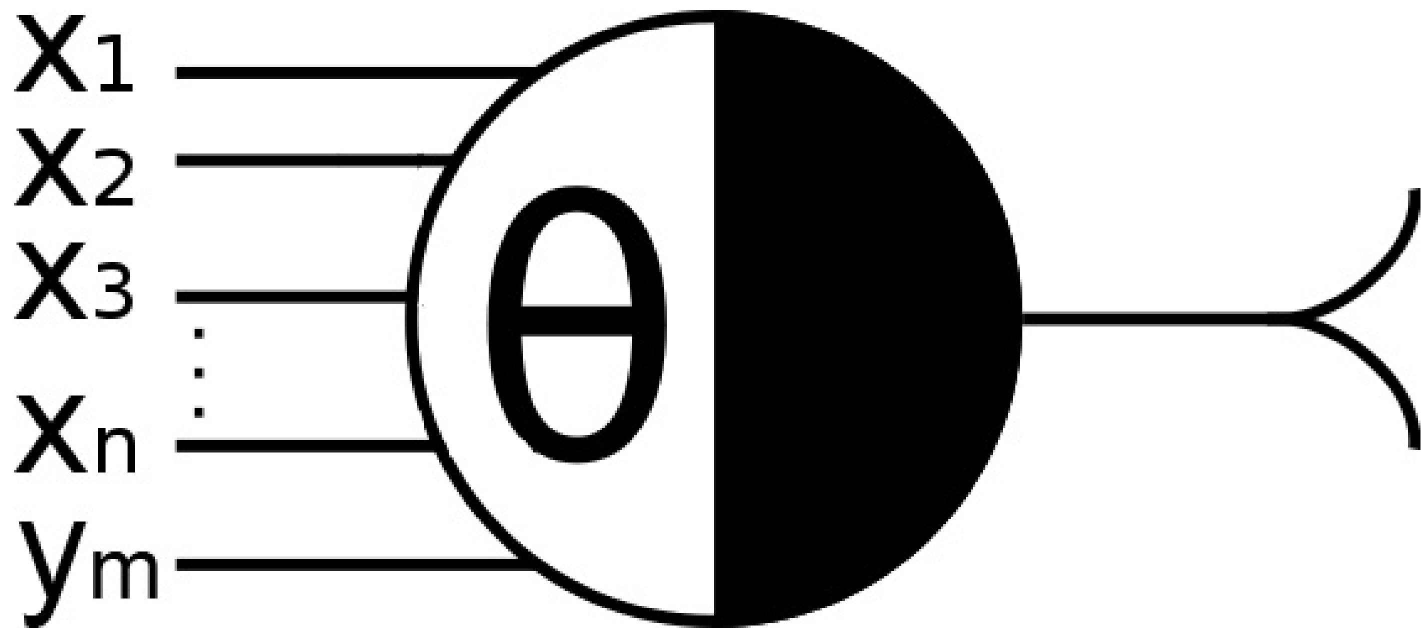
Artificial Neural Networks

McCulloch-Pitts Cell

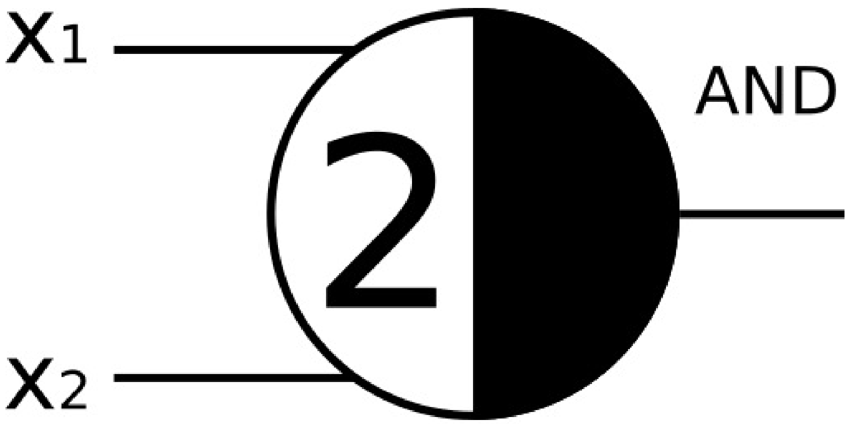
- Introduced 1943 by Warren McCulloch and Walter Pitts
- Simplest model to represent a biological neuron as a computational unit consists of:

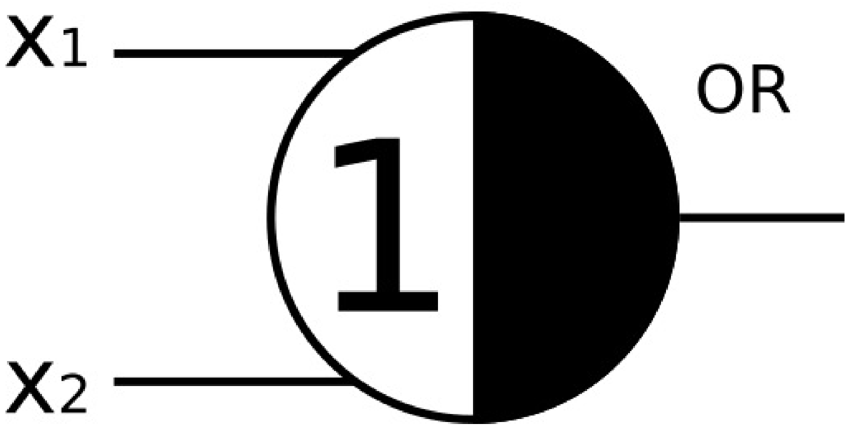
- a node
- *n* exiting binary inputs $x_1, \ldots, x_n$
- *m* inhibitory binary inputs $y_1, \ldots, y_m$
- integer threshold value θ
- binary output

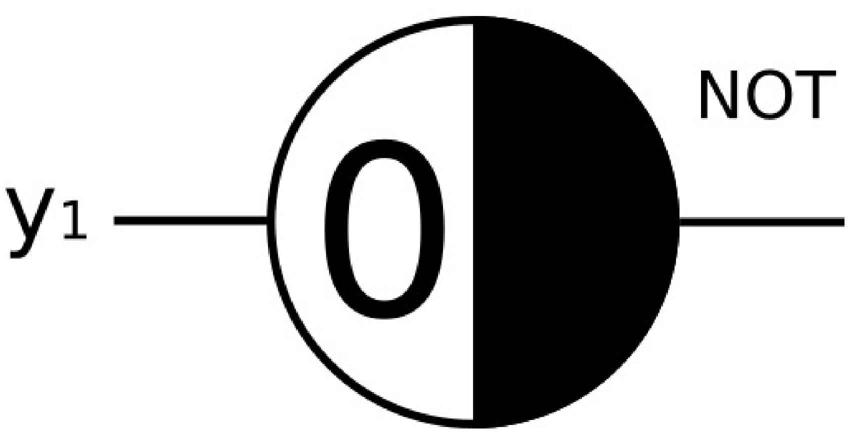


calculation procedure

- if one or more inhibitory inputs *y* are present → output is negative
- else: exciting inputs xn are summed up - if: $\Sigma x < \theta$ → *output is negative*
  $\Sigma x > \theta$ → *output is positive*



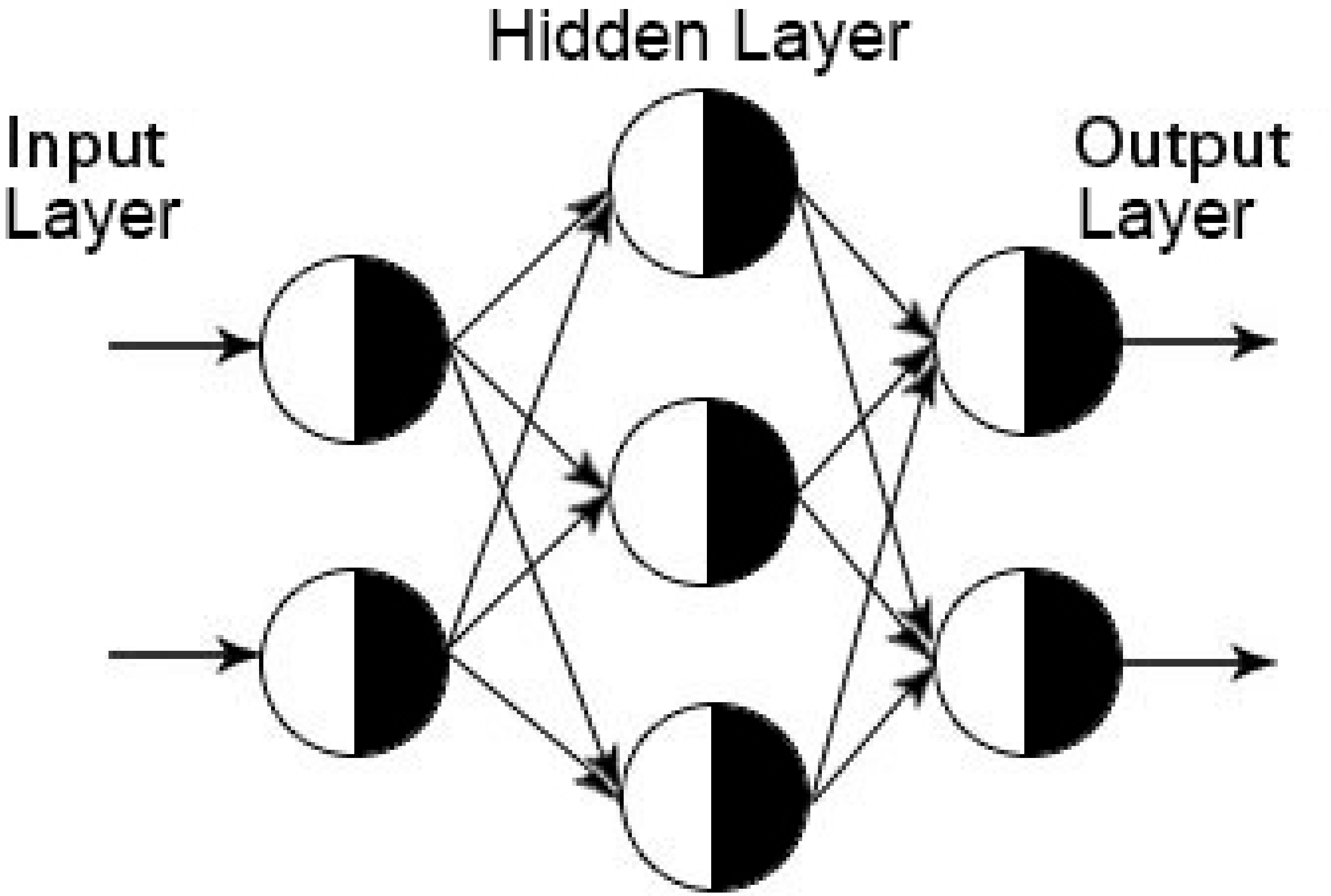




- can perform the basic boolean operations ***AND, OR, NOT***
- thereby supplies a complete basis of boolean algebra (good)
- a single MP cell is NOT able to perform the ***XOR*** function proven by Minsky and Papert in 1969 (bad)

McCulloch-Pitts Network

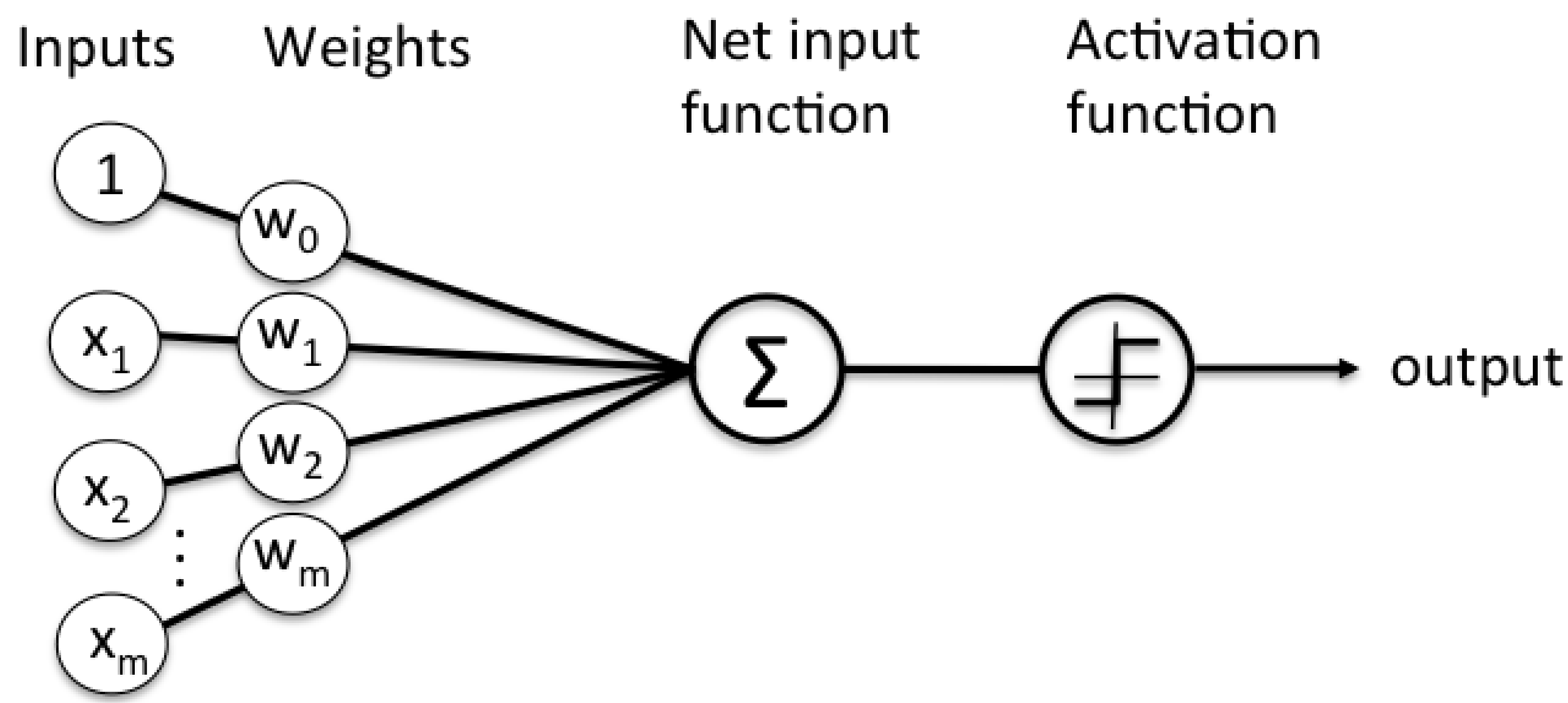
- network of McCulloch Pitts Cells, connected by ***synapses***
- if information is flowing in one direction, it's a ***feed-forward neural network***
- synapses can also form loops or feed information backward ***(feed-back network)***
- input layers
  hidden layers
  output layers
- threshold-controlled networks of boolean operations



Perceptron after Frank Rosenblatt

extends the idea of MC networks by adding:

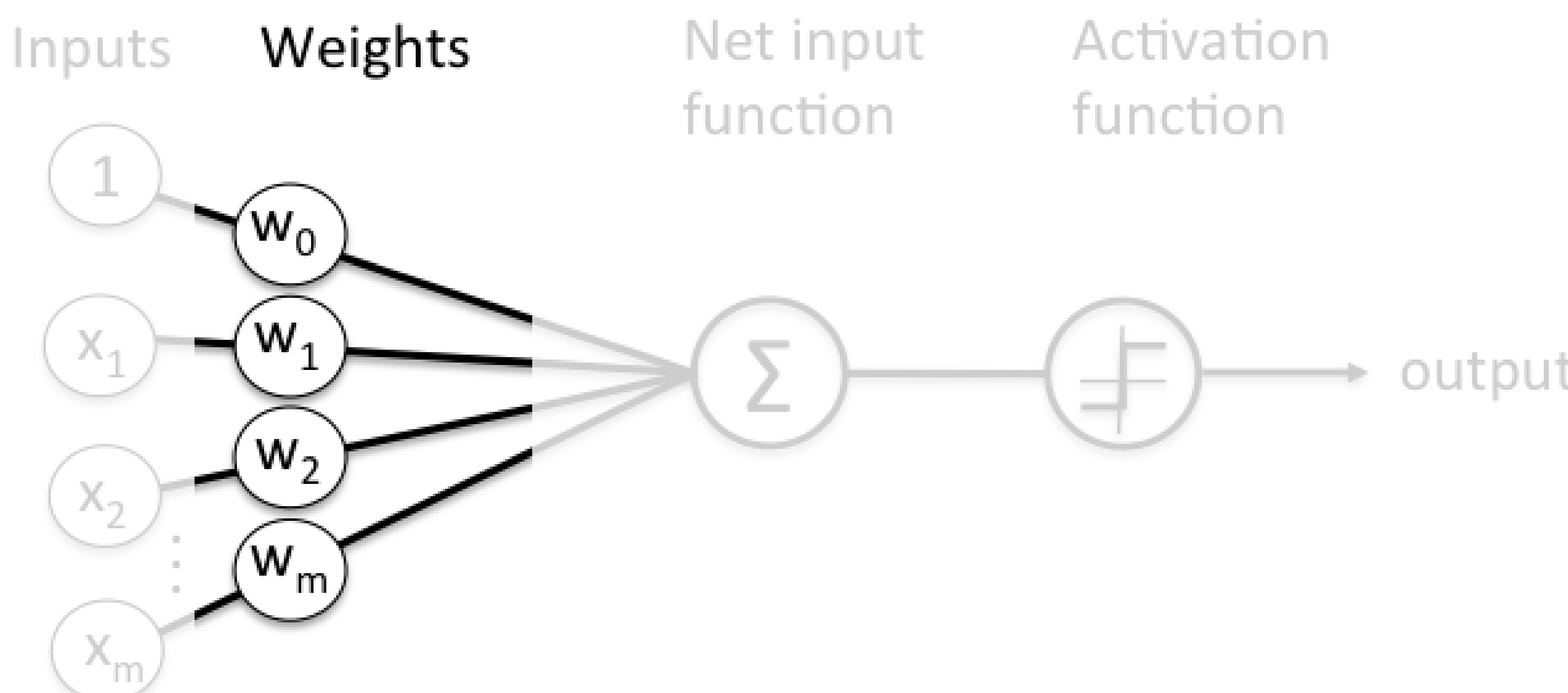
- non-binary inputs / outputs
- synaptic weight *- inputs are multiplied with defined weight*
- activation function *sum of inputs is evaluated using an additional function to generate output*



Sebastian Raschka
Schematics of Rosenblatt's Perceptron
www.sebastianraschka.com/Articles/2015_singlelayer_neurons.html

Perceptron Synaptic Weights

- determine how strongly a certain input is taken into consideration
- are ***updated*** over time to perform ***learning***

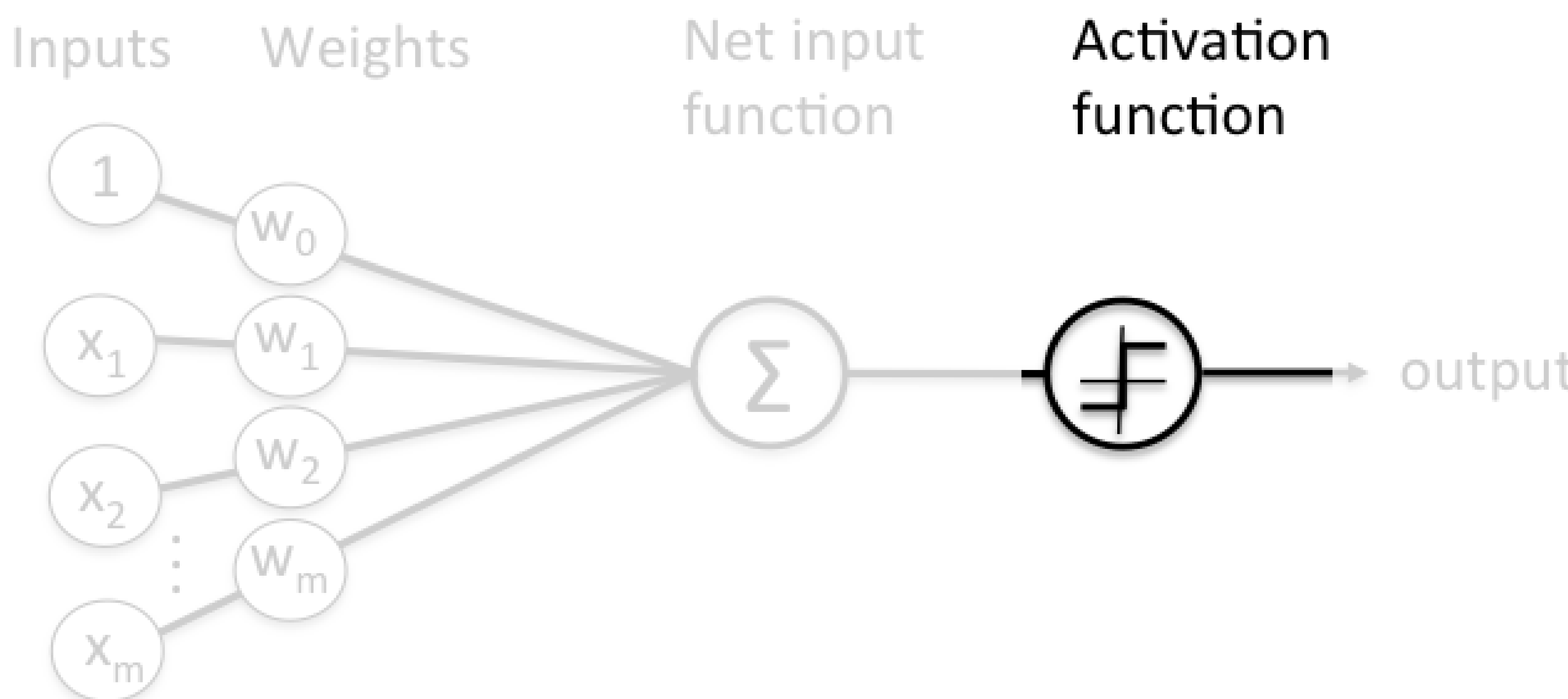


Perceptron Activation Function

- weighted input sum is evaluated using a pre-defined function

- choice of suitable activation function is task specific

- most common:
  - Linear Function
  - Sigmoid
  - Sine
  - Logarithmic
  - TanH



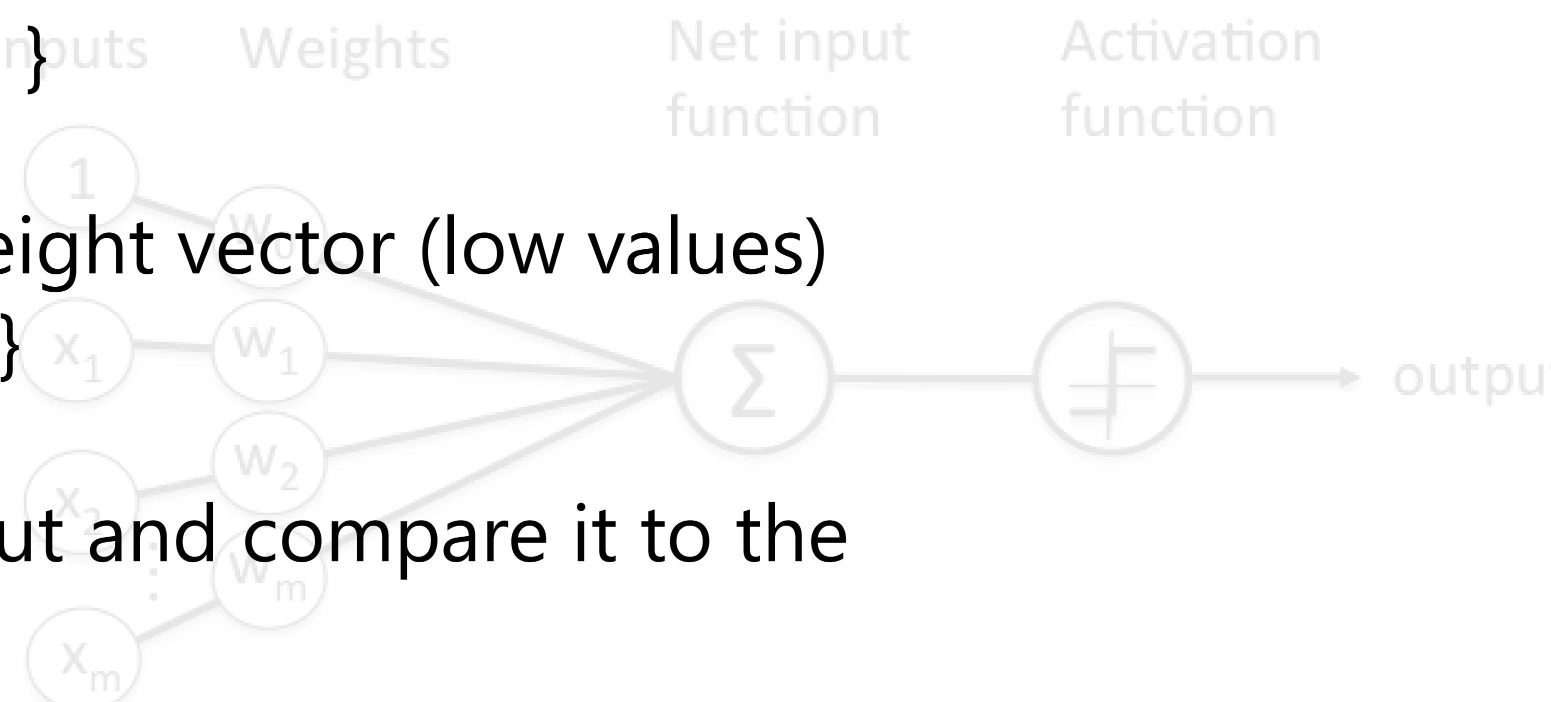
Perceptron Supervised Learning

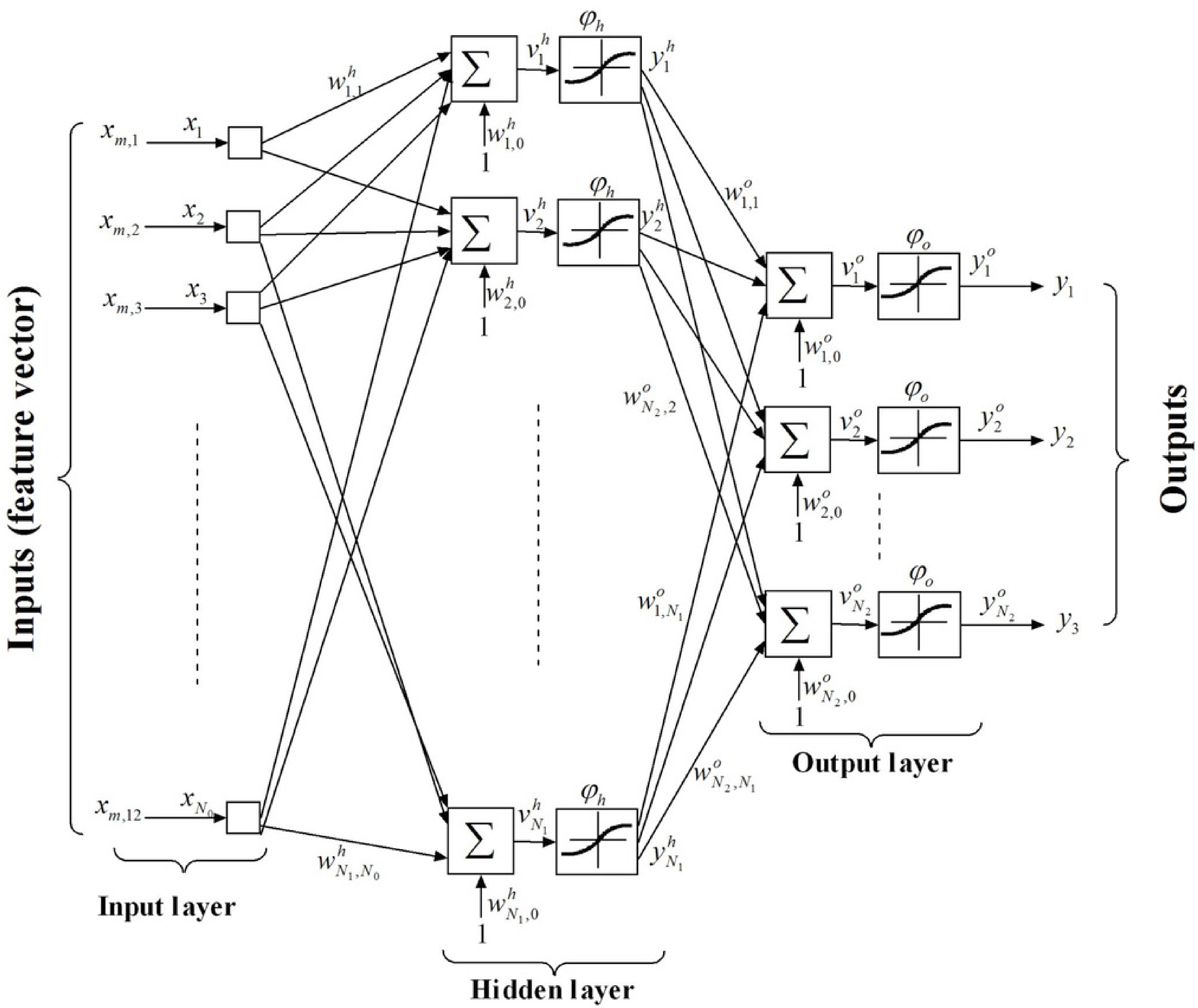
- define an input vector
  X_{in} = {0.67, 0.32, ... }

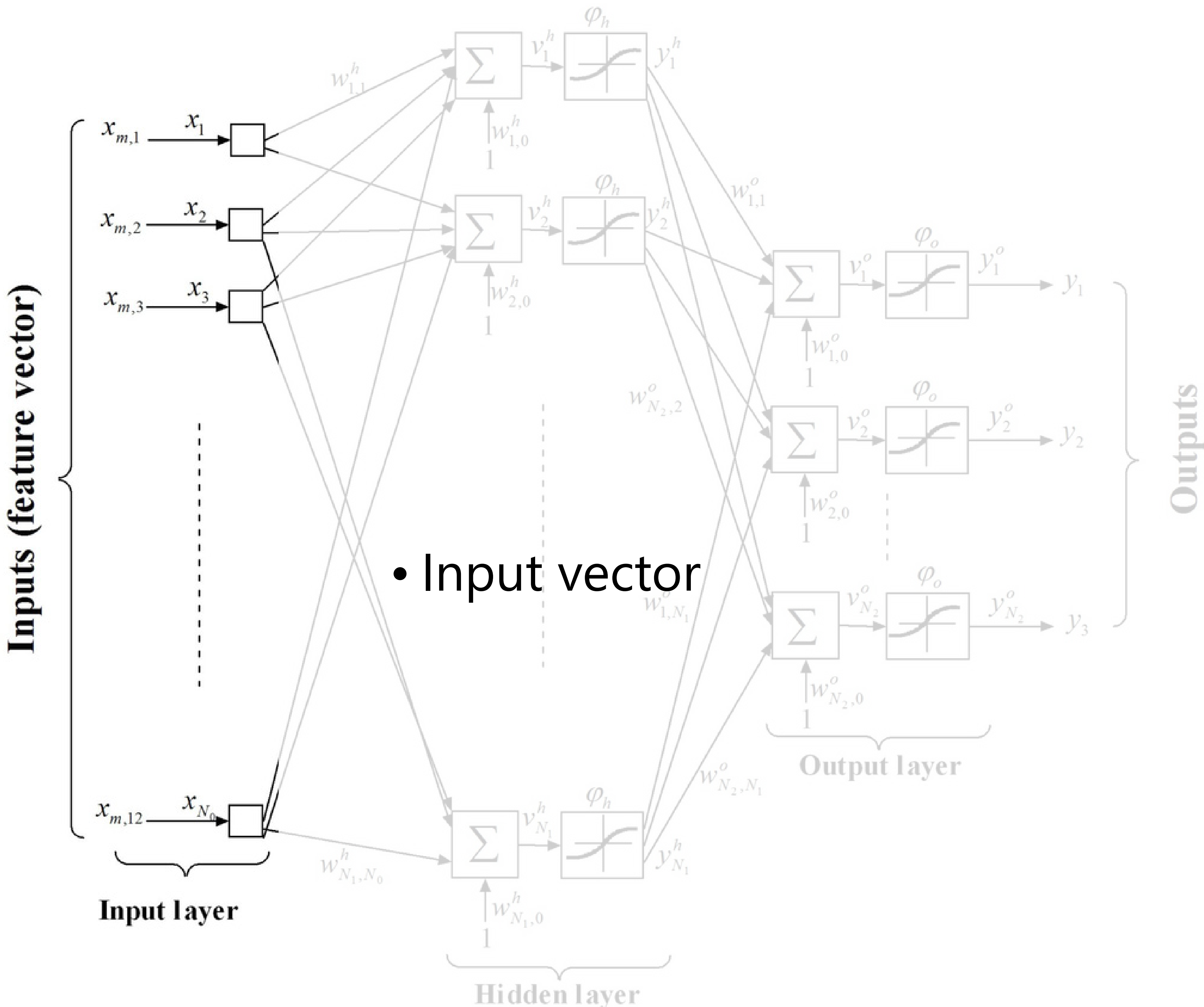
- define an initial weight vector (low values)
  w = {0.00, 0.01, ... }

