BLG 527E Machine Learning

FALL 2024-2025

Assoc. Prof. Yusuf Yaslan

Multi Layer Perceptron

Introduction

- Artificial Neural Networks take their inspiration from the brain.
- Our aim is not to understand the brain per se but to build useful machines.

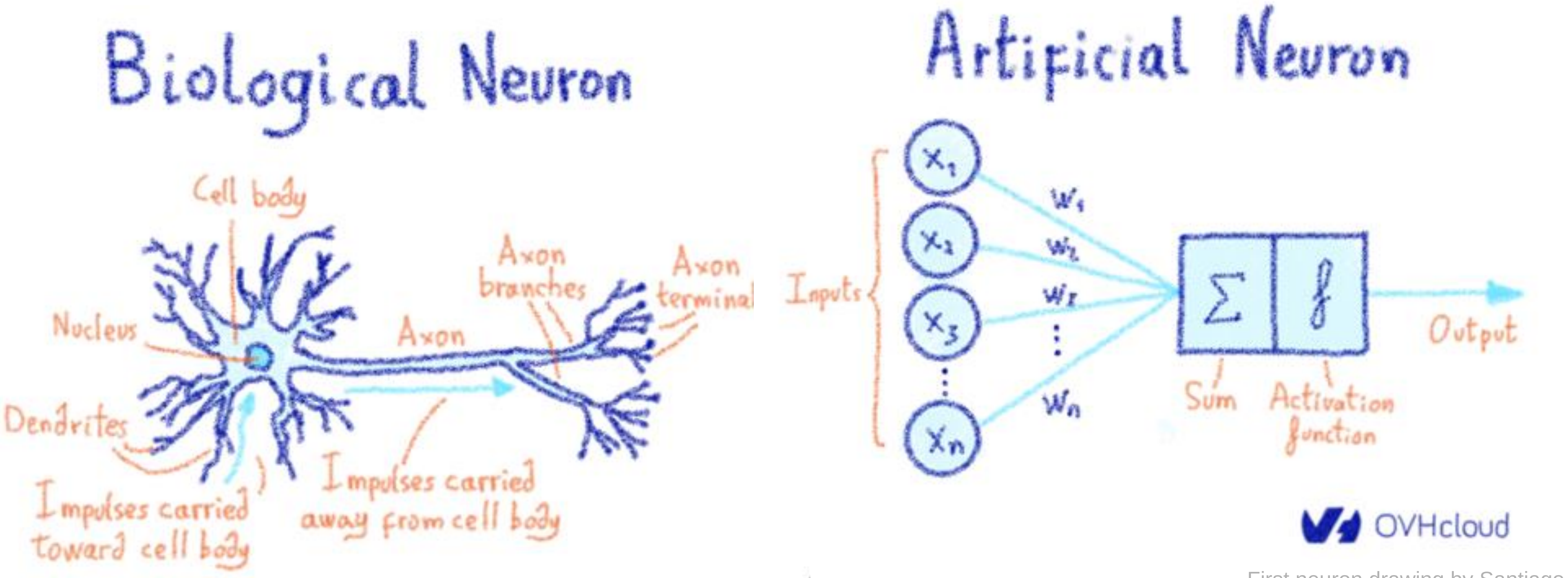
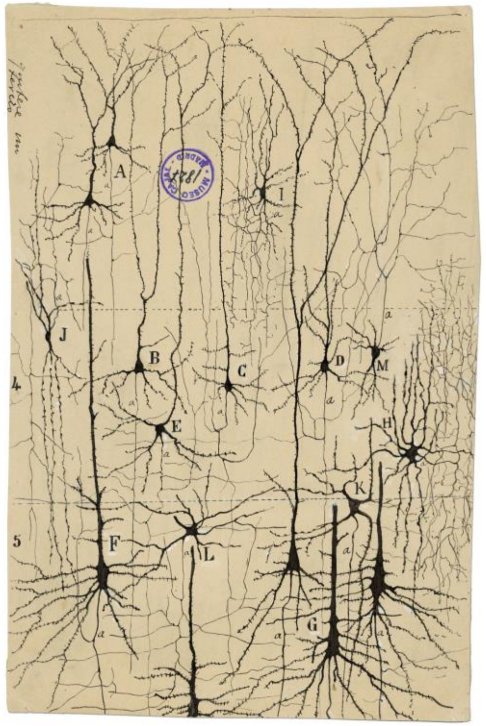


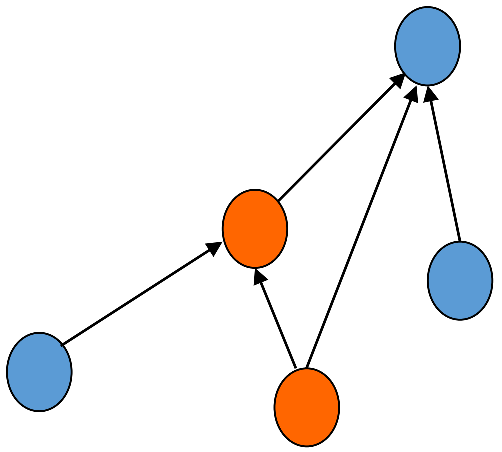
Image from: https://blog.ovhcloud.com/what-does-training-neural-networks-mean/



First neuron drawing by Santiago Ramón y Cajal
Image from: https://www.quantamagazine.org/why-the-first-drawings-of-neurons-were-defaced-20170928/

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) Understanding an information processing system
  1. Computational theory: Corresponds to goal of computation
  2. Representation and algorithm: How the input and output represented
  3. Hardware implementation: Physical realization of the system
- One example is sorting: The computational theory is to order a given set of elements. The representation may use integers, and the algorithm may be Quicksort. After compilation, the executable code for a particular processor sorting integers represented in binary is one hardware implementation.

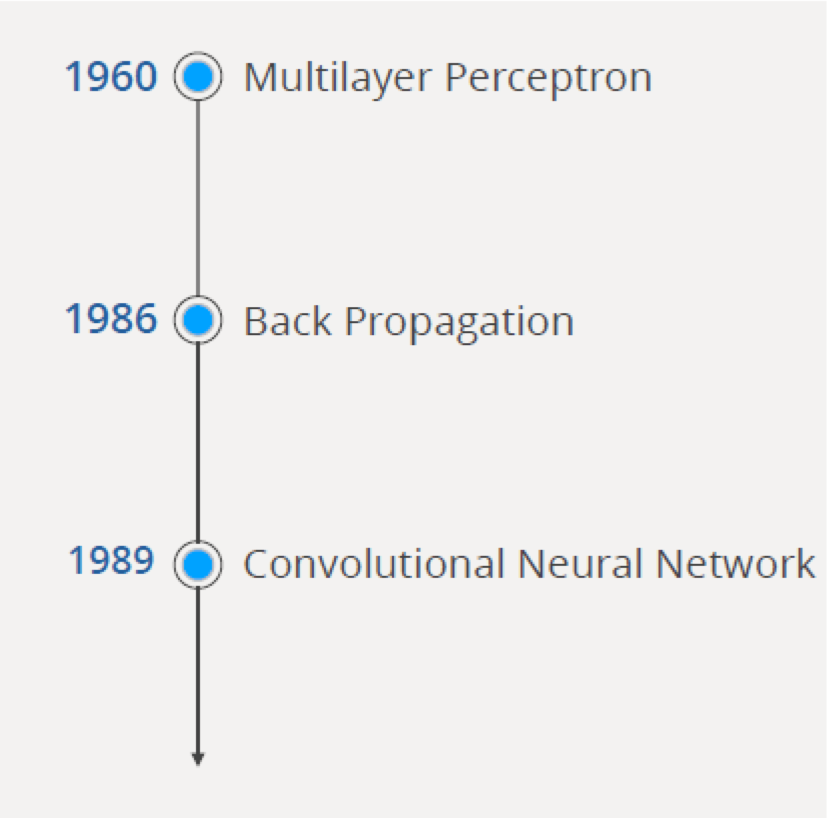
Understanding the Brain

- The brain is one hardware implementation for learning or pattern recognition.
- Reverse engineering: From hardware to theory
- Parallel processing: SIMD vs MIMD
    - In single instruction, multiple data (SIMD) machines, all processors execute the same instruction but on different pieces of data.
    - In multiple instruction, multiple data (MIMD) machines, different processors may execute different instructions on different data.

Neural net: SIMD with modifiable local memory

Learning: Update by training/experience

The Seasons of Neural Networks

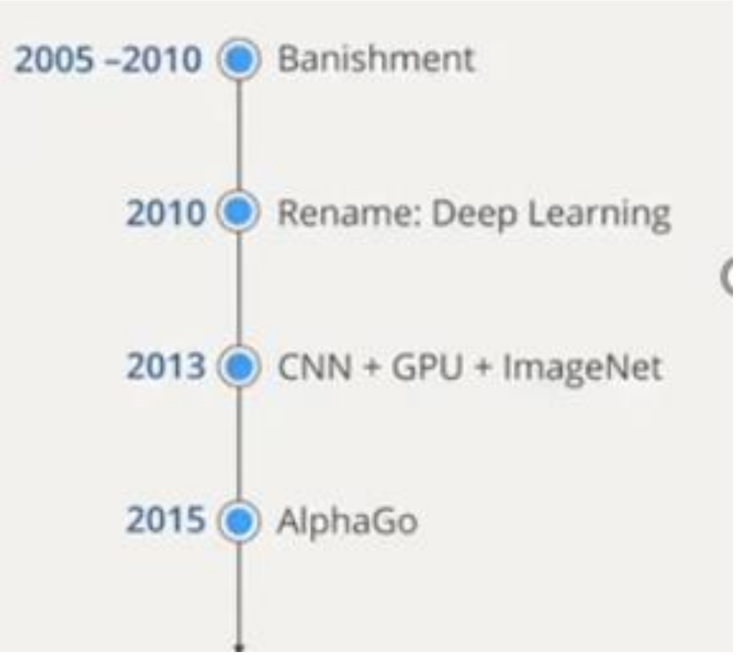


This slide is adopted from the following course: Coursera Introduction to Machine Learning Course by Duke University

The Seasons of Neural Networks

- 1990 –1994 Neural Nets in the Wild
- 1995 Long Short-Term Memory
- 1998 – 2005 More Neural Nets in the Wild

