Computational Intelligence & Machine Learning

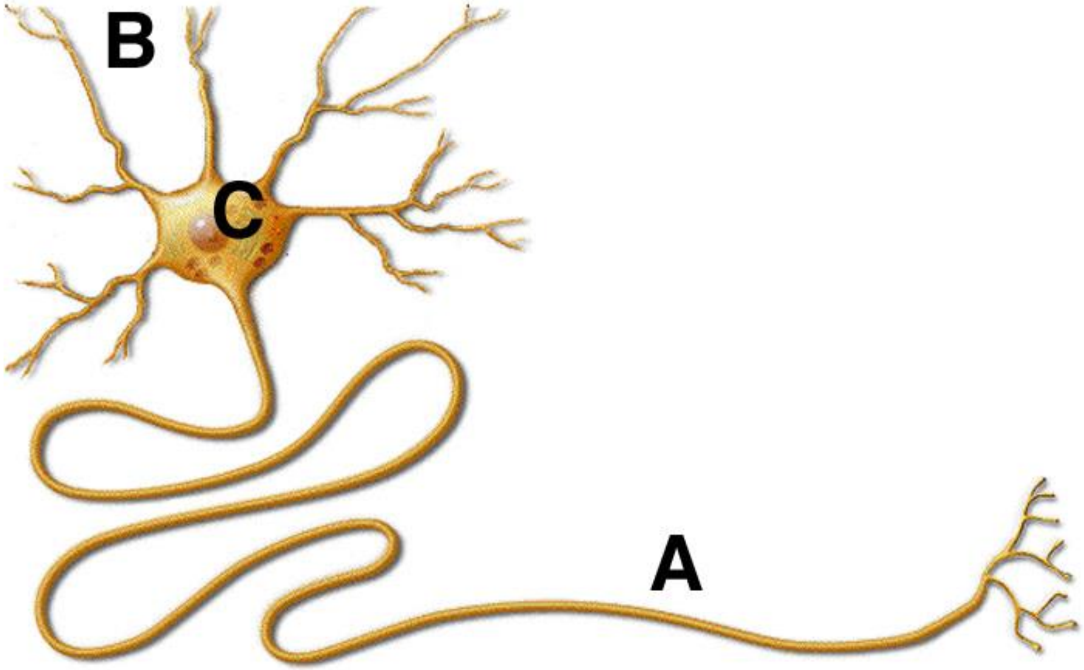
Artificial Neural Networks

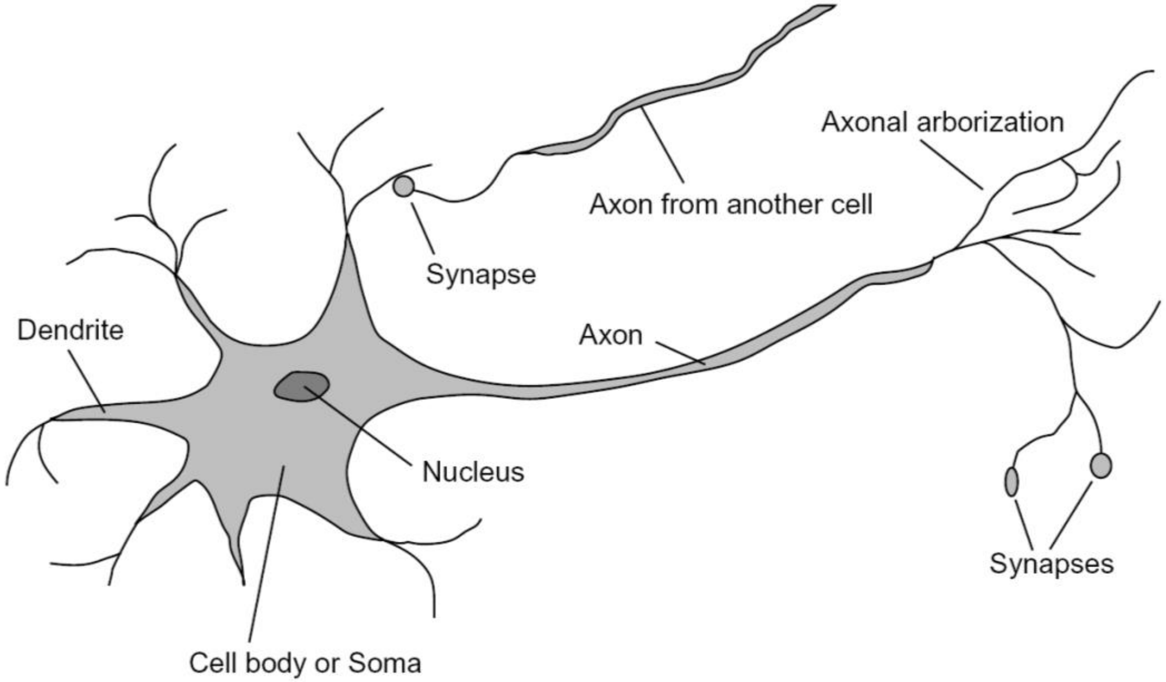
Adopted from Garrison Cotrell - Darwish Ashraf - Nielsen Morten – ‘Data Mining: Concepts & Techniques’

Introduction

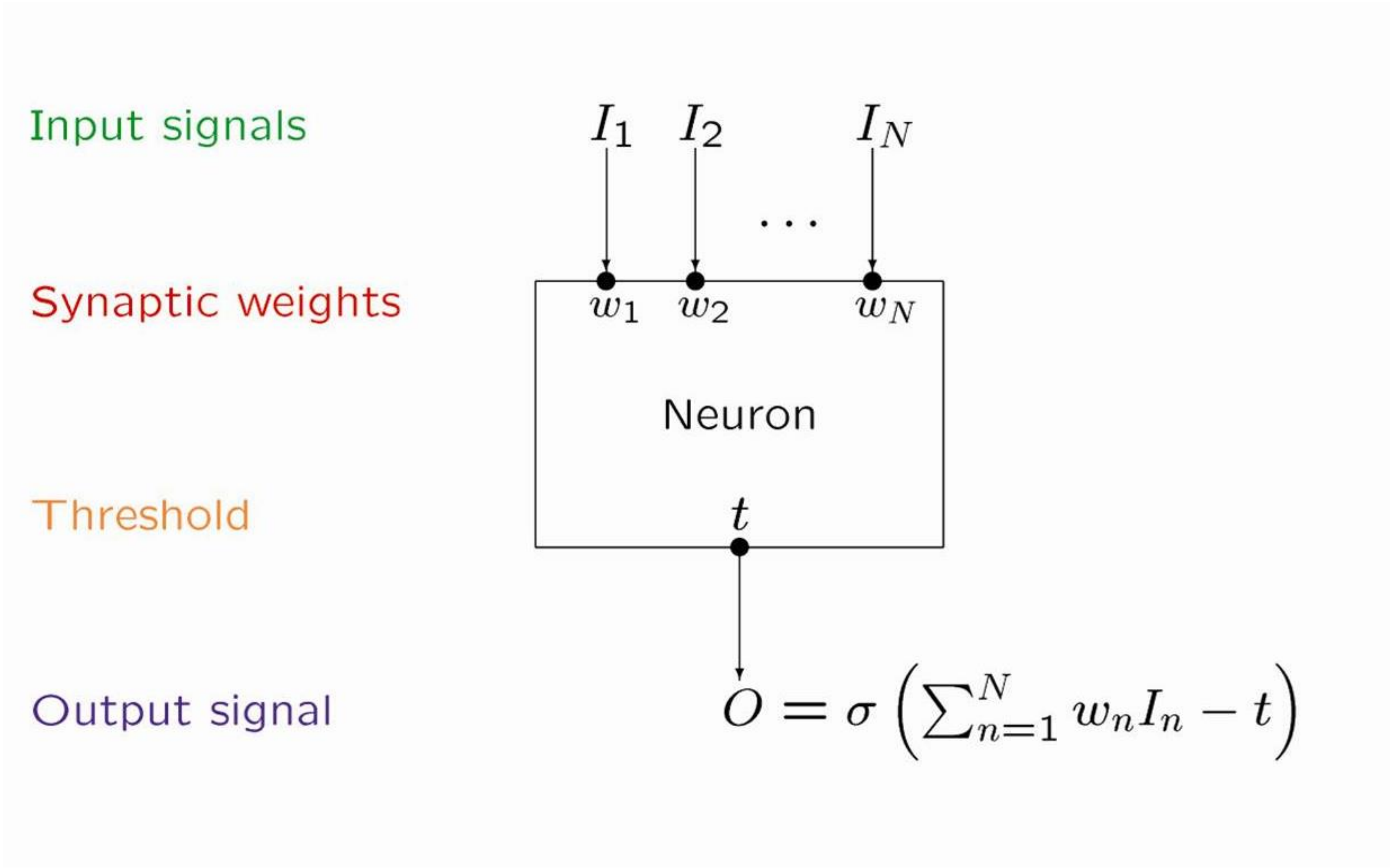
- We have billions and billions of neurons that somehow work together to create the mind.
- These neurons are connected by 10^{14} - 10^{15} synapses, which we think encode the “knowledge” in the network - too many for us to explicitly program them in our models
- Rather we need some way to ***indirectly*** set them via a procedure that will achieve some goal by changing the synaptic strengths (which we call weights).
- This is called ***learning*** in these systems.

