CISC6000 Deep Learning
Neural Networks

Dr. Yijun Zhao

Fordham University

Review of Gradient Descent

Formal Problem Setup

Given N observations

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$$

a regression problem tries to uncover the function

$$y_i = f(\mathbf{x_i}) \quad \forall i = 1, 2. \ldots, n$$

such that for a new input value $\mathbf{x}_*$, we can accurately predict the corresponding value

$$y_* = f(\mathbf{x}_*).$$

Error Measure

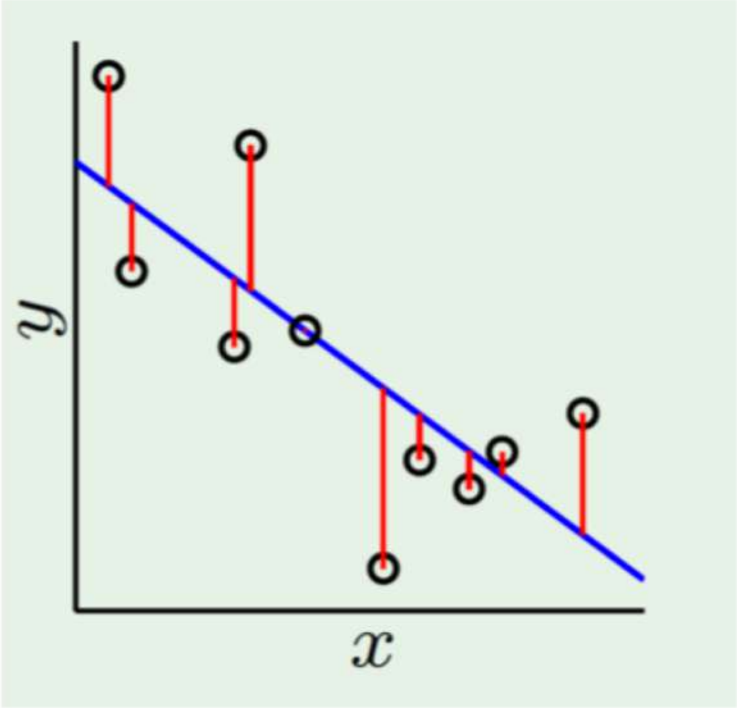
Mean Squared Error (MSE):

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2$$

$$= \frac{1}{N}\parallel \mathbf{X}\mathbf{w} - \mathbf{y} \parallel^2$$

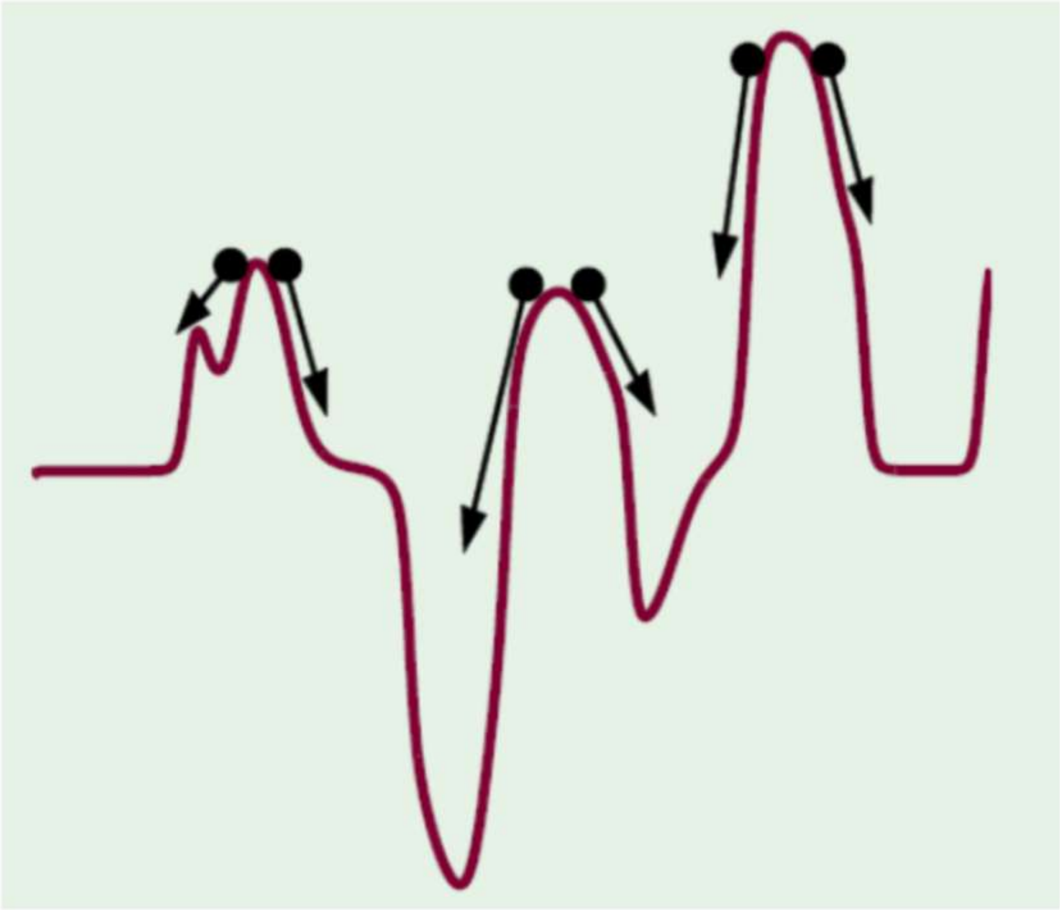
where

$$\mathbf{X} = \begin{bmatrix} — \mathbf{x}_1{}^T — \\ — \mathbf{x}_2{}^T — \\ \dots \\ — \mathbf{x}_N{}^T — \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{bmatrix}$$



Gradient Descent

- Minimize our target function ($E(w)$) by moving down in the steepest direction



Gradient Descent

Gradient Descent Algorithm

- Initialize the $\mathbf{w}(0)$ for time $t = 0$
- for $t = 0, 1, 2, \ldots$ do
- Compute the gradient $\mathbf{g}_t = \nabla E(\mathbf{w}(t))$
- Set the direction to move, $\mathbf{v}_t = -\mathbf{g}_t$
- Update $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$
- Stop if converged
- Return the final weights $\mathbf{w}$

Regression vs. Classification

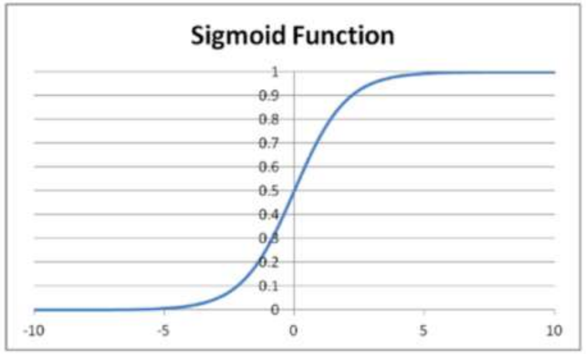
Recall once again regression vs. classification:

				Regression	Classification
Age	Education	Marital Status	Major	Income	Income
35	MS	M	Art	70K	High (1)
40	BS	M	Engineer	65K	High (1)
25	BS	S	Art	40K	Low (0)
50	PhD	D	Engineer	100K	High (1)
30	MS	S	Science	80K	High (1)
28	BS	S	Art	50K	Low (0)
45	PhD	M	Science	90K	High (1)
60	BS	M	Management	120K	High (1)
20	HS	S	Engineer	30K	Low (0)

Logistic Regression

Task: find $\mathbf{w}$ that minimizes $E(\mathbf{w})$

$\sigma(x) = \frac{1}{1+e^{-x}}$ maps $x \in R \rightarrow (0,1)$



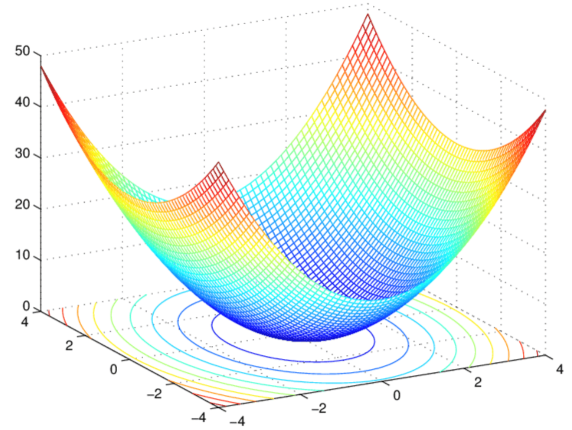
$$-\sum_{i=1}^{N} y_i \log(\sigma(\mathbf{w}^\mathsf{T}\mathbf{x}_i)) - \sum_{i=1}^{N}(1-y_i)\log(1-\sigma(\mathbf{w}^\mathsf{T}\mathbf{x}_i))$$

- No closed-form solution
- Common iterative solutions
  - Gradient Descent: $E(\mathbf{w})$ is convex, so there is one global minimum
  - Newton-Raphson Method (Bishop 4.3.3)
  - Matlab *glmfit* function

Gradient Descent Method

Let $\mathbf{w} = (\omega_0, \omega_1, \omega_2, \ldots, \omega_d)^T$

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



Repeat until convergence

{

$$\omega_j \leftarrow \omega_j - \eta \frac{1}{N} \sum_{i=1}^{N} x_j^i (P(y^i{=}1|\mathbf{x}^i) - y^i)$$

where $j = 0, 1, 2, \ldots, d$ and η is the learning rate (e.g. 0.01)

}

Stochastic Gradient Descent Method

- Batch Gradient Descent:

$$\omega_j \leftarrow \omega_j - \eta \frac{1}{N} \sum_{i=1}^{N} x_j^i (P(y^i=1|\mathbf{x}^i) - y^i)$$

- Stochastic Gradient Descent:
  - Repeat until convergence
    - Get a sample point X_j

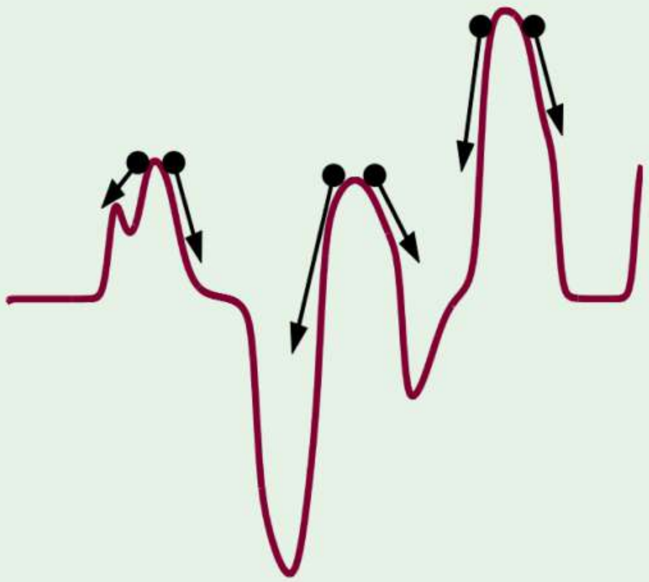
$$\omega_j \leftarrow \omega_j - \eta \, x_j^i (P(y^i=1|\mathbf{x}^i) - y^i)$$

