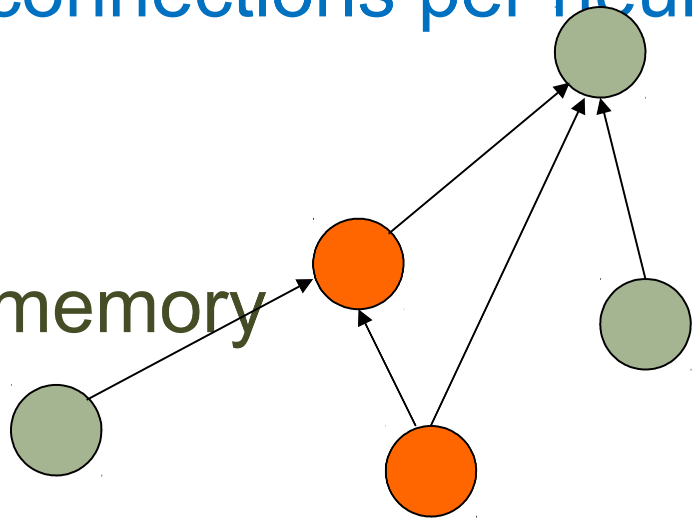
CHAPTER 11:

MULTİLAYER PERCEPTRONS

Neural Networks - Inspired by the human brain

- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: 10^{10} (other estimates say10^{11})
- Large connectitivity: 10^5 connections per neuron: synapses
- Parallel processing
- Distributed computation/memory
- Robust to noise, failures



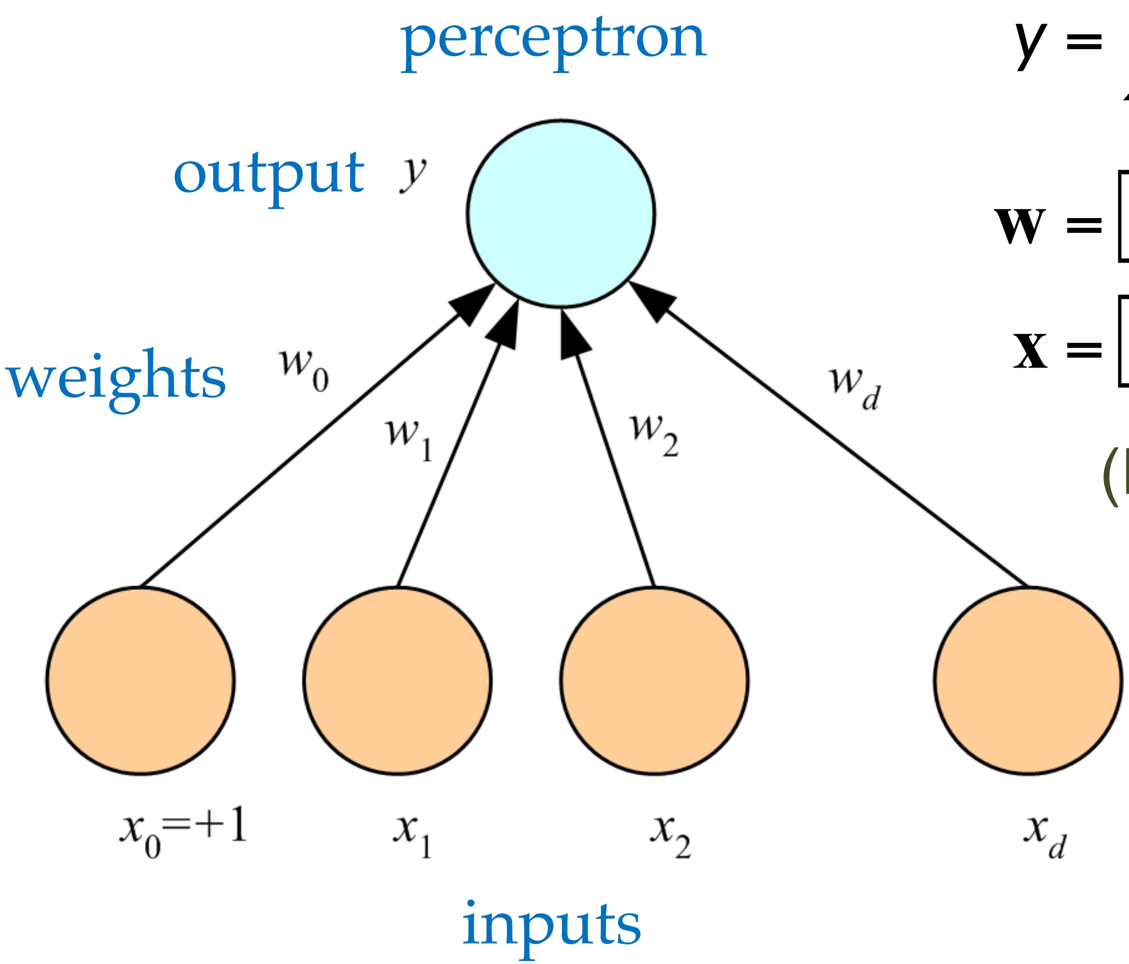
Understanding the Brain

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

  Neural net: SIMD with modifiable local memory

  Learning: Update by training/experience

Perceptron = one "neuron" or computational unit



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

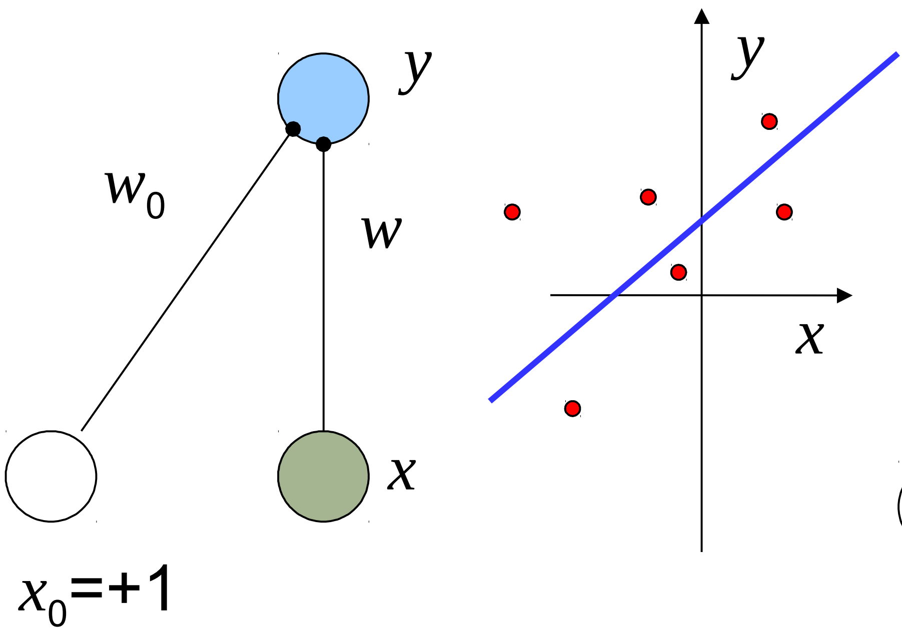
$$\mathbf{w} = \left[w_0, w_1, \ldots, w_d\right]^T$$

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

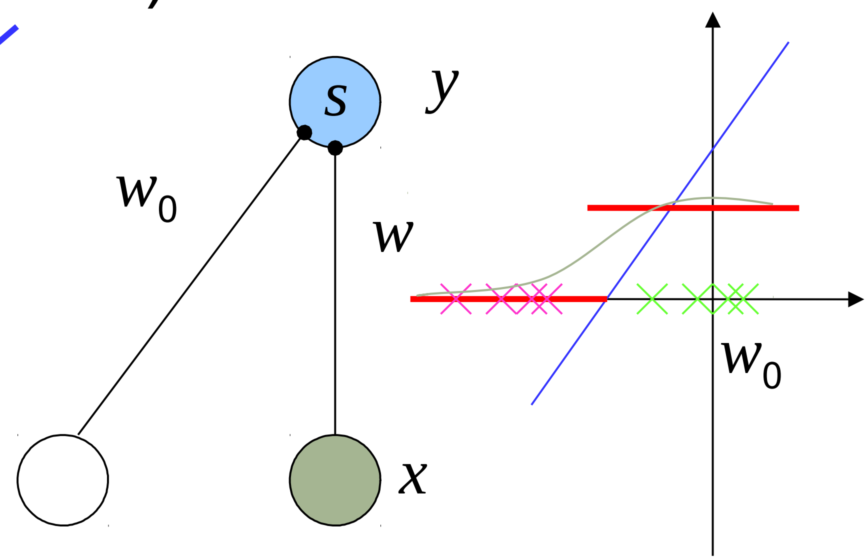
(Rosenblatt, 1962)

What a Perceptron Does

- Regression: $y=wx+w_0$



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



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

K Outputs

Regression:

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

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

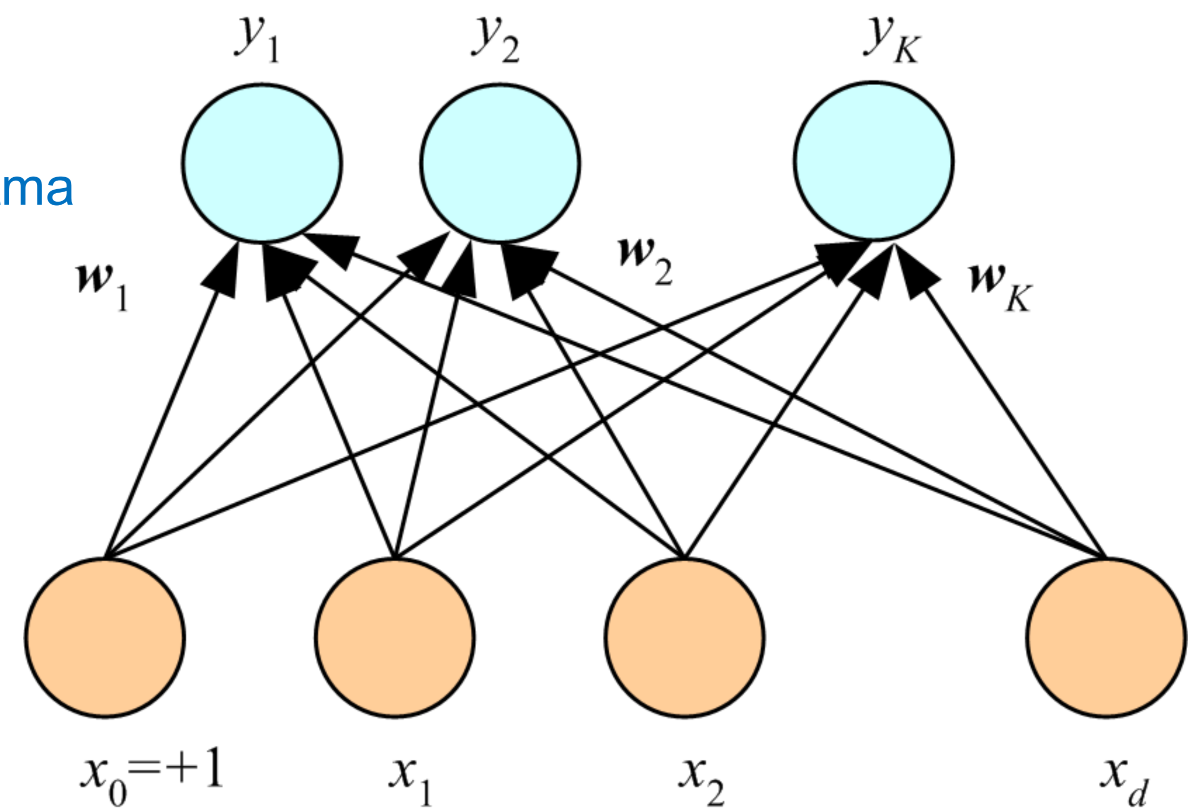
Classification:

