Training Neural Networks

SS 2020 – Machine Perception

Otmar Hilliges

12 March 2020

Advanced Interactive Technologies

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Last lecture

- Perceptron learning algorithm
- MLP as engineering model of a neural network

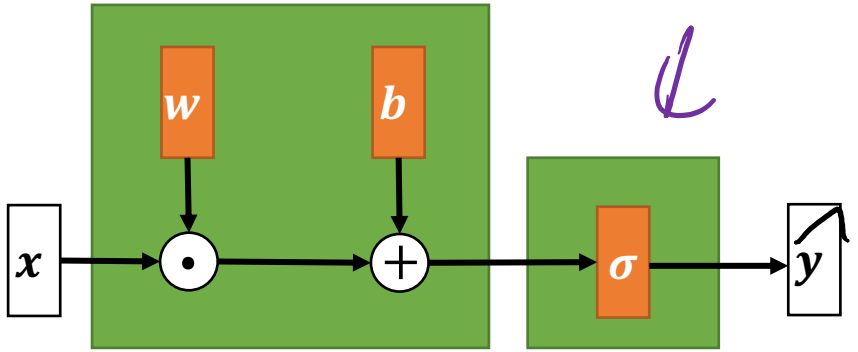
This lecture

- What types of functions can be approximated by neural networks?
    - Universal approximation theorem
- How do we train neural networks?
    - Backprop algorithm

Perceptron - Block diagram

$$w = [w_0, w_1, w_2, \ldots, w_n]$$

$$x = [x_0, x_1, x_2, \ldots, x_n]$$



activation

$$\hat{y} = \sigma(w^T x + b)$$

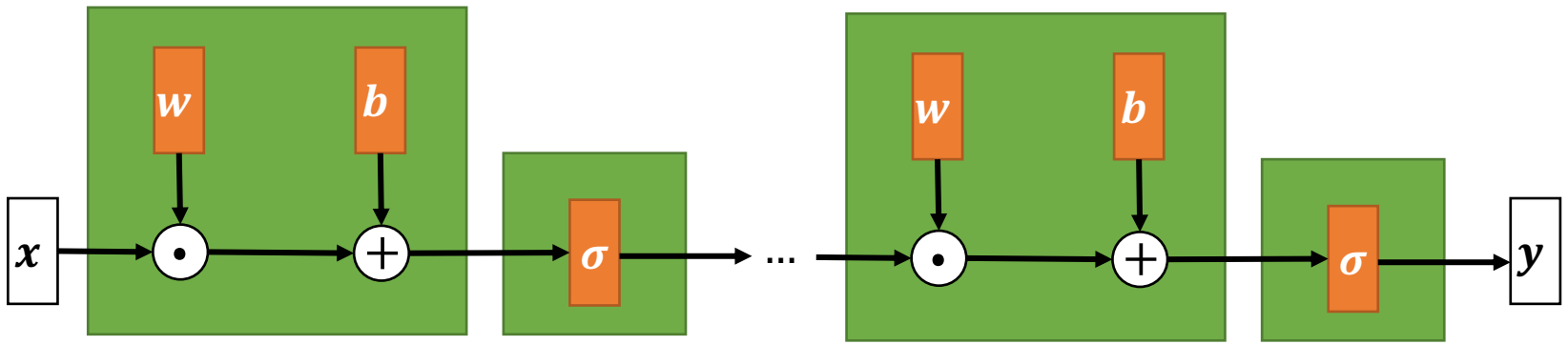
Multi-layered Perceptron - Block diagram

We can combine several layers:

With $\boldsymbol{x}^{(0)} = \boldsymbol{x}$,

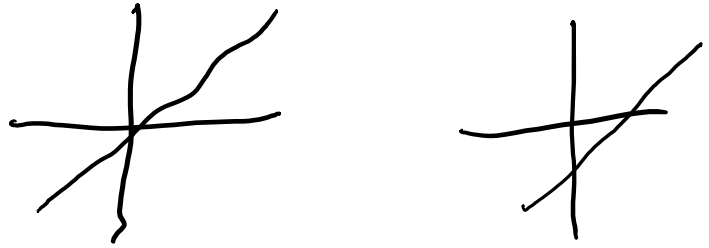
$$\forall l = 1, \dots, L, \quad \boldsymbol{x}^{(l)} = \sigma\left(\boldsymbol{w}^{(l)^T} \boldsymbol{x}^{(l-1)} + b^{(l)}\right)$$

And $f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^{(L)}$



Can we get rid of activation function?
or use other simple function for that?

Linear activation functions?



Assuming that σ is a *linear* transform,

$$\forall x \in \mathbb{R}^n, \sigma(\boldsymbol{x}) = \alpha \boldsymbol{x} + \beta I$$

with $\alpha, \beta \in \mathbb{R}$, we get:

$$\forall l = 1, \dots, L, \quad \boldsymbol{x}^{(l)} = \alpha \boldsymbol{w}^{(l)^T} \boldsymbol{x}^{(l-1)} + \alpha b^{(l)} + \beta I,$$

which results in an *affine* mapping:

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = A^{(L)} \boldsymbol{x} + B^{(L)},$$

where $A^{(0)} = I, B^{(0)} = 0$ and

$$\forall l < L \begin{cases} A^{(l)} = \alpha \boldsymbol{w}^{(l)} A^{(l-1)} \\ B^{(l)} = \alpha \boldsymbol{w}^{(l)} B^{(l-1)} + \alpha b^{(l)} + \beta I \end{cases}$$

Important message: The activation function should be non-linear, or the resulting MLP is an affine mapping with a peculiar parametrization!

Linear activation functions?

Assuming that σ is a *linear* transform,

$$\forall x \in \mathbb{R}^n, \sigma(\boldsymbol{x}) = \alpha \boldsymbol{x} + \beta I$$

with $\alpha, \beta \in \mathbb{R}$, we get:

$$\forall l = 1, \ldots, L, \quad \boldsymbol{x}^{(l)} = \alpha \boldsymbol{w}^{(l)^T} \boldsymbol{x}^{(l-1)} + \alpha b^{(l)} + \beta I,$$

which results in an *affine* mapping:

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = A^{(L)} \boldsymbol{x} + B^{(L)},$$

where $A^{(0)} = I, B^{(0)} = 0$ and

$$\forall l < L \begin{cases} A^{(l)} = \alpha \boldsymbol{w}^{(l)} A^{(l-1)} \\ B^{(l)} = \alpha \boldsymbol{w}^{(l)} B^{(l-1)} + \alpha b^{(l)} + \beta I \end{cases}$$

Important message: The activation function should be non-linear, or the resulting MLP is an affine mapping with a peculiar parametrization!

上页: $\sigma(x) = \alpha x + \beta I$

$x^L = \alpha(w^L x^{L-1} + b^L) + \beta I$

$= \alpha w^L x^{L-1} + \alpha b^L + \beta I = A^L x^{L-1} + B^L$

$\therefore x^{L+1} = \alpha(w^{L+1} x^L + b^{L+1}) + \beta I$

$= \alpha(w^{L+1} \boxed{A^L x^{L-1} + B^L} + b^{L+1}) + \beta I$

$= \underbrace{\alpha w^{L+1} A^L}_{A^{L+1}} \cdot x^{L-1} + \underbrace{\alpha w^{L+1} B^L + \alpha b^{L+1} + \beta I}_{B^{L+1}}$

$\therefore A^L = \alpha w^L A^{L-1}$ $B^L = \alpha w^L B^{L-1} + \alpha b^L + \beta I$

So what happens *with* a non-linearity?

Example: we have non-linearity

and

we don't have non-linerity

Example: Solving 'XOR'

Task learn function $y = f^*(\boldsymbol{x})$ that maps binary variables x_1, x_2 to "true" or "false" if one and only one $x_i == 1$

We will use function $f(\boldsymbol{x}; \Theta)$ and find parameters Θ to make $f^* \approx f$

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

异或门（英語：Exclusive-OR gate，簡稱XOR gate，又稱EOR gate、ExOR gate）是数字逻辑中实现逻辑异或的逻辑门

输入		输出
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Attempt I: Use a linear model?

Assuming a linear function: mean square error

$$J(\Theta) = \frac{1}{(N=4)} \sum_{x \in X} \left(f^*\left(x^{(i)}\right) - f\left(x^{(i)}; \Theta\right)\right)^2 \quad \text{with } f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^T \boldsymbol{w} + b$$

Solving via normal equation:

$$\boldsymbol{w} = \mathbf{0} \text{ and } b = \frac{1}{2}$$

Normal Equation

Given a matrix equation

$$\mathbf{A}\,\mathbf{x} = \mathbf{b},$$

the normal equation is that which minimizes the sum of the square differences between the left and right sides:

