Foundations of Machine Learning (ECE 5984)

- Neural Networks -

Eunbyung Park

Assistant Professor

School of Electronic and Electrical Engineering

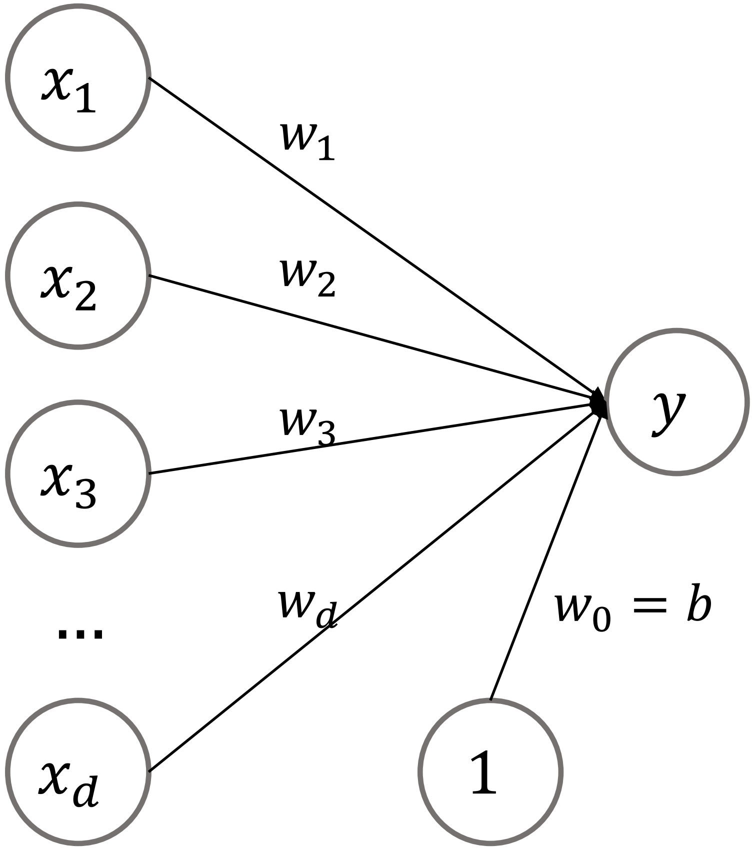
Eunbyung Park (silverbottlep.github.io)

성균관대학교
SUNGKYUNKWAN UNIVERSITY

Multi-Layer Perceptron

Linear Models as Shallow Neural Networks

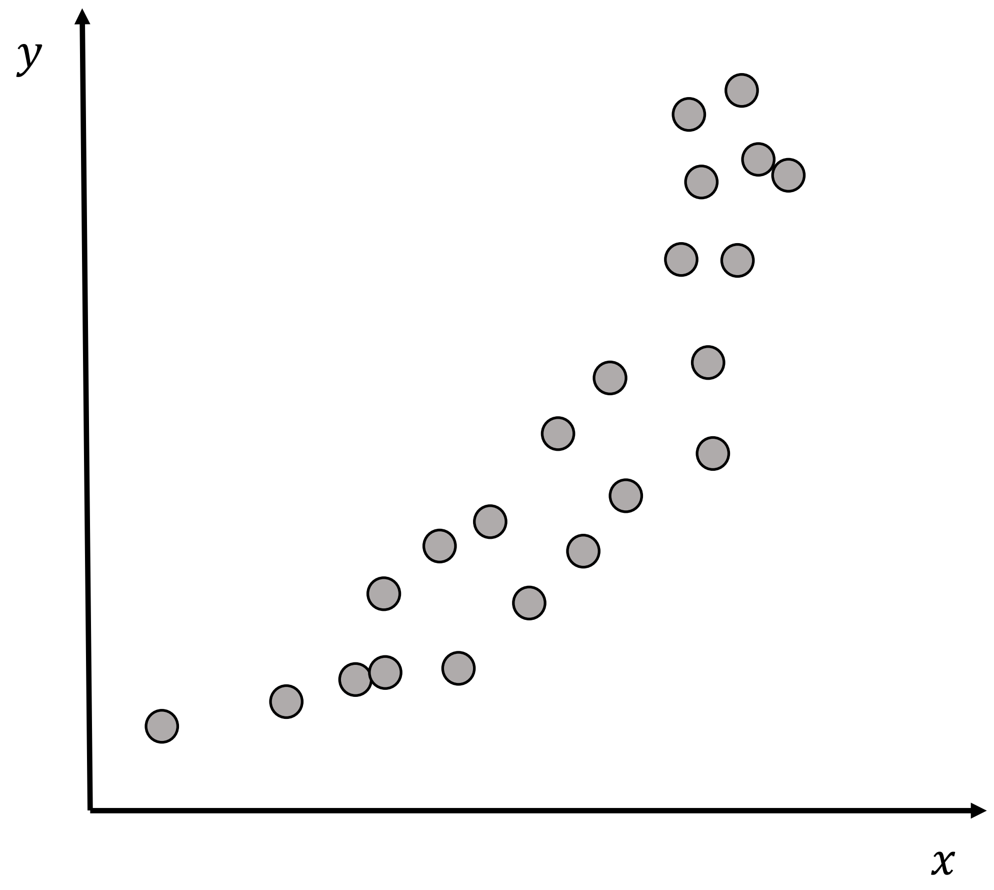
- It is a single layer neural network

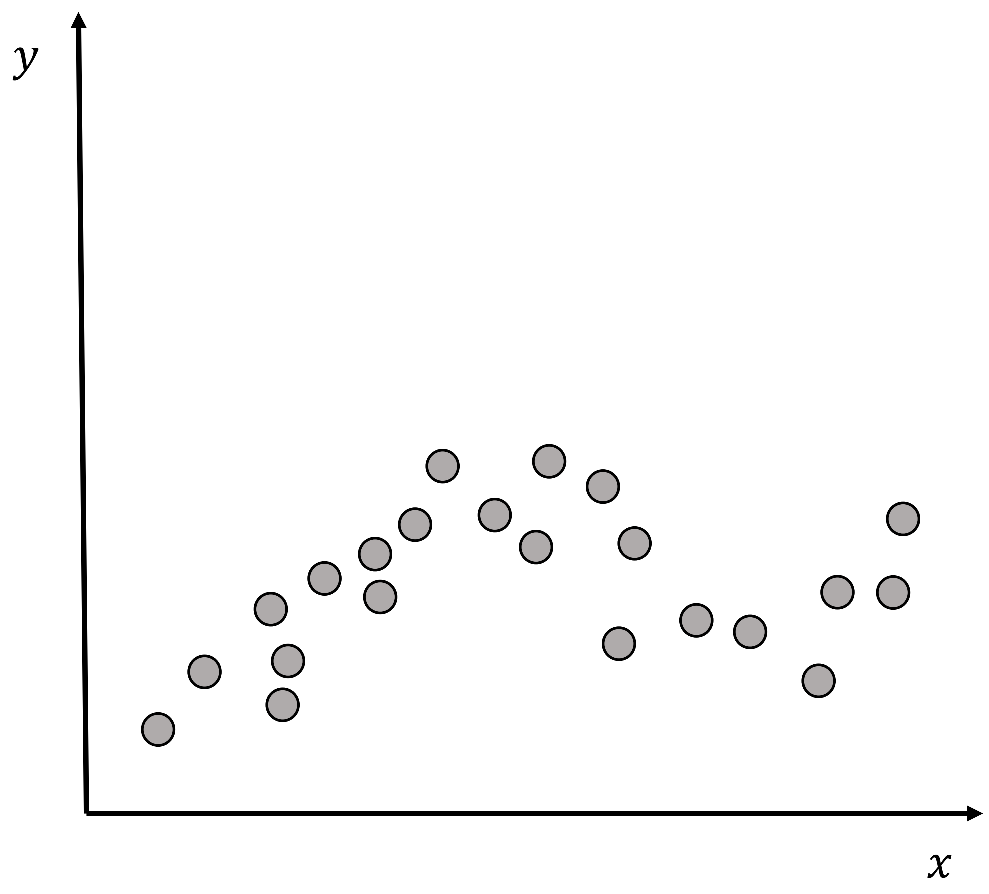


$$y = w^\top x$$

Linear Models

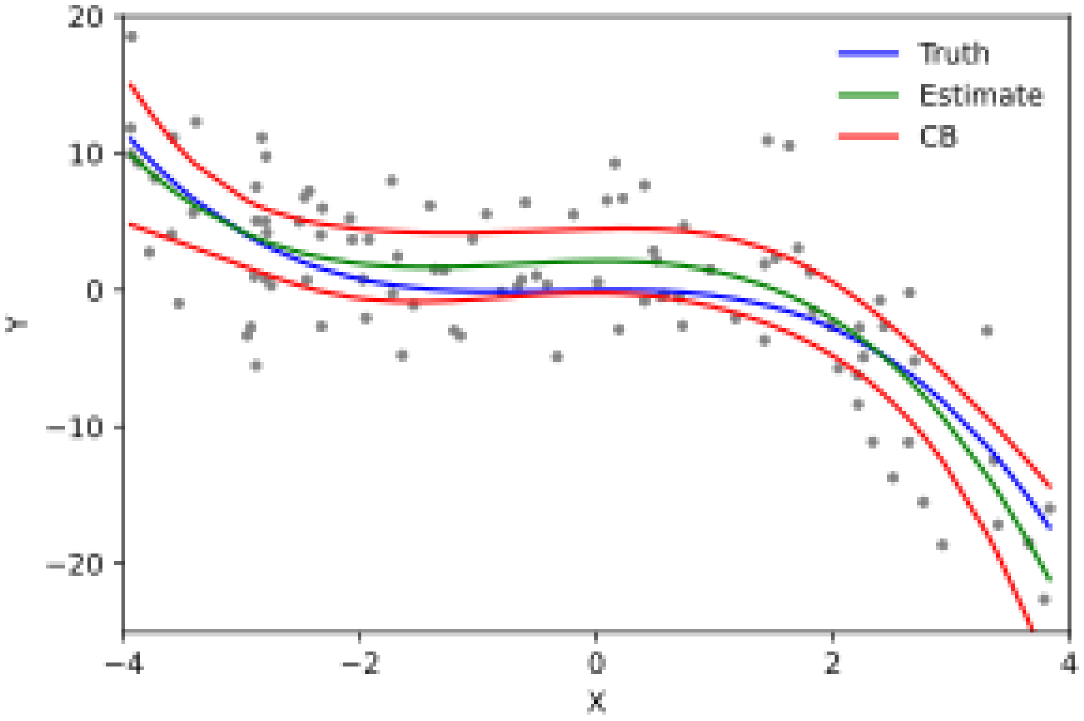
- Is linear model a good for all?





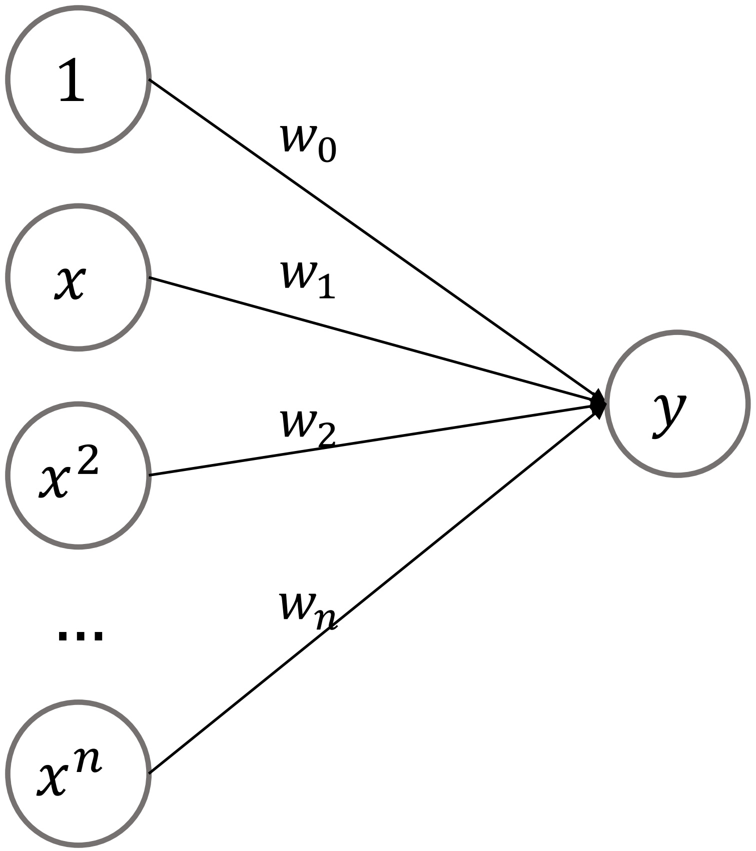
Nonlinear Models

- nth-degree Polynomial regression



$$f(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_n x^n$$

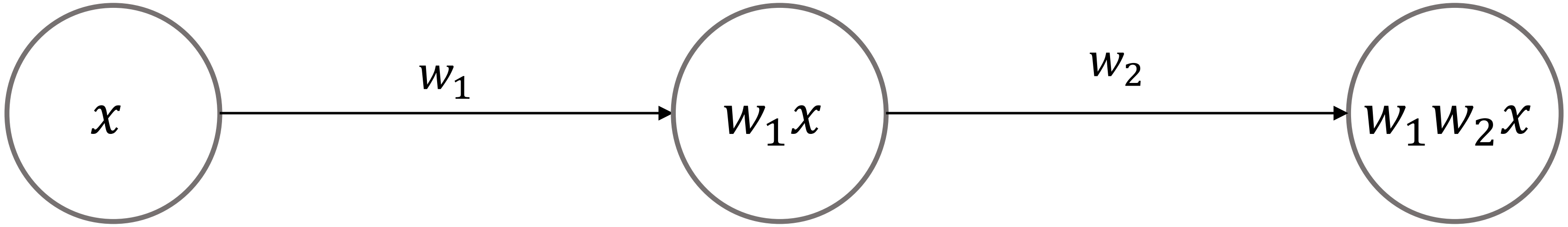
Polynomals as Neural Network



$$f(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_n x^n$$

- Feature engineering is hard
- Can we make it non-linear w/o feature engineering?

Feed-Forward Neural Network

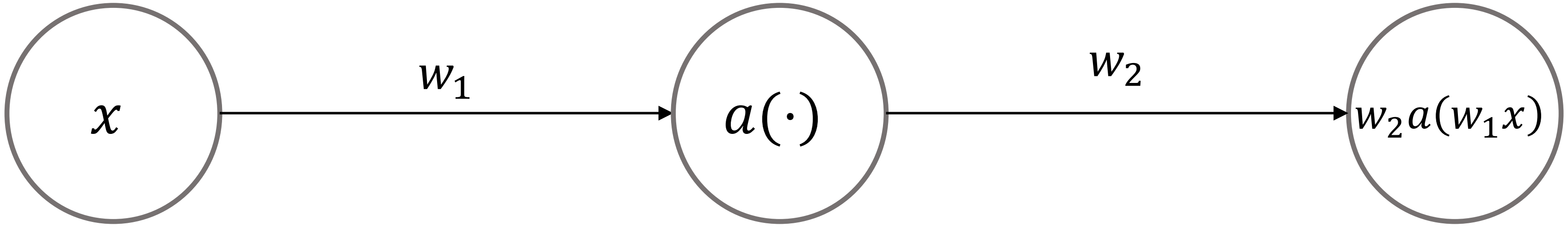


$$f(x) = w_1 w_2 x$$

Is it non-linear in x?

Feed-Forward Neural Network

- Using non-linear activation function



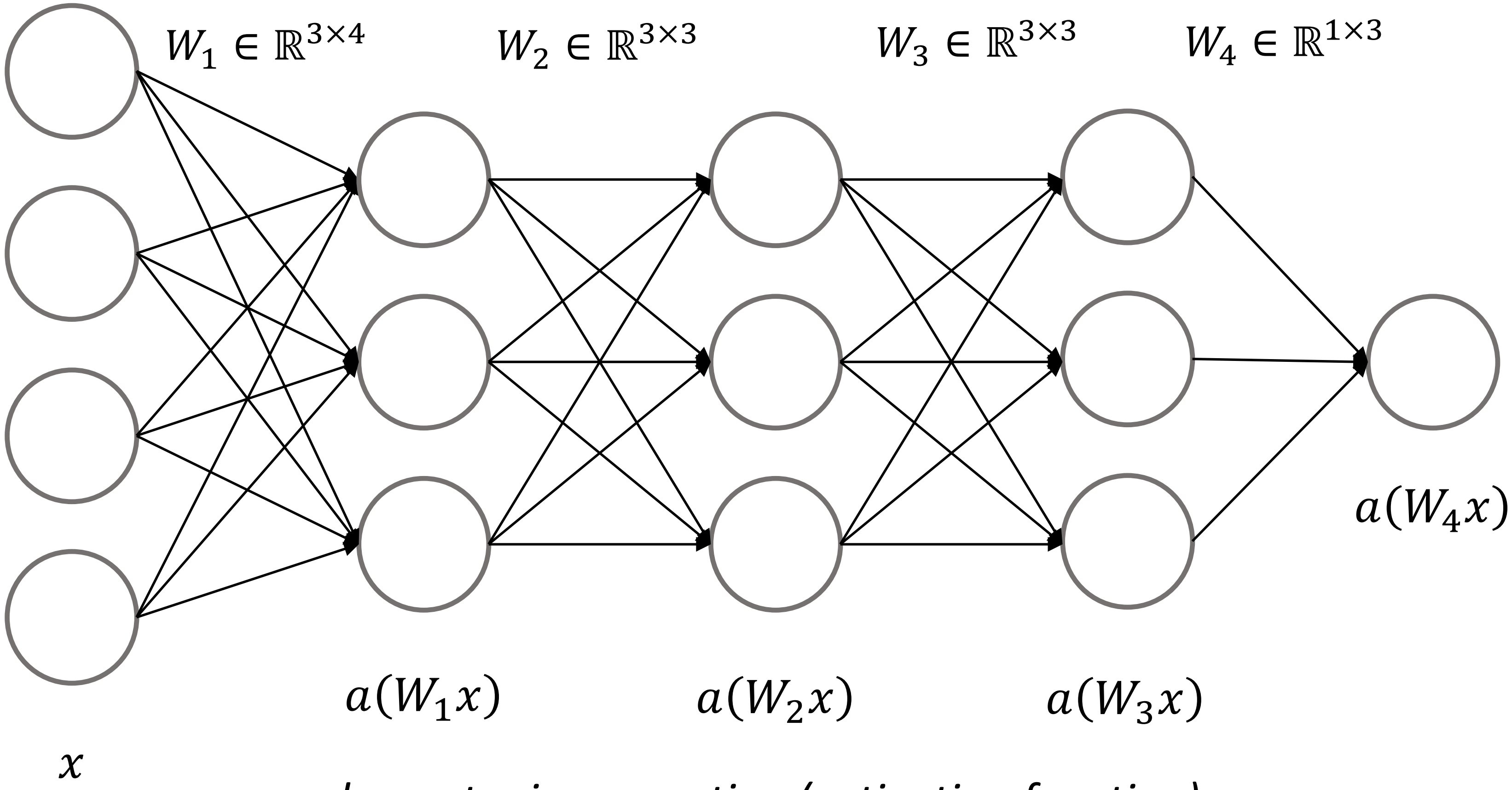
$$f(x) = w_2 a(w_1 x)$$

$$a(x) = \max(0, x)$$ (Rectifier Linear Unit)

