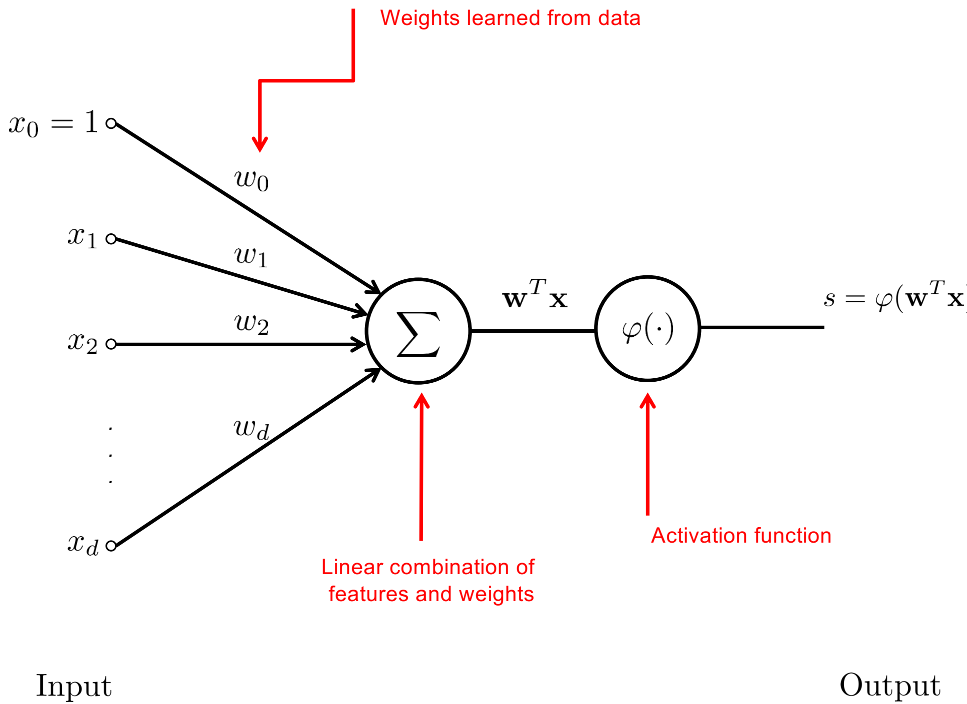
NEURAL NETWORKS

CS6140

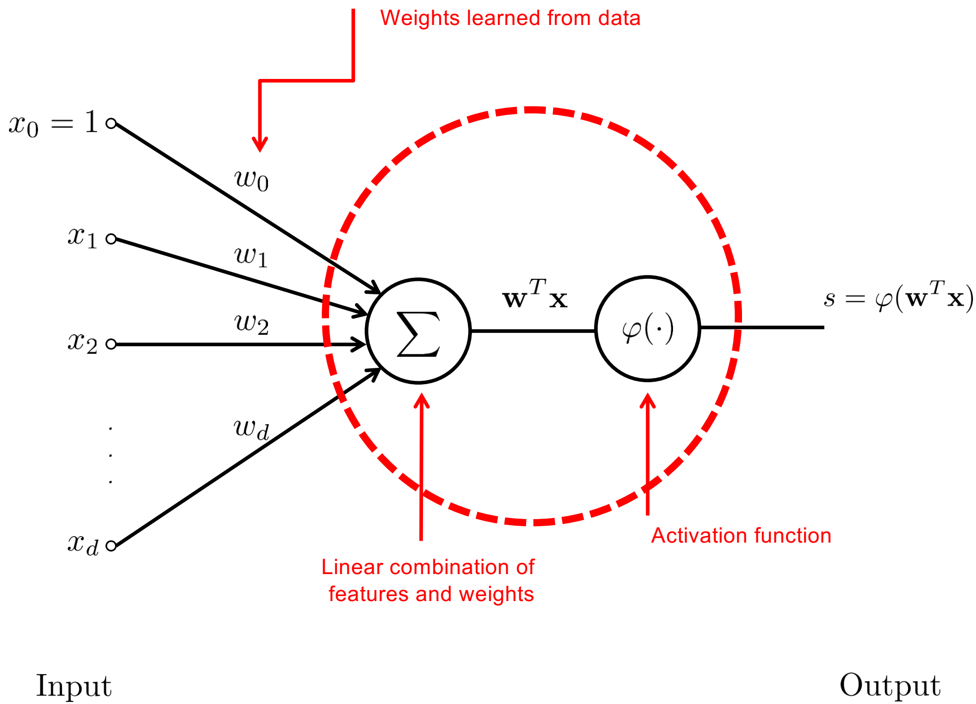
Predrag Radivojac
KHOURY COLLEGE OF COMPUTER SCIENCES
NORTHEASTERN UNIVERSITY

