Machine Learning

Artificial Neural Networks

ADF

Outline

- History and Motivation
  - Linear Regression
  - Linear Classifier
  - Logistic Regression
- Neuron Model
- Neural Network Architectures
- Activation Function

History and Motivation:

Linear Regression

Linear Regression

- Remember Linear Regression
  - Modelling the relation that best fit the data using a single line (linear function)

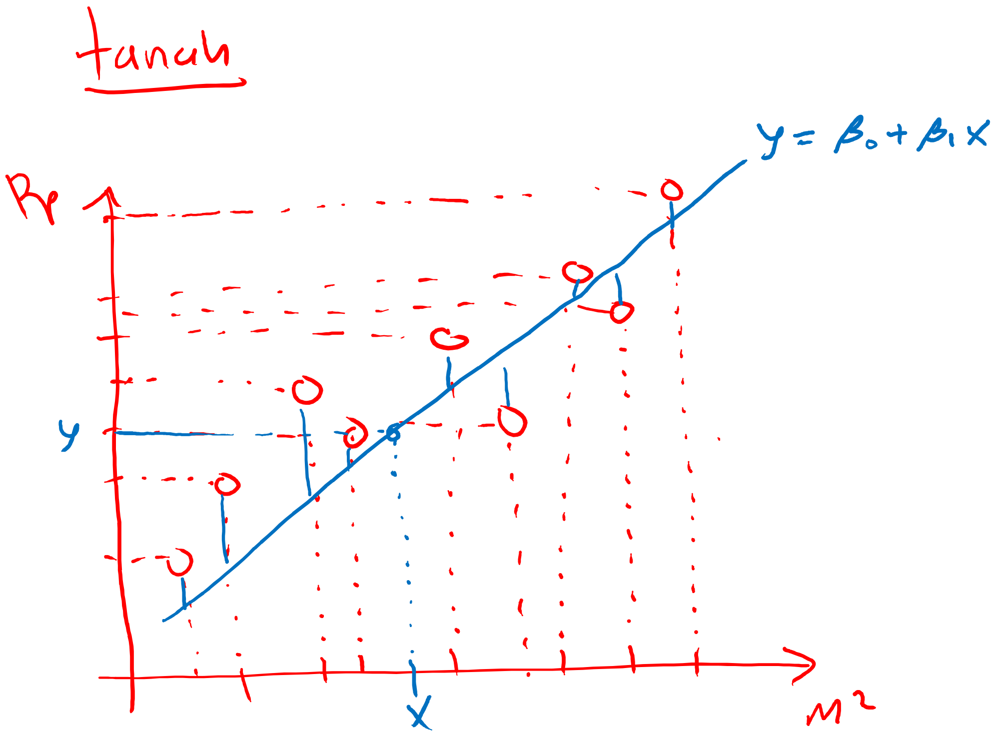
$$\boxed{y = \beta_0 + \beta_1 x}$$

    - β_0 : Population Y-Intercept (intercept, bias, …)
    - β_1 : Population slope (weight vector,…)

  - In a multivariate (multidimensional) x, we'll have

$$\boxed{y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d}$$

Univariate Linear Regression



Linear Regression

- Intuition:
  - Modelling the relation that best fit the data using a single line (linear function)
- Problem:
  - What is the best weights (parameters)
- 1st Solution: Least Square Error
  - Define a Cost/Loss/Error function (SSE, MSE, etc.)
  - Find weights that minimize the Cost Function
  - Use the First derivative

$$\boxed{\hat{w} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}}$$

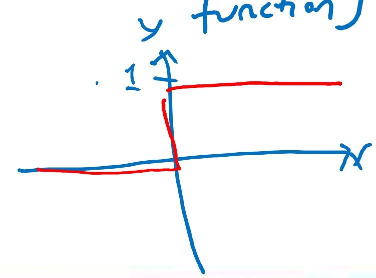
History and Motivation:

Linear Classification

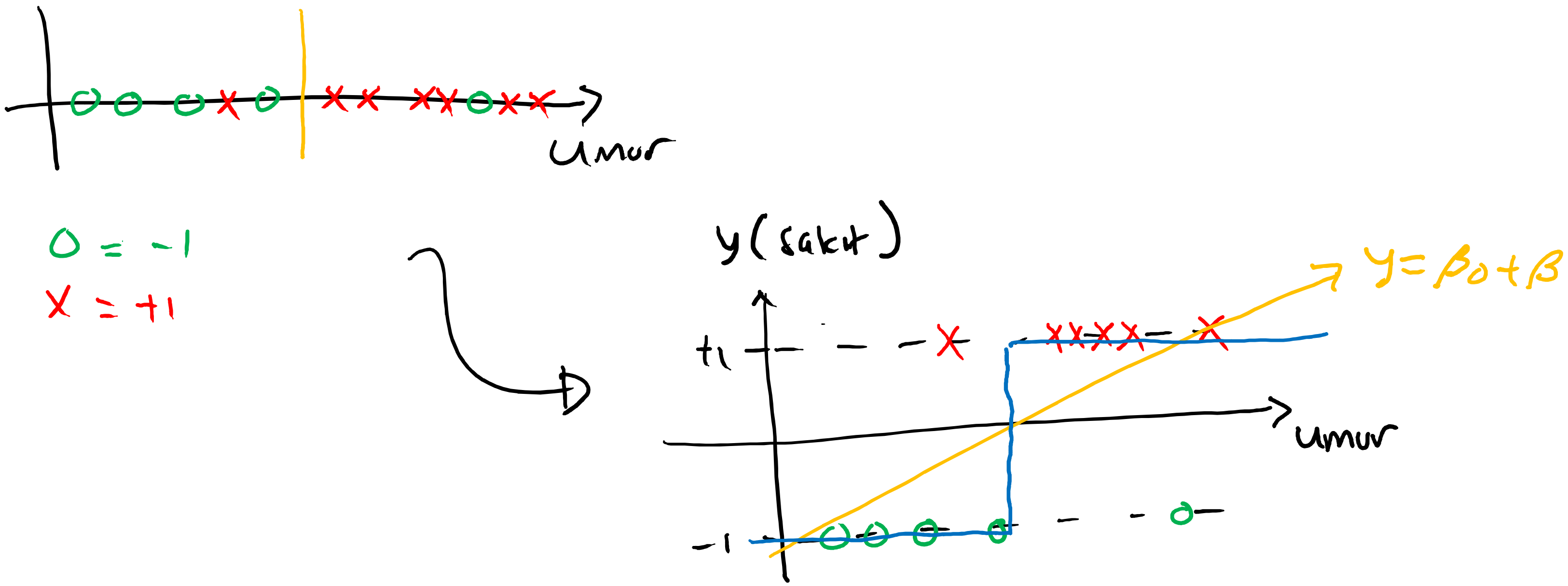
Linear Classification

- Intuition:
  - Modelling the relation that best fit the data using a single line
  - Find weights that minimize the Cost Function
- Problem:
  - For binary classification, y is categorical $\{-1, +1\}$
- Solution:
  - Use discriminative function to decide which class example (step function)

$$\boxed{\left.\begin{array}{l} f(x_i) > 0 \Leftrightarrow \hat{y}_i = +1 \\ f(x_i) < 0 \Leftrightarrow \hat{y}_i = -1 \end{array}\right\} \text{i.e. } \hat{y}_i = sign\big(f(x_i)\big)}$$



Linear Classification



History and Motivation:

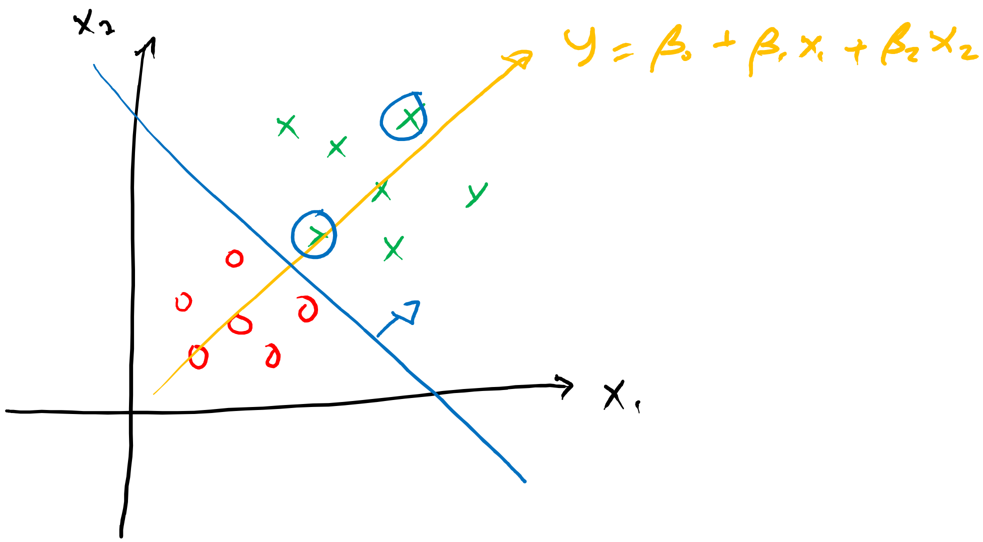
Logistic Regression

Linear Classification

- Intuition:
  - Modelling the relation that best fit the data using a single line
  - Use discriminative (sign) function to decide which class example
  - Find weights that minimize the Cost Function
- Problem:
  - For binary classification, when using linear regression, the examples far from the decision boundary have a huge impact on $\hat{y}$.
  - How to limit their influence?

Linear Classification

- Problem:
  - For binary classification, when using linear regression, the examples far from the decision boundary have a huge impact on $\hat{y}$.
  - How to limit their influence?



Logistic Regression

- Intuition:
  - Modelling the relation that best fit the data using a single line
  - Use discriminative (sign) function to decide which class example
  - Find weights that minimize the Cost Function
- Problem:
  - For binary classification, when using linear regression, the examples far from the decision boundary have a huge impact on $\hat{y}$.
  - How to limit their influence?
- Solution:
  - Use a transformation of the values of linear function

Logistic Regression

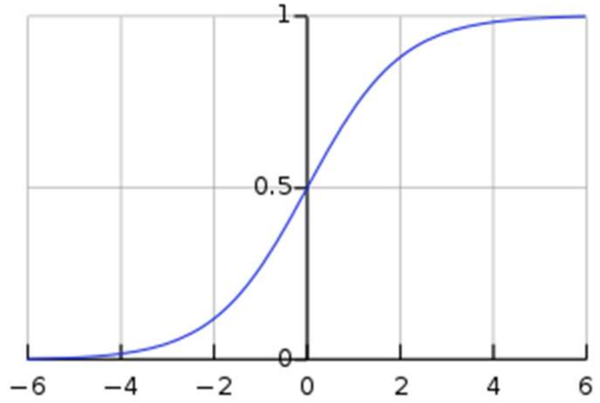
- Limit output $[0 \le f(x) \le 1]$ by inserting the affine function to a sigmoid function

$$\boxed{f(x) = \sigma(wx + \beta)}$$

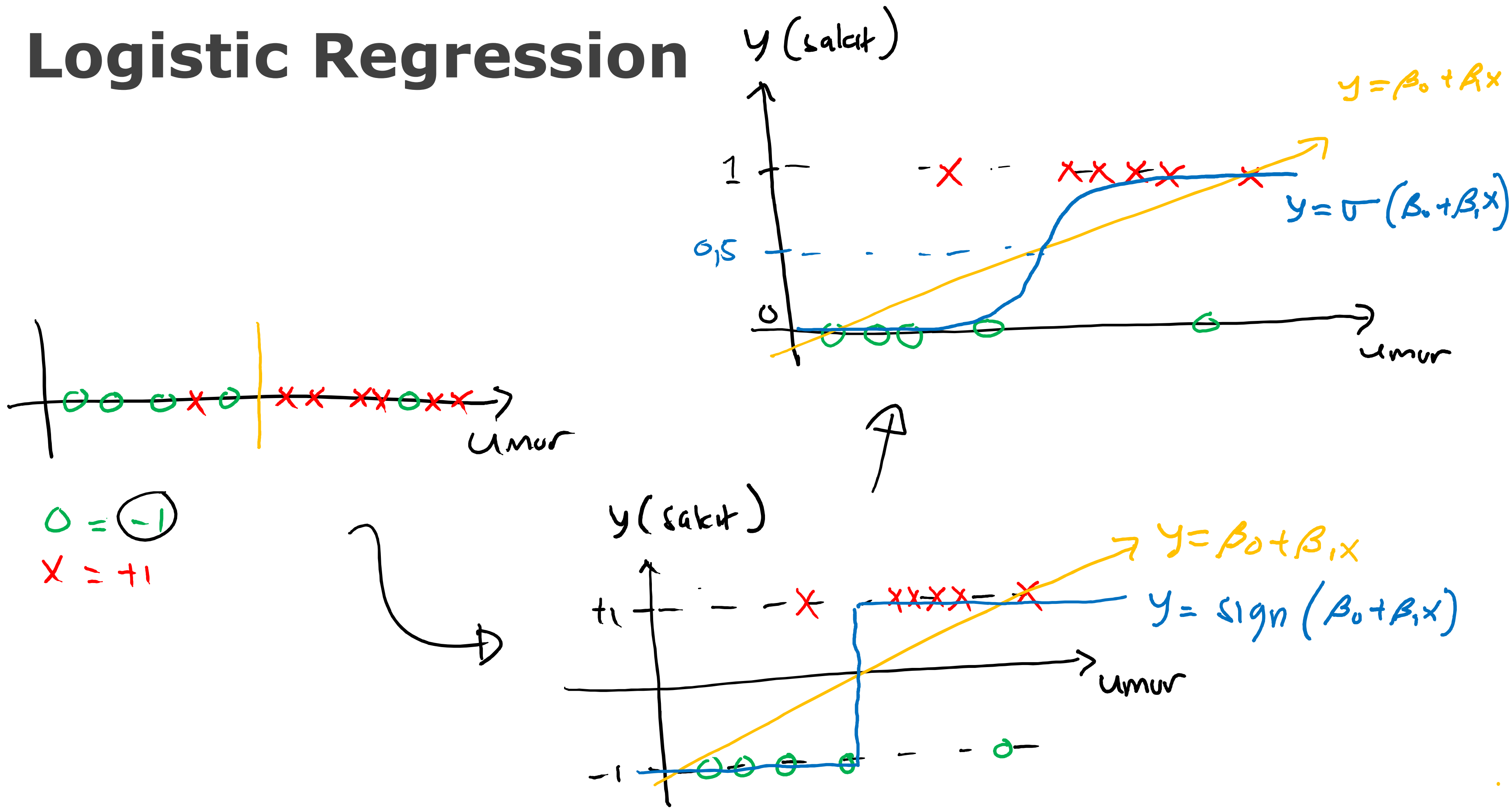
- where

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

  - is the sigmoid function (a.k.a logistic function)

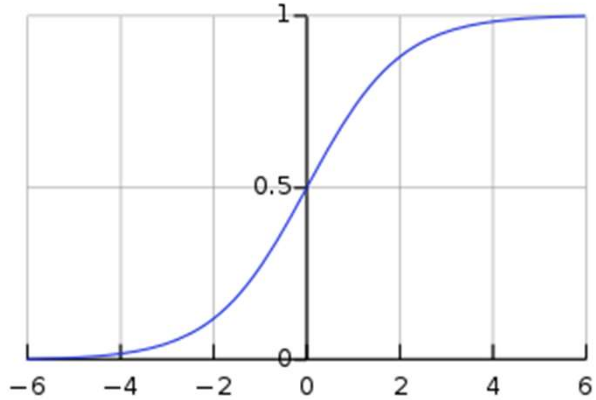


Logistic Regression



Logistic Regression

$$\boxed{f(x) = \frac{1}{1 + e^{-(wx+a)}}}$$



- Interpretation of the model:
  - $f(x)$ estimates the probability that x belongs to class 1.
  - Logistic regression is a classification model
  - The discrimination function $f(x)$ itself is not linear anymore; but the decision boundary is still linear!