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

Feed-Forward Neural Network

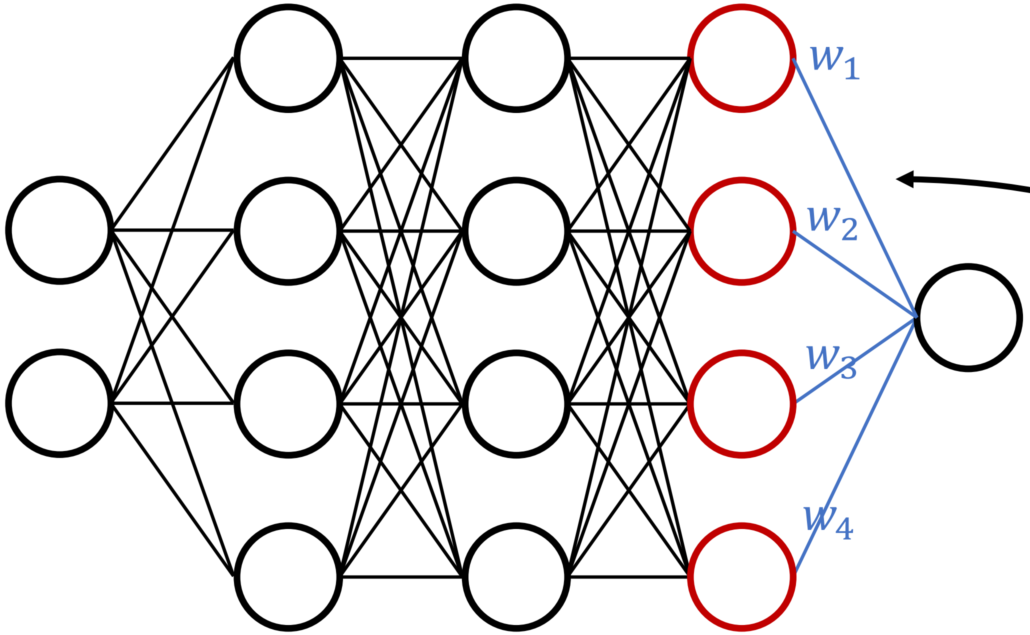
- AKA, Multi-Layer Perceptron



a: element-wise operation (activation function)

Feed-Forward Neural Network

- Learning feature representations

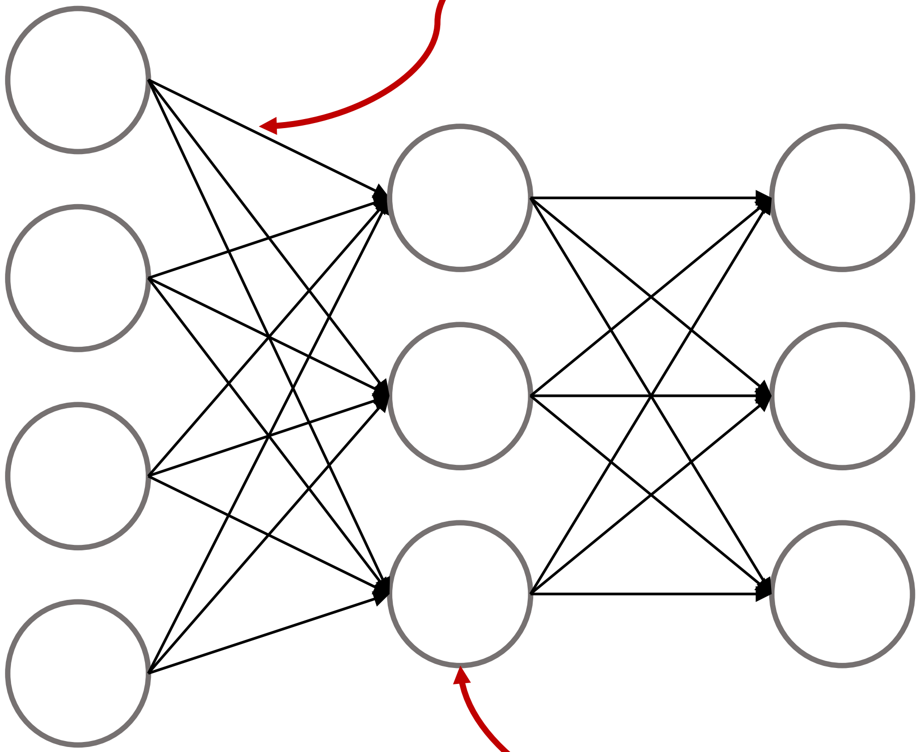


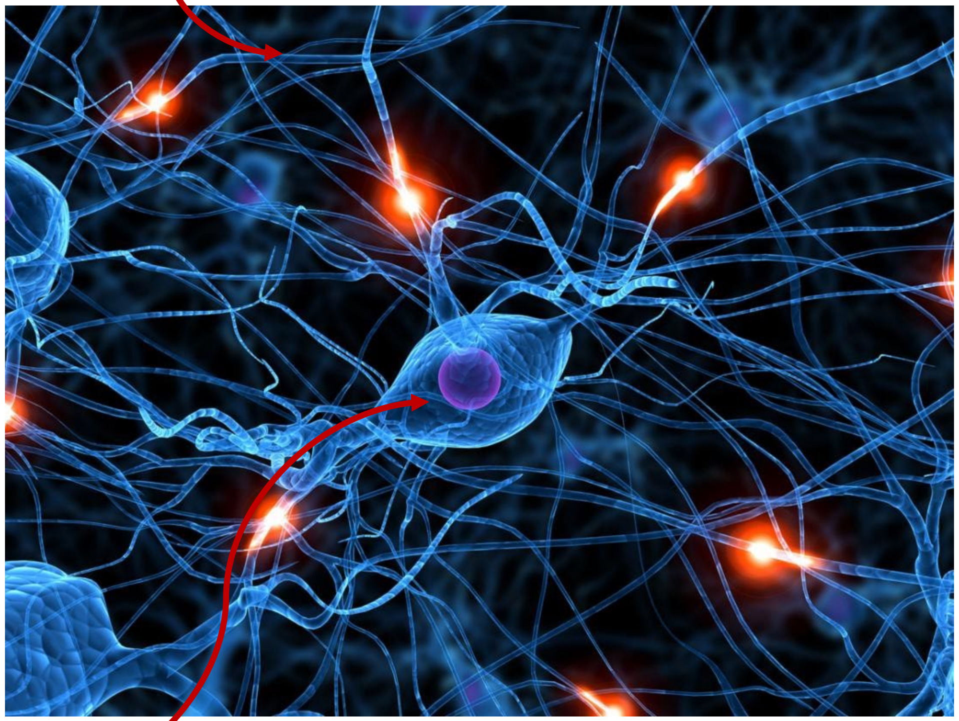
$$\sum_{j=1}^{d} w_j \phi_j(x)$$

Feed-Forward Neural Network

- Biology Inspired

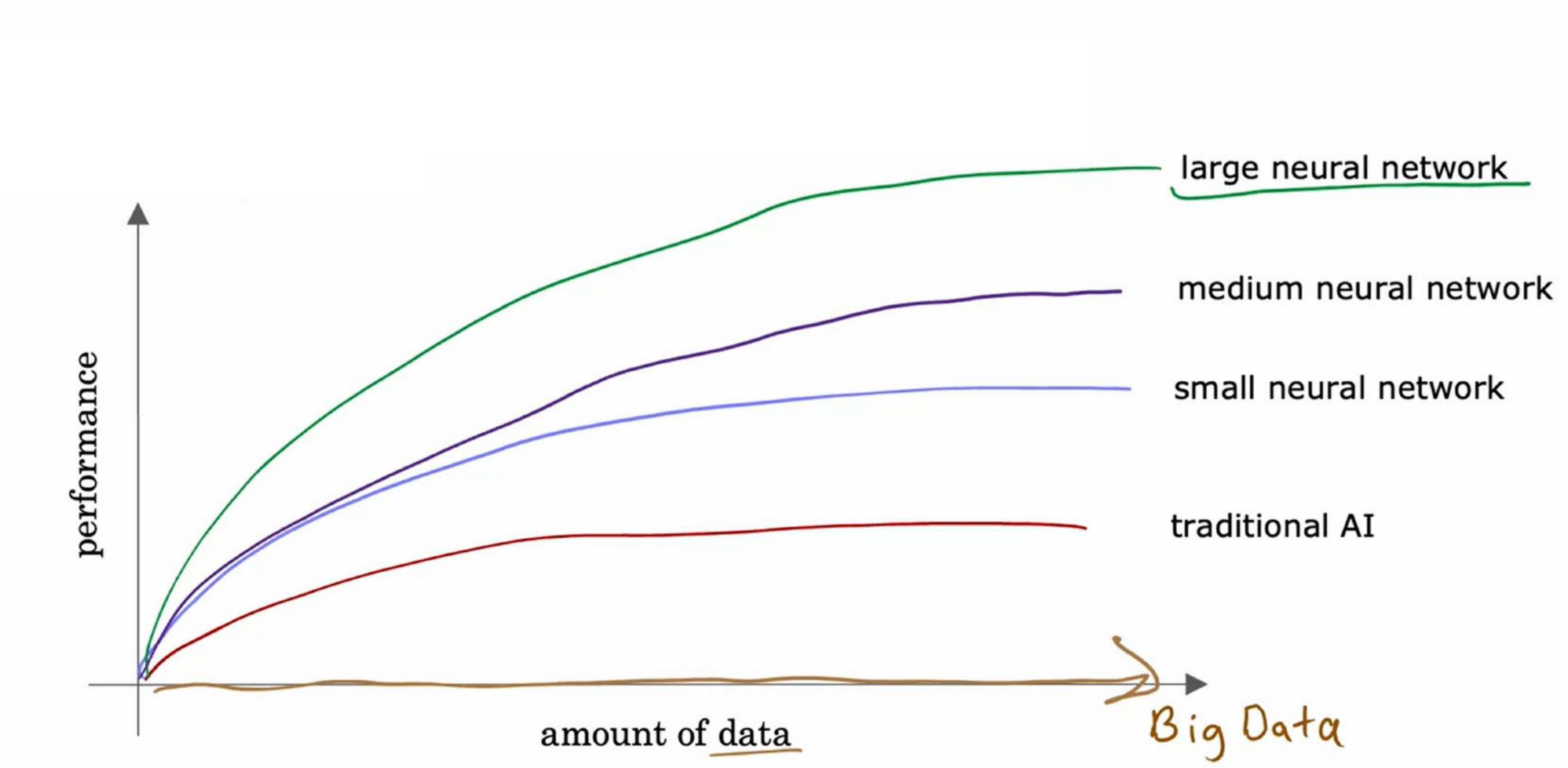
Weights ≈ Axons





Units ≈ Neurons

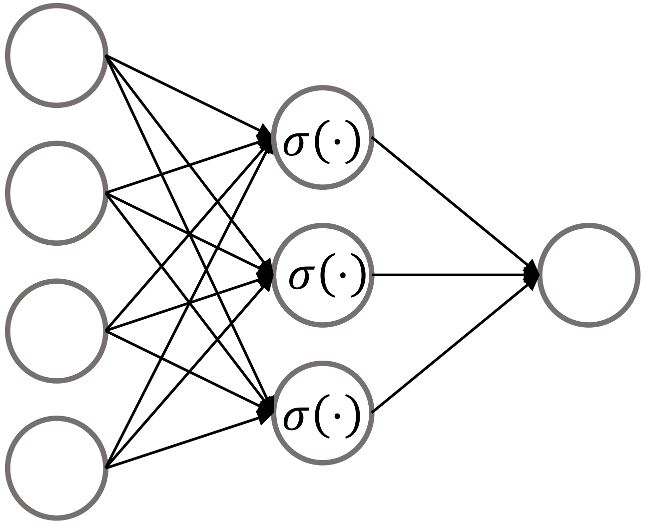
Scaling Laws



[Machine Learning Specialization [3 courses] (Stanford) | Coursera]

Feed-Forward Neural Network

- Regression with two layers MLP



$$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$$

$$x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R}, X \in \mathbb{R}^{N\times d}, Y \in \mathbb{R}^N$$

$$\theta = \{W_1, W_2\}, W_1 \in \mathbb{R}^{h\times d}, W_2 \in \mathbb{R}^{1\times h}$$

$$f_\theta(x) = W_2\sigma(W_1 x)$$

$$f_\theta : \mathbb{R}^d \to \mathbb{R}$$

$$L(\theta) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - f_\theta(x^{(i)})\right)^2 = \frac{1}{2}(Y - \sigma(W_1X^\top)^\top W_2^\top)^\top(Y - \sigma(W_1X^\top)^\top W_2^\top)$$

Feed-Forward Neural Network

- Regression with two layers MLP

$$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$$
$$x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R}, X \in \mathbb{R}^{N\times d}, Y \in \mathbb{R}^N$$
$$\theta = \{W_1, W_2\}, W_1 \in \mathbb{R}^{h\times d}, W_2 \in \mathbb{R}^{1\times h}$$
$$f_\theta(x) = W_2\sigma(W_1 x)$$
$$f_\theta : \mathbb{R}^d \to \mathbb{R}$$

1. Can you take the gradients w.r.t θ?
2. Does it have a closed form solution?
3. Is it a convex function?

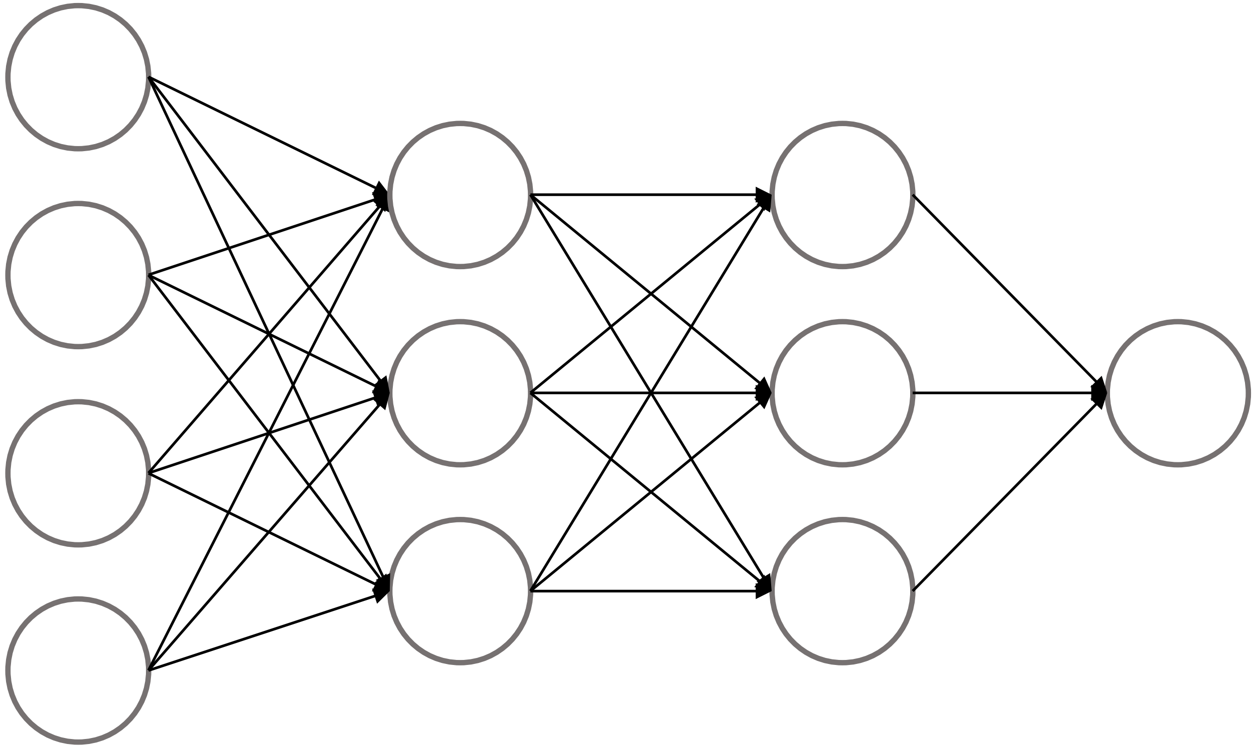
$$L(W_1, W_2) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - f_\theta(x^{(i)})\right)^2 = \frac{1}{2}(Y - \sigma(W_1 X^\top)^\top W_2^\top)^\top (Y - \sigma(W_1 X^\top)^\top W_2^\top)$$

Gradient Descent

- We are using *gradient descent* for training deep neural networks

$$W := W - \alpha\left(\frac{\partial L}{\partial W}\right)$$

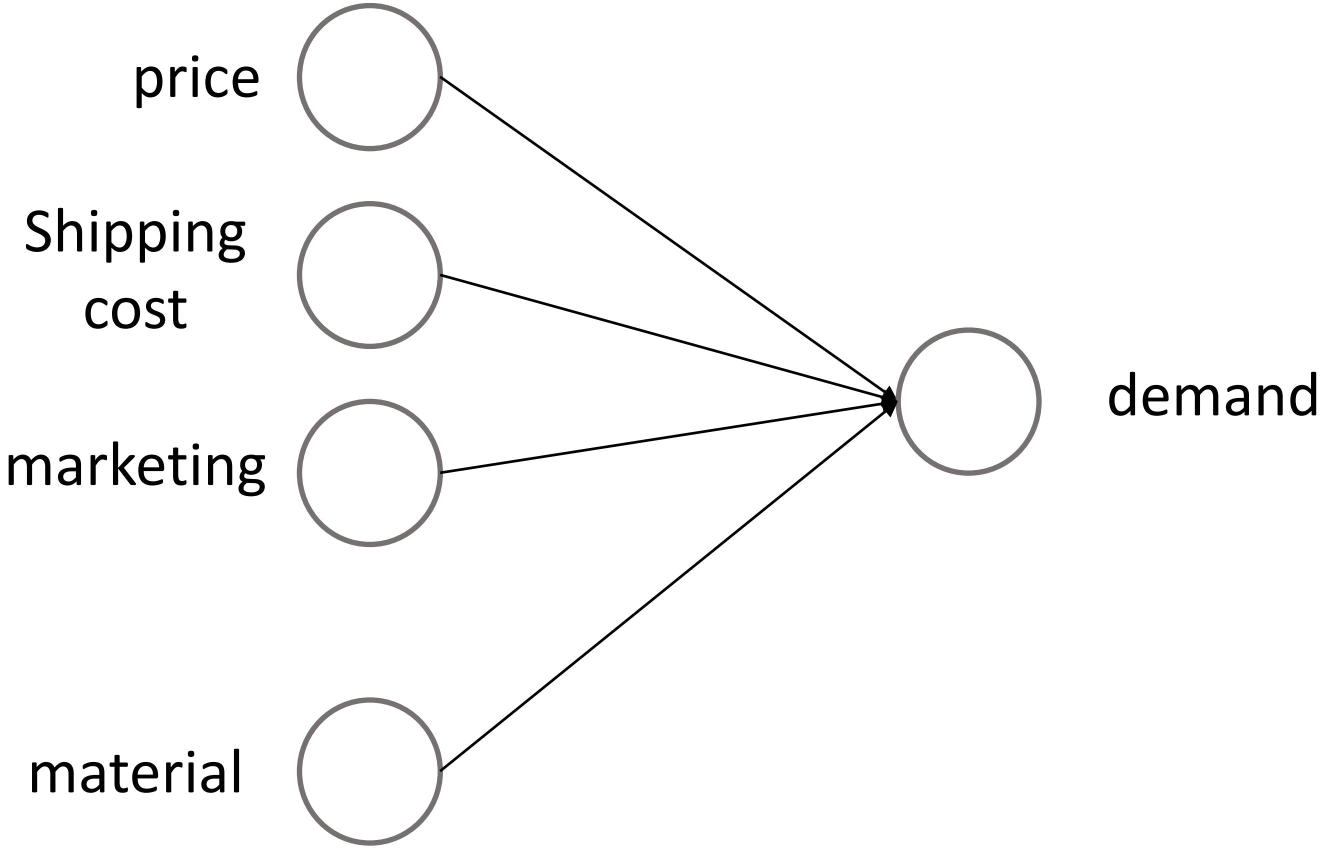
(descent) (step-size) (gradient)



