**Topics: Self Organizing Maps (SOM)**

## What we are going to cover in this lecture?

1. What is Self Organizing Maps?
2. SOMs Network Architecture.
3. How Self Organizing Maps work.
4. Practical Implementation of SOMs.

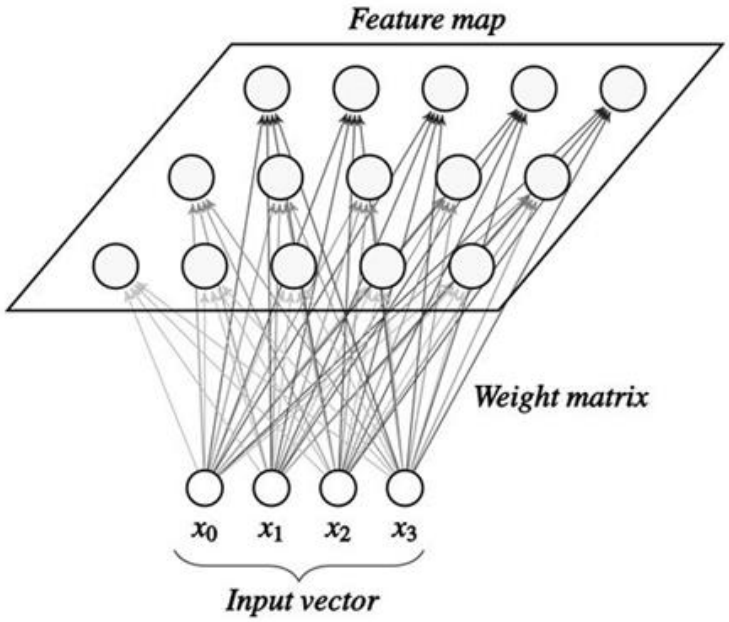
## 1: What is Self Organization Maps?

The Self Organizing Map is one of the most popular neural models. It belongs to the category of the competitive learning network. The SOM is based on unsupervised learning, which means that no human intervention is needed during the training and those little needs to be known about characterized by the input data. We could, for example, use the SOM for clustering membership of the input data. The SOM can be used to detect features inherent to the problem and thus has also been called SOFM the Self Origination Feature Map.

The Self Organized Map was developed by professor Kohonen which is used in many applications.

the purpose of SOM is that it's providing a data visualization technique that helps to understand high dimensional data by reducing the dimension of data to map. SOM also represents the clustering concept by grouping similar data together.

Therefore it can be said that Self Organizing Map reduces data dimension and displays similarly among data.




## 3. Self Organization Maps Network Architecture?

For this purpose, we'll be discussing a two-dimensional SOM. The network is created from a 2D lattice of 'nodes', each of which is fully connected to the input layer. The below Figure shows a very small

Kohonen network of 4 X 4 nodes connected to the input layer (shown in green) representing a two-dimensional vector.

Each node has a specific topological position (an x, y coordinate in the lattice) and contains a vector of weights of the same dimension as the input vectors. That is to say if the training data consists of vectors, *V*, of *n* dimensions:

V1, V2, V3…Vn

Then each node will contain a corresponding weight vector *W,* of *n* dimensions:

W1, W2, W3…Wn

The lines connecting the nodes in the above Figure are only there to represent adjacency and do not signify a connection as normally indicated when discussing a neural network. There are no lateral connections between nodes within the lattice.

## 4. How Self Organization Maps work?

### 4.1: Learning Algorithm Overview

A SOM does not need a target output to be specified unlike many other types of networks. Instead, where the node weights match the input vector, that area of the lattice is selectively optimized to more closely resemble the data for the class the input vector is a member of. From an initial distribution of random weights, and over many iterations, the SOM eventually settles into a map of stable zones. Each zone is effectively a feature classifier, so you can think of the graphical output as a type of feature map of the input space.

**Training occurs in several steps and over many iterations:**

1. Each node's weights are initialized.

2. A vector is chosen at random from the set of training data and presented to the lattice.

3. Every node is examined to calculate which ones weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).

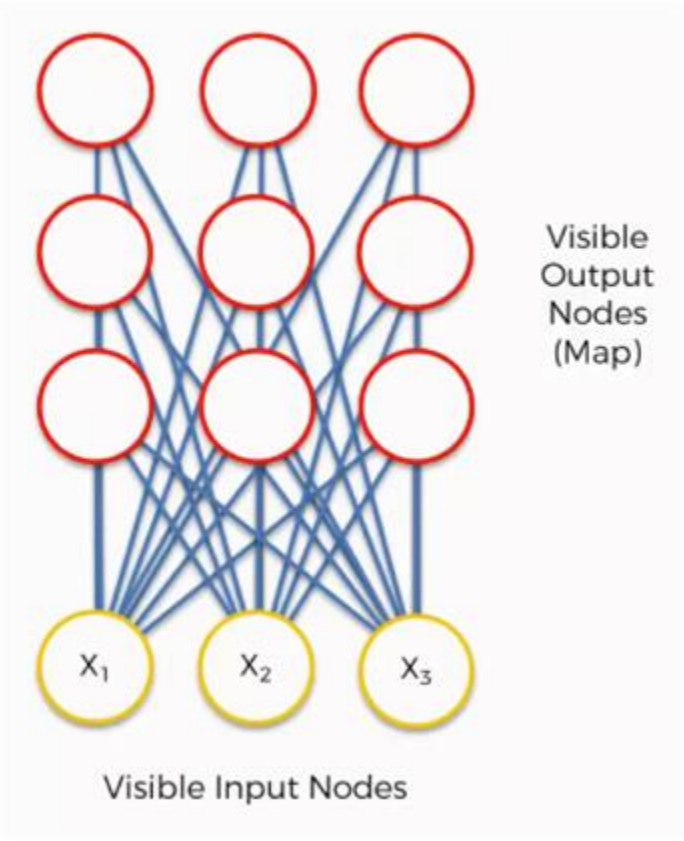
4. The radius of the neighbourhood of the BMU is now calculated. This is a value that starts large, typically set to the 'radius' of the lattice, but diminishes each time step. Any nodes found within this radius are deemed to be inside the BMU's neighbourhood.