History and Motivation:

Linear Regression/Classification is a Weighted Sum

Linear Classification

$$\boxed{y = sign(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d)}$$

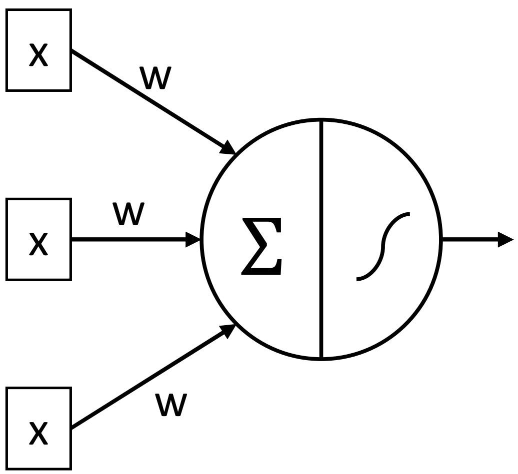
X_1	X_2	X_3	y	
2	1	-3	1	✓
1	1	-2	1	✓
2	-1	-2	0	✓
1	1	-2	1	✓
1	-1	1	0	✓

$y = sign(\quad)$ → $1, > 0$; $0, \leq 0$

$$\underbrace{(1)}_{\beta_1} X_1 + \underbrace{(5)}_{\beta_2} X_2 + \underbrace{(-1)}_{\beta_3} X_3$$

Logistic Regression

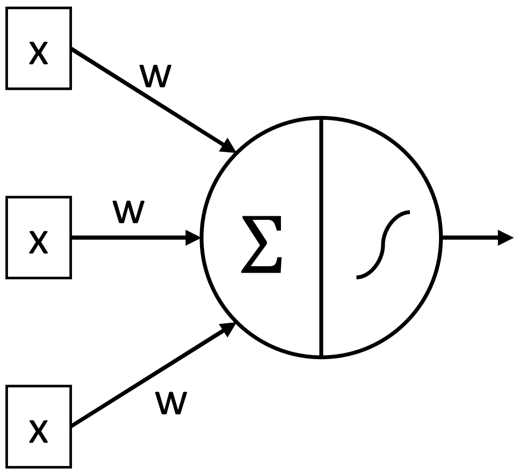
- **Weighted Sum** to determine **YES**(1) or **NO**(0)
  - **Receive** input from outside
  - **Multiplying** by the weight connected to it
  - **Sum** up all of them
  - **Limit** output in the range [0-1]



- If **output total high**, then class = **close to YES**

Logistic Regression

- Weight is a **feature identifier**
  - a value that states the correlation of the attributes associated with it
  - if its input is important (correlated to class 1), then increase it



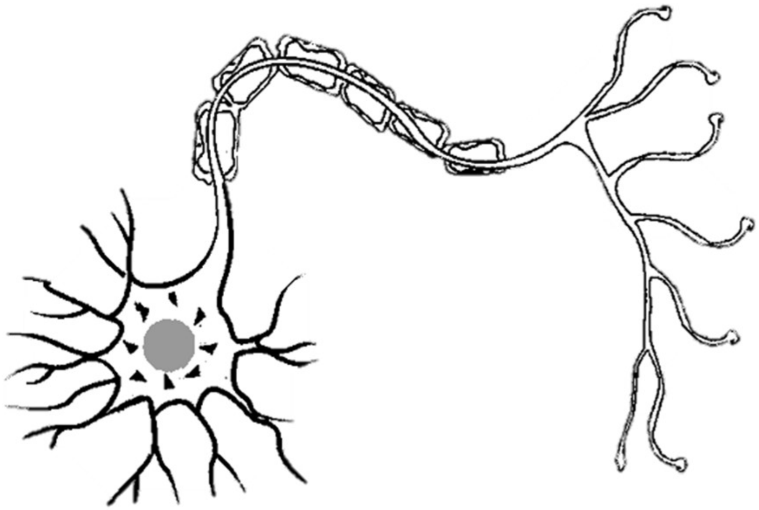
Preview:
Gradient Descent Optimization

Logistic Regression

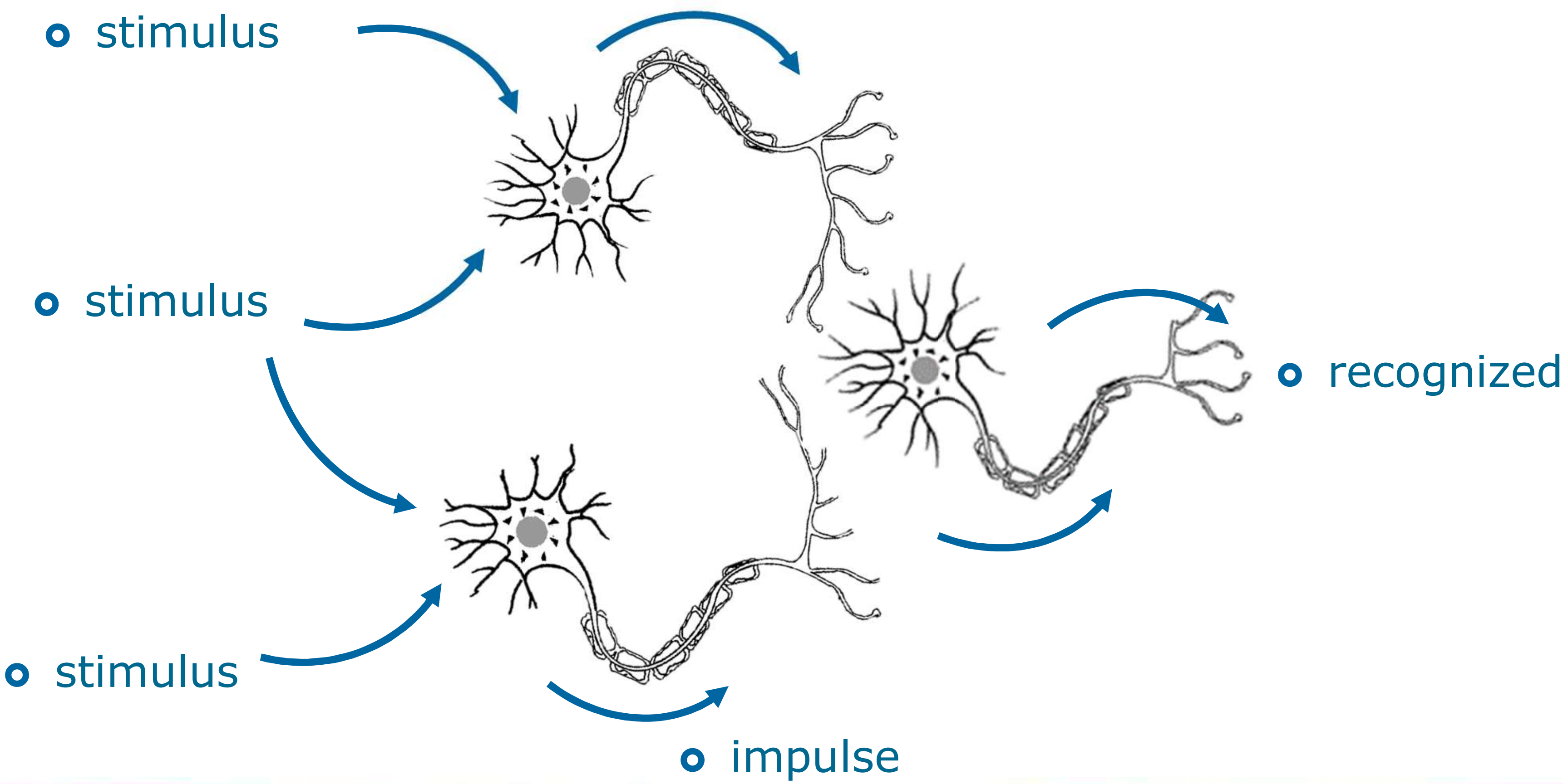
- Intuition:
  - Class score $\hat{y}$ = weighted sum of the attributes (x)
  - Use a transformation of the values of linear function
  - Find weights that minimize the Cost Function
- Problem: $\boxed{\hat{w} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}}$
  - Expensive calculation with the increasing of dimension in x,
  - Need to come up with another technique
- Solution:
  - Gradient Descent Optimization