Learning Representations

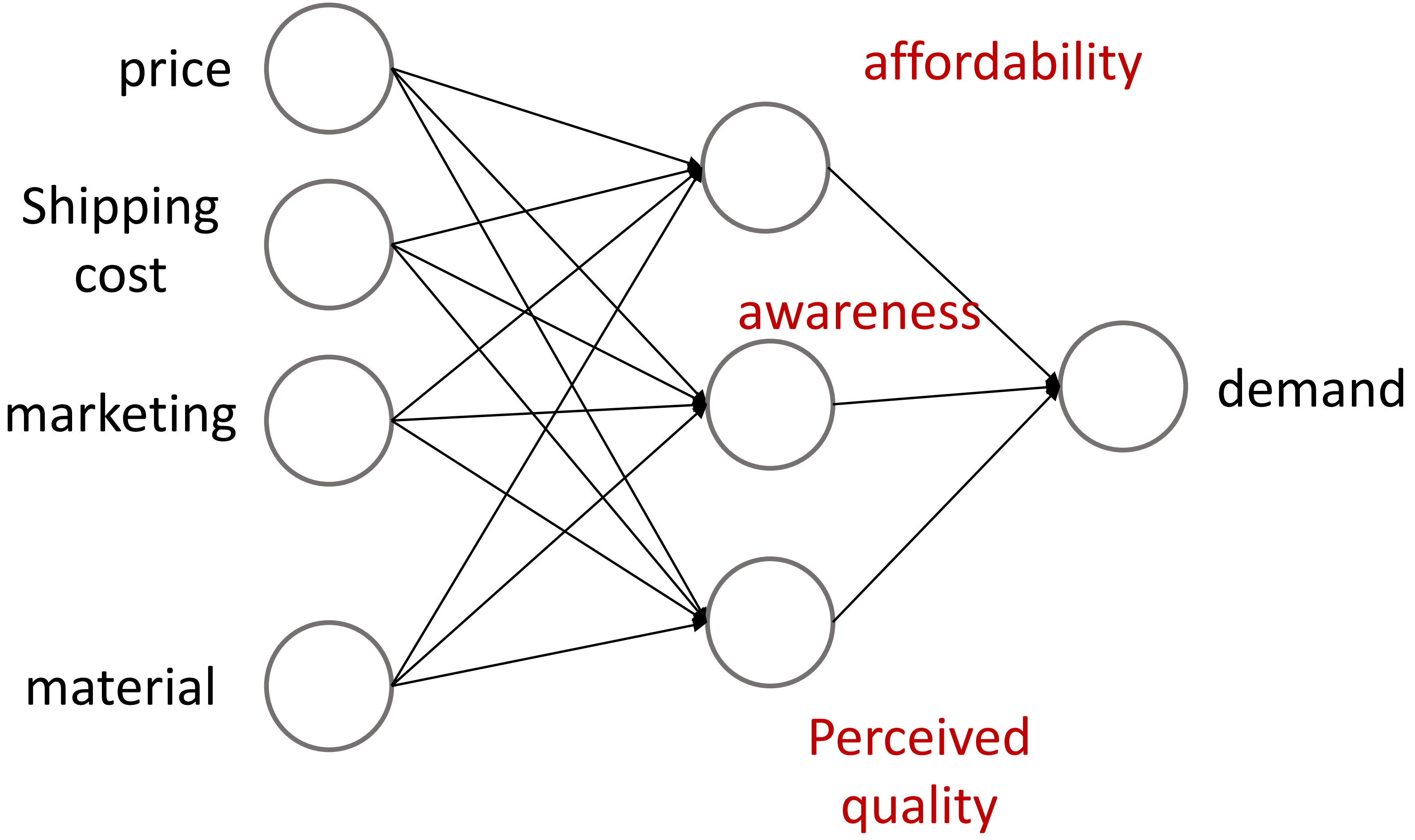
Demand Prediction

- Linear regression

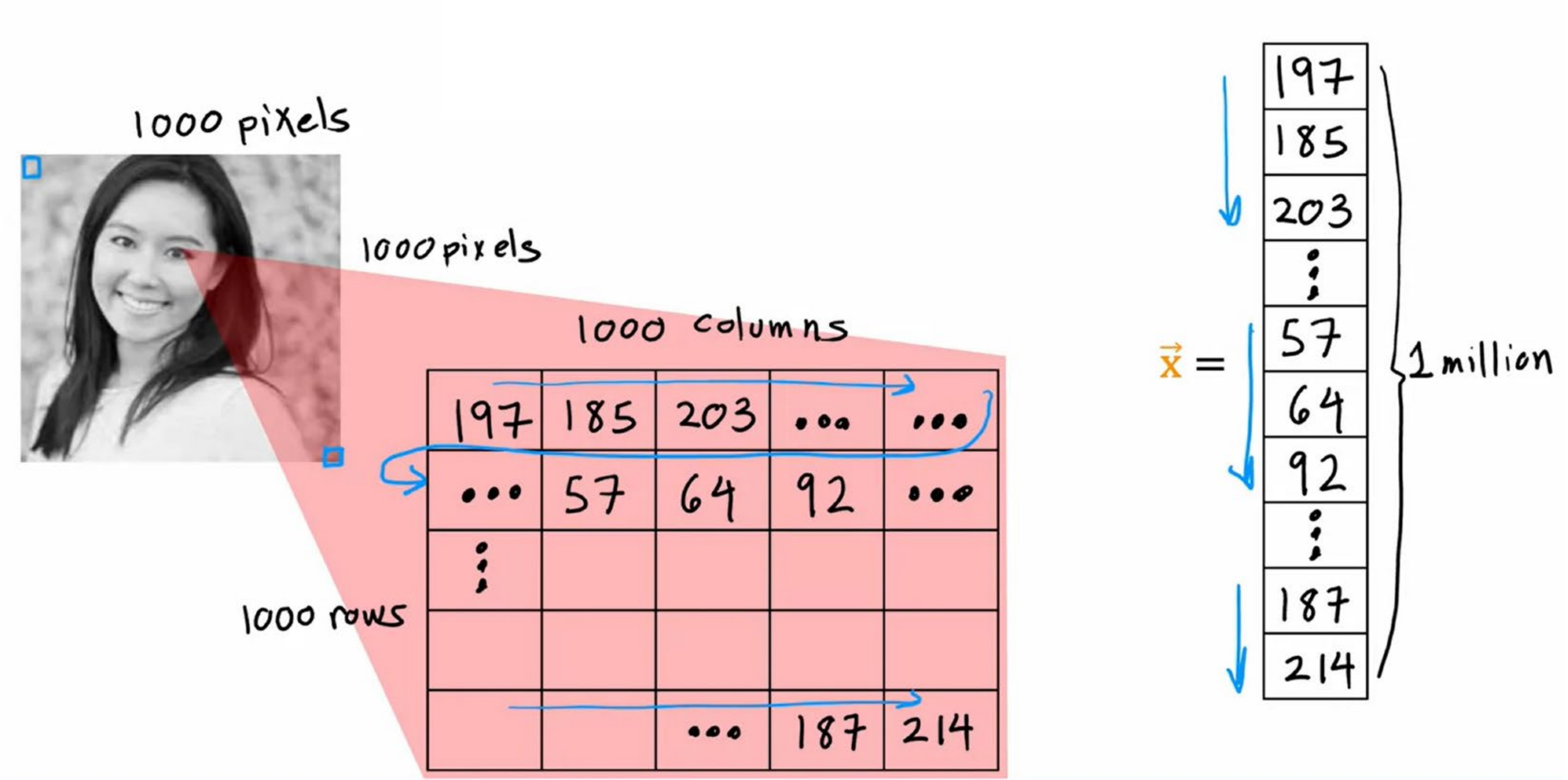


Demand Prediction

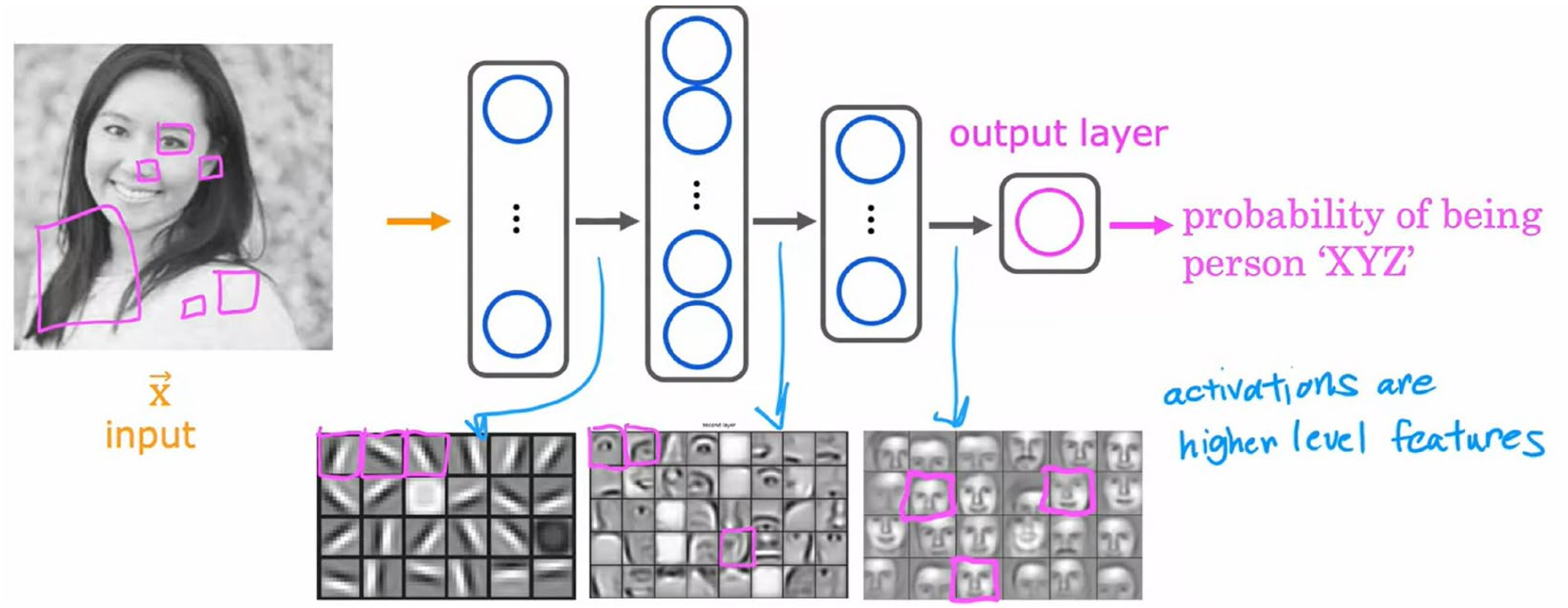
- Linear regression



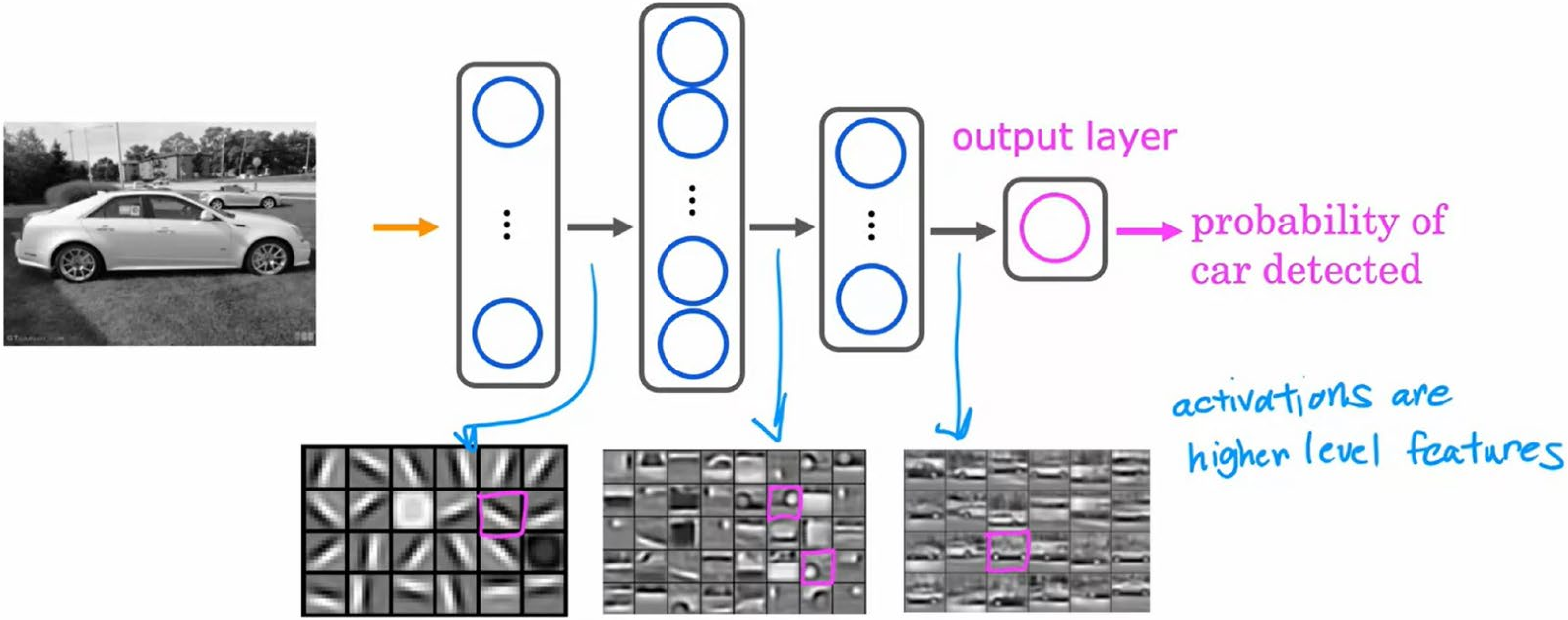
Face Recognition



Face Recognition



Car Classification



Exercise

Neural Networks for Boolean Logic Gate Functions

- Which one is linearly separable?
- How many layers required to model 'AND'? And 'XOR'?

Suppose we have binary inputs that only take on values of 0 or 1. Below are truth tables and plots for the Boolean logic gate functions **AND** and **XOR**.

x_1	x_2	**AND**
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	**XOR**
0	0	0
0	1	1
1	0	1
1	1	0

Neural Networks for Boolean Logic Gate Functions

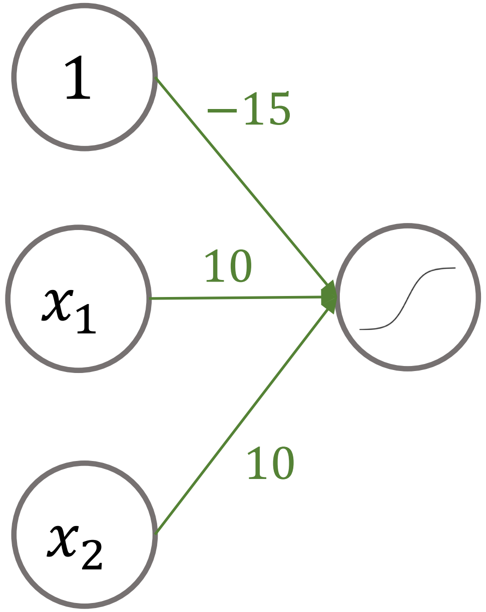
Notice that **AND** appears linearly separable (you could draw a line through the figure separating the positive and negative examples) whereas **XOR** does not. Thus, a simple neural network to model the **AND** function might not have a hidden layer whereas a simple neural network to model the **XOR** function might have a hidden layer. Which of the following Boolean logic gate functions are linearly separable?

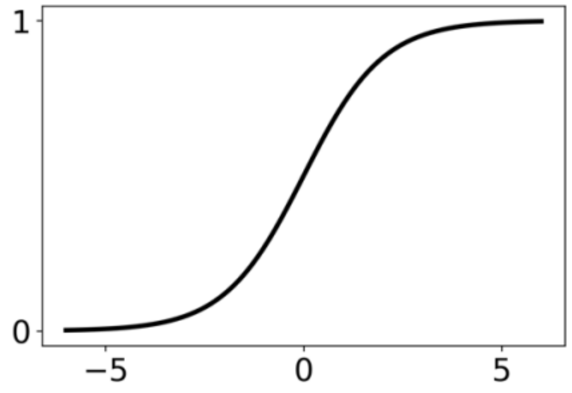
- NAND
- OR
- NOR
- XNOR

Neural Networks for Boolean Logic Gate Functions

- Which gate function does the neural network make?

$x_1 \in \{0,1\}, x_2 \in \{0,1\}$



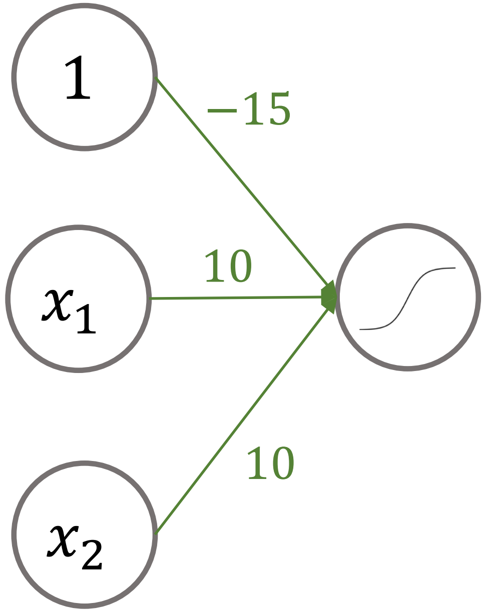


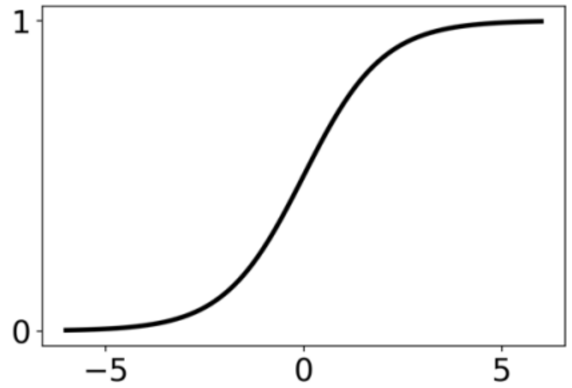
Sigmoid

Neural Networks for Boolean Logic Gate Functions

- Which gate function does the neural network make?

$x_1 \in \{0,1\}, x_2 \in \{0,1\}$





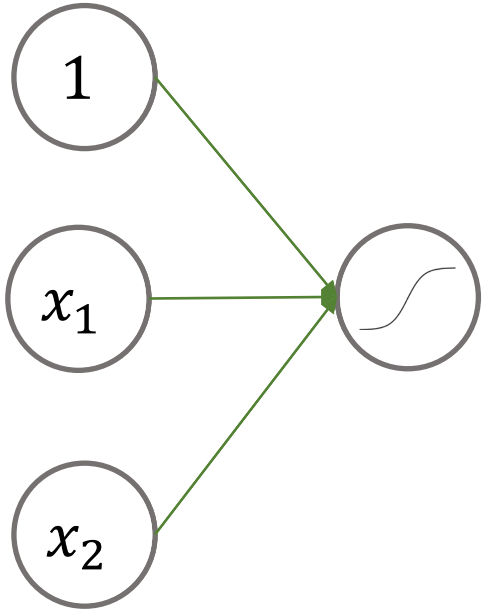
Sigmoid

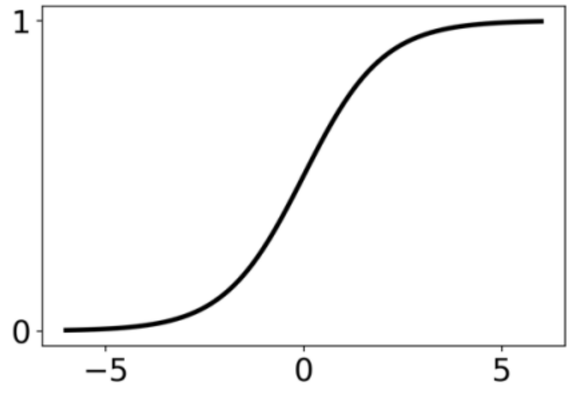
x_1	x_1	output
0	0	-15
0	1	-5
1	0	-5
1	1	5

Neural Networks for Boolean Logic Gate Functions

- Could you make 'OR'?

$$x_1 \in \{0,1\}, x_2 \in \{0,1\}$$



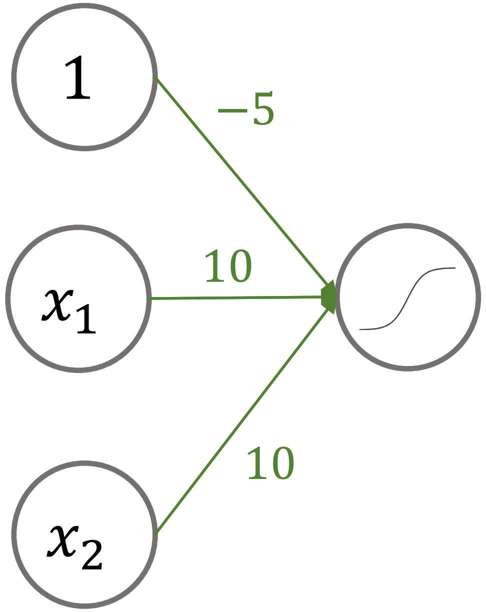


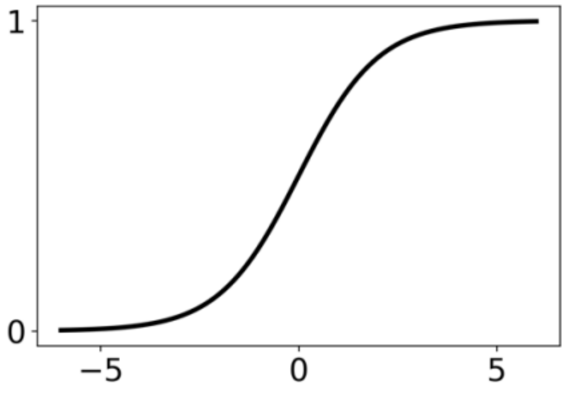
Sigmoid

Neural Networks for Boolean Logic Gate Functions

- Could you make 'OR'?

