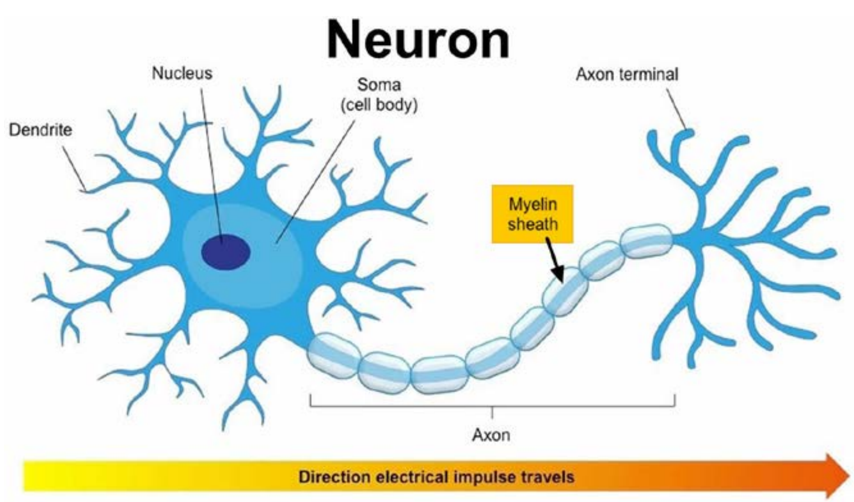
CHAPTER 11:

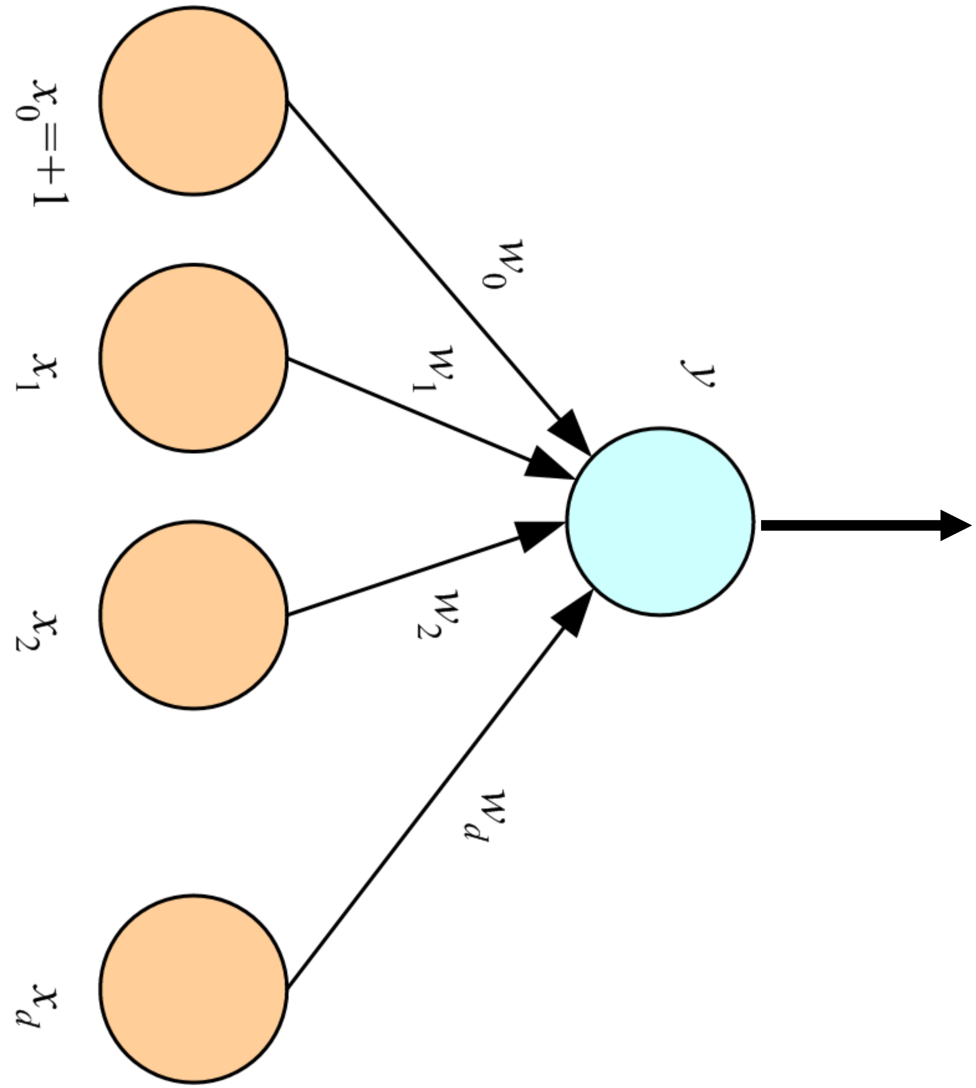
Multilayer Perceptrons

Functioning of brain

- Interconnected Billions of neurons
  - Individual neurons: Simple
  - Perform parallel processing
  - Distributed associative memory