Fall 2024

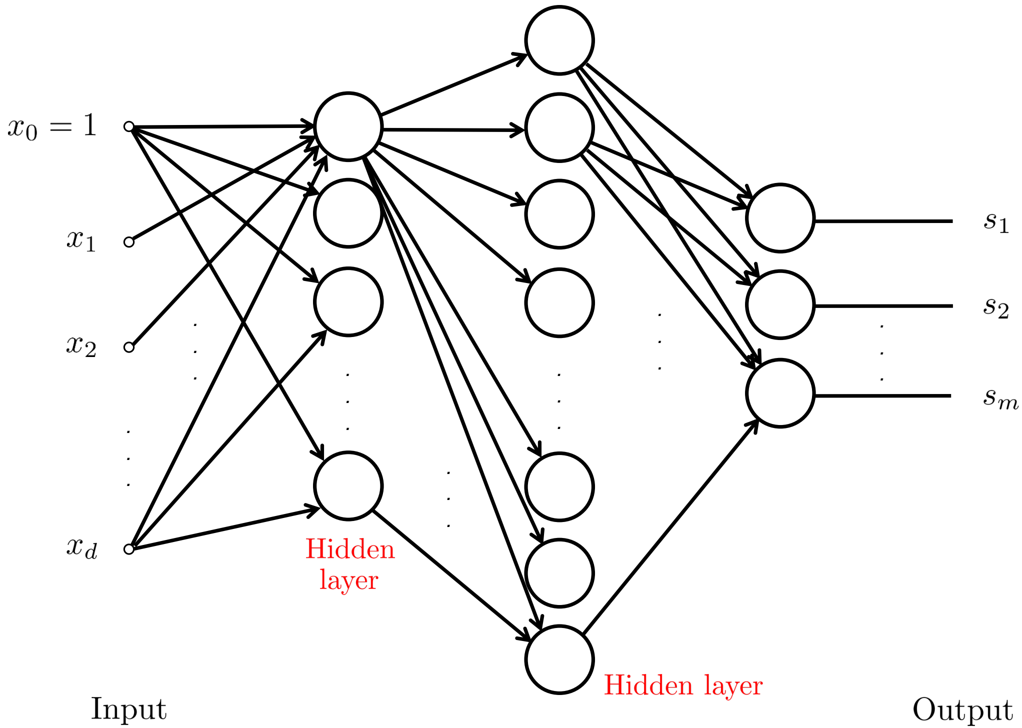
Linear Model



Linear Model



Feed-Forward Neural Network

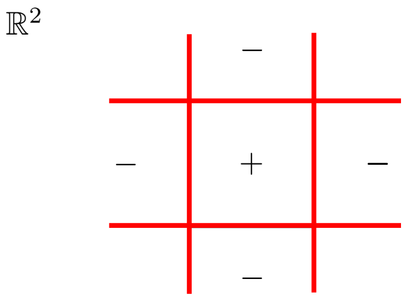


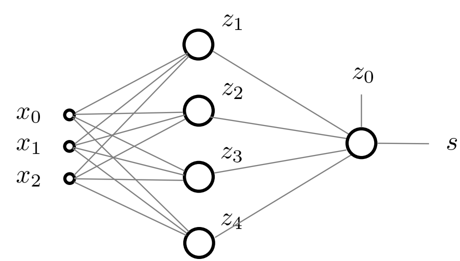
IDEA

Perceptron:

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

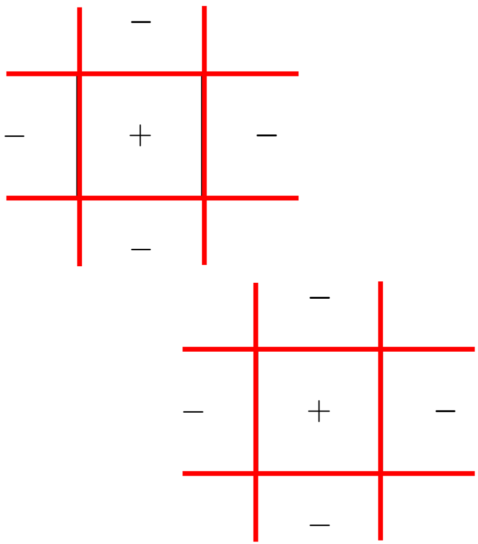
Example concept to learn:

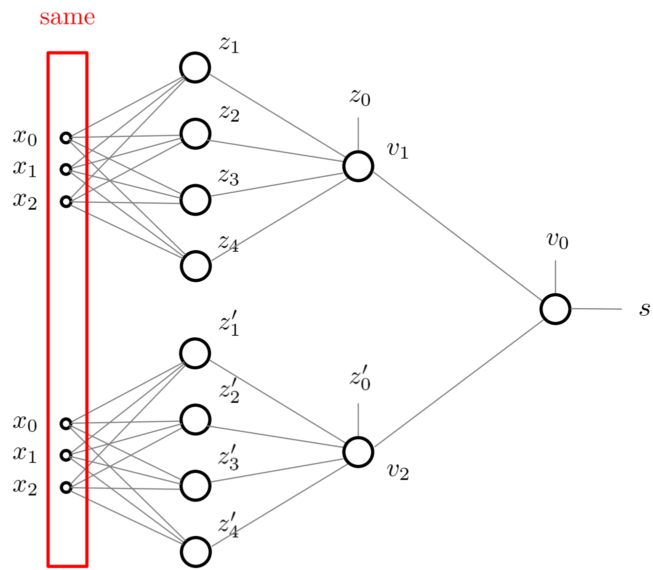




Idea

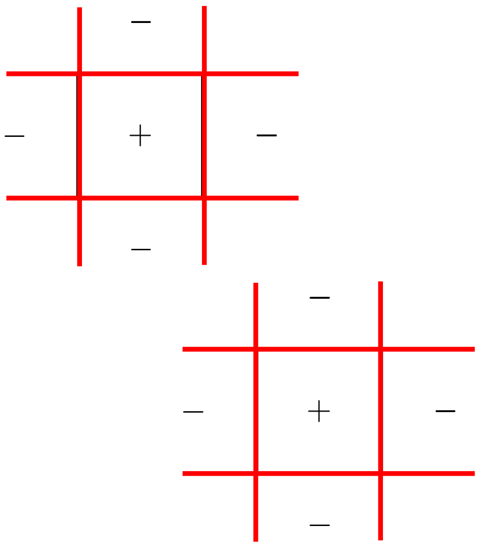
How about this concept?