5. Each neighbouring node's (the nodes found in step 4) weights are adjusted to make them more like the input vector. The closer a node is to the BMU; the more its weights get altered.

6. Repeat step 2 for N iterations.

## 4.2: Learning Algorithm in Details.

Now it's time for us to learn how SOMs learn. Are you ready? Let's begin. Right here we have a very basic self-organizing map.

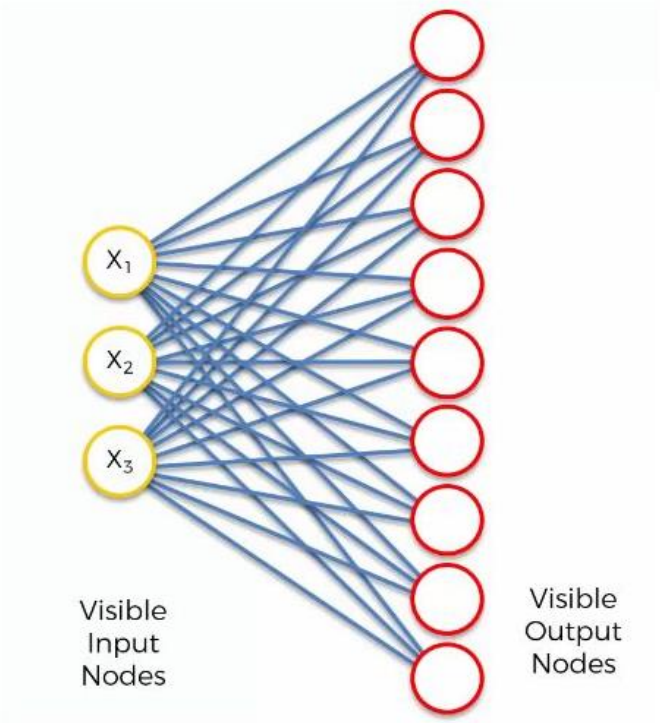



Our input vectors amount to three features, and we have nine output nodes.

That being said, it might confuse you to see how this example shows three input nodes producing nine output nodes. Don't get puzzled by that. The three input nodes represent three columns

(dimensions) in the dataset, but each of these columns can contain thousands of rows. The output nodes in a SOM are always two-dimensional.
Now what we'll do is turn this SOM into an input set that would be more familiar to you from when we discussed the supervised machine learning methods (artificial, convolutional, and recurrent neural networks) in earlier chapters.

Consider the Structure of Self Organizing which has 3 visible input nodes and 9 outputs that are connected directly to input as shown below fig.



Our input nodes values are:

