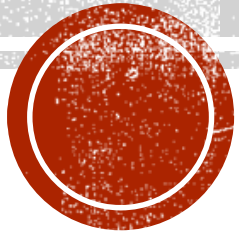
NEURAL NETWORK

Saeedeh Heydarian

"Intelligent methods for health information systems"

Spring 2024



WHAT IS A NEURAL NETWORK?

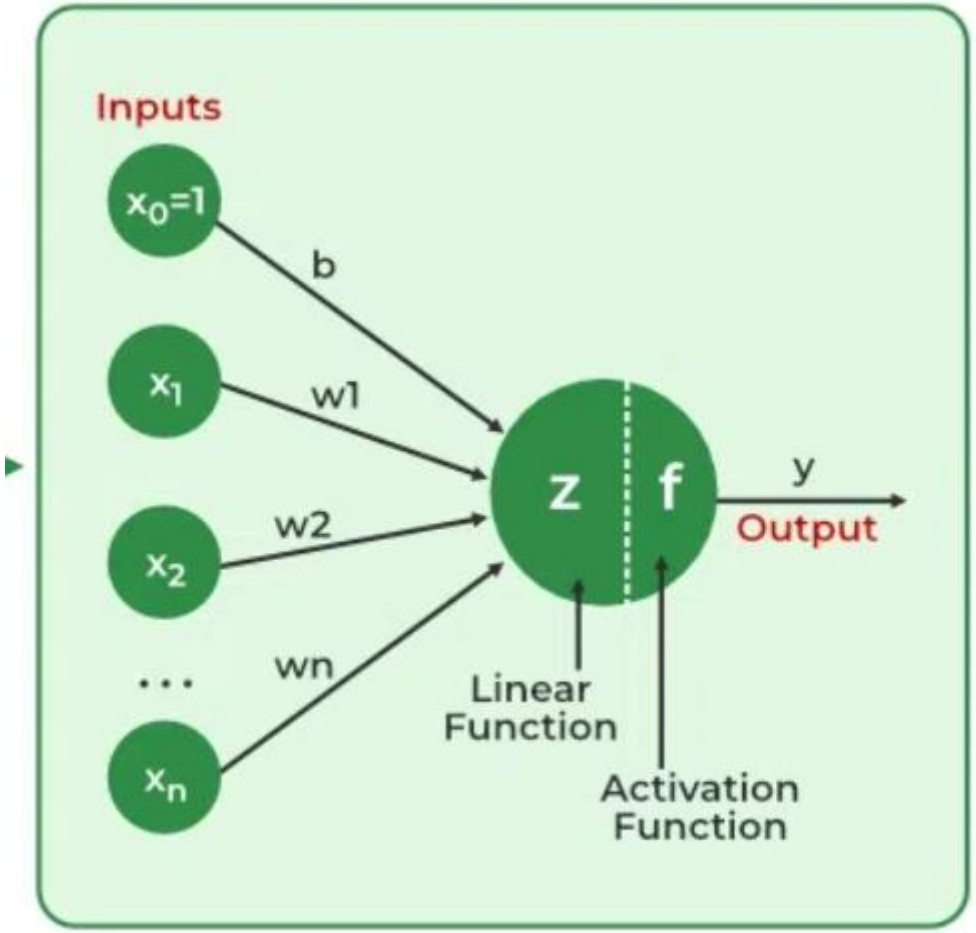
- A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain.
- It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain.
- Neural Networks are computational models that mimic the complex functions of the human brain.

EVOLUTION OF NEURAL NETWORKS

- **1940s-1950s: Early Concepts**
  Neural networks began with the introduction of the first mathematical model of artificial neurons by McCulloch and Pitts. But computational constraints made progress difficult.
- **1960s-1970s: Perceptrons**
  This era is defined by the work of Rosenblatt on perceptrons. Perceptrons are single-layer networks whose applicability was limited to issues that could be solved linearly separately.
- **1980s: Backpropagation and Connectionism**
  Multi-layer network training was made possible by Rumelhart, Hinton, and Williams' invention of the backpropagation method. With its emphasis on learning through interconnected nodes, connectionism gained appeal.

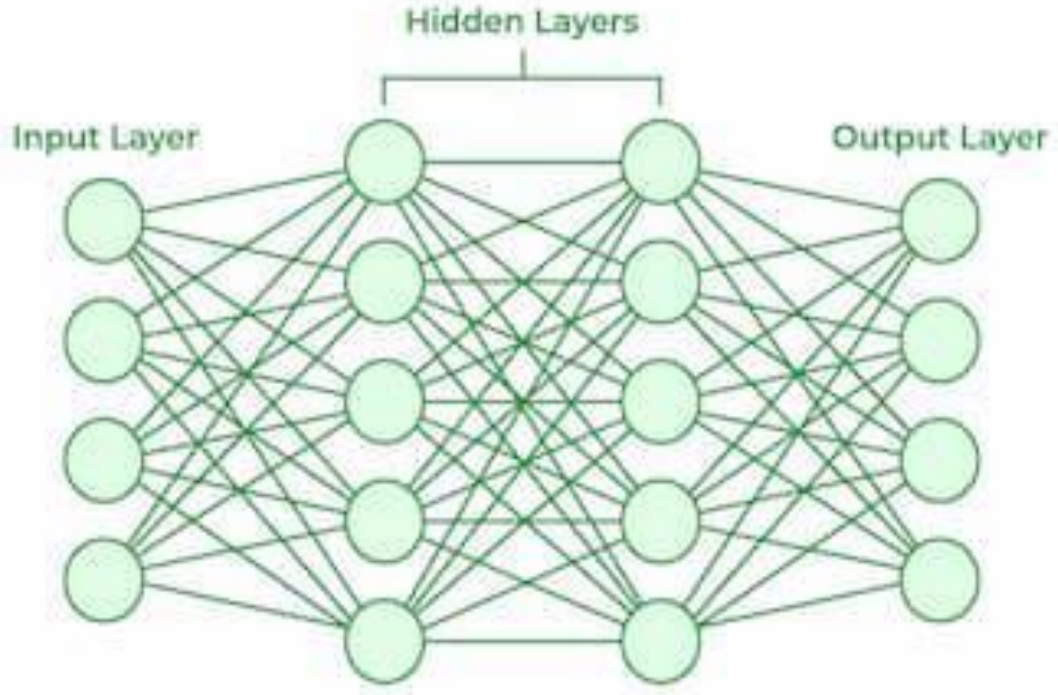
WHAT ARE NEURAL NETWORK

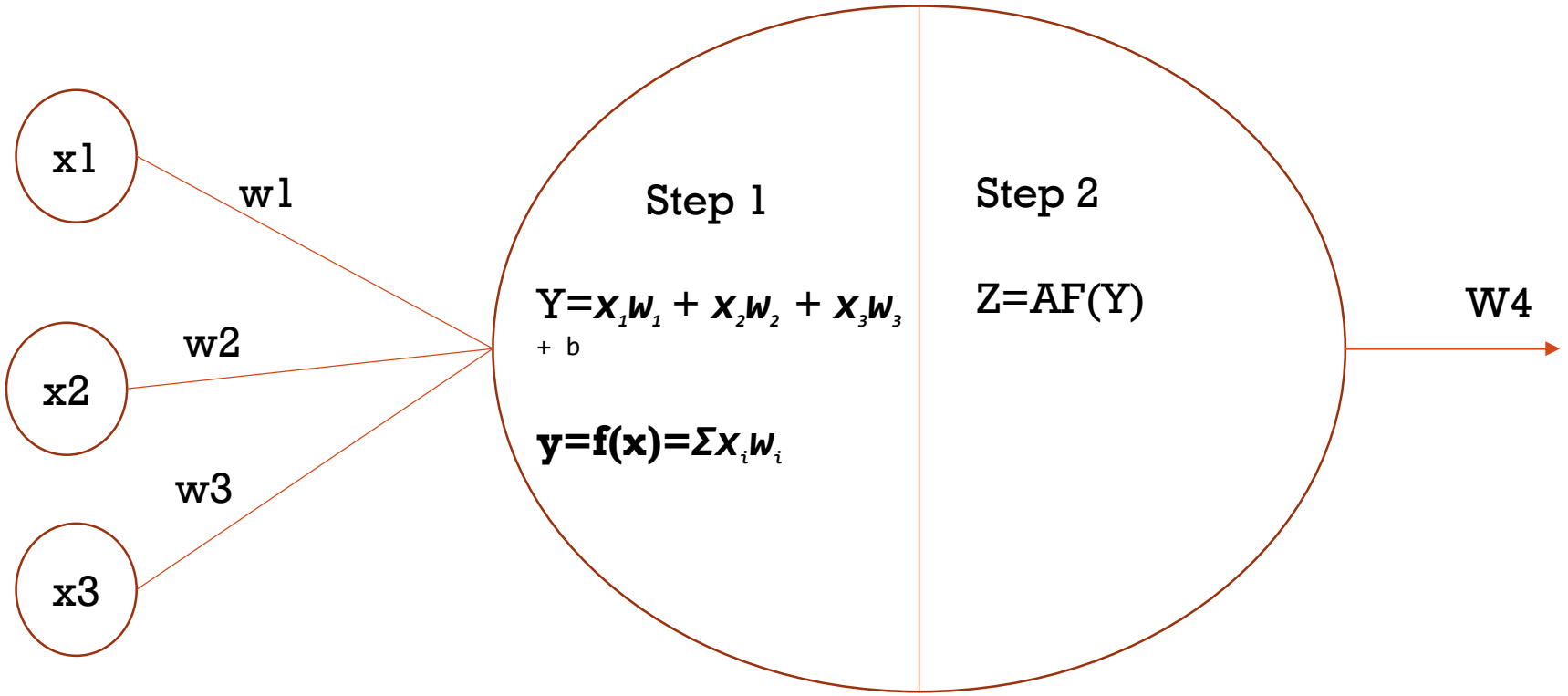
- Neural networks extract identifying features from data, lacking pre-programmed understanding.
- Network components include neurons, connections, weights, biases, propagation functions, and a learning rule.
- A perceptron is a basic unit of a neural network, capable of binary classification through a linear decision boundary.



WORKING OF A NEURAL NETWORK

- Neural networks are complex systems that mimic some features of the functioning of the human brain. It is composed of an input layer, one or more hidden layers, and an output layer made up of layers of artificial neurons that are coupled.
- The two stages of the basic process are called backpropagation and forward propagation.

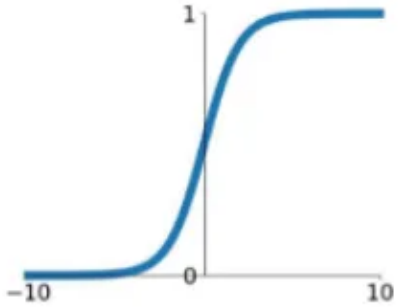



x1
x2
x3
w1
w2
w3
Step 1
Y=$x_1w_1 + x_2w_2 + x_3w_3$ + b
y=f(x)=$\Sigma x_i w_i$
Step 2
Z=AF(Y)
W4