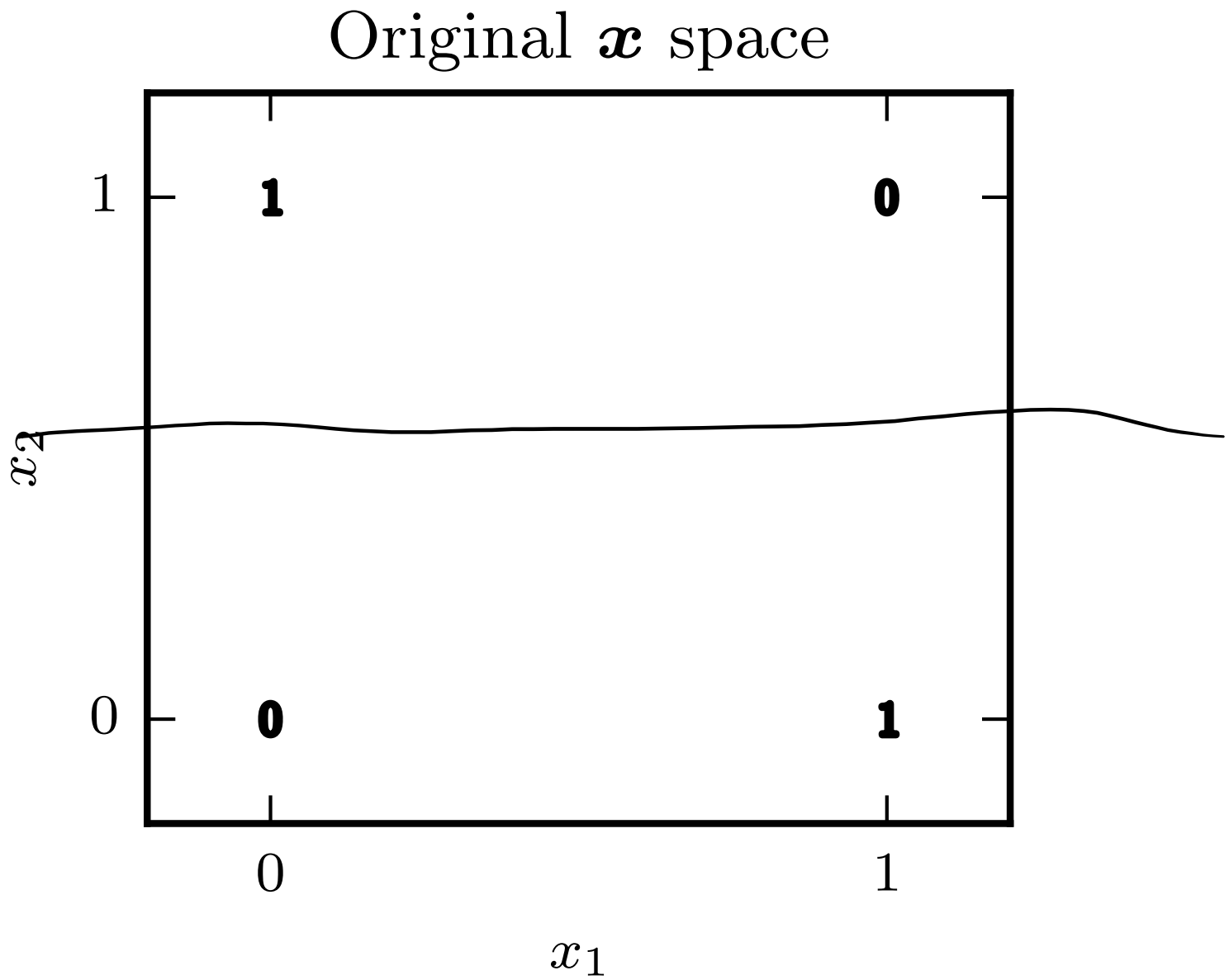
$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\,\mathbf{x} = \mathbf{A}^{\mathrm{T}}\mathbf{b}.$$

It is called a normal equation because $\mathbf{b} - \mathbf{A}\,\mathbf{x}$ is normal to the range of $\mathbf{A}$.

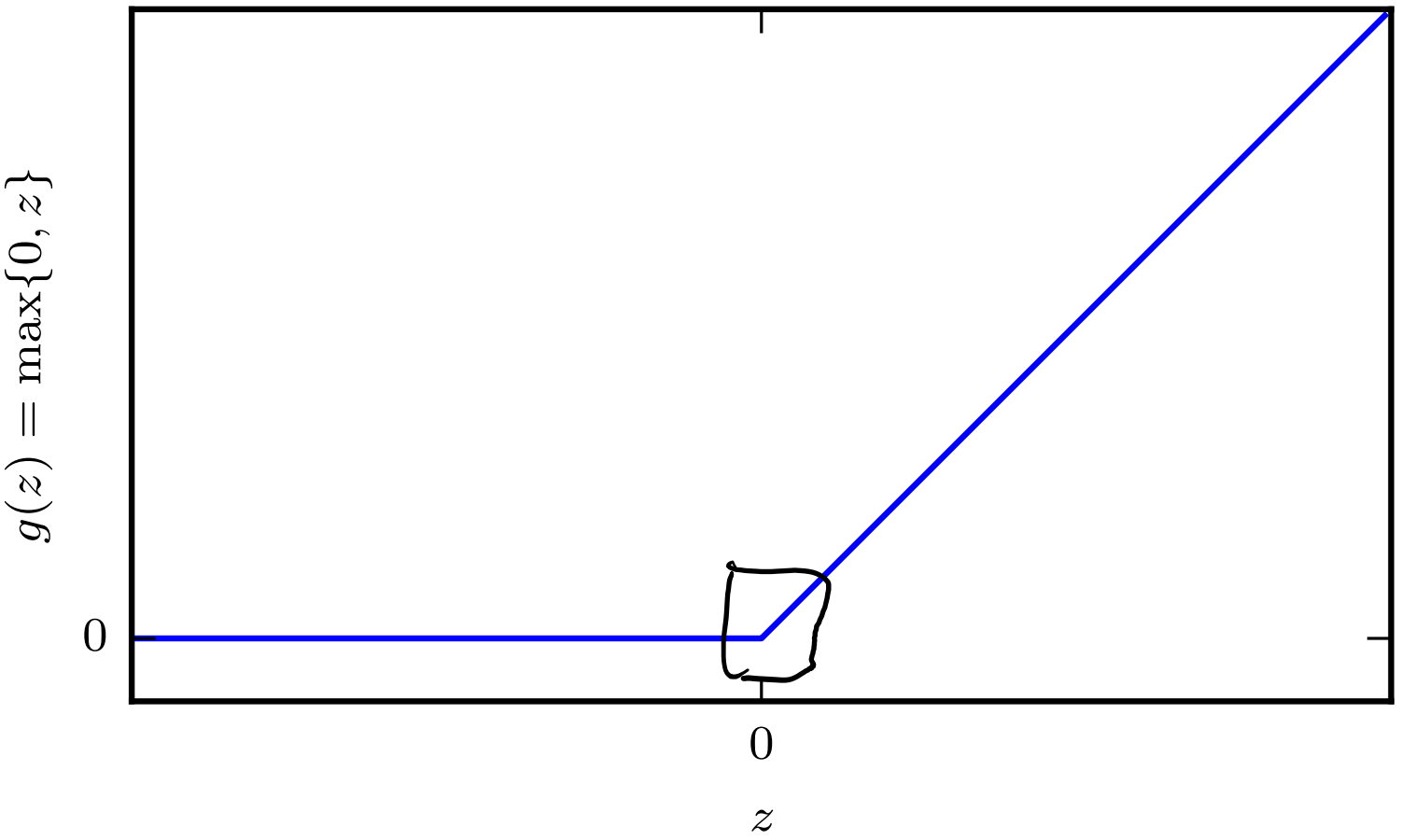
Here, $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is a normal matrix.

Visualization

bold output one the plane is the value of y.



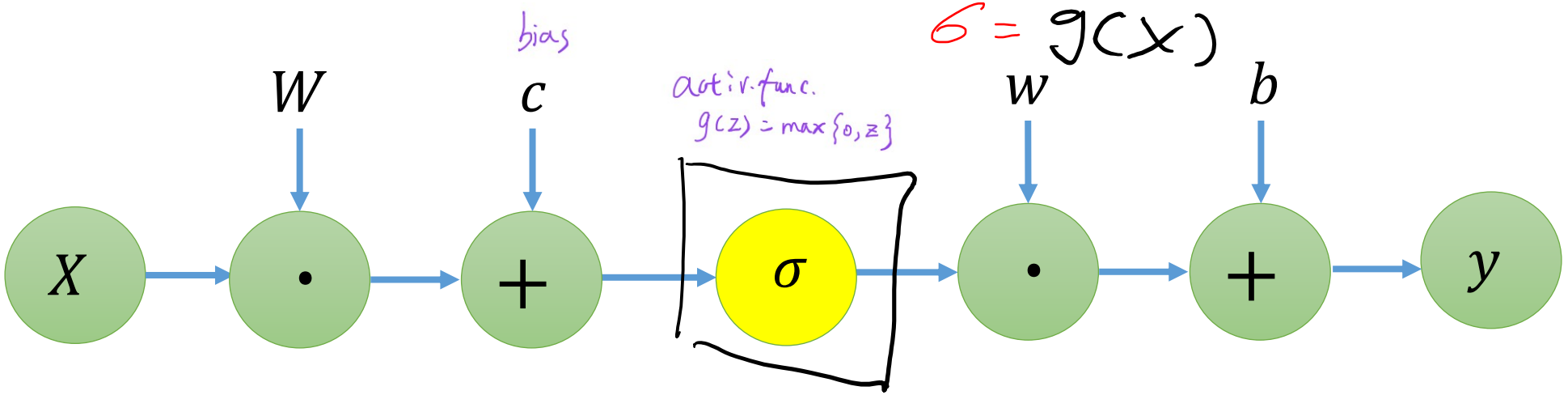
ReLu (non-linear)



$$g(z) = \max\{0, z\}$$

Feedforward Neural Network

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, XW + \boldsymbol{c}\} + b$$



The XOR Multi-Layered Perceptron

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Function we want to learn:

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, XW + \boldsymbol{c}\} + b$$

Oracle solution:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad b = 0$$

.