$X_1 = 0.7$

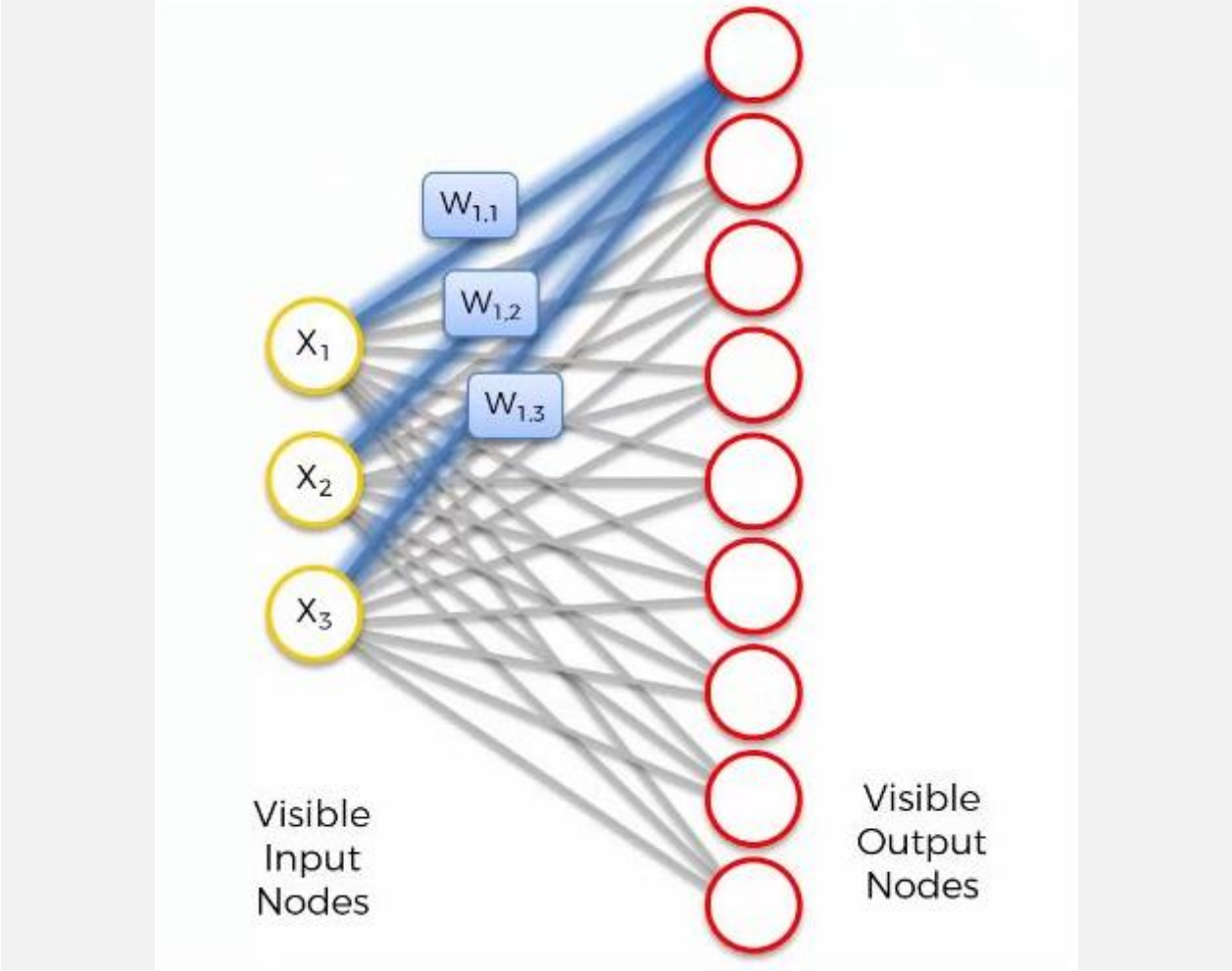
$X_2 = 0.6$

$X_3 = 0.9$

Now let's take a look at each step in detail.

## Step 1: Initializing the Weights

Now, let's take the topmost output node and focus on its connections with the input nodes. As you can see, there is a weight assigned to each of these connections.

