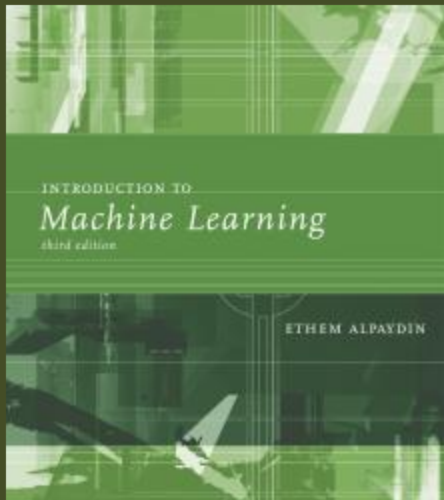


Lecture Slides for

INTRODUCTION TO MACHINE LEARNING

3RD EDITION

ETHEM ALPAYDIN

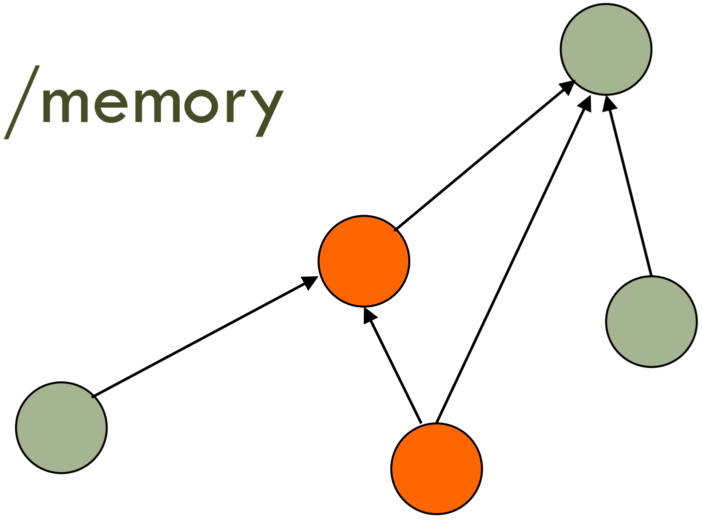
alpaydin@boun.edu.tr
http://www.cmpe.boun.edu.tr/~ethem/i2ml3e

CHAPTER 11:

MULTILAYER PERCEPTRONS

Neural Networks

- 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



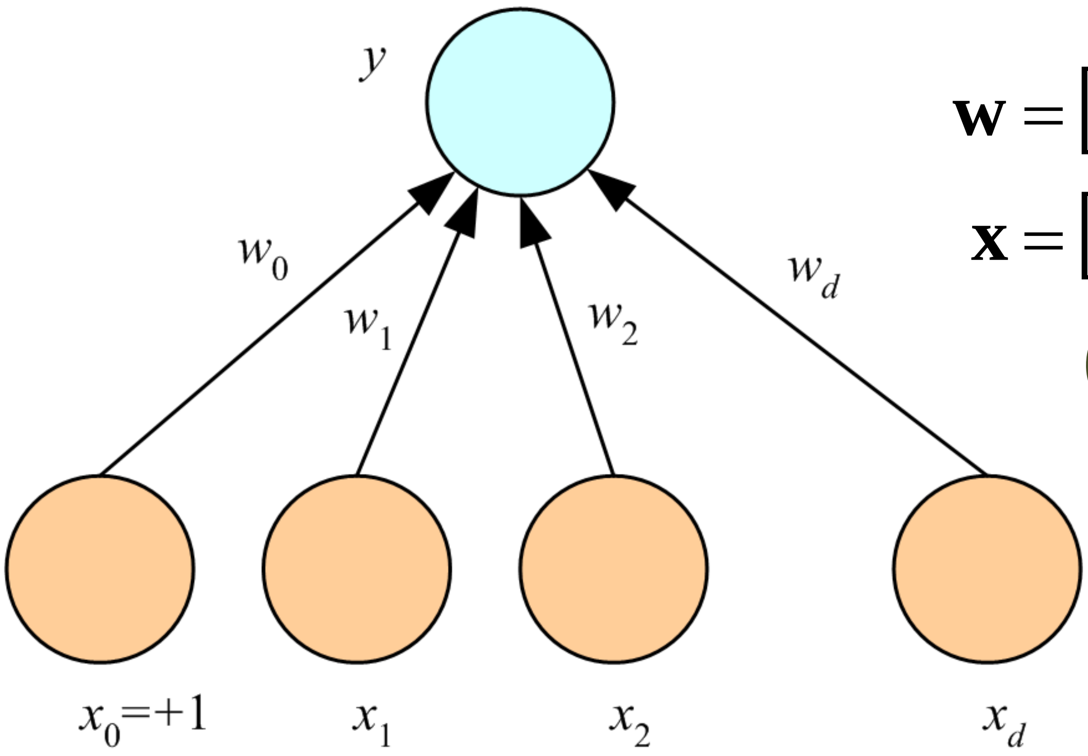
Understanding the Brain

- Levels of analysis (Marr, 1982)
  1. Computational theory
  2. Representation and algorithm
  3. Hardware implementation
- Reverse engineering: From hardware to theory
- Parallel processing: SIMD vs MIMD

  Neural net: SIMD with modifiable local memory

  Learning: Update by training/experience

Perceptron



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

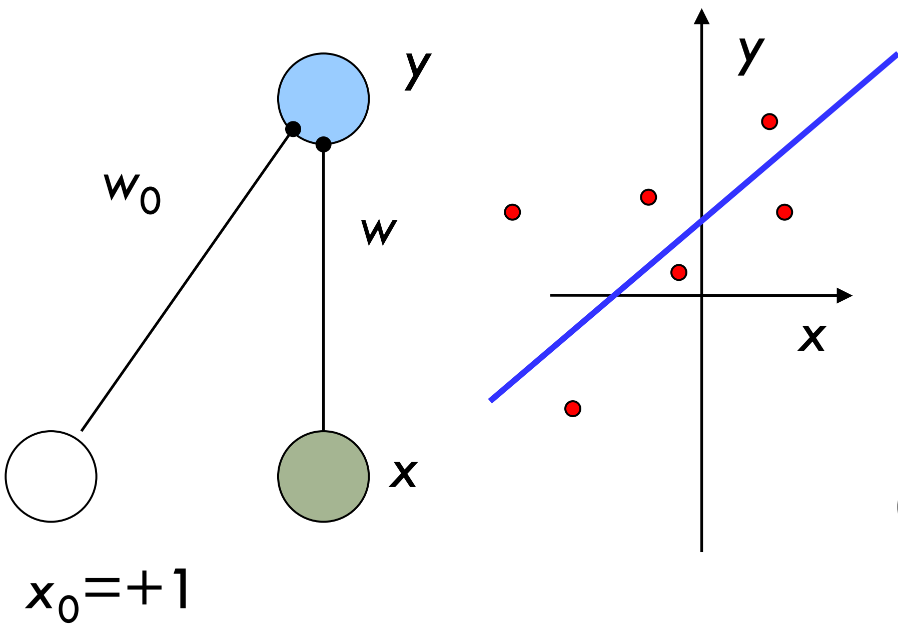
$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$$

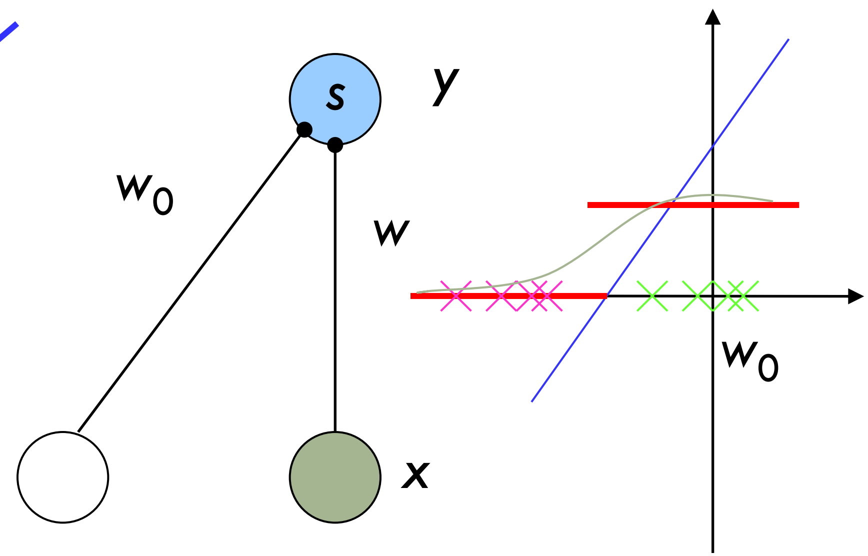
$$\mathbf{x} = [1, x_1, \ldots, x_d]^T$$

(Rosenblatt, 1962)

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:

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

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

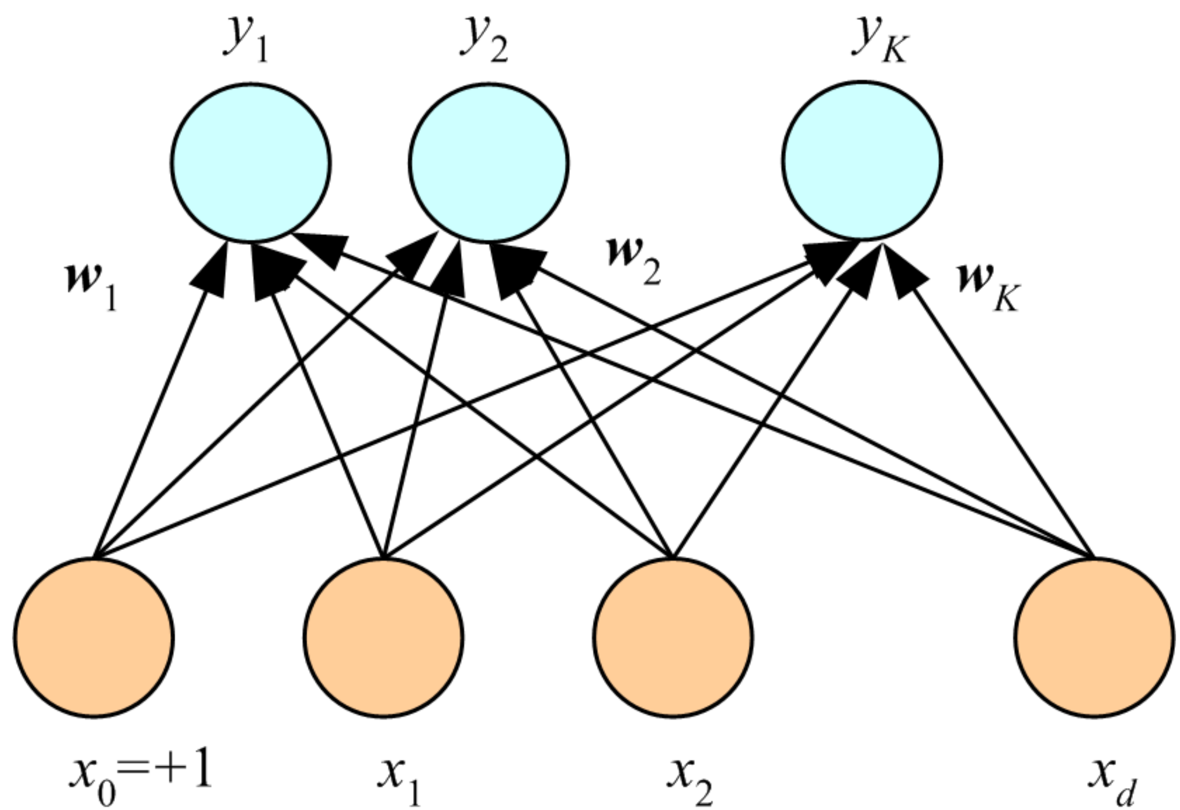
Classification:

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

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

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):

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

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

Classification

- Single sigmoid output

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

$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, \mathbf{r}^t\right) = -r^t \log y^t - \left(1 - r^t\right) \log\left(1 - y^t\right)$$

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

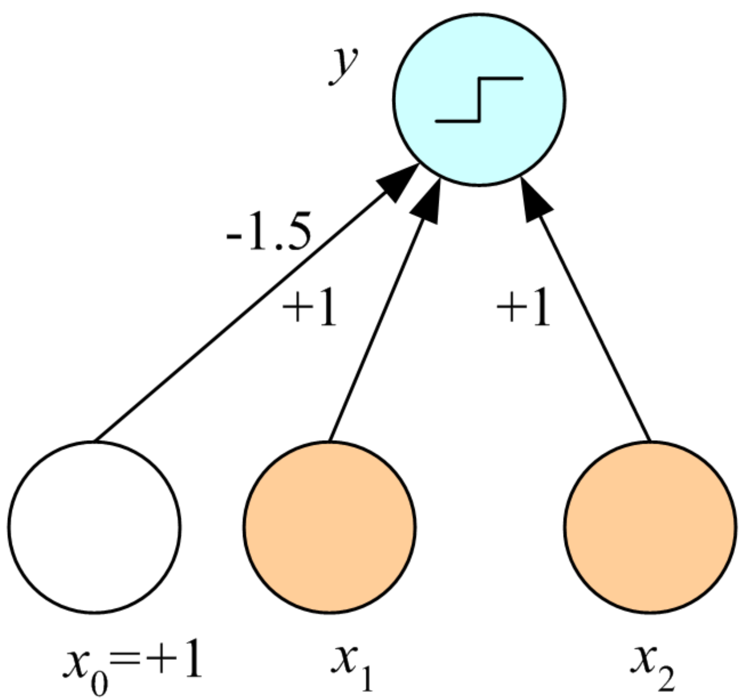
- *K*>2 softmax outputs

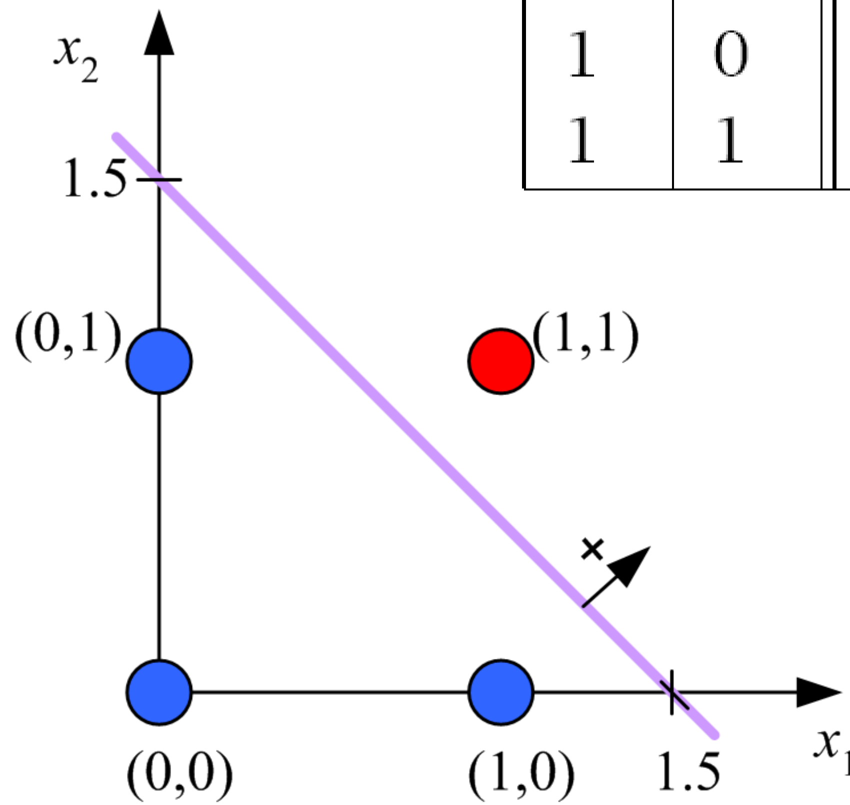
$$y^t = \frac{\exp \mathbf{w}_i^T \mathbf{x}^t}{\sum_k \exp \mathbf{w}_k^T \mathbf{x}^t} \qquad E^t\left(\{\mathbf{w}_i\}_i \mid \mathbf{x}^t, \mathbf{r}^t\right) = -\sum_i r_i^t \log y_i^t$$

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

Learning Boolean AND

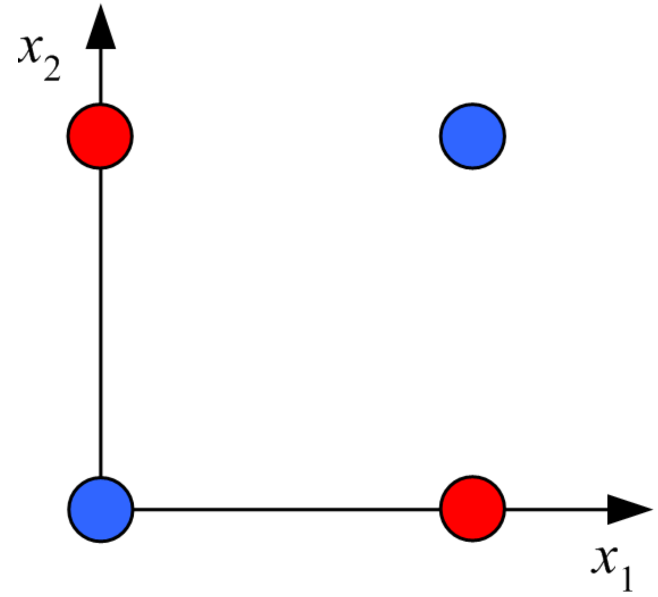
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





