Deep Learning

- Linear Models and Multi-Layer Perceptron-

Eunbyung Park

Assistant Professor

School of Electronic and Electrical Engineering

Eunbyung Park (silverbottlep.github.io)

Linear Regression

Linear Models

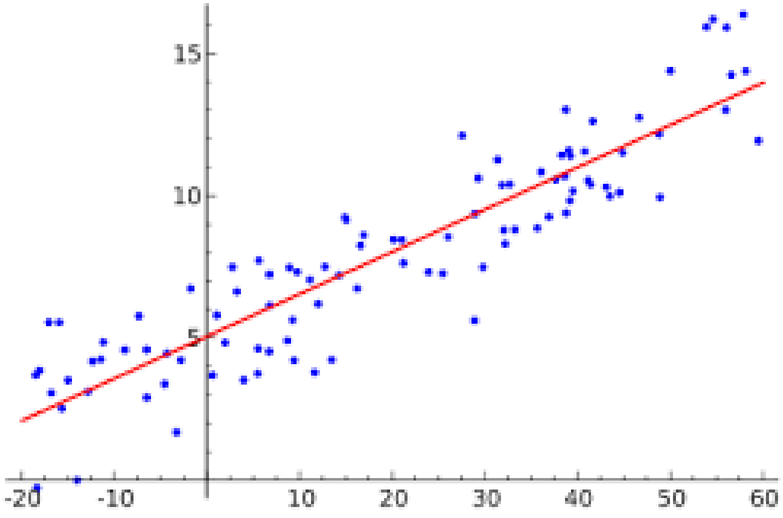
Linear Models

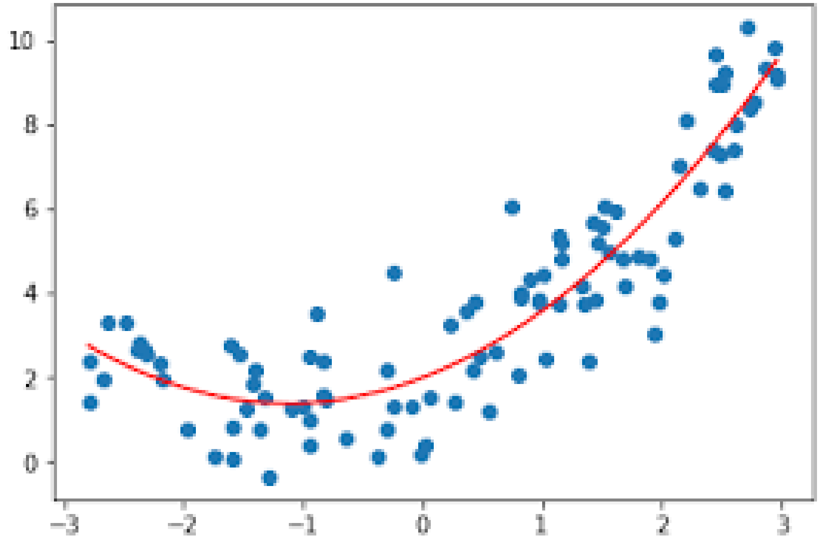
Linear (comes from 'Line')

Mathematical models

Regression

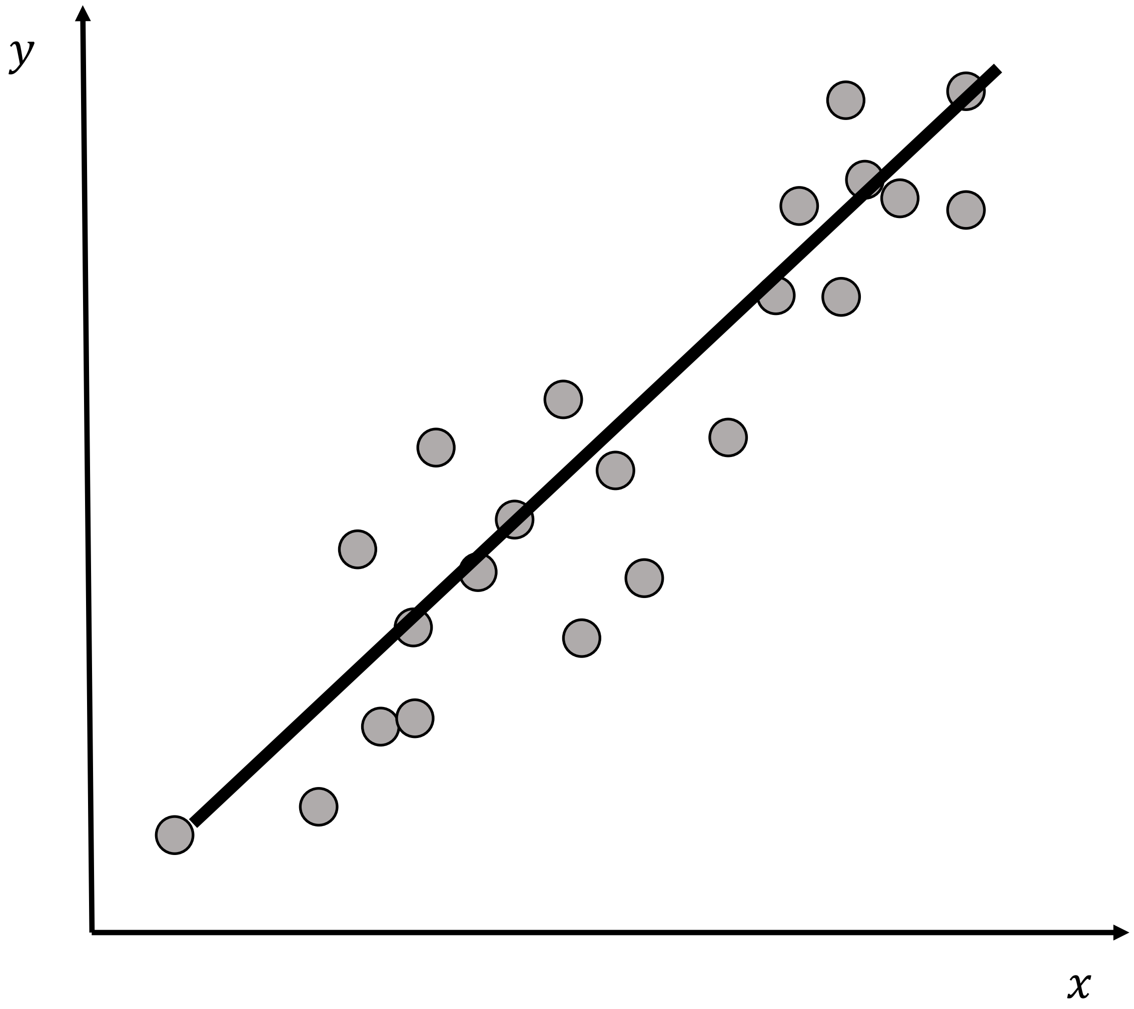
- Regression is a (statistical) method of fitting curves through data points
- The term "regression" was coined by Francis Galton in the 19th century to describe a biological phenomenon.
  - The taller the parents, the taller the children, but shorter than their parents
  - The shorter the parents, the shorter the children, but taller than their parents
  - "regression to the mean"





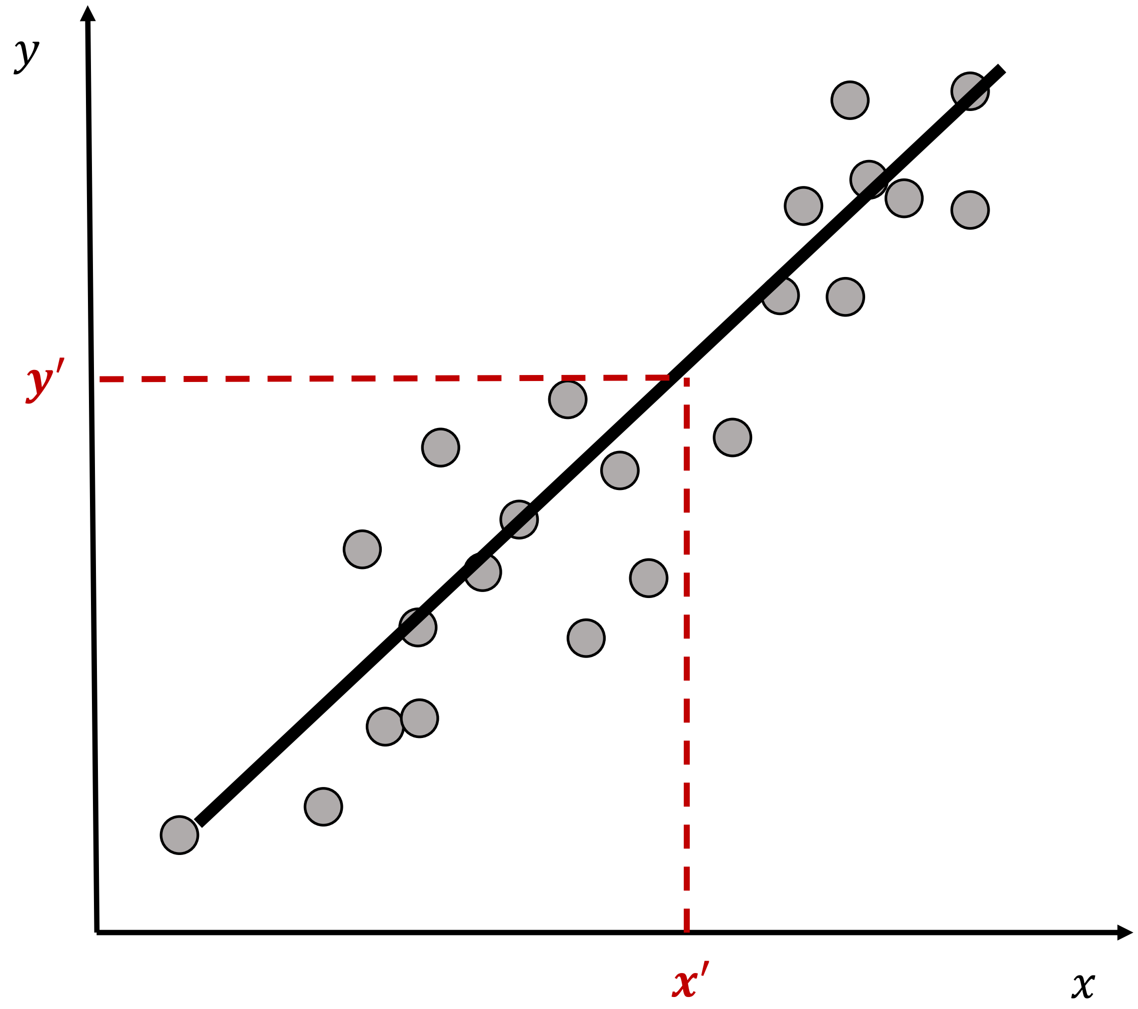
[JMPer Cable Summer 98: Why is it called Regression? (jhsph.edu)]

1D Linear Regression



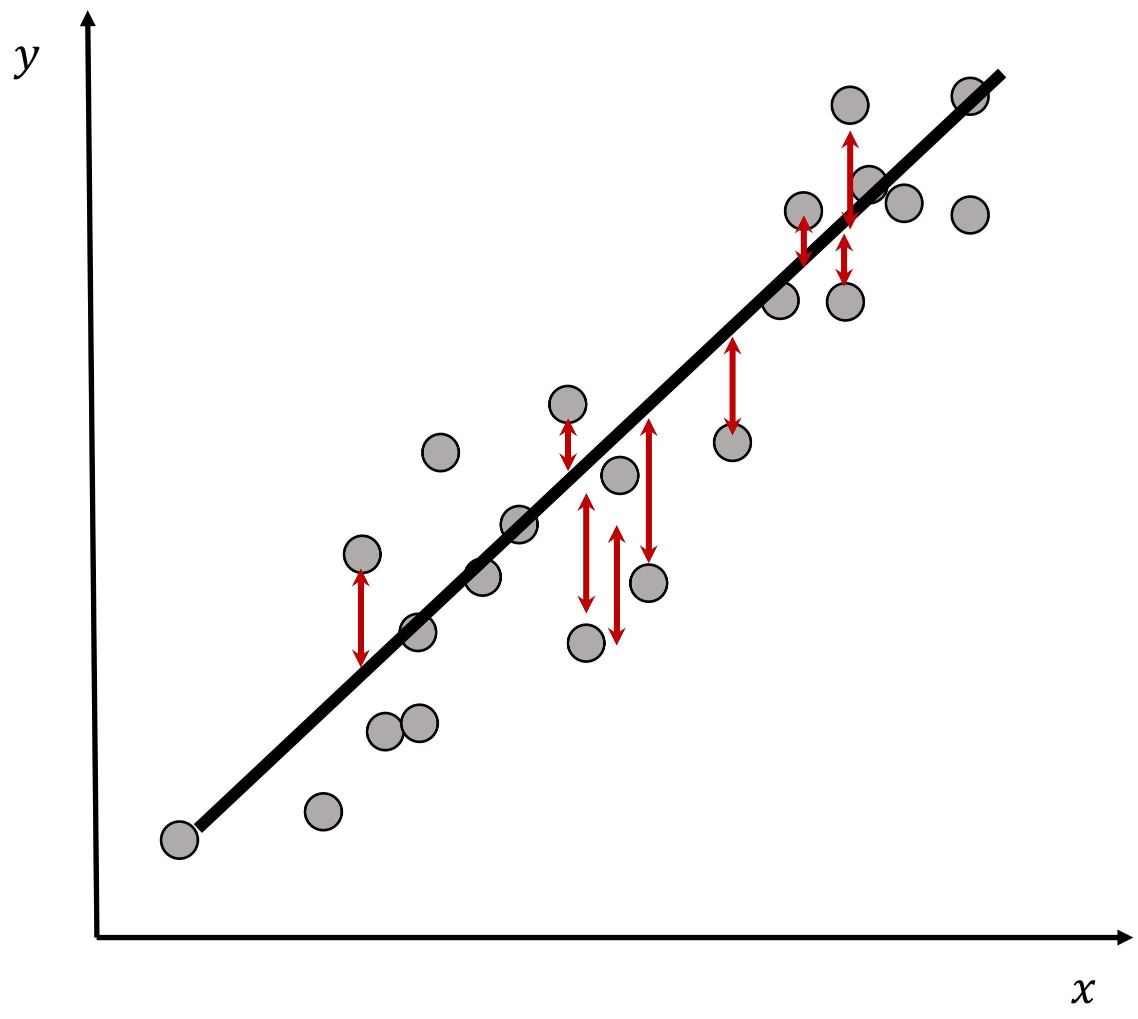
- Fitting a *line* that explains the data

1D Linear Regression



- Fitting a *line* that explains the data
- Given a new data $\boldsymbol{x'}$, predict $\boldsymbol{y'}$

1D Linear Regression



- Fitting a line that explains the data $\{(x^{(i)}, y^{(i)})\}$

$$f(\mathrm{x}) = wx$$

- What is the best line?
  - A line that is close to all data points 'on average'
  - Mean squared error (MSE) loss

$$w^* = \arg\min_w \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - wx^{(i)}\right)^2$$

1D Linear Regression

$$L(w) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - wx^{(i)}\right)^2$$

$$w^* = \arg\min_{w} L(w)$$

- The least squares method
  - L2 Loss function
- N and $\{(x^{(i)}, y^{(i)})\}$ are constants (given), and only w is 'unknown'
- We are going to find w that minimizes the loss function $L(w)$
- Then, how?

1D Linear Regression

$$L(w) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - wx^{(i)}\right)^2 = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)}\right)^2 + w^2\left(x^{(i)}\right)^2 - 2wx^{(i)}y^{(i)}$$

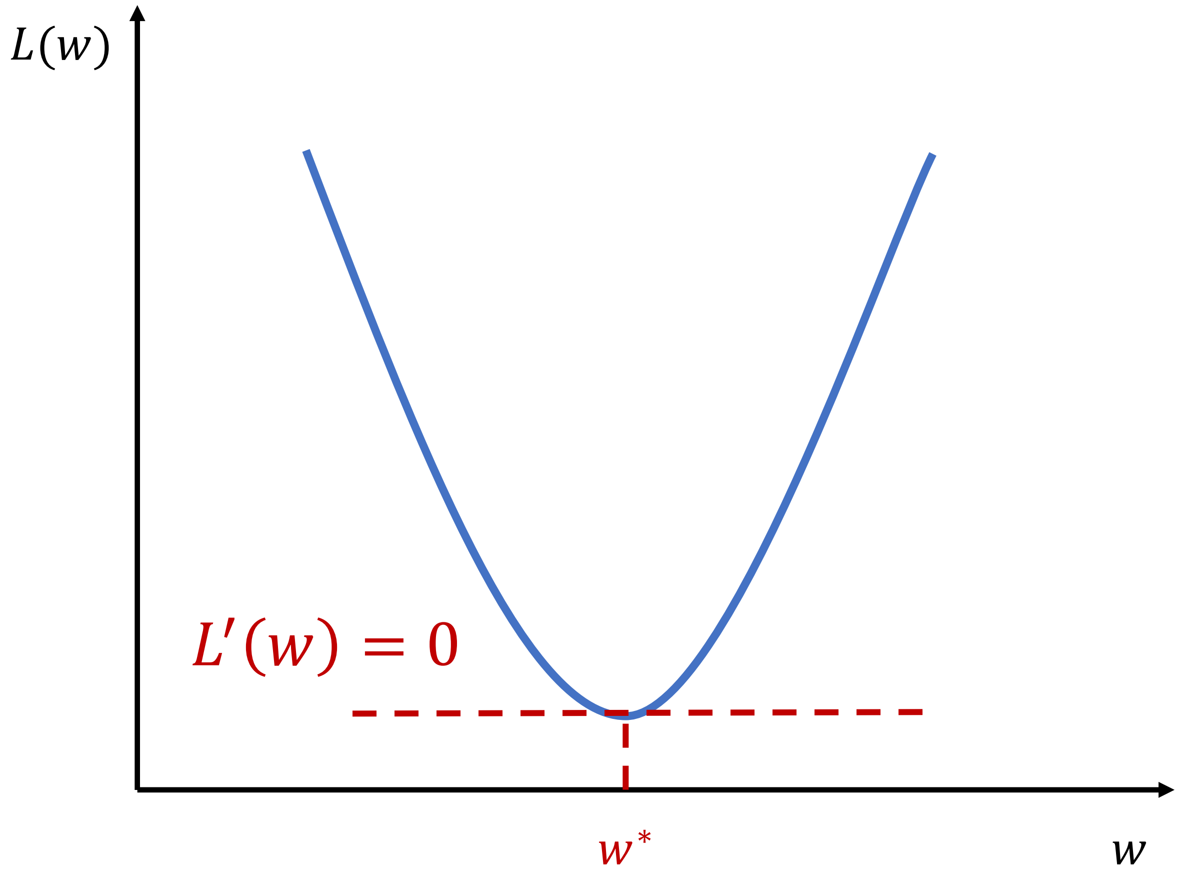
$$= \frac{1}{2}\left(\sum_{i=1}^{N}\left(x^{(i)}\right)^2\right)w^2 + \left(\sum_{i=1}^{N} x_i y^{(i)}\right)w + \frac{1}{2}\left(\sum_{i=1}^{N}\left(y^{(i)}\right)^2\right)$$

$L(w)$ is a quadradic function

How to minimize a quadratic function?

1D Linear Regression

- Minimizing a quadratic function
  - Take a derivative, and set it to zero



Does it have a solution?

If so, is it an unique solution?

