



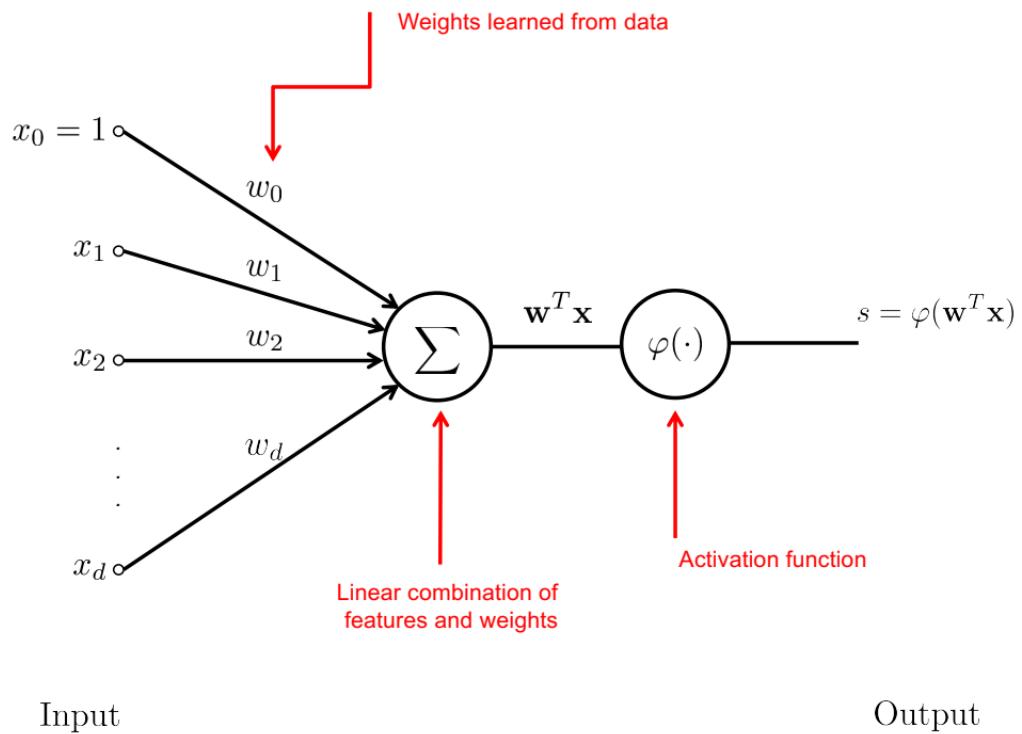
NEURAL NETWORKS

CS6140

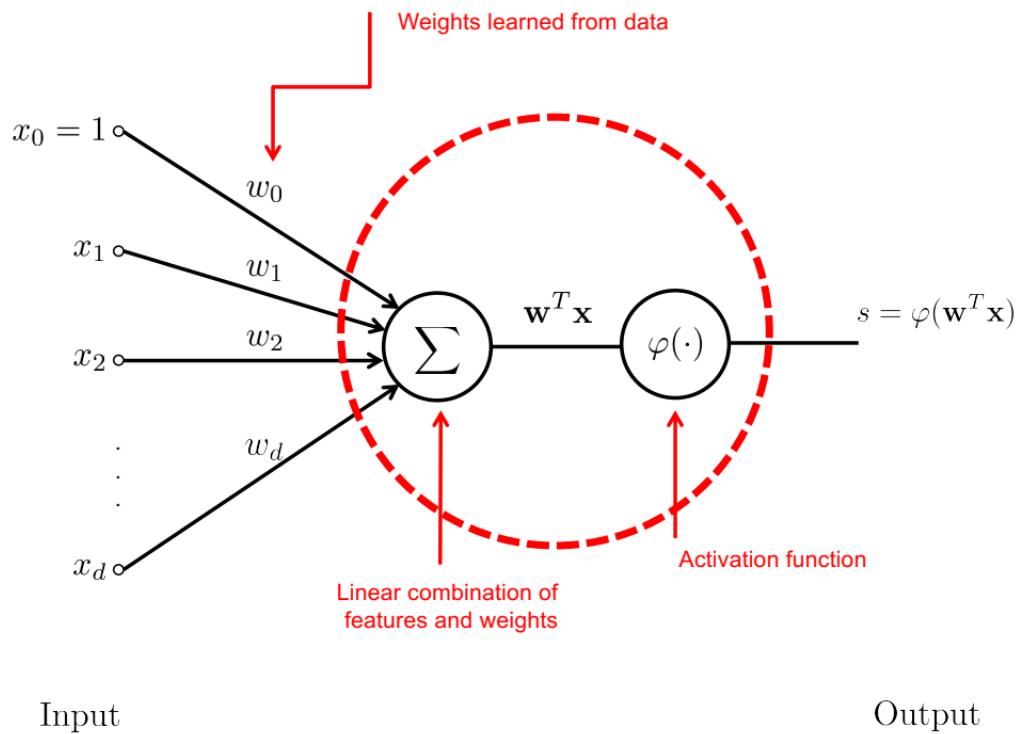
Predrag Radivojac
KHOURY COLLEGE OF COMPUTER SCIENCES
NORTHEASTERN UNIVERSITY