Artificial Neural Network

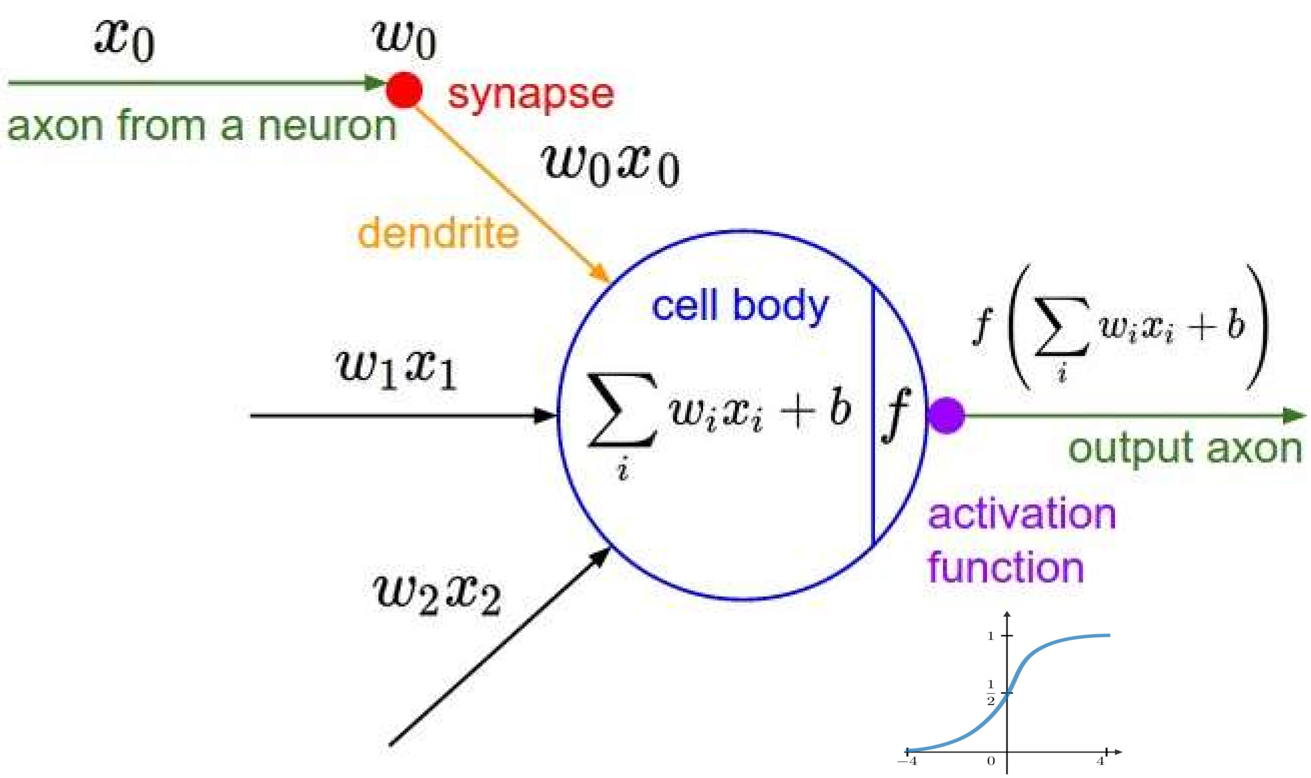
Neuron in our Brain



How Human Brain Works (?)

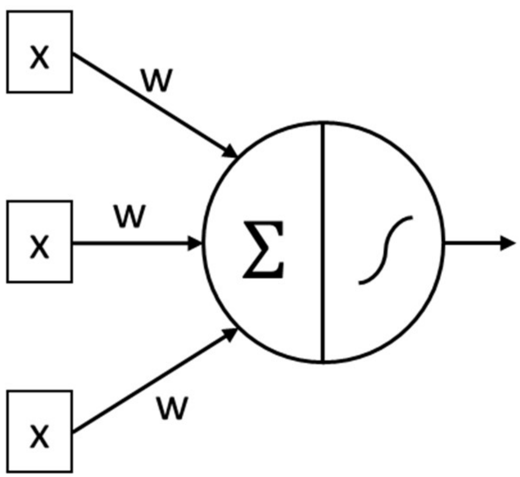


Neurons


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
$\sum_i w_i x_i + b$
f
output axon
activation
function
$w_2 x_2$

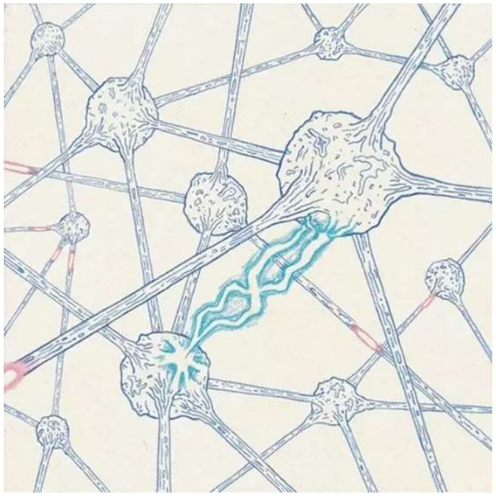
Neuron

- The main component of Neural Network
- Which is, actually, just a simple linear function ("rename" from logistic regression)

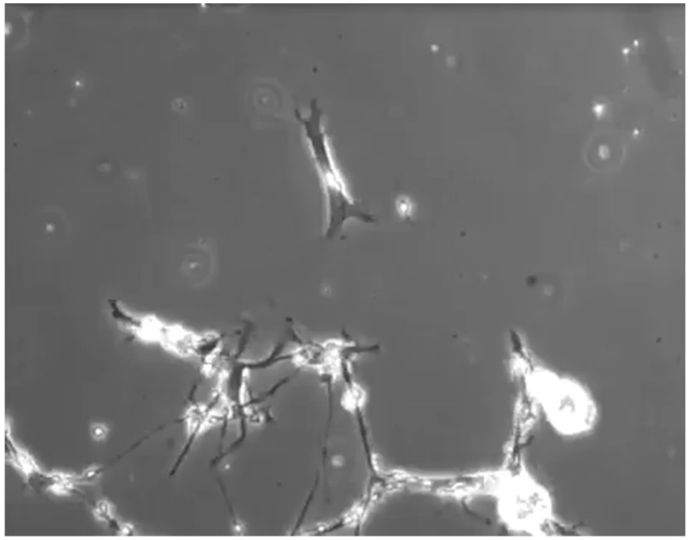


"Brain" analogies

- Be **very careful** with your analogies
- It is **inspired** by how the brain works, but **do not say** that it works like a brain



