-

$$d(p,q)=\sum_{i=1}^{n}|p_i-q_i|=\|.\|^1$$

- 2. K-median clustering choose the **median instead of the mean for the centers.**
- 3. K-median clustering need to optimize the following problem :-

$$argmin_C=\sum_{i=1}^{n}\sum_{i\in C_i}|x-median(C_i)|$$

Hierarchical clustering

- Hierarchical clustering is another **unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster** and also known as **hierarchical cluster analysis or HCA.**

- In this algorithm, we develop the hierarchy of clusters in the form of a **tree**, and this tree-shaped structure is known as the **dendrogram**.

- Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. **As there is no requirement to predetermine the number of clusters.**

- The hierarchical clustering technique has two approaches:

- **Agglomerative:** Agglomerative is a bottom-up approach, in which the **algorithm starts with taking all data points as single clusters and merging them until one cluster is left.**

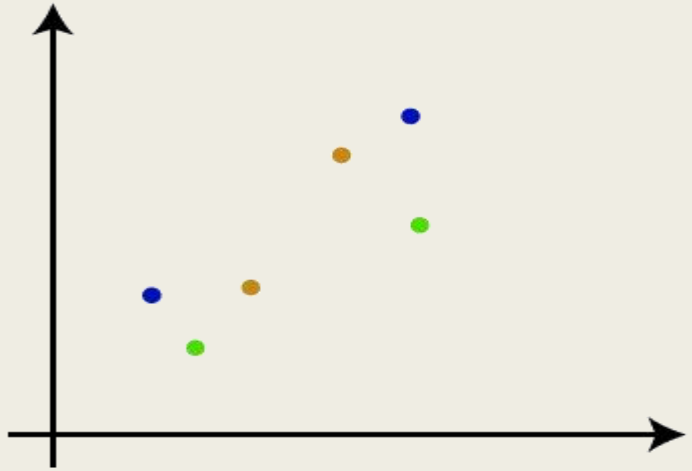
- **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

Agglomerative Hierarchical clustering

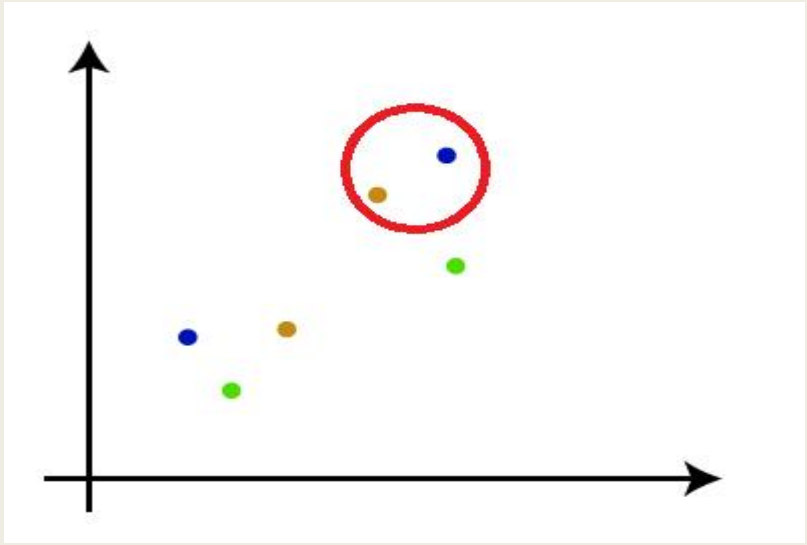
- The agglomerative hierarchical clustering algorithm is a popular example of HCA.
- To group the datasets into clusters, it follows the bottom-up approach.
- It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together.
- It does this until all the clusters are merged into a single cluster that contains all the datasets.
- This hierarchy of clusters is represented in the form of the dendrogram.

Steps of Agglomerative Hierarchical clustering

- Step-1: Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.

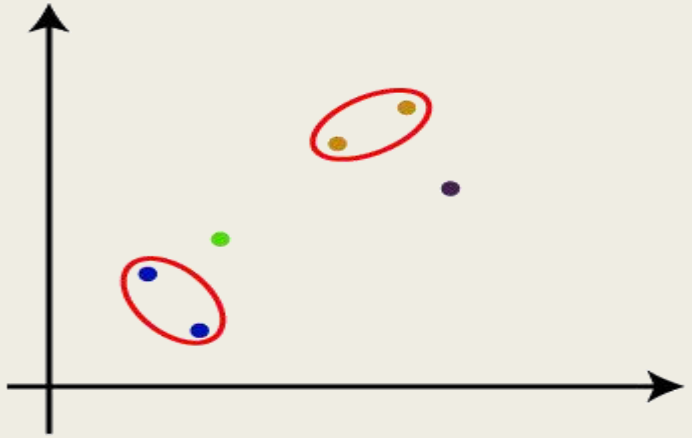


- Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.

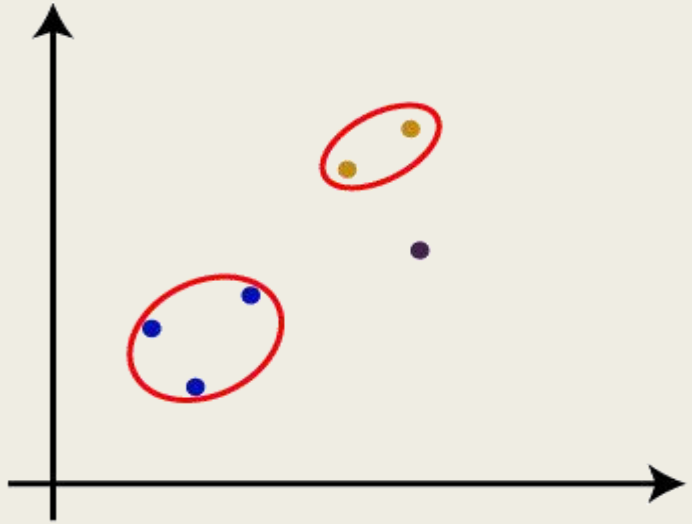


Steps of Agglomerative Hierarchical clustering

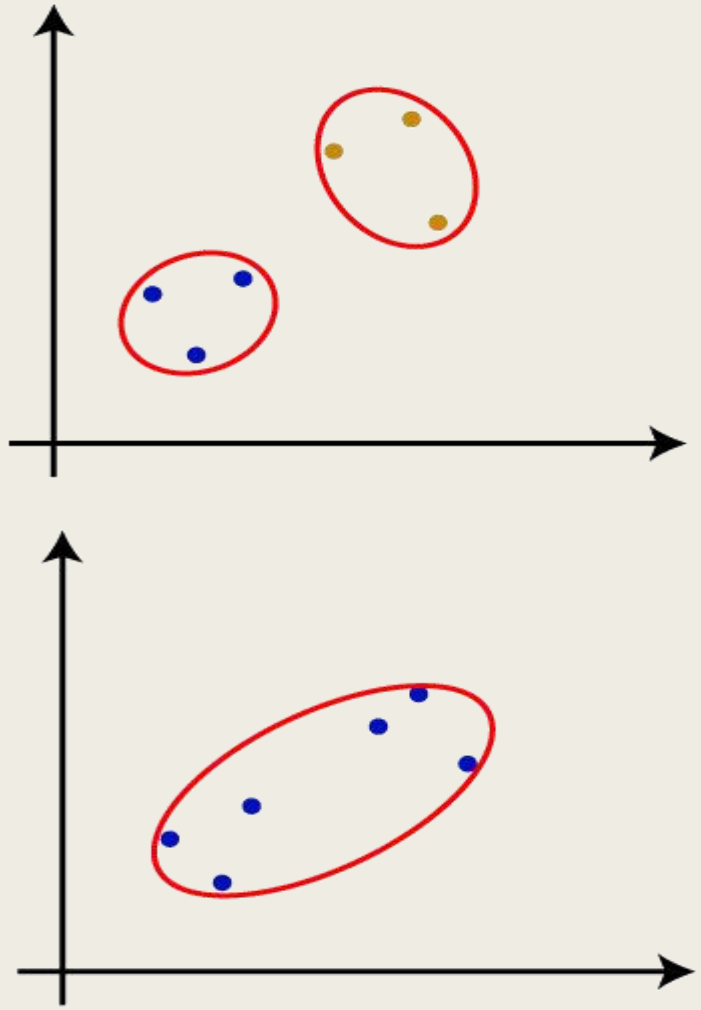
- Step-3: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.



- Step-4: Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:



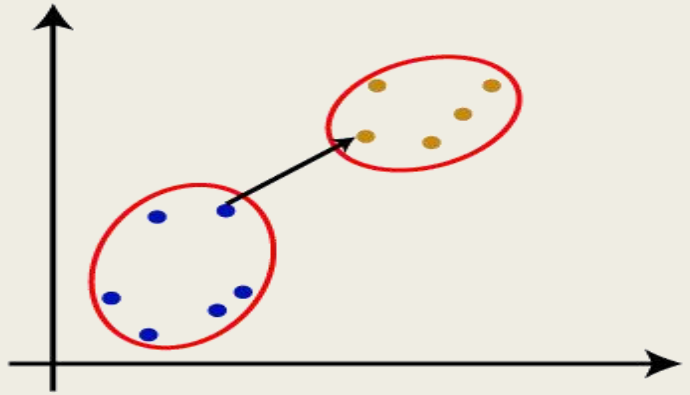
Steps of Agglomerative Hierarchical clustering



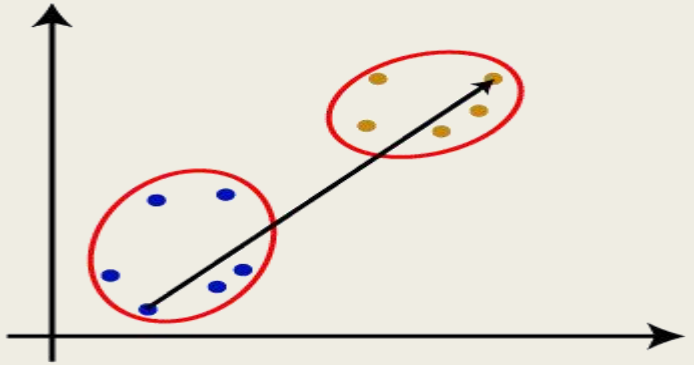
- Step-5: Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

Measure for the distance between two clusters

- There are **various ways to calculate the distance between two clusters**, and these ways decide the rule for clustering. These measures are called **Linkage methods**. Some of the popular linkage methods are given below:
- **Single Linkage:** It is the **Shortest Distance between the closest points of the clusters.** Consider the below image:

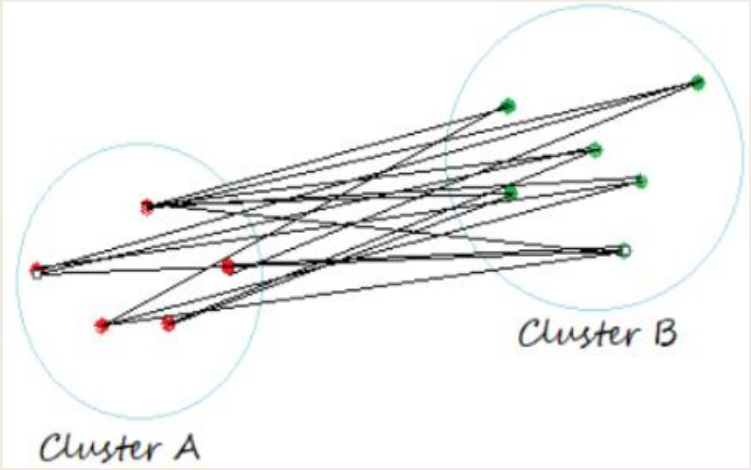


- **Complete Linkage:** It is the **farthest distance between the two points of two different clusters.** It is one of the popular linkage methods as **it forms tighter clusters than single-linkage.**

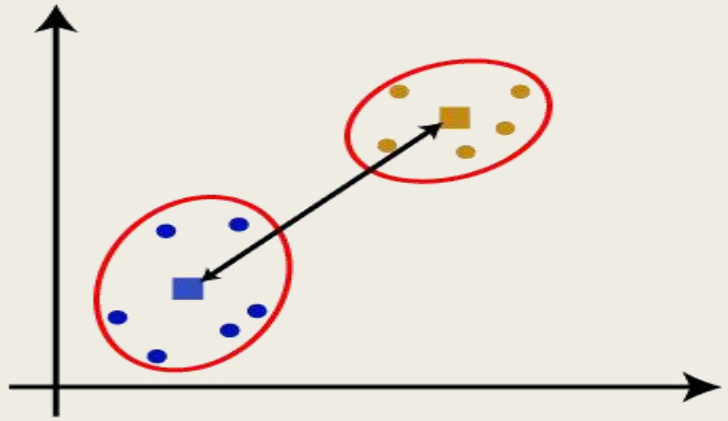


Measure for the distance between two clusters

- **Average Linkage:** It is the linkage method in which the **distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters.** It is also one of the most popular linkage methods.

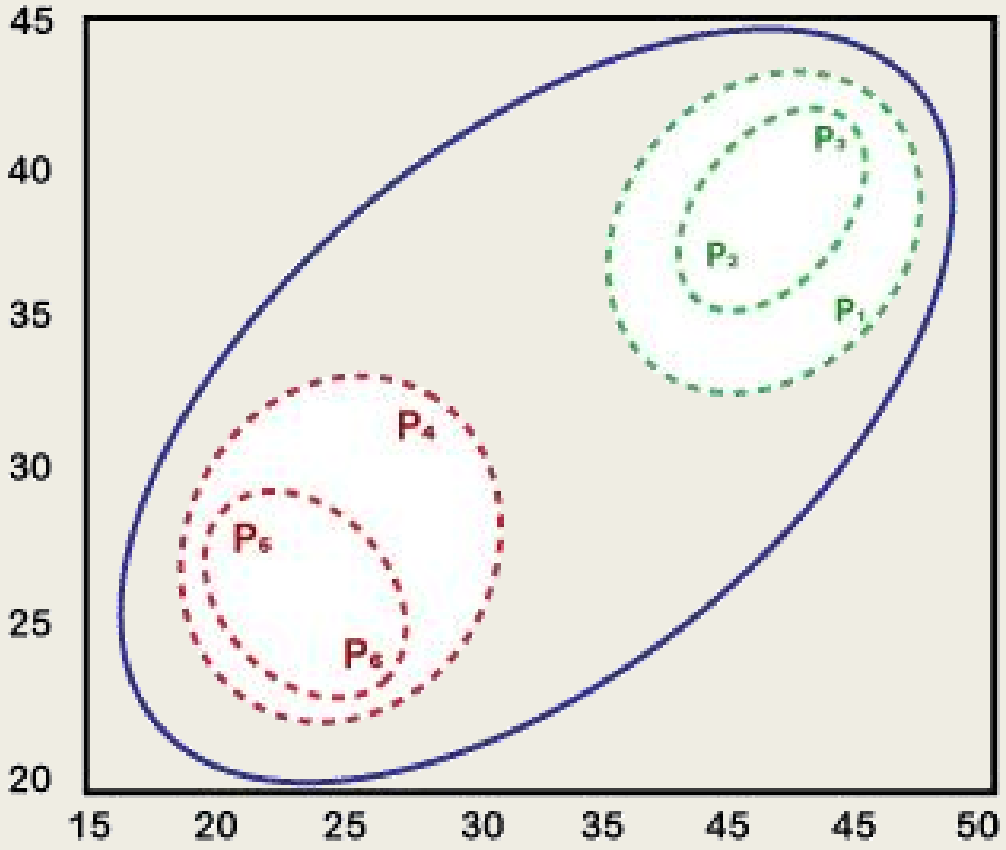


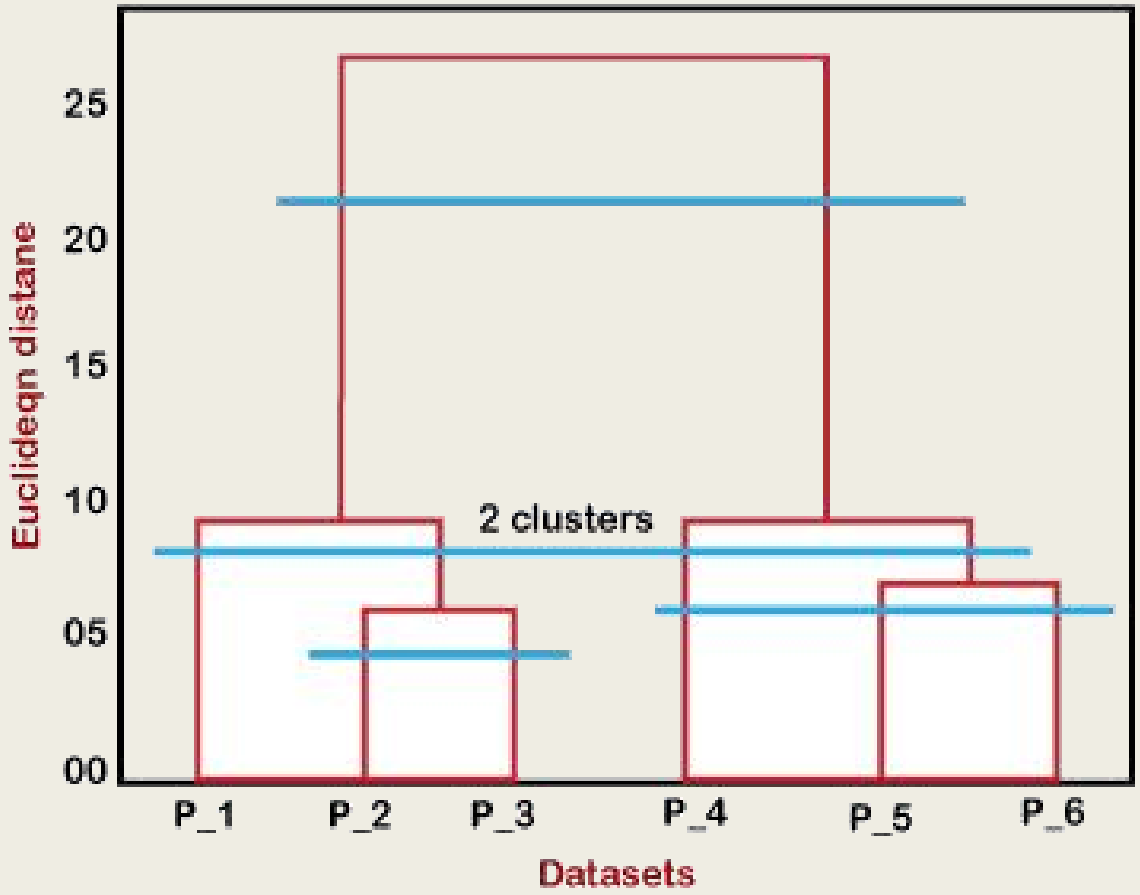
- **Centroid Linkage:** It is the linkage method in which the **distance between the centroid of the clusters is calculated.** Consider the below image:



Woking of Dendrogram in Hierarchical clustering

- The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.
- The working of the dendrogram can be explained using the below diagram:

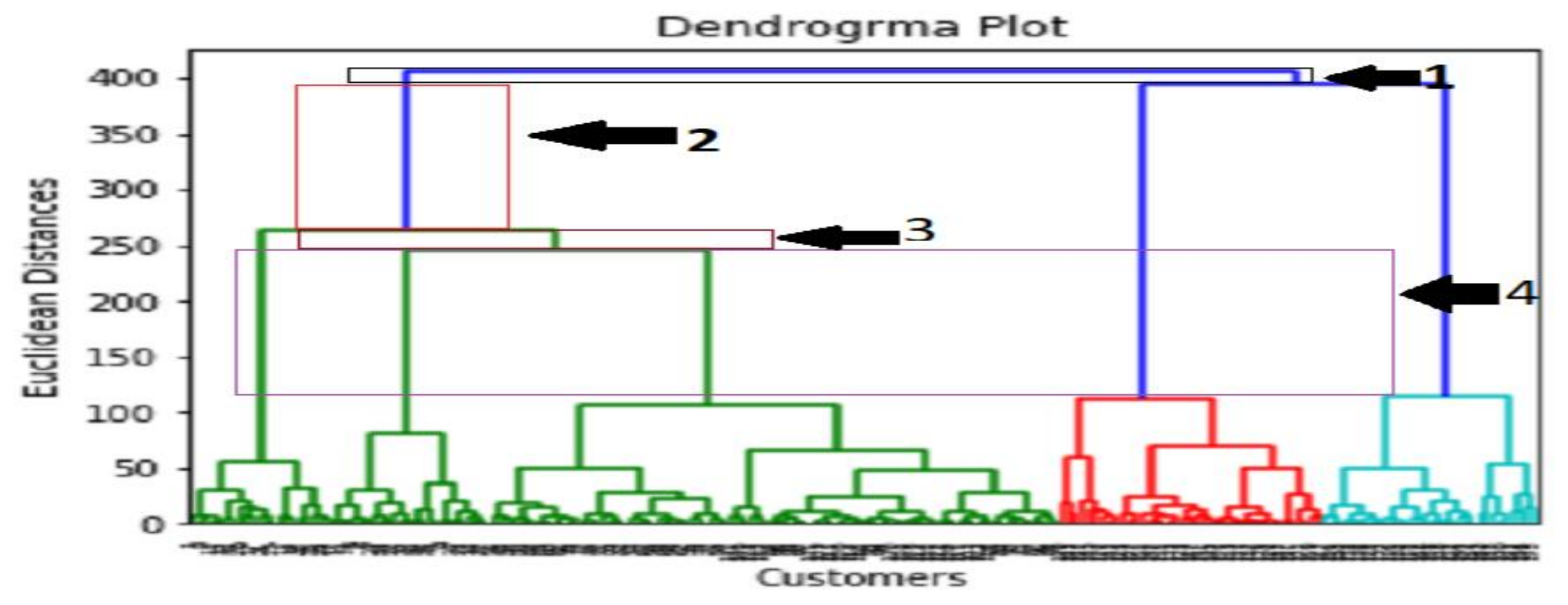




Woking of Dendrogram in Hierarchical clustering

- In the previous diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.
- Firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The hight is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
- Again, two new dendrograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.
- We can cut the dendrogram tree structure at any level as per our requirement.

Finding the optimal number of clusters using the Dendrogram



- In the above diagram, we have shown the **vertical distances that are not cutting their horizontal bars.**
- As we can visualize, the **4th distance is looking the maximum, so according to this, the number of clusters will be 5(the vertical lines in this range).**
- So, **the optimal number of clusters will be 5**, and we will train the model in the next step, using the same.

Divisive hierarchical clustering

- The **divisive method starts at the top and at each level recursively split one of the existing clusters at that level into two new clusters.**
- If there are N observations in the dataset, there the divisive method also will produce N − 1 levels in the hierarchy.
- The **split is chosen to produce two new groups with the largest "between-group dissimilarity".**
- For example, the divisive method is shown in Figure 13.11. Each nonterminal node has two daughter nodes.
- The two daughters represent the two groups resulting from the split of the parent.

Divisive hierarchical clustering

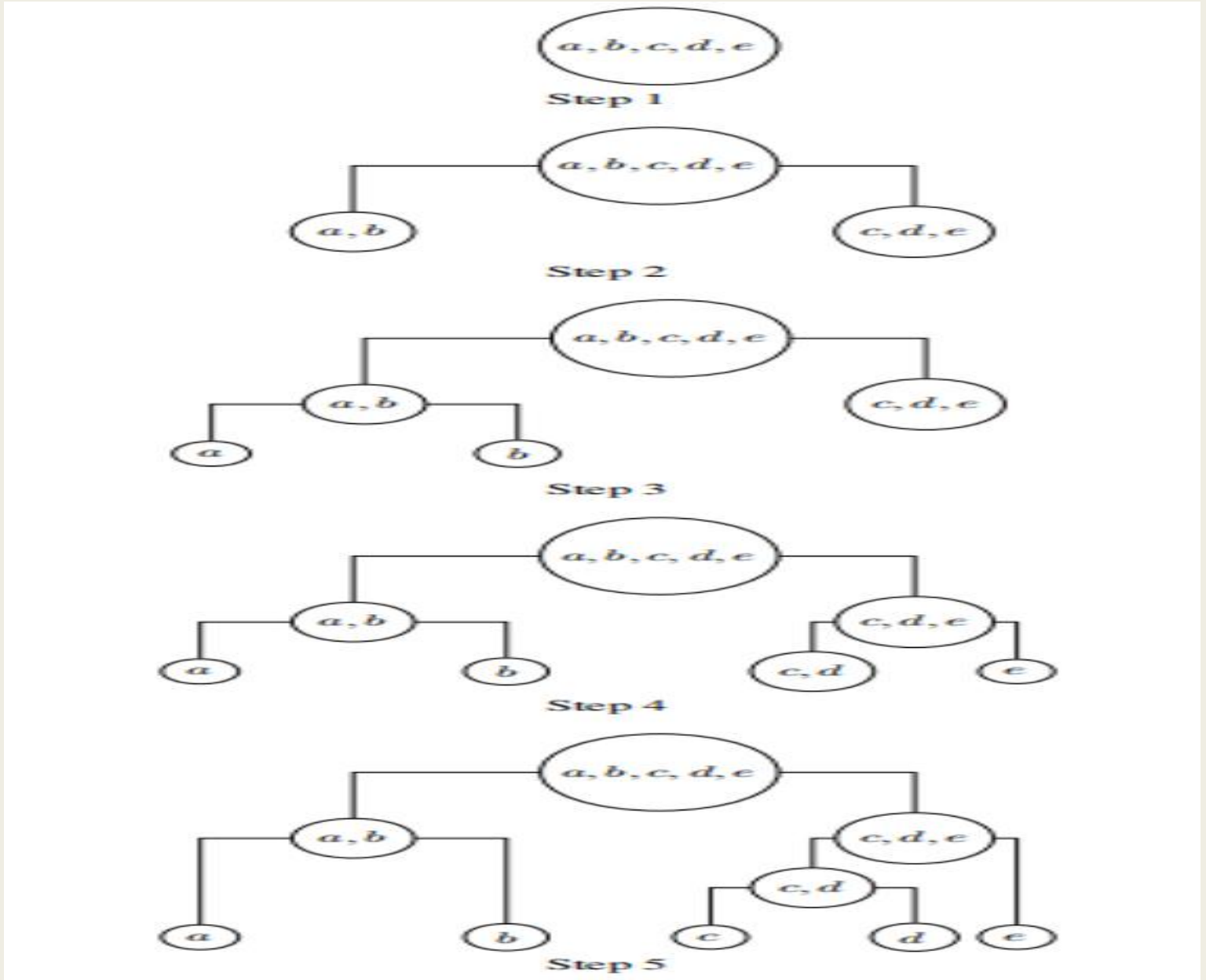


Figure 13.11: Hierarchical clustering using divisive method

Algorithm for divisive hierarchical clustering

- Divisive clustering algorithms begin with the entire data set as a single cluster, and recursively divide one of the existing clusters into two daughter clusters at each iteration in a top-down fashion.

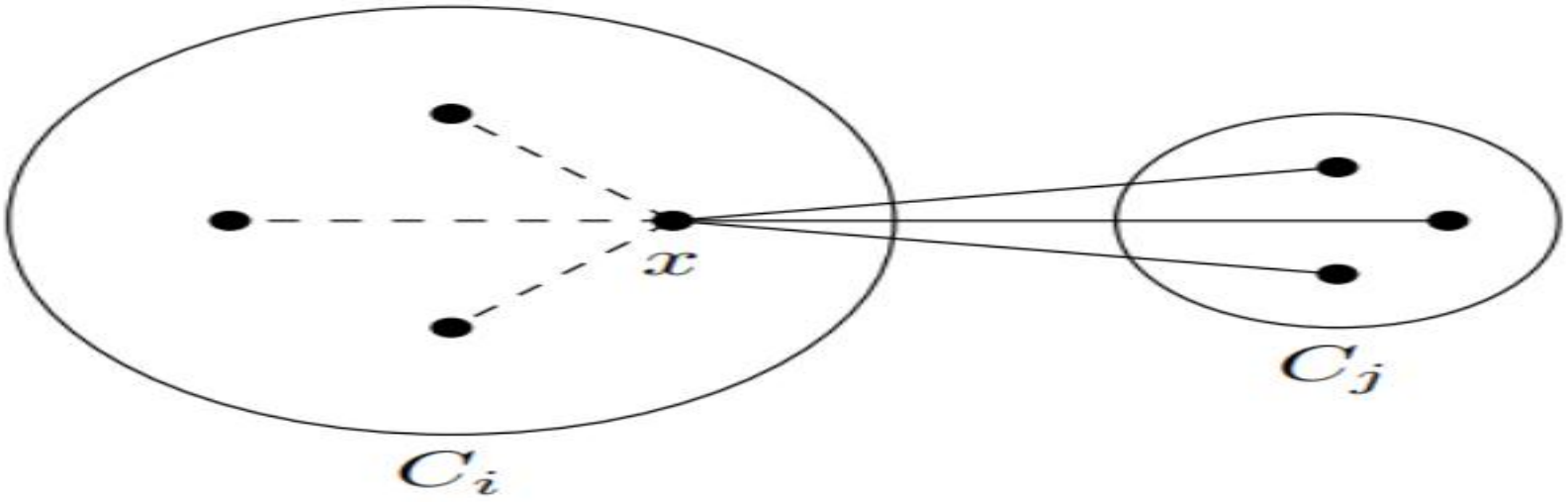
- To apply this procedure, we need a separate algorithm to divide a given dataset into two clusters.

- The divisive algorithm may be implemented by using the k-means algorithm with k = 2 to perform the splits at each iteration.

- However, it would not necessarily produce a splitting sequence that possesses the monotonicity property required for dendrogram representation.

DIANA (DIvisive ANAlysis)

- DIANA is a divisive hierarchical clustering technique. Here is an outline of the algorithm.
- Step 1. Suppose that cluster Cl is going to be split into clusters Ci and Cj .
- Step 2. Let Ci = Cl and Cj = ∅ .
- Step 3. For each object x ∈ Ci:
- (a) For the first iteration, compute the average distance of x to all other objects.
- (b) For the remaining iterations, compute
- Dx = average {d(x, y) : y ∈ Ci} − average{d(x, y) : y ∈ Cj}.



D_x= (average of dashed lines) − (average of solid lines)

DIANA (DIvisive ANAlysis)

- Step 4. (a) For the first iteration, move the object with the maximum average distance to Cj .
- (b) For the remaining iterations, find an object x in Ci for which Dx is the largest. If Dx > 0 then move x to Cj .
- Step 5. Repeat Steps 3(b) and 4(b) until all differences Dx are negative. Then Cl is split into Ci and Cj .
- Step 6. Select the smaller cluster with the largest diameter. (The diameter of a cluster is the largest dissimilarity between any two of its objects.) Then divide this cluster, following Steps 1-5.
- Step 7. Repeat Step 6 until all clusters contain only a single object.