$$x_1 \in \{0,1\}, x_2 \in \{0,1\}$$





Sigmoid

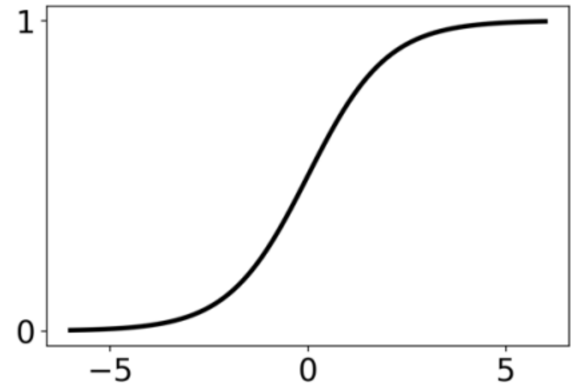
x_1	x_1	output
0	0	-5
0	1	5
1	0	5
1	1	15

Neural Networks for Boolean Logic Gate Functions

- Could you make ‘XOR’?

$$x_1 \in \{0,1\}, x_2 \in \{0,1\}$$

x_1	x_1	output
0	0	0
0	1	1
1	0	1
1	1	0

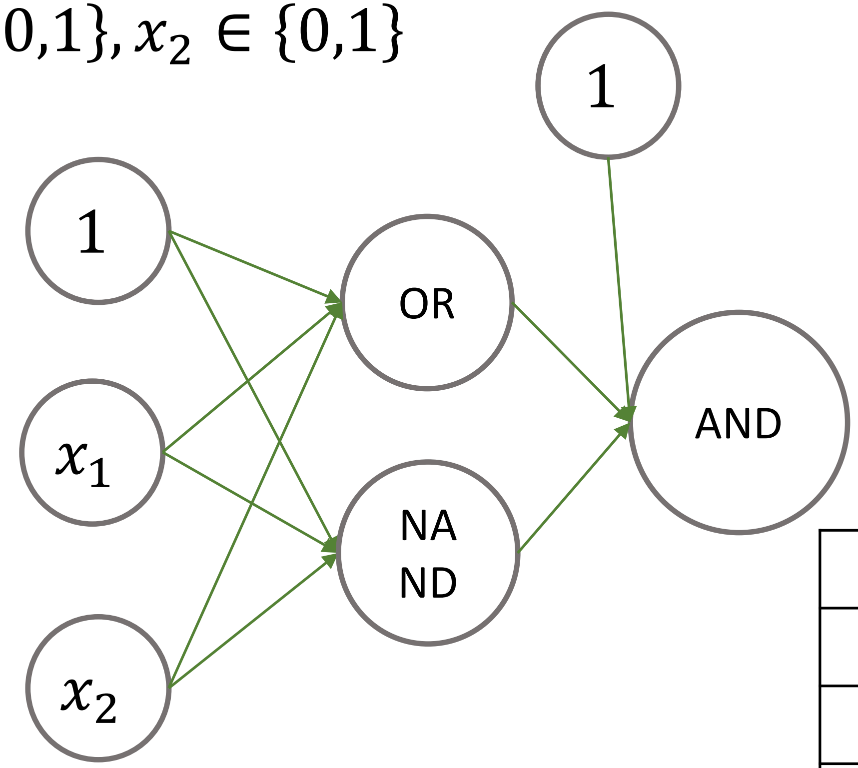


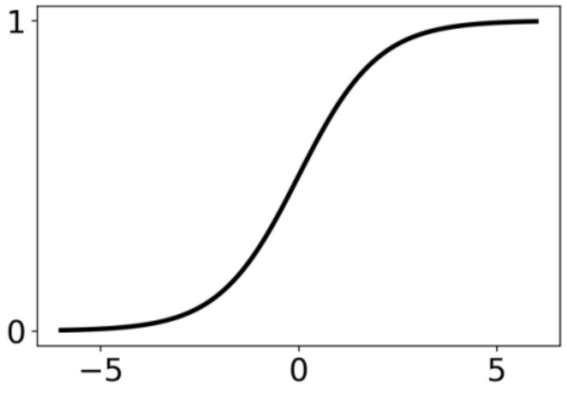
Sigmoid

Neural Networks for Boolean Logic Gate Functions

- Could you make ‘XOR’?

$x_1 \in \{0,1\}, x_2 \in \{0,1\}$





Sigmoid

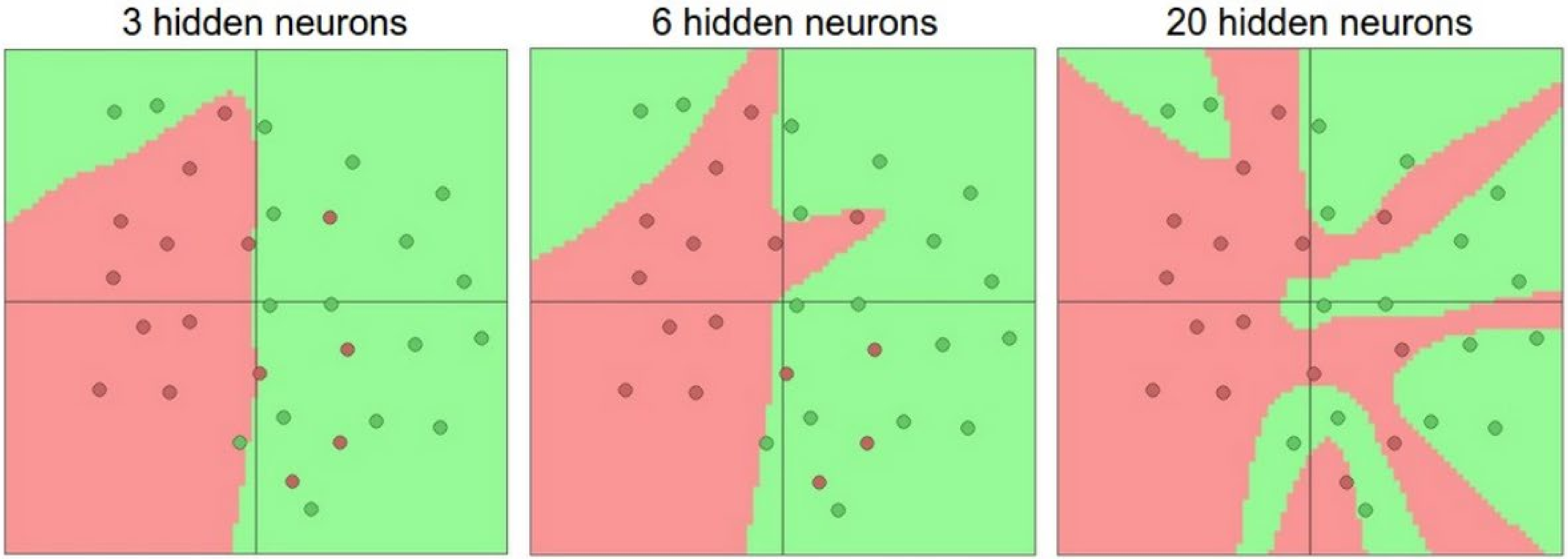
x_1	x_2	OR	NAND	AND
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

The Universal Approximator

The Universal Approximation Theorem

- A single hidden layer neural network can approximate any continuous function arbitrarily well, given enough hidden units.
- This holds for many different activation functions, e.g. sigmoid, tanh, ReLU, etc.

The Universal Approximation Theorem



Cybenko Theorem

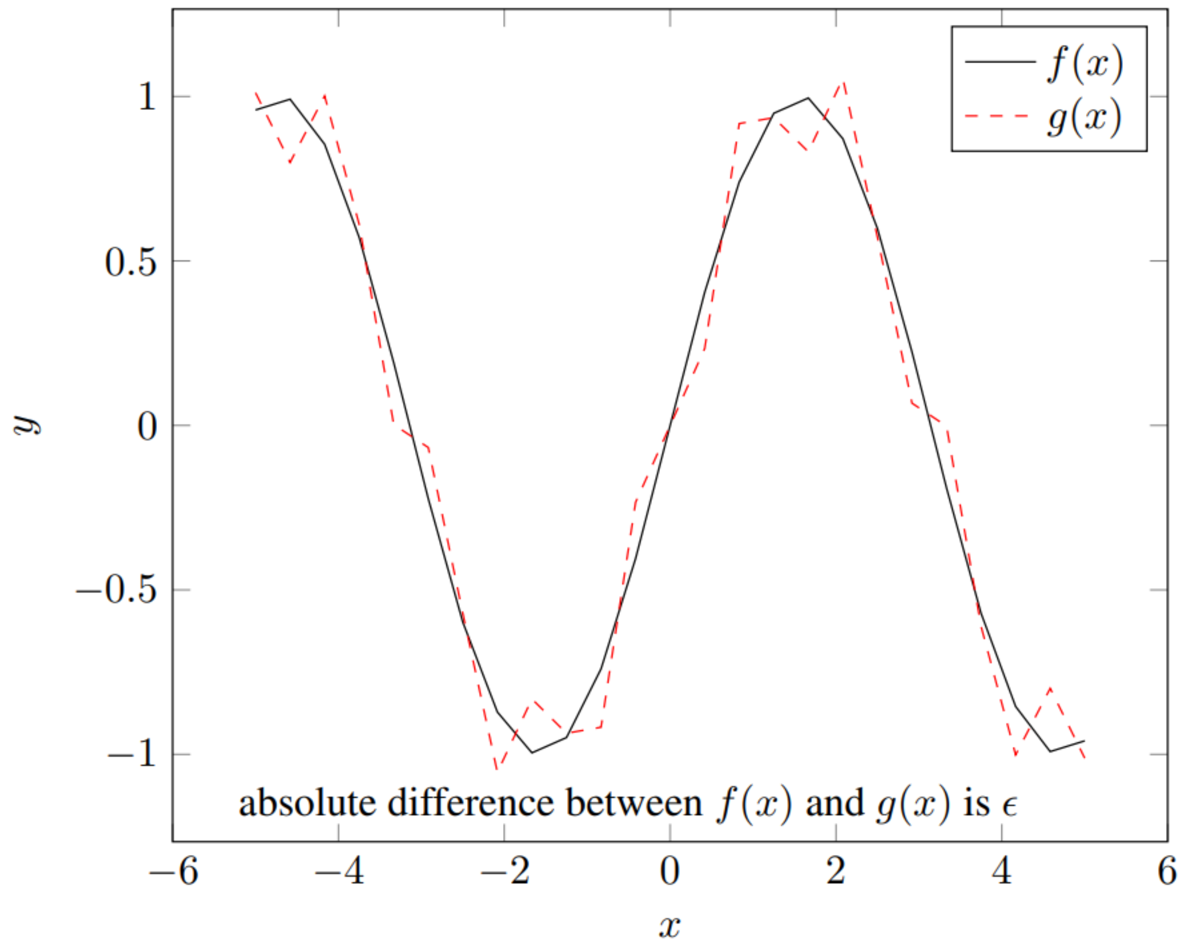
- Cybenko Approximation by Superposition of Sigmoidal Function

Let $C([0,1]^n)$ *denote the set of all continuous function* $[0,1]^n \to \mathbb{R}$, *let* σ *be any sigmoidal activation function then the finite sum of the form* $f(x) = \sum_{i=1}^{N} \alpha_i \, \sigma(w_i^{\top} x + b_i)$ *is dense in* $C([0,1]^n)$

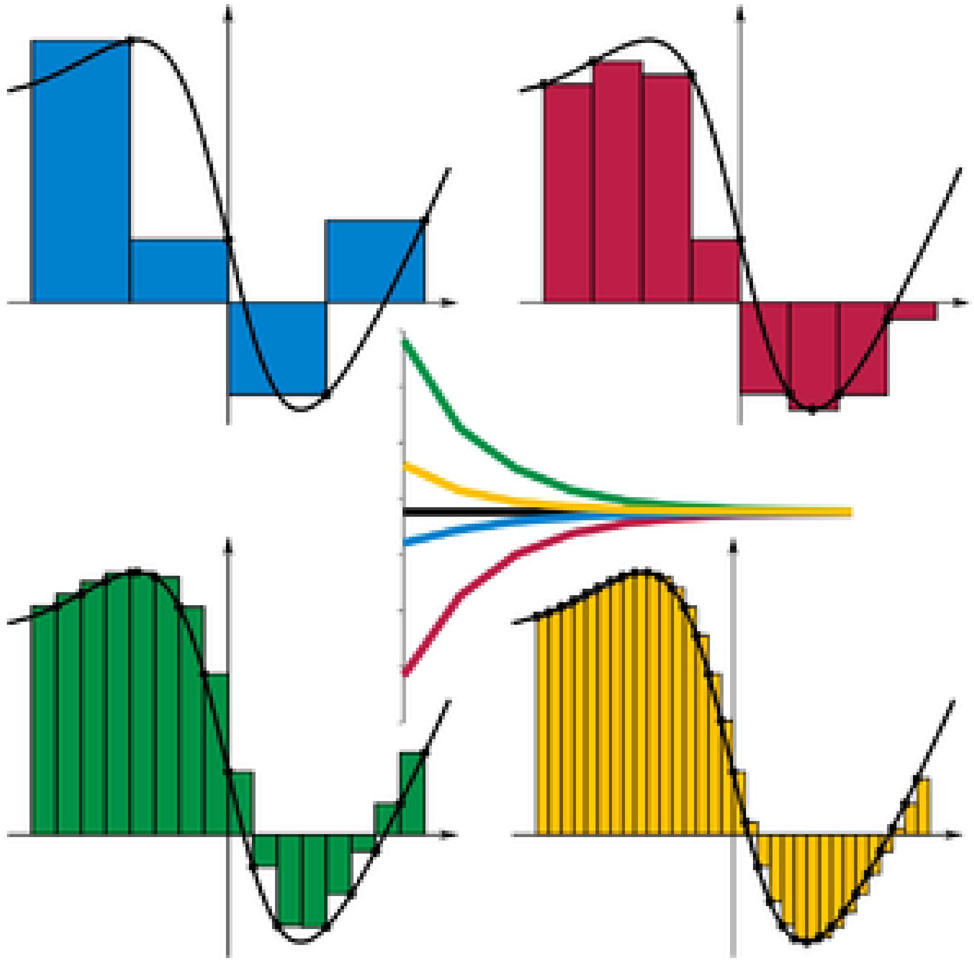
For any $g \in C([0,1]^n)$ *and any* $\epsilon > 0$, *there exists* $f: x \to \sum_{i=1}^{N} \alpha_i \, \sigma\left(w_i^{\top} x + b_i\right)$, *such that* $|f(x) - g(x)| < \epsilon$ *for all* $x \subset [0,1]^n$.

Cybenko Theorem

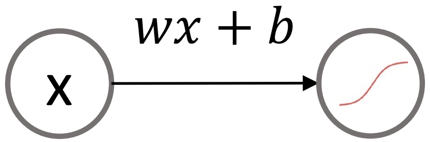
- Cybenko Approximation by Superposition of Sigmoidal Function



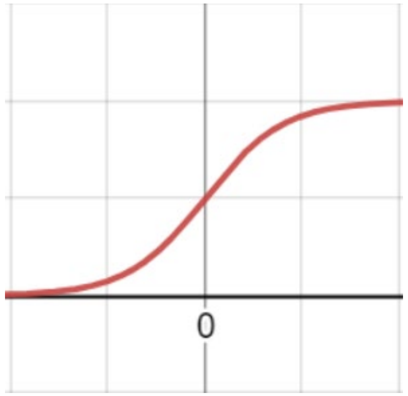
The Universal Approximation Theorem



The Universal Approximation Theorem



$w = 5, b = 0$

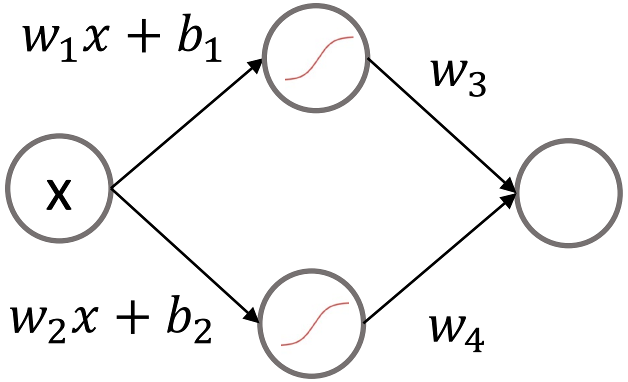


$w = 5, b = 3$

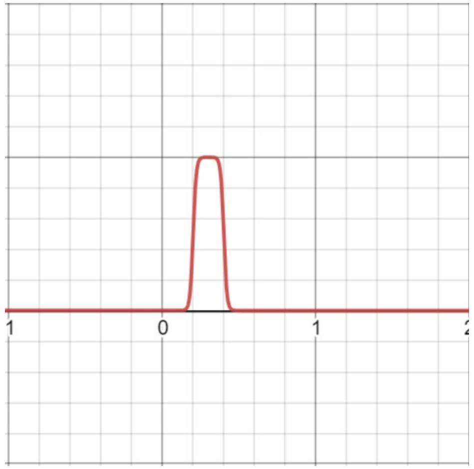
$w = 10, b = -7$

$w = 100, b = -20$

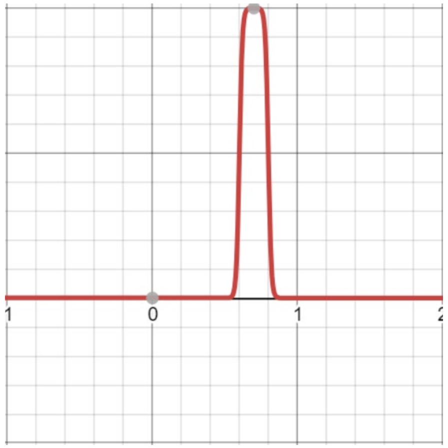
The Universal Approximation Theorem



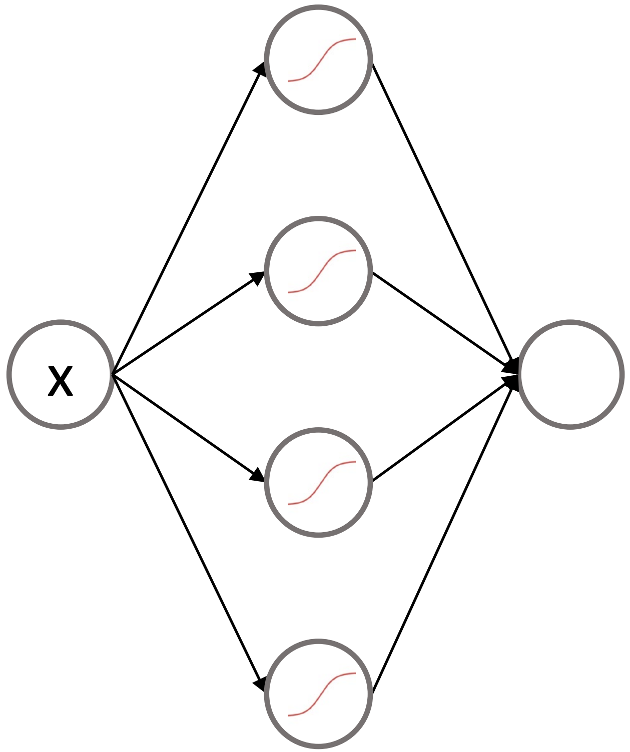
$$w_1 = 100, b_1 = -20$$
$$w_2 = 100, b_2 = -40$$
$$w_3 = 1, w_4 = -1$$