Introduction



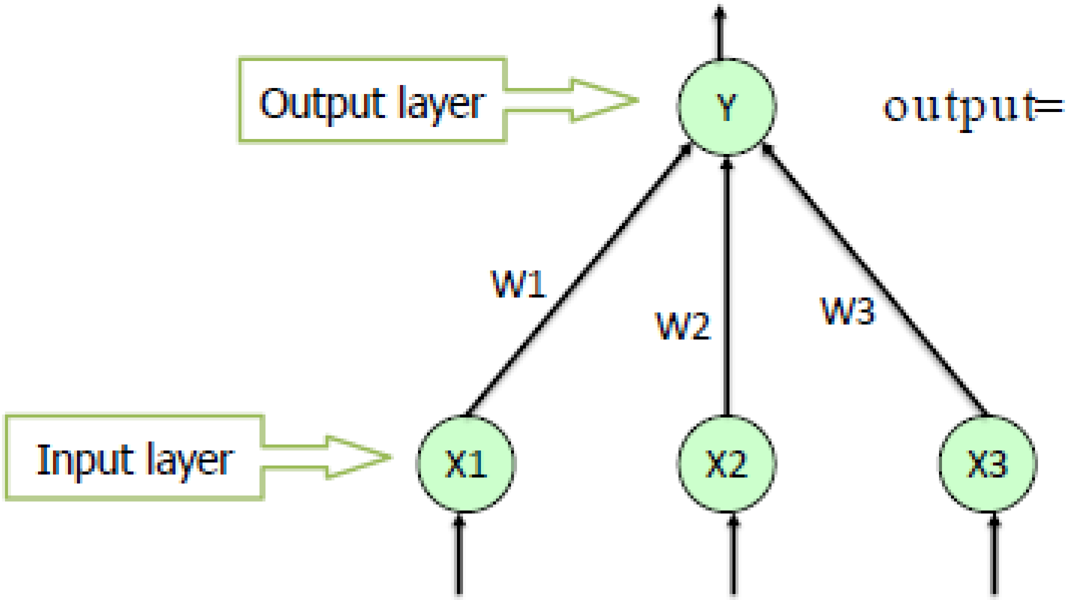


Introduction



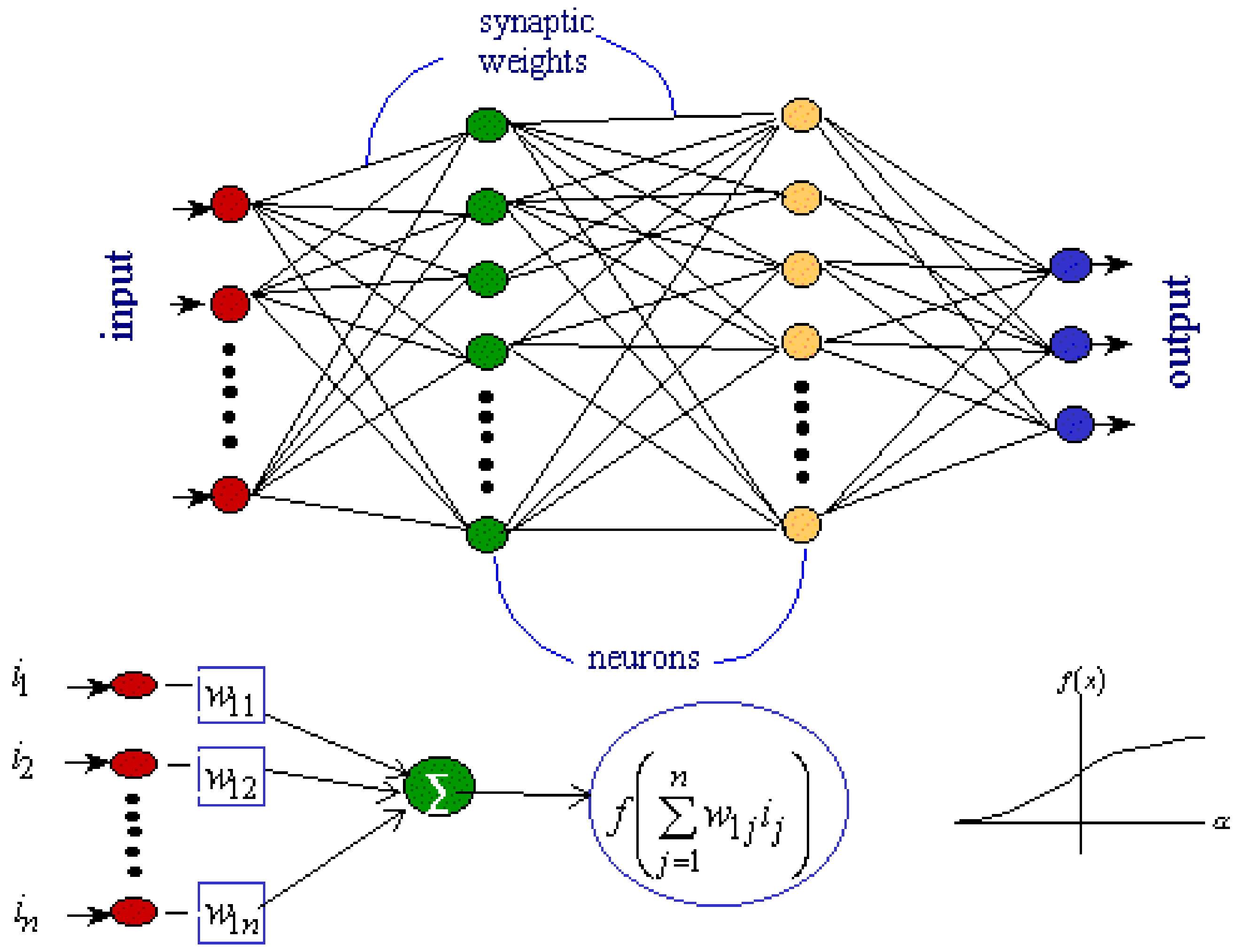
Perceptron

Single Layer Perceptron



$$\text{output} = \begin{cases} 1 & \text{if } \sum w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