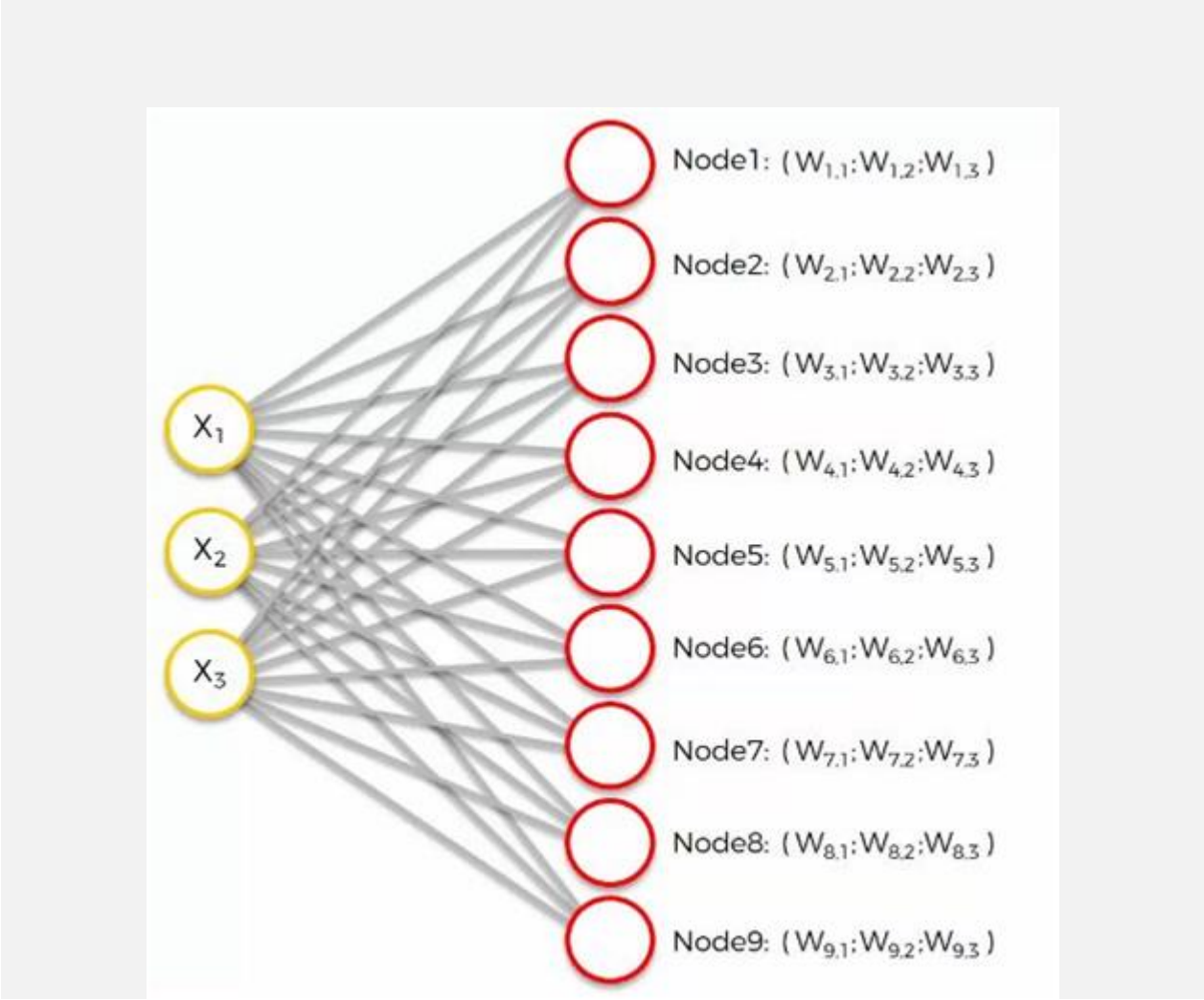



Again, the word “weight” here carries a whole other meaning than it did with artificial and convolutional neural networks. For instance, with artificial neural networks we multiplied the input node’s value by the weight and, finally, applied an activation function. With SOMs, on the other hand, there is no activation function.

Weights are not separate from the nodes here. In a SOM, the weights belong to the output node itself. Instead of being the result of adding up the weights, the output node in a SOM contains the weights as its coordinates. Carrying these weights, it sneakily tries to find its way into the input space.

In this example, we have a 3D dataset, and each of the input nodes represents an x-coordinate. The SOM would compress these into a single output node that carries three weights. If we happen to deal with a 20-dimensional dataset, the output node, in this case, would carry 20 weight coordinates.

Each of these output nodes does not exactly become parts of the input space, but try to integrate into it nevertheless, developing imaginary places for themselves.




We have randomly initialized the values of the weights (close to 0 but not 0).

| | | |
|---|---|---|
| $W_{1,1} = 0.31$ | $W_{1,2} = 0.22$ | $W_{1,3} = 0.10$ |
| $W_{2,1} = 0.21$ | $W_{2,2} = 0.34$ | $W_{2,3} = 0.19$ |
| $W_{3,1} = 0.39$ | $W_{3,2} = 0.42$ | $W_{3,3} = 0.45$ |
| $W_{4,1} = 0.25$ | $W_{4,2} = 0.32$ | $W_{4,3} = 0.62$ |
| $W_{5,1} = 0.24$ | $W_{5,2} = 0.31$ | $W_{5,3} = 0.16$ |
| $W_{6,1} = 0.52$ | $W_{6,2} = 0.33$ | $W_{6,3} = 0.42$ |
| $W_{7,1} = 0.31$ | $W_{7,2} = 0.22$ | $W_{7,3} = 0.10$ |
| $W_{8,1} = 0.12$ | $W_{8,2} = 0.41$ | $W_{8,3} = 0.19$ |
| $W_{9,1} = 0.34$ | $W_{9,2} = 0.40$ | $W_{9,3} = 0.51$ |

**Step 2: Calculating the Best Matching Unit**

The next step is to go through our dataset. For each of the rows in our dataset, we'll try to find the node closest to it.

Say we take row number 1, and we extract its value for each of the three columns we have. We'll then want to find which of our output nodes is closest to that row.

To determine the best matching unit, one method is to iterate through all the nodes and calculate the Euclidean distance between each node's weight vector and the current input vector. The node with a weight vector closest to the input vector is tagged as the BMU.

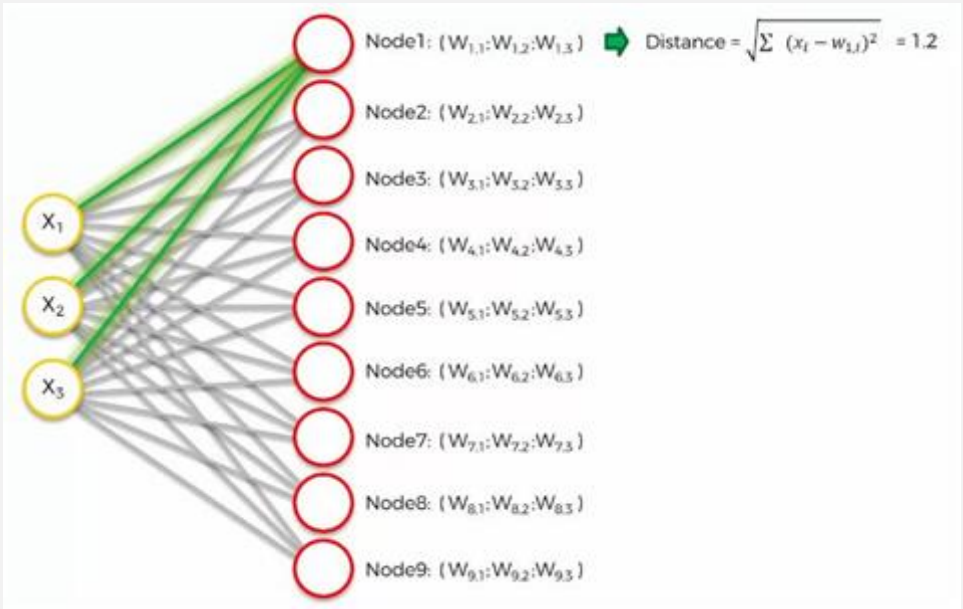
**The Euclidean distance is given as:**

$$\text{Distance} = \sqrt{\sum_{i=0}^{i=n}(X_i - W_i)^2}$$

Where X is the current input vector and W is the node's weight vector.

