UNIVERSITÄT ZU LÜBECK

GenAI in Education, Science, and Society

Aktuelle Themen Data Science und
KI / CS5070-KP04

Prof. Dr. Sylvia Melzer

SoSe2026

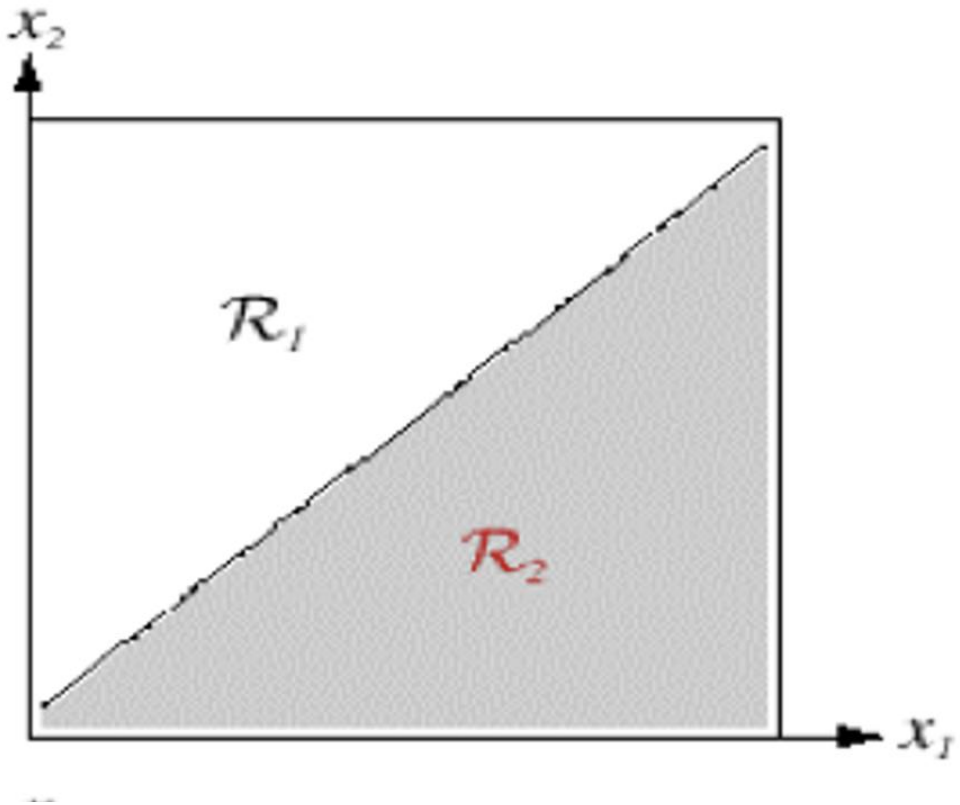
Training Embedding Architectures

GenAI in Education, Science, and Society

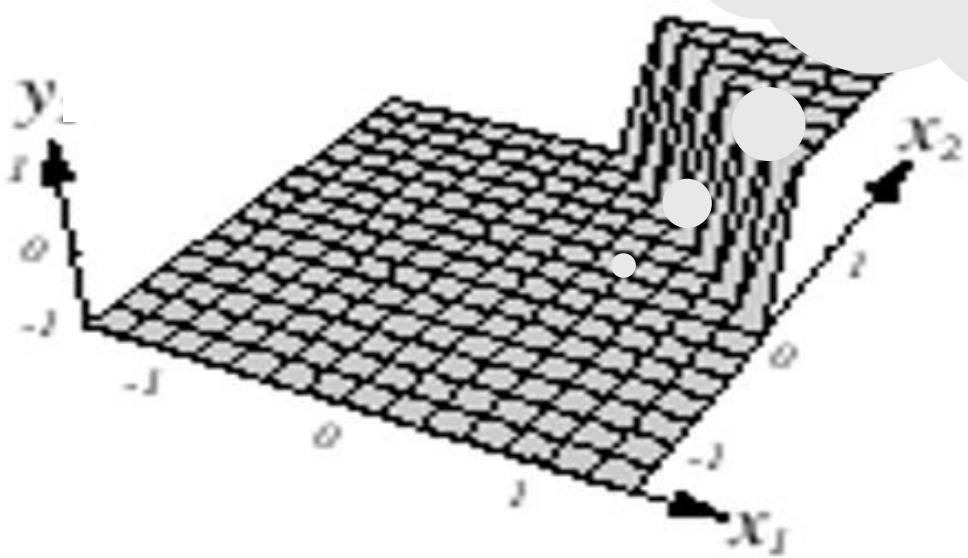
Acknowledgements

These lecture notes were prepared in collaboration with Prof. Dr. habil. Ralf Möller.

Linear Separation (Linear Models)