Example of DIANA (DIvisive ANAlysis)

- **Problem :**Given the dataset {a, b, c, d, e} and the distance matrix in Table 13.4, construct a dendrogram by the divisive analysis algorithm.

	a	b	c	d	e
a	0	9	3	6	11
b	9	0	7	5	10
c	3	7	0	9	2
d	6	5	9	0	8
e	11	10	2	8	0

Table 13.4: Example for distance matrix

- **Solution:**
- 1. We have, initially
- Cl = {a, b, c, d, e}
- 2. We write
- Ci = Cl, Cj = ∅ .
- 3. Division into clusters

Example of DIANA (DIvisive ANAlysis)

- (a) Initial iteration
- Let us calculate the average dissimilarities of the objects in Ci with the other objects in Ci.
- Average dissimilarity of a $= \frac{1}{4}(d(a,b) + d(a,c) + d(a,e)) = \frac{1}{4}(9 + 3 + 6 + 11) = 7.25$

- Similarly we have :
- Average dissimilarity of b = 7.75
- Average dissimilarity of c = 5.25
- Average dissimilarity of d = 7.00
- Average dissimilarity of e = 7.75
- The highest average distance is 7.75 and there are two corresponding objects. We choose one of them, b, arbitrarily. We move b to Cj .

- We now have
- Ci = {a, c, d, e}, Cj = ∅ ∪ {b} = {b}

Example of DIANA (DIvisive ANAlysis)

- (b) Remaining iterations
- (i) 2-nd iteration.

$$D_a = \frac{1}{3}(d(a,c) + d(a,d) + d(a,e)) - \frac{1}{1}(d(a,b)) = \frac{20}{3} - 9 = -2.33$$
$$D_c = \frac{1}{3}(d(c,a) + d(c,d) + d(c,e)) - \frac{1}{1}(d(c,b)) = \frac{14}{3} - 7 = -2.33$$
$$D_d = \frac{1}{3}(d(d,a) + d(d,c) + d(d,e)) - \frac{1}{1}(d(c,b)) = \frac{23}{3} - 7 = 0.67$$
$$D_e = \frac{1}{3}(d(e,a) + d(e,c) + d(e,d)) - \frac{1}{1}(d(e,b)) = \frac{21}{3} - 7 = 0$$

- Dd is the largest and Dd > 0. So we move, d to Cj .
- We now have
- Ci = {a, c, e}, Cj = {b} ∪ {d} = {b, d}.
- (ii) 3-rd iteration

$$D_a = \frac{1}{2}(d(a,c) + d(a,e)) - \frac{1}{2}(d(a,b) + d(a,d)) = \frac{14}{2} - \frac{15}{2} = -0.5$$
$$D_c = \frac{1}{2}(d(c,a) + d(c,e)) - \frac{1}{2}(d(c,b) + d(c,d)) = \frac{5}{2} - \frac{16}{2} = -13.5$$
$$D_e = \frac{1}{2}(d(e,a) + d(e,c)) - \frac{1}{2}(d(e,b) + d(e,d)) = \frac{13}{2} - \frac{18}{2} = -2.5$$

- All are negative. So we stop and form the clusters Ci and Cj .

Example of DIANA (DIvisive ANAlysis)

- 4. To divide, Ci and Cj , we compute their diameters.

$$\begin{aligned}\text{diameter}(C_i) &= \max\{d(a,c), d(a,e), d(c,e)\} \\ &= \max\{3, 11, 2\} \\ &= 11 \\ \text{diameter}(C_j) &= \max\{d(b,d)\} \\ &= 5\end{aligned}$$

- The cluster with the largest diameter is Ci.
- So we now split Ci.
- We repeat the process by taking Cl = {a, c, e}.
- The remaining computations are left as an exercise to the students.

Example of AGNES (AGglomerative NESting)

- **Problem 1 :**Given the dataset {a, b, c, d, e} and the following distance matrix, construct a dendrogram by complete linkage hierarchical clustering using the agglomerative method.

	a	b	c	d	e
a	0	9	3	6	11
b	9	0	7	5	10
c	3	7	0	9	2
d	6	5	9	0	8
e	11	10	2	8	0

Table 13.4: Example for distance matrix

- **Solution:**
- The complete-linkage clustering uses the "maximum formula", that is, the following formula to compute the distance between two clusters A and B:
- $d(A, B) = \max\{d(x, y) : x \in A, y \in B\}$
- 1. Initial clustering (singleton sets)
- Dataset : {a, b, c, d, e}.
- C1: {a}, {b}, {c}, {d}, {e}.

Example of AGNES (AGglomerative NESting)

- 2. The following table gives the distances between the various clusters in C1:

	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$
$\{a\}$	0	9	3	6	11
$\{b\}$	9	0	7	5	10
$\{c\}$	3	7	0	9	**2**
$\{d\}$	6	5	9	0	8
$\{e\}$	11	10	**2**	8	0

- In the above table, the minimum distance is the distance between the clusters {c} and {e}.
- Also
- d({c}, {e}) = 2.
- We merge {c} and {e} to form the cluster {c, e}.
- The new set of clusters C2: {a}, {b}, {d}, {c, e}.

Example of AGNES (AGglomerative NESting)

- 3. Let us compute the distance of {c, e} from other clusters.
- d({c, e}, {a}) = max{d(c, a), d(e, a)} = max{3, 11} = 11.
- d({c, e}, {b}) = max{d(c, b), d(e, b)} = max{7, 10} = 10.
- d({c, e}, {d}) = max{d(c, d), d(e, d)} = max{9, 8} = 9.
- The following table gives the distances between the various clusters in C2.

	$\{a\}$	$\{b\}$	$\{d\}$	$\{c, e\}$
$\{a\}$	0	9	6	11
$\{b\}$	9	0	**5**	10
$\{d\}$	6	**5**	0	9
$\{c, e\}$	11	10	9	0

- In the above table, the minimum distance is the distance between the clusters {b} and {d}.
- Also
- d({b}, {d}) = 5.
- We merge {b} and {d} to form the cluster {b, d}.
- The new set of clusters C3: {a}, {b, d}, {c, e}.

Example of AGNES (AGglomerative NESting)

- 4. Let us compute the distance of {b, d} from other clusters.
- d({b, d}, {a}) = max{d(b, a), d(d, a)} = max{9, 6} = 9.
- d({b, d}, {c, e}) = max{d(b, c), d(b, e), d(d, c), d(d, e)} = max{7, 10, 9, 8} = 10.
- The following table gives the distances between the various clusters in C3.

	$\{a\}$	$\{b, d\}$	$\{c, e\}$
$\{a\}$	0	**9**	11
$\{b, d\}$	**9**	0	10
$\{c, e\}$	11	10	0

- In the above table, the minimum distance is the distance between the clusters {a} and {b, d}.
- Also
- d({a}, {b, d}) = 9.
- We merge {a} and {b, d} to form the cluster {a, b, d}.
- The new set of clusters C4: {a, b, d}, {c, e}

Example of AGNES (AGglomerative NESting)

- 5. Only two clusters are left. We merge them form a single cluster containing all data points. We have
- $d(\{a, b, d\}, \{c, e\}) = \max\{d(a, c), d(a, e), d(b, c), d(b, e), d(d, c), d(d, e)\}$

  $= \max\{3, 11, 7, 10, 9, 8\}$

  $= 11$
- 6. Figure 13.14 shows the dendrogram of the hierarchical clustering.

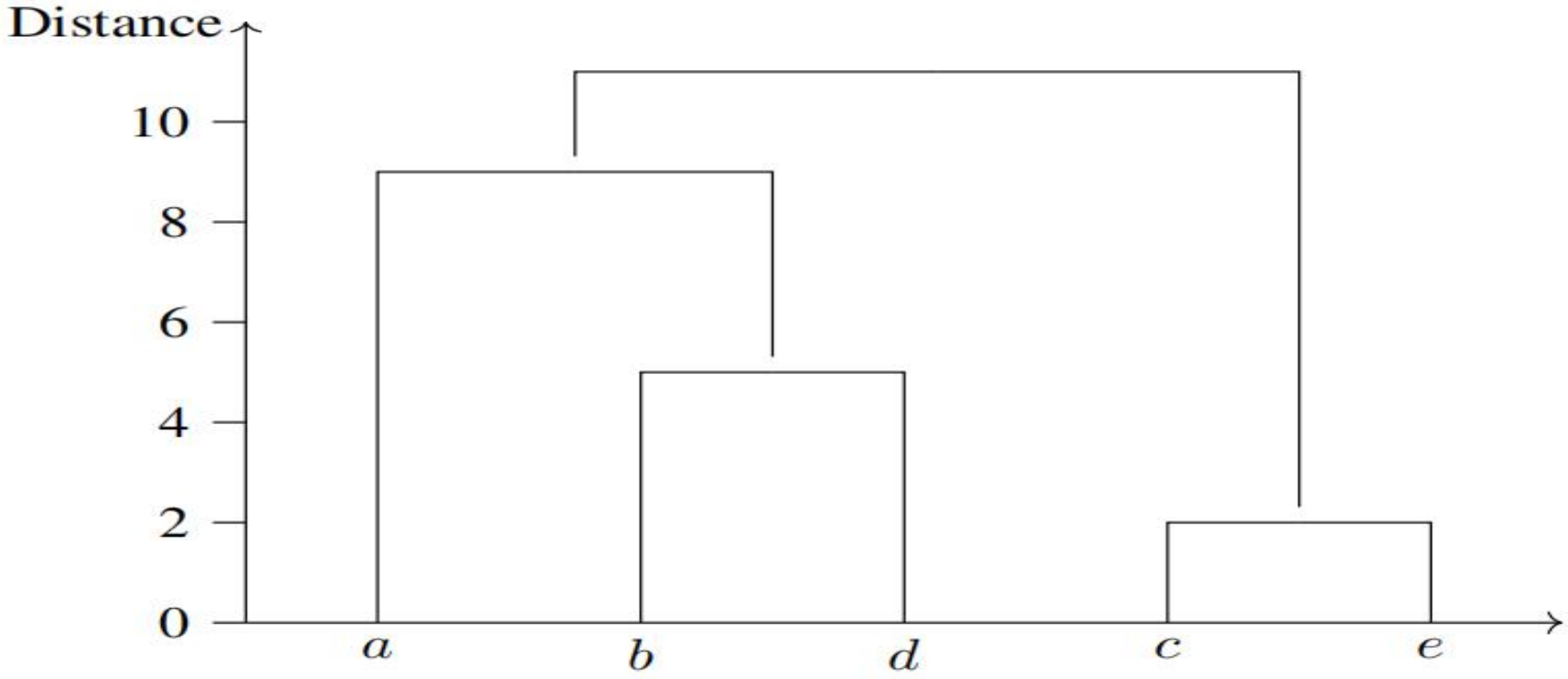


Figure 13.14: Dendrogram for the data given in Table 13.4 (complete linkage clustering)

Example of AGNES (AGglomerative NESting)

- **Problem 2 :**Given the dataset {a, b, c, d, e} and the distance matrix given in Table 13.4, construct a dendrogram by single-linkage hierarchical clustering using the agglomerative method.

	a	b	c	d	e
a	0	9	3	6	11
b	9	0	7	5	10
c	3	7	0	9	2
d	6	5	9	0	8
e	11	10	2	8	0

Table 13.4: Example for distance matrix

- **Solution:**
- The complete-linkage clustering uses the "maximum formula", that is, the following formula to compute the distance between two clusters A and B:
- $d(A, B) = \min\{d(x, y) : x \in A, y \in B\}$
- 1.Initial clustering (singleton sets)
- Dataset : {a, b, c, d, e}.
- C1: {a}, {b}, {c}, {d}, {e}.

Example of AGNES (AGglomerative NESting)

- 2. The following table gives the distances between the various clusters in C1:

	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$
$\{a\}$	0	9	3	6	11
$\{b\}$	9	0	7	5	10
$\{c\}$	3	7	0	9	**2**
$\{d\}$	6	5	9	0	8
$\{e\}$	11	10	**2**	8	0

- In the above table, the minimum distance is the distance between the clusters {c} and {e}.
- Also
- d({c}, {e}) = 2.
- We merge {c} and {e} to form the cluster {c, e}.
- The new set of clusters C2: {a}, {b}, {d}, {c, e}.

Example of AGNES (AGglomerative NESting)

- 3. Let us compute the distance of {c, e} from other clusters.
- d({c, e}, {a}) = min{d(c, a), d(e, a)} = max{3, 11} = 3.
- d({c, e}, {b}) = min{d(c, b), d(e, b)} = max{7, 10} = 7.
- d({c, e}, {d}) = min{d(c, d), d(e, d)} = max{9, 8} = 8.
- The following table gives the distances between the various clusters in C2.

	$\{a\}$	$\{b\}$	$\{d\}$	$\{c, e\}$
$\{a\}$	0	9	6	**3**
$\{b\}$	9	0	5	7
$\{d\}$	6	5	0	8
$\{c, e\}$	**3**	7	8	0

- In the above table, the minimum distance is the distance between the clusters {a} and {c, e}.
- Also
- d({a}, {c, e}) = 3.
- We merge {a} and {c, e} to form the cluster {a, c, e}.
- The new set of clusters C3: {a, c, e}, {b}, {d}.

Example of AGNES (AGglomerative NESting)

- 4. Let us compute the distance of {a, c, e} from other clusters.
- d({a, c, e}, {b}) = min{d(a, b), d(c, b), d(e, b)} = {9, 7, 10} = 7
- d({a, c, e}, {d}) = min{d(a, d), d(c, d), d(e, d)} = {6, 9, 8} = 6
- The following table gives the distances between the various clusters in C3.

	$\{a, c, e\}$	$\{b\}$	$\{d\}$
$\{a, c, e\}$	0	7	6
$\{b\}$	7	0	**5**
$\{d\}$	6	**5**	0

- In the above table, the minimum distance is between {b} and {d}. Also
- d({b}, {d}) = 5.
- We merge {b} and {d} to form the cluster {b, d}.
- The new set of clusters C4: {a, c, e}, {b, d}

Example of AGNES (AGglomerative NESting)

- 5. Only two clusters are left. We merge them form a single cluster containing all data points. We have
- $d(\{a, c, e\}, \{b, d\}) = \min\{d(a, b), d(a, d), d(c, b), d(c, d), d(e, b), d(e, d)\}$

  $= \min\{9, 6, 7, 9, 10, 8\}$

  $= 6$
- 6. Figure 13.15 shows the dendrogram of the hierarchical clustering.

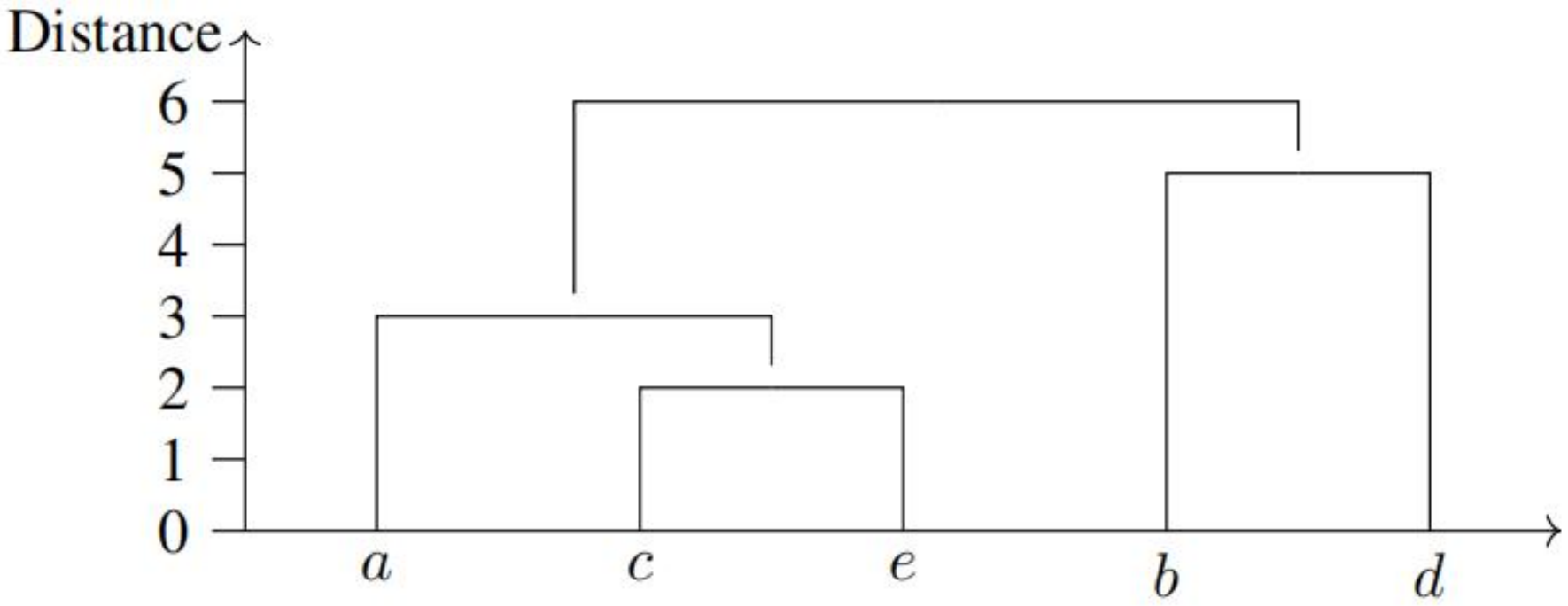


Figure 13.15: Dendrogram for the data given in Table 13.4 (single linkage clustering)

Support Vector Machine Algorithm

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is **used for Classification as well as Regression problems.**

- However, primarily, it is used for Classification problems in Machine Learning.

- The **goal of the SVM** algorithm is to **create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category** in the future.

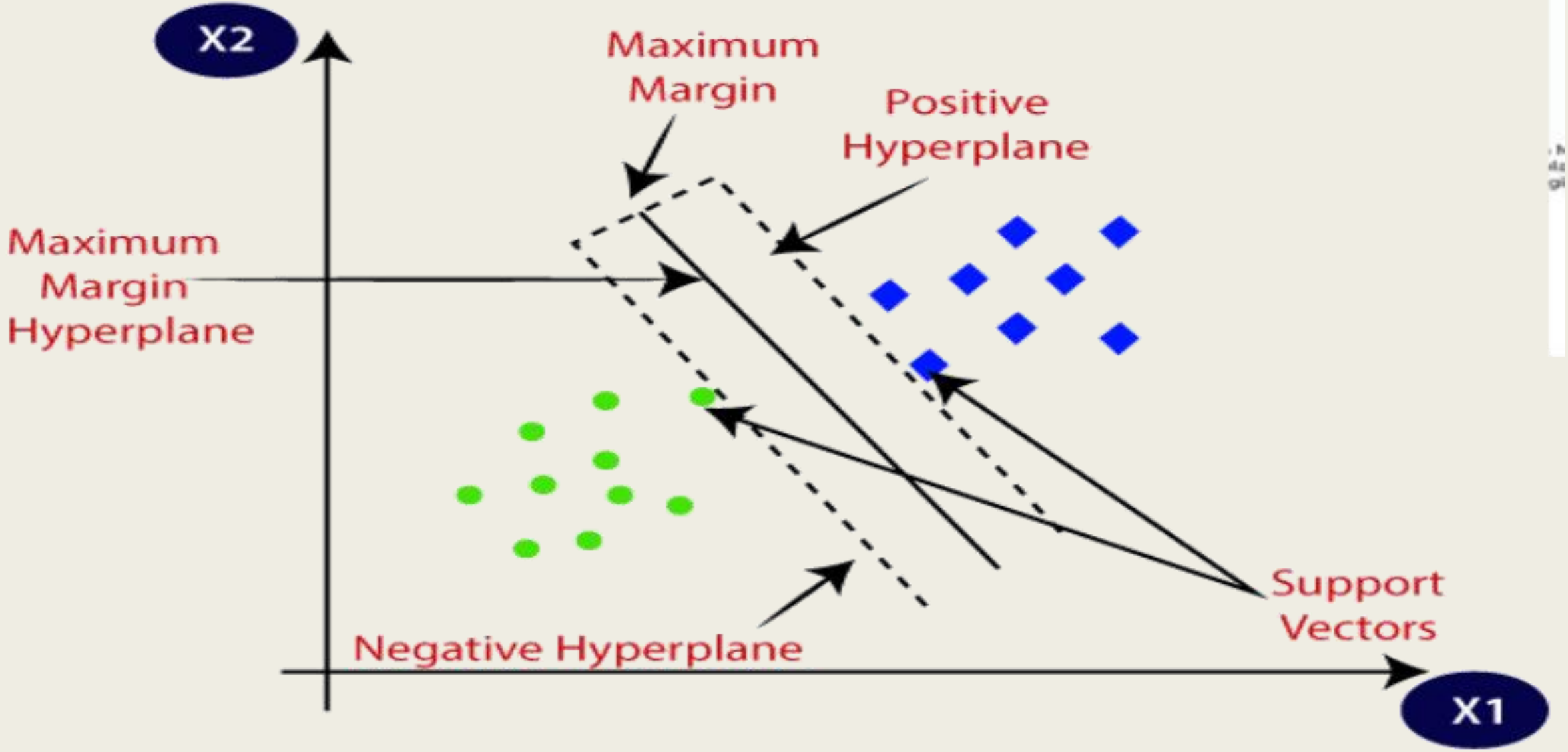
- This **best decision boundary is called a hyperplane.**

- SVM **chooses the extreme points/vectors that help in creating the hyperplane.**

- These **extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.**

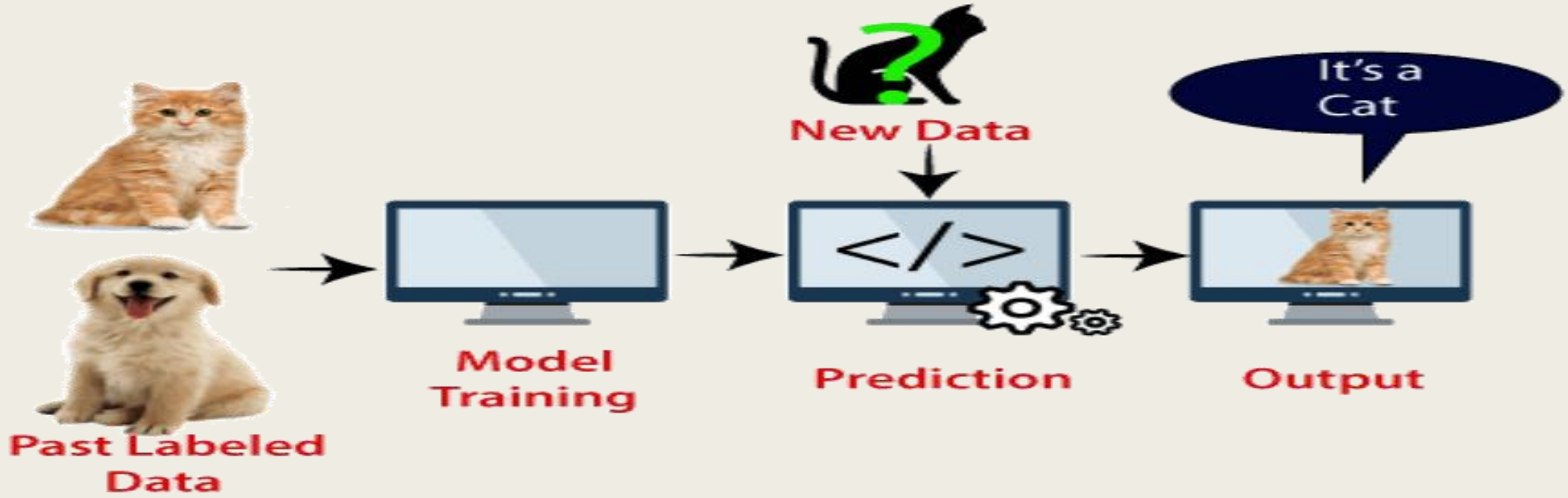
Support Vector Machine Algorithm

- Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Support Vector Machine Algorithm

- **Example:** Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, a model can be created by using the SVM algorithm.
- We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature.
- So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog.
- On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



- SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

- SVM can be of two types:
- **Linear SVM:** Linear SVM is **used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line,** then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

- **Non-linear SVM:** Non-Linear SVM is **used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line,** then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

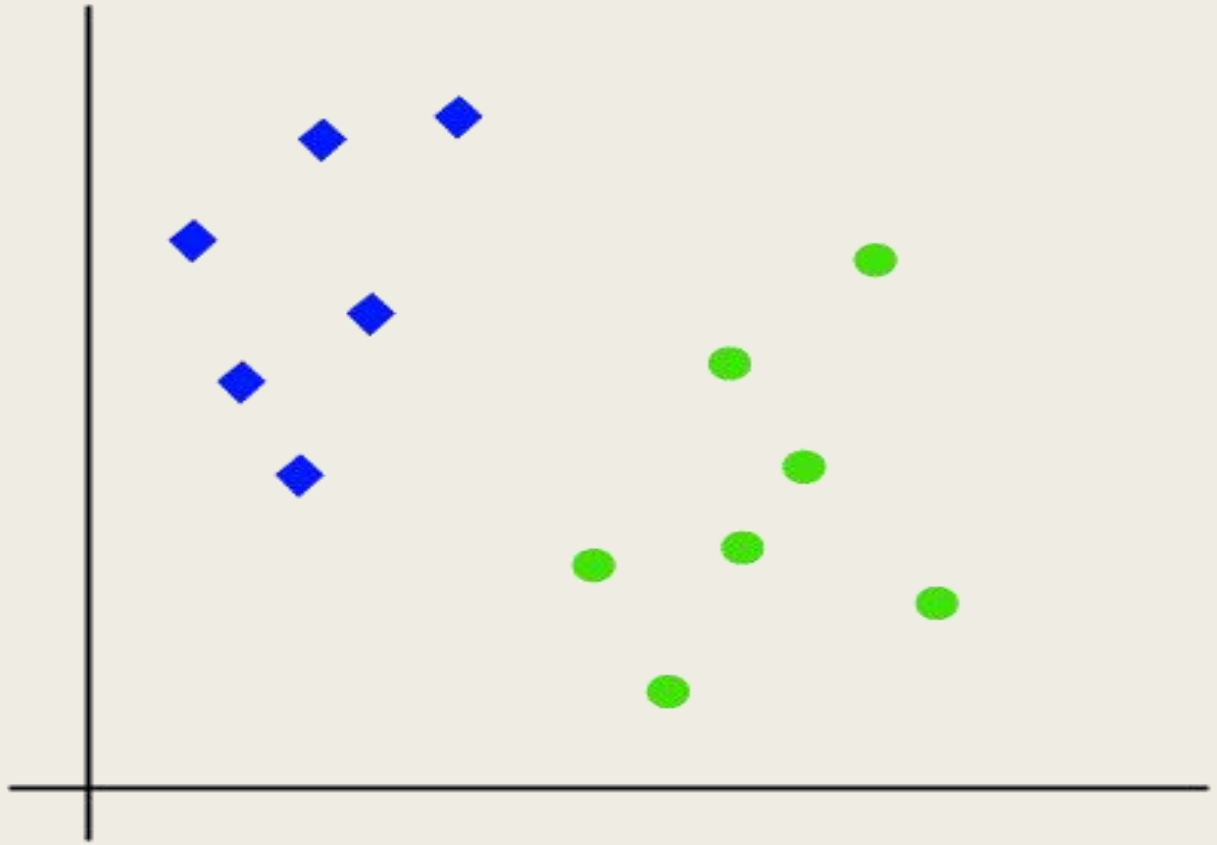
Hyperplane and Support Vectors in the SVM algorithm

- **Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points.
- This best boundary is known as the hyperplane of SVM.
- **The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features, then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.**
- We always **create a hyperplane that has a maximum margin, which means the maximum distance between the data points.**

- **Support Vectors:**
- The data points or vectors that are the **closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector.**
- Since **these vectors support the hyperplane, hence called a Support vector.**

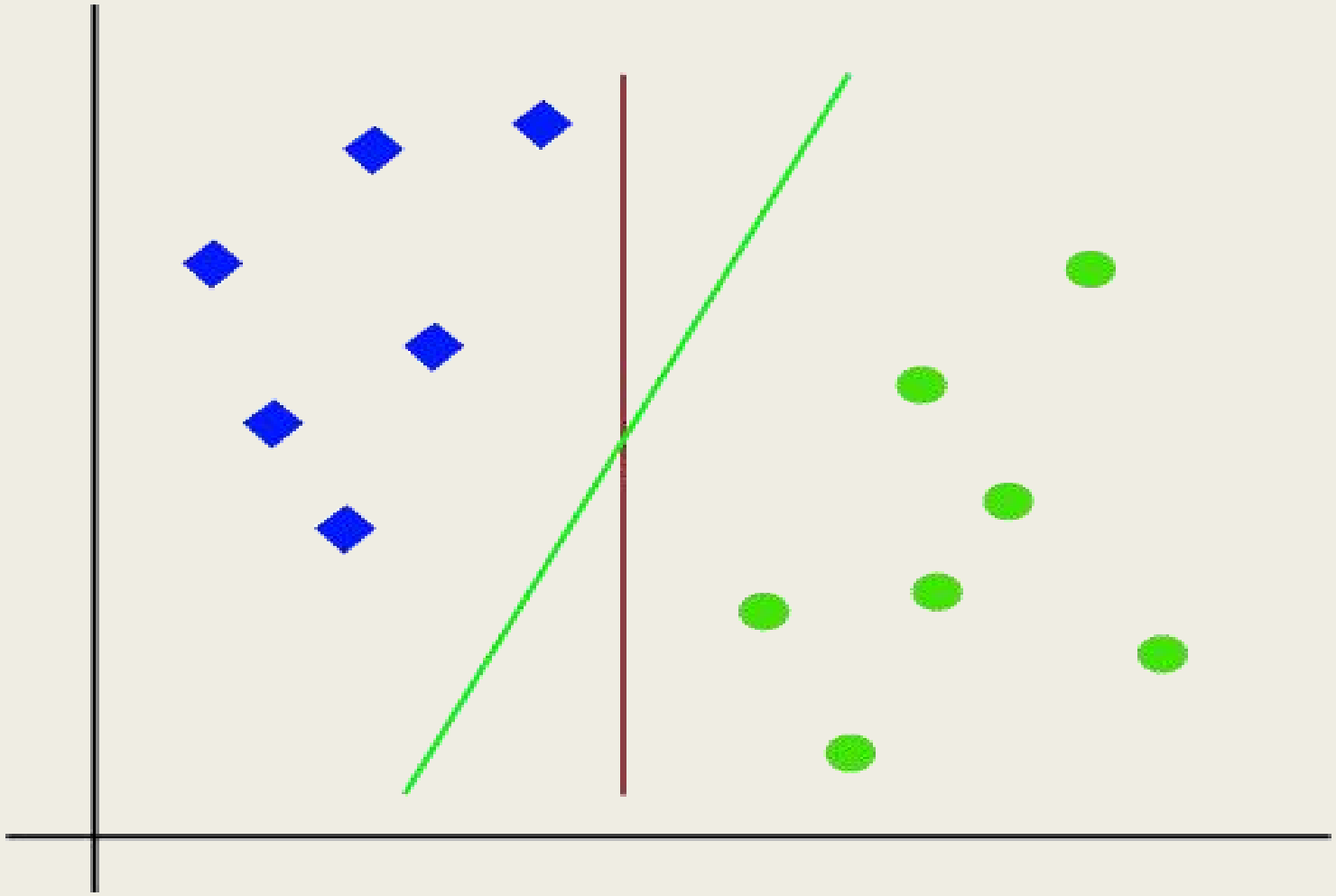
How does SVM works?