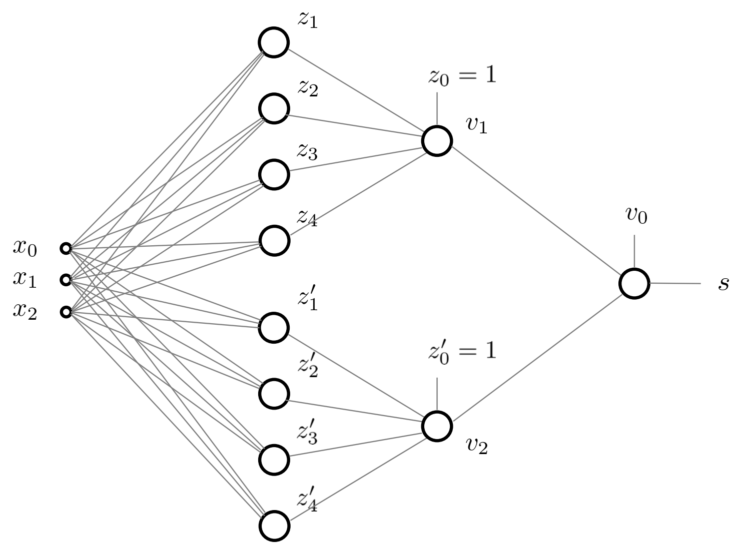




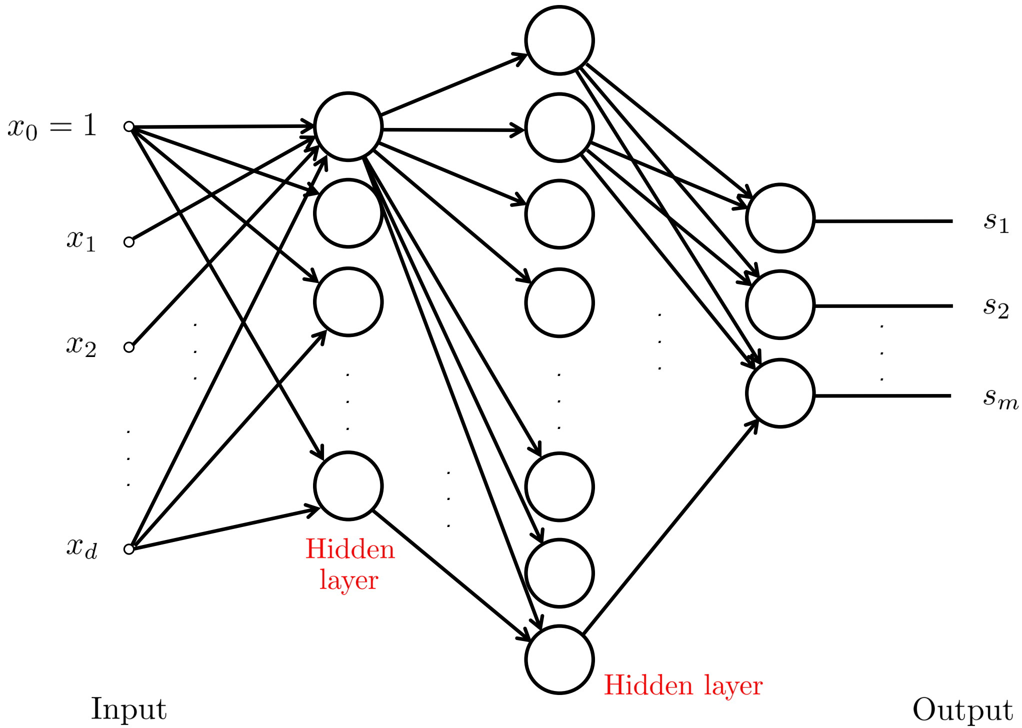
Idea

How about this concept?





Feed-Forward Neural Network



A Feed-Forward Neural Network

$b_1 = (b_{10}, b_{11}, ..., b_{1h})$

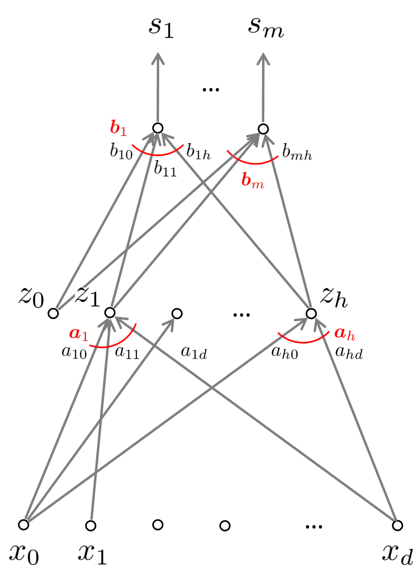
$\vdots$

$b_m = (b_{m0}, b_{m1}, ..., b_{mh})$

$a_1 = (a_{10}, a_{11}, ..., a_{1d})$

$\vdots$

$a_h = (a_{h0}, a_{h1}, ..., a_{hd})$



$\boldsymbol{s} = (s_1, s_2, ..., s_m)$

$s_l = \varphi(\sum_{k=0}^{h} b_{lk} z_k)$ $\quad l = 1 \dots m$

$\boldsymbol{z} = (z_0, z_1, z_2, ..., z_h)$

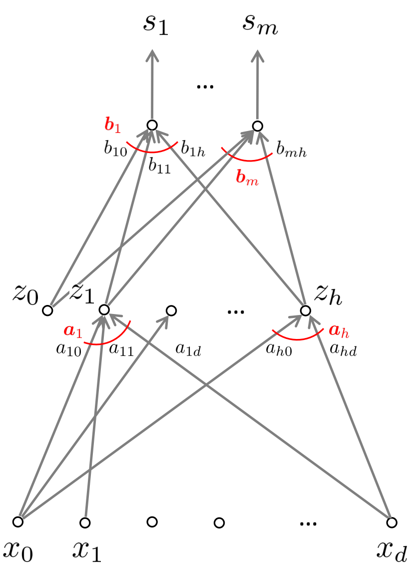
$z_k = \varphi(\sum_{j=0}^{d} a_{kj} x_j)$ $\quad k = 1 \dots h$

$z_0 = 1$

$\boldsymbol{x} = (x_0, x_1, x_2, ..., x_d)$

$x_0 = 1$

A FEED-FORWARD NEURAL NETWORK



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

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

$$\mathbf{s} = \boldsymbol{\varphi}\left(\mathbf{Bz}\right) = \boldsymbol{\varphi}(\mathbf{B}_{(*,0)} + \mathbf{B}_{(*,1:h)}\boldsymbol{\varphi}(\mathbf{Ax}))$$

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

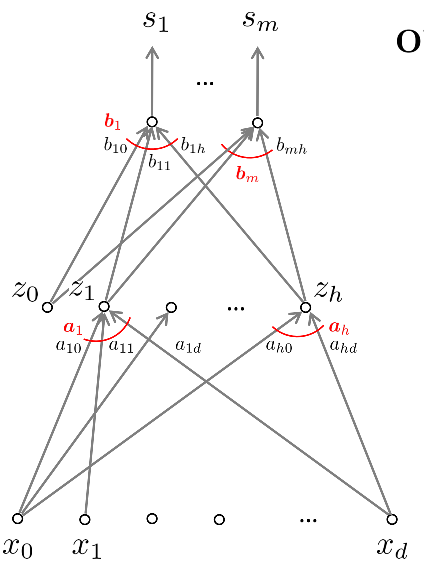
$$z_k = \varphi\left(\sum_{j=0}^{d} a_{kj} x_j\right) = \varphi\left(\mathbf{a}_k^T \mathbf{x}\right) \quad k \neq 0$$

$$\mathbf{z} = (1, \boldsymbol{\varphi}(\mathbf{Ax}))$$

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

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

How Do We Learn Weights?



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

Objective: Minimize sum of squared errors

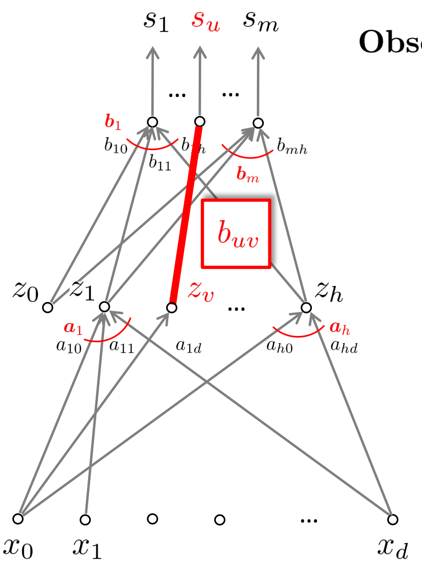
$$E = \sum_{i=1}^{n} \underbrace{\sum_{l=1}^{m} (y_{il} - s_{il})^2}_{e_i} = \sum_{i=1}^{n} e_i$$

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

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

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

Deriving Gradient Descent

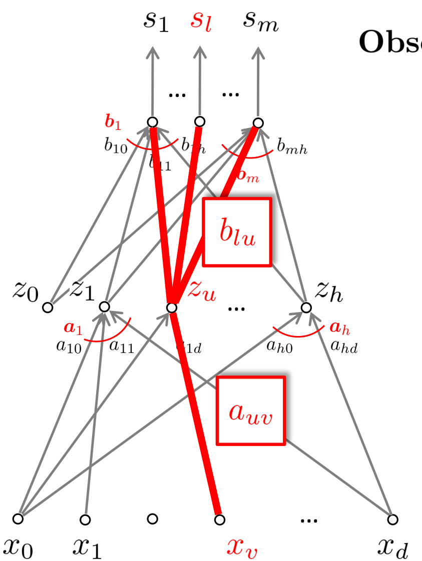


Given: Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, $\boldsymbol{x} \in \mathbb{R}^d$ and $\boldsymbol{y} \in \{0,1\}^m$