The Seasons of Neural Networks

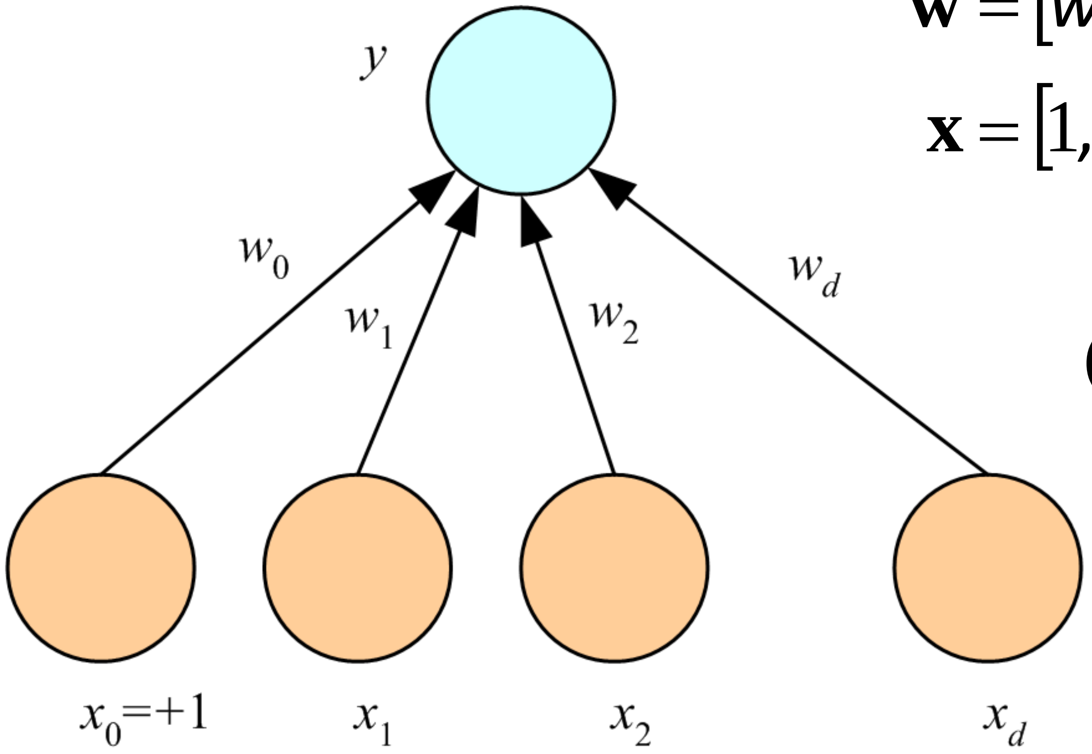


Perceptron

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

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

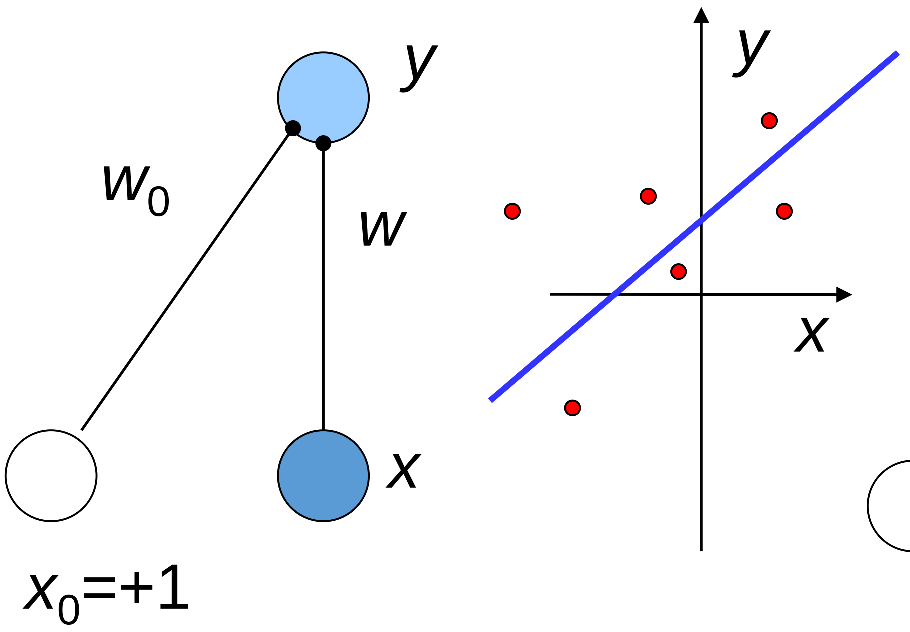
$$\mathbf{x} = [1, x_1, \ldots, x_d]^T$$



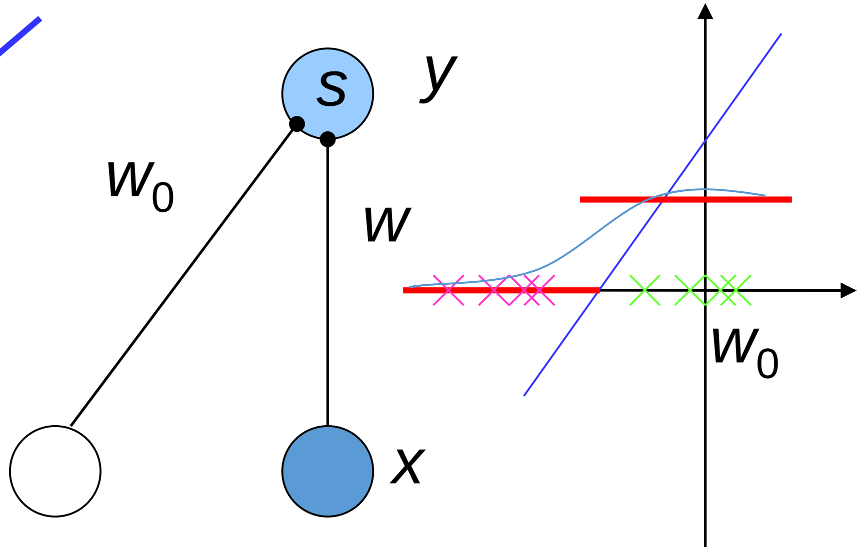
(Rosenblatt, 1962)

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:

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

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

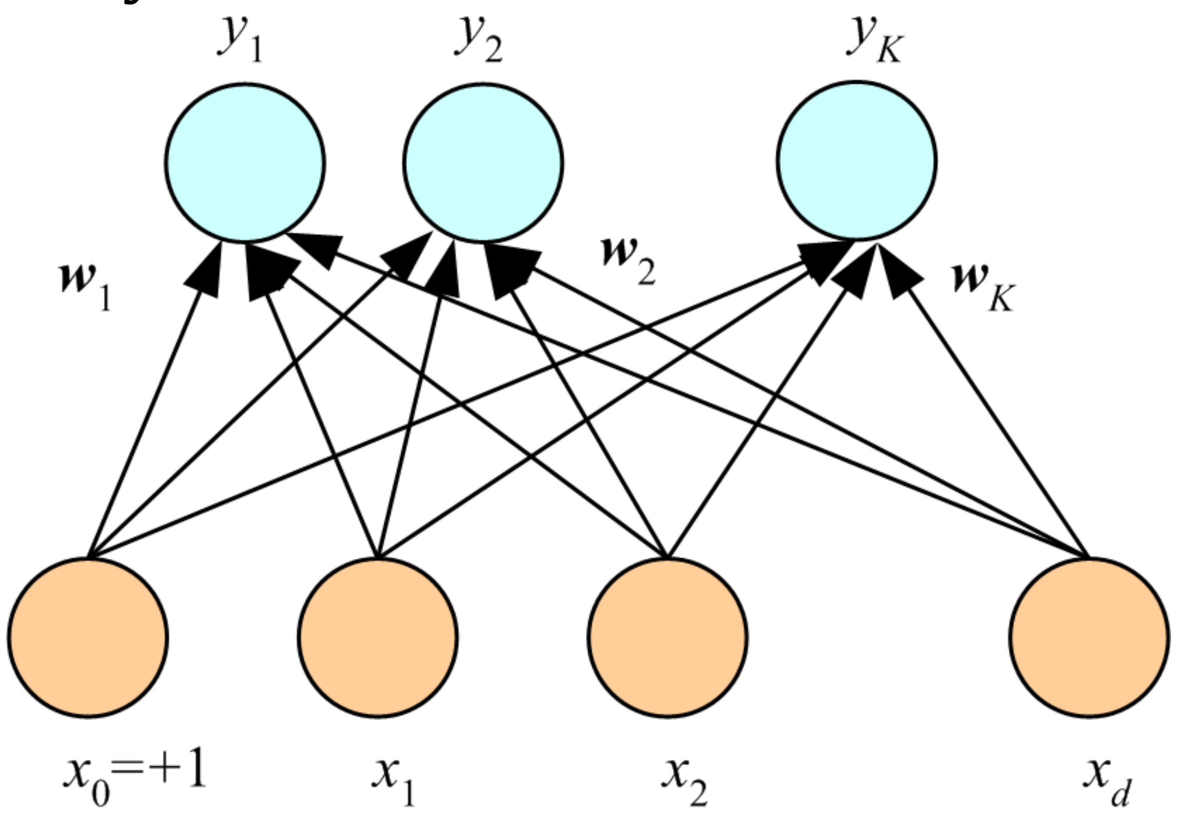
Classification:

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

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

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

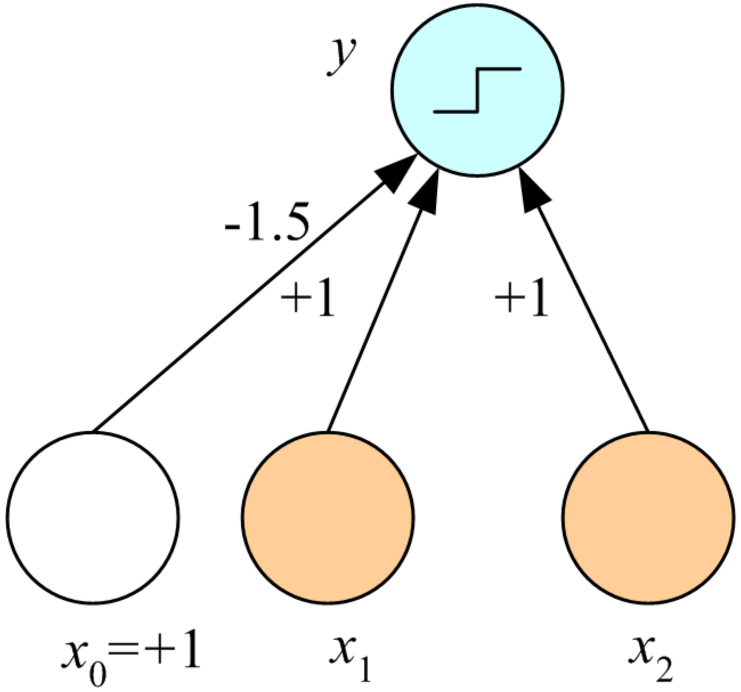
$$y^t = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}^t\right)$$
$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, \mathbf{r}^t\right) = -r^t \log y^t - \left(1 - r^t\right) \log\left(1 - y^t\right)$$
$$\Delta w_j^t = \eta\left(r^t - y^t\right) x_j^t$$

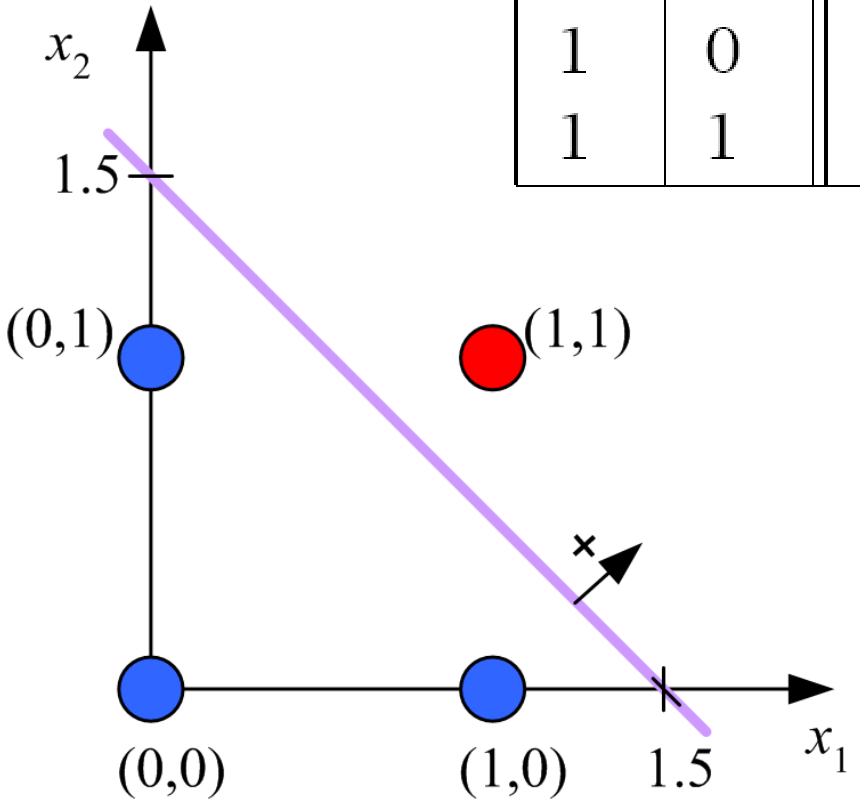
- K>2 softmax outputs

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

Learning Boolean AND

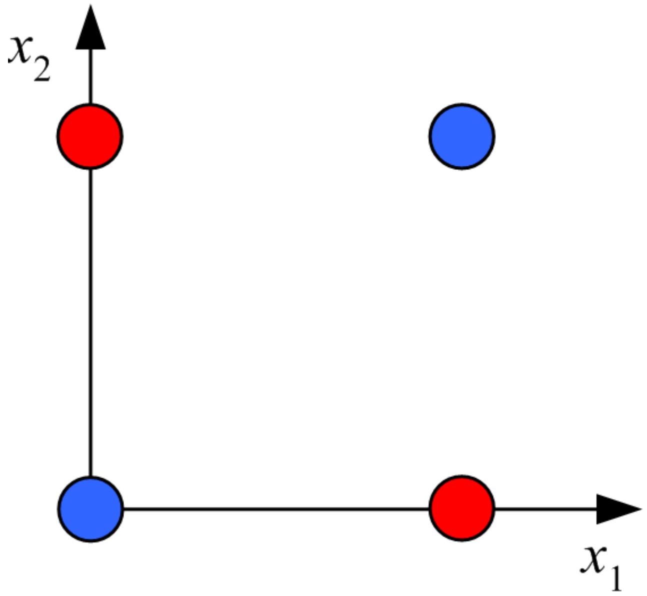
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