Biological vs Artificial

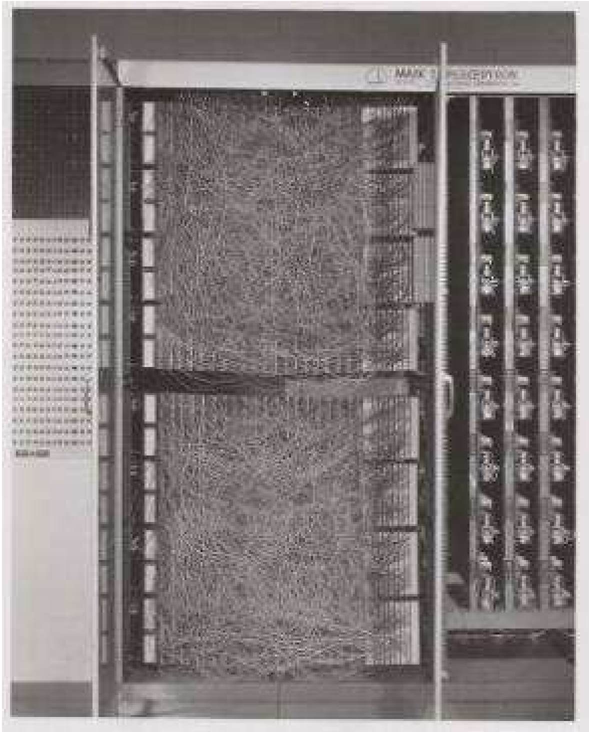


- Biological Neurons:
  - Complex connectivity Pattern
  - Many different types
  - Dendrites can perform complex nonlinear computations
  - Synapses are not a single weight but a complex non-linear dynamical system
  - Rate code may not be adequate

History and Motivation: Mark I Perceptron

Perceptron Algorithm

- Frank Rosenblatt, ~1957:
  - the first implementation of perceptron algorithm
  - The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.
  - Recognized letters of the alphabet



Perceptron Algorithm

- Ad hoc Learning Rule
  - Online or Offline learning
  - Starts with weights initialized to 0 (or to a small random value)
  - For each example at a step,

    calculate $$\boxed{f(x_i) = \begin{cases} 1 & if\ w.x + b > 0 \\ 0 & otherwise \end{cases}}$$

  - Update rule: $w_d(t+1) = w_d(t) + \alpha\big(y_i - \hat{y}_i(t)\big)x_{id}$

    for all features $0 \le d \le D$

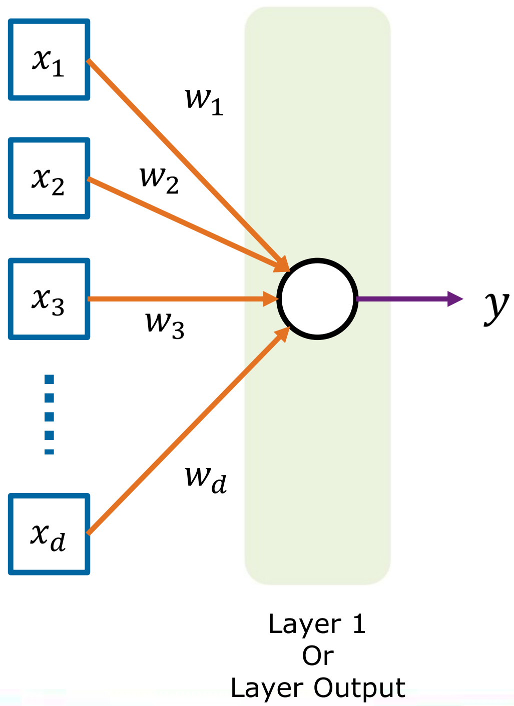
Perceptron Algorithm

- Perceptron algorithm eventually finds a hyperplane that separates 2 classes of points, if such a hyperplane exists.
- If no separating hyperplane exists, the algorithm cannot converge and will iterate forever (until max iteration).

Neural Nets Architectures

Single Layer Perceptron

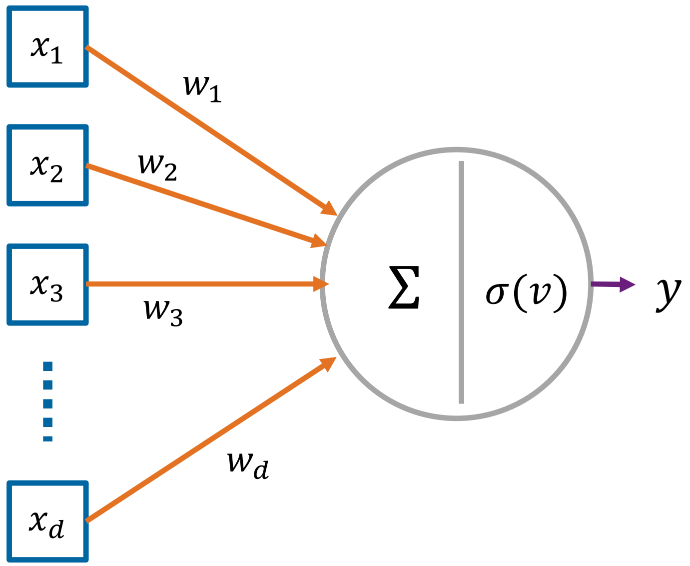
- 1-layer processing
  - 1 layer containing neuron set
  - Input layer is not counted as Layer
  - If it only has a single output neuron, it can be seen as single linear function



Single Layer Perceptron

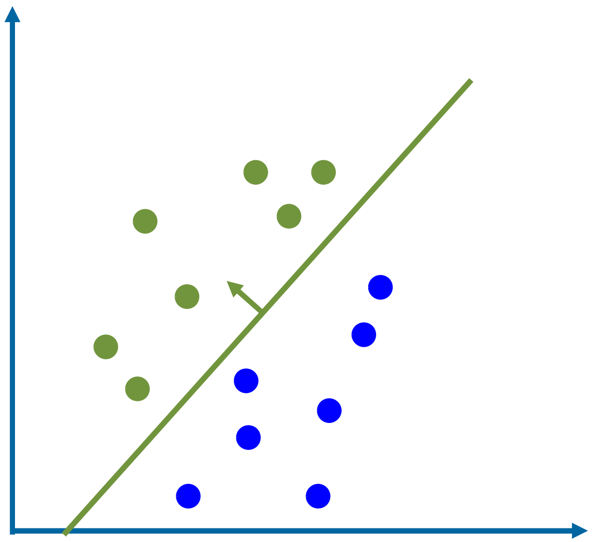
- Input: $X \in \mathbb{R}^d$
- Output: $y \in \{0,1\}$

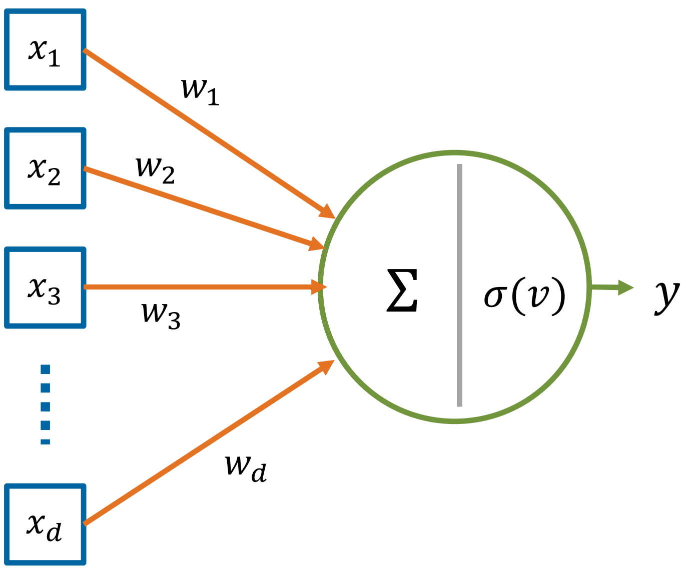
i	x_1	x_2	…	x_d	y
1	0.5	0.2	…	0.7	1
2	0.1	0.3	…	0.2	0
3	0.2	0.6	…	0.8	1
…	…	…	…	…	…
…	…	…	…	…	…
N			…		1