XOR

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

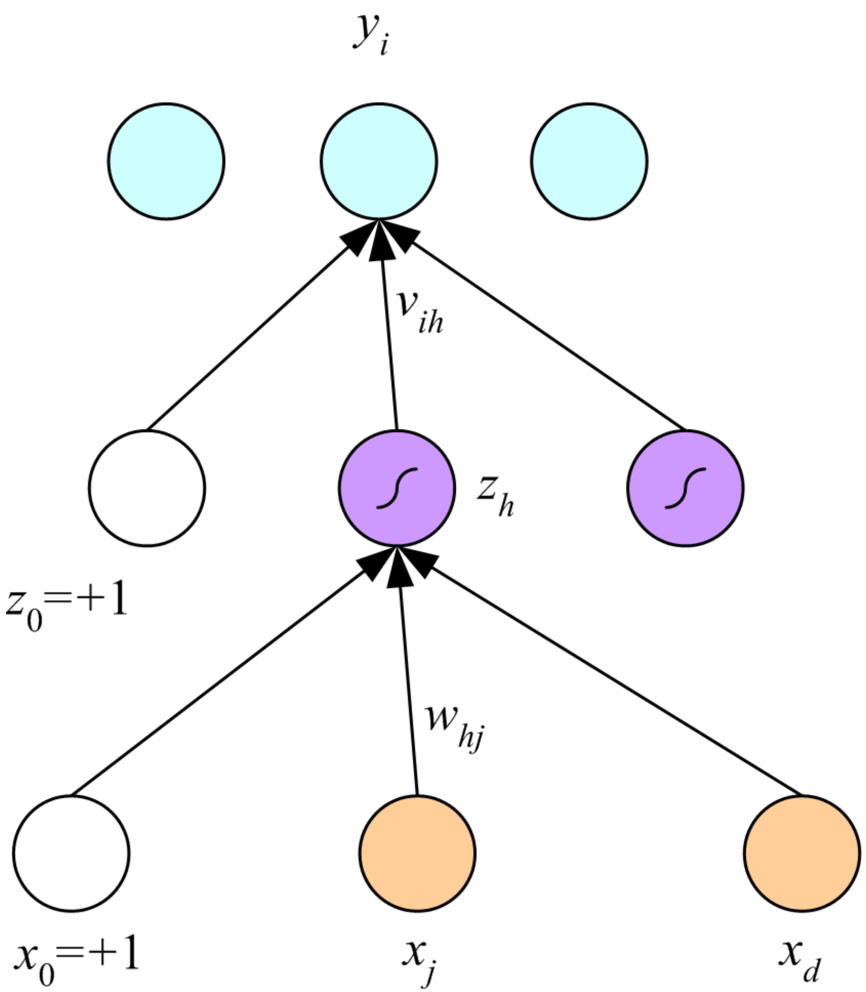


(Minsky and Papert, 1969)

- No w_0, w_1, w_2 satisfy:

$$
\begin{aligned}
w_0 &\le 0 \\
w_2 + w_0 &> 0 \\
w_1 + w_0 &> 0 \\
w_1 + w_2 + w_0 &\le 0
\end{aligned}
$$

Multilayer Perceptrons



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

Output Layer

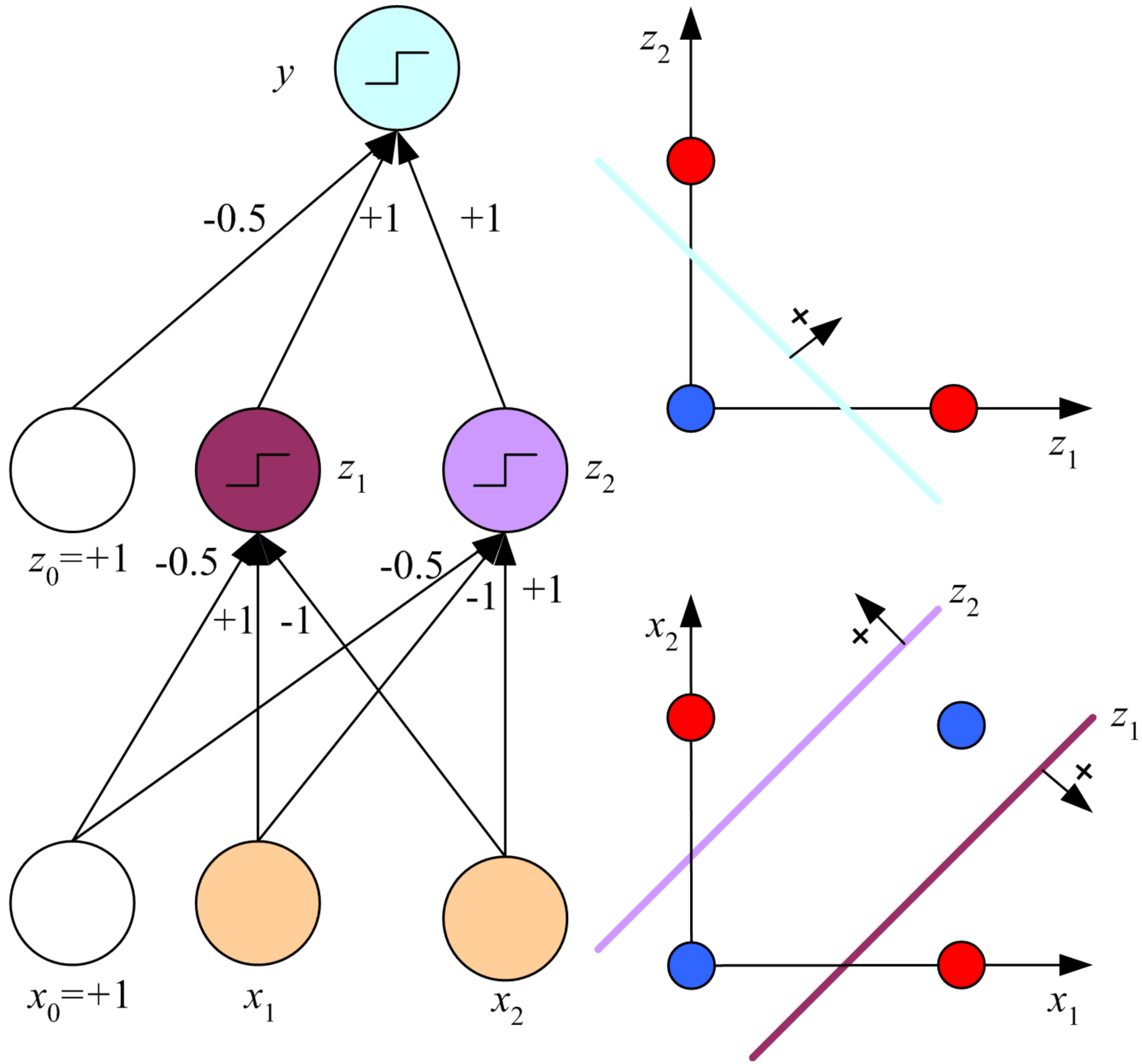
$$z_h = \text{sigmoid}\left(\mathbf{w}_h^T \mathbf{x}\right)$$

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

Hidden

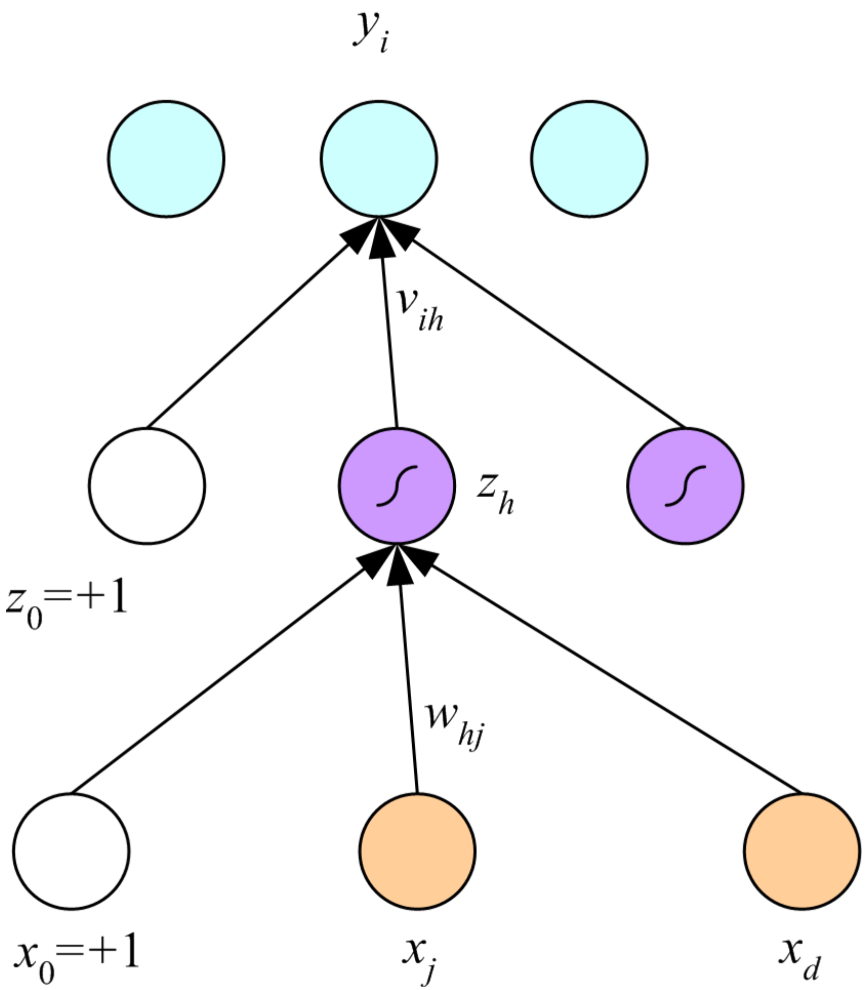
(Rumelhart et al., 1986)