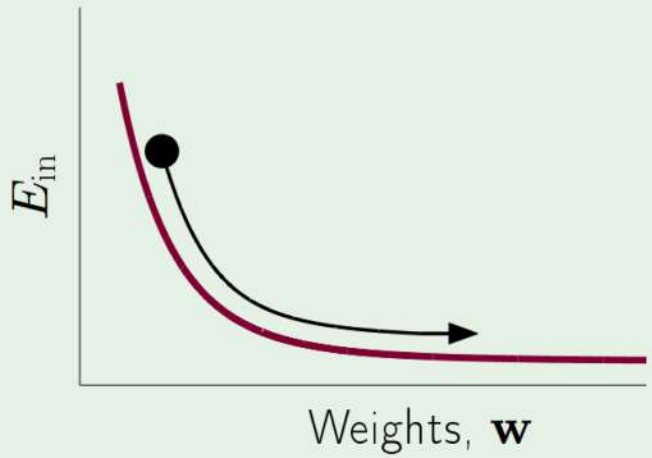
- Mini Batch Gradient Descent: choose a number between 1 and *N*

Benefits of SGD

1. cheaper computation

2. randomization





randomization helps

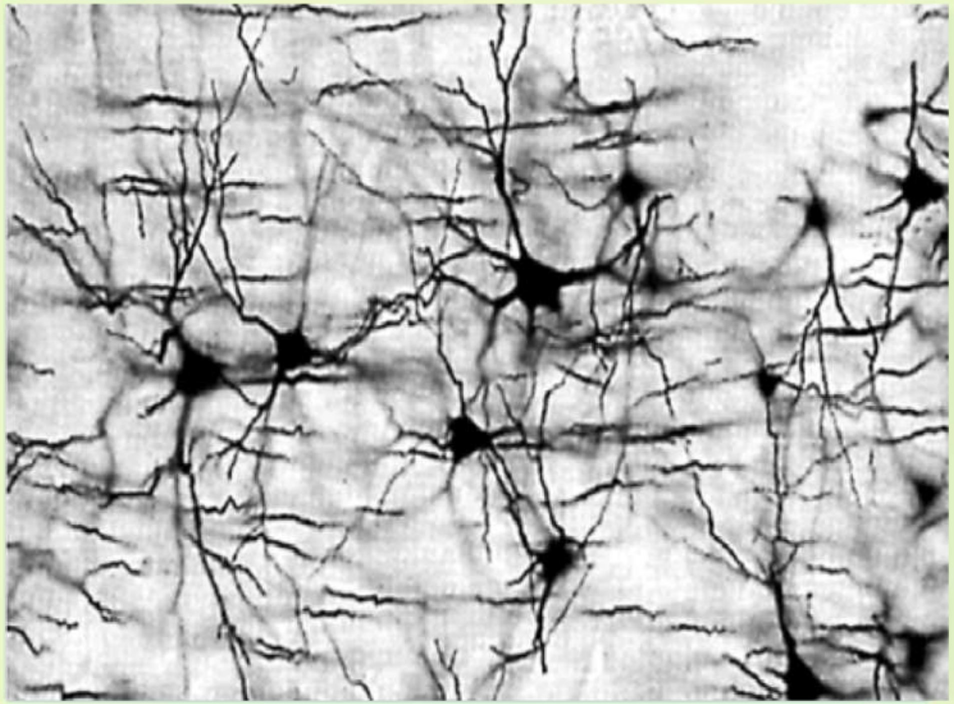
Neural Networks

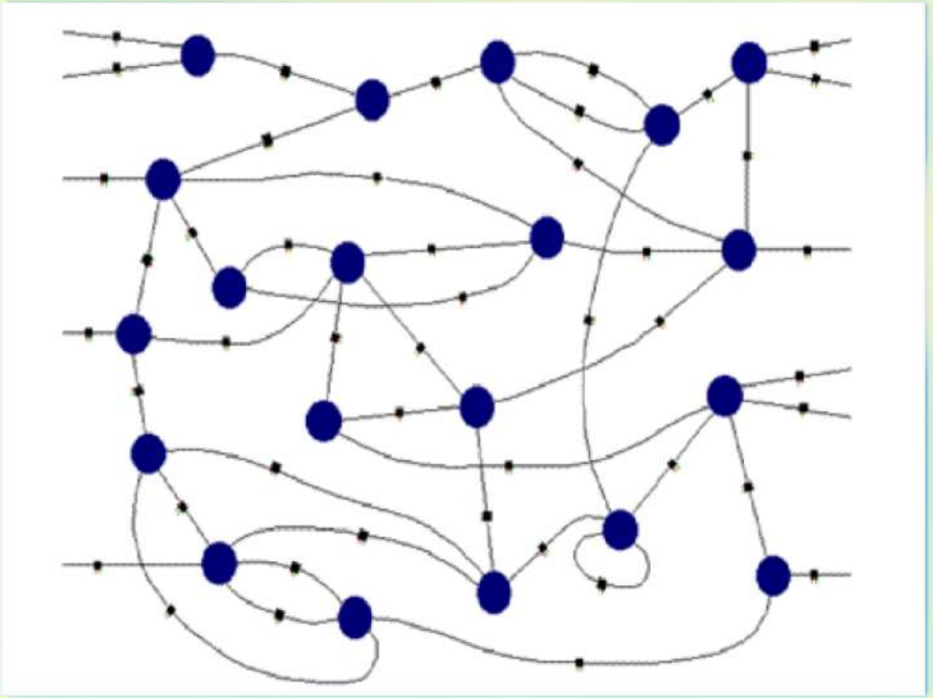
Biological inspiration

biological function $\longrightarrow$ biological structure

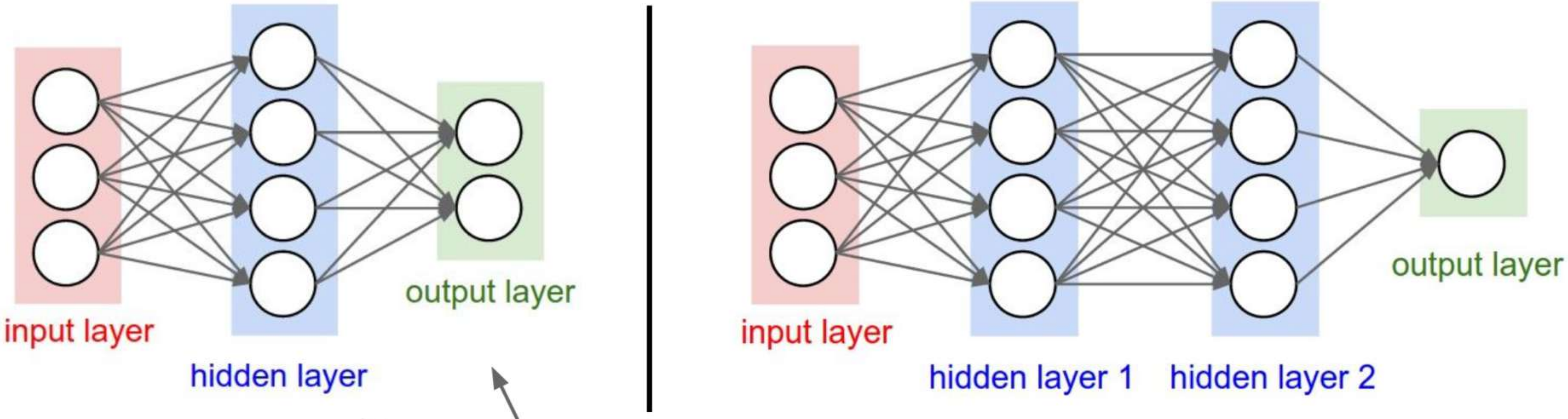
Biological inspiration

biological function $\longrightarrow$ biological structure





Neural Network Architecture

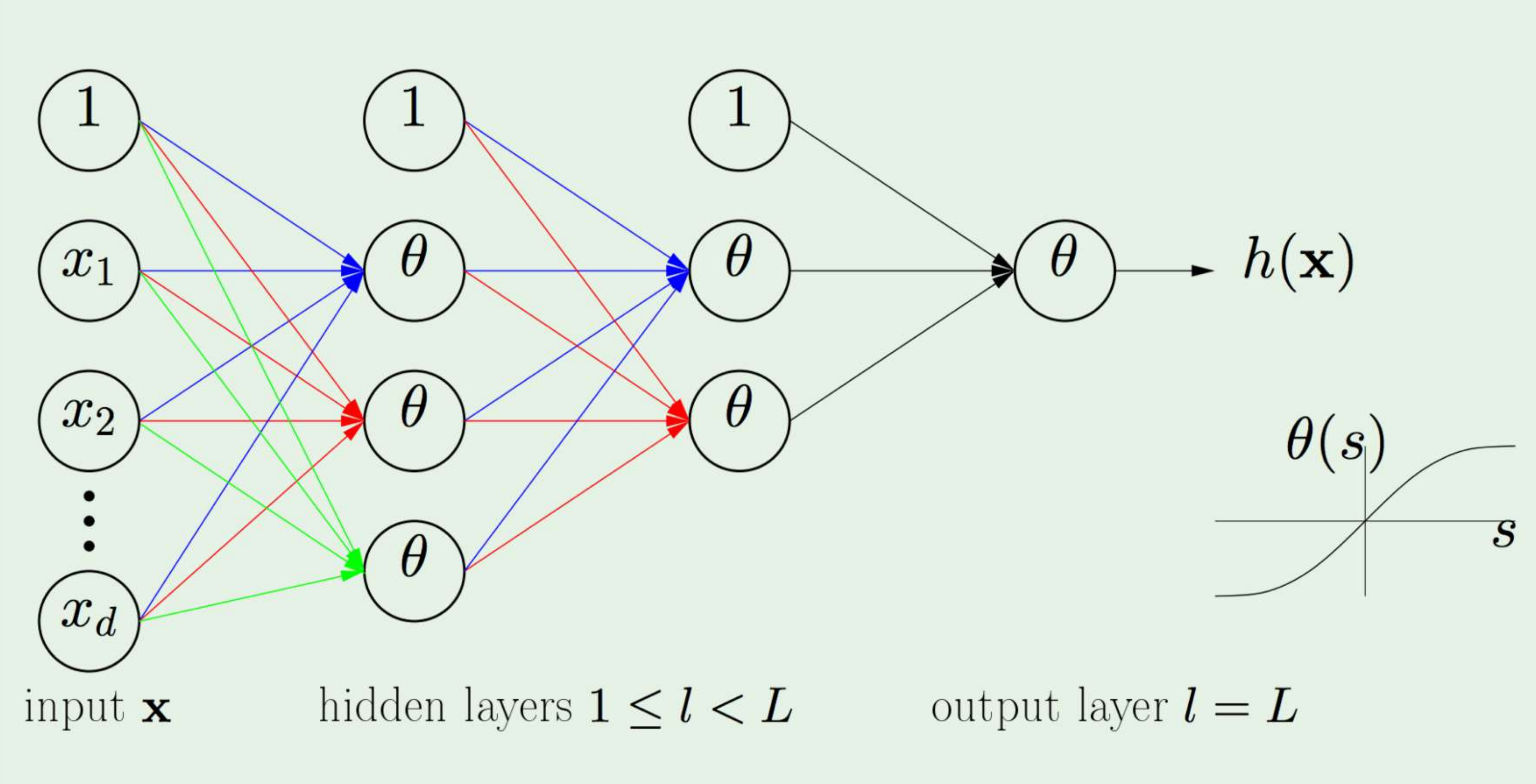


"2-layer Neural Net", or
"1-hidden-layer Neural Net"