1D Linear Regression

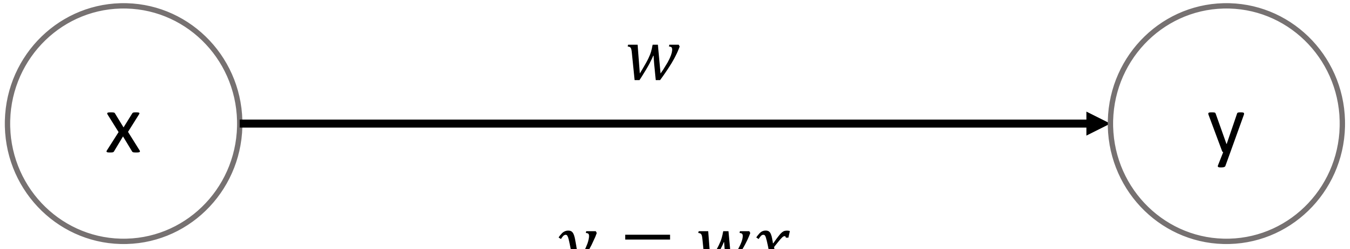
$$L(w) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - wx^{(i)}\right)^2$$

$$L'(w) = \frac{dL(w)}{dw} = \sum_{i=1}^{N}\left(y^{(i)} - wx^{(i)}\right)x^{(i)} = 0$$

$$w^* = \frac{\sum_1^N x^{(i)}y^{(i)}}{\sum_1^N (x^{(i)})^2}$$

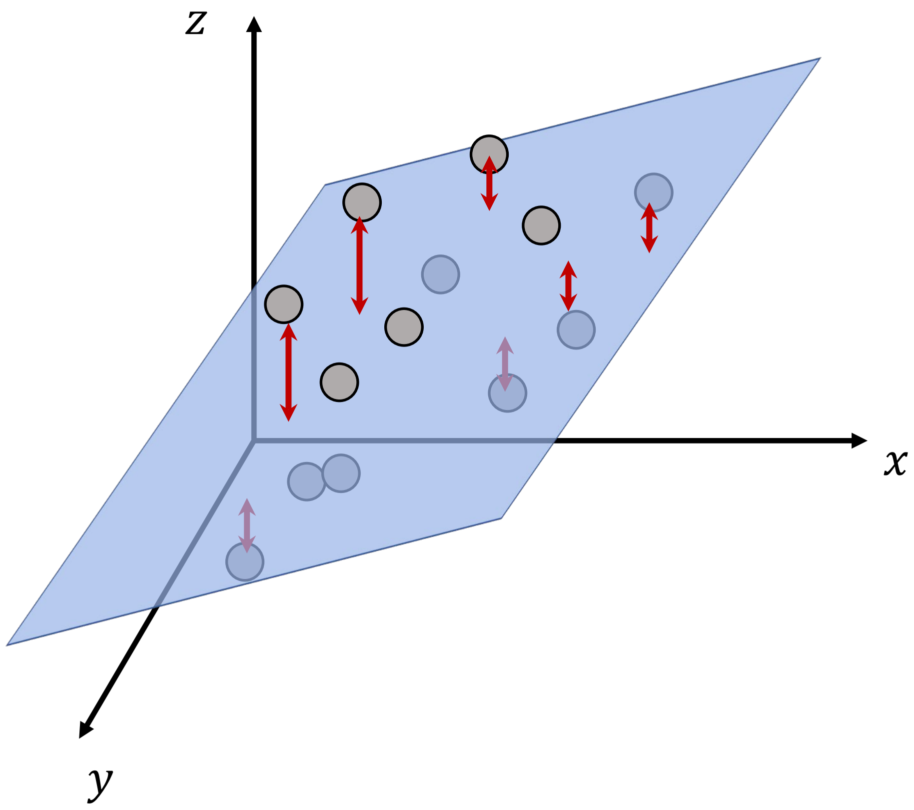
1D Linear Regression

- It is the simplest possible neural network



$$y = wx$$

2D Linear Regression



$$z = w_2 x + w_1 y$$

$$L(w_1, w_2) = \frac{1}{2} \sum_{i=1}^{N} \left(z^{(i)} - w_2 x^{(i)} - w_1 y^{(i)} \right)^2$$

2D Linear Regression

$$z = w_2 x + w_1 y$$

$$L(w_1, w_2) = \frac{1}{2} \sum_{i=1}^{N} \left(z^{(i)} - w_2 x^{(i)} - w_1 y^{(i)}\right)^2$$

$$\frac{\partial L}{\partial w_1} = \sum_{i=1}^{N} \left(z^{(i)} - w_2 x^{(i)} - w_1 y^{(i)}\right)\left(-y^{(i)}\right) = 0$$

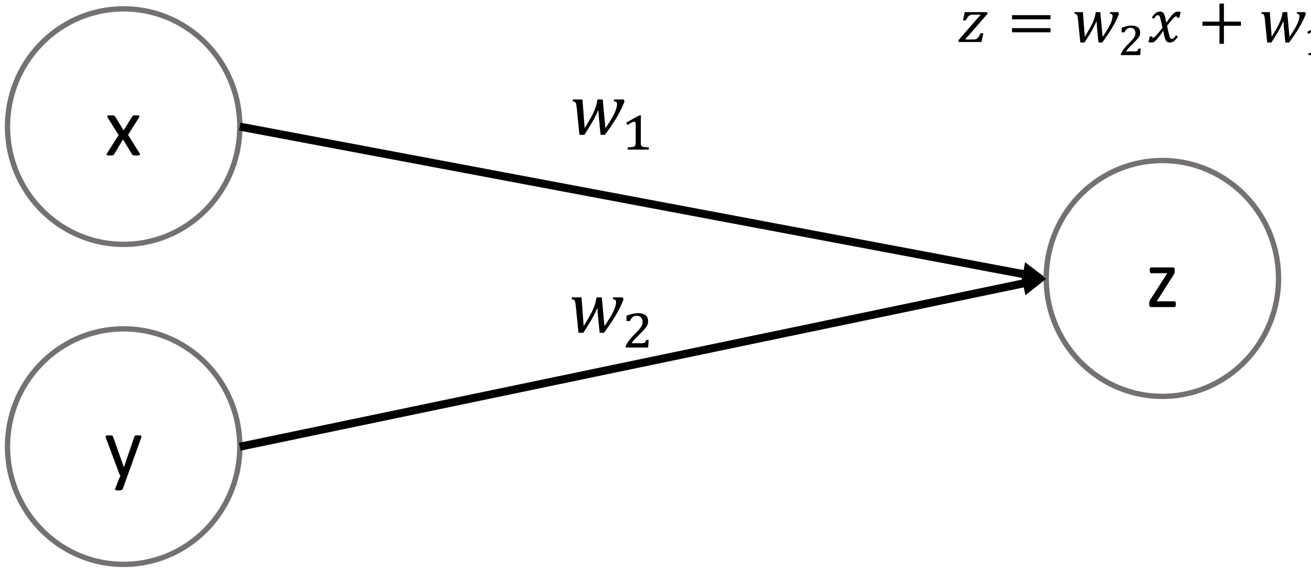
$$\frac{\partial L}{\partial w_2} = \sum_{i=1}^{N} \left(z^{(i)} - w_2 x^{(i)} - w_1 y^{(i)}\right)\left(-x^{(i)}\right) = 0$$

$$w_1 = \frac{\sum_{i=1}^{N} y^{(i)} z^{(i)} - w_2 \sum_{i=1}^{N} z^{(i)} y^{(i)}}{\sum_{i=1}^{N} \left(y^{(i)}\right)^2}$$

$$w_2 = \frac{\sum_{i=1}^{N} x^{(i)} z^{(i)} - w_1 \sum_{i=1}^{N} x^{(i)} y^{(i)}}{\sum_{i=1}^{N} \left(x^{(i)}\right)^2}$$

2D Linear Regression

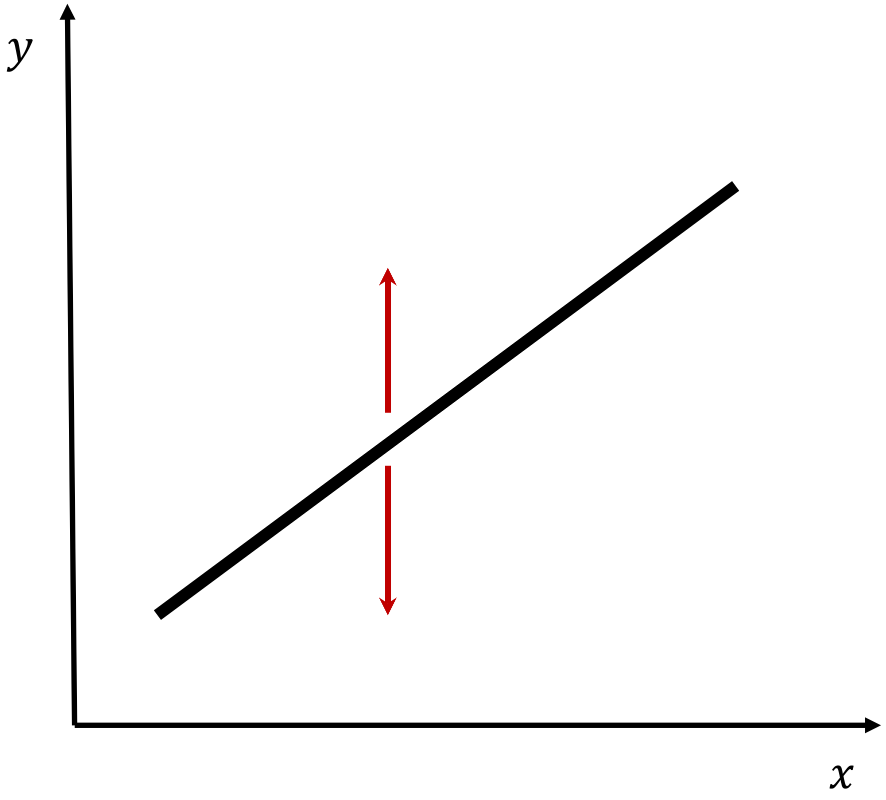
- Single-layer neural network w/ two input units



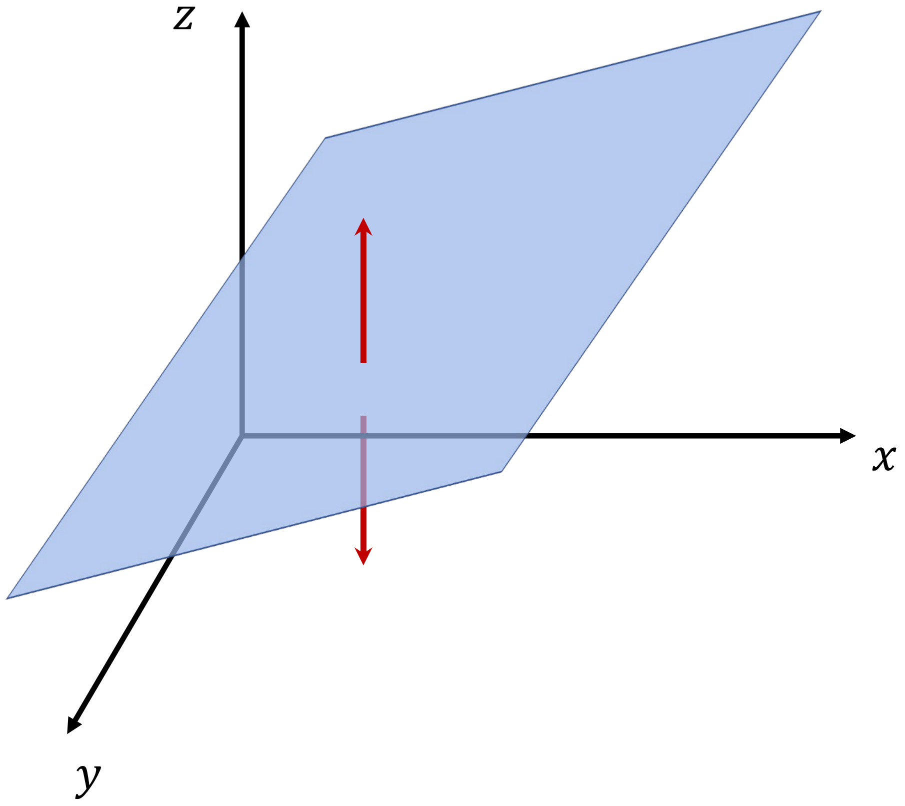
$$z = w_2 x + w_1 y$$

Bias term and Higher Dimension

$$y = w_1 x + w_0$$



$$z = w_2 x + w_1 y + w_0$$



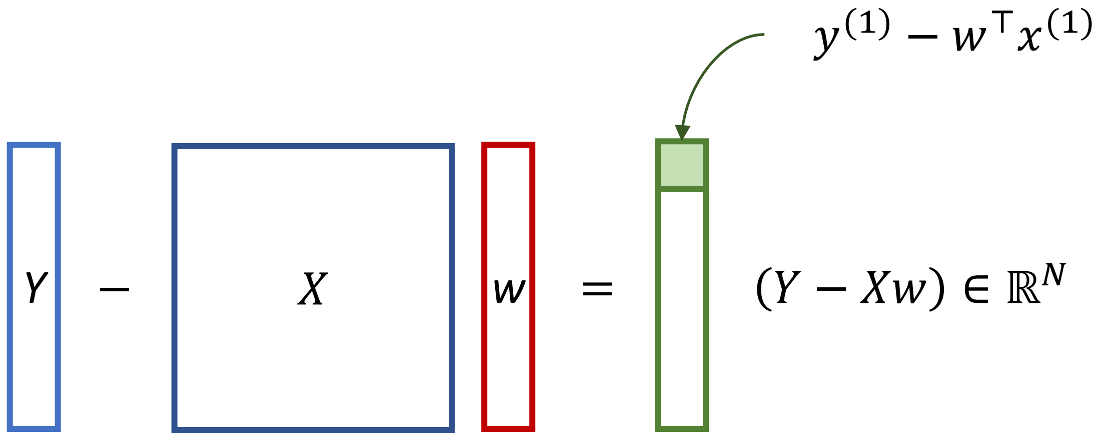
Linear Algebra

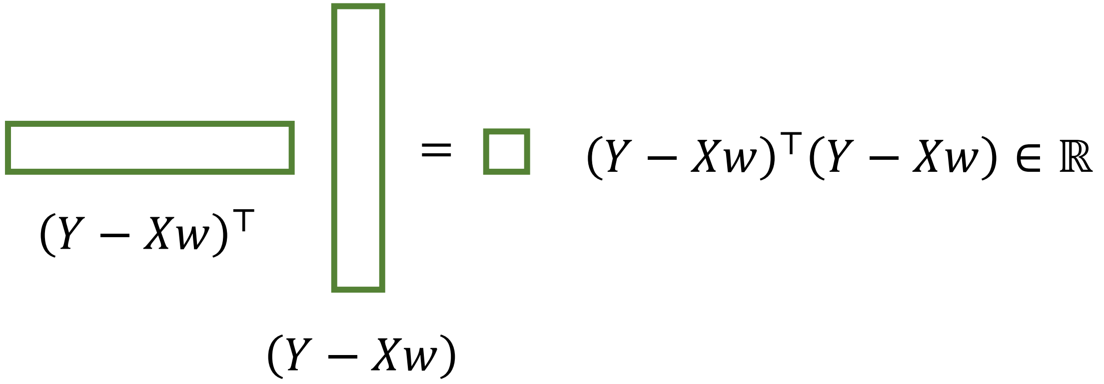
- Linear algebra comes to the rescue!
- Problem setup

$$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$$
$$x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R}, w \in \mathbb{R}^d$$
$$X \in \mathbb{R}^{N\times d}, Y \in \mathbb{R}^N$$

$$L(w) = \frac{1}{2}(Y - Xw)^\top (Y - Xw)$$

$$= \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - w^\top x^{(i)}\right)^2$$





Linear Algebra

- Linear algebra comes to the rescue!
- Problem setup

$$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$$
$$x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R}, w \in \mathbb{R}^d$$
$$X \in \mathbb{R}^{N \times d}, Y \in \mathbb{R}^N$$

$$L(w) = \frac{1}{2}(Y - Xw)^\top (Y - Xw)$$
$$= \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - w^\top x^{(i)}\right)^2$$

$$\arg\min_w L(w)$$

$$L(w) = \frac{1}{2}(Y^\top Y - Y^\top Xw - w^\top X^\top Y + w^\top X^\top Xw)$$
$$= \frac{1}{2}(Y^\top Y - 2Y^\top Xw + w^\top X^\top Xw)$$

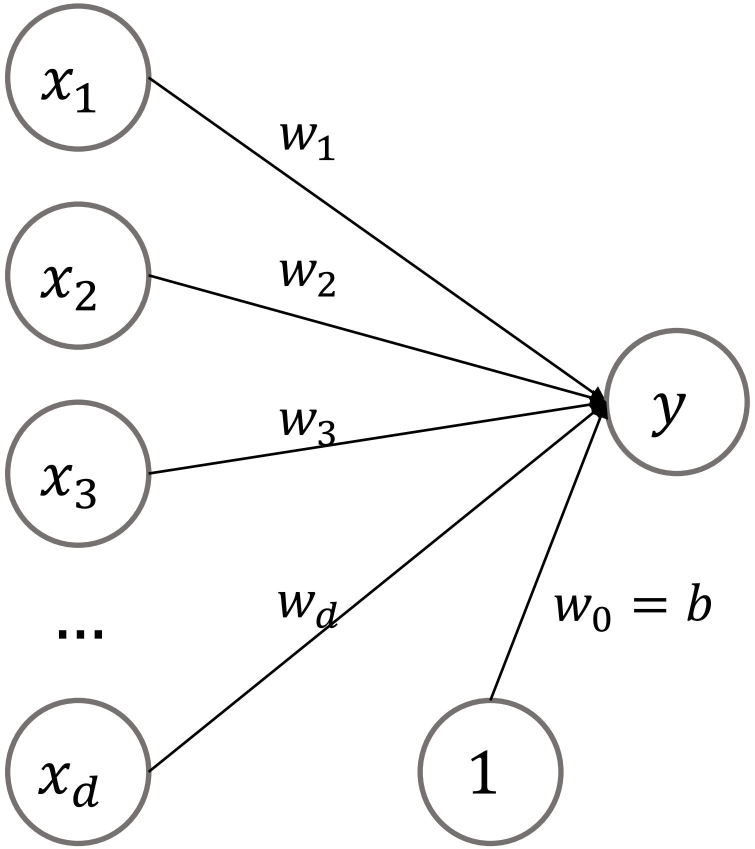
$$\nabla L(w) = -Y^\top X + X^\top Xw = 0$$

$$w^* = (X^\top X)^{-1} Y^\top X = (X^\top X)^{-1} X^\top Y$$

(normal equation)

Shallow Neural Network

- It is a single layer neural network



$$y = w^{\top} x$$

What’s Wrong with It?

$$w^* = (X^\top X)^{-1} X^\top Y$$

Gradient Descent

- For convex optimization, it is guaranteed to converge to global optima

$$L(w) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - w^{\top}x^{(i)}\right)^2 = \frac{1}{2}(Y - Xw)^{\top}(Y - Xw)$$

$$\frac{dL}{dw} = \sum_{i=1}^{N}\left(w^{\top}x^{(i)} - y^{(i)}\right)x^{(i)} = X^{\top}(Xw - Y)$$

$$w \coloneqq w - \alpha\left(\frac{dL}{dw}\right)$$

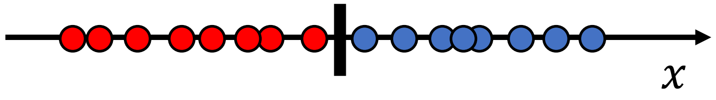
(descent) (step-size) (gradient)