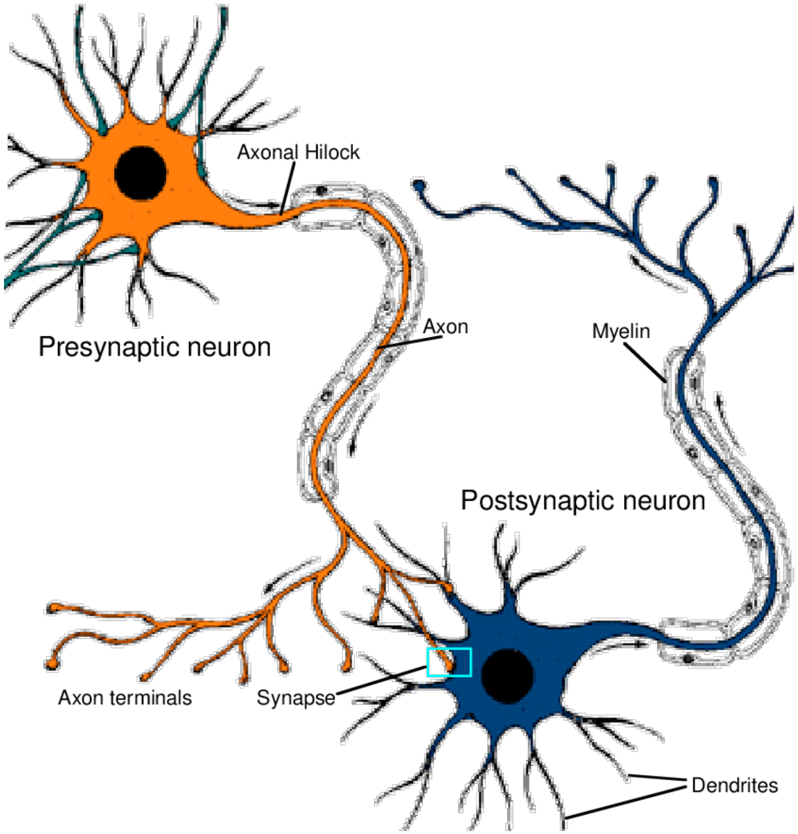
Model of a Neuron





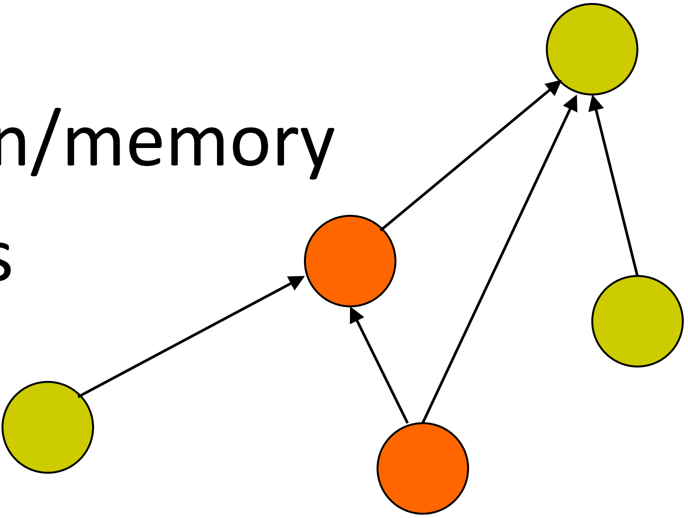
Artificial Neural Network

- Neurons → Nodes
- Synapse → weights



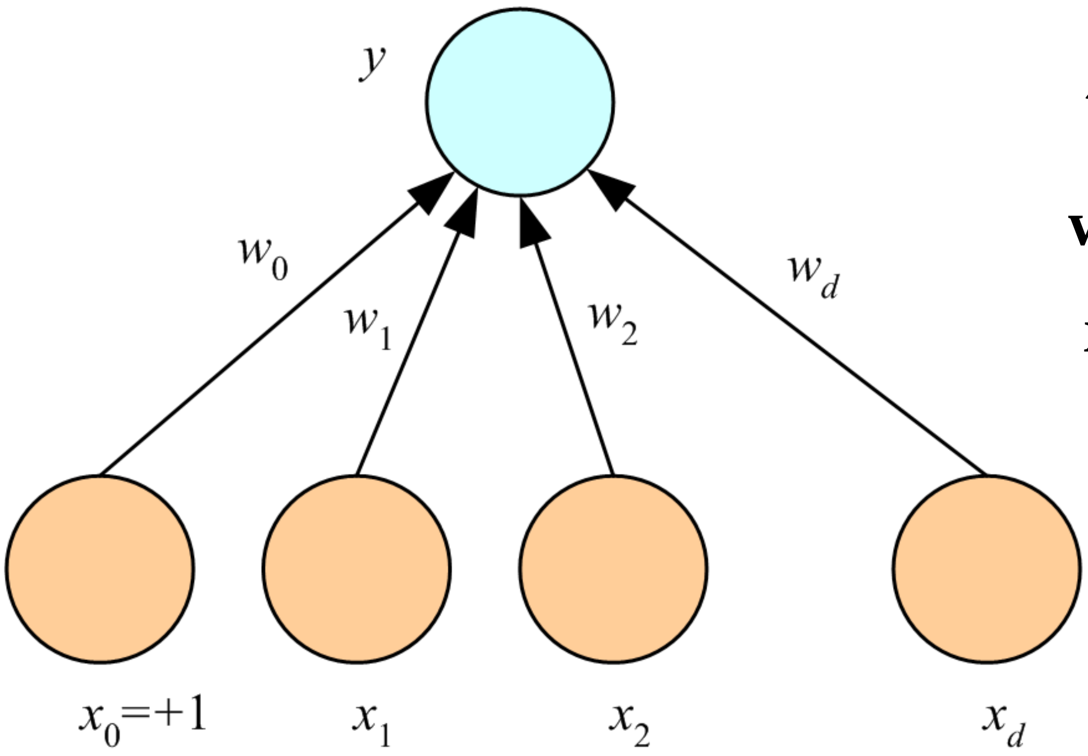
Neural Networks: Features

- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: 10^{10}
- Large connectitivity: 10^5
- Parallel processing
- Distributed computation/memory
- Robust to noise, failures



Perceptron

- Basic unit in neural network



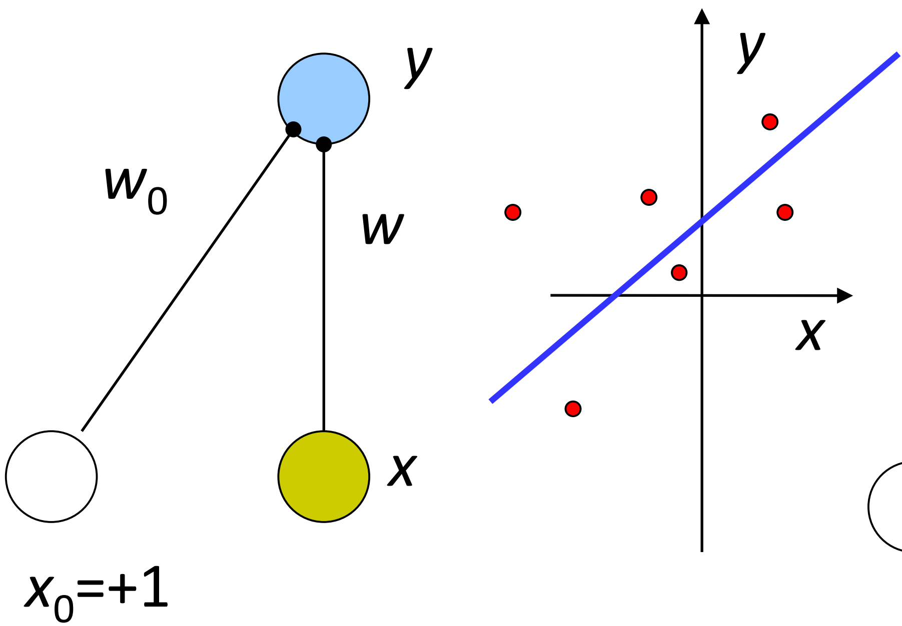
$$y = \sum_{j=1}^{d} w_j x_j + w_0 = \mathbf{w}^T \mathbf{x}$$

$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$$

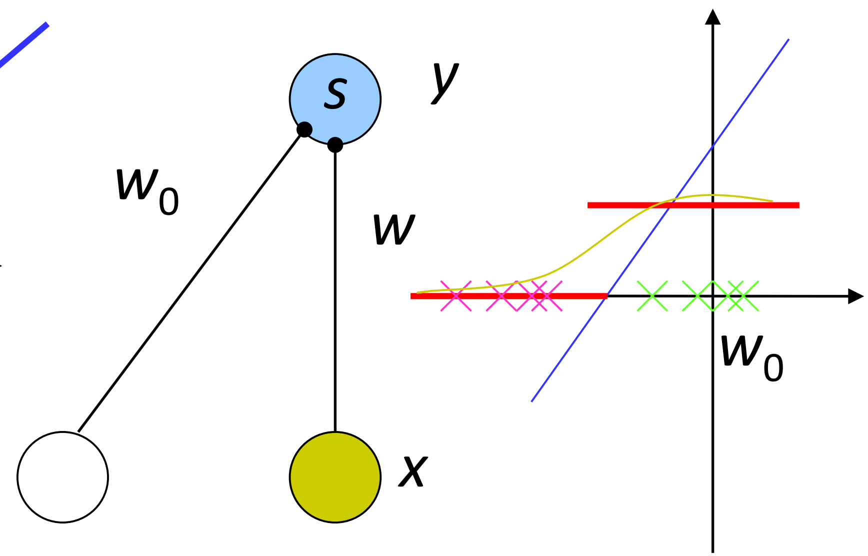
$$\mathbf{x} = [1, x_1, \ldots, x_d]^T$$