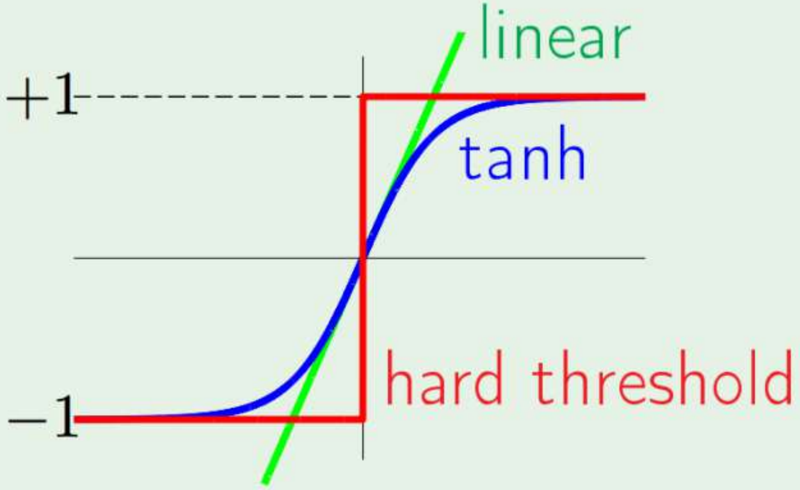
"Fully-connected" layers

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Neural Network



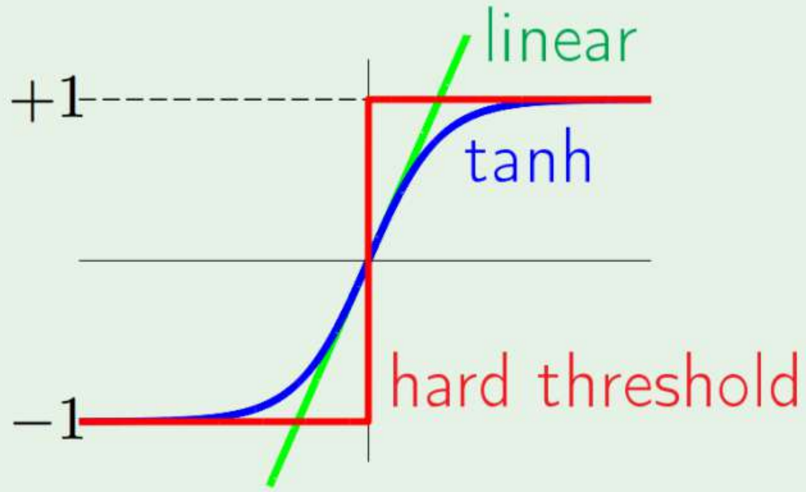
Neural Network



$$\theta(s) = \tanh(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}}$$

Neural Network

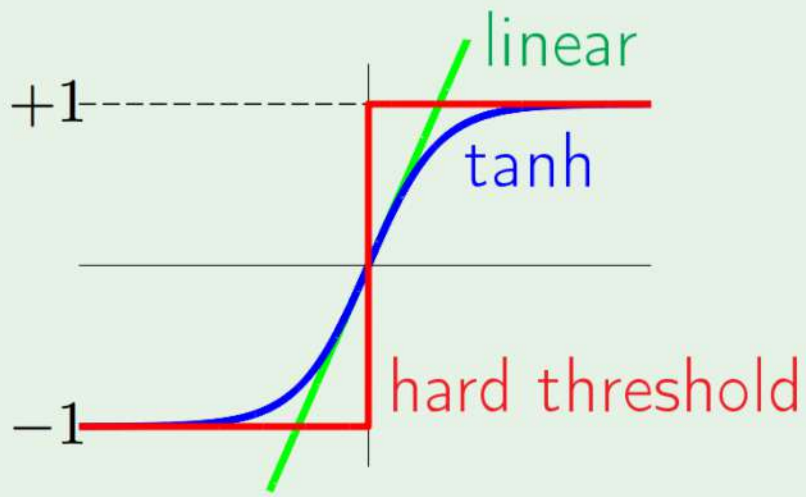
$$w_{ij}^{(l)} \begin{cases} l & \text{layers} \\ i & \text{inputs} \\ j & \text{outputs} \end{cases}$$



$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

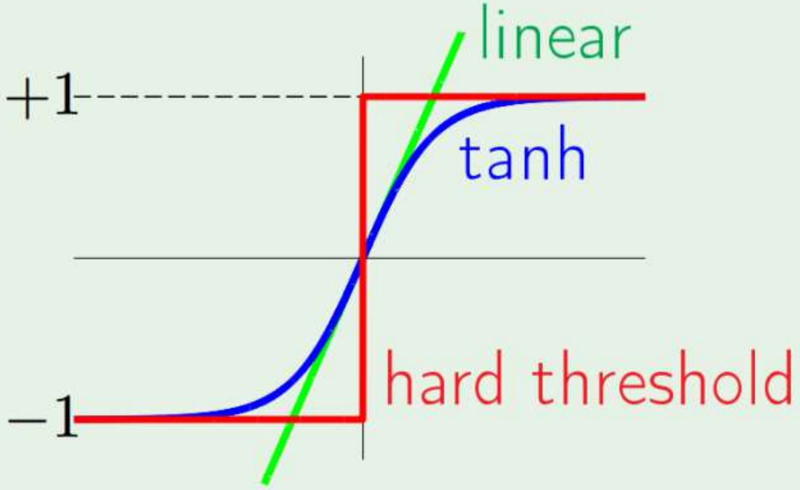
$$w_{ij}^{(l)} \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ \quad\; i & \text{inputs} \\ \quad\; j & \text{outputs} \end{cases}$$



$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

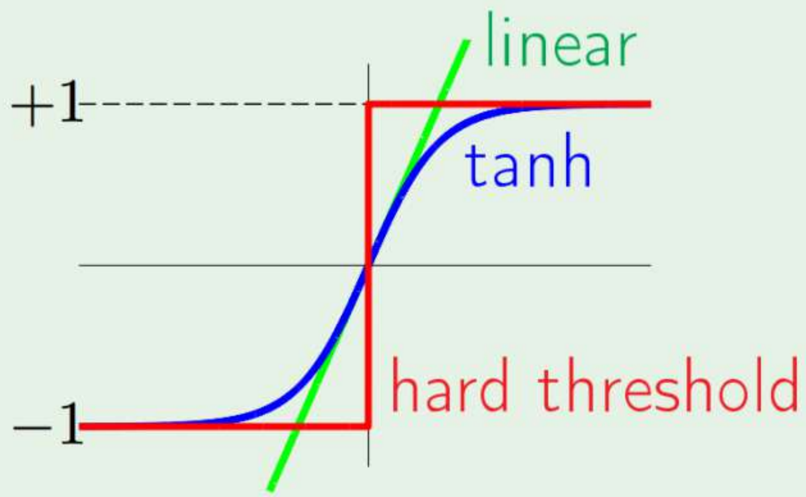
$$w_{ij}^{(l)} \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ \quad j & \text{outputs} \end{cases}$$



$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

$$w_{ij}^{(l)} \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

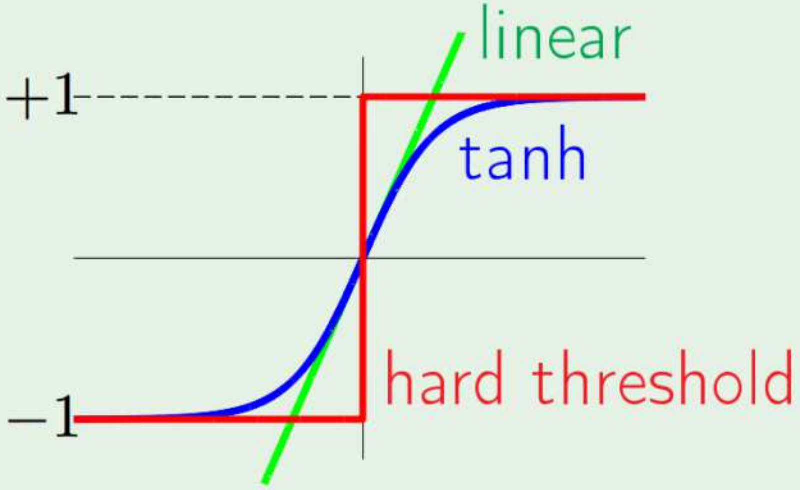


$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

$$w_{ij}^{(l)} \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \qquad x_i^{(l-1)}$$

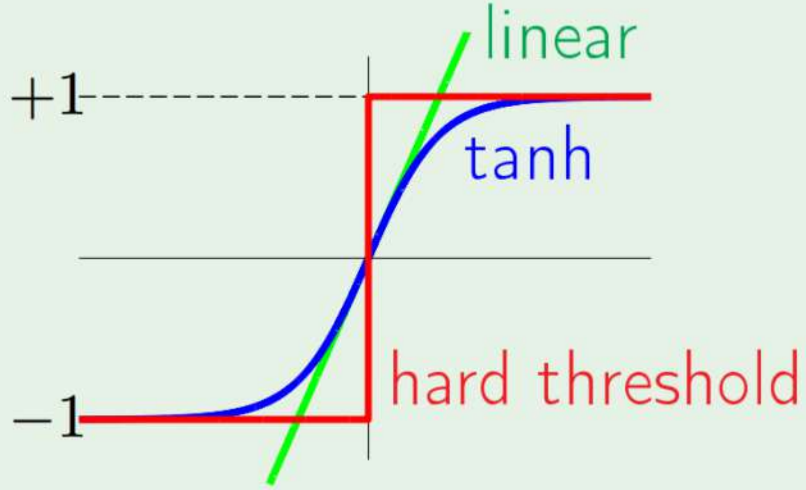


$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

$$w_{ij}^{(l)} \quad \begin{cases} 1 \le l \le L & \text{layers} \\ 0 \le i \le d^{(l-1)} & \text{inputs} \\ 1 \le j \le d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} \, x_i^{(l-1)}$$