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

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

choose C_i

if $y_i = \max_k y_k$



softmax function or **normalized exponential function** converts a real vector into one with values in the range (0, 1) that add up to 1

Training (i.e., learning the right weights)

- Online (instances seen one by one) vs batch (whole sample) learning: In online (= incremental) learning:
  - No need to store the whole sample
  - Problem may change in time
  - Wear and degradation in system components
- Stochastic gradient-descent (= incremental gradient descent): Update after a single data instance
- 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

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

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

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

cross-entropy cost function: used instead of the quadratic error function to speed up learning

- K>2 softmax outputs

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

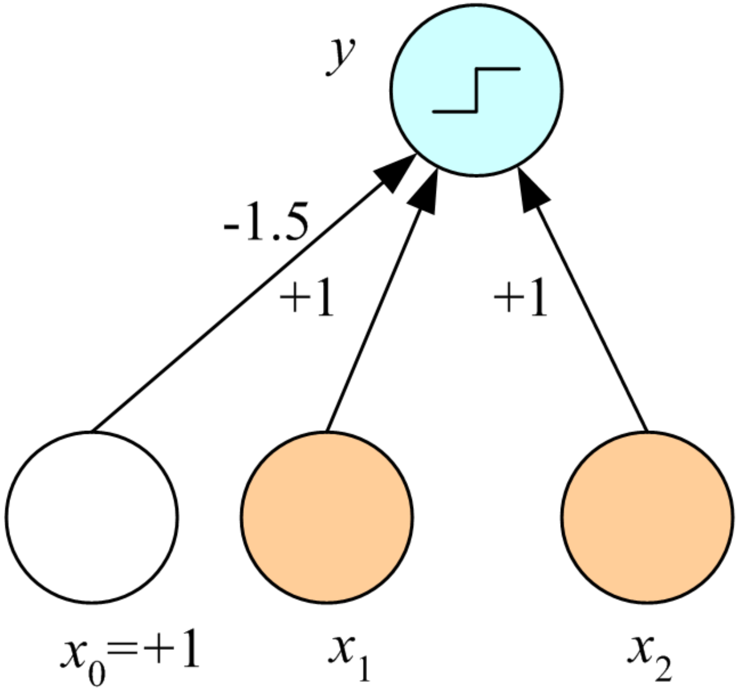
cross-entropy cost function

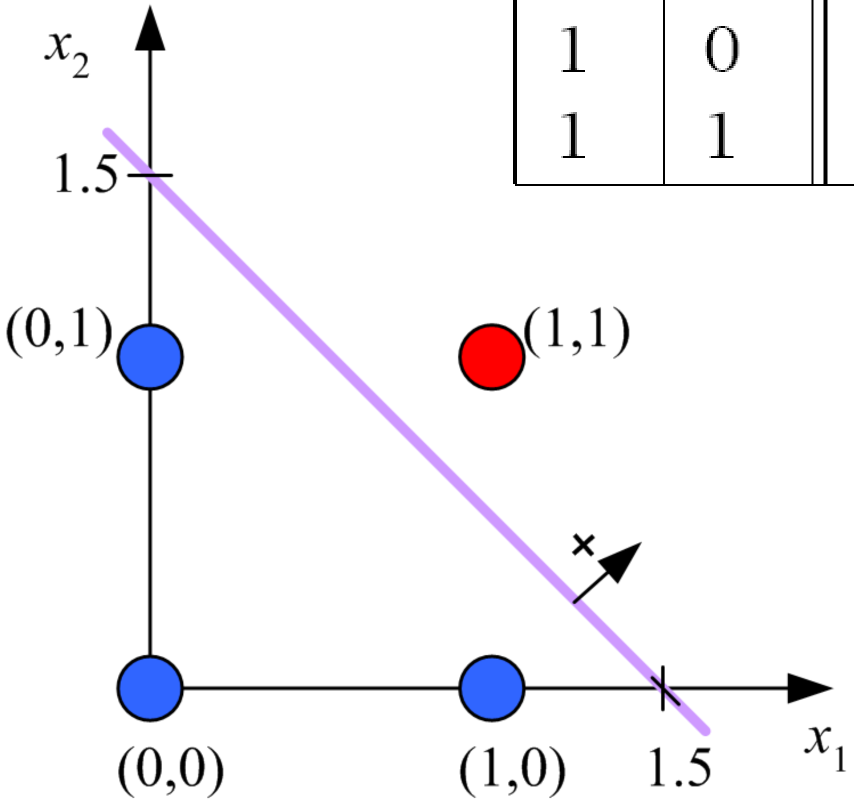
$$\Delta w_{ij}^t = \eta\left(r_i^t - y_i^t\right) x_j^t$$

see
Chapter 3 of Michael Nielsen's "Neural Networks and Deep Learning" online

Learning Boolean AND

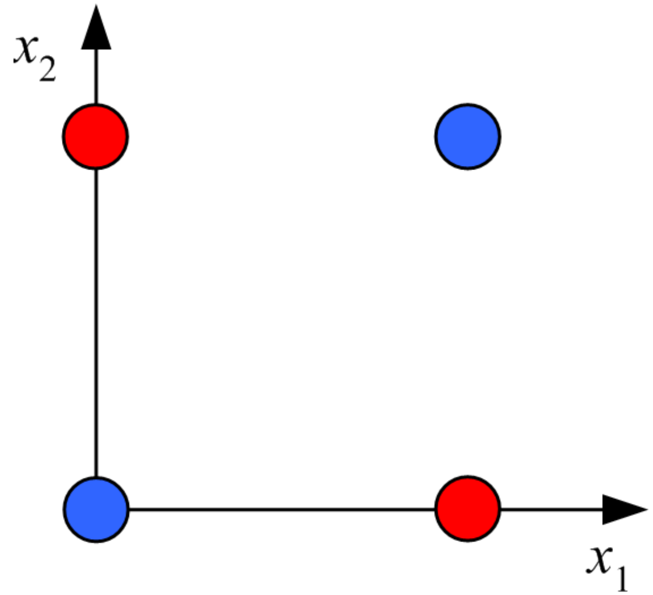
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





XOR

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

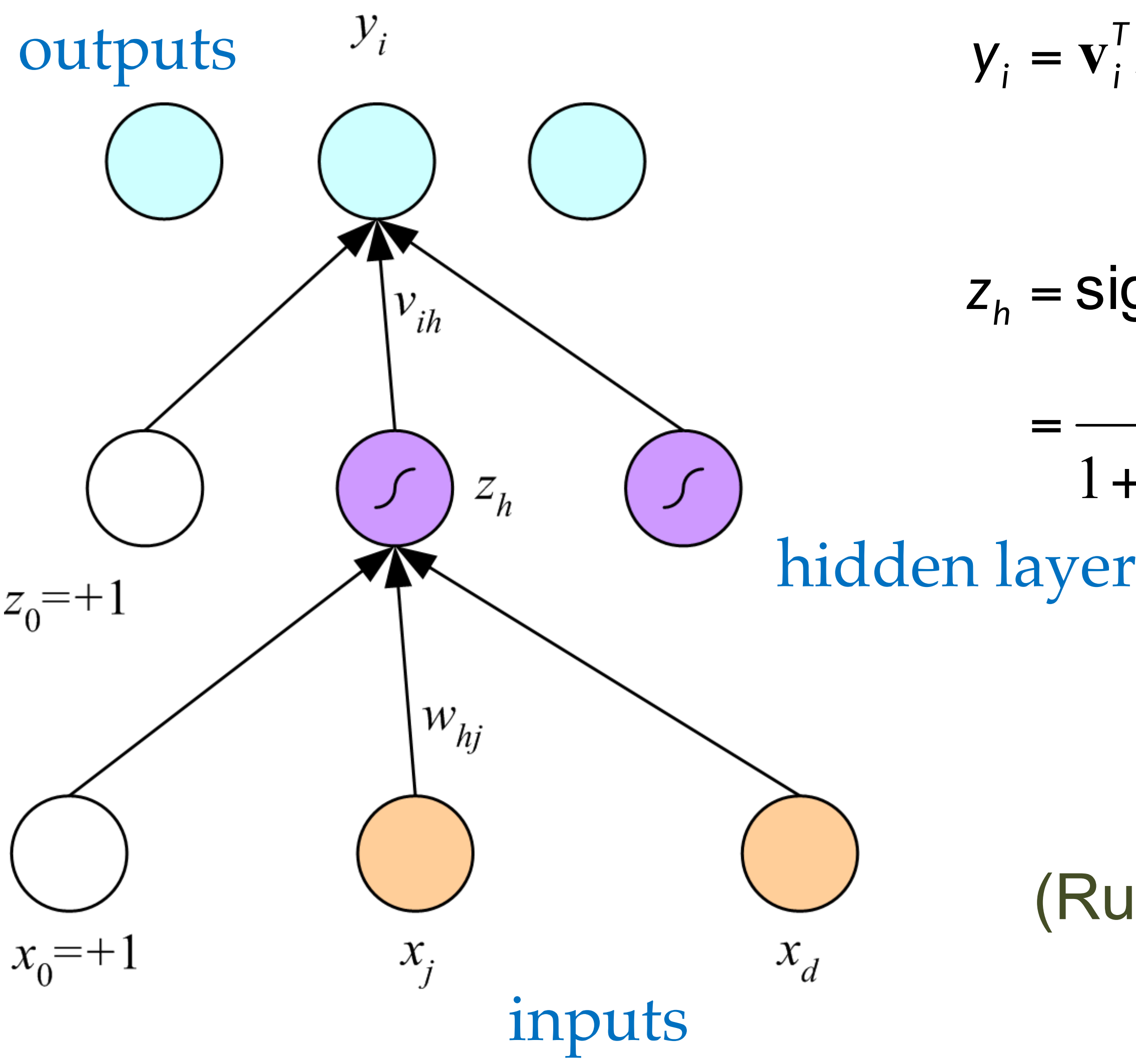


- No w_0, w_1, w_2 satisfy:

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

(Minsky and Papert, 1969)

Multilayer Perceptrons

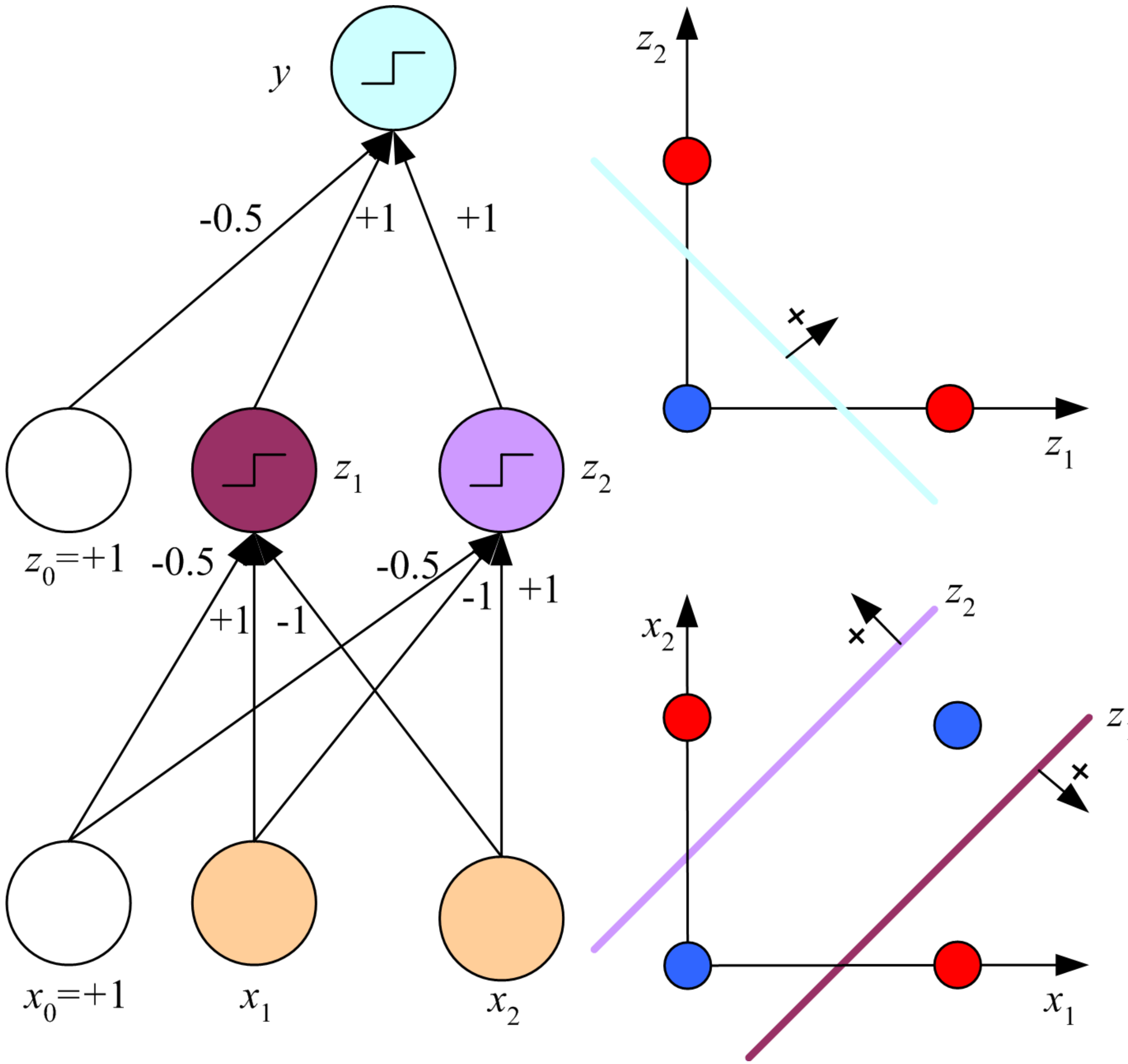


$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

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

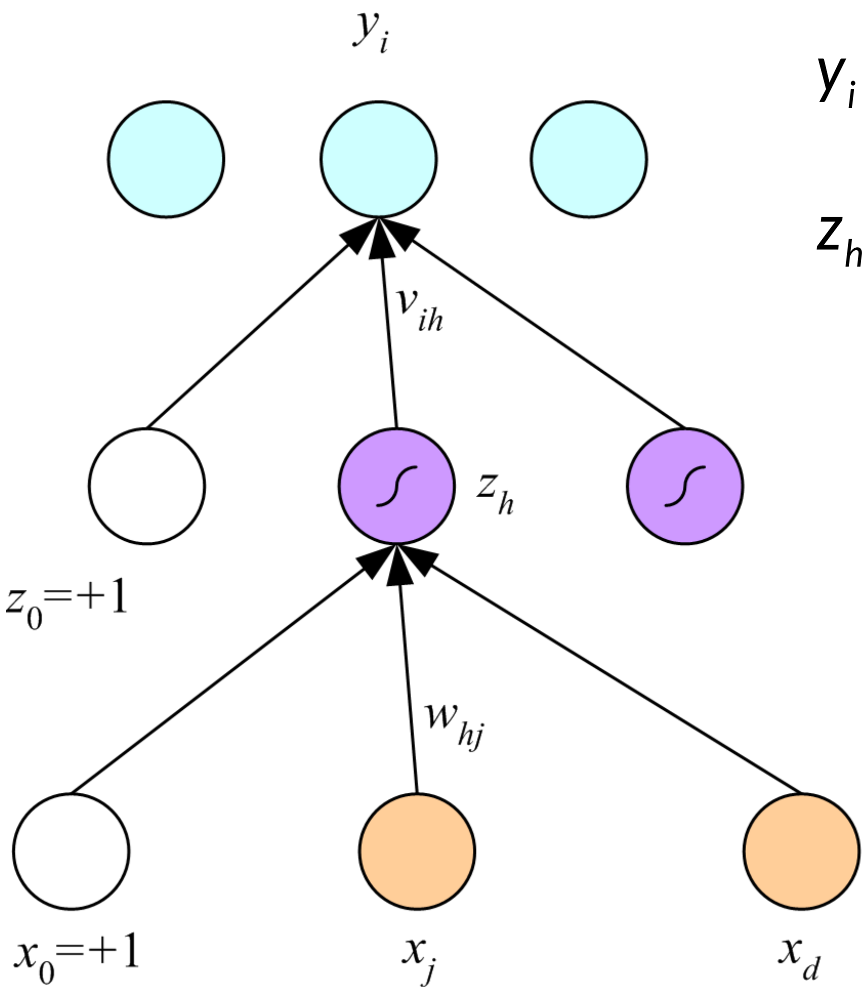
$$= \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

(Rumelhart, Hinton, and Williams, 1986)



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

Backpropagation (Rumelhart, Hinton & Williams, 1986)



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

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

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

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

gradient of the error function

Regression

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

Forward

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

$\boldsymbol{x}$

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

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

Backward

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

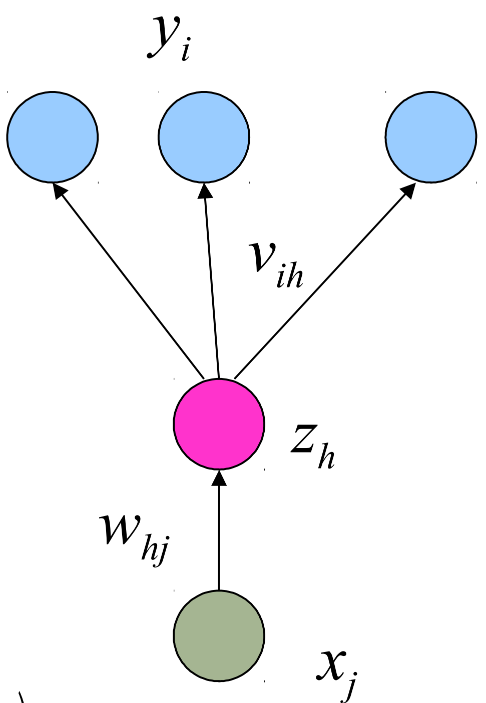
Regression with Multiple Outputs

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

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

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

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



Error Back Propagation Algorithm using Stochastic Gradient Descent

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

Repeat

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

forward propagation of inputs

For $h = 1, \ldots, H$ (h: index over hidden units)

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

For $i = 1, \ldots, K$ (i: index over output units)