X W

$$XW = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

The XOR Multi-Layered Perceptron

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Function we want to learn:

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, XW + \boldsymbol{c}\} + b$$

Oracle solution:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad b = 0$$

$$XW + c \qquad = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

The XOR Multi-Layered Perceptron

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Function we want to learn:

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, XW + \boldsymbol{c}\} + b$$

Oracle solution:

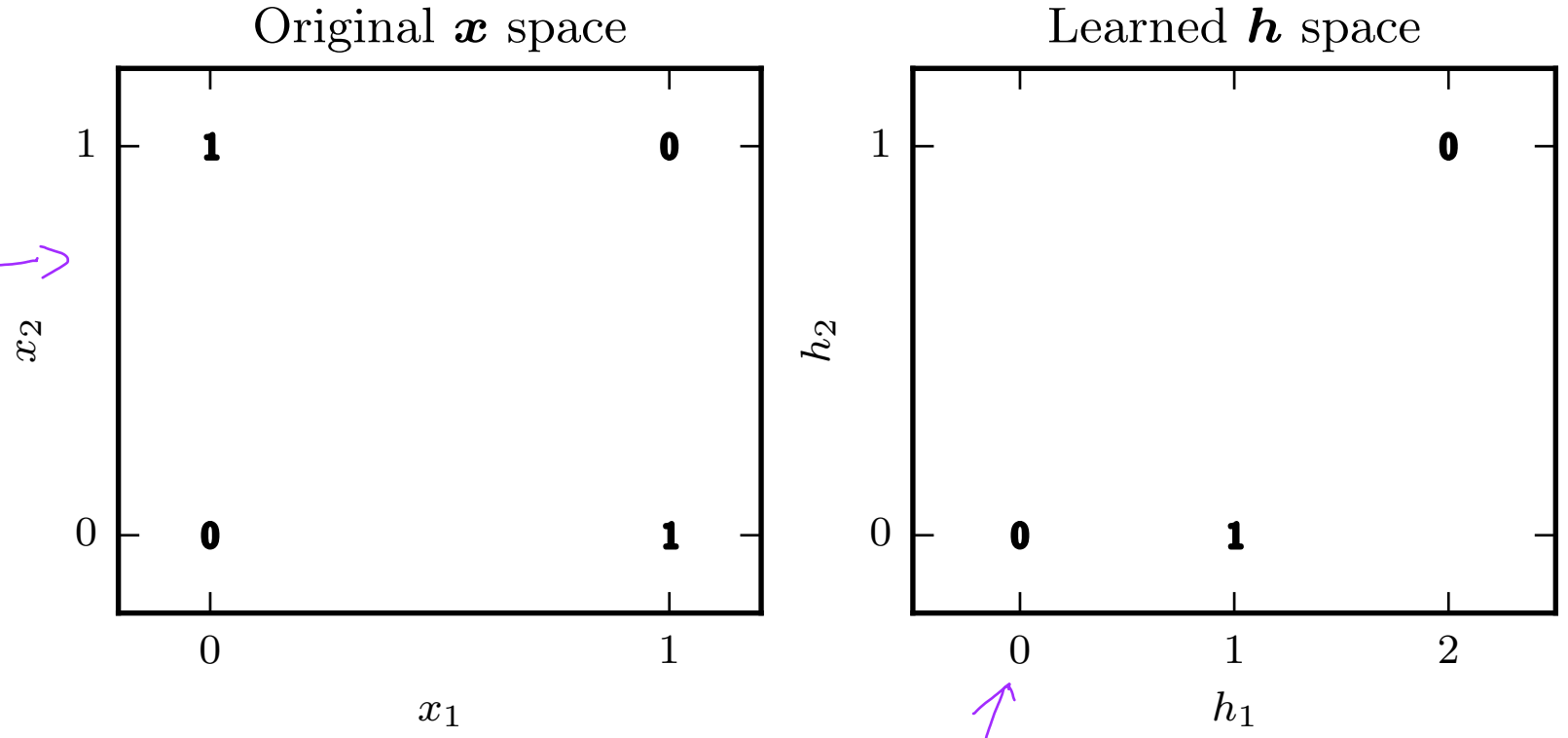
$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad b = 0$$

$$\max\{0,\ XW + c\} \quad = \max\{O,\ \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$\max\{0, -1\}$

$h_1 \quad h_2$

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$



$$\max\{0, XW + c\} = \max\{O, \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}\} = \begin{matrix} h_1 & h_2 \\ \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{matrix}$$

The XOR Multi-Layered Perceptron

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Function we want to learn:

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, XW + \boldsymbol{c}\} + b$$

Oracle solution:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad b = 0$$

$$\mathrm{w}^{\mathrm{T}} \max\{0, WX + c\} + b = [1 \quad -2] \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} + 0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = Y$$

identical

we solve the problem by introduce the non-linearity into the network.

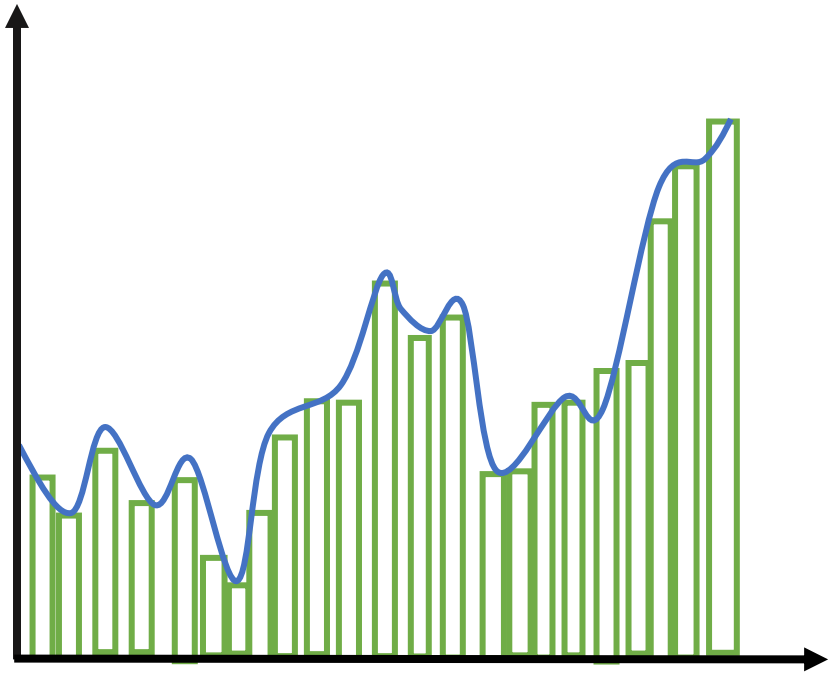
So far: The non-linearity is critically important

Next: What *type* of functions can we approximate?

Universal approximation theorem

Given that $\sigma \in C^{\infty}(\mathbb{R})$ is non-linear (e.g., sigmoid), what type of functions can we learn?

as long as $f(x)$ continuous function



Given enough hidden units.
One layer is enough in theory.
In practice deeper is better.