Activation Functions

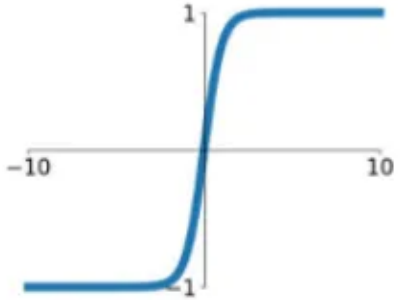
Sigmoid

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



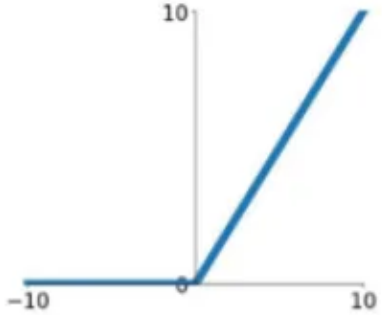
tanh

$$\tanh(x)$$



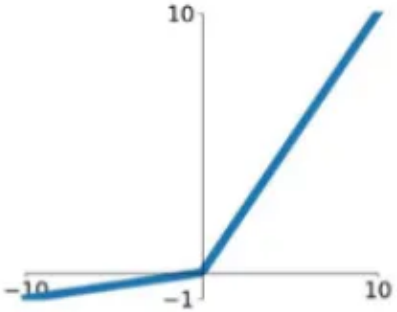
ReLU

$$\max(0, x)$$



Leaky ReLU

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

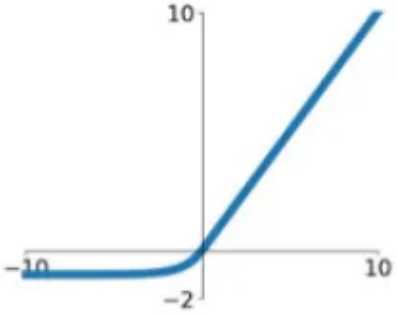


Maxout

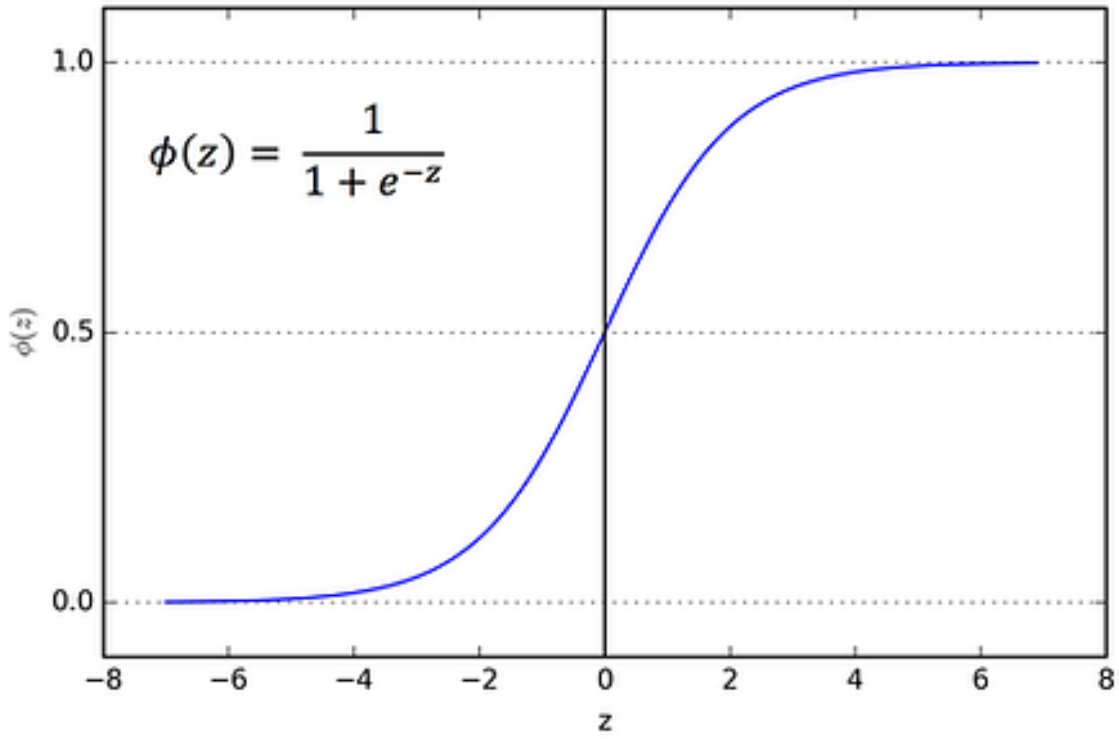
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

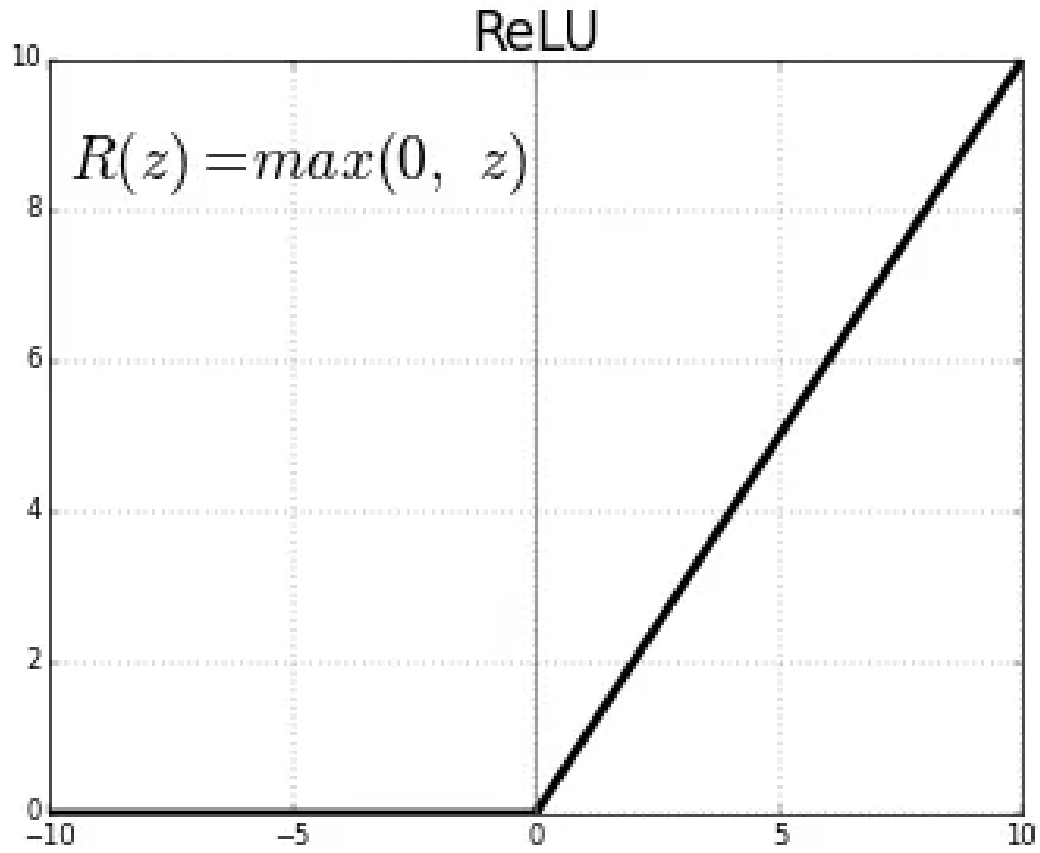
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



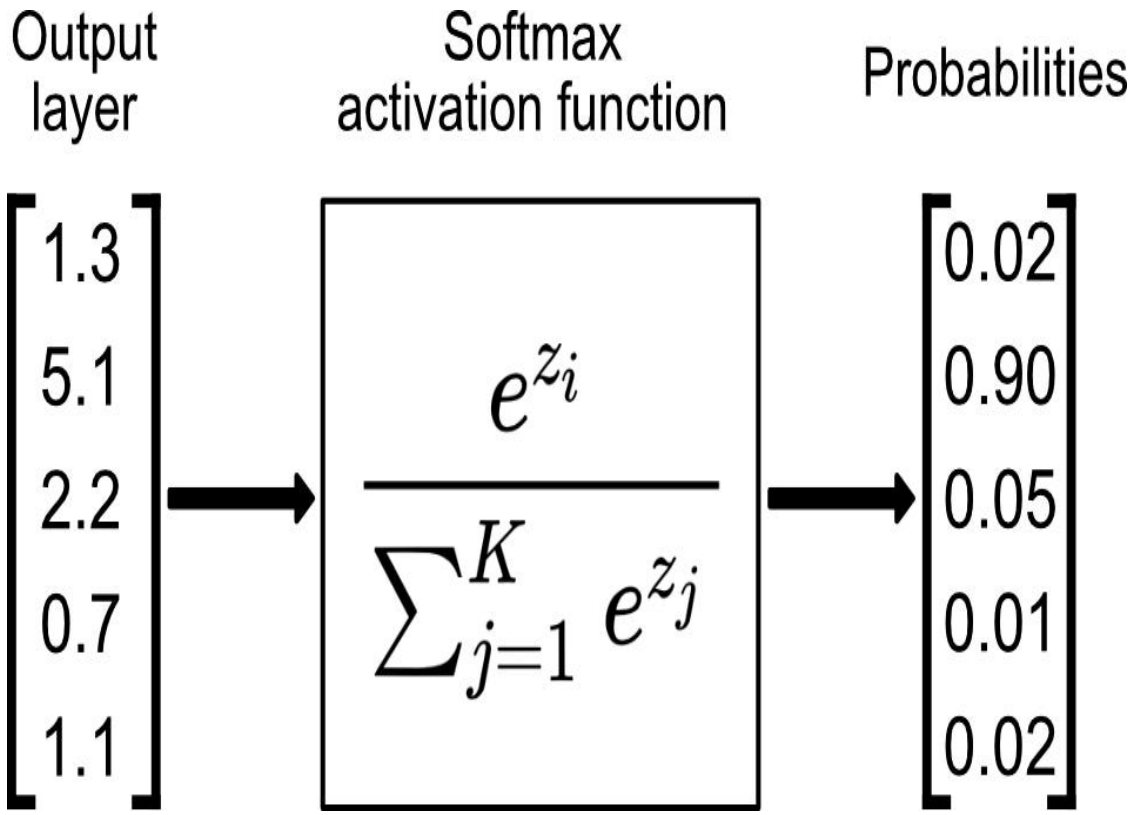
- **Sigmoid**
- Also known as Logistic Activation Function
- Range: Between 0 and 1



- **Rectified Linear Units (ReLU)**
- Simple and efficient: It is said to have 6 times improvement in convergence from tanh function
- Range: [0, infinity)



- **Softmax**
- converts a vector of values to a probability distribution.
- Softmax is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution.



FORWARD PROPAGATION

- **Input Layer:** Each feature in the input layer is represented by a node on the network, which receives input data.
- **Weights and Connections:** The weight of each neuronal connection indicates how strong the connection is. Throughout training, these weights are changed.
- **Hidden Layers:** Each hidden layer neuron processes inputs by multiplying them by weights, adding them up, and then passing them through an activation function. By doing this, non-linearity is introduced, enabling the network to recognize intricate patterns.
- **Activation Functions:** Model non-linearity is introduced by activation functions like the rectified linear unit (ReLU) or sigmoid. Their decision on whether to "fire" a neuron is based on the whole weighted input.
- **Output:** The final result is produced by repeating the process until the output layer is reached.

BACKPROPAGATION

- **Loss Function**: loss function is a term used in math optimization/decision theory. It is sometimes called cost function, fitness function or even error function.
- **Gradient Descent:** Gradient descent is then used by the network to reduce the loss. To lower the inaccuracy, weights are changed based on the derivative of the loss with respect to each weight.
- **Adjusting weights:** The weights are adjusted at each connection by applying this iterative process, or backpropagation, backward across the network.
- **Training:** During training with different data samples, the entire process of forward propagation, loss calculation, and backpropagation is done iteratively, enabling the network to adapt and learn patterns from the data.

