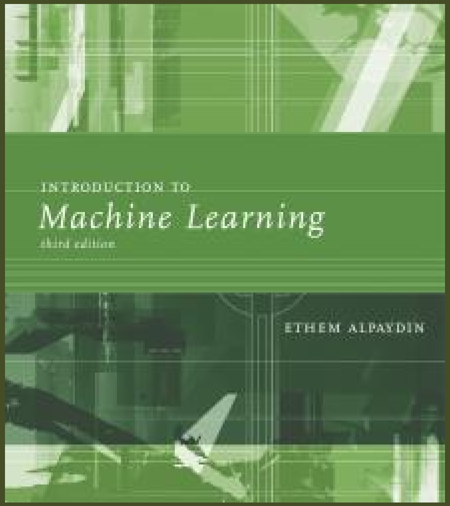


Lecture Slides for

INTRODUCTION TO MACHINE LEARNING

3RD EDITION

ETHEM ALPAYDIN

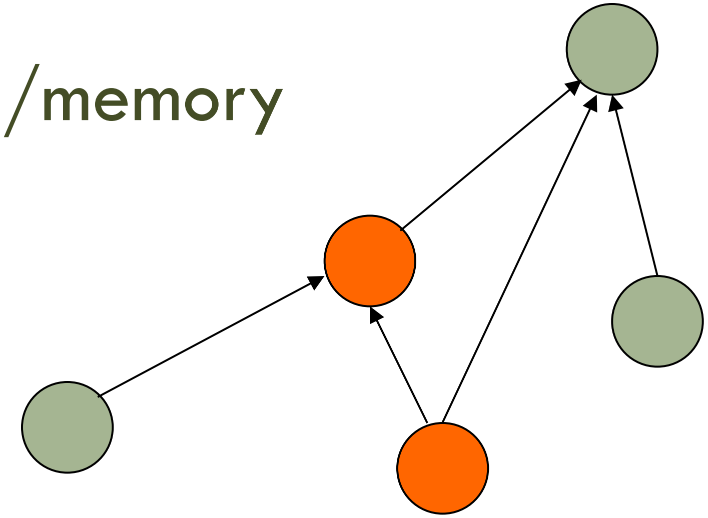
alpaydin@boun.edu.tr
http://www.cmpe.boun.edu.tr/~ethem/i2ml3e

