LAB 1

Title: **K nearest Neighbor (KNN)**

Objectives:

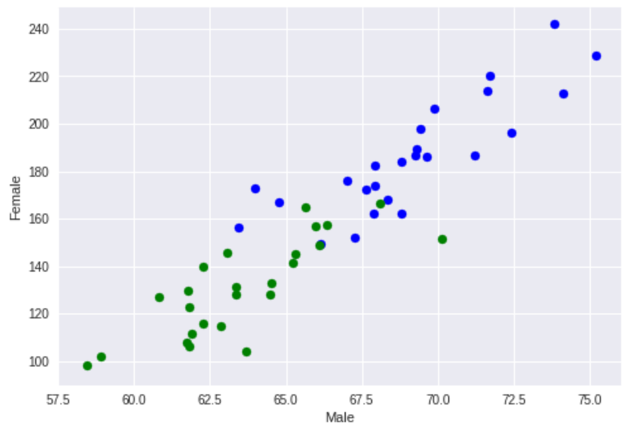
1. K nearest neighbor can classify data by measuring distance.
2. Easy to implement.
3. Easiest classification algorithm to understand.

Methodology: K Nearest Neighbors outputs a class membership of a feature set by a majority vote of its closest neighbors. Here k is a number that indicates the number of data points that are closed or under the minimum distance area. There are some steps for KNN.

1. Input and Output

Height	Weight	Gender
73.84701702	241.89356318	Male
68.78190405	162.31047252	Male
66.1038728	148.64518257	Female
64.52718203	132.68086824	Female

2. Visualizing Dataset



3. Now we will predict gender for unknown data. Suppose our test data is :

Height	Weight
76	252.1

We will use Euclidean distance for measuring distance between test and train data. Euclidean Distance:

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

So 1st distance d1 is=

$$d1 = \sqrt{(76 - 73.1)^2 + (252.1 - 241.89356)^2}$$
= 11.41

According to this rule d2= 90.07, d3= 119.48 and d4=105.53

4. The next step is sorting distance ascending and choosing best k distance with their gender from top. So here k=3 so best 3 distances are d1= [11.41, male], d2= [90.07, male] and d3= [105.53, female].

5. The last step is voting occurrence according to their gender. Here, male is 2 times and female is one times. So, male is winner. The predicted gender of test dataset is male.

Result:

Training	**Testing**	**Accuracy**
100%	100%	94%
80%	20%	80%

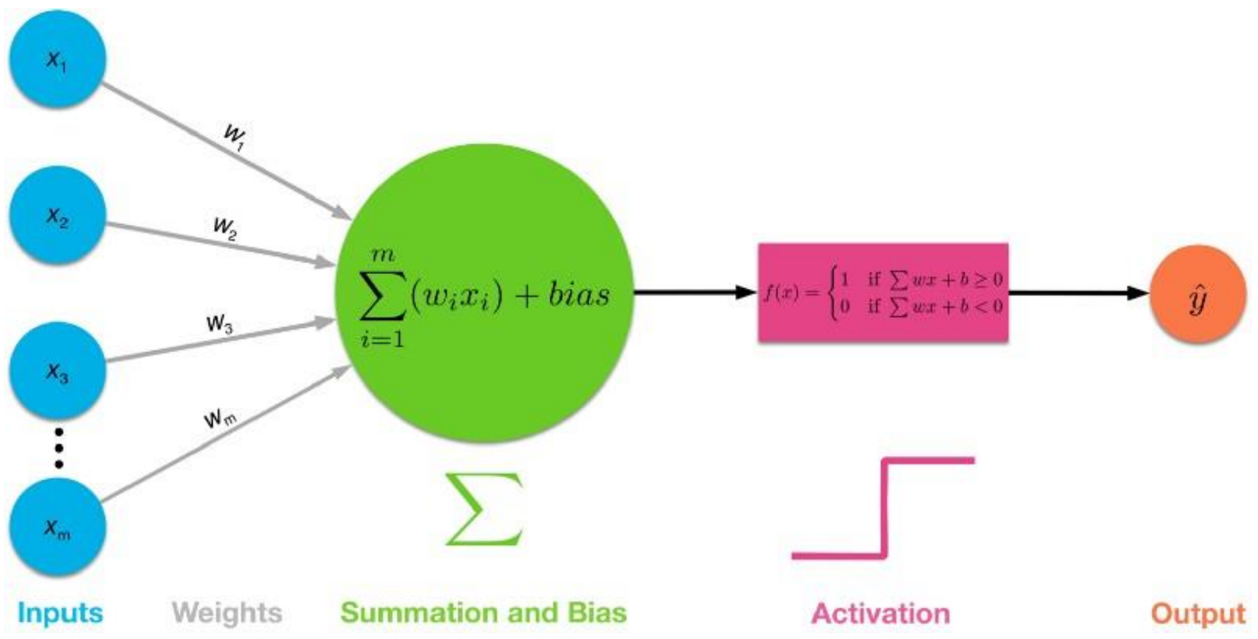
Conclusion: While KNN produces good classifiers, it takes a lot of time to predict outputs for large data sets. This is because Euclidean distance must be calculated for all training samples in order to find the nearest samples in the data.