- compute the output and compare it to the desired output

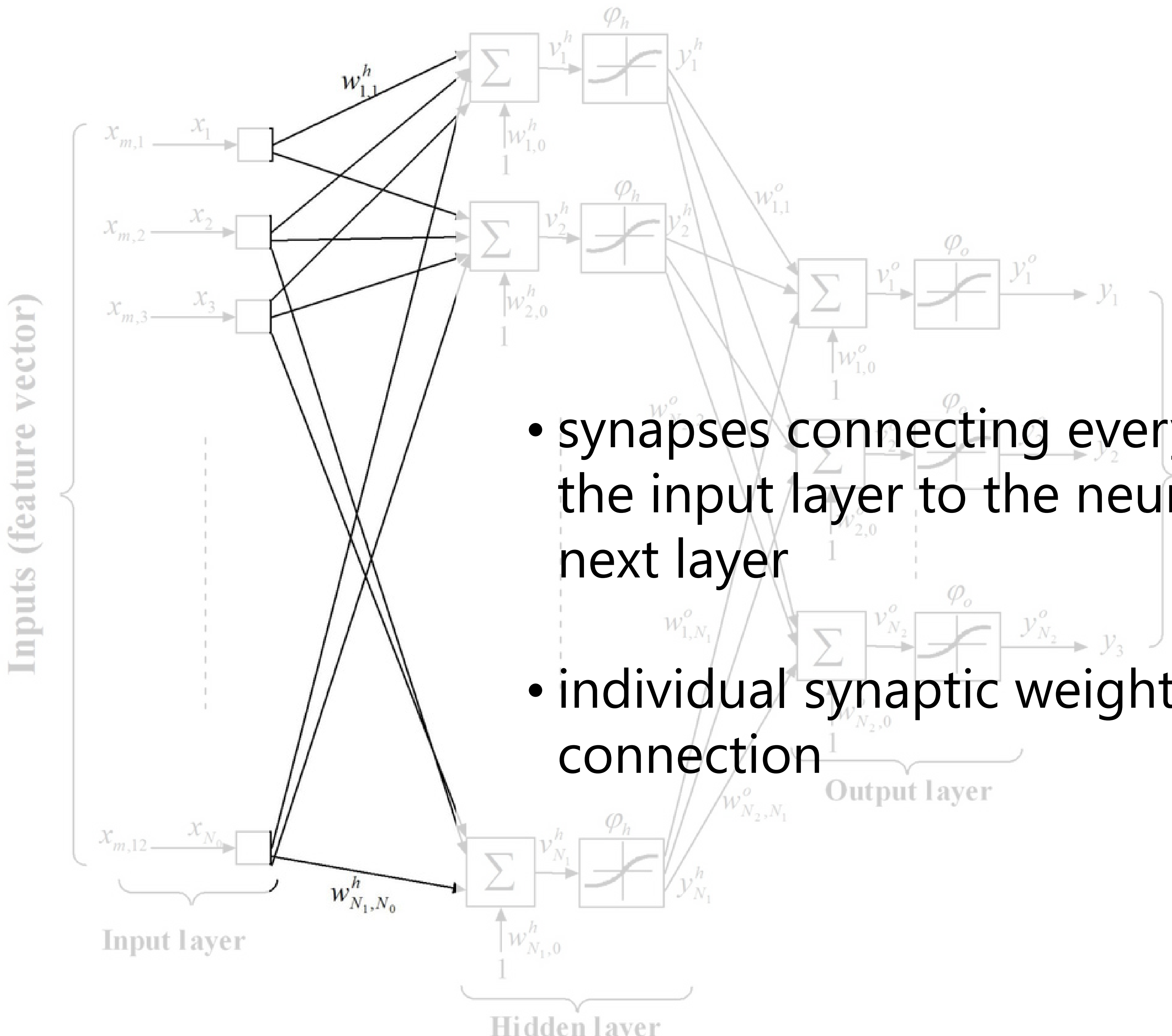
- update the weights until the desired output is generated




Inputs (feature vector)
$x_{m,1}$
x_1
$x_{m,2}$
x_2
$x_{m,3}$
x_3
$x_{m,12}$
x_{N_0}
Input layer
$w^h_{1,1}$
$w^h_{N_1,N_0}$
$w^h_{1,0}$
$w^h_{2,0}$
$w^h_{N_1,0}$
v^h_1
v^h_2
$v^h_{N_1}$
φ_h
y^h_1
y^h_2
$y^h_{N_1}$
Hidden layer
$w^o_{1,1}$
$w^o_{N_2,2}$
w^o_{1,N_1}
$w^o_{N_2,N_1}$
$w^o_{1,0}$
$w^o_{2,0}$
$w^o_{N_2,0}$
v^o_1
v^o_2
$v^o_{N_2}$
φ_o
y^o_1
y^o_2
$y^o_{N_2}$
y_1
y_2
y_3
Output layer
Outputs

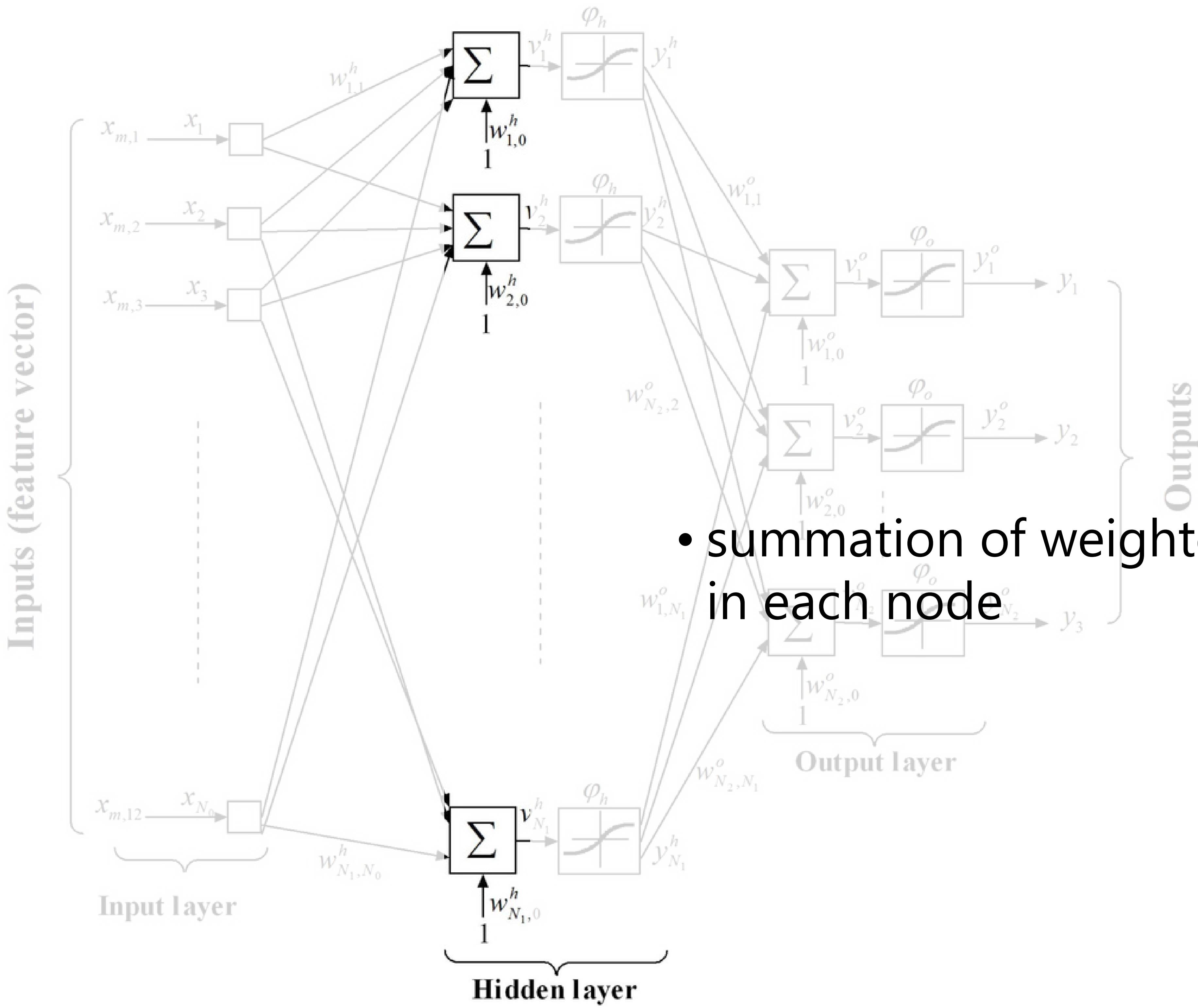


- Input vector

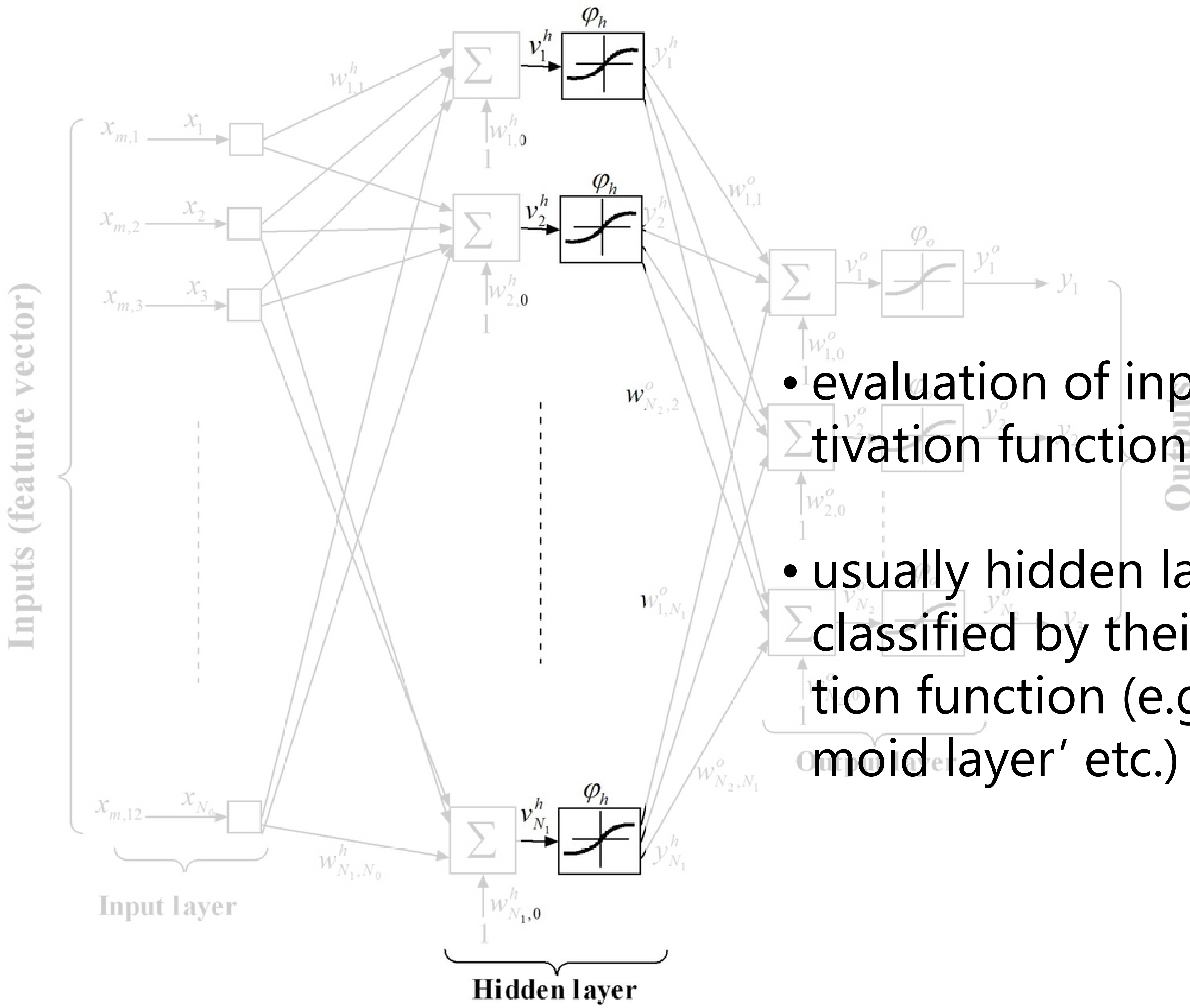


- synapses connecting every neuron of the input layer to the neuron of the next layer

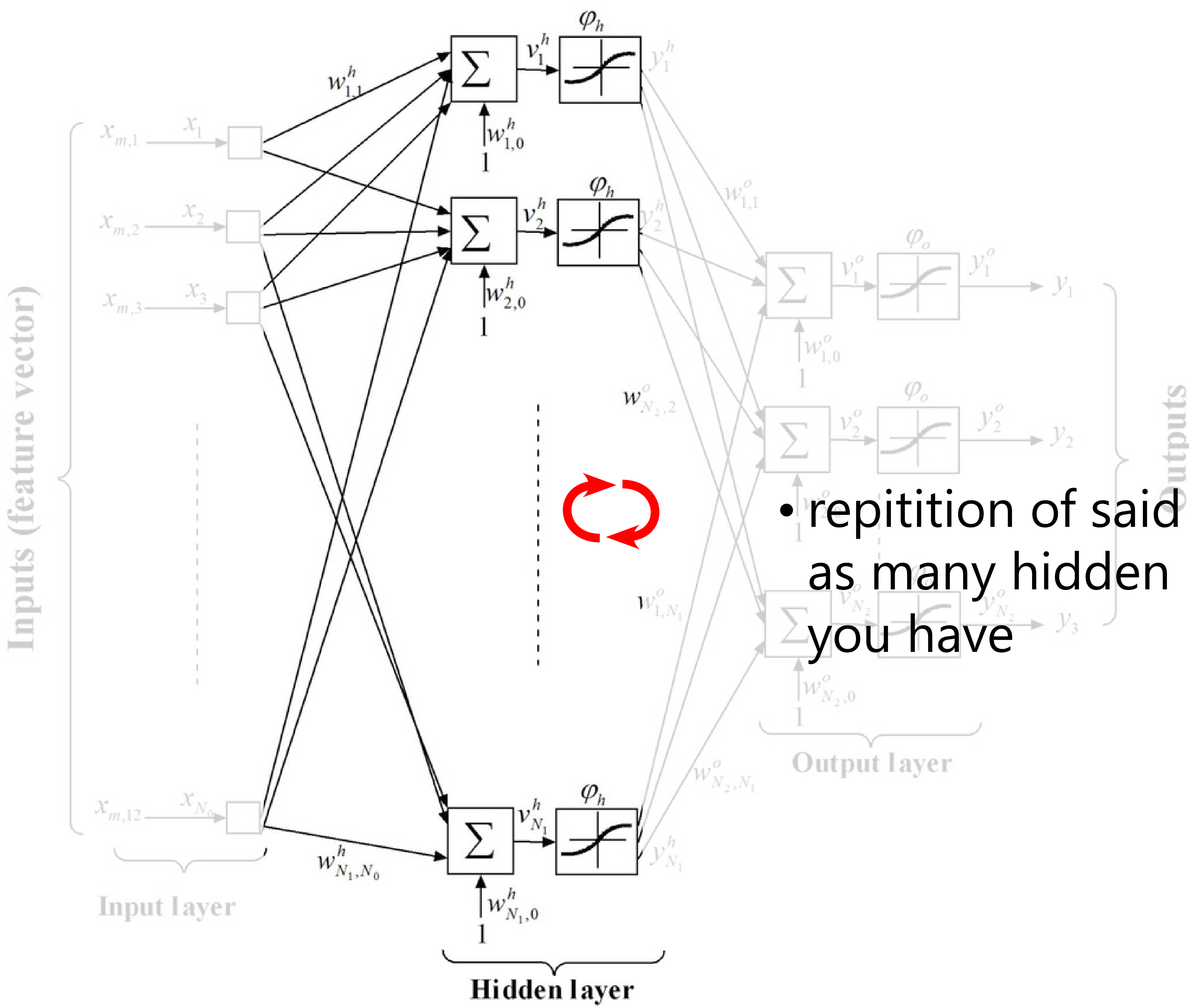
- individual synaptic weight for each connection