- **Linear SVM:**
- The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2.
- We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue.
- Consider the below image:



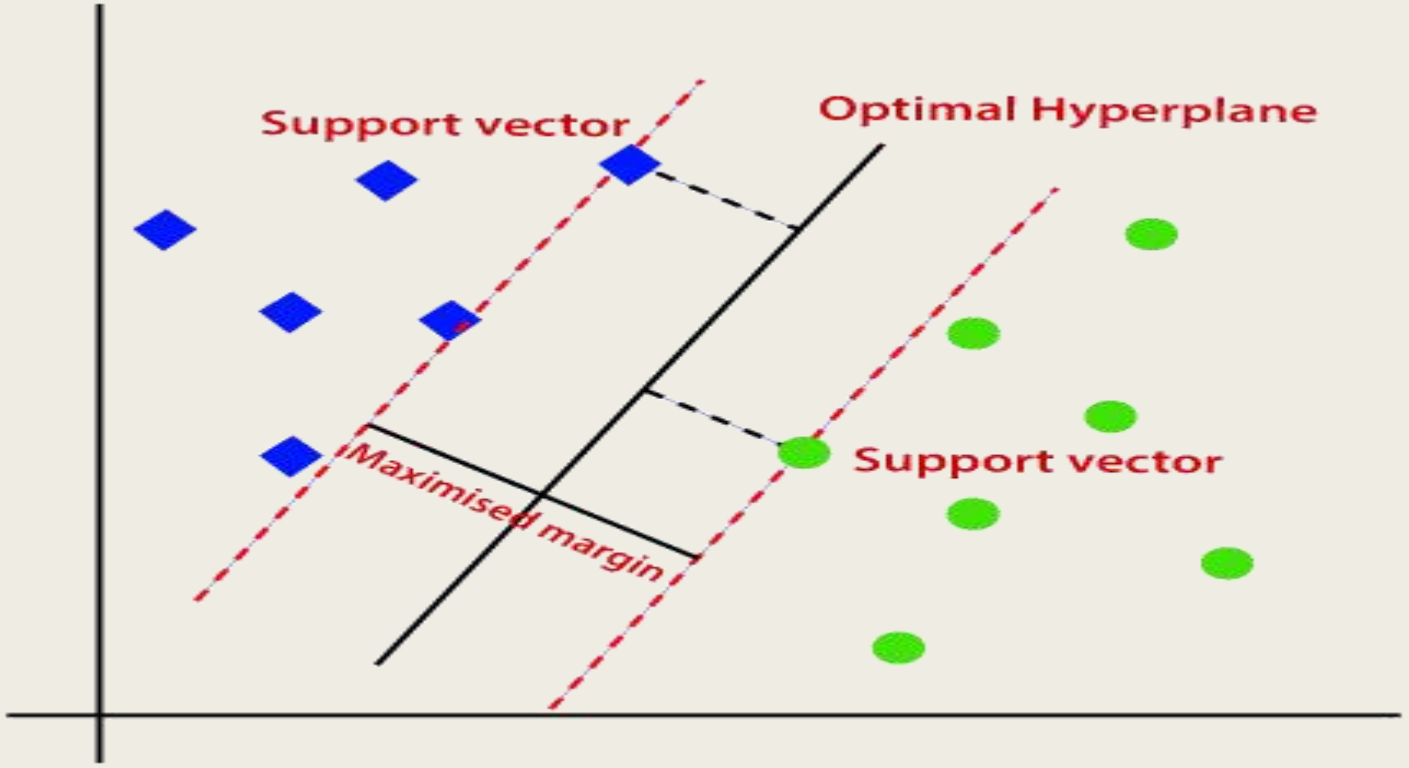
How does SVM works?

- So as it is 2-d space so by just using a straight line, we can easily separate these two classes.
- But there can be multiple lines that can separate these classes. Consider the below image:



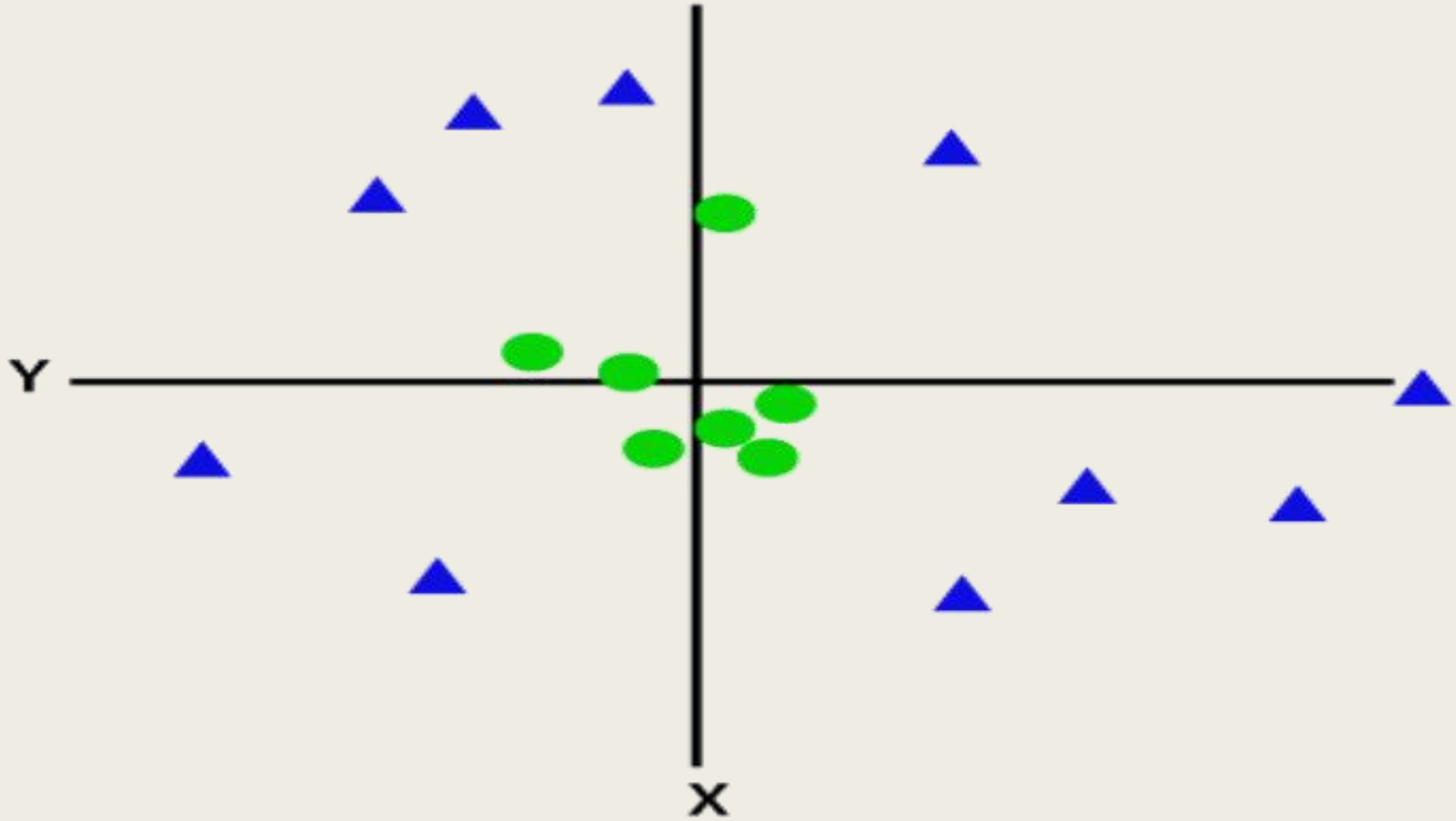
How does SVM works?

- The SVM algorithm helps to **find the best line or decision boundary; this best boundary or region is called as a hyperplane.**
- SVM algorithm **finds the closest point of the lines from both the classes. These points are called support vectors.**
- The **distance between the vectors and the hyperplane is called as margin.**
- And the **goal of SVM is to maximize this margin.**
- The **hyperplane with maximum margin is called the optimal hyperplane.**



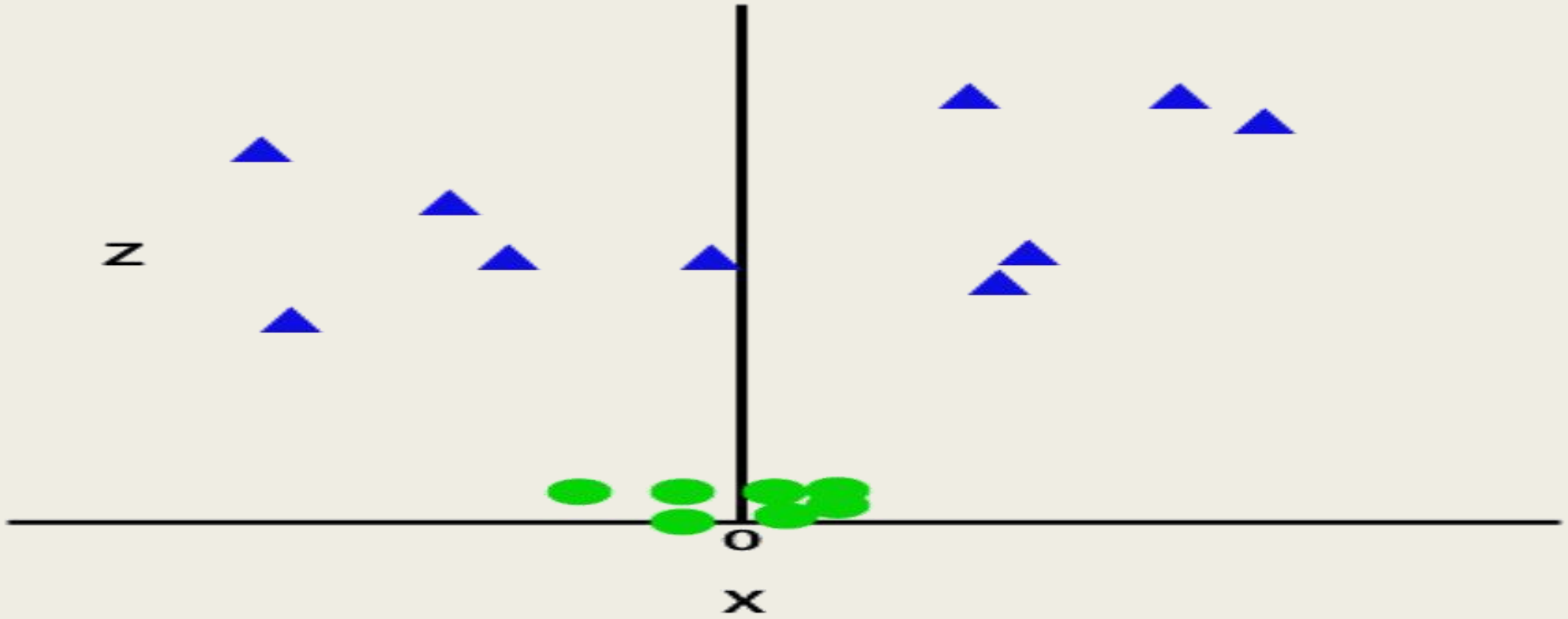
How does SVM works?

- **Non-Linear SVM:** If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line.
- Consider the below image:



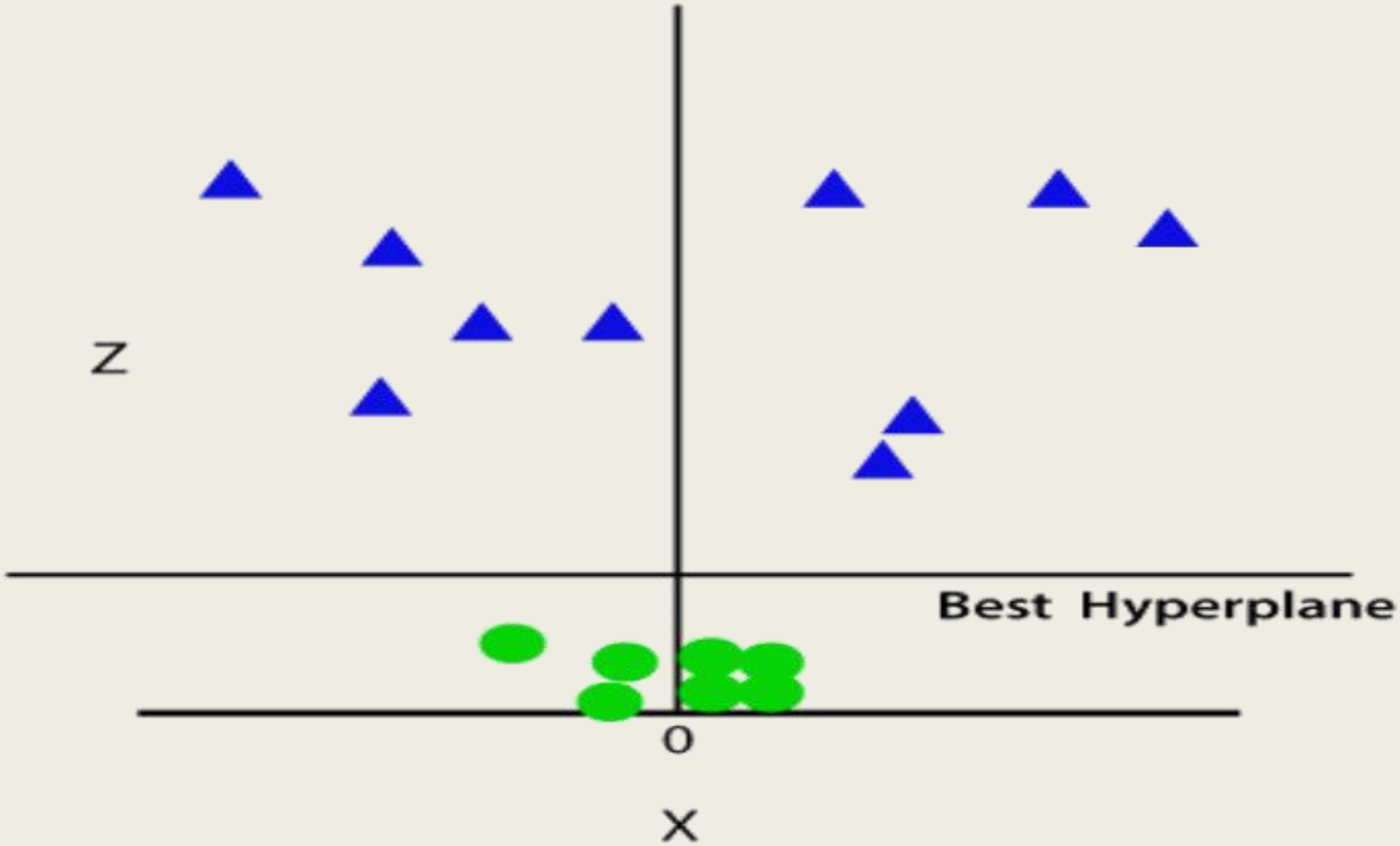
How does SVM works?

- So to separate these data points, we need to add one more dimension.
- For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:
- $z=x^2+y^2$
- By adding the third dimension, the sample space will become as below image:



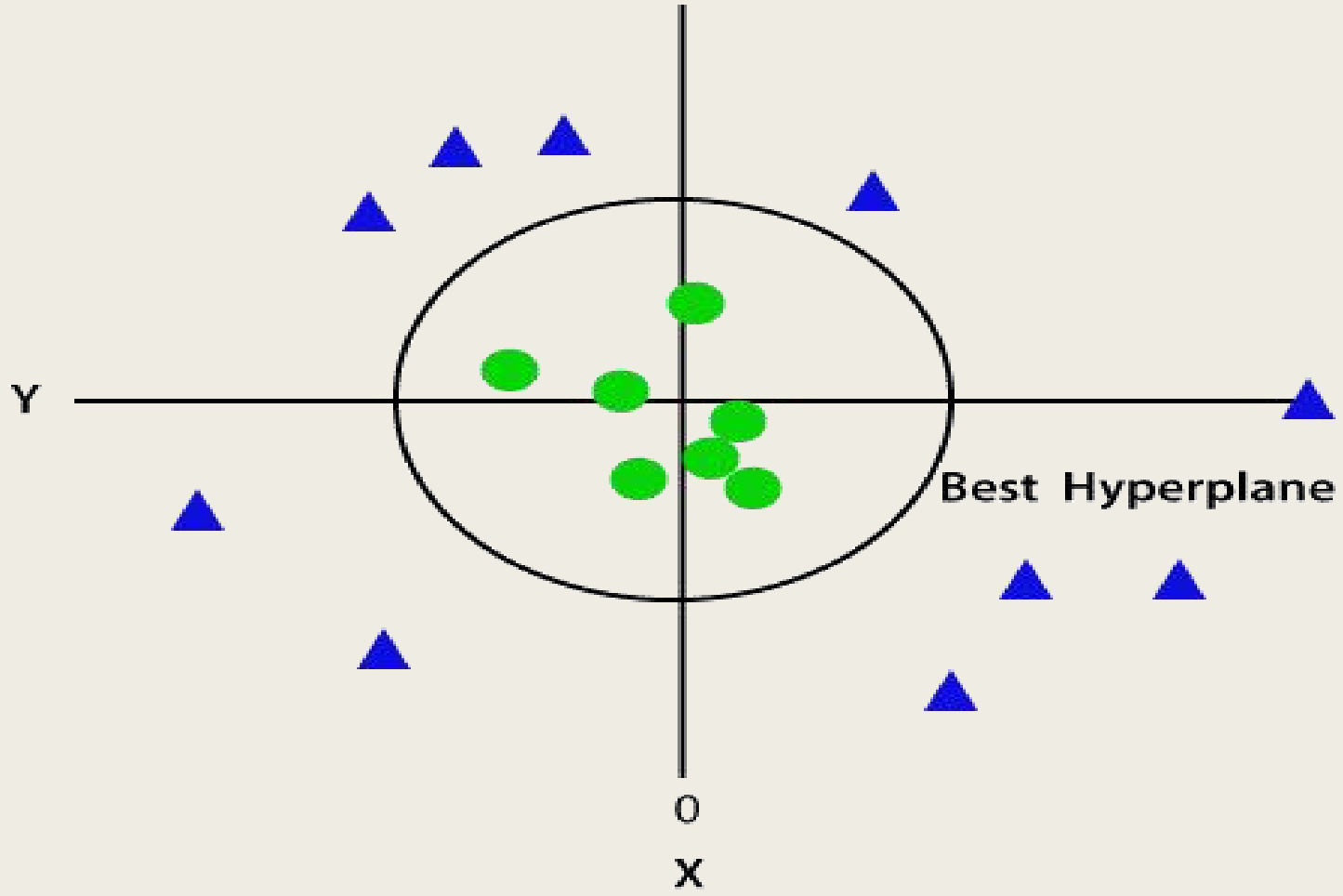
How does SVM works?

- So now, SVM will divide the datasets into classes in the following way.
- Consider the below image:



How does SVM works?

- Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis.
- If we convert it in 2d space with z=1, then it will become as:



- Hence we get a circumference of radius 1 in case of given non-linear data.

Numerical Example of Linear SVM

Suppose we are given the following positively labeled data points in $\Re^2$:

$$\left\{ \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 6 \\ -1 \end{pmatrix} \right\}$$

and the following negatively labeled data points in $\Re^2$ (see Figure 1):

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$$

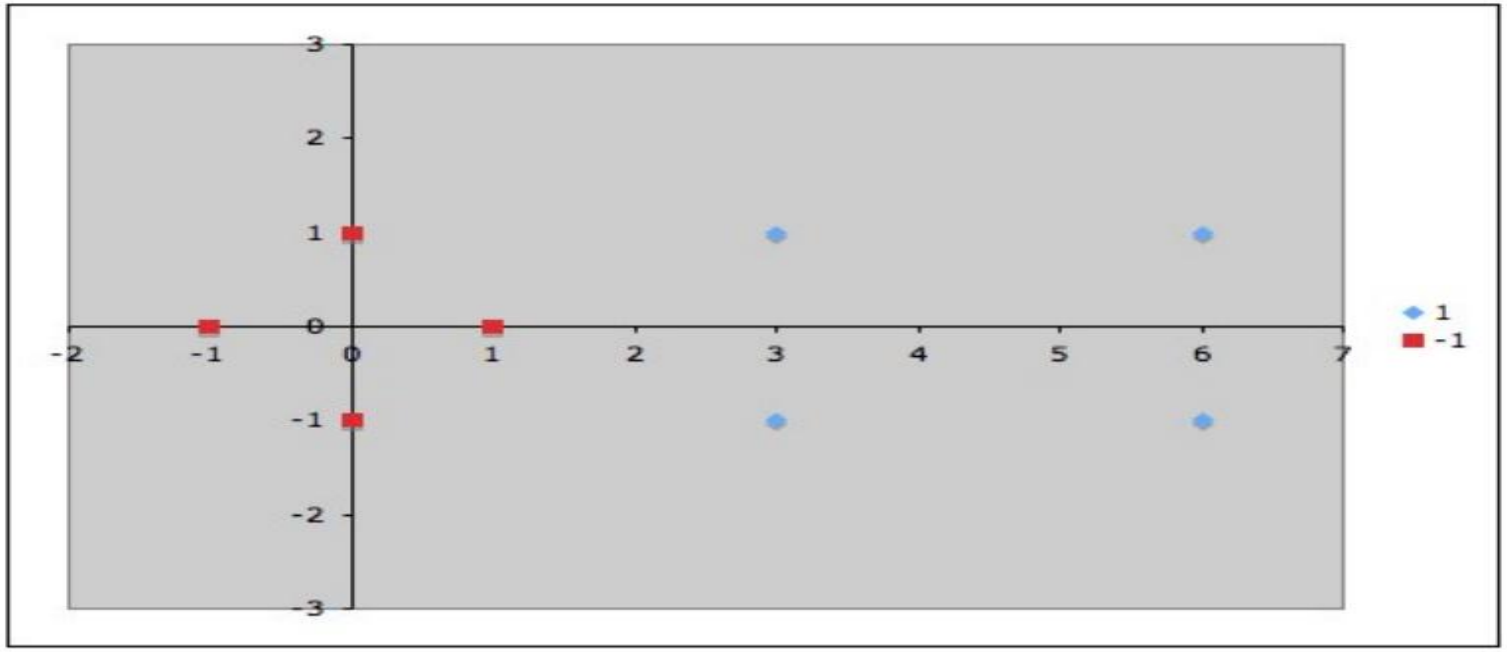


Figure 1: Sample data points in $\Re^2$. Blue diamonds are positive examples and red squares are negative examples.

Numerical Example of Linear SVM

We would like to discover a simple SVM that accurately discriminates the two classes. Since the data is linearly separable, we can use a linear SVM (that is, one whose mapping function $\Phi()$ is the identity function). By inspection, it should be obvious that there are three support vectors (see Figure 2):

$$\left\{ s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, s_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}, s_3 = \begin{pmatrix} 3 \\ -1 \end{pmatrix} \right\}$$

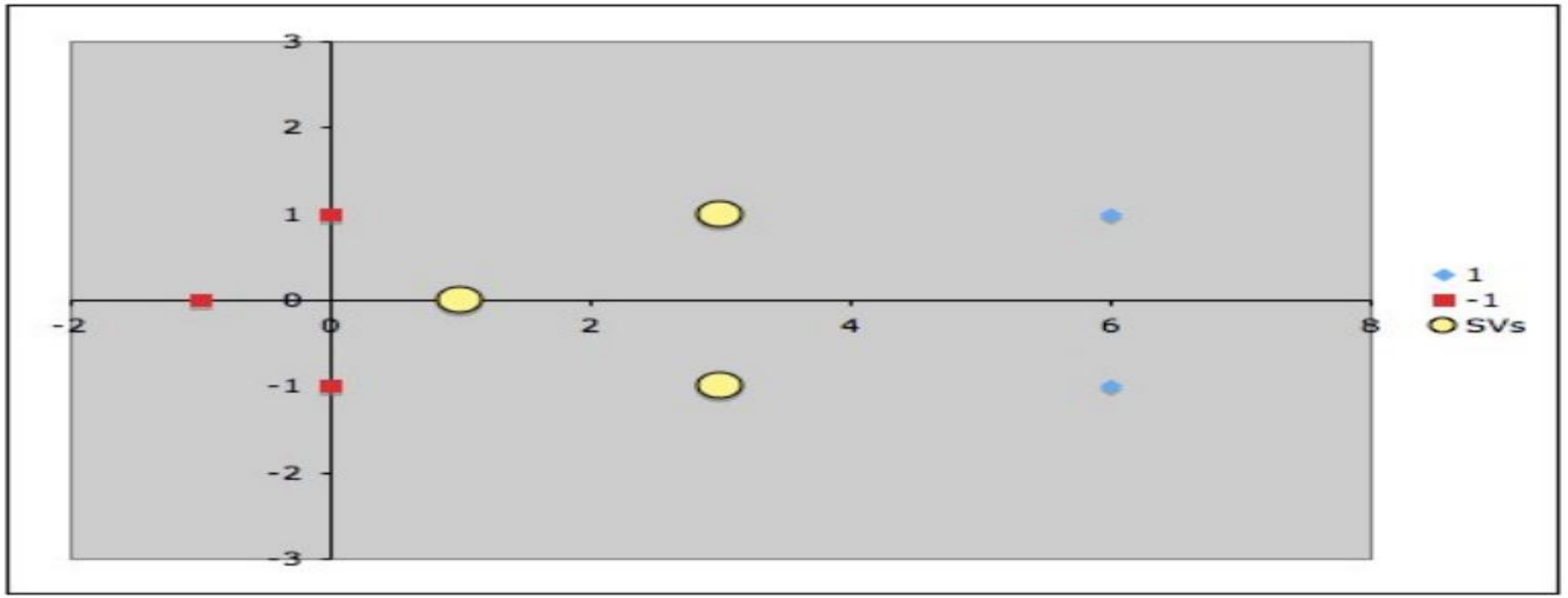


Figure 2: The three support vectors are marked as yellow circles.

Numerical Example of Linear SVM

- Each vector is augmented with a 1 as a bias input
- So, $s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, then $\tilde{s_1} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$
- Similarly,
- $s_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$, then $\tilde{s_2} = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$ and $s_3 = \begin{pmatrix} 3 \\ -1 \end{pmatrix}$, then $\tilde{s_3} = \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$

Numerical Example of Linear SVM

our task is to find values for the α_i such that

$$\begin{aligned}
\alpha_1\Phi(s_1)\cdot\Phi(s_1)+\alpha_2\Phi(s_2)\cdot\Phi(s_1)+\alpha_3\Phi(s_3)\cdot\Phi(s_1) &= -1\\
\alpha_1\Phi(s_1)\cdot\Phi(s_2)+\alpha_2\Phi(s_2)\cdot\Phi(s_2)+\alpha_3\Phi(s_3)\cdot\Phi(s_2) &= +1\\
\alpha_1\Phi(s_1)\cdot\Phi(s_3)+\alpha_2\Phi(s_2)\cdot\Phi(s_3)+\alpha_3\Phi(s_3)\cdot\Phi(s_3) &= +1
\end{aligned}$$

Since for now we have let $\Phi() = I$, this reduces to

$$\begin{aligned}
\alpha_1\tilde{s_1}\cdot\tilde{s_1}+\alpha_2\tilde{s_2}\cdot\tilde{s_1}+\alpha_3\tilde{s_3}\cdot\tilde{s_1} &= -1\\
\alpha_1\tilde{s_1}\cdot\tilde{s_2}+\alpha_2\tilde{s_2}\cdot\tilde{s_2}+\alpha_3\tilde{s_3}\cdot\tilde{s_2} &= +1\\
\alpha_1\tilde{s_1}\cdot\tilde{s_3}+\alpha_2\tilde{s_2}\cdot\tilde{s_3}+\alpha_3\tilde{s_3}\cdot\tilde{s_3} &= +1
\end{aligned}$$

Numerical Example of Linear SVM

$$\alpha_1 \tilde{s}_1 \cdot \tilde{s}_1 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_1 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_1 = -1 \qquad \alpha_1(1+0+1)+\alpha_2(3+0+1)+\alpha_3(3+0+1) = -1$$

$$\alpha_1 \tilde{s}_1 \cdot \tilde{s}_2 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_2 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_2 = +1 \qquad \alpha_1(3+0+1)+\alpha_2(9+1+1)+\alpha_3(9-1+1) = 1$$

$$\alpha_1 \tilde{s}_1 \cdot \tilde{s}_3 + \alpha_2 \tilde{s}_2 \cdot \tilde{s}_3 + \alpha_3 \tilde{s}_3 \cdot \tilde{s}_3 = +1 \qquad \alpha_1(3+0+1)+\alpha_2(9-1+1)+\alpha_3(9+1+1) = 1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} = -1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}3\\1\\1\end{pmatrix} = 1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}3\\1\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}3\\-1\\1\end{pmatrix}\begin{pmatrix}3\\-1\\1\end{pmatrix} = 1$$

$$2\alpha_1 + 4\alpha_2 + 4\alpha_3 = -1$$

$$4\alpha_1 + 11\alpha_2 + 9\alpha_3 = 1$$

$$4\alpha_1 + 9\alpha_2 + 11\alpha_3 = 1$$

$$\alpha_1 = -3.5$$

$$\alpha_2 = 0.75$$

$$\alpha_3 = 0.75$$

Numerical Example of Linear SVM

$$\begin{aligned}\tilde{w} &= \sum_i \alpha_i \tilde{s}_i \\ &= -3.5 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix}\end{aligned}$$

Finally, remembering that our vectors are augmented with a bias, we can equate the last entry in $\tilde{w}$ as the hyperplane offset b and write the separating hyperplane equation $y = wx + b$ with $w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $b = -2$. Plotting the line gives the expected decision surface (see Figure 4).