XOR

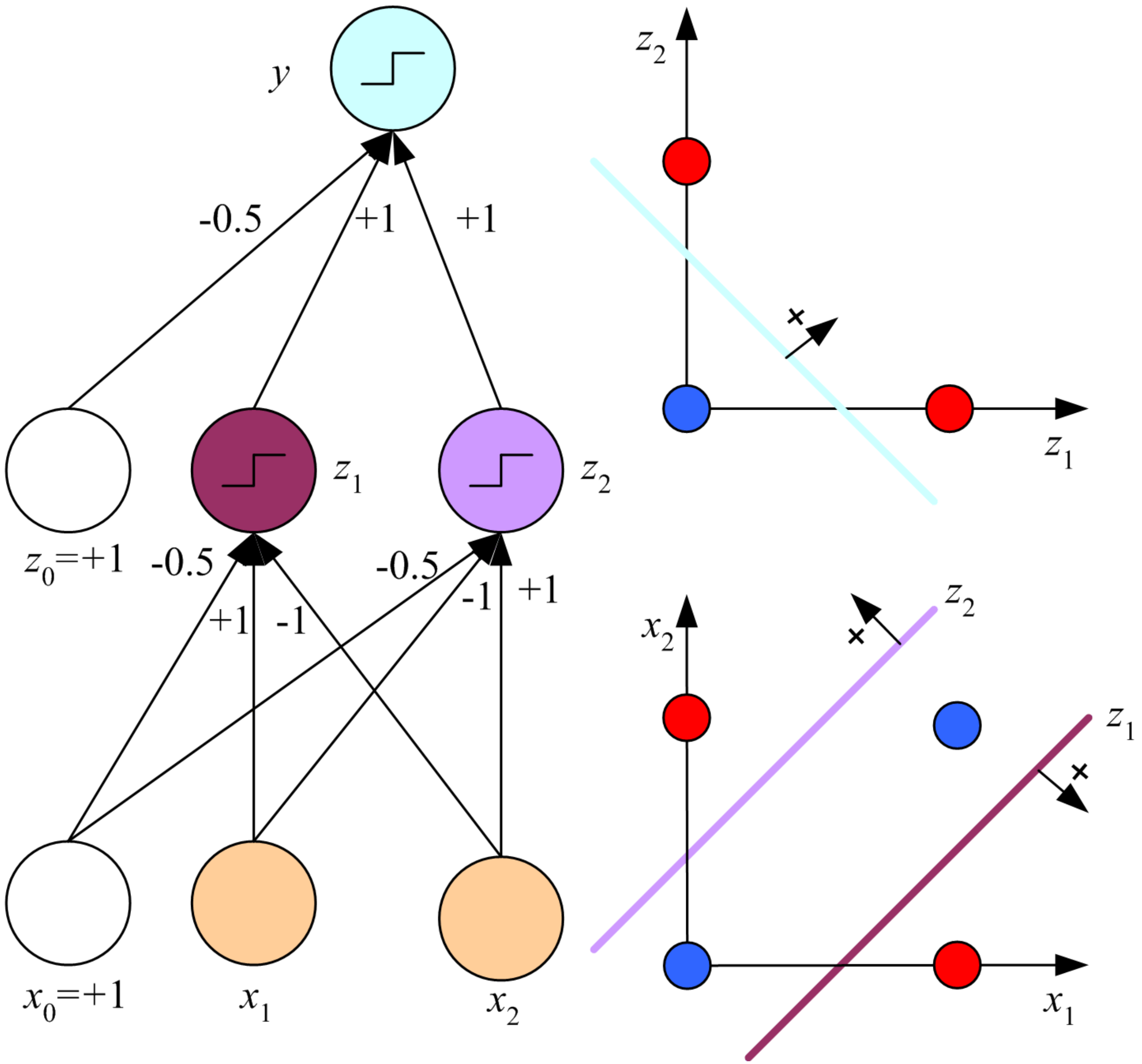
x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



- No w_0, w_1, w_2 satisfy:

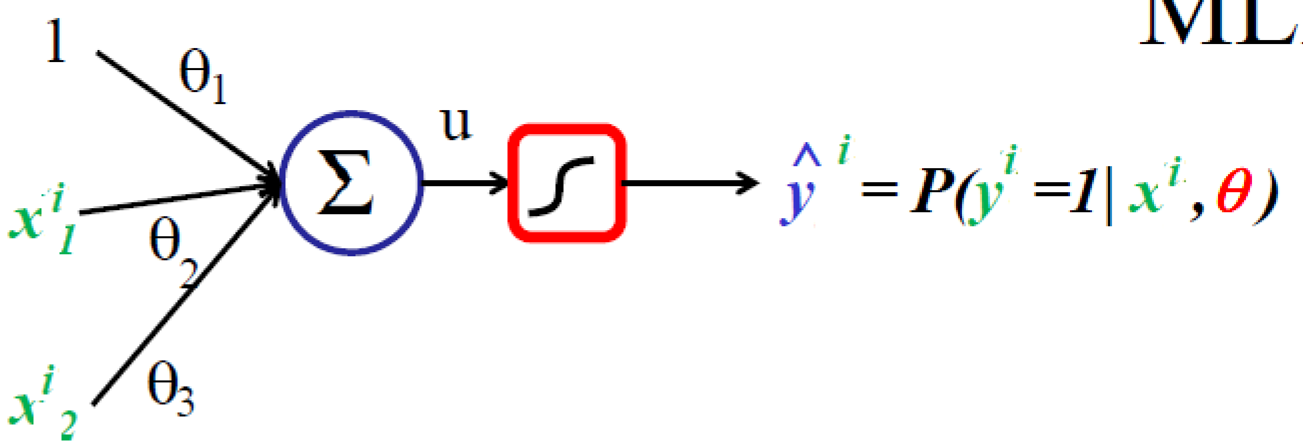
$$\begin{aligned} w_0 &\le 0 \\ w_2 + w_0 &> 0 \\ w_1 + w_0 &> 0 \\ w_1 + w_2 + w_0 &\le 0 \end{aligned}$$

(Minsky and Papert, 1969)

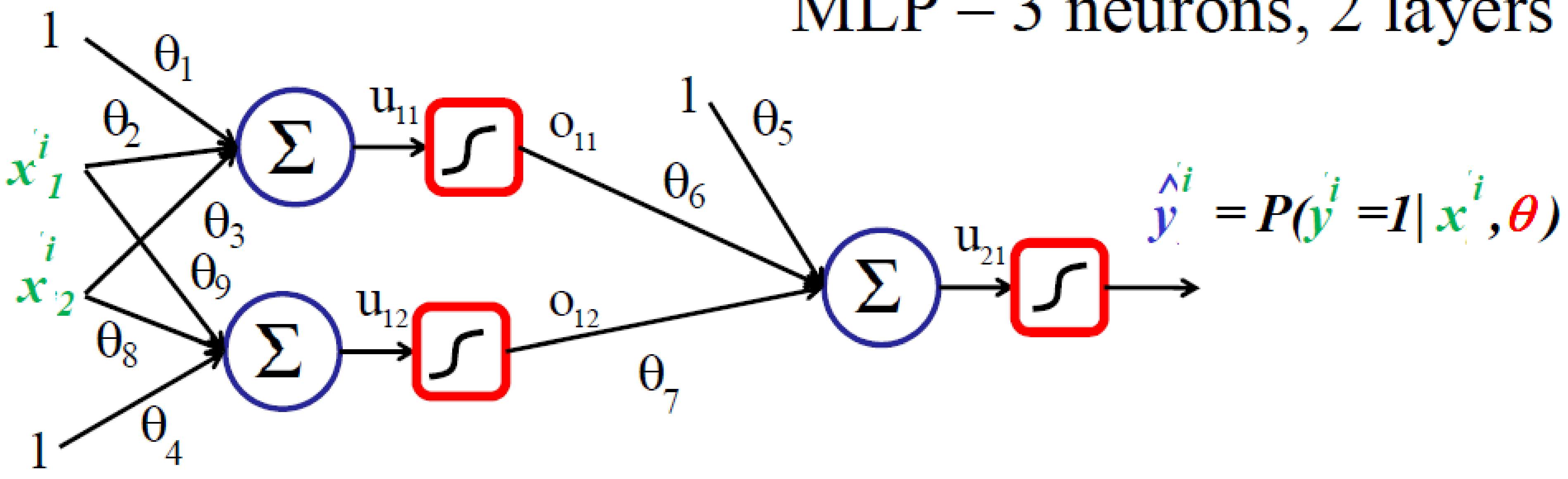


x_1 XOR x_2 = (x_1 AND ~x_2) OR (~x_1 AND x_2)