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

backward propagation of errors

For $i = 1, \ldots, K$

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

For $h = 1, \ldots, H$

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

weights update

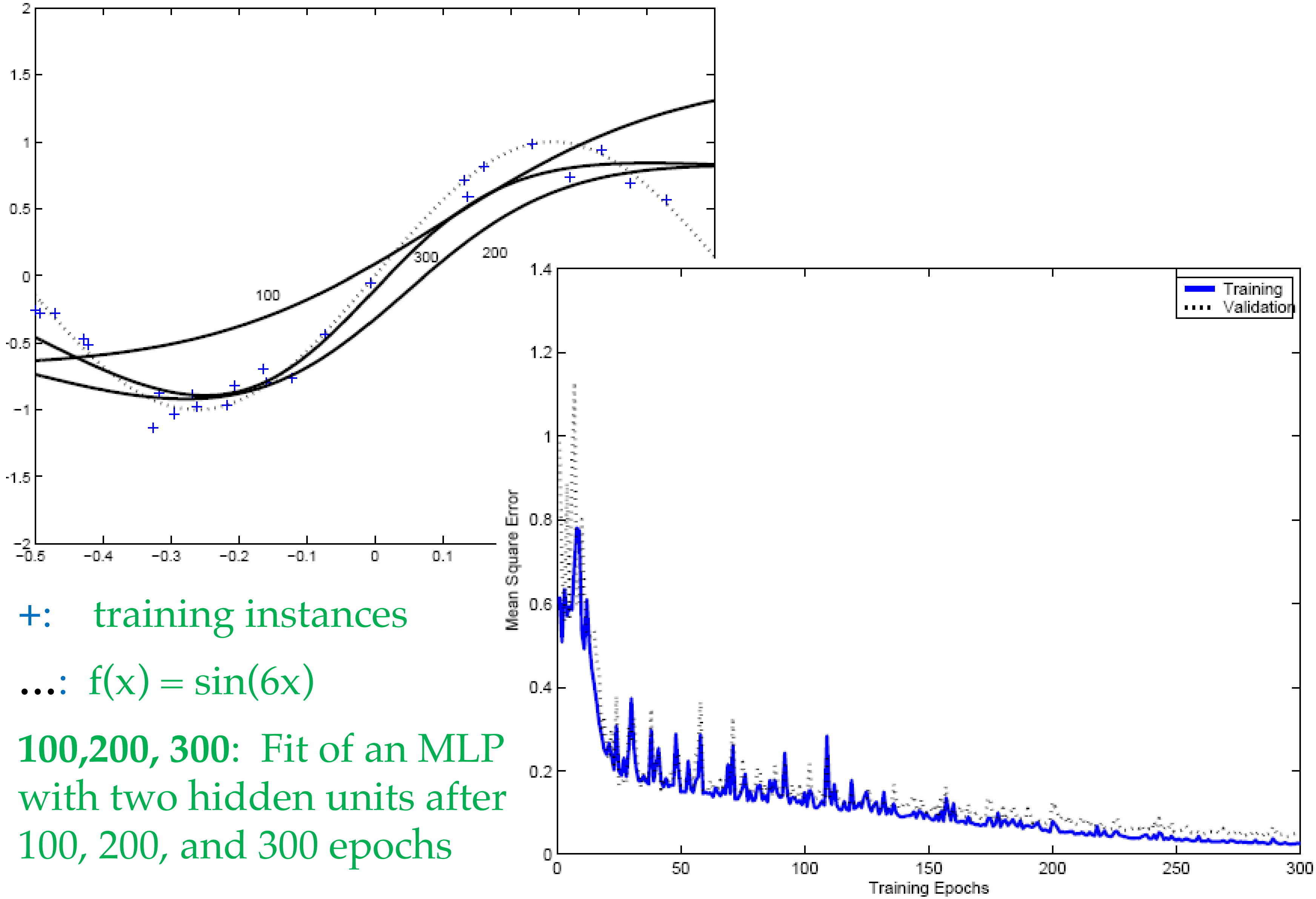
For $i = 1, \ldots, K$

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

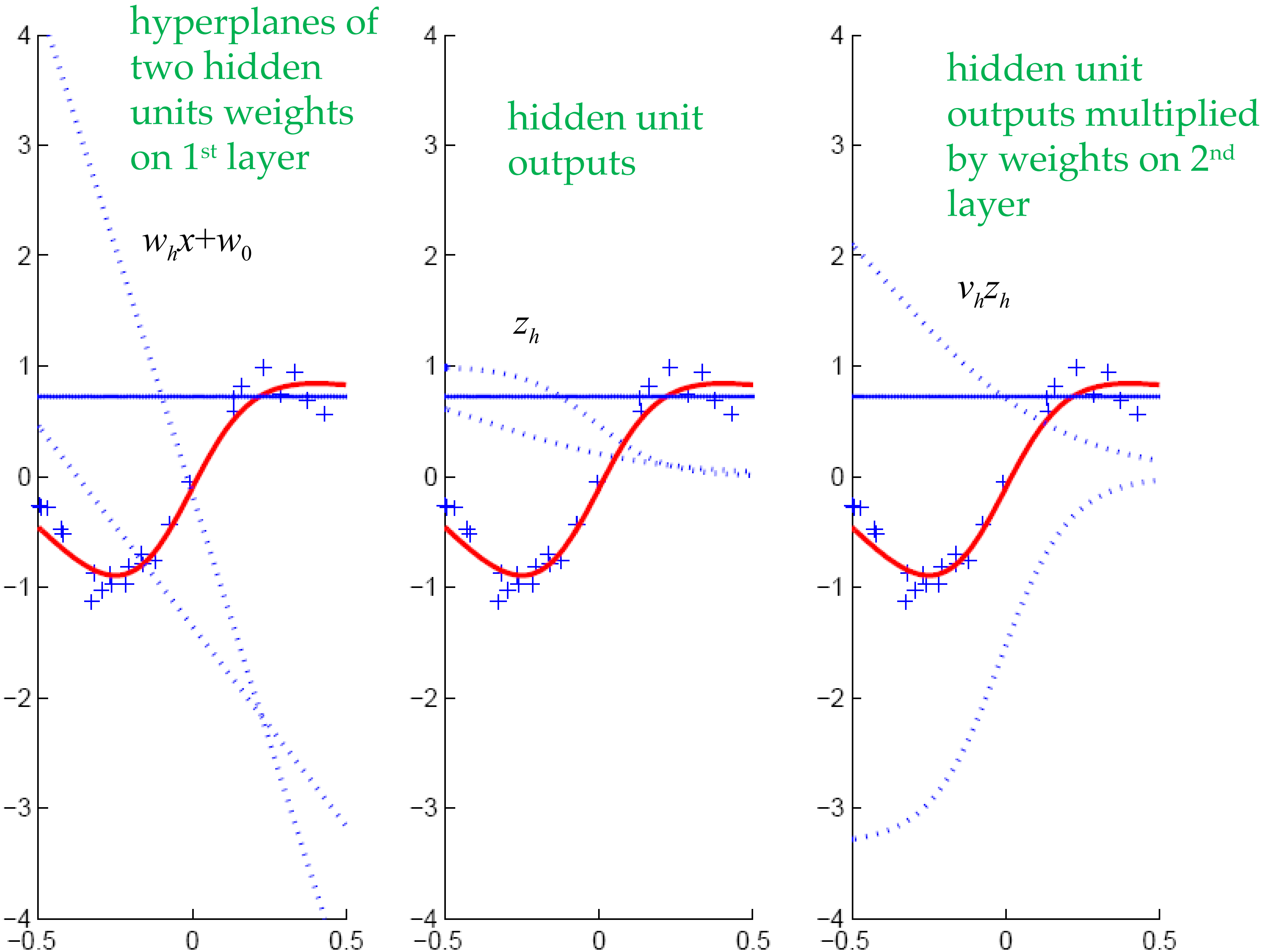
For $h = 1, \ldots, H$

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

Until convergence let's talk about termination criteria



- +: training instances
- ...: f(x) = sin(6x)
- **100,200, 300**: Fit of an MLP with two hidden units after 100, 200, and 300 epochs



MLP fit is formed as the sum of the outputs of the hidden units

Two-Class Discrimination

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

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

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

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

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

K>2 Classes

softmax

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

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

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

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

Multiple Hidden Layers

- MLP (multi-layer perceptrons) 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