Linear Classification

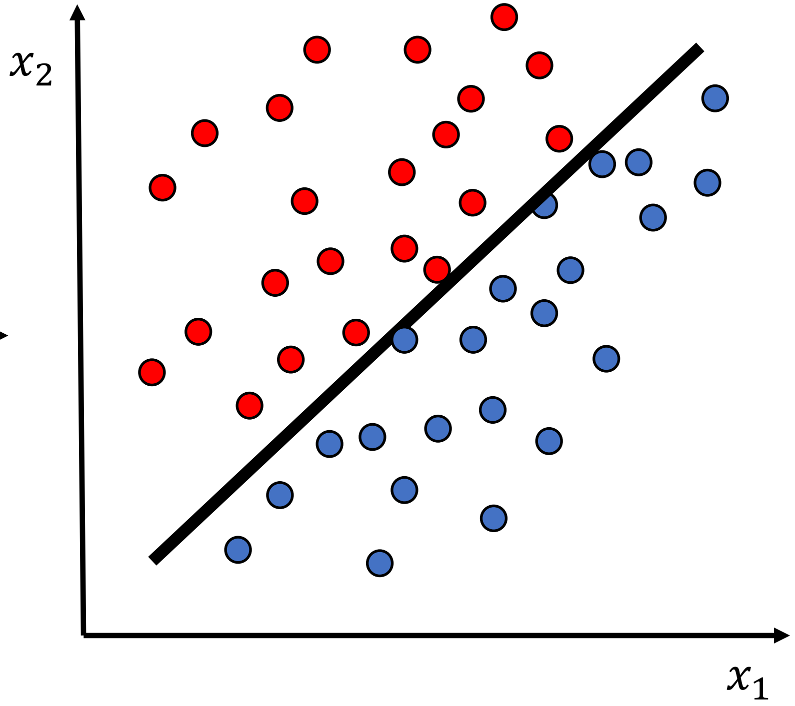
- Linear Models and Multi-Layer Perceptron-

Linear Classification

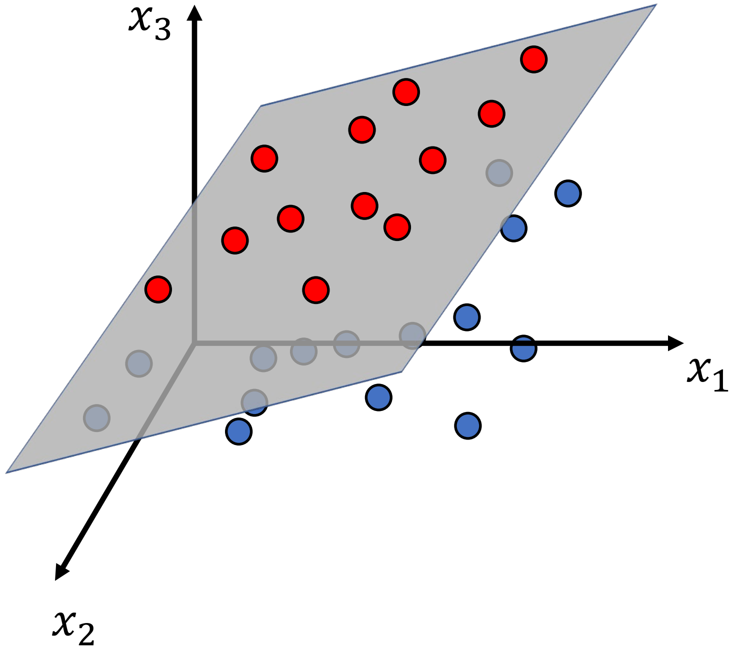
- Linear decision boundary



$x + b = 0$



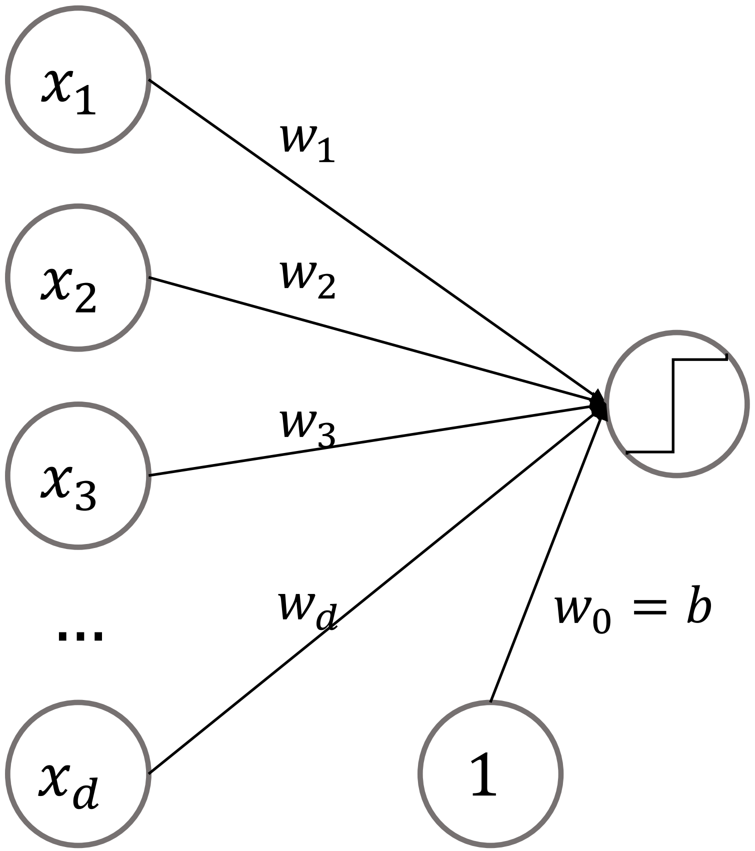
$w_1 x_1 + w_2 x_2 + b = 0$



$w_1 x_1 + w_2 x_2 + w_3 x_3 + b = 0$

Rosenblatt's Perceptron

- A single perceptron as a linear decision boundary (hyperplane)

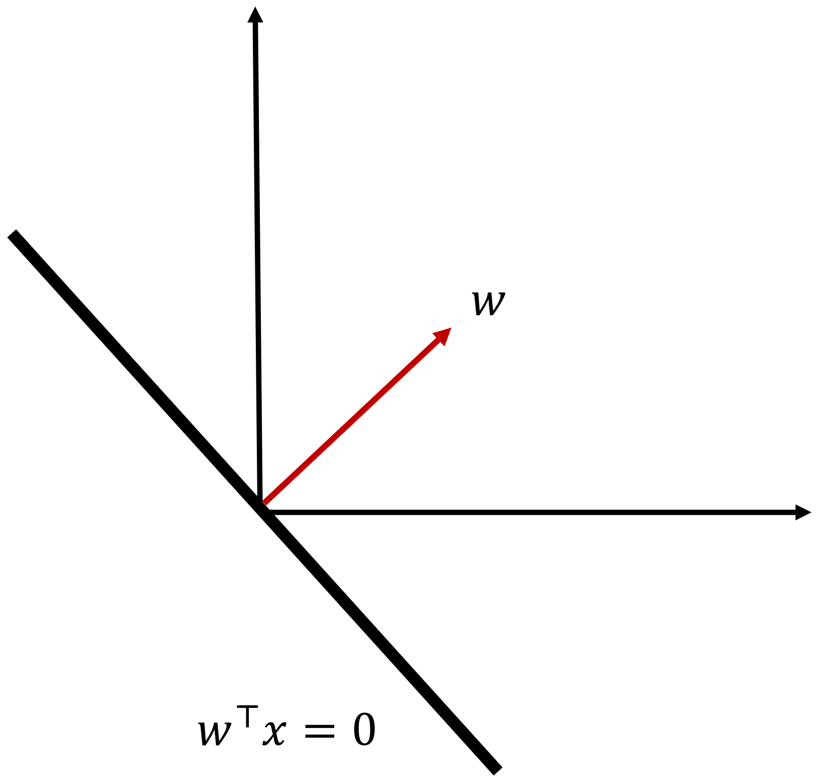


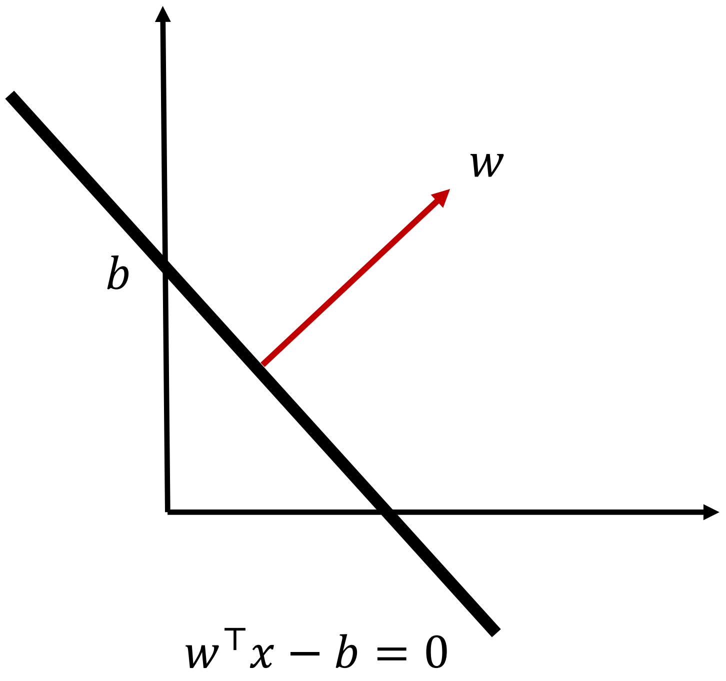
$$f(x) = \begin{cases} 1, & w^\top x \geq 0 \\ 0, & w^\top x < 0 \end{cases}$$

Rosenblatt, Psychological Review 1958

Perceptron

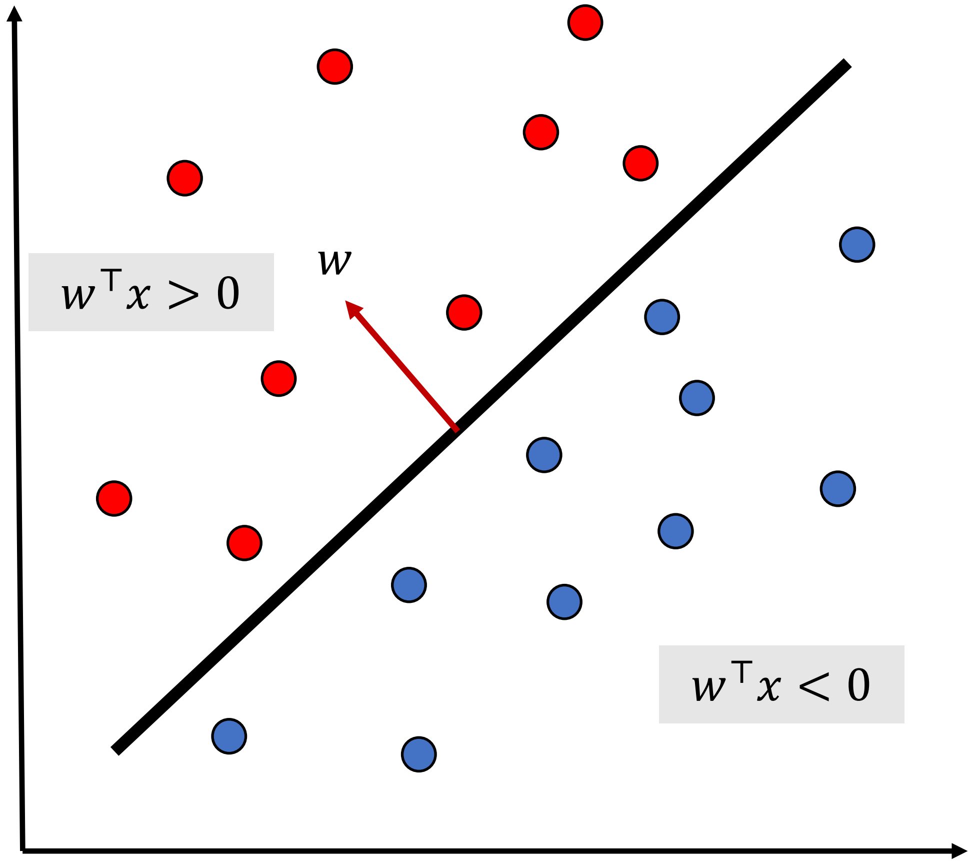
- Weight vector is orthogonal to the hyperplane





Perceptron

- Find a separating hyperplane

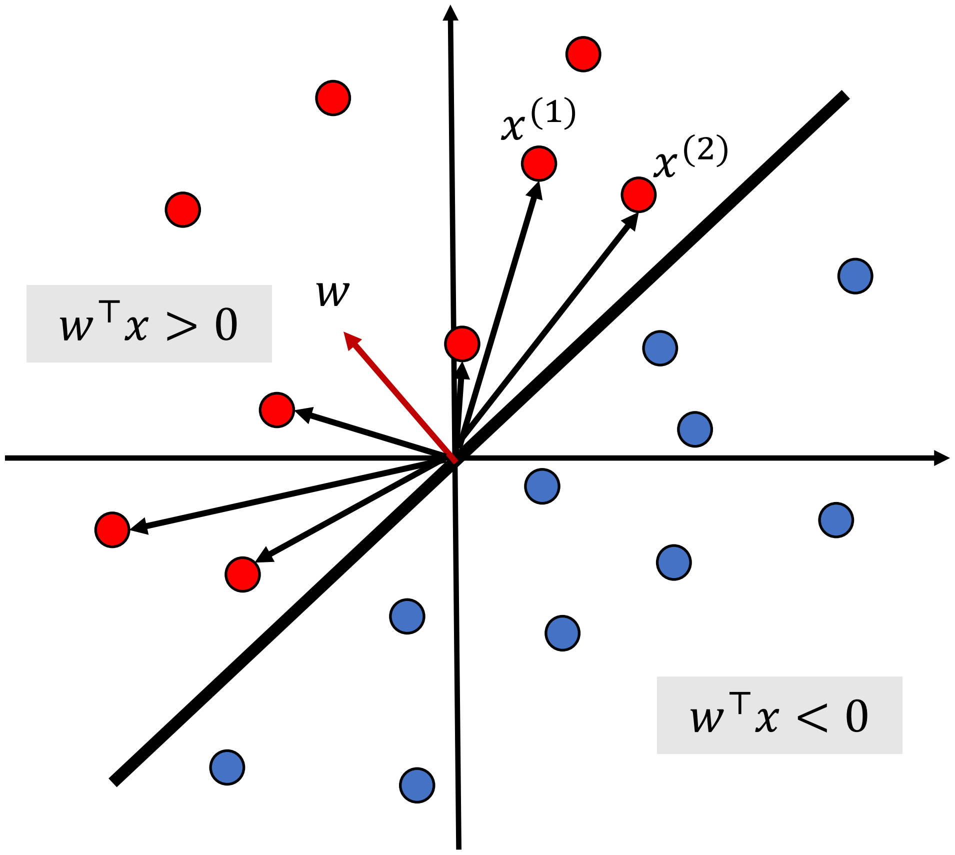


Perceptron

- Find a separating hyperplane

Angles between all positive examples $x^{(i)}$ and w should be less then 90 degree

Angles between all negative examples $x^{(i)}$ and w should be greater then 90 degree



Perceptron Learning Algorithm

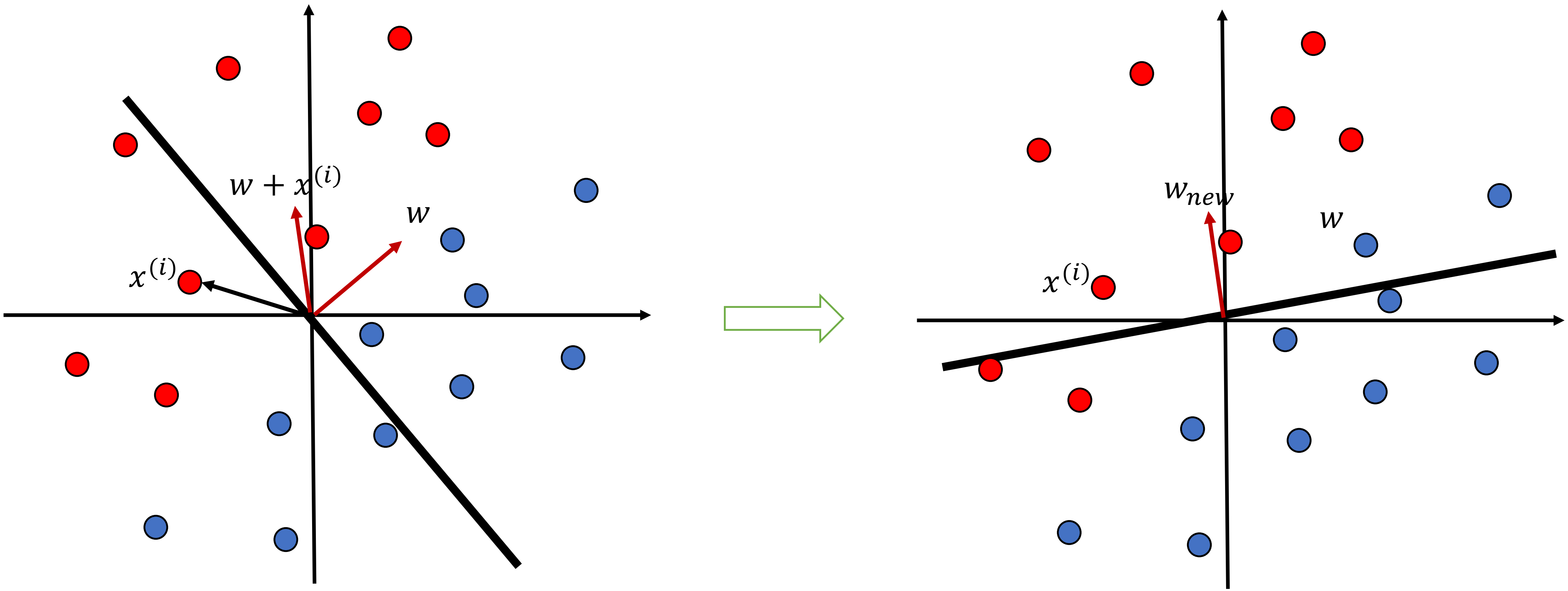
- Find the w vector that perfectly classify training examples

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
        $\boxed{\mathbf{w} = \mathbf{w} + \mathbf{x} \;;}$
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\boxed{\mathbf{w} = \mathbf{w} - \mathbf{x} \;;}$
    end
end
//the algorithm converges when all the inputs are classified correctly