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



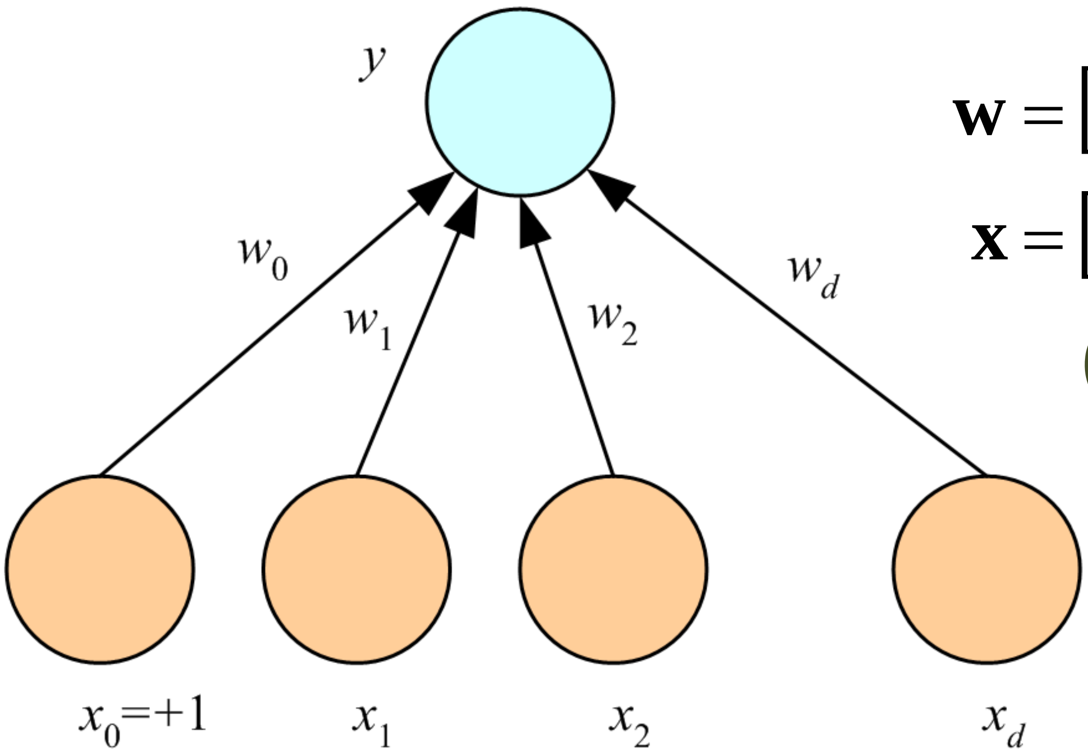
Understanding the Brain

- 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



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

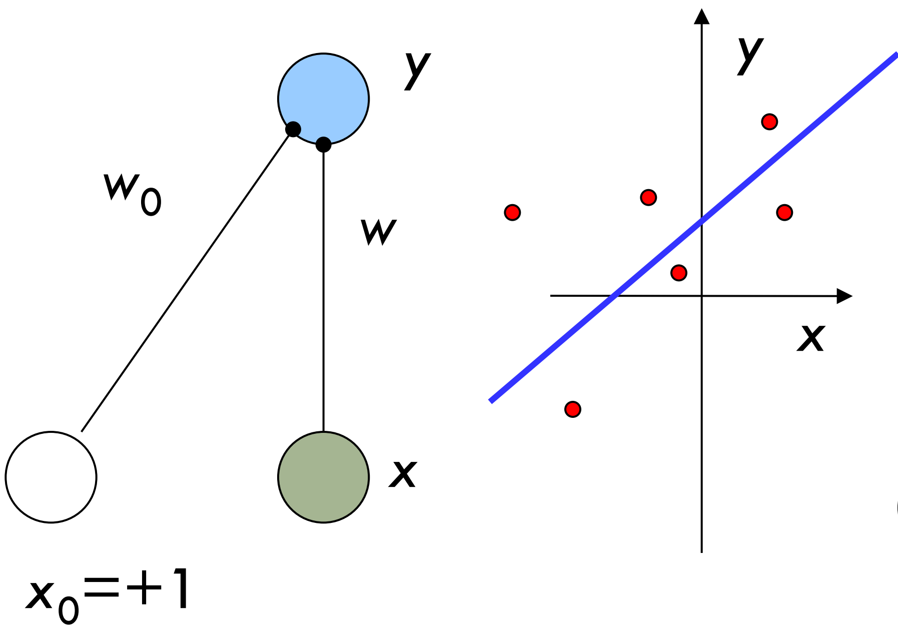
$$\mathbf{w} = \left[w_0, w_1, \ldots, w_d\right]^T$$

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

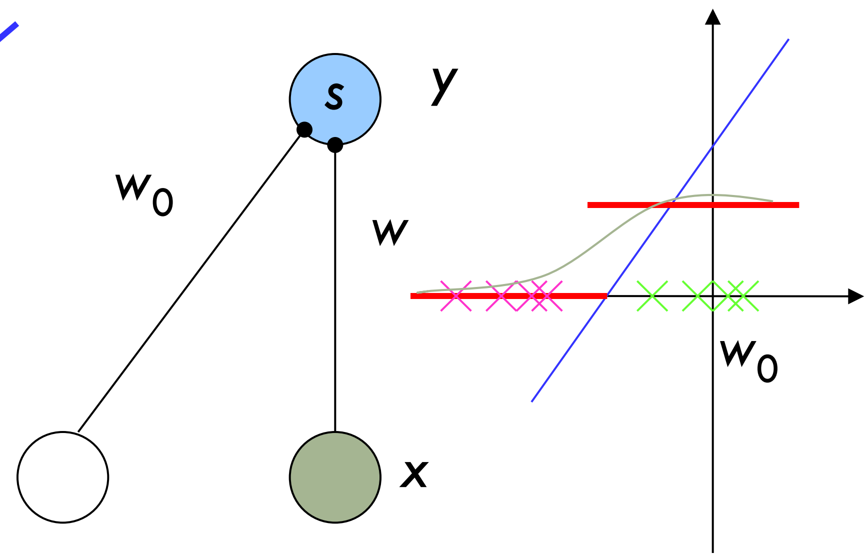
(Rosenblatt, 1962)