Spring 2021

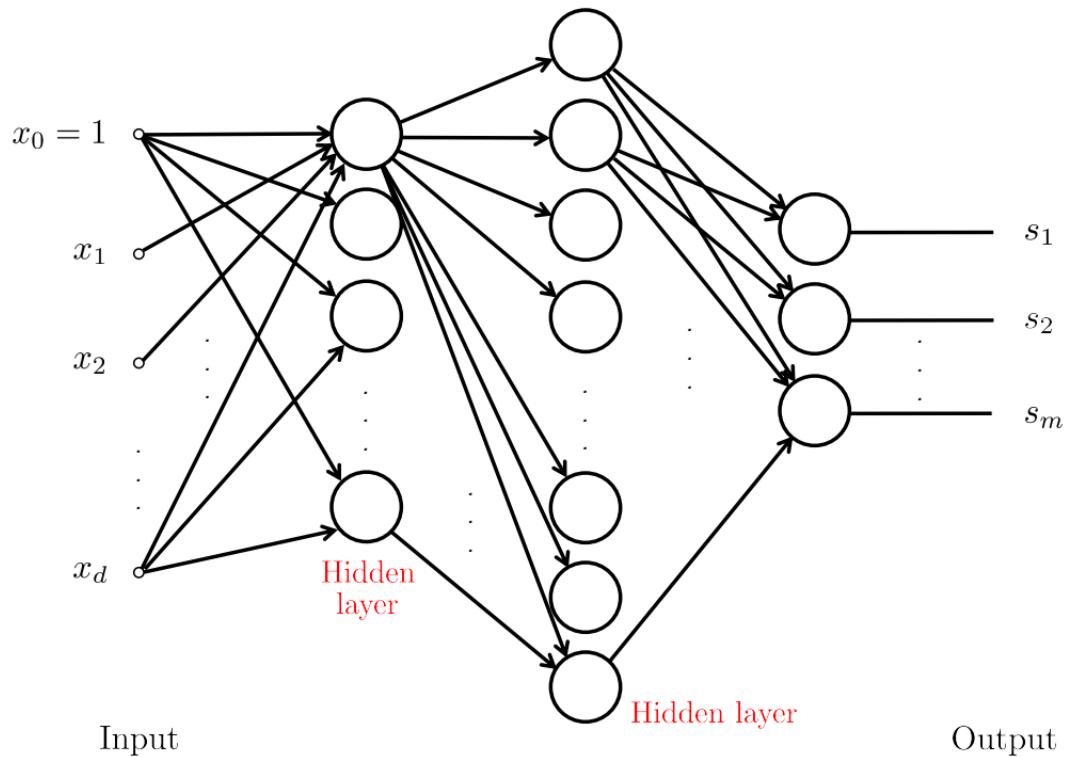
LINEAR MODEL



LINEAR MODEL



FEED-FORWARD NEURAL NETWORK

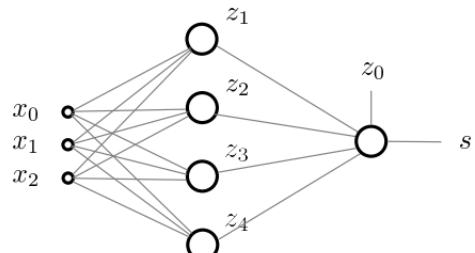
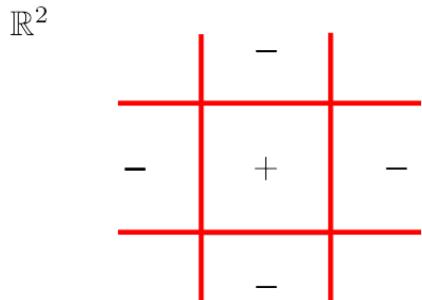


IDEA

Perceptron:

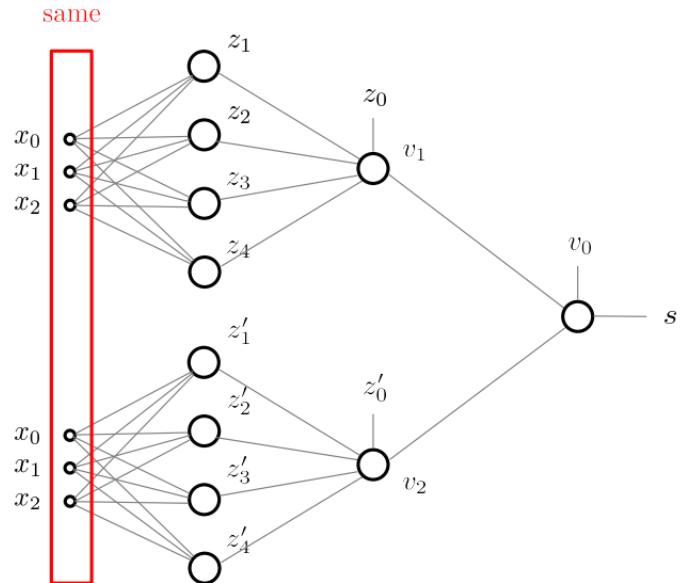
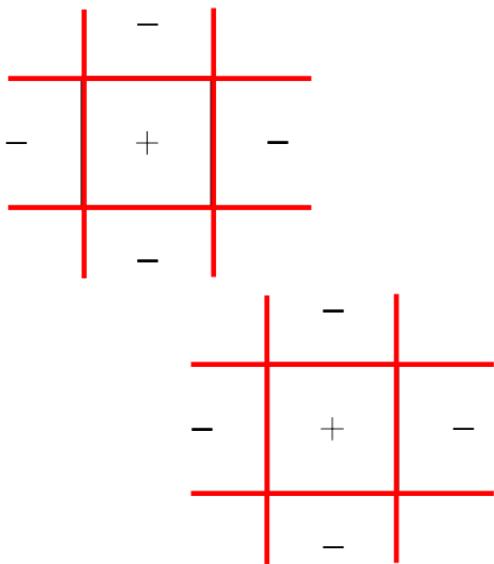
- provides a linear boundary
- easily models m -out-of- d functions

Example concept to learn:



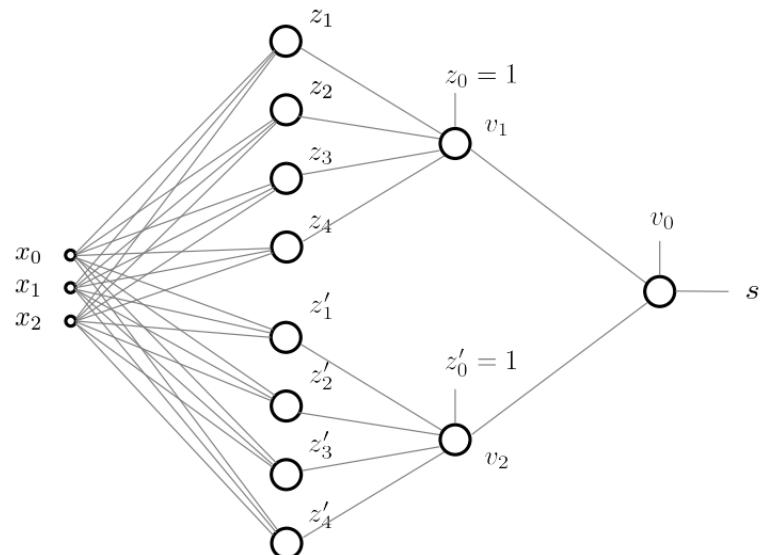
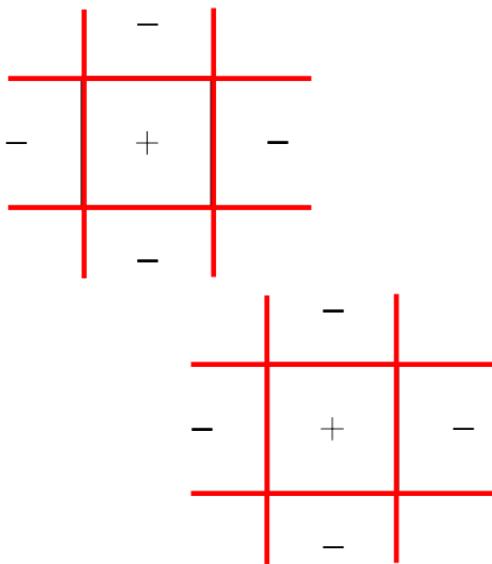
IDEA

How about this concept?