Observation: b_{uv} only affects output s_u

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

DERIVING GRADIENT DESCENT



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

Observation: a_{uv} affects all outputs

$$\frac{\partial e_i}{\partial a_{uv}} = -2 \sum_{l=1}^{m} (y_{il} - s_{il}) \frac{\partial}{\partial a_{uv}} \varphi(\mathbf{b}_l^T \mathbf{z}_i)$$

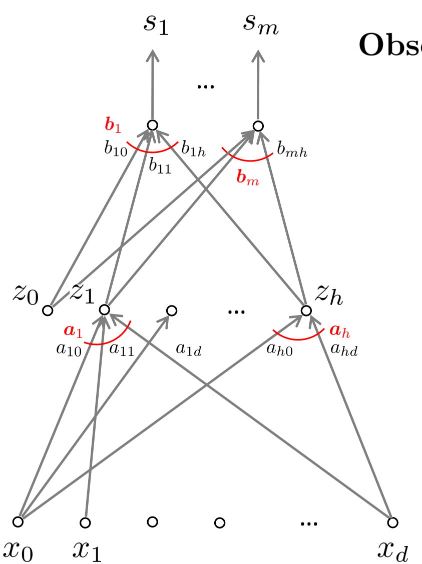
$$= -2 \sum_{l=1}^{m} (y_{il} - s_{il}) \varphi'(\mathbf{b}_l^T \mathbf{z}_i) \frac{\partial}{\partial a_{uv}} \left(\sum_{k=0}^{h} b_{lk} z_{ik} \right)$$

$$\frac{\partial}{\partial a_{uv}} \left(\sum_{k=0}^{h} b_{lk} z_{ik} \right) = b_{lu} \frac{\partial}{\partial a_{uv}} z_{iu}$$

$$= b_{lu} \frac{\partial}{\partial a_{uv}} \varphi(\mathbf{a}_u^T \mathbf{x}_i)$$

$$= b_{lu} \varphi'(\mathbf{a}_u^T \mathbf{x}_i) x_{iv}$$

Deriving Gradient Descent



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

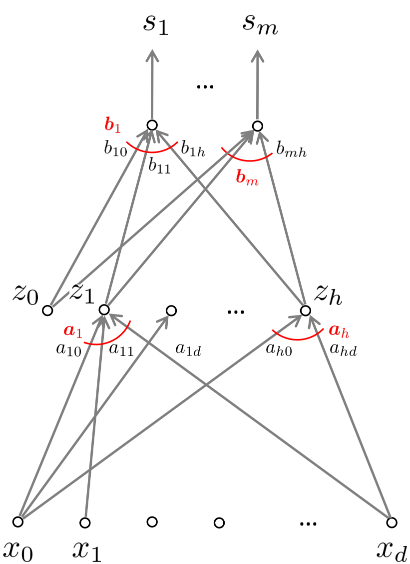
Observation: a_{uv} affects all outputs

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

$$= \sum_{l=1}^{m} \beta_{il} b_{lu} \varphi'(\mathbf{a}_u^T \mathbf{x}_i) x_{iv}$$

$$= \alpha_{iu} x_{iv}$$

Update Rules



Basic formulation:

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

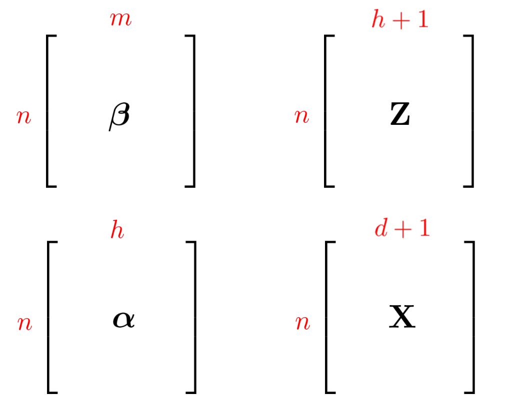
$$a_{uv} \leftarrow a_{uv} - \eta \sum_{i=1}^{n} \alpha_{iu} x_{iv} \qquad \forall u \in \{1, 2, \ldots, h\}, \forall v \in \{0, 1, \ldots, d\}$$

Matrix formulation:

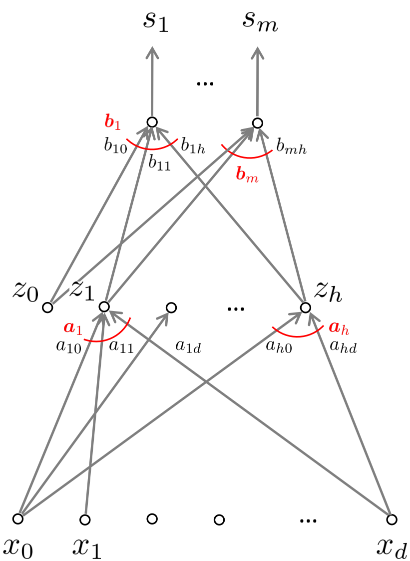
$$\mathbf{B} \leftarrow \mathbf{B} - \eta \boldsymbol{\beta}^T \mathbf{Z}$$

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

One update cycle = one epoch



Update Rules



Forward pass:

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

Backward pass:

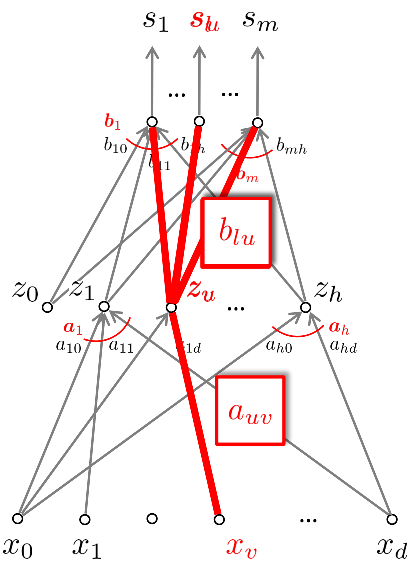
- ∘ compute $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using $\mathbf{Y}$, $\mathbf{S}$ and $\mathbf{Z}$

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



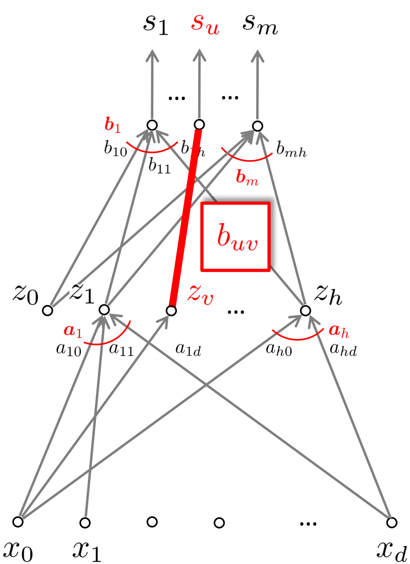
What is backpropagated?