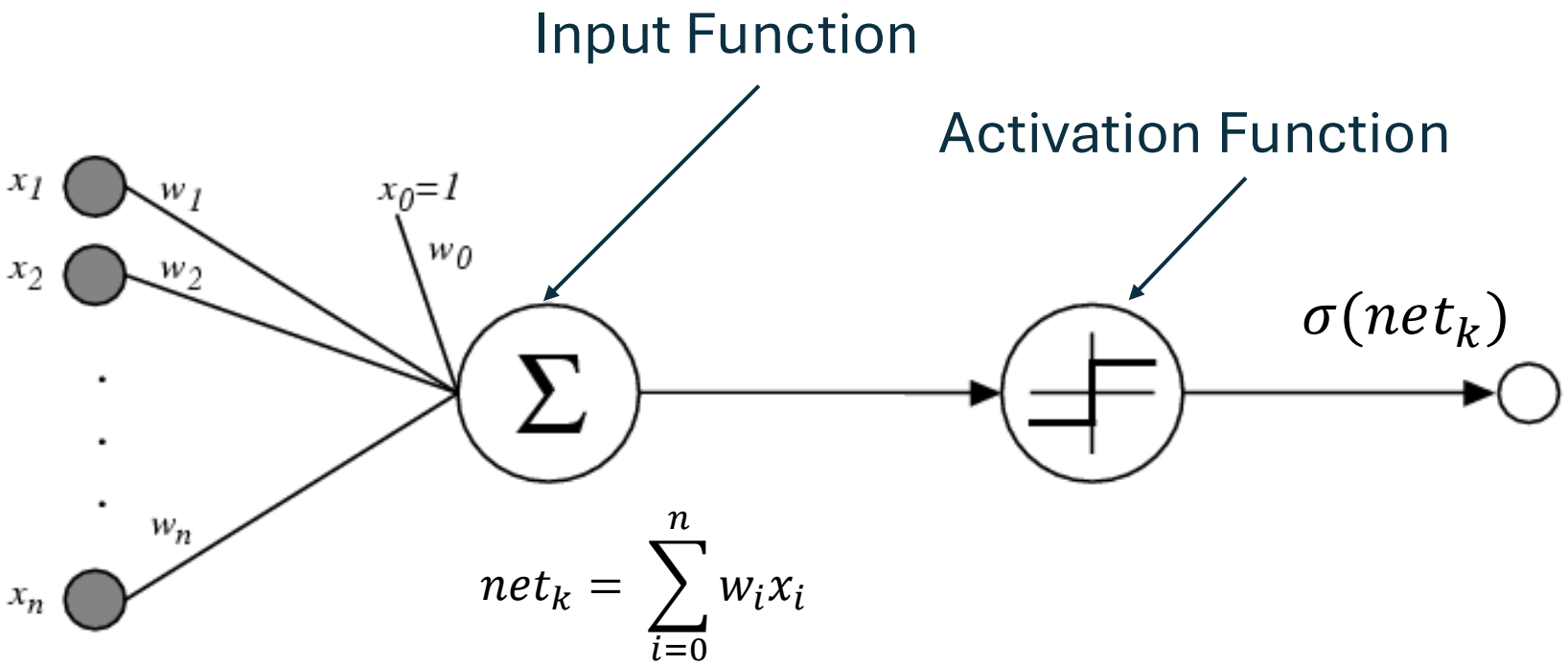


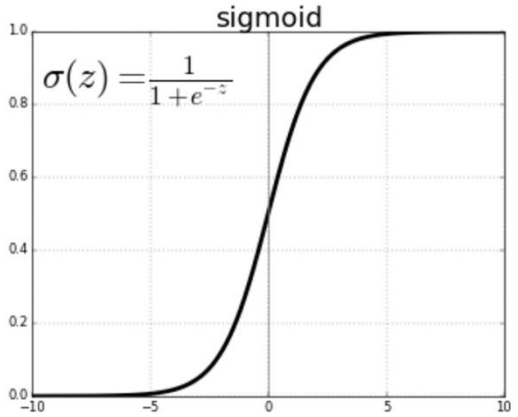
Embedding (x_1, x_2) data into y



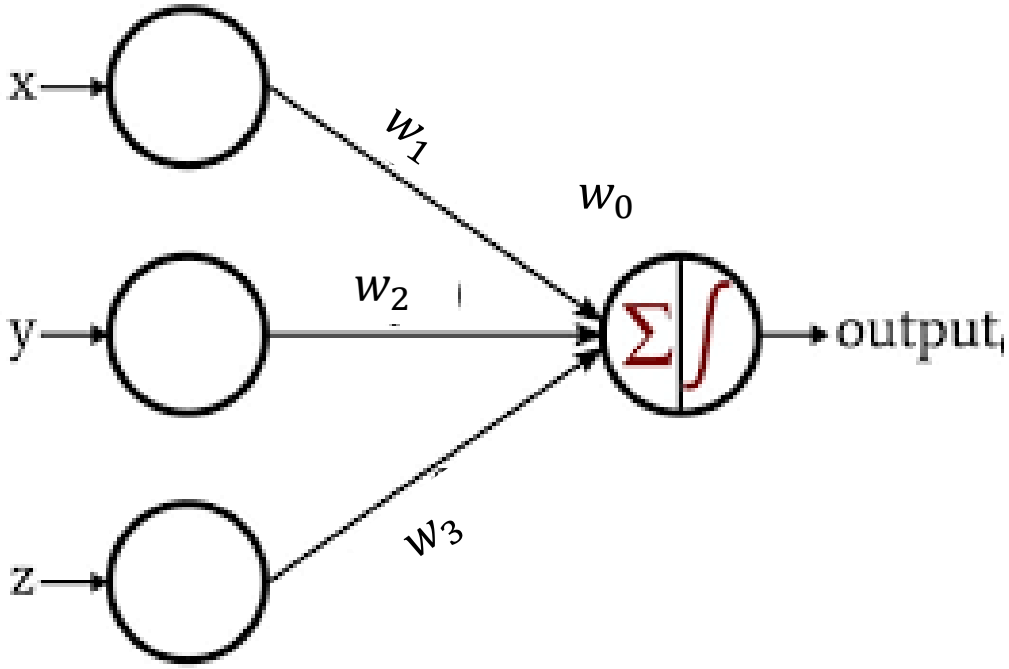
Perceptron

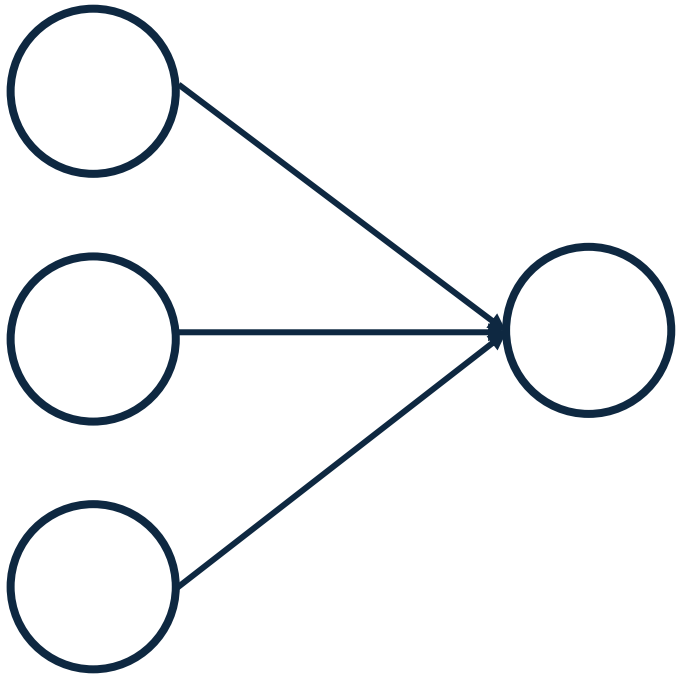
Frank Rosenblatt, The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory, **1957**





Perceptron (simplified views)



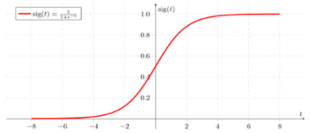


Other Examples for Activation Functions

- Linear activation: $a_k = f_{act}(net_k) = c_k \cdot net_k$

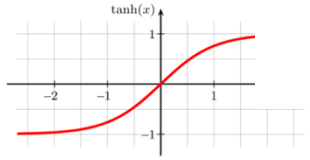
Threshold (often 0)

- Threshold activation: $a_k = f_{act}(net_k) = \begin{cases} 1, & \text{falls } net_k \geq \theta_k, \\ 0, & \text{sonst.} \end{cases}$
- Logistic activation: $a_k = f_{act}(net_k) = \frac{1}{1+e^{-\frac{net_k}{T}}}$



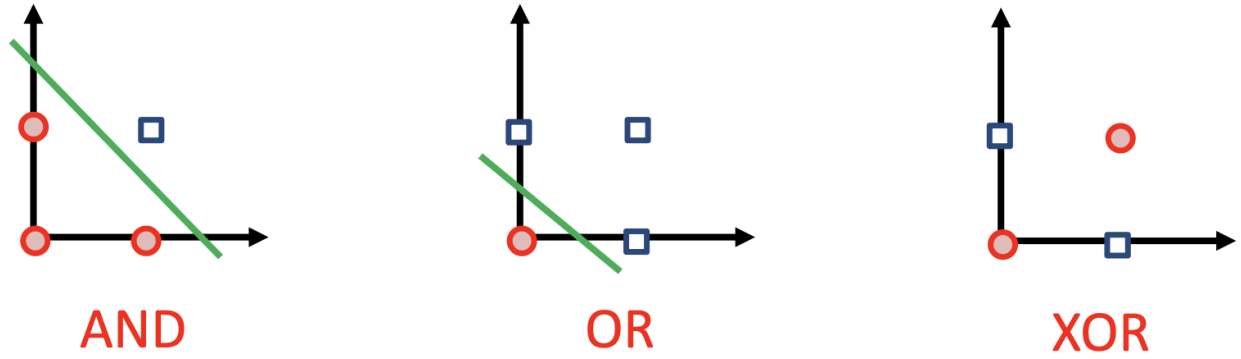
Special case Sigmoid: T=1

- Hyperbolic tangent: $a_k = f_{act}(net_k) = \frac{e^{net_k} - e^{-net_k}}{e^{net_k} + e^{-net_k}} = \frac{1+\tanh(net_k)}{2}$