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

where t is time (= epoch) and α is a constant, say between 0.5 and 1

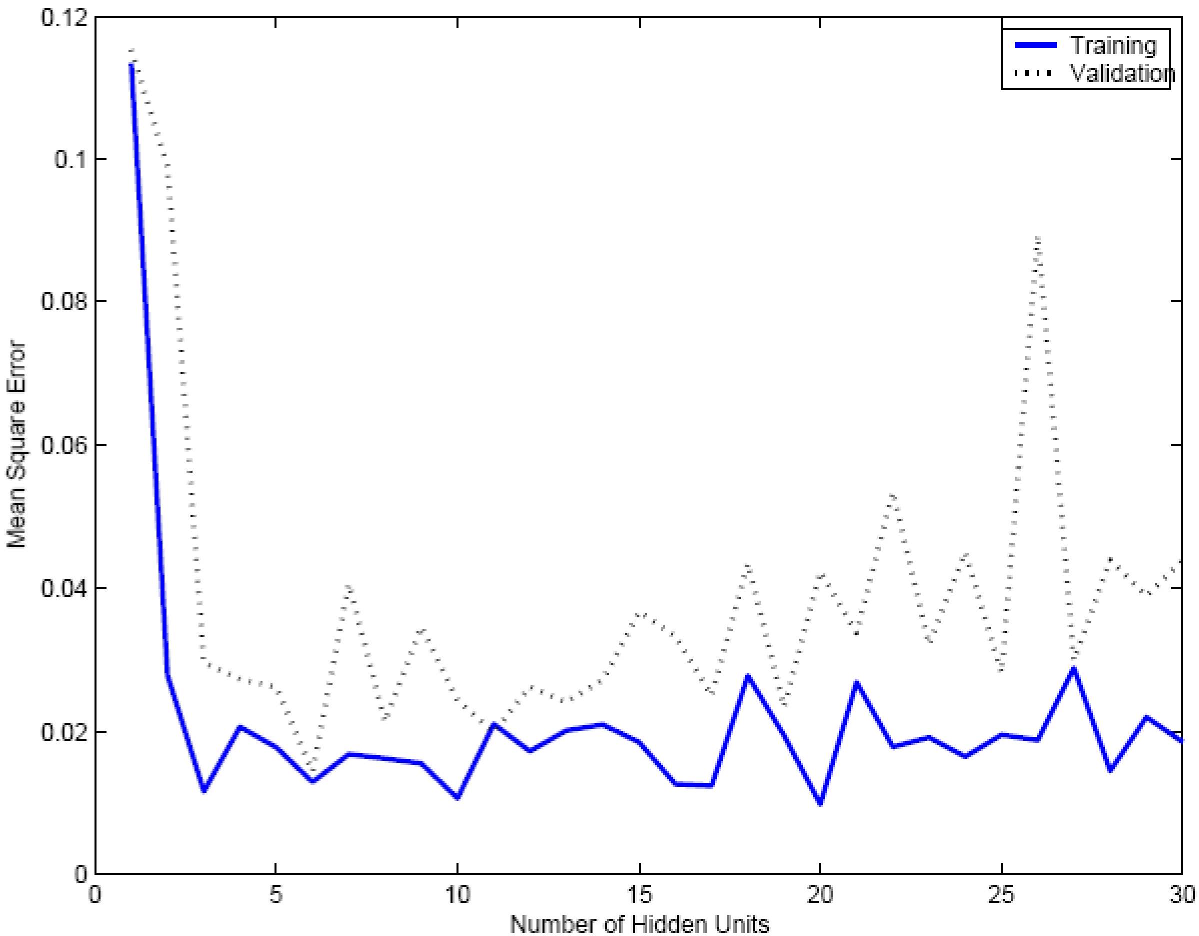
- Adaptive learning rate

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

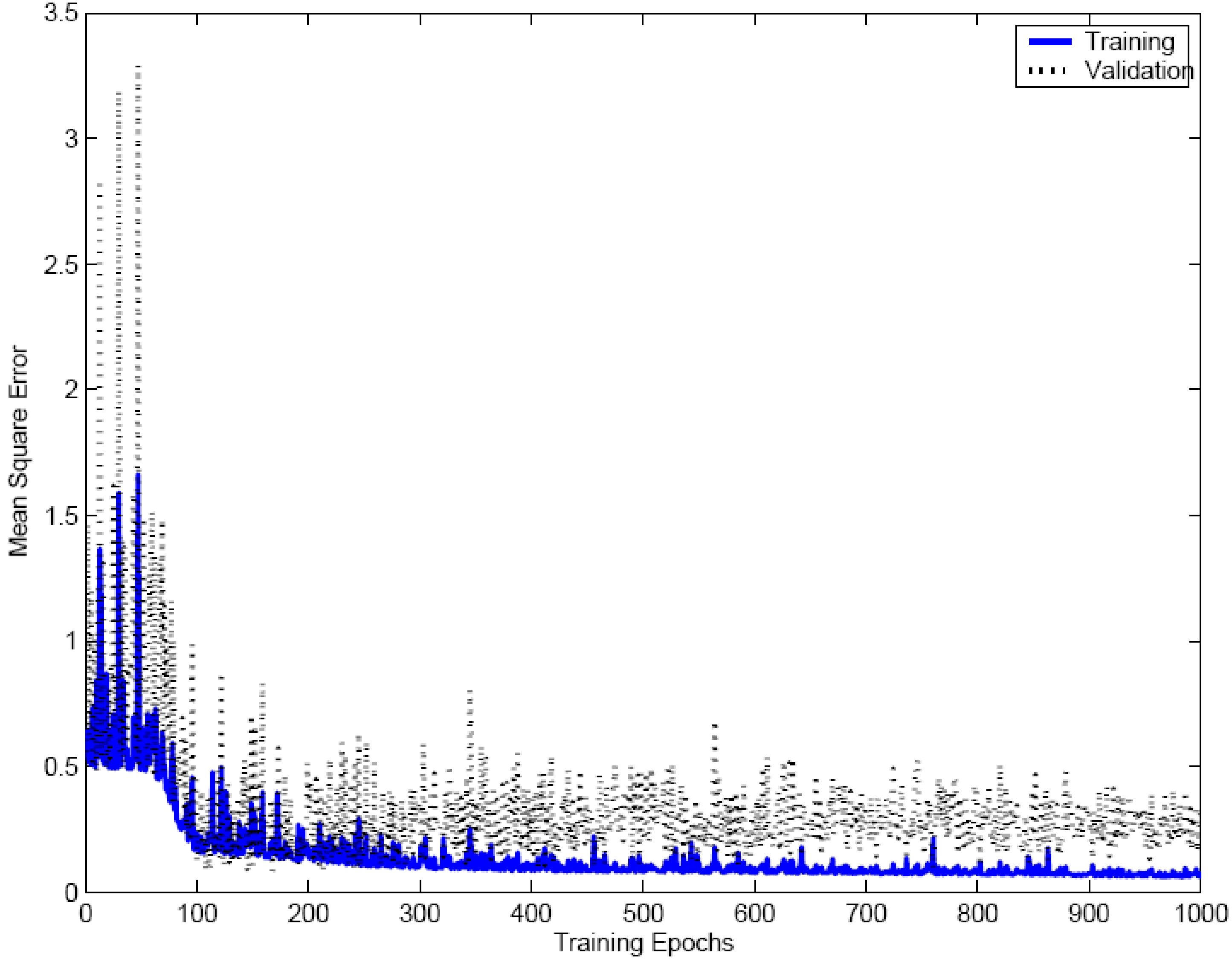
if error rate is decreasing, increase learning rate by a constant amount; otherwise, decrease learning rate geometrically

Overfitting/Overtraining

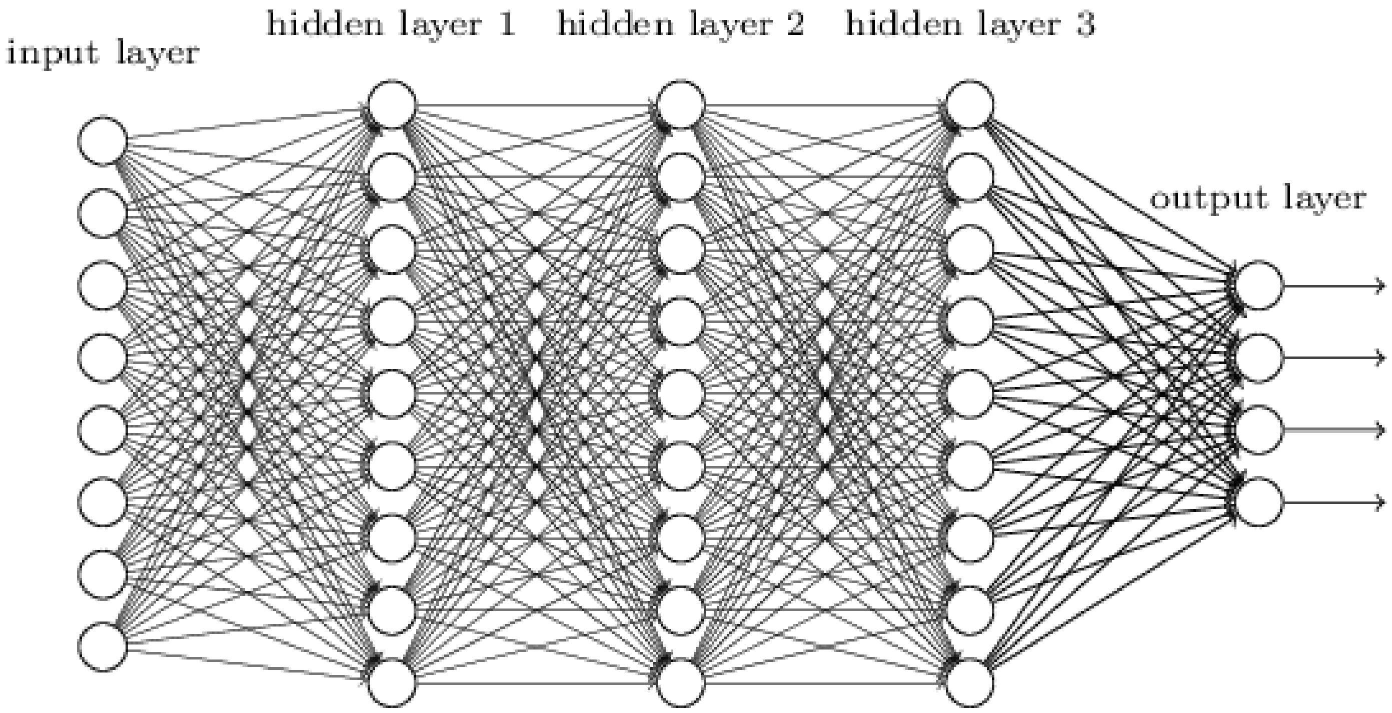
Number of weights: $H\,(d+1)+(H+1)K$



MLP with d inputs, H hidden units, and K outputs: has Hx(d+1) weights in 1st layer and Kx(H+1) in 2nd layer


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

Fully Connected, Layered MLP Architecture



Taken from
[Chapter 6 of Michael Nielsen's "Neural Networks and Deep Learning" online texbook]()

Expressive Capabilities of ANNs

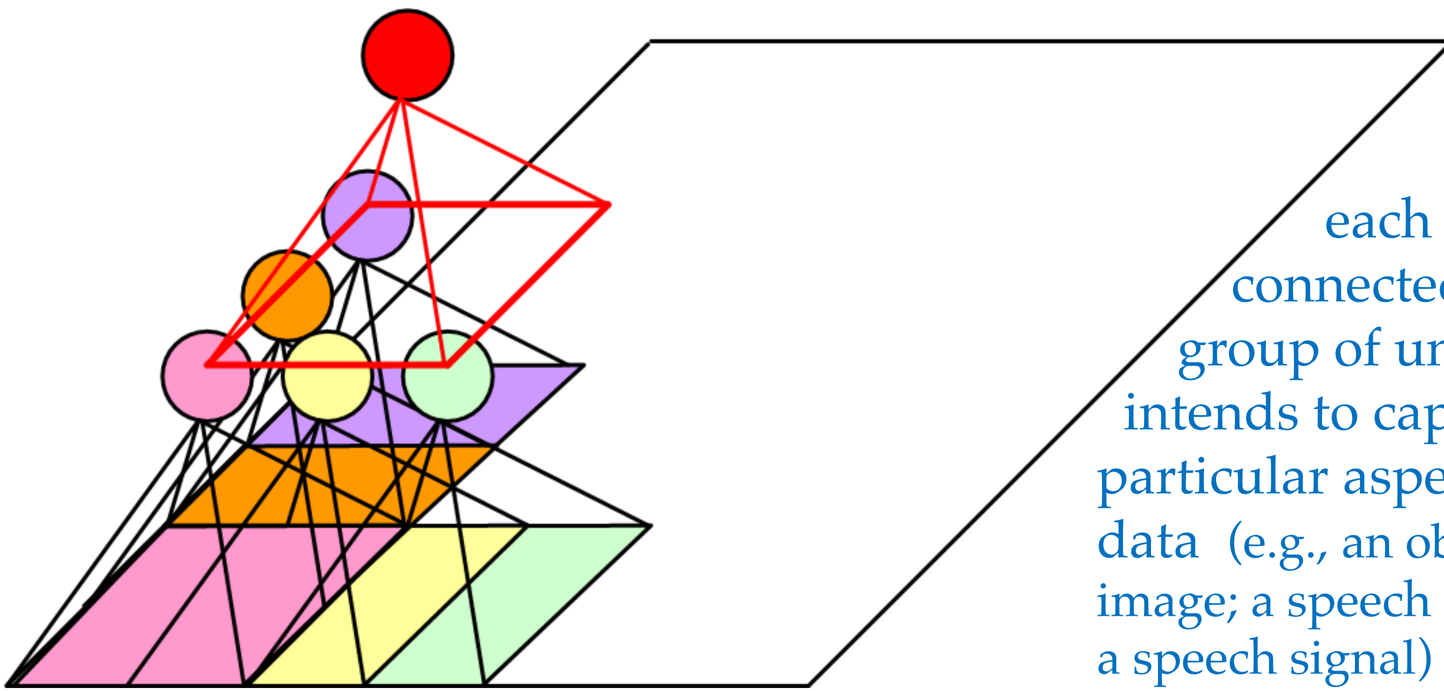
(Taken from Tom Mitchell's "Machine Learning" Textbook)