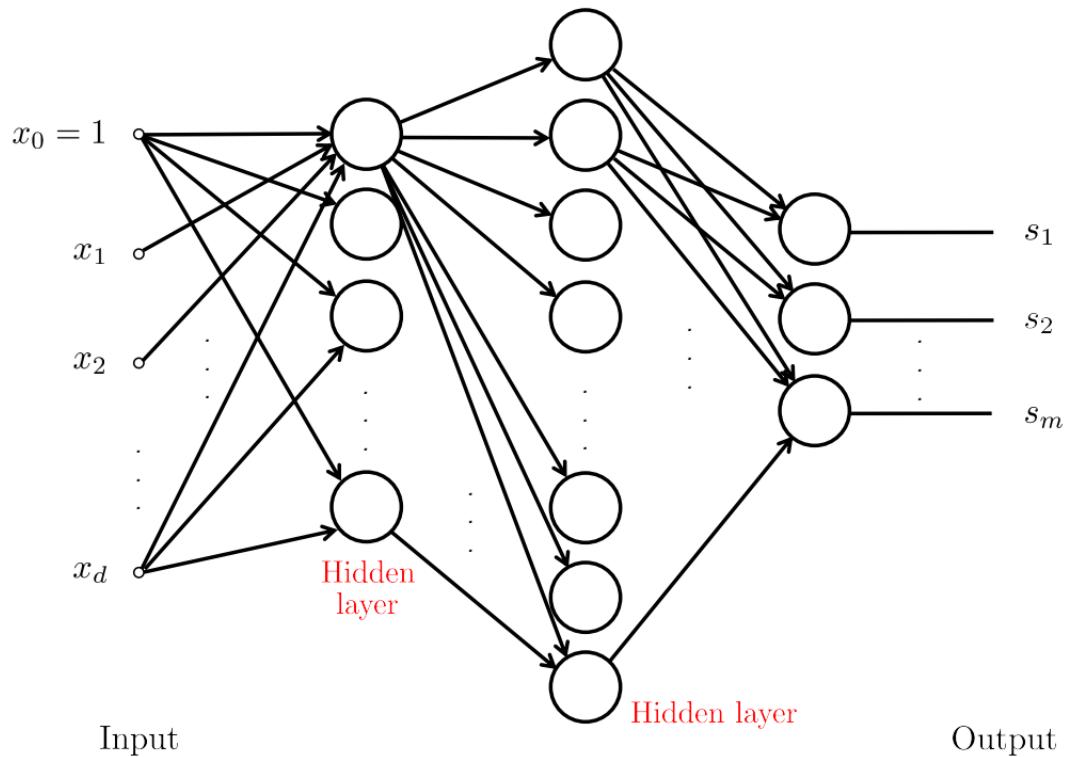


IDEA

How about this concept?



FEED-FORWARD NEURAL NETWORK



A FEED-FORWARD NEURAL NETWORK

$$\mathbf{b}_1 = (b_{10}, b_{11}, \dots, b_{1h})$$

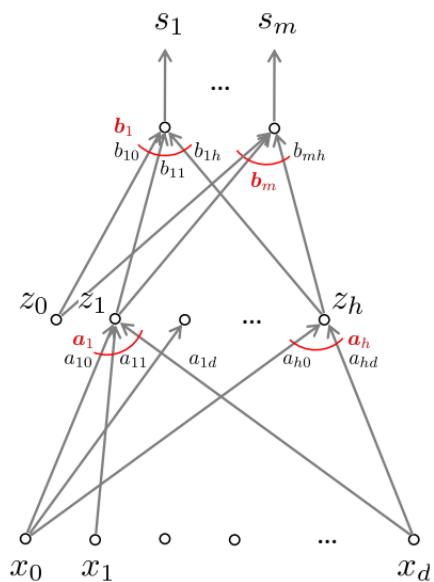
⋮

$$\mathbf{b}_m = (b_{m0}, b_{m1}, \dots, b_{mh})$$

$$\mathbf{a}_1 = (a_{10}, a_{11}, \dots, a_{1d})$$

⋮

$$\mathbf{a}_h = (a_{h0}, a_{h1}, \dots, a_{hd})$$



$$\mathbf{s} = (s_1, s_2, \dots, s_m)$$

$$s_l = \varphi(\sum_{k=0}^h b_{lk} z_k)$$

$$l = 1 \dots m$$

$$\mathbf{z} = (z_0, z_1, z_2, \dots, z_h)$$

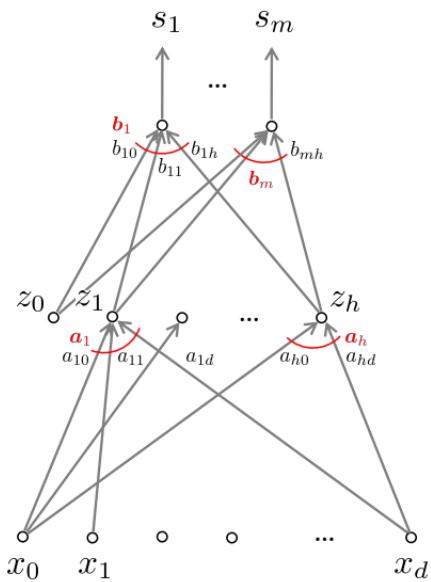
$$z_k = \varphi(\sum_{j=0}^d a_{kj} x_j)$$

$$z_0 = 1$$

$$\mathbf{x} = (x_0, x_1, x_2, \dots, x_d)$$

$$x_0 = 1$$

A FEED-FORWARD NEURAL NETWORK



$$\mathbf{s} = (s_1, s_2, \dots, s_m)$$

$$s_l = \varphi \left(\sum_{k=0}^h b_{lk} z_k \right) = \varphi \left(\mathbf{b}_l^T \mathbf{z} \right)$$

$$\mathbf{s} = \varphi(\mathbf{B}\mathbf{z}) = \varphi(\mathbf{B}_{(*,0)} + \mathbf{B}_{(*,1:h)}\varphi(\mathbf{A}\mathbf{x}))$$

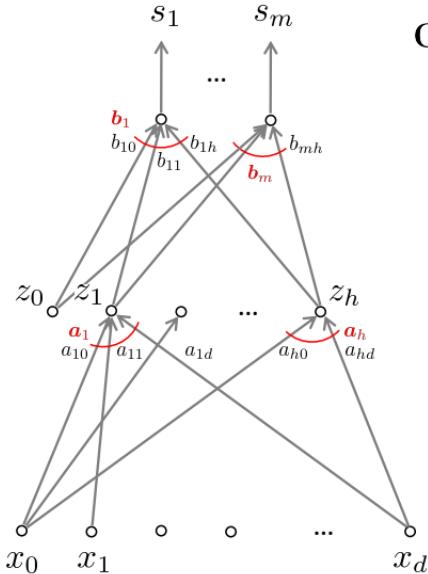
$$\mathbf{B} = \begin{bmatrix} b_{10} & b_{11} & b_{12} & & \dots & b_{1h} \\ b_{20} & b_{21} & b_{22} & & & b_{2h} \\ \vdots & & & \ddots & & \\ b_{m0} & b_{m1} & b_{m2} & & \dots & b_{mh} \end{bmatrix} = (\mathbf{b}_1^T, \mathbf{b}_2^T, \dots, \mathbf{b}_m^T)$$

$$z_k = \varphi \left(\sum_{j=0}^d a_{kj} x_j \right) = \varphi \left(\mathbf{a}_k^T \mathbf{x} \right) \quad k \neq 0$$

$$\mathbf{A} = \begin{bmatrix} a_{10} & a_{11} & a_{12} & & \dots & a_{1d} \\ a_{20} & a_{21} & a_{22} & & & a_{2d} \\ \vdots & & & \ddots & & \\ a_{h0} & a_{h1} & a_{h2} & & \dots & a_{hd} \end{bmatrix} = (\mathbf{a}_1^T, \mathbf{a}_2^T, \dots, \mathbf{a}_h^T)$$

$$\mathbf{x} = (x_0 = 1, x_1, x_2, \dots, x_d)$$

How Do WE LEARN WEIGHTS?