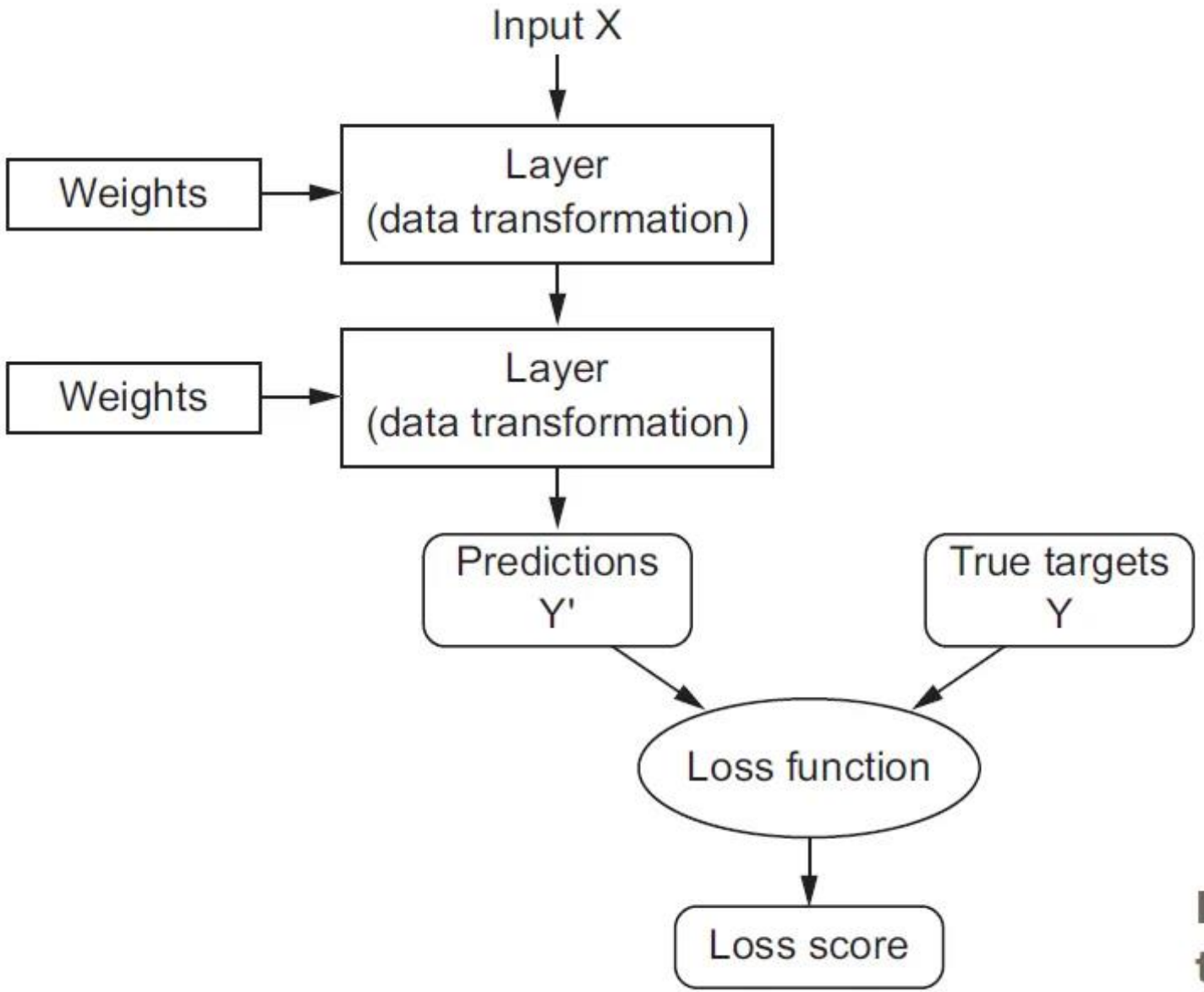


Figure 1.8 A loss function measures the quality of the network's output.

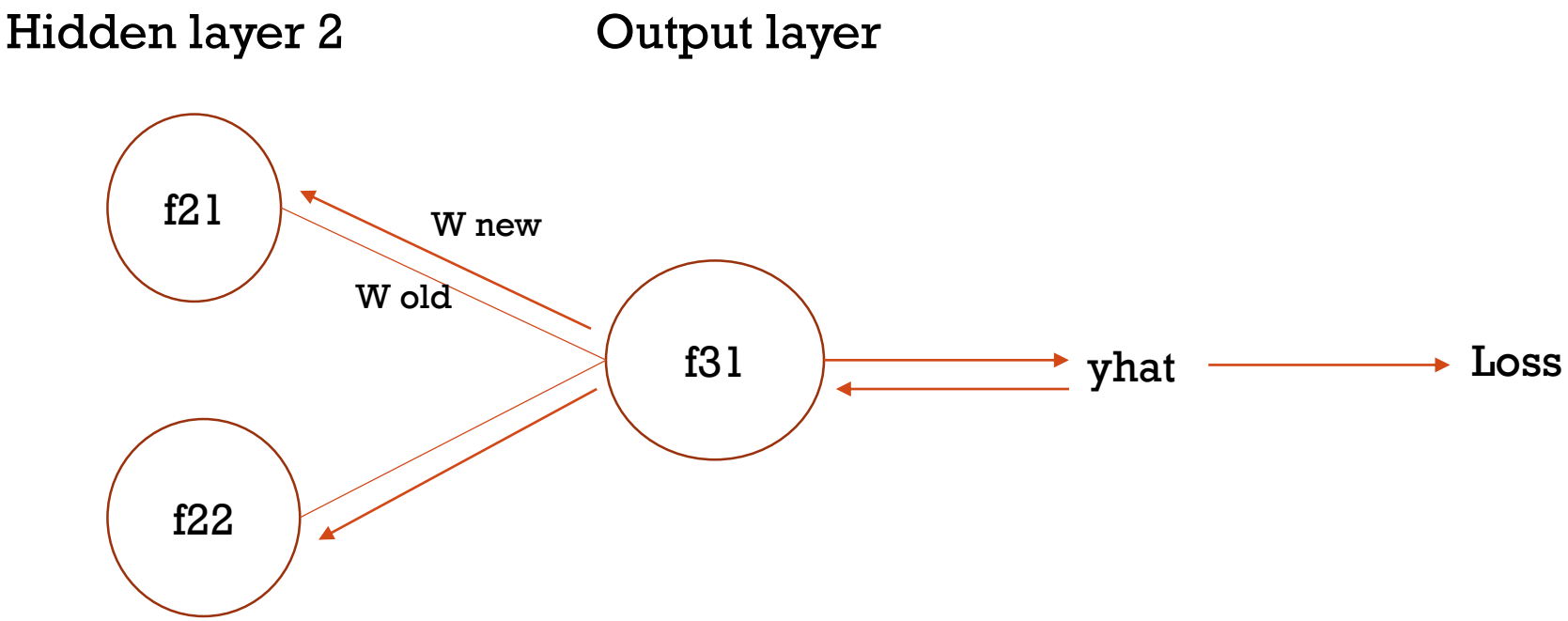
LOSS FUNCTIONS

- There are different loss functions which are commonly used as part of deep learning algorithms — following are a couple of examples. With regression example loss functions are: **MSE** (Mean square error), **MAE** (Mean Absolute Error) and **Hubber loss**. With classification examples loss functions are**: binary cross entropy** and **categorical cross-entropy.**
- **Loss= (Yi –Yihat)^2**

OPTIMIZERS

- **Gradient Descent**: Gradient descent is a simple optimization algorithm that updates the model's parameters to minimize the loss function.
- **Stochastic Gradient Descent:** Stochastic gradient descent (SGD) is a variant of gradient descent that involves updating the parameters based on a small, randomly-selected subset of the data (i.e., a "mini-batch") rather than the full dataset.
- **Stochastic Gradient Descent with Momentum**
- **Mini-Batch Gradient Descent:** similar to SGD, but instead of using a single sample to compute the gradient, it uses a **small, fixed-size "mini-batch" of samples.**
- **Adagrad**
- **RMSProp**
- **AdaDelta**
- **Adam**


Starting point
Loss
Value of weight
Point of convergence
i.e. where the cost
function is at its
minimum


Hidden layer 2
Output layer
f21
W new
W old
f31
yhat
Loss
f22

$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

The value of the updated weight

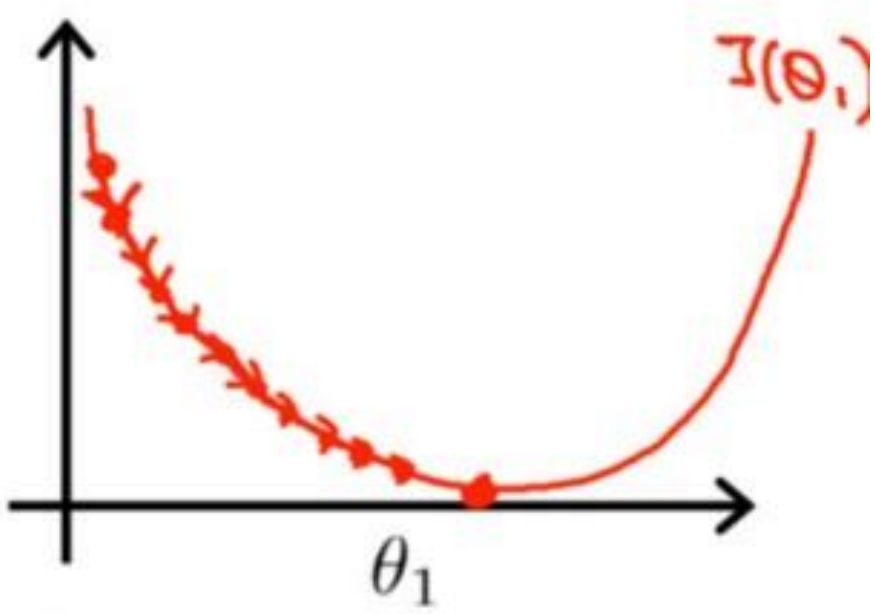
The old value of the weight

Learning rate

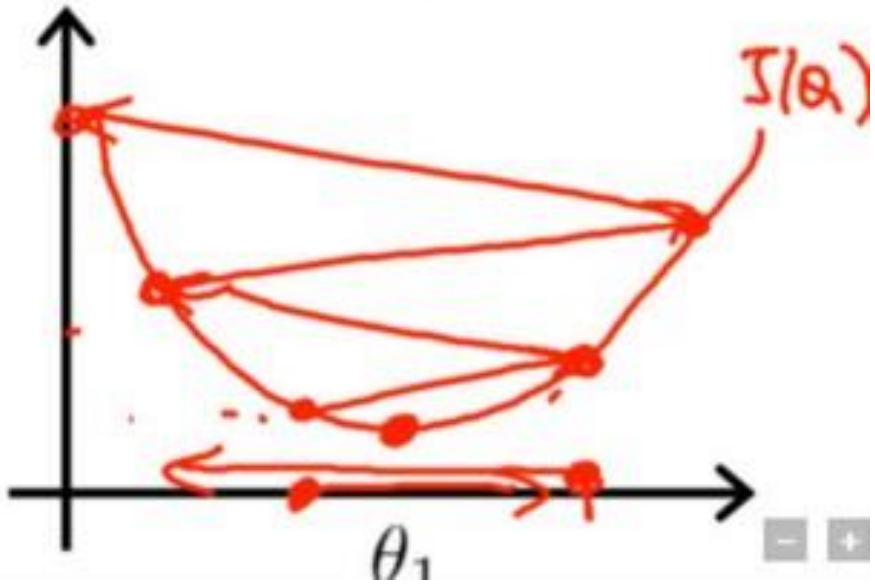
Derivative loss with respect to the weight

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



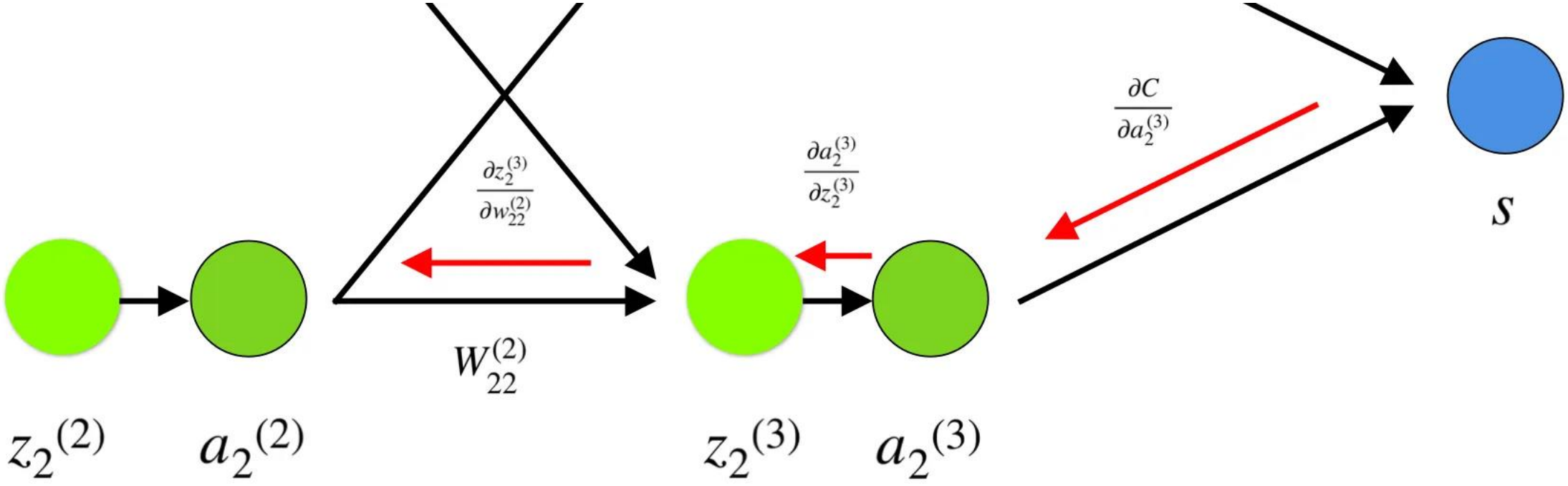
CHAIN RULE

- Compute those gradients happens using a technique called chain rule.
- Compute the gradient of the loss with respect to the parameters (*W* and *b*)

- $\frac{dL}{dW} = \frac{dL}{dz} \cdot \frac{dz}{dW} = x \cdot \frac{dL}{dz}$
- $\frac{dL}{db} = \frac{dL}{dz} \cdot \frac{dz}{db} = \frac{dL}{dz}$

Update the parameters *W* and *b* using the computed gradients and an optimization algorithm.