Input L



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

Backpropagation



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

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

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

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

Regression

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

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

Backward

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

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

Forward

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

$$\mathbf{x}$$

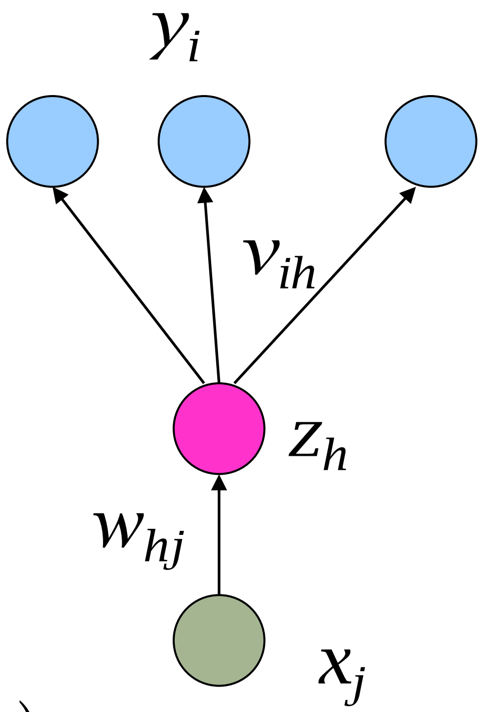
Regression with Multiple Outputs

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

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

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

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



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 $h = 1, \ldots, H$

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

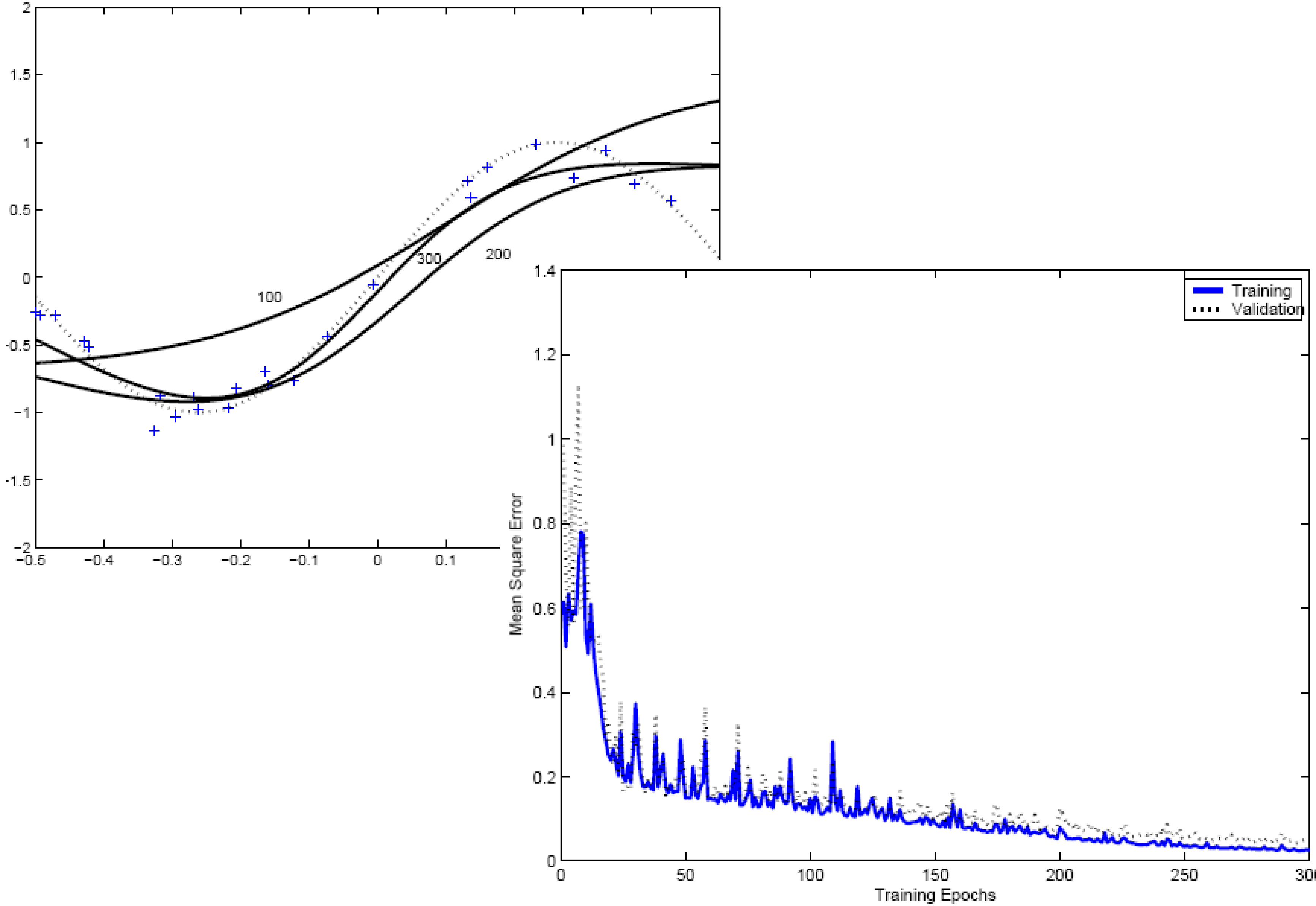
        For $i = 1, \ldots, K$

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

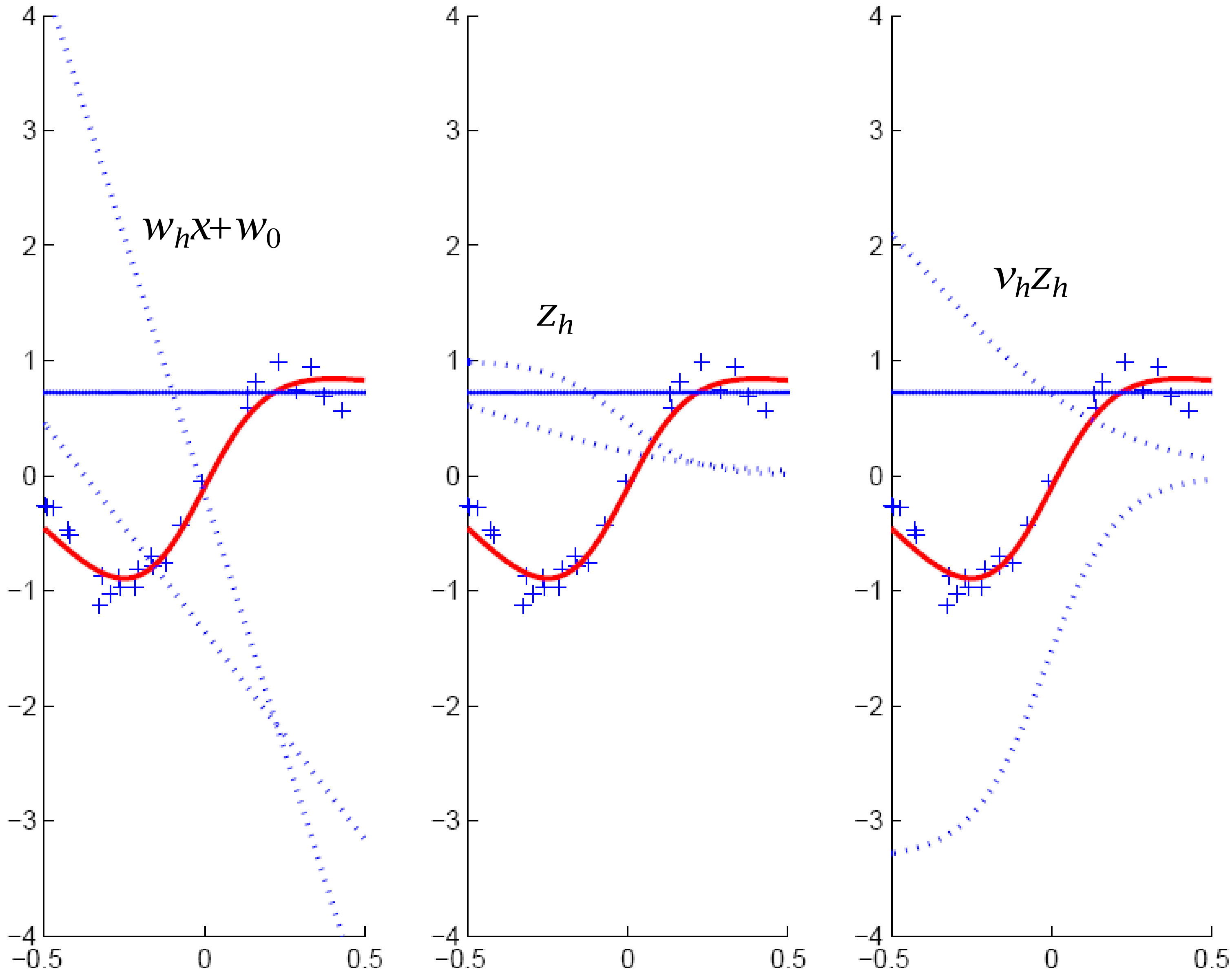
        For $h = 1, \ldots, H$

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

Until convergence



100
200
300
Training
Validation
Mean Square Error
Training Epochs



$w_h x + w_0$
z_h
$v_h z_h$
4
3
2
1
0
−1
−2
−3
−4
−0.5
0
0.5

Two-Class Discrimination

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

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

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

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

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

K>2 Classes

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

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

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

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

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

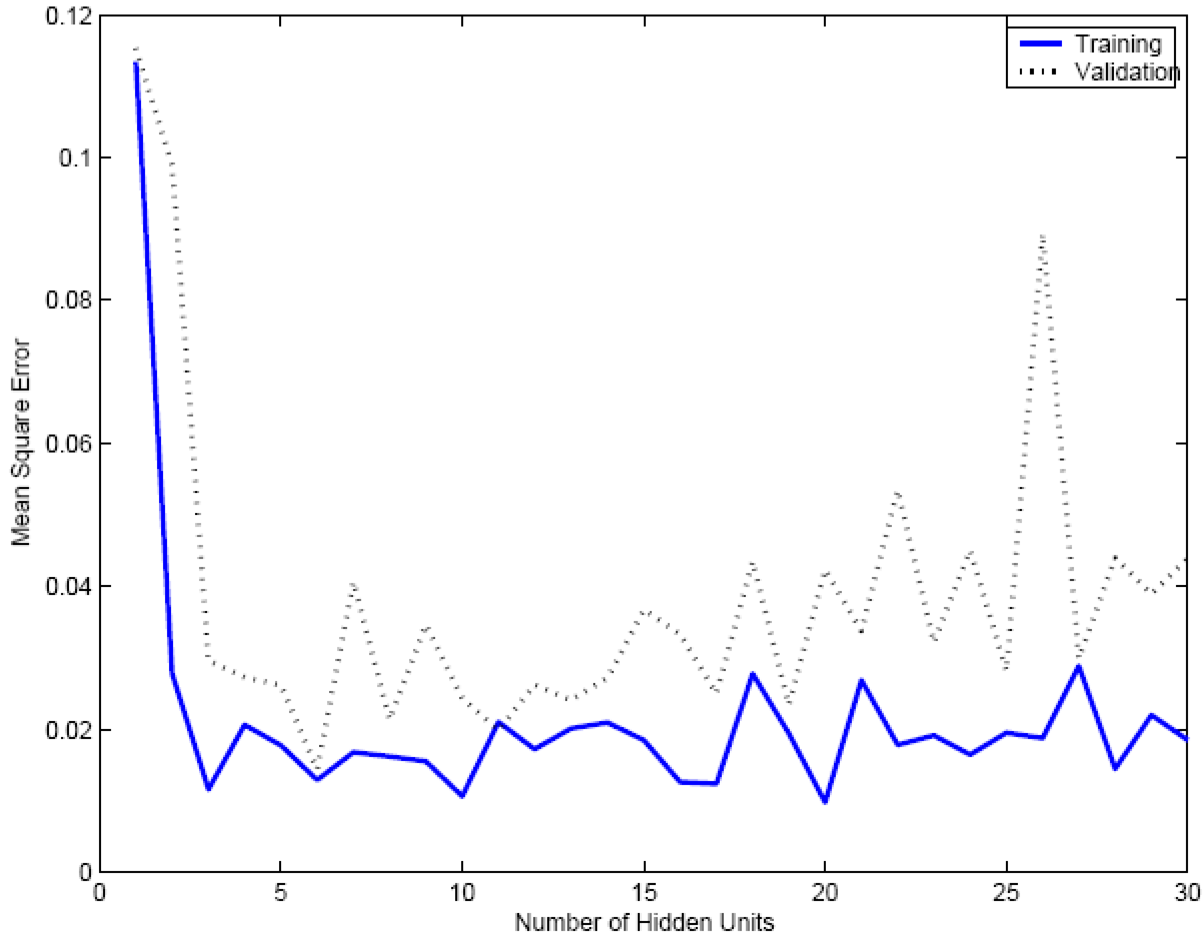
$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