$$\exists\, g(x)\ as\ NN,\ g(x) \approx f(x),\ and\ |g(x) - f(x)| < \epsilon$$

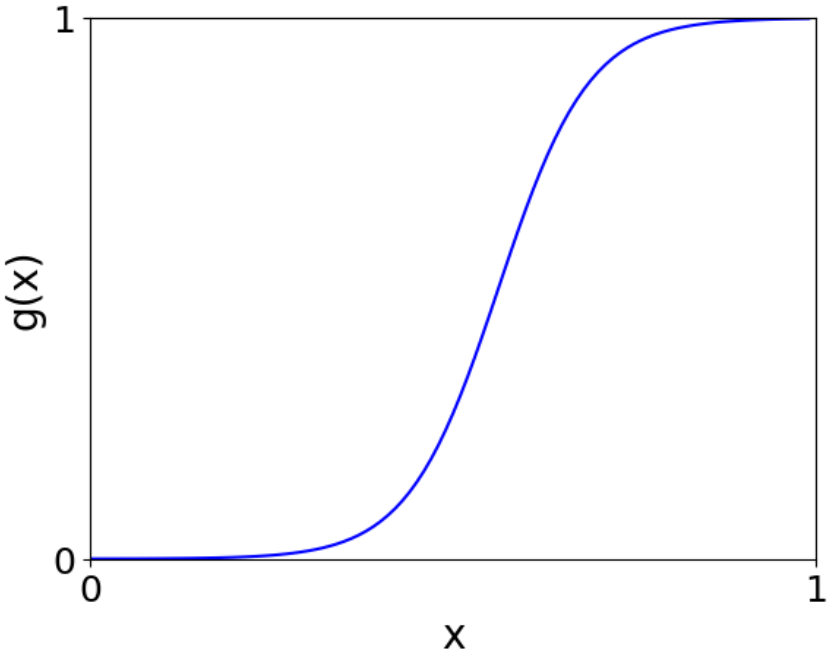
It is an approximation

Original proof: Multilayer feedforward networks are universal approximators

Universal approximation theorem

$$g(x) = \sigma(w^T x + b) = \frac{\exp(w^T x + b)}{\exp(w^T x + b) + 1}$$

Where σ is the sigmoid function



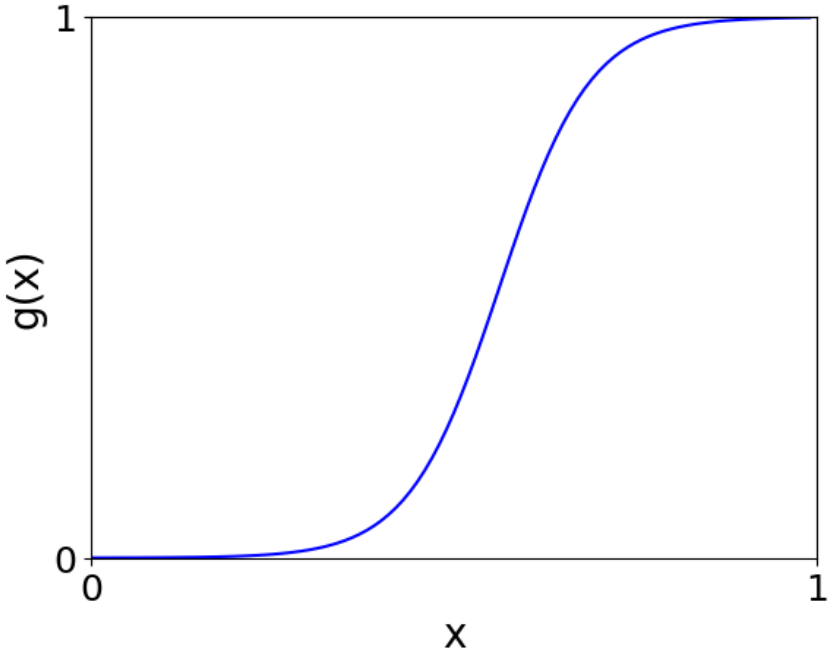
$$w = 16, b = -9$$

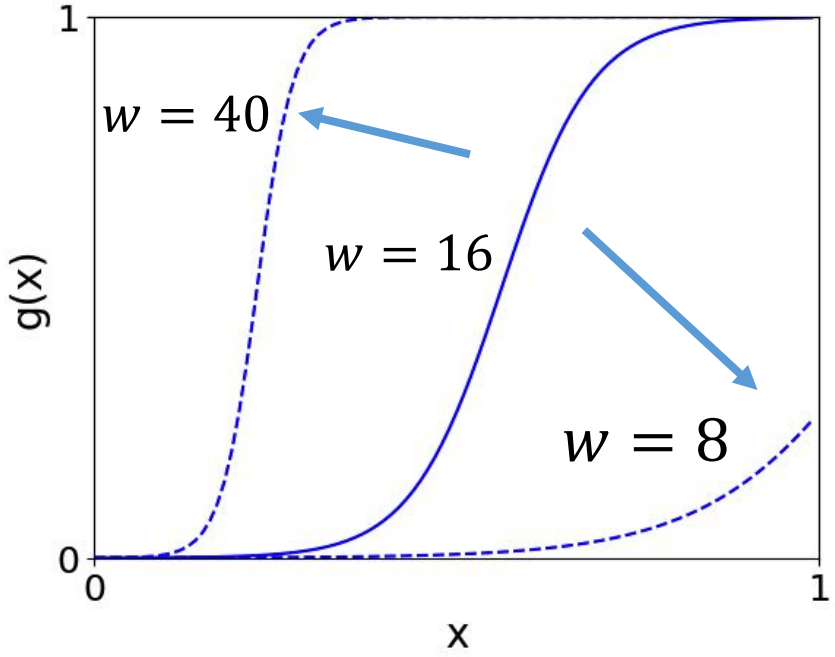
$\sigma = \frac{e^x}{e^x + 1}$

[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

Universal approximation theorem

$$g(x) = \sigma(w^T x + b) = \frac{\exp(w^T x + b)}{\exp(w^T x + b) + 1}$$





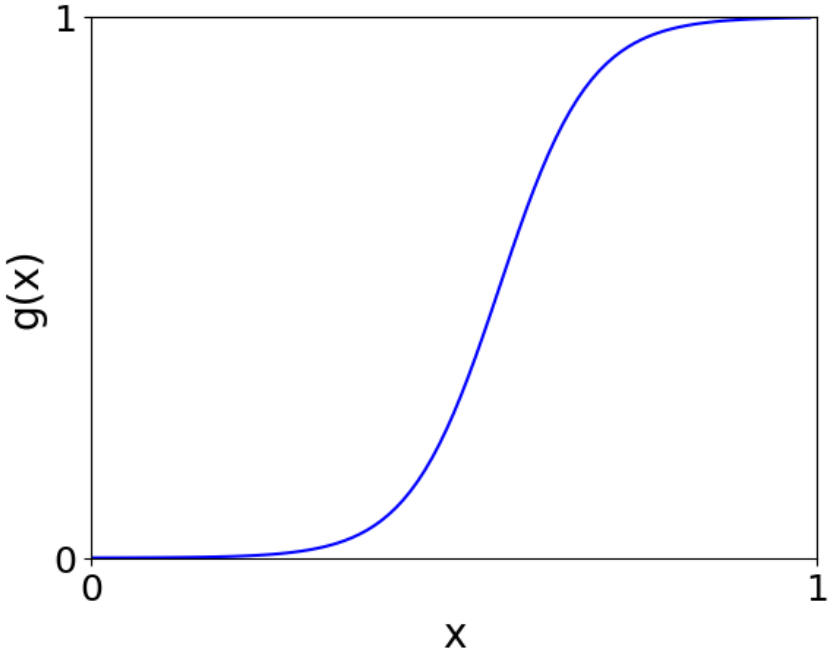
$$w = 16, b = -9$$

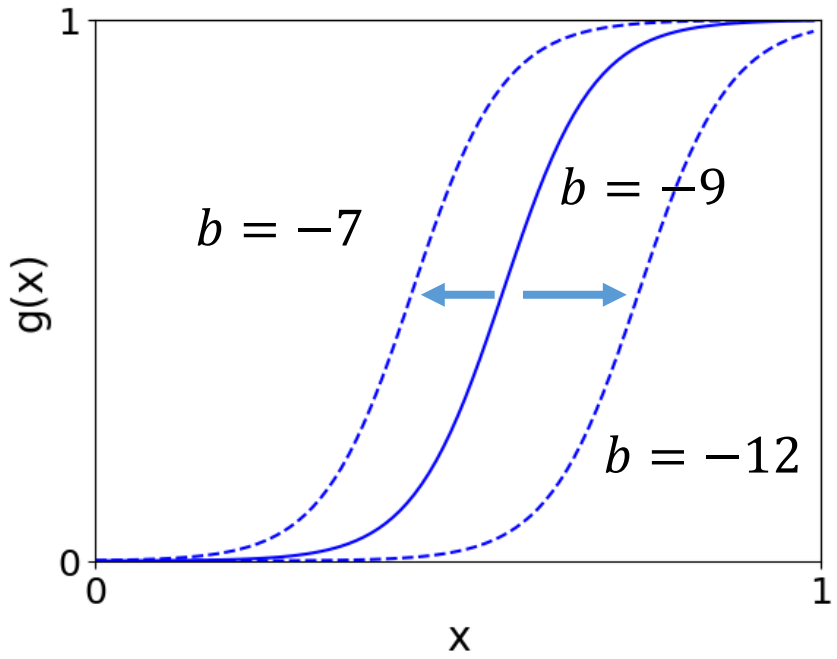
[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