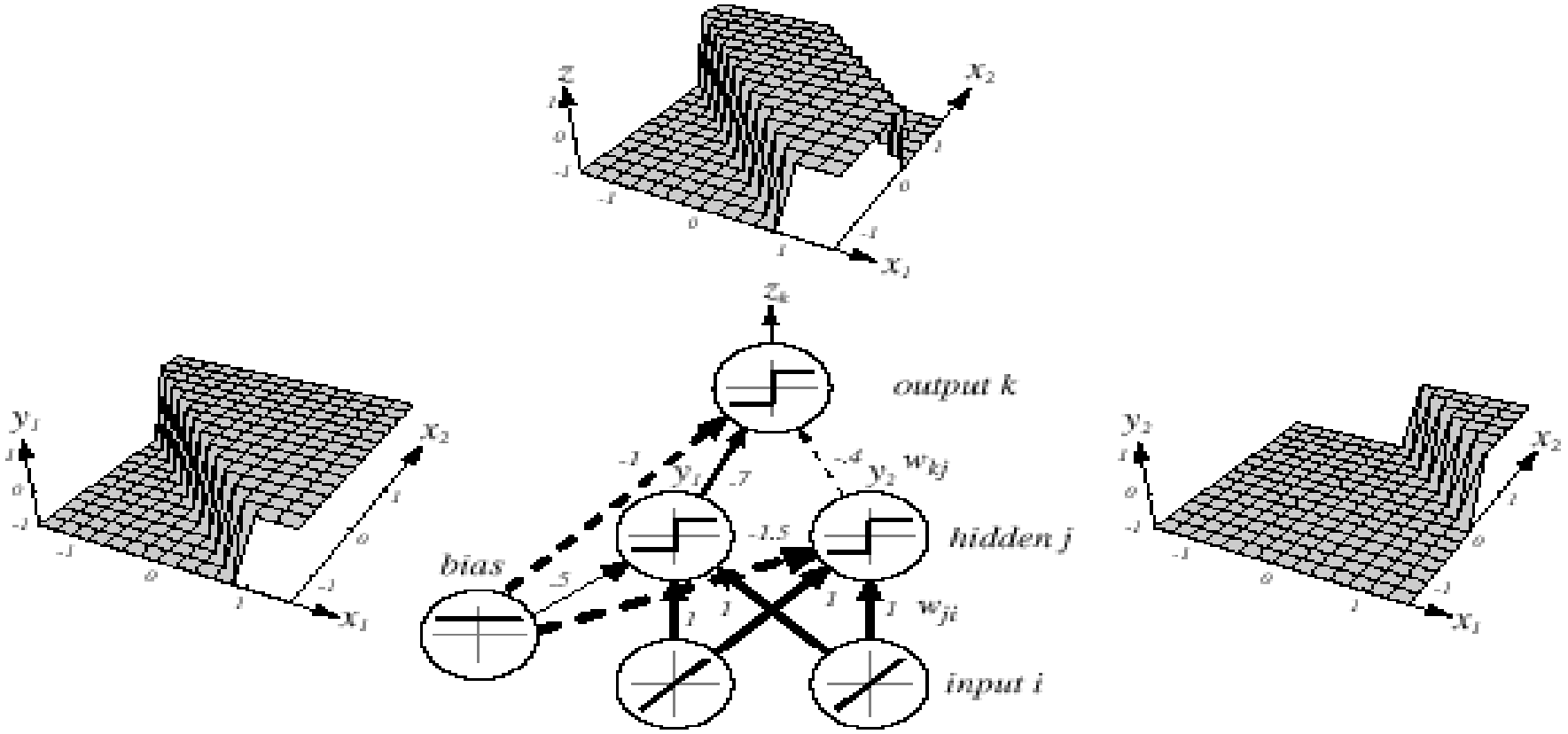
Limitations of Linear Models

Some function are linearly separable, but many are not

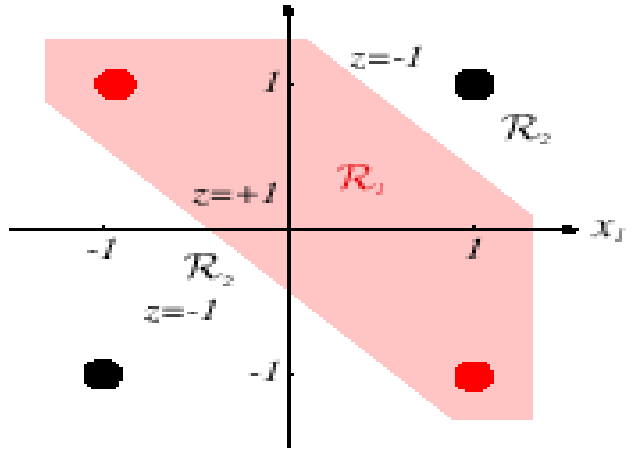


Possible solution: composition

$$x_1 \text{ XOR } x_2 = (x_1 \text{ OR } x_2) \text{ AND not}(x_1 \text{ AND } x_2)$$



Z = x1 XOR x2 = (x1 OR x2) AND NOT(x1 AND x2)



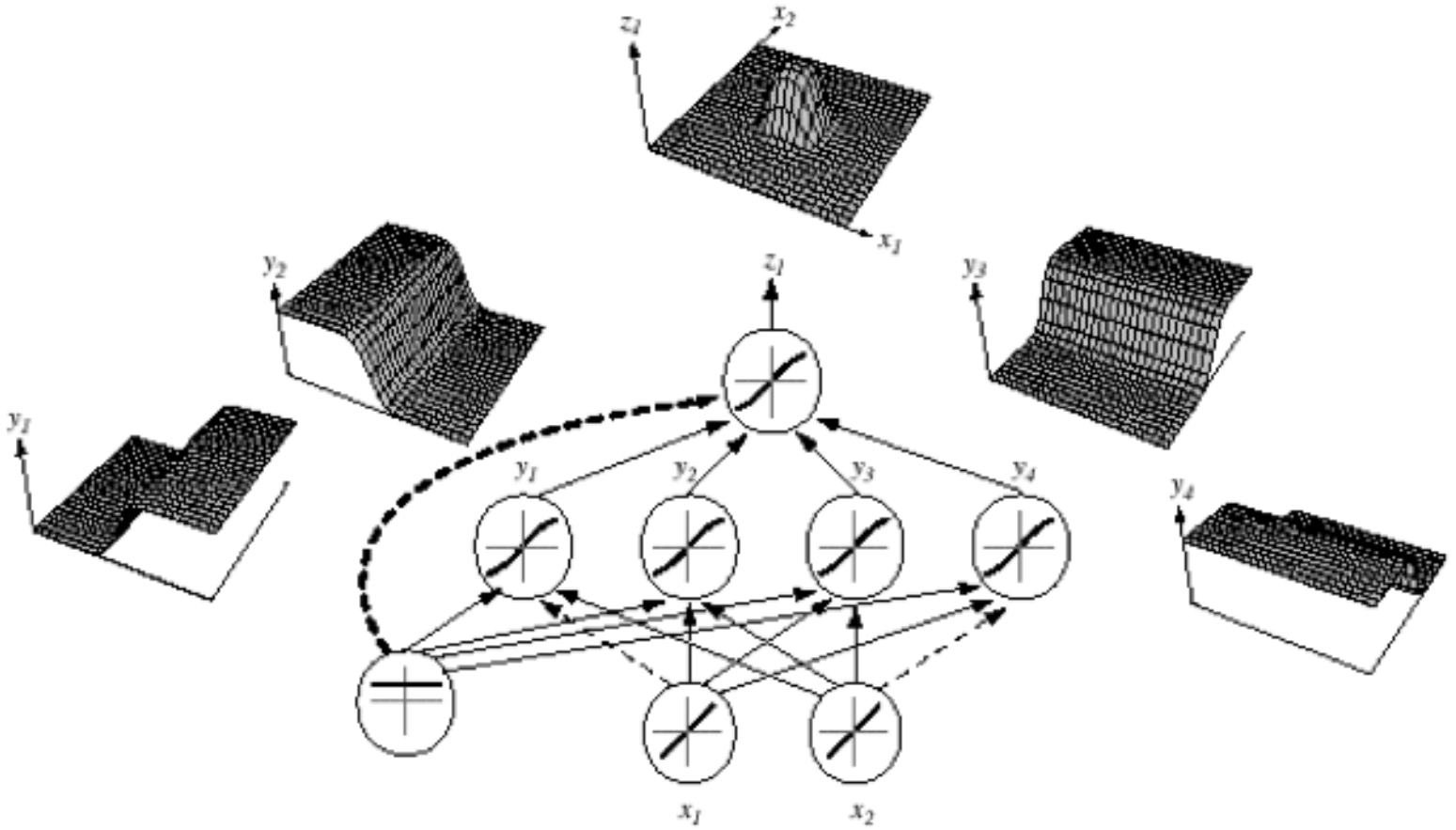


FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a "bump" at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