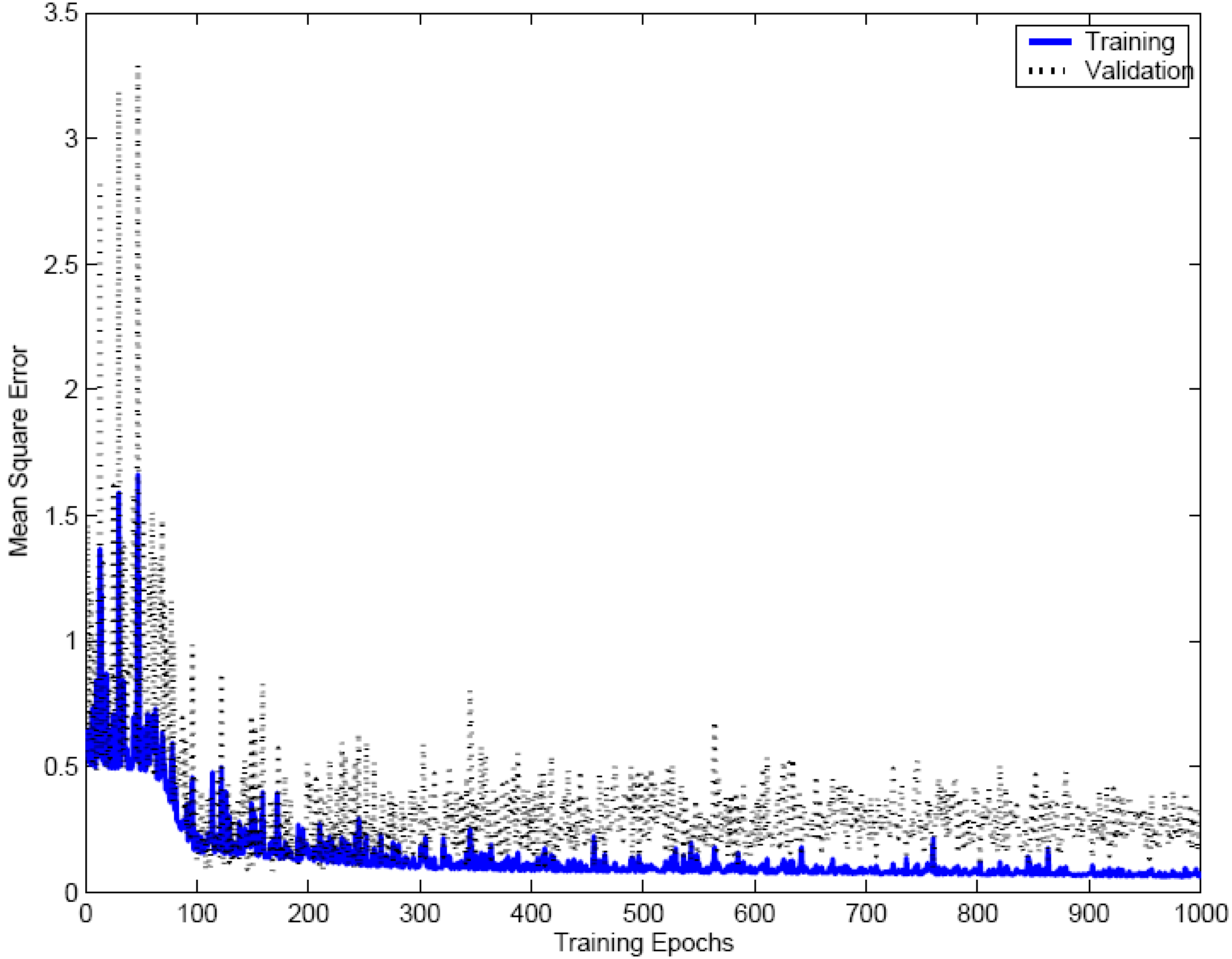
- Adaptive learning rate

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

Overfitting/Overtraining

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

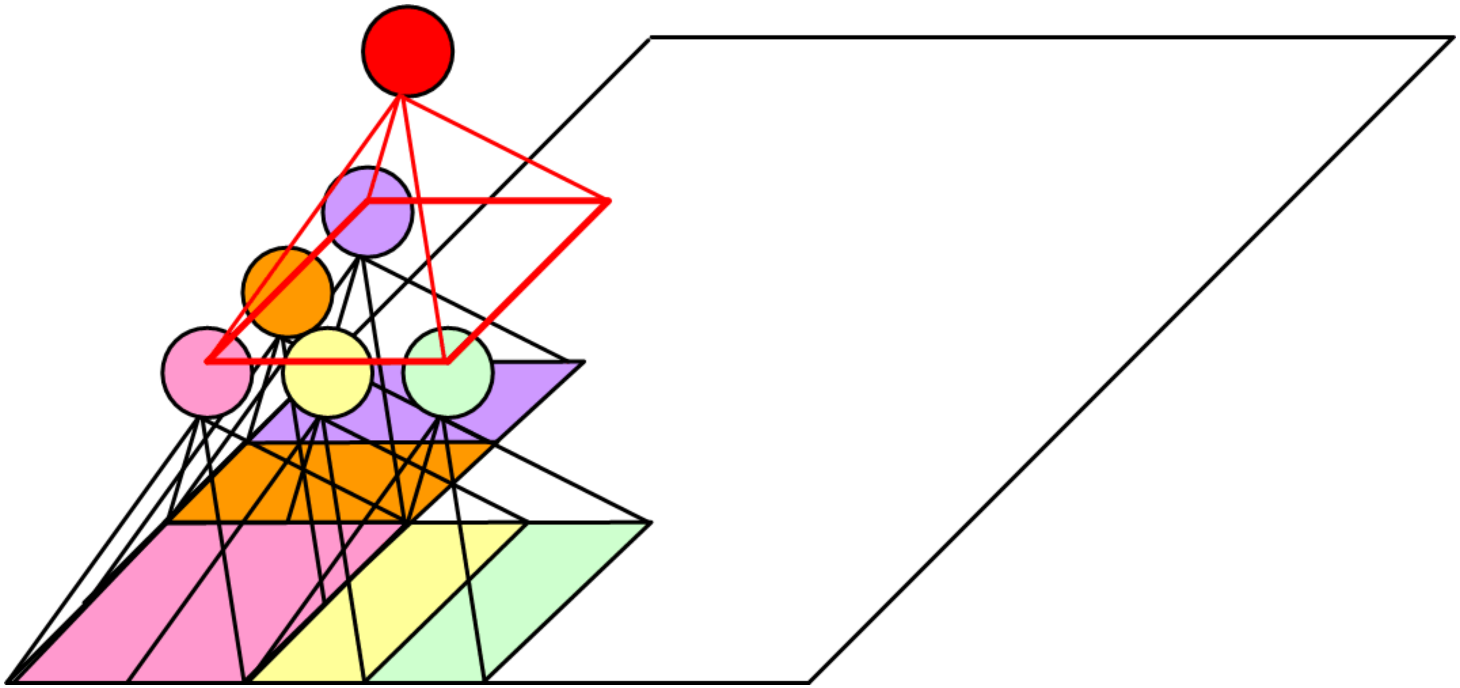




Training
Validation
Mean Square Error
Training Epochs
0
0.5
1
1.5
2
2.5
3
3.5
100
200
300
400
500
600
700
800
900
1000

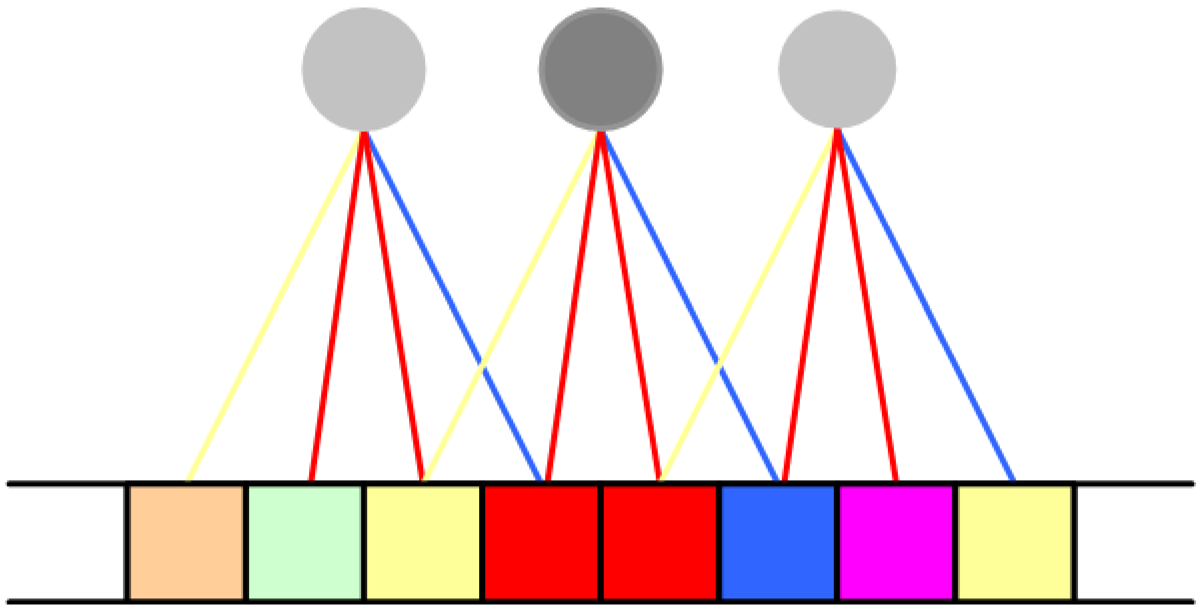
Structured MLP

- Convolutional networks (Deep learning)



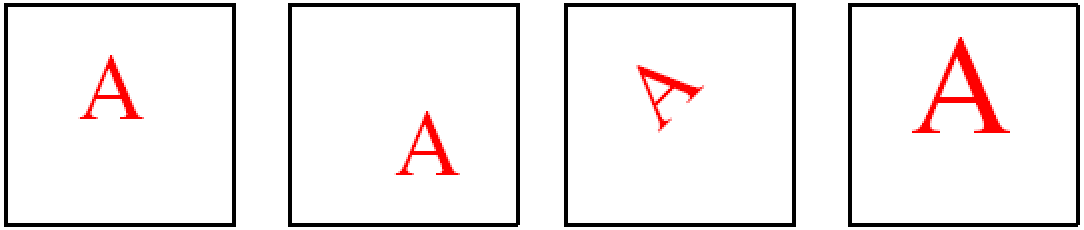
(Le Cun et al, 1989)

Weight Sharing



Hints

- Invariance to translation, rotation, size



- Virtual examples (Abu-Mostafa, 1995)
- Augmented error: $E' = E + \lambda_h E_h$

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

Approximation hint:

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

Tuning the Network Size