What a Perceptron Does

- Regression: $y = wx + w_0$



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



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

K Outputs

Regression:

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

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

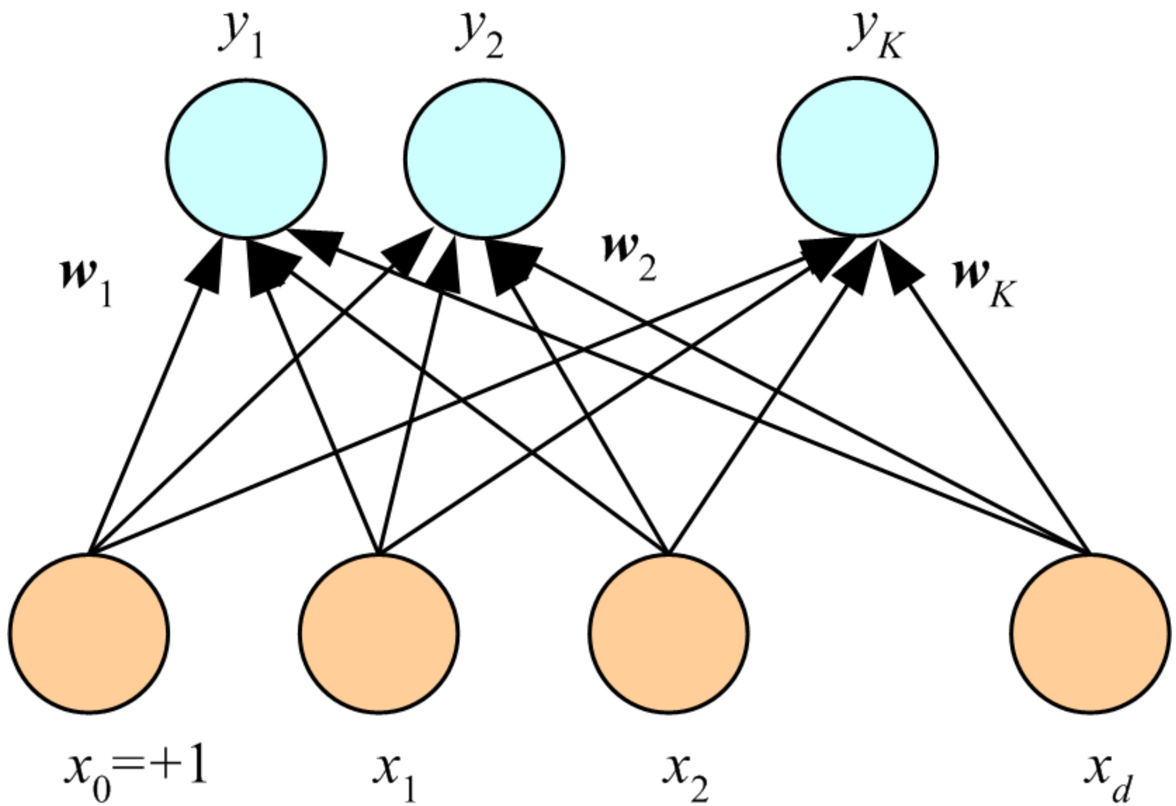
Classification:

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

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

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):

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

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

- Stochastic optimization, online learning
  - Update weights based on one sample at a time
  - Stochastic gradient descent (SGD)