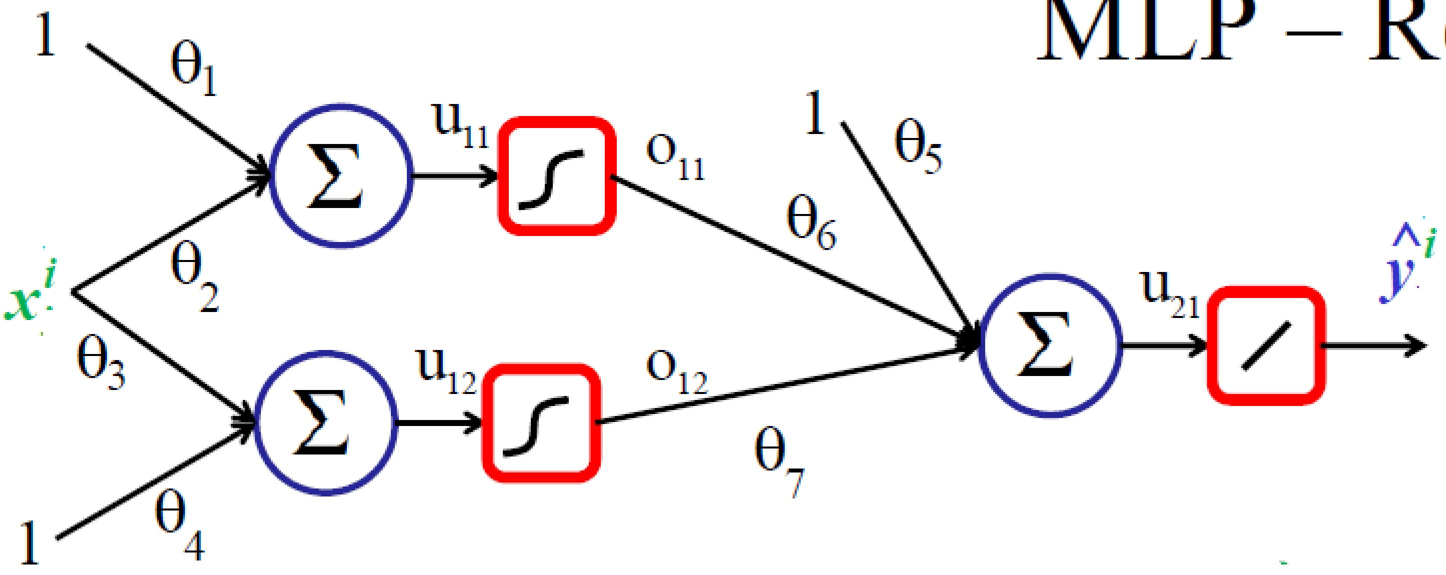
MLP – 1 neuron



MLP – 3 neurons, 2 layers



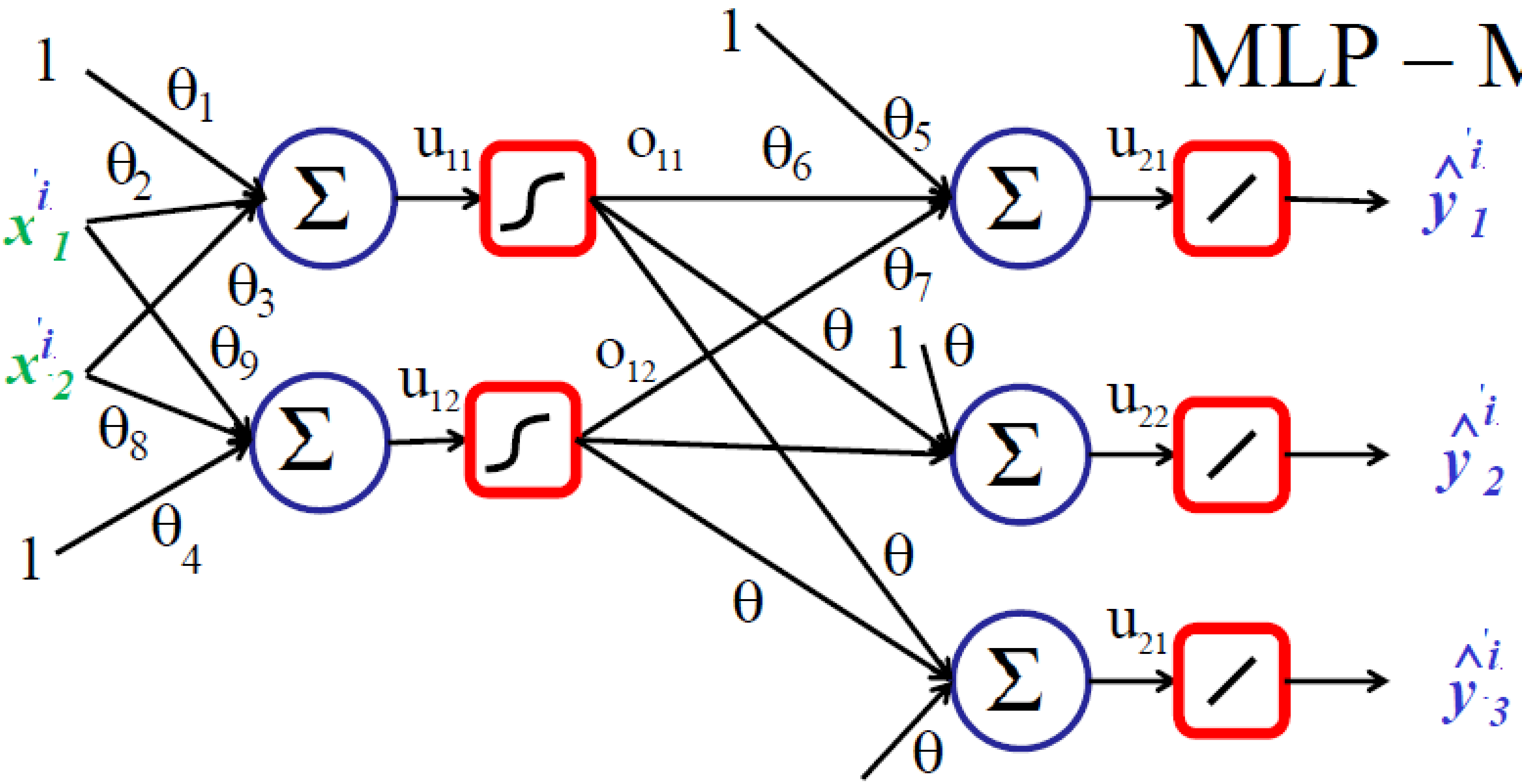
MLP – Regression



$\hat{y}^i = \mathbf{E}(y^i \mid x^i, \theta)$

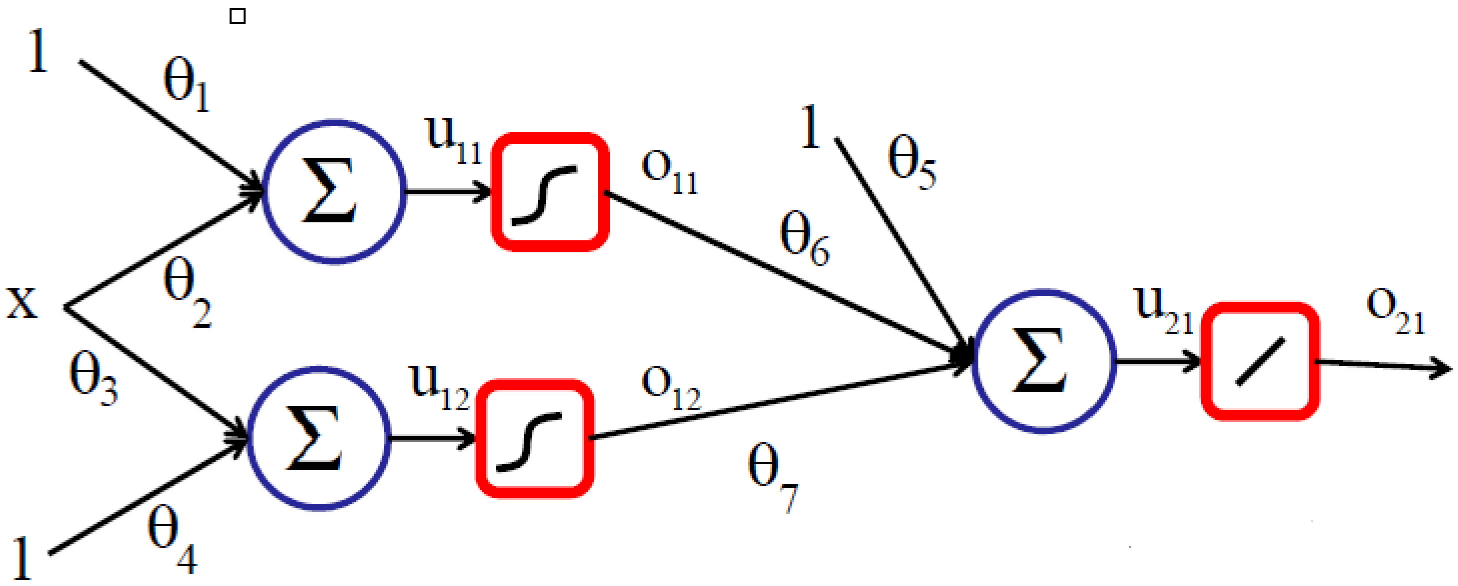
This slide is obtained from Nando Freitas Lecture Notes