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

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

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

Activation Functions

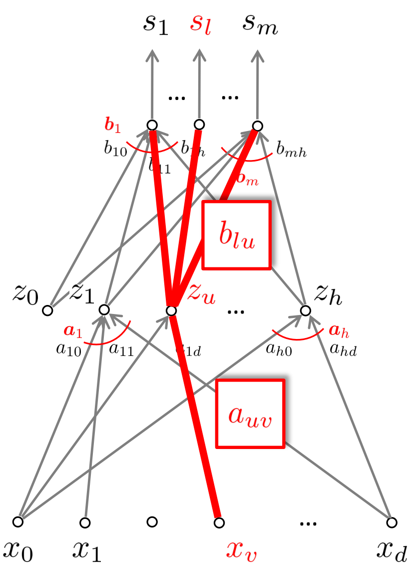


Sigmoid function:

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

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

Activation Functions



Sigmoid function:

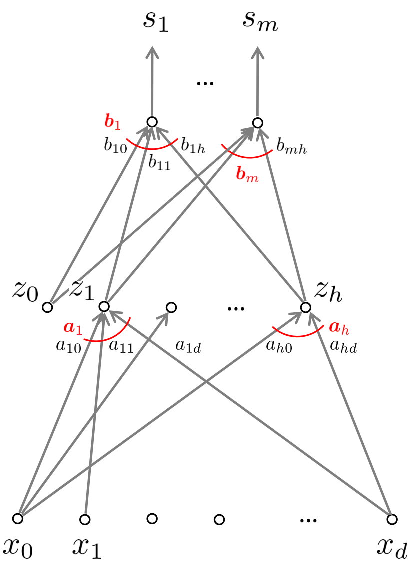
- $\varphi(x) = \frac{1}{1+e^{-x}}$
- $\varphi'(x) = \varphi(x)(1-\varphi(x))$

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

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

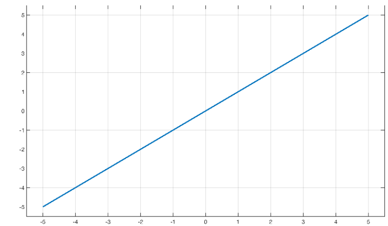
$$= -2 \sum_{l=1}^{m} (y_{il} - s_{il}) s_{il}(1 - s_{il}) b_{lu} z_{iu}(1 - z_{iu}) x_{iv}$$

Activation Functions



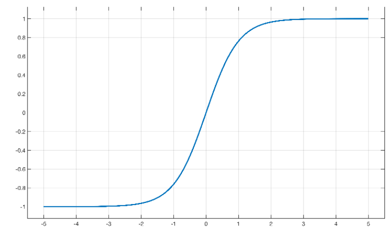
Identity (linear):

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



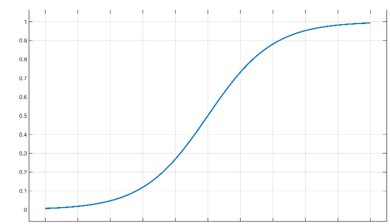
Hyperbolic tangent:

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

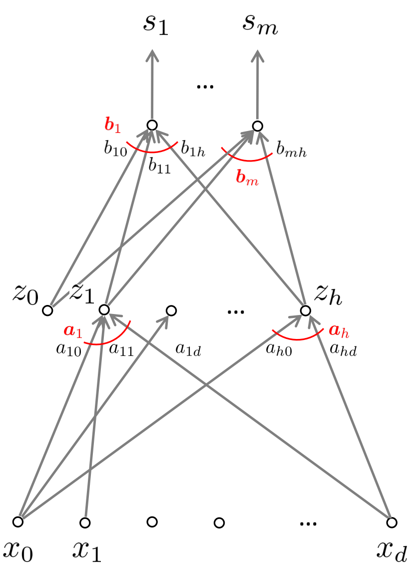


Sigmoid (logistic):

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

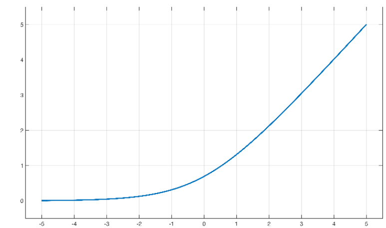


Activation Functions



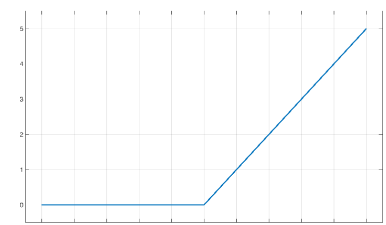
SoftPlus:

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



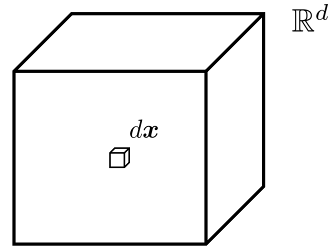
Rectified linear unit (ReLU):

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



Minimizers of Euclidean Distance Approximate Posteriors

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



$$\mathbb{E}\left[e^2(\boldsymbol{w})\right] = \int_{\mathcal{X}} \sum_{\mathcal{Y}} \left(y - s(\boldsymbol{x}, \boldsymbol{w})\right)^2 p(\boldsymbol{x}, y) d\boldsymbol{x},$$

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

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

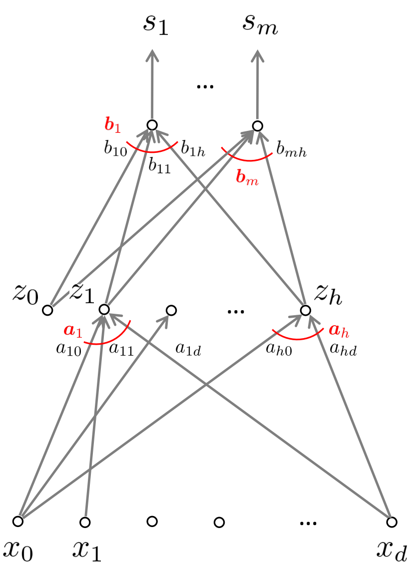
Minimize: $s^2(\boldsymbol{x}, \boldsymbol{w}) p(0|\boldsymbol{x}) + (1 - s(\boldsymbol{x}, \boldsymbol{w}))^2 p(1|\boldsymbol{x})$

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

- have a flexible model
- find global minimum
- have a large representative data set (not required)

Same result for maximum likelihood estimation...