Multilayer Perceptrons

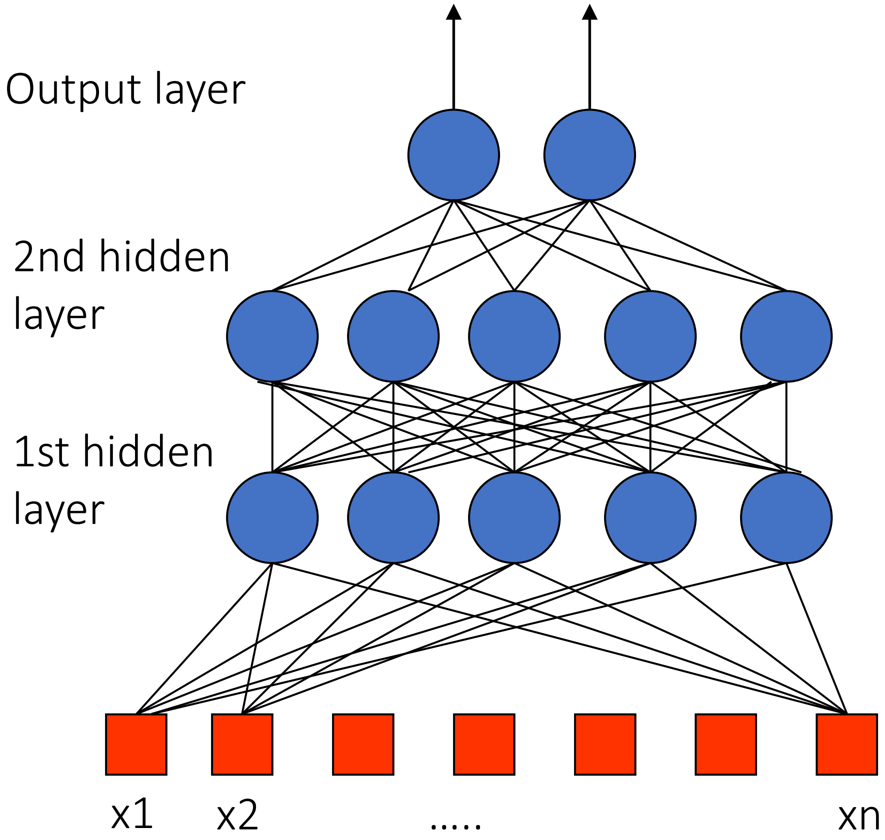


Calculations

- A mathematical model to solve engineering problems
  - Group of highly connected neurons to realize compositions of non linear functions
- Tasks
  - Classification
  - Discrimination
  - Estimation
- 2 types of networks
  - Feed forward Neural Networks
  - Recurrent Neural Networks

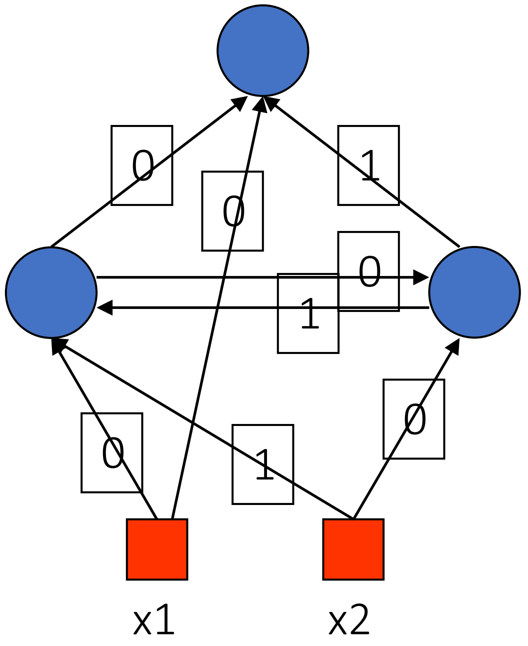
Feed Forward Neural Networks

- The information is propagated from the inputs to the outputs
- Computations of No non linear functions from n input variables by compositions of Nc algebraic functions
- Time has no role (NO cycle between outputs and inputs)



Recurrent Neural Networks

- Can have arbitrary topologies
- Can model systems with internal states (dynamic ones)
- Delays are associated to a specific weight
- Training is more difficult
- Performance may be problematic
  - Stable Outputs may be more difficult to evaluate
  - Unexpected behavior (oscillation, chaos, …)



Learning

- The procedure that consists in estimating the parameters of neurons so that the whole network can perform a specific task

- 2 types of learning
  - **The supervised learning**
  - **The unsupervised learning**

- The Learning process (supervised)
  - Present the network a number of inputs and their corresponding outputs
  - See how closely the actual outputs match the desired ones
  - Modify the parameters to better approximate the desired outputs

Supervised Learning

- The desired response of the neural network in function of particular inputs is well known.
- A “Professor” may provide examples and teach the neural network how to fulfill a certain task

Unsupervised Learning

- Idea : group typical input data in function of resemblance criteria un-known a priori
- Data clustering
- No need of a professor
  - The network finds itself the correlations between the data
  - Examples of such networks :
    - Kohonen feature maps

Learning

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

Pros & Cons

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs *and outputs*
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks

Architectures

- Perceptron
- Multi-Layer Perceptron
- Radial Basis Function (RBF)
- Kohonen Features maps
- Other architectures
  - An example : Shared weights neural networks

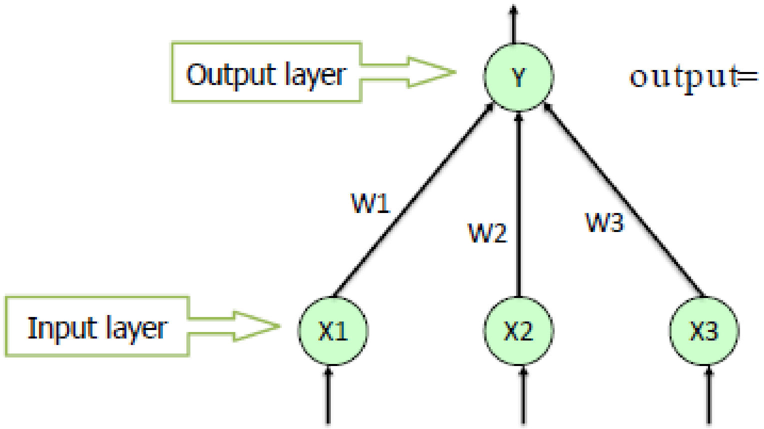
Perceptron

- Rosenblatt (1962) discovered a learning rule for perceptrons called the ***perceptron convergence procedure***.

- Guaranteed to learn anything computable (by a two-layer perceptron)

- Unfortunately, not everything was computable (Minsky & Papert, 1969)

Single Layer Perceptron



$$\text{output} = \begin{cases} 1 & \text{if } \sum w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

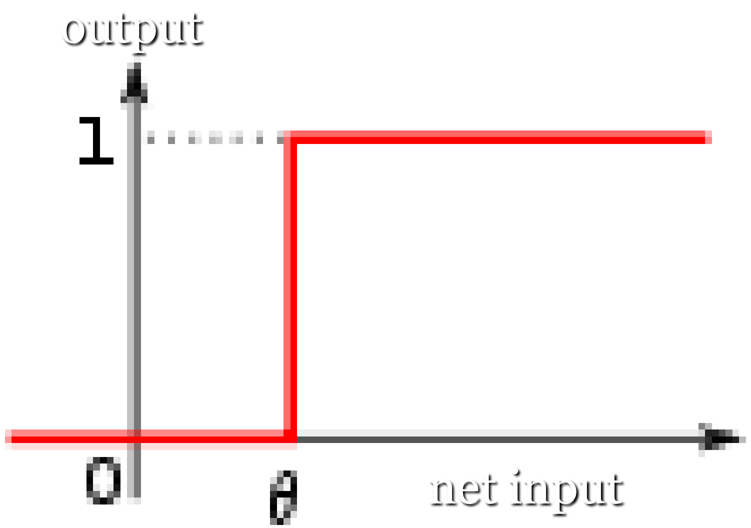
Perceptron

- Output activation rule:
  - First, compute the *net input* to the output unit:

    $\Sigma_i \, w_i x_i = net$
  - Then, compute the output as:

    If *net* $\geq \theta$ then output = 1

    else output = 0



Perceptron

- Perceptrons only be 100% accurate only on linearly separable problems.

- **Multi-layer networks (often called *multi-layer perceptrons*, or *MLPs*) can represent any target function**.

- However, in multi-layer networks, there is no guarantee of convergence to minimal error weight vector.