What a Perceptron Does

- Regression: $y = wx + w_0$



- Classification: $y = 1(wx + w_0 > 0)$



$$y = \text{sigmoid}\,(o) = \frac{1}{1 + \exp\left[-\mathbf{w}^T \mathbf{x}\right]}$$

K Outputs

Regression:

$$y_i = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = \mathbf{w}_i^T \mathbf{x}$$

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

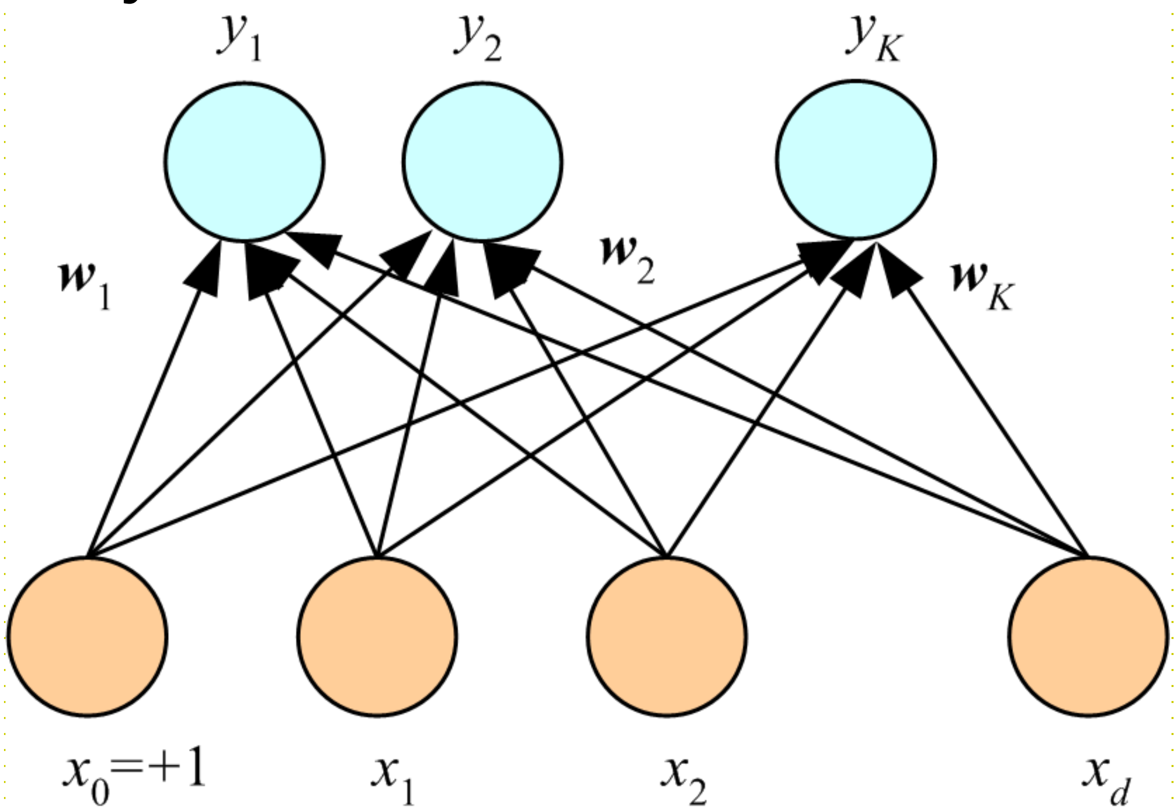
Classification:

$$o_i = \mathbf{w}_i^T \mathbf{x}$$

$$y_i = \frac{\exp o_i}{\sum_k \exp o_k}$$

choose C_i

if $y_i = \max_k y_k$



Training

- Online (instances seen one by one) vs batch (whole sample) learning:
  - No need to store the whole sample
  - Problem may change in time
  - Wear and degradation in system components
- Stochastic gradient-descent: Update after a single pattern
- Generic update rule (LMS rule):

$$\Delta w_{ij}^t = \eta\left(r_i^t - y_i^t\right)x_j^t$$

$$\text{Update} = \text{LearningFactor} \cdot \left(\text{DesiredOutput} - \text{ActualOutput} \right) \cdot \text{Input}$$

Training a Perceptron: Regression

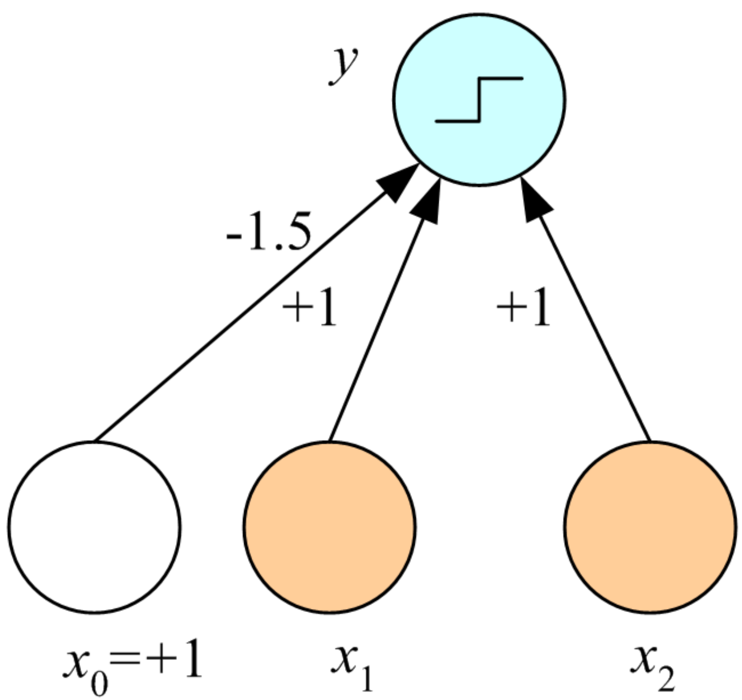
- Regression (Linear output):