Given: Training data: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \{0, 1\}^m$

Objective: Minimize sum of squared errors

$$E = \sum_{i=1}^n \underbrace{\sum_{l=1}^m (y_{il} - s_{il})^2}_{e_i} = \sum_{i=1}^n e_i$$

Idea: Gradient descent. Find gradient $\nabla E(\mathbf{A}, \mathbf{B})$

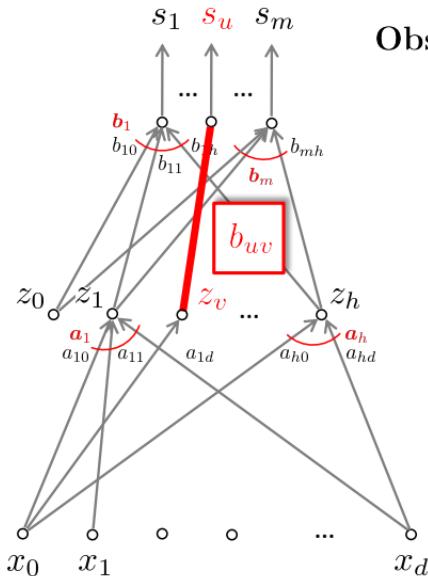
$$\frac{\partial e_i}{\partial a_{uv}} \quad \forall uv \in \{1, \dots, h\} \times \{0, 1, \dots, d\}$$

$$\frac{\partial e_i}{\partial b_{uv}} \quad \forall uv \in \{1, \dots, m\} \times \{0, 1, \dots, h\}$$

DERIVING GRADIENT DESCENT

Given: Training data: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \{0, 1\}^m$

Observation: b_{uv} only affects output s_u

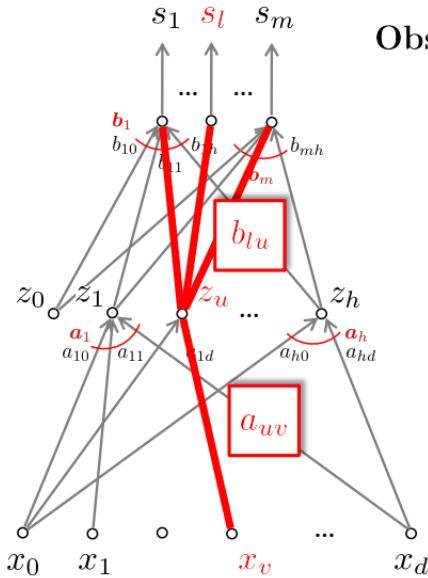


$$\begin{aligned}
 \frac{\partial e_i}{\partial b_{uv}} &= \frac{\partial}{\partial b_{uv}} ((y_{i1} - s_{i1})^2 + (y_{i2} - s_{i2})^2 + \dots + (y_{im} - s_{im})^2) \\
 &= -2(y_{iu} - s_{iu}) \frac{\partial}{\partial b_{uv}} \varphi(\mathbf{b}_u^T \mathbf{z}_i) \\
 &= -2(y_{iu} - s_{iu}) \varphi'(\mathbf{b}_u^T \mathbf{z}_i) z_{iv} \\
 &= \beta_{iu} z_{iv}
 \end{aligned}$$

DERIVING GRADIENT DESCENT

Given: Training data: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \{0, 1\}^m$

Observation: a_{uv} affects all outputs



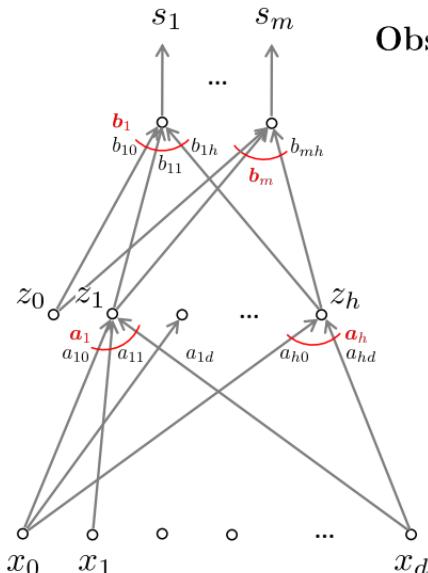
$$\begin{aligned}\frac{\partial e_i}{\partial a_{uv}} &= -2 \sum_{l=1}^m (y_{il} - s_{il}) \frac{\partial}{\partial a_{uv}} \varphi(\mathbf{b}_l^T \mathbf{z}_i) \\ &= -2 \sum_{l=1}^m (y_{il} - s_{il}) \varphi'(\mathbf{b}_l^T \mathbf{z}_i) \frac{\partial}{\partial a_{uv}} \left(\sum_{k=0}^h b_{lk} z_{ik} \right)\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial a_{uv}} \left(\sum_{k=0}^h b_{lk} z_{ik} \right) &= b_{lu} \frac{\partial}{\partial a_{uv}} z_{iu} \\ &= b_{lu} \frac{\partial}{\partial a_{uv}} \varphi(\mathbf{a}_u^T \mathbf{x}_i) \\ &= b_{lu} \varphi'(\mathbf{a}_u^T \mathbf{x}_i) x_{iv}\end{aligned}$$

DERIVING GRADIENT DESCENT

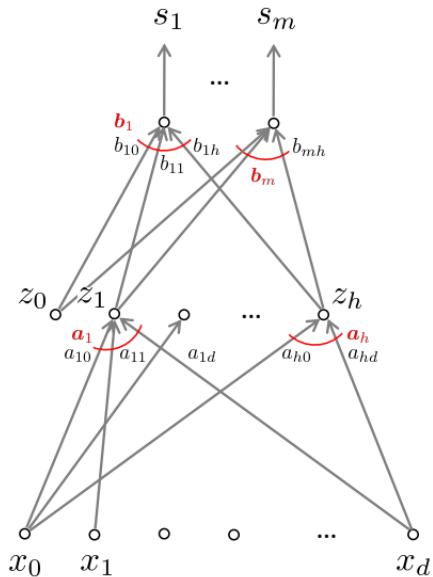
Given: Training data: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \{0, 1\}^m$

Observation: a_{uv} affects all outputs



$$\begin{aligned}\frac{\partial e_i}{\partial a_{uv}} &= -2 \sum_{l=1}^m (y_{il} - s_{il}) \varphi'(\mathbf{b}_l^T \mathbf{x}_i) b_{lu} \varphi'(\mathbf{a}_u^T \mathbf{x}_i) x_{iv} \\ &= \sum_{l=1}^m \beta_{il} b_{lu} \varphi'(\mathbf{a}_u^T \mathbf{x}_i) x_{iv} \\ &= \alpha_{iu} x_{iv}\end{aligned}$$