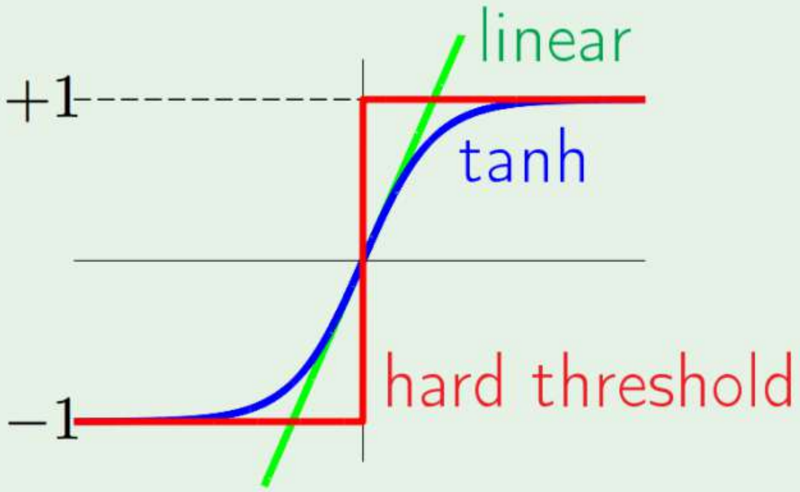


$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

$$w_{ij}^{(l)} \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} \, x_i^{(l-1)}\right)$$

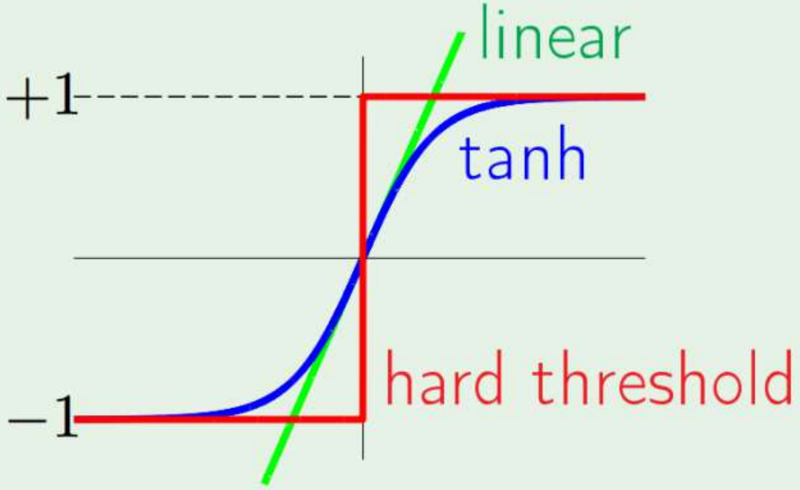


$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

$$w_{ij}^{(l)} \quad \begin{cases} 1 \leq l \leq L & \text{layers} \\ 0 \leq i \leq d^{(l-1)} & \text{inputs} \\ 1 \leq j \leq d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)}\, x_i^{(l-1)}\right)$$



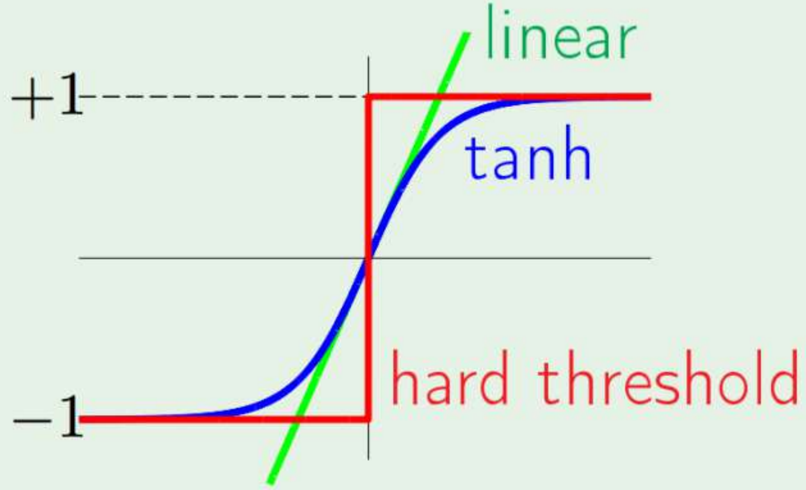
$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

$$w_{ij}^{(l)} \quad \begin{cases} 1 \le l \le L & \text{layers} \\ 0 \le i \le d^{(l-1)} & \text{inputs} \\ 1 \le j \le d^{(l)} & \text{outputs} \end{cases}$$

$$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)}\, x_i^{(l-1)}\right)$$

Apply $\mathbf{x}$ to $x_1^{(0)} \cdots x_{d^{(0)}}^{(0)} \rightarrow \rightarrow x_1^{(L)} = h(\mathbf{x})$



$$\theta(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

Neural Network

All the weights $\mathbf{w} = \{w_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Neural Network

All the weights $\mathbf{w} = \{w_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Error on example $(\mathbf{x}_n, y_n)$ is

$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

Neural Network

All the weights $\mathbf{w} = \{w_{ij}^{(l)}\}$ determine $h(\mathbf{x})$

Error on example $(\mathbf{x}_n, y_n)$ is

$$\mathbf{e}\left(h(\mathbf{x}_n), y_n\right) = \mathbf{e}(\mathbf{w})$$

To implement SGD, we need the gradient

$$\nabla \mathbf{e}(\mathbf{w}): \quad \frac{\partial\ \mathbf{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}} \quad \text{for all } i, j, l$$

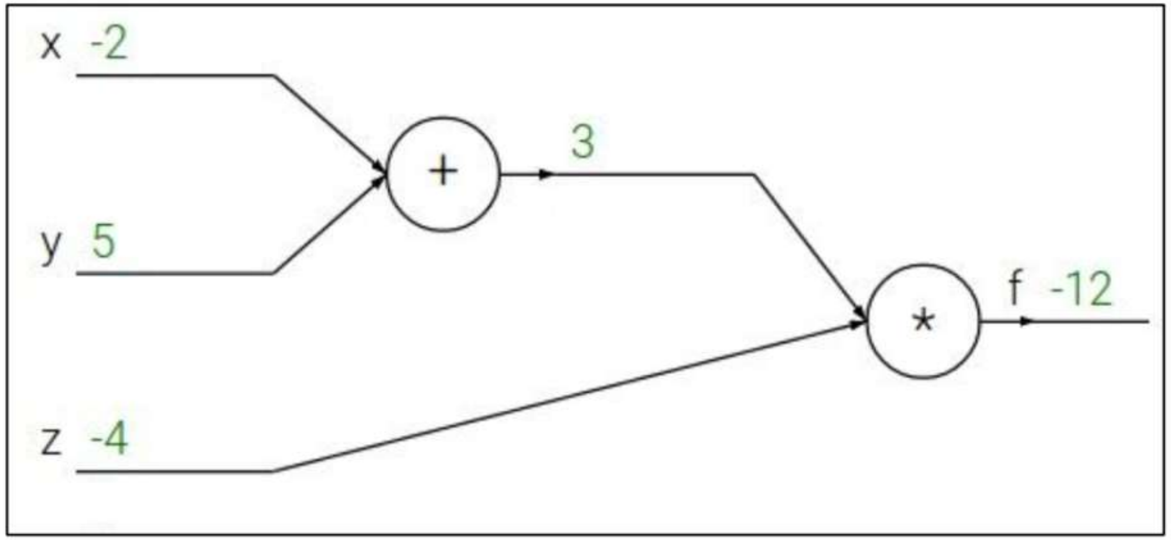
Backpropagation Algorithm

Computing $\dfrac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}}$

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Backpropagation: a simple example

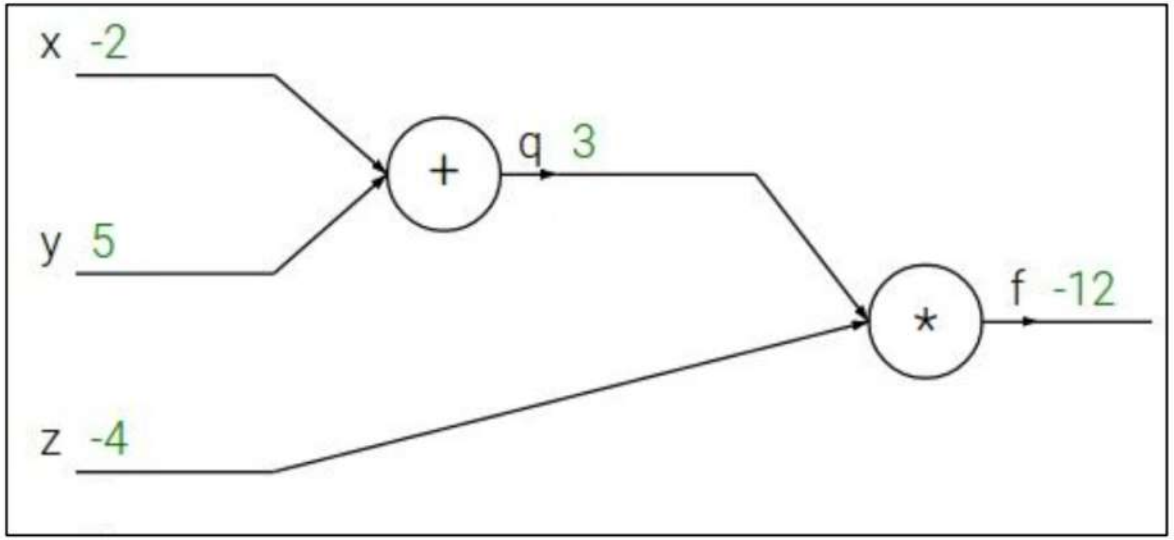
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



Backpropagation: a simple example

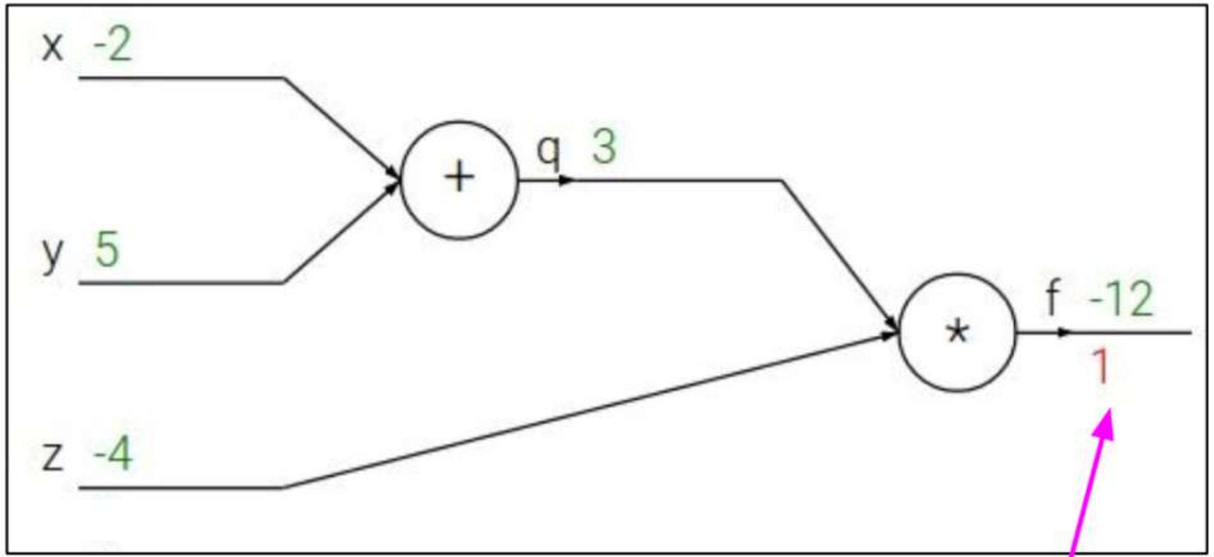
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial f}$$