- Destructive

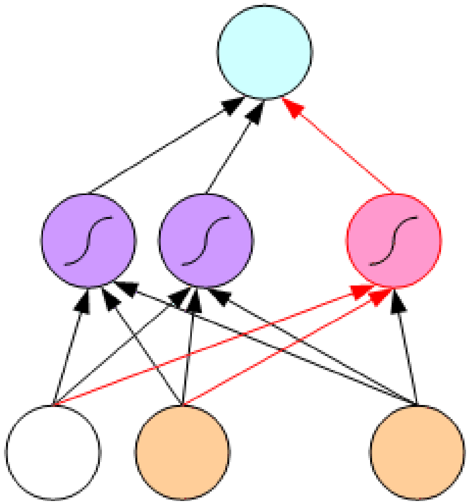
  Weight decay:

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

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

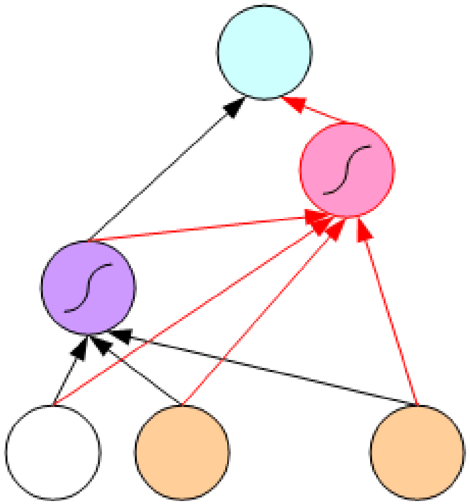
- Constructive

  Growing networks



Dynamic Node Creation

(Ash, 1989)



Cascade Correlation

(Fahlman and Lebiere, 1989)

Bayesian Learning

- Consider weights w_i as random vars, prior $p(w_i)$

$$p(\mathbf{w}|\mathcal{X}) = \frac{p(\mathcal{X}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{X})} \quad \hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log p(\mathbf{w}|\mathcal{X})$$

$$\log p(\mathbf{w}|\mathcal{X}) = \log p(\mathcal{X}|\mathbf{w}) + \log p(\mathbf{w}) + C$$

$$p(\mathbf{w}) = \prod_i p(w_i) \text{ where } p(w_i) = c \cdot \exp\left[-\frac{w_i^2}{2(1/2\lambda)}\right]$$