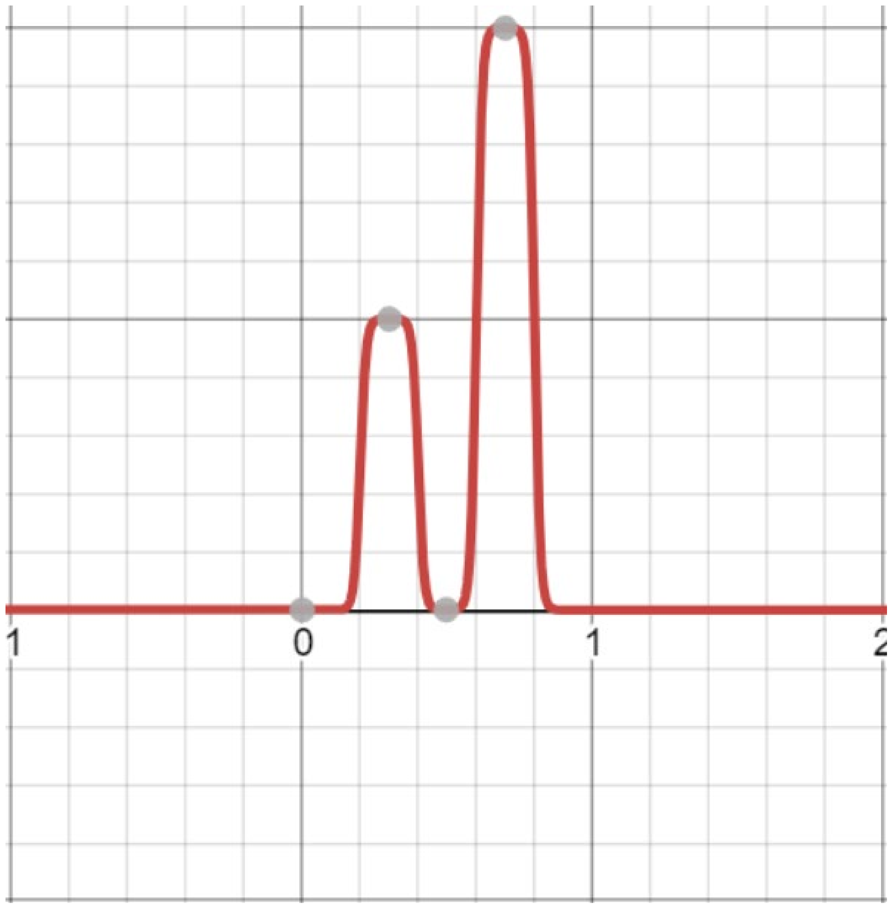


$$w_1 = 100, b_1 = -60$$
$$w_2 = 100, b_2 = -80$$
$$w_3 = 2, w_4 = -2$$

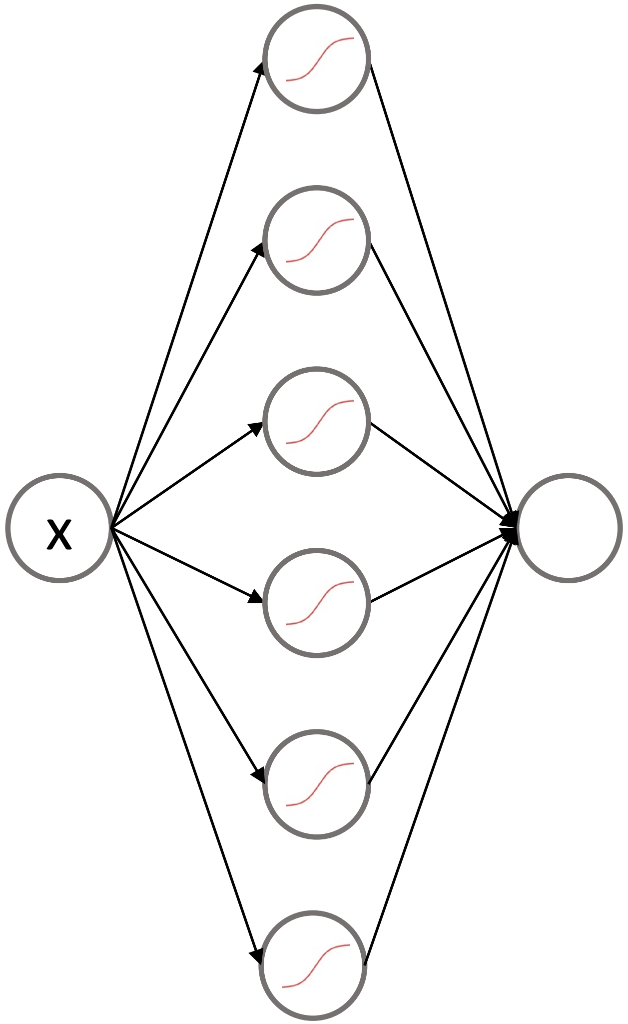


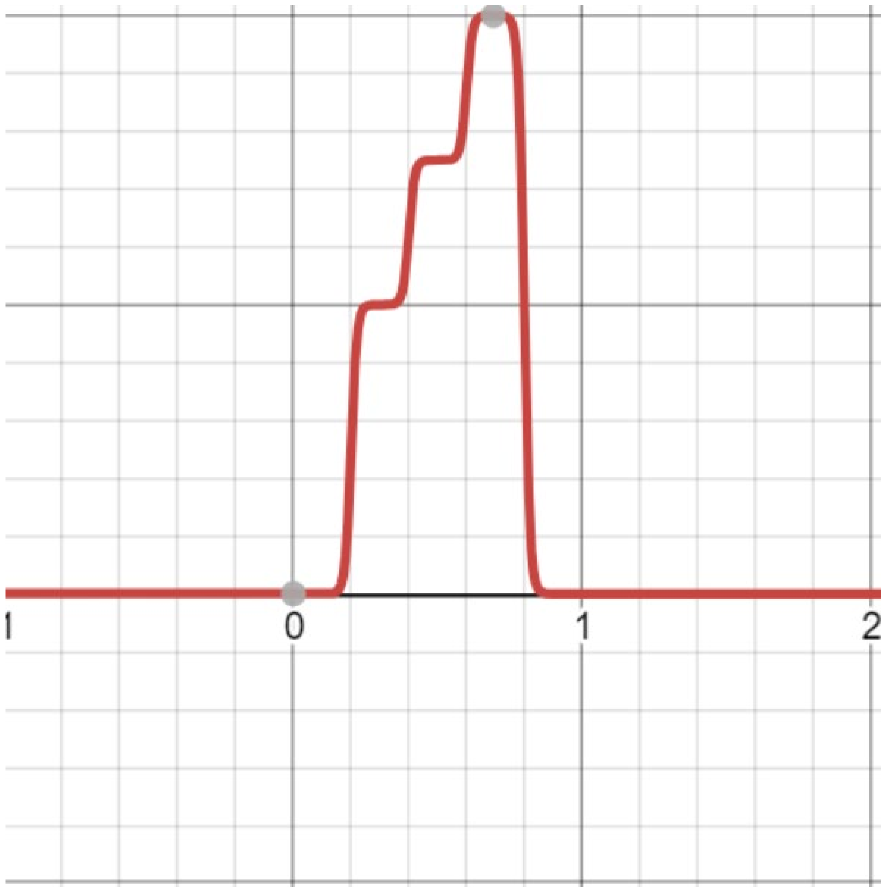
The Universal Approximation Theorem



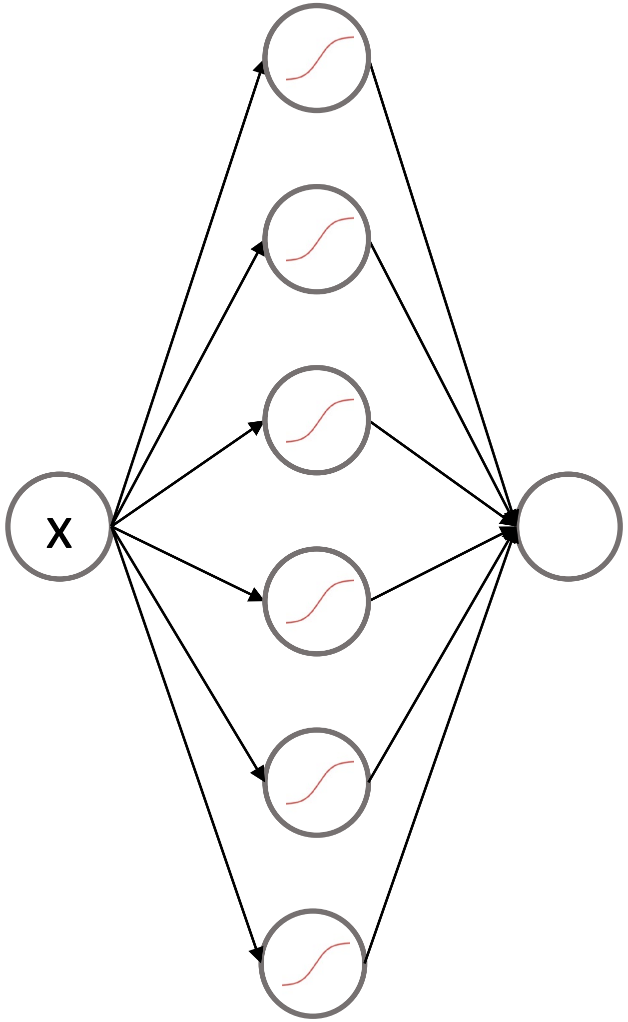


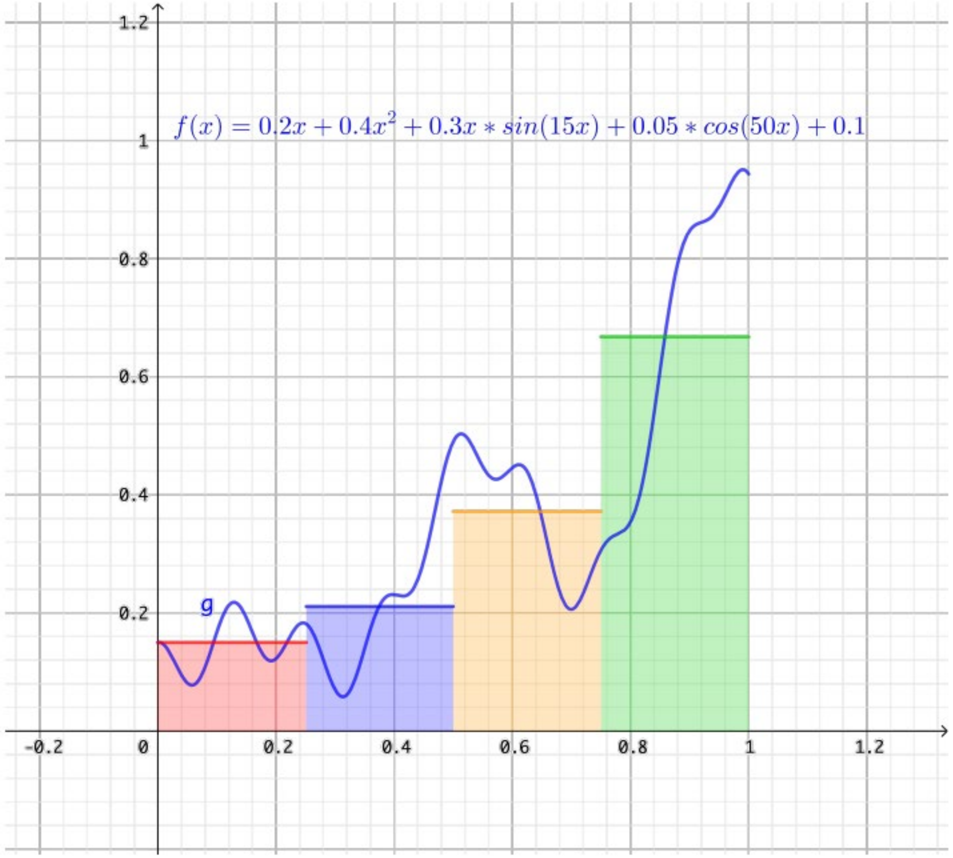
The Universal Approximation Theorem



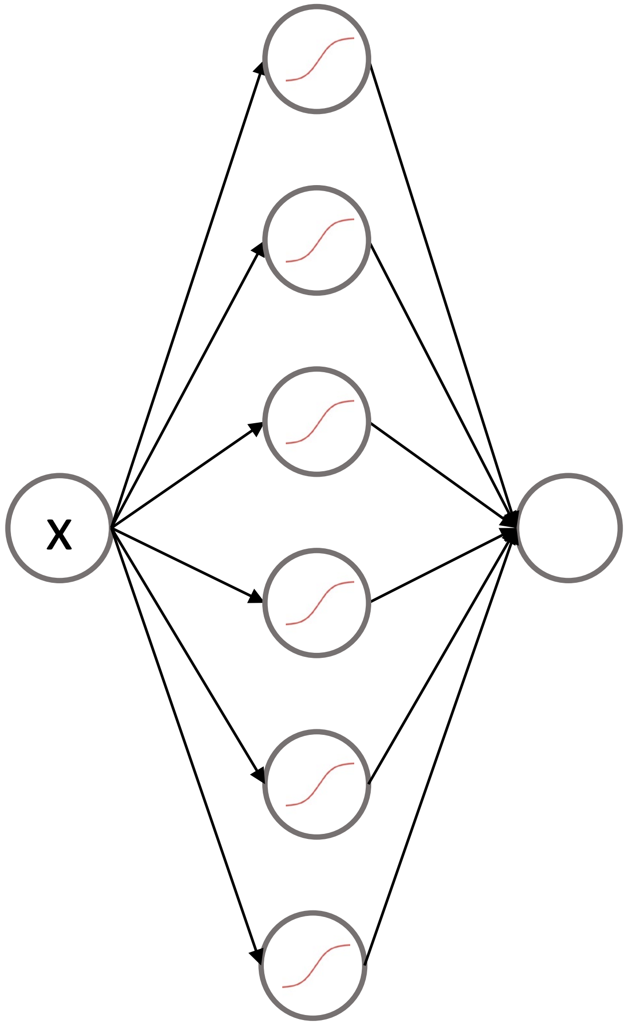


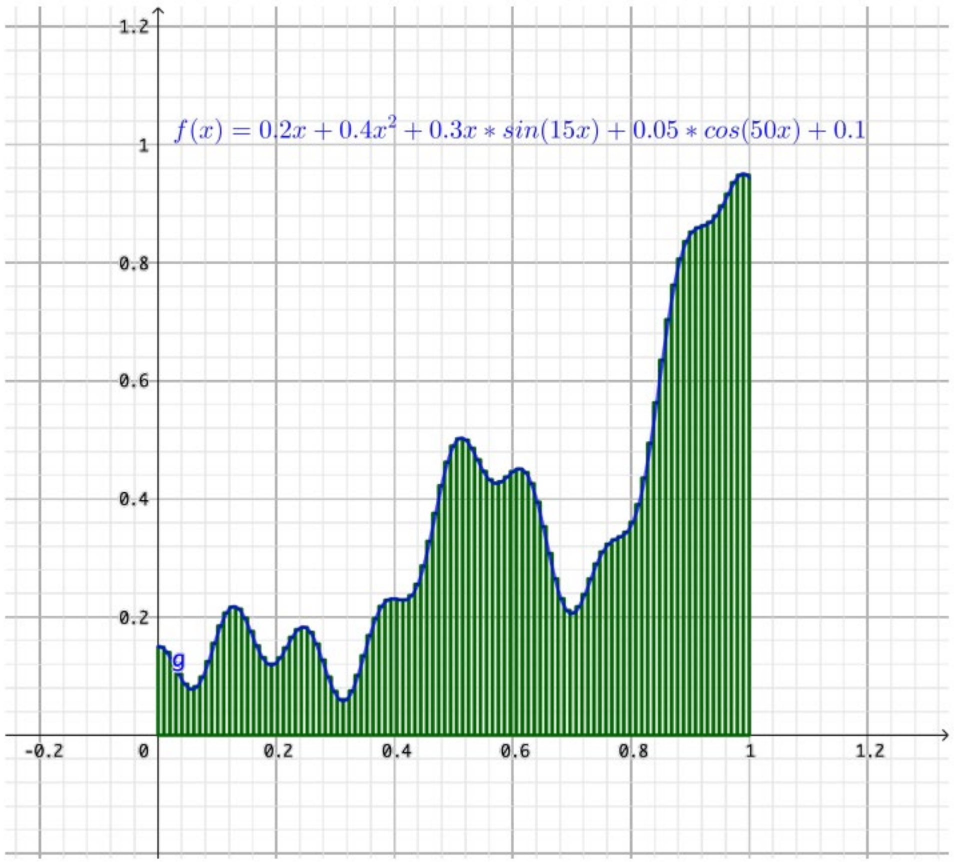
The Universal Approximation Theorem



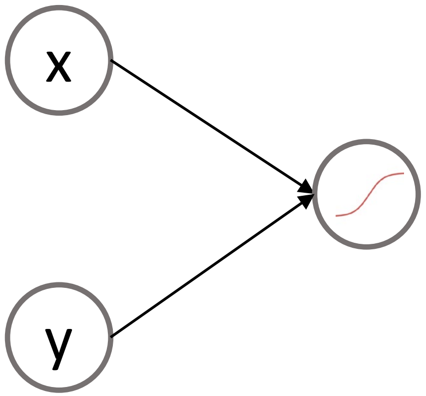


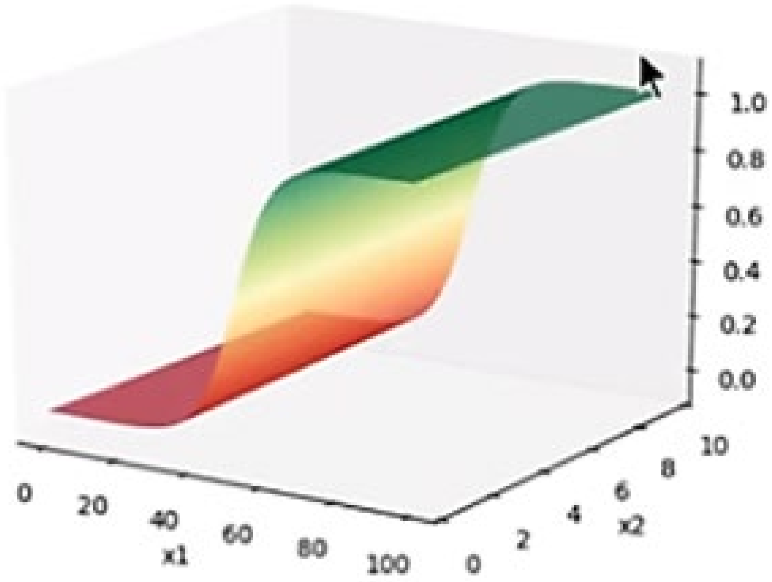
The Universal Approximation Theorem



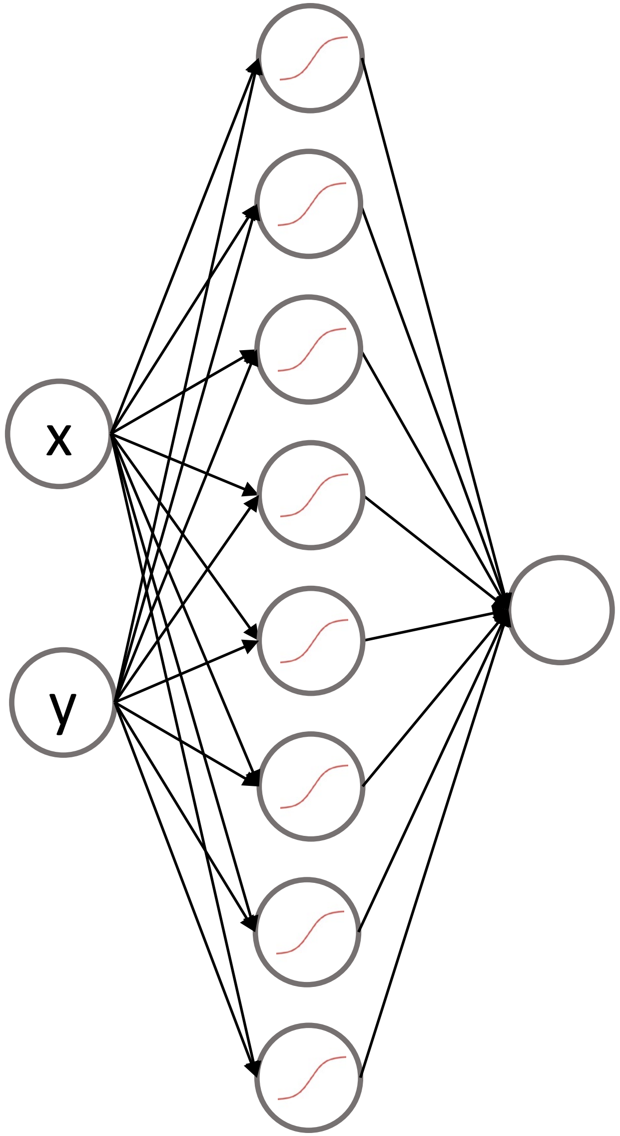


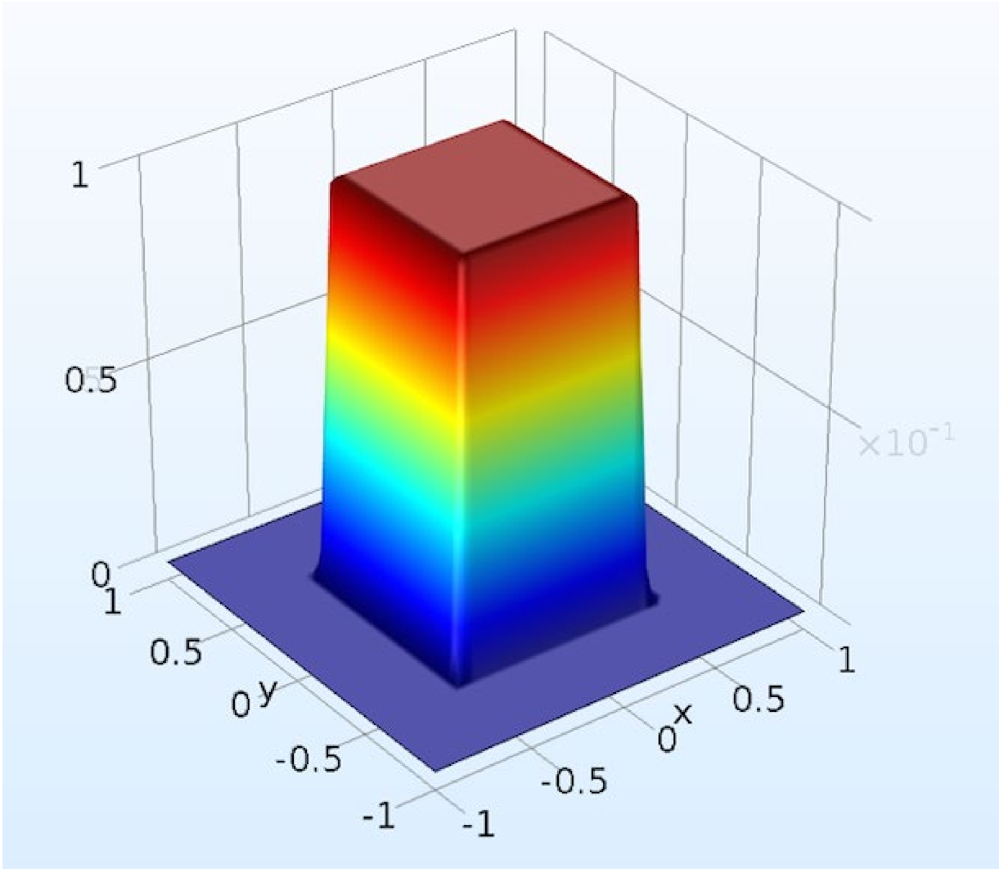
The Universal Approximator in 2D



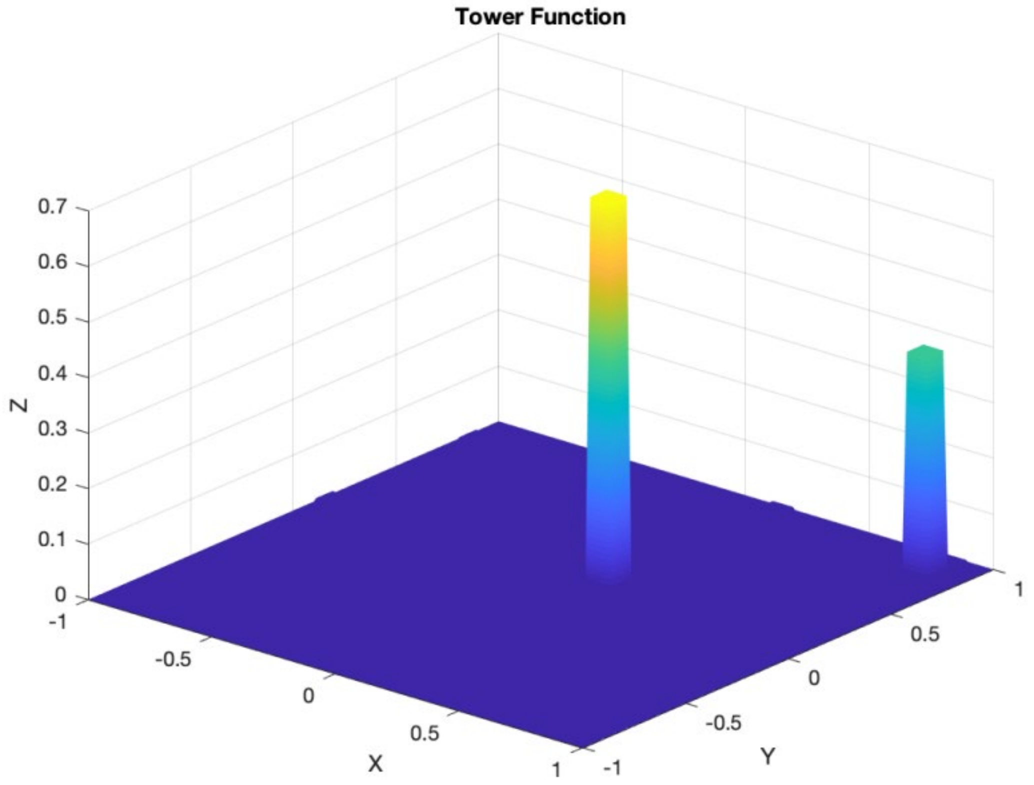


The Universal Approximator in 2D

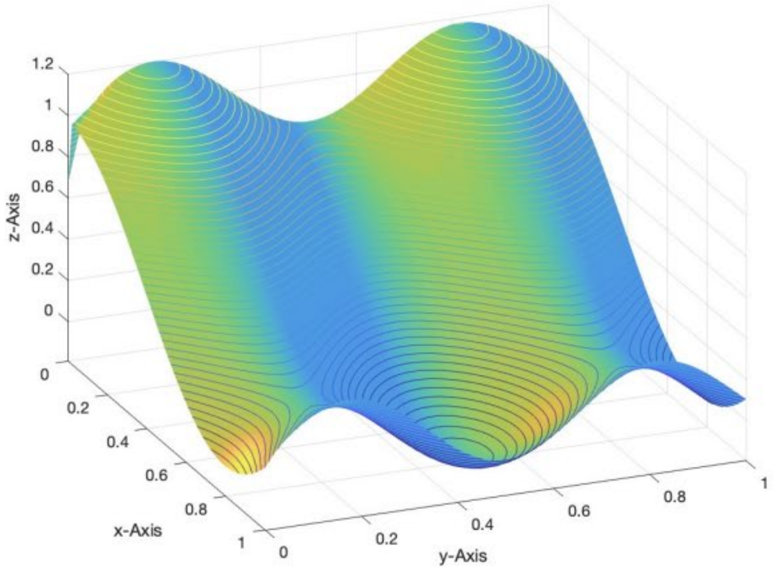


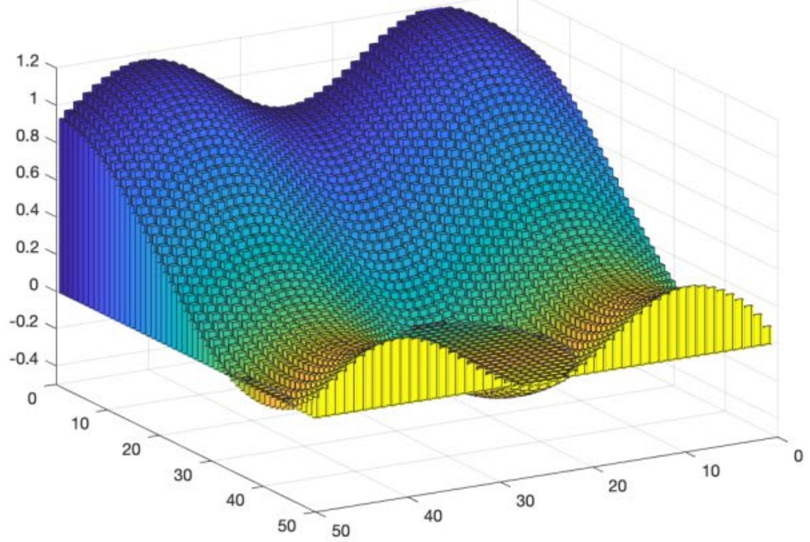


The Universal Approximator in 2D



The Universal Approximator in 2D





The Universal Approximation Theorem

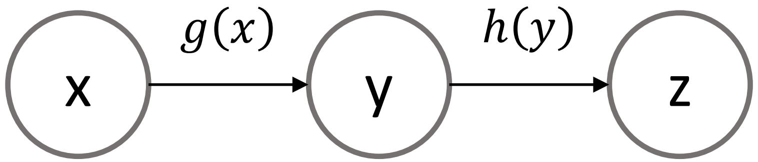
- Single layer might be enough, but it requires ‘enough’ neurons.
- Informally, ‘shallower and wider’ networks require exponentially more hidden units to compute ‘narrower and deeper’ neural networks
  - [Lecture 2 | The Universal Approximation Theorem - YouTube]

The Chain Rule

The Chain Rule

- A single variable chain rule

$$f, g, h: \mathbb{R} \to \mathbb{R}$$



$$f: h \circ g$$