MLP – Multiclass

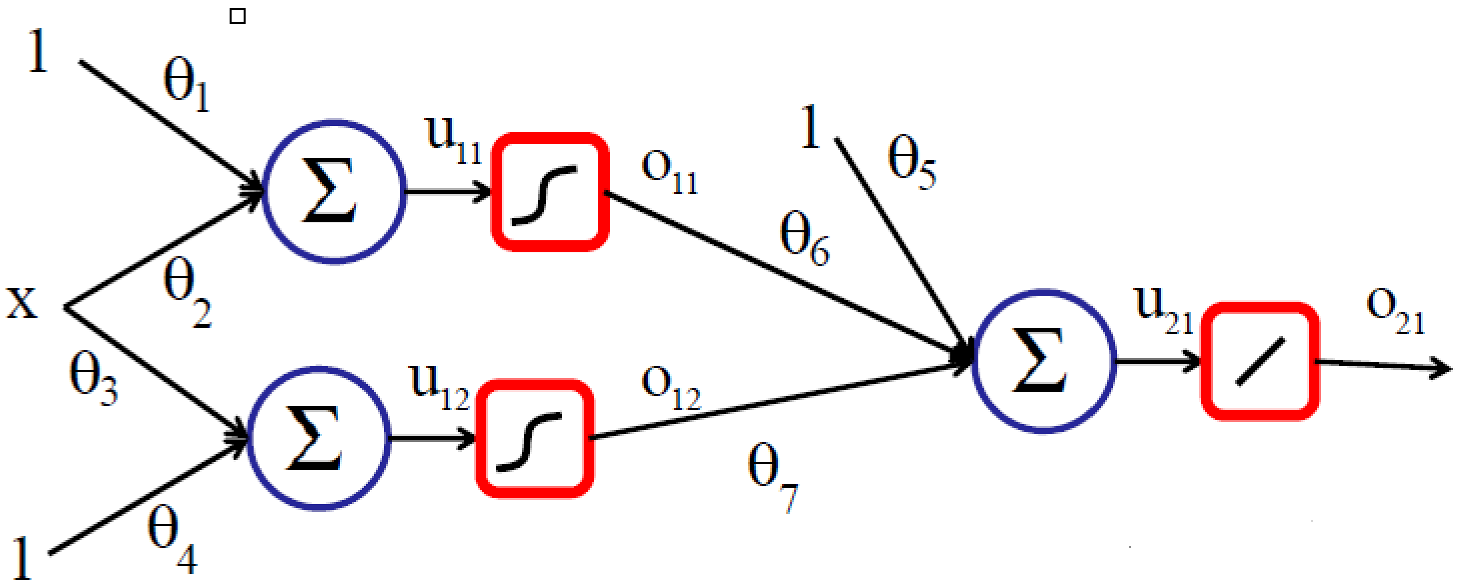


This slide is obtained from Nando Freitas Lecture Notes

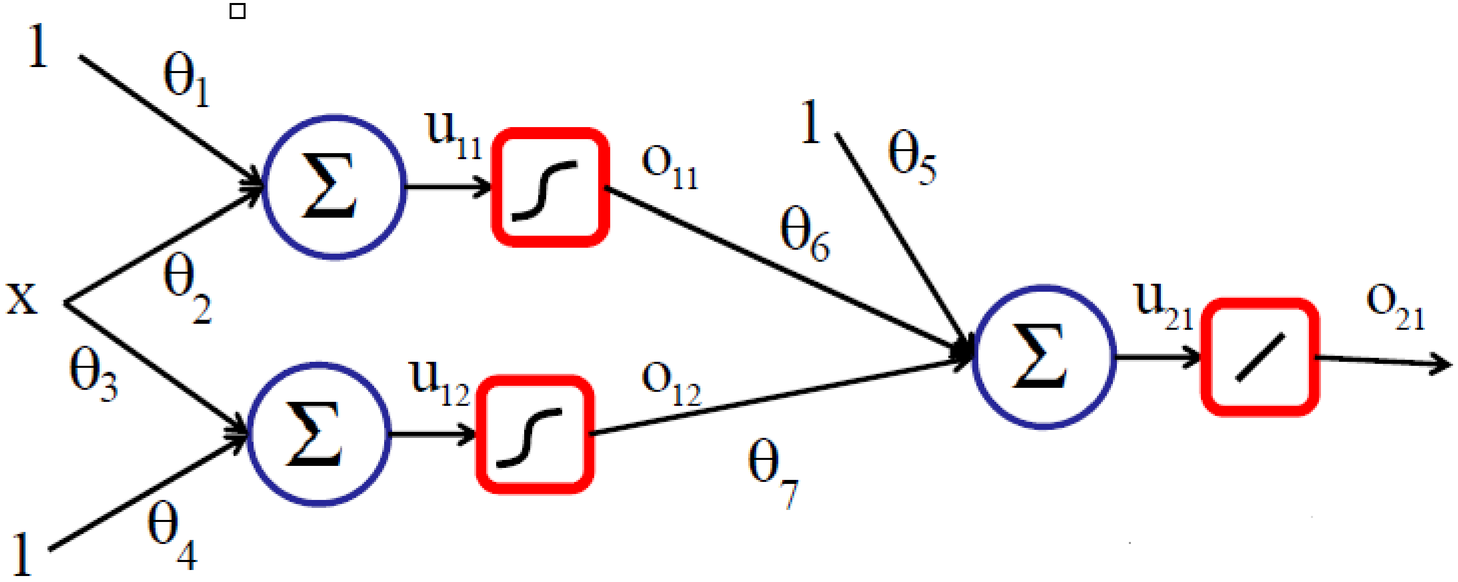
Backpropagation



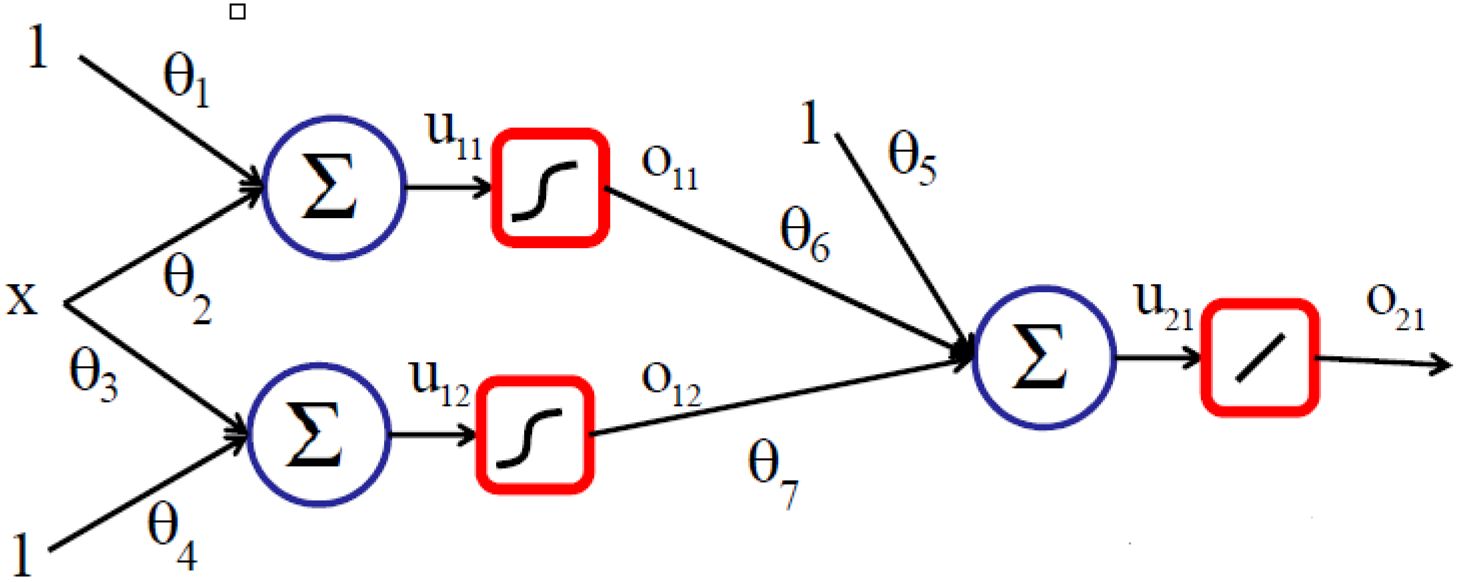
Backpropagation



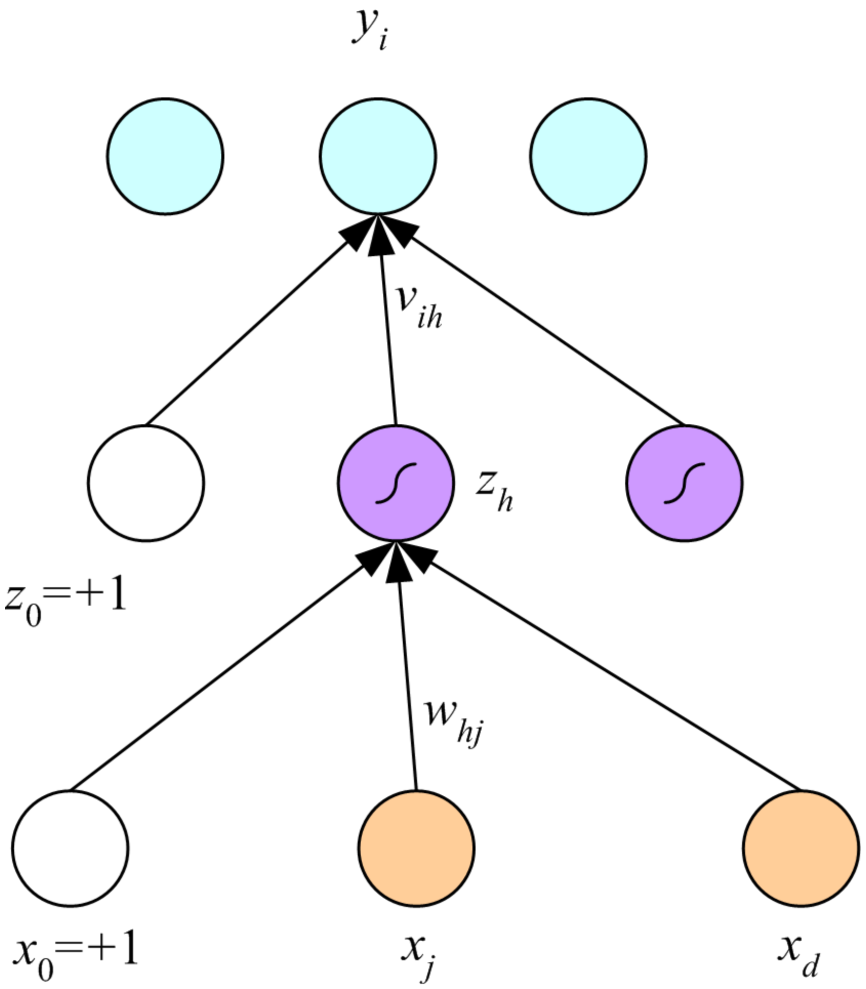
Backpropagation



Backpropagation



Multilayer Perceptrons

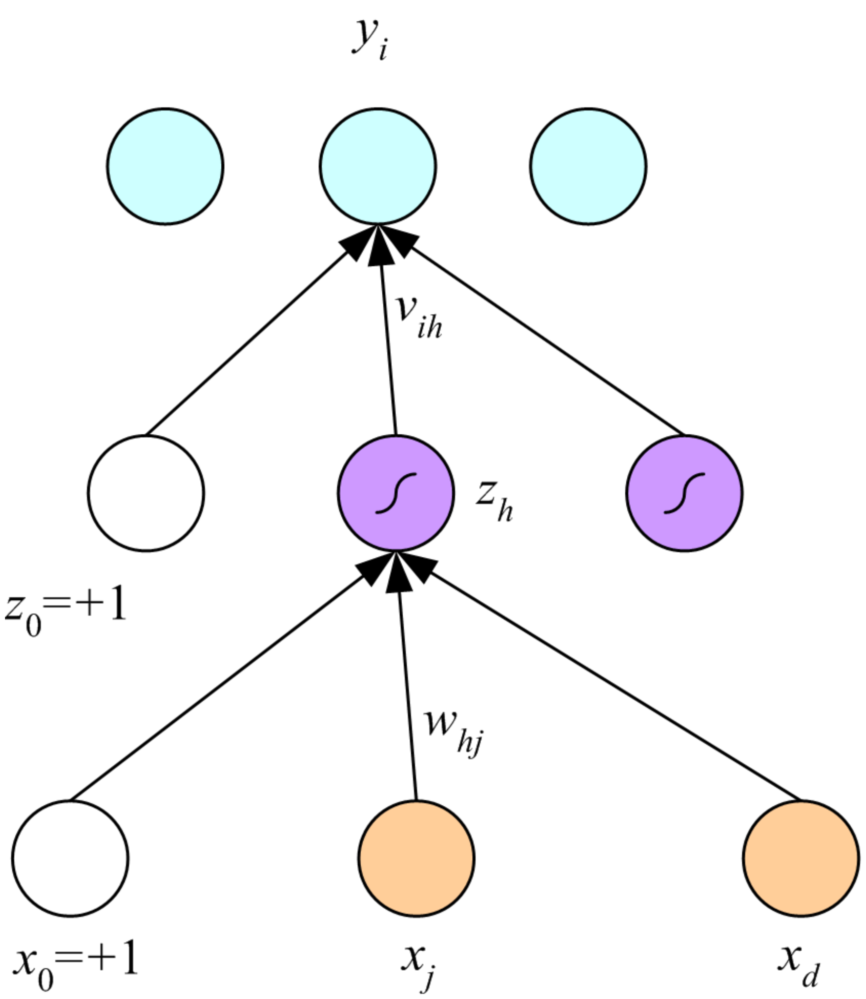


$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

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

(Rumelhart et al., 1986)