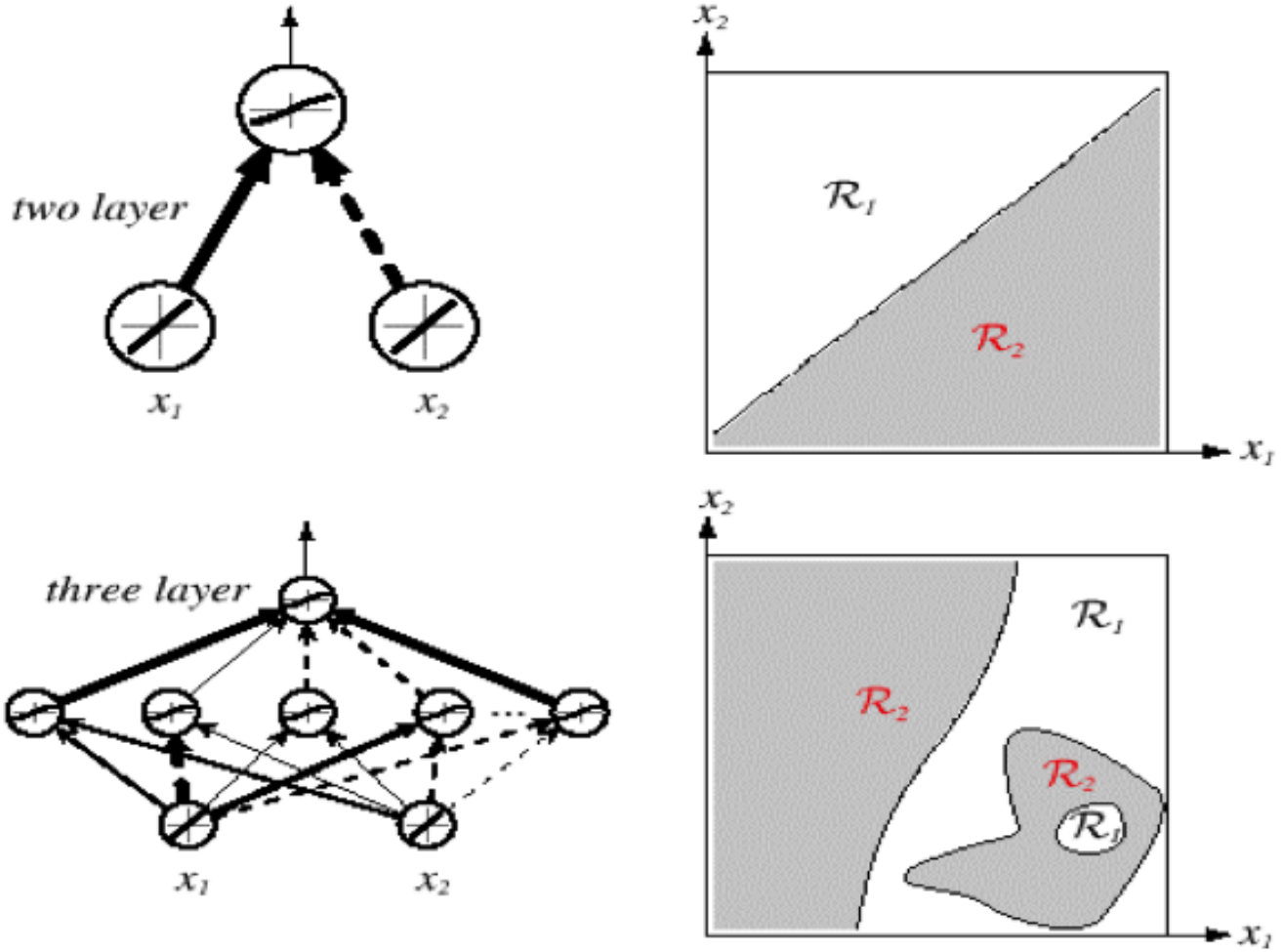
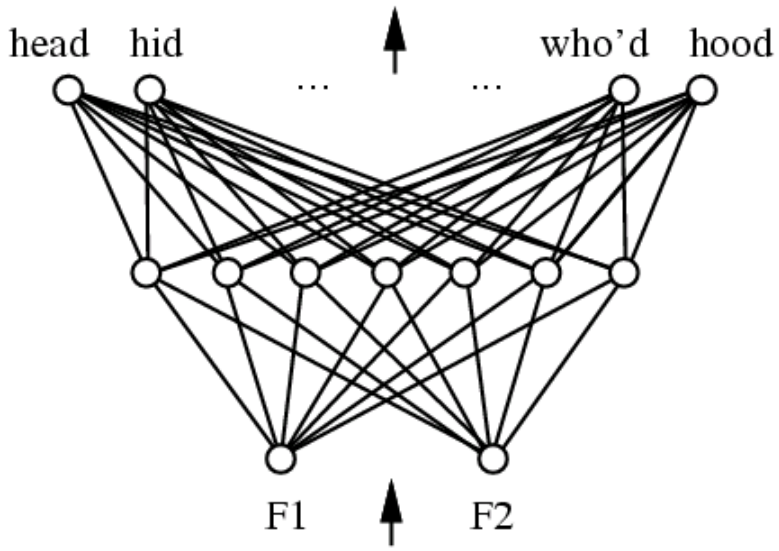
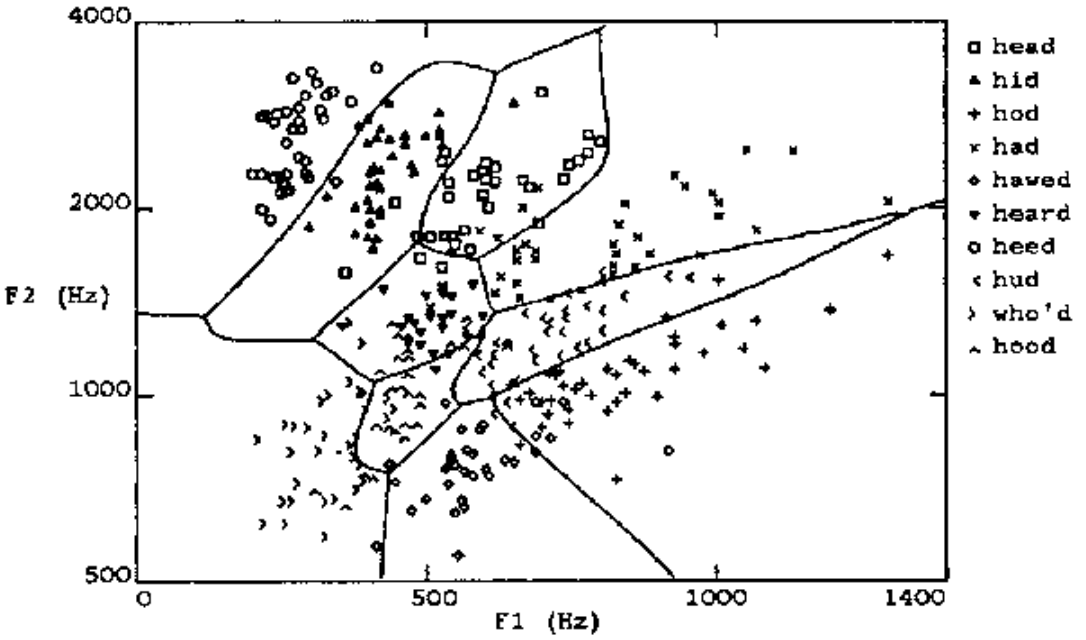


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Multi-level networks of sigmoid units



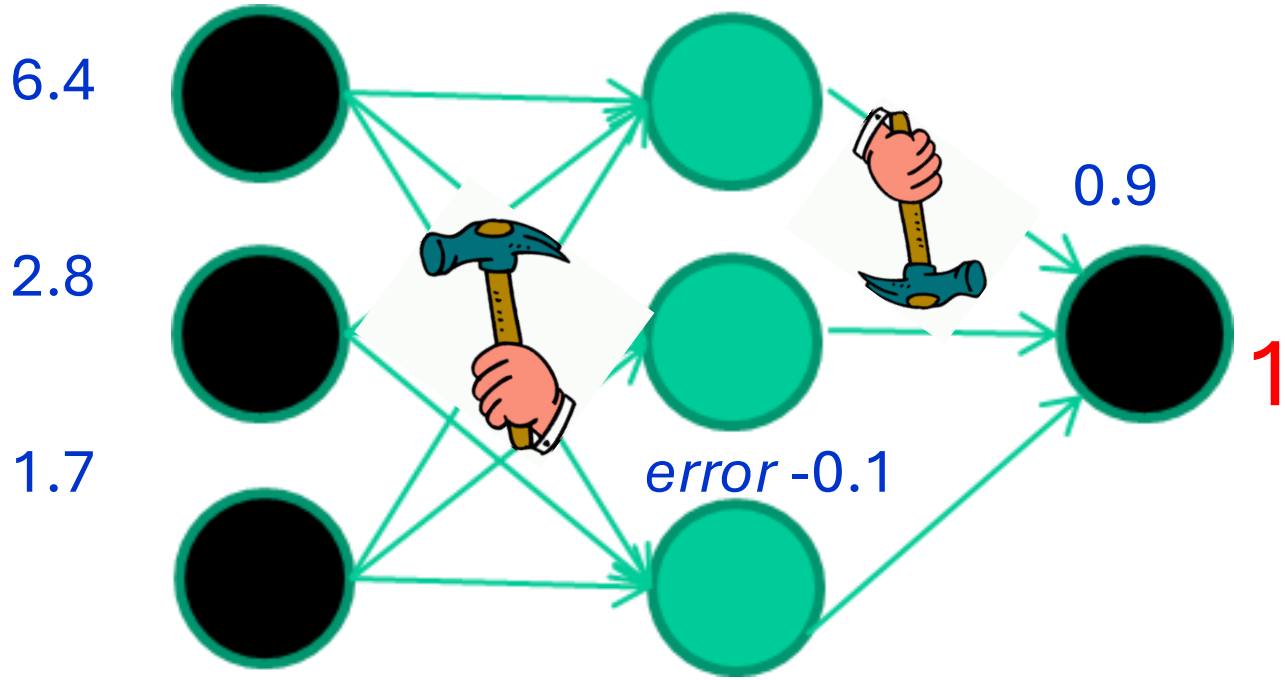


Adaptation of mapping parameters (“weights”)

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

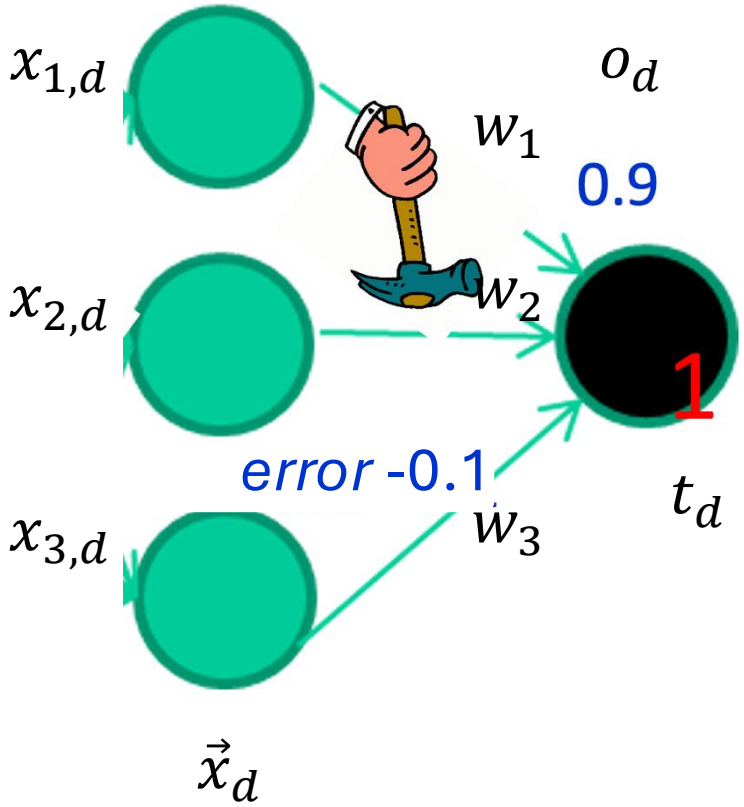
Simple case $o = Wx$



Delta Learning Rule for Perceptrons

- Given: Set T of training data $\{(\vec{x}_1, t_1), \ldots, (\vec{x}_N, t_N)\}$
- Select training datum $d \in T$:
  - Input $\vec{x}_d$
  - Target t_d
- Output o_d
- ${w_i}^{k+1} = \Delta w_i + {w_i}^k$

How to compute Δw_i?