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

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

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

- Stochastic gradient descent: one sample at a time

Classification using Square Loss

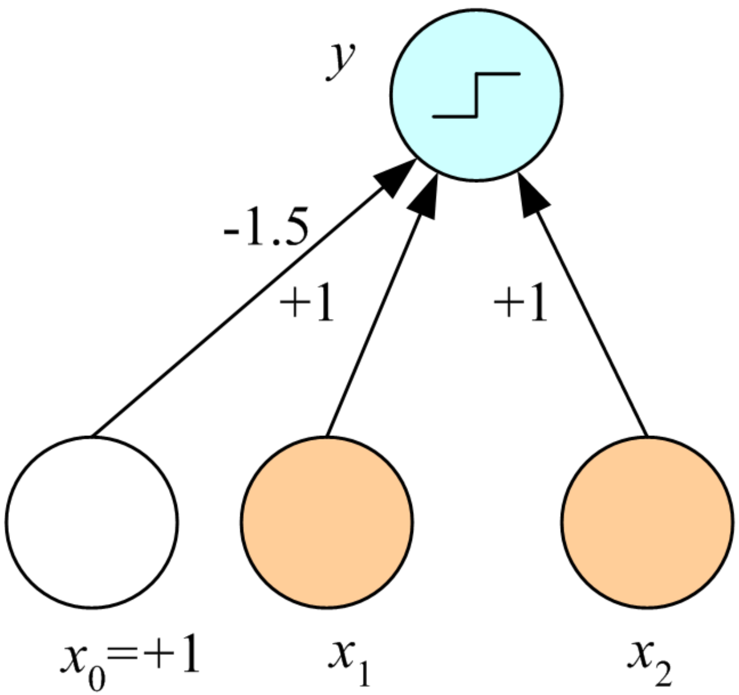
- Single sigmoid output

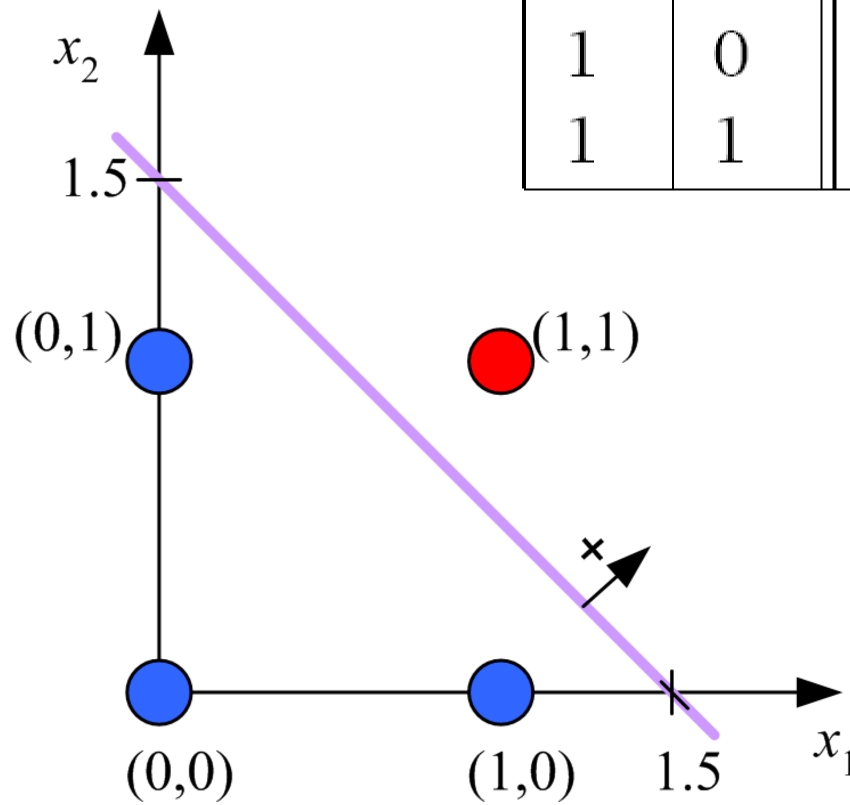
$$y^t = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}^t\right)$$
$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, \mathbf{r}^t\right) = (r^t - y^t)^2$$
$$\Delta w_j^t = \eta\left(r^t - y^t\right) y^t (1 - y^t) x_j^t$$