$$f'(x) = h'\big(g(x)\big)g'(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \qquad y = g(x), z = h(y)$$

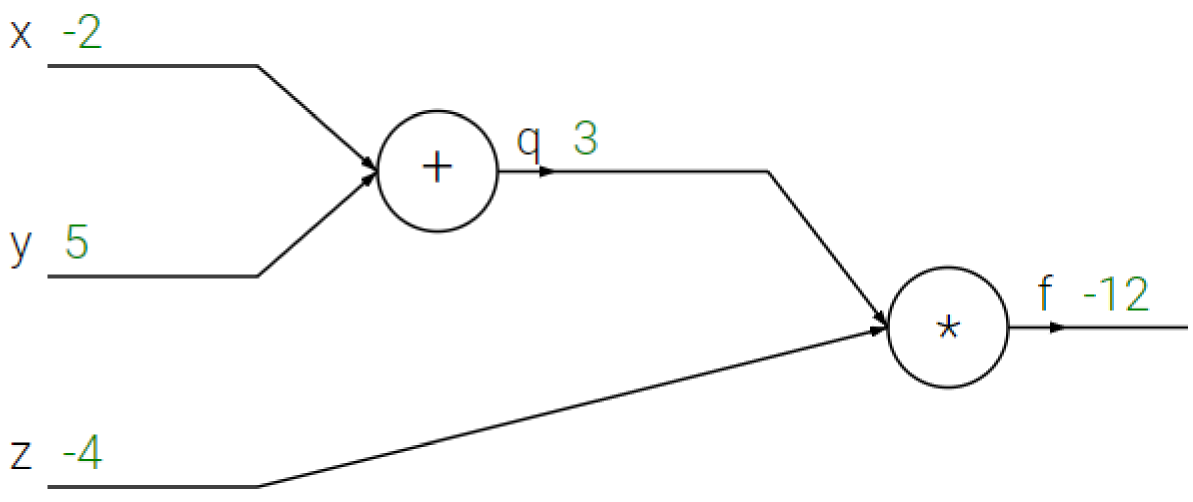
Simple Example

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

$$q = x + y, \qquad f = qz$$

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

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



Simple Example

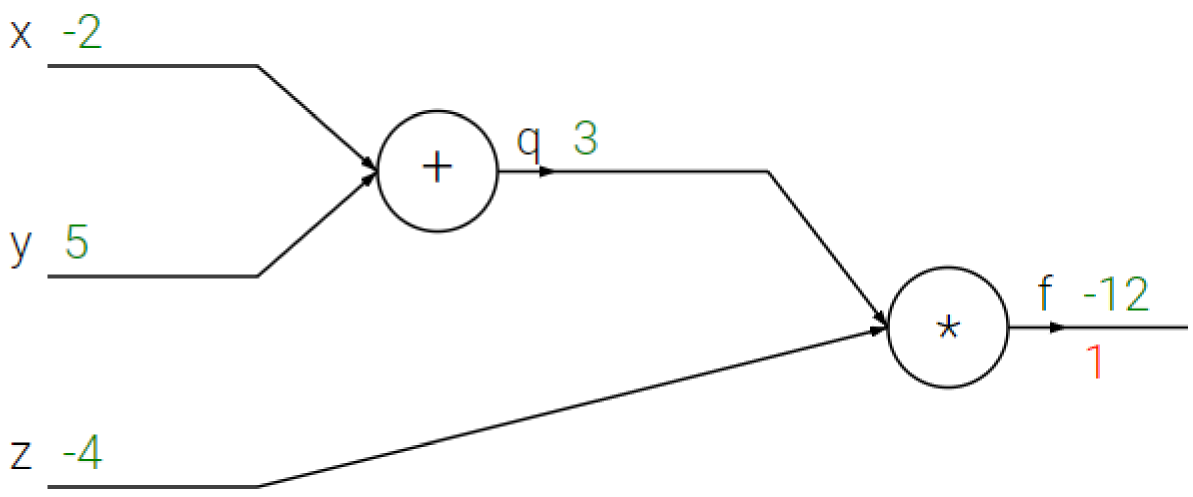
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

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

$$q = x + y, \qquad f = qz$$

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

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



Simple Example

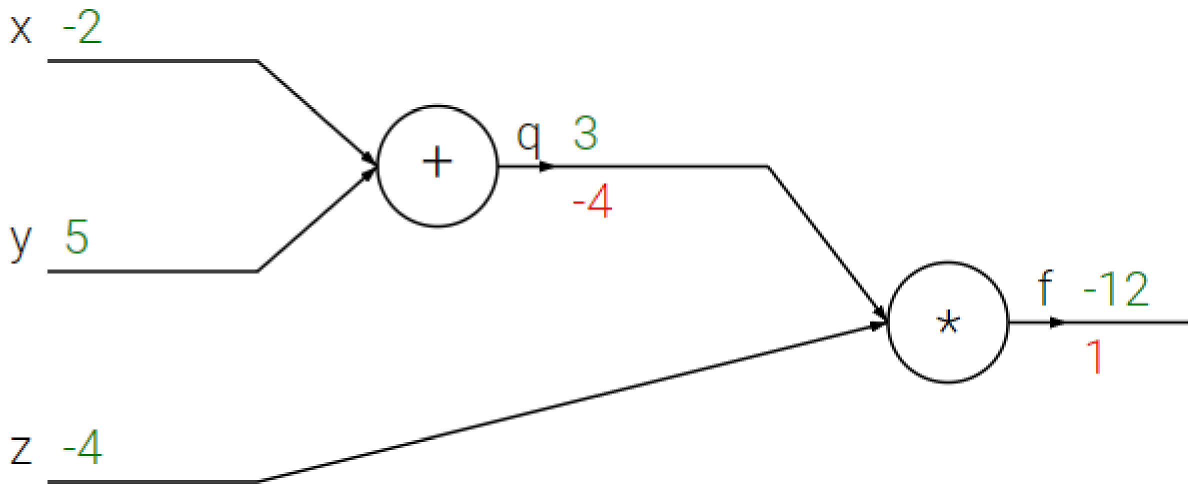
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

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

$$q = x + y, \qquad f = qz$$

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

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



Simple Example

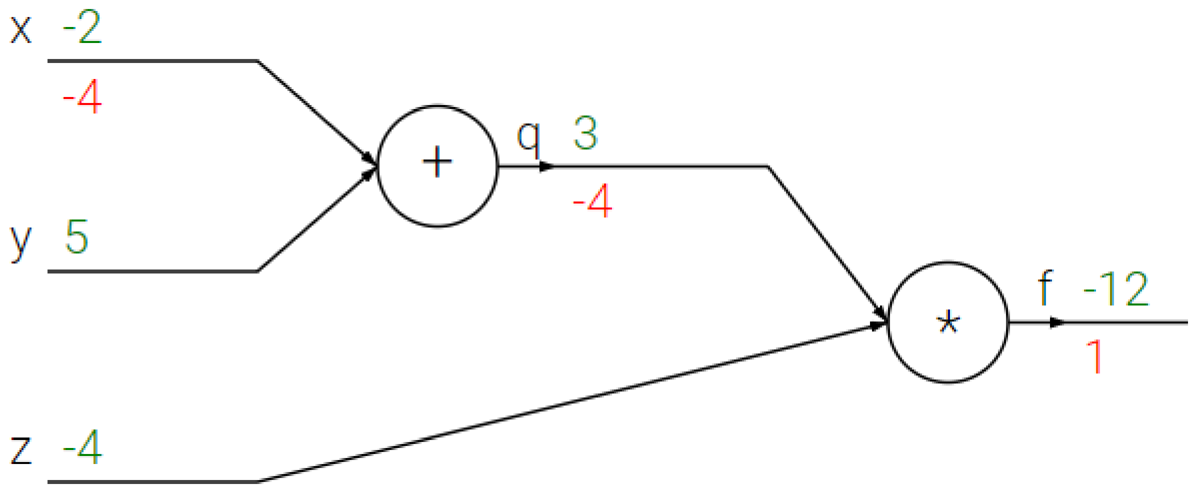
$$f(x, y, z) = (x + y)z$$

$$q = x + y, \qquad f = qz$$

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

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

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



Simple Example

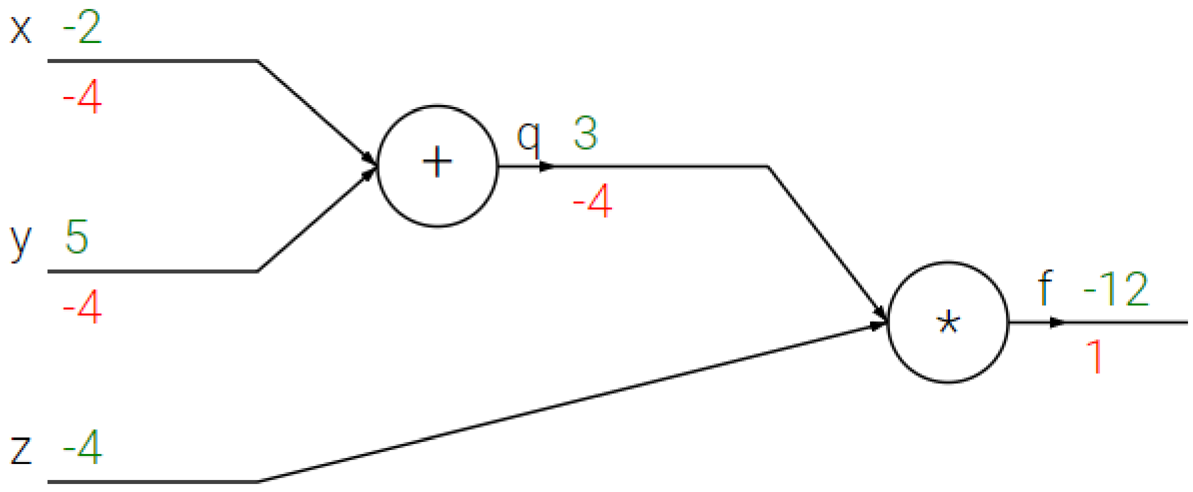
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial f}\frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