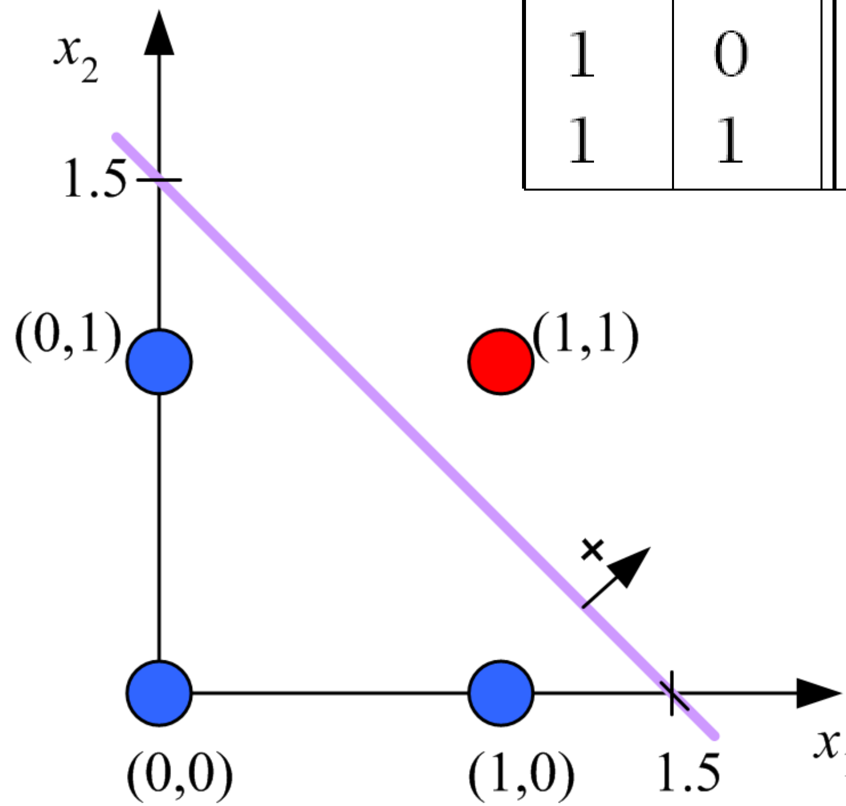
$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, r^t\right) = \frac{1}{2}\left(r^t - y^t\right)^2 = \frac{1}{2}\left[r^t - \left(\mathbf{w}^T \mathbf{x}^t\right)\right]^2$$

$$\Delta w_j^t = \eta\left(r^t - y^t\right)x_j^t$$

Single Layer Perceptron: AND

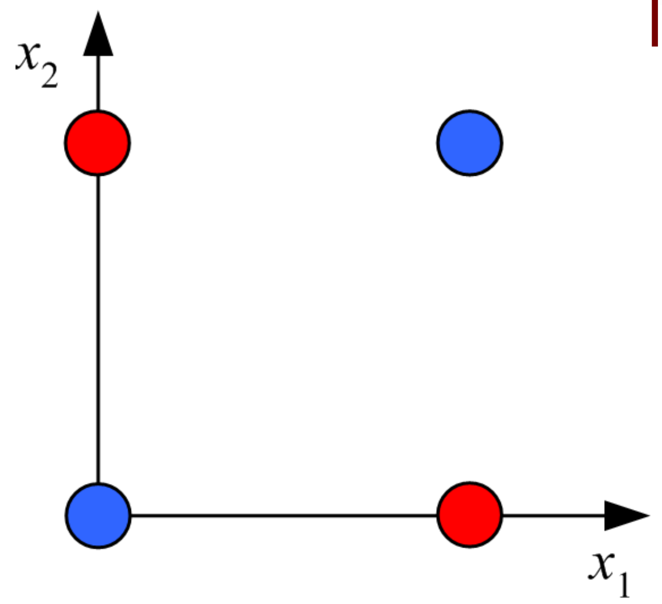
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



- **Problem: No w_0, w_1, w_2 that can satisfy:**

$$
\begin{array}{rrrl}
 & & w_0 & \leq 0 \\
 & w_2 + & w_0 & > 0 \\
w_1 + & & w_0 & > 0 \\
w_1 + & w_2 + & w_0 & \leq 0
\end{array}
$$

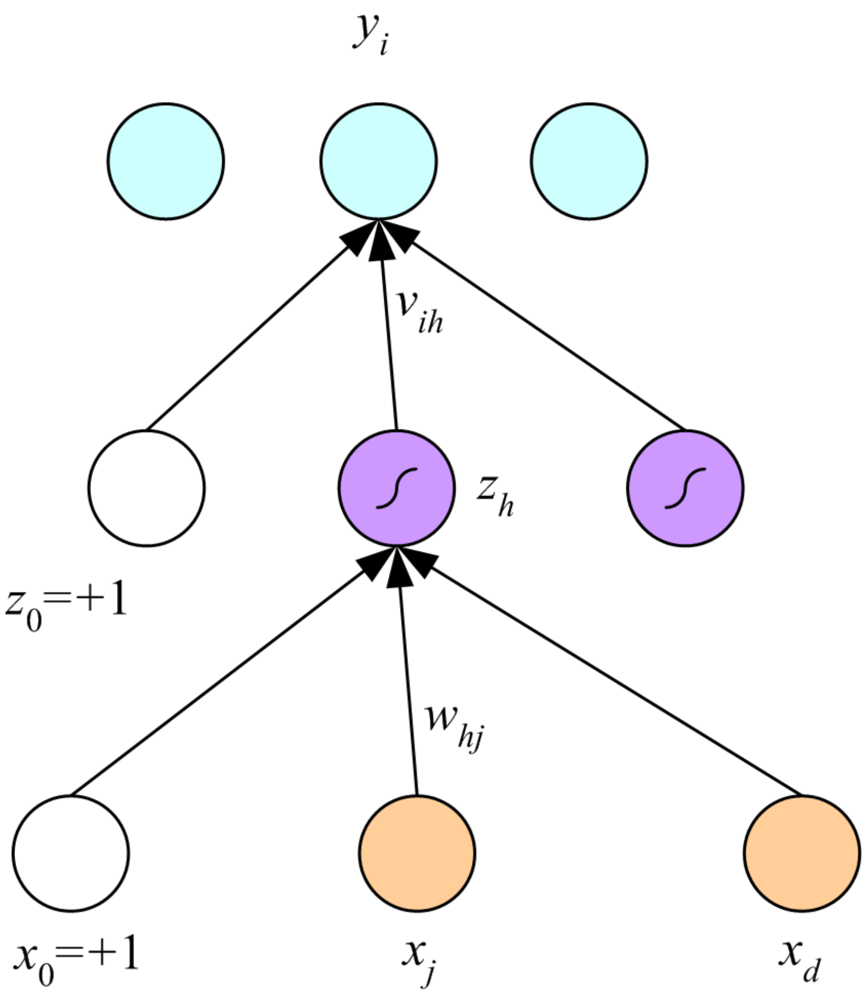
(Minsky and Papert, 1969)

A single layer network can not represent XOR.

Multilayer : Why ?

- Single layer can represent linearly separable data
  - Can not represent data which can not be linearly separable: Limited representation ability
- Multilayer
  - Can represent non linear separation
  - A 2-layer perceptron can represent any bounded continuous function with arbitrarily small error