- Training: Square loss vs. Cross entropy
  - Cross entropy leads to a convex optimization problem
  - Square loss leads to a *non-convex* optimization problem
- Classification with one layer: Logistic regression
  - Soft-max/sigmoid transfer function
  - Cross entropy as loss function

Learning Boolean AND

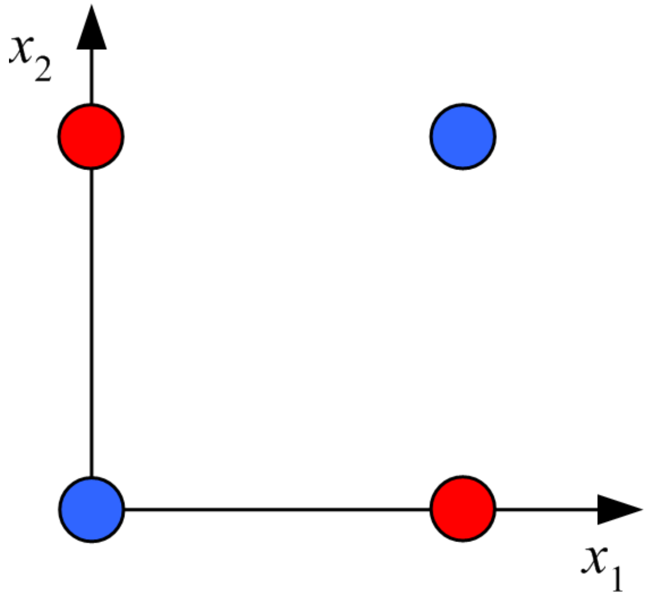
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





XOR

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

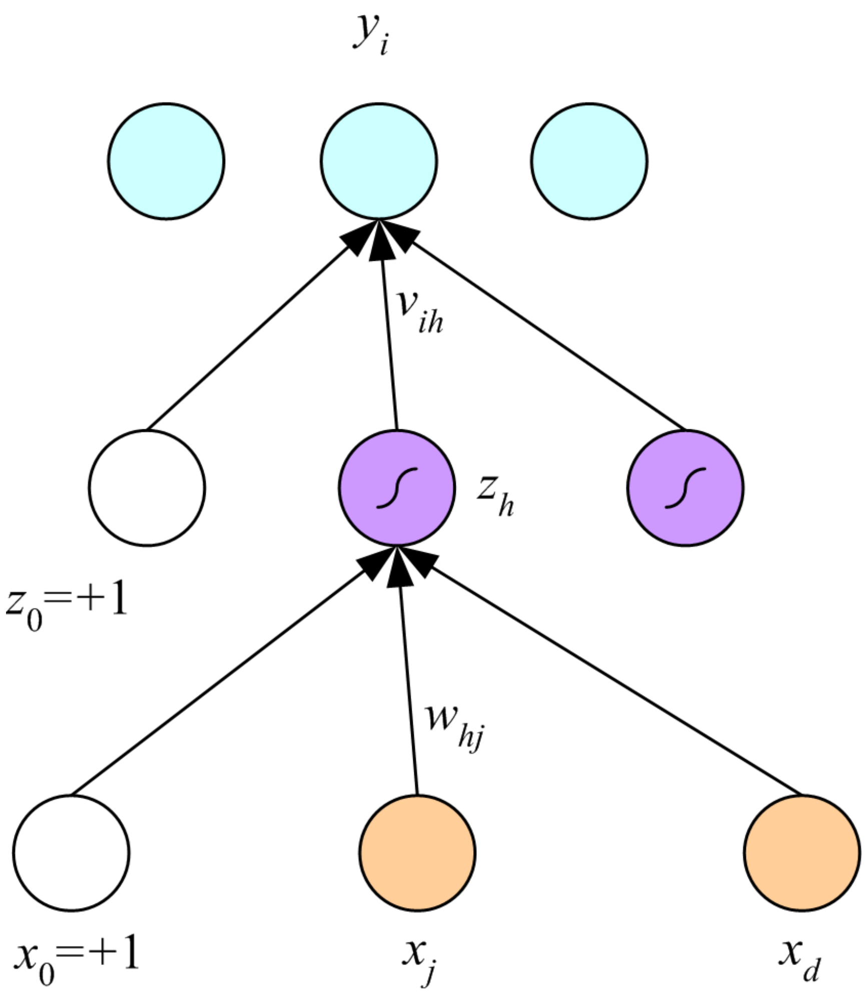


(Minsky and Papert, 1969)