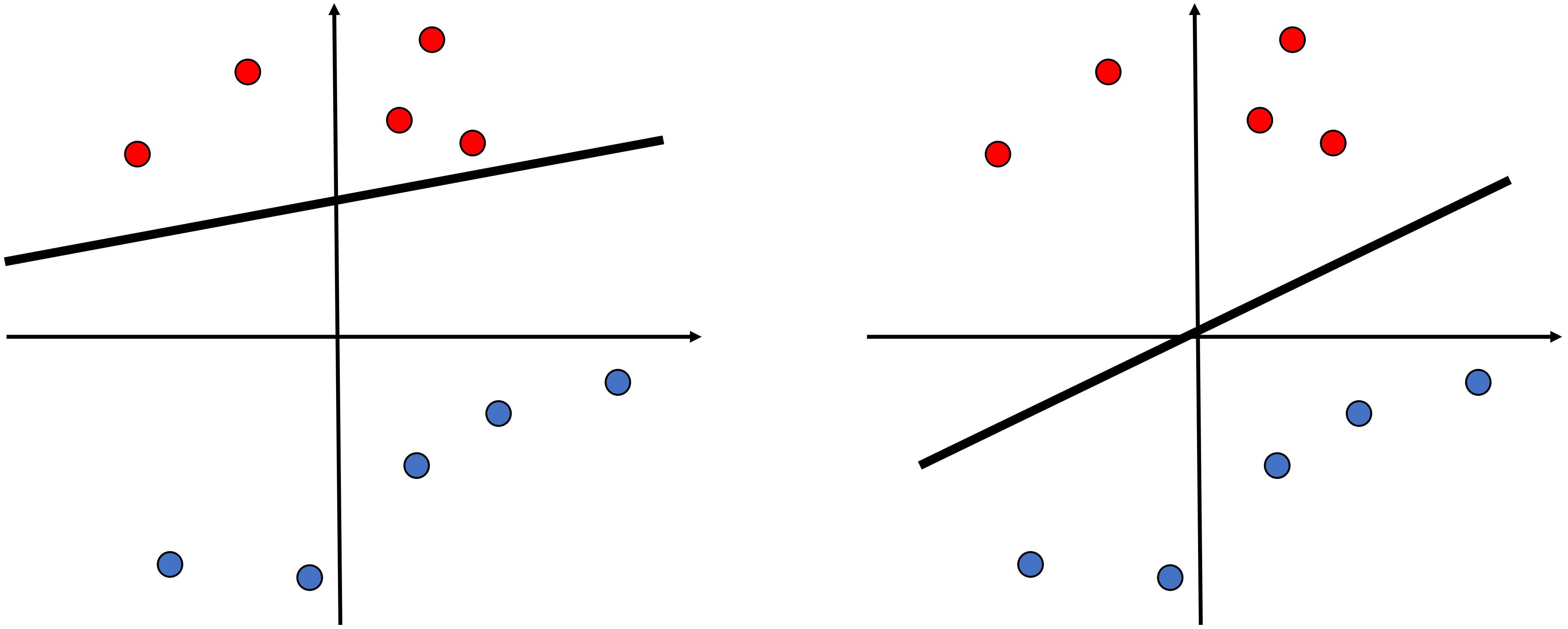
Perceptron Learning Algorithm

- Find a separating hyperplane



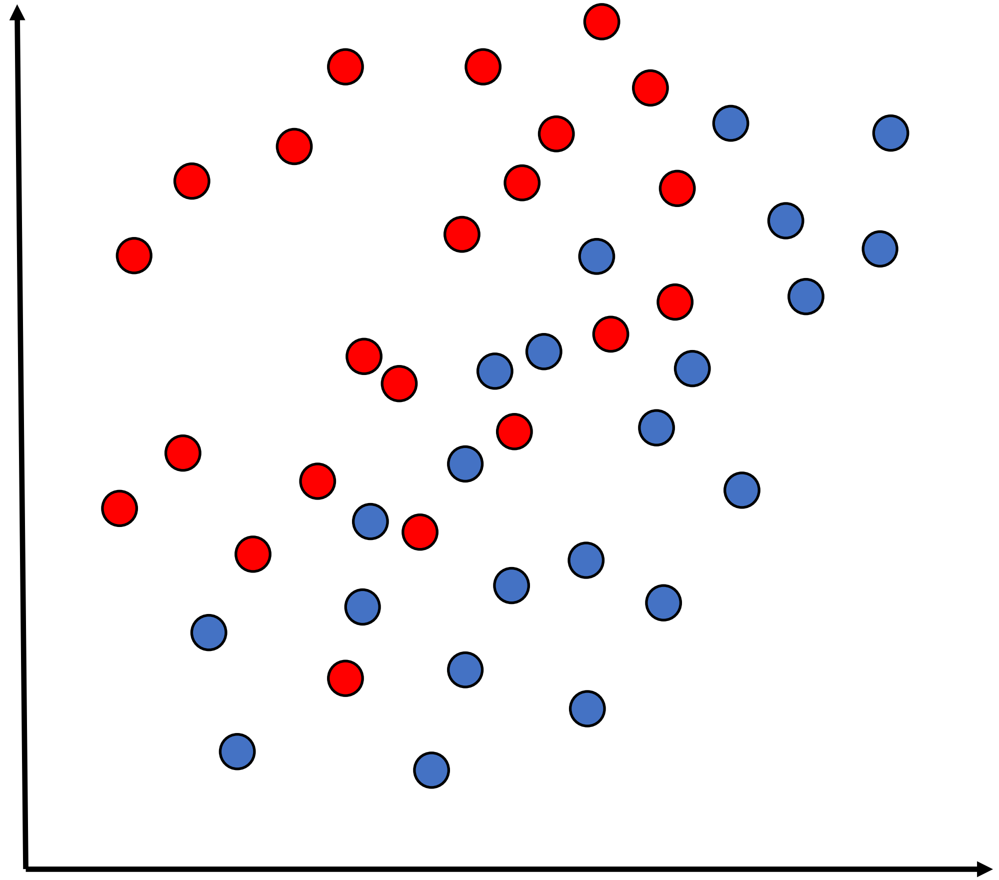
Problems

- Which one is better?



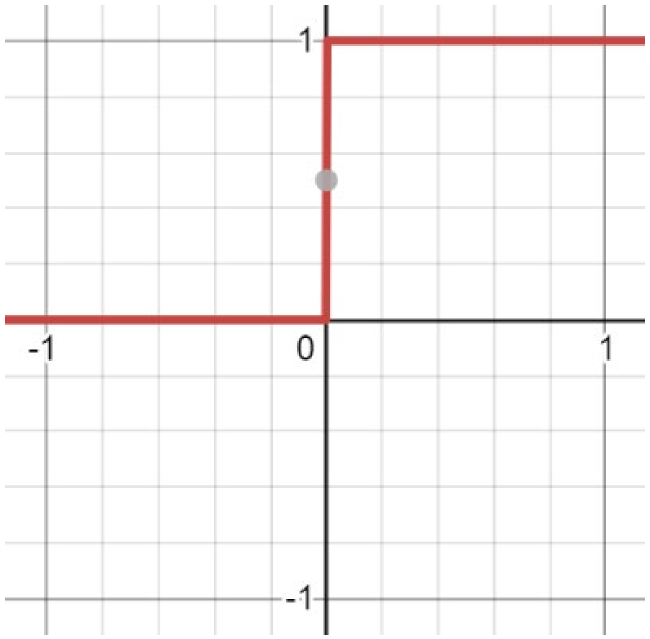
Problems

- What about not linearly separable cases?

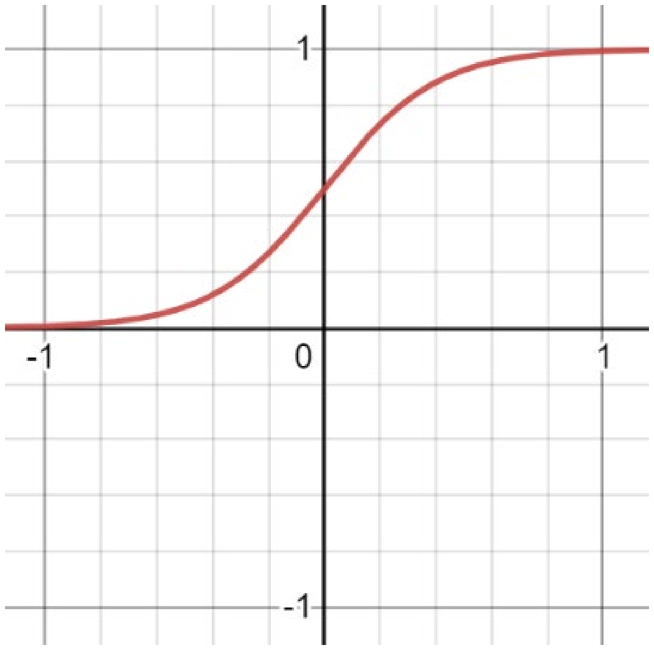


Logistic Function (aka Sigmoid)

- Squeezing the output of a 'linear equation' between 0 and 1



$$\text{step}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

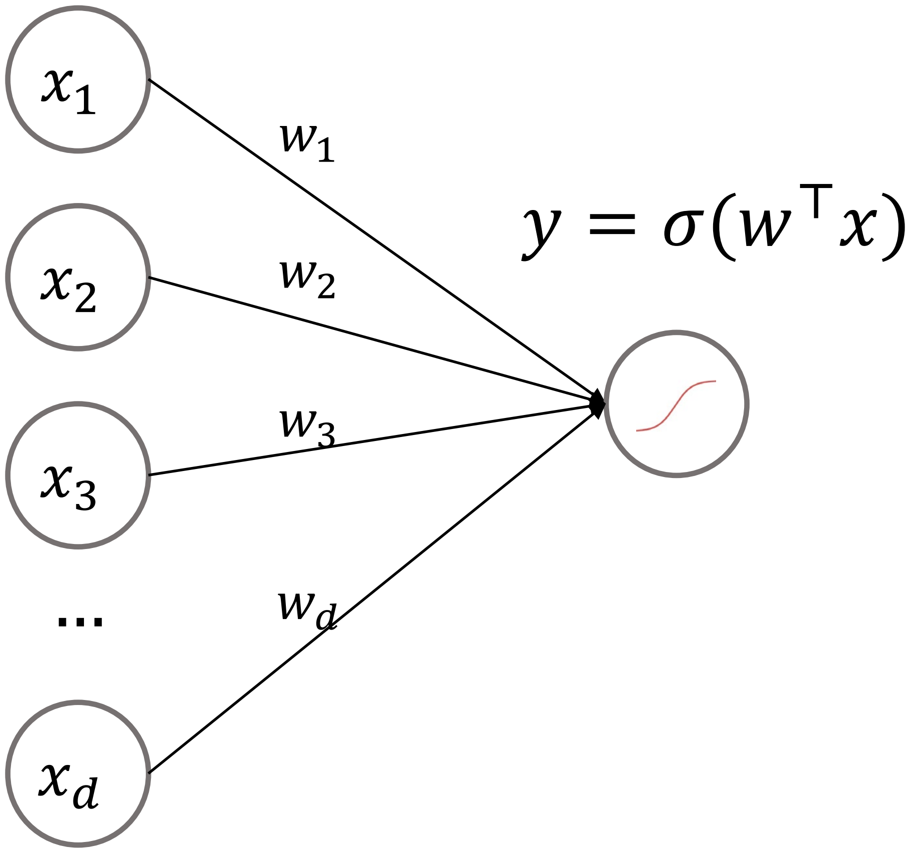


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Logistic Regression

- Using the 'logistic function' to squeeze the output of a 'linear equation'
  - $\sigma(w^\top x) \in [0,1]$ (w/ sigmoid)
  - $\text{step}(w^\top x) \in \{0,1\}$ (thresholding)

- So, now it's more like probability
  - $p(y = 1|x; w) = \sigma(w^\top x)$
  - $p(y = 0|x; w) = 1 - \sigma(w^\top x)$



MSE Loss for Logistic Regression

- Can we apply MSE loss function to logistic regression?

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

$$x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{0,1\}, w \in \mathbb{R}^d$$

$$X \in \mathbb{R}^{N \times d}, Y \in \{0,1\}^N$$

$$\mathrm{MSE}(w) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \sigma(w^\top x^{(i)})\right)^2$$

MSE Loss for Logistic Regression

- Can we apply MSE loss function to logistic regression?

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

$$x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{0,1\}, w \in \mathbb{R}^d$$

$$X \in \mathbb{R}^{N \times d}, Y \in \{0,1\}^N$$

$$\mathrm{MSE}(w) = \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - \sigma(w^\top x^{(i)})\right)^2$$

It is not a convex function
(convince yourself)

Log Loss (Binary Cross Entropy)

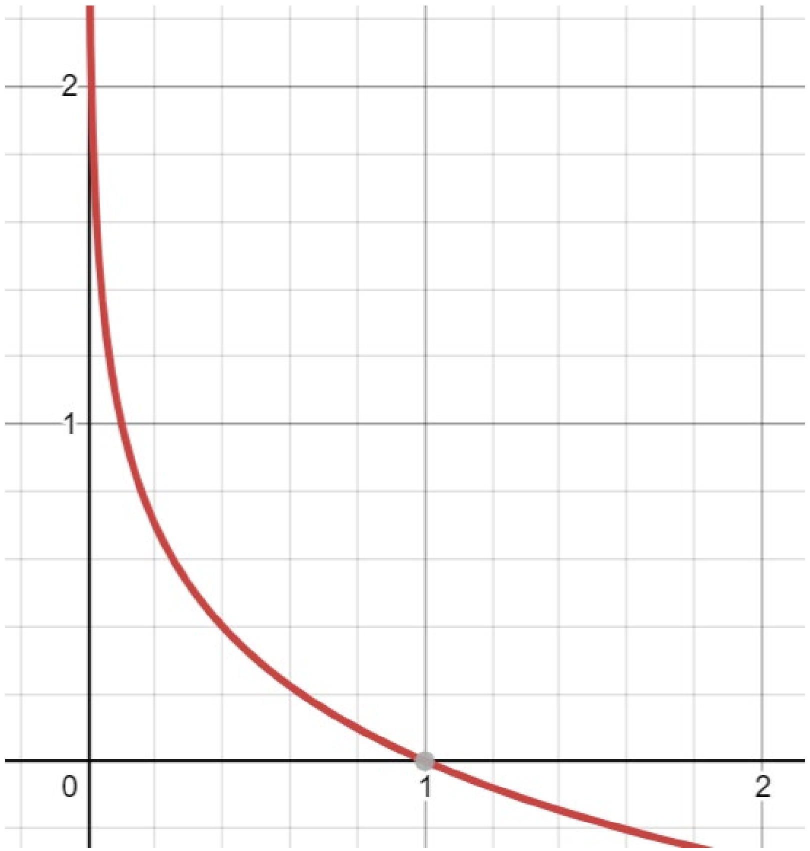
$$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$$

$$x^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{0,1\}, w \in \mathbb{R}^d$$

$$X \in \mathbb{R}^{N \times d}, Y \in \{0,1\}^N$$

$$\hat{y}^{(i)} = \sigma\left(w^\top x^{(i)}\right)$$

$$\text{BCE}(w) = -\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$



$-\log(x)$

Log Loss (Binary Cross Entropy)

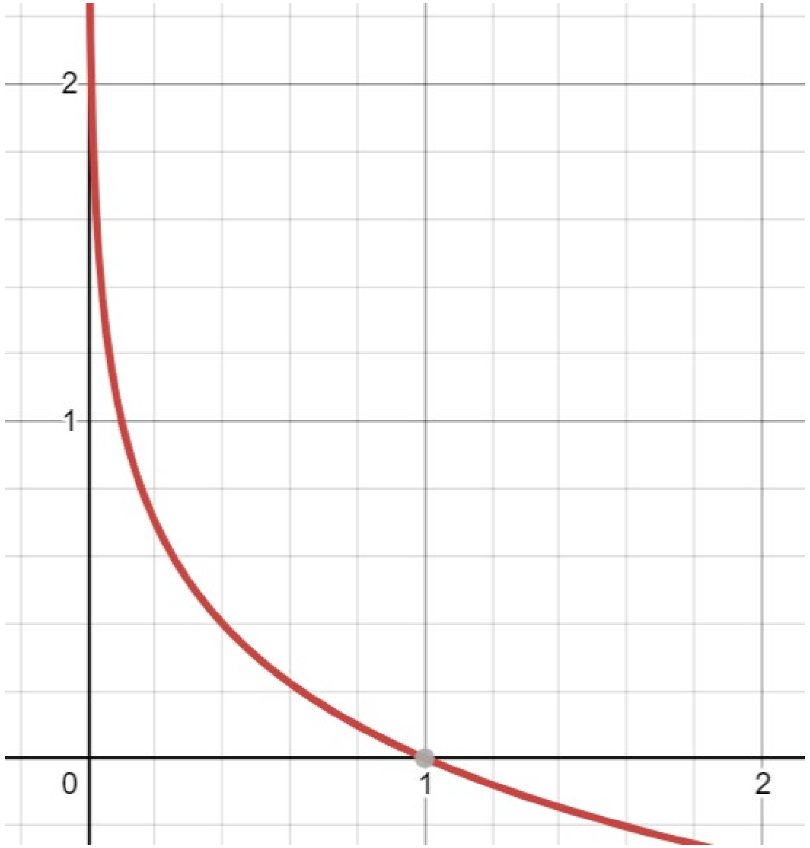
- Can we apply MSE loss function to logistic regression?

$$\hat{y}^{(i)} = \sigma\left(w^\top x^{(i)}\right)$$

$$\mathrm{BCE}(w) = -\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

$$-\log(1-\hat{y}), \qquad y^{(i)} = 0$$

$$-\log(\hat{y}), \qquad y^{(i)} = 1$$



$-\log(x)$

Log Loss (Binary Cross Entropy)

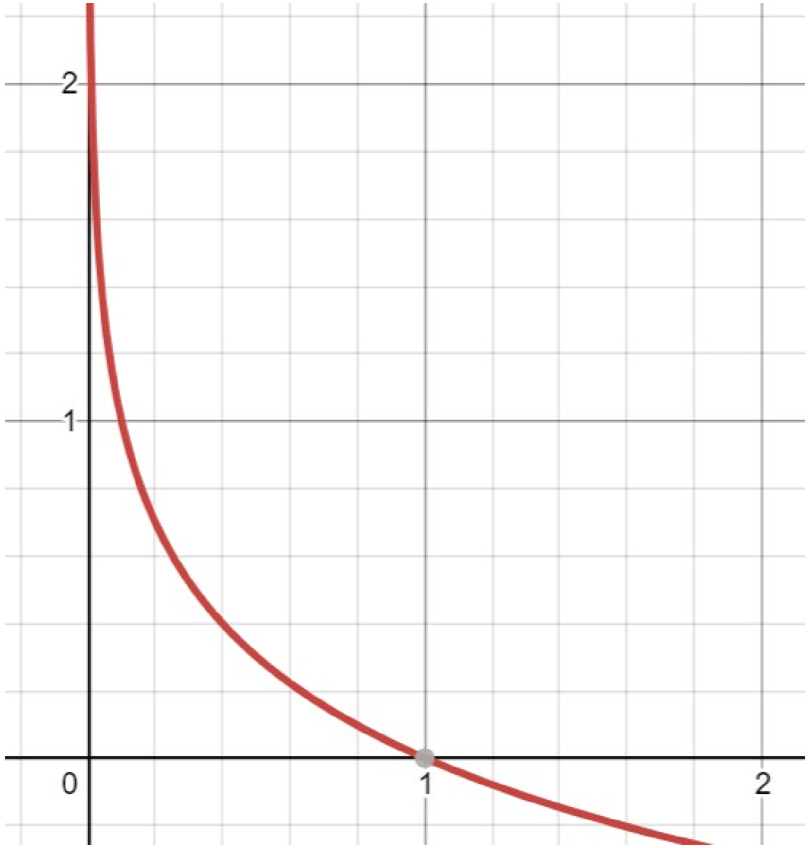
- Can we apply MSE loss function to logistic regression?

$$\hat{y}^{(i)} = \sigma\left(w^\top x^{(i)}\right)$$

$$\mathrm{BCE}(w) = -\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

$$-\log(1-\hat{y}), \qquad y^{(i)} = 0$$

$$-\log(\hat{y}), \qquad y^{(i)} = 1$$



$-\log(x)$

It is a convex function
(convince yourself)

Derivative of Sigmoid Function

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

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{(1 + e^{-x})}\frac{e^{-x}}{(1 + e^{-x})} = \sigma(x)(1 - \sigma(x))$$

BCE Loss

$$\mathrm{BCE}(w) = -\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

$$\hat{y}^{(i)} = \sigma\left(w^{\top} x^{(i)}\right)$$

$$\frac{\partial \mathrm{BCE}(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial \mathrm{BCE}(w)}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial w_j}$$

$$= -\sum_{i=1}^{N} \left(\frac{1}{\hat{y}^{(i)}} y^{(i)} + \frac{1}{1 - \hat{y}^{(i)}} \left(1 - y^{(i)}\right) \right) \frac{\partial \hat{y}^{(i)}}{\partial w_j}$$

$$= -\sum_{i=1}^{N} \left(\frac{1}{\hat{y}^{(i)}} y^{(i)} + \frac{1}{1 - \hat{y}^{(i)}} \left(1 - y^{(i)}\right) \right) \hat{y}^{(i)} \left(1 - \hat{y}^{(i)}\right) x_j^{(i)}$$

$$= \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

BCE Loss

$$\mathrm{BCE}(w) = -\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

$$\hat{y}^{(i)} = \sigma\left(w^{\top} x^{(i)}\right)$$

$$\frac{\partial \mathrm{BCE}(w)}{\partial w_j} = \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

$$\frac{\partial \mathrm{BCE}(w)}{\partial w} = X^{\top}(\sigma(Xw) - Y)$$

Can you solve this as we did before?
(take the gradients, and set to zero)