Multilayer Perceptrons

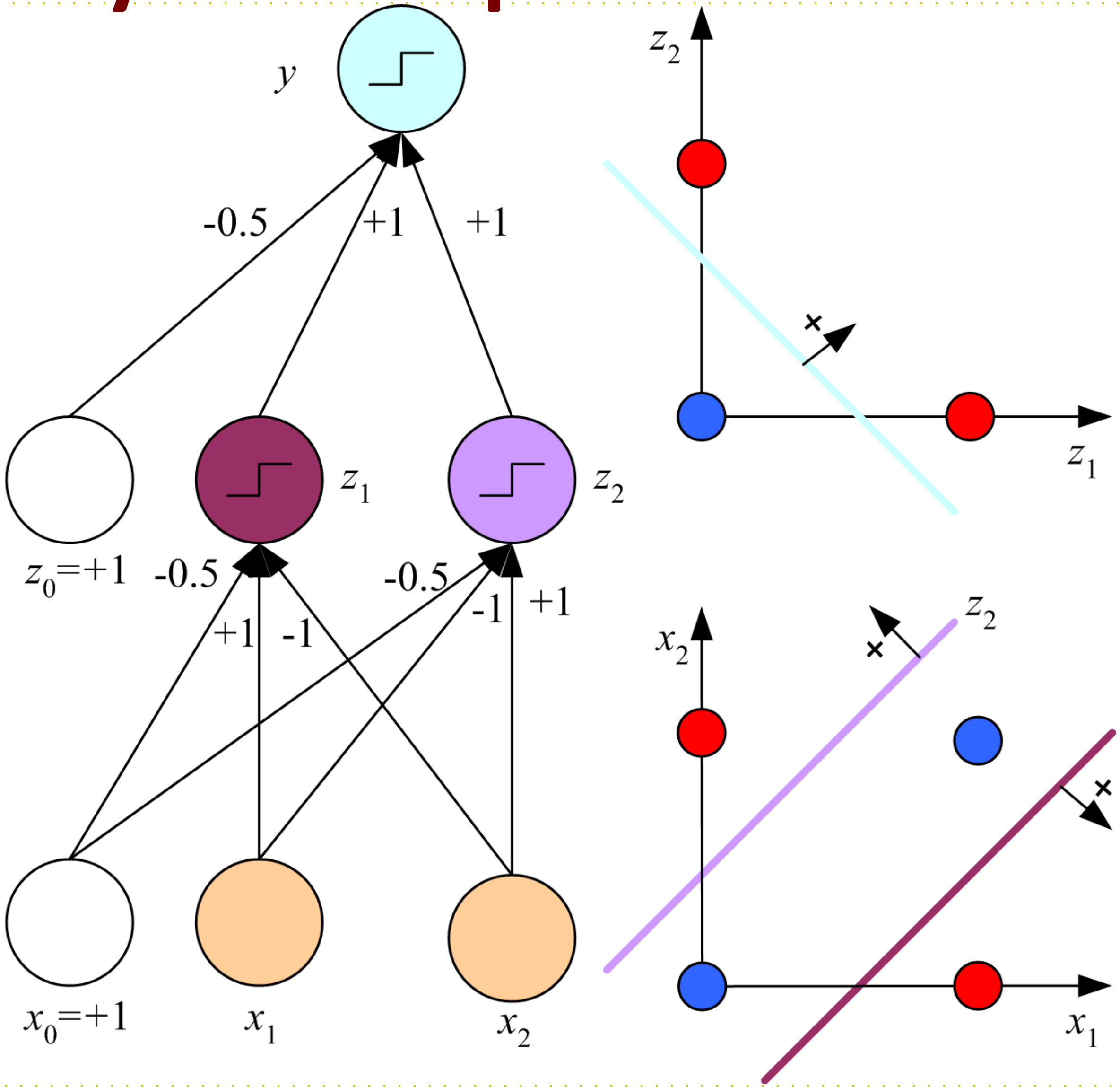


$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\mathbf{w}_h^T \mathbf{x}\right) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

(Rumelhart et al., 1986)

Multilayer Perceptrons: XOR

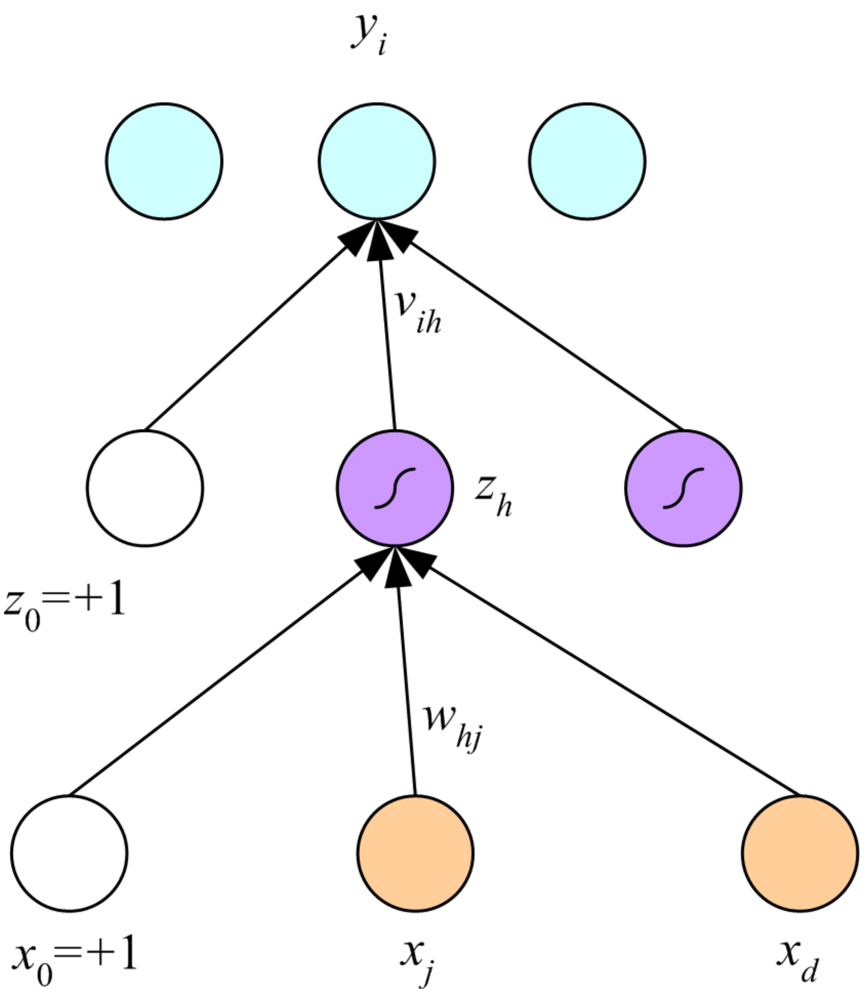


x_1 XOR x_2 = (x_1 AND ~x_2) OR (~x_1 AND x_2)

Thresholding function

- Linear combination of inputs will result into another input combinations
  - Can not represent nonlinearity
- Thresholding function aka transfer function
  - Must be continuous and differentiable (Because of gradient descent)
  - Popular functions: Sigmoid (Most popular), Hyperbolic tangent function (tanh),.. Etc.

Training Multilayer: Backpropagation



$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\mathbf{w}_h^T \mathbf{x}\right)$$

$$= \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

$$\boxed{\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}}$$

Multilayer Training: Regression

$$E(\mathbf{W},\mathbf{v}\mid\mathcal{X})=\frac{1}{2}\sum_t\left(r^t-y^t\right)^2$$

$$\Delta v_h=\sum_t\left(r^t-y^t\right)z_h^t$$

Backward

$$y^t=\sum_{h=1}^{H}v_h z_h^t+v_0$$

Forward

$$z_h=\boxed{\text{sigmoid}}\left(\mathbf{w}_h^T\mathbf{x}\right)$$

$\boldsymbol{x}$

$$\begin{aligned}\Delta w_{hj}&=-\eta\frac{\partial E}{\partial w_{hj}}\\&=-\eta\sum_t\frac{\partial E}{\partial y^t}\frac{\partial y^t}{\partial z_h^t}\frac{\partial z_h^t}{\partial w_{hj}}\\&=-\eta\sum_t-\left(r^t-y^t\right)v_h\boxed{z_h^t\left(1-z_h^t\right)}x_j^t\\&=\eta\sum_t\left(r^t-y^t\right)v_h z_h^t\left(1-z_h^t\right)x_j^t\end{aligned}$$