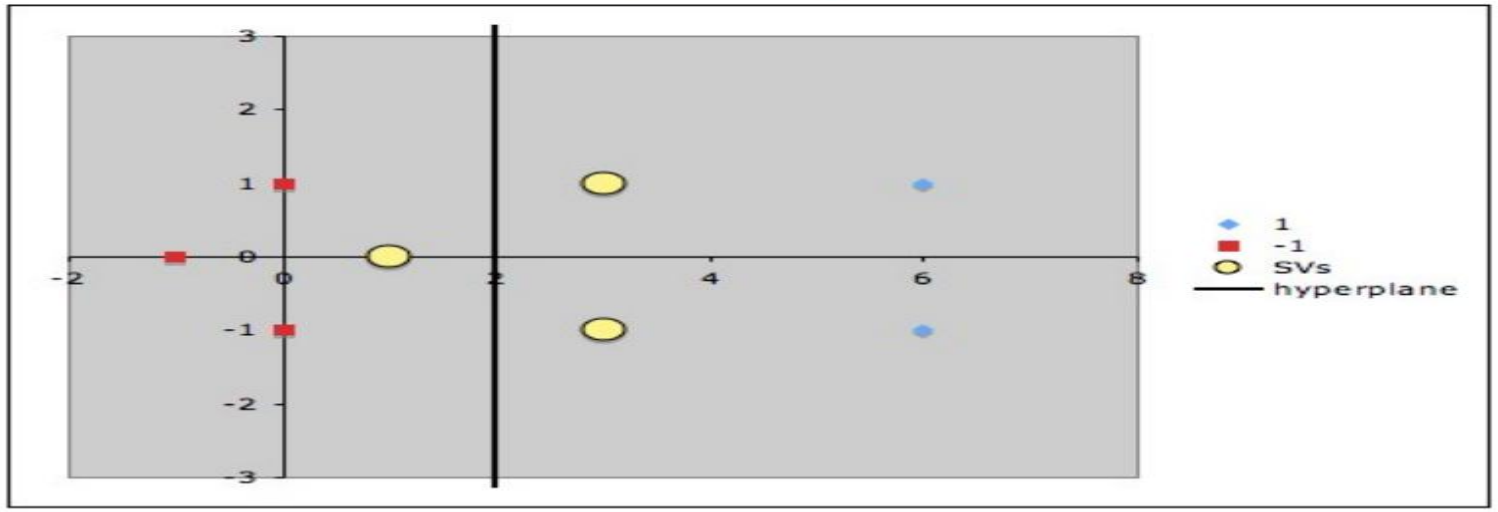
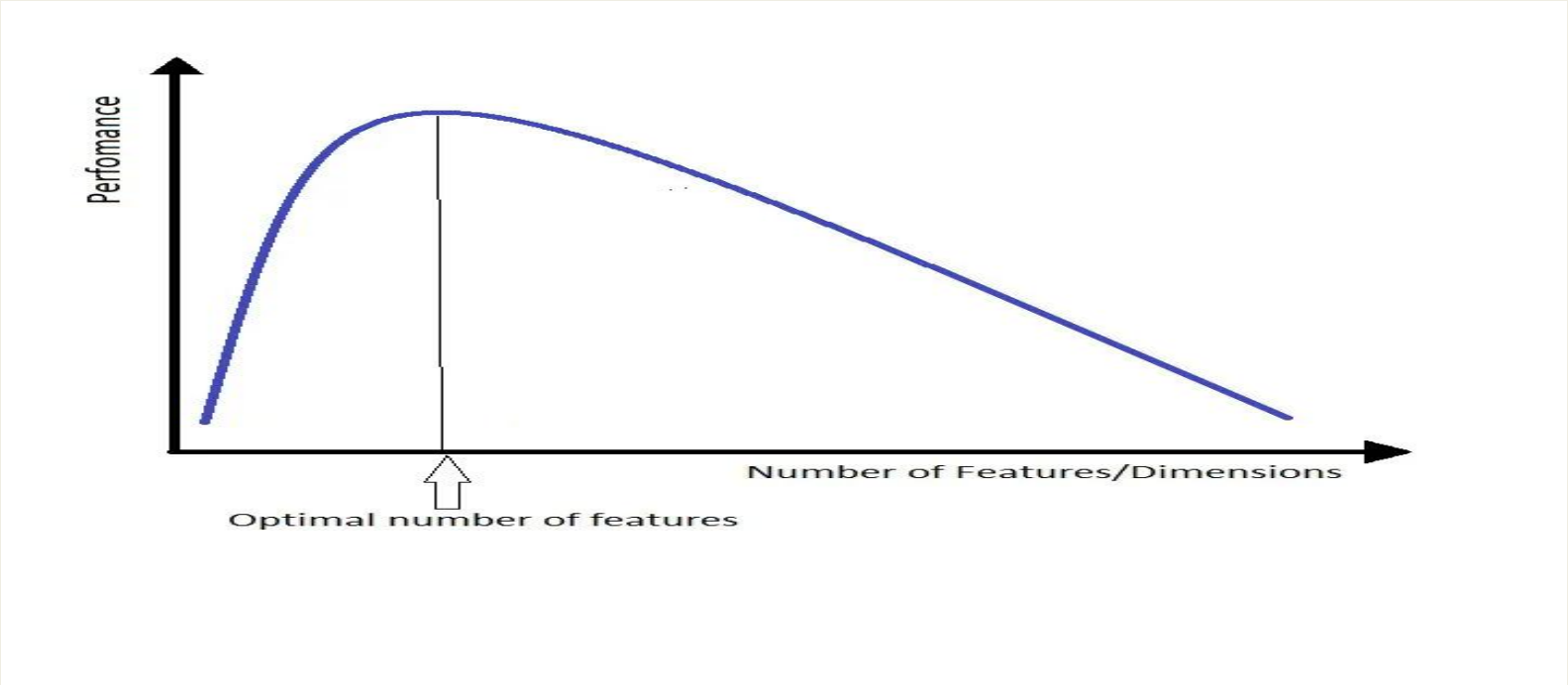


Figure 4: The discriminating hyperplane corresponding to the values $\alpha_1 = -3.5, \alpha_2 = 0.75$ and $\alpha_3 = 0.75$.

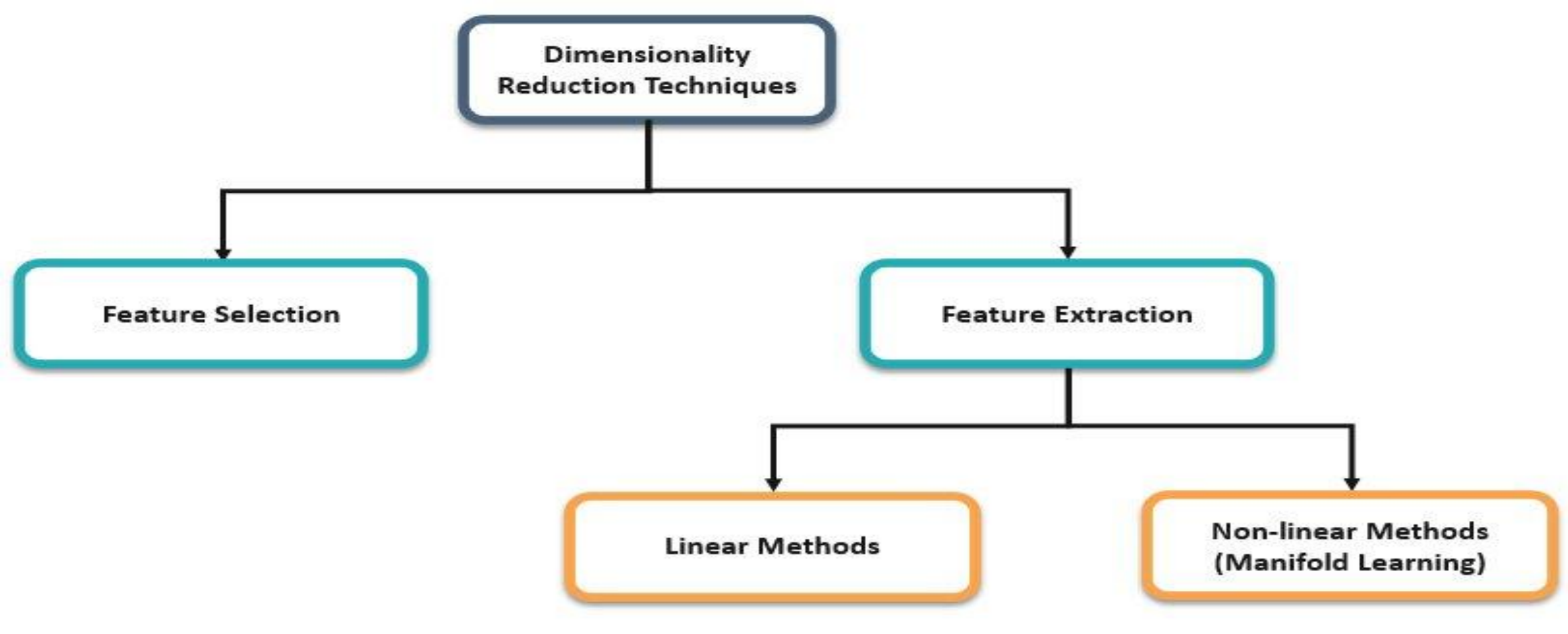
Curse of Dimensionality and Need of Dimension Reduction

- As the **dimensionality increases, the number of data points required for good performance of any machine learning algorithm increases exponentially.**
- The reason is that, we would need more number of data points for any given combination of features, for any machine learning model to be valid.
- Hughes (1968) in his study concluded that **with a fixed number of training samples, the predictive power of any classifier first increases as the number of dimensions increase, but after a certain value of number of dimensions, the performance deteriorates.**
- Thus, the phenomenon of **curse of dimensionality is also known as Hughes phenomenon.**



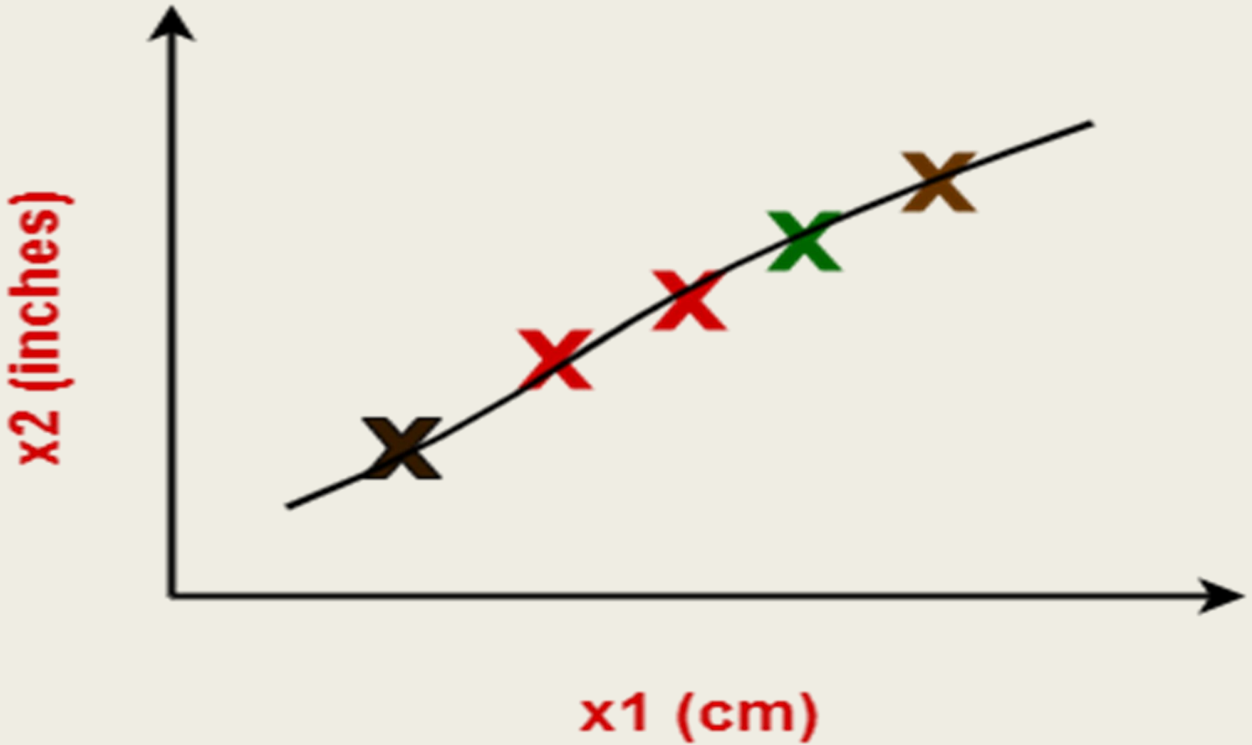
Solution to Curse of Dimensionality

- Several techniques can be employed for dimensionality reduction depending on the problem and the data.
- These techniques are divided into two broad categories:
- **Feature Selection: Choosing the most important features from the data**
- **Feature Extraction: Combining features to create new superfeatures.**



Dimension Reduction

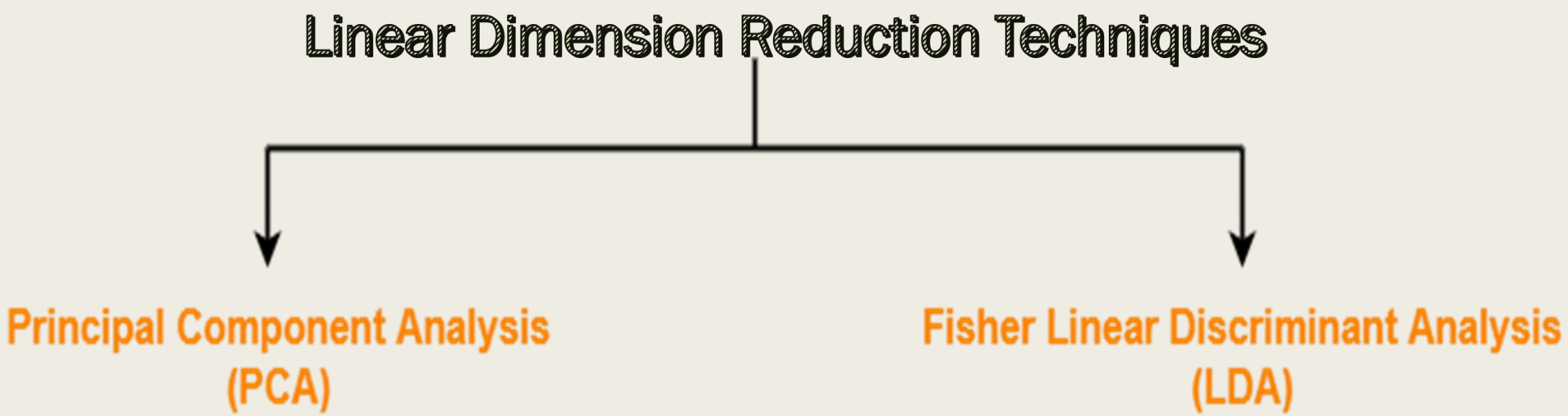
- **It is a process of converting a data set having vast dimensions into a data set with lesser dimensions.**
- It ensures that the converted data set conveys similar information concisely.
- Example-
- The following graph shows two dimensions x1 and x2.
- x1 represents the measurement of several objects in cm.
- x2 represents the measurement of several objects in inches.



Dimension Reduction

- Using both these dimensions convey similar information.
- Also, they introduce a lot of noise in the system.
- So, it is better to use just one dimension.
- Using dimension reduction techniques-
- We convert the dimensions of data from 2 dimensions (x1 and x2) to 1 dimension (z1).
- It makes the data relatively easier to explain.

- **Benefits-**
- It **compresses the data** and thus **reduces the storage space requirements.**
- It **reduces the time required for computation** since less dimensions require less computation.
- It **eliminates the redundant features.**
- It **improves the model performance.**



- **Principal Component Analysis-**
- Principal Component Analysis is a well-known **dimension reduction technique.**
- It **transforms the variables into a new set of variables called as principal components.**
- These principal components are **linear combination of original variables and are orthogonal.**
- The **first principal component accounts for most of the possible variation of original data.**
- The **second principal component does its best to capture the variance in the data.**
- There **can be only two principal components for a two-dimensional data set.**

PCA Algorithm

- **The steps involved in PCA Algorithm are as follows-**
- Step-01: Get data.
- Step-02: Compute the mean vector (μ).
- Step-03: Subtract mean from the given data.
- Step-04: Calculate the covariance matrix.
- Step-05: Calculate the eigen vectors and eigen values of the covariance matrix.
- Step-06: Choosing components and forming a feature vector.
- Step-07: Deriving the new data set.

Example on PCA Algorithm

- **Problem-01:** Given data = { 2, 3, 4, 5, 6, 7; 1, 5, 3, 6, 7, 8 }.
- F1={ 2, 3, 4, 5, 6, 7}
- F2= {1, 5, 3, 6, 7, 8 }.
- Compute the principal component using PCA Algorithm.

OR

- Consider the two dimensional patterns (2, 1), (3, 5), (4, 3), (5, 6), (6, 7), (7, 8).
- Compute the principal component using PCA Algorithm.

OR

- Compute the principal component of following data-
- CLASS 1
- X = 2 , 3 , 4
- Y = 1 , 5 , 3
- CLASS 2
- X = 5 , 6 , 7
- Y = 6 , 7 , 8

OR

- Reduce the following dataset step by step from 2 dimension to 1 using PCA.

Feature	Example1	Example2	Example3	Example4	Example5	Example6
X	2	3	4	5	6	7
Y	1	5	3	6	7	8

Example on PCA Algorithm

- **Solution-**
- We use the above discussed PCA Algorithm-
- **Step-01:**
- **Get data.**
- The given feature vectors are-
- x1 = (2, 1)
- x2 = (3, 5)
- x3 = (4, 3)
- x4 = (5, 6)
- x5 = (6, 7)
- x6 = (7, 8)

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \begin{bmatrix} 3 \\ 5 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \end{bmatrix} \begin{bmatrix} 6 \\ 7 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix}$$

Example on PCA Algorithm

- **Step-02:**
- **Calculate the mean vector (μ).**
- Mean vector (μ) = ((2 + 3 + 4 + 5 + 6 + 7) / 6, (1 + 5 + 3 + 6 + 7 + 8) / 6) = (4.5, 5)
- Thus, Mean vector (μ) = $\begin{bmatrix} 4.5 \\ 5 \end{bmatrix}$

- **Step-03:**
- **Subtract mean vector (μ) from the given feature vectors.**
- x1 – μ = (2 – 4.5, 1 – 5) = (-2.5, -4)
- x2 – μ = (3 – 4.5, 5 – 5) = (-1.5, 0)
- x3 – μ = (4 – 4.5, 3 – 5) = (-0.5, -2)
- x4 – μ = (5 – 4.5, 6 – 5) = (0.5, 1)
- x5 – μ = (6 – 4.5, 7 – 5) = (1.5, 2)
- x6 – μ = (7 – 4.5, 8 – 5) = (2.5, 3)
- Feature vectors (xi) after subtracting mean vector (μ) are- $\begin{bmatrix} -2.5 \\ -4 \end{bmatrix}$ $\begin{bmatrix} -1.5 \\ 0 \end{bmatrix}$ $\begin{bmatrix} -0.5 \\ -2 \end{bmatrix}$ $\begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$ $\begin{bmatrix} 1.5 \\ 2 \end{bmatrix}$ $\begin{bmatrix} 2.5 \\ 3 \end{bmatrix}$

Example on PCA Algorithm

- **Step-04:**
- **Calculate the covariance matrix.**
- Covariance matrix is given by-

$$\text{Covariance Matrix} = \frac{\sum (x_i - \mu)(x_i - \mu)^t}{n}$$

$$m_1 = (x_1 - \mu)(x_1 - \mu)^t = \begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \begin{bmatrix} -2.5 & -4 \end{bmatrix} = \begin{bmatrix} 6.25 & 10 \\ 10 & 16 \end{bmatrix}$$

$$m_2 = (x_2 - \mu)(x_2 - \mu)^t = \begin{bmatrix} -1.5 \\ 0 \end{bmatrix} \begin{bmatrix} -1.5 & 0 \end{bmatrix} = \begin{bmatrix} 2.25 & 0 \\ 0 & 0 \end{bmatrix}$$

$$m_3 = (x_3 - \mu)(x_3 - \mu)^t = \begin{bmatrix} -0.5 \\ -2 \end{bmatrix} \begin{bmatrix} -0.5 & -2 \end{bmatrix} = \begin{bmatrix} 0.25 & 1 \\ 1 & 4 \end{bmatrix}$$

$$m_4 = (x_4 - \mu)(x_4 - \mu)^t = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \begin{bmatrix} 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$m_5 = (x_5 - \mu)(x_5 - \mu)^t = \begin{bmatrix} 1.5 \\ 2 \end{bmatrix} \begin{bmatrix} 1.5 & 2 \end{bmatrix} = \begin{bmatrix} 2.25 & 3 \\ 3 & 4 \end{bmatrix}$$

$$m_6 = (x_6 - \mu)(x_6 - \mu)^t = \begin{bmatrix} 2.5 \\ 3 \end{bmatrix} \begin{bmatrix} 2.5 & 3 \end{bmatrix} = \begin{bmatrix} 6.25 & 7.5 \\ 7.5 & 9 \end{bmatrix}$$

Example on PCA Algorithm

- Covariance matrix = (m1 + m2 + m3 + m4 + m5 + m6) / 6

$$\text{Covariance Matrix} = \frac{1}{6}\begin{bmatrix} 17.5 & 22 \\ 22 & 34 \end{bmatrix}$$

$$\text{Covariance Matrix} = \begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix}$$

- **Step-05:**
- **Calculate the eigen values and eigen vectors of the covariance matrix.**
- **λ is an eigen value for a matrix M if it is a solution of the characteristic equation $|M - \lambda I| = 0$.**

$$\begin{vmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{vmatrix} - \begin{vmatrix} \lambda & 0 \\ 0 & \lambda \end{vmatrix} = 0$$

$$\begin{vmatrix} 2.92 - \lambda & 3.67 \\ 3.67 & 5.67 - \lambda \end{vmatrix} = 0$$

Example on PCA Algorithm

- $(2.92 - \lambda)(5.67 - \lambda) - (3.67 \times 3.67) = 0$
- $16.56 - 2.92\lambda - 5.67\lambda + \lambda 2 - 13.47 = 0$
- $\lambda^2 - 8.59\lambda + 3.09 = 0$
- Solving this quadratic equation, we get $\lambda = 8.22, 0.38$
- Thus, **two eigen values are $\lambda 1 = 8.22$ and $\lambda 2 = 0.38$.**
- Clearly, the **second eigen value is very small compared to the first eigen value.**
- So, the second eigen vector can be left out.
- **Eigen vector corresponds to the greatest eigen value is principal component for the given data set.**
- **So. we find the eigen vector corresponding to eigen value $\lambda 1$.**
- We use the following equation to find the eigen vector-
- **$MX = \lambda X$**
- where-
- M = Covariance Matrix
- X = Eigen vector
- λ = Eigen value

Example on PCA Algorithm

- Substituting the values in the above equation, we get-

$$\begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} = 8.22 \begin{bmatrix} X1 \\ X2 \end{bmatrix}$$

- Solving these, we get-
- $2.92X1 + 3.67X2 = 8.22X1$
- $3.67X1 + 5.67X2 = 8.22X2$
- On simplification, we get-
- $5.3X1 = 3.67X2$ ……(1)
- $3.67X1 = 2.55X2$ ……(2)
- $X1/2.55 = X2/3.67$
- From (1) and (2), $X1 = 0.69X2$
- From (2), the eigen vector is-

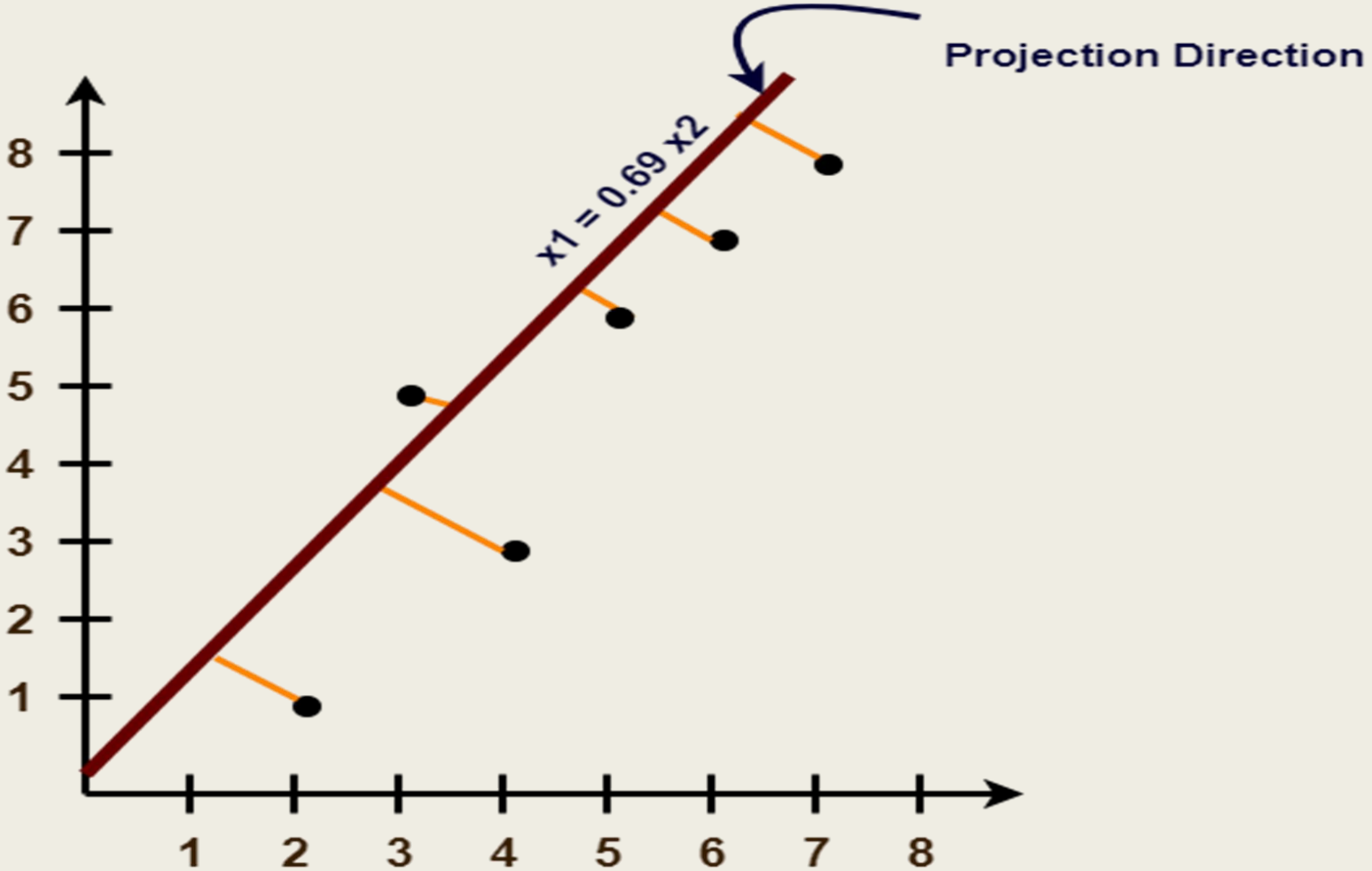
$$\text{Eigen Vector :} \quad \begin{bmatrix} X1 \\ X2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$$

- Thus, principal component for the given data set is-

$$\text{Principal Component :} \quad \begin{bmatrix} X1 \\ X2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$$

Example on PCA Algorithm

- Lastly, we project the data points onto the new subspace as-



Example on PCA Algorithm

- **Problem-02:**
- Use PCA Algorithm to transform the pattern (2, 1) onto the eigen vector in the previous question.
- **Solution-**
- The given feature vector is (2, 1).

$$\text{Given Feature Vector :} \quad \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

- The feature vector gets transformed to = Transpose of Eigen vector x (Feature Vector – Mean Vector)

$$= \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}^T \times \left(\begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 4.5 \\ 5 \end{bmatrix} \right)$$

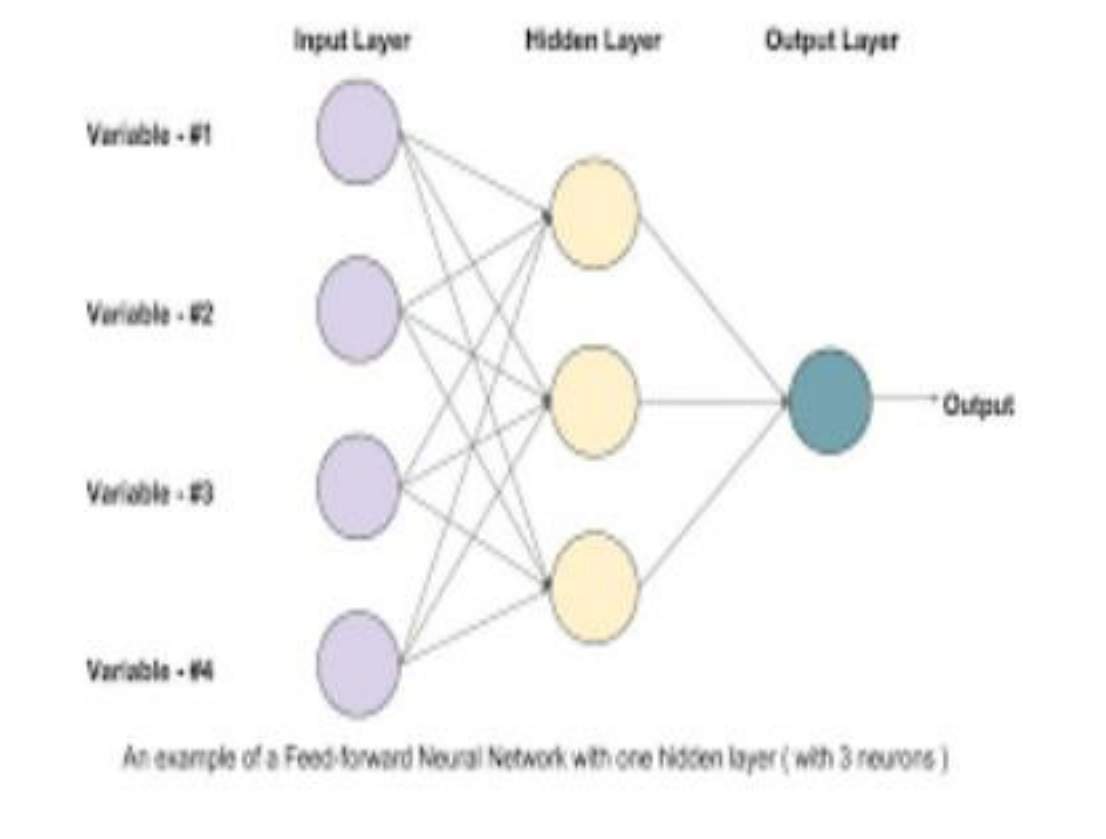
$$= \begin{bmatrix} 2.55 & 3.67 \end{bmatrix} \times \begin{bmatrix} -2.5 \\ -4 \end{bmatrix}$$

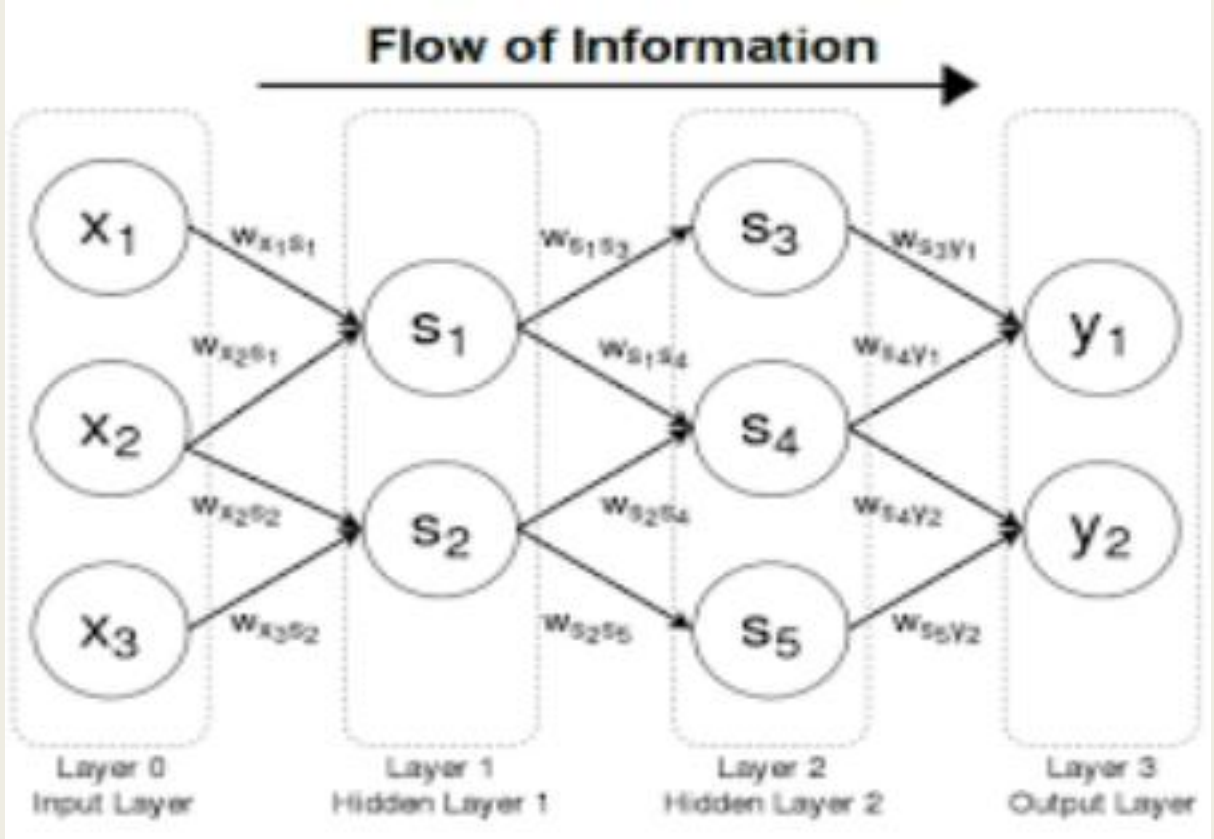
$$= -21.055$$

What is Feed-forward Neural Networks?