Backpropagation: a simple example

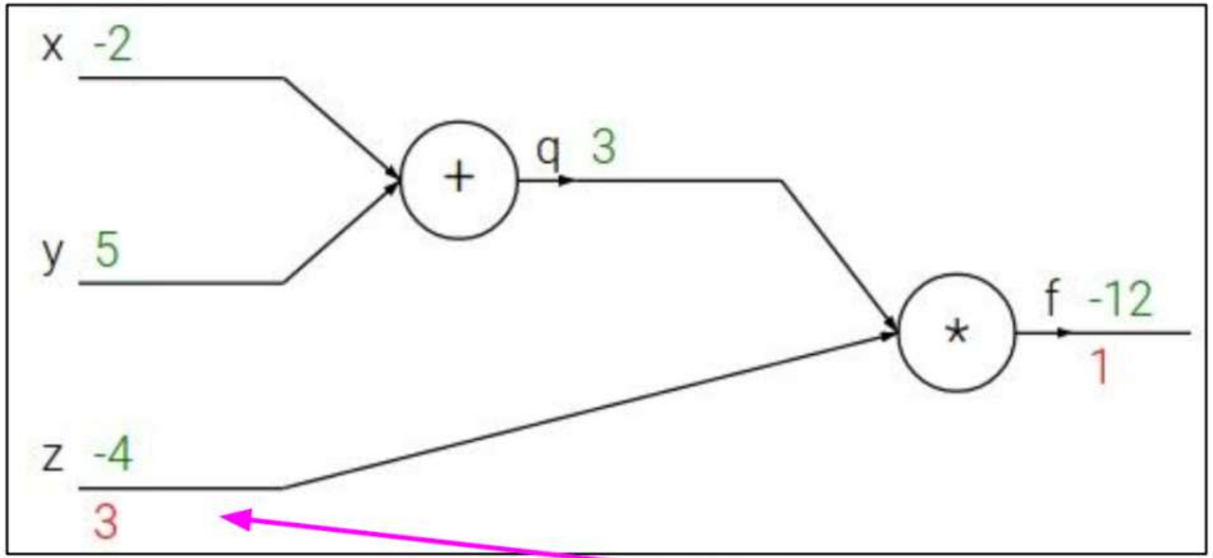
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

Backpropagation: a simple example

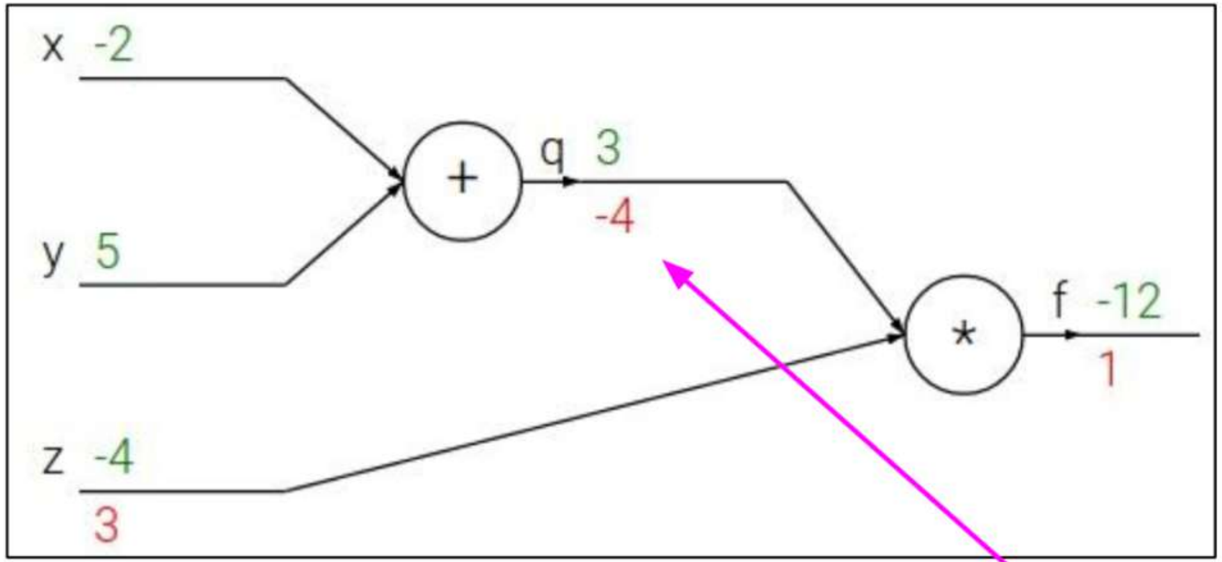
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Backpropagation: a simple example

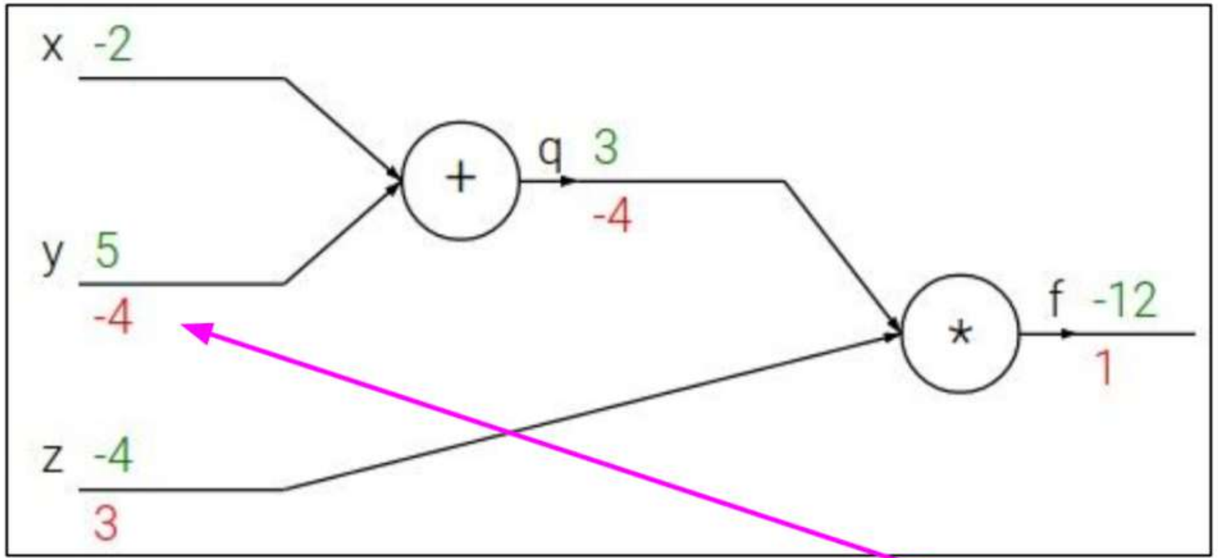
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Backpropagation: a simple example

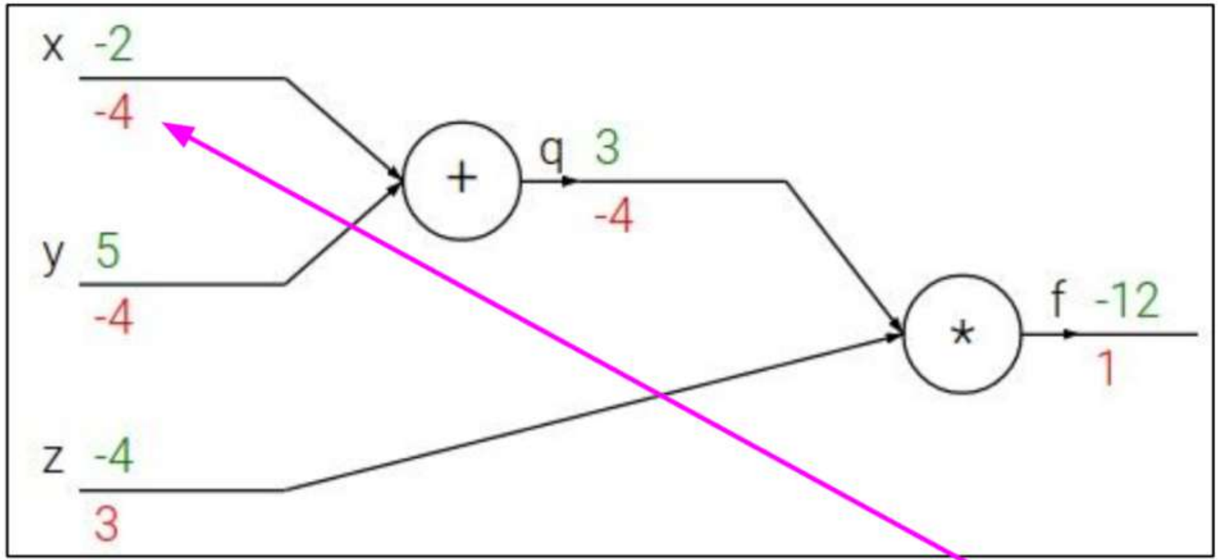
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

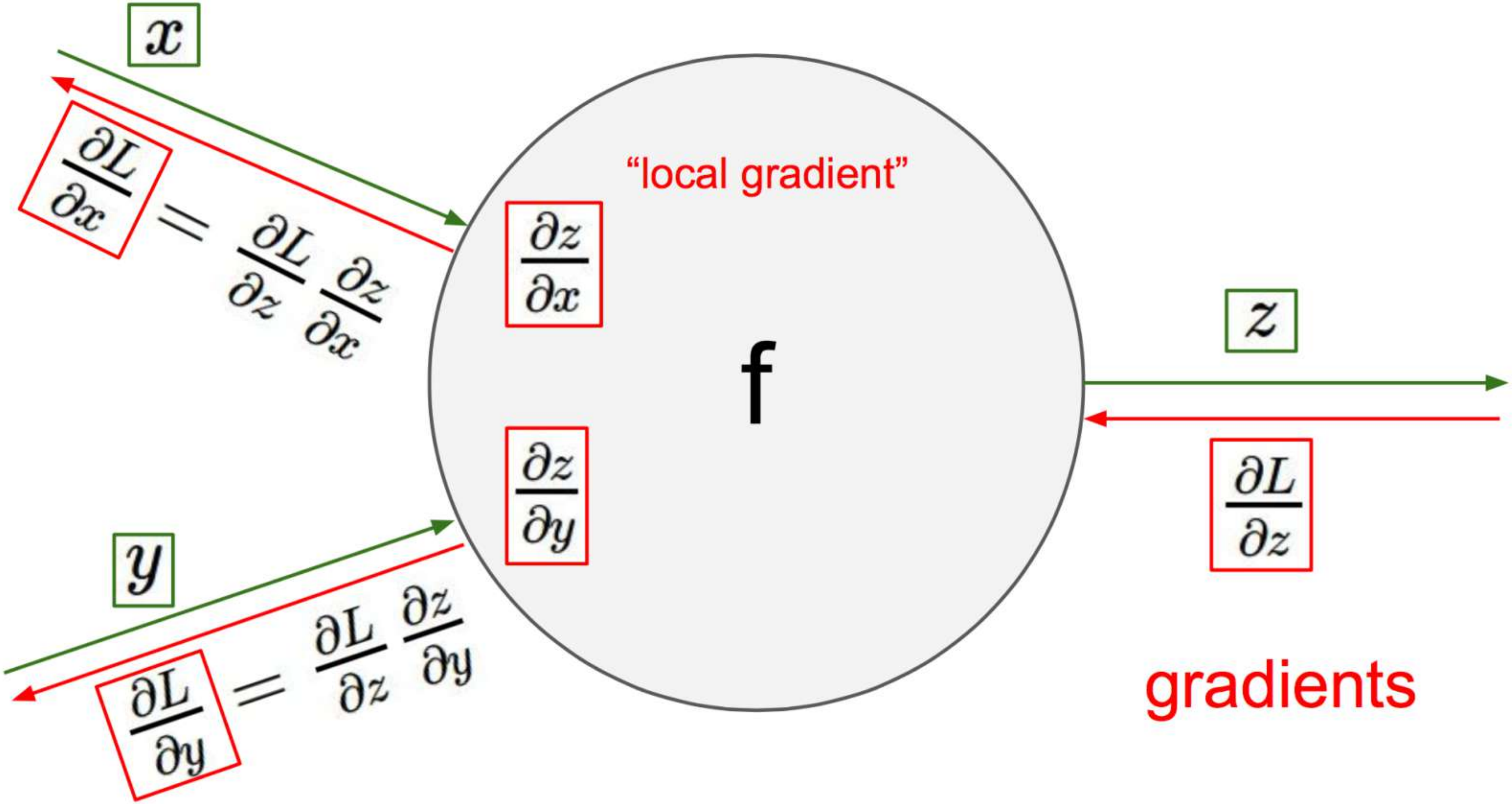
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