Code: https://github.com/shawon100/machine-learning/tree/master/K%20Nearest%20Neighbour

Title: **Single Layer Perceptron (Widrow Hoff delta rule)**

Objectives:

1. Single layer perceptron can classify binary classes.
2. It is used for linear relationship classification.
3. It has 0/1 threshold function that is used on weighted sum identification.


x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
Inputs
Weights
Summation and Bias
Activation
Output

Input and Output:

X	Y
0 0 0 0 0 0 0 0 0 0	0
0 0 0 0 0 0 0 0 0 1	0
…………….	
0 0 0 0 0 0 1 0 0 1	1
0 0 0 0 0 0 1 0 1 0	1

Methodology:

1. Initialize weights and thresholds randomly.
2. Present Input x0, x1, x2 … and desired output d(t)
3. Calculate actual output

$$\boldsymbol{y(t) = f_h \sum_{i=1}^{n} w_i(t) x_i(t)}$$

$\boldsymbol{f_h}$ is a hard limiting threshold function.

4. Adapt Weights

$$\Delta = \boldsymbol{d(t) - y(t)}$$

$$\boldsymbol{w_i(t+1) = w_i(t) + \alpha \Delta x_i(t)}$$

$$d(t) = \begin{cases} +1, & if\ input\ from\ class\ A \\ 0, & if\ input\ from\ class\ B \end{cases}$$

Result:

Training	Testing	Accuracy
100%	100%	98%

Conclusion: Though single layer perceptron can classify binary classes, it cannot classify those binary classes that have nonlinear relationship. Because it has no such a layer that can store common features and information.

Code: https://github.com/shawon100/Neural-Network/tree/master/Lab%201%20(single%20layer%20neural%20network)

LAB 2

Title: **Multi-Layer Perceptron (Backpropagation)**

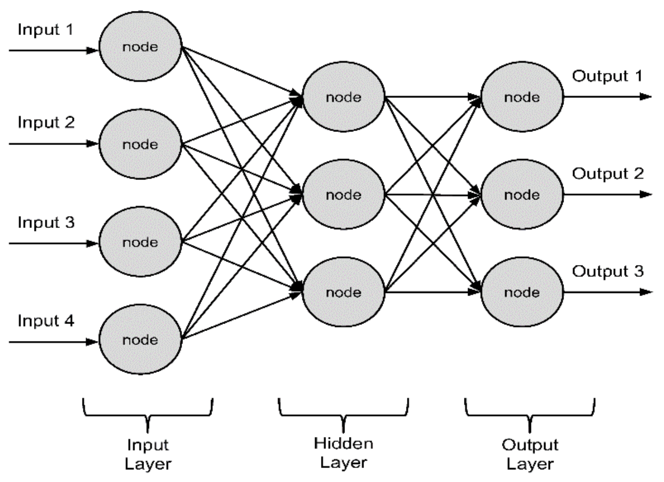
Objectives:

1. MLP can solve more complex problems.
2. It can classify multi class.
3. It can also solve nonlinear relationship based problems using sigmoid function.