D. Hebb: The organization of behavior. A neuropsychological theory. Erlbaum Books, Mahwah, N.J., **1949**

Derivation of a suitable update value Δw_i

- Idea: minimize quadratic error

  .

  - t_d value for $d \in T$ an arbitrarily selected ("sampled") datum $d = (\vec{x}_d, t_d)$ from dataset T
  - o_d output for $\vec{x}_d$

$$L[\vec{w}] = \frac{1}{2}(t_d - o_d)^2$$

$$\bar{} \quad \overline{d \in D}$$

- Determine minimum with 1st derivative

Gradient Descent

$$L[\vec{w}] = \frac{1}{2}(t_d - o_d)^2$$

$L(w_i)$

- Gradient

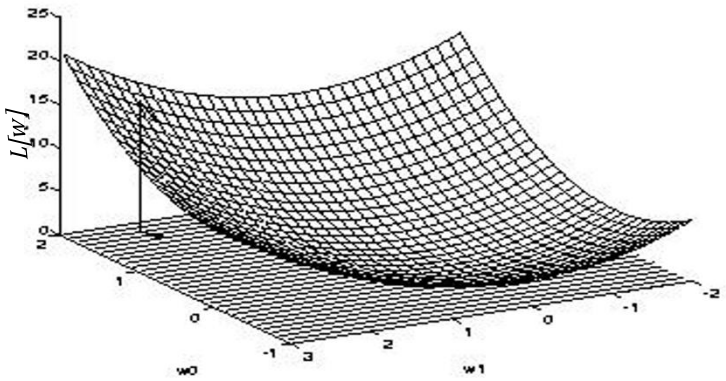
$$\Delta L[\vec{w}] \equiv \left[\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \dots, \frac{\partial L}{\partial wn}\right]$$

- Learning rule

$$\Delta \vec{w} = -\eta \, \Delta L[\vec{w}]$$

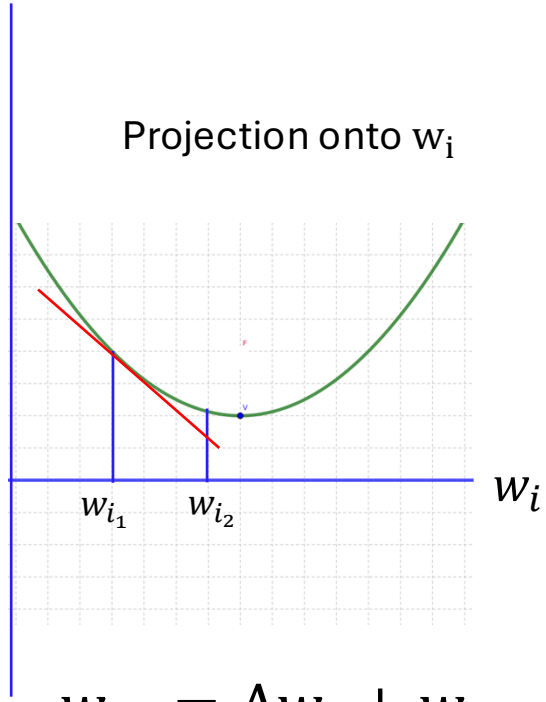
- i.e.

$$\Delta \vec{w_i} = -\eta \frac{\partial L}{\partial wi}(\vec{w})$$



Visualization of L for $\vec{w} = (w_0, w_1)$

How to compute the derivative



$$w_{i_2} = \Delta w_i + w_{i_1}$$

What if only linear mappings were considered?

$$\frac{\partial L}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}(t_d - o_d)^2$$
$$= \frac{1}{2}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$
$$= \frac{1}{2}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$
$$= (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$
$$= (t_d - o_d)\left(-x_{i,d}\right) = -(t_d - o_d)x_{i,d}$$

Multivariate linear regression

$$\Delta\vec{w_i} = -\eta\frac{\partial L}{\partial wi}(\vec{w})$$

Iteration for sampled data points d

$$\Delta w_i = \eta(t_d - o_d)x_{i,d}$$

With sigmoid

$$L[\vec{w}] = \frac{1}{2}(t_d - o_d)^2$$

$$\sigma(x) = 1/(1 - e^{-x})$$

$$\sigma'(x) = \sigma(x)(1\text{-}\,\sigma(x))$$

Multivariate logistic regression

$$\begin{aligned}
\frac{\partial L}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}(t_d - o_d)^2 \\
&= \frac{1}{2}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \sigma(\vec{w}\cdot\vec{x}_d)) \\
&= (t_d - o_d)\left(-x_{i,d}\sigma'(\vec{w}\cdot\vec{x}_d)\right) \\
&= -(t_d - o_d)x_{i,d}\sigma'(\vec{w}\cdot\vec{x}_d) \\
&= -(t_d - o_d)x_{i,d}\sigma(\vec{w}\cdot\vec{x}_d)\left(1 - \sigma(\vec{w}\cdot\vec{x}_d)\right) \\
&= -(t_d - o_d)x_{i,d}o_d(1 - o_d)
\end{aligned}$$

Stochastic Gradient Descent (for sampled data point)

- Gradient

$$\Delta L[\vec{w}] \equiv \left[\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial wn}\right]$$

- Learning rule

$$\Delta \vec{w} = -\eta \, \Delta L[\vec{w}]$$

- i.e.

$$\Delta \vec{w_i} = -\eta \frac{\partial L}{\partial wi}(\vec{w})$$

$$\frac{\partial L}{\partial w_i} = -(t_d - o_d) x_{i,d} o_d (1 - o_d)$$

Iteration for sampled data points d

$$\Delta w_i = \eta (t_d - o_d) x_{i,d} o_d (1 - o_d)$$

Stochastic Gradient Descent for batch of data

$$\Delta w_i = \eta \frac{1}{N} \sum_{d \in D} (t_d - o_d) o (1 - o_d) x_{i,d}$$

Given **batch of data** D sampled from from training set T:

$$D = \{d_1, \ldots, d_N\} \subseteq T:$$

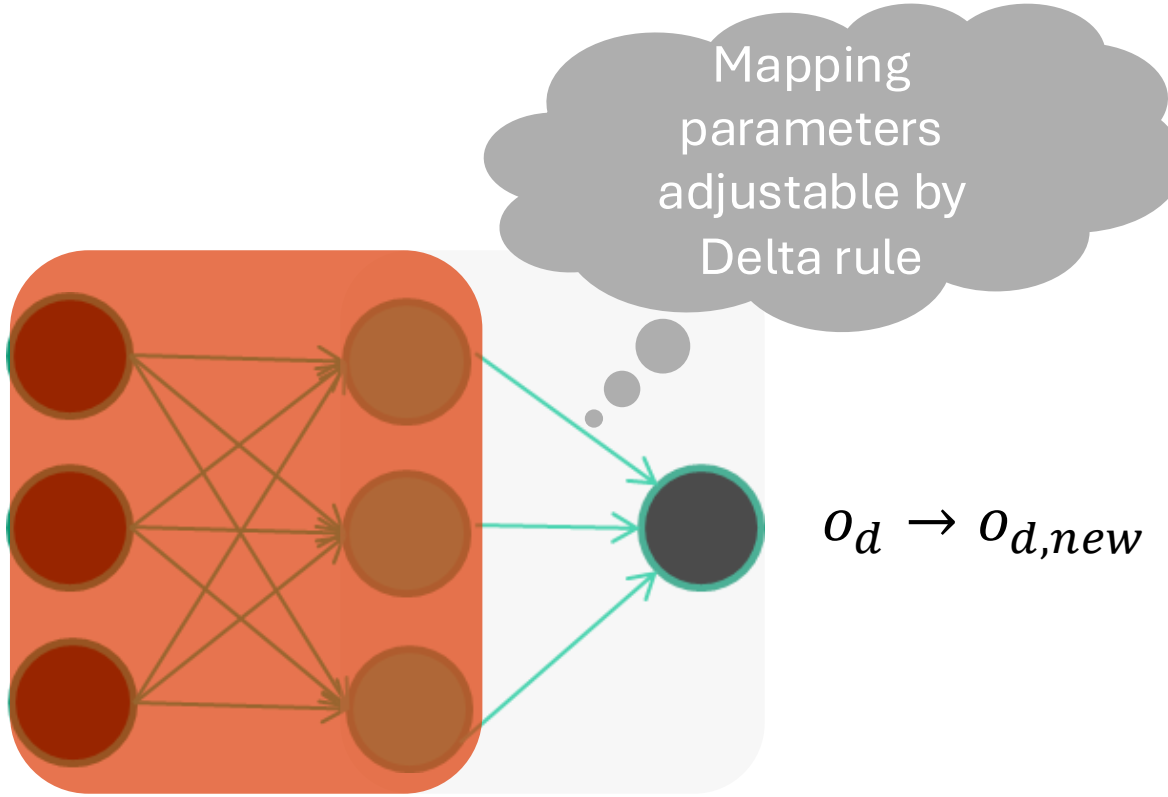
Learning Parameters (“Hyper parameters”)