- 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



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

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

(Rumelhart et al., 1986)

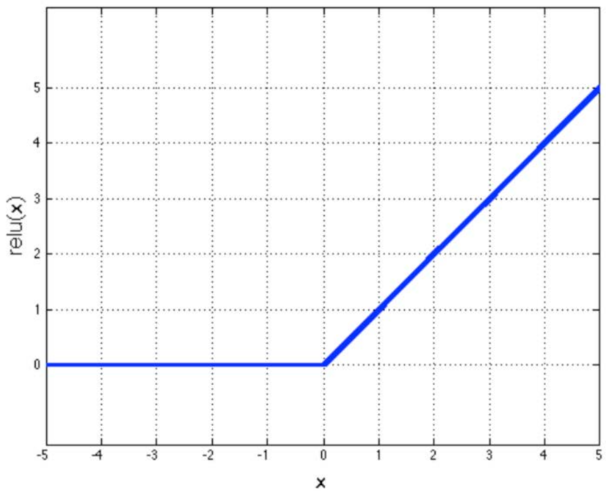
MLP Transfer: Sigmoid vs ReLU

- Traditional transfer function: sigmoid (or tanh)

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

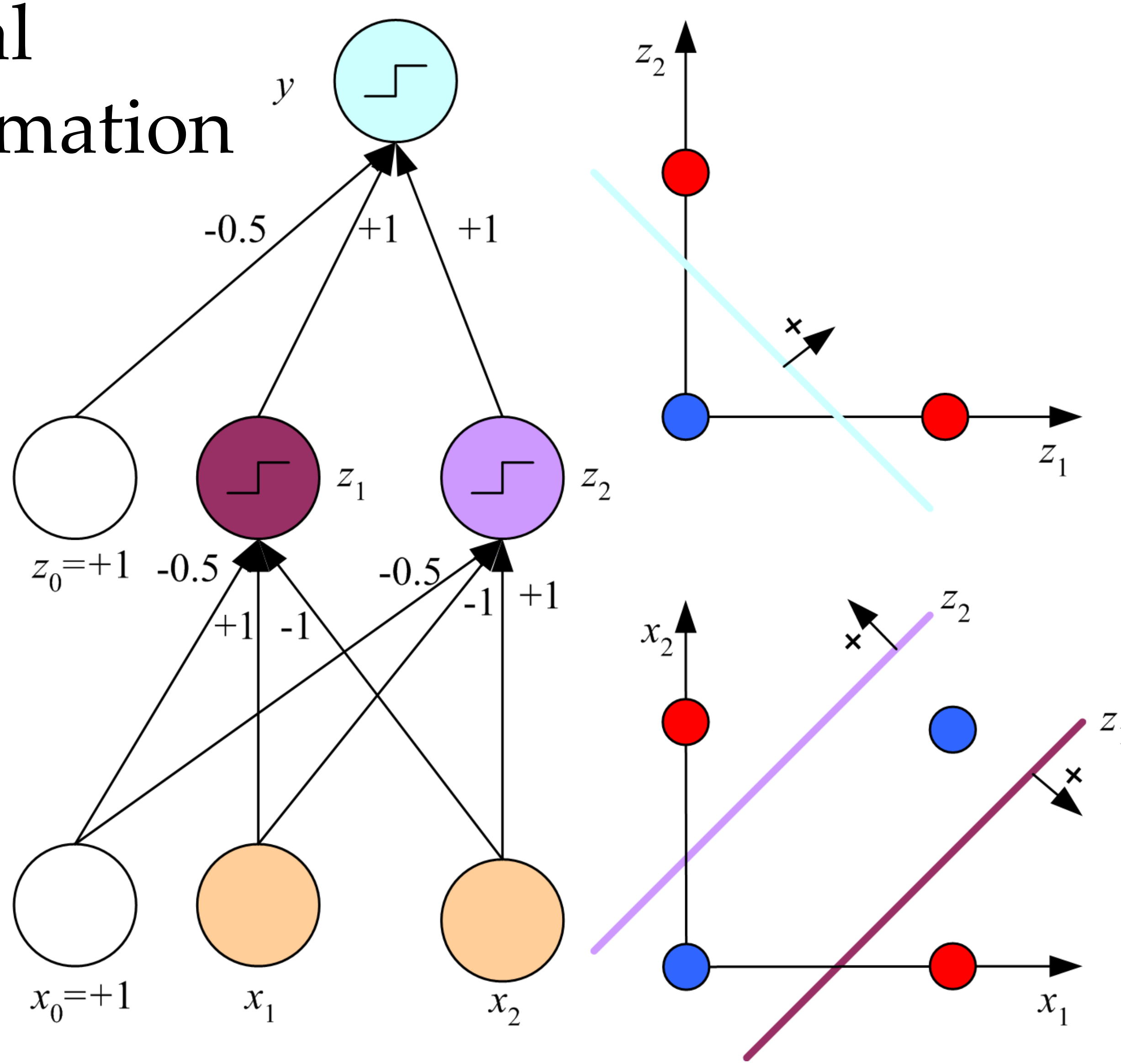
- Rectified Linear Unit (ReLU)