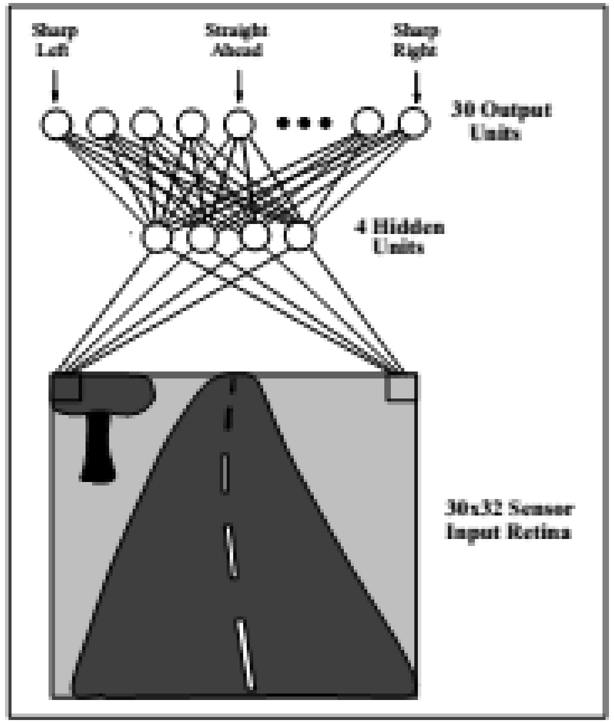
Perceptron

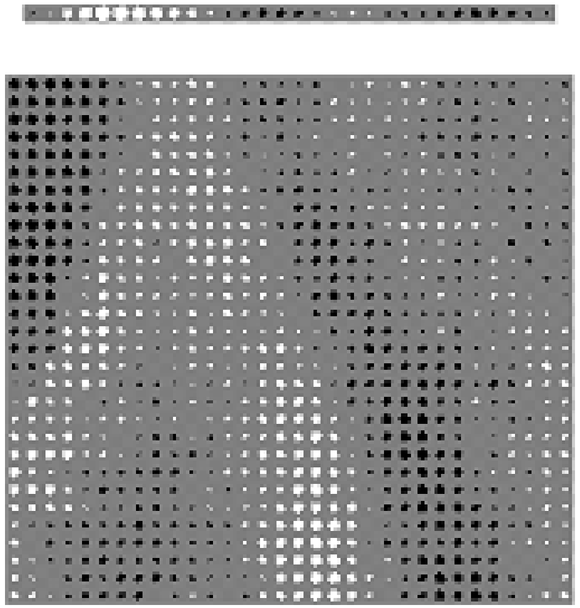
- Learning rule:
  - If output is 1 and should be 0, then *lower* weights to active inputs and *raise* the threshold θ
  - If output is 0 and should be 1, then *raise* weights to active inputs and *lower* the threshold θ

(“active input” means $x_i = 1$, not 0)

Perceptron

- Each output unit correspond to a particular steering direction.
- The most highly activated one gives the direction to steer.





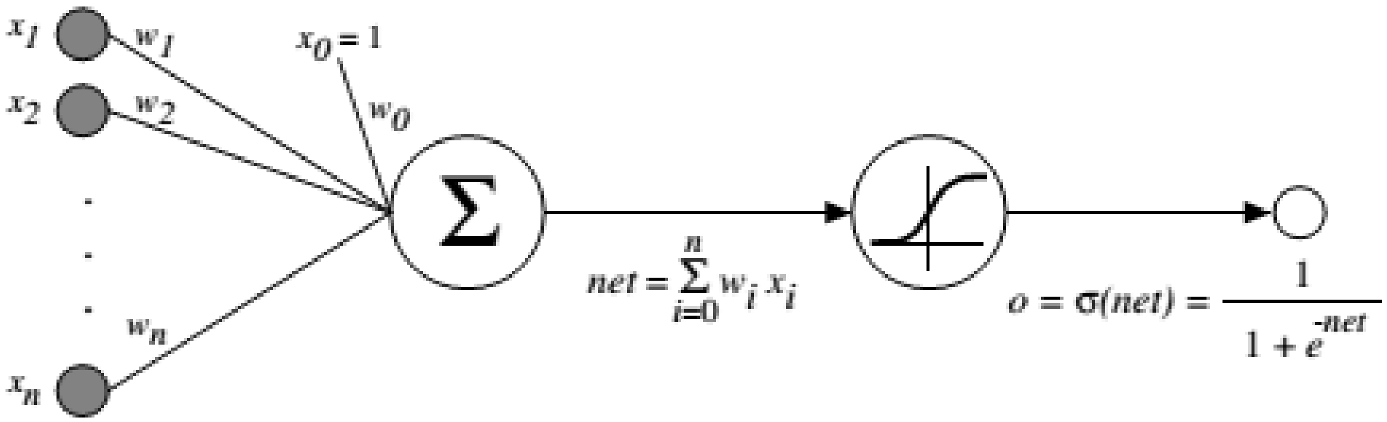
(Note: bias units and weights not shown)

Perceptron

- Before 2009: ANNs typically with 2-3 layers
  - Reason 1: computation times
  - Reason 2: problems of the backpropagation algorithm
    - Local optimization only (needs a good initialization, or re-initialization)
    - Prone to over-fitting (too many parameters to estimate, too few labeled examples)
  - => Skepticism: A deep network often performed worse than a shallow one
- After 2009: Deep neural networks
  - Fast GPU-based implementations
  - Weights can be initialized better (Use of unlabeled data, Restricted Boltzmann Machines)
  - Large collections of labeled data available
  - Reducing the number of parameters by weight sharing
  - Improved backpropagation algorithm
  - Success in different areas, e.g. traffic sign recognition, handwritten digits problem

Perceptron

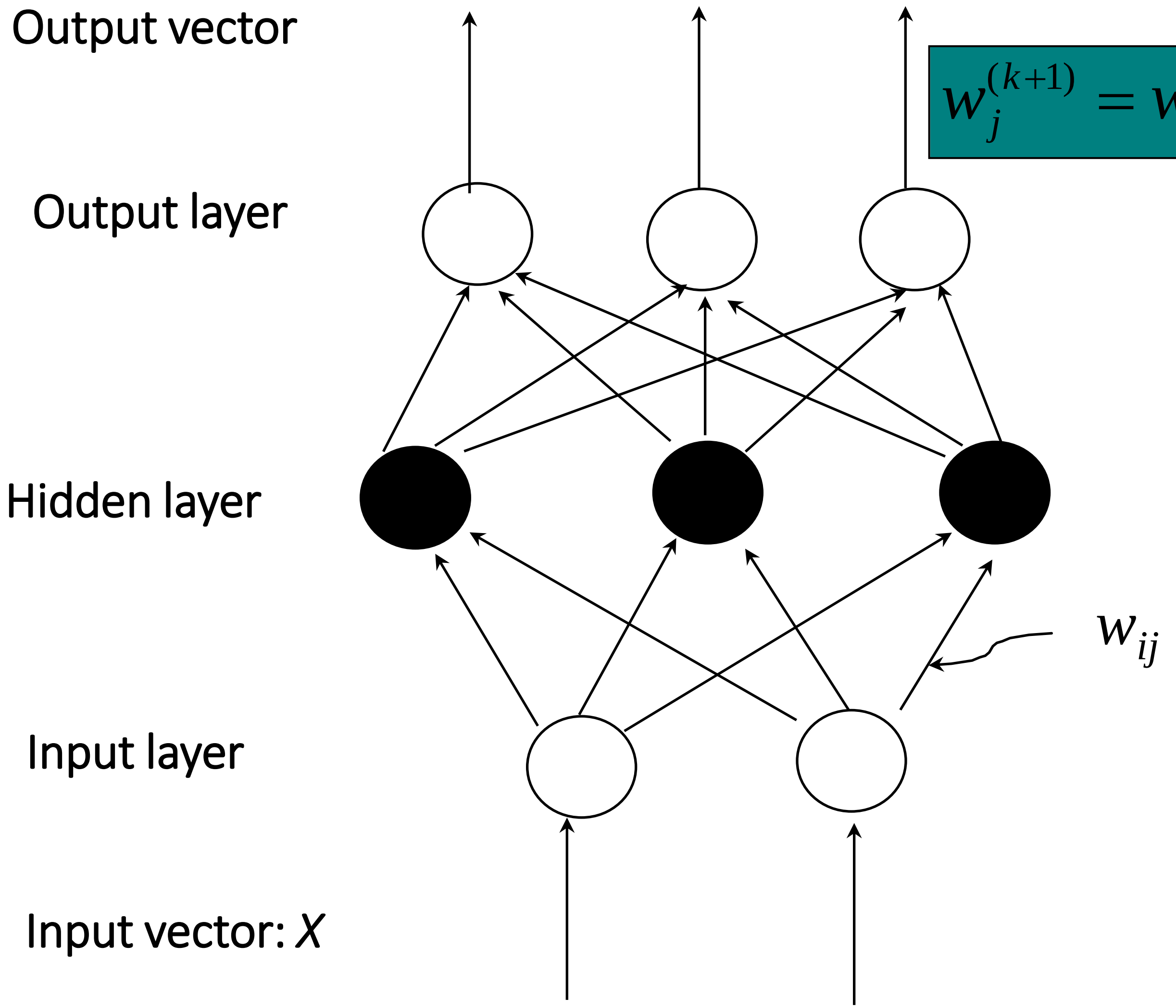
- Network node in detail



$\sigma(x)$ is the sigmoid function

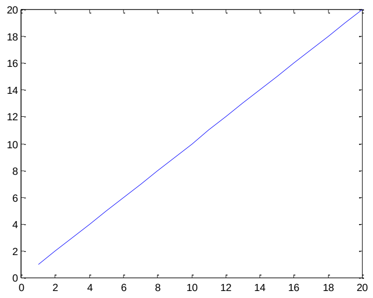
- Network learning process = tuning the synaptic weights
  - Initialize randomly
  - Repeatedly compute the ANN result for a given task, compare with ground truth, update ANN weights by *backpropagation* algorithm to improve ANN performance