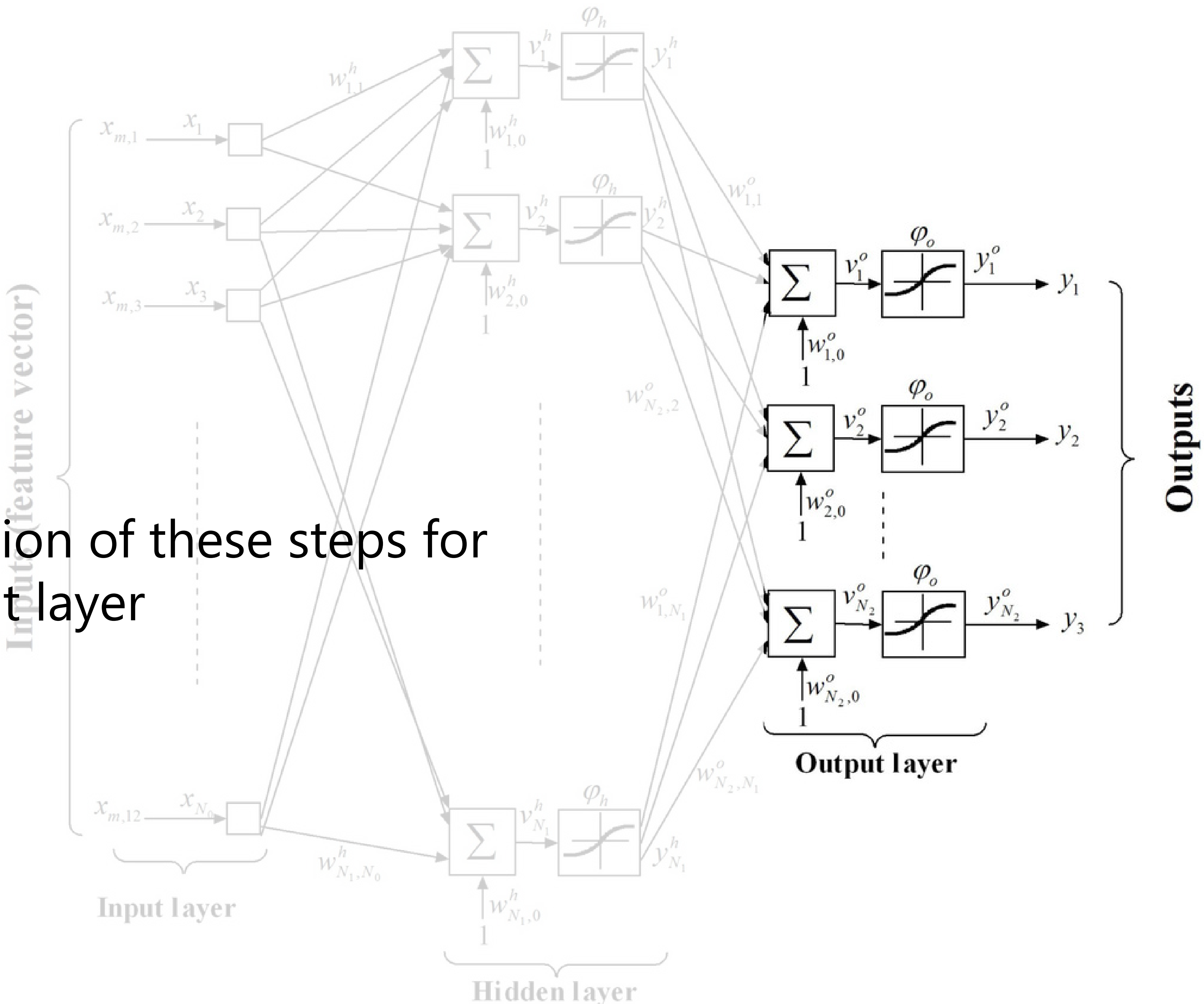
- summation of weighted inputs in each node



- evaluation of inputs in activation function
- usually hidden layers are classified by their activation function (e.g. 'Sigmoid layer' etc.)



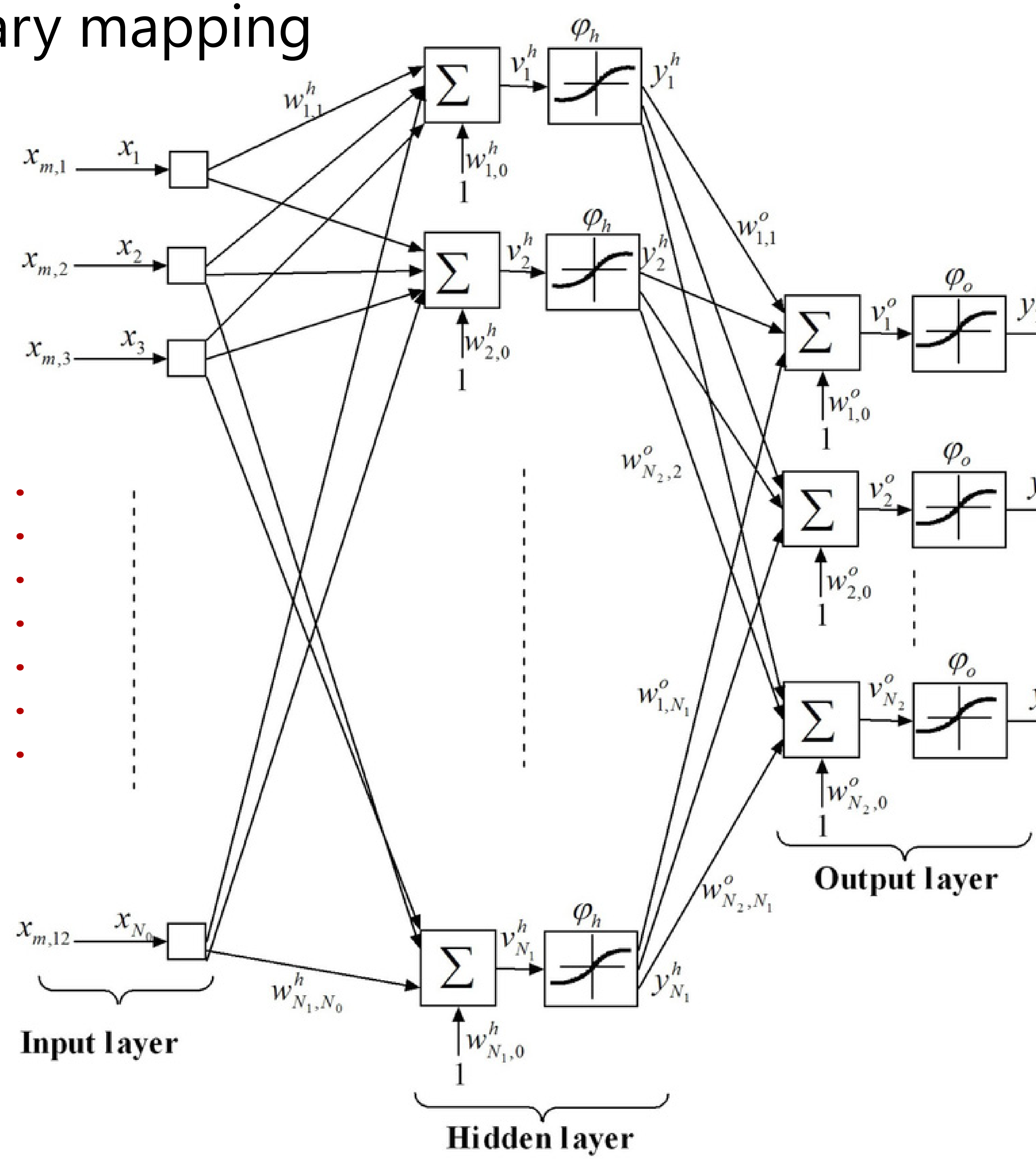
- repitition of said steps for as many hidden layers as you have



- repitition of these steps for output layer

Example: Binary mapping

1 0 1 **1**
0 0 0 **1**
1 1 0 **0**
1 0 0 **1**



0.32 **0** 1 0 1
0.87 **0** 0 1 1
0.03 **1** 0 0 0

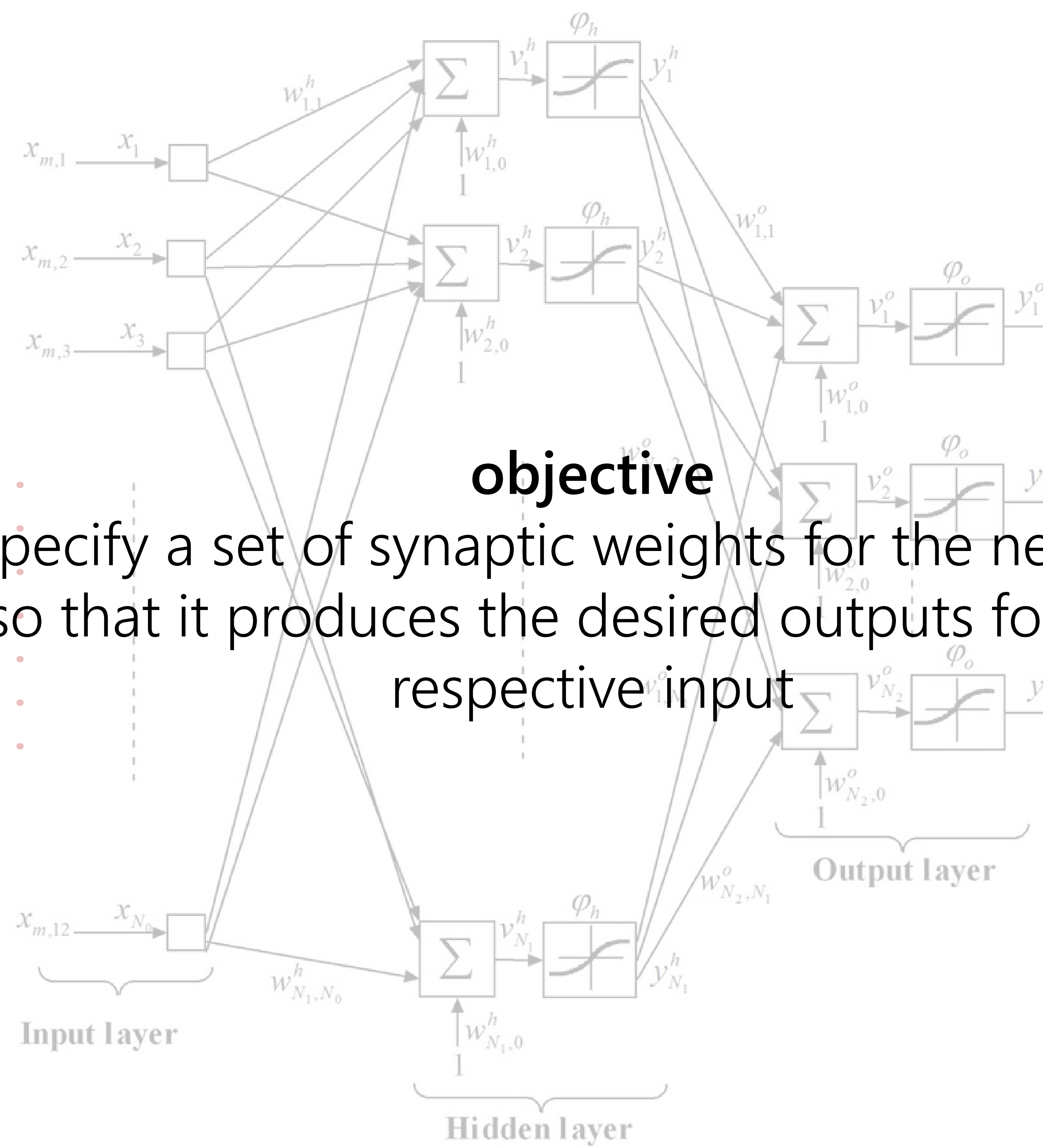
desired output

Binary input

network output

1 0 1 1
0 0 0 1
1 1 0 0
1 0 0 1

Binary input



desired output

0.32 0 1 0 1
0.87 0 0 1 1
0.03 1 0 0 0

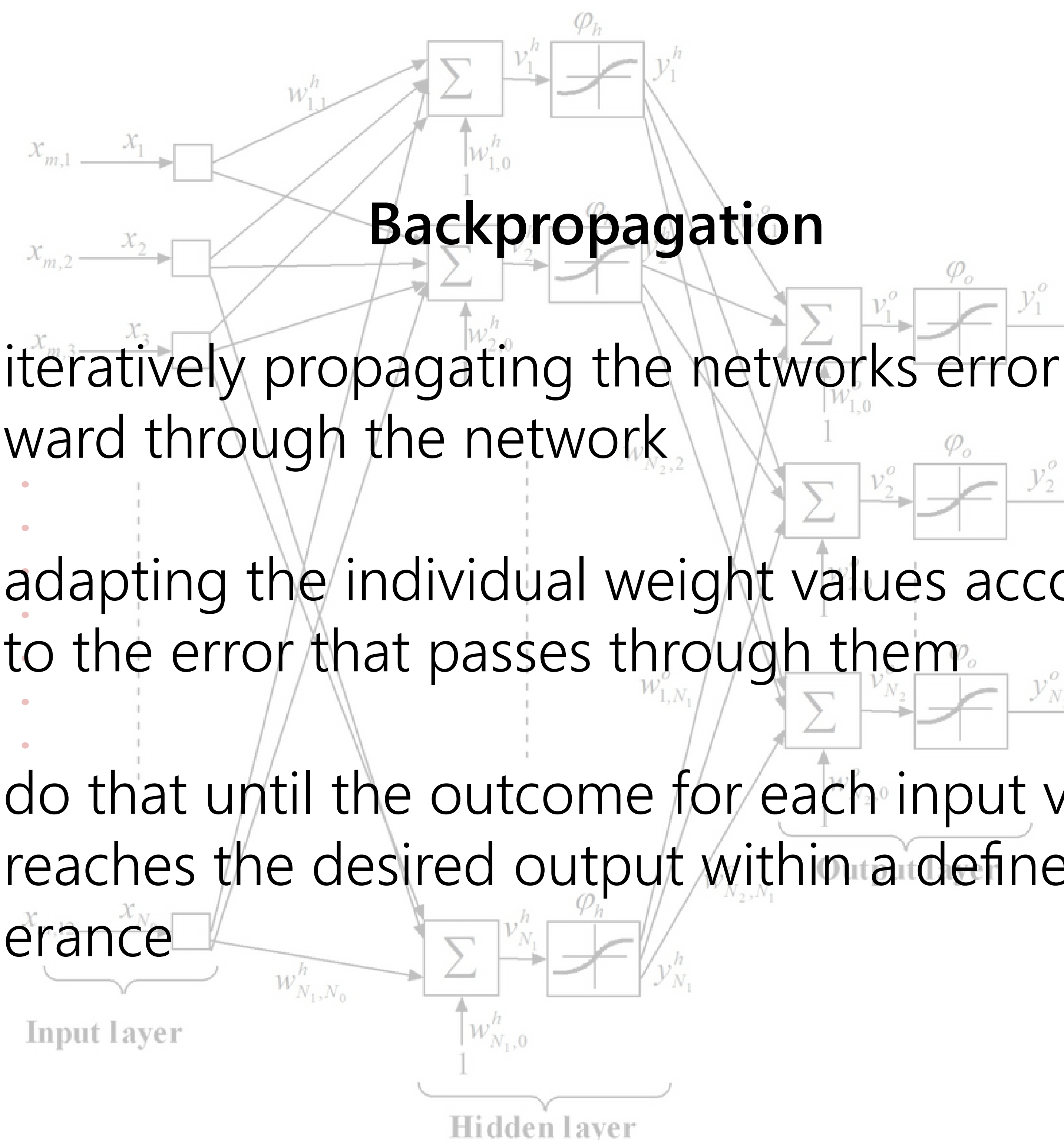
network output

objective

specify a set of synaptic weights for the network, so that it produces the desired outputs for each respective input

Backpropagation

- iteratively propagating the networks error back-ward through the network
- adapting the individual weight values according to the error that passes through them
- do that until the outcome for each input vector reaches the desired output within a defined tol-erance