Chain rule:

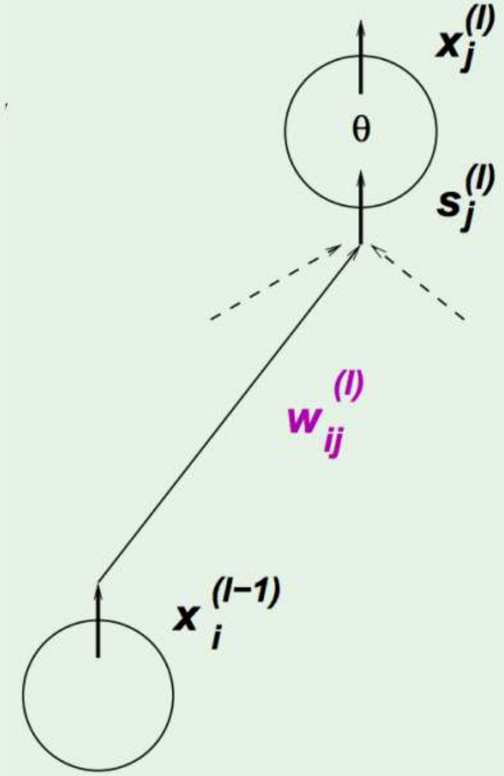
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Backpropagation Algorithm

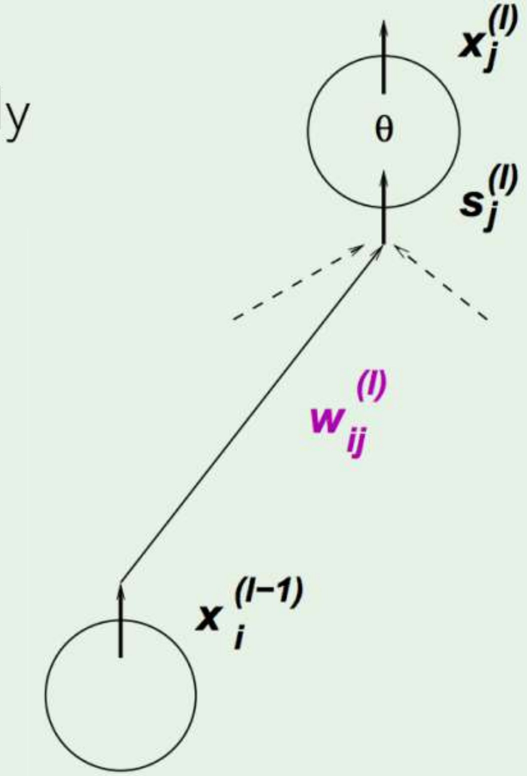
Computing $\dfrac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}}$



Backpropagation Algorithm

Computing $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$

We can evaluate $\dfrac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ w_{ij}^{(l)}}$ one by one: analytically or numerically



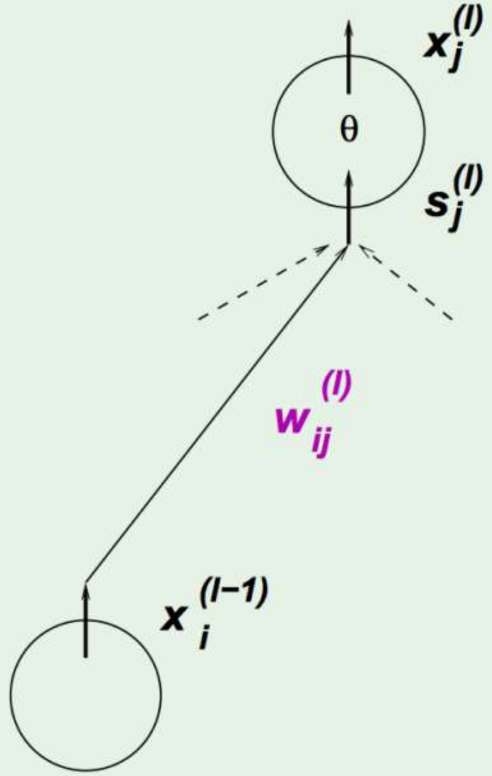
Backpropagation Algorithm

Computing $\dfrac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}}$

We can evaluate $\dfrac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}}$ one by one: analytically or numerically

A trick for efficient computation:

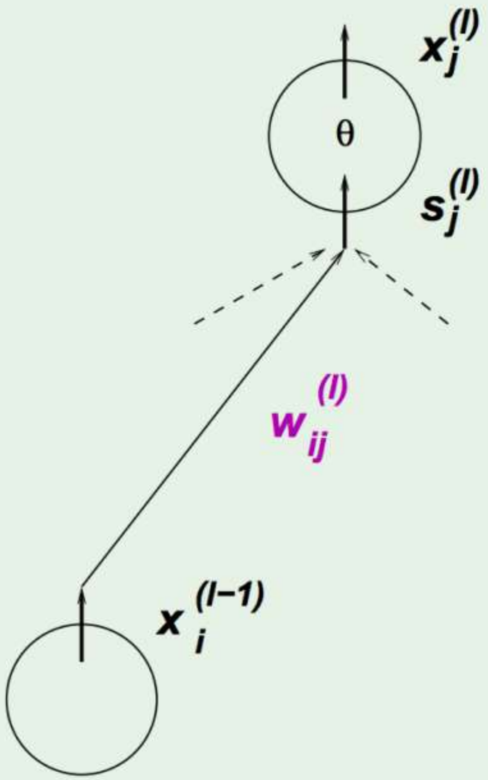
$$\frac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}} = \frac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ s_j^{(l)}} \times \frac{\partial \ s_j^{(l)}}{\partial \ w_{ij}^{(l)}}$$



Backpropagation Algorithm

Computing $\dfrac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}}$

We can evaluate $\dfrac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}}$ one by one: analytically or numerically



A trick for efficient computation:

$$\frac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ w_{ij}^{(l)}} = \frac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ s_j^{(l)}} \times \frac{\partial \ s_j^{(l)}}{\partial \ w_{ij}^{(l)}}$$

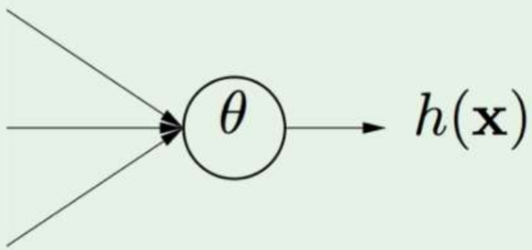
We have $\dfrac{\partial \ s_j^{(l)}}{\partial \ w_{ij}^{(l)}} = x_i^{(l-1)}$ We only need: $\dfrac{\partial \ \mathrm{e}(\mathbf{w})}{\partial \ s_j^{(l)}} = \delta_j^{(l)}$

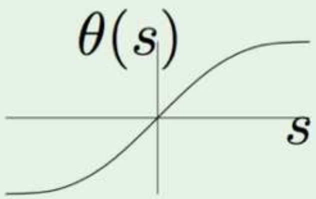
Backpropagation Algorithm

δ for the final layer

$$\delta_j^{(l)} = \frac{\partial \, \mathrm{e}(\mathbf{w})}{\partial \, s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:





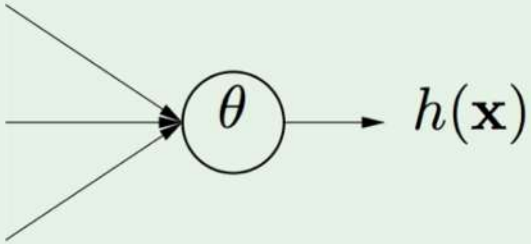
Backpropagation Algorithm

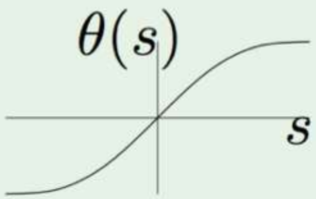
δ for the final layer

$$\delta_j^{(l)} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_1^{(L)}}$$





Backpropagation Algorithm

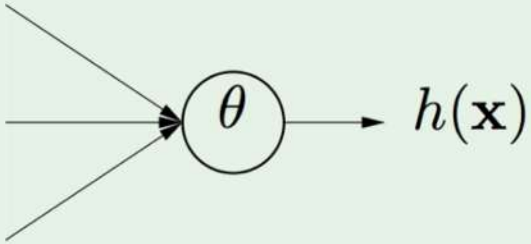
δ for the final layer

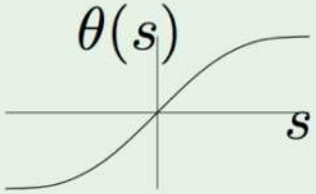
$$\delta_j^{(l)} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial\ \mathrm{e}(\mathbf{w})}{\partial\ s_1^{(L)}}$$

$$\mathrm{e}(\mathbf{w}) = (\ x_1^{(L)} - y_n)^2$$





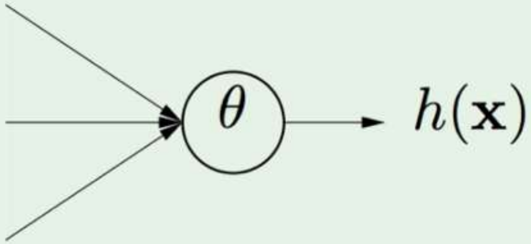
Backpropagation Algorithm

δ for the final layer

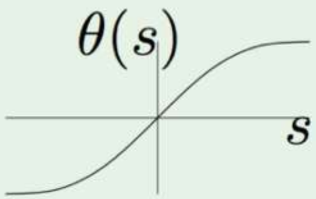
$$\delta_j^{(l)} = \frac{\partial \, \mathrm{e}(\mathbf{w})}{\partial \, s_j^{(l)}}$$