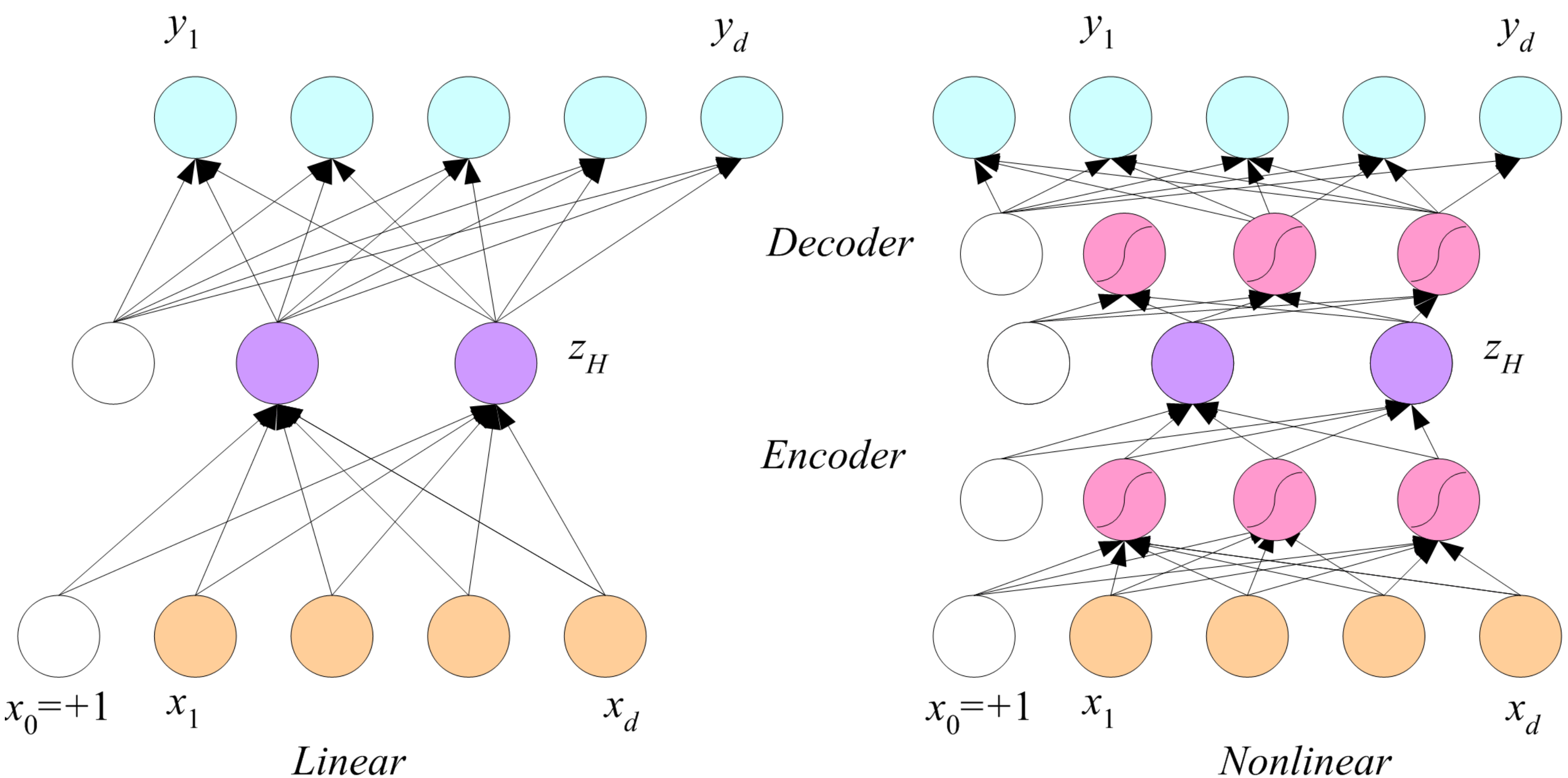
$$E' = E + \lambda \|\mathbf{w}\|^2$$

- Weight decay, ridge regression, regularization

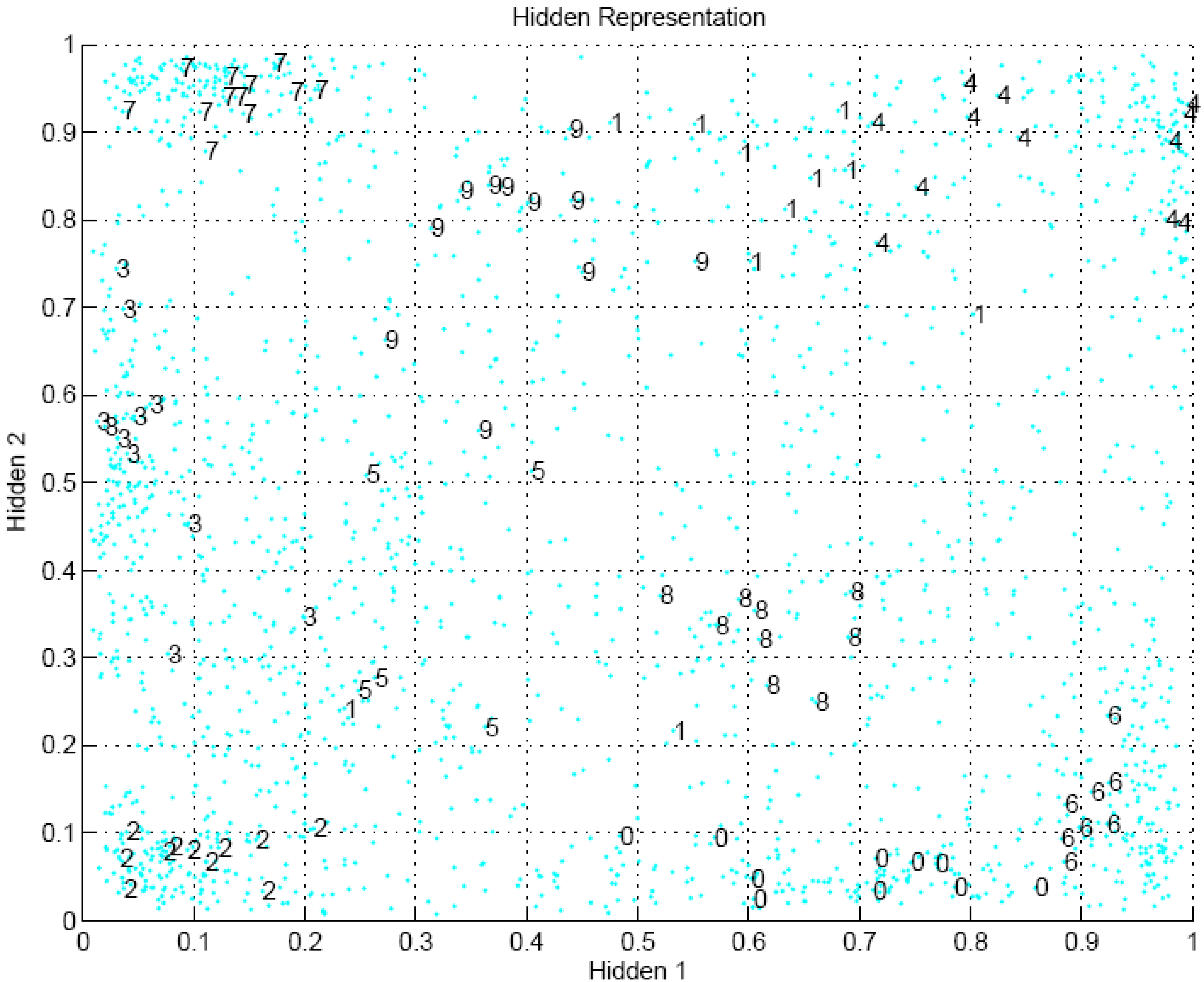
  cost=data-misfit + λ complexity

  More about Bayesian methods in chapter 14

Dimensionality Reduction



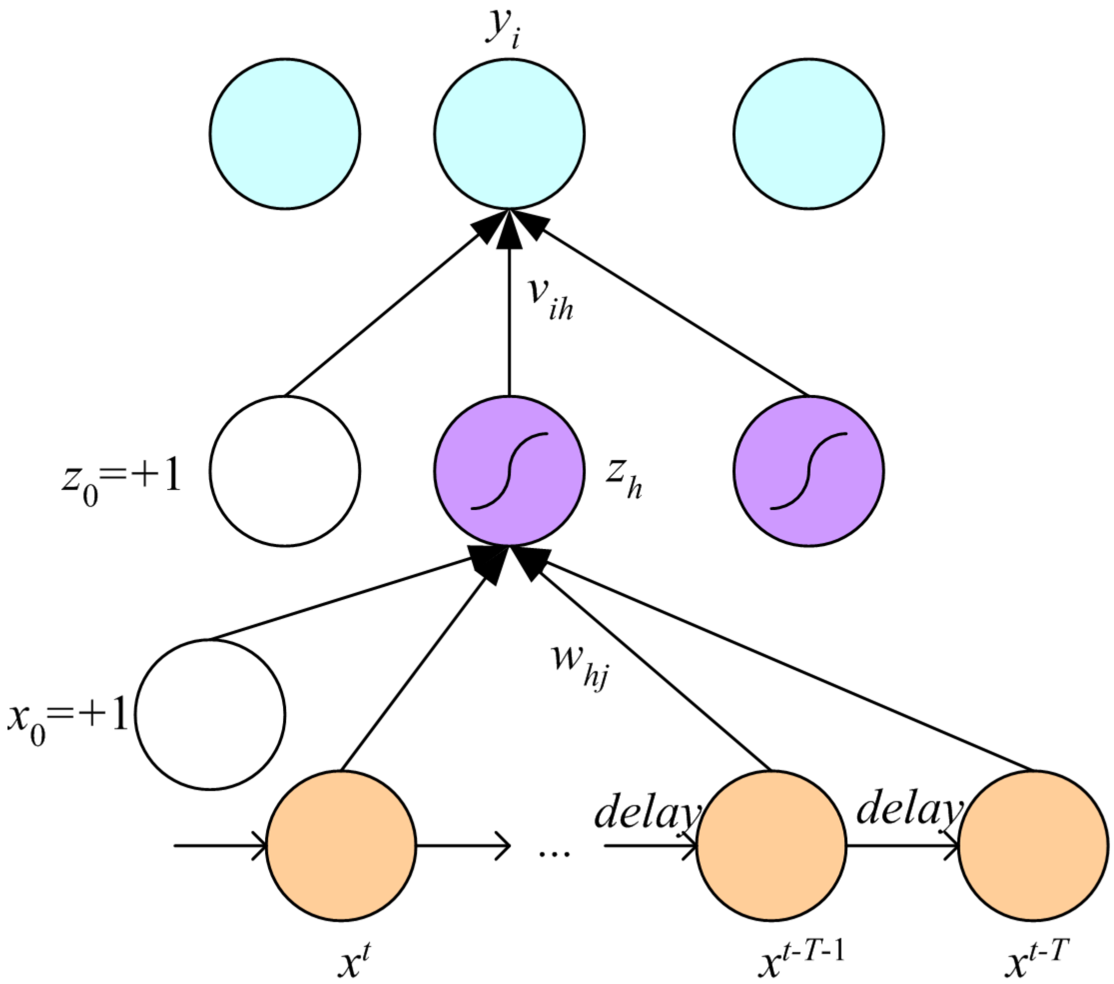
Autoencoder networks


Hidden Representation
Hidden 1
Hidden 2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

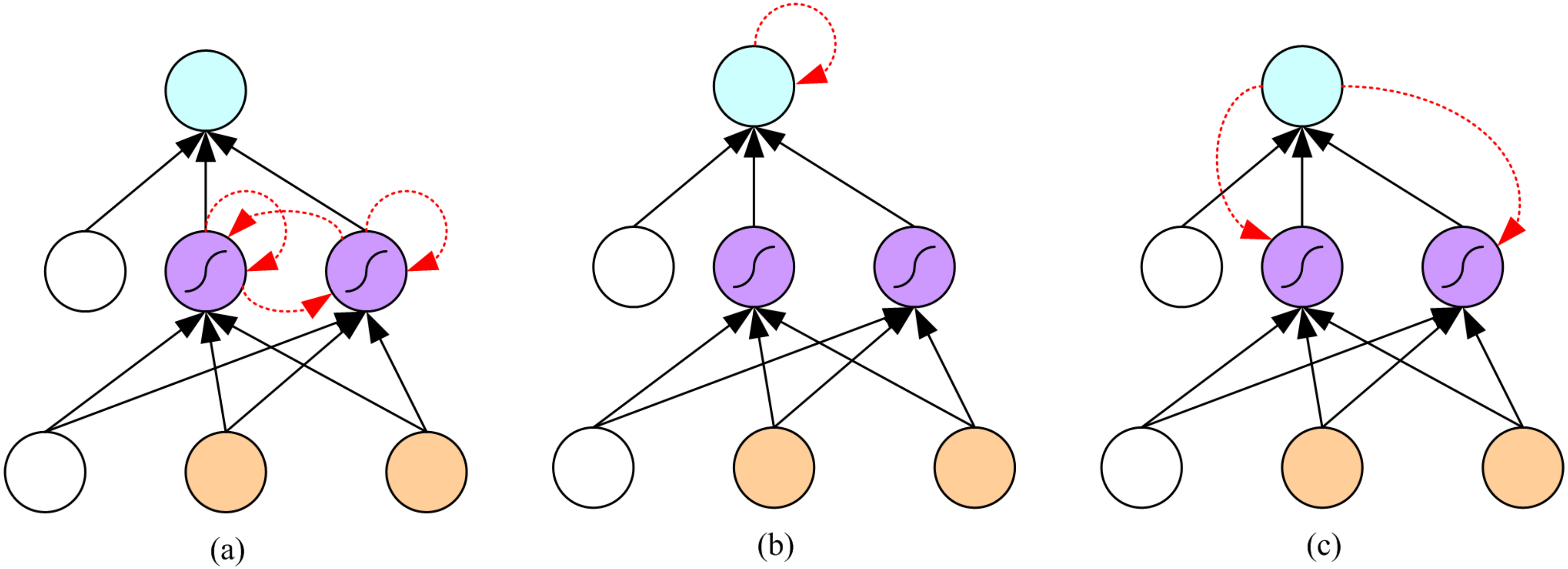
Learning Time

- Applications:
  - Sequence recognition: Speech recognition
  - Sequence reproduction: Time-series prediction
  - Sequence association
- Network architectures