- A feedforward neural network is a type of artificial neural network in which **nodes' connections do not form a loop.**
- Often referred to as a multi-layered network of neurons, feedforward neural networks are so named because **all information flows in a forward manner only.**
- The **data enters the input nodes, travels through the hidden layers, and eventually exits the output nodes.**
- The network is devoid of links that would allow the information exiting the output node to be sent back into the network.
- The **purpose of feedforward neural networks is to approximate functions.**
- There is a **classifier using the formula y = f* (x).**
- **This assigns the value of input x to the category y.**
- **The feedforward network will map y = f (x; θ). It then memorizes the value of θ that most closely approximates the function.**

Diagram of Feed-forward Neural Networks?





Components of Feedforward Neural Network

- **Input layer**
- It **contains the neurons that receive input.**
- The data is subsequently passed on to the next layer.
- The **input layer's total number of neurons is equal to the number of variables in the dataset.**
- **Hidden layer**
- This is the **intermediate layer, which is concealed between the input and output layers.**
- This layer has a **large number of neurons that perform alterations on the inputs.**
- They then communicate with the output layer.
- **Output layer**
- It is the **last layer and is depending on the model's construction**.
- Additionally, **the output layer is the expected feature**, as you are aware of the desired outcome.
- **Neurons weights**
- Weights are used to describe the **strength of a connection between neurons**.
- The **range of a weight's value is from 0 to 1.**

Cost Function in Feedforward Neural Network

- The cost function is an important factor of a feedforward neural network.
- Generally, minor adjustments to weights and biases have little effect on the categorized data points.
- Thus, to determine a method for improving performance by making minor adjustments to weights and biases using a smooth cost function.
- The mean square error cost function is defined as follows:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

- Where,
- w = weights collected in the network
- b = biases
- a = output vectors
- x = input
- ‖v‖ = usual length of vector v.

Loss Function in Feedforward Neural Network

Cross Entropy Loss:

$$L(\Theta) = \begin{cases} -log(\hat{y}) & \text{if } y = 1 \\ -log(1-\hat{y}) & \text{if } y = 0 \end{cases}$$

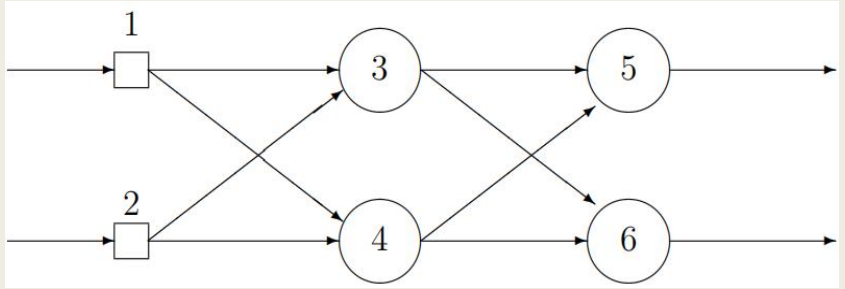
- The cross-entropy loss associated with multi-class categorization is as follows:

Cross Entropy Loss:

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

Numerical on Feedforward Neural Network

- The following diagram represents a feed-forward neural network with one hidden layer:



- A weight on connection between nodes i and j is denoted by w_{ij} , such as w_{13} is the weight on the connection between nodes 1 and 3. The following table lists all the weights in the network:

$w_{13} = -2$	$w_{35} = 1$
$w_{23} = 3$	$w_{45} = -1$
$w_{14} = 4$	$w_{36} = -1$
$w_{24} = -1$	$w_{46} = 1$

- Each of the nodes 3, 4, 5 and 6 uses the following activation function:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- where v denotes the weighted sum of a node. Each of the input nodes (1 and 2) can only receive binary values (either 0 or 1). Calculate the output of the network (y_5 and y_6) for each of the input patterns:

Pattern:	P_1	P_2	P_3	P_4
Node 1:	0	1	0	1
Node 2:	0	0	1	1

Numerical on Feedforward Neural Network

- **Answer:**
- In order to find the output of the network it is necessary to calculate weighted sums of hidden nodes 3 and 4:
- $v_3 = w_{13}x_1 + w_{23}x_2$, $v_4 = w_{14}x_1 + w_{24}x_2$
- Then find the outputs from hidden nodes using activation function ϕ:
- $y_3 = \phi(v_3)$, $y_4 = \phi(v_4)$.
- Use the outputs of the hidden nodes y_3 and y_4 as the input values to the output layer (nodes 5 and 6), and find weighted sums of output nodes 5 and 6:
- $v_5 = w_{35}y_3 + w_{45}y_4$, $v_6 = w_{36}y_3 + w_{46}y_4$
- Finally, find the outputs from nodes 5 and 6 (also using ϕ):
- $y_5 = \phi(v_5)$, $y_6 = \phi(v_6)$

Numerical on Feedforward Neural Network

P_1: *Input pattern* $(0, 0)$

$$v_3 = -2 \cdot 0 + 3 \cdot 0 = 0, \qquad y_3 = \varphi(0) = 1$$
$$v_4 = 4 \cdot 0 - 1 \cdot 0 = 0, \qquad y_4 = \varphi(0) = 1$$
$$v_5 = 1 \cdot 1 - 1 \cdot 1 = 0, \qquad y_5 = \varphi(0) = 1$$
$$v_6 = -1 \cdot 1 + 1 \cdot 1 = 0, \qquad y_6 = \varphi(0) = 1$$

The output of the network is $(1, 1)$.

P_2: *Input pattern* $(1, 0)$

$$v_3 = -2 \cdot 1 + 3 \cdot 0 = -2, \qquad y_3 = \varphi(-2) = 0$$
$$v_4 = 4 \cdot 1 - 1 \cdot 0 = 4, \qquad y_4 = \varphi(4) = 1$$
$$v_5 = 1 \cdot 0 - 1 \cdot 1 = -1, \qquad y_5 = \varphi(-1) = 0$$
$$v_6 = -1 \cdot 0 + 1 \cdot 1 = 1, \qquad y_6 = \varphi(1) = 1$$

The output of the network is $(0, 1)$.

Numerical on Feedforward Neural Network

P_3: *Input pattern* $(0, 1)$

$$v_3 = -2 \cdot 0 + 3 \cdot 1 = 3, \qquad y_3 = \varphi(3) = 1$$
$$v_4 = 4 \cdot 0 - 1 \cdot 1 = -1, \qquad y_4 = \varphi(-1) = 0$$
$$v_5 = 1 \cdot 1 - 1 \cdot 0 = 1, \qquad y_5 = \varphi(1) = 1$$
$$v_6 = -1 \cdot 1 + 1 \cdot 0 = -1, \qquad y_6 = \varphi(-1) = 0$$

The output of the network is $(1, 0)$.

P_4: *Input pattern* $(1, 1)$

$$v_3 = -2 \cdot 1 + 3 \cdot 1 = 1, \qquad y_3 = \varphi(1) = 1$$
$$v_4 = 4 \cdot 1 - 1 \cdot 1 = 3, \qquad y_4 = \varphi(3) = 1$$
$$v_5 = 1 \cdot 1 - 1 \cdot 1 = 0, \qquad y_5 = \varphi(0) = 1$$
$$v_6 = -1 \cdot 1 + 1 \cdot 1 = 0, \qquad y_6 = \varphi(0) = 1$$

The output of the network is $(1, 1)$.

Activation Functions

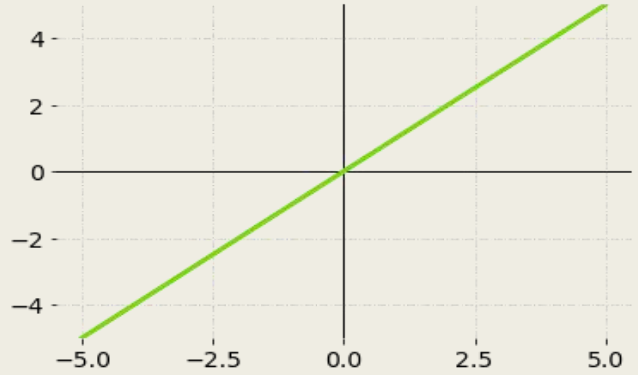
- In artificial neural networks, **each neuron forms a weighted sum of its inputs and passes the resulting scalar value through a function referred to as an activation function**.

- **Why do we need activation functions?**
- An activation function **determines if a neuron should be activated or not activated.**
- This implies that **it will use some simple mathematical operations to determine if the neuron's input to the network is relevant or not relevant in the prediction process.**
- The ability to **introduce non-linearity to an artificial neural network and generate output from a collection of input values fed to a layer** is the purpose of the activation function.

- **Types of Activation functions**
- Activation functions can be divided into three types:
- **Linear Activation Function**
- **Binary Step Function**
- **Non-linear Activation Functions**

Linear Activation Function

- The linear activation function, often called the **identity activation function**, is **proportional to the input.**
- The range of the linear activation function will be (-∞ to ∞).
- The linear activation function simply adds up the weighted total of the inputs and returns the result.



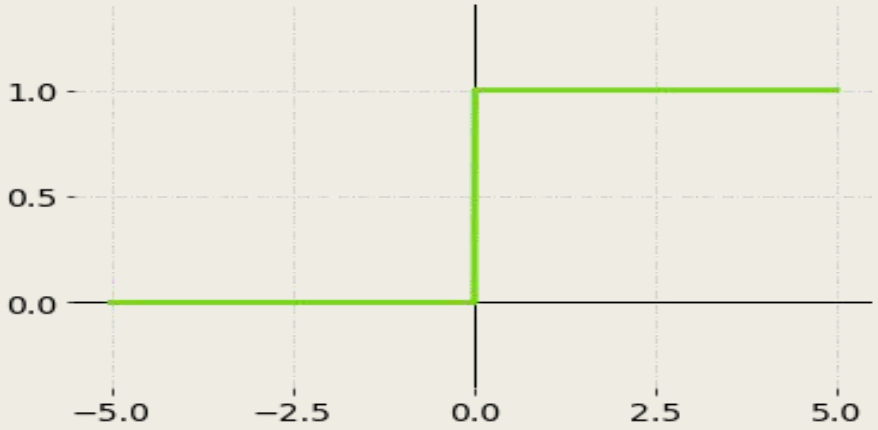
- Mathematically, it can be represented as:

$$f(x) = x$$

- **<u>Pros and Cons</u>**
- It is not a binary activation because the linear activation function only delivers a range of activations.
- We can surely connect a few neurons together, and if there are multiple activations, we can calculate the max (or soft max) based on that.
- The derivative of this activation function is a constant. That is to say, the gradient is unrelated to the x (input).

Binary Step Activation Function

- A **threshold value determines whether a neuron should be activated or not activated** in a binary step activation function.
- The activation function compares the input value to a threshold value.
- If the input value is greater than the threshold value, the neuron is activated.
- It's disabled if the input value is less than the threshold value, which means its output isn't sent on to the next or hidden layer.



- Mathematically, the binary activation function can be represented as:

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

- **Pros and Cons**
- It cannot provide multi-value outputs—for example, it cannot be used for multi-class classification problems.
- The step function's gradient is zero, which makes the back propagation procedure difficult.

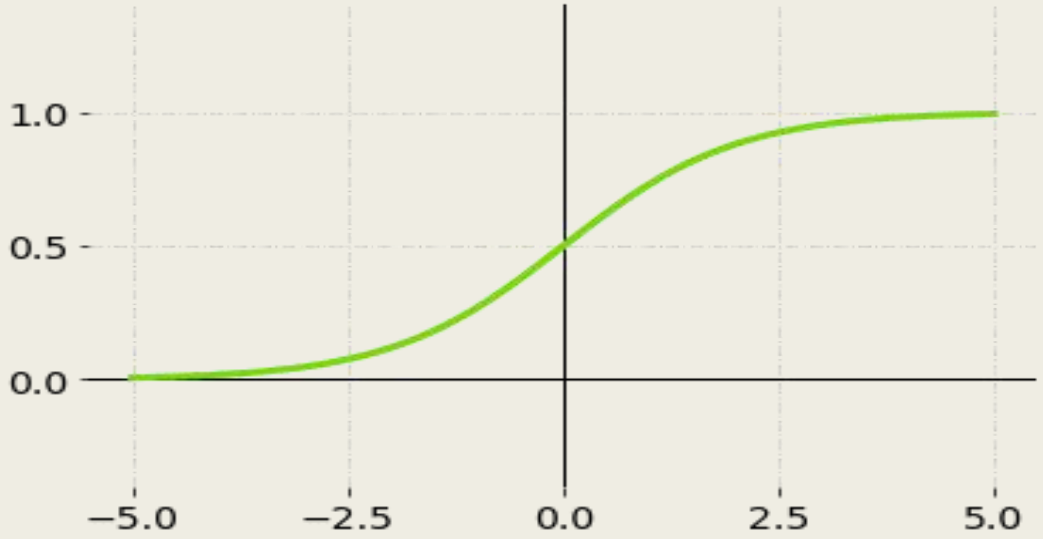
Non - linear Activation Functions

- The non-linear activation functions are the **most-used activation functions.**
- They make it uncomplicated for an artificial neural network model to adapt to a variety of data and to differentiate between the outputs.
- Non-linear activation functions **allow the stacking of multiple layers of neurons, as the output would now be a non-linear combination of input** passed through multiple layers.
- Any output can be represented as a functional computation output in a neural network.
- These activation functions are **mainly divided basis on their range and curves.**
- Some of the most used non-linear activation functions are:-

1. **Sigmoid**
2. **TanH**
3. **ReLU**
4. **Leaky ReLU**
5. **ELU**
6. **Softmax**
7. **Swish**

Sigmoid (Logistic Activation Function)

- Sigmoid **accepts a number as input and returns a number between 0 and 1.**
- It's simple to use and has all the desirable qualities of activation functions: nonlinearity, continuous differentiation, monotonicity, and a set output range.
- This is **mainly used in binary classification problems.**
- This sigmoid function gives the probability of an existence of a particular class.



- Mathematically, it can be represented as:

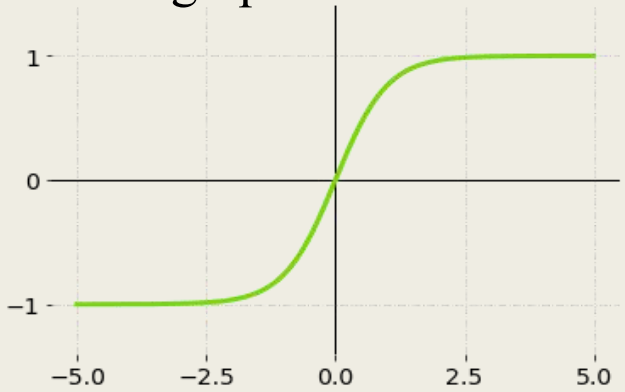
$$f(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid (Logistic Activation Function)

- **Pros and Cons**
- It is non-linear in nature.
- Combinations of this function are also non-linear, and it will give an analogue activation, unlike binary step activation function.
- It has a smooth gradient too, and It's good for a classifier type problem.
- The output of the activation function is always going to be in the range (0,1) compared to $(-\infty, \infty)$ of linear activation function. As a result, we've defined a range for our activations.
- Sigmoid function gives rise to a problem of "**Vanishing gradients**" and Sigmoids **saturate and kill gradients.**
- Its output isn't zero centred, and it makes the gradient updates go too far in different directions.
- The output value is between zero and one, so it makes optimization harder.
- The network either refuses to learn more or is extremely slow.

TanH (Hyperbolic Tangent)

- TanH compress a real-valued number to the range [-1, 1].
- It's non-linear, But it's different from Sigmoid,and its output is zero-centered.
- The main advantage of this is that the negative inputs will be mapped strongly to the negative and zero inputs will be mapped to almost zero in the graph of TanH.



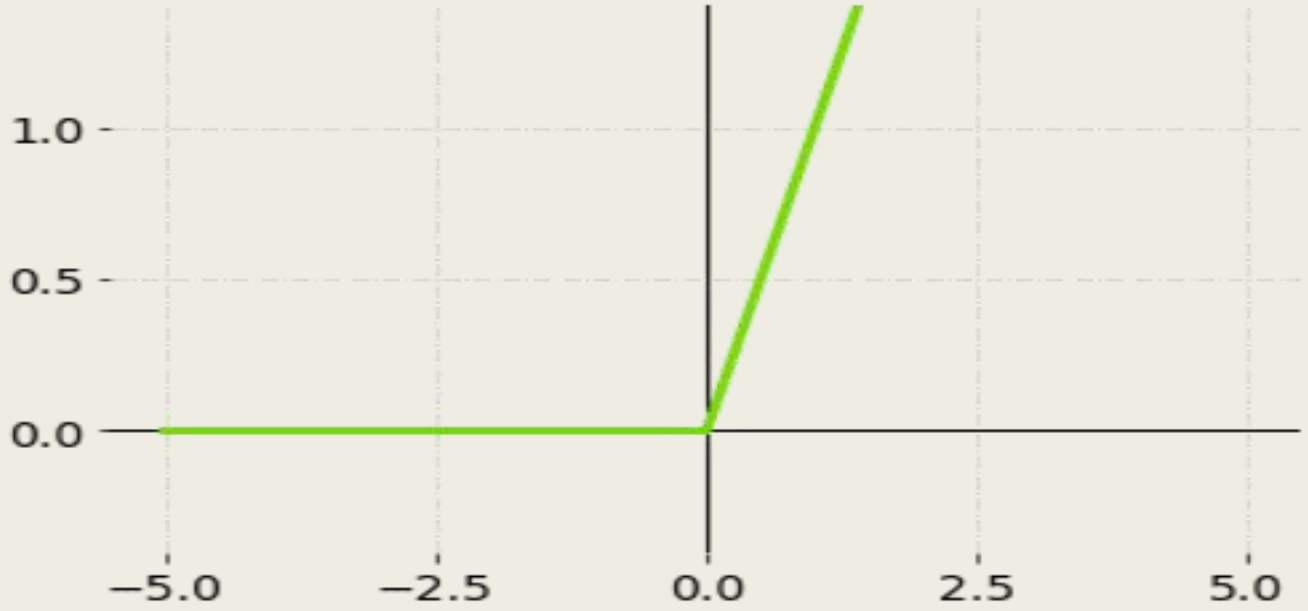
- Mathematically, TanH function can be represented as:

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

- **Pros and Cons**
- TanH also has the **vanishing gradient problem**, but the gradient is stronger for TanH than sigmoid (derivatives are steeper).
- TanH is zero-centered, and gradients do not have to move in a specific direction.

ReLU (Rectified Linear Unit)

- ReLU stands for Rectified Linear Unit and is one of the most commonly used activation function in the applications.
- It's solved the problem of vanishing gradient because the maximum value of the gradient of ReLU function is one.
- It also solved the problem of saturating neuron, since the slope is never zero for ReLU function. The range of ReLU is between 0 and infinity.



- Mathematically, it can be represented as:

$$f(x) = max\,(0, x)$$

ReLU (Rectified Linear Unit)

- **Pros and Cons**
- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and TanH functions.
- ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.
- One of its limitations is that it should only be used within hidden layers of an artificial neural network model.
- Some gradients can be fragile during training.
- In other words, For activations in the region ($x<0$) of ReLu, the gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons, which go into that state will stop responding to variations in input (simply because the gradient is 0, nothing changes.) This is called the **dying ReLu problem**.

Leaky ReLU

- Leaky ReLU is an upgraded version of the ReLU activation function to **solve the dying ReLU problem**, as it has a **small positive slope in the negative area.**
- But, the consistency of the benefit across tasks is presently ambiguous.

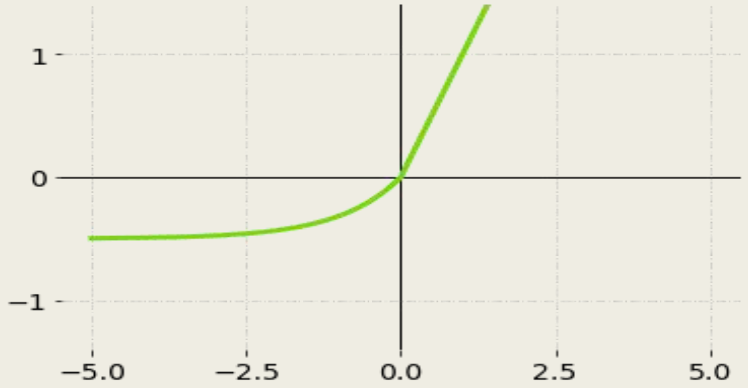
- Mathematically, it can be represented as,

$$f(x) = max\,(0.1x, x)$$

- **<u>Pros and Cons</u>**
- The advantages of Leaky ReLU are the same as that of ReLU, in addition to the fact that **it does enable back propagation, even for negative input values.**
- Making minor modification of negative input values, the gradient of the left side of the graph comes out to be a real (non-zero) value. As a result, there would be no more dead neurons in that area.
- The predictions may not be steady for negative input values.

ELU (Exponential Linear Units)

- ELU is also one of the variations of ReLU which also solves the dead ReLU problem.
- ELU, just like leaky ReLU also considers negative values by introducing a new alpha parameter and multiplying it will another equation.
- ELU is slightly more computationally expensive than leaky ReLU, and it's very similar to ReLU except negative inputs. They are both in identity function shape for positive inputs.



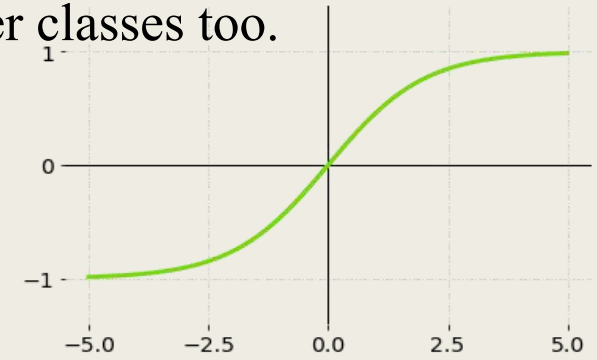
- Mathematically, it can be represented as:

$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha(e^x - 1) & for\ x < 0 \end{cases}$$

- **Pros and Cons**
- ELU is a strong alternative to ReLU. Different from the ReLU, ELU can produce negative outputs.
- Exponential operations are there in ELU, So it increases the computational time.
- No learning about the 'a' value takes place, and exploding gradient problem.

Softmax

- A **combination of many sigmoids** is referred to as the Softmax function. It determines **relative probability.** Similar to the sigmoid activation function, the Softmax returns the probability of each class.
- In **multi-class classification, softmax activation function is most commonly used for the last layer of the neural network.**
- The **softmax function gives the probability of the current class with respect to others**. This means that it also considers the possibility of other classes too.



- Mathematically, it can be represented as:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- **<u>Pros and Cons</u>**
- It mimics the one encoded label better than the absolute values.
- We would lose information if we used absolute (modulus) values, but the exponential takes care of this on its own.
- The softmax function **should be used for multi-label classification and regression task as well.**

Swish

- Swish **allows for the propagation of a few numbers of negative weights**, whereas ReLU sets all non-positive weights to zero.
- This is a crucial property that determines the success of non-monotonic smooth activation functions, such as Swish's, in progressively deep neural networks.
- It's a **self-gated activation function created by Google researchers.**

- Mathematically, it can be represented as:

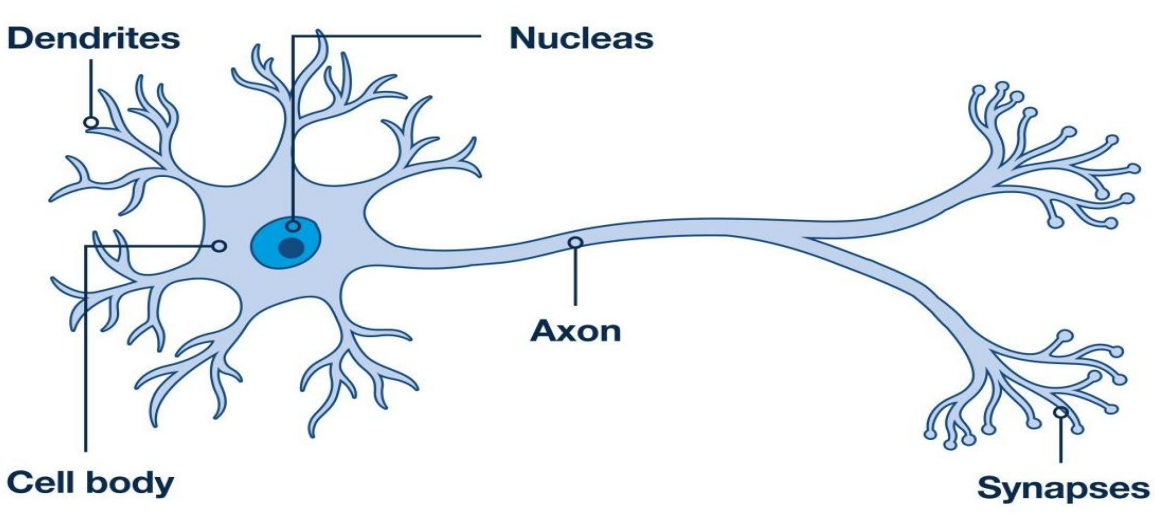
$$\sigma(x) = \frac{x}{1+e^{-x}}$$

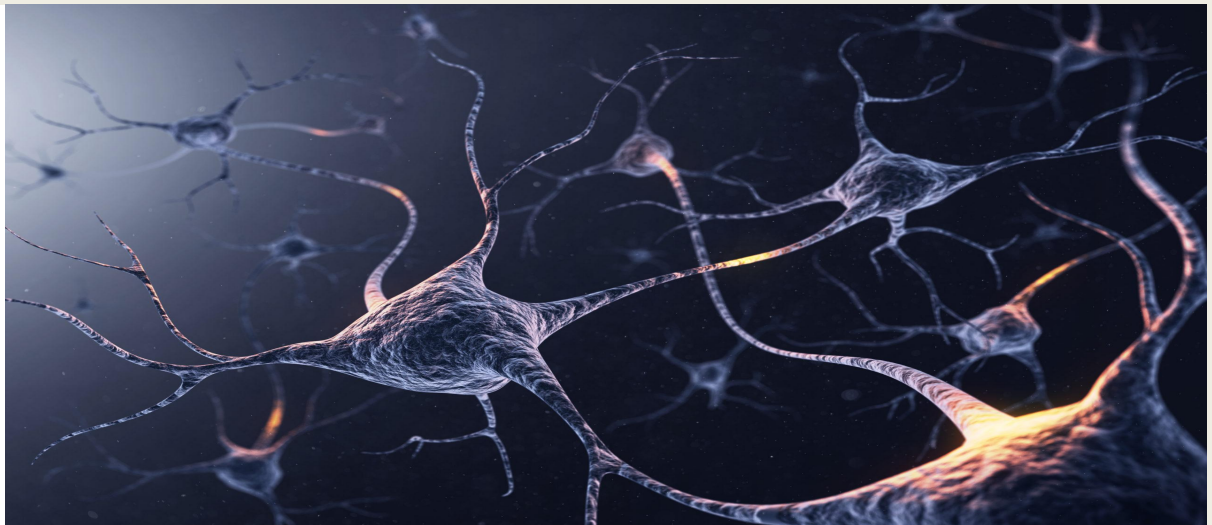
Swish

- **Pros and Cons**
- Swish is a smooth activation function that means that it **does not suddenly change direction like ReLU does near x equal to zero.**
- Rather, **it smoothly bends from 0 towards values < 0 and then upwards again.**
- Non-positive values were zeroed out in ReLU activation function.
- Negative numbers, on the other hand, may be valuable for detecting patterns in the data.
- Because of the sparsity, large negative numbers are wiped out, resulting in a win-win situation.
- The swish activation function being non-monotonous enhances the term of input data and weight to be learnt.
- Slightly more computationally expensive and more problems with the algorithm will probably arise given time.