For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial \, \mathrm{e}(\mathbf{w})}{\partial \, s_1^{(L)}}$$



$$\mathrm{e}(\mathbf{w}) = (\, x_1^{(L)} - y_n)^2$$



$$x_1^{(L)} = \theta(s_1^{(L)})$$

Backpropagation Algorithm

δ for the final layer

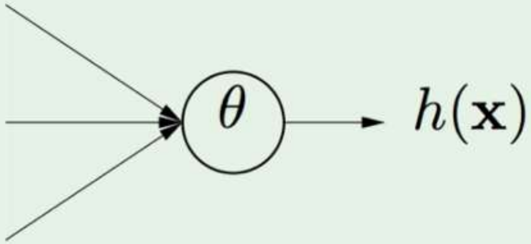
$$\delta_j^{(l)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}}$$

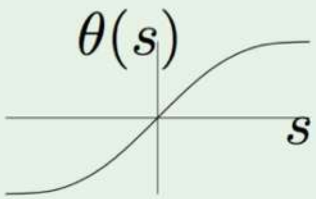
For the final layer $l = L$ and $j = 1$:

$$\delta_1^{(L)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_1^{(L)}}$$

$$\mathrm{e}(\mathbf{w}) = (\, x_1^{(L)} - y_n)^2$$

$$x_1^{(L)} = \theta(s_1^{(L)})$$



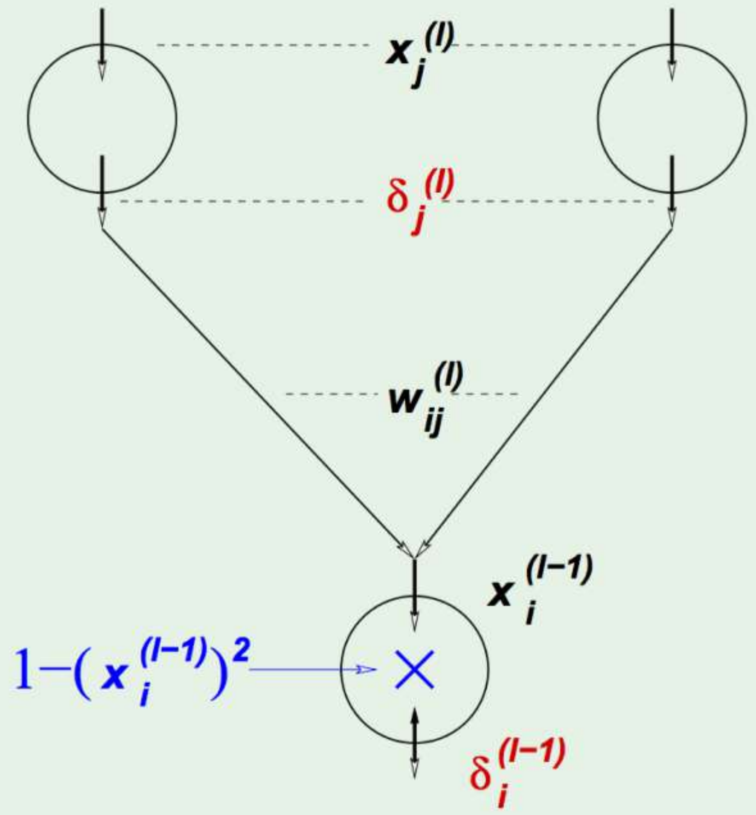


$$\theta'(s) = 1 - \theta^2(s)$$ for the tanh

Backpropagation Algorithm

Back propagation of δ

$$\delta_i^{(l-1)} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_i^{(l-1)}}$$

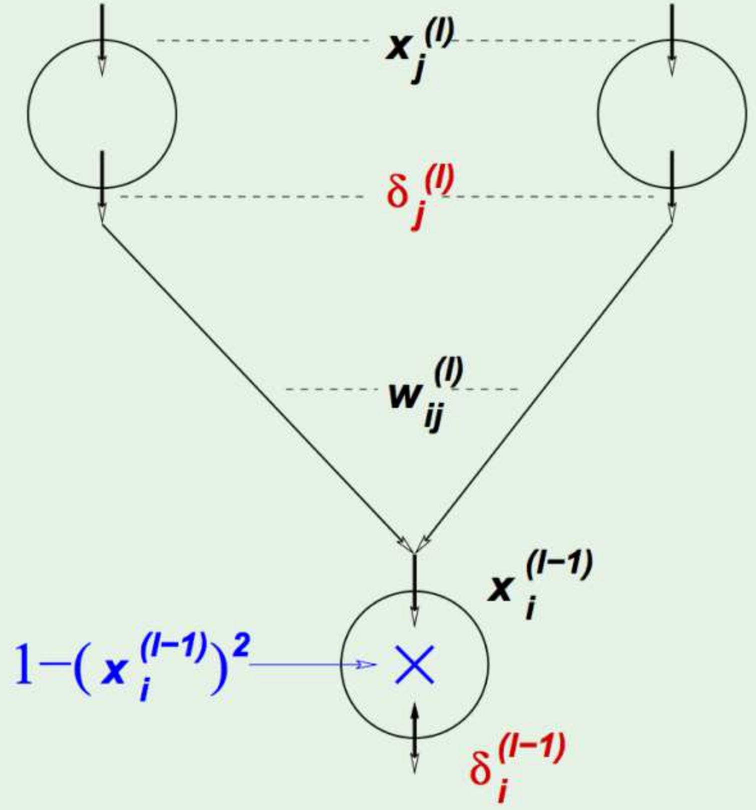


Backpropagation Algorithm

Back propagation of δ

$$
\begin{aligned}
\delta_i^{(l-1)} &= \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, x_i^{(l-1)}} \times \frac{\partial\, x_i^{(l-1)}}{\partial\, s_i^{(l-1)}} \\
&= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)})
\end{aligned}
$$

$$
\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)}\, \delta_j^{(l)}
$$



Backpropagation Algorithm

Initialize all weights $w_{ij}^{(l)}$ at random

for $t = 0, 1, 2, \ldots$ **do**

  Pick $n \in \{1, 2, \cdots, N\}$

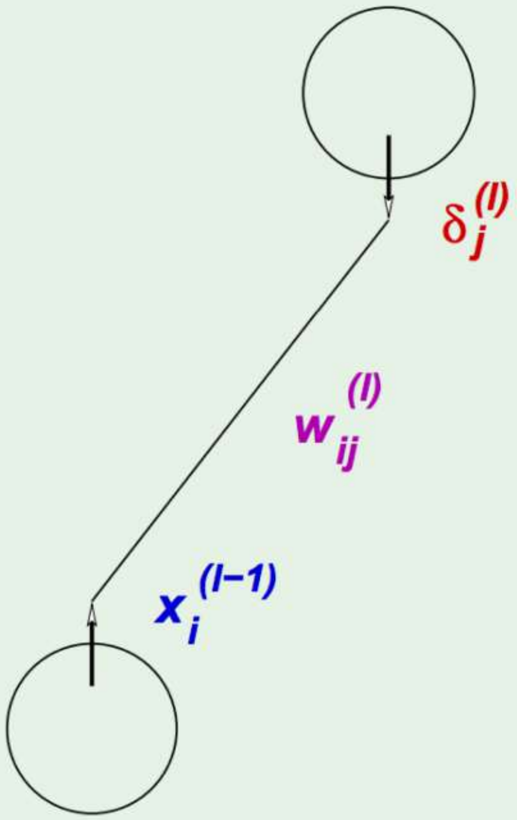
  Forward: Compute all $x_j^{(l)}$

  Backward: Compute all $\delta_j^{(l)}$

  Update the weights: $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\, x_i^{(l-1)} \delta_j^{(l)}$