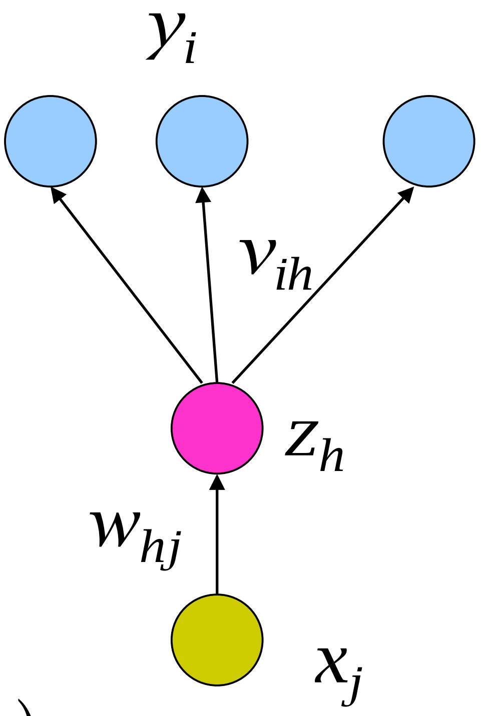
Regression with Multiple Outputs

$$E(\mathbf{W},\mathbf{V}|\mathcal{X}) = \frac{1}{2}\sum_t \sum_i \left(r_i^t - y_i^t\right)^2$$

$$y_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0}$$

$$\Delta v_{ih} = \eta \sum_t \left(r_i^t - y_i^t\right) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) v_{ih} \right] z_h^t \left(1 - z_h^t\right) x_j^t$$



Backpropagation: Algorithm

Initialize all v_{ih} and w_{hj} to rand$(-0.01, 0.01)$

Repeat

For all $(\boldsymbol{x}^t, r^t) \in \mathcal{X}$ in random order

For $h = 1, \ldots, H$

$z_h \leftarrow \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}^t)$

For $i = 1, \ldots, K$

$y_i = \boldsymbol{v}_i^T \boldsymbol{z}$

For $i = 1, \ldots, K$

$\Delta \boldsymbol{v}_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$

For $h = 1, \ldots, H$

$\Delta \boldsymbol{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \boldsymbol{x}^t$

For $i = 1, \ldots, K$

$\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta \boldsymbol{v}_i$

For $h = 1, \ldots, H$

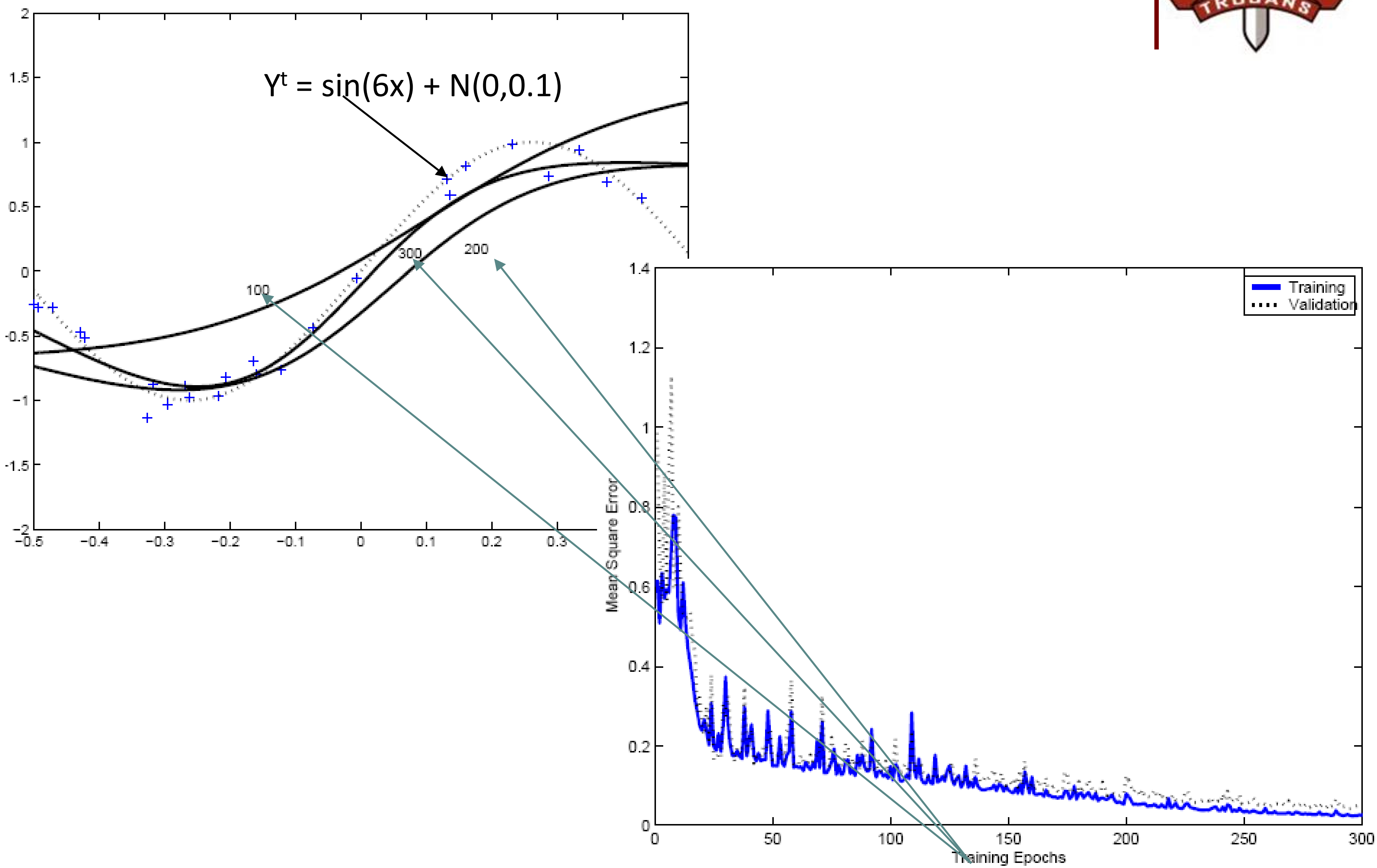
$\boldsymbol{w}_h \leftarrow \boldsymbol{w}_h + \Delta \boldsymbol{w}_h$

Until convergence

Learning: Issues & Approaches

- Number of epochs
  - Fixed number vs target validation error
- Batch vs Stochastic
- Avoid Overfitting: avoid divergence of errors between training set and validation set.