Neural networks

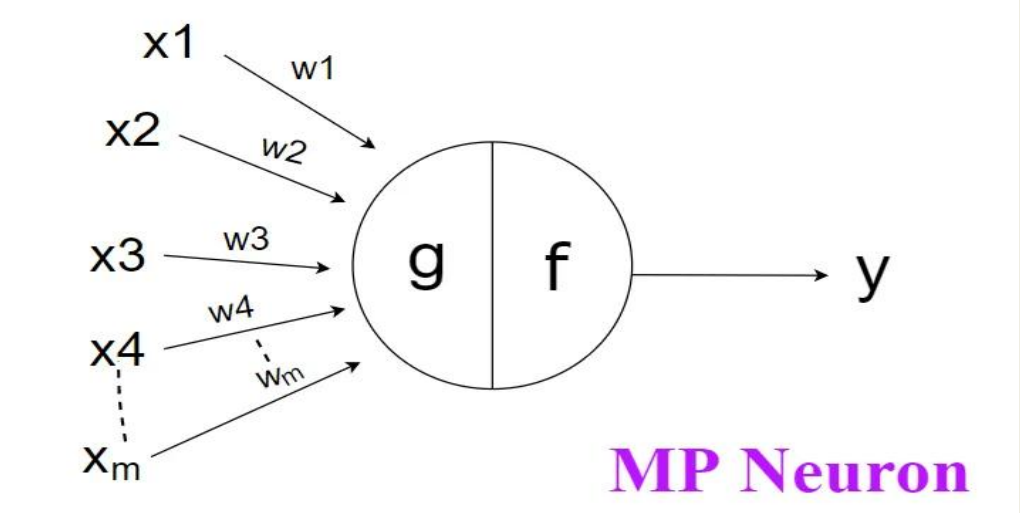
- An **Artificial Neural Network (ANN) models the relationship between a set of input signals and an output signal** using a model derived from our **understanding of how a biological brain responds to stimuli from sensory inputs.**
- Just as a brain uses a network of interconnected cells called neurons to create a massive parallel processor, **ANN uses a network of artificial neurons or nodes to solve learning problems.**
- **<u>Biological motivation</u>**
- In the cell, the **incoming signals are received by the cell's dendrites** through a biochemical process.
- The process allows the **impulse to be weighted according to its relative importance or frequency.**
- As the **cell body begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the axon.**
- At the axon's terminals, the **electric signal is again processed as a chemical signal to be passed to the neighboring neurons across a tiny gap known as a synapse.**





McCulloch - Pitts Neuron (MP Neuron)

- The MP neuron is **mankind's first simplified mathematical model of the neuron.**
- This model was **developed by McCulloch and Pitts in 1943.**
- The MP neuron model is also known as the **linear threshold gate model.**
- They are widely used in proving logic functions.



- Now let's look into the model. It has **4 basic components :**
- The model **takes inputs (x1, x2, …., xm)**
- **Applies Adder function(g)** and
- Takes **decision in Activation function (f)**
- **Gives an output Y**

Characteristics of MP Neuron

- When MP neurons are modeled as neural networks, they are connected by directed weighted paths in a neural network.
- When we pass the Adder function value in the Activation function of an MP neuron, there are 2 possibilities: the neuron may fire (label 1) or it does not fire (label 0).
- The activation function is based on the threshold value.
- There is a **fixed threshold for each neuron and if the net input to the neuron is greater than the threshold then the neuron fires.**
- **Weights(w):** It is the parameter that shows the **contributing power of the input feature towards the output.**
- Low weight value will have no change on the input and high weight will have a more significant change on the output.
- **Adder Function(g):** It is an Aggregation function that performs the **sum of the product of the inputs with the weights** and gets Adder value.
- **Activation Function(f):** It is the **mathematical function** that **decides whether neuron input is relevant for model prediction or not.**
- **Threshold value(b):** It is the value in the activation function based on which the activation function takes its decision.
- Initially, we take any value from 0 to n (where n is maximum adder value) and then iterate over it and find the total loss of the model.
- Then we can **choose the value of the threshold, such that the loss is minimum.** This is the brute force method by which we calculate the threshold value.

Characteristics of MP Neuron

$$g(x) = \sum_{i=1}^{m} w_i x_i$$

$$y = f(g(x)) = 1 \text{ if } g(x) \geq b$$

$$= 0 \text{ if } g(x) < b$$

- From the above image :
- **g(x) is the aggregated sum of all weighted inputs**
- **y is the final value predicted by an activation function (f).**
- **b is the threshold value which is calculated by the brute force method**

AND function using MP Neuron

- Consider the truth table for AND function
- The M–P neuron has no particular training algorithm
- In M-Pneuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = 1.

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

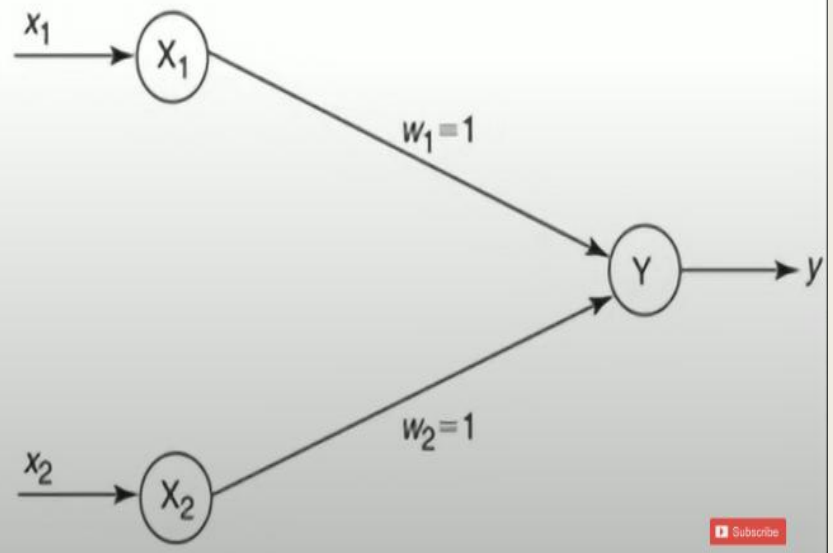
$$(1, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$$

$$(1, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1$$

$$(0, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 1 \times 1 = 1$$

$$(0, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0$$

Threshold value is set equal to 2 ($\theta = 2$).



ANDNOT function using MP Neuron

- Consider the truth table for ANDNOT function
- The M–P neuron has no particular training algorithm
- In M-P neuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = 1.

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

$$y_{in} = x_1 w_1 + x_2 w_2$$

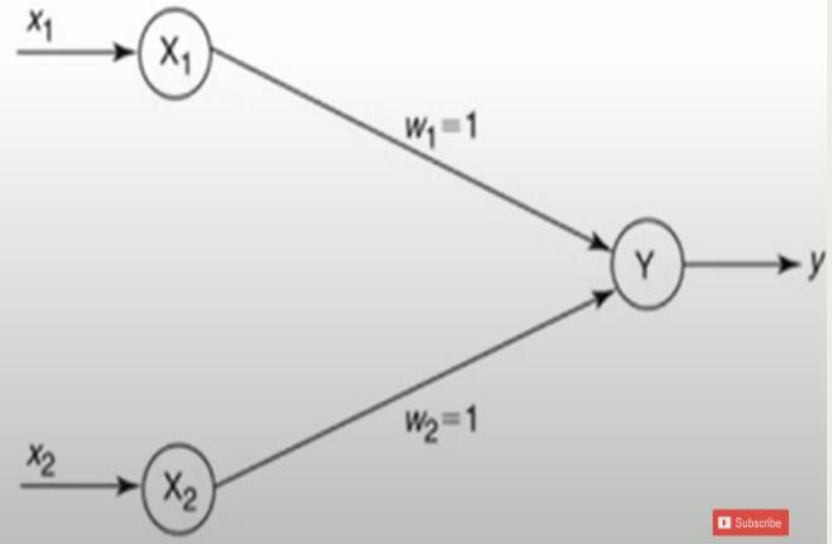
$(1, 1),\ y_{in} = 1 \times 1 + 1 \times 1 = 2$

$(1, 0),\ y_{in} = 1 \times 1 + 0 \times 1 = 1$

$(0, 1),\ y_{in} = 0 \times 1 + 1 \times 1 = 1$

$(0, 0),\ y_{in} = 0 \times 1 + 0 \times 1 = 0$

From the calculated net inputs, it is not possible to fire the neuron for input (1, 0) only. Hence, these weights are not suitable.



ANDNOT function using MP Neuron

- Consider the truth table for ANDNOT function
- The M–P neuron has no particular training algorithm
- In M-P neuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = -1.

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

$$y_{in} = x_1 w_1 + x_2 w_2$$

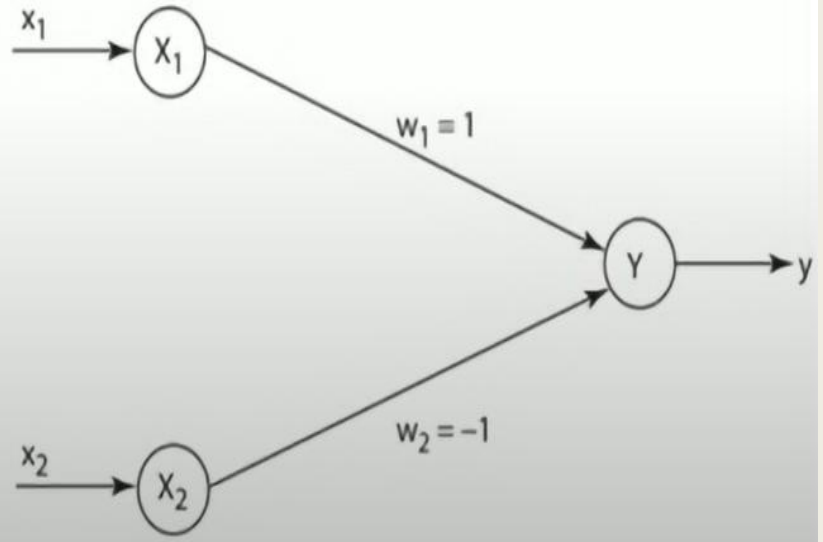
$(1, 1),\ y_{in} = 1 \times 1 + 1 \times -1 = 0$

$(1, 0),\ y_{in} = 1 \times 1 + 0 \times -1 = 1$

$(0, 1),\ y_{in} = 0 \times 1 + 1 \times -1 = -1$

$(0, 0),\ y_{in} = 0 \times 1 + 0 \times -1 = 0$

From the calculated net inputs, now it is possible to fire the neuron for input (1, 0) only by fixing a threshold of 1, i.e., $\theta \geq 1$ for Y unit.



Limitations and Solution of M-P Neuron

- **Limitations of MP Neuron**
- What about non-boolean (say, real) inputs?
- Do we always need to hand code the threshold?
- Are all inputs equal? What if we want to assign more importance to some inputs?
- What about functions which are not linearly separable? Say XOR function.

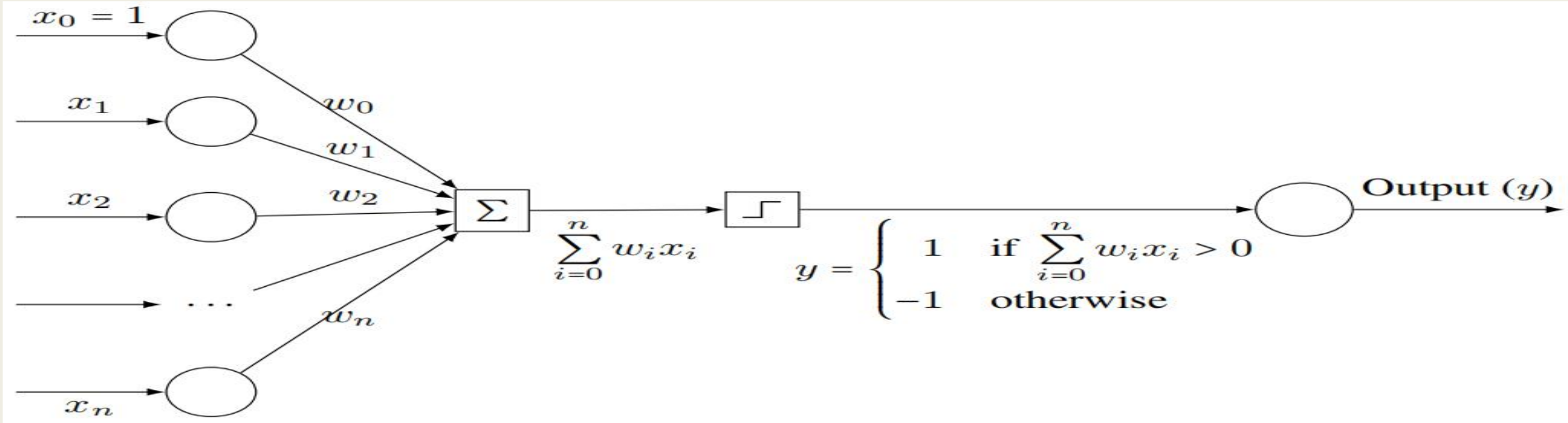
- **Solution for limitations of MP Neuron**
- Overcoming the limitations of the M-P neuron, **Frank Rosenblatt**, an American psychologist, proposed the **classical perception model, the mighty artificial neuron (perceptron), in 1957.**
- It is **more generalized computational model than the McCulloch-Pitts neuron where weights and thresholds can be learnt over time.**

Perceptron Model

- A perceptron is an artificial neuron in which the **activation function is the threshold function.**
- Consider an artificial neuron having x1, x2, ⋯ , xn as the input signals and w1, w2, ⋯ , wn as the associated weights.
- Let w0 be some constant (known as **bias**).
- The neuron is called a perceptron if the output of the neuron is given by the following function:

$$o(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n \leq 0 \end{cases}$$

- Below figure shows the schematic representation of a perceptron.



Perceptron Training Rule

```
Perceptron_training_rule (X, η)
initialize w (wi ← an initial (small) random value)
repeat
    for each training instance (x, tx) ∈ X
        compute the real output ox = Activation(Summation(w.x))
        if (tx ≠ ox)
            for each wi
                wi ← wi + Δwi
                Δwi ← η (tx - ox)xi
            end for
        end if
    end for
until all the training instances in X are correctly classified
return w
```

AND gate using Perceptron Model

w1 = 1.2, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*1.2 + 0*0.6 = 0*
   - This is not greater than the threshold of 1, so the output = 0
2. A=0, B=1 and Target = 0
   - *wi.xi = 0*1.2 + 1*0.6 = 0.6*
   - This is not greater than the threshold of 1, so the output = 0

AND gate using Perceptron Model

w1 = 1.2, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

3. A=1, B=0 and Target = 0

- $wi.xi = 1*1.2 + 0*0.6 = 1.2$
- This is greater than the threshold of 1, so the output = 1

$$wi = wi + n(t - o)xi$$

$$w1 = 1.2 + 0.5(0 - 1)1 = 0.7$$

$$w2 = 0.6 + 0.5(0 - 1)0 = 0.6$$

AND gate using Perceptron Model

w1 = 0.7, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*0.7 + 0*0.6 = 0*
   - This is not greater than the threshold of 1, so the output = 0
2. A=0, B=1 and Target = 0
   - *wi.xi = 0*0.7 + 1*0.6 = 0.6*
   - This is not greater than the threshold of 1, so the output = 0

AND gate using Perceptron Model

w1 = 0.7, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

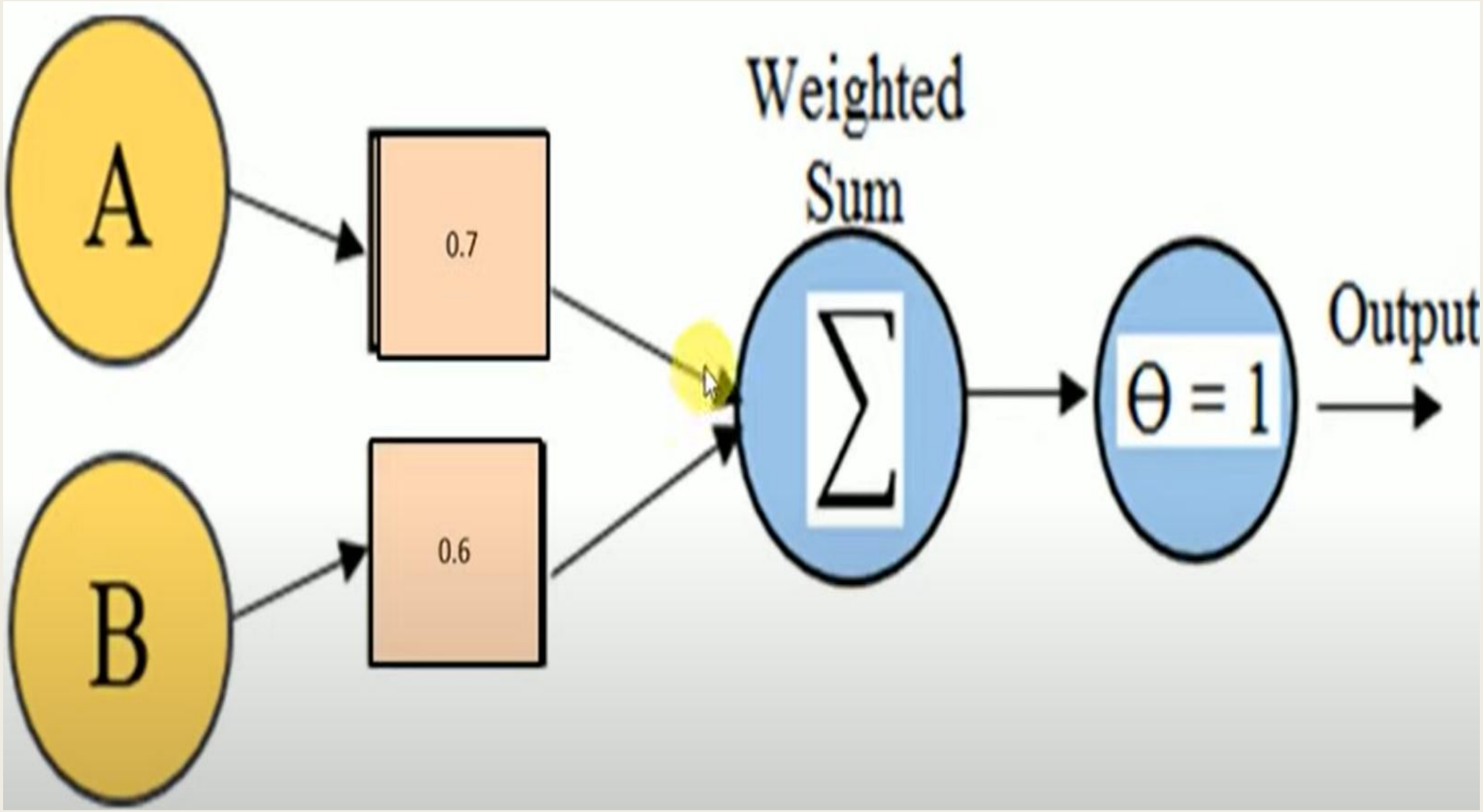
3. A=1, B=0 and Target = 0

   - *wi.xi = 1*0.7 + 0*0.6 = 0.7*
   - This is not greater than the threshold of 1, so the output = 0

4. A=1, B=1 and Target = 1

   - *wi.xi = 1*0.7 + 1*0.6 = 1.3*
   - This is greater than the threshold of 1, so the output = 1

AND gate using Perceptron Model



Characteristics and Limitations of Perceptron

- **Characteristics of Perceptron:**
- Perceptron is a machine learning algorithm for **supervised learning of binary classifiers.**
- In Perceptron, the **weight coefficient is automatically learned.**
- Initially, **weights are multiplied with input features, and the decision is made whether the neuron is fired or not.**
- The **activation function applies a step rule to check whether the weight function is greater than zero.**
- The **linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.**
- **If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.**

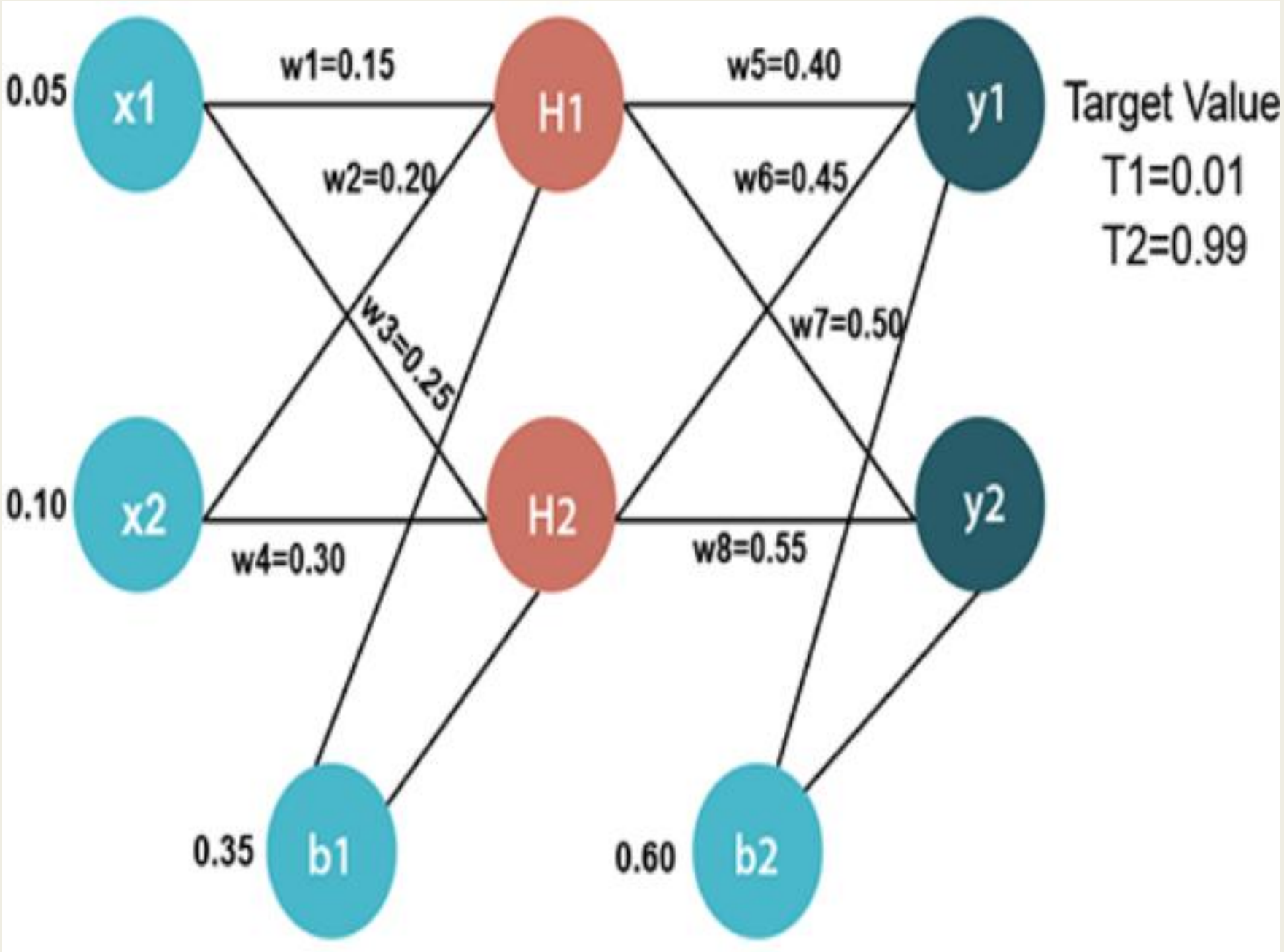
- **Limitations of Perceptron Model**
- The **output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.**
- Perceptron can **only be used to classify the linearly separable sets of input vectors.**
- **If input vectors are non-linear, it is not easy to classify them properly.**

Backpropagation

- The backpropagation algorithm was **discovered in 1985-86.**
- Here is an outline of the algorithm.
- 1. **Initially the weights are assigned at random.**
- 2. Then the algorithm **iterates through many cycles of two processes until a stopping criterion is reached. Each cycle is known as an epoch**. Each epoch includes:
- (a) A **forward phase** in which the **neurons are activated in sequence from the input layer to the output layer, applying each neuron's weights and activation function along the way**. Upon reaching the final layer, an output signal is produced.
- (b) A **backward phase** in which the network's output signal resulting from the forward phase is compared to the true target value in the training data. **The difference between the network's output signal and the true value results in an error that is propagated backwards in the network to modify the connection weights** between neurons and reduce future errors.
- 3. The technique used to determine how much a weight should be changed is known as **gradient descent method.** At every stage of the computation, the error is a function of the weights. **If we plot the error against the wights, we get a higher dimensional analog of something like a curve or surface. At any point on this surface, the gradient suggests how steeply the error will be reduced or increased for a change in the weight. The algorithm will attempt to change the weights that result in the greatest reduction in error.**

Backpropagation Example

- **Input values**
- X1=0.05
- X2=0.10
- **Initial weight**
- W1=0.15 w5=0.40
- W2=0.20 w6=0.45
- W3=0.25 w7=0.50
- W4=0.30 w8=0.55
- **Bias Values**
- b1=0.35
- b2=0.60
- **Target Values**
- T1=0.01
- T2=0.99



Backpropagation Example Solution

- **Forward Pass**
- To find the value of H1 we first multiply the input value from the weights as
- $H1 = x1 \times w1 + x2 \times w2 + b1 = 0.05 \times 0.15 + 0.10 \times 0.20 + 0.35 = 0.3775$
- To calculate the final result of H1, we performed the sigmoid function as

$$H1_{final} = \frac{1}{1 + \frac{1}{e^{H1}}}$$

$$H1_{final} = \frac{1}{1 + \frac{1}{e^{0.3775}}}$$

$$\mathbf{H1_{final} = 0.593269992}$$

- We will calculate the value of H2 in the same way as H1
- $H2 = x1 \times w3 + x2 \times w4 + b1 = 0.05 \times 0.25 + 0.10 \times 0.30 + 0.35 = 0.3925$
- To calculate the final result of H1, we performed the sigmoid function as

$$H2_{final} = \frac{1}{1 + \frac{1}{e^{H2}}}$$

$$H2_{final} = \frac{1}{1 + \frac{1}{e^{0.3925}}}$$

$$\mathbf{H2_{final} = 0.596884378}$$

Backpropagation Example Solution

- Now, we calculate the values of y1 and y2 in the same way as we calculate the H1 and H2.
- To find value of y1, we first multiply the input value i.e., the outcome of H1 and H2 from the weights as
- y1=H1×w5+H2×w6+b2 =0.593269992×0.40+0.596884378×0.45+0.60 =1.10590597
- To calculate the final result of y1 we performed the sigmoid function as

$$y1_{final} = \frac{1}{1 + \frac{1}{e^{y1}}}$$

$$y1_{final} = \frac{1}{1 + \frac{1}{e^{1.10590597}}}$$

$$y1_{final} = 0.75136507$$

- We will calculate the value of y2 in the same way as y1
- y2=H1×w7+H2×w8+b2 =0.593269992×0.50+0.596884378×0.55+0.60 =1.2249214
- To calculate the final result of y2, we performed the sigmoid function as

$$y2_{final} = \frac{1}{1 + \frac{1}{e^{y2}}}$$

$$y2_{final} = \frac{1}{1 + \frac{1}{e^{1.2249214}}}$$

$$y2_{final} = 0.772928465$$

Backpropagation Example Solution

- Our target values are 0.01 and 0.99. Our y1 and y2 value is not matched with our target values T1 and T2.
- Now, we will find the total error, which is simply the difference between the outputs from the target outputs. The total error is calculated as

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

- So, the total error is

$$= \frac{1}{2}(t1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

$$= \frac{1}{2}(0.01 - 0.75136507)^2 + \frac{1}{2}(0.99 - 0.772928465)^2$$

$$= 0.274811084 + 0.0235600257$$

$$\mathbf{E_{total} = 0.29837111}$$

- Now, we will backpropagate this error to update the weights using a backward pass.

Backpropagation Example Solution

- **Backward pass at the output layer**
- To update the weight, we calculate the error correspond to each weight with the help of a total error. The error on weight w is calculated by differentiating total error with respect to w.

$$\text{Error}_w = \frac{\partial E_{total}}{\partial w}$$

- We perform backward process so first consider the last weight w5 as

$$\text{Error}_{w5} = \frac{\partial E_{total}}{\partial w5} \ldots\ldots\ldots (1)$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2 \ldots\ldots\ldots (2)$$

- From equation two, it is clear that we cannot partially differentiate it with respect to w5 because there is no any w5. We split equation one into multiple terms so that we can easily differentiate it with respect to w5 as

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1\_final}{\partial y1} \times \frac{\partial y1}{\partial w5} \ldots\ldots\ldots (3)$$

Backpropagation Example Solution

- Now, we calculate each term one by one to differentiate Etotal with respect to w5 as

$$\frac{\partial E_{total}}{\partial y1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial y1_{final}}$$

$$= 2 \times \frac{1}{2} \times (T1 - y1_{final})^{2-1} \times (-1) + 0$$

$$= -(T1 - y1_{final})$$

$$= -(0.01 - 0.75136507)$$

$$\frac{\partial E_{total}}{\partial y1_{final}} = 0.74136507 \ldots\ldots\ldots (4)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}} \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

$$\frac{\partial y1_{final}}{\partial y1} = \frac{\partial(\frac{1}{1 + e^{-y1}})}{\partial y1}$$

$$= \frac{e^{-y1}}{(1 + e^{-y1})^2}$$

$$= e^{-y1} \times (y1_{final})^2 \ldots\ldots\ldots\ldots\ldots (6)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}}$$

$$e^{-y1} = \frac{1 - y1_{final}}{y1_{final}} \ldots\ldots\ldots\ldots\ldots (7)$$

Backpropagation Example Solution

- Putting the value of e^{-y} in equation (5)

$$= \frac{1 - y1_{final}}{y1_{final}} \times (y1_{final})^2$$

$$= y1_{final} \times (1 - y1_{final})$$

$$= 0.75136507 \times (1 - 0.75136507)$$

$$\frac{\partial y1_{final}}{\partial y1} = \mathbf{0.186815602} \ldots\ldots\ldots (8)$$

$$y1 = H1_{final} \times w5 + H2_{final} \times w6 + b2 \ldots\ldots\ldots\ldots\ldots\ldots (9)$$

$$\frac{\partial y1}{\partial w5} = \frac{\partial (H1_{final} \times w5 + H2_{final} \times w6 + b2)}{\partial w5}$$

$$= H1_{final}$$

$$\frac{\partial y1}{\partial w5} = \mathbf{0.596884378} \ldots\ldots\ldots (10)$$

- So, we put the values of $\frac{\partial E_{total}}{\partial y1_{final}}$, $\frac{\partial y1_{final}}{\partial y1}$, and $\frac{\partial y1}{\partial w5}$ in equation no (3) to find the final result.