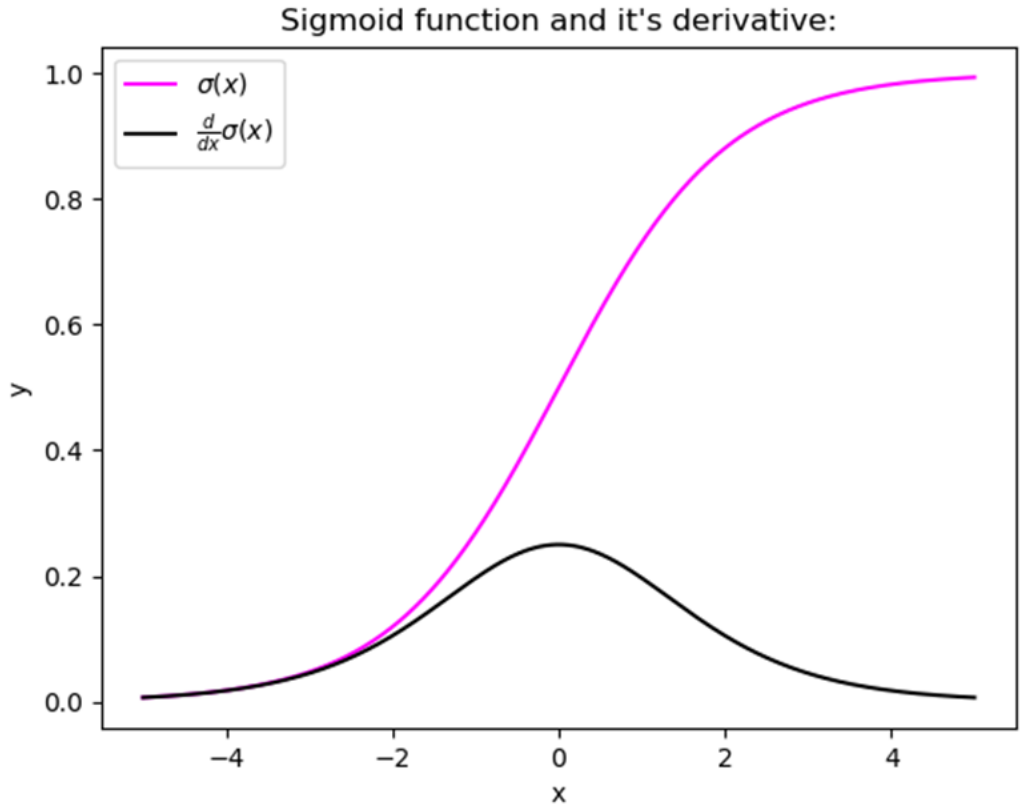
$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$



$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$

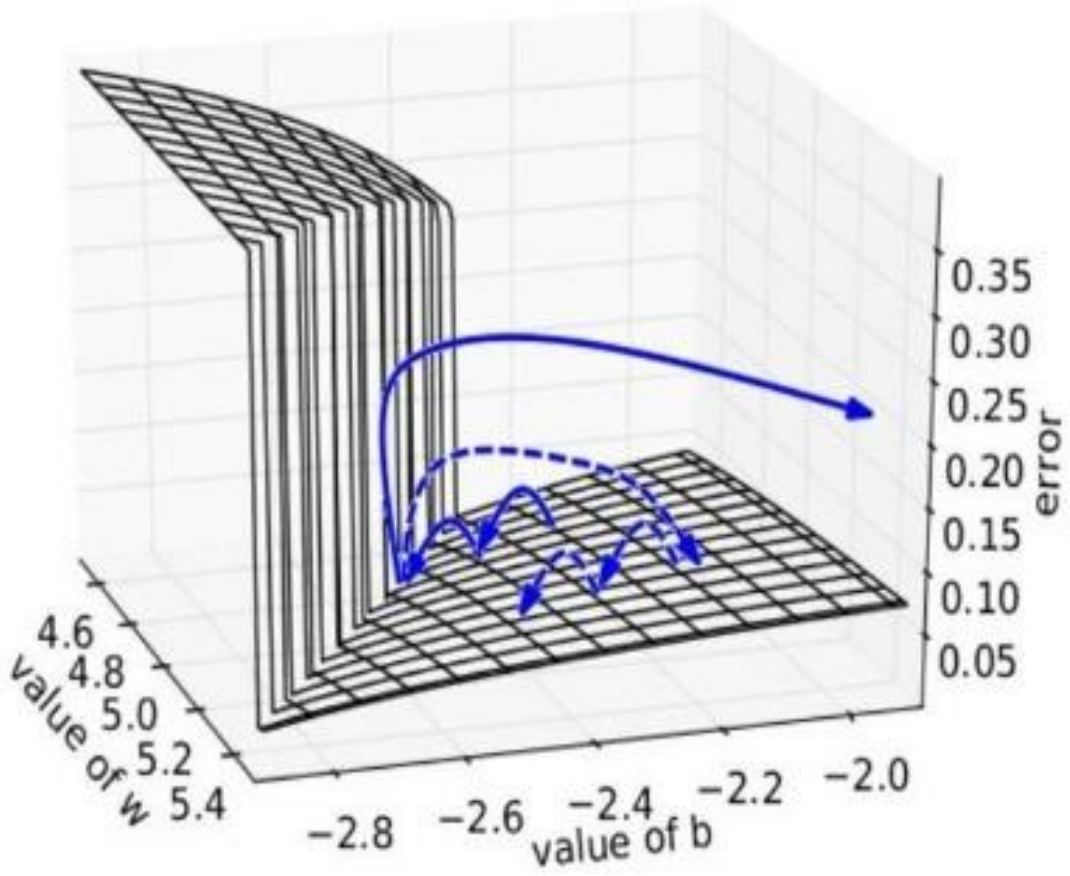
THE CHALLENGE OF VANISHING/EXPLODING GRADIENTS IN DEEP NEURAL NETWORKS

- **Vanishing:**
- 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.



Exploding:

Conversely, in the context of exploding gradients, the gradients of the loss function with respect to the model's parameters become extremely large during backpropagation.



DEEP NEURAL NETWORK

- Deep neural networks, or deep learning networks, have several hidden layers with millions of artificial neurons linked together. A number, called weight, represents the connections between one node and another.
- The weight is a positive number if one node excites another, or negative if one node suppresses the other. Nodes with higher weight values have more influence on the other nodes.

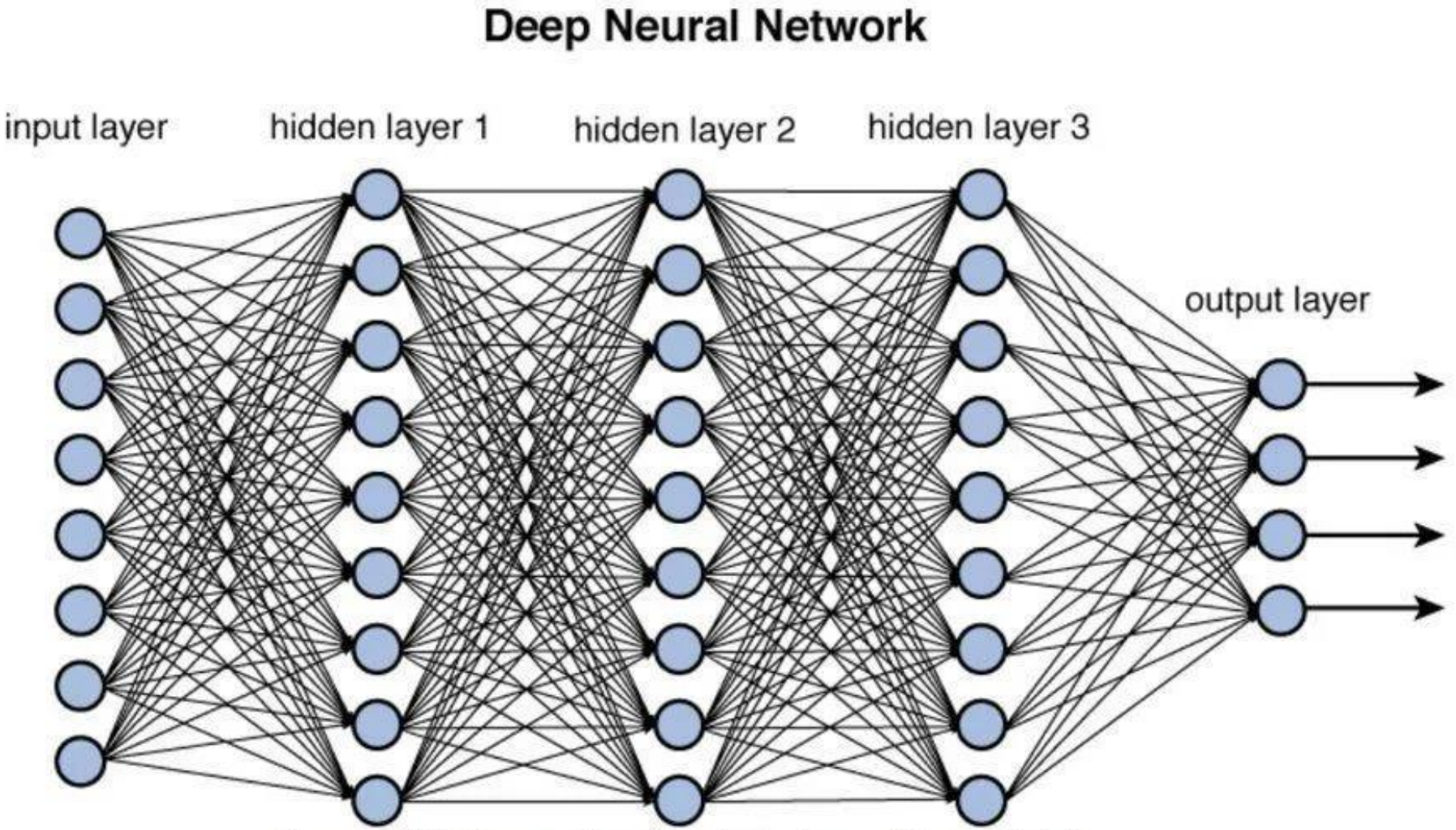


Figure 12.2 Deep network architecture with multiple layers.

REGULARIZATION TECHNIQUE

- Regularization is a set of techniques that can prevent overfitting in neural networks and thus improve the accuracy of a Deep Learning model when facing completely new data from the problem domain.
- The most popular regularization techniques which are called L1, L2, and dropout.
-

L2&L1 REGULARIZATION

- L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.
- *Cost function = Loss (say, binary cross entropy) + Regularization term*
- In L2, we have:

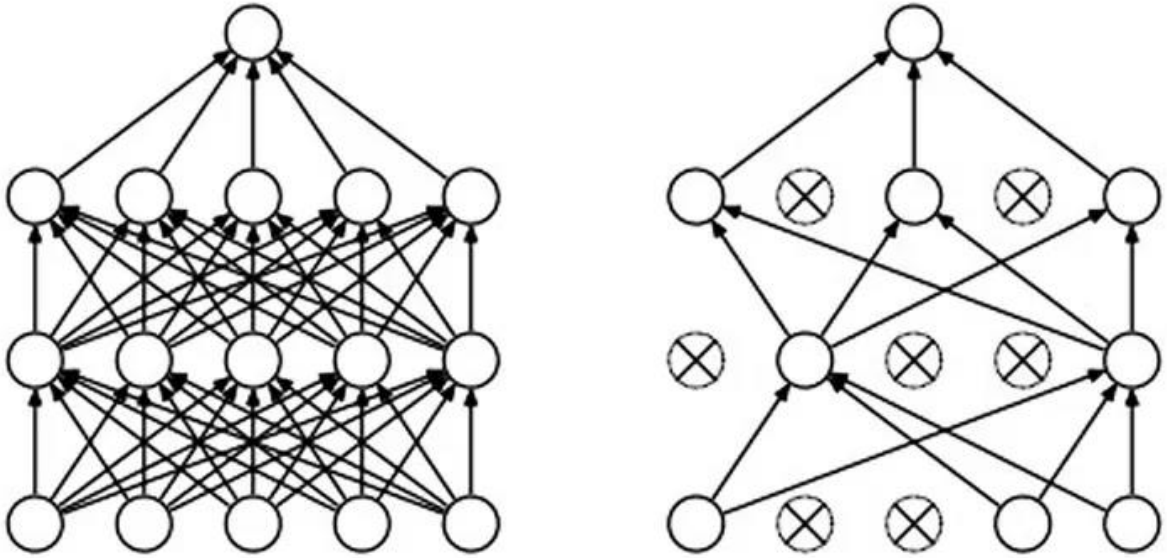
$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

- In L1, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

DROP OUT

- The key idea is to randomly drop unit
  while training the network so that we a
  working with smaller neural network a
  each iteration. To drop a unit is same a
  to ignore those units during forward
  propagation or backward propagatior
  a sense this prevents the network from
  adapting to some specific set of features.