$$z_h = \text{ReLU}\left(\mathbf{w}_h^T \mathbf{x} + w_{h,0}\right) \qquad \text{ReLU}\left(a\right) = \max(0, a)$$



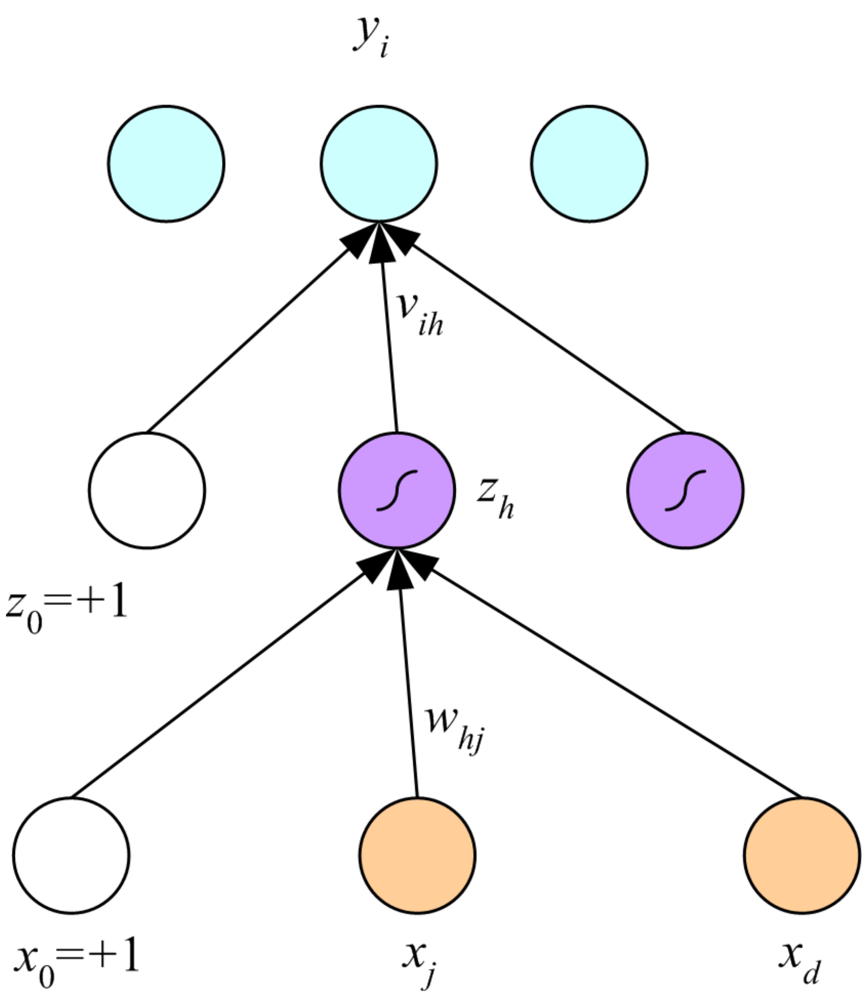
- Use ReLU (and variants)
  - Simplifies gradient descent
  - Makes learning faster
  - Avoids (sigmoid) saturation