Universal approximation theorem

$$g(x) = \sigma(w^T x + b) = \frac{\exp(w^T x + b)}{\exp(w^T x + b) + 1}$$

Where σ is the sigmoid function



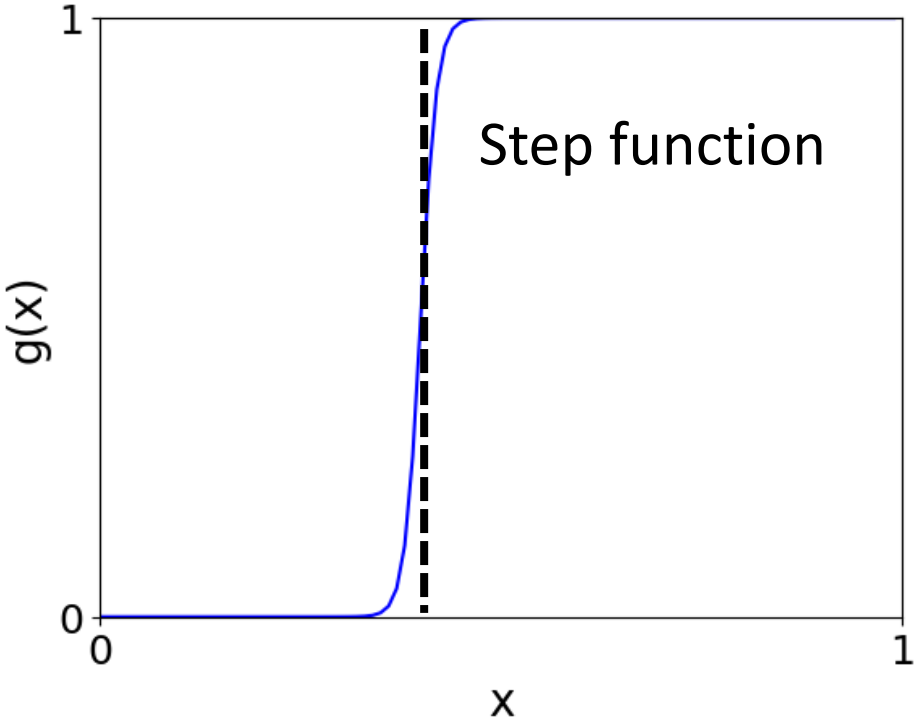


$w = 16, b = -9$

[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

Universal approximation theorem

Leading edge: $s = -\frac{b}{w} = 0.4$

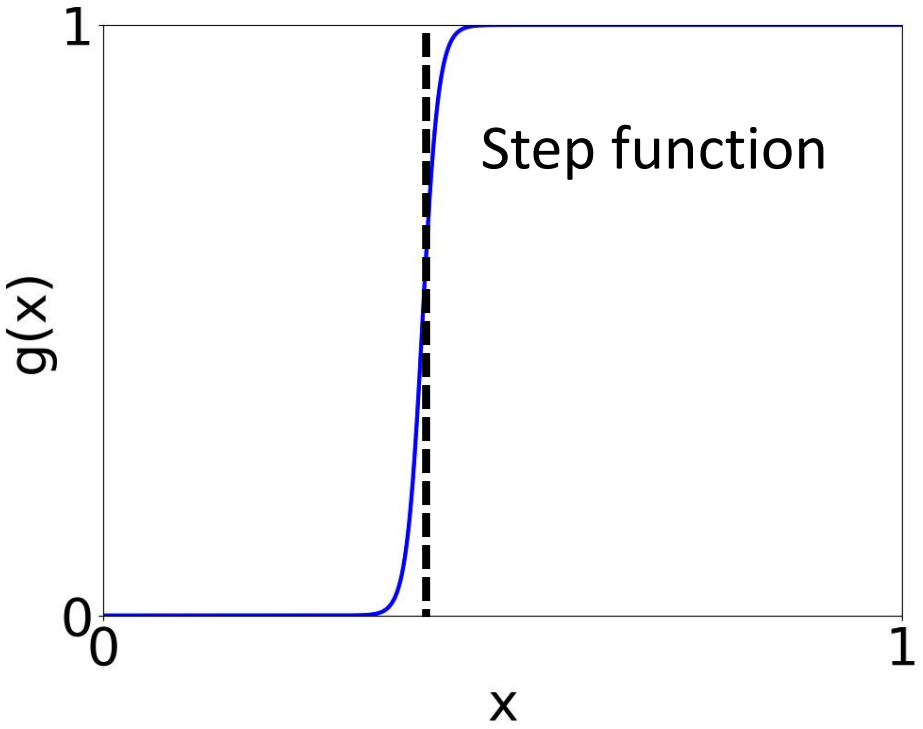


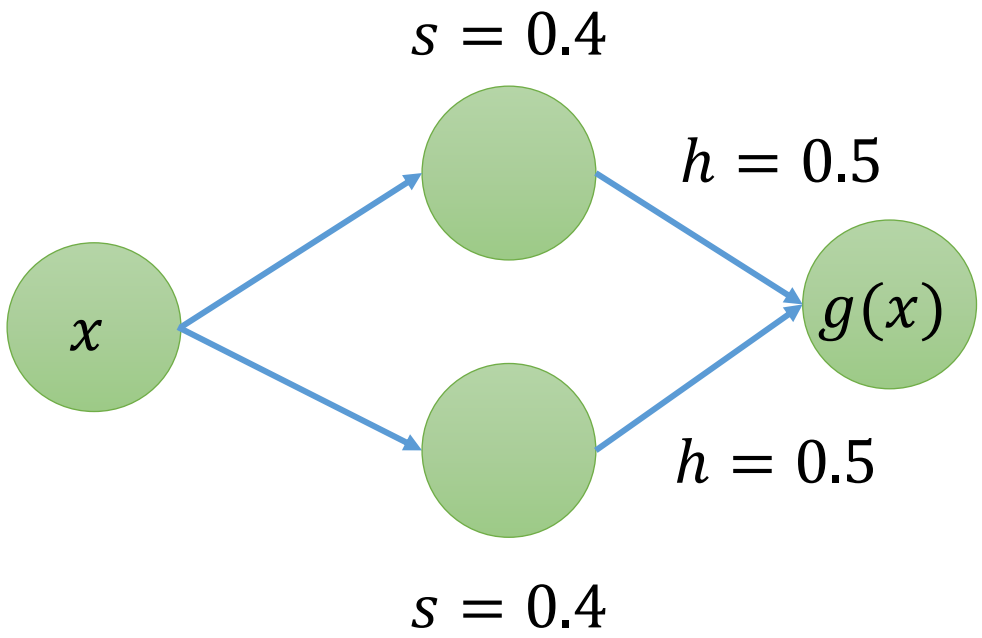
$w = 100, b = -40$

[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

Universal approximation theorem

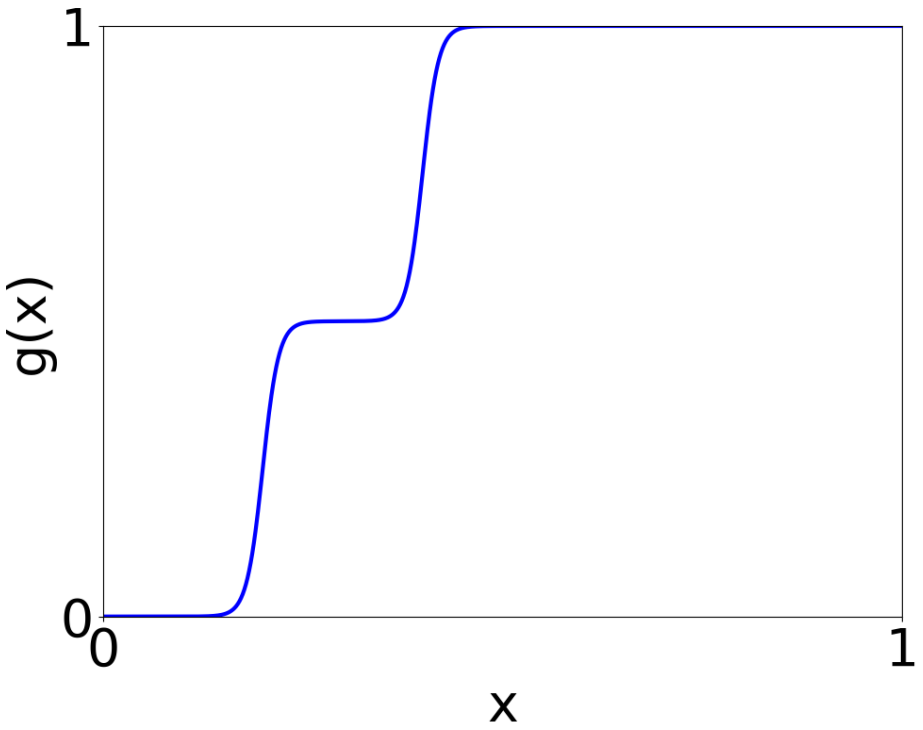
Output layer
is always assumed