Backpropagation

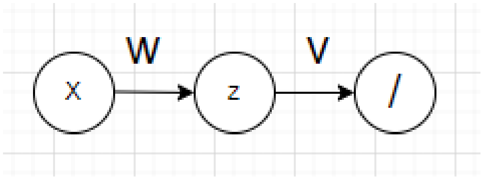


$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

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

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

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


W
V
x
z
/

Regression

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

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

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

Backward

Forward

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

$$\boldsymbol{x}$$

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

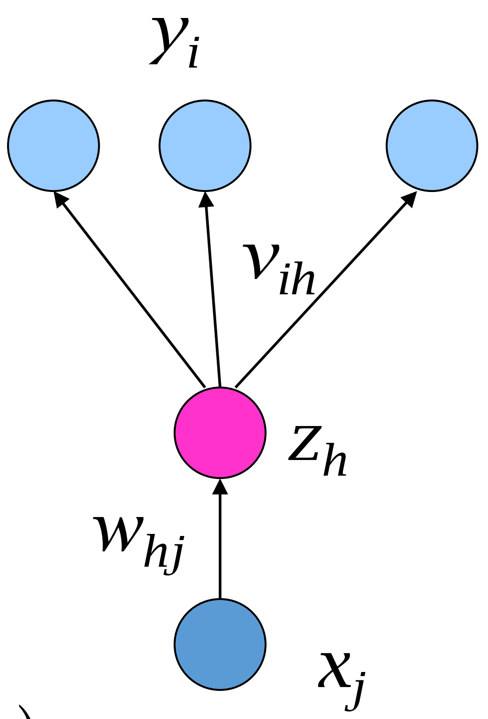
Regression with Multiple Outputs

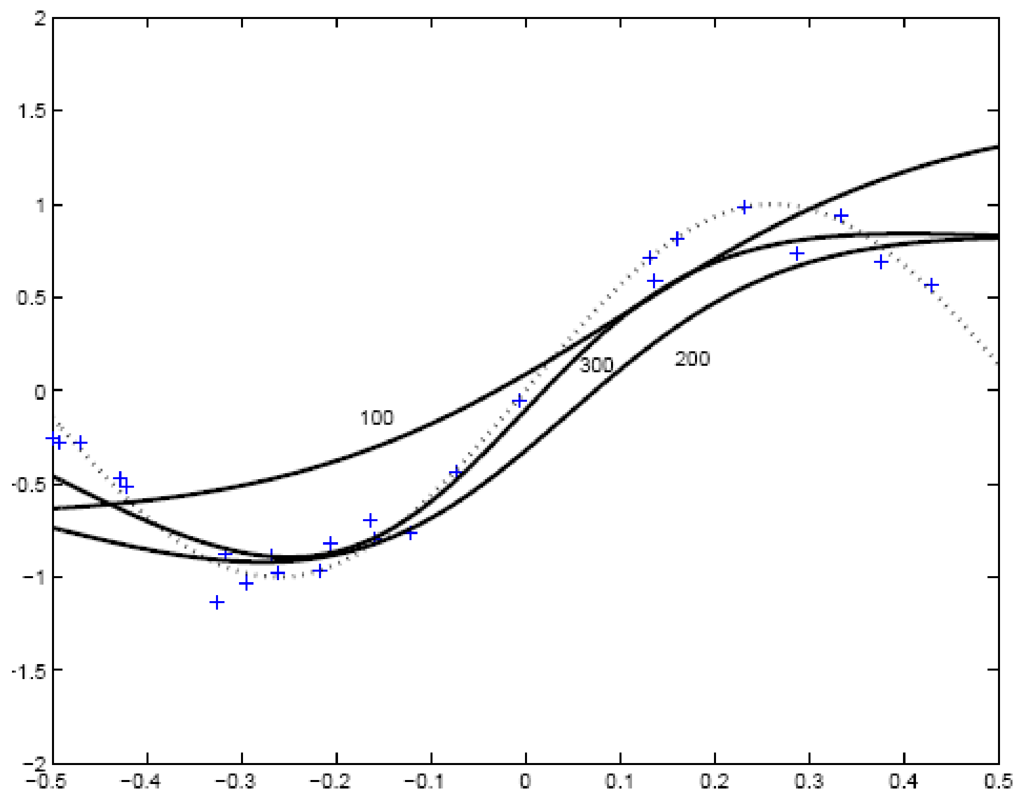
$$E(\mathbf{W},\mathbf{V} \mid \mathcal{X}) = \frac{1}{2}\sum_t \sum_i \left(r_i^t - y_i^t\right)^2$$

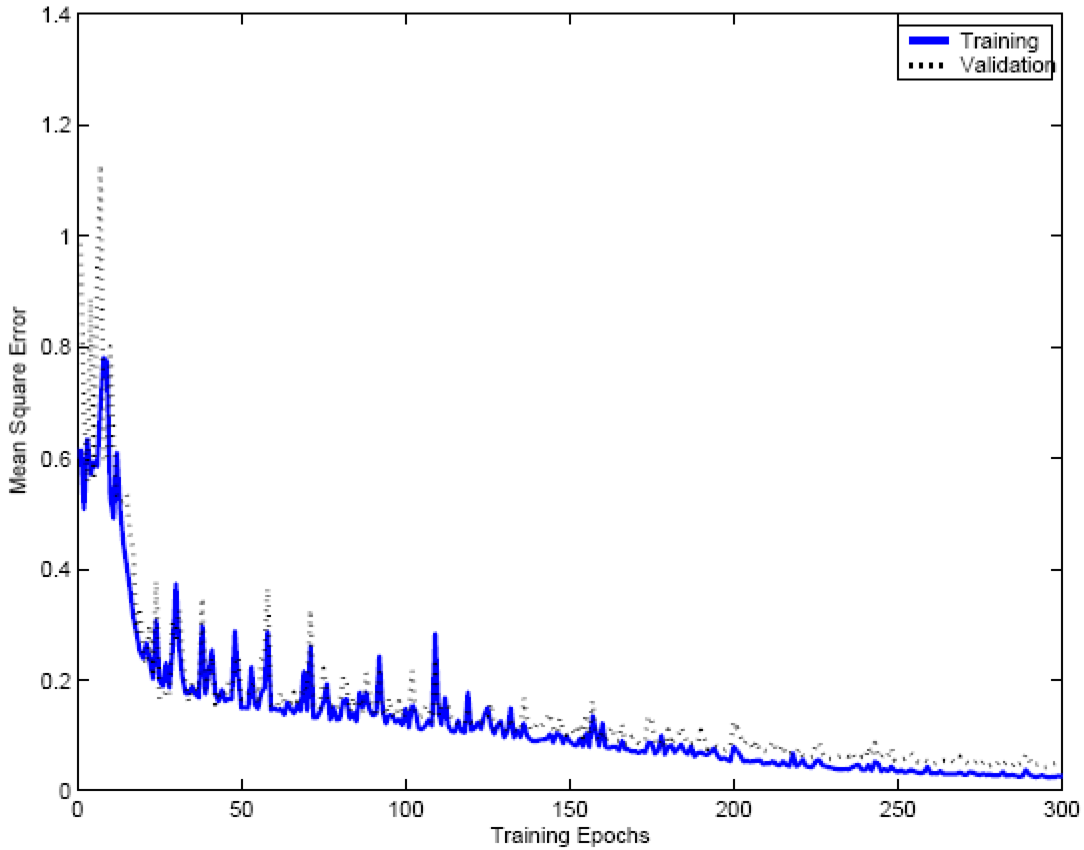
$$y_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0}$$

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

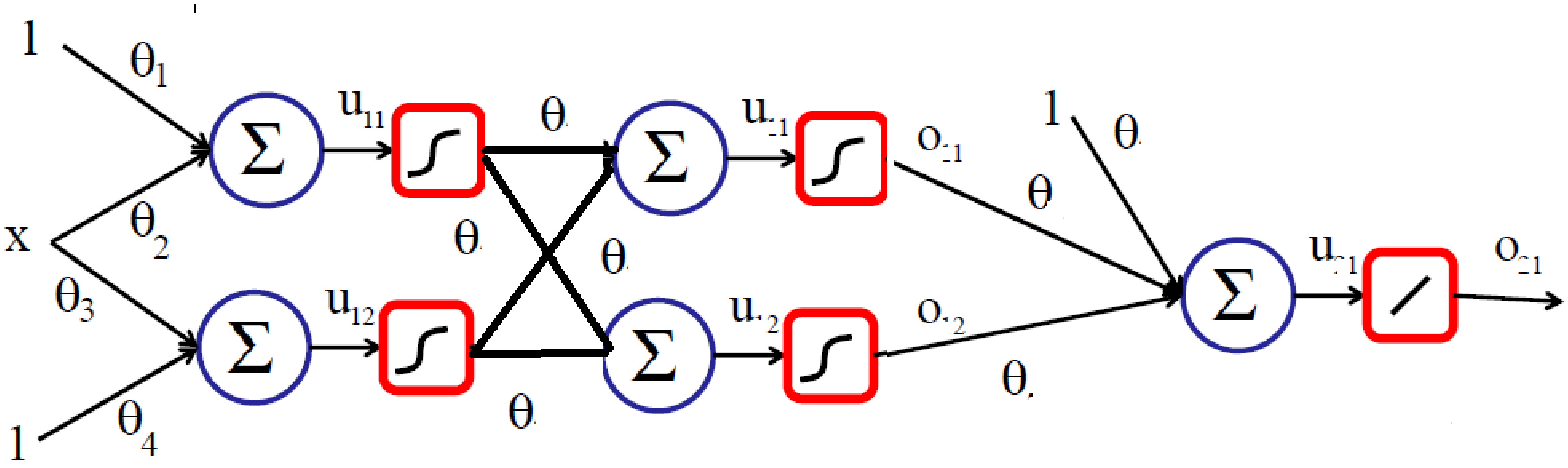
$$\Delta w_{hj} = \eta \sum_t \left[\sum_i \left(r_i^t - y_i^t\right) v_{ih}\right] z_h^t \left(1 - z_h^t\right) x_j^t$$