Input and Output:

X	Y
0 0 0 0 0 0 0 0 0 0	0 0 0
0 0 0 0 0 0 0 0 0 1	0 0 1
0 0 0 0 0 0 0 0 1 0	0 1 0
0 0 0 0 0 0 0 0 1 1	0 1 1
0 0 0 0 0 0 0 1 0 0	1 0 0
0 0 0 0 0 0 0 1 0 1	1 0 1
0 0 0 0 0 0 0 1 1 0	1 1 0
0 0 0 0 0 0 0 1 1 1	1 1 1
0 0 0 0 0 0 1 0 0 1	0 0 0
0 0 0 0 0 0 1 0 1 0	0 0 1
0 0 0 0 0 0 1 0 1 1	0 1 0

Methodology:



1. For better understanding, we will use a simple network. This architecture has 10 input nodes, one hidden layer with 3 nodes and an output layer with 3 nodes. So there are total 8 classes. The first step is randomly initializing weights W_{ij} , W_{jk} and threshold values.

2. In second step, the network provides the input patterns and the desired respective output patterns.

3. In third step, the input patterns are connected to the hidden nodes through weights W_{ij} . In the hidden layer each nodes value is computed according to weighted sum net_{aj}

$$net_{aj} = \sum_{i=1}^{n} W_{ij}\, O_{ai} \qquad ------------ (1)$$

4. Here O_{ai} is the input of unit I for pattern a. The threshold of each node is added to it's weighted sum.

$$activ_j = net_{aj} + uh_j \qquad ------------ (2)$$

5. Here uh_j is the hidden threshold weights. Passing this value to sigmoid function we can set the value in the range of 0 to 1.

$$O_{aj} = \frac{1}{1 + e^{-k1*activ_j}} \qquad ----------(3)$$

6. Here k_1 is called the spread factors. These O_{aj} is now served as the input to the output computation. Now we will calculate weighted sum for weight W_{jk}

$$net_{ak} = \sum_{i=1}^{n} W_{jk}\, O_{aj} \qquad ------------ (4)$$

7. Now output threshold uo_k will be added with net_{ak}

$$activ_k = net_{ak} + uo_k \qquad ------------ (5)$$

8. The actual output O_{ak} is computed by sigmoid function

$$O_{ak} = \frac{1}{1 + e^{-k2*activ_k}} \qquad ----------(6)$$

9. After computing the feed forward propagation, an error is computed by comparing the output O_{ak} with respective output t_{ak}

$$\delta_{ak} = t_{ak} - O_{ak} \text{ ------------------------------------ } (7)$$

10. This error is then used to adjust the weights vector W_{jk} using the equation,

$$\Delta W_{jk} = n_2\, k_2\, \delta_{ak} O_{aj} O_{ak}(1 - O_{ak}) \text{ ------------------------------- } (8)$$

11. So the updated new weights are

$$W_{jk} = W_{jk} + \Delta W_{jk} \text{ ------------------------------ } (9)$$

12. Threshold should also be updated like weights.

$$\Delta uo_k = n_2\, k_2\, \delta_{ak} O_{ak}(1 - O_{ak}) \text{ ----------------------------- } (10)$$

13. So the new threshold is now changed.

$$uo_k = uo_k + \Delta uo_k \text{ ---------------------- } (11)$$

14. Next we will update weights of W_{ij}

$$\Delta W_{ij} = n_1\, k_1\, O_{ai} O_{aj}(1 - O_{aj}) \sum_{k=0}^{n-1} \delta_{ak}\, W_{jk} \text{ ------------------------------- } (12)$$

15. The updated Weights

$$W_{ij} = W_{ij} + \Delta W_{ij} \text{ --------------------------- } (13)$$

16. Threshold updating

$$\Delta uh_j = n_1\, k_1\, O_{aj}(1 - O_{aj}) \sum_{k=0}^{n-1} \delta_{ak}\, W_{jk} - - - - - - - - - - - - (13)$$

17. Updated threshold

$$uh_j = uh_j + \Delta uh_j \text{ ---------------------- } (14)$$

18. Thus the network will update weights and thresholds until error rate reaches to minimum.

Result:

Training	**Testing**	**Accuracy**
100%	100%	83.33%
60%	40%	65.77%
10%	90%	49.34%

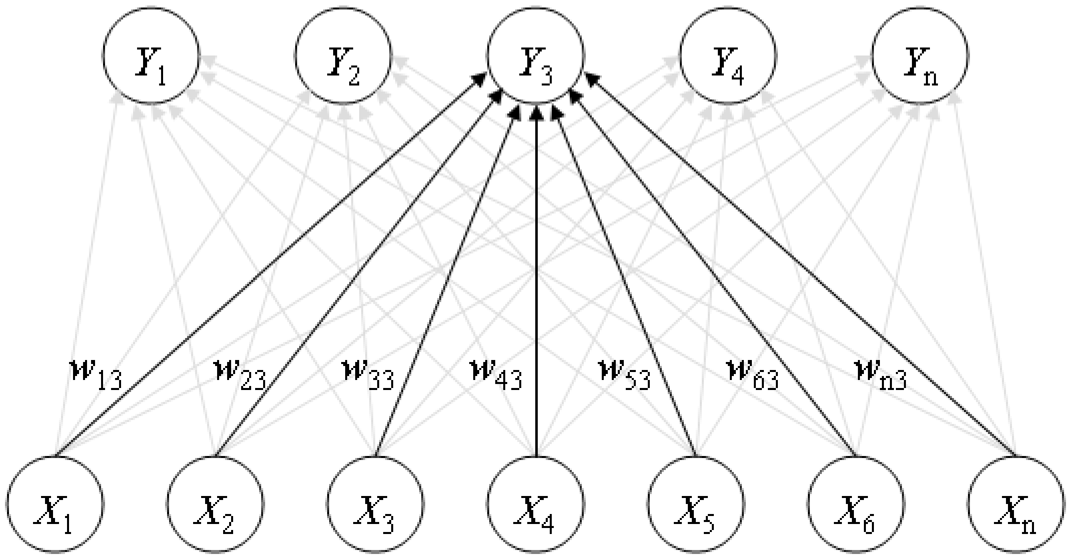
Conclusion: The performance of multi-layer perceptron depends on number of hidden layers, number of hidden nodes, learning rate, weight initialization, error calculation etc. This result can be improved by adding more hidden nodes or changing mean square error to cross entropy error.

Code: https://github.com/shawon100/Neural-Network/tree/master/Lab%202%20Multilayer%20Perception

LAB 3

Title: **Kohenen Self Organizing Network**

Objectives: The objective of a Kohonen network is to map input vectors (patterns) of arbitrary dimension N onto a discrete map with 1 or 2 dimensions. Kohenen can extract common features and cluster by unsupervised learning.



Methodology:

1. Initialize Network: Initialize weights randomly and set the radius large.
2. Define inputs.
3. Calculate Distances :

$$d_j = \sum_{i=0}^{n-1} (x_i (t) - w_{ij} (t))^2$$

4. Select minimum distance.
5. According to minimum distance select the weight column and update

$$\boldsymbol{w_{ij}(t+1) = w_{ij}(t) + \alpha(t)}(x_i (t) - w_{ij} (t))$$

The gain term decreases in time so slowing the weight adaptation. Radius also decreases and thus it localizes the area of maximum activity.

6. Repeat by going to 2.

Result:

Input	Output
0 0 0 0 0 0 0 0 0 0	3
0 0 0 0 0 0 0 0 0 1	3
0 0 0 0 0 0 0 1 0 0	1
0 0 0 0 0 0 0 1 0 1	1
0 0 0 0 0 0 0 1 1 0	1
0 0 0 0 0 0 0 1 1 1	1
0 0 0 0 0 0 1 0 1 1	5

Training	**Testing**	**Accuracy**
100%	100%	64%
60%	40%	26.66%

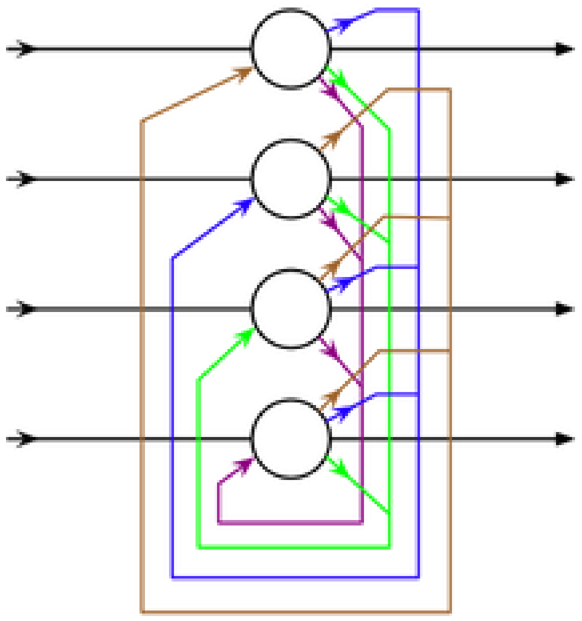
Conclusion: Self-organizing neural networks are used to cluster input patterns into groups of similar patterns. They're called "maps" because they assume a topological structure among their cluster units; effectively mapping weights to input data. The Kohonen network is probably the best example, because it's simple, yet introduces the concepts of self-organization and unsupervised learning easily.