Regularization



Weight decay: penalizes large weights

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

γ = regularization parameter

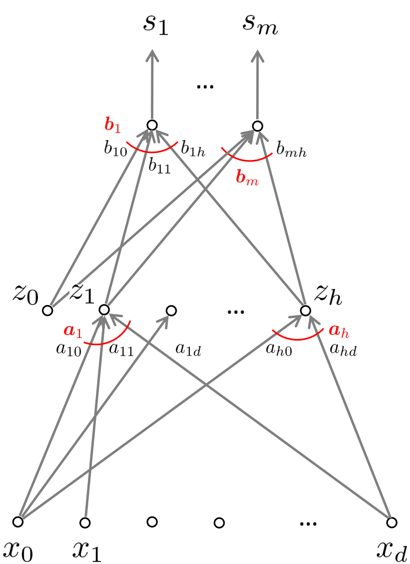
$\boldsymbol{w}$ = all weights

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

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

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

Initialization



How should we initialize weights?

- ∘ initial weights should be small and close to zero
- ∘ shouldn't drive the activation function into saturation
- ∘ initial weights should be different to break symmetries

Let's look at the variance of hidden node Z_k

$$\mathbb{V}[\sum_{\text{fan-in}} w_{kj} X_j] \approx \sum_{\text{fan-in}} \mathbb{V}[w_{kj} X_j] = \qquad \textcolor{red}{\text{(independent features)}}$$

$$= \sum_{\text{fan-in}} w_{kj}^2 \mathbb{V}[\overset{\textcolor{red}{1}}{X_j}] =$$

$$= \sum_{\text{fan-in}} w_{kj}^2 \leq \text{maximum} \overset{\textcolor{red}{1}}{\ } \text{magnitude}$$

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

⟵ use uniform distribution but consider sparsity

INITIALIZATION: EXPERIMENT BY GLOROT & BENGIO (2010)

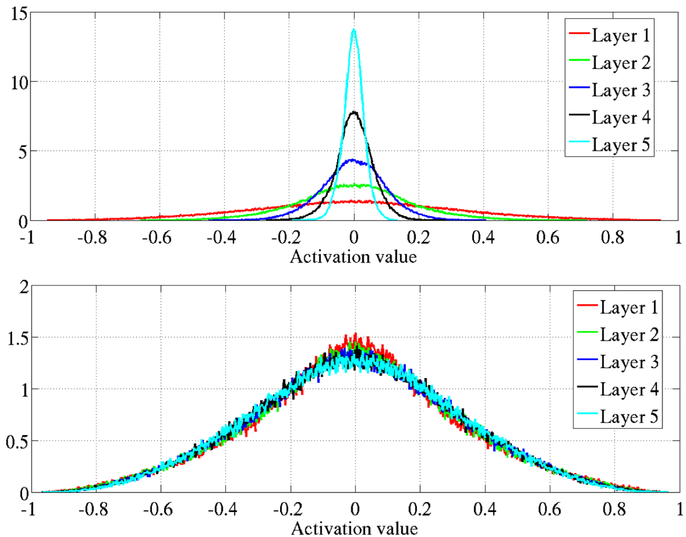


Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

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

$$|w_{kj}^{(0)}| \leq \frac{\sqrt{6}}{\sqrt{|\text{fan-in}|_k + |\text{fan-out}|_k}}$$

Glorot & Bengio, AISTATS, 2010.

INITIALIZATION: EXPERIMENT BY GLOROT & BENGIO (2010)

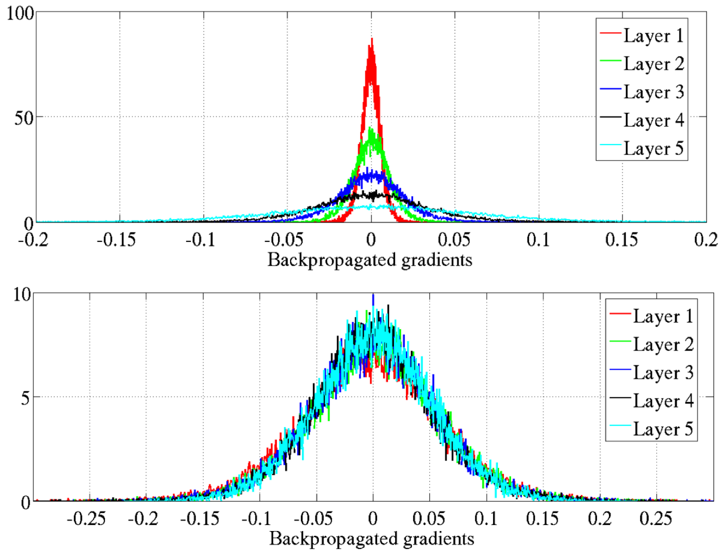


Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

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

$$|w_{kj}^{(0)}| \leq \frac{\sqrt{6}}{\sqrt{|\text{fan-in}|_k + |\text{fan-out}|_k}}$$

Glorot & Bengio, AISTATS, 2010.

INITIALIZATION: EXPERIMENT BY GLOROT & BENGIO (2010)

Table 1: *Test error with different activation functions and initialization schemes for deep networks with 5 hidden layers. N after the activation function name indicates the use of normalized initialization. Results in bold are statistically different from non-bold ones under the null hypothesis test with* $p = 0.005$.

TYPE	Shapeset	MNIST	CIFAR-10	ImageNet
Softsign	**16.27**	**1.64**	55.78	**69.14**
Softsign N	**16.06**	**1.72**	**53.8**	**68.13**
Tanh	27.15	**1.76**	55.9	70.58
Tanh N	**15.60**	**1.64**	**52.92**	**68.57**
Sigmoid	82.61	2.21	57.28	70.66

SoftSign:

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

Hyperbolic tangent:

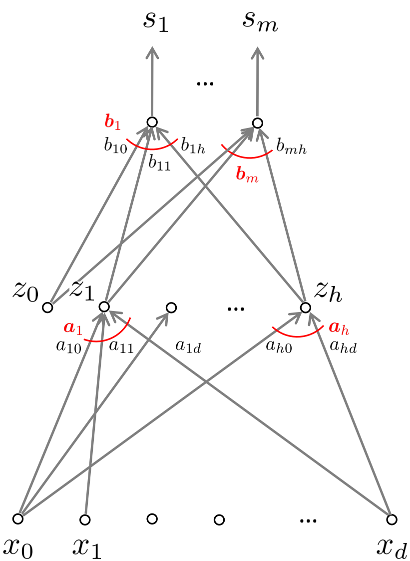
- $\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $\varphi'(x) = 1 - \varphi^2(x)$

Sigmoid (logistic):

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

Glorot & Bengio, AISTATS, 2010.