- Boolean Functions:
  - Every Boolean function can be represented by a network with a single hidden layer
  - but might require exponential (in number of inputs) hidden units
- Continuous Functions:
  - Every bounded continuous function can be approximated with arbitrary small error by a network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Other ANN Architectures: Structured MLP

- Convolutional networks (Deep learning)

Useful on data that has "local structure" (e.g., images; sound)



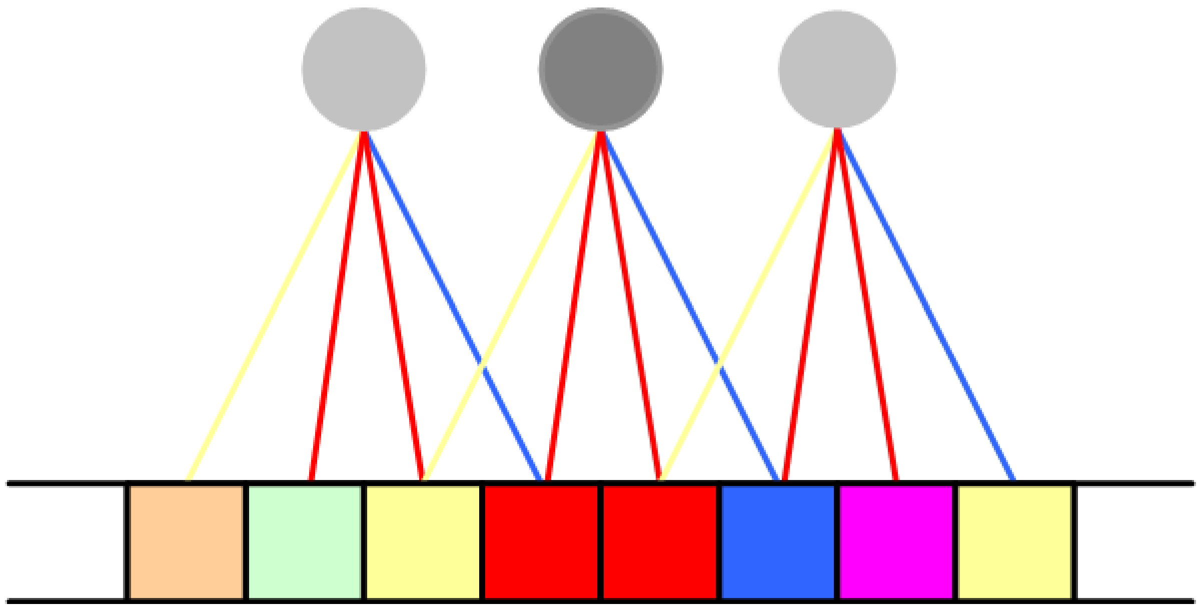
each unit is connected to a local group of units, and intends to capture a particular aspect of the data (e.g., an object in an image; a speech phoneme in a speech signal)

(Le Cun et al, 1989)

see [Chapter 6 of Michael Nielsen's "Neural Networks and Deep Learning" online]

Weight Sharing

three translations over the data of the same "filter"

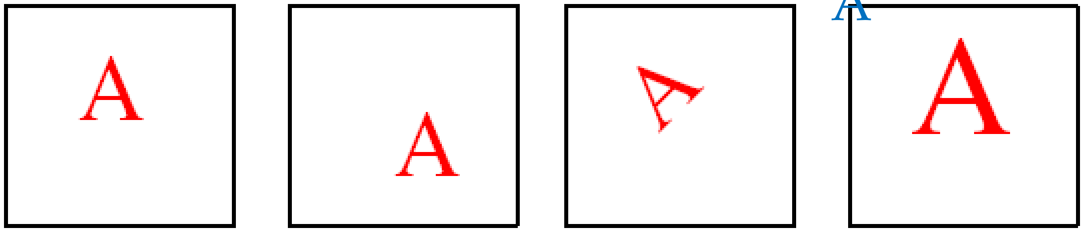


Each "filter" is applied on overlapping segments of the data, and uses the same weights each time that is applied during the same epoch ("translation invariance").
These weights are updated after each epoch.

Hints

- Invariance to translation, rotation, size

the "A" remains an "A"



- Virtual examples

(Abu-Mostafa, 1995)

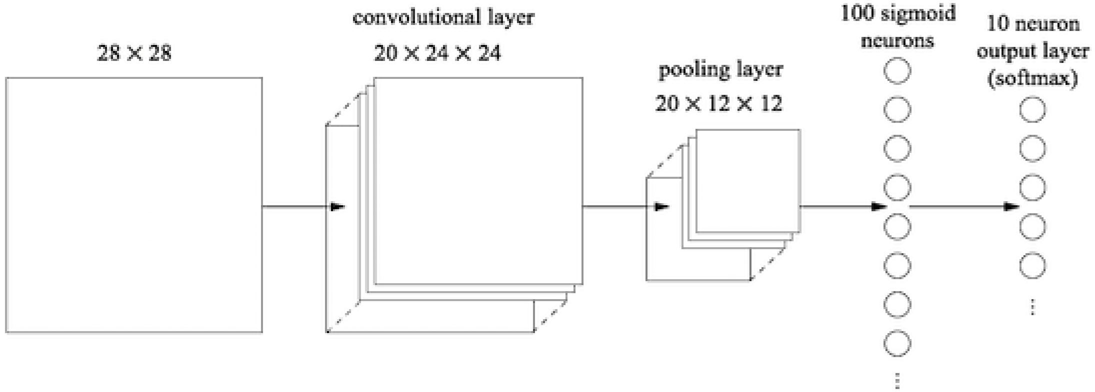
- Augmented error: $E' = E + \lambda_h E_h$

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

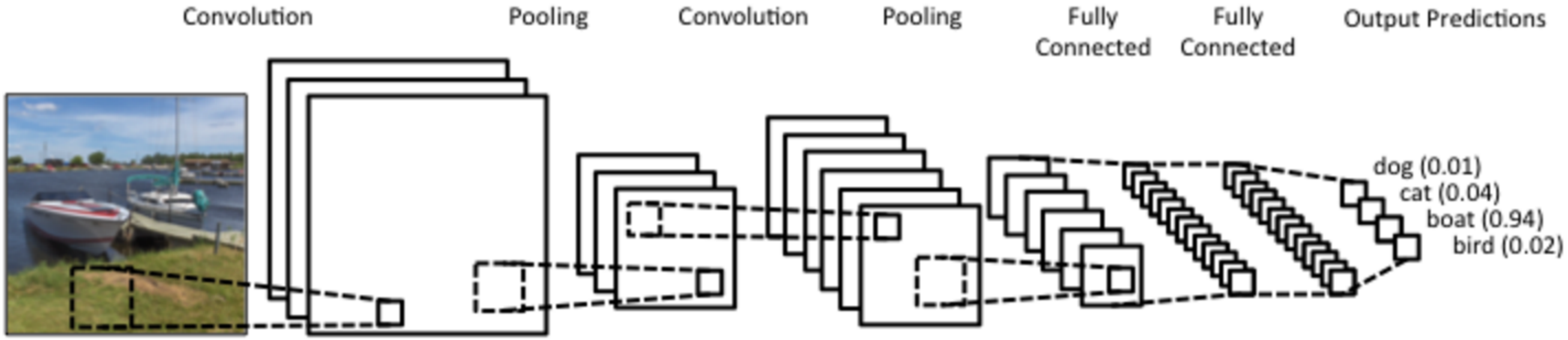
Approximation hint:

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

Convolutional Neural Networks in Practice



Taken from
[Chapter 6 of Michael Nielsen's "Neural Networks and Deep Learning" online texbook]()



Taken from ["Understanding Convolutional Neural Networks for NLP"]()

Tuning the Network Size

- Destructive

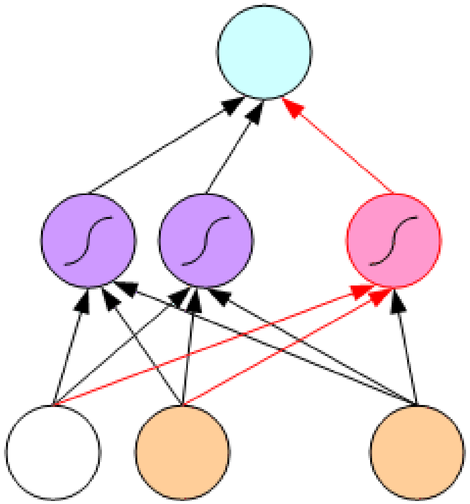
  Weight decay:

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

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

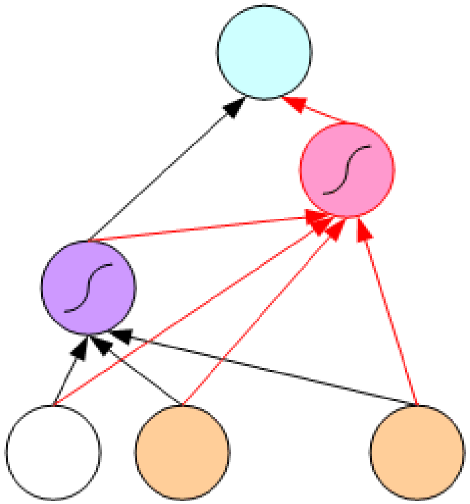
- Constructive

  Growing networks



Dynamic Node Creation

(Ash, 1989)



Cascade Correlation

(Fahlman and Lebiere, 1989)

Bayesian Learning

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

$$p(\mathbf{w} \mid X) = \frac{p(X \mid \mathbf{w})\, p(\mathbf{w})}{p(X)} \qquad \hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log p(\mathbf{w} \mid X)$$