linear

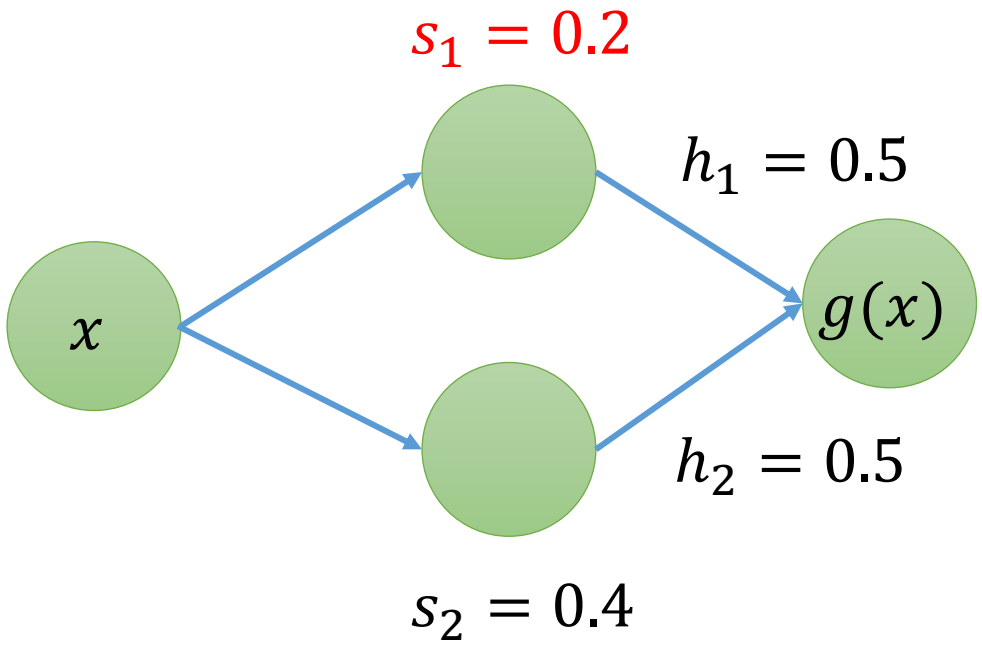




[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

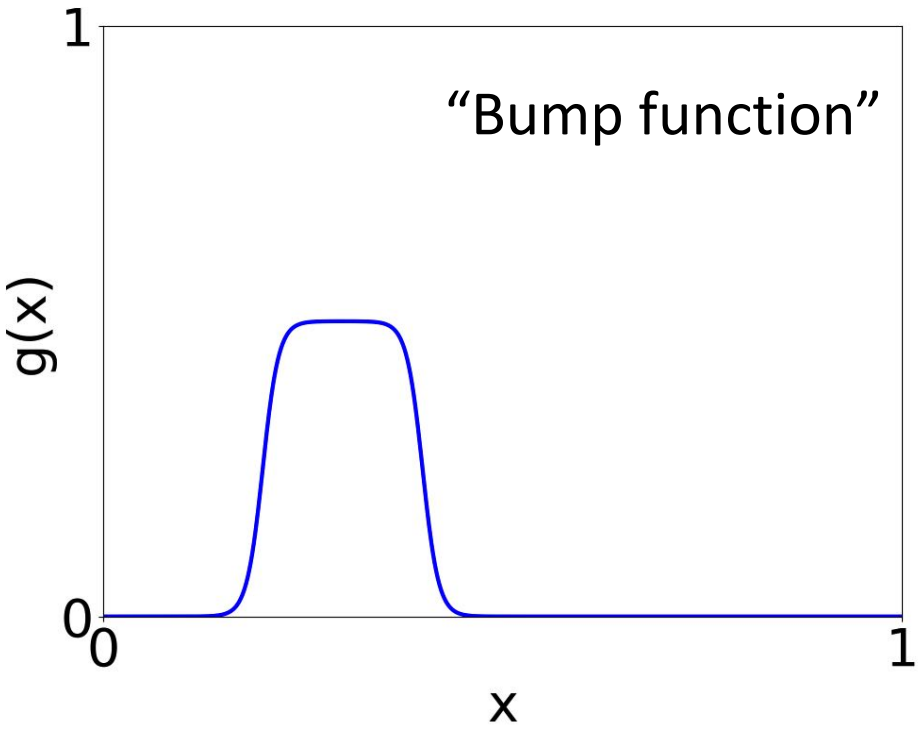
Universal approximation theorem

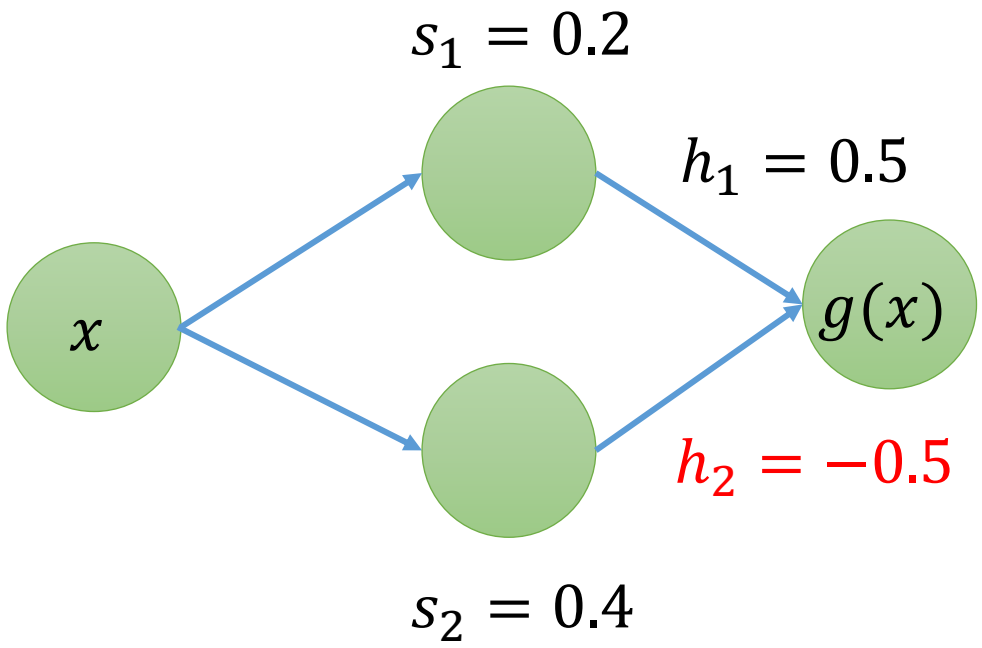




[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

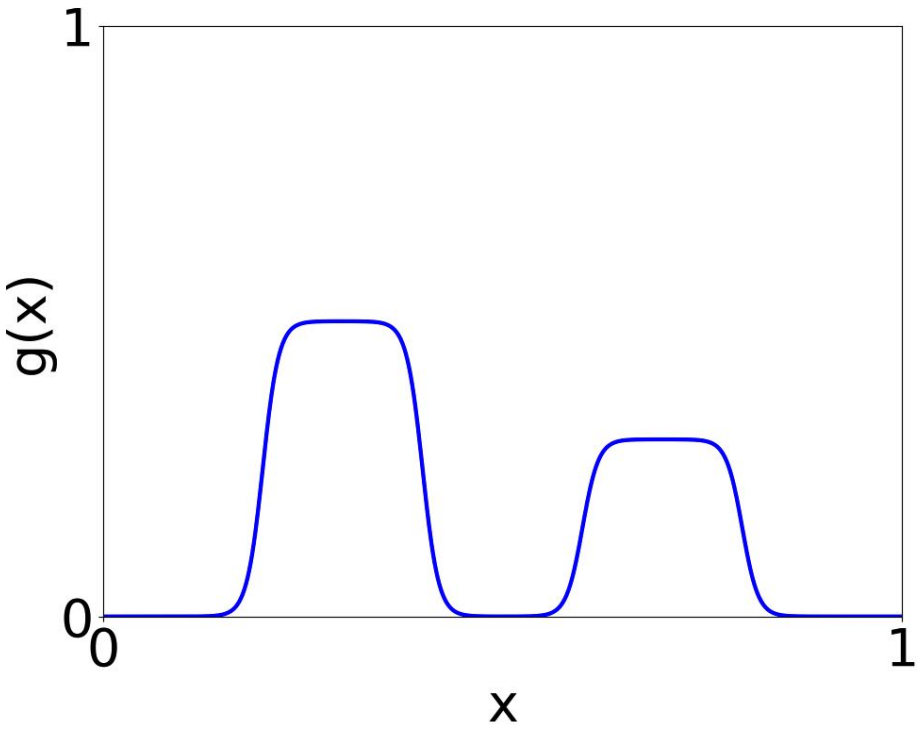
Universal approximation theorem

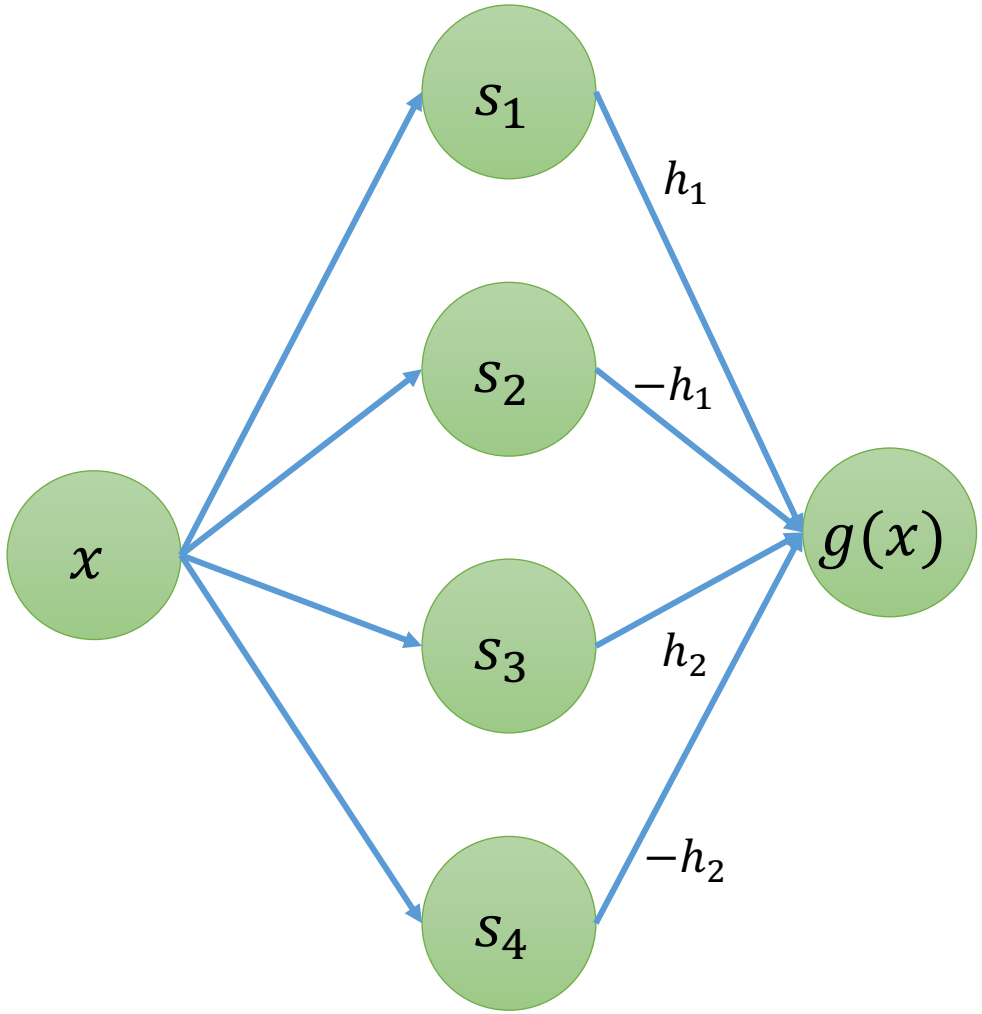




[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

Universal approximation theorem





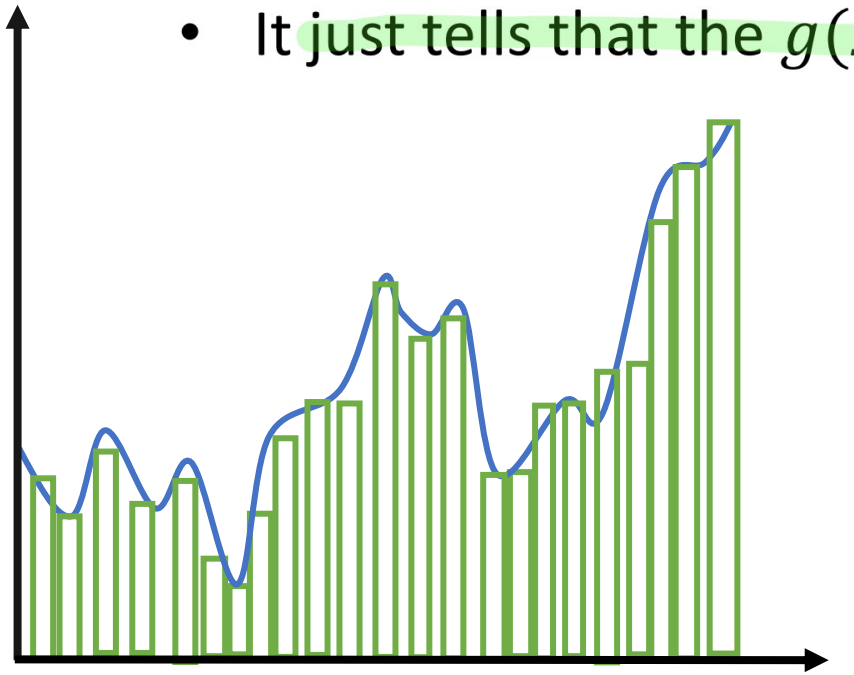
[see also: http://neuralnetworksanddeeplearning.com/chap4.html]