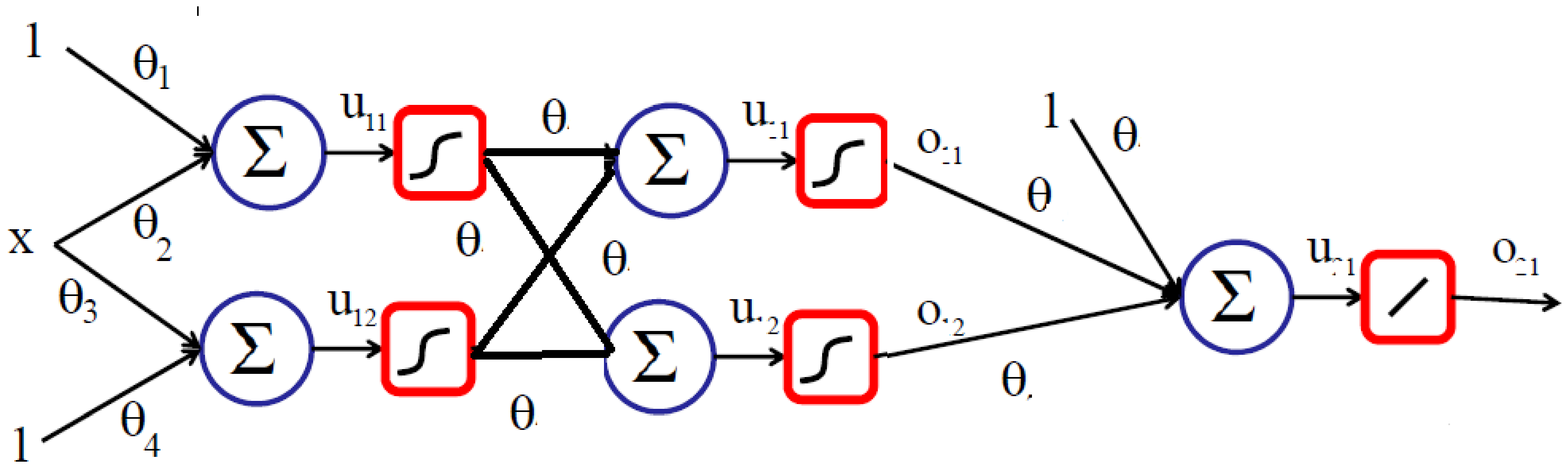



100
200
300

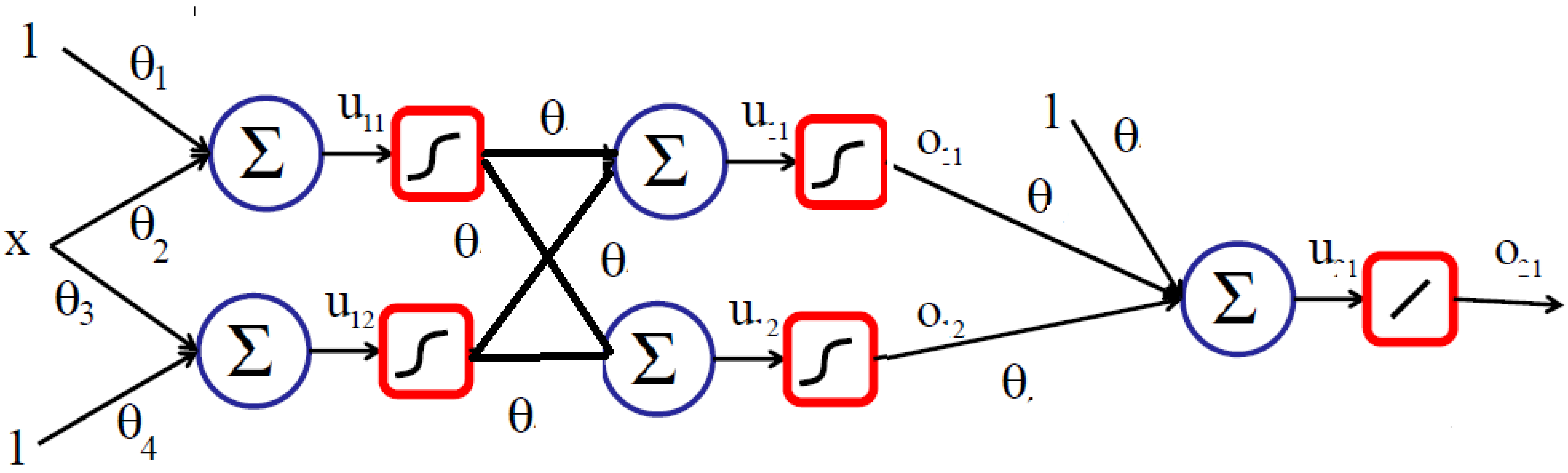

Training
Validation
Mean Square Error
Training Epochs

MLP with 2 Hidden Layer

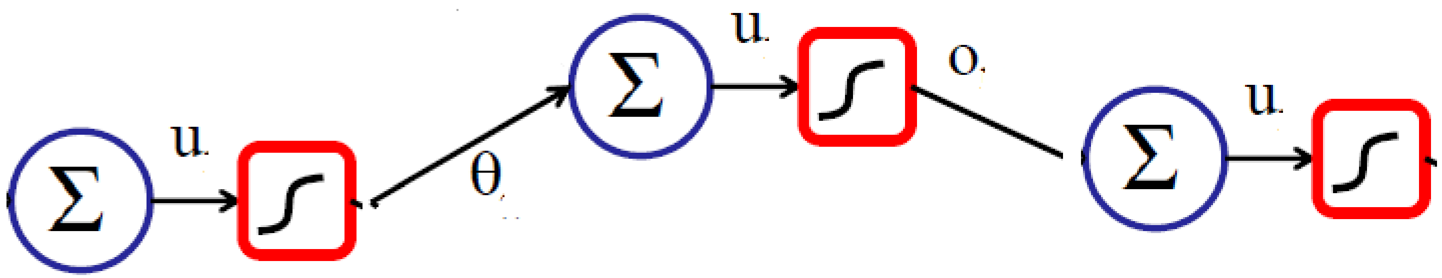




This slide is adopted from Nando Freitas Lecutre Notes



This slide is adopted from Nando Freitas Lecutre Notes


u
θ
u
o
u

Improving Convergence

- Momentum: At each parameter update, successive Δ*wt i* values may be so different that large oscillations may occur and slow convergence. *t* is the time index that is the epoch number in batch learning and the iteration number in online learning.

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

Improving Convergence

Adaptive learning rate: In gradient descent, the learning factor η determines the magnitude of change to be made in the parameter. It is generally taken between 0.0 and 1.0, mostly less than or equal to 0.2. It can be made adaptive for faster convergence, where it is kept large when learning takes place and is decreased when learning slows down

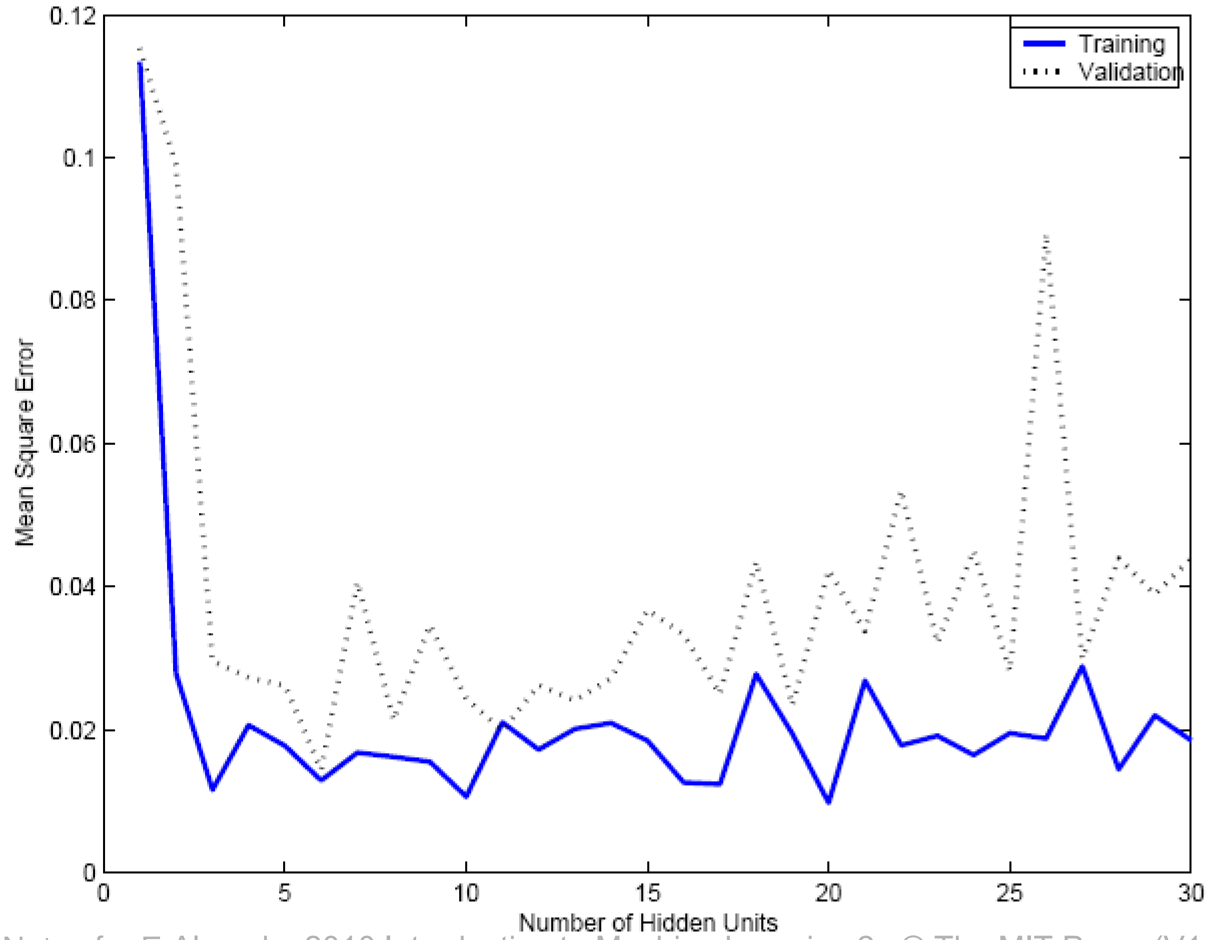
$$\Delta\eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^{t} \\ -b\eta & \text{otherwise} \end{cases}$$

Overfitting/Overtraining

- We know from previous chapters that an overcomplex model memorizes the noise in the training set and does not generalize to the validation set.
- Similarly in an MLP, when the number of hidden units is large, the generalization accuracy deteriorates

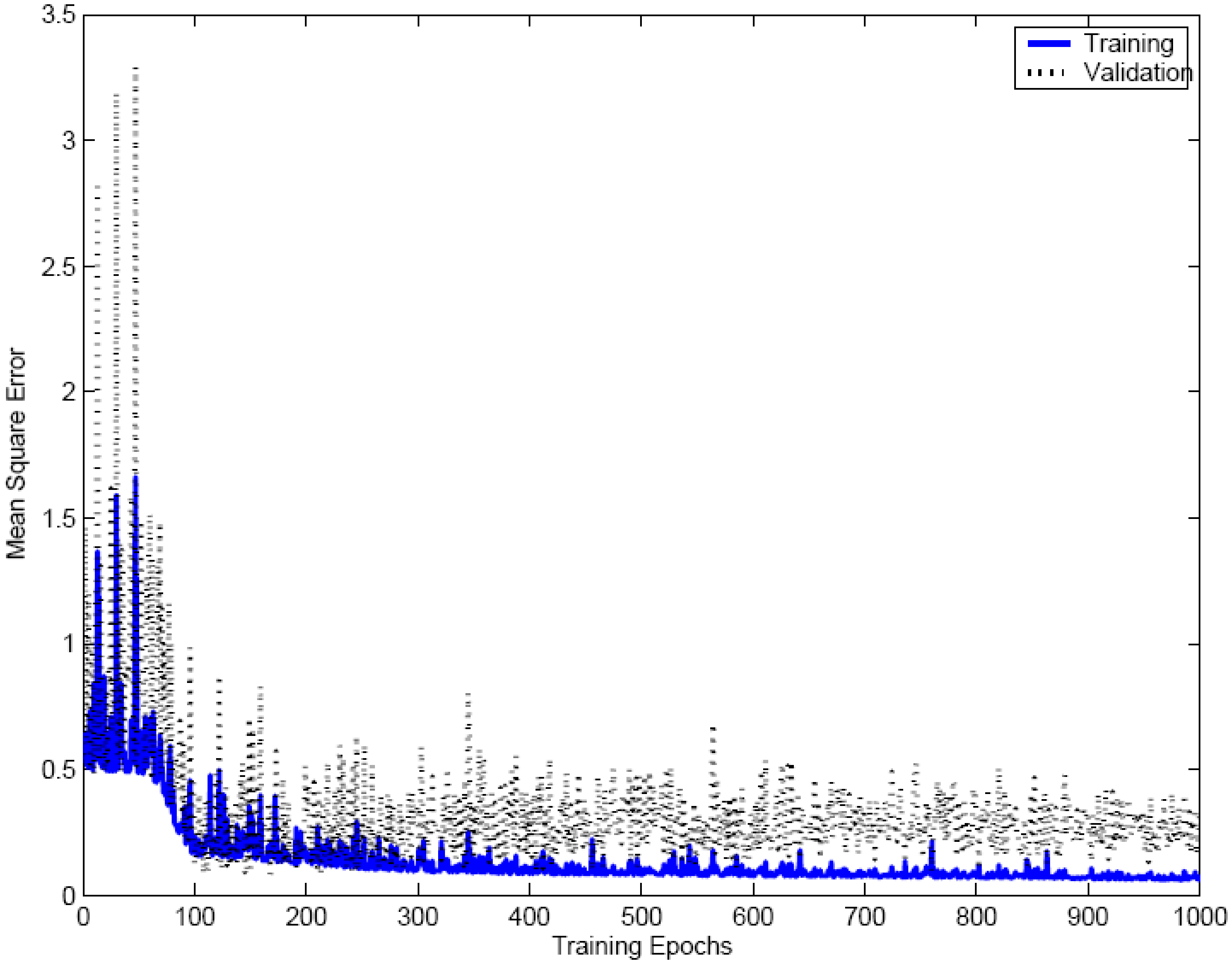
Overfitting/Overtraining

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



Overfitting/Overtraining

- A similar behavior happens when training is continued too long: As
- more training epochs are made, the error on the training set decreases, but the error on the validation set starts to increase beyond a certain point.
- Early stopping: Learning should be *stopped early* to alleviate this problem of *overtraining*.



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

Tuning the Network Size

- To find the optimal network size, the most common approach is to try many different architectures, train them all on the training set, and choose the one that generalizes best to the validation set.
- Another approach is to incorporate this *structural adaptation* into the learning algorithm.
- In the *destructive* approach, we start with a large network and gradually remove units and/or connections that are not necessary
- In the *constructive* approach, we start with a small network and gradually add units and/or connections to improve performance.