Perceptron



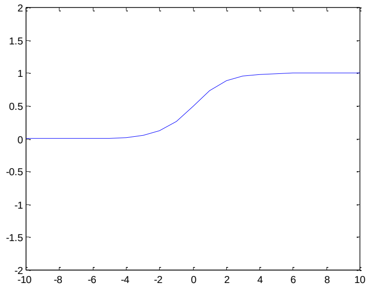
$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$

Perceptron



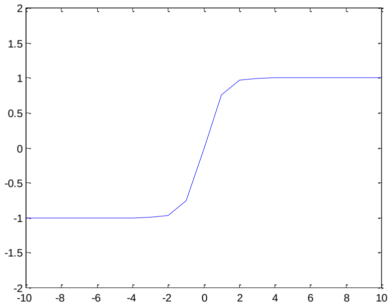
Linear

$$y = x$$



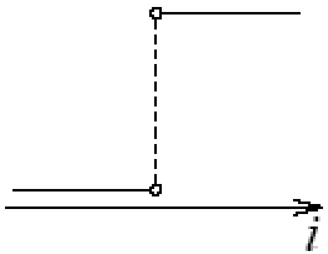
Logistic

$$y = \frac{1}{1 + \exp(-x)}$$

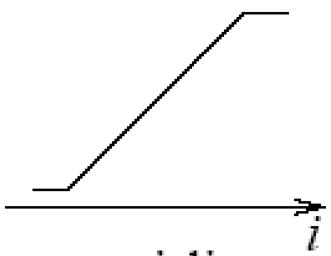


Hyperbolic tangent

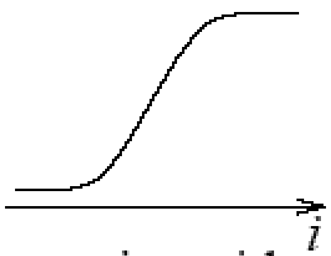
$$y = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



sgn



semi-linear

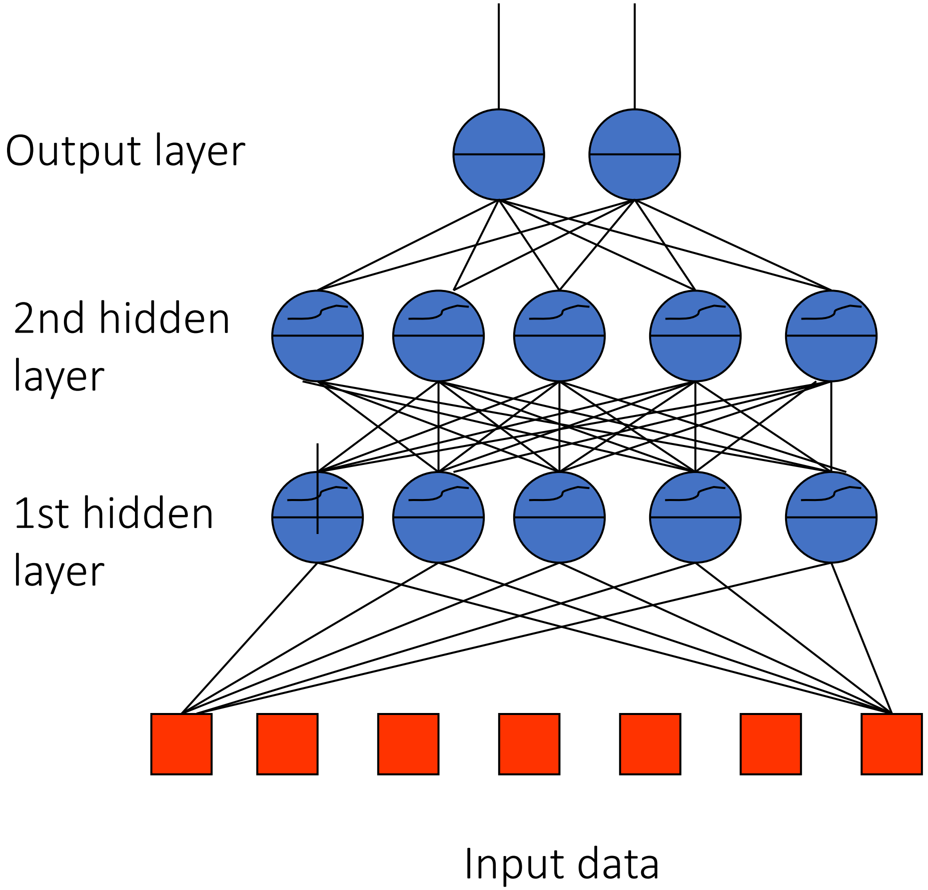


sigmoid

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

Multi-Layer Perceptrons

- One or more hidden layers
- Sigmoid activations functions



Network Topology

- Decide the **network topology:** Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- One **input** unit per domain value, each initialized to 0
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the “**backwards**” direction: from the output layer, through each hidden layer down to the first hidden layer, hence “**backpropagation**”
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Backpropagation

Credit assignment

$$\delta_j = -\frac{\partial E}{\partial net_j}$$

$$net_j = w_{j0} + \sum_i^n w_{ji} o_i$$

$$o_j = f_j(net_j)$$

$$\Delta w_{ji} = -\alpha \frac{\partial E}{\partial w_{ji}} = -\alpha \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \alpha \delta_j o_i$$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\delta_j = -\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} = -\frac{\partial E}{\partial o_j} f'(net_j)$$

$$E = \frac{1}{2}(t_j - o_j)^2 => \frac{\partial E}{\partial o_j} = -(t_j - o_j)$$

$$\delta_j = (t_j - o_j) f'(net_j)$$

If the jth node is an output unit

Backpropagation (differentiable perceptron)

Define total classification error or loss on the training set:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j)\right)^2, \; f_{\mathbf{w}}(\mathbf{x}_j) = \sigma(\mathbf{w} \cdot \mathbf{x}_j), \;\; \sigma(t) = \frac{1}{1+e^{-t}}$$

Update weights by *gradient descent*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{j=1}^{N} \left[-2\left(y_j - f(\mathbf{x}_j)\right)\sigma'(\mathbf{w} \cdot \mathbf{x}_j) \frac{\partial}{\partial \mathbf{w}}(\mathbf{w} \cdot \mathbf{x}_j) \right]$$

$$= \sum_{j=1}^{N} \left[-2\left(y_j - f(\mathbf{x}_j)\right)\sigma(\mathbf{w} \cdot \mathbf{x}_j)(1-\sigma(\mathbf{w} \cdot \mathbf{x}_j))\mathbf{x}_j \right]$$

For a single training point, the update is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left(y - f(\mathbf{x})\right)\sigma(\mathbf{w} \cdot \mathbf{x})(1-\sigma(\mathbf{w} \cdot \mathbf{x}))\mathbf{x}$$

Backpropagation (differentiable perceptron)

- For a single training point, the update is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(y - f(\mathbf{x})\big)\sigma(\mathbf{w}\cdot\mathbf{x})(1-\sigma(\mathbf{w}\cdot\mathbf{x}))\mathbf{x}$$

- Compare with update rule with non-differentiable perceptron:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(y - f(\mathbf{x})\big)\mathbf{x}$$