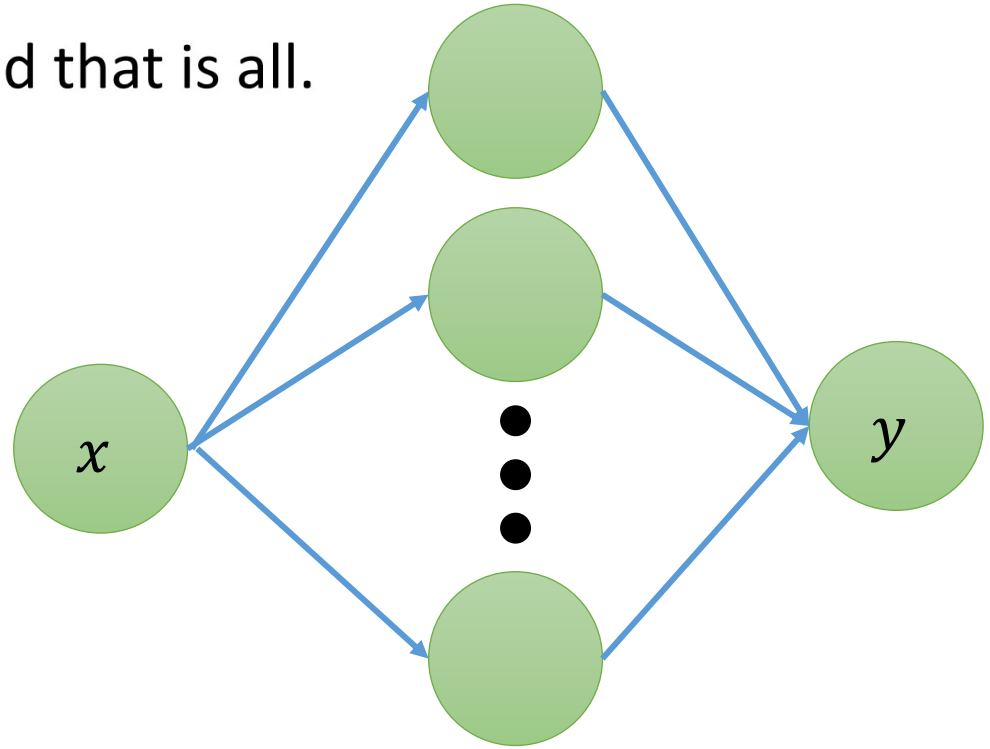
Universal approximation theorem

- Tells us that feed forward networks can approximate $f(x)$
- It *does not* tell us anything about the chances of learning the correct parameters

So as long as f(x) continous.

- It just tells that the $g(x)$ exists, and that is all.





More precisely…

[Cybenko, G. "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 1989.]

Let $\sigma: \mathbb{R} \to \mathbb{R}$ be a non-constant, bounded and continous (activation) function

I_m denotes the m-dimensional unit hypercube $[0,1]^m$ and the space of real-valued functions on I_m is denoted with $C(I_m)$.

Then any function $f \in C(I_m)$ can be approximated given any $\epsilon > 0$, integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \dots, N$:

$$f(x) \approx g(x) = \sum_{i=1}^{N} v_i \sigma(w_i^T x + b_i)$$

And: $|g(x) - f(x)| < \epsilon \quad$, for all $x \in I_m$

However…

Networks with a single-hidden layer, need to have exponential width

In practice, deeper networks work better.

Lot's of ongoing work to provide theory on what network properties lead to which approximation capabilities.

E.g., [Lu et al. "The Expressive Power of Neural Networks: A View from the Width". NeurIPS 2017]

So far: NNs are universal approximators

Next: How do we find the network parameters?

2019 notes

- Previous: What types of functions can be approximated by neural networks? any kind of f(x) with continuity
- Next: How do we train neural networks?

General procedure

Iterative gradient descent to find parameters Θ:

Initialize weights with small random values

Initialize biases with 0 or small positive values

Compute gradients

Update parameters with SGD

from experience

SGD in a nutshell:

Compute the negative gradient at θ^0

$\rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\rightarrow -\eta \nabla C(\theta^0)$

The nonlinearity of a neural network causes most interesting loss functions to become non-convex

Chain rule

$$y = g(x) \text{ and } z = f\big(g(x)\big) = f(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

For vector types

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial \mathrm{x}}$$

Two ways to compute gradients

Consider the scalar function:

$$f = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$$

Symbolic differentiation gives us:

$$\frac{df}{dx} =$$

$$\exp(\exp(x) + \exp(x)^2)\,(\exp(x) + 2\exp(x)^2) + \cos(\exp(x) + \exp(x)^2)\,(\exp(x) + 2\exp(x)^2)$$

$[\exp(x)^2]' = 2\exp(x) \cdot \exp(x) = 2\exp(x)^2$

If we were to write a program…

Consider the scalar function:

$$f = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$$

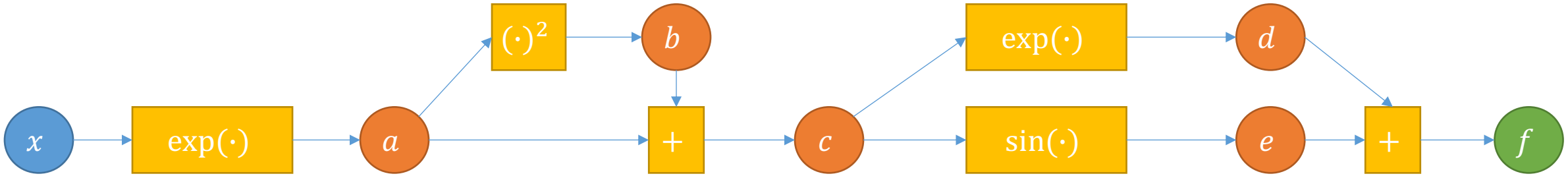
We would define and compute intermediate variables:

$$a = \exp(x)$$
$$b = a^2$$
$$c = a + b$$
$$d = \exp(c)$$
$$e = \sin(c)$$
$$f = d + e.$$

We can draw this as graph

$$f = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$$

$$a = \exp(x)$$
$$b = a^2$$
$$c = a + b$$
$$d = \exp(c)$$
$$e = \sin(c)$$
$$f = d + e.$$



Mechanically writing down derivatives

$$\frac{df}{dd} = 1$$

$$\frac{df}{de} = 1$$

$$\frac{df}{dc} = \frac{df}{dd}\frac{dd}{dc} + \frac{df}{de}\frac{de}{dc}$$

$$\frac{df}{db} = \frac{df}{dc}\frac{dc}{db}$$

$$\frac{df}{da} = \frac{df}{dc}\frac{dc}{da} + \frac{df}{db}\frac{db}{da}$$

$$\frac{df}{dx} = \frac{df}{da}\frac{da}{dx}$$

$$\frac{df}{dd} = 1$$

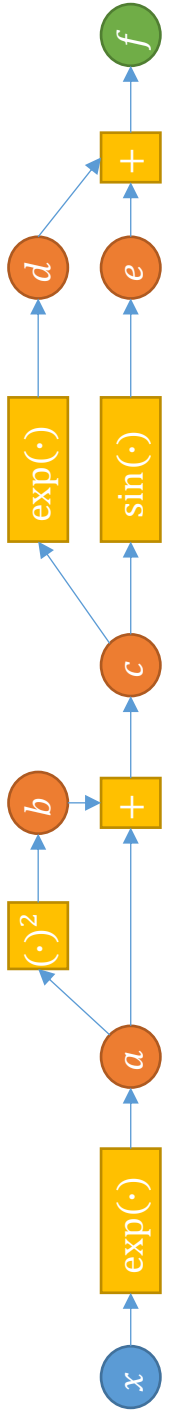
$$\frac{df}{de} = 1$$

$$\frac{df}{dc} = \frac{df}{dd}\exp(c) + \frac{df}{de}\cos(c)$$

$$\frac{df}{db} = \frac{df}{dc}$$

$$\frac{df}{da} = \frac{df}{dc} + \frac{df}{db}2a$$

$$\frac{df}{dx} = \frac{df}{da}\exp(x)$$



Backpropagation in Neural Networks

Example & Derivation on the blackboard

① Backpropagation – single unit

$$* f^{(3)} = \|z^{(2)} - y\|^2 = z^{(2)T} z^{(2)} - 2 z^{(2)} y + y^T y$$

$$\frac{\partial z^{(3)}}{\partial z^{(2)}}^{*} = 2 z^{(2)} - 2y = 2(z^{(2)} - y)$$

$$\frac{\partial z^{(2)}}{\partial z^{(1)}} = \sigma(z^{(1)})(1 - \sigma(z^{(1)}))$$

$$\frac{\partial z^{(1)}}{\partial \theta^{(1)}} = z^{(0)} = x$$

$z^{(3)} = C(\theta)$

$f^{(3)} = \|z^{(2)} - y\|^2$

$z^{(2)}$

$f^{(2)} = \sigma(z^{(1)})$

$z^{(1)}$

$f^{(1)} = \sum \theta^{(1)} z^{(0)}$

$z^{(0)} = x$

$$\frac{\partial C}{\partial C} = \frac{\partial z^{(3)}}{\partial z^{(3)}} = 1$$