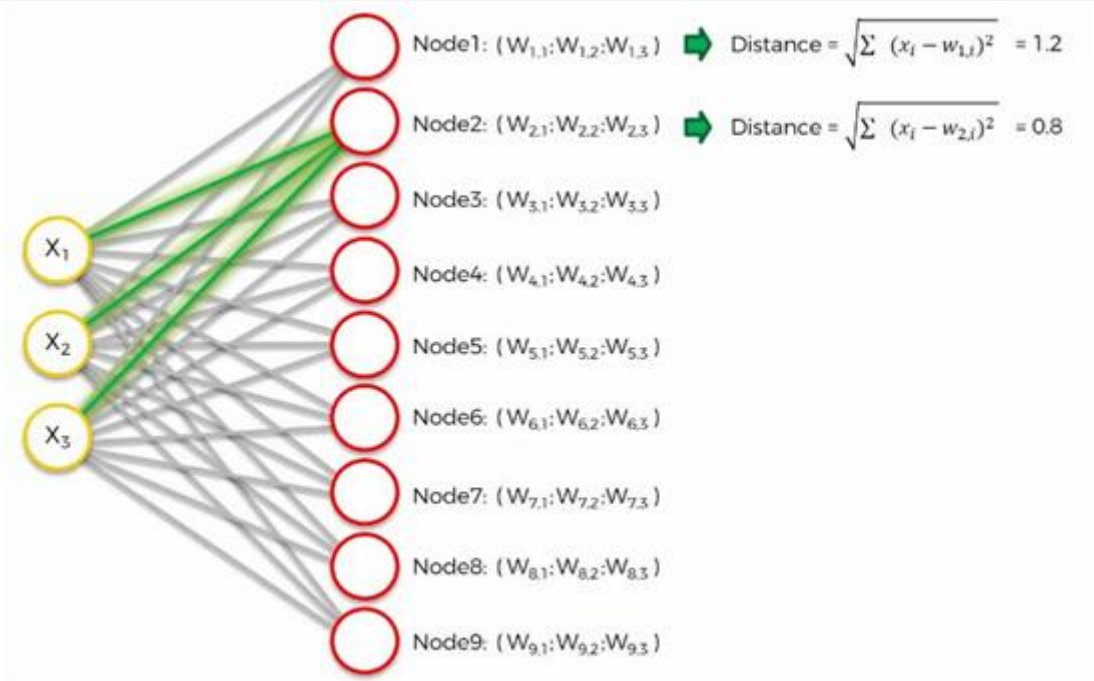
Let's calculate the Best Match Unit using the Distance formula.

For 1st Nodes:



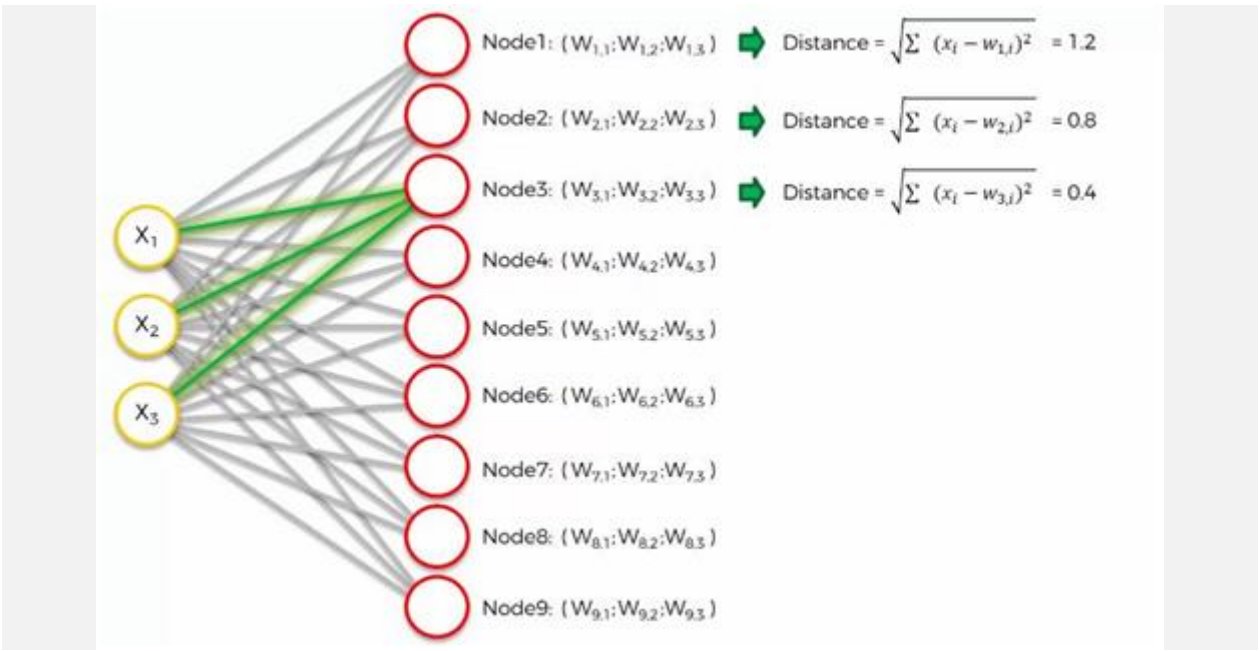

Distance = sqrt((x1-w1,1,)^2+(x2-w12)^2+(x3-w13)^2)