Single Layer Perceptron

- Input: $X \in \mathbb{R}^d$
- Output: $y \in \mathbb{R}^1 \{0,1\}$





Multiclass Classification

Multiclass Single Layer Perceptron

- Input: $X \in \mathbb{R}^d$
- Output: $y \in \{1, 2, \dots, C\}$

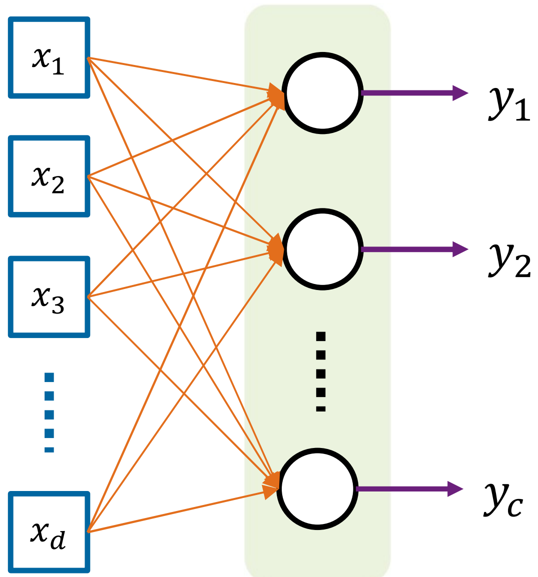
i	x_1	x_2	...	x_d	y
1	0.5	0.2	...	0.7	1
2	0.1	0.3	...	0.2	2
3	0.2	0.6	...	0.8	1
4	0.2	0.5	...	0.2	3
...	...	...	...	...	...
N			...		3

- Input: $X \in \mathbb{R}^d$
- Output: $y \in C \times \{0,1\}$

i	x_1	x_2	...	x_d	y_1	y_2	y_3
1	0.5	0.2	...	0.7	1	0	0
2	0.1	0.3	...	0.2	0	1	0
3	0.2	0.6	...	0.8	1	0	0
4	0.2	0.5	...	0.2	0	0	1
...	...	...	...	...	...		
N			...		0	0	1

Multiclass Single Layer Perceptron

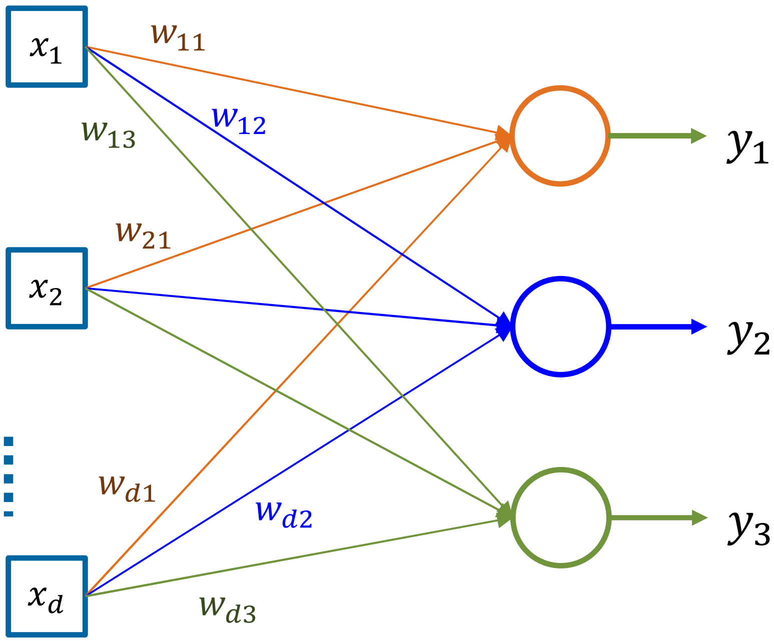
- 1-layer processing
  - 1 layer containing neuron set
  - If it has multiple neurons, it can be seen as multiple linear functions



Multiclass Single Layer Perceptron

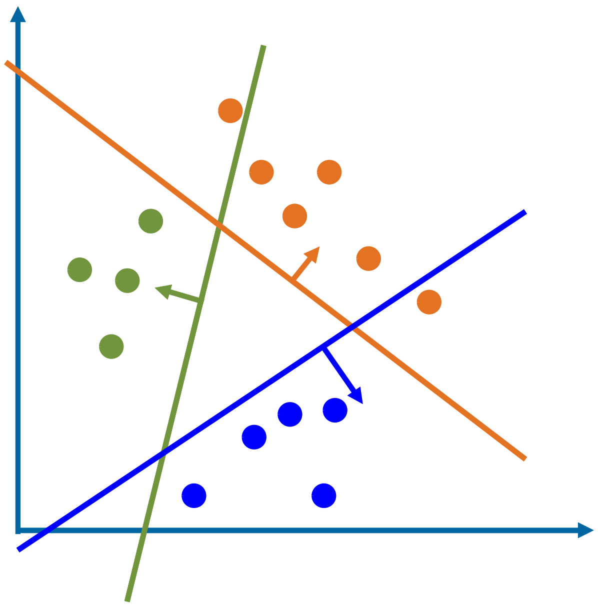
- Input: $X \in \mathbb{R}^d$
- Output: $y \in \mathbb{R}^1\{1, 2, 3\}$

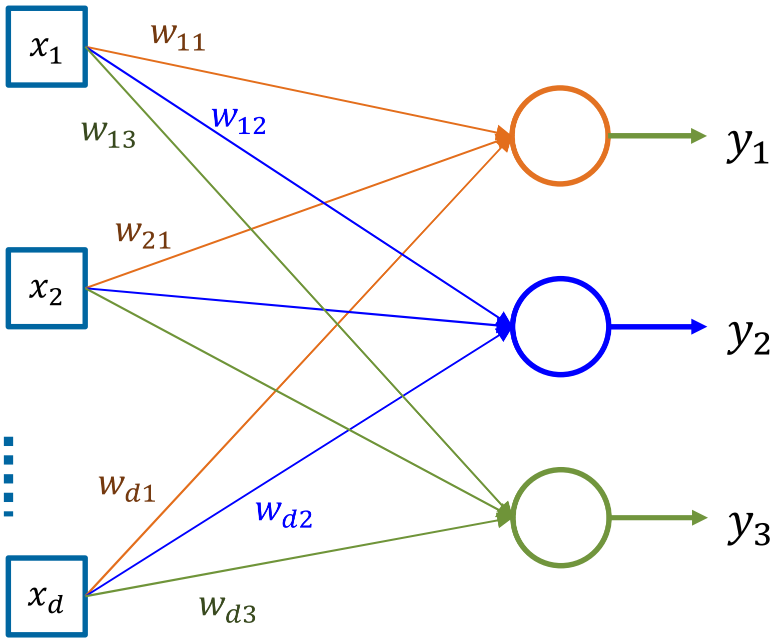
i	x_1	x_2	…	x_d	y_1	y_2	y_3
1	0.5	0.2	…	0.7	1	0	0
2	0.1	0.3	…	0.2	0	1	0
3	0.2	0.6	…	0.8	1	0	0
4	0.2	0.5	…	0.2	0	0	1
…	…	…	…	…	…		
N			…		0	0	1