Backpropagation

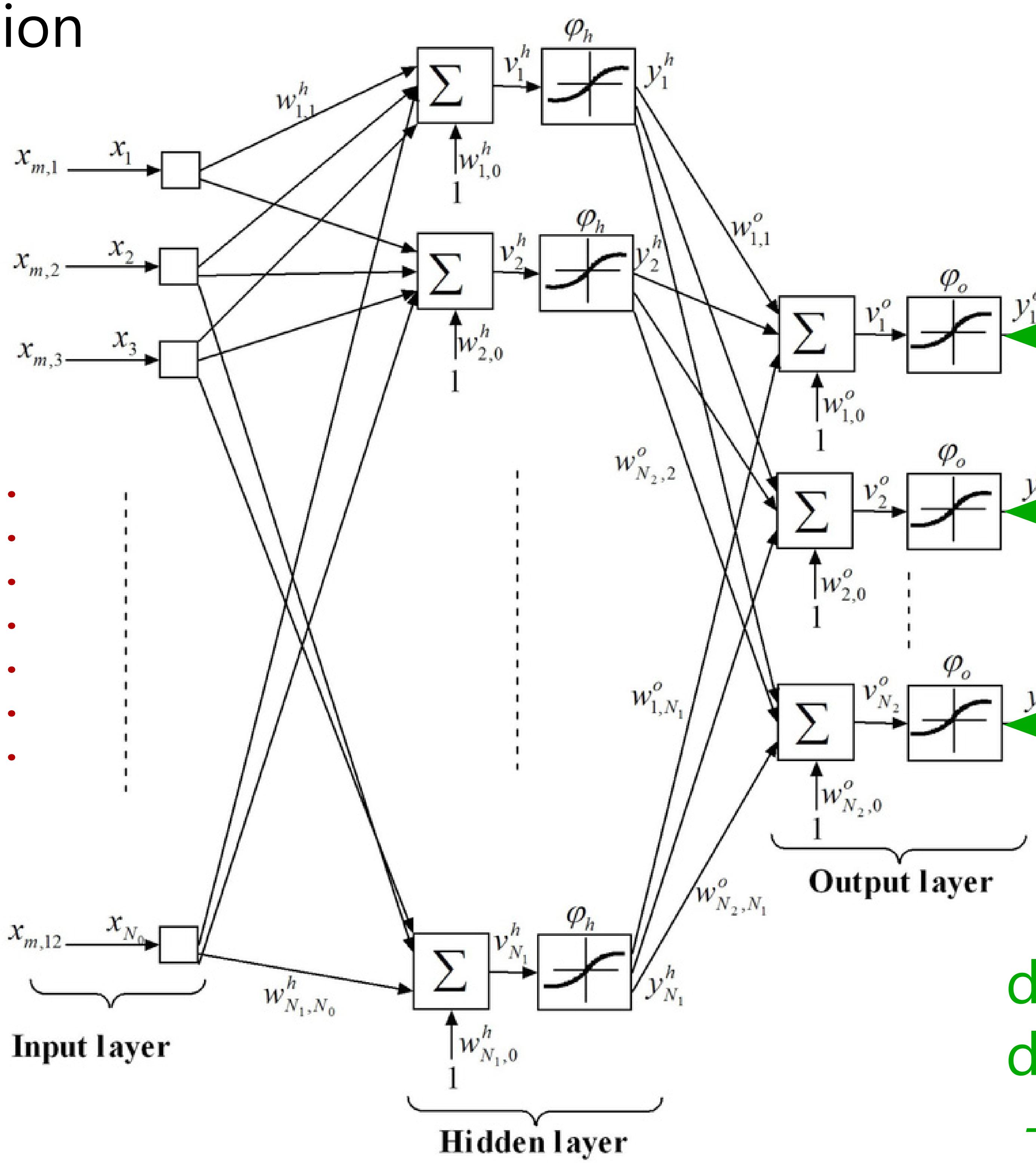
1 0 1 **1**

0 0 0 **1**

1 1 0 **0**

1 0 0 **1**

Binary input



0.32 **0** 1 0 1

0.87 **0** 0 1 1

0.97 **1** 0 0 0

desired output

deviation from desired output
- *error* -

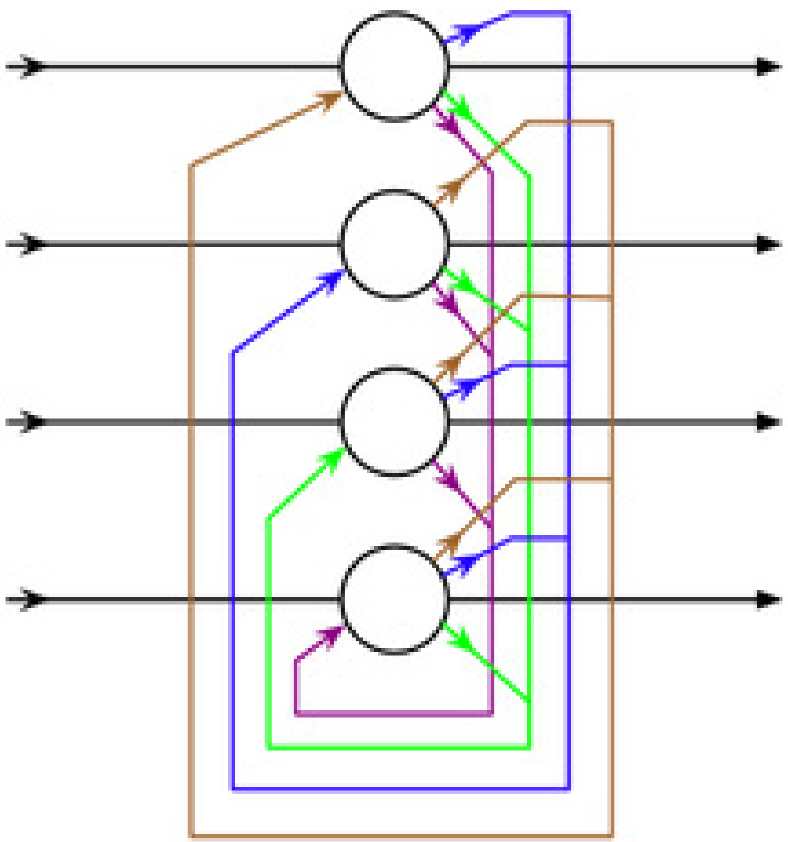
Video … ?

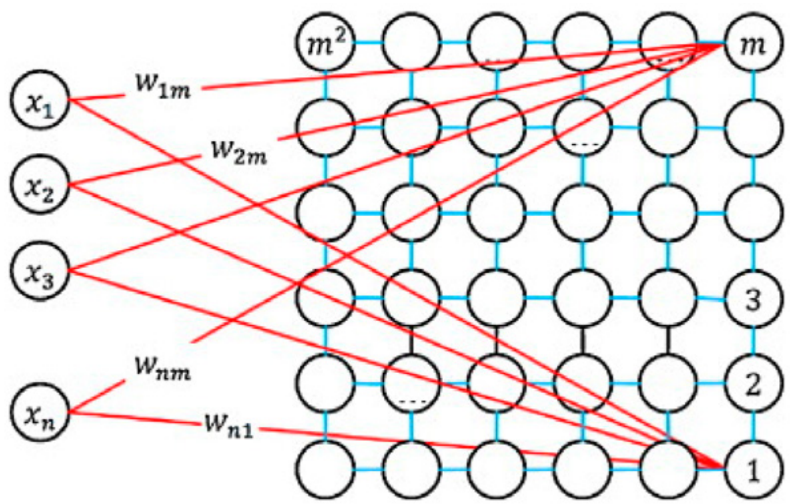
...but why?

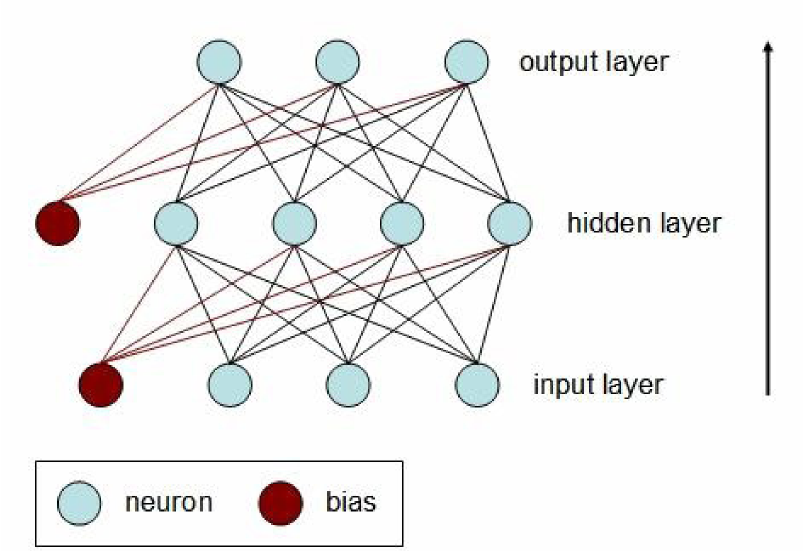
- by performing BP training a network can learn a *specific vector mapping*
- specific vector mapping is nothing but *function approximation*
- interpolation between data samples works to a certain degree
  → data can be classified in a *fuzzy* way

Examples one and two:
150523_Crow_Showcase_Backpropagation
150523_Crow_Showcase_Backpropagation2

More topologies







Hopfield Network

- recurrent network
- every node feeds back into every node
- used for content adress-able memory

Kohonen Network

- single layer network
- neurons are connected to neighbors through a neighborhood function
- unsupervised learning

Constant Bias

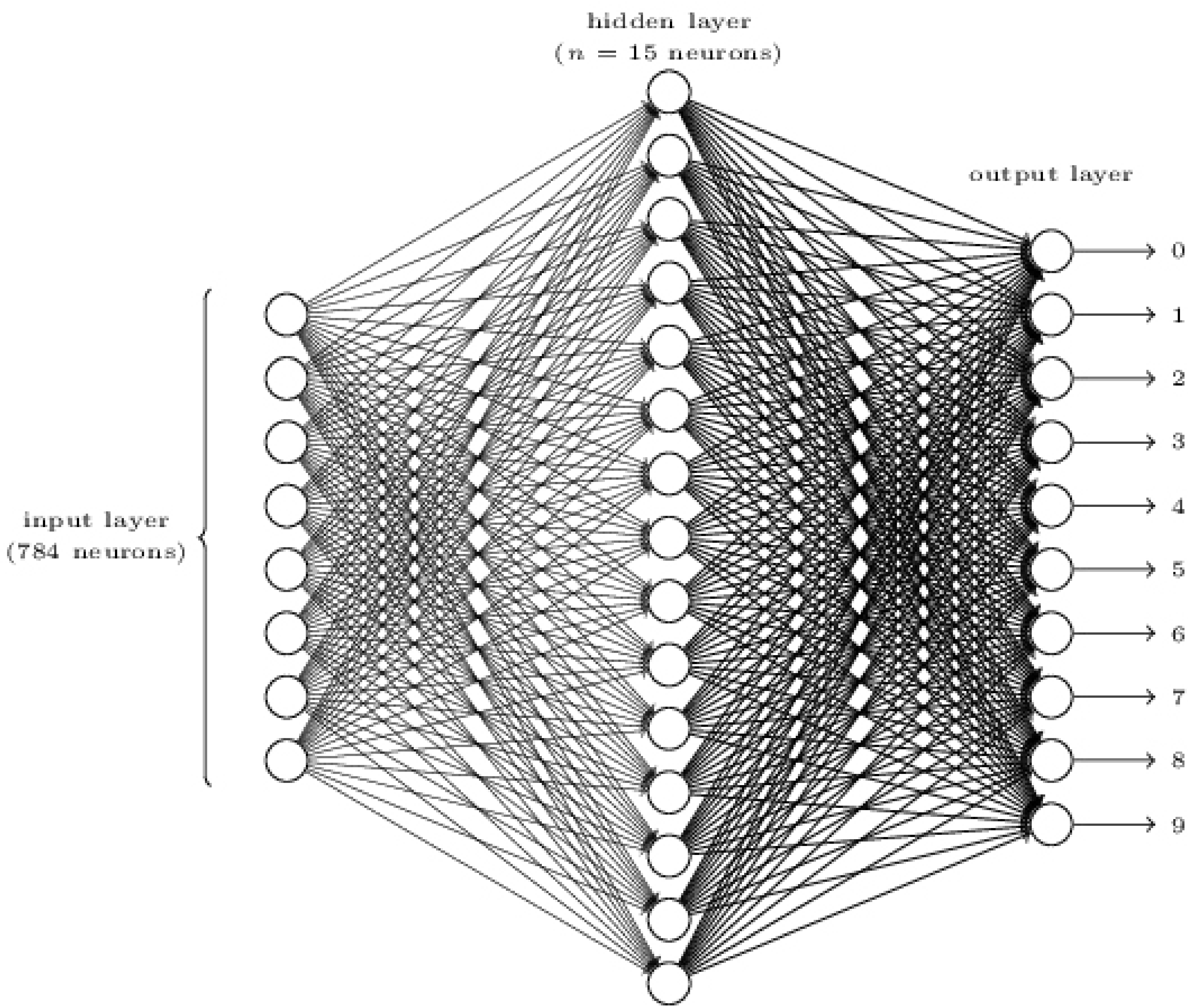
- static value affects network behavior
- used if needed

Handwriting recognition - Digits

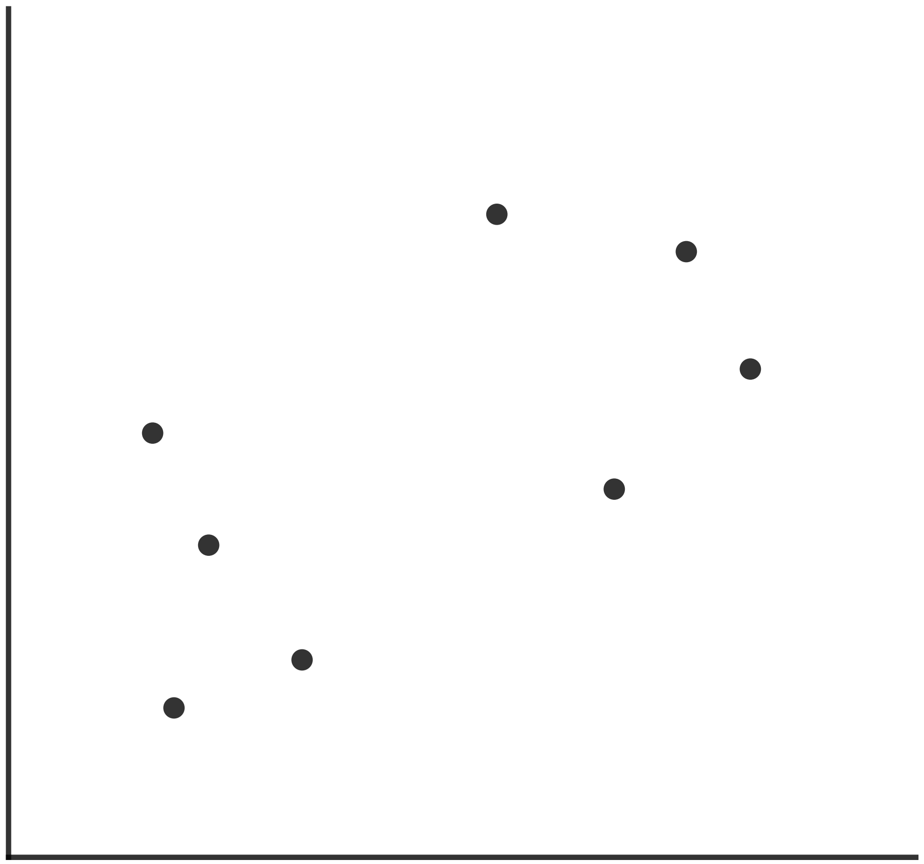
each digit represents a ***labelled training sample***

input vector:
flattened pixel information
0.0 to 1.0 grey scale

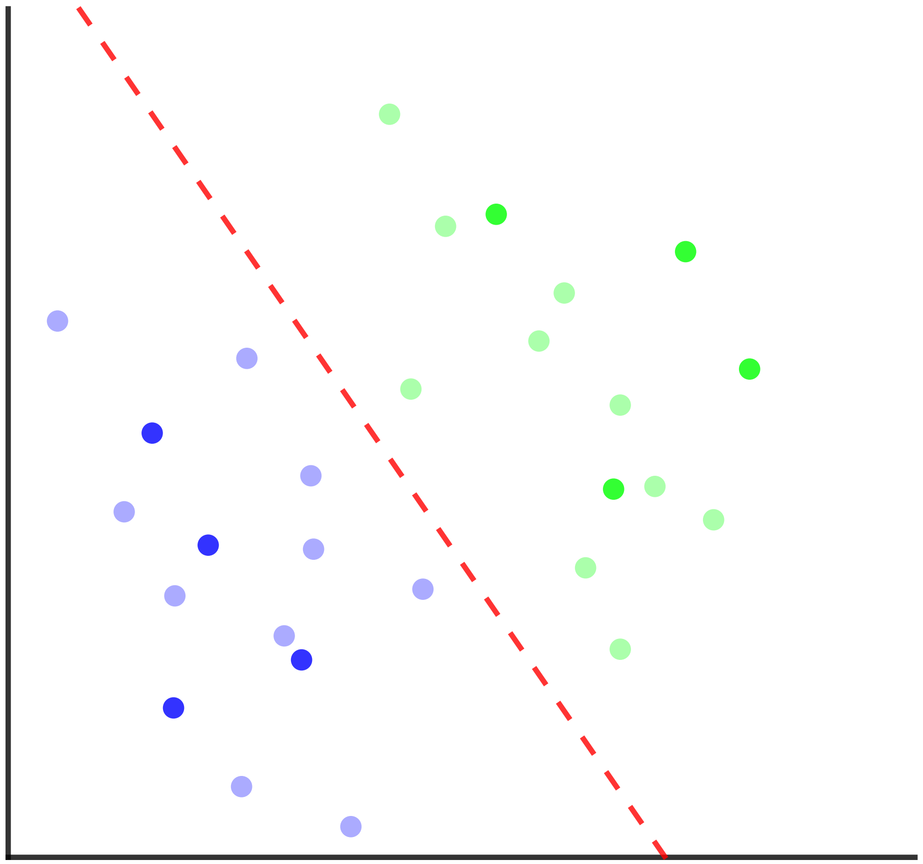
(data collection: MNIST - standard for benchmarking algorithms)


hidden layer
(n = 15 neurons)
output layer
input layer
(784 neurons)
0
1
2
3
4
5
6
7
8
9

Essentially....