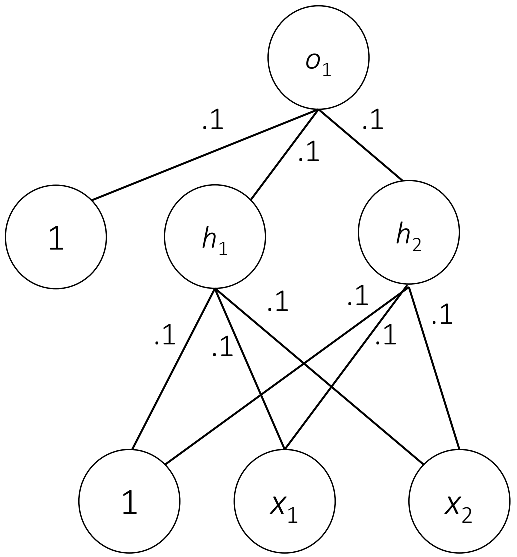
Example

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Example

Training set:

1	0	Label: Positive
0	1	Label: Negative

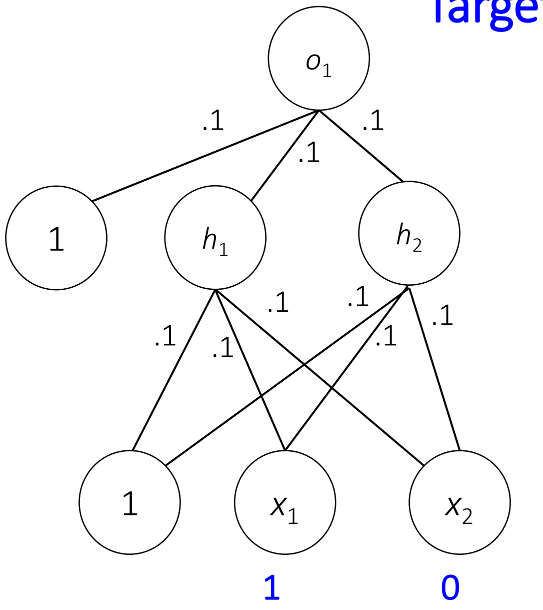


Example

Training set:

1	0	Label: Positive
0	1	Label: Negative

Target: .9



Label: Positive

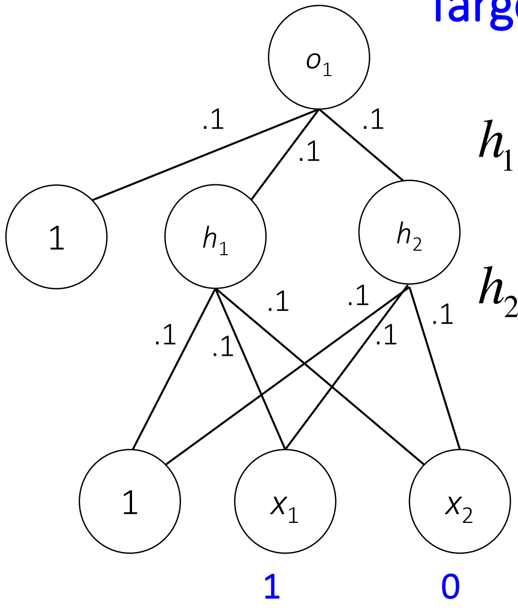
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



$$h_1 = \sigma\big((1)(.1)+(1)(.1)+(0)(.1)\big) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

$$h_2 = \sigma\big((1)(.1)+(1)(.1)+(0)(.1)\big) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

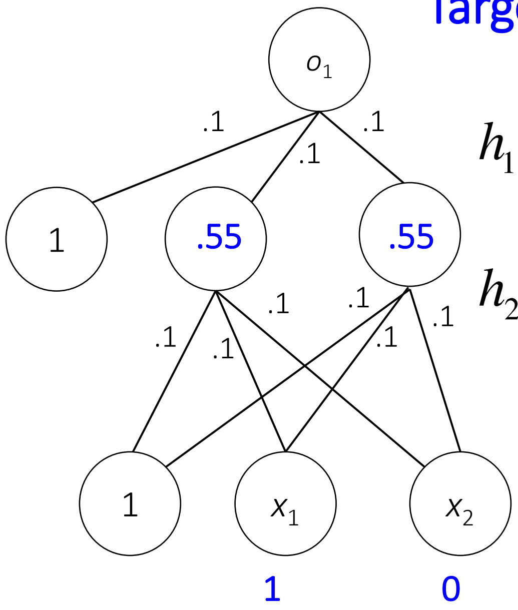
Label: Positive

Example

Training set:

1	0	Label: Positive
0	1	Label: Negative

Target: .9



$$h_1 = \sigma\big((1)(.1) + (1)(.1) + (0)(.1)\big) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

$$h_2 = \sigma\big((1)(.1) + (1)(.1) + (0)(.1)\big) = \sigma(.2) = \frac{1}{1+e^{-.2}} = .55$$

Label: Positive

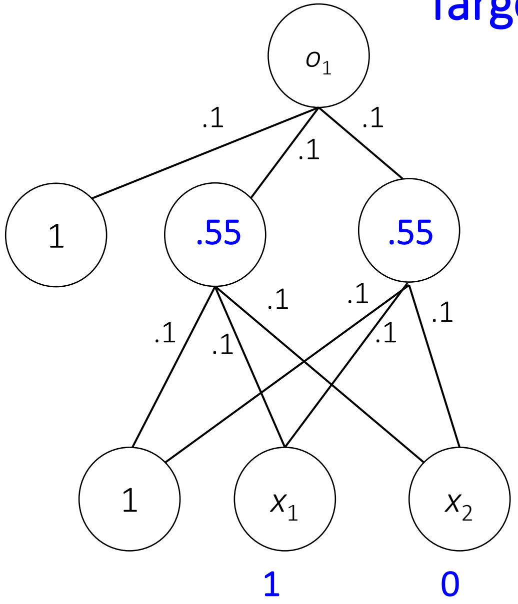
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Label: Positive

Example

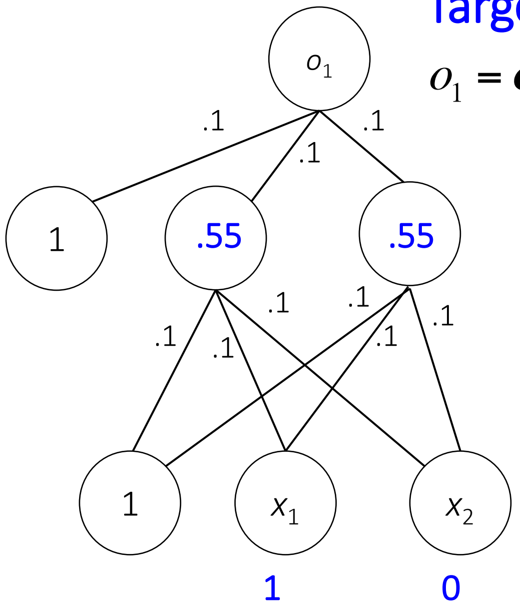
Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9

$$o_1 = \sigma\left((1)(.1) + (.55)(.1) + (.55)(.1)\right) = \sigma(.21) = \frac{1}{1+e^{-.21}} = .552$$



Label: Positive

Example

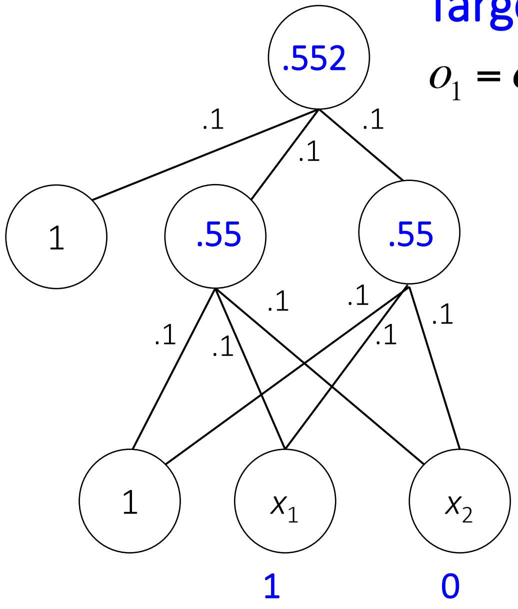
Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9

$$o_1 = \sigma\big((1)(.1) + (.55)(.1) + (.55)(.1)\big) = \sigma(.21) = \frac{1}{1+e^{-.21}} = .552$$