(a) Standard Neural Net

(b) After applying dropout.

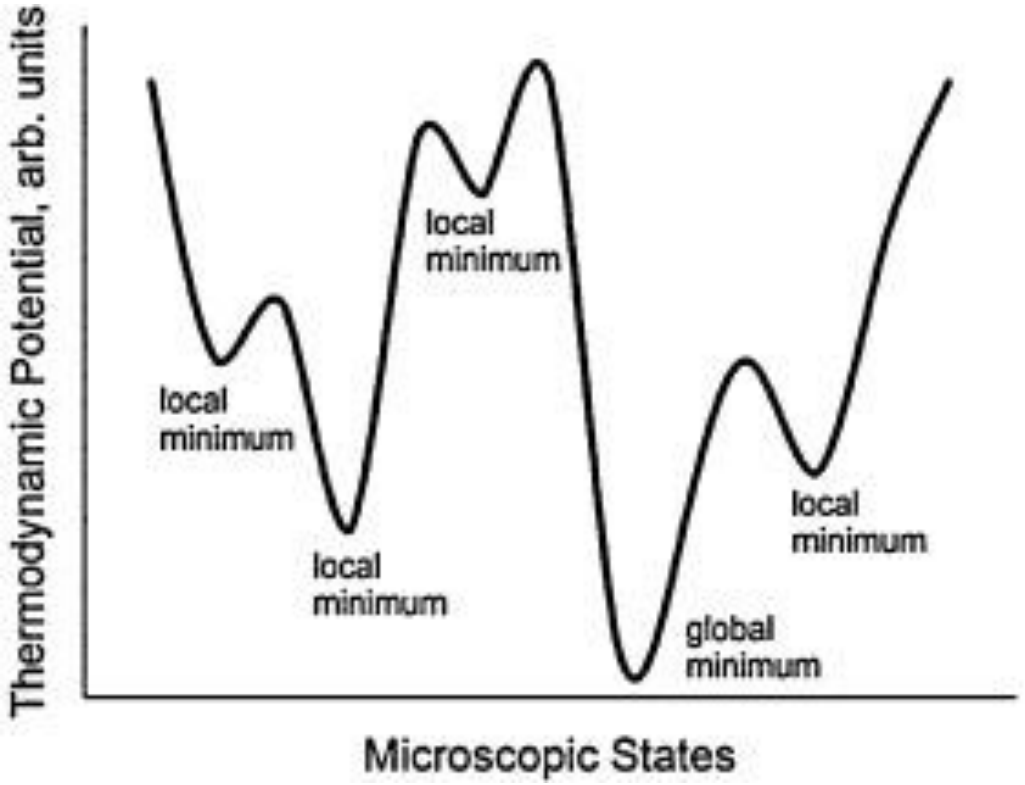
GLOBAL MINIMA & LOCAL MINIMA

Local Optima

- A **local optimum** (local minimum or maximum) is a solution that is better than any other feasible solutions in its immediate vicinity, but not necessarily the best possible solution across the entire search space.

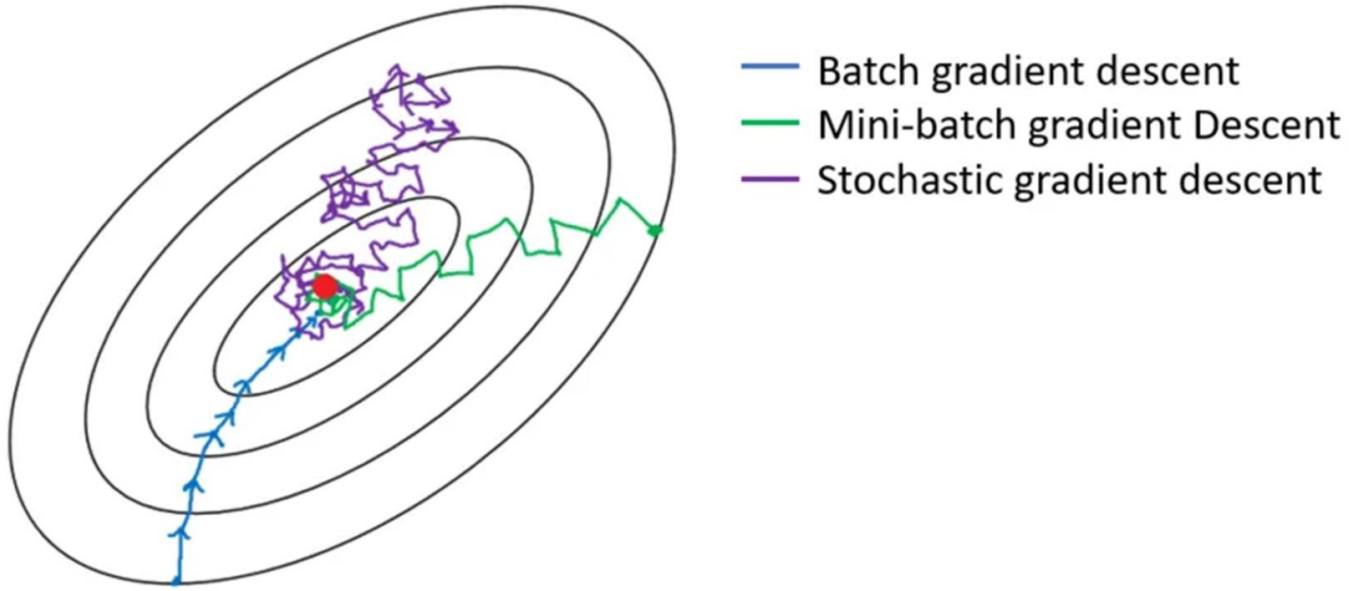
Global Optima

- A **global optimum** is the best possible solution across the entire search space. It represents the absolute highest or lowest value of the objective function over all possible inputs.
- **Global Minimum**: The point where the function attains its lowest value over the entire domain.



GD & SGD & MINI BTACH GD

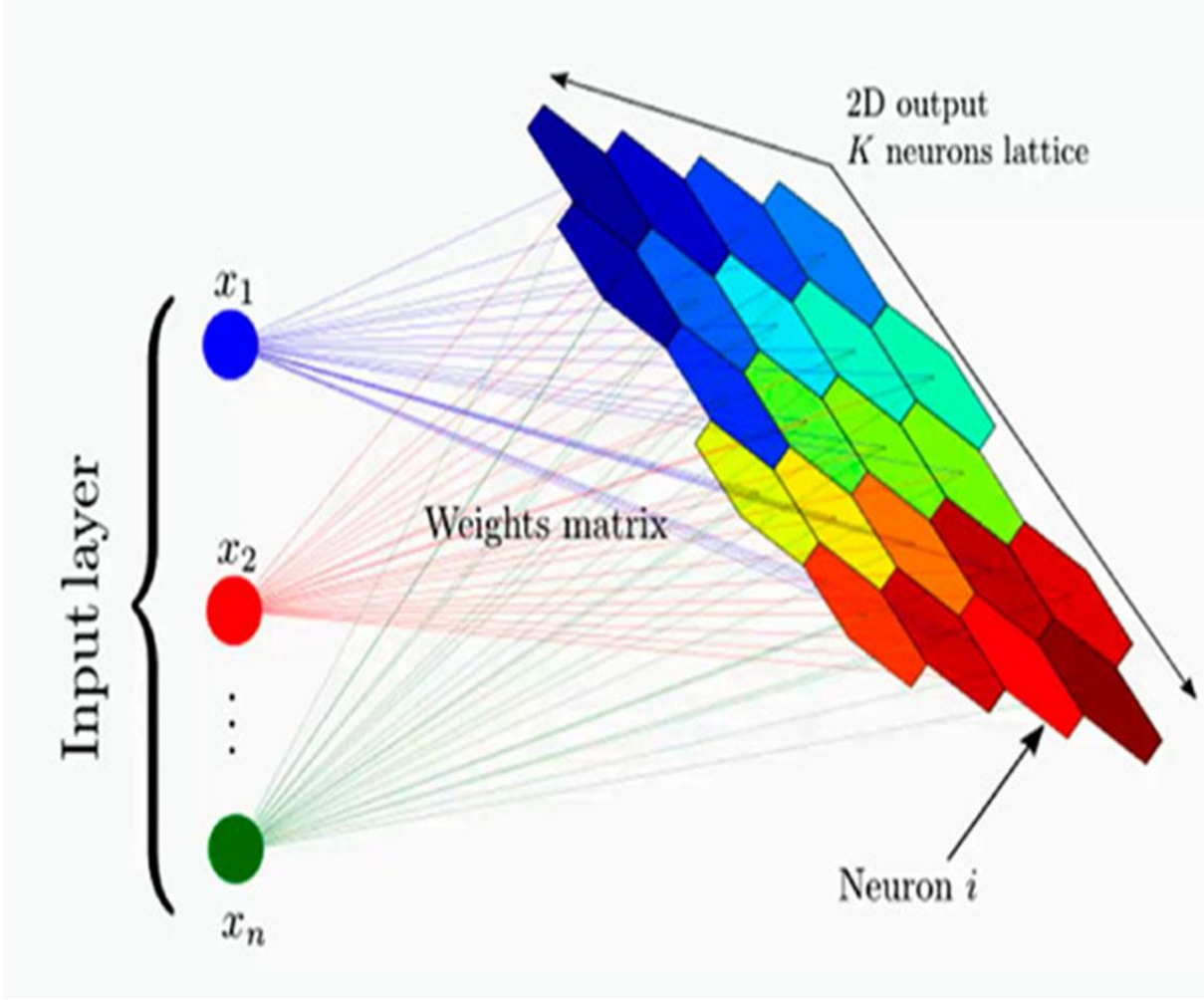
- Mini Batch Gradient Descent is considered to be the cross-over between GD and SGD. In this approach instead of iterating through the entire dataset or one observation, we split the dataset into small subsets (batches) and compute the gradients for each batch.