BCE Loss

$$\mathrm{BCE}(w) = -\sum_{i=1}^{N} y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

$$\hat{y}^{(i)} = \sigma\left(w^\top x^{(i)}\right)$$

$$\frac{\partial \mathrm{BCE}(w)}{\partial w_j} = \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)} = 0$$

$$\frac{\partial \mathrm{BCE}(w)}{\partial w} = X^\top(\sigma(Xw) - Y) = 0$$

$$w_j \coloneqq w_j - \alpha\left(\sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}\right)$$

$$w \coloneqq w - \alpha(X^\top(\sigma(Xw) - Y))$$

(Gradient Descent)

Probabilistic Interpretation

Maximum Likelihood Estimation (MLE)

Probability

A (probability density/mass) function of the data given the fixed parameters

$$p(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

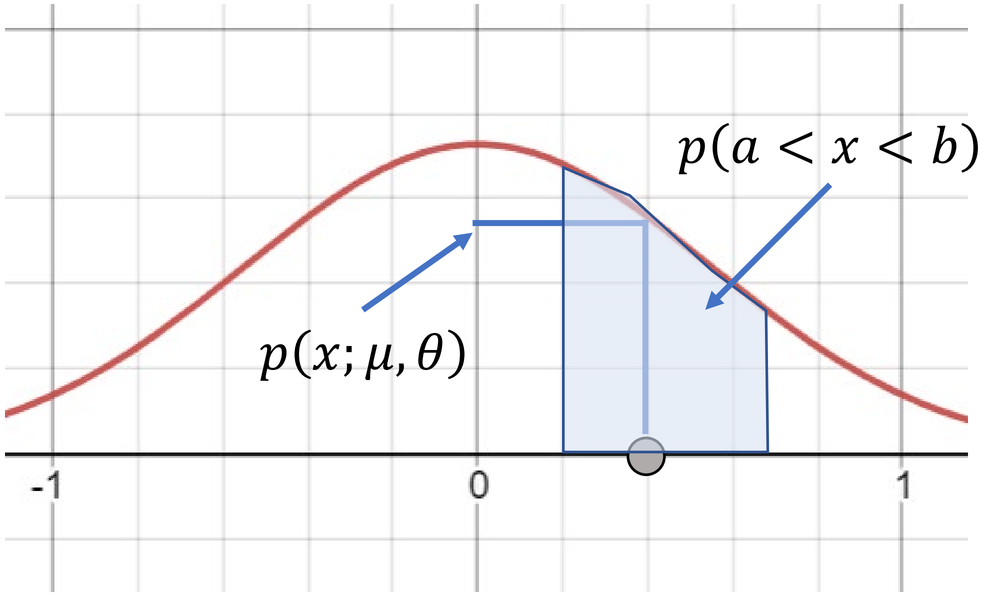
Likelihood

A (probability density /mass) function of parameters given the data

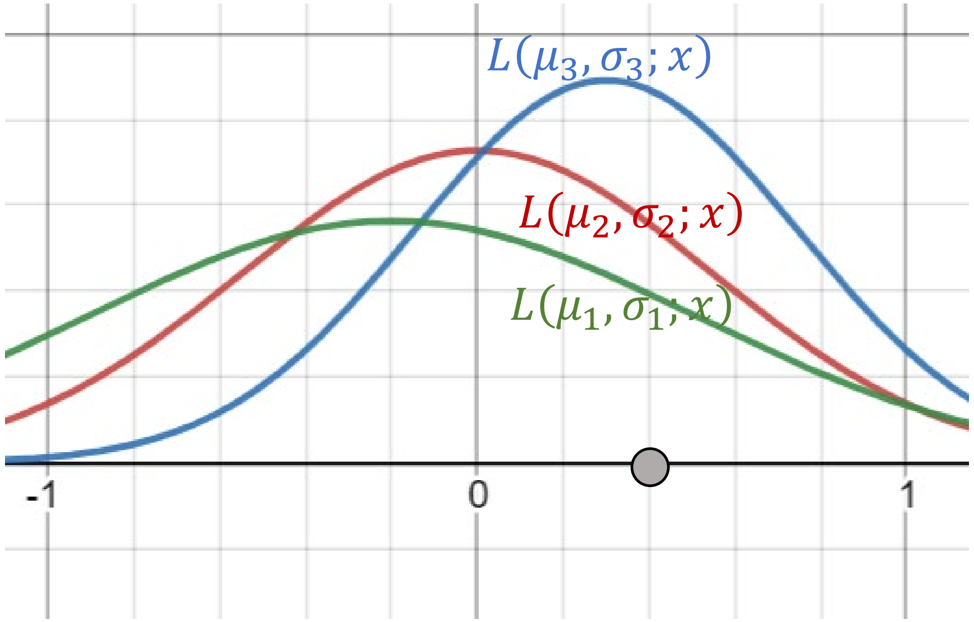
$$L(\mu,\sigma;x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Maximum Likelihood Estimation (MLE)

Probability Density Function



Likelihood



Maximum Likelihood Estimation (MLE)

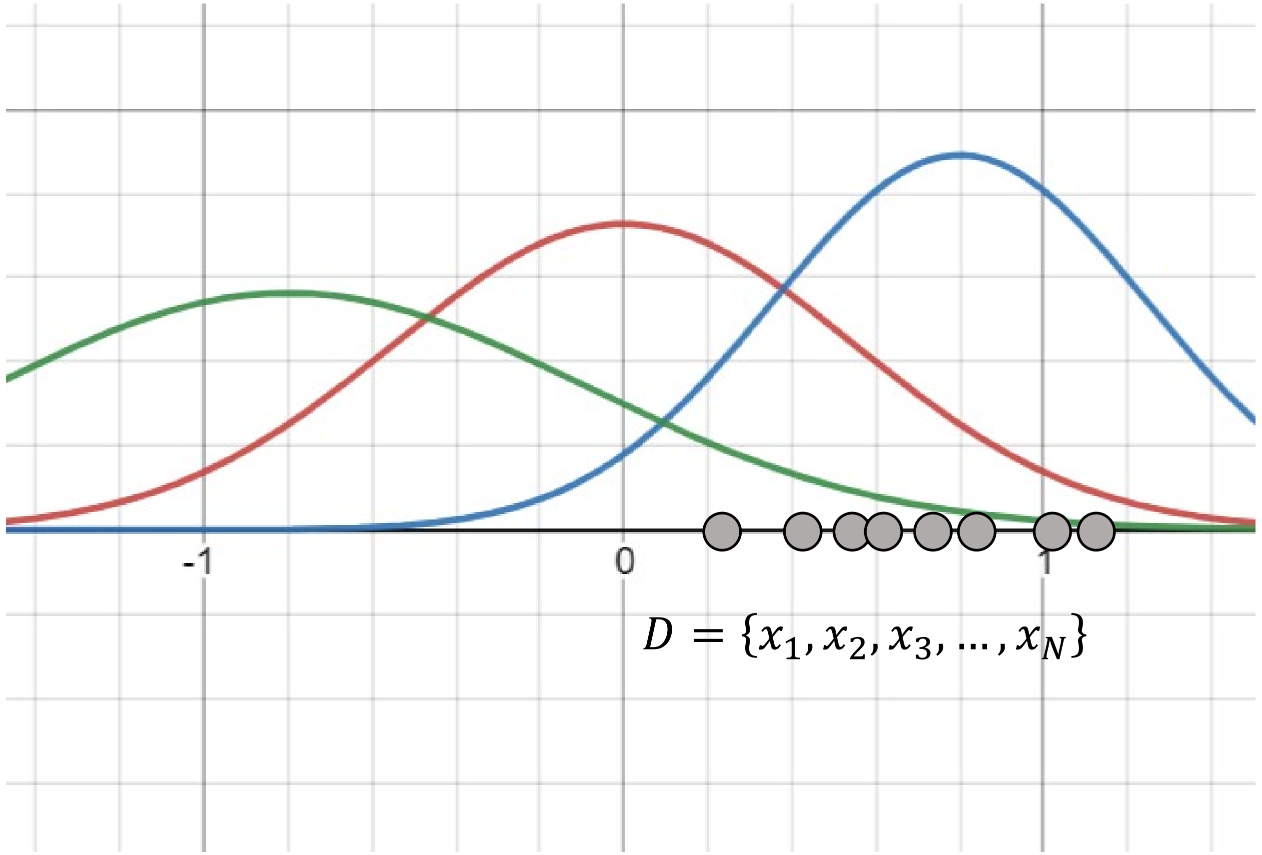
- Finding the parameters that maximize the probability (density/mass) function

I.I.D assumption

$$\arg\max_{\theta} L(\theta; x) = \arg\max_{\theta} \prod_{i=1}^{N} p(x_i; \theta)$$

$$= \arg\max_{\theta} \log \prod_{i=1}^{N} p(x_i; \theta)$$

$$= \arg\max_{\theta} \sum_{i=1}^{N} \log p(x_i; \theta)$$



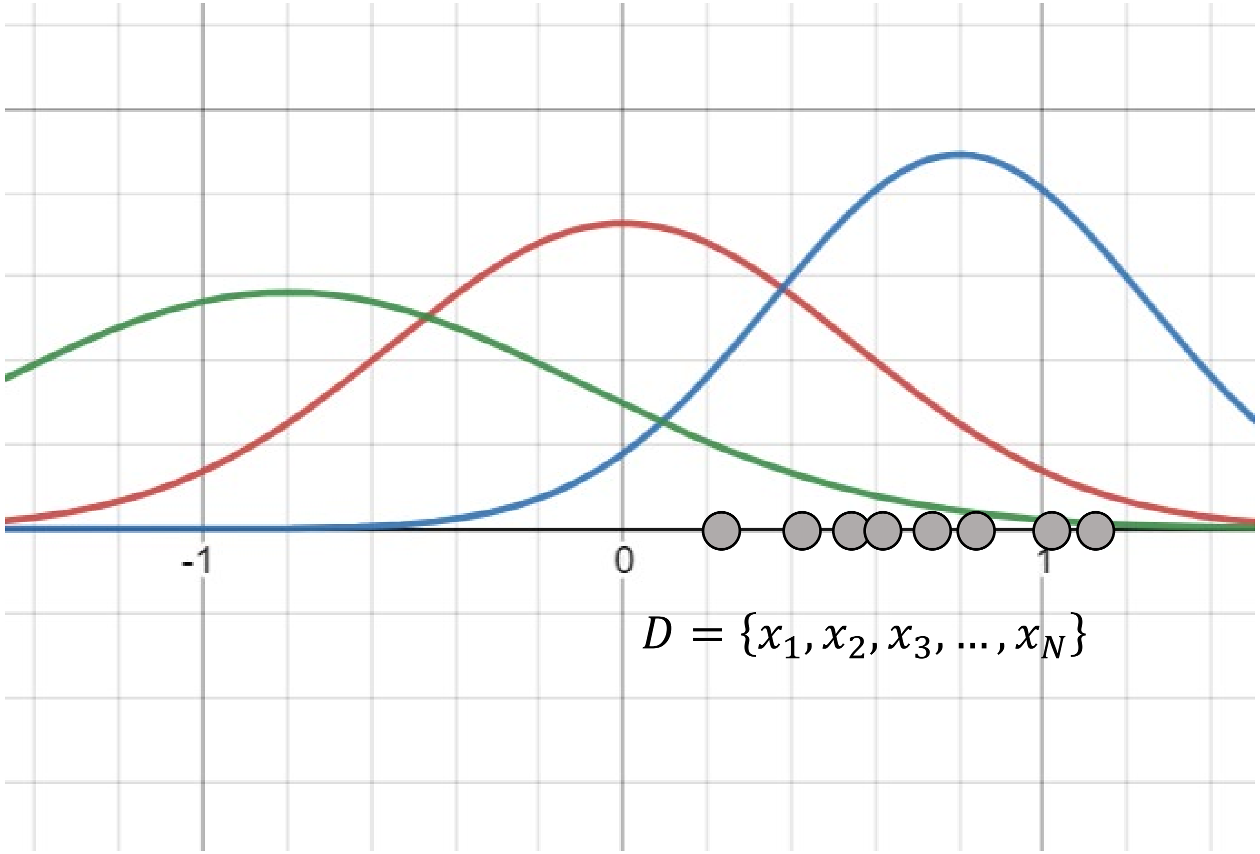
Maximum Likelihood Estimation (MLE)

- Finding the parameters that maximize the probability (density/mass) function

$$\arg\max_{\theta} \sum_{i=1}^{N} \log p(x_i; \theta)$$

$$= \arg\max_{\theta} \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}\right)$$

$$= \arg\max_{\theta} \sum_{i=1}^{N} -\frac{(x_i-\mu)^2}{2\sigma^2} - \log\left(\sqrt{2\pi\sigma^2}\right)$$



Maximum Likelihood Estimation (MLE)

- Finding the parameters that maximize the probability (density/mass) function

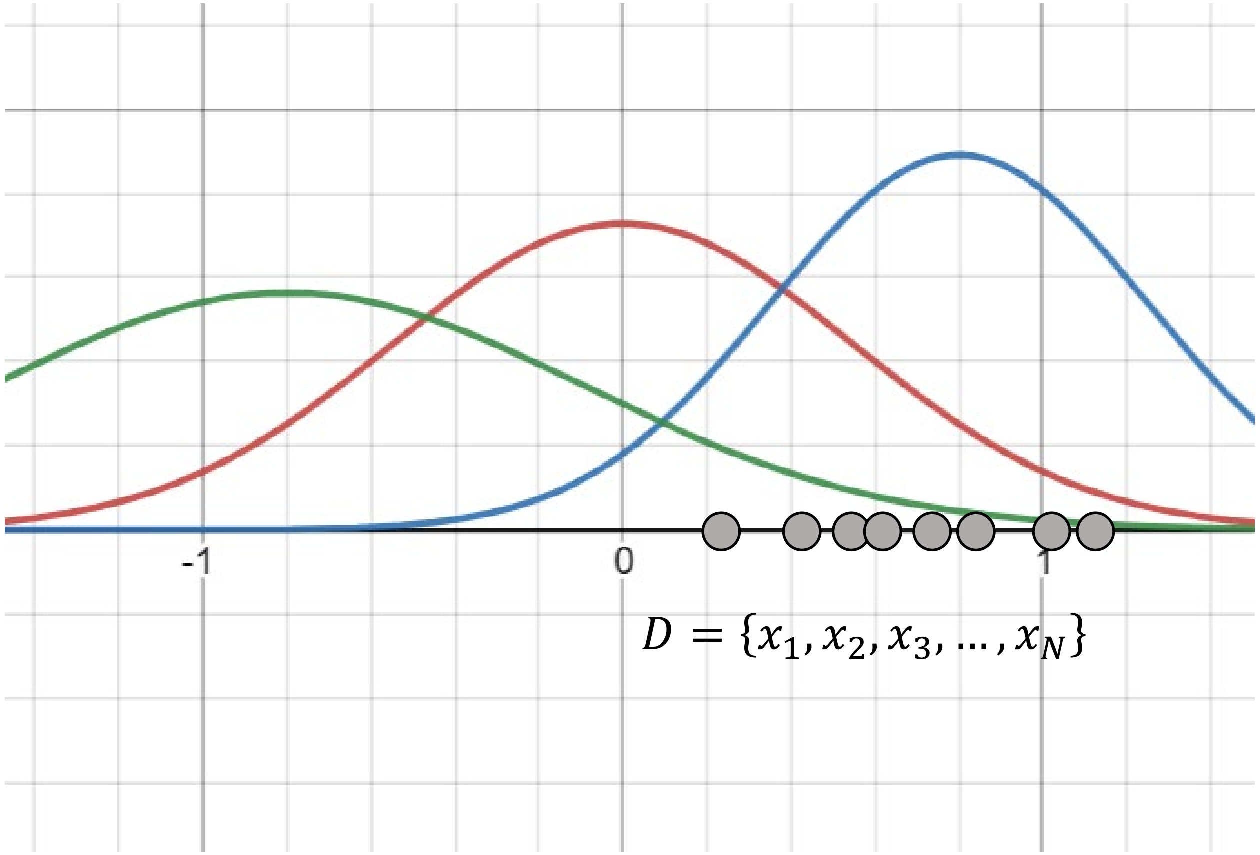
$$\arg\max_{\mu} \sum_{i=1}^{N} -\frac{(x_i - \mu)^2}{2\sigma^2} - \log\left(\sqrt{2\pi\sigma^2}\right)$$

$$\frac{1}{d\mu} \sum_{i=1}^{N} -\frac{(x_i - \mu)^2}{2\sigma^2} - \log\left(\sqrt{2\pi\sigma^2}\right)$$

$$= \sum_{i=1}^{N} \frac{(x_i - \mu)}{\sigma^2} = 0$$

$$\sum_{i=1}^{N} x_i - N\mu = 0$$

$$\mu^* = \frac{1}{N} \sum_{i=1}^{N} x_i$$



$D = \{x_1, x_2, x_3, \dots, x_N\}$

Maximum Likelihood Estimation (MLE)

- Finding the parameters that maximize the probability (density/mass) function

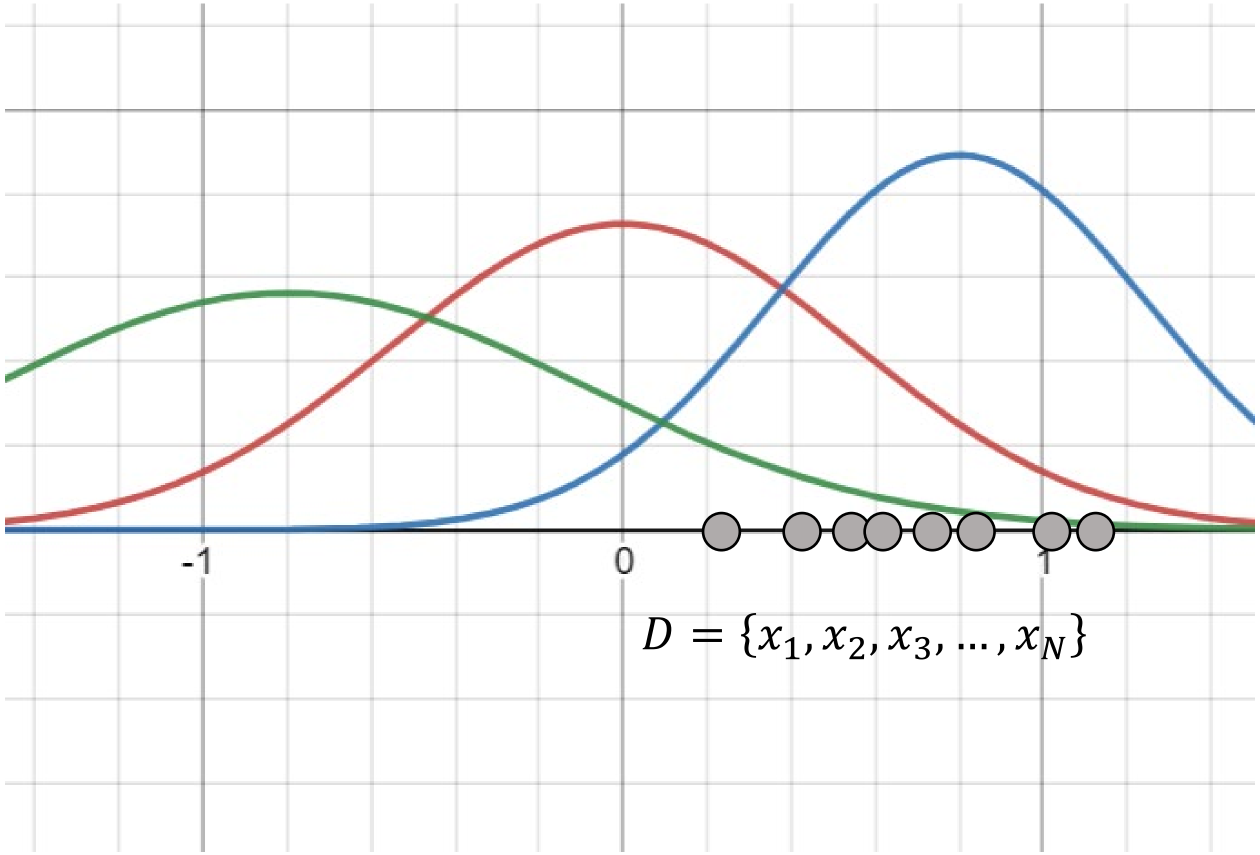
$$\underset{\sigma}{\arg\max} \sum_{i=1}^{N} -\frac{(x_i-\mu)^2}{2\sigma^2} - \log\left(\sqrt{2\pi\sigma^2}\right)$$

$$\frac{1}{d\sigma}\sum_{i=1}^{N} -\frac{(x_i-\mu)^2}{2\sigma^2} - \log\left(\sqrt{2\pi\sigma^2}\right)$$

$$= \frac{1}{\sigma^3}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{N}{\sigma} = 0$$

$$\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2 = \sigma^2$$

$$\sigma^* = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$$



MLE for Linear Regression

- Finding the parameters that maximize 'conditional likelihood'