Multiclass Single Layer Perceptron

- Input: $X \in \mathbb{R}^d$
- Output: $y \in \mathbb{R}^1\{1, 2, 3\}$

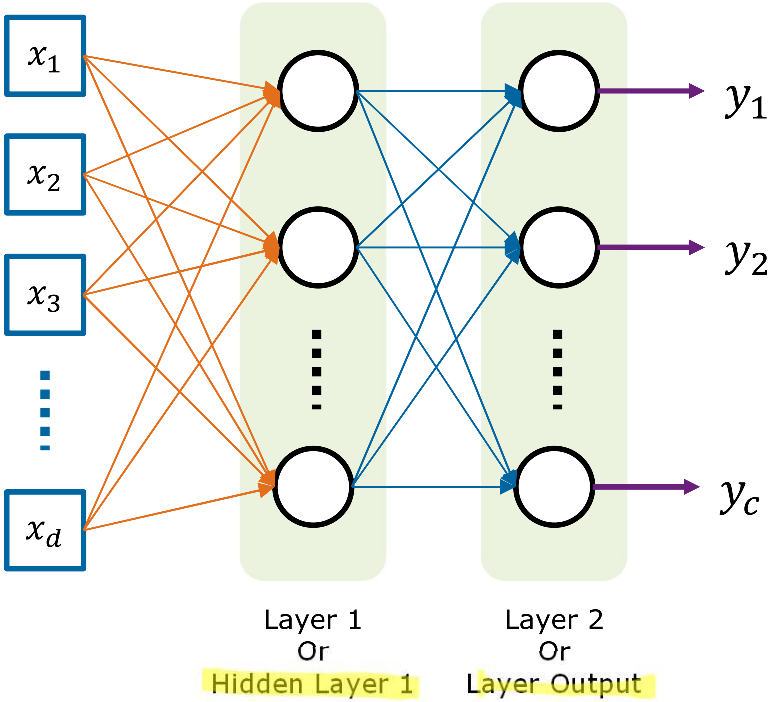




Multi Layer Perceptron

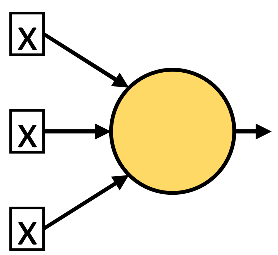
Multi Layer Perceptron

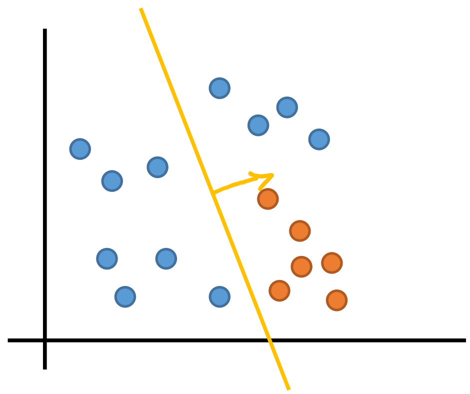
- N layer processing
  - Stacked of layers containing neuron set
  - Each connection between neuron is precedented by a non-linear function
  - The intermediate layer(s) behind output layer is often called hidden layer



Multi Layer Perceptron

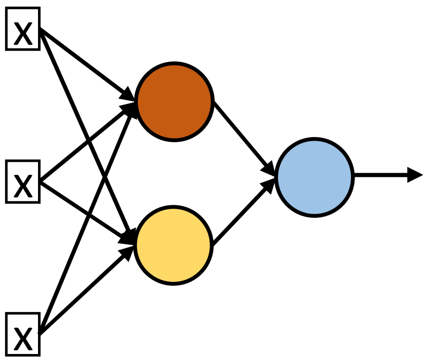
- Before, if one neuron was able to make a line to divide class 1 and 0,

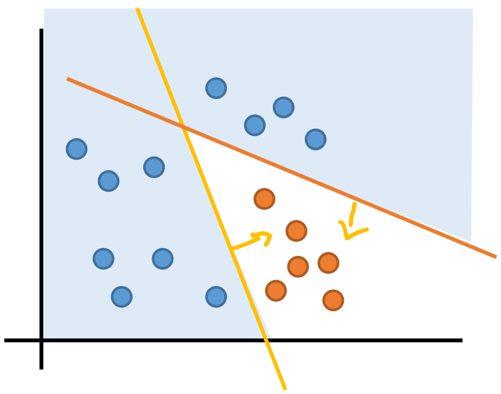




Multi Layer Perceptron

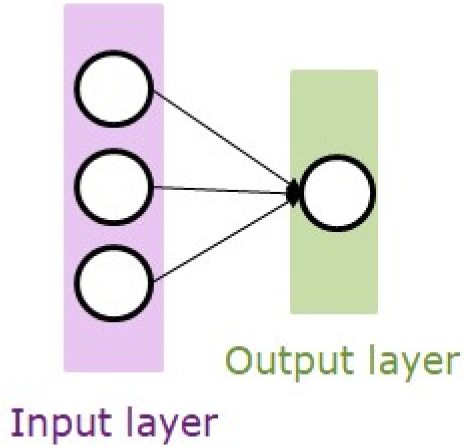
- Then two neurons can create two lines
- And combine them back to a single classification



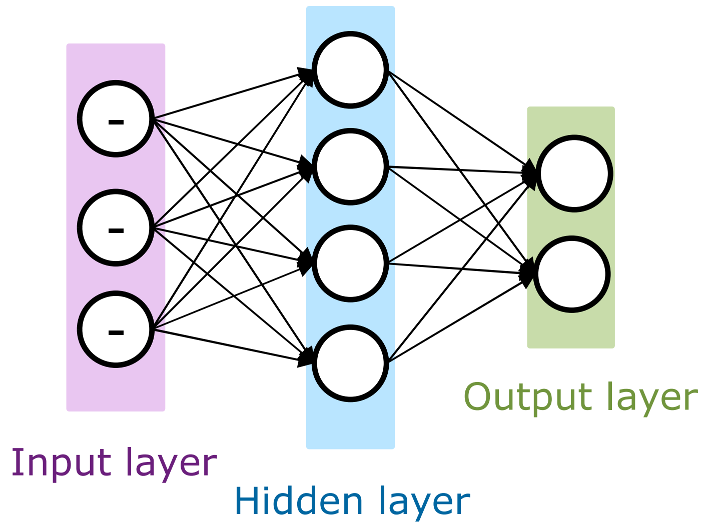


Neural Network

- (**Before**) Linear Score Function
  - $\hat{y} = W.x$



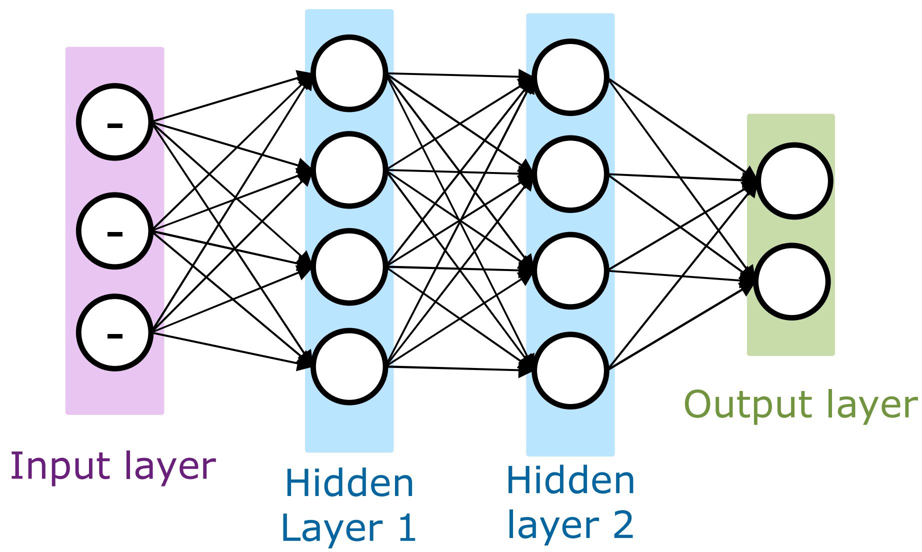
- (**Now**) 2-Layer Neural Network
  - $\hat{y} = W_2.f(W_1.x)$