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

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

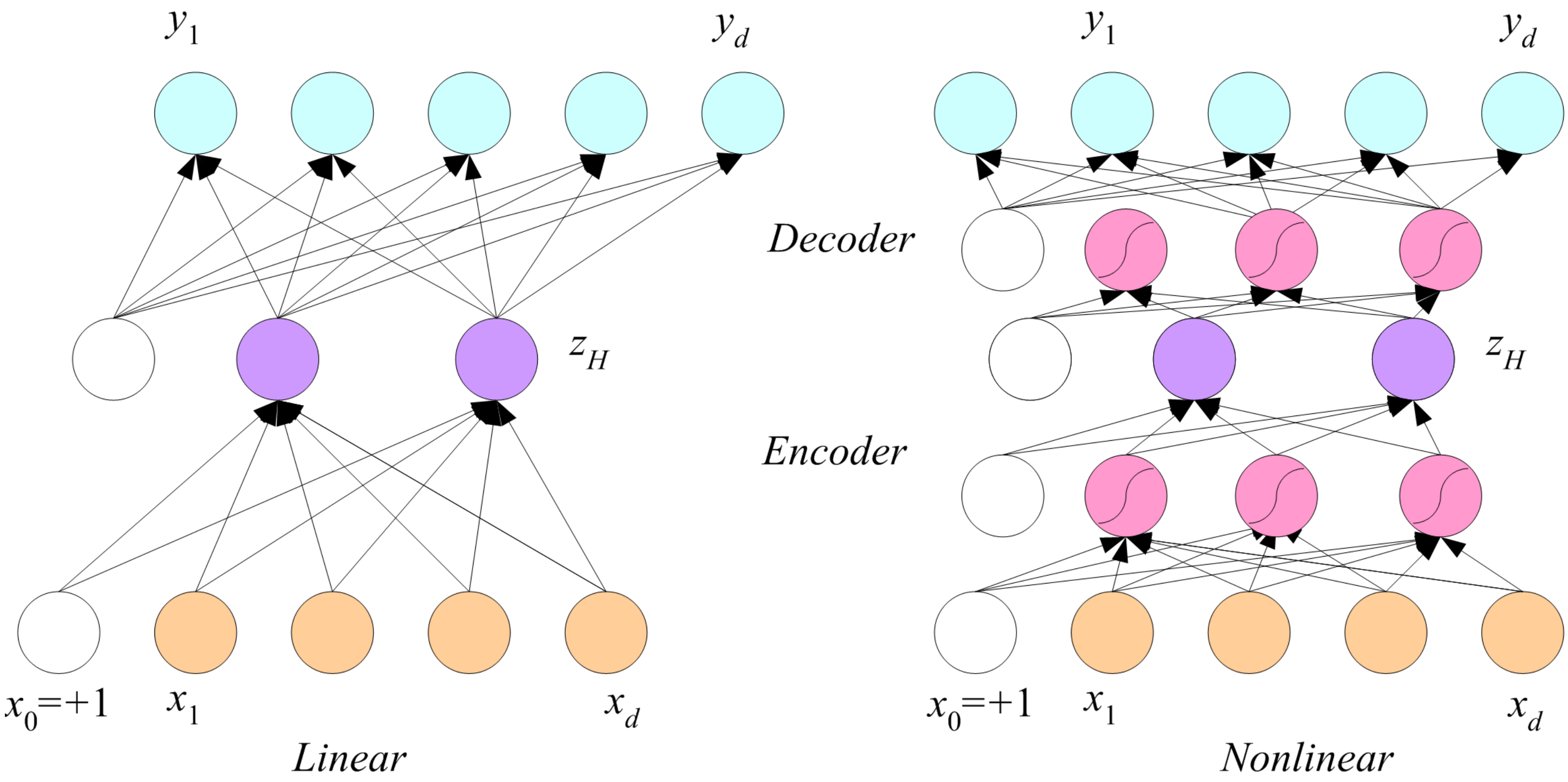
$$E' = E + \lambda \|\mathbf{w}\|^2$$

- Weight decay, ridge regression, regularization

  cost=data-misfit + λ complexity

  More about Bayesian methods in chapter 14

Dimensionality Reduction



Autoencoder networks

Autoencoder: An Example

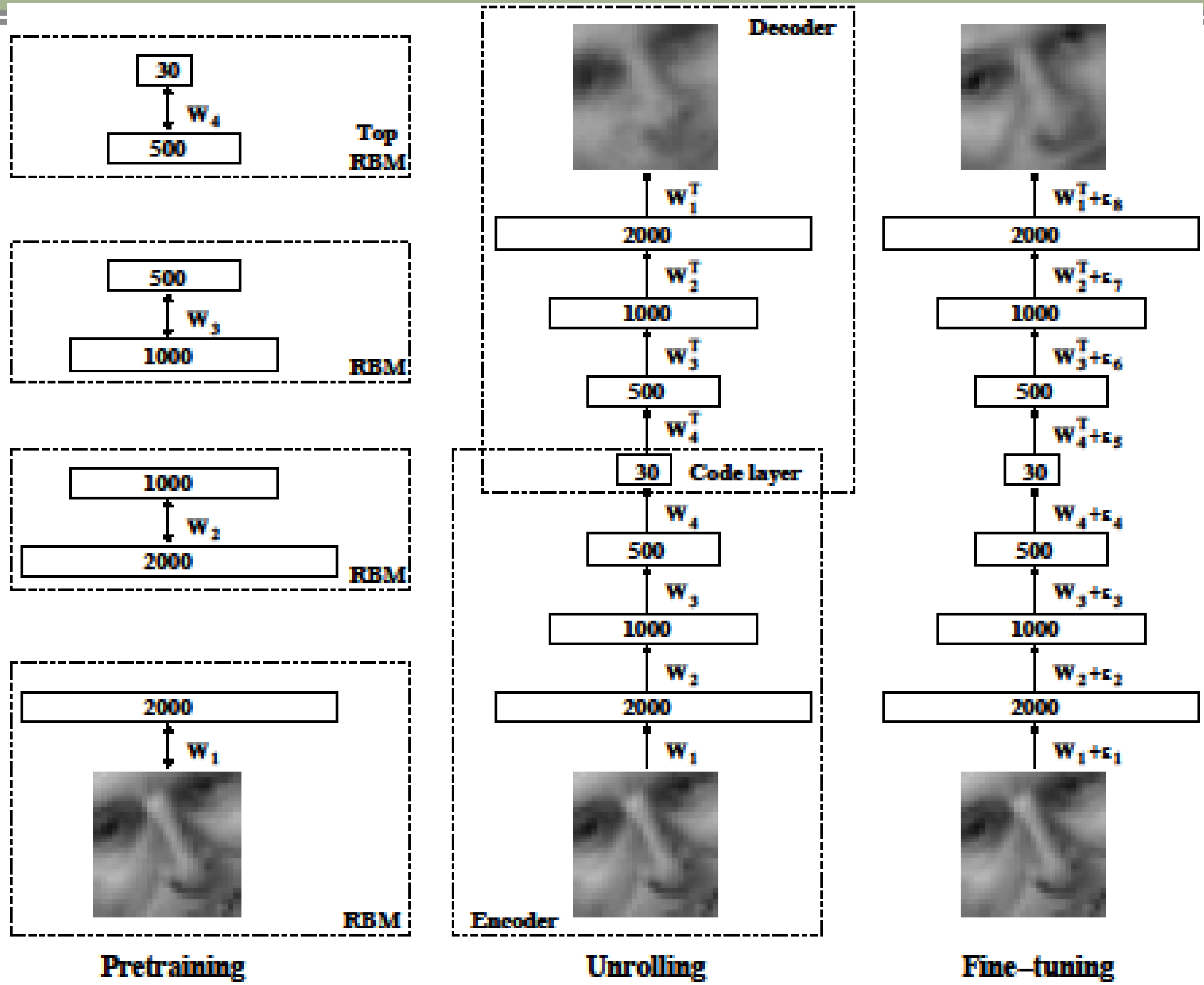
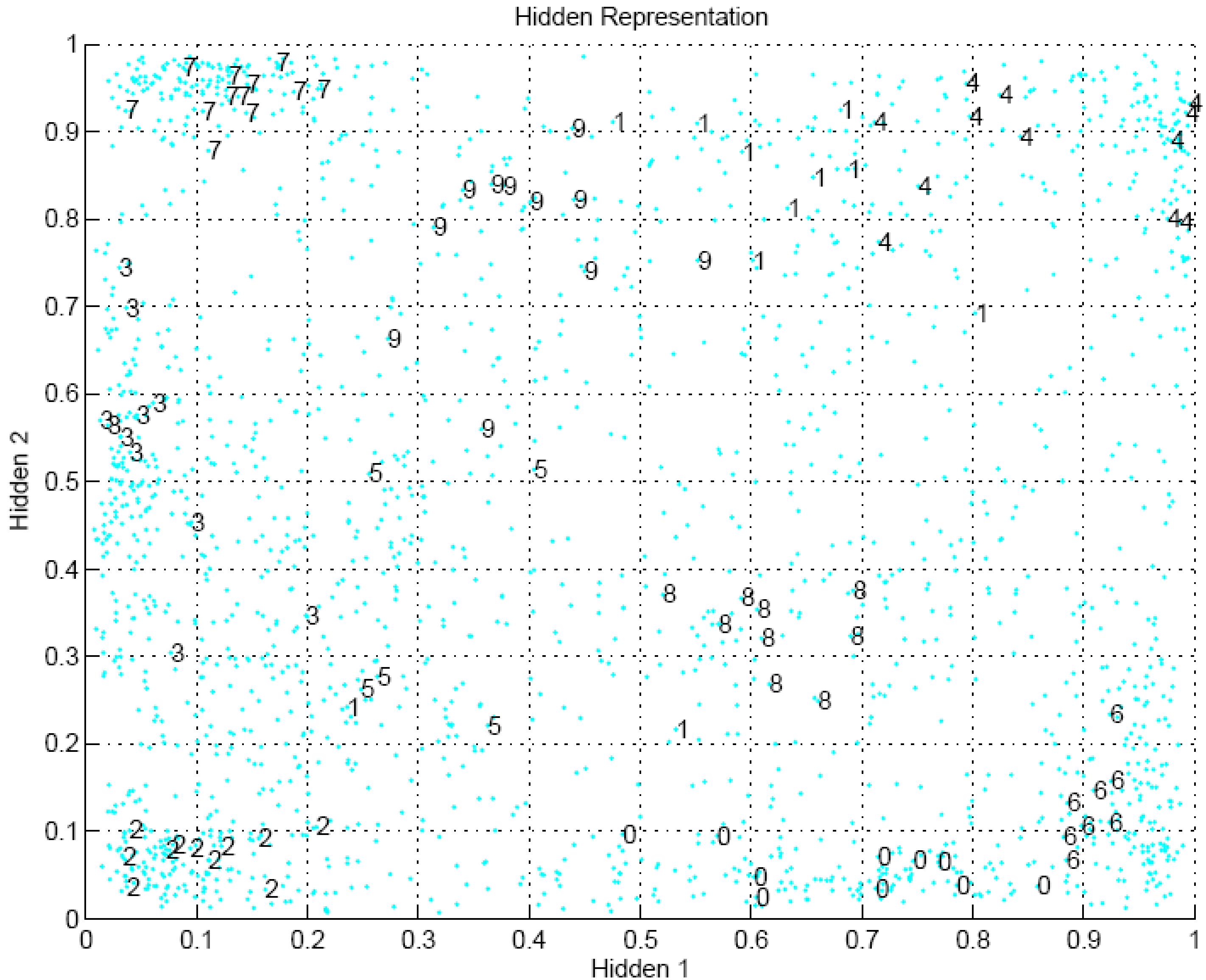