Label: Positive

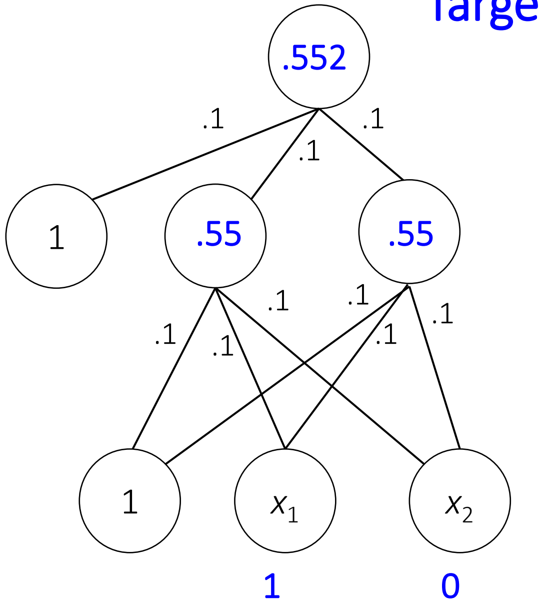
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Label: Positive

Here we interpret $o_1 > .5$ as “positive”.

Classification is correct.

But we still update weights.

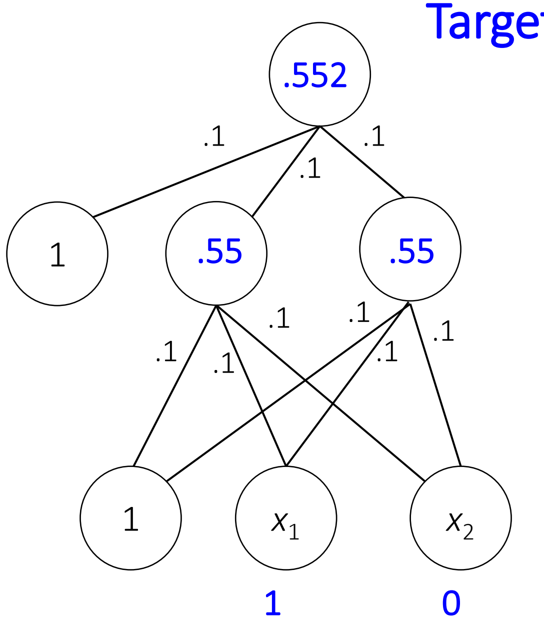
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Label: Positive

Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

$$\delta_{j=2} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

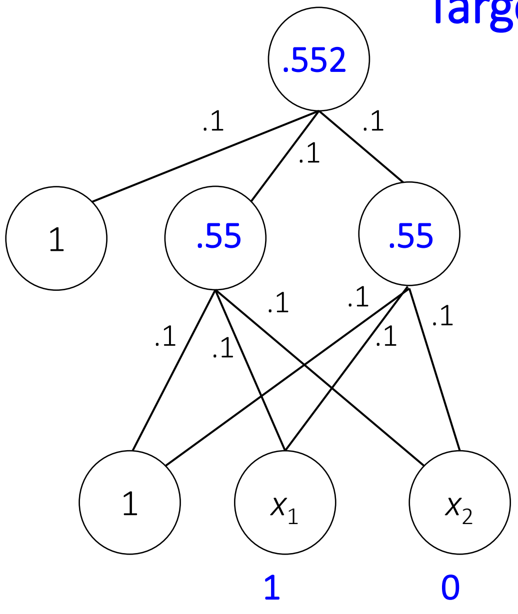
Example

Training set:

1 0 Label: Positive

0 1 Label: Negative

Target: .9



Label: Positive

Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

$$\delta_{j=2} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

Update hidden-to-output weights (*learning rate* = 0.2; *momentum* = 0.9):

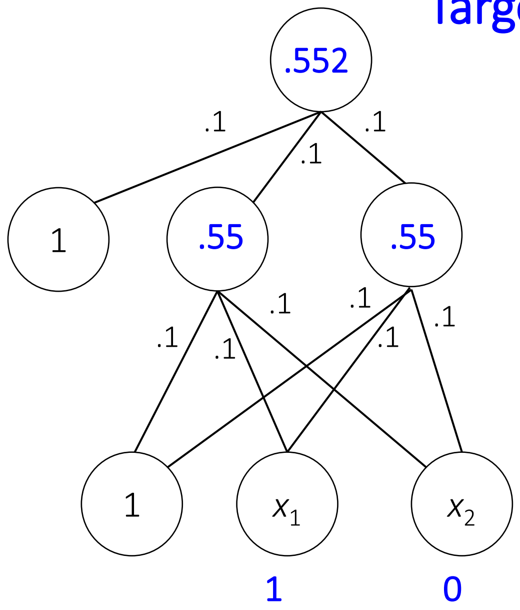
Training set:

1 0 Label: Positive

0 1 Label: Negative

Example

Target: .9



Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

$$\delta_{j=2} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

Label: Positive

Update hidden-to-output weights (*learning rate* = 0.2; *momentum* = 0.9):

$$\Delta w^1_{k=1,j=0} = (.2)\left(.086\right)(1) + (.9)(0) = .0172$$

$$\Delta w^1_{k=1,j=1} = (.2)\left(.086\right)(.55) + (.9)(0) = .0095$$

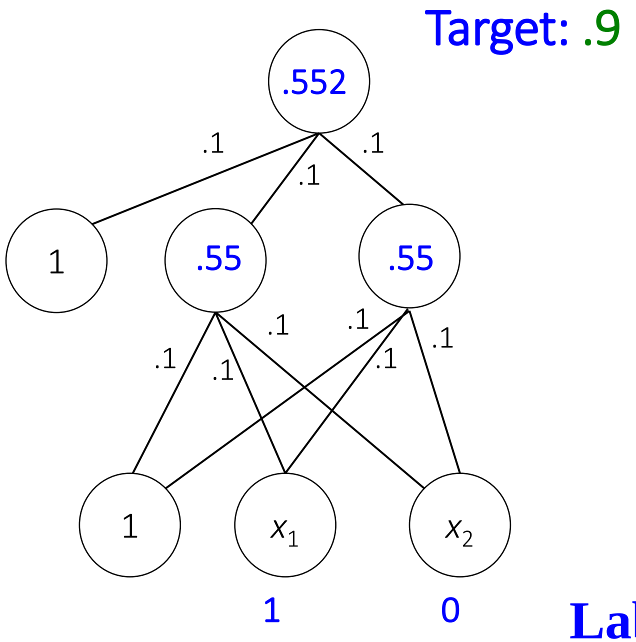
$$\Delta w^1_{k=1,j=2} = (.2)\left(.086\right)(.55) + (.9)(0) = .0095$$

Training set:

1 0 Label: Positive

0 1 Label: Negative

Example



Label: Positive

Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

$$\delta_{j=2} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

Update hidden-to-output weights (*learning rate* = 0.2; *momentum* = 0.9):

$$\Delta w^{1}_{k=1,j=0} = (.2)\left(.086\right)(1) + (.9)(0) = .0172 \qquad w^{1}_{k=1,j=0} = .1 + .0172 = .1172$$

$$\Delta w^{1}_{k=1,j=1} = (.2)\left(.086\right)(.55) + (.9)(0) = .0095 \qquad w^{1}_{k=1,j=1} = .1 + .0095 = .1095$$