UPDATE RULES



Basic formulation:

$$b_{uv} \leftarrow b_{uv} - \eta \sum_{i=1}^n \beta_{iu} z_{iv} \quad \forall u \in \{1, 2, \dots, m\}, \forall v \in \{0, 1, \dots, h\}$$

$$a_{uv} \leftarrow a_{uv} - \eta \sum_{i=1}^n \alpha_{iu} x_{iv} \quad \forall u \in \{1, 2, \dots, h\}, \forall v \in \{0, 1, \dots, d\}$$

Matrix formulation:

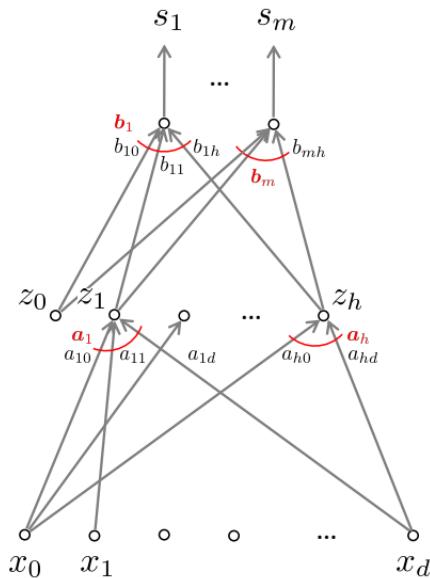
$$\mathbf{B} \leftarrow \mathbf{B} - \eta \boldsymbol{\beta}^T \mathbf{Z}$$

$$\mathbf{A} \leftarrow \mathbf{A} - \eta \boldsymbol{\alpha}^T \mathbf{X}$$

One update cycle = one epoch

$$\begin{array}{ccc} & \begin{matrix} m \\ \boldsymbol{\beta} \end{matrix} & \begin{matrix} h+1 \\ \mathbf{Z} \end{matrix} \\ \begin{matrix} n \\ \boldsymbol{\alpha} \end{matrix} & & \begin{matrix} d+1 \\ \mathbf{X} \end{matrix} \end{array}$$

UPDATE RULES



Forward pass:

- compute \mathbf{Z} and \mathbf{S} using \mathbf{X} and current weights

Backward pass:

- compute $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using \mathbf{Y}, \mathbf{S} and \mathbf{Z}

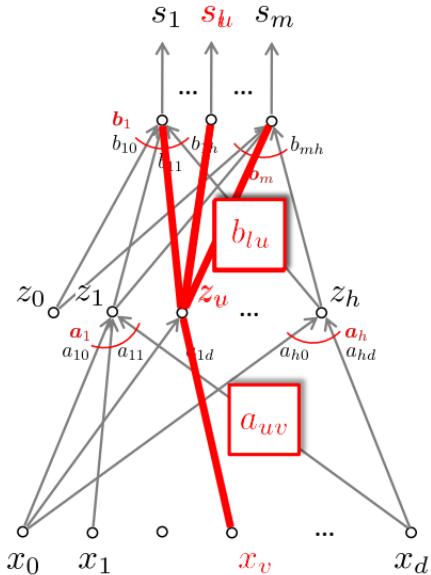
Computational complexity per epoch:

- compute \mathbf{Z} : $O(nhd)$
 - compute \mathbf{S} : $O(nmh)$
 - compute $\boldsymbol{\beta}$: $O(nmh)$
 - compute $\boldsymbol{\alpha}$: $O(nmh)$
 - update \mathbf{B} : $O(nmh)$
 - update \mathbf{A} : $O(nhd)$
- $\Rightarrow O(n \cdot \text{total weights})$

BACKPROPAGATION

What is backpropagated?

- error at the last layer
- distributed error at internal layers



$$\frac{\partial e_i}{\partial b_{uv}} = -2(y_{iu} - s_{iu})\varphi'(\mathbf{b}_u^T \mathbf{z}_i)z_{iv}$$

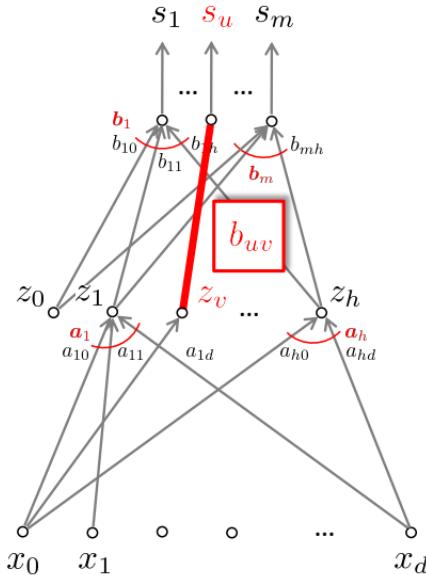
$$\frac{\partial e_i}{\partial a_{uv}} = -2 \sum_{l=1}^m (y_{il} - s_{il})\varphi'(\mathbf{b}_l^T \mathbf{z}_i)\mathbf{b}_{lu}\varphi'(\mathbf{a}_u^T \mathbf{x}_i)x_{iv}$$

ACTIVATION FUNCTIONS

Sigmoid function:

$$\circ \varphi(x) = \frac{1}{1+e^{-x}}$$

$$\circ \varphi'(x) = \varphi(x)(1 - \varphi(x))$$

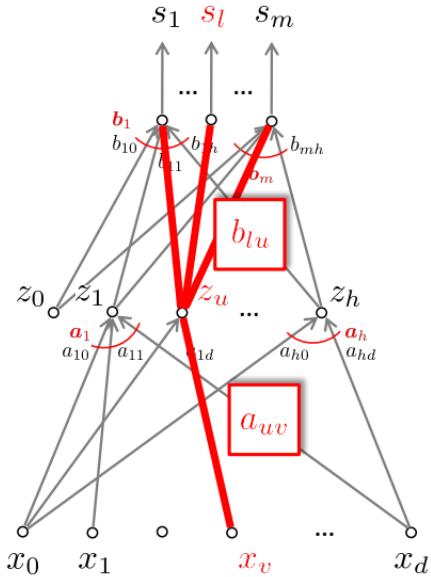


$$\frac{\partial e_i}{\partial b_{uv}} = -2(y_{iu} - s_{iu})\varphi'(\mathbf{b}_u^T \mathbf{z}_i)z_{iv} = -2(y_{iu} - s_{iu})\mathbf{s}_{iu}(1 - \mathbf{s}_{iu})z_{iv}$$

ACTIVATION FUNCTIONS

Sigmoid function:

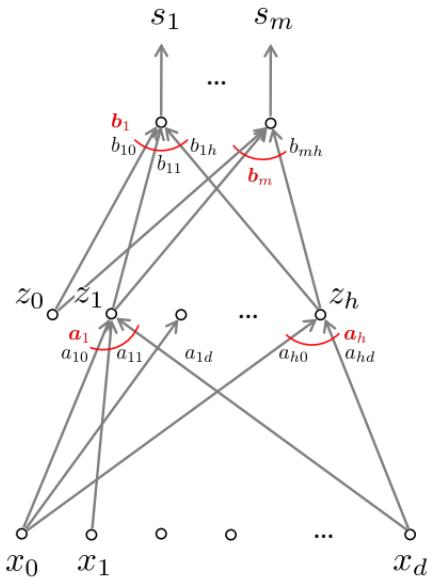
- $\varphi(x) = \frac{1}{1+e^{-x}}$
- $\varphi'(x) = \varphi(x)(1 - \varphi(x))$



$$\frac{\partial e_i}{\partial b_{uv}} = -2(y_{iu} - s_{iu})\varphi'(\mathbf{b}_u^T \mathbf{z}_i)z_{iv} = -2(y_{iu} - s_{iu})\mathbf{s}_{iu}(1 - s_{iu})z_{iv}$$

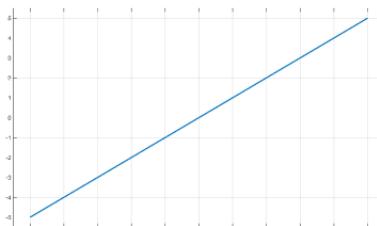
$$\begin{aligned} \frac{\partial e_i}{\partial a_{uv}} &= -2 \sum_{l=1}^m (y_{il} - s_{il})\varphi'(\mathbf{b}_l^T \mathbf{z}_i)b_{lu}\varphi'(\mathbf{a}_u^T \mathbf{x}_i)x_{iv} \\ &= -2 \sum_{l=1}^m (y_{il} - s_{il})\mathbf{s}_{il}(1 - s_{il})b_{lu}z_{iu}(1 - z_{iu})x_{iv} \end{aligned}$$

ACTIVATION FUNCTIONS



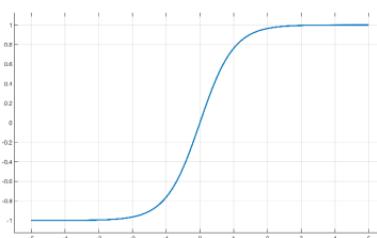
Identity (linear):

- $\varphi(x) = x$
- $\varphi'(x) = 1$



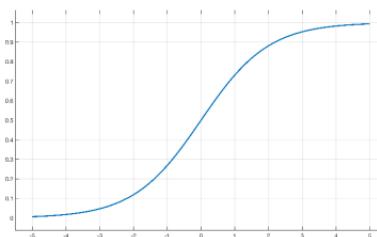
Hyperbolic tangent:

- $\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $\varphi'(x) = 1 - \varphi^2(x)$

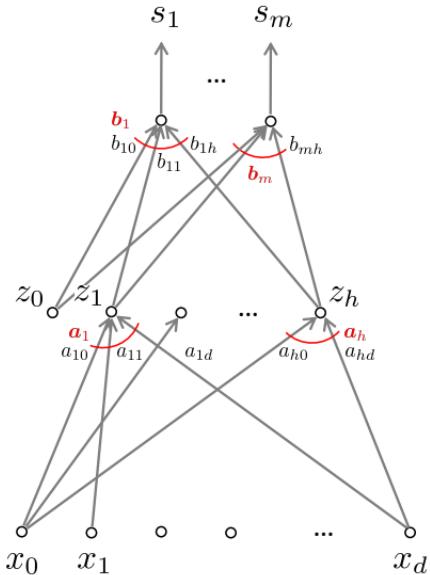


Sigmoid (logistic):

- $\varphi(x) = \frac{1}{1+e^{-x}}$
- $\varphi'(x) = \varphi(x)(1 - \varphi(x))$

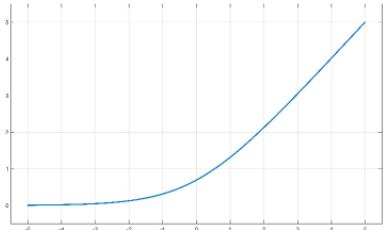


ACTIVATION FUNCTIONS



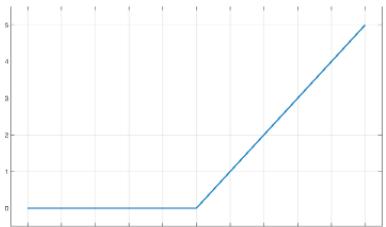
SoftPlus:

- $\varphi(x) = \ln(1 + e^x)$
- $\varphi'(x) = \frac{1}{1+e^{-x}}$



Rectified linear unit (ReLU):

- $\varphi(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$
- $\varphi'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$

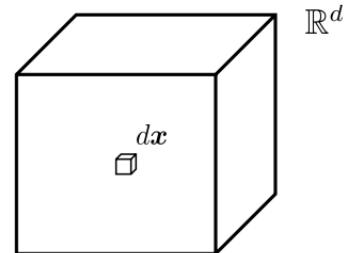


MINIMIZERS OF EUCLIDEAN DISTANCE APPROXIMATES POSTERIORS

- consider binary classification on \mathbb{R}^d
- split \mathbb{R}^d into small volumes $d\mathbf{x}$
- let's look at the expected error

$$\mathbb{E}[e^2(\mathbf{w})] = \int_{\mathcal{X}} \sum_y (y - s(\mathbf{x}, \mathbf{w}))^2 p(\mathbf{x}, y) d\mathbf{x},$$

$$y - s(\mathbf{x}, \mathbf{w}) = \begin{cases} -s(\mathbf{x}, \mathbf{w}) & y = 0 \\ 1 - s(\mathbf{x}, \mathbf{w}) & y = 1 \end{cases}$$



$$\mathbb{E}[e^2(\mathbf{w})] = \int_{\mathcal{X}} (s^2(\mathbf{x}, \mathbf{w})p(0|\mathbf{x}) + (1 - s(\mathbf{x}, \mathbf{w}))^2 p(1|\mathbf{x})) p(\mathbf{x}) d\mathbf{x}.$$

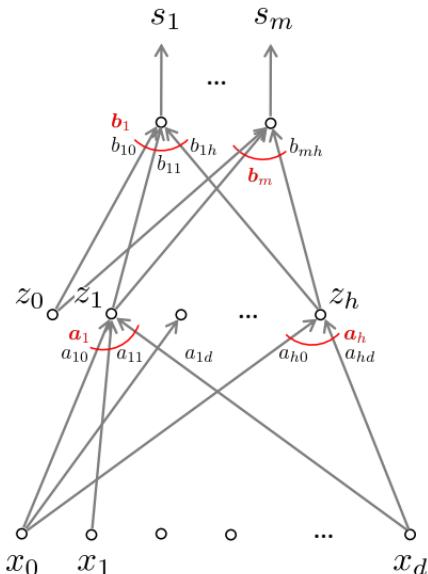
Minimize: $s^2(\mathbf{x}, \mathbf{w})p(0|\mathbf{x}) + (1 - s(\mathbf{x}, \mathbf{w}))^2 p(1|\mathbf{x})$

Solution: $s(\mathbf{x}, \mathbf{w}^*) = p(1|\mathbf{x})$

- have a flexible model
- find global minimum
- have a large representative data set

Same result for maximum likelihood estimation...