Backpropagation: Fitting a curve



Two-Class Discrimination

- One sigmoid output y^t for $P(C_1|\boldsymbol{x}^t)$ and $P(C_2|\boldsymbol{x}^t) \equiv 1-y^t$

$$y^t = \text{sigmoid}\left(\sum_{h=1}^{H} v_h z_h^t + v_0\right)$$

$$E(\mathbf{W},\mathbf{v}|\mathcal{X}) = -\sum_t r^t \log y^t + \left(1-r^t\right)\log\left(1-y^t\right)$$

$$\Delta v_h = \eta \sum_t \left(r^t - y^t\right) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left(r^t - y^t\right) v_h z_h^t \left(1 - z_h^t\right) x_j^t$$

K>2 Classes

$$o_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \qquad y_i^t = \frac{\exp o_i^t}{\sum_k \exp o_k^t} \equiv P(C_i \mid \mathbf{x}^t)$$

$$E(\mathbf{W}, \mathbf{v} \mid \mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t$$

$$\Delta v_{ih} = \eta \sum_t \left(r_i^t - y_i^t\right) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) v_{ih} \right] z_h^t \left(1 - z_h^t\right) x_j^t$$

Multiple Hidden Layers

- MLP with one hidden layer is a universal approximator (Hornik et al., 1989), but using multiple layers may lead to simpler networks

$$z_{1h} = \text{sigmoid}\left(\mathbf{w}_{1h}^T \mathbf{x}\right) = \text{sigmoid}\left(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0}\right), h = 1, \ldots, H_1$$

$$z_{2l} = \text{sigmoid}\left(\mathbf{w}_{2l}^T \mathbf{z}_1\right) = \text{sigmoid}\left(\sum_{h=1}^{H_1} w_{2lh} z_{1h} + w_{2l0}\right), l = 1, \ldots, H_2$$

$$y = \mathbf{v}^T \mathbf{z}_2 = \sum_{l=1}^{H_2} v_l z_{2l} + v_0$$

Improving Convergence

- Momentum

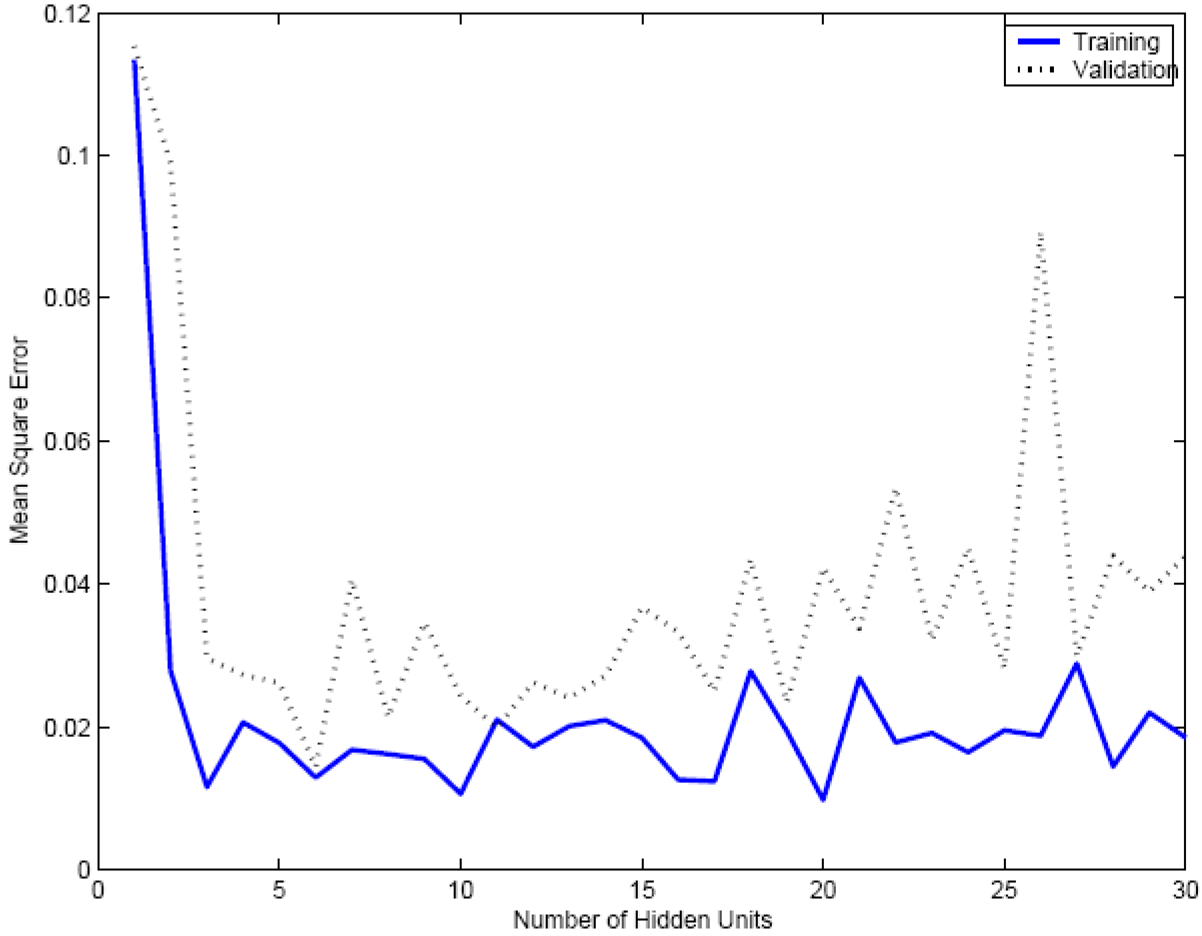
$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