Momentum



Momentum:

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

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

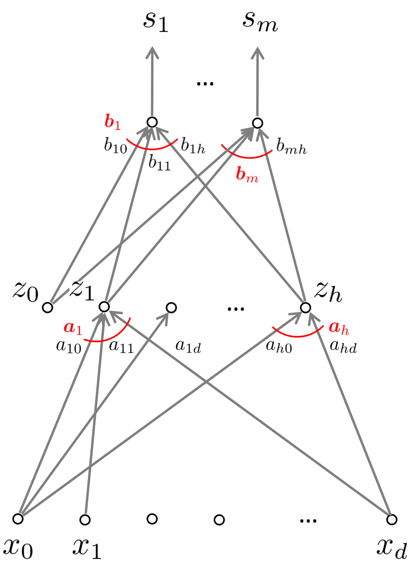
- new weight update

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

$$\mu = \text{momentum constant}$$

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

Vanishing and Exploding Gradient



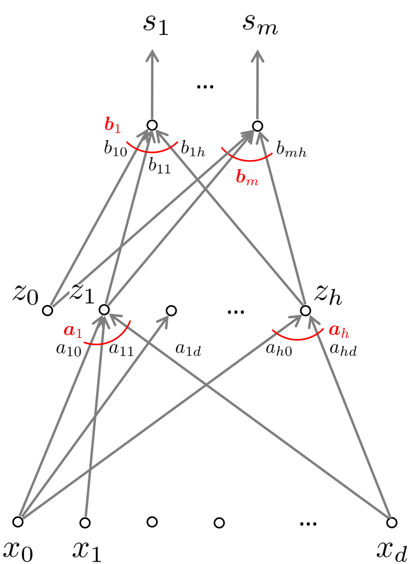
Why it happens?

- $\varphi(\cdot)$ return gradient values in their saturation region
- $\varphi(\cdot) > 1$ can accumulate, cause large gradient, instability

How to combat it?

- unorthodox $\varphi(\cdot)$ such as ReLU or leaky ReLU
- batch normalization
- gradient clipping
- tinker with learning rates

Batch Normalization



Composition of layers:

- gradient: update weights when other weights are unchanged
- practice: all weights are updated simultaneously
- problem: when network very large

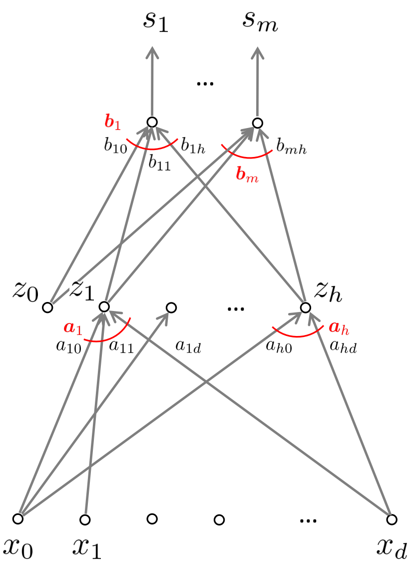
Example:

- deep network w/ one weight w per layer (linear activation)
- output: $x \cdot w_1 \cdots w_{|\text{layers}|}$
- updated output: $x \cdot (w_1 - \eta \frac{\partial E}{\partial w_1}) \cdots (w_{|\text{layers}|} - \eta \frac{\partial E}{\partial w_{|\text{layers}|}})$

Approach:

- z-score normalize outouts in each layer, backpropagate through those operations

Additional Heuristics



Early stopping:

- use validation set to prevent overfitting
- validation set size picked wisely (e.g., $n < |\boldsymbol{w}|$ 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. 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

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

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

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

$t = 1$

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

repeat until convergence

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

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

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

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

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

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

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

else

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

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

end

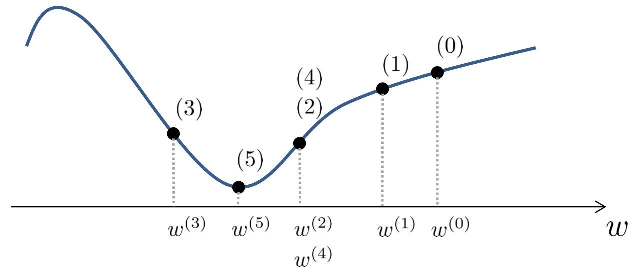
$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$

$t = t + 1$

end

Termination: for $\forall w_{ij}$

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



0) $\frac{\partial E^{(0)}}{\partial w^{(0)}} > 0 \quad \rightarrow \quad \Delta^{(0)} = \Delta_0 \quad \rightarrow \quad w^{(1)} = w^{(0)} - \Delta_0$

1) $\frac{\partial E^{(1)}}{\partial w^{(1)}} > 0 \quad \rightarrow \quad \Delta^{(1)} = \eta_+ \Delta_0 \quad \rightarrow \quad w^{(2)} = w^{(1)} - \eta_+ \Delta_0$

2) $\frac{\partial E^{(2)}}{\partial w^{(2)}} > 0 \quad \rightarrow \quad \Delta^{(2)} = \eta_+^2 \Delta_0 \quad \rightarrow \quad w^{(3)} = w^{(2)} - \eta_+^2 \Delta_0$

3) $\frac{\partial E^{(3)}}{\partial w^{(3)}} < 0 \quad \rightarrow \quad \Delta^{(3)} = \eta_- \eta_+^2 \Delta_0 \quad \rightarrow \quad w^{(4)} = w^{(3)} + \eta_+^2 \Delta_0$

$\frac{\partial E^{(3)}}{\partial w^{(3)}} \leftarrow 0$

4) $\frac{\partial E^{(4)}}{\partial w^{(4)}} > 0 \quad \rightarrow \quad \Delta^{(4)} = \eta_- \eta_+^2 \Delta_0 \quad \rightarrow \quad w^{(5)} = w^{(4)} - \eta_- \eta_+^2 \Delta_0$

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

Riedmiller & Braun, IEEE ICNN, 1993.

Optimization: Stochastic Gradient Descent (SGD)

Algorithm. Stochastic gradient descent.

Input:

Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$
Learning rate schedule: $\eta_1, \eta_2, \ldots \in (0, 1]$
Termination criteria; e.g., the maximum number of steps

Initialization: $\boldsymbol{w} \leftarrow$ initial values

Weight learning:

$t \leftarrow 1$
repeat until termination criteria are satisfied
draw a minibatch $\mathcal{B}$ of m examples $(\boldsymbol{x}, y)$ from $\mathcal{D}$
compute gradient $\nabla E(\boldsymbol{w})$ using $\mathcal{B}$
$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_t \nabla E(\boldsymbol{w})$
$t \leftarrow t + 1$
end

Output: weights $\boldsymbol{w}$

Learning rate schedule:

$$\eta_t = (1 - \alpha)\eta_0 + \alpha\eta_\tau$$

where $\alpha = \frac{t}{\tau}$

Convergence guaranteed if:

$$\sum_{k=1}^{\infty} \eta_k = \infty$$

$$\sum_{k=1}^{\infty} \eta_k^2 < \infty$$

Goodfellow et al. Deep learning, 2016.