Figure from Hinton and Salakhutdinov, Science, 2006

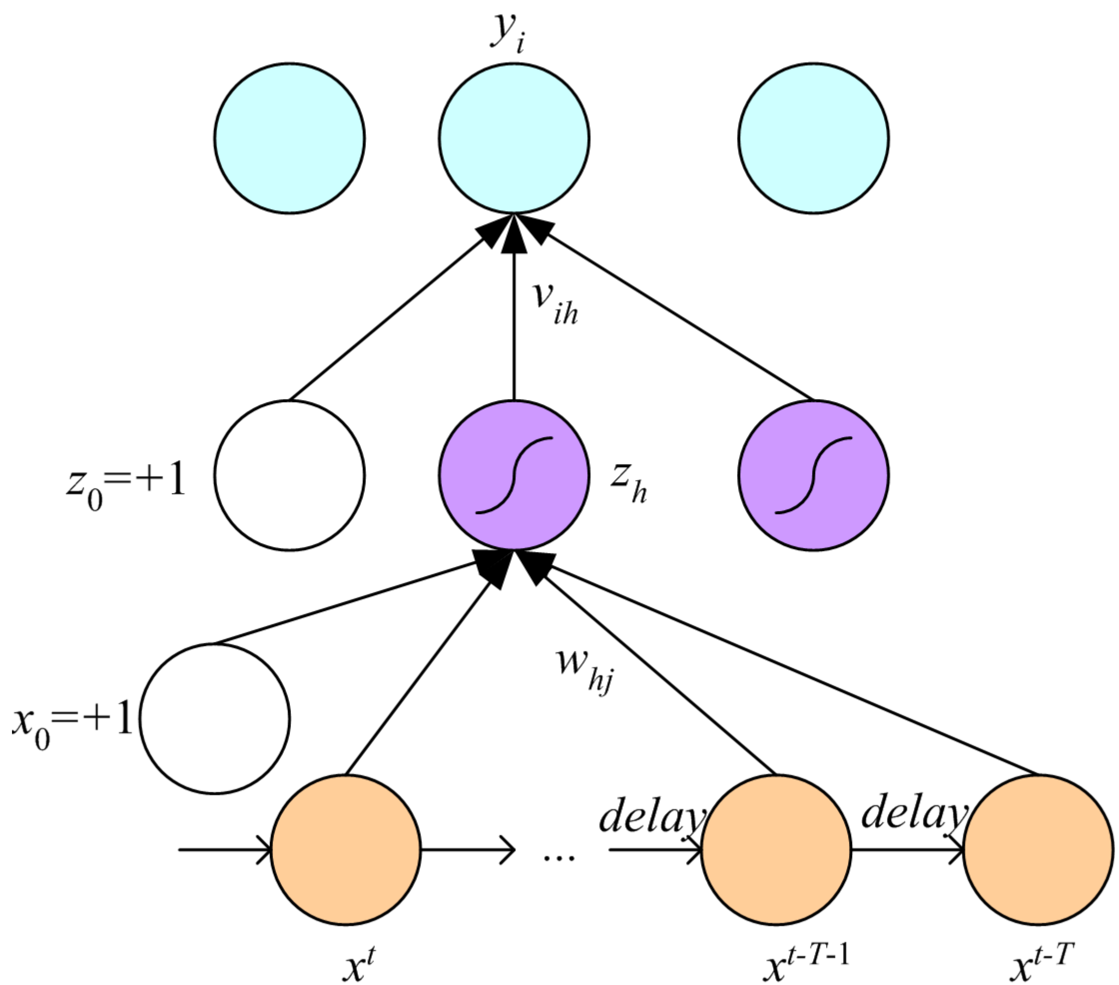


Hidden Representation
Hidden 1
Hidden 2

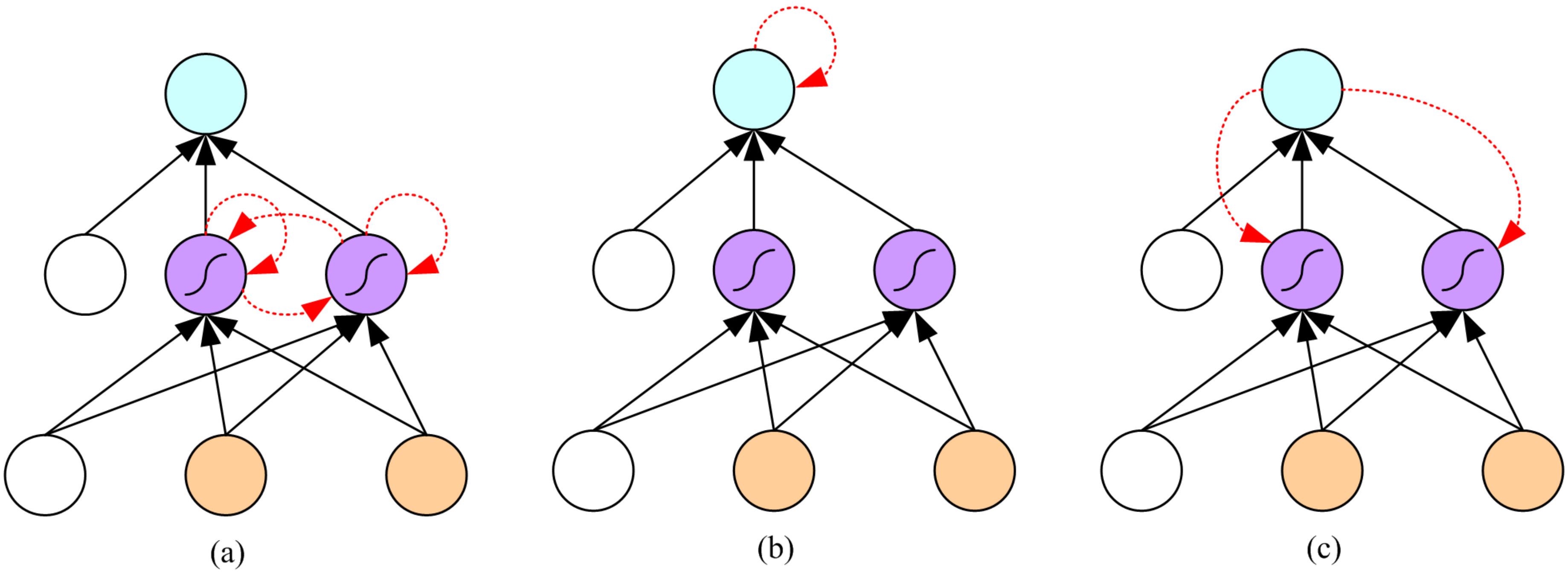
Learning Time

- 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)

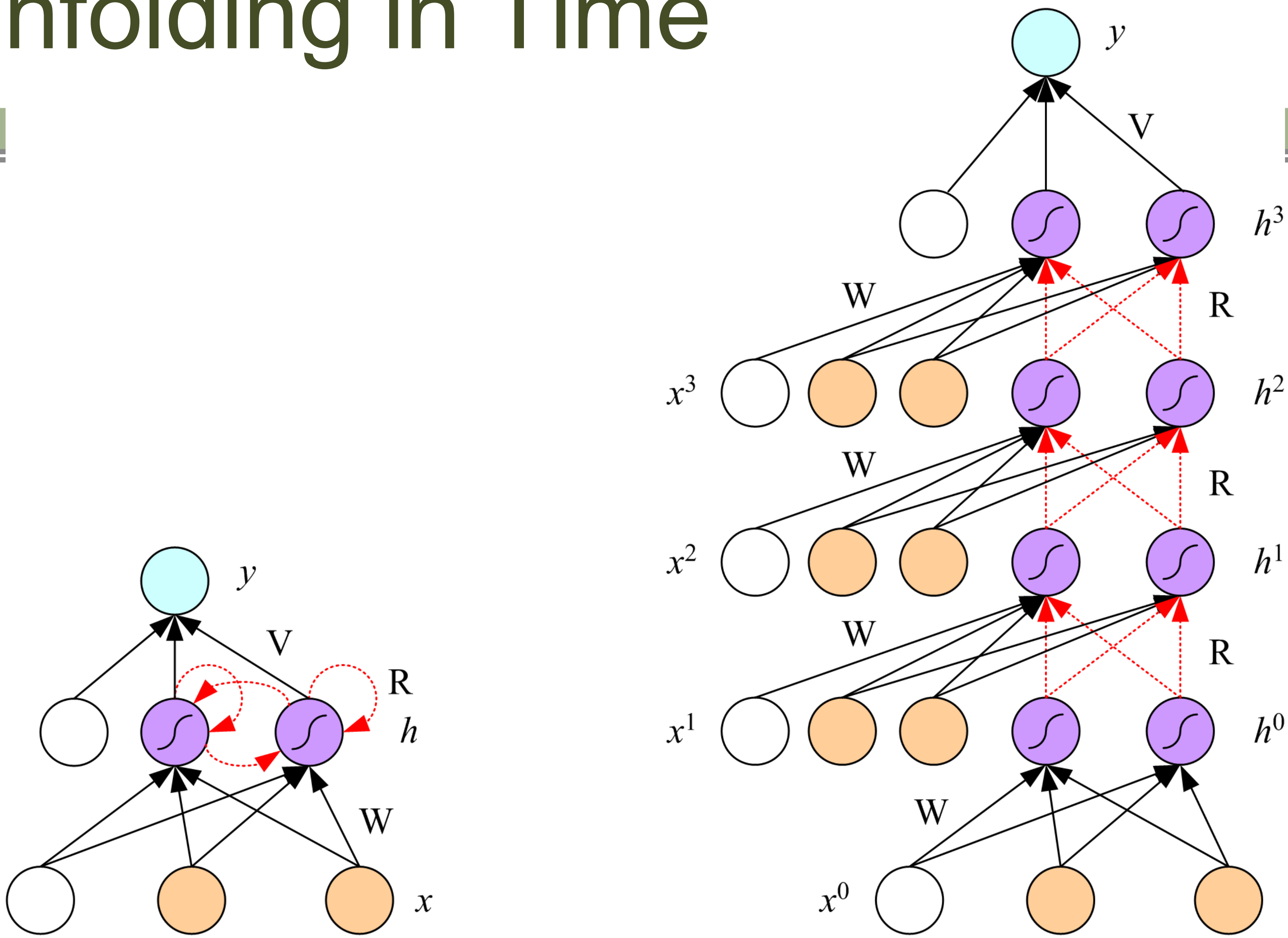
Time-Delay Neural Networks



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, ...