REGULARIZATION



Weight decay: penalizes large weights

$$E = \sum_i \sum_j (y_{ij} - s_{ij}(\mathbf{x}, \mathbf{w}))^2 + \gamma \|\mathbf{w}\|^2$$

γ = regularization parameter

\mathbf{w} = all weights

Smoothing: penalizes large k^{th} derivatives of outputs

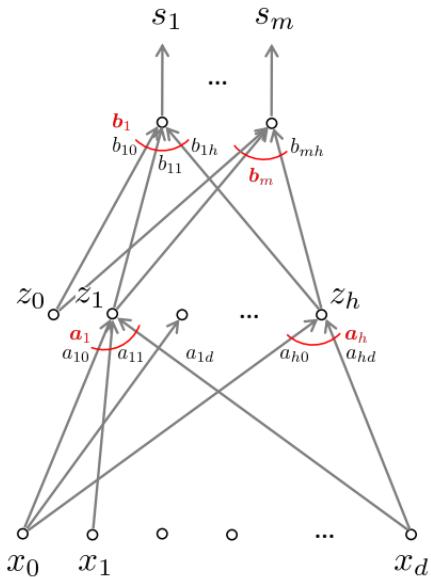
$$E = \sum_i \sum_j (y_{ij} - s_{ij}(\mathbf{x}, \mathbf{w}))^2 + \gamma \int \left\| \frac{\partial^k}{\partial \mathbf{x}^k} s(\mathbf{x}, \mathbf{w}) \right\|^2 \mu(\mathbf{x}) d\mathbf{x}$$

$\mu(\mathbf{x})$ = weighting function

INITIALIZATION

How should we initialize weights?

- o initial weights should be small and close to zero
- o shouldn't drive the activation function into saturation



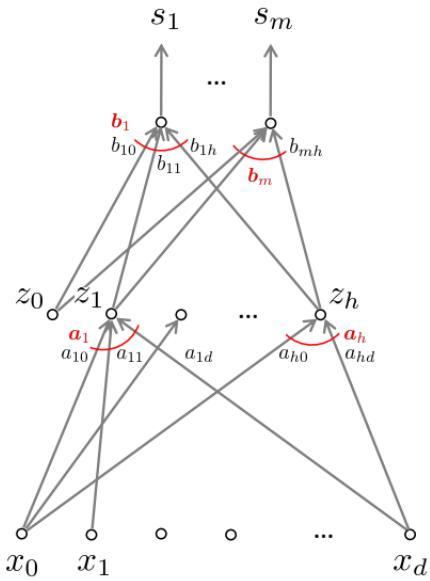
Let's look at the variance of hidden node Z_k

$$\begin{aligned}\mathbb{V}[\sum_{\text{fan-in}} w_{kj} X_j] &\approx \sum_{\text{fan-in}} \mathbb{V}[w_{kj} X_j] = && \text{(independent features)} \\ &= \sum_{\text{fan-in}} w_{kj}^2 \mathbb{V}[X_j] = \\ &= \sum_{\text{fan-in}} w_{kj}^2 \leq \text{maximum magnitude } \mathbf{1}\end{aligned}$$

$$|w_{kj}^{(0)}| \leq \frac{1}{\sqrt{|\text{fan-in}|_k}}$$

MOMENTUM

Momentum:



- escaping local optima or flat regions on the error surface
- adds inertia to the magnitude of the weight update

$$w^{(t+1)} = w^{(t)} - \frac{\partial E^{(t)}}{\partial w^{(t)}}, \text{ where } \Delta w^{(t)} = -\frac{\partial E^{(t)}}{\partial w^{(t)}}$$

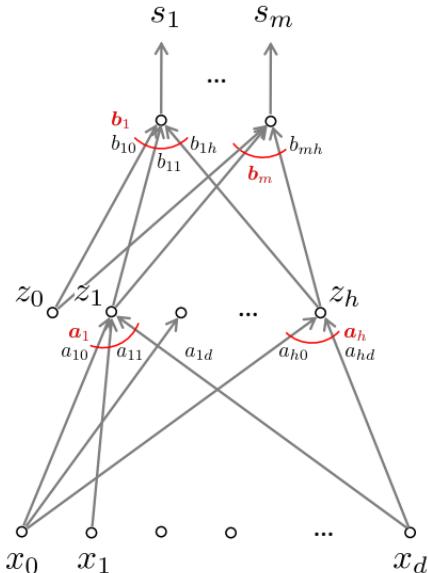
- new weight update

$$w^{(t+1)} = w^{(t)} - \frac{\partial E^{(t)}}{\partial w^{(t)}} + \mu \Delta w^{(t-1)} \quad 0 \leq \mu < 1$$

μ = momentum constant

$$\Delta w^{(t)} = -\eta \sum_{i=0}^t \mu^{t-i} \frac{\partial E^{(i)}}{\partial w^{(i)}}$$

ADDITIONAL HEURISTICS



Early stopping:

- use validation set to prevent overfitting
- validation set size picked wisely (e.g., $n < |\mathbf{w}|$ requires additional attention)

Learning rate:

- each weight has its learning rate
- learning rate varies over time (e.g., increase if gradient sign is unchanged over consecutive iterations)

ADAPTIVE TECHNIQUES

Standard update:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial E^{(t)}}{\partial w^{(t)}}$$

Two questions:

- Can we avoid using gradient? Yes, but we need the sign.
- Can we adaptively change learning rates? Yes.

Idea: observe gradient

- if the sign stays the same in t and $t + 1$: increase step size
- if sign changes: return to previous position and decrease step size
- if gradient is 0: stop

Algorithms:

- resilient propagation (RPROP)

Algorithm 1 Resilient propagation algorithm (RPROP). Recommended parameter values are $\Delta_0 = 0.1$, $\Delta_{\min} = 10^{-6}$, $\Delta_{\max} = 50$, $\eta_- = 0.5$, $\eta_+ = 1.2$. Note that $\text{sign}(0) = 0$.

Initialization: for $\forall w_{ij}$

Set $w_{ij}^{(0)}$ randomly

$$\Delta_{ij}^{(0)} = \Delta_0$$

$$\Delta w_{ij}^{(0)} = -\text{sign}\left(\frac{\partial E}{\partial w_{ij}}\right)^{(0)} \cdot \Delta_0$$