For 2nd Nodes:

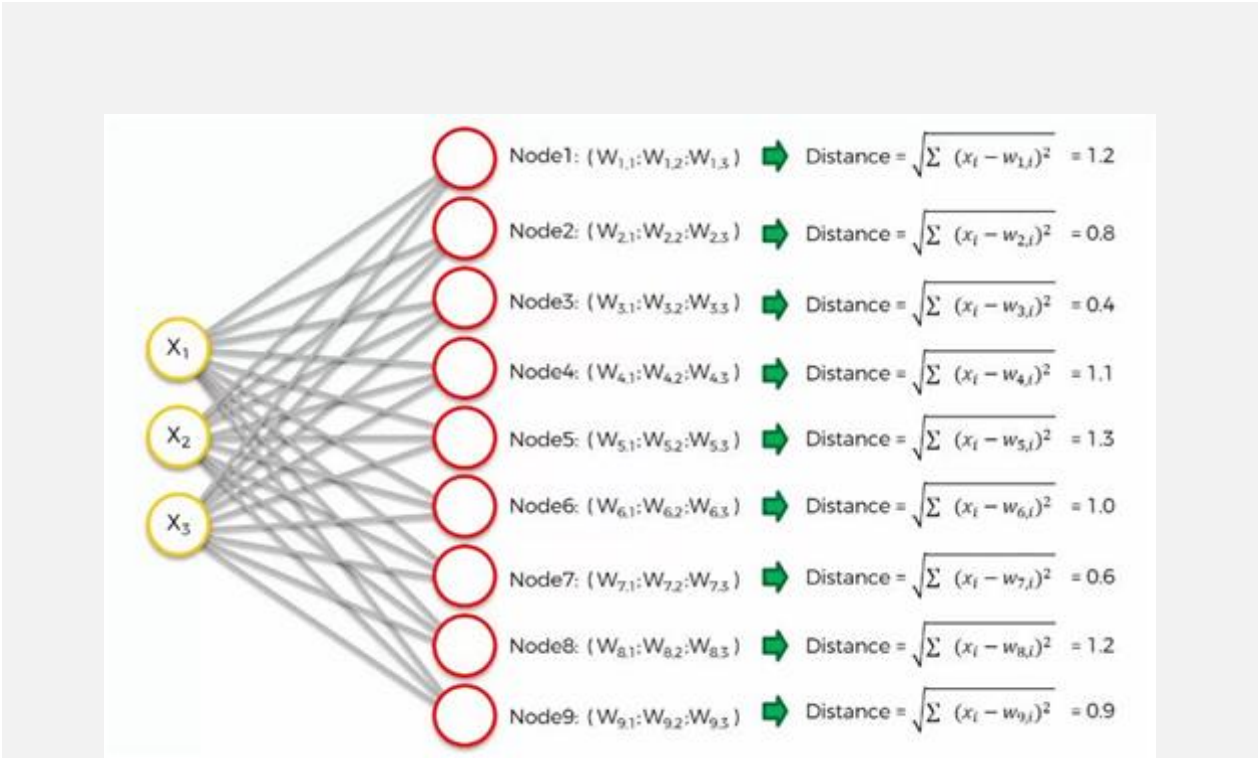



For 3rd Nodes:

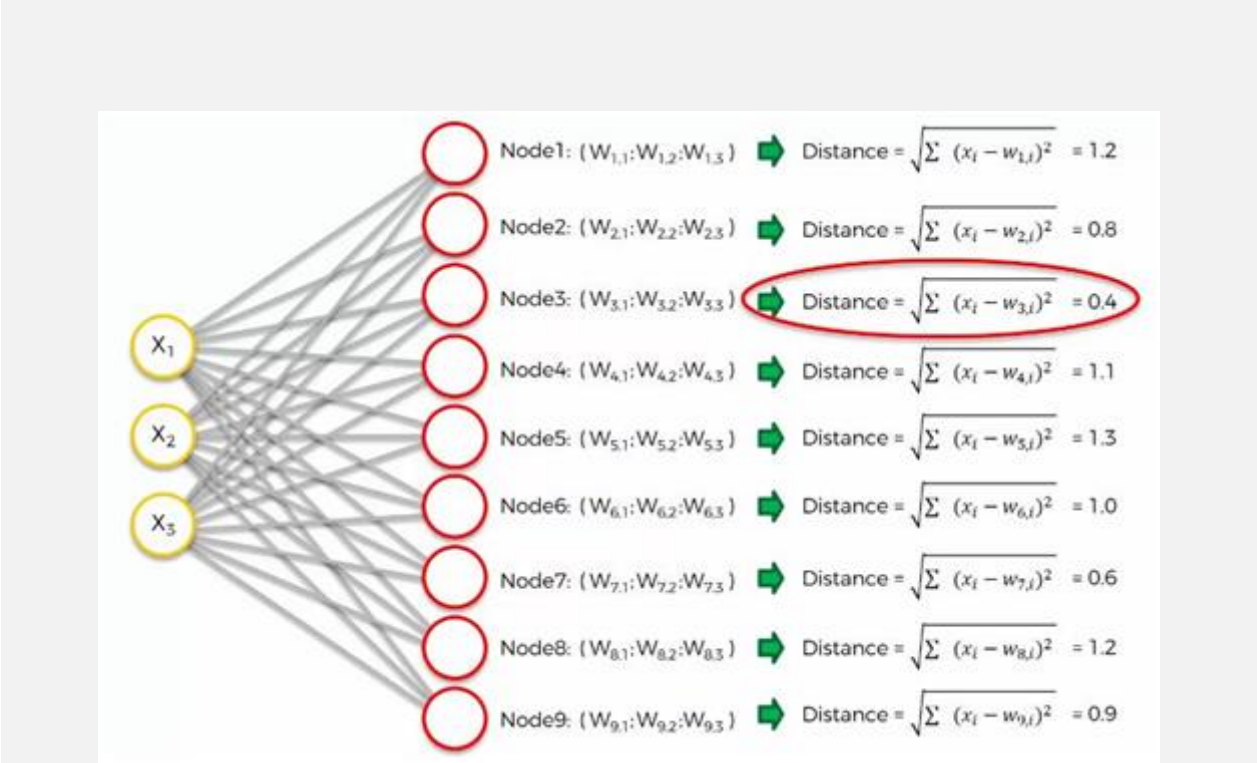
Similarly, way we calculate all remaining Nodes the same way as you can see below.