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1} \times \frac{\partial y1}{\partial w5}$$

$$= 0.74136507 \times 0.186815602 \times 0.593269992$$

$$\mathbf{Error_{w5}} = \frac{\partial E_{total}}{\partial w5} = \mathbf{0.0821670407} \ldots\ldots\ldots (11)$$

Backpropagation Example Solution

- Now, we will calculate the updated weight w5new with the help of the following formula

$$w5_{new} = w5 - \eta \times \frac{\partial E_{total}}{\partial w5} \text{ Here, } \eta = \text{learning rate} = 0.5$$

$$= 0.4 - 0.5 \times 0.0821670407$$

$$\mathbf{w5_{new} = 0.35891648 \ldots\ldots\ldots (12)}$$

- In the same way, we calculate $w6_{new}$,$w7_{new}$, and $w8_{new}$ and this will give us the following values
- $w5_{new}$=0.35891648
- $w6_{new}$=408666186
- $w7_{new}$=0.511301270
- $w8_{new}$=0.561370121

Backpropagation Example Solution

- **Backward pass at Hidden layer**
- Now, we will backpropagate to our hidden layer and update the weight w1, w2, w3, and w4 as we have done with w5, w6, w7, and w8 weights.
- We will calculate the error at w1 as

$$\text{Error}_{w1} = \frac{\partial E_{total}}{\partial w1}$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

- From eqn (2), it is clear that we cannot partially differentiate it with respect to w1 because there is no any w1. We split equation (1) into multiple terms so that we can easily differentiate it with respect to w1 as

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1\_final}{\partial H1} \times \frac{\partial H1}{\partial w1} \dots\dots\dots\dots (13)$$

- Now, we calculate each term one by one to differentiate E_{total} with respect to w1 as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial H1} \dots\dots\dots\dots (14)$$

- We again split this because there is no any $H1_{final}$ term in E_{toatal} as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}} \dots\dots\dots (15)$$

Backpropagation Example Solution

- $\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ will again split because in E1 and E2 there is no H1 term. Splitting is done as

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}} \dots\dots\dots (16)$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final} \dots\dots\dots (17)$$

- We again Split both $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ because there is no any y1 and y2 term in E1 and E2. We split it as

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1} \dots\dots\dots (18)$$

$$\frac{\partial E_2}{\partial y2} = \frac{\partial E_2}{\partial y2_{fianl}} \times \frac{\partial y2_{final}}{\partial y2} \dots\dots\dots (19)$$

- Now, we find the value of $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ by putting values in equation (18) and (19) as
- From equation (18)

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2)}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= 2 \times \frac{1}{2}(T1 - y1_{final}) \times (-1) \times \frac{\partial y1_{final}}{\partial y1}$$

Backpropagation Example Solution

- From equation (8)

$$= 2 \times \frac{1}{2}(0.01 - 0.75136507) \times (-1) \times 0.186815602$$

$$\frac{\partial E_1}{\partial y1} = \mathbf{0.138498562} \ldots\ldots\ldots (20)$$

- From equation (19)

$$\frac{\partial E_2}{\partial y2} = \frac{\partial E_2}{\partial y2_{fianl}} \times \frac{\partial y2_{final}}{\partial y2}$$

$$= \frac{\partial(\frac{1}{2}(T2 - y2_{final})^2)}{\partial y2_{final}} \times \frac{\partial y2_{final}}{\partial y2}$$

$$= 2 \times \frac{1}{2}(T2 - y2_{final}) \times (-1) \times \frac{\partial y2_{final}}{\partial y2} \ldots\ldots\ldots (21)$$

$$y2_{final} = \frac{1}{1 + e^{-y2}} \ldots\ldots\ldots\ldots\ldots (22)$$

$$\frac{\partial y2_{fianl}}{\partial y2} = \frac{\partial(\frac{1}{1 + e^{-y2}})}{\partial y2}$$

$$= \frac{e^{-y2}}{(1 + e^{-y2})^2}$$

$$= e^{-y2} \times (y2_{final})^2 \ldots\ldots\ldots\ldots\ldots (23)$$

$$y2_{final} = \frac{1}{1 + e^{-y2}}$$

$$e^{-y2} = \frac{1 - y2_{final}}{y2_{final}} \ldots\ldots\ldots\ldots\ldots (24)$$

Backpropagation Example Solution

Putting the value of e^{-y2} in equation (23)

$$= \frac{1 - y2_{final}}{y2_{final}} \times (y2_{final})^2$$

$$= y2_{final} \times (1 - y2_{final})$$

$$= 0.772928465 \times (1 - 0.772928465)$$

$$\frac{\partial y2_{fianl}}{\partial y2} = \mathbf{0.175510053} \ldots\ldots\ldots (25)$$

From equation (21)

$$= 2 \times \frac{1}{2}(0.99 - 0.772928465) \times (-1) \times \ 0.175510053$$

$$\frac{\partial E_1}{\partial y1} = \mathbf{-0.038098236612641 4} \ldots\ldots\ldots (26)$$

Backpropagation Example Solution

- Now from equation (16) and (17)

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial (H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial (H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times w5$$

$$= 0.138498562 \times 0.40$$

$$\boldsymbol{\frac{\partial E_1}{\partial H1_{final}} = 0.0553994248 \ldots\ldots\ldots (27)}$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final}$$

$$= -0.0380982366126414 \times \frac{\partial (H1_{final} \times w_7 + H2_{final} \times w_8 + b2)}{\partial H1_{final}}$$

$$= -0.0380982366126414 \times w7$$

$$= -0.0380982366126414 \times 0.50$$

$$\boldsymbol{\frac{\partial E_2}{\partial H1_{final}} = -0.0190491183063207 \ldots\ldots\ldots (28)}$$

Backpropagation Example Solution

Put the value of $\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ in equation (15) as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}}$$

$$= 0.0553994248 + (-0.0190491183063207)$$

$$\frac{\partial E_{total}}{\partial H1_{final}} = \mathbf{0.0364908241736793} \ldots\ldots\ldots (29)$$

We have $\frac{\partial E_{total}}{\partial H1_{final}}$; we need to figure out $\frac{\partial H1\_final}{\partial H1}$, $\frac{\partial H1}{\partial w1}$ as

$$\frac{\partial H1_{final}}{\partial H1} = \frac{\partial(\frac{1}{1+e^{-H1}})}{\partial H1}$$

$$= \frac{e^{-H1}}{(1+e^{-H1})^2}$$

$$e^{-H1} \times (H1_{final})^2 \ldots\ldots\ldots (30)$$

$$H1_{final} = \frac{1}{1+e^{-H1}}$$

$$e^{-H1} = \frac{1 - H1_{final}}{H1_{final}} \ldots\ldots\ldots\ldots (31)$$

Backpropagation Example Solution

Putting the value of e^{-H1} in equation (30)

$$= \frac{1 - \mathrm{H1_{final}}}{\mathrm{H1_{final}}} \times (\mathrm{H1_{final}})^2$$

$$= \mathrm{H1_{final}} \times (1 - \mathrm{H1_{final}})$$

$$= 0.593269992 \times (1 - 0.593269992)$$

$$\frac{\partial \mathrm{H1_{final}}}{\partial \mathrm{H1}} = \mathbf{0.241300708592319 9}$$

We calculate the partial derivative of the total net input to H1 with respect to w1 the same as we did for the output neuron:

$$\mathrm{H1} = \mathrm{H1_{final}} \times \mathrm{w5} + \mathrm{H2_{final}} \times \mathrm{w6} + \mathrm{b2} \ldots\ldots\ldots\ldots\ldots\ldots (32)$$

$$\frac{\partial \mathrm{y1}}{\partial \mathrm{w1}} = \frac{\partial(\mathrm{x1} \times \mathrm{w1} + \mathrm{x2} \times \mathrm{w3} + \mathrm{b1} \times 1)}{\partial \mathrm{w1}}$$

$$= \mathrm{x1}$$

$$\frac{\partial \mathrm{H1}}{\partial \mathrm{w1}} = \mathbf{0.05} \ldots\ldots\ldots (33)$$

Backpropagation Example Solution

So, we put the values of $\frac{\partial E_{total}}{\partial H1_{final}}$, $\frac{\partial H1_{final}}{\partial H1}$, and $\frac{\partial H1}{\partial w1}$ in equation (13) to find the final result.

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1_{final}}{\partial H1} \times \frac{\partial H1}{\partial w1}$$

$$= 0.036490824173679 3 \times 0.241300708592319 9 \times 0.05$$

$$\mathbf{Error_{w1}} = \frac{\partial E_{total}}{\partial w1} = \mathbf{0.000438568} \ldots\ldots\ldots (34)$$

Now, we will calculate the updated weight $w1_{new}$ with the help of the following formula

$$w1_{new} = w1 - \eta \times \frac{\partial E_{total}}{\partial w1} \text{ Here } \eta = \text{learning rate} = 0.5$$

$$= 0.15 - 0.5 \times 0.000438568$$

$$\mathbf{w1_{new}} = \mathbf{0.149780716} \ldots\ldots\ldots (35)$$

In the same way, we calculate $w2_{new}$, $w3_{new}$, and w4 and this will give us the following values

$\mathbf{w1_{new}}$**=0.149780716**

$\mathbf{w2_{new}}$**=0.19956143**

$\mathbf{w3_{new}}$**=0.24975114**

$\mathbf{w4_{new}}$**=0.29950229**

Backpropagation Example Solution

- We have updated all the weights.
- We found the error 0.298371109 on the network when we fed forward the 0.05 and 0.1 inputs.
- In the first round of Backpropagation, the total error is down to 0.291027924.
- **After repeating this process 10,000, the total error is down to 0.0000351085.**
- At this point, the outputs neurons generate 0.159121960 and 0.984065734 i.e., nearby our target value when we feed forward the 0.05 and 0.1.

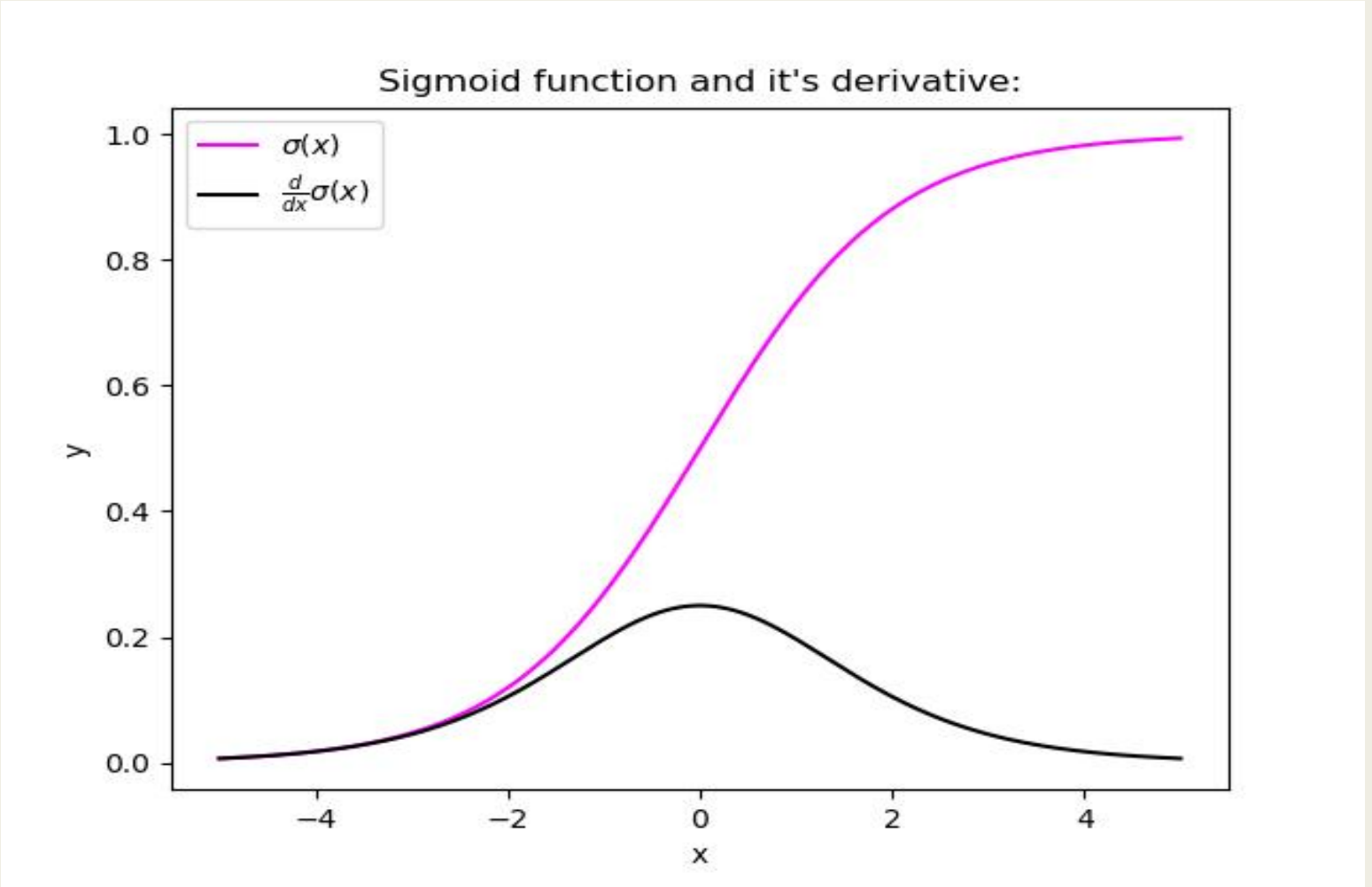
Understanding the Vanishing and Exploding gradient Problems

- **Vanishing gradients problem**
- As the backpropagation algorithm advances downwards(or backward) from the output layer towards the input layer, **the gradients often get smaller and smaller and approach zero which eventually leaves the weights of the initial or lower layers nearly unchanged.**
- As a result, the gradient descent never converges to the optimum.
- This is known as the **vanishing gradients problem.**

- **Exploding gradients problem**
- In some cases, the **gradients keep on getting larger and larger as the backpropagation algorithm progresses.**
- This, in turn, **causes very large weight updates and causes the gradient descent to diverge.**
- This is known as the **exploding gradients problem.**

Why Do the Gradients Even Vanish/Explode?

- Certain activation functions, like the **logistic function (sigmoid), have a very huge difference between the variance of their inputs and the outputs.**
- In simpler words, **they shrink and transform a larger input space into a smaller output space that lies between the range of [0,1].**



Why Do the Gradients Even Vanish/Explode?

- Observing the above graph of the Sigmoid function, we can see that **for larger inputs (negative or positive), it saturates at 0 or 1 with a derivative very close to zero.**
- Thus, when the backpropagation algorithm chips in, it virtually has **no gradients to propagate backward in the network, and whatever little residual gradients exist keeps on diluting as the algorithm progresses down through the top layers.**
- So, this **leaves nothing for the lower layers.**
- Similarly, in **some cases suppose the initial weights assigned to the network generate some large loss.**
- Now **the gradients can accumulate during an update and result in very large gradients which eventually results in large updates to the network weights and leads to an unstable network.**
- The **parameters can sometimes become so large that they overflow and result in NaN values.**

Signs of Exploding/Vanishing Gradient Problem

- Following are some signs that can indicate that our gradients are exploding/vanishing :

Exploding	Vanishing
There is an exponential growth in the model parameters.	The parameters of the higher layers change significantly whereas the parameters of lower layers would not change much (or not at all).
The model weights may become NaN during training.	The model weights may become 0 during training.
The model experiences avalanche learning.	The model learns very slowly and perhaps the training stagnates at a very early stage just after a few iterations.

How to avoid Exploding/Vanishing Gradient Problem

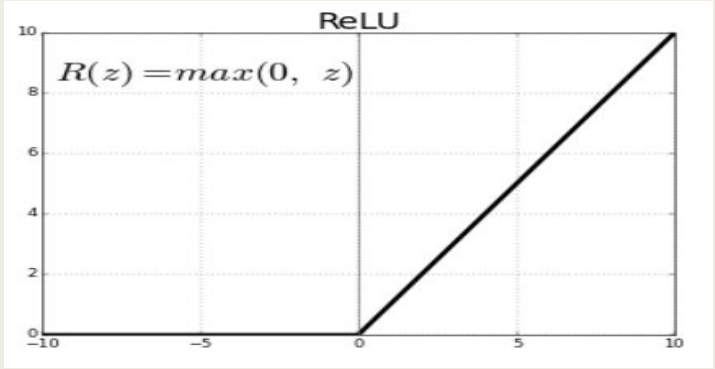
- **Proper Weight Initialization**
- For the proper flow of the signal:
- The **variance of outputs of each layer should be equal to the variance of its inputs.**
- The **gradients should have equal variance before and after flowing through a layer in the reverse direction.**
- To satisfy both the conditions, Xavier Glorot et al. proposed an initialization method known as **Xavier initialization** (after the author's first name) or **Glorot initialization** (after his last name).
- **Uniform Xavier Initialization**
- We can initialize the weights by drawing them from a **random uniform distribution within a specific range, which is determined by the formula:**

$$x = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$$

- x is calculated using above formula.
- n_{inputs} : Number of Input in the input layer
- n_{output} : Number of Output in the Output layer
- For each weight in network, we draw a random value w from a uniform distribution in the range [-x, x].

How to avoid Exploding/Vanishing Gradient Problem

- **Using Non-saturating Activation Functions**
- we observed that sigmoid activation function's **nature of saturating for larger inputs (negative or positive) came out to be a major reason behind the vanishing of gradients** thus making it non-recommendable to use in the hidden layers of the network.
- So to **tackle the issue regarding the saturation of activation functions like sigmoid and tanh, we must use some other non-saturating functions like ReLu and its alternatives.**



- Unfortunately, the **ReLu function is also not a perfect pick for the intermediate layers of the network "in some cases"**.
- It suffers from a problem known as **dying ReLus wherein some neurons just die out, meaning they keep on throwing 0 as outputs with the advancement in training.**
- Some popular alternative functions of the ReLU that mitigates the problem of vanishing gradients when used as activation for the intermediate layers of the network are LReLU, PReLU, ELU, SELU.

How to avoid Exploding/Vanishing Gradient Problem

- Using **He initialization along with any variant of the ReLU activation function can significantly reduce the chances of vanishing/exploding problems** at the beginning. However, it does not guarantee that the problem won't reappear during training.
- In **2015, Sergey Ioffe and Christian Szegedy proposed a paper in which they introduced a technique known as Batch Normalization to address the problem of vanishing/exploding gradients.**
- The **Following key points** explain the intuition behind BN and how it works:
- It consists of **adding an operation in the model just before or after the activation function of each hidden layer.**
- **This operation simply zero-centers and normalizes each input, then scales and shifts the result using two new parameter vectors per layer: one for scaling, the other for shifting.**
- In other words, the **operation lets the model learn the optimal scale and mean of each of the layer's inputs.**
- **To zero-center and normalize the inputs, the algorithm needs to estimate each input's mean and standard deviation.**
- **It does so by evaluating the mean and standard deviation of the input over the current mini-batch (hence the name "Batch Normalization").**

How to avoid Exploding/Vanishing Gradient Problem

- **Gradients clipping**
- To **prevent gradients from exploding, one of the most effective ways is gradient clipping.**
- In a nutshell, gradient clipping **caps the derivatives to a threshold and uses the capped gradients to update the weights throughout.**

- **L2 norm regularization**
- In addition to weight initialization, another excellent approach is employing L2 regularization, which **penalizes large weight values by imposing a squared term of model weights to the loss function:**

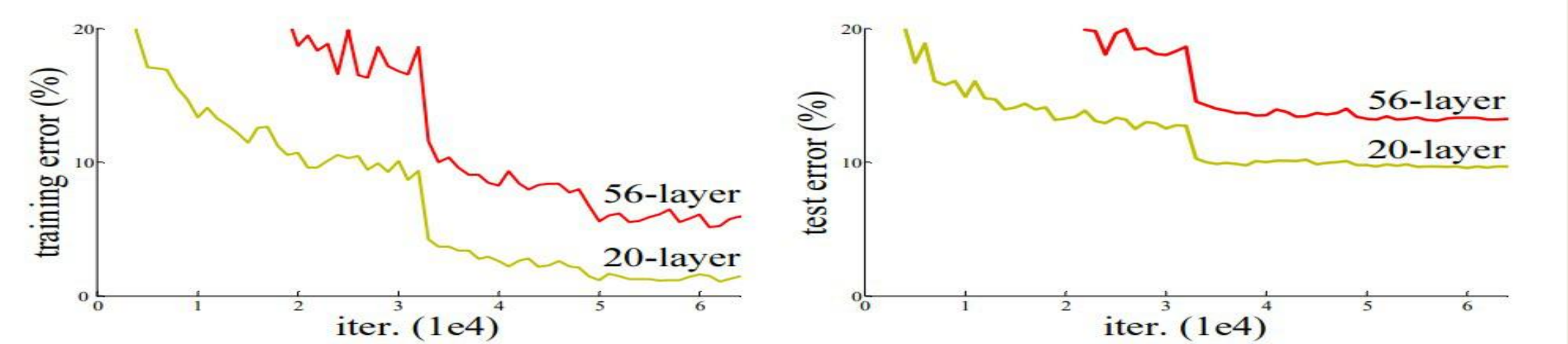
$$Loss = Error(Y - \hat{Y}) + \lambda \sum_{1}^{n} w_i^2$$

- **Adding the L2 norm oftentimes will result in smaller weight updates throughout the network.**

- **Using less number of layers**
- Using fewer layer will ensure that the **gradient is not multiplied too many times.**
- This **may stop the gradient from vanishing or exploding, but it affect the ability of network to understand complex features.**

Problem with deep neural networks

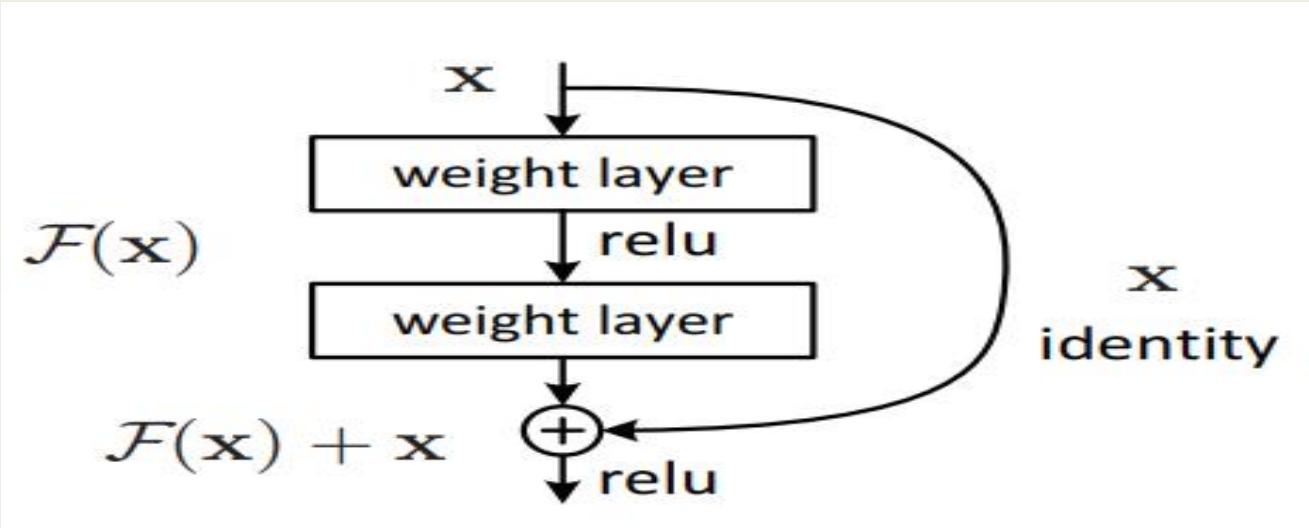
- After the **first CNN-based architecture (AlexNet) that win the ImageNet 2012 competition, Every subsequent winning architecture uses more layers in a deep neural network to reduce the error rate.**
- This works for less number of layers, but **when we increase the number of layers, there is a common problem in deep learning associated with that called the Vanishing/Exploding gradient.**
- This causes the gradient to become 0 or too large.
- Thus when we increases number of layers, the training and test error rate also increases



- In the above plot, we can observe that a **56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture.**
- After analyzing more on error rate the authors were able to reach conclusion that it is **caused by vanishing/exploding gradient.**

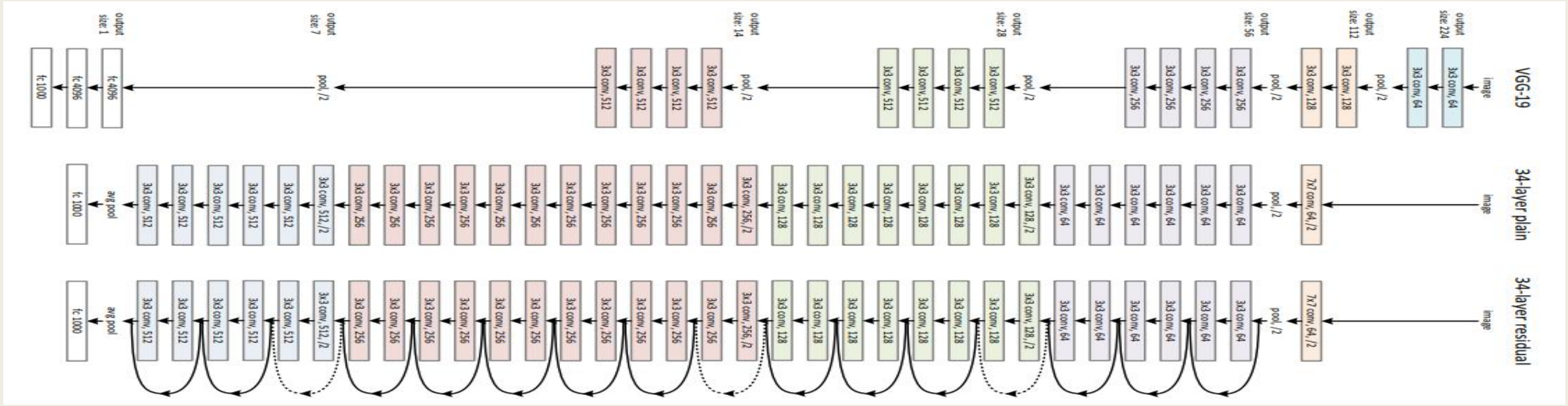
Residual Network (ResNet)

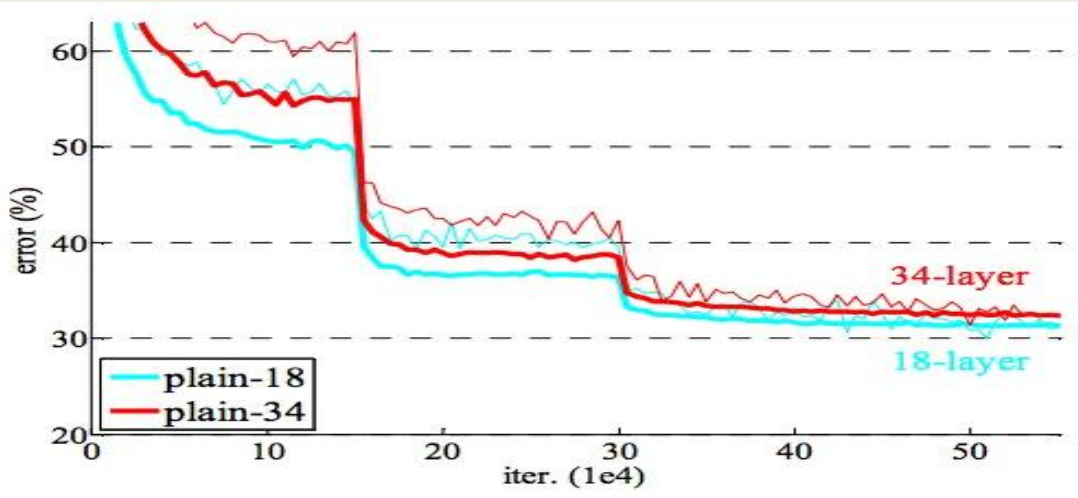
- In order to solve the problem of the vanishing/exploding gradient, **Microsoft Research introduced a new architecture called Residual Network (ResNet) in 2015.**
- In this network, we use a technique called **skip connections.**
- The **skip connection connects activations of a layer to further layers by skipping some layers in between.** This forms a **residual block.**
- Resnets are made by **stacking these residual blocks together.**
- The approach behind this network is **instead of layers learning the underlying mapping, we allow the network to fit the residual mapping.**
- So, instead of say H(x), initial mapping, let the network fit,
- F(x) := H(x) - x which gives H(x) := F(x) + x.

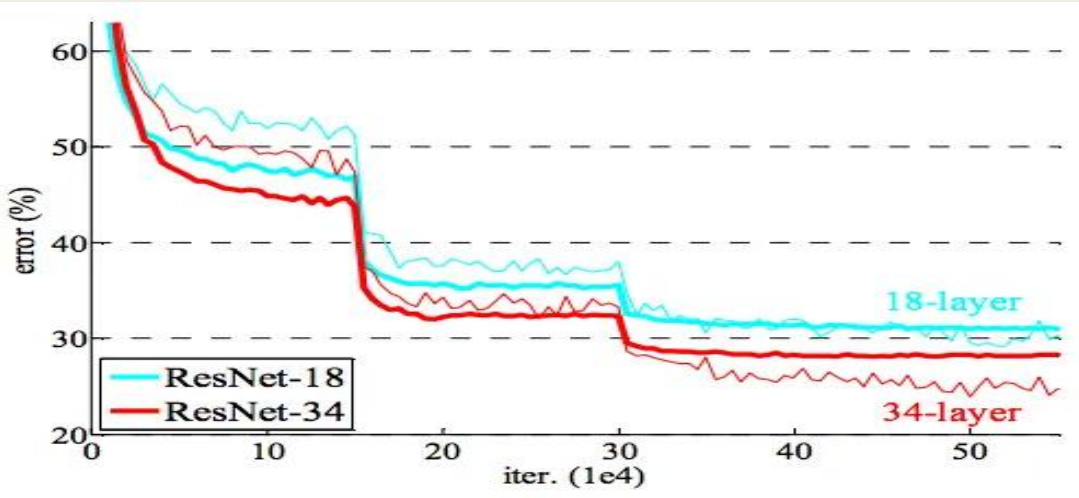


Residual Network (ResNet)

- The advantage of adding this type of skip connection is that **if any layer hurt the performance of architecture then it will be skipped by regularization.**
- This results in training a very deep neural network without the problems by vanishing/exploding gradient.