- Learning rate η
- Batch size
  - Number of samples processed before model is updated
- Epochs
  - Number of complete passes through the training dataset

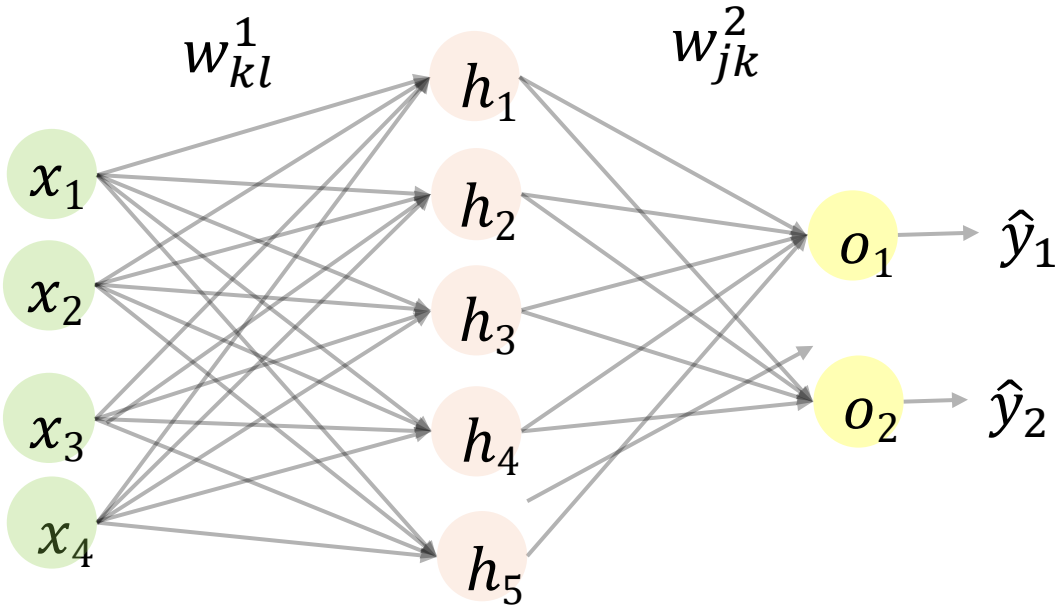
$$\Delta w_i = \eta \frac{1}{N} \sum_{d \in D} (t_d - o_d) o (1 - o_d) x_{i,d}$$

Multilayer Perceptron

- General layer: Parallel composition of perceptrons
- Multilayer perceptron: series of parallel compositions
- How to train based on forward pass?
- We can adjust the mapping parameters of the last layer
- How to adjust parameters of previous layers?
  - Problem: no targets t for previous layer given
  - Idea: Propagate change from o_d to $o_{d,new}$ to input value changes (green circles) to obtain a target t for previous layer (→ Backpropagation)



Input layer | Hidden layer(s) | Output layer



$$\mathbb{R}^4 \xrightarrow{\boldsymbol{g}} \mathbb{R}^5 \xrightarrow{\boldsymbol{f}} \mathbb{R}^2$$
$$\boldsymbol{x} \mapsto \boldsymbol{h} \mapsto \hat{\boldsymbol{y}}$$

k: Layer k
$\boldsymbol{W}^k$: Weight matrix for k

Matrix form computations

$$\hat{\boldsymbol{y}} = \boldsymbol{f}\left(\boldsymbol{g}\left(\boldsymbol{x}; \boldsymbol{W}^1\right); \boldsymbol{W}^2\right)$$
$$= \boldsymbol{\sigma}(\boldsymbol{W}^2\boldsymbol{\sigma}(\boldsymbol{W}^1\boldsymbol{x}))$$

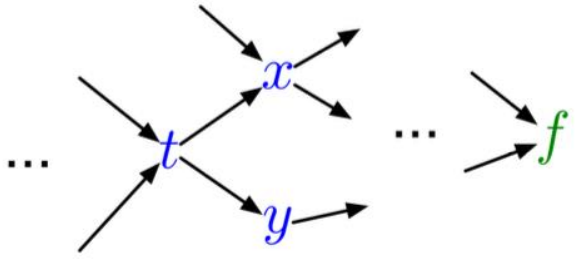
Recap: Multivariate chain rule $f(x, y)$

- In the context of backpropagation:

Mathematical expressions to be evaluated

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

Values already computed by our program

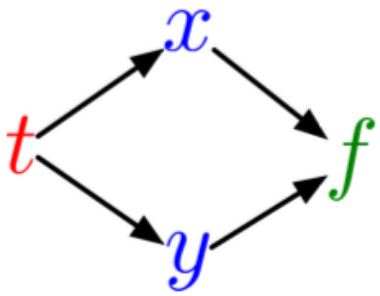


- In our notation:

$$\bar{t} = \bar{x}\frac{\mathrm{d}x}{\mathrm{d}t} + \bar{y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

- Suppose we have a function $f(x, y)$ and functions $x(t)$ and $y(t)$. (All the variables here are scalar-valued.) Then

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$



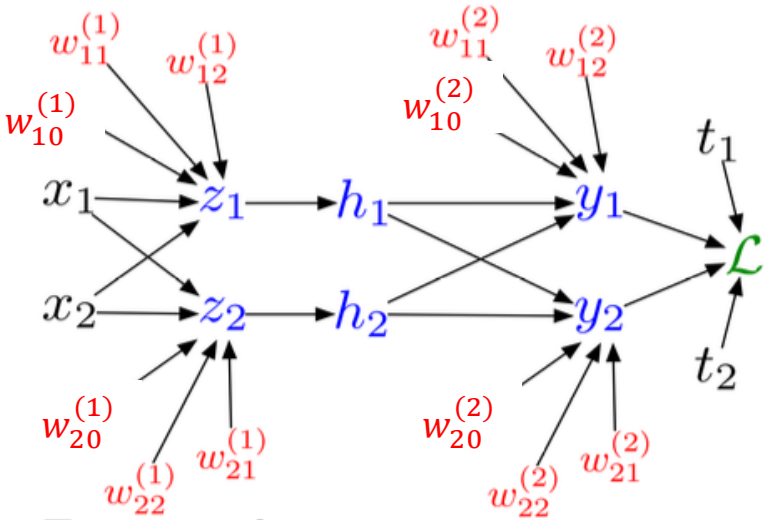
- Example:

$$f(x, y) = y + e^{xy}$$
$$x(t) = \cos t$$
$$y(t) = t^2$$

- Plug in to Chain Rule:

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}t} &= \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t\end{aligned}$$

Multilayer Perceptron (multiple outputs):



Forward pass:

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}} (y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k} h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

$$\overline{z_i} = \overline{h_i} \sigma'(z_i)$$

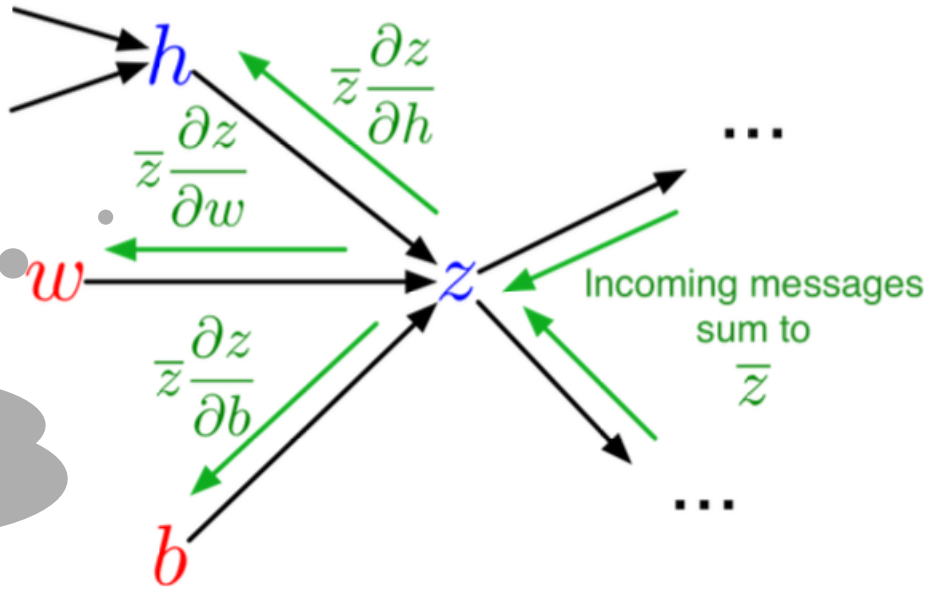
$$\overline{w_{ij}^{(1)}} = \overline{z_i} x_j$$

$$\overline{b_i^{(1)}} = \overline{z_i}$$

Backprop as message passing:

Adaptation of mapping parameters

Derivation of function term for z w.r.t. w as discussed in the above examples



- Each node receives a bunch of messages from its children, which it aggregates to get its error signal. It then passes messages to its parents.
- This provides modularity, since each node only has to know how to compute derivatives with respect to its arguments, and doesn't have to know anything about the rest of the graph.