Similarly, we calculate all remaining Nodes the same way as you can see below.



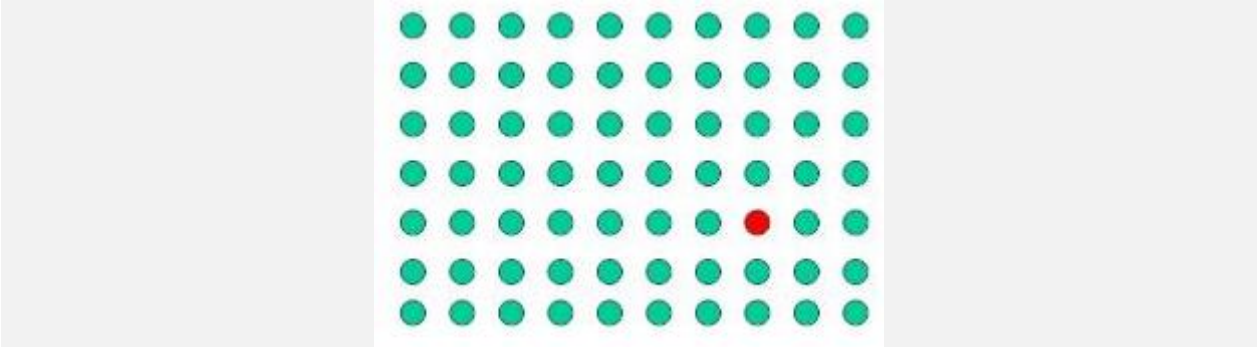
Since we have calculated all the values of respected Nodes. Now it's time to calculate the Best Match Unit.




As we can see, node number 3 is the closest with a distance of 0.4. We will call this node our BMU (best-matching unit).

## What happens next?

To understand this next part, we'll need to use a larger SOM.



Supposedly you now understand what the difference is between weights in the SOM context as opposed to the one we were used to when dealing with supervised machine learning.

The red circle in the figure above represents this map's BMU. Now, the new SOM will have to update its weights so that it is even closer to our dataset's first row. The reason we need this is that our input nodes cannot be updated, whereas we have control over our output nodes.

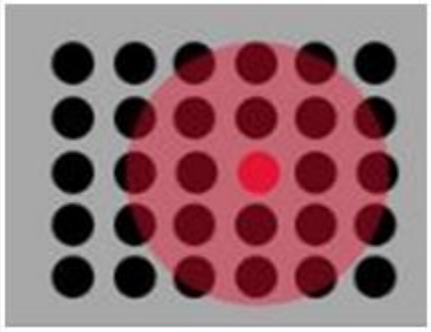
In simple terms, our SOM is drawing closer to the data point by stretching the BMU towards it. The end goal is to have our map as aligned with the dataset as we see in the image on the far right

**Step 3: Calculating the size of the neighbourhood around the BMU.**

This is where things start to get more interesting! Each iteration, after the BMU has been determined, the next step is to calculate which of the other nodes are within the BMU's neighbourhood. All these nodes will have their weight vectors altered in the next step.

So how do we do that? Well, it's not too difficult… first, you calculate what the radius of the neighbourhood should be and then it's a simple application of good ol' Pythagoras to determine if each node is within the radial distance or not.

The figure shows an example of the size of a typical neighbourhood close to the commencement of training.



You can see that the neighbourhood shown above is centred around the BMU (red-point) and encompasses most of the other nodes and the circle show radius.