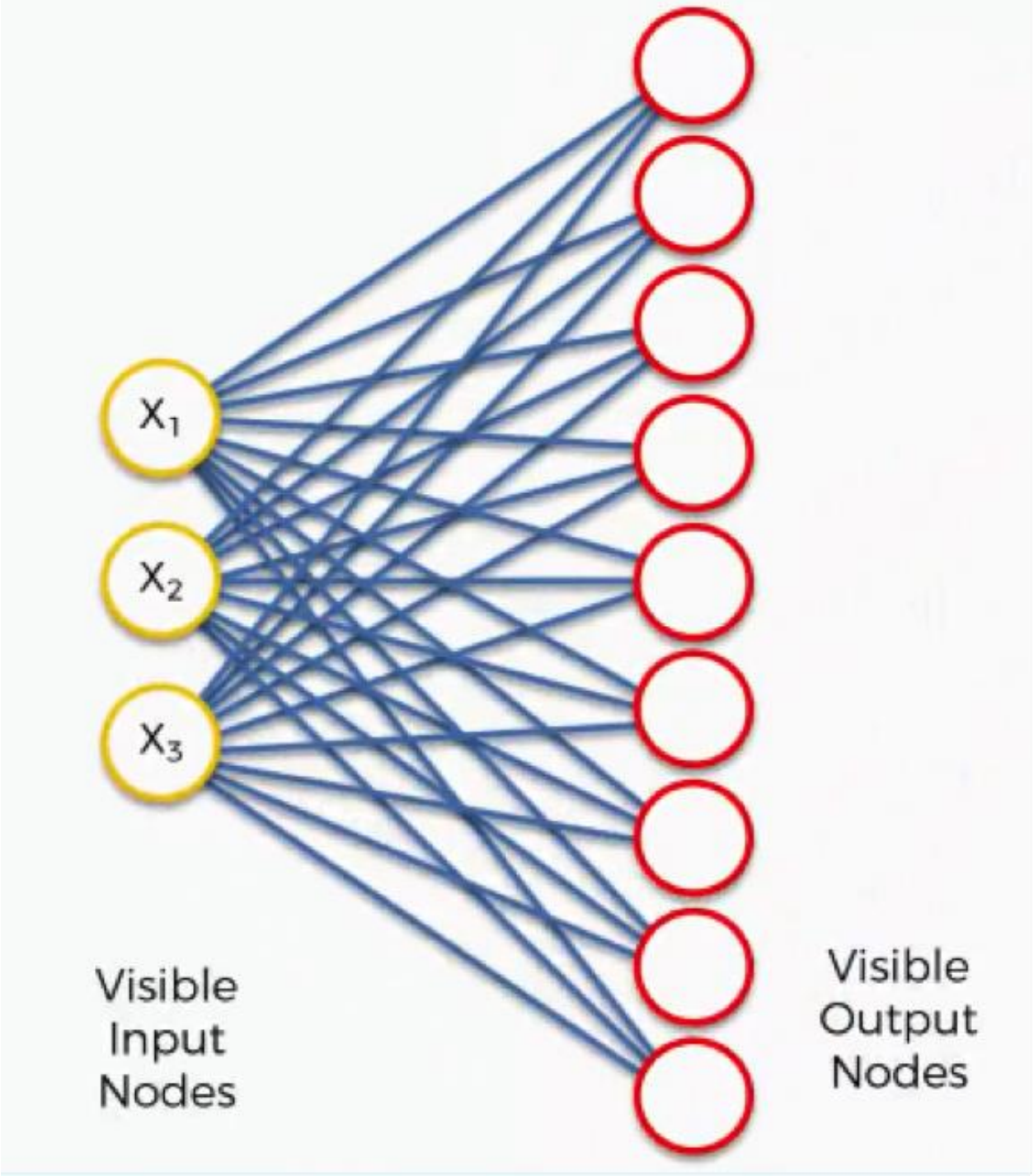


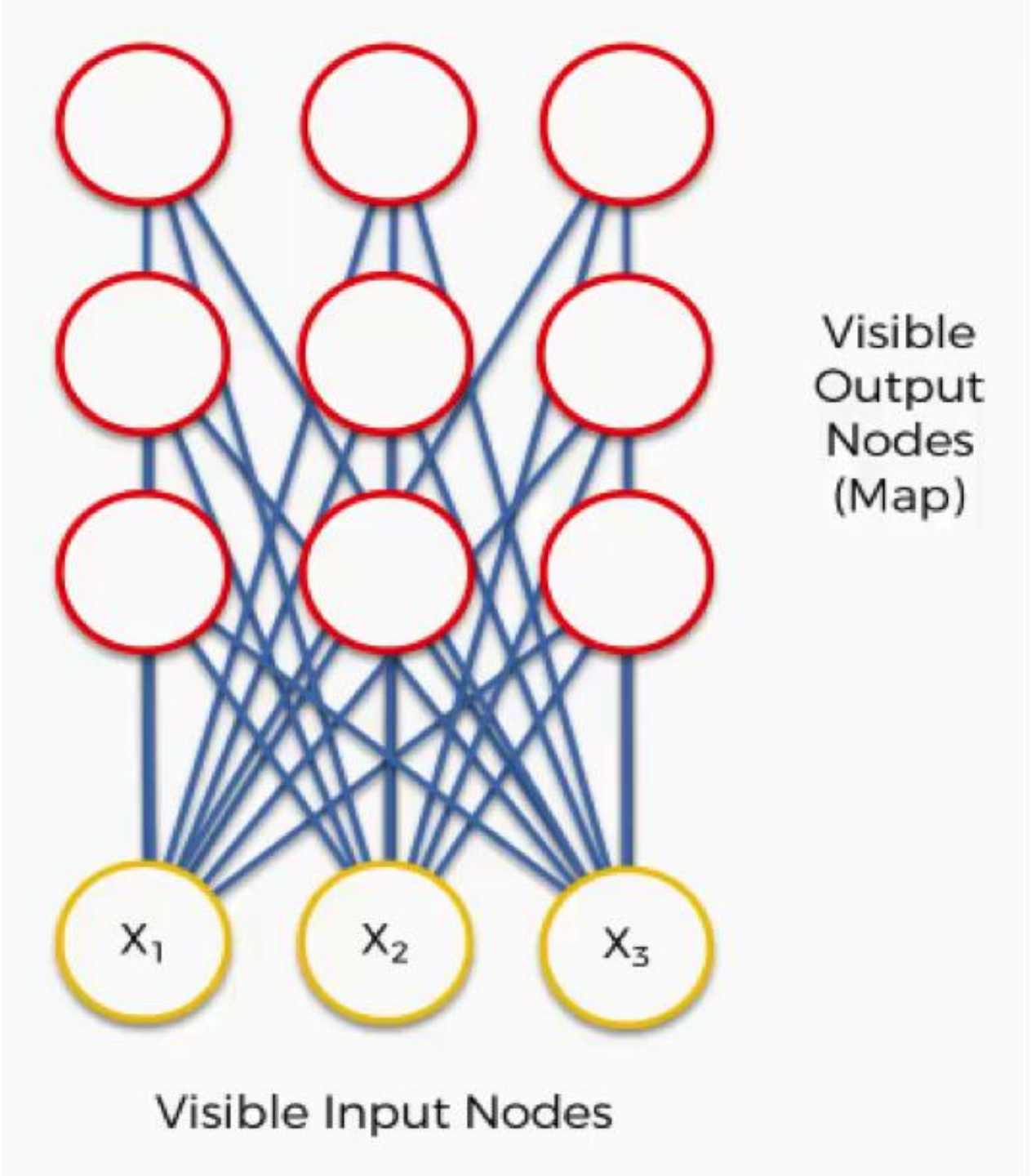
SELF ORGANIZING MAPS (SOM)

- Self Organizing Maps or Kohenin's map is a type of artificial neural networks introduced by Teuvo Kohonen in the 1980s.
- A **self-organizing map** (**SOM**) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional)
- it is a little bit different from other artificial neural networks, SOM doesn't learn by backpropagation with SGD, it use competitive learning to adjust weights in neurons.
- SOM are used to reduce dimensionality & make clusters.
- ***it is a very useful technique for clustering analysis, and exploring data.***

- SOM has two layers, one is the Input layer and the other one is the Output layer.
- Input nodes represents the features of the dataset.
- Each synapse of input node of input layers carries some weights.
- Euclidian distances between the input node & weights are calculated.
- Unlike supervised deep learning framework, it does not have any cost function or cross entropy function. So no concept of Backpropagation with gradient descent comes here.




X_1
X_2
X_3
Visible Input Nodes
Visible Output Nodes

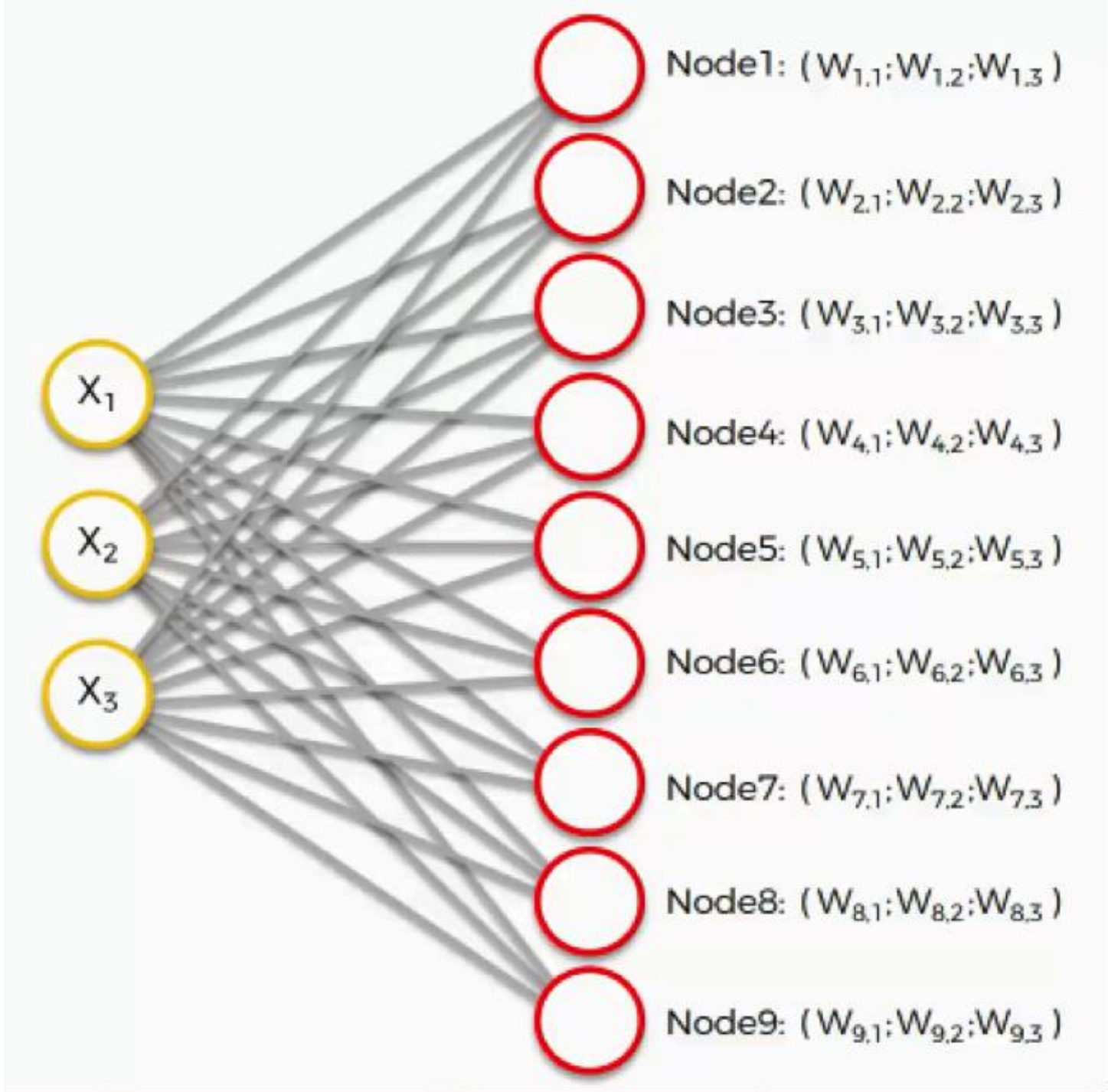

X_1
X_2
X_3
Visible Input Nodes
Visible Output Nodes (Map)