Full backpropagation algorithm:

Let $v_1, \ldots, v_N$ be a topological ordering of the computation graph (i.e. parents come before children.)

v_N denotes the variable we're trying to compute derivatives of (e.g. loss).

forward pass

For $i = 1, \ldots, N$

Compute v_i as a function of $\mathrm{Pa}(v_i)$

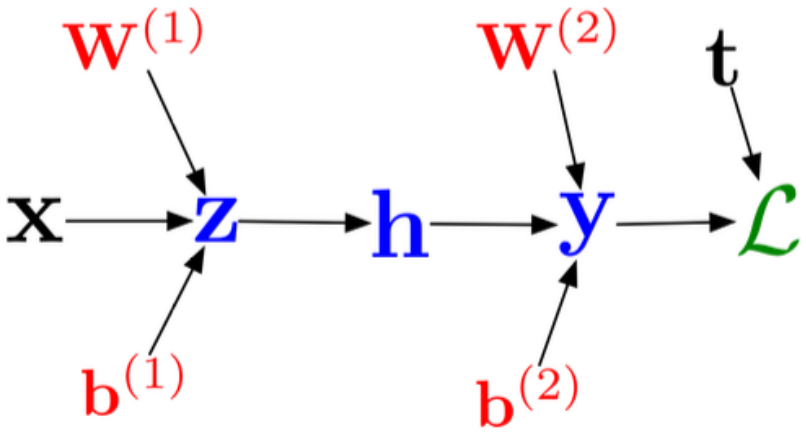
backward pass

$\overline{v_N} = 1$

For $i = N-1, \ldots, 1$

$\overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$

In vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$

$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$

$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$

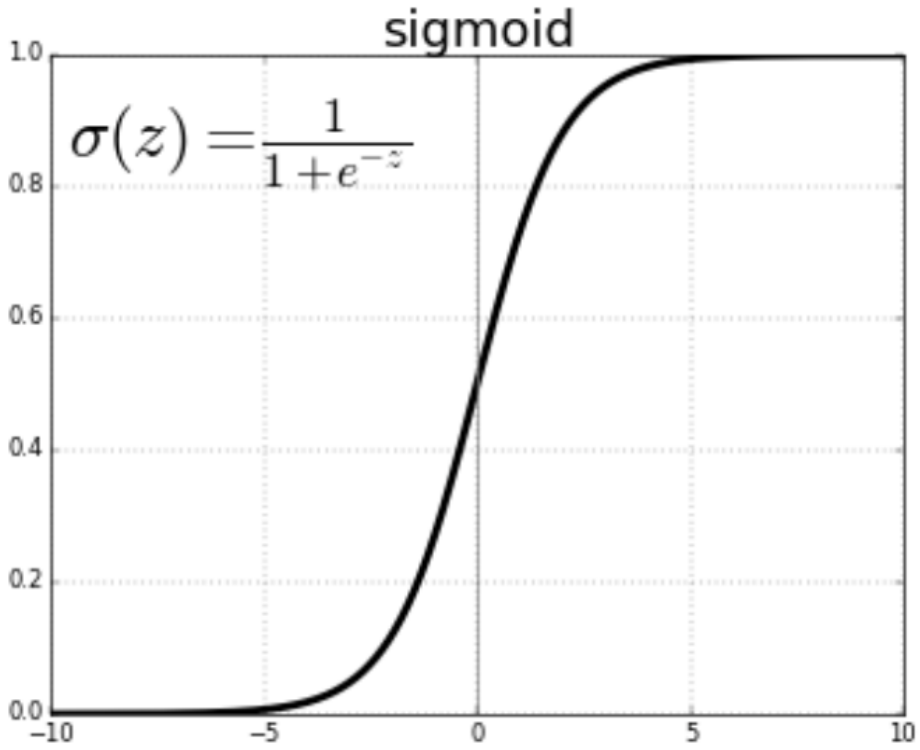
$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$

$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$

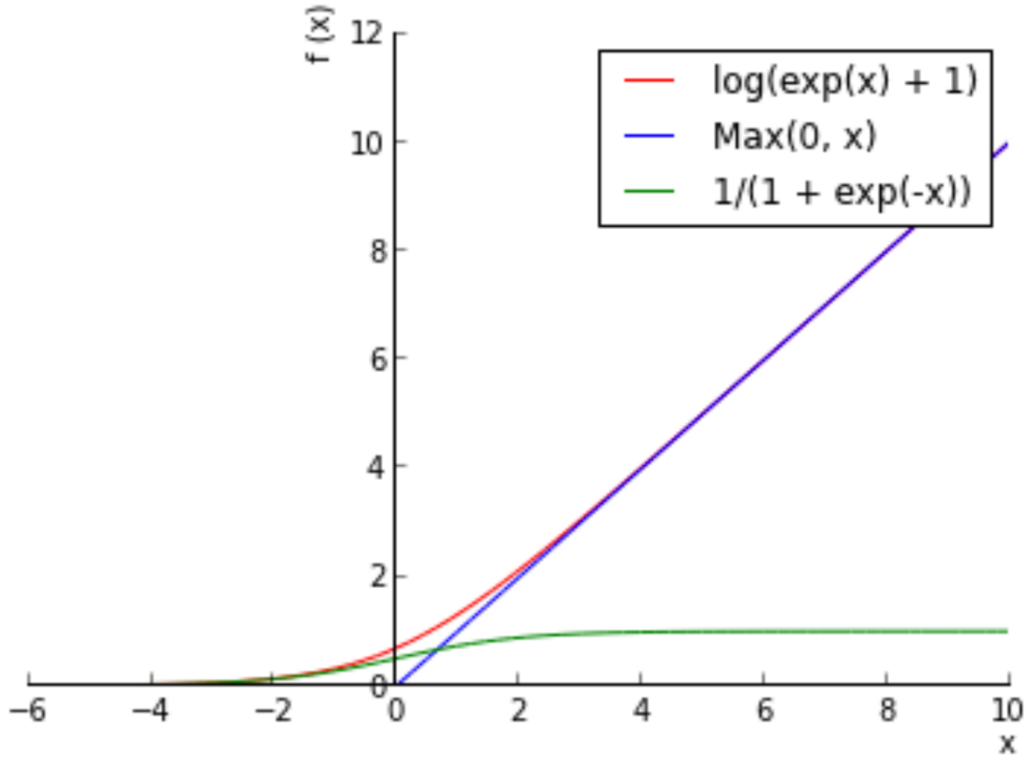
$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$

$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Sigmoid vs. ReLU (Rectified Linear Unit)