The size of the neighbourhood around the BMU is decreasing with an exponential decay function. It shrinks on each iteration until reaching just the BMU

$$\sigma(t) = \sigma_0 \exp(-\frac{t}{\lambda})$$

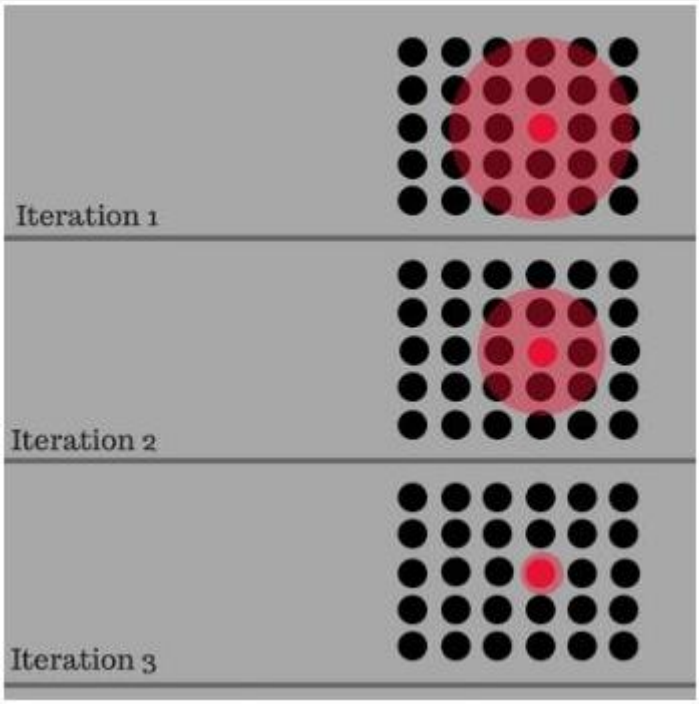
$\sigma_0$ = *the width of lattice at time zero*

$t$ = *the current time step*

$\lambda$ = *the time constant*

Where t = 0, 1, 2, 3….

The figure below shows how the neighbourhood decreases over time after each iteration




Over time the neighbourhood will shrink to the size of just one node… the BMU.

Now we know the radius, it's a simple matter to iterate through all the nodes in the lattice to determine if they lay within the radius or not. If a node is found to be within the neighbourhood then its weight vector is adjusted as follows in Step 4.

## How to set the radius value in the self-organizing map?

It depends on the range and scale of your input data. If you are mean-zero standardizing your feature values, then try $\sigma=4$. If you are normalizing feature values to a range of [0, 1] then you can still try $\sigma=4$, but a value of $\sigma=1$ might be better. Remember, you have to decrease the learning rate $\alpha$ and the size of the neighbourhood function with increasing iterations, as none of the metrics stays constant throughout the iterations in SOM.

It also depends on how large your SOM is. If it's a 10 by 10, then use for example $\sigma=5$. Otherwise, if it's a 100 by 100 map, use $\sigma=50$.

In unsupervised classification, $\sigma$ is sometimes based on the Euclidean distance between the centroids of the first and second closest clusters.

## Step 4: Adjusting the Weights

Every node within the BMU's neighbourhood (including the BMU) has its weight vector adjusted according to the following equation:

New Weights = Old Weights + Learning Rate (Input Vector — Old Weights)

W(t+1) = W(t) + L(t) ( V(t) — W(t) )

Where t represents the time-step and L is a small variable called the learning rate, which decreases with time. What this equation is sayiWhatnewly adjusted weight for the node is equal to the old weight (W), plus a fraction of the difference (L) between the old weight and the input vector (V).

So according to our example are Node 3 is the Best Match Unit (as you can see in step 2) corresponding to their weights:

| | | |
|---|---|---|
| $W_{3,1}= 0.39$ | $W_{3,2}= 0.42$ | $W_{3,3}= 0.45$ |
| Input Vector: $X_1= 0.7$ | $X_2= 0.6$ | $X_3= 0.9$ |

Learning rate = 0.5

So update that weight according to the above equation

For W3,1

New Weights = Old Weights + Learning Rate (Input Vector1 — Old Weights)

New Weights = 0.39 + 0.5 (0.7–0.39)

New Weights = 0.545

For W3,2

New Weights = Old Weights + Learning Rate (Input Vector2 — Old Weights)

New Weights = 0.42 + 0.5 (0.6–0.42)

New Weights = 0.51

For W3,3

New Weights = Old Weights + Learning Rate (Input Vector3 — Old Weights)

New Weights = 0.45 + 0.5 (0.9–0.45)

New Weights = 0.675

**Updated weights:**

$W_{3,1} = 0.545$ $W_{3,2} = 0.51$ $W_{3,3} = 0.67$

So in this way, we update the weights.

The decay of the learning rate is calculated for each iteration using the following equation:

$$L(t) = L_0 \exp(-\frac{t}{\lambda})$$

As training goes on, the neighbourhood gradually shrinks. At the end of the training, the neighbourhoods have shrunk to zero sizes.

$$\Theta(t) = \exp\left(-\frac{dist^2}{2\sigma^2(t)}\right)$$

$\Theta(t) =$ *Influence rate*

$\sigma(t) =$ *width of the lattice at time t*

The influence rate shows the amount of influence a node's distance from the BMU has on its learning. In the simplest form, the influence rate is equal to 1 for all the nodes close to the BMU and

zero for others, but a Gaussian function is common too. Finally, from a random distribution of weights and through many iterations, SOM can arrive at a map of stable zones. In the end, interpretation of data is to be done by a human but SOM is a great technique to present the invisible patterns in the data.

**References**


1. https://medium.com/machine-learning-researcher/self-organizing-map-som-c296561e2117

2. https://github.com/AmirAli5/Deep-Learning/blob/master/2.Unsupervised%20Deep%20Learning/4.Self%20Organization%20Map/Fraud%20Detection.ipynb