Code: https://github.com/shawon100/Neural-Network/tree/master/Lab%203

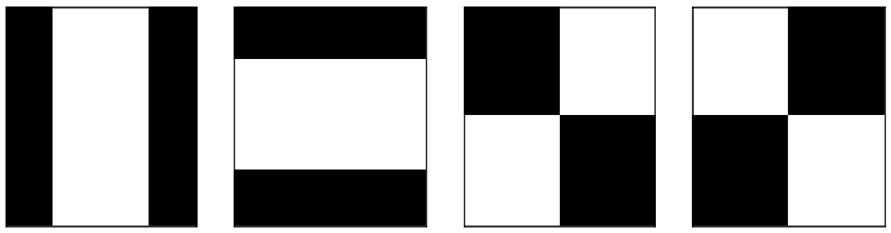
LAB 4

Title: **Hopfield Network**

Objectives: The objective of Hopfield network is to convert a wrong pattern to a right pattern. Recovering pattern is the important task of Hopfield network. It's a fully connected network.



Input:



Methodology:

1. Assign connection weights

$$w_{ij} = \begin{cases} \sum_{s=0}^{M-1} x_i\, x_j & i \neq j \\ 0 & i = j \end{cases}$$

This equation is described below:
Suppose inputs are x1= [+1, -1 ,+1 ,-1] and x2= [+1,+1,+1,+1] . Our matrix will be 4*4.
$w_{11} = 0,\ w_{22} = 0\ ,\ w_{33} = 0\ ,\ w_{44} = 0$

For x1, $w_{12} = -1\ ,\ w_{13}$ =1, $w_{14} = -1\ ,\ w_{23}$ = -1 $w_{24} = 1\ ,\ w_{34} = -1$
For x2, $w_{12} = 1\ ,\ w_{13}$ =1, $w_{14} = 1\ ,\ w_{23}$ =1, $w_{24} = 1,\ w_{34} = 1$

After summation, the final matrix will look like:

$$w_{ij} = \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \\ 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

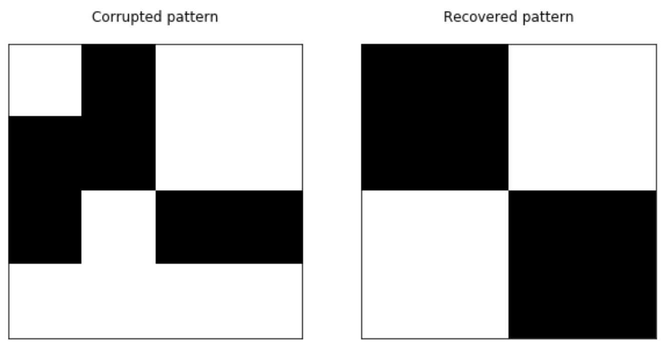
2. Initialize with unknown pattern
   Suppose unknown pattern is μ= [-1, +1, +1, -1]

3. Iterate until convergence

$$\mu_i(\boldsymbol{t}+\mathbf{1}) = \boldsymbol{f_h} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}-\mathbf{1}} \boldsymbol{w_{ji}}(\boldsymbol{t})\mu_j(\boldsymbol{t})$$

Result:

After some iteration, the recovered pattern is = [+1, -1, +1,-1] that is similar to input pattern.



Conclusion: By Hopfield network corrupted patterns are guaranteed to converge to a local minimum, but will sometimes converge to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum). Hopfield networks also provide a model for understanding human memory.

Code: https://github.com/shawon100/Neural-Network/tree/master/Lab%204%20Hopfield%20Network