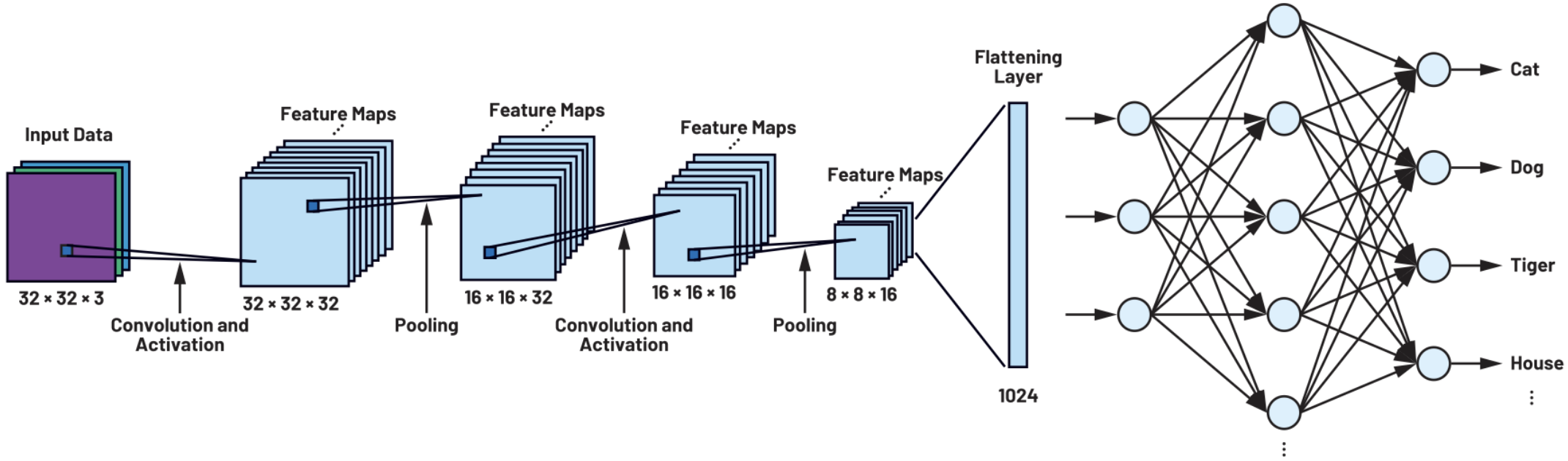
ReLU

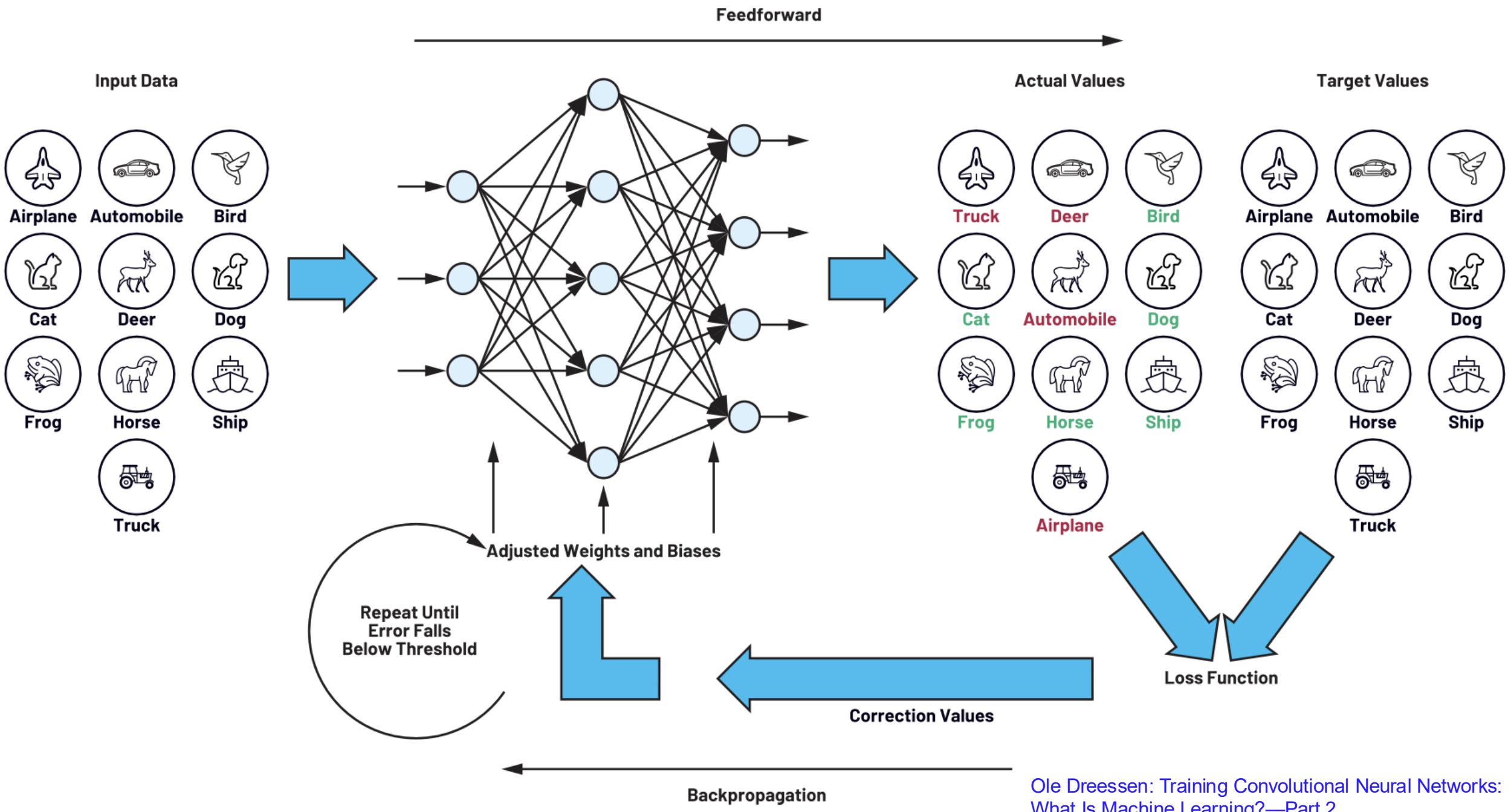


Ole Dreessen: Training Convolutional Neural Networks: What Is Machine Learning?—Part 2

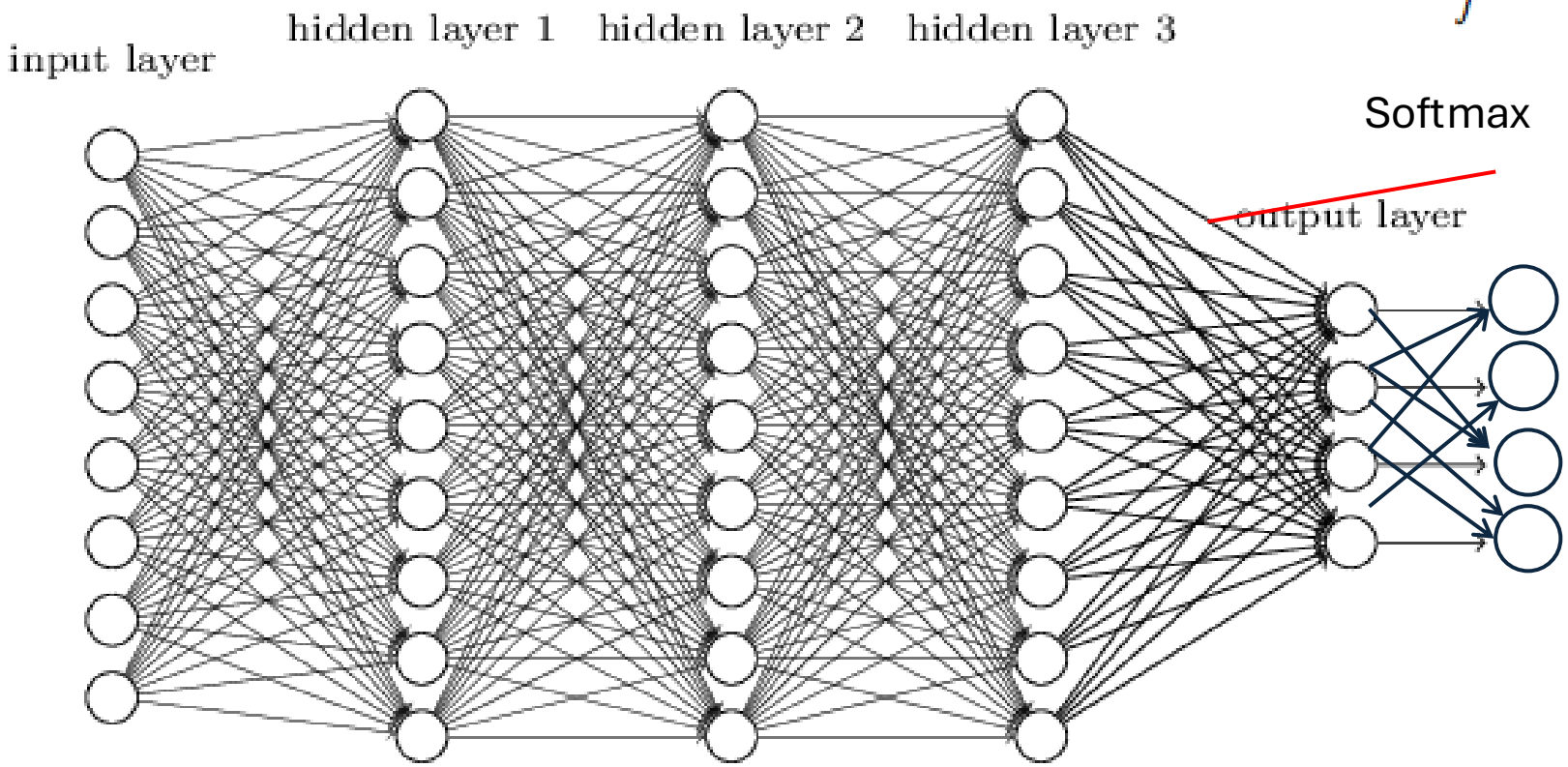
Autoencoder

- Training of convolutional networks
- Need good starting points





Ole Dreessen: Training Convolutional Neural Networks: What Is Machine Learning?—Part 2

Softmax output

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$