...labeled data is used to generate a decision rule...



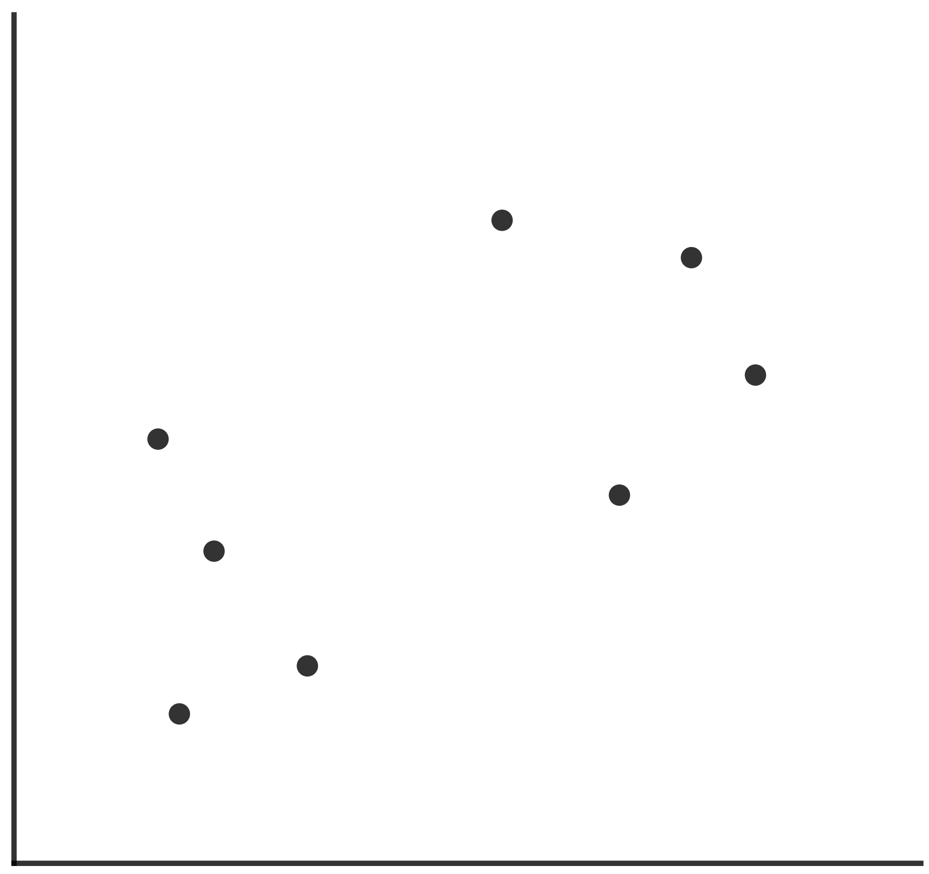
...on how to classify unla-beld data

Unsupervised Learning Concept

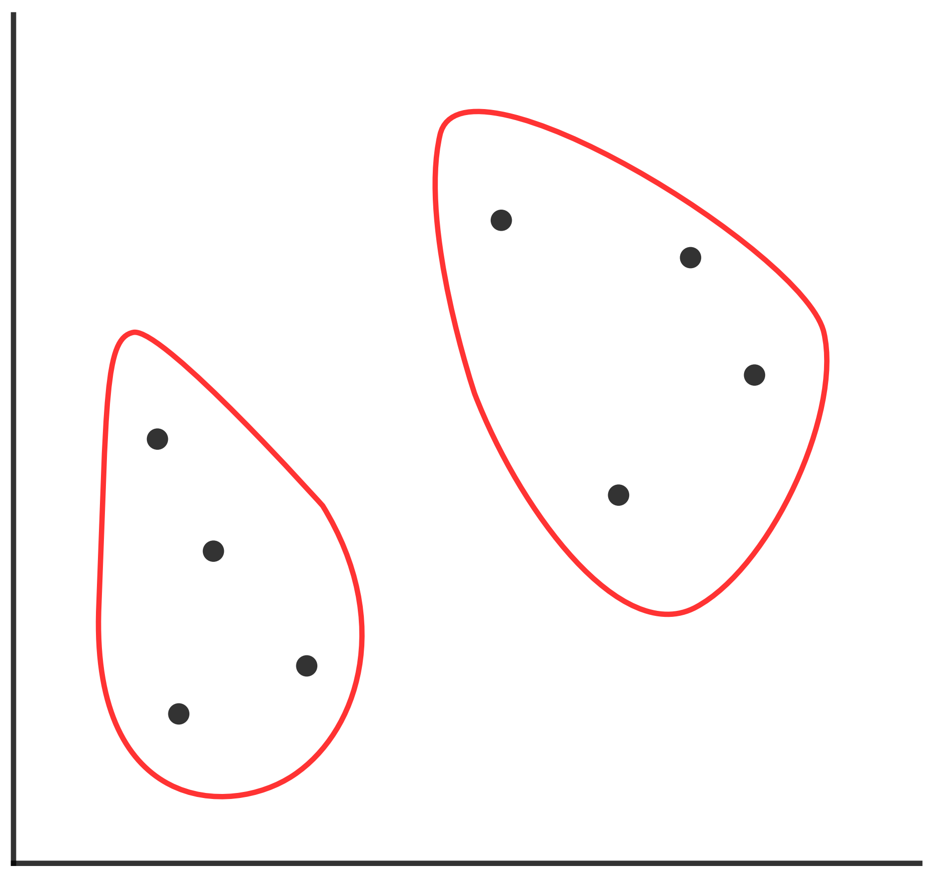
- general term in machine learning for a function describing *hidden* structure in *unlabeled* data

- no reward or error signal to evaluate a potential solution

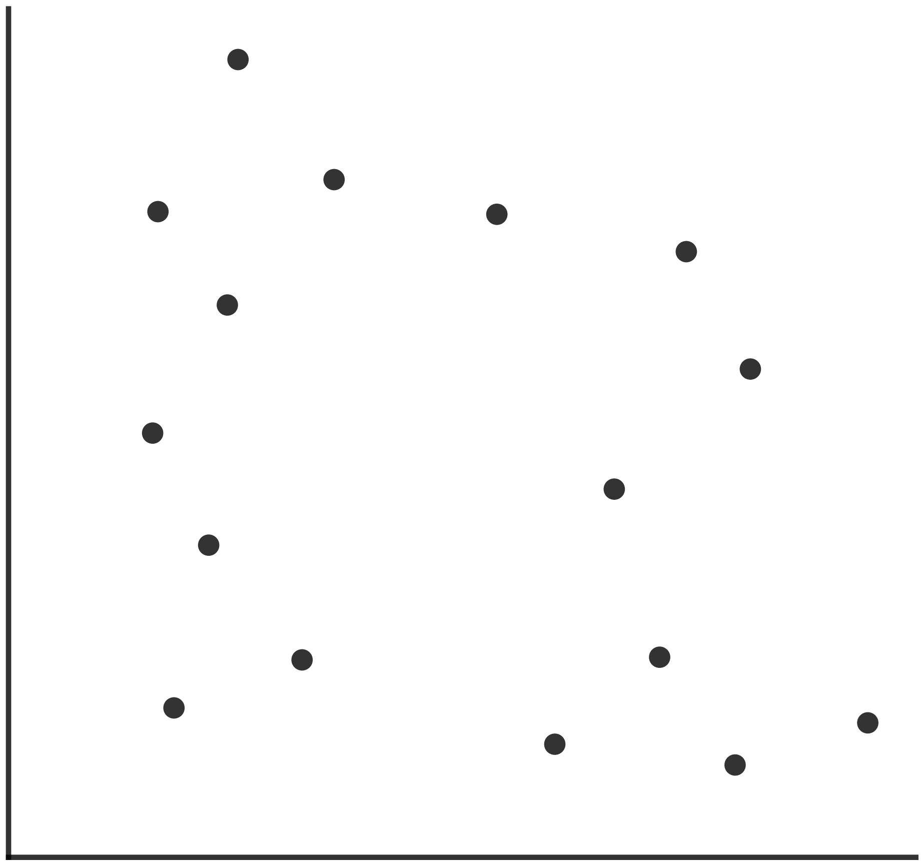
- in neural networks:
  - Self-Organizing Maps
  - Adaptive Resonance Theory



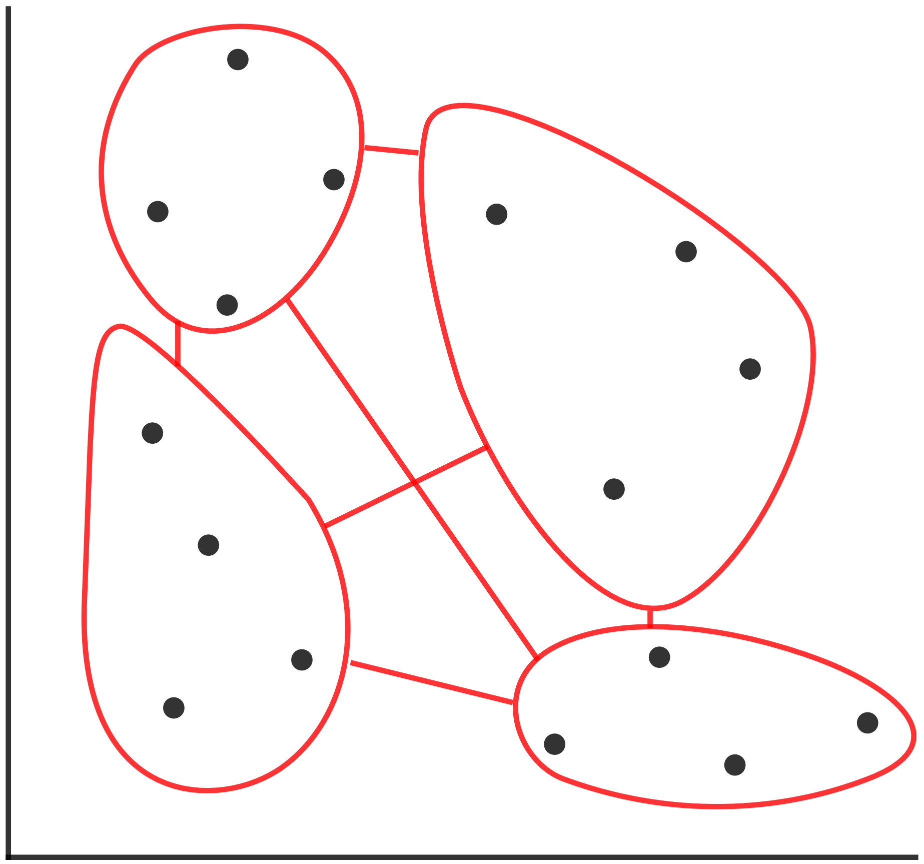
unlabeled data set
(2D)