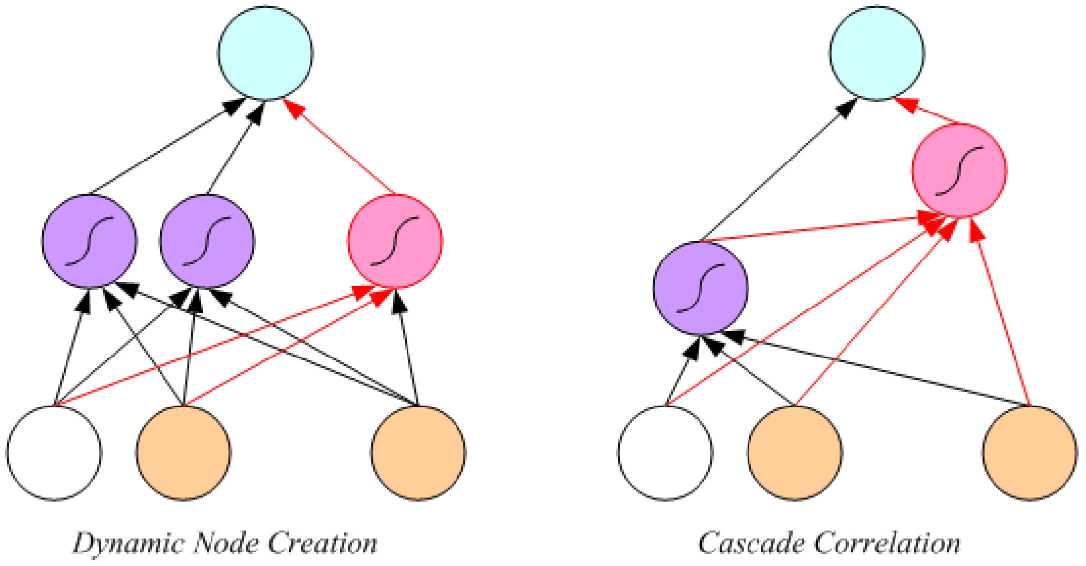
Tuning the Network Size

- Destructive
- Weight decay:

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

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

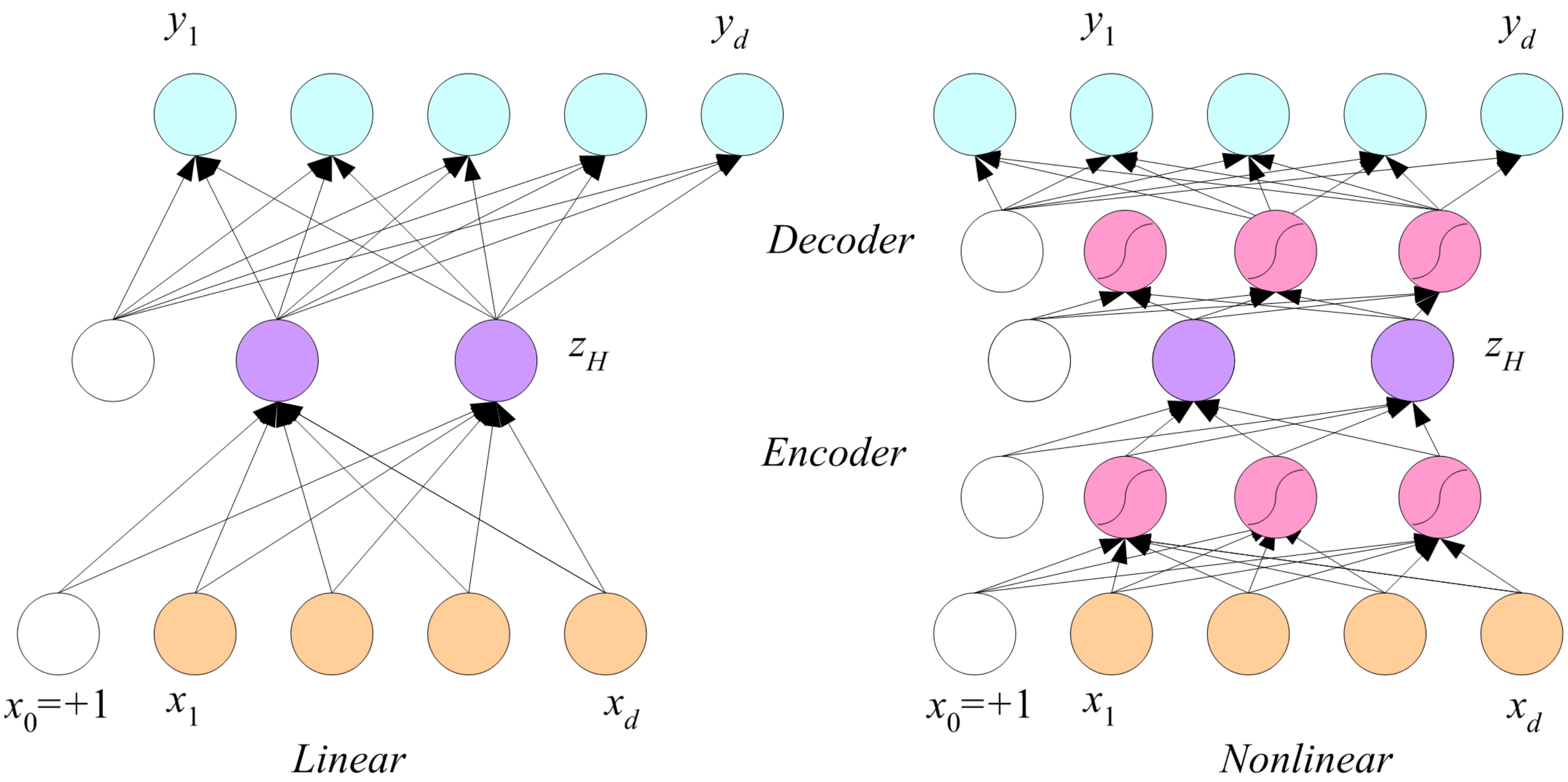
- Constructive
- Growing networks

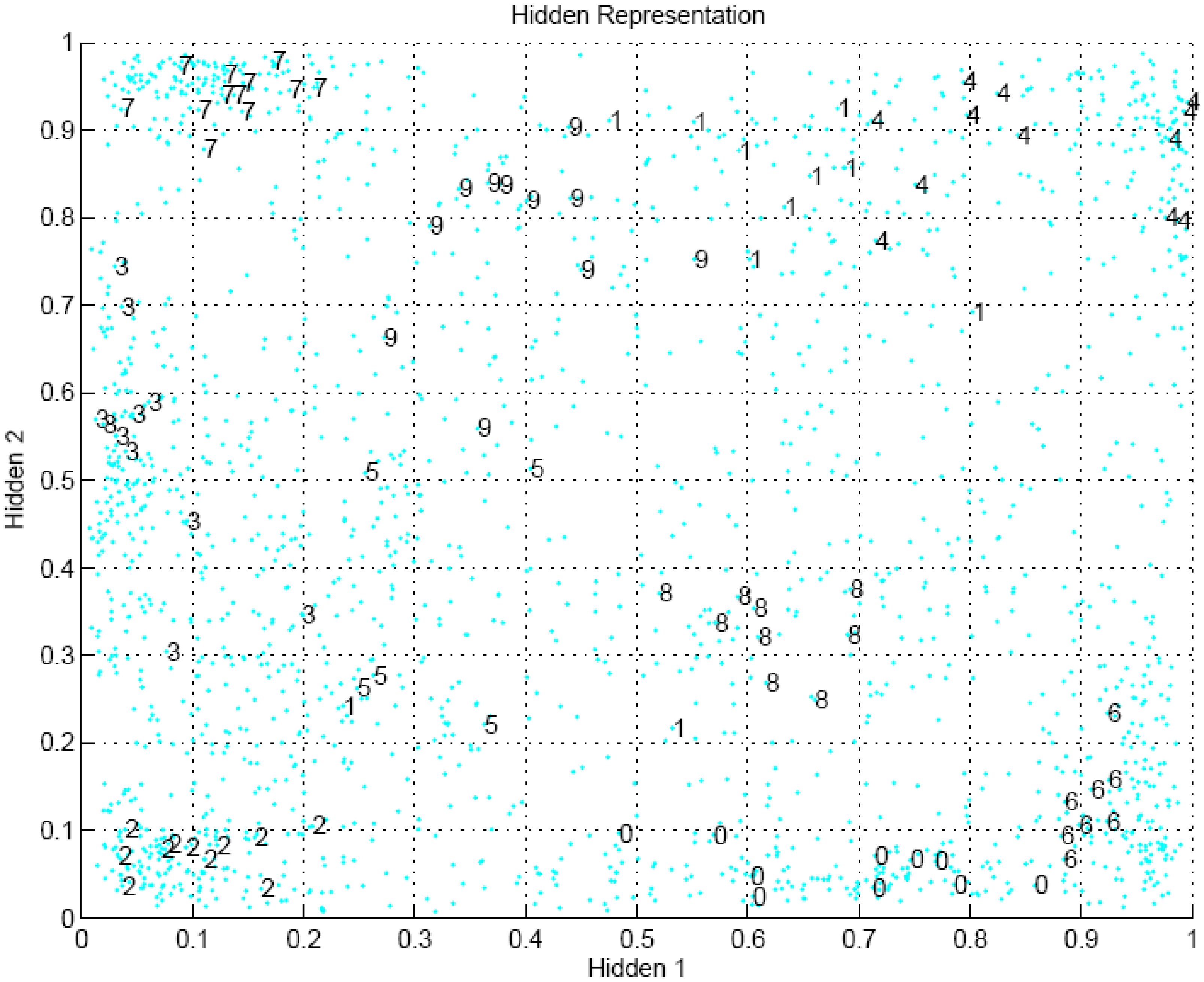


(Ash, 1989)

(Fahlman and Lebiere, 1989)

Dimensionality Reduction

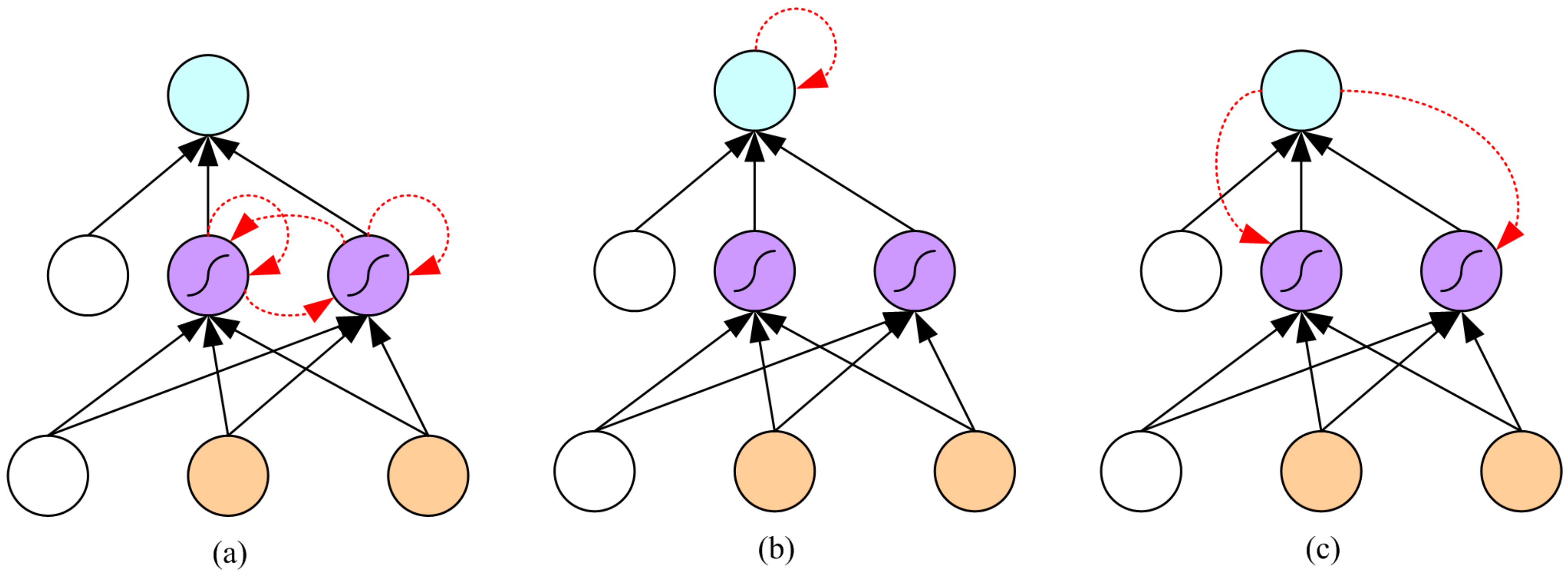



Hidden Representation
Hidden 1
Hidden 2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

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)

Recurrent Networks



Recurrent Networks

If the sequences have a small maximum length, then *unfolding in time* can be used to convert an arbitrary recurrent network to an equivalent feedforward network.

The resulting network can be trained with backpropagation with the additional requirement that all copies of each connection should remain identical.

The solution is to sum up the different weight changes in time and change the weight by the average

Unfolding in Time