Weight update step: for $\forall w_{ij}$

$$t = 1$$

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)}$$

repeat until convergence

$$\text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} > 0$$

$$\Delta_{ij}^{(t)} = \min \left\{ \Delta_{\max}, \Delta_{ij}^{(t-1)} \cdot \eta_+ \right\}$$

$$\Delta w_{ij}^{(t)} = -\text{sign}\left(\frac{\partial E}{\partial w_{ij}}\right)^{(t)} \cdot \Delta_{ij}^{(t)}$$

$$\text{elseif } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$$

$$\Delta_{ij}^{(t)} = \max \left\{ \Delta_{\min}, \Delta_{ij}^{(t-1)} \cdot \eta_- \right\}$$

$$\Delta w_{ij}^{(t)} = -\Delta w_{ij}^{(t-1)}$$

$$\frac{\partial E}{\partial w_{ij}}^{(t)} = 0$$

$$\text{else}$$

$$\Delta_{ij}^{(t)} = \Delta_{ij}^{(t-1)}$$

$$\Delta w_{ij}^{(t)} = -\text{sign}\left(\frac{\partial E}{\partial w_{ij}}\right)^{(t)} \cdot \Delta_{ij}^{(t)}$$

$$\text{end}$$

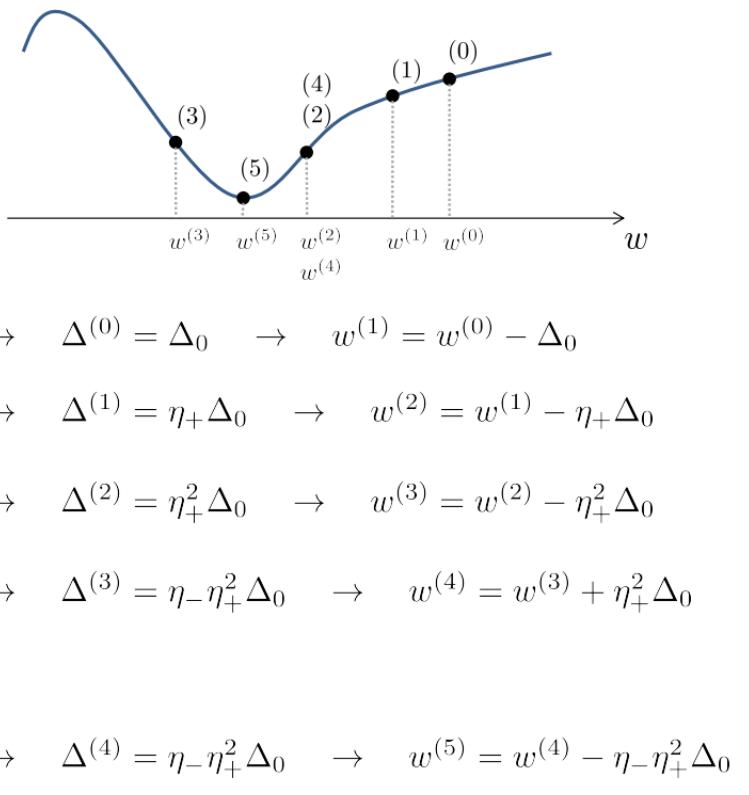
$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$$

$$t = t + 1$$

$$\text{end}$$

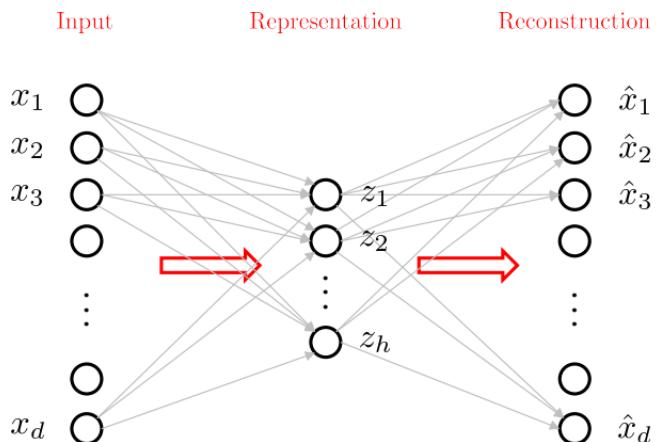
Termination: for $\forall w_{ij}$

Output $w_{ij}^{(t)}$



$$5) \quad \frac{\partial E}{\partial w^{(5)}} = 0 \quad \rightarrow \quad \text{stop}$$

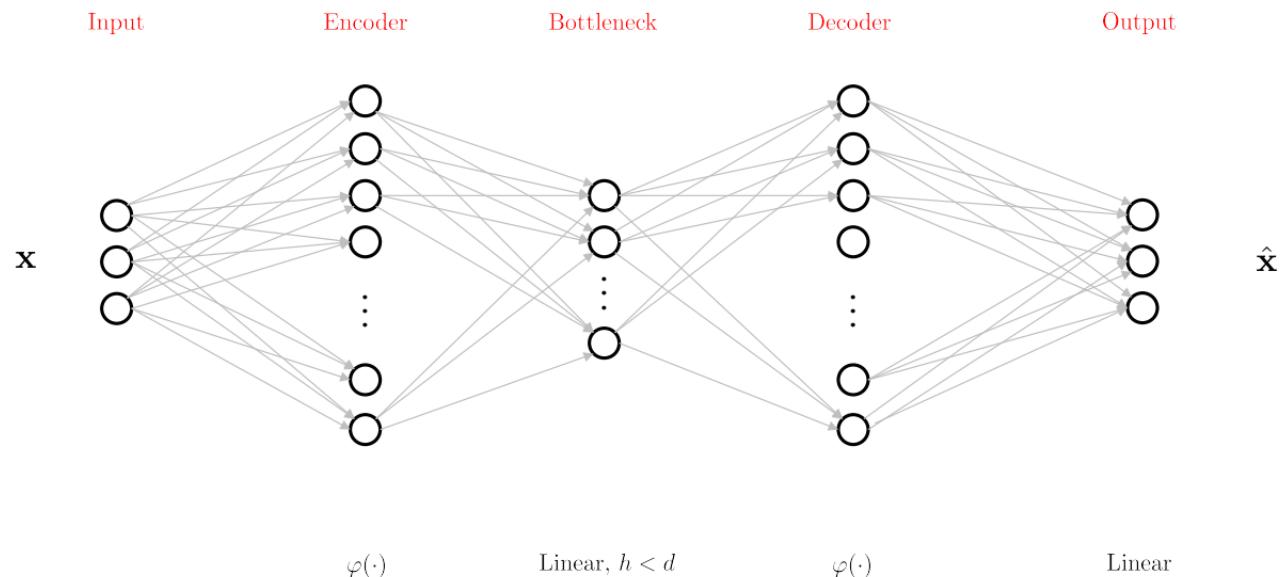
COMPRESSION



Input	Hidden			Output
0 0 0 0 0 0 1	.10	.00	.91	0 0 0 0 0 0 1
0 0 0 0 0 1 0	.10	.95	.07	0 0 0 0 0 1 0
0 0 0 0 1 0 0	.10	.89	.91	0 0 0 0 1 0 0
0 0 0 0 1 0 0 0	.94	.03	.13	0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0	.94	.03	.93	0 0 0 1 0 0 0 0
	⋮		⋮	⋮

Learned representation \mathbf{z}

AUTOASSOCIATOR (AUTOENCODER)



ON NEURAL NETWORKS

Networks:

- inspired in part by how brain works
- massive parallelism, graceful degradation
- good generalization, noise-tolerant, can incorporate prior knowledge
- not transparent

Network structure:

- feed-forward
- recurrent
- deterministic vs. stochastic activation

Expressiveness:

- 2-layer networks can learn any continuous function
- 3-layer networks can learn any function (universal approximators)
- $O(\frac{2^d}{d})$ neurons needed to learn all binary functions with d inputs