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

$$q = x + y, \qquad f = qz$$

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

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



Simple Example

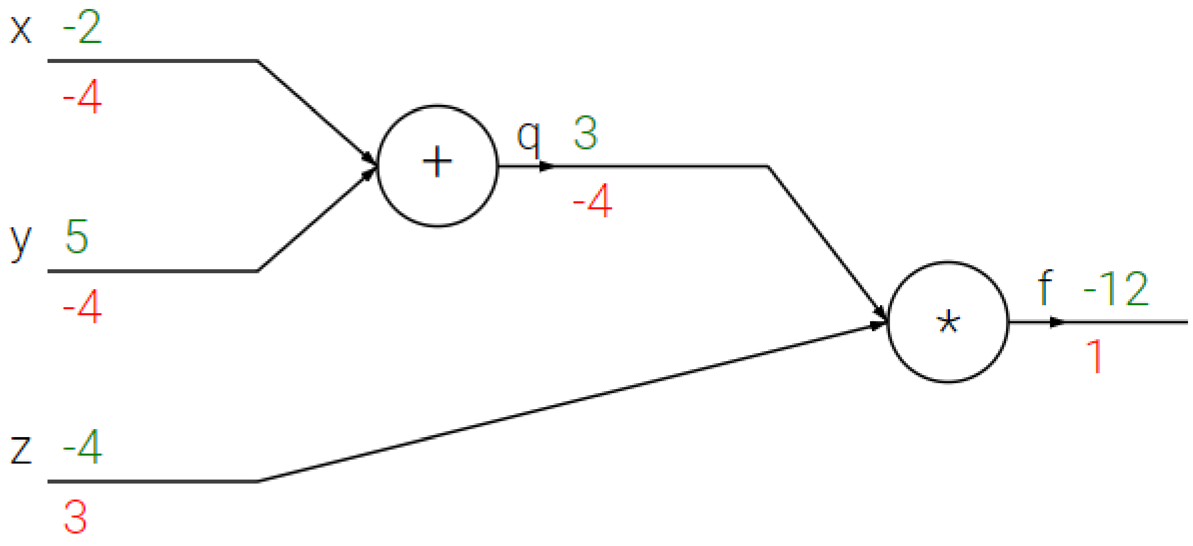
$$f(x,y,z) = (x+y)z$$

$$q = x + y, \qquad f = qz$$

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

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

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



[CS231n Convolutional Neural Networks for Visual Recognition]()

Sigmoid Example

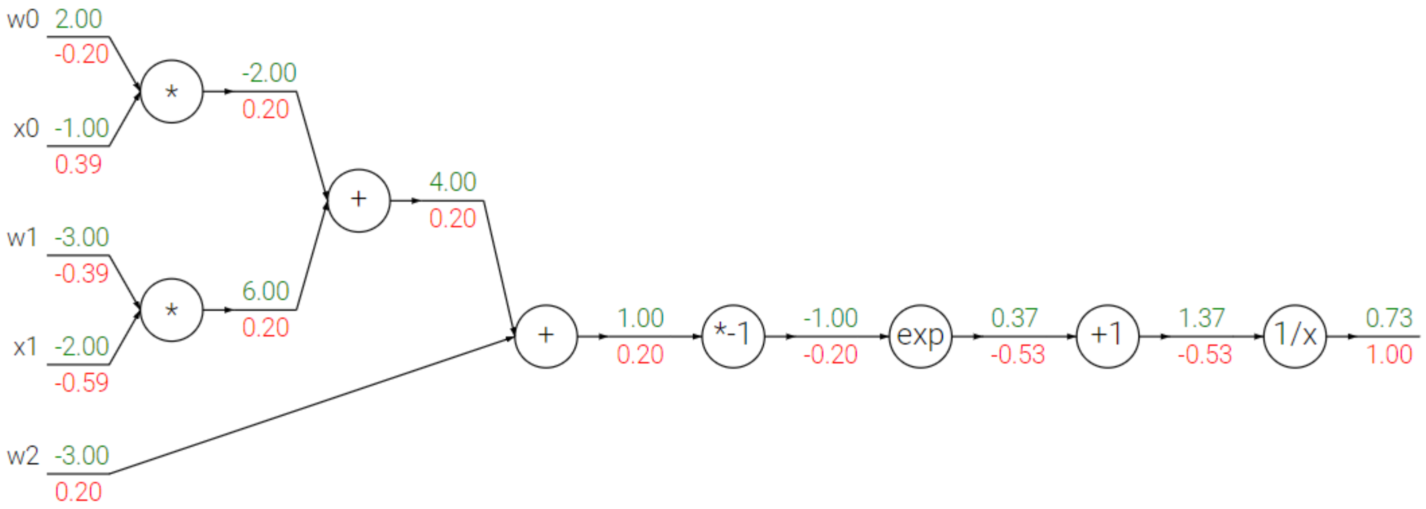
$$\sigma(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$f(x) = \frac{1}{x}, \qquad g(x) = 1 + x, \qquad h(x) = e^{-x}, \qquad i(x) = w_0 x_0 + w_1 x_1 + w_2$$

Sigmoid Example

$$\sigma(x, w) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$f(x) = \frac{1}{x}, \qquad g(x) = 1 + x, \qquad h(x) = e^{-x}, \qquad i(x) = w_0 x_0 + w_1 x_1 + w_2$$



Backpropatagion

(Regression w/ MLP)

Gradient

- In vector calculus, the *gradient* of a *scalar-valued* differentiable function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ at the point x

$$\nabla f: \mathbb{R}^n \rightarrow \mathbb{R}^n \qquad \nabla f = \frac{\partial f}{\partial x} = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n}\right]$$

Jacobian

- In vector calculus, the *Jacobian* of a *vector-valued* differentiable function is the matrix of all its first-order partial derivatives.

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Matrix Calculus

$X \in \mathbb{R}^{n \times m}, y \in \mathbb{R}$

$f: \mathbb{R}^{n \times m} \to \mathbb{R}$

$y = f(x)$

$$\begin{array}{c} n \times m \\ \frac{\partial y}{\partial X} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \cdots & \frac{\partial y}{\partial X_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial X_{n1}} & \cdots & \frac{\partial y}{\partial X_{nm}} \end{bmatrix} \end{array}$$

Matrix Calculus

$X \in \mathbb{R}^{n \times m}, y \in \mathbb{R}^{l}$

$f: \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^{l}$

$y = f(x)$

$$\begin{array}{cc} & n \times m \\ \dfrac{\partial y_1}{\partial X} = & \begin{bmatrix} \dfrac{\partial y_1}{\partial X_{11}} & \cdots & \dfrac{\partial y_1}{\partial X_{1m}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_1}{\partial X_{n1}} & \cdots & \dfrac{\partial y_1}{\partial X_{nm}} \end{bmatrix} \end{array}$$

$\dfrac{\partial y}{\partial X}$ $\quad l \times n \times m$ (3 dim tensor)

Finite Difference

- Numerical method to compute the gradients based on the definition of gradients

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Forward difference

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}$$

$$\frac{df}{dx} \approx \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

Backward difference

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

Central difference

Finite Difference

- Numerical method to compute the gradients based on the definition of gradients

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Forward difference

$$\frac{df}{dx} \approx \frac{f(x) - f(x-\Delta x)}{\Delta x}$$

Backward difference

$$\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

Central difference

What's wrong with this approach?

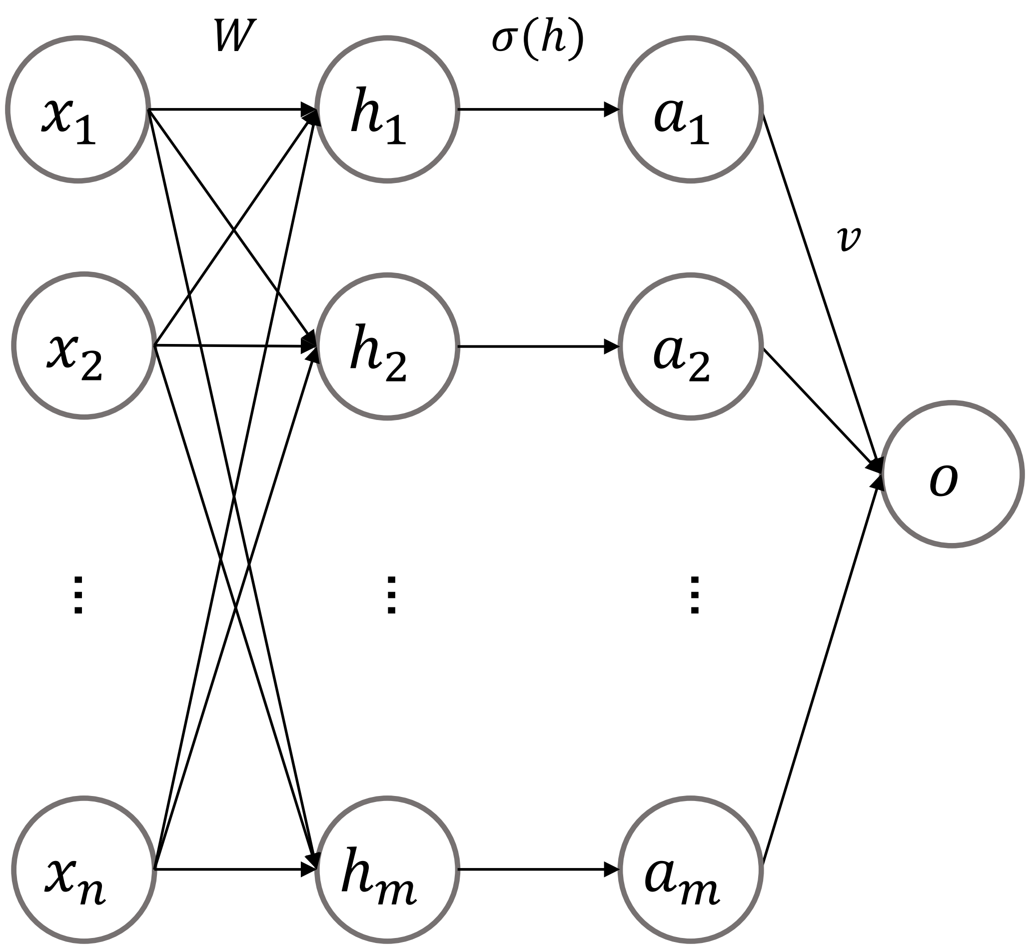
Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m\times n}, v \in \mathbb{R}^m$

$h = Wx \quad a = \sigma(h) \quad o = v^\top a$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial W}? \frac{\partial L}{\partial v}? \frac{\partial L}{\partial x}?$$



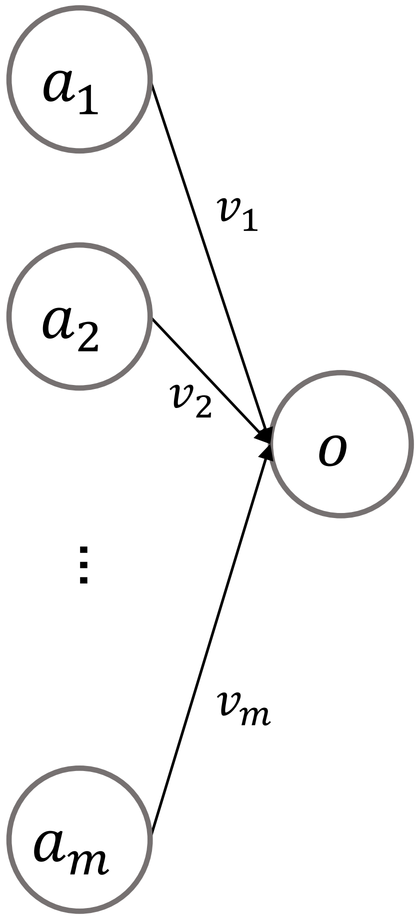
Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial v_i} = (o - y)\frac{\partial o}{\partial v_i} = (o - y)a_i$$



Two Layers MLP

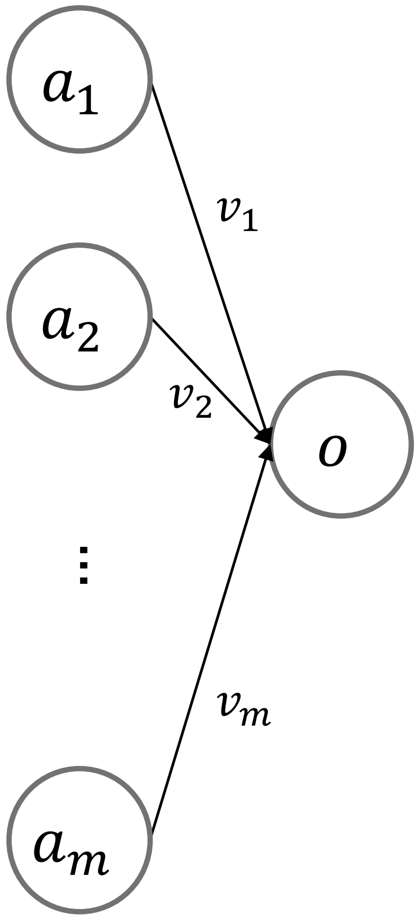
$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial v_i} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial v_i} = (o - y)\frac{\partial o}{\partial v_i} = (o - y)a_i$$

$$\frac{\partial L}{\partial v} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial v} = (o - y)\frac{\partial o}{\partial v} = (o - y)a$$



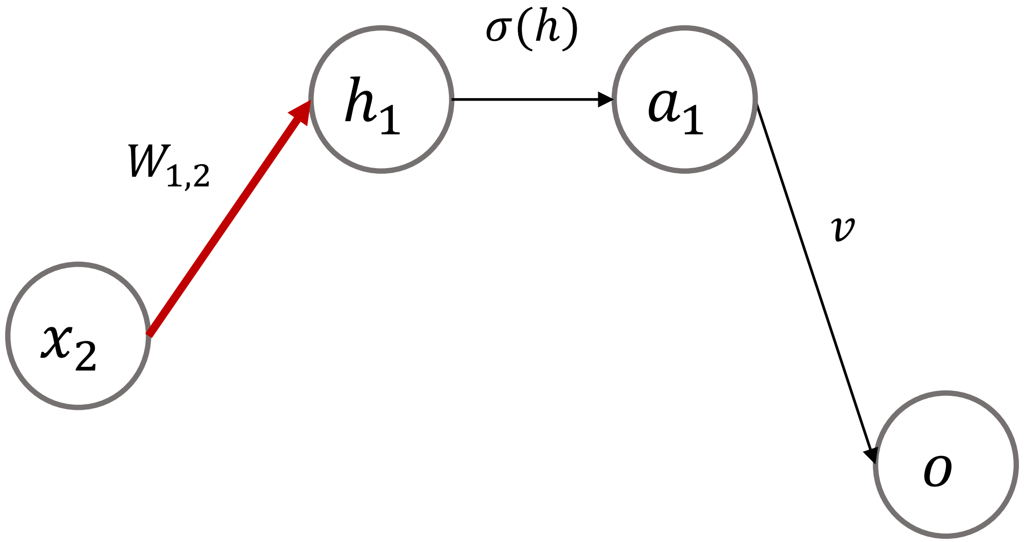
Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial W_{ij}}$$



Two Layers MLP

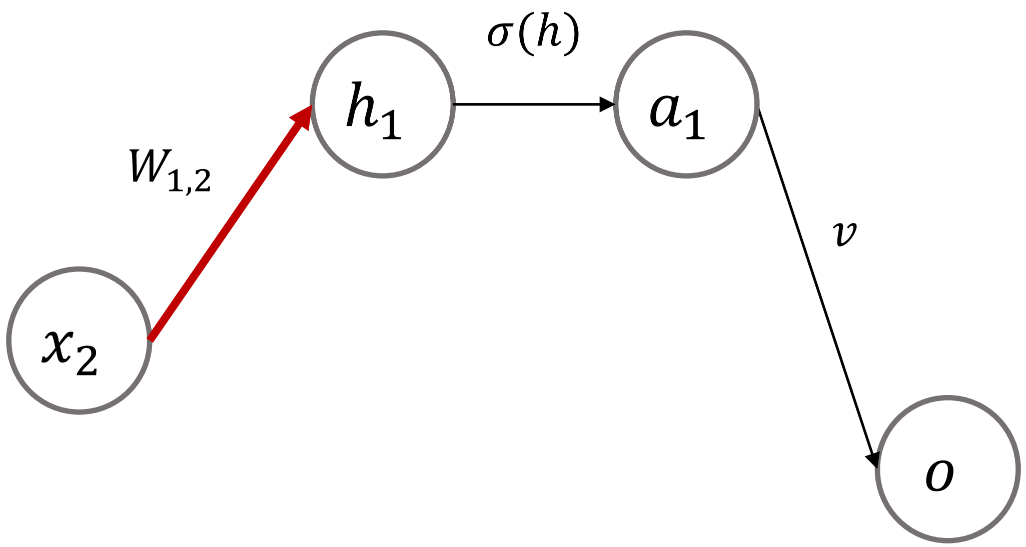
$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial a} \frac{\partial a}{\partial h} \frac{\partial h}{\partial W_{ij}}$$

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial a_i} \frac{\partial a_i}{\partial h_i} \frac{\partial h_i}{\partial W_{ij}}$$



Two Layers MLP

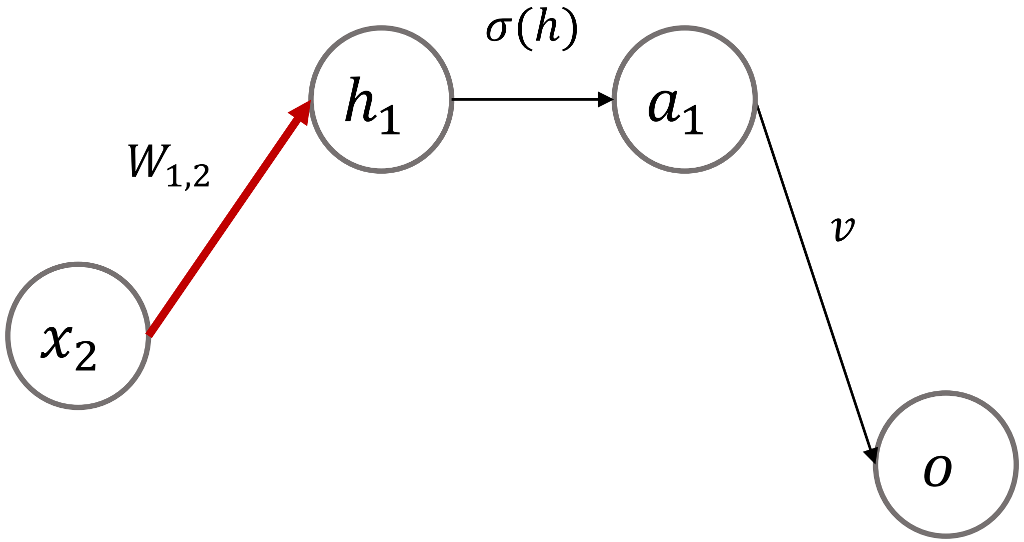
$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a_i}\frac{\partial a_i}{\partial h_i}\frac{\partial h_i}{\partial W_{ij}}$$

$$\frac{\partial L}{\partial W_{ij}} = (o - y)v_i\left(\sigma(h_i)\big(1 - \sigma(h_i)\big)\right)x_j$$