Assumption1: $p(y|x)$ is a normal distribution

Assumption2: I.I.D

$$L(\theta) = \sum_{i=1}^{N} \log p(y_i|x_i;\theta) = \sum_{i=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\theta^\top x_i)^2}{2\sigma^2}}\right)$$

$$= -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \theta^\top x_i)^2 - \mathrm{N}\log\left(\sqrt{2\pi\sigma^2}\right)$$

$\sigma = 1$, we recover MSE Loss

MLE for Logistic Regression

- Finding the parameters that maximize 'conditional likelihood'

Assumption1: $p(y|x)$ is a Bernoulli distribution

Assumption2: I.I.D

$$L(\theta) = \sum_{i=1}^{N} \log p(y_i|x_i;\theta) = \sum_{i=1}^{N} \log \sigma(\theta^\top x_i)^{y_i}\big(1-\sigma(\theta^\top x_i)\big)^{1-y_i}$$

$$= \sum_{i=1}^{N} y_i \log \sigma(\theta^\top x_i) + (1-y_i)\log\big(1-\sigma(\theta^\top x_i)\big)$$

We recover BCE Loss

Multiclass Classification

Multiclass (Multinomial) Classification

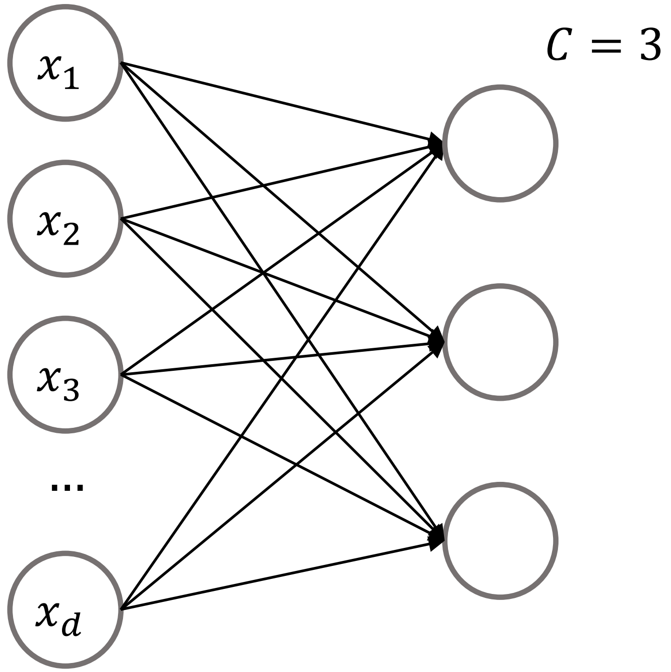
- Cross Entropy Loss
  - BCE is a special case of CE (two classes)

$$\mathrm{CE}(w) = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} \log\left(\hat{y}_c^{(i)}\right)$$

$$\mathrm{BCE}(w) = -\sum_{i=1}^{N}\sum_{c=1}^{2} y_c^{(i)} \log\left(\hat{y}_c^{(i)}\right)$$

$$\hat{y}^{(i)} = \sigma\left(w^\top x^{(i)}\right) \in \mathbb{R}^C$$

$$y^{(i)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$ (one-hot vector)



Softmax Function

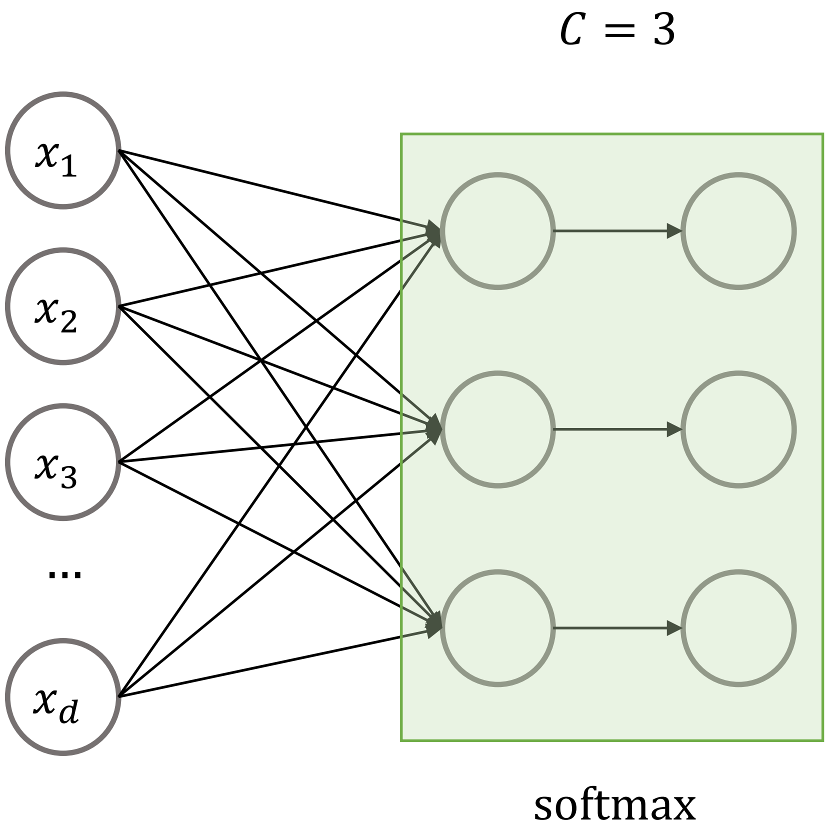
- 'Soft' 'Max' function
  - $[1,2,3,2,1] \rightarrow [0.0674, 0.183\ , 0.498, 0.183\ , 0.0674]$

$$\text{softmax}: \mathbb{R}^C \rightarrow [0,1]^C$$

$$|\text{softmax}(x)| = 1$$

$$\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{i=1}^{C} e^{x_i}}$$

$$\text{softmax}(x) = \begin{bmatrix} \frac{e^{x_1}}{\sum_{i=1}^{C} e^{x_i}} \\ \frac{e^{x_2}}{\sum_{i=1}^{C} e^{x_i}} \\ \vdots \\ \frac{e^{x_C}}{\sum_{i=1}^{C} e^{x_i}} \end{bmatrix} \in [0,1]^C$$



Derivative of the Softmax Function

$$y_j = \frac{e^{x_j}}{\sum_{i=1}^{C} e^{x_i}}$$

$$\frac{\partial y_i}{\partial x_j} = \begin{cases} y_i(1 - y_i), & i = j \\ -y_i y_j, & i \neq j \end{cases} = y_i\big(1\{i = j\} - y_j\big)$$

$$\frac{dy}{dx} = \begin{bmatrix} y_1(1 - y_1) & \cdots & y_1 y_4 \\ \vdots & \ddots & \vdots \\ -y_4 y_1 & \cdots & y_C(1 - y_C) \end{bmatrix}$$

Convince yourself ☺

Cross-Entropy + Softmax

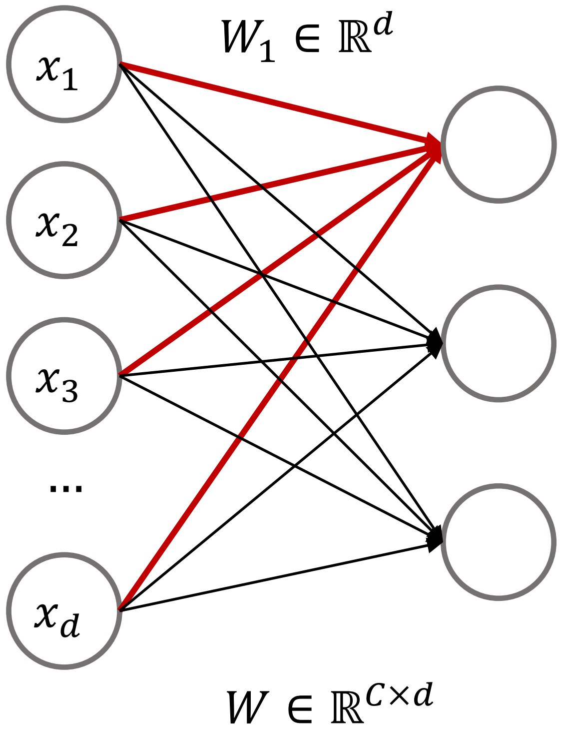
$$L(w) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \qquad \hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \qquad z_c = W_c^{\top} x$$

$$\frac{\partial \hat{y}_i}{\partial x_j} = \begin{cases} \hat{y}_i(1-\hat{y}_i), & i = j \\ -\hat{y}_i \hat{y}_j, & i \neq j \end{cases}$$

$$\frac{\partial L}{\partial z_j} = -\frac{\partial}{\partial z_j}\sum_{i=1}^{C} y_i \log(\hat{y}_i) = -\sum_{i=1}^{C} y_i \frac{\partial \log(\hat{y}_i)}{\partial z_j} = -\sum_{i=1}^{C} \frac{y_i}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_j}$$

$$= -\frac{y_j}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_j} - \sum_{i\neq j}^{C} \frac{y_i}{\hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_j} = -\frac{y_j}{\hat{y}_j}\hat{y}_j\left(1-\hat{y}_j\right) + \sum_{i \neq j}^{C} \frac{y_i}{\hat{y}_i}\hat{y}_i\hat{y}_j$$

$$= -y_j + y_j\hat{y}_j + \sum_{i\neq j}^{C} y_i\hat{y}_j = -y_j + \hat{y}_j \sum_{i=1}^{C} y_i = \boldsymbol{\hat{y}_j - y_j}$$



$$\boxed{\frac{dL}{dz} = \hat{y} - y}$$

Relationship to Logistic Regression

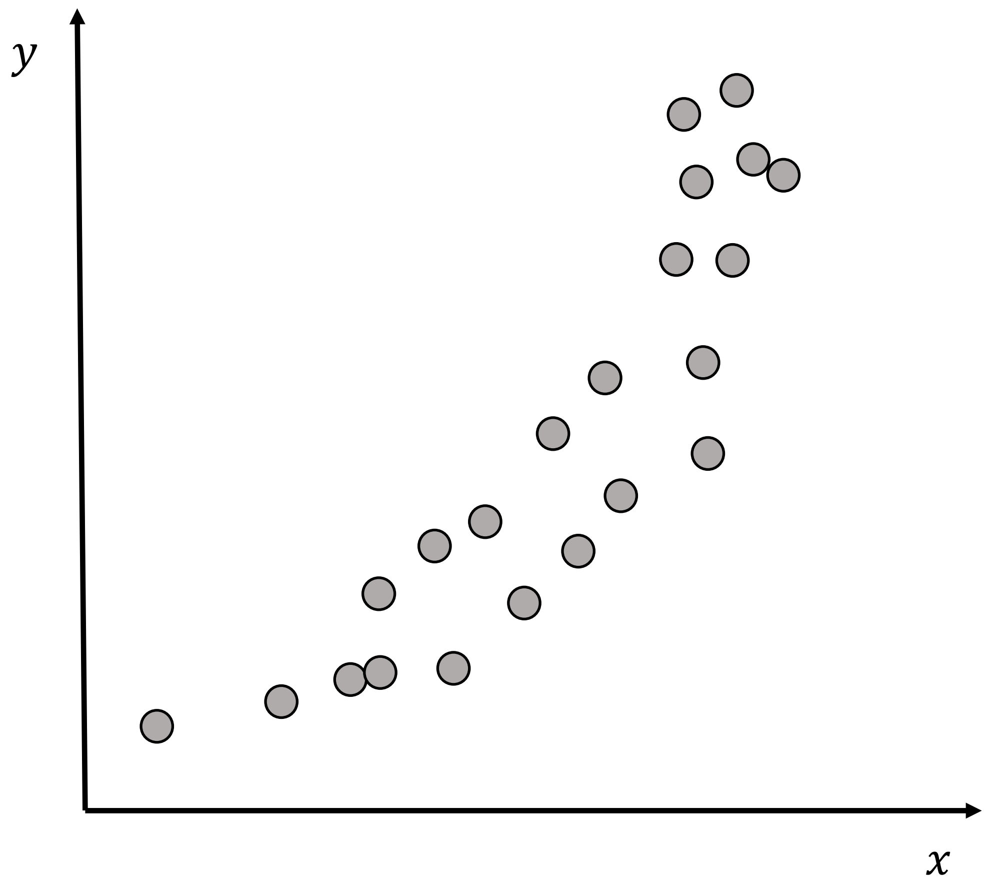
- Logistic regression is a special case of CE+Softmax classification when $C = 2$

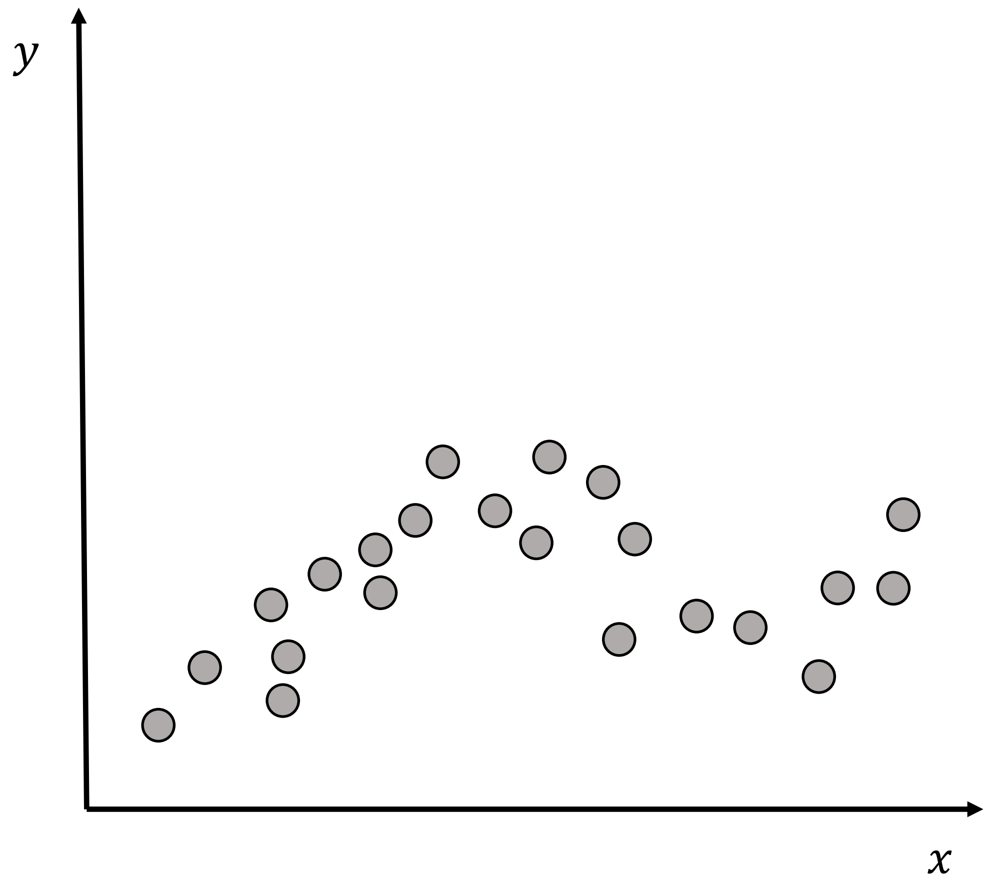
Convince yourself ☺

Multi-Layer Perceptron

Linear Models

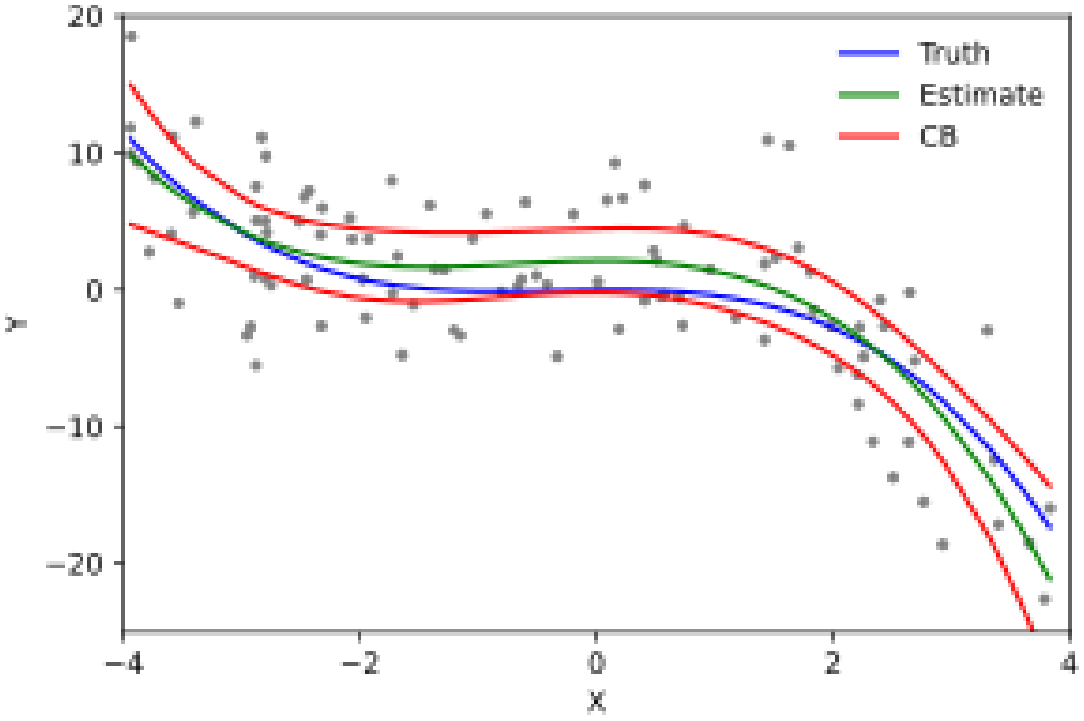
- Is linear model a good for all?