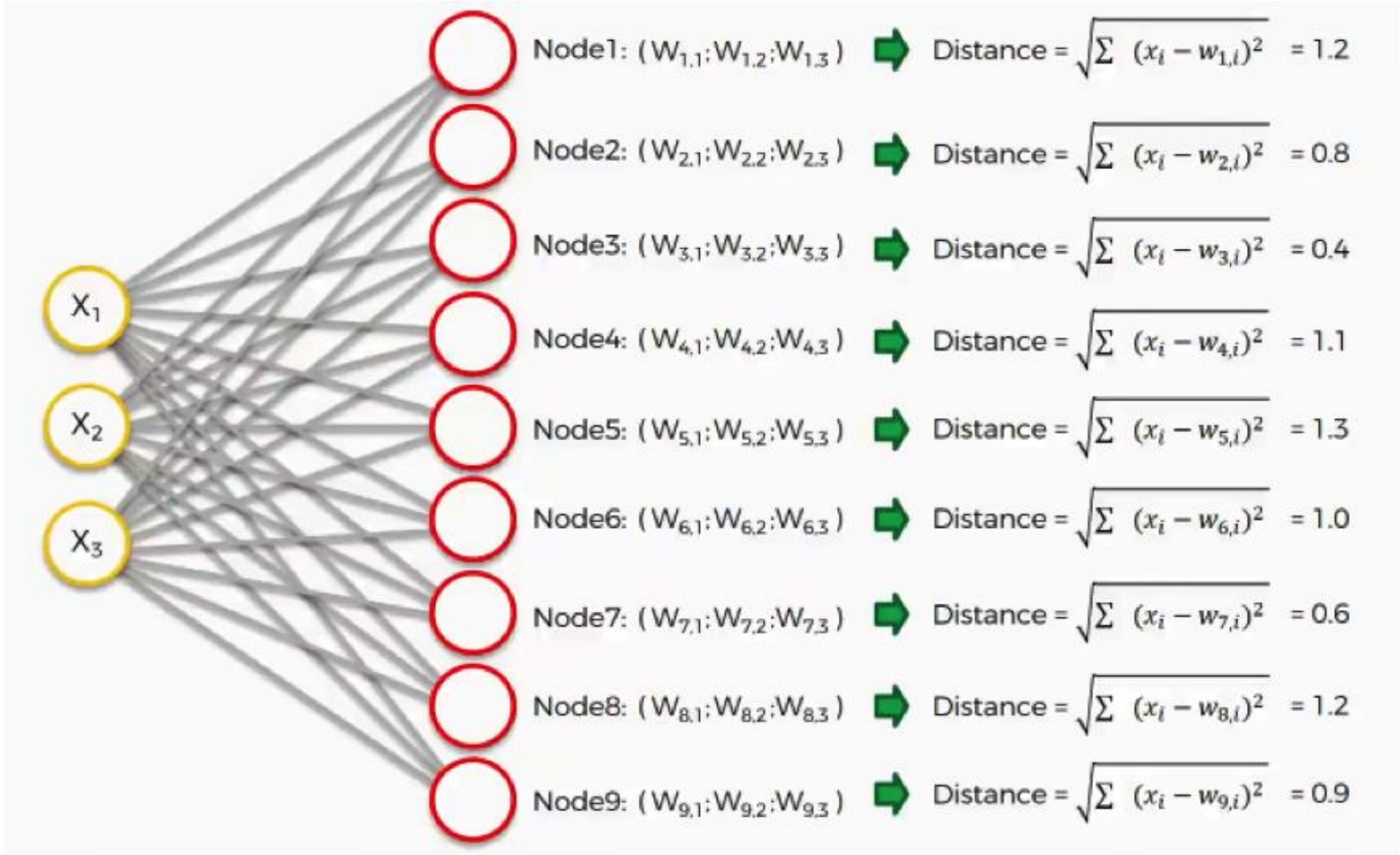
THE TRAINING PROCESS OF SOMS FOLLOWS:

1. Firstly, randomly initialize all the weights.
2. Select an input vector x = [x1, x2, x3, … , xn] from the training set.
3. Compare x with the weights wj by calculating Euclidean distance for each neuron j. The neuron having the least distance is declared as the winner. The winning neuron is known as Best Matching Unit(BMU)
4. Update the neuron weights so that the winner becomes and resembles the input vector x.
5. The weights of the neighbouring neuron are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its weights get altered. The parameters adjusted are learning rate and neighbourhood function. (more below on neighbourhood function)
6. Repeat from step 2 until the map has converged for the given iterations or there are no changes observed in the weights.

1.COMPETITION

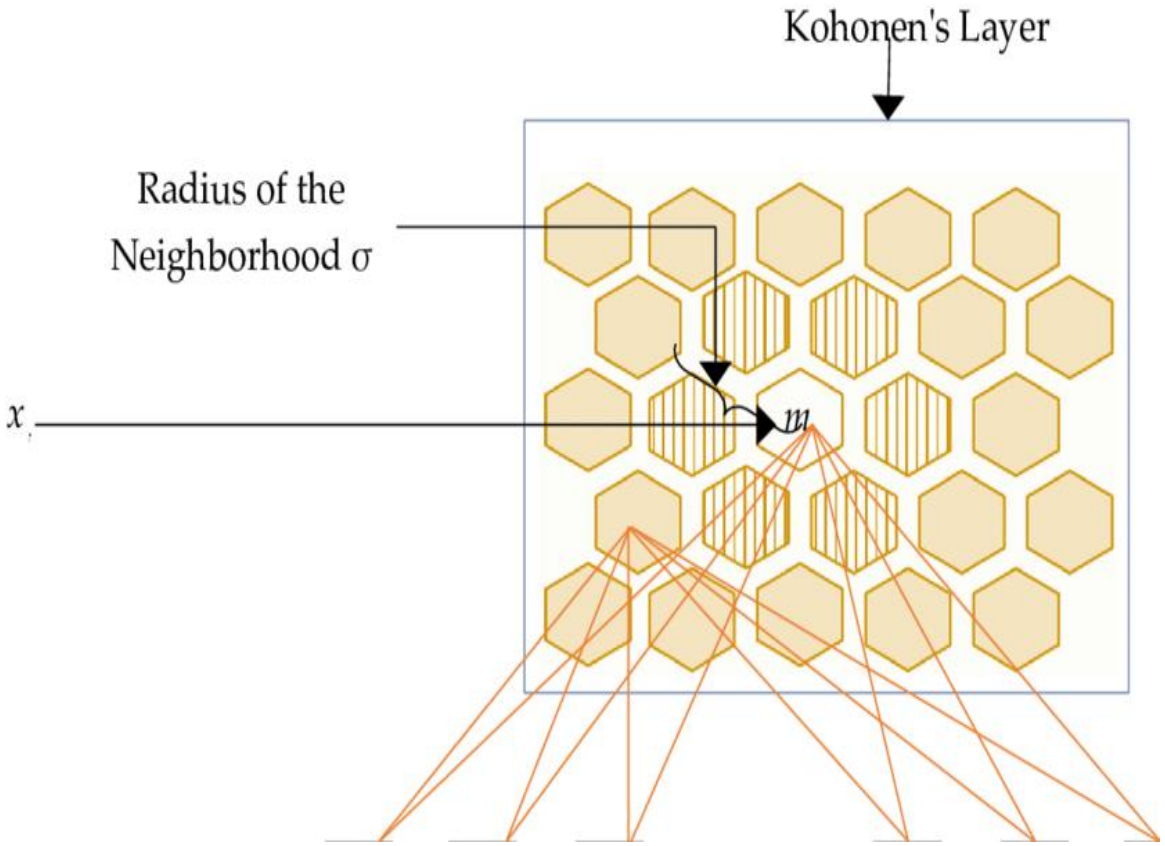
- In this stage, we go over all the input data points. Then, for each, we measure the distance between its vector and each neuron in the lattice (e.g., Euclidean distance).
- After calculating the distances, we find the neuron closest to the input vector neuron. This stage is called competition because **the neurons compete in similarity to our data point.**


X1
X2
X3
Node1: (W1,1:W1,2:W1,3)
Node2: (W2,1:W2,2:W2,3)
Node3: (W3,1:W3,2:W3,3)
Node4: (W4,1:W4,2:W4,3)
Node5: (W5,1:W5,2:W5,3)
Node6: (W6,1:W6,2:W6,3)
Node7: (W7,1:W7,2:W7,3)
Node8: (W8,1:W8,2:W8,3)
Node9: (W9,1:W9,2:W9,3)



Index(x)=argmax (wjx)=argmin(wj-x)

- As we saw above the distance of each of the nodes (or raw data point) is calculated from all the output nodes.
- The node or neuron which is identified with the least distance is the Best Matching Unit (BMU) or neighbourhood.
- $BMUindex = arg_j min \|\mathbf{w}_j - \mathbf{x}\|$
- From this BMU, a radius (or sigma) is defined. All the nodes that fall in the radius of the BMU get updated according to their respective distance from the BMU.



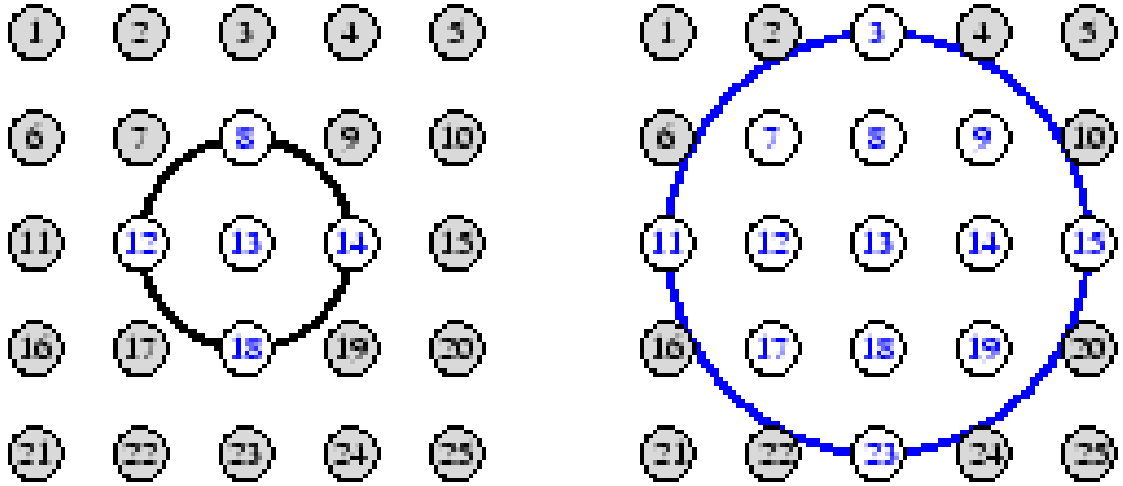
2. COOPERATION

- As we saw above the distance of each of the nodes (or raw data point) is calculated from all the output nodes. The node or neuron which is identified with the least distance is the Best Matching Unit (BMU) or neighbourhood. It is depicted as (m) in the following figure:
- **The cooperation phase begins after finding the winning neuron.** In it, we find the winning neuron's neighbors. The Gaussian function is a common choice for a neighbourhood function. S
- The neighborhood is defined through the neighborhood function h. It quantifies the degree to which a neuron can be considered a neighbor of the winning neuron. While doing so, h follows two rules:
- it should be symmetrical about the winning neuron
- its value should decrease with the distance to the neuron

Any nodes found within this radius are deemed to be inside the BMU's neighborhood.

After each iteration, the radius shrinks and the BMU pulls lesser nodes so the process becomes more and more accurate and that shrinks the neighbourhood that reduces the mapping of those segments of those data points which belong to that particular segment.