- clustering
  detection of hidden structure through unsupervised learning



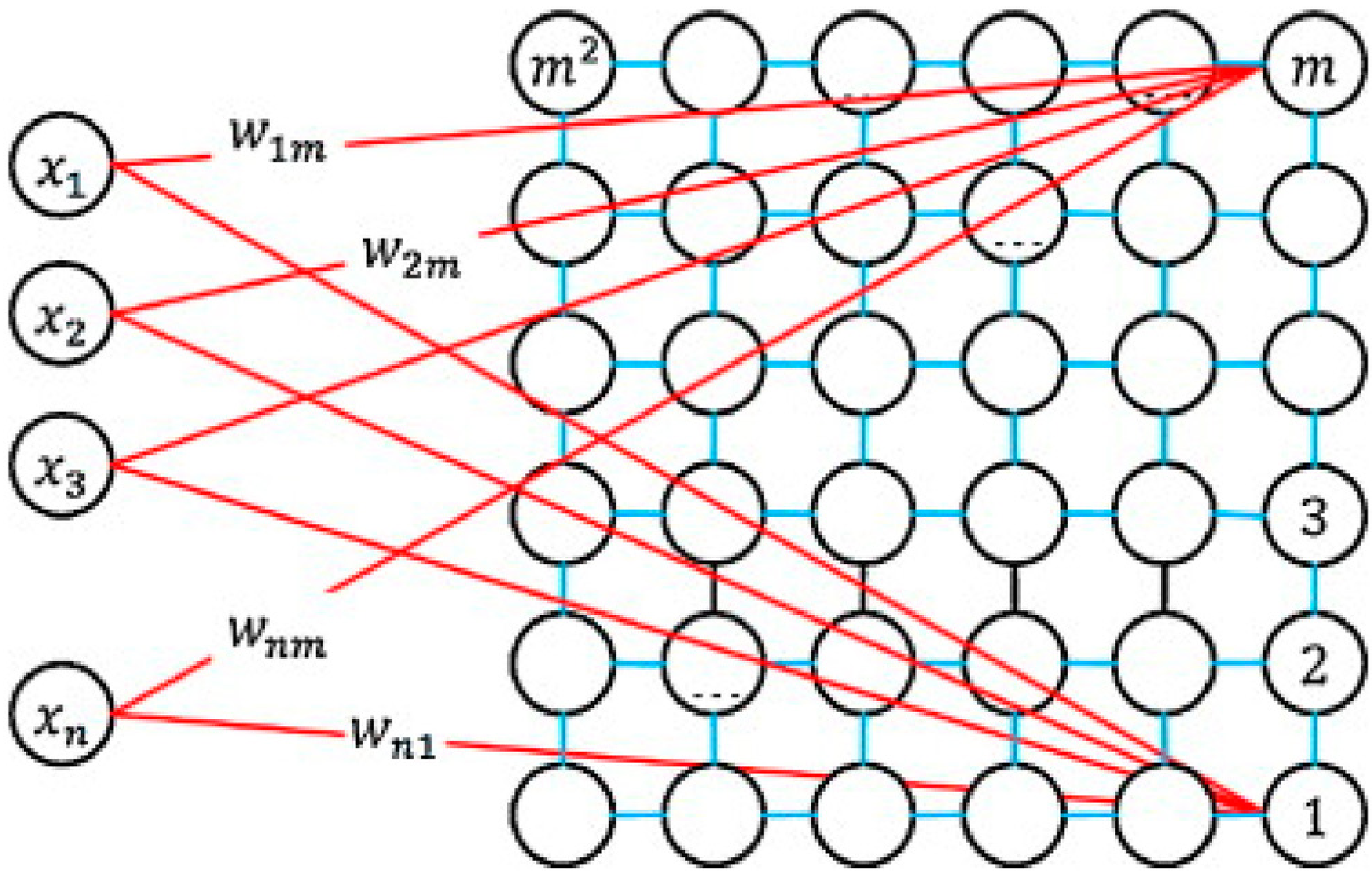
unlabeled data set

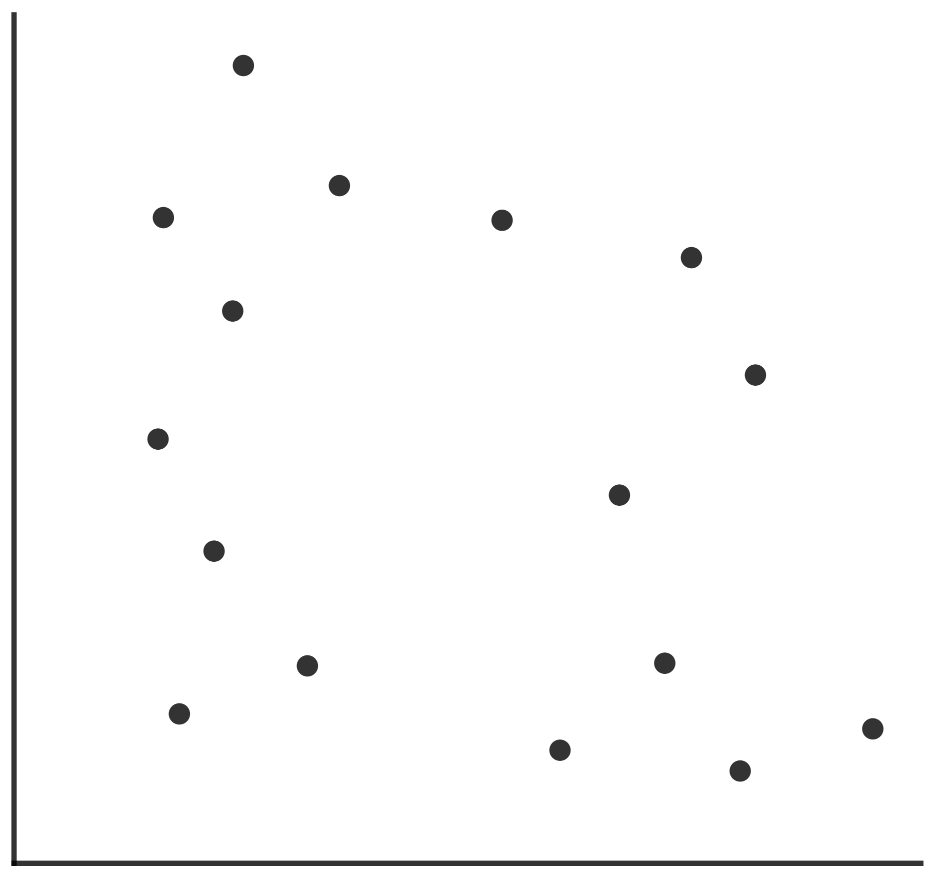


- clustering + topological relation between clusters

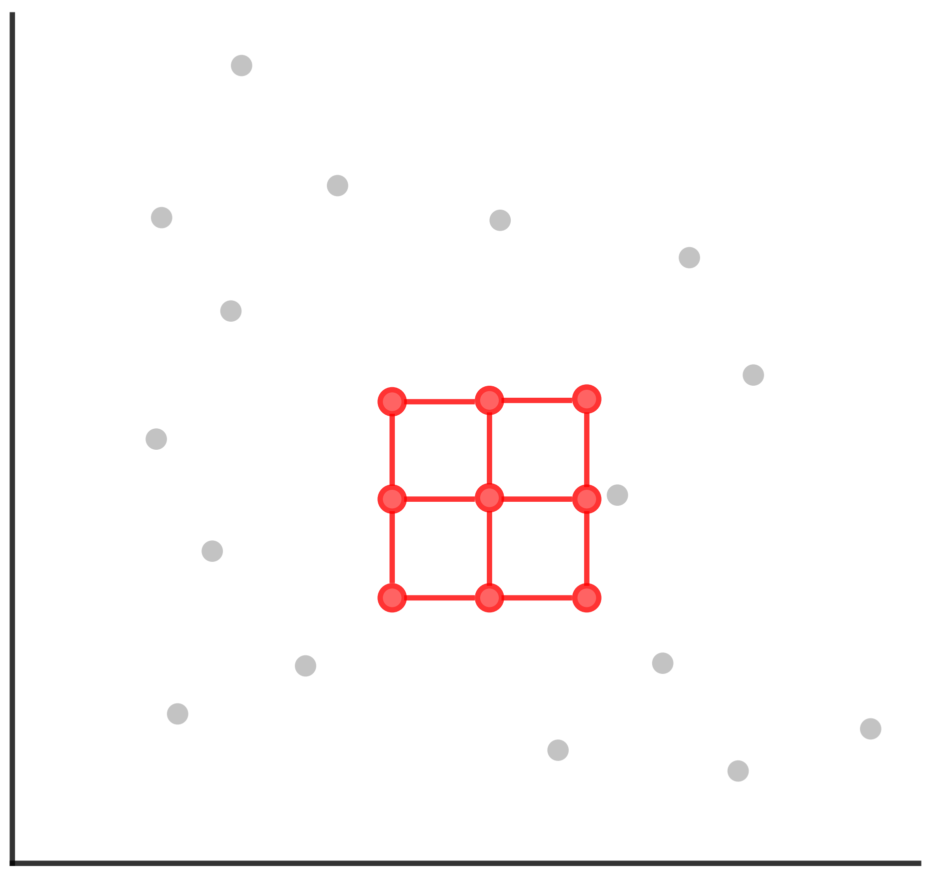
How to do that? Self-Organizing Maps

- fixed grid topology between neurons
- neighborhood connections modified through a neighborhood function (gaussian or mexican hat)
- each input represents a data vector of n dimensions
- each neuron is also reprented in this n-dimensional space

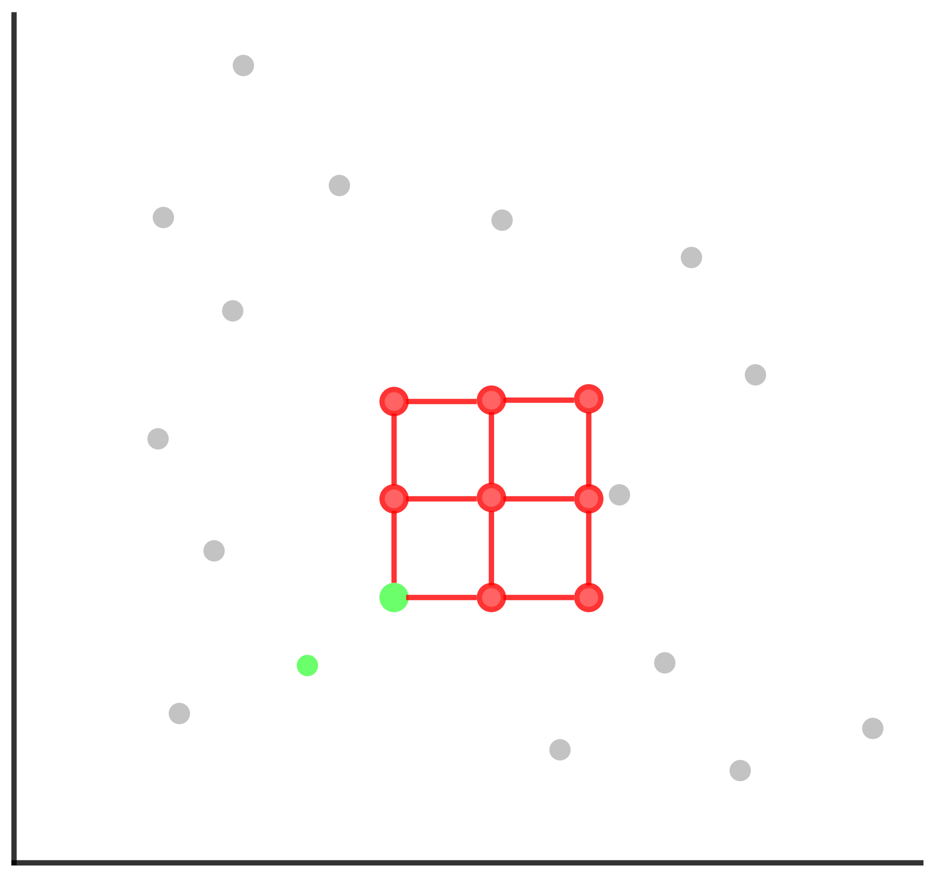




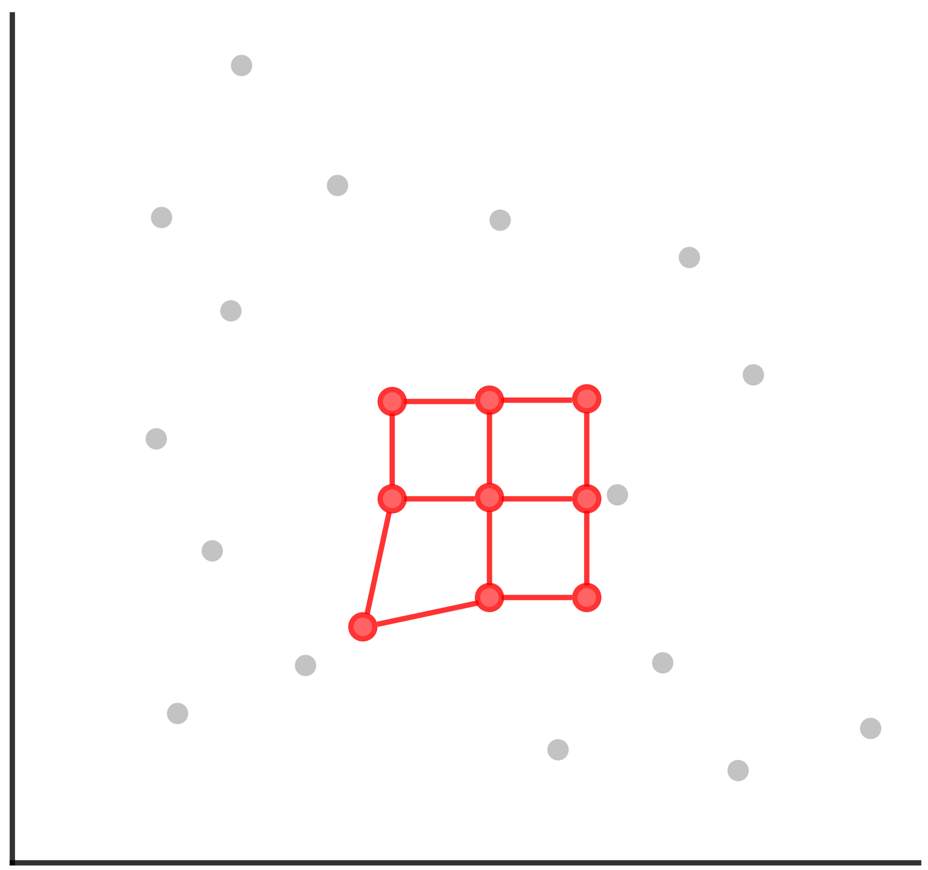
unlabeled data set
(2D)



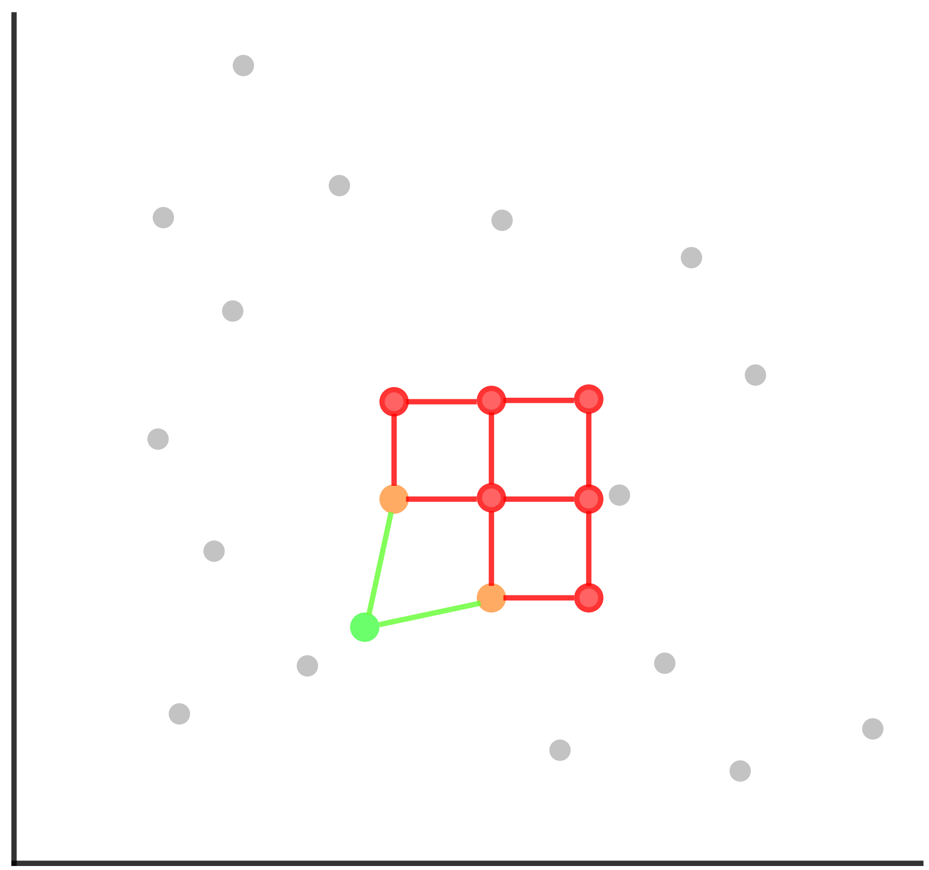
- defined neural network topology, e.g.: 3x3 network of two di-mensions



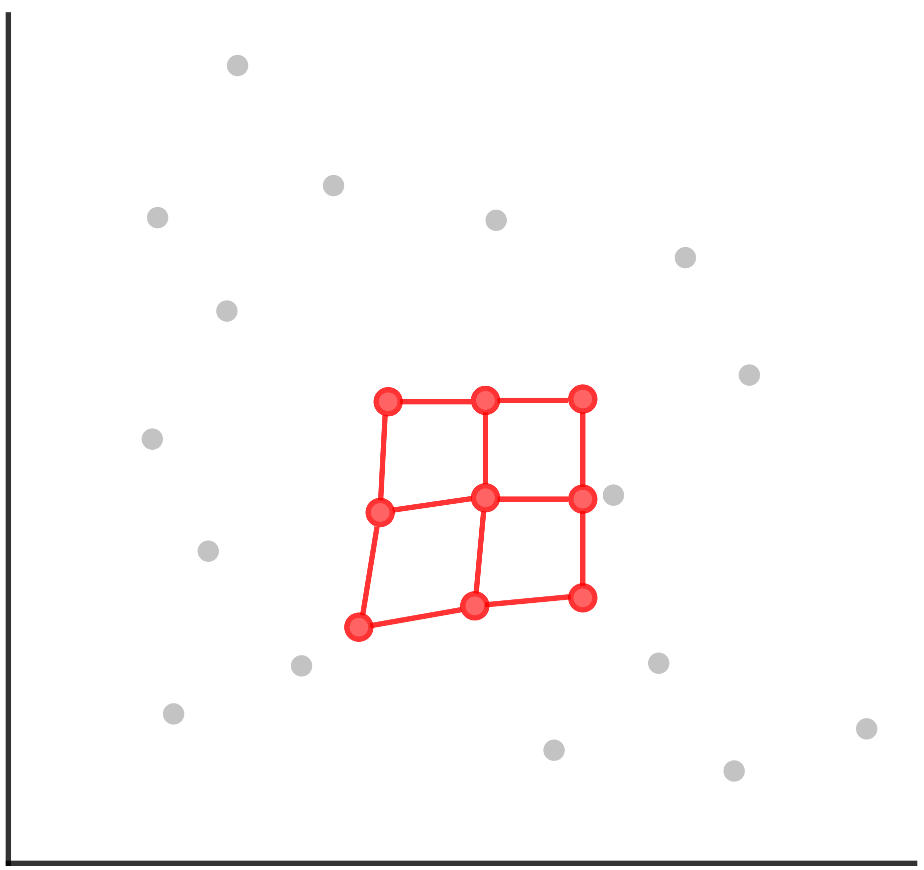
- start comparing data samples to neurons in random order
- find closest fit in set of neurons (winner neuron)



- adjust winner neurons position to respective data sample according to predefined *learning rate*