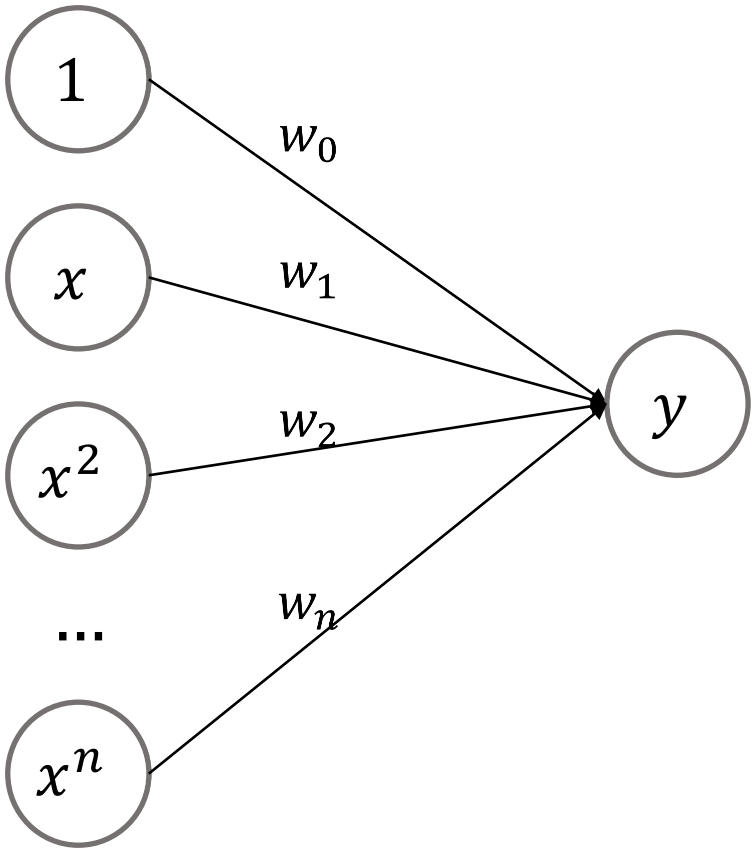
Nonlinear Models

- nth-degree Polynomial regression



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

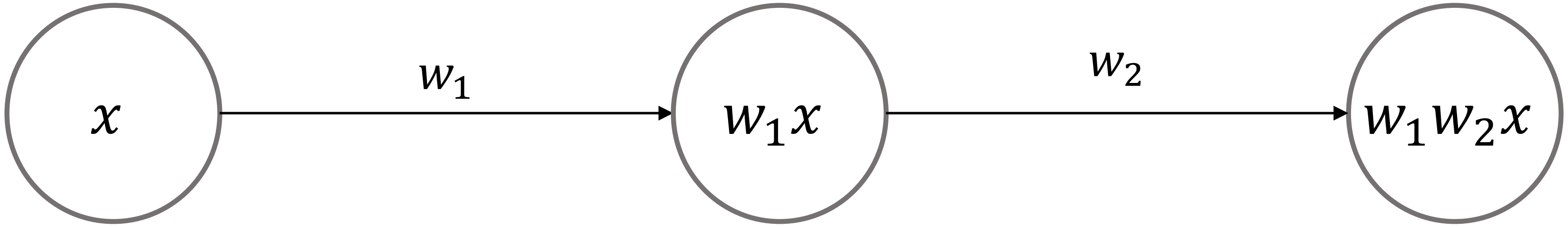
Polynomals as Neural Network



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

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

Feed-Forward Neural Network

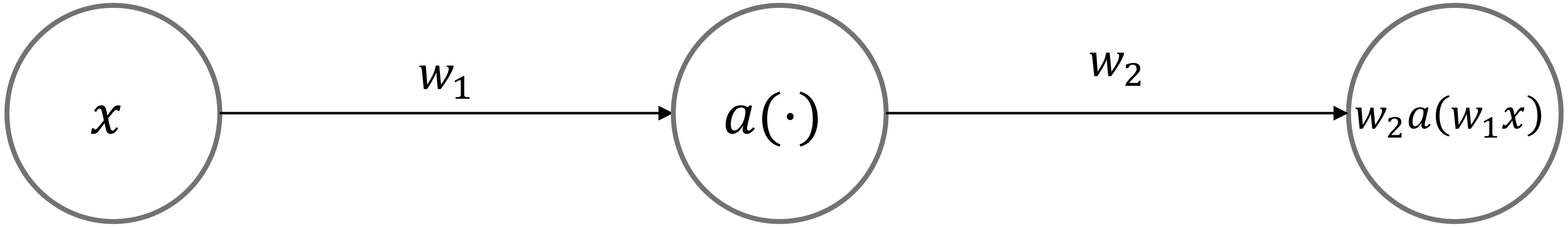


$$f(x) = w_1 w_2 x$$

Is it non-linear in x?

Feed-Forward Neural Network

- Using non-linear activation function



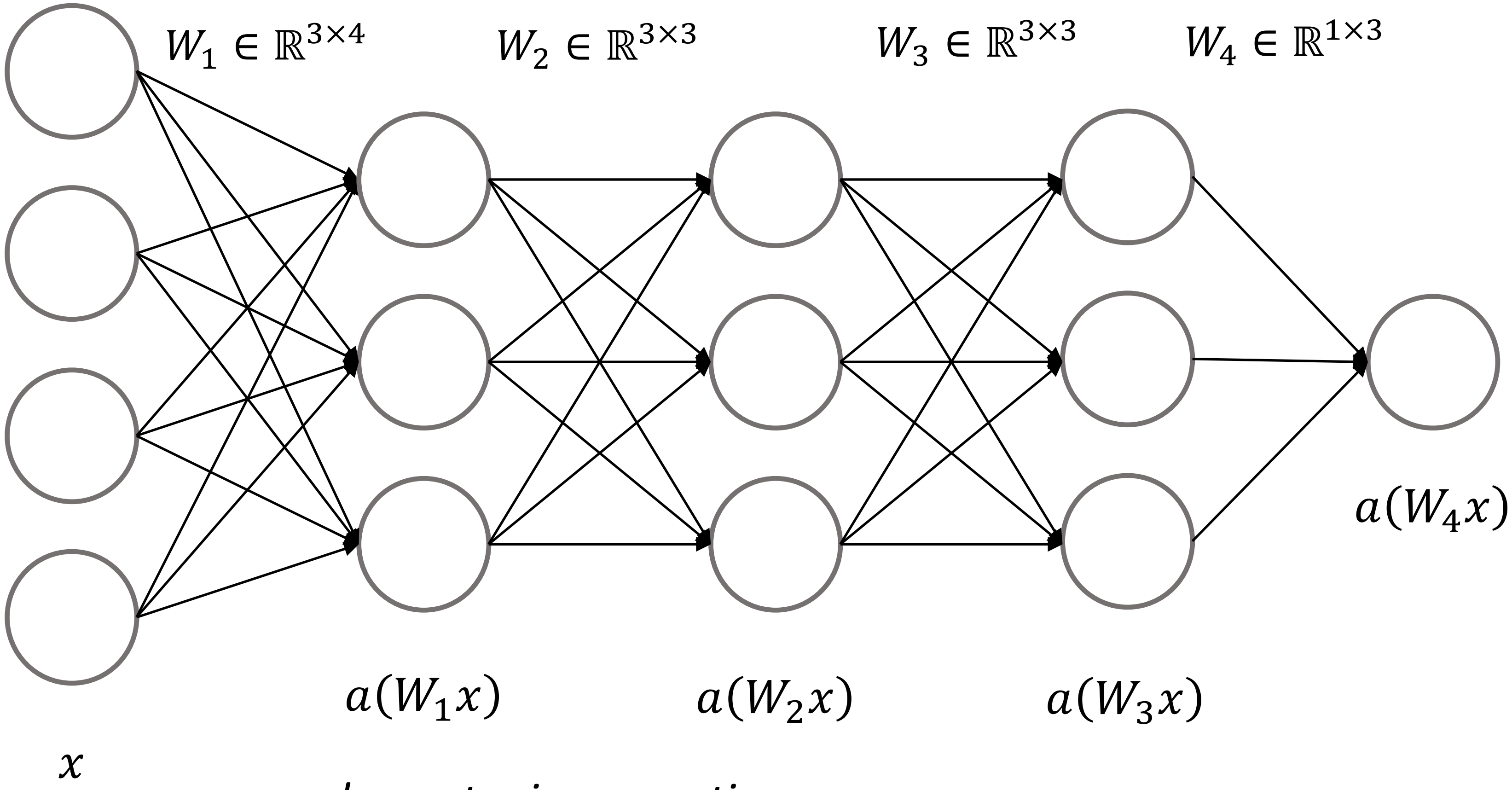
$$f(x) = w_2 a(w_1 x)$$

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

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

Feed-Forward Neural Network

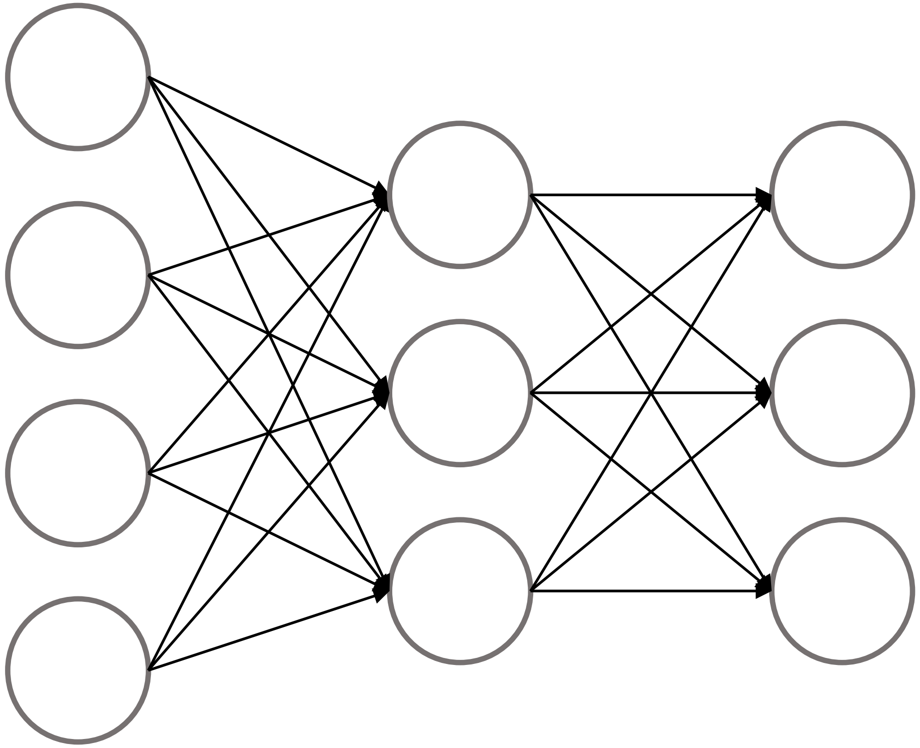
- AKA, Multi-Layer Perceptron

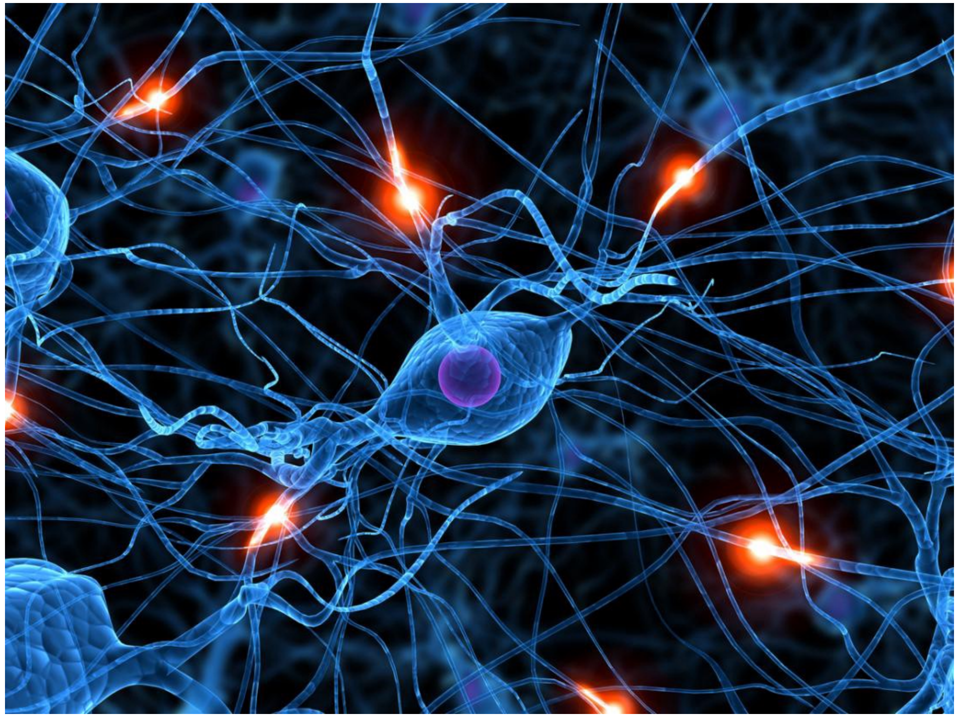


a: element-wise operation

Feed-Forward Neural Network

- AKA, Multi-Layer Perceptron





Feed-Forward Neural Network

- Regression with two layers MLP

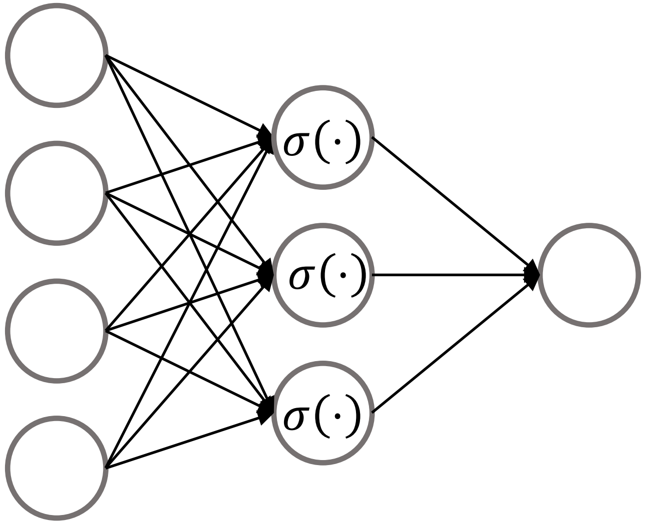
$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$

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

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

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

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



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

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?
2. Does it have a closed form solution?
3. Is it a convex function?

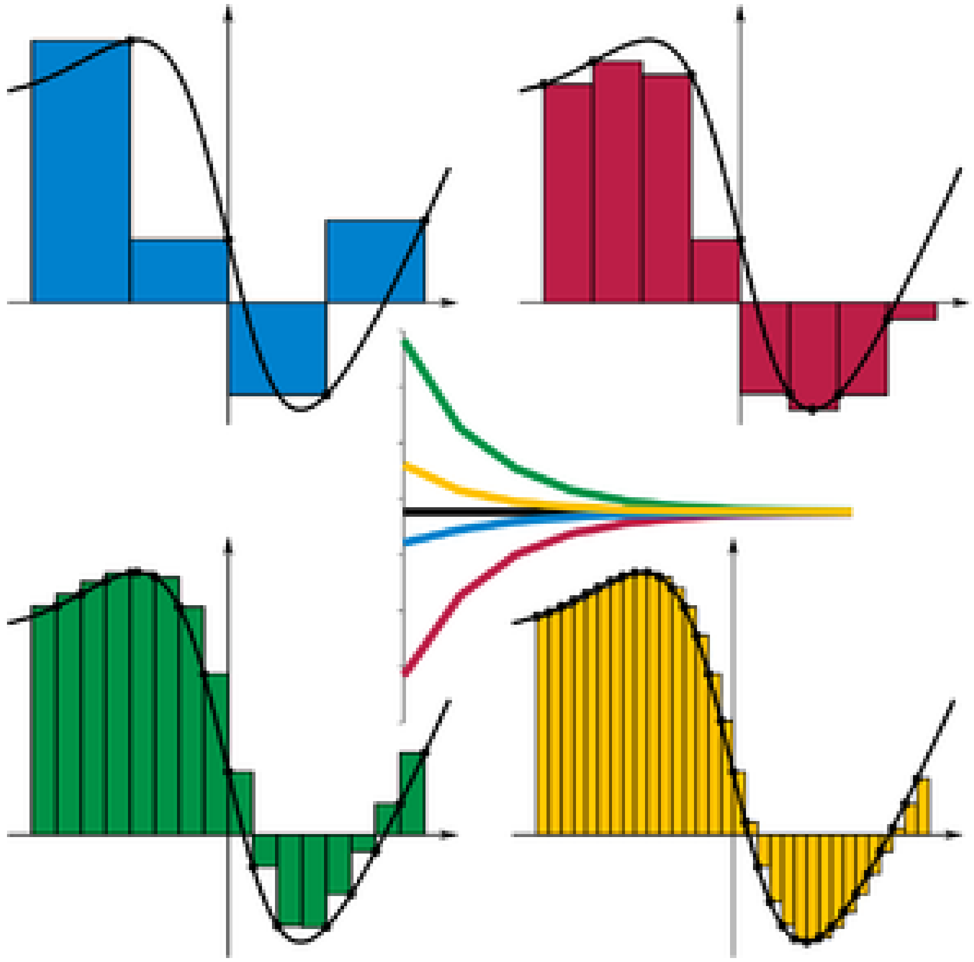
$$L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - f_\theta(x^{(i)})\right)^2 = \frac{1}{2}(Y - \sigma(W_1 X^\top)^\top W_2^\top)^\top (Y - \sigma(W_1 X^\top)^\top W_2^\top)$$

The Universal Approximator

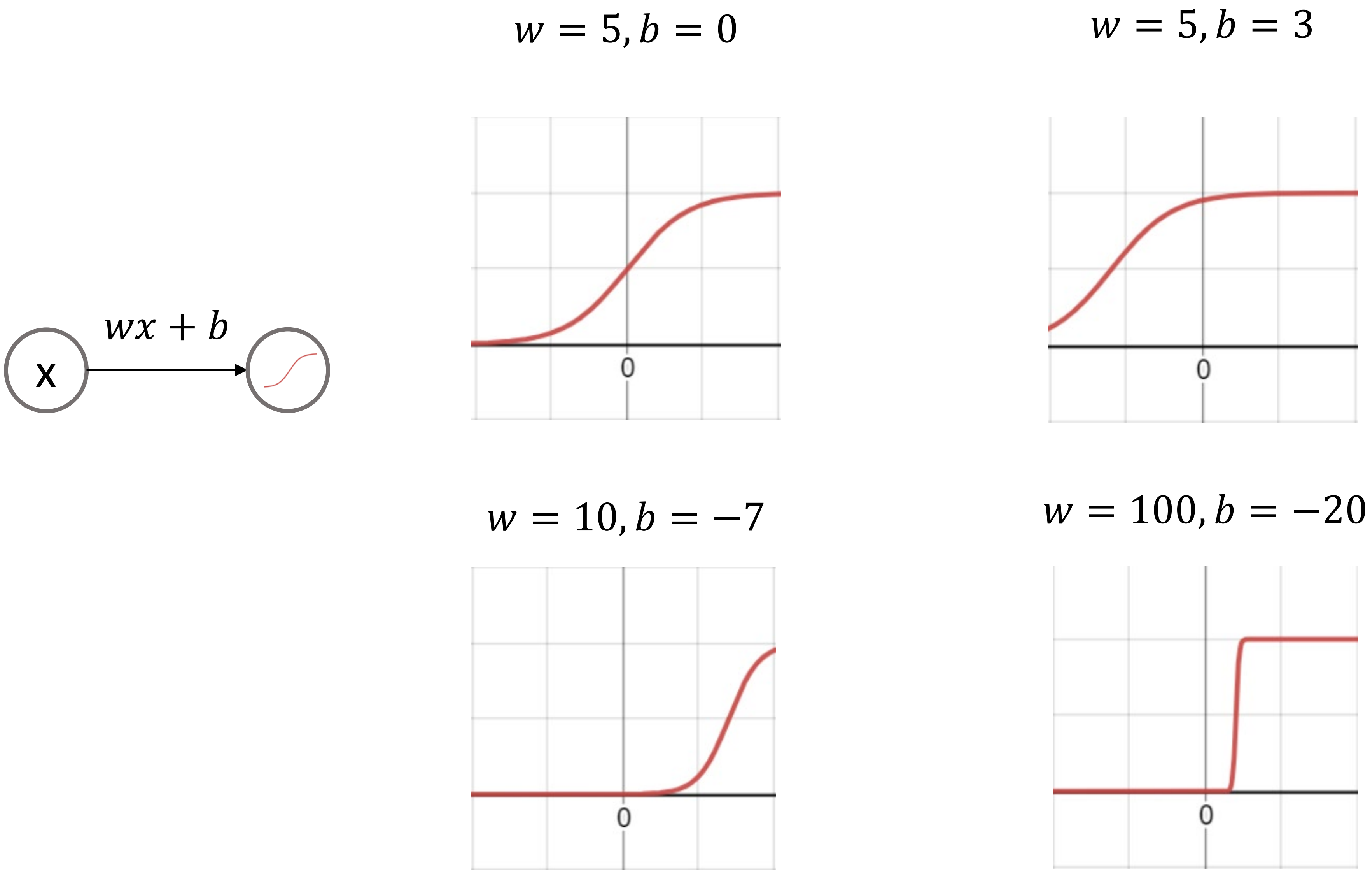
The Universal Approximation Theorem

- 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.