- Adaptive learning rate

$$\Delta \eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases}$$

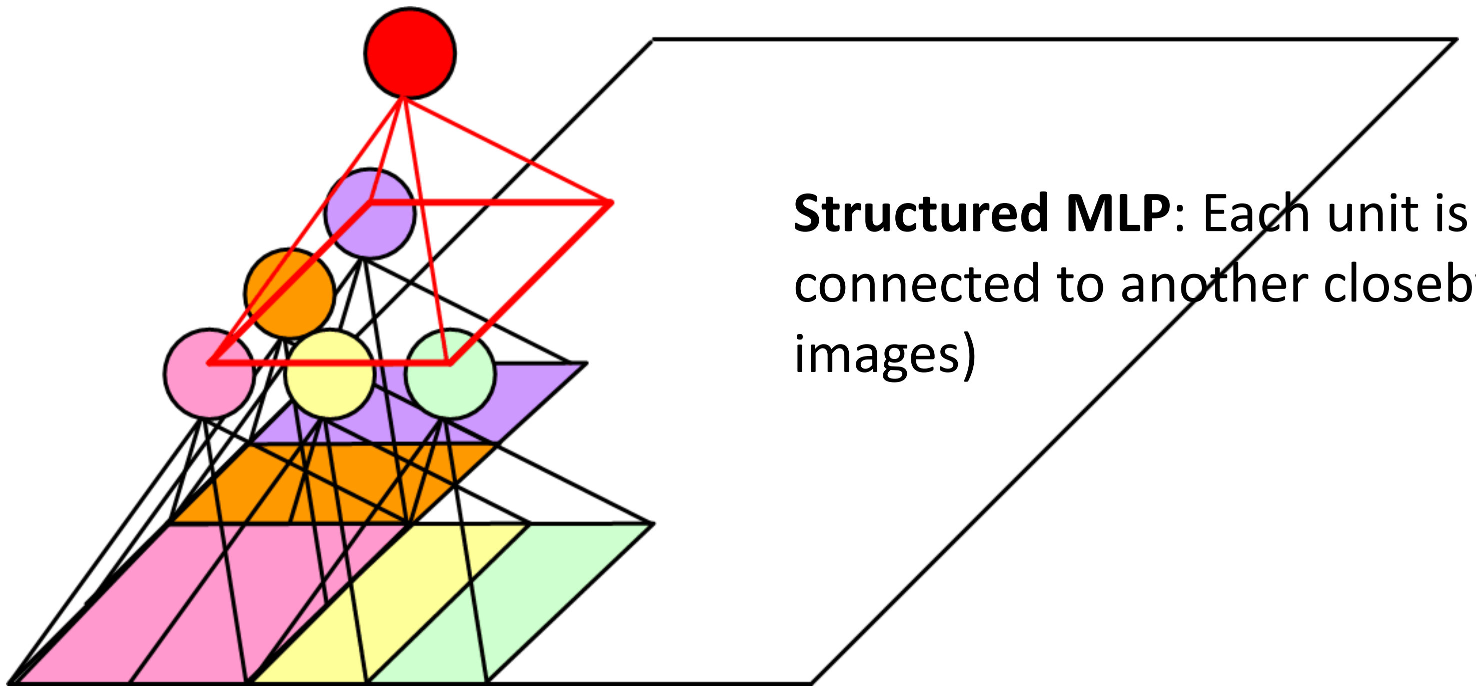
Overfitting/Overtraining

Number of weights: $H(d+1)+(H+1)K$



Structuring the MLP

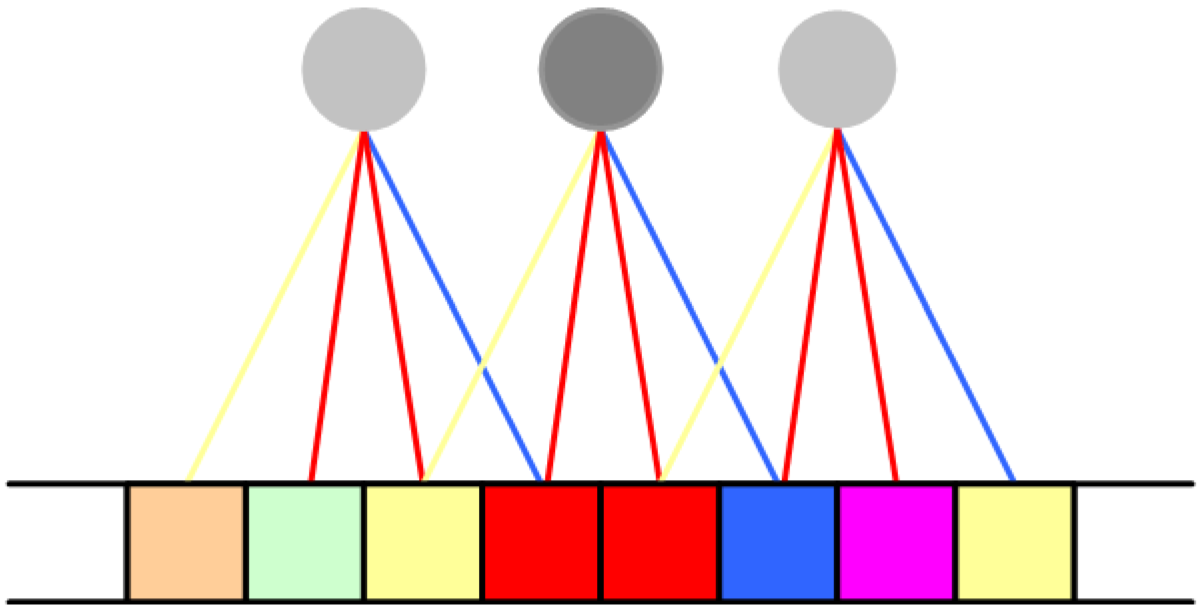
- Structure MLP using domain knowledge
- Large structures are hard to train
- Use hints from the local structures
  - Invariance to translation, rotation, size



Structured MLP: Each unit is connected to another closeby unit (ex: images)

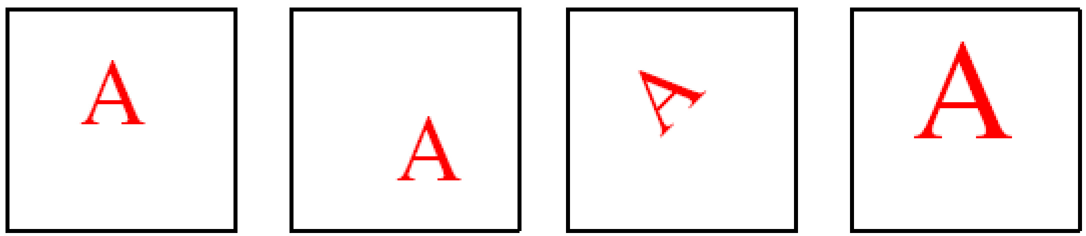
Structure the MLP

Weight sharing: shares the same weight but connected to different inputs



Hints

(Abu-Mostafa, 1995)



- Virtual
- Augmented error: $E' = E + \lambda_h E_h$

If ***x'*** and ***x*** are the "same": $E_h = [g(x|\theta) - g(x'|\theta)]^2$

Approximation hint:

$$E_h = \begin{cases} 0 & \text{if } g(x|\theta) \in [a_x, b_x] \\ \left(g(x|\theta) - a_x\right)^2 & \text{if } g(x|\theta) < a_x \\ \left(g(x|\theta) - b_x\right)^2 & \text{if } g(x|\theta) > b_x \end{cases}$$

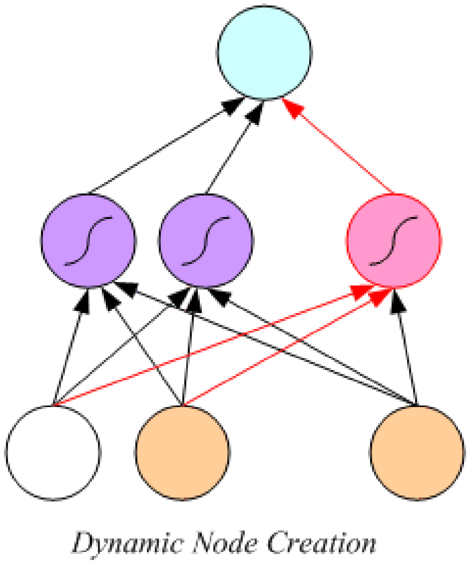
Tuning the Network Size

- Destructive
- Weight decay:

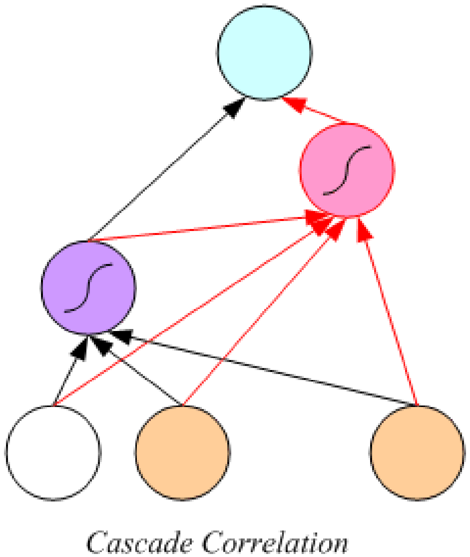
$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i$$

$$E' = E + \frac{\lambda}{2} \sum_i w_i^2$$

- Constructive
- Growing networks



(Ash, 1989)



(Fahlman and Lebiere, 1989)