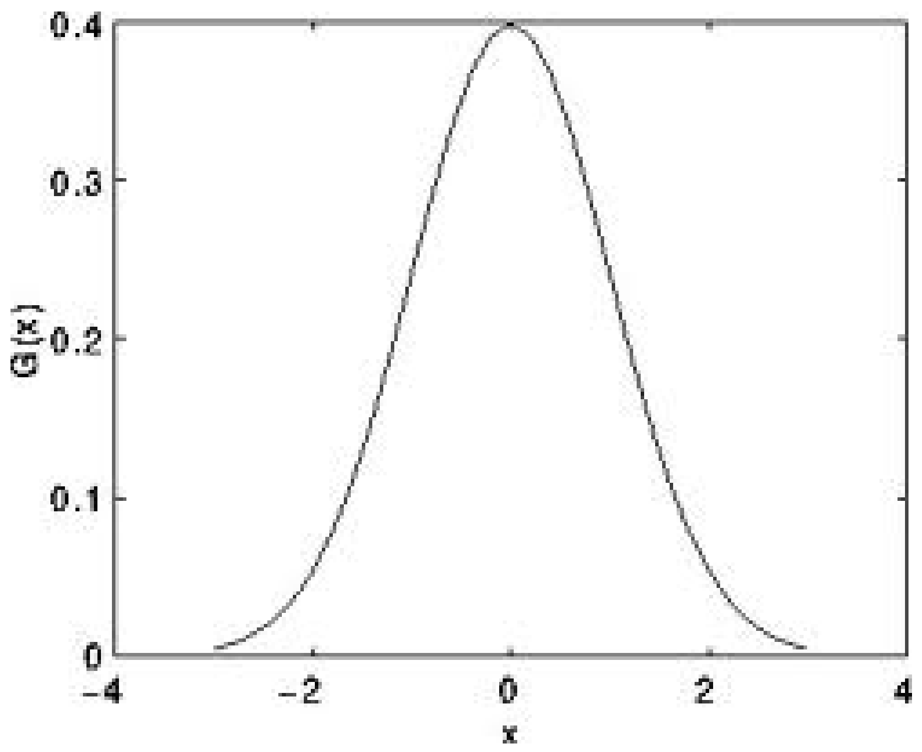


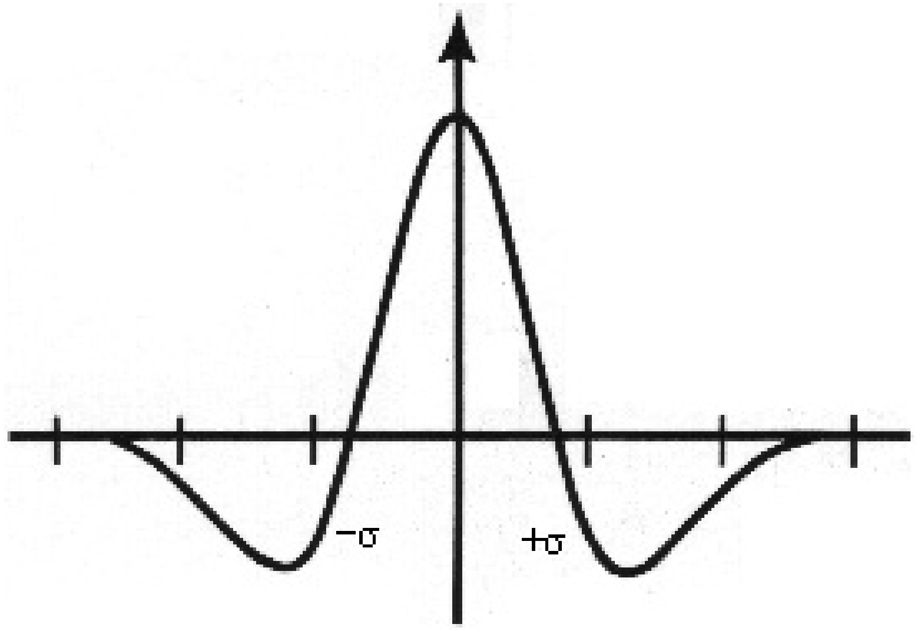
- spread out the position update information to the winner neurons to-polical neighbors



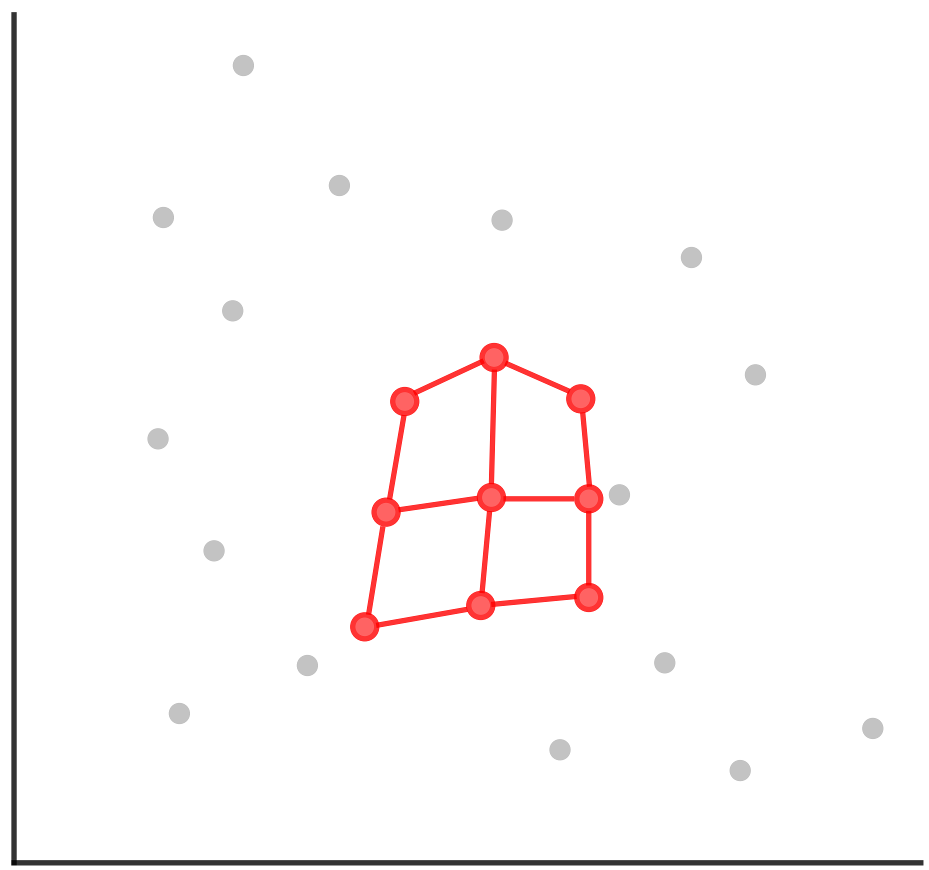
- adjust neighbor neuron positions according to pre-defined neighborhood function

Common Neighborhood Functions

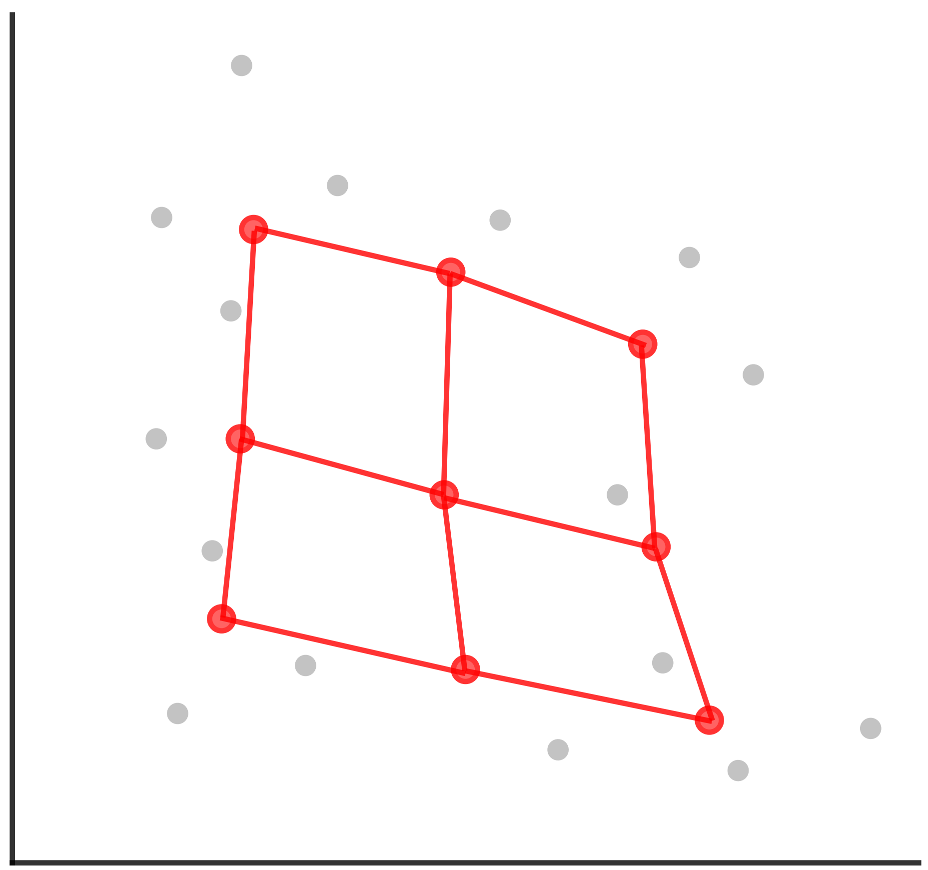




- Gaussian Function
- Mexican Hat Function

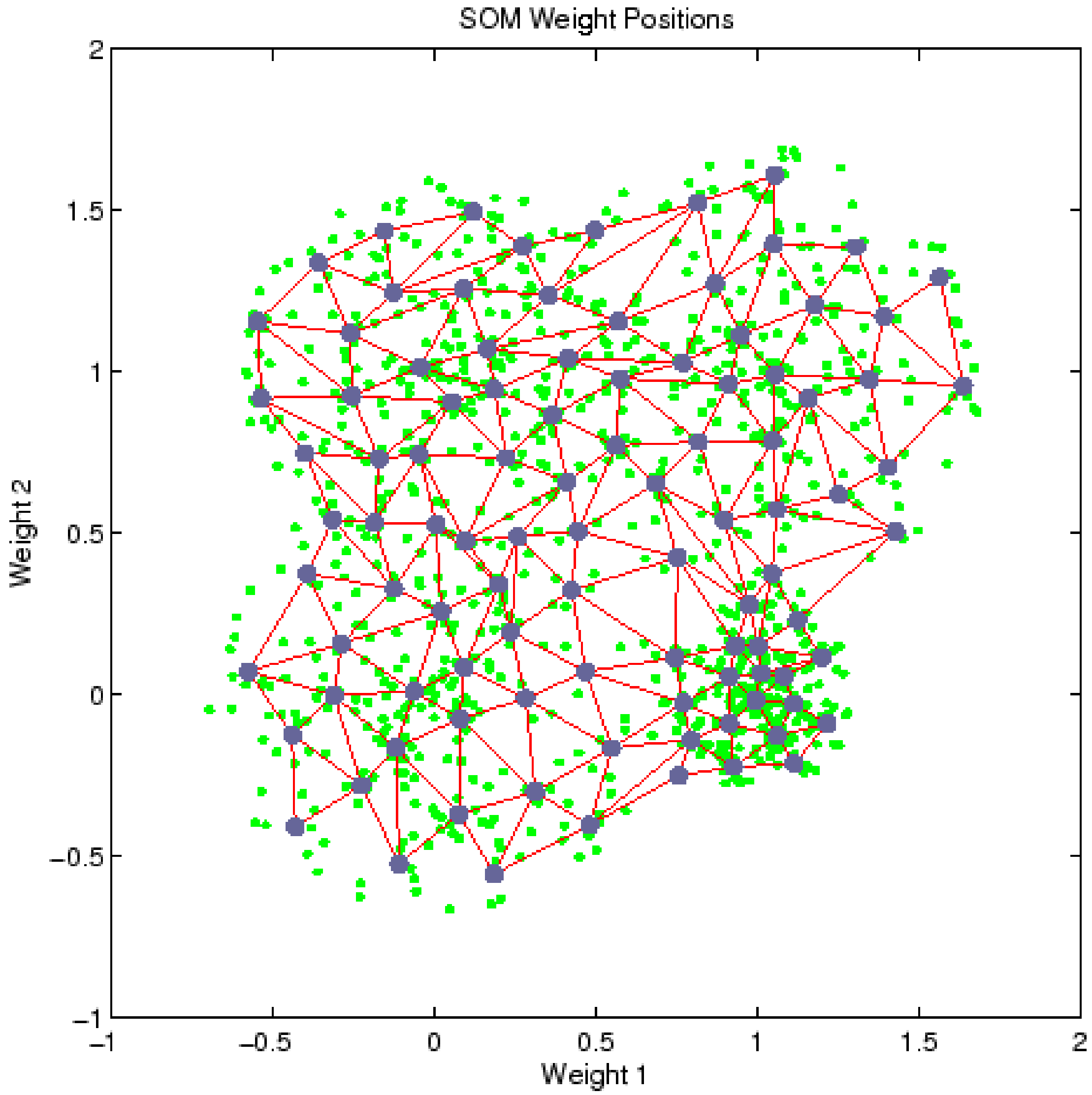


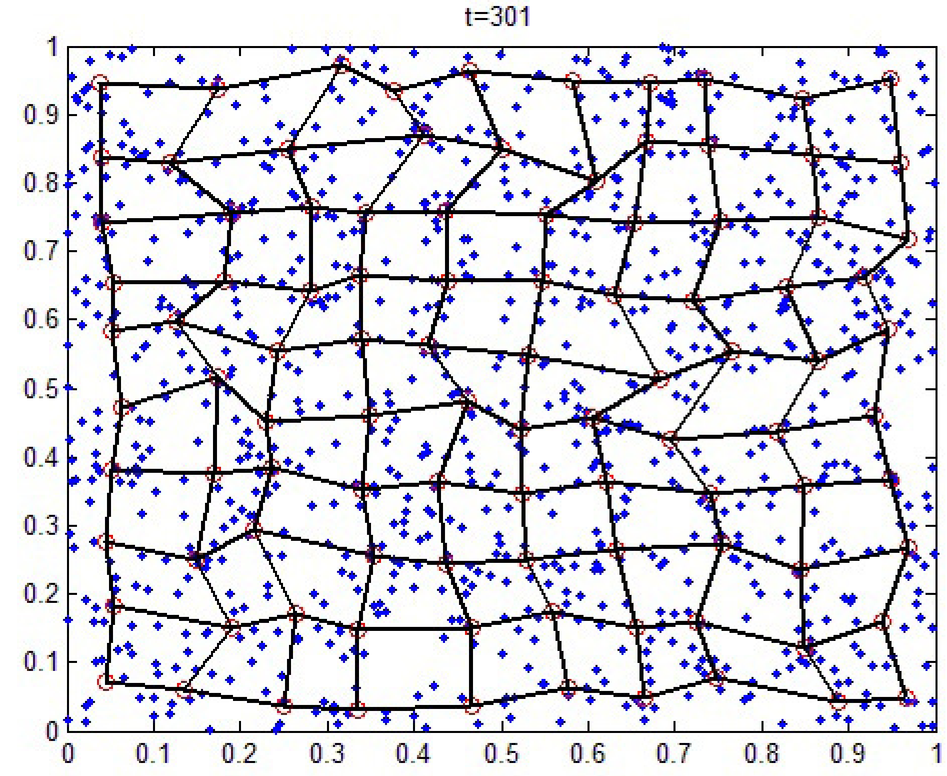
- repeat for different data sample
- repeat repeat repeat ....



- network topology will (hopefully) fit and approxi-mate the data

Examples


SOM Weight Positions
Weight 2
Weight 1
2
1.5
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1
1.5
2


t=301
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Example three:
150523_Crow_Showcase_Kohonen2D

Video … ?

Example three:
150523_Crow_Showcase_KohonenND

...but why?

- structuring large amounts of data
- classification of new data

- Special case:
  high dimensional interpolation
  (if many more neurons than data are supplied)

Exercise: Camera calibration

- objective:
  get rid of the distortion in a camera lense to get a rectified image (important for image processing algorithms)

Exercise: Camera calibration

- objective:
  - get rid of the distortion in a camera lense to get a rectified image (important for image processing algorithms)
  → find the function that maps unrectified pixels into rectified pixels

- steps:
  - generate labelled data (markers of known position)
  - interpolate labelled data to get more data using SOM
  - label new data
  - approximate function that maps camera pixel to rectified pixel