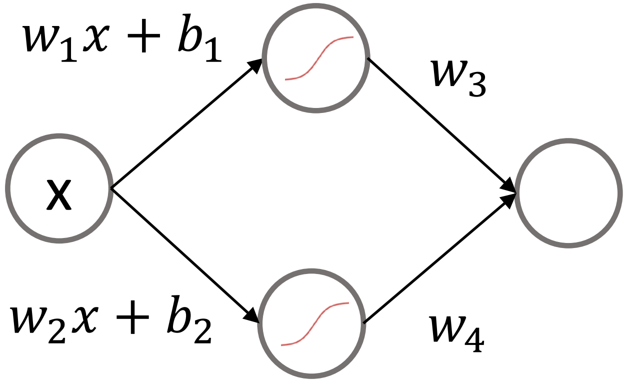
The Universal Approximation Theorem



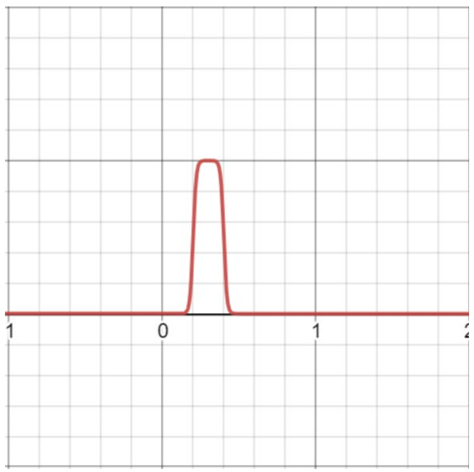
The Universal Approximation Theorem



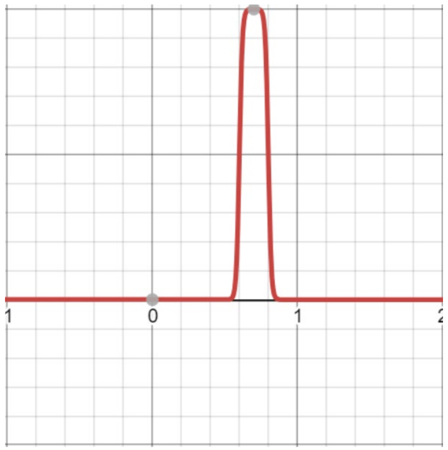
The Universal Approximation Theorem



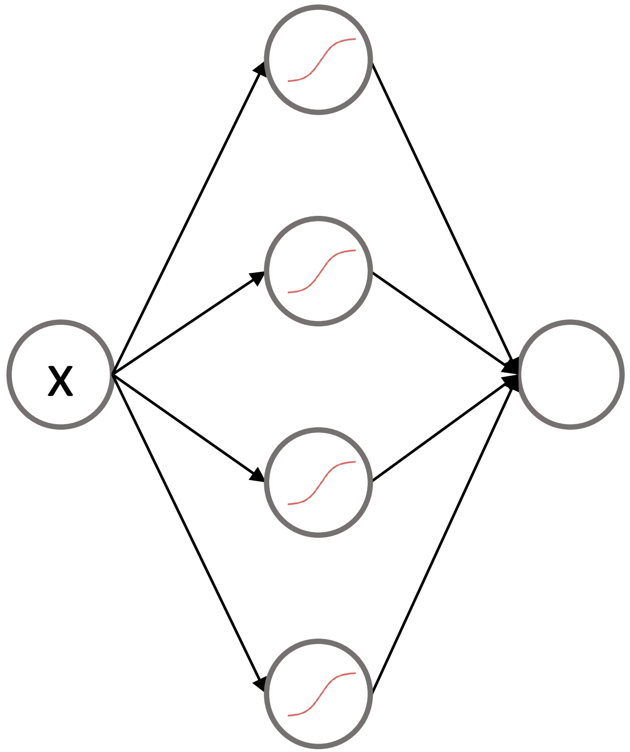
$w_1 = 100, b_1 = -20$
$w_2 = 100, b_2 = -40$
$w_3 = 1, w_4 = -1$

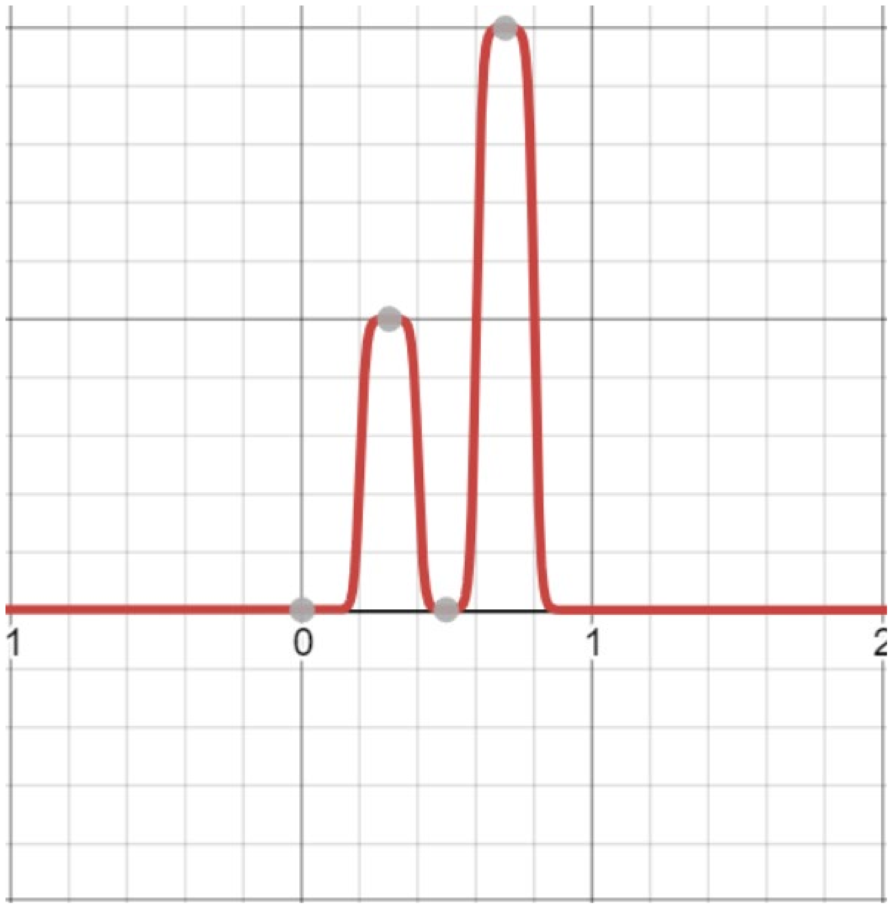


$w_1 = 100, b_1 = -60$
$w_2 = 100, b_2 = -80$
$w_3 = 2, w_4 = -2$

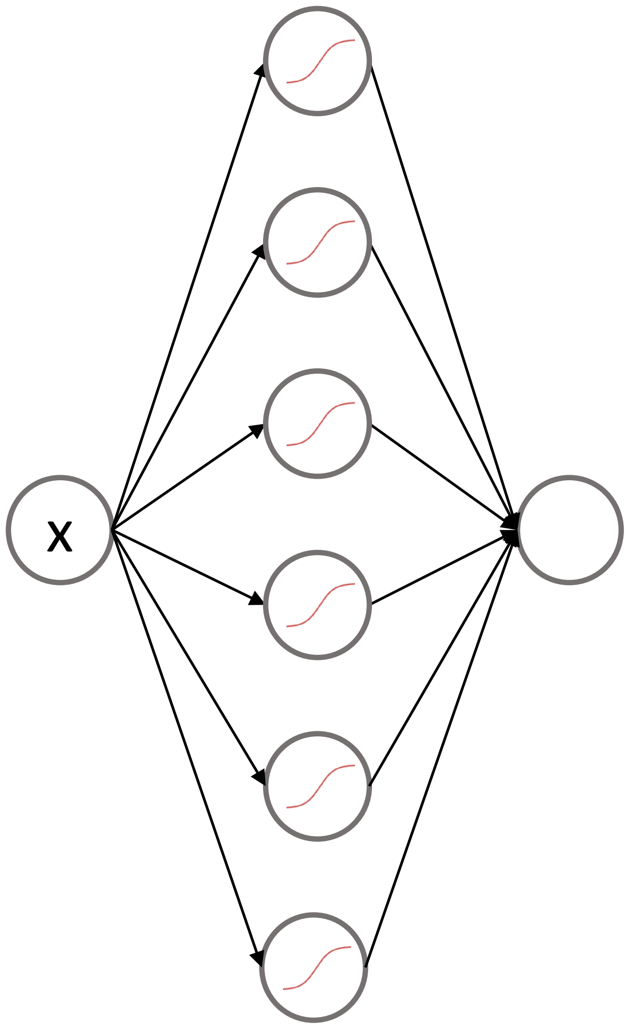


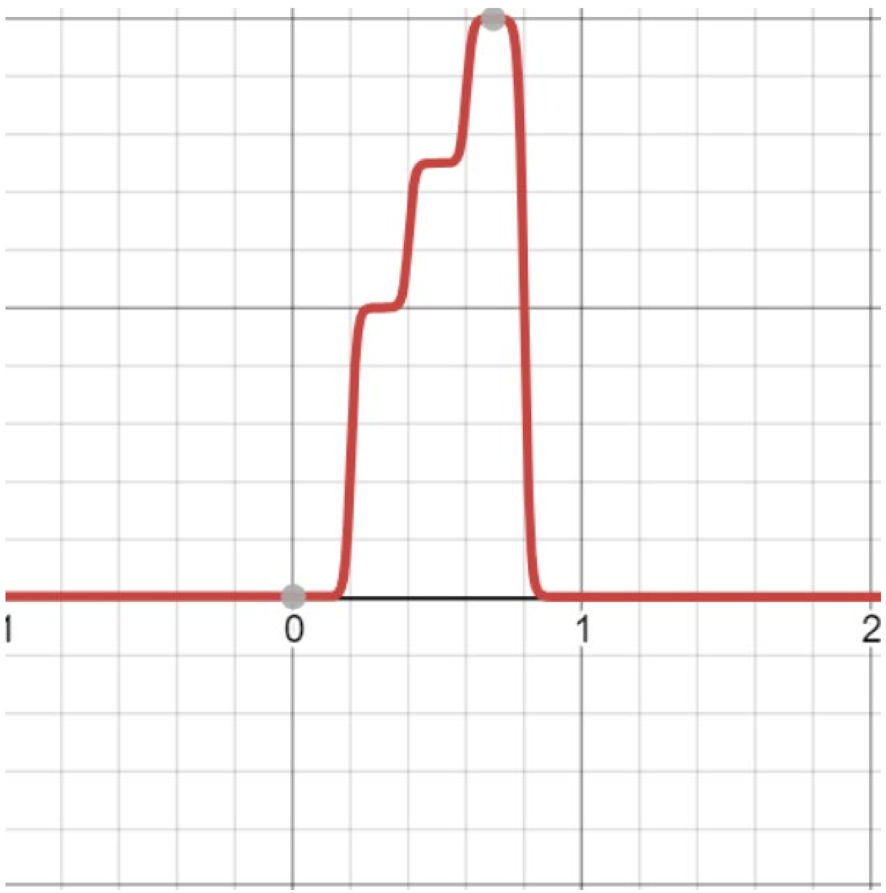
The Universal Approximation Theorem



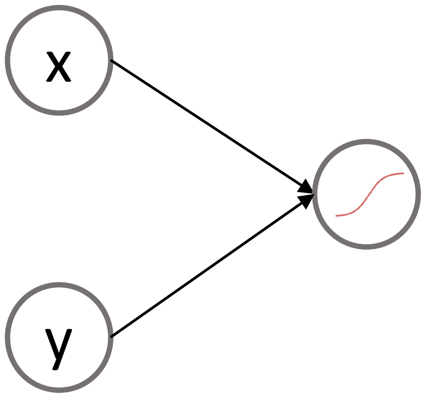


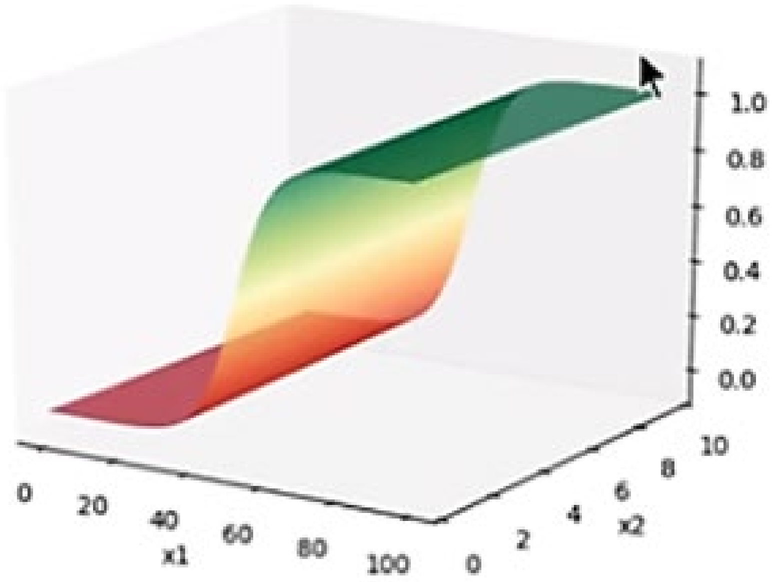
The Universal Approximation Theorem



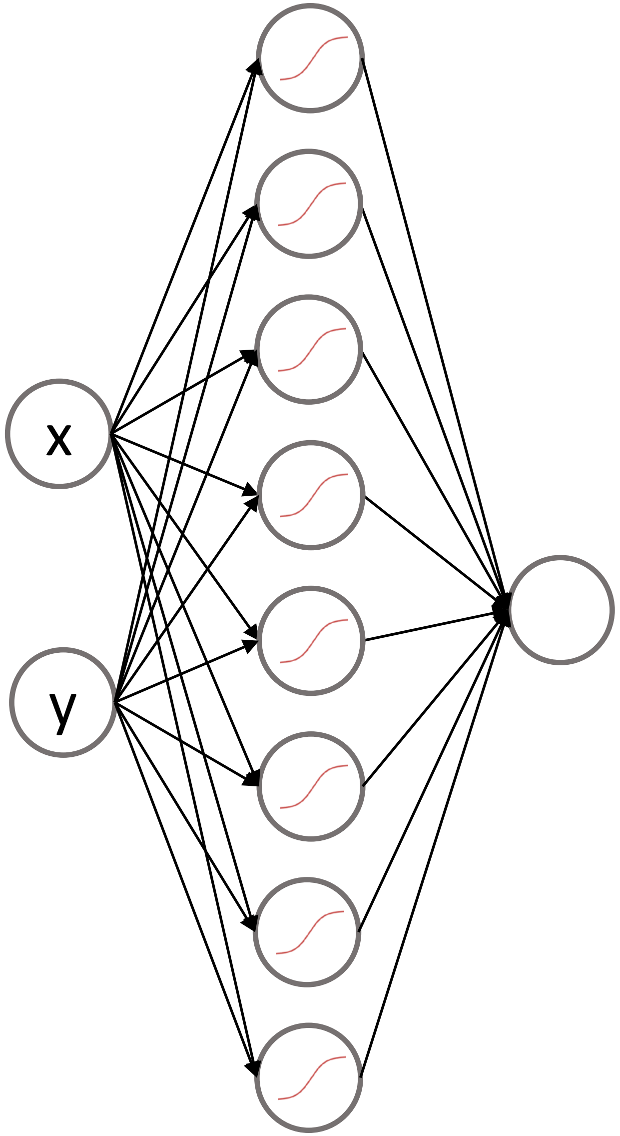


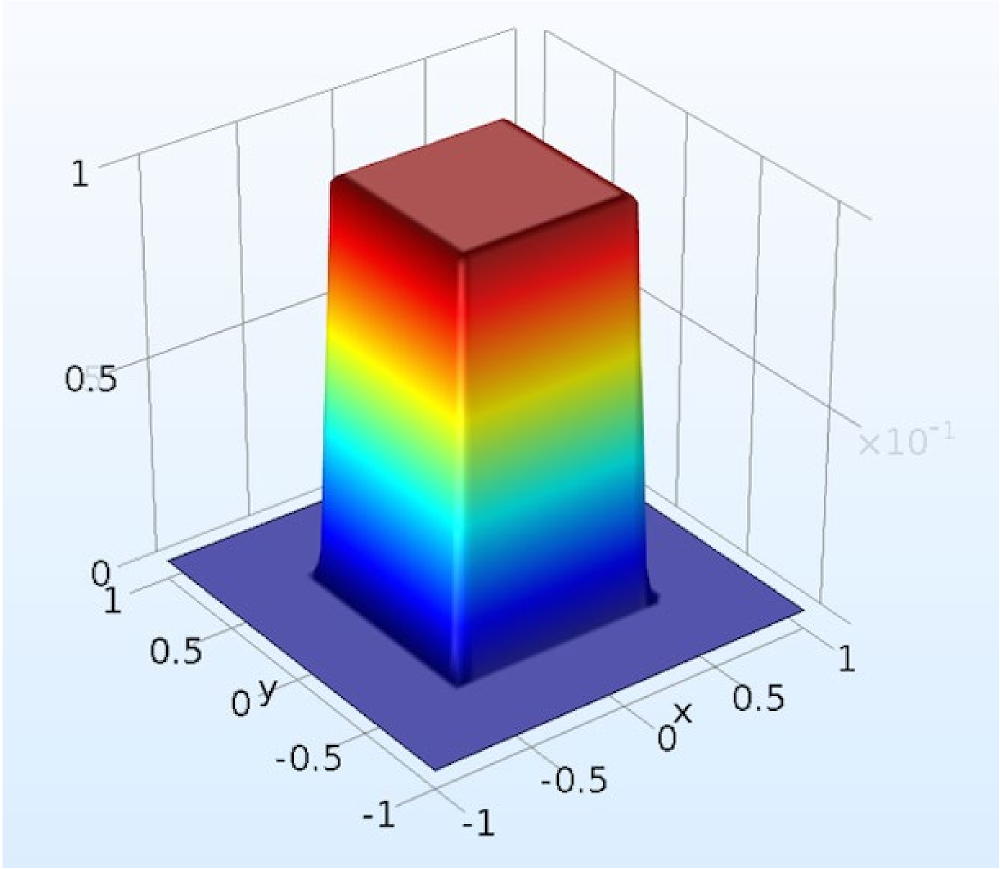
The Universal Approximator in 2D





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