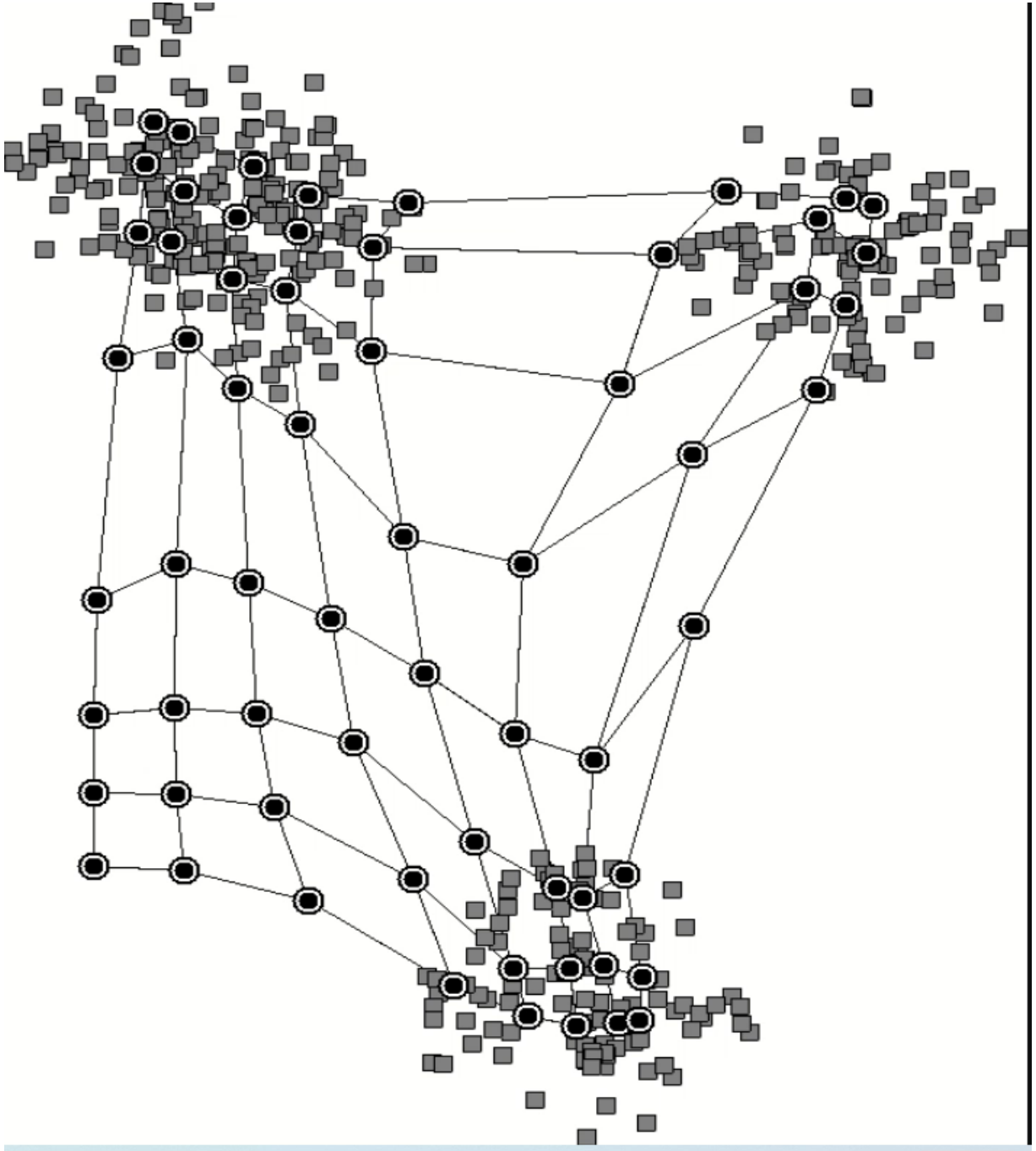
This makes the map more and more specific and ultimately it starts converging and becomes like the training data (or input).



$\sigma(t)=\sigma 0$ exp(−time constantiteration number)

```
sigma = initial_sigma * np.exp(-iteration / time_constant

 return sigma
```



3. ADAPTATION

- Here, we update the neurons according to:

$$w_j(n+1) = w_j(n) + \alpha h_{ij}\Big(d\left(x, w_j(n)\right)\Big)$$

where:

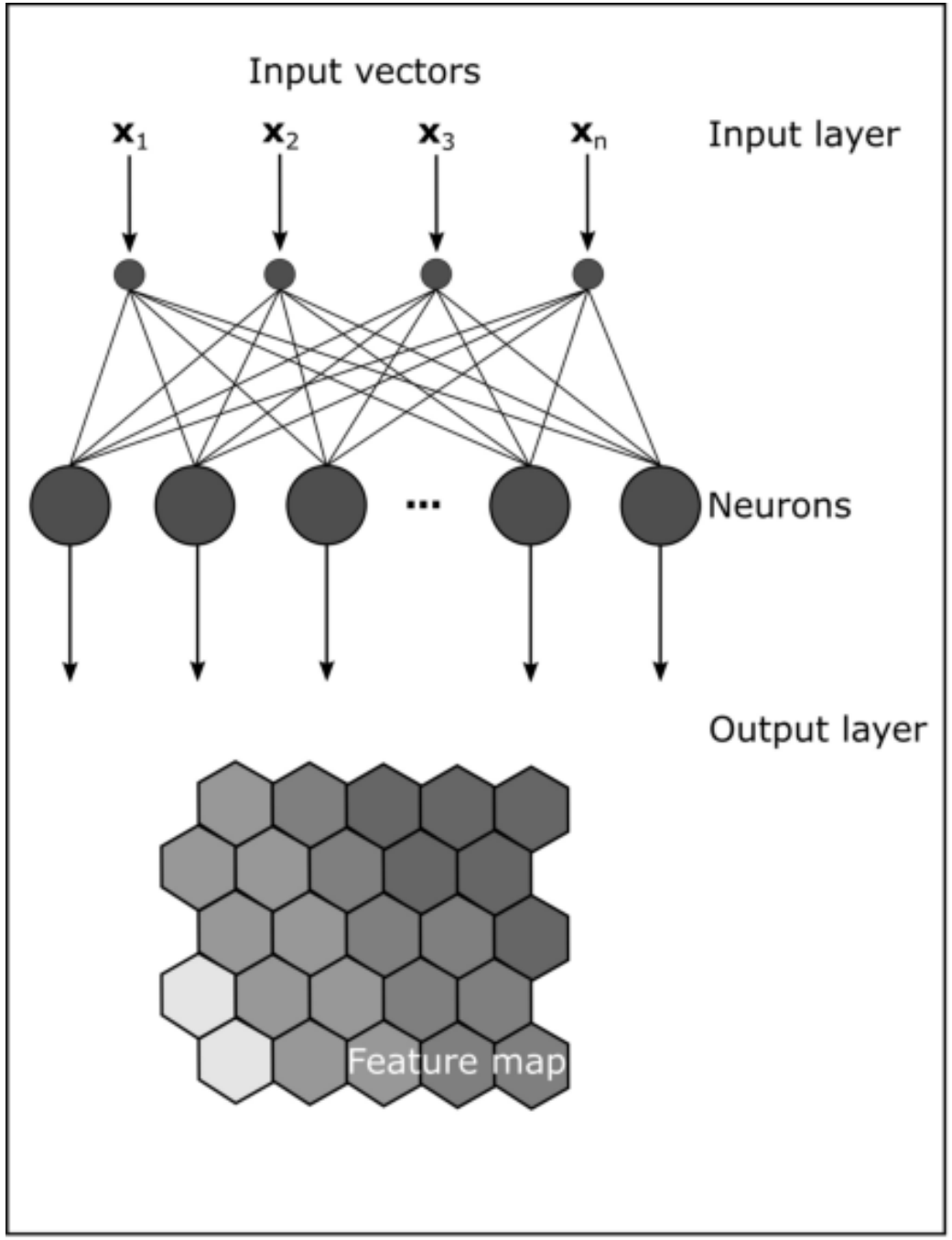
x is the input vector
w_{j}(n) is the vector representation of neuron j at iteration n (w_i(n) is the winning neuron)
\alpha is the learning rate
d is a distance function
For instance, we can use the Gaussian function to focus the winner's influence on the neurons close to it:

$$h(d(i,j)) = e^{-\frac{d^2(i,j)}{2\sigma^2}}$$

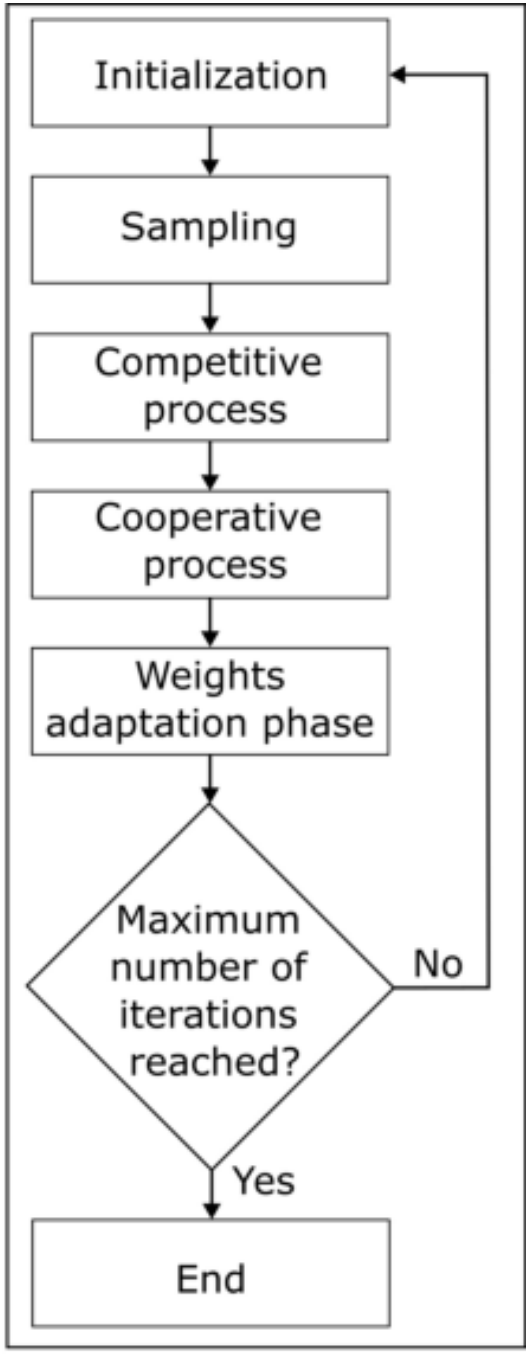
where d(i, j) is the Euclidean distance between neighbor j and the winning neuron i, and \sigma is the standard deviation of the Gaussian function.

In some variations of the algorithm, \sigma decreases with time.

Update the neuron weights so that the winner becomes and resembles the input vector x.

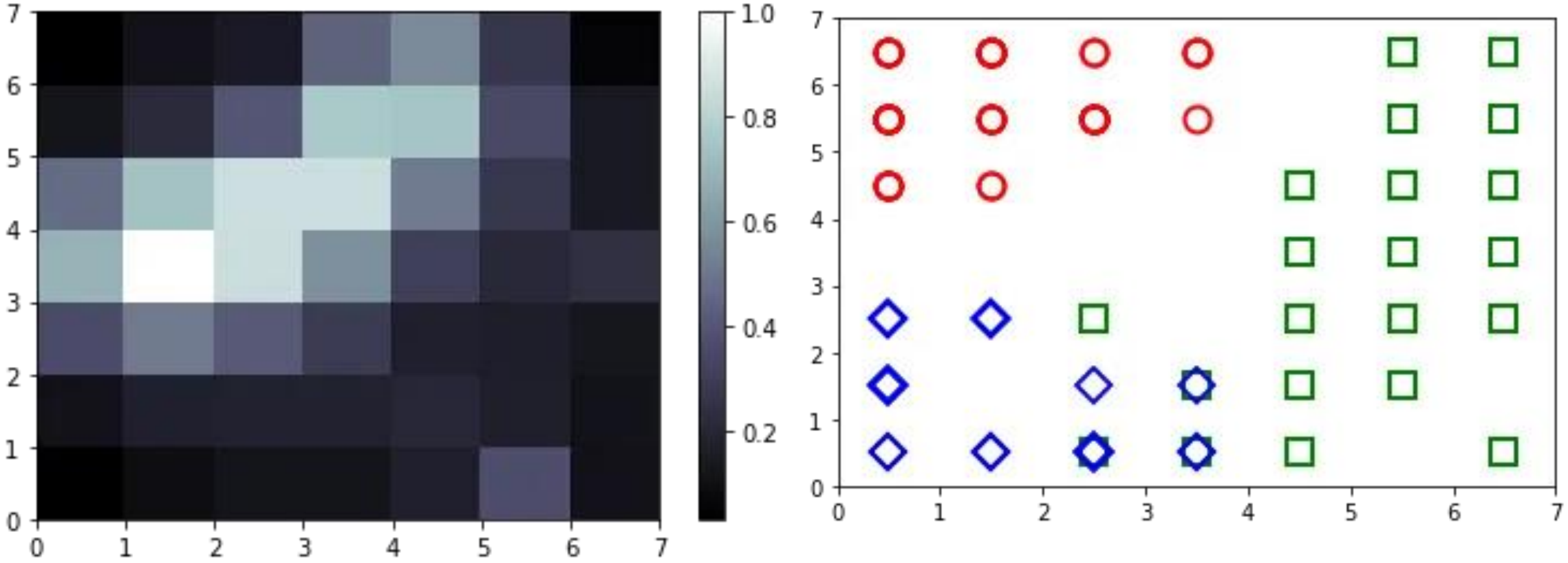


a)



b)

IRIS DATASET



- The self-organizing maps algorithm projects the high-dimensional data into a two-dimensional map while retaining the topology of the data so that similar data points are mapped to the nearby locations on the map.
- The technique classifies the data without any supervision and there is no target vector and hence no backward propagation.
- The output of the SOMs is a two-dimensional map and color-coding is used to identify any specific group of data points.