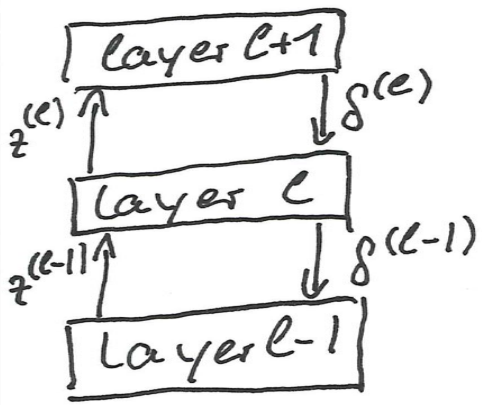
$$\frac{\partial z^{(3)}}{\partial z^{(2)}} \cdot 1$$

$$\frac{\partial z^{(2)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(3)}}{\partial z^{(2)}} \cdot 1$$

$$\frac{\partial z^{(1)}}{\partial \theta^{(1)}} \cdot \frac{\partial z^{(2)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(3)}}{\partial z^{(2)}} \cdot 1 = \frac{\partial C}{\partial \theta} = \frac{\partial z^{(3)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial \theta^{(1)}}$$

$$= 2(z^{(2)} - y)\sigma(z^{(1)})(1 - \sigma(z^{(1)}))x$$

(II) BP layer wise

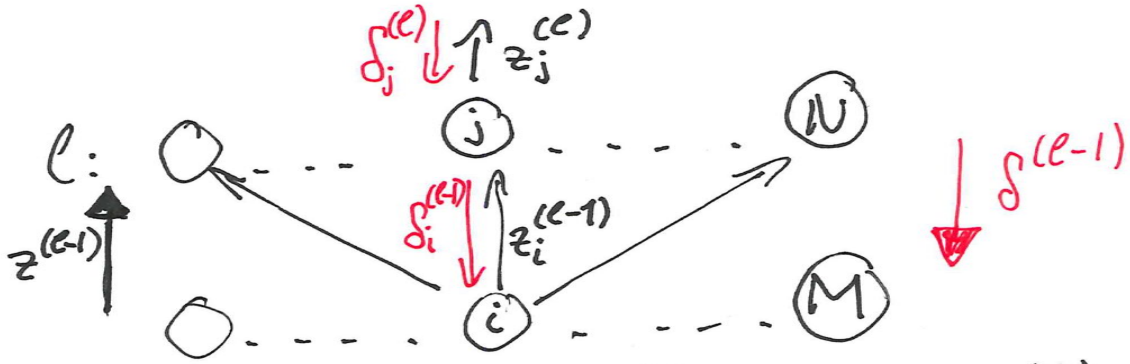


$\delta^{(\ell-1)}$: Gradient that flows from layer (ℓ) to $(\ell-1)$

$\delta_i^{(\ell-1)}$: Gradient wrt to the i^{th} unit in the $(\ell-1)^{th}$-layer

Example: "Layer 2 receives inputs $z^{(1)}$ and passes down gradients $\delta^{(1)}$".

(IIa)



$$\delta_i^{(\ell-1)} = \frac{\partial C}{\partial z_i^{(\ell-1)}} = \sum_{j=1}^{N} \underbrace{\frac{\partial C}{\partial z_j^{(\ell)}}}_{\delta_j^{(\ell)}} \frac{\partial z_j^{(\ell)}}{\partial z_i^{(\ell-1)}}$$

(IIb)

$$\delta^{(\ell-1)} = \sum_{j=1}^{N} \delta_j^{(\ell)} \frac{\partial z_j^{(\ell)}}{\partial z^{(\ell-1)}}$$

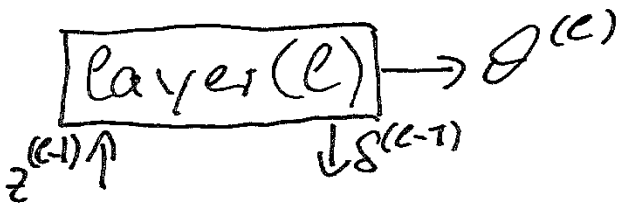
$$= \underbrace{\frac{\partial C}{\partial z^{(\ell)}}}_{\in \mathbb{R}^{1\times n}} \underbrace{\frac{\partial z^{(\ell)}}{\partial z^{(\ell-1)}}}_{\in \mathbb{R}^{n\times m}}$$

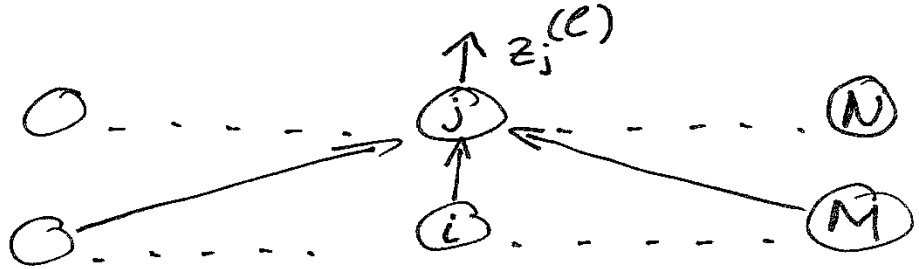
$$z_j^{(\ell)} = \sigma(\theta_j^{(\ell)} z^{(\ell-1)}) \qquad \langle 1 \rangle \quad \langle 1, M \rangle \langle M, 1 \rangle$$

$$z^{(\ell)} = \sigma(\theta^{(\ell)} z^{(\ell-1)}) \qquad \langle N, 1 \rangle \quad \langle N, M \rangle \langle M, 1 \rangle$$

$$\sigma\left(\begin{pmatrix} \theta_1, \theta_2, \dots \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ \vdots \end{pmatrix} \right)$$

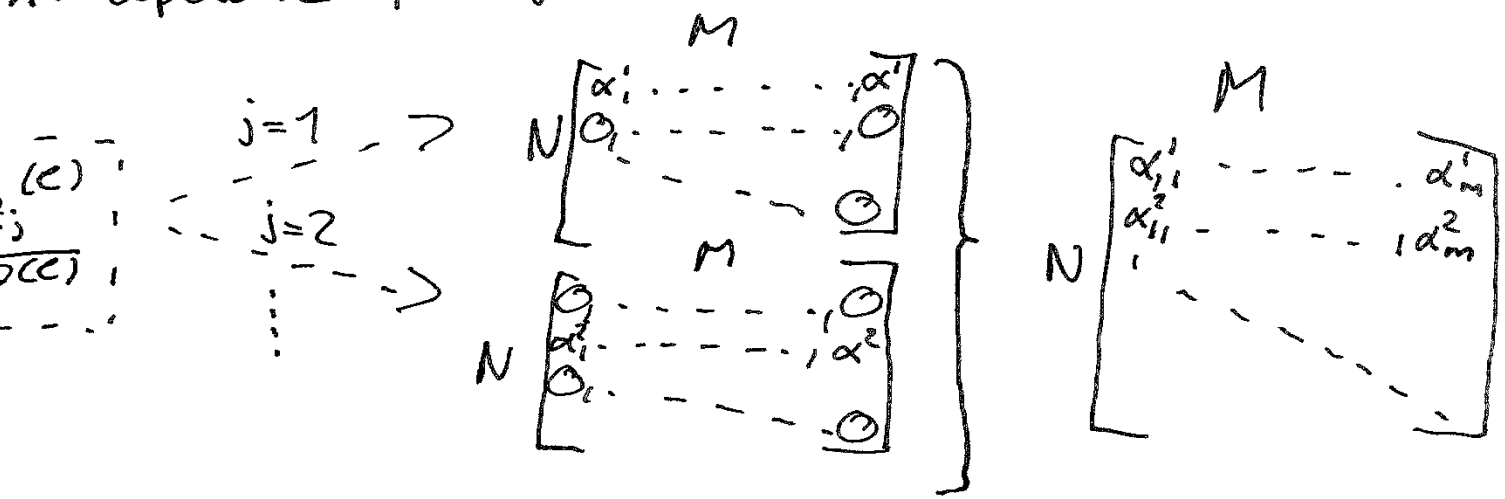
(III) BP - parameter update





$\frac{\partial C}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial \theta_j^{(l)}}$ } weight update for j^{th} unit

$$\frac{\partial C}{\partial \theta^{(l)}} = \sum_{j=1}^{N} \frac{\partial C}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial \theta^{(l)}}$$



$$= \sum_{j=1}^{N} \delta_j^{(l)} \frac{\partial z_j^{(l)}}{\partial \theta^{(l)}}$$

$$= \frac{\partial C}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial \theta^{(l)}}$$

Next week

Convolutional Neural Networks