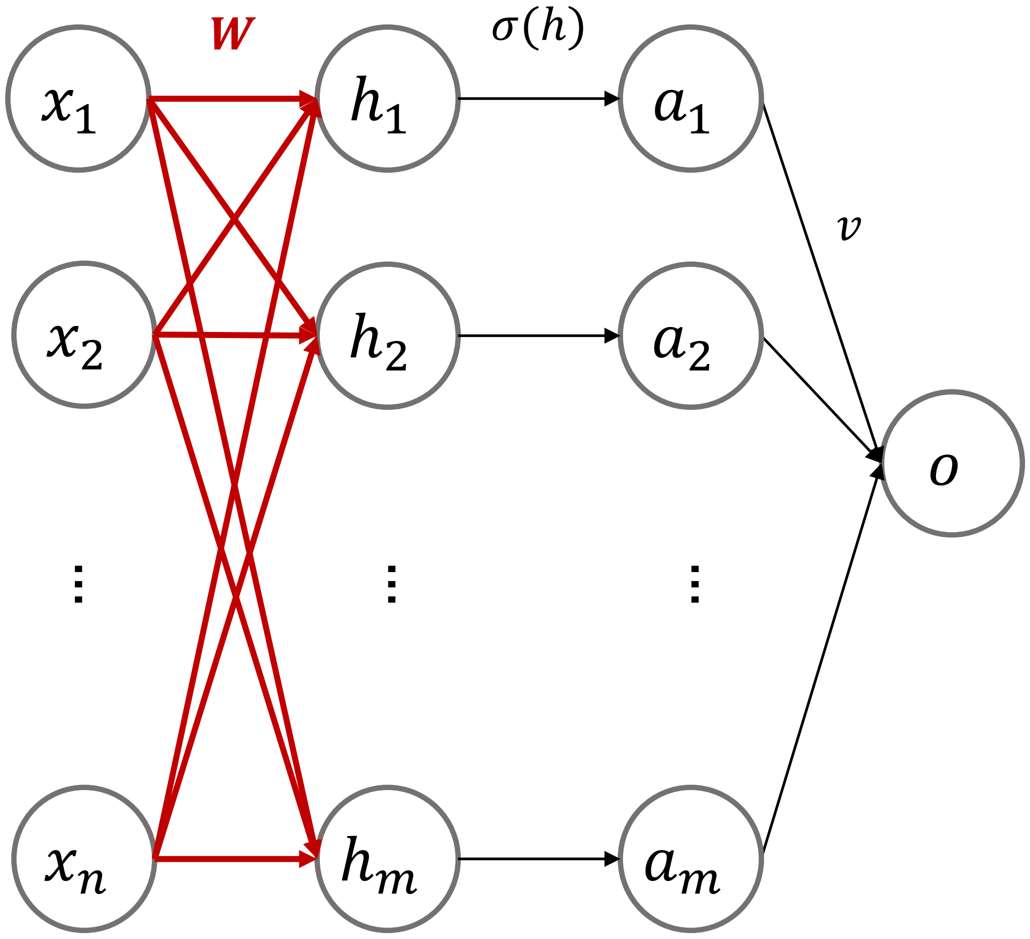


Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$\frac{\partial L}{\partial W} = (o-y)\begin{bmatrix} \sigma(h_1)\big(1-\sigma(h_1)\big)v_1 x_1 & \cdots & \sigma(h_1)\big(1-\sigma(h_1)\big)v_1 x_n \\ \vdots & \ddots & \vdots \\ \sigma(h_m)\big(1-\sigma(h_m)\big)v_m x_1 & \cdots & \sigma(h_m)\big(1-\sigma(h_m)\big)v_m x_n \end{bmatrix}$$

$$\underbrace{\frac{\partial L}{\partial W}}_{m \times n} = \underbrace{(o-y)\Big(v \odot \sigma(h)\big(1-\sigma(h)\big)\Big)x^\top}_{m \times n}$$



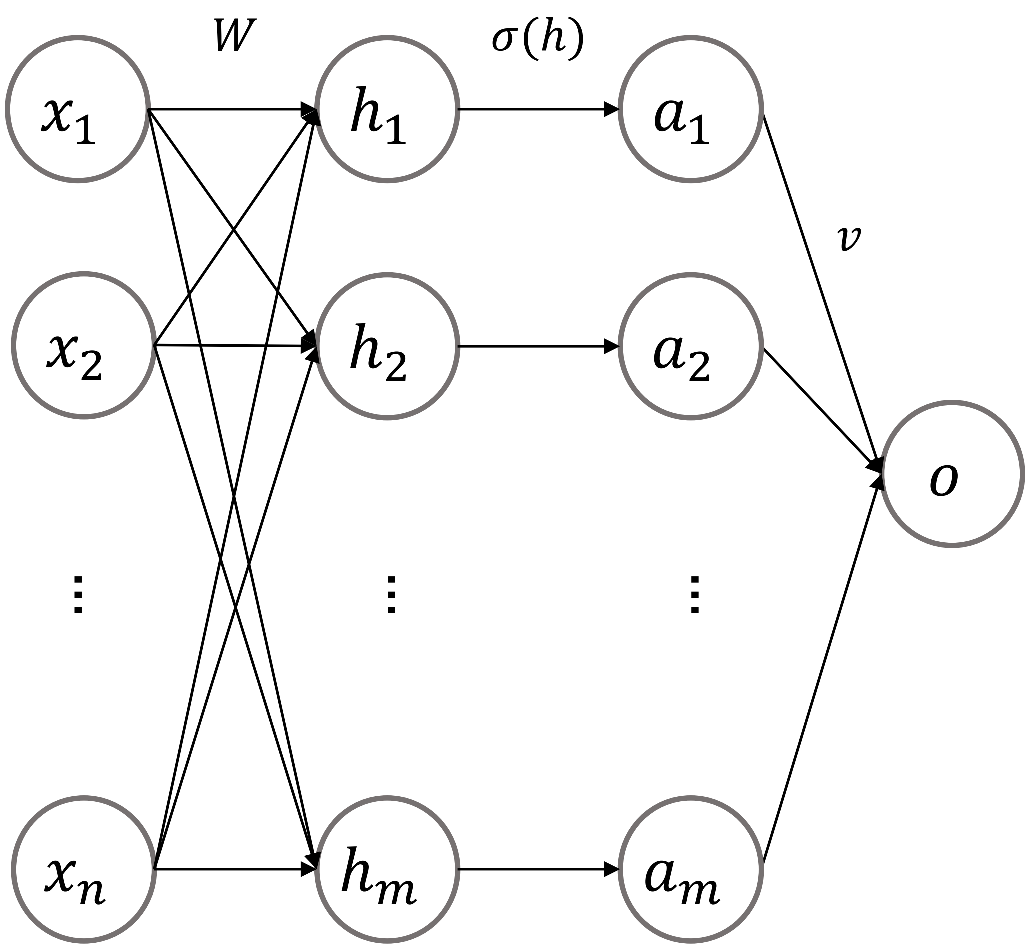
Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \quad a = \sigma(h) \quad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial x}$$



Two Layers MLP

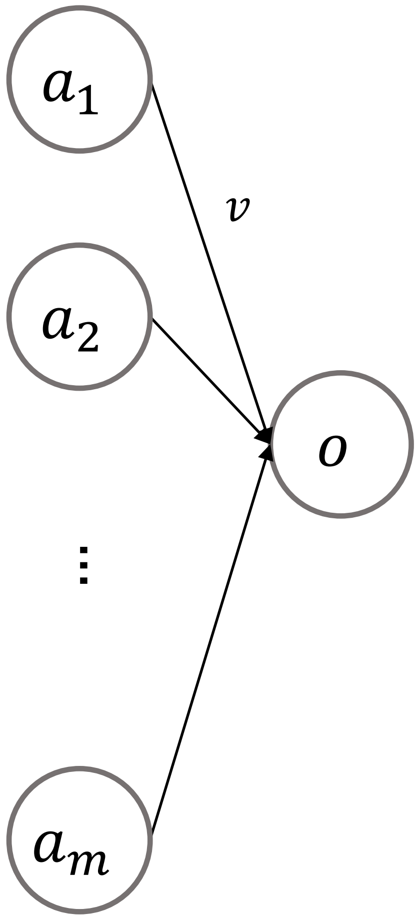
$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a_i} = (o - y)v_i$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a} = (o - y)v$$



Two Layers MLP

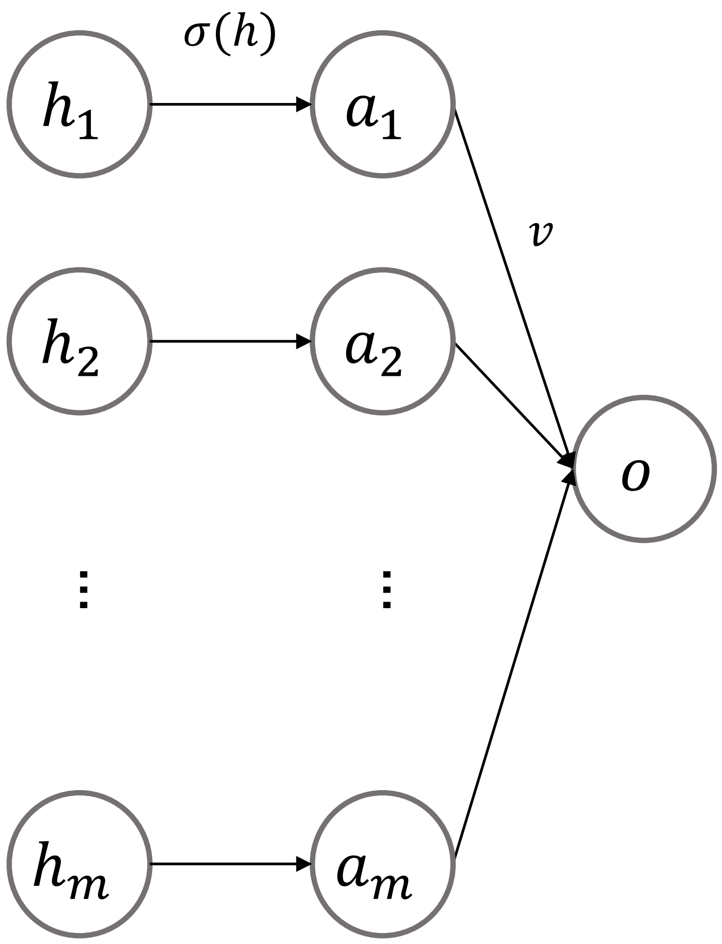
$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial a} \frac{\partial a}{\partial h}$$

$$= (o - y)\Big(v \odot \sigma(h)\big(1 - \sigma(h)\big)\Big)$$



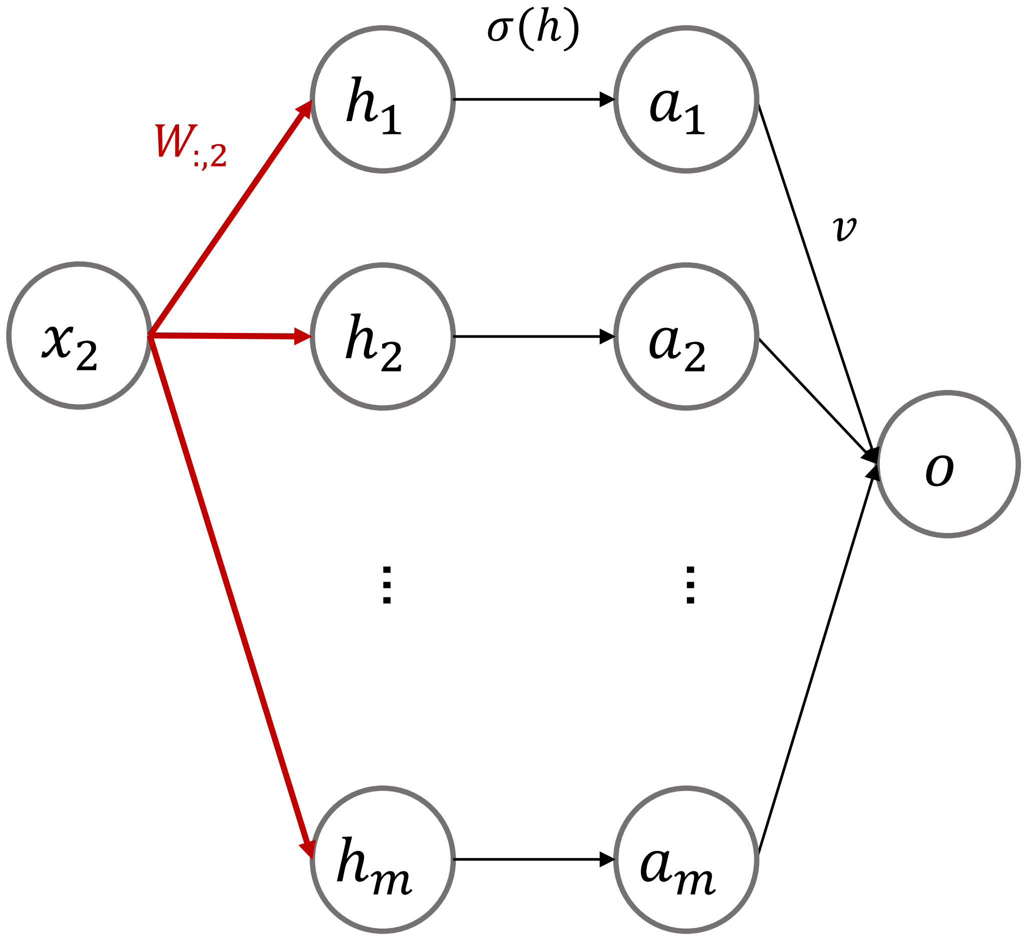
Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial x_i}$$



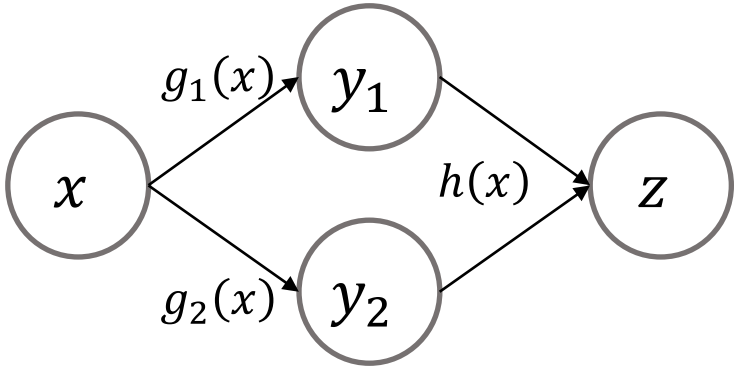
The Multi-variable Chain Rule

- Multi-variable chain rule

$$f, g_1, g_2 : \mathbb{R} \to \mathbb{R}, \qquad h : \mathbb{R}^2 \to \mathbb{R}$$

$$y_1 = g_1(x), \qquad y_2 = g_2(y)$$

$$z = h(y_1, y_2)$$



$$\frac{dz}{dx} = \frac{dz}{dy_1}\frac{dy_1}{dx} + \frac{dz}{dy_2}\frac{dy_2}{dx}$$ (Total derivative)

The Multi-variable Chain Rule

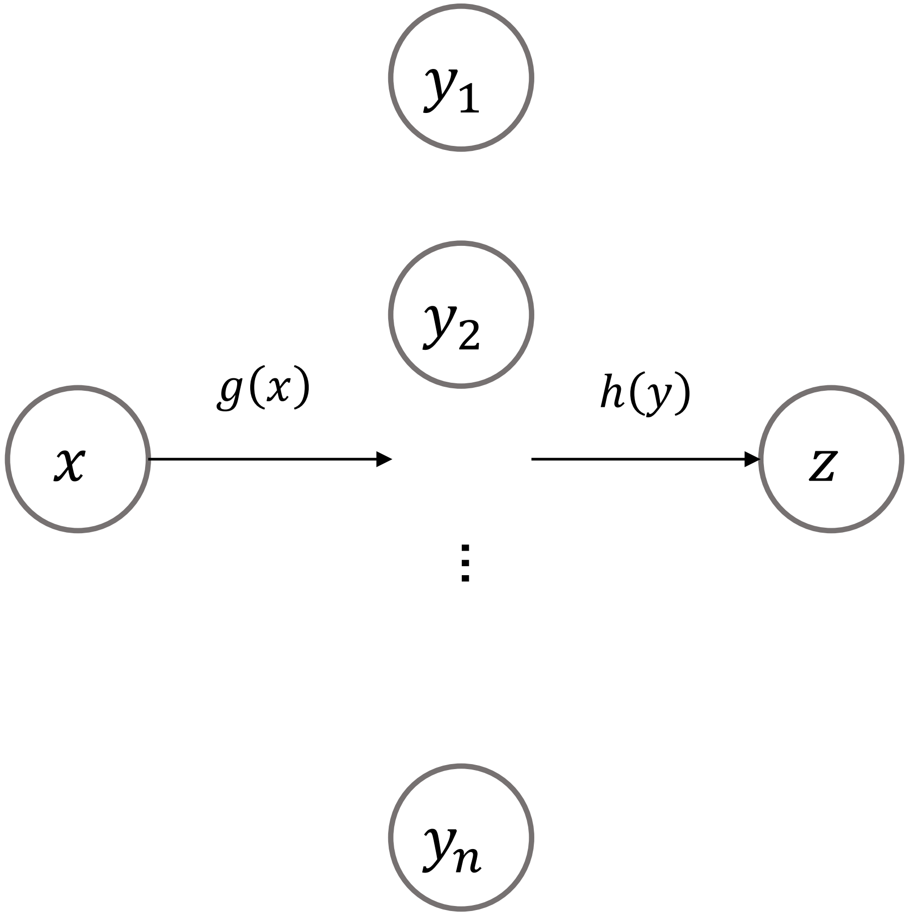
- Multi-variable chain rule

$$x \in \mathbb{R}, y \in \mathbb{R}^n, z \in \mathbb{R}$$

$$g: \mathbb{R} \to \mathbb{R}^n, \qquad y = g(x)$$

$$h: \mathbb{R}^n \to \mathbb{R}, \qquad z = h(y)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$



Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m \times n}, v \in \mathbb{R}^m$

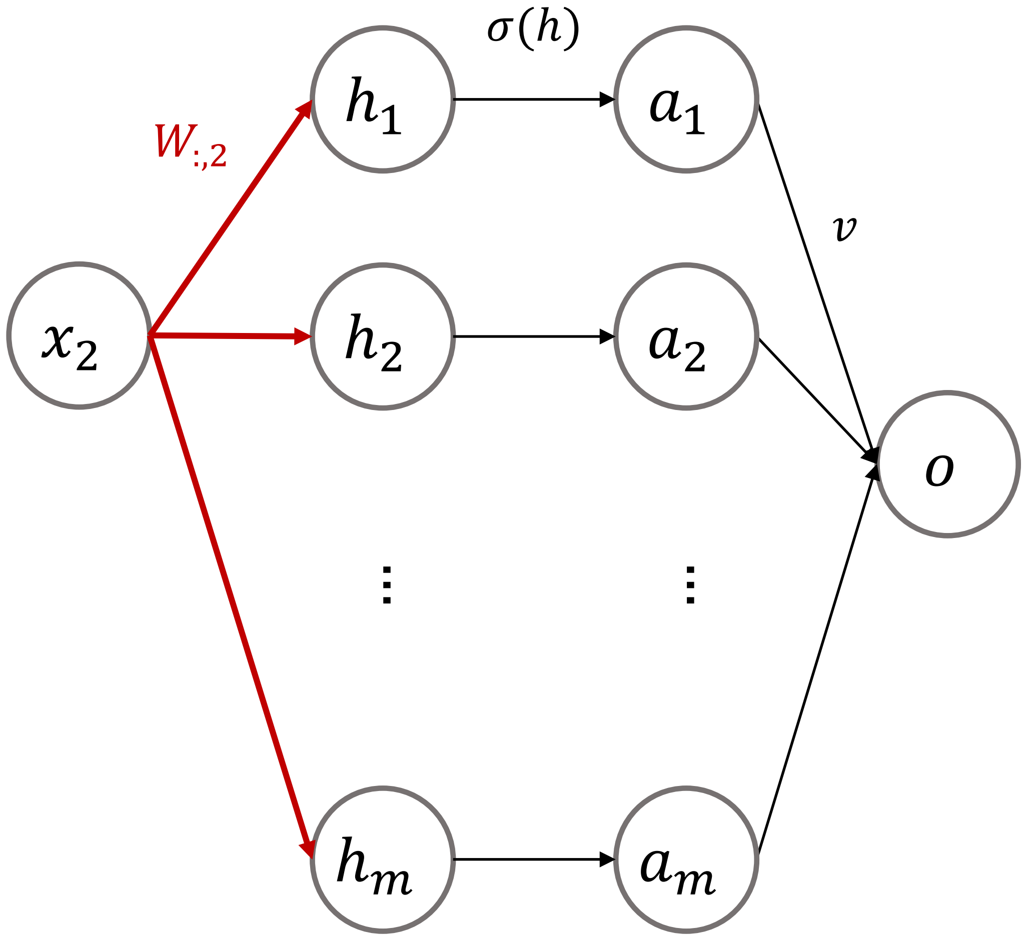
$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W, v) = \frac{1}{2}(y - o)^2$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial x_i}$$

$W_{:,i} \in \mathbb{R}^m$

$$= (o - y)\left(v \odot \sigma(h)(1 - \sigma(h))\right)^\top W_{:,i}$$



Two Layers MLP

$x \in \mathbb{R}^n, y \in \mathbb{R}, h \in \mathbb{R}^m, a \in [0,1]^m, o \in \mathbb{R}$
$W \in \mathbb{R}^{m\times n}, v \in \mathbb{R}^m$

$$h = Wx \qquad a = \sigma(h) \qquad o = v^\top a$$

$$L(W,v) = \frac{1}{2}(y-o)^2$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial x}$$

$$= (o-y)\Big(v \odot \sigma(h)\big(1-\sigma(h)\big)\Big)^\top W$$