  - Time-delay networks (Waibel et al., 1989)
  - Recurrent networks (Rumelhart et al., 1986)

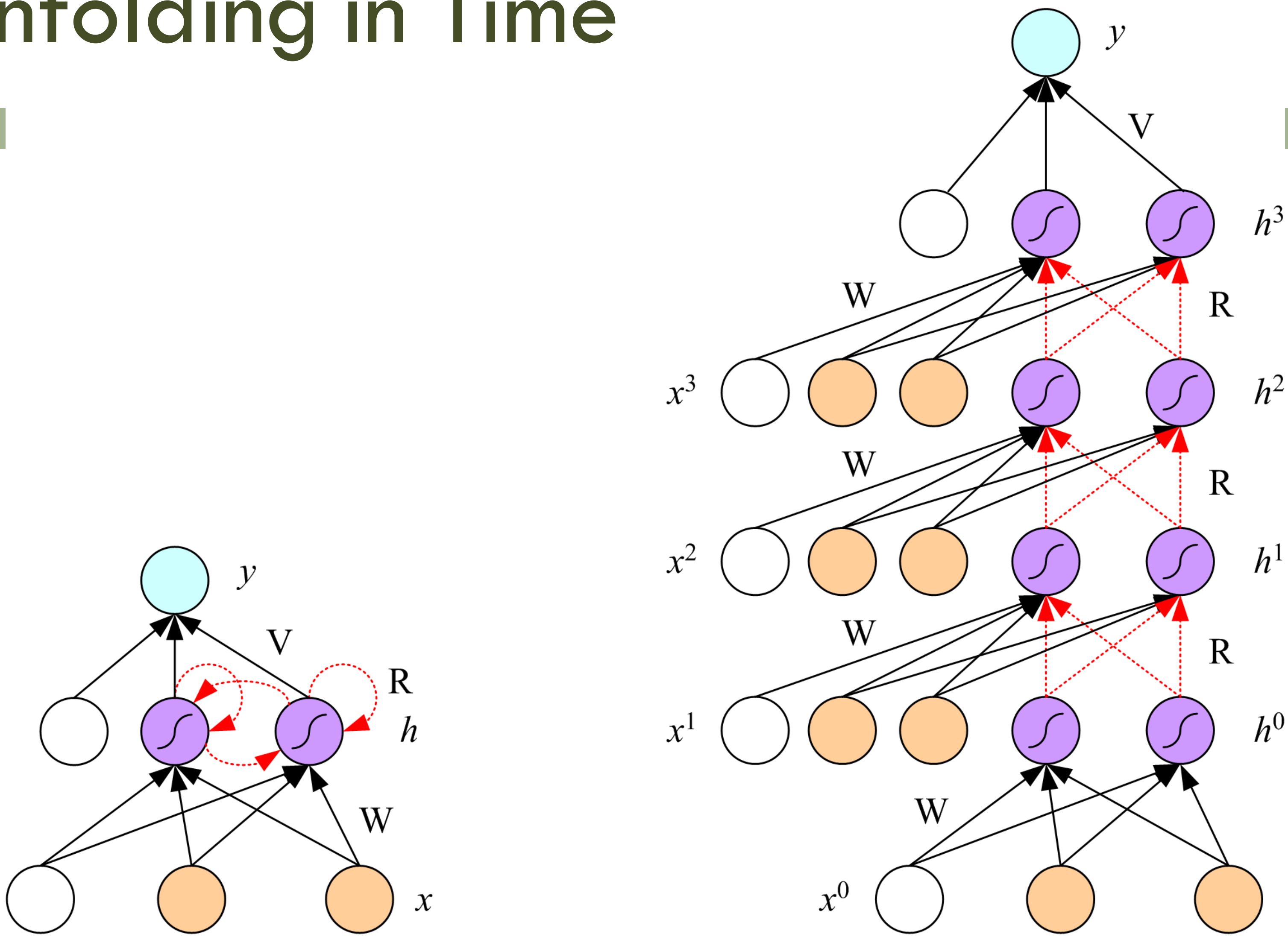
Time-Delay Neural Networks



Recurrent Networks



Unfolding in Time



(a) (b)

Deep Networks

- Layers of feature extraction units
- Can have local receptive fields as in convolution networks, or can be fully connected
- Can be trained layer by layer using an autoencoder in an unsupervised manner
- No need to craft the right features or the right basis functions or the right dimensionality reduction method; learns multiple layers of abstraction all by itself given a lot of data and a lot of computation
- Applications in vision, language processing, …