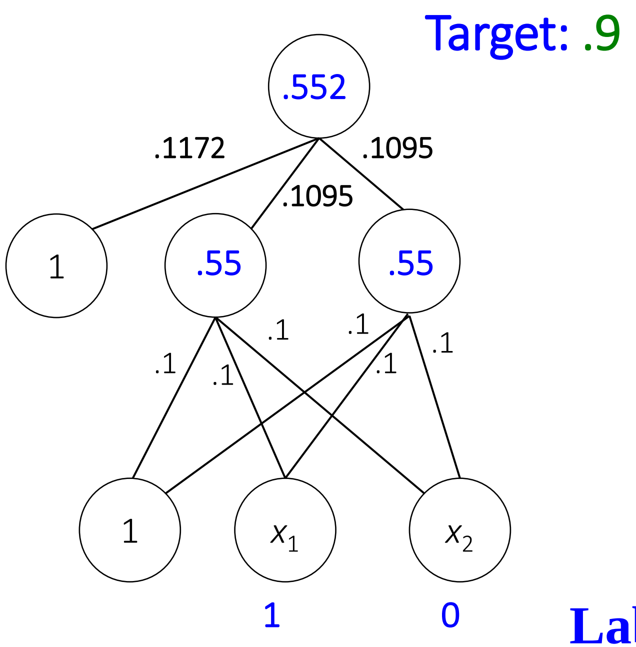
$$\Delta w^{1}_{k=1,j=2} = (.2)\left(.086\right)(.55) + (.9)(0) = .0095 \qquad w^{1}_{k=1,j=2} = .1 + .0095 = .1095$$

Training set:

1 0 Label: Positive

0 1 Label: Negative

Example



Label: Positive

Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = (.55)(.45)(.1)(.086) = .002$$

$$\delta_{j=2} = (.55)(.45)(.1)(.086) = .002$$

$$\Delta w^1_{k=1,j=0} = (.2)(.086)(1) + (.9)(0) = .0172 \qquad w^1_{k=1,j=0} = .1 + .0172 = .1172$$

$$\Delta w^1_{k=1,j=1} = (.2)(.086)(.55) + (.9)(0) = .0095 \qquad w^1_{k=1,j=1} = .1 + .0095 = .1095$$

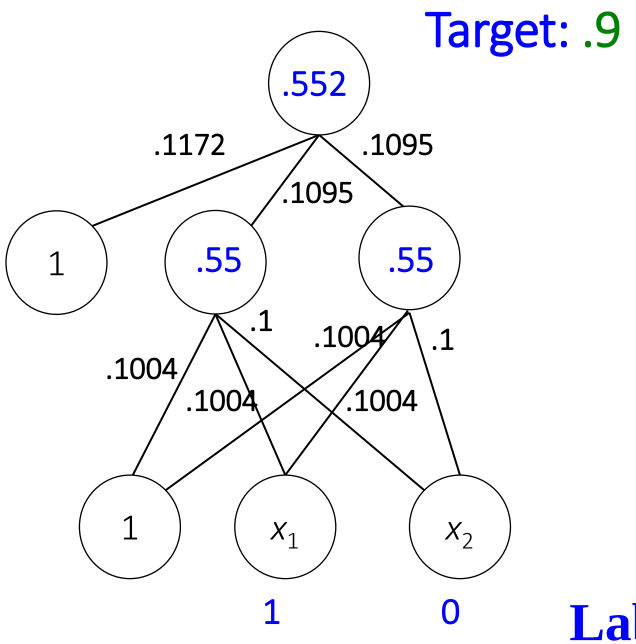
$$\Delta w^1_{k=1,j=2} = (.2)(.086)(.55) + (.9)(0) = .0095 \qquad w^1_{k=1,j=2} = .1 + .0095 = .1095$$

Training set:

1 0 Label: Positive

0 1 Label: Negative

Example



Label: Positive

Calculate error terms:

$$\delta_{k=1} = (.552)(.448)(.9 - .552) = .086$$

$$\delta_{j=1} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

$$\delta_{j=2} = \left(.55\right)(.45)(.1)\left(.086\right) = .002$$

Update input-to-hidden weights (*learning rate* = 0.2; *momentum* = 0.9):

$$\Delta w^1_{j=1,i=0} = (.2)\left(.002\right)(1) + (.9)(0) = .0004 \qquad w^1_{j=1,i=0} = .1 + .0004 = .1004$$

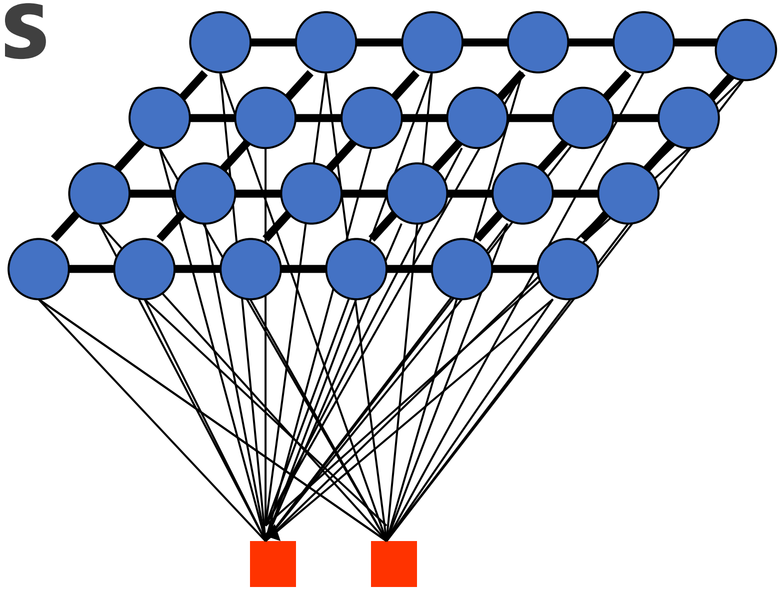
$$\Delta w^1_{j=1,i=1} = (.2)\left(.002\right)(1) + (.9)(0) = .0004 \qquad w^1_{j=1,i=1} = .1 + .0004 = .1004$$

$$\mathrm{D}w^1_{j=1,i=2} = (.2)\left(.002\right)(0) + (.9)(0) = 0 \qquad w^1_{j=1,\,i=2} = .1 \qquad w^1_{j=2,\,i=2} = .1$$

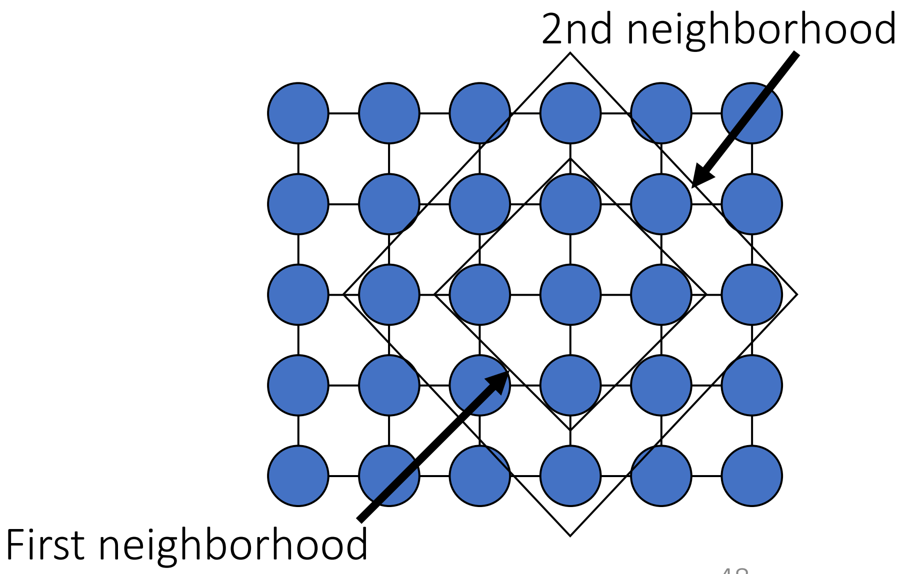
Self Organizing Maps

- The purpose of SOM is to map a multidimensional input space onto a topology preserving map of neurons
  - Preserve a topological so that neighboring neurons respond to « similar »input patterns
  - The topological structure is often a 2 or 3 dimensional space
- Each neuron is assigned a weight vector with the same dimensionality of the input space
- Input patterns are compared to each weight vector and the closest wins (Euclidean Distance)

Self Organizing Maps



- The activation of the neuron is spread in its direct neighborhood =>neighbors become sensitive to the same input patterns
- Block distance
- The size of the neighborhood is initially large but reduce over time => Specialization of the network



Self Organizing Maps

- During training, the “winner” neuron and its neighborhood adapts to make their weight vector more similar to the input pattern that caused the activation
- The neurons are moved closer to the input pattern
- The magnitude of the adaptation is controlled via a learning parameter which decays over time