Optimization: SGD w/ Momentum

Algorithm. Stochastic gradient descent with momentum.

Input:

Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$
Learning rate $\eta \in (0, 1]$, momentum $\mu \in [0, 1)$
Termination criteria; e.g., the maximum number of steps

Initialization: $\boldsymbol{w}, \Delta\boldsymbol{w} \leftarrow$ initial values

Weight learning:

repeat until termination criteria are satisfied
  draw a minibatch $\mathcal{B}$ of m examples $(\boldsymbol{x}, y)$ from $\mathcal{D}$
  compute gradient $\nabla E(\boldsymbol{w})$ using $\mathcal{B}$
  $\Delta\boldsymbol{w} \leftarrow \mu\Delta\boldsymbol{w} - \eta\nabla E(\boldsymbol{w})$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w}$
end

Output: weights $\boldsymbol{w}$

Momentum choice:

$\mu \in \{0.5, 0.9, 0.99\}$

Consider:

$\nabla E(\boldsymbol{w})$ doesn't change in time

Then:

$\frac{\eta||\nabla E(\mathbf{w})||}{1-\mu}$ is terminal momentum

Thus:

$\mu = 0.9$ leads to $10\times$ update

Goodfellow et al. Deep learning, 2016.

Optimization: Adam

Algorithm. The Adam algorithm. Recommended parameter values are $\eta_0 = 0.001$, $\rho_1 = 0.9$, $\rho_2 = 0.999$, $\delta = 10^{-8}$.

Input:

Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$
Learning rate $\eta_0 \in (0, 1]$, $\rho_1, \rho_2 \in [0, 1)$, δ
Termination criteria; e.g., the maximum number of steps

Initialization: $\boldsymbol{w} \leftarrow$ initial values, $t \leftarrow 0$, $\boldsymbol{u} \leftarrow \mathbf{0}$, $\boldsymbol{v} \leftarrow \mathbf{0}$

Weight learning:

repeat until termination criteria are satisfied
  draw a minibatch $\mathcal{B}$ of m examples $(\boldsymbol{x}, y)$ from $\mathcal{D}$
  compute gradient $\nabla E(\boldsymbol{w})$ using $\mathcal{B}$
  $t \leftarrow t + 1$
  $\boldsymbol{u} \leftarrow \rho_1 \boldsymbol{u} + (1 - \rho_1) \nabla E(\boldsymbol{w})$
  $\boldsymbol{v} \leftarrow \rho_2 \boldsymbol{v} + (1 - \rho_2) \nabla E(\boldsymbol{w})^2$ ← component-wise operation
  $\eta_t \leftarrow \eta_0 \sqrt{1 - \rho_2^t} / (1 - \rho_1^t)$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_t \frac{\boldsymbol{u}}{\sqrt{\boldsymbol{v}} + \delta}$ ← component-wise operation
end

Output: weights $\boldsymbol{w}$

Adam = Adaptive Moment estimation

Insights:

ρ_1, ρ_2 determine η_t

$\eta_t \approx \eta_0$ for large t

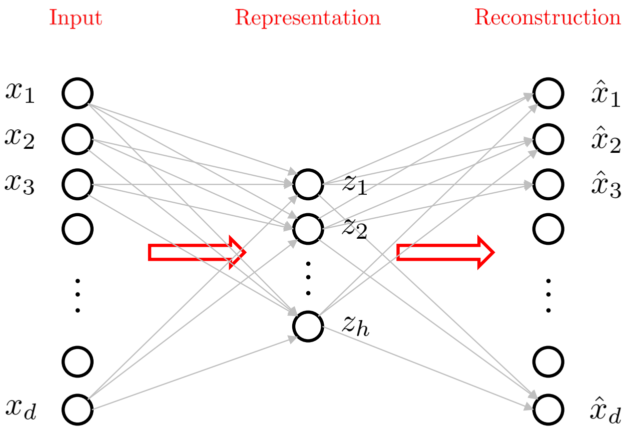
$\frac{\boldsymbol{u}}{\sqrt{\boldsymbol{v}}} \approx \mathbf{sign}(\nabla E(\boldsymbol{w}))$

$\boldsymbol{u}$ = slowly becomes $\nabla E(\boldsymbol{w})$

$\boldsymbol{v}$ = slowly becomes $\nabla E(\boldsymbol{w})^2$

Kingma & Ba, ICLR, 2015.

COMPRESSION

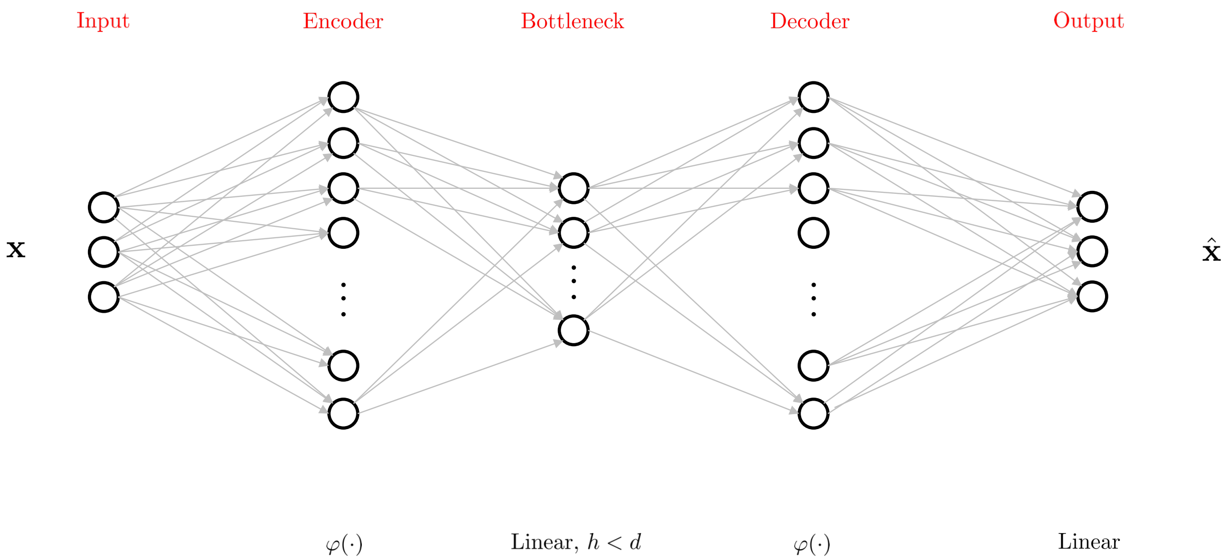


Input	**Hidden**			**Output**
0 0 0 0 0 0 0 1	.10	.00	.91	0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0	.10	.95	.07	0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0	.10	.89	.91	0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0	.94	.03	.13	0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0	.94	.03	.93	0 0 0 1 0 0 0 0
⋮		⋮		⋮

↑

Learned representation **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