  Iterate to the next step until it is time to stop

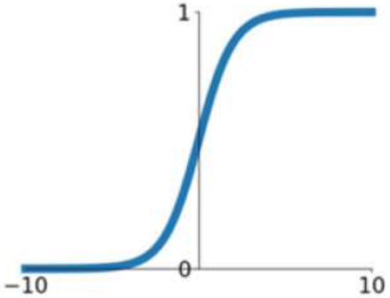
Return the final weights $w_{ij}^{(l)}$



More Activation Functions

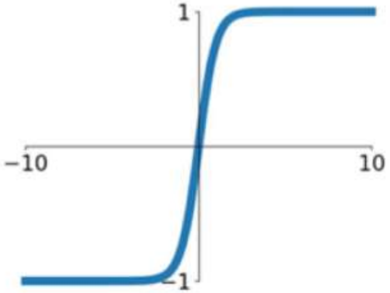
Sigmoid

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



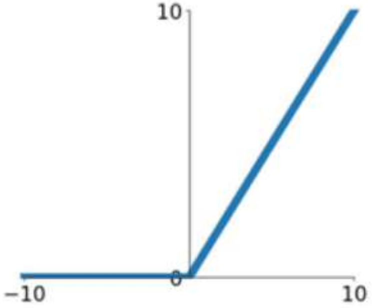
tanh

$\tanh(x)$



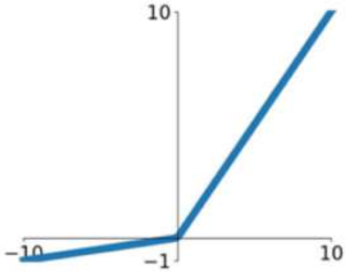
ReLU

$\max(0, x)$



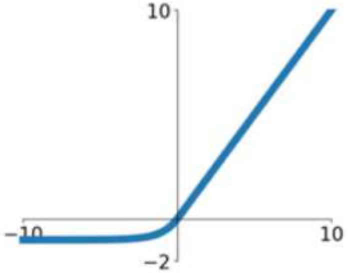
Leaky ReLU

$\max(0.1x, x)$



ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Softmax Function

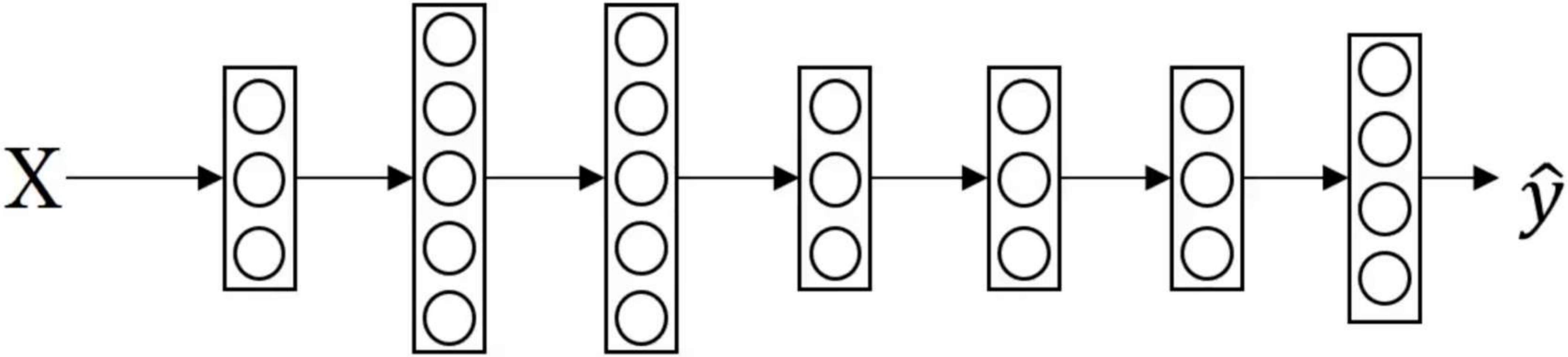
Recognizing cats, dogs, and baby chicks

3 1 2 0 3 2 0 1

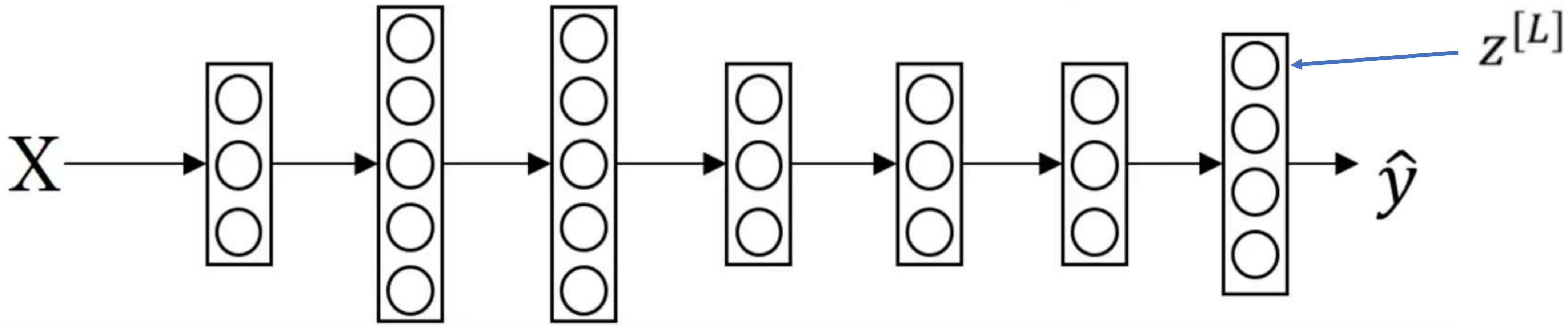
Softmax Function

Recognizing cats, dogs, and baby chicks

3 1 2 0 3 2 0 1



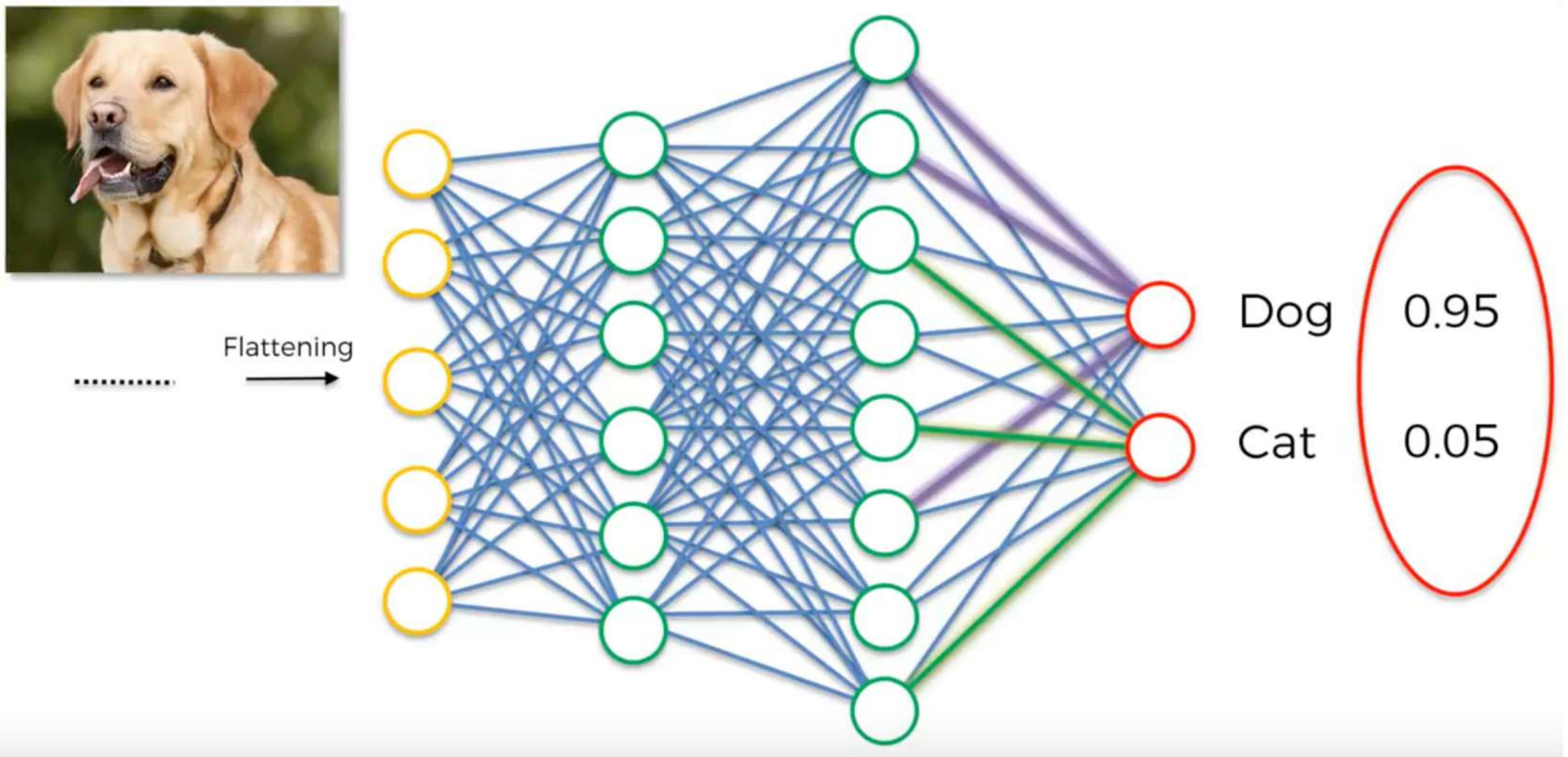
Softmax Function



$$z^{[L]} = W^{[L]}X^{[L-1]} + b^{[L]}$$

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} \qquad \hat{y} = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

Cross Entropy Loss



Cross Entropy Loss

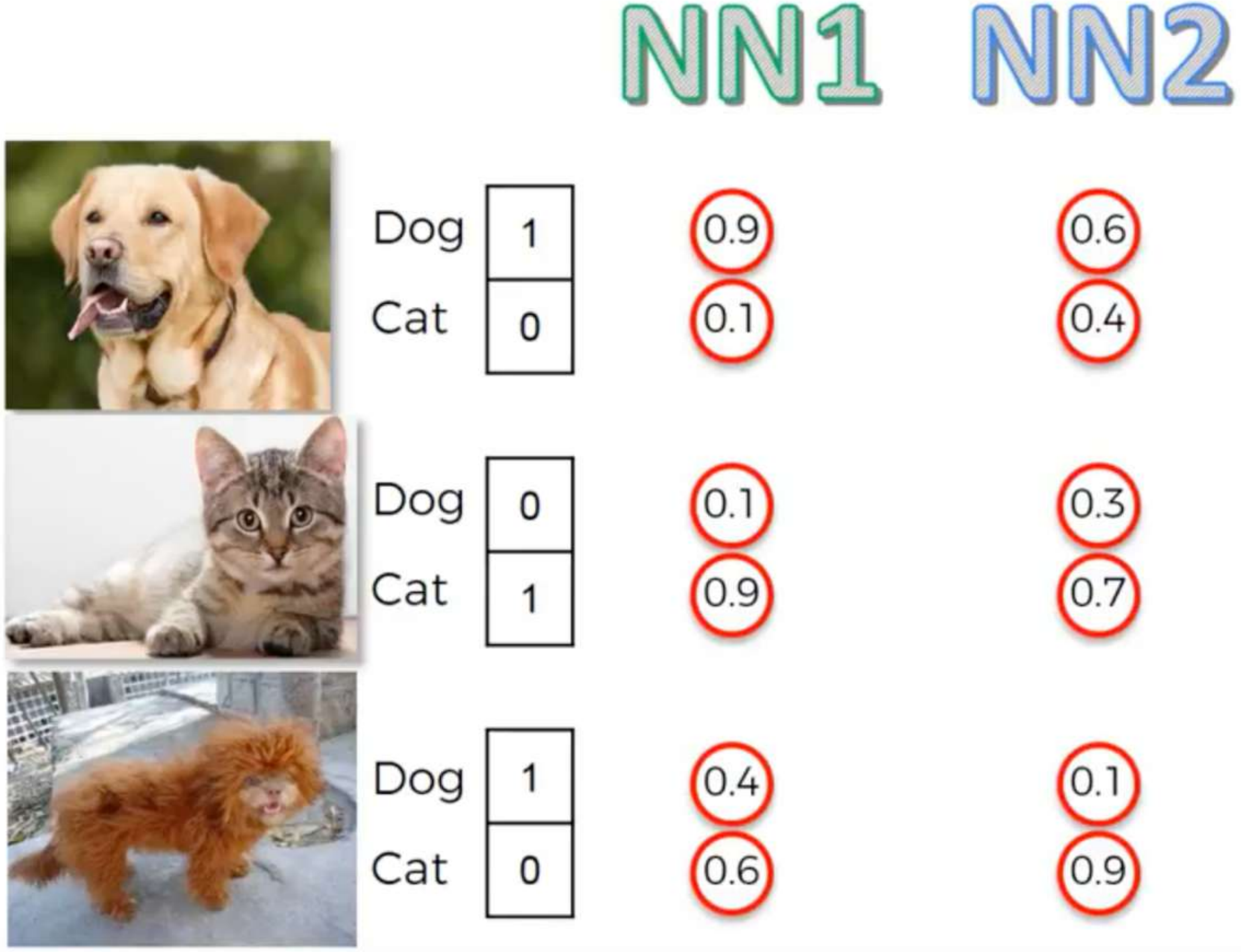
Dog 0.9

Cat 0.1

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

1
0

Cross Entropy Loss



Cross Entropy Loss

NN1

Row	Dog^	Cat^	Dog	Cat
#1	0.9	0.1	1	0
#2	0.1	0.9	0	1
#3	0.4	0.6	1	0

NN2

Row	Dog^	Cat^	Dog	Cat
#1	0.6	0.4	1	0
#2	0.3	0.7	0	1
#3	0.1	0.9	1	0

Classification Error

NN1: 1/3 = 0.33 — NN2: 1/3 = 0.33

Mean Squared Error

NN1: 0.25 — NN2: 0.71

Cross-Entropy

NN1: 0.38 — NN2: 1.06

- Q: what happens when W=0 init is used?