Previous Year Question (Neural Network)

❖ 1. Implement AND function using McCulloch–Pitts neuron (take binary data). Consider the below given truth table for AND function.

X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

❖ Ans:

- Consider the truth table for AND function
- The M–P neuron has no particular training algorithm
- In M-Pneuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = 1.

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

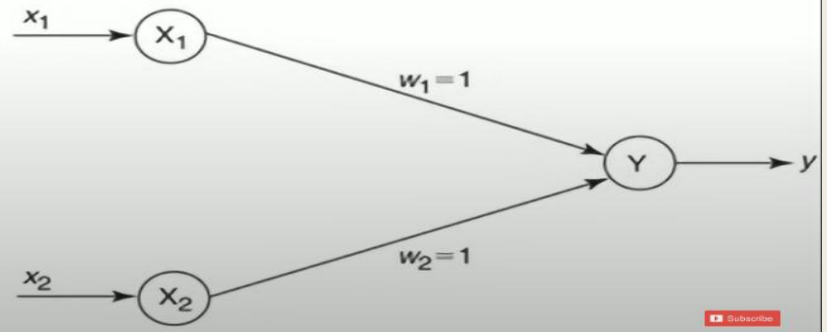
$(1, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$

$(1, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1$

$(0, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 1 \times 1 = 1$

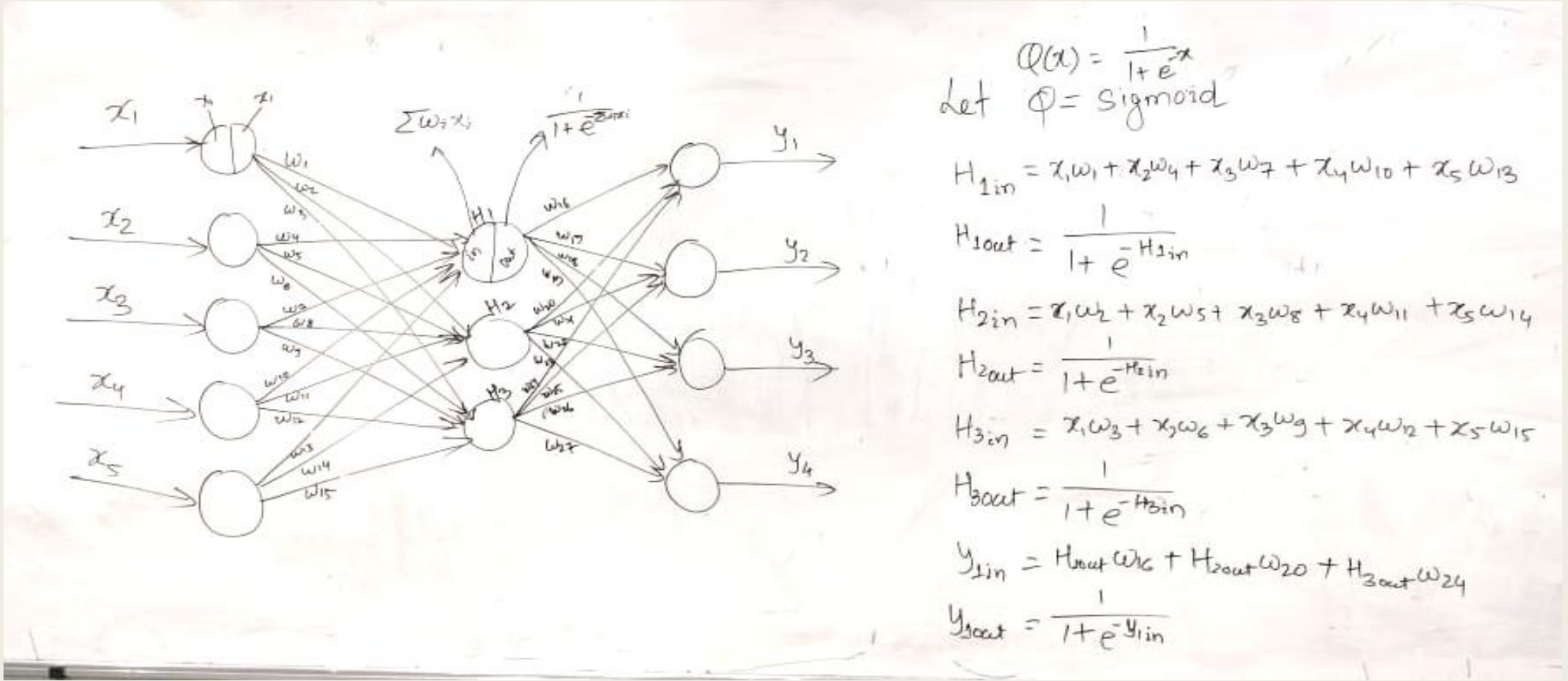
$(0, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0$

Threshold value is set equal to 2 (θ = 2).



Previous Year Question (Neural Network)

- 1. Construct a feed-forward network with five input nodes, three hidden nodes and four output nodes with necessary mathematical expressions and explanations.
- Ans:



Previous Year Question (Neural Network)

- 1. Draw an ANN architecture for 3 inputs, 2 hidden layer each with 2 neurons and one output. Derive the estimated output using sigmoid transfer function.
- Ans:

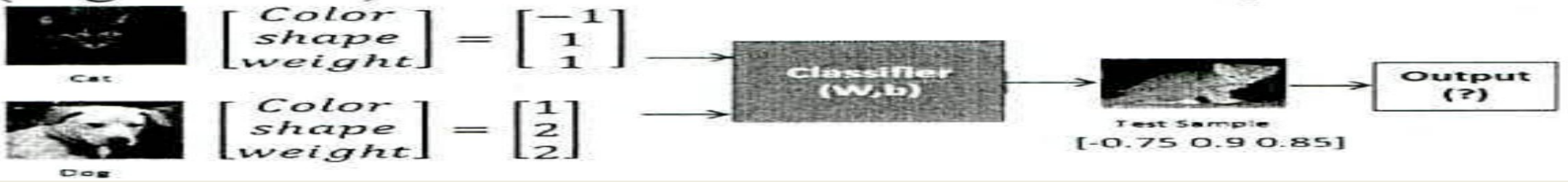
Previous Year Question (Neural Network)

- 1. Differentiate between linearly and non-linearly separable datasets. A two input single output neuron model has weights value [-1.5 2.0] and bias of -2.5. It is given an input [2.23.1]. What will be the output if the binary step function threshold=1 is used?

- Ans: g(x) = 2.2 x -1.5 + 3.1 x 2.0 + (-2.5) = 0.4
- F(x) = Binary step function
- Output = F(g(x)) = F(0.4) = 0

Previous Year Question (Neural Network)

- 1. Solve the problem: Use a Three-Input/Single-Neuron Perceptron with weights $w_{11}=1$, $w_{12}=0.5$ and $w_{13}=1$. Draw the perceptron, decision boundary and compute whether test sample [-0.75 0.9 0.85] is cat (target class 0) or dog (target class 1). Use transfer function hardlim().



- Ans:- Here F(x) = hardlim()
- For Cat case
- g(x) = -1 x 1 + 1 x 0.5 + 1 x 1 = 0.5
- Output = F(g(x)) = F(0.5) = 0 (cat)
- For Dog case
- g(x) = 1 x 1 + 2 x 0.5 + 2 x 1 = 4
- Output = F(g(x)) = F(4) = 1 (dog)
- From above two example, we can see that the threshold should be 1.
- For test sample
- g(x) = -0.75 x 1 + 0.9 x 0.5 + 0.85 x 1 = 0.55
- Output = F(g(x)) = F(0.55) = 0 (cat)

Previous Year Question (Neural Network)

- 1. Design a three inputs, two layers with two-two neurons and one output ANN model with its input to output relationship using F() as an activation function.
- Ans:-

Previous Year Question (Neural Network)

- 1. b) Derive Back propagation algorithm for a NN having 3 inputs, 2 hidden layers each having three neurons and output layer having two neurons. You may ignore the bias term. Sigmoid is used as the activation function in the network. [2+6]

- Ans:-

Previous Year Question (K - Means Clustering)

- 1. Divide the given sample data set into two(2) clusters using K means Algorithm by using Euclidean distance. Use Cluster 1 (185, 72) and Cluster 2 (170, 56) as initial cluster centers.

Height	Weight
185	72
170	56
168	60
179	68
182	72
188	77

- Ans: Computing the distances from initial cluster centers

xi	Data Point	Distance from cluster-1 (185,72)	Distance from cluster-2 (170,56)	Min Dist	Assigned Cluster
X1	(185,72)	0	21.93	0	Cluster-1
X2	(170,56)	21.93	0	0	Cluster-2
X3	(168,60)	20.81	4.47	4.47	Cluster-2
X4	(179,68)	7.21	15	7.21	Cluster-1
X5	(182,72)	3	20	3	Cluster-1
X6	(188,77)	5.83	27.66	5.83	Cluster-1

Previous Year Question (K - Means Clustering)

- The cluster centers are recalculated as follows:-
- Cluster-1 new center = {(185+179+182+188)/4,(72+68+72+77)/4} = (183.5,72.25)
- Cluster-2 new center = {(170+168)/2,(56+60)/2} = (169,58)
- Computing the distances from new cluster centers

xi	Data Point	Distance from cluster-1 (183.5,72.25)	Distance from cluster-2 (169,58)	Min Dist	Assigned Cluster
X1	(185,72)	1.52	21.26	1.52	Cluster-1
X2	(170,56)	21.13	2.24	2.24	Cluster-2
X3	(168,60)	19.76	2.24	2.24	Cluster-2
X4	(179,68)	6.19	14.14	6.19	Cluster-1
X5	(182,72)	1.52	19.10	1.52	Cluster-1
X6	(188,77)	6.54	26.87	6.54	Cluster-1

- Since there is no reassignment of data points to different clusters so we will stop the algorithm.
- The Final clusters are as follows:-
- Cluster-1={X1,X4,X5,X6} represented by v1 = (183.5,72.25)
- Cluster-2={X2,X3} represented by v2 = (169,58)

Previous Year Question (K - Means Clustering)

- 2. b) Apply K-means clustering algorithm on given data for K=3. Use C1(2), C2(16), C3(38) as initial cluster centers. Data: 2, 4, 6, 3, 31, 12, 15, 16, 38, 35, 14, 21, 23, 25, 33. [5]
- Ans:

Previous Year Question (K - Means Clustering)

- 3. (b) Cluster the given dataset of e-commerce sales with the following features:

Product_ID: ID of the product sold
Price: Price of the product (in dollars)
Quantity: Quantity of the product sold
Category: Category of the product (A, B, or C)

We want to use K-means clustering to group the products into three clusters based on their price and quantity. We will use K=3 for the K-means algorithm.

The sales data:

Product_ID	Price	Quantity	Category
1	50	10	A
2	40	8	A
3	60	12	A
4	70	15	B
5	80	18	B
6	90	20	C
7	100	22	C
8	110	25	C

- Ans:

Previous Year Question (K - Means Clustering)

- 4. 5. a) Apply K-means clustering algorithm on given data for K=2. Use C1(4), C2(12) as initial cluster centers. Data:{2, 3, 4, 10, 11, 12, 20, 25, 30} [3 Marks]
- Ans:

Previous Year Question (PCA)

- 1. Explain the Principal Component Analysis (PCA) and reduce the following dataset step-by-step from 2 dimensions to 1. [4]

Feature	Example 1	Example 2	Example 3	Example 4
x	4	8	13	7
y	11	4	5	14

- **Ans: Step-01:**
- **Get data.**
- **The given feature vectors are-**
- x1 = (4, 11)
- x2 = (8, 4)
- x3 = (13, 5)
- x4 = (7, 14)
- **Step-02:**
- **Calculate the mean vector (μ).**
- Mean vector (μ) = ((4 + 8 + 13 + 7) / 4, (11 + 4 + 5 + 14) / 4) = (8, 8.5)

Previous Year Question (PCA)

- **Step-03:**
- **Subtract mean vector (μ) from the given feature vectors.**
- $x1 - \mu = (4-8, 11-8.5) = (-4, 2.5)$
- $x2 - \mu = (8-8, 4-8.5) = (0, -4.5)$
- $x3 - \mu = (13-8, 5-8.5) = (5, -3.5)$
- $x4 - \mu = (7-8, 14-8.5) = (-1, 5.5)$

- **Step-04:**
- **Calculate the covariance matrix.**
- $m1 = (x1 - \mu)(x1 - \mu)^T = (-4, 2.5)(-4, 2.5)^T = \begin{bmatrix} 16 & -10 \\ -10 & 6.25 \end{bmatrix}$
- $m2 = (x2 - \mu)(x2 - \mu)^T = (0, -4.5)(0, -4.5)^T = \begin{bmatrix} 0 & 0 \\ 0 & 20.25 \end{bmatrix}$
- $m3 = (x3 - \mu)(x3 - \mu)^T = (5, -3.5)(5, -3.5)^T = \begin{bmatrix} 25 & -17.5 \\ -17.5 & 12.25 \end{bmatrix}$
- $m4 = (x4 - \mu)(x4 - \mu)^T = (-1, 5.5)(-1, 5.5)^T = \begin{bmatrix} 1 & -5.5 \\ -5.5 & 30.25 \end{bmatrix}$
- Covariance matrix = $(m1 + m2 + m3 + m4) / 4 = \begin{bmatrix} 10.5 & -8.25 \\ -8.25 & 17.25 \end{bmatrix}$

Previous Year Question (PCA)

- **Step-05:**
- **Calculate the eigen values and eigen vectors of the covariance matrix.**
- **λ is an eigen value for a matrix M if it is a solution of the characteristic equation $|M - \lambda I| = 0$.**

$$\left| \begin{bmatrix} 10.5 & -8.25 \\ -8.25 & 17.25 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$

- $(10.5 - \lambda)(17.25 - \lambda) - (-8.25)(-8.25) = 0$
- $113.0625 - 27.75\lambda + \lambda^2 = 0$
- Solving this quadratic equation, we get $\lambda = 22.7886, 4.96135$
- Thus, two eigen values are $\lambda 1 = 22.7886$ and $\lambda 2 = 4.96135$.
- Clearly, the second eigen value is very small compared to the first eigen value.
- So, the second eigen vector can be left out.
- Eigen vector corresponds to the greatest eigen value is principal component for the given data set.
- So. we find the eigen vector corresponding to eigen value $\lambda 1$.

Previous Year Question (PCA)

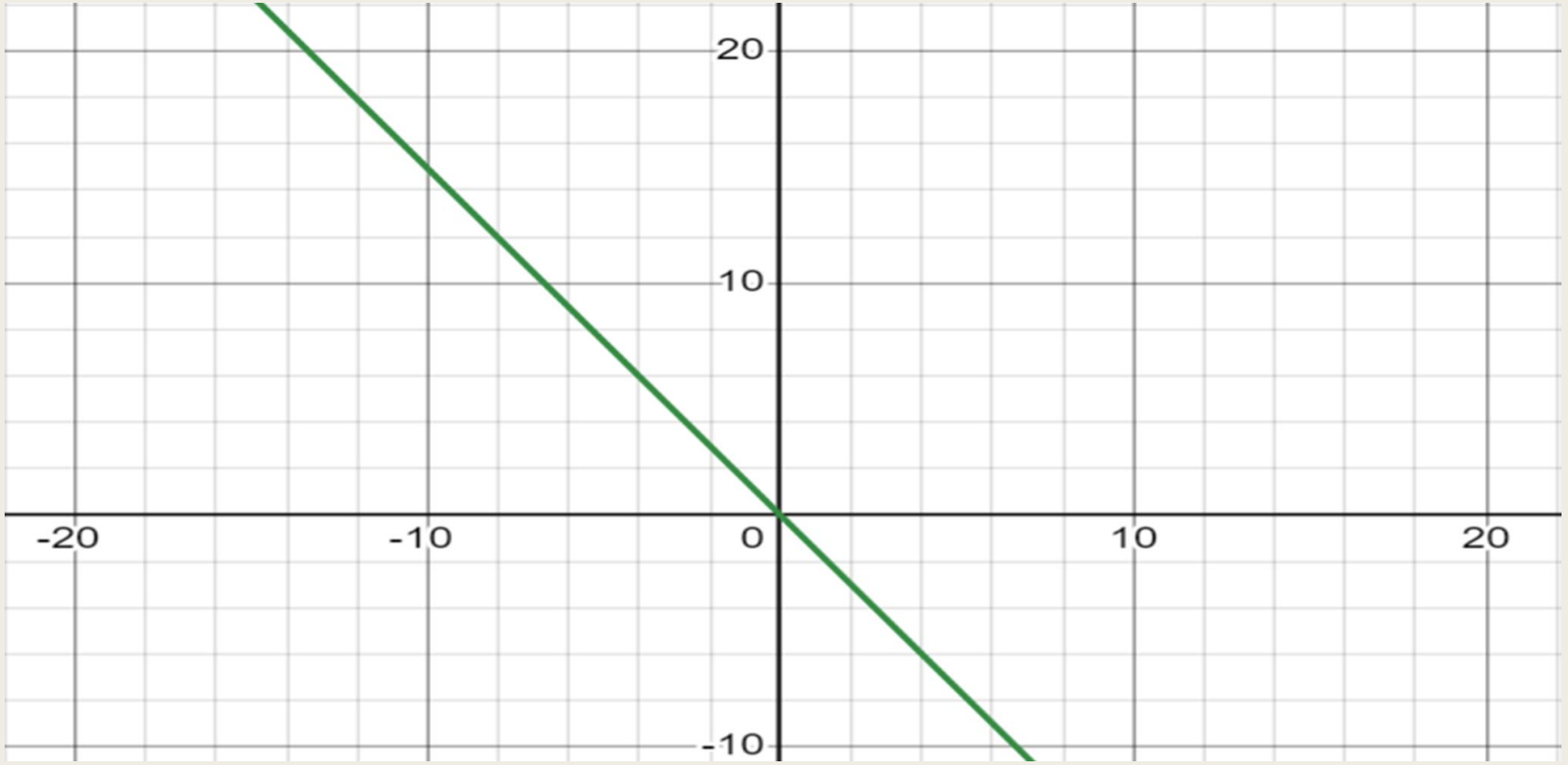
- We use the following equation to find the eigen vector-
- $MX = \lambda X$
- where- M = Covariance Matrix
- X = Eigen vector
- λ = Eigen value
- Substituting the values in the above equation, we get-

$$\begin{bmatrix} 10.5 & -8.25 \\ -8.25 & 17.25 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} = 22.79 \begin{bmatrix} X1 \\ X2 \end{bmatrix}$$

- Solving these, we get-
- 10.5X1 - 8.25X2 = 22.79X1
- -8.25X1 + 17.25X2 = 22.79X2
- On simplification, we get-
- 12.29X1 = -8.25X2 ……(1)
- -8.25X1 = 5.54X2 ……(2)
- X1/5.54=X2/-8.25
- From (1) and (2), X1 = -0.67X2

Previous Year Question (PCA)

- From (2), the eigen vector is- $\begin{bmatrix} 5.54 \\ -8.25 \end{bmatrix}$
- Thus, principal component for the given data set is- $\begin{bmatrix} 5.54 \\ -8.25 \end{bmatrix}$
- Lastly, we project the data points onto the new subspace as-



Previous Year Question (PCA)

- 2. Explain the Principal Component Analysis (PCA) and reduce the following dataset step-by-step from 2 dimensions to 1.

Feature	Example 1	Example 2	Example 3	Example 4
x	2	1	0	-1
y	4	3	1	0.5

- **Ans: Homework.**

- 3. Create the reduced dimension of the given data from 4 to 2 using Principal Component Analysis (PCA).

Sampales	A	B	C	D
1	14	18	13	7
2	40	4	5	14
3	87	11	13	23
4	8	15	8	45

Previous Year Question (PCA)

- **Ans:-**
- **Step-01:**
- **Get data.**
- The given feature vectors are-
- x1 = (14, 18, 13, 7)
- x2 = (40, 4, 5, 14)
- x3 = (87, 11, 13, 23)
- x4 = (8, 15, 8, 45)

- **Step-02:**
- **Calculate the mean vector (μ).**
- Mean vector (μ) = ((14 + 40 + 87 + 8) / 4, (18 + 4 + 11 + 15) / 4, (13 + 5 + 13 + 8) / 4, (7 + 14 + 23 + 45) / 4) = (37.25, 12, 9.75, 22.25)

Previous Year Question (PCA)

- **Step-03:**
- **Subtract mean vector (μ) from the given feature vectors.**
- $x1 - \mu = (14-37.25, 18-12, 13-9.75, 7-22.25) = (-23.25, 6, 3.25, 15.25)$
- $x2 - \mu = (40-37.25, 4-12, 5-9.75, 14-22.25) = (2.75, -8, -4.75, -8.25)$
- $x3 - \mu = (87-37.25, 11-12, 13-9.75, 23-22.25) = (49.75, -1, 3.25, 0.75)$
- $x4 - \mu = (8-37.25, 15-12, 8-9.75, 45-22.25) = (-29.25, 3, -1.75, 22.75)$

- **Step-04:**
- **Calculate the covariance matrix.**
- $m1 = (x1 - \mu)(x1 - \mu)T = (-23.25, 6, 3.25, 15.25)(-23.25, 6, 3.25, 15.25)T =$

$$\begin{bmatrix} 540.5625 & -139.5 & -75.5625 & -354.5625 \\ -139.5 & 36 & 19.5 & 91.5 \\ -75.5625 & 19.5 & 10.5625 & 49.5625 \\ -354.5625 & 91.5 & 49.5625 & 232.5625 \end{bmatrix}$$

- $m2 = (x2 - \mu)(x2 - \mu)T = (2.75, -8, -4.75, -8.25)(2.75, -8, -4.75, -8.25)T =$

$$\begin{bmatrix} 7.5625 & -22 & -13.0625 & -22.6875 \\ -22 & 64 & 38 & 66 \\ -13.0625 & 38 & 22.5625 & 39.1875 \\ -22.6875 & 66 & 39.1875 & 68.0625 \end{bmatrix}$$

Previous Year Question (PCA)

- $m3 = (x3 - \mu)(x3 - \mu)^T = (49.75, -1, 3.25, 0.75)(49.75, -1, 3.25, 0.75)^T =$

2475.0625	-49.75	161.6875	37.3125
-49.75	1	-3.25	-0.75
161.6875	-3.25	10.5625	2.4375
37.3125	-0.75	2.4375	0.5625

- $m4 = (x4 - \mu)(x4 - \mu)^T = (-29.25, 3, -1.75, 22.75)(-29.25, 3, -1.75, 22.75)^T =$

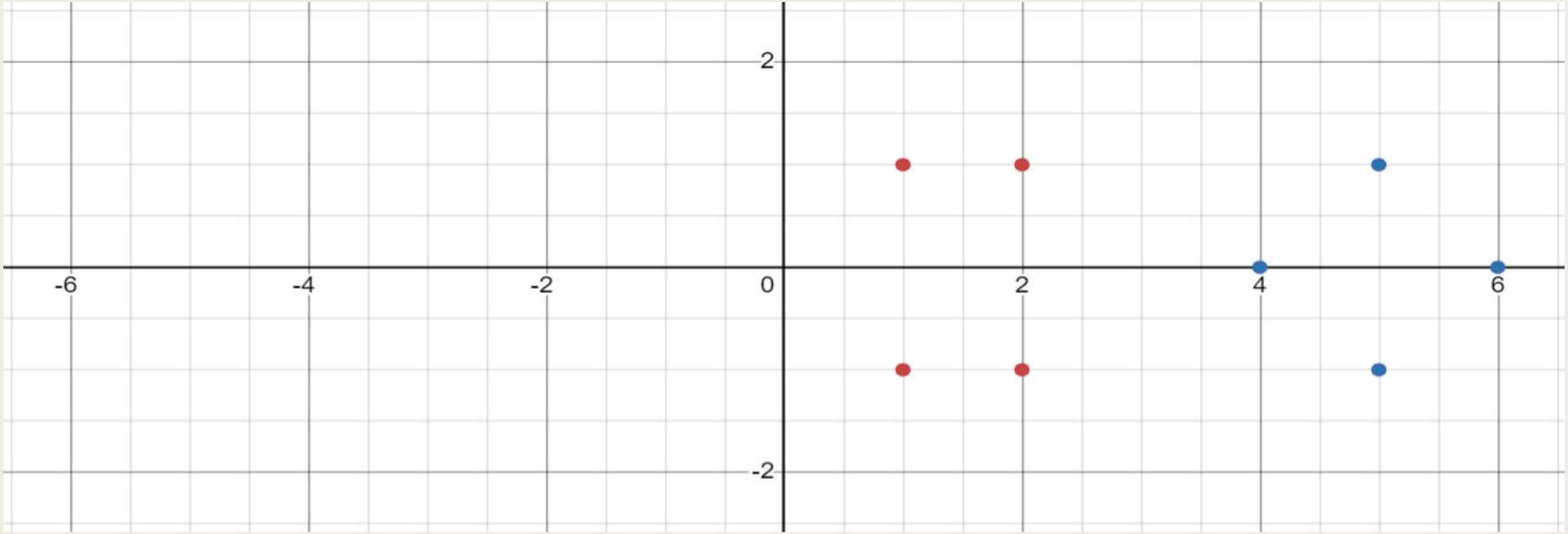
855.5625	-87.75	51.1875	-665.4375
-87.75	9	-5.25	68.25
51.1875	-5.25	3.0625	-39.8125
-665.4375	68.25	-39.8125	517.5625

- Covariance matrix = (m1 + m2 + m3 + m4) / 4
- **From here rest of the solution is Homework.**

Previous Year Question (SVM)

- 1. Plot a hyper plane for the given data set (1,1) (2,1) (1,-1) (2,-1) (4,0) (5,1) (5,-1) (6,0) by using SVM.

- **Ans: First we plot the data points on a 2-D plane as below:-**



- S1 = $\begin{pmatrix} \mathbf{2} \\ \mathbf{1} \end{pmatrix}$, S2 = $\begin{pmatrix} \mathbf{2} \\ \mathbf{-1} \end{pmatrix}$, S3 = $\begin{pmatrix} \mathbf{4} \\ \mathbf{0} \end{pmatrix}$,

Previous Year Question (SVM)

- **Step2:- Augment each vector with 1 as bias input:**
- $\widetilde{S1} = \begin{pmatrix}2\\1\\1\end{pmatrix}, \widetilde{S2} = \begin{pmatrix}2\\-1\\1\end{pmatrix}, \widetilde{S3} = \begin{pmatrix}4\\0\\1\end{pmatrix}$
- **Step3:- Our task is to find values for the α_i such that**

$\alpha 1.\ \widetilde{S1}.\widetilde{S1} + \alpha 2.\ \widetilde{S2}.\widetilde{S1} + \alpha 3.\ \widetilde{S3}.\widetilde{S1} = -1$

$\alpha 1.\ \widetilde{S1}.\widetilde{S2} + \alpha 2.\ \widetilde{S2}.\widetilde{S2} + \alpha 3.\ \widetilde{S3}.\widetilde{S2} = -1$

$\alpha 1.\ \widetilde{S1}.\widetilde{S3} + \alpha 2.\ \widetilde{S2}.\widetilde{S3} + \alpha 3.\ \widetilde{S3}.\widetilde{S3} = 1$

$\alpha 1. \begin{pmatrix}2\\1\\1\end{pmatrix}.\begin{pmatrix}2\\1\\1\end{pmatrix} + \alpha 2. \begin{pmatrix}2\\-1\\1\end{pmatrix}.\begin{pmatrix}2\\1\\1\end{pmatrix} + \alpha 3. \begin{pmatrix}4\\0\\1\end{pmatrix}.\begin{pmatrix}2\\1\\1\end{pmatrix} = -1$

$\alpha 1. \begin{pmatrix}2\\1\\1\end{pmatrix}.\begin{pmatrix}2\\-1\\1\end{pmatrix} + \alpha 2. \begin{pmatrix}2\\-1\\1\end{pmatrix}.\begin{pmatrix}2\\-1\\1\end{pmatrix} + \alpha 3. \begin{pmatrix}4\\0\\1\end{pmatrix}.\begin{pmatrix}2\\-1\\1\end{pmatrix} = -1$

$\alpha 1. \begin{pmatrix}2\\1\\1\end{pmatrix}.\begin{pmatrix}4\\0\\1\end{pmatrix} + \alpha 2. \begin{pmatrix}2\\-1\\1\end{pmatrix}.\begin{pmatrix}4\\0\\1\end{pmatrix} + \alpha 3. \begin{pmatrix}4\\0\\1\end{pmatrix}.\begin{pmatrix}4\\0\\1\end{pmatrix} = 1$

Previous Year Question (SVM)

$\alpha1(4+1+1) + \alpha2(4-1+1) + \alpha3(8+0+1) = -1$

$\alpha1(4-1+1) + \alpha2(4+1+1) + \alpha3(8+0+1) = -1$

$\alpha1(8+0+1) + \alpha2(8+0+1) + \alpha3(16+0+1) = 1$

$6\alpha1 + 4\alpha2 + 9\alpha3 = -1$

$4\alpha1 + 6\alpha2 + 9\alpha3 = -1$

$9\alpha1 + 9\alpha2 + 17\alpha3 = 1$

By solving above eqn we get,

$\alpha1 = -3.25$

$\alpha2 = -3.25$

$\alpha3 = 3.5$

$\tilde{w} = \alpha1.\widetilde{S1} + \alpha2.\widetilde{S2} + \alpha3.\widetilde{S3}$

$$= -3.25\begin{pmatrix}2\\1\\1\end{pmatrix} + -3.25\begin{pmatrix}2\\-1\\1\end{pmatrix} + 3.5\begin{pmatrix}4\\0\\1\end{pmatrix} = \begin{pmatrix}1\\0\\-3\end{pmatrix}$$

- So the line eqn is
- $1.x + 0.y - 3 = 0$ or **$x-3 = 0$**

Previous Year Question (SVM)

- **The optimum hyperplane is shown below:-**