Universal Approximation



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

Regression, Backpropagation



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

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

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

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

E is the loss at the output, e.g., $(r^t - y^t)^2$

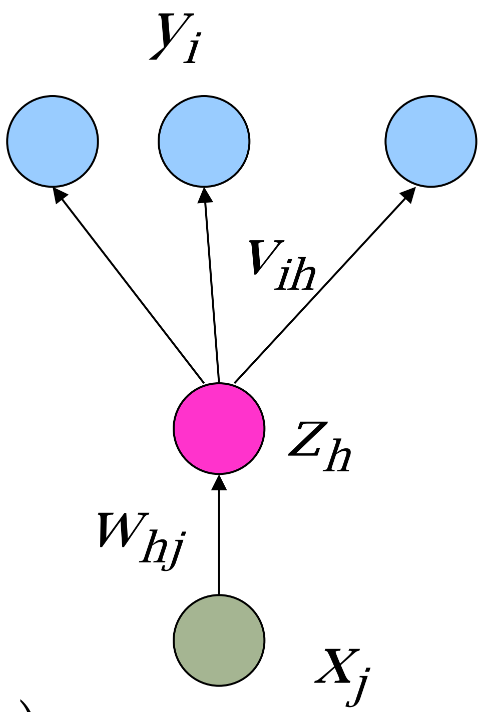
Regression with Multiple Outputs

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

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

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

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



Backprogation

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

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

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

Backward

Forward

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

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

$\mathbf{x}$

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

Backward

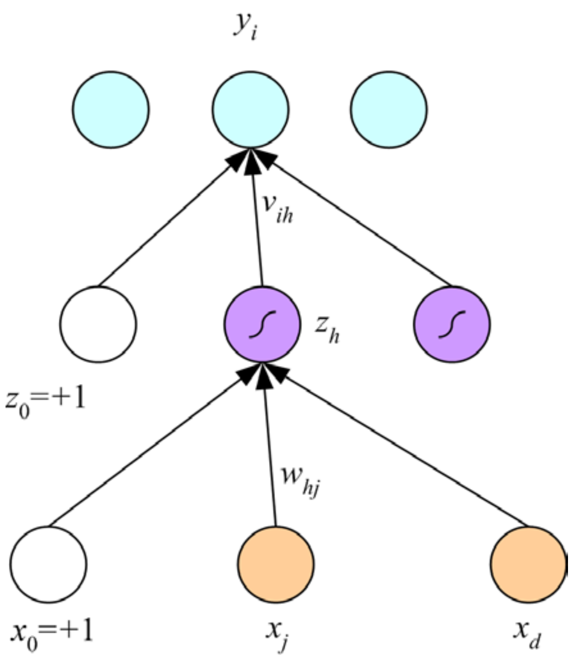
$$\boxed{\Delta v_{i,h}=\eta(r_i^t-y_i^t)z_h^t}$$

$$\boxed{\Delta v_{i,h}=\eta\,\Delta_i^t\ z_h^t=\eta\times\text{error}\times\text{input}}$$

$$\boxed{\Delta_i^t=(r_i^t-y_i^t)=\text{error}}$$

Backward

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



$$\boxed{\Delta w_{h,j}=\eta\,\Delta_h^t\ x_j^t=\eta\times\text{error}\times\text{input}}$$

$$\boxed{\Delta_h^t=\sum_i\Delta_i^t v_{ih}\,g'(a_h)=\text{backprogated error}}$$

g : transfer (e.g., sigmoid)

$$a_h=\sum_j w_{hj}x_j+x_0$$

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

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, …