Neural Network

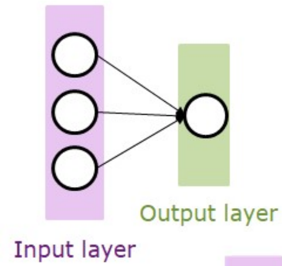
- (and further) 3-Layer Neural Network

$$-\hat{y} = W_3 . f(W_2 . f(W_1 . x))$$

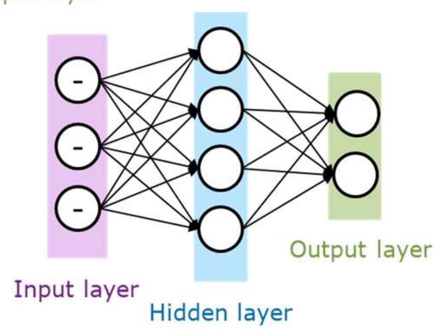


Neural Network Naming

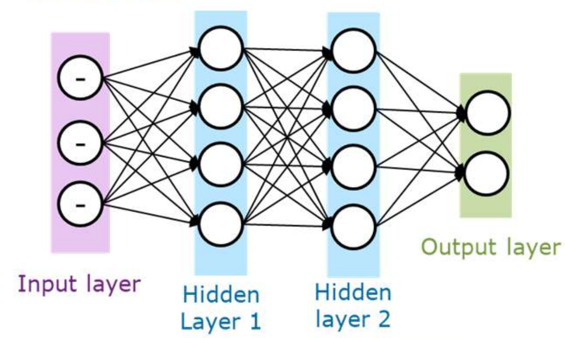
- 1-layer Neural Net
  - Single Layer Perceptron



- 2-layer Neural Net
  - 1 Hidden Layer Neural Net

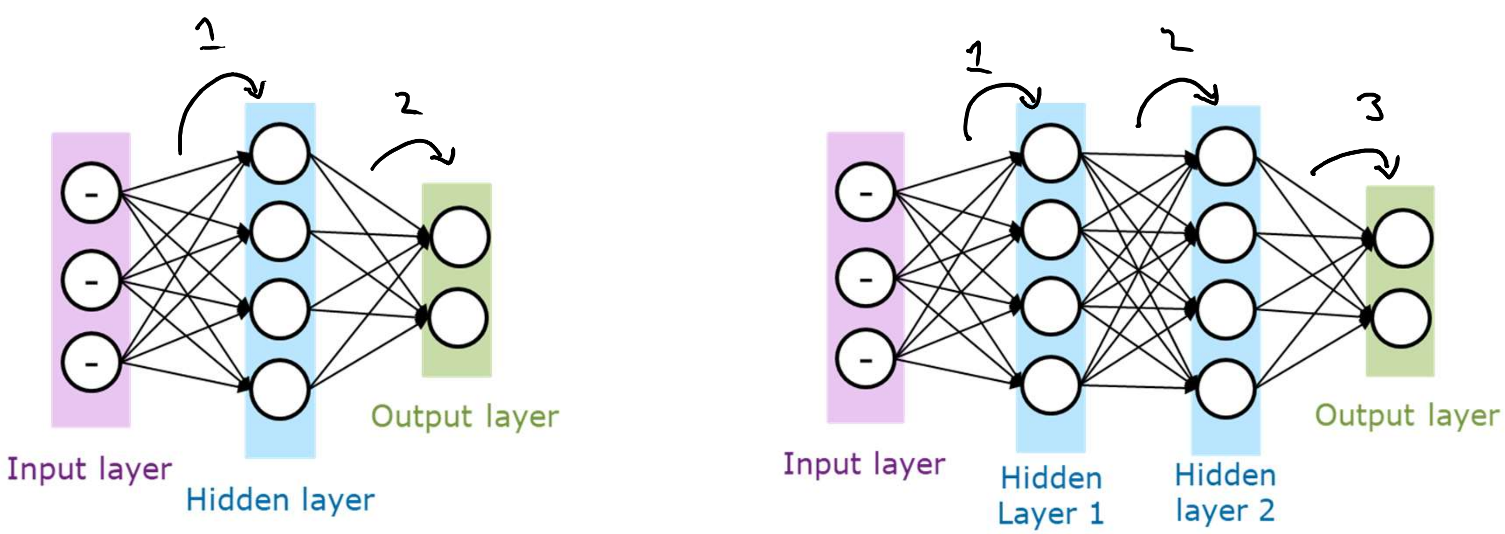


- 3-layer Neural Net
  - 2 Hidden Layer Neural Net
  - And so on



Neural Network Naming

- Layer is where the weights attached
- A network with two or more layers is referred to as a Multi-Layer Perceptron (MLP)



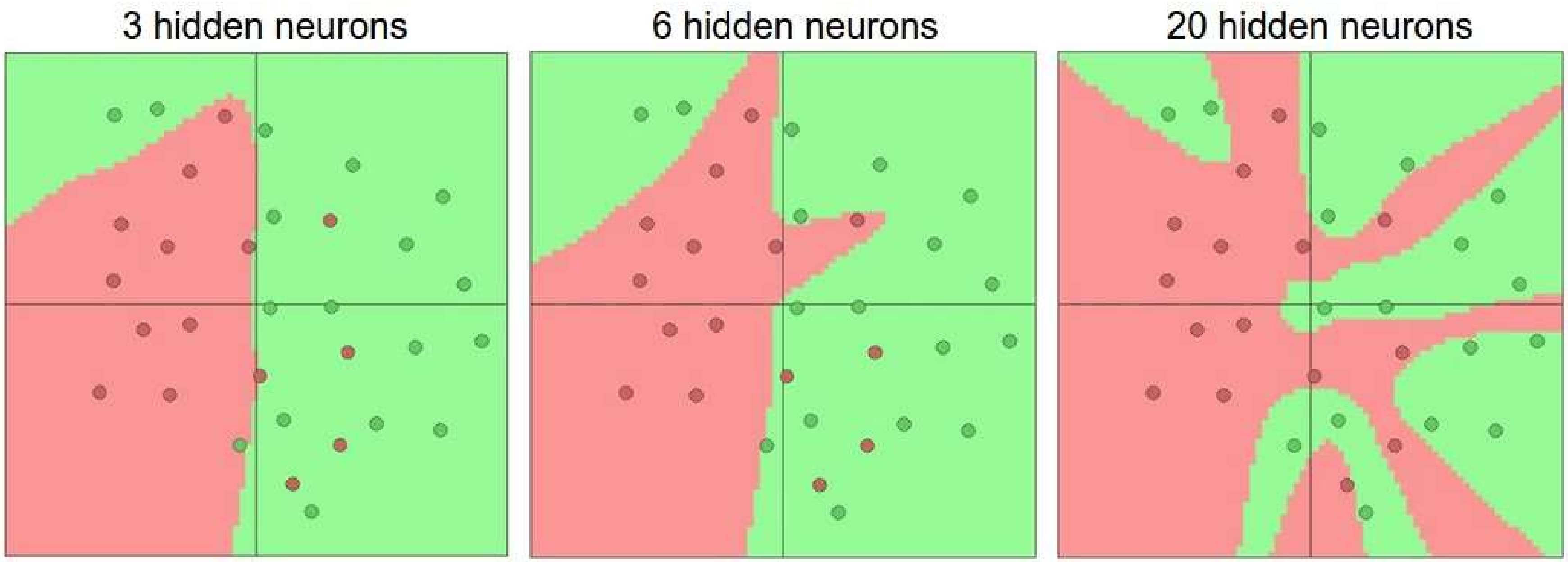
Layer Number and Sizes

Layer Number and Size

- Increase the size and number **==** increase the network capacity
  - Neural Networks with more neurons can express more complicated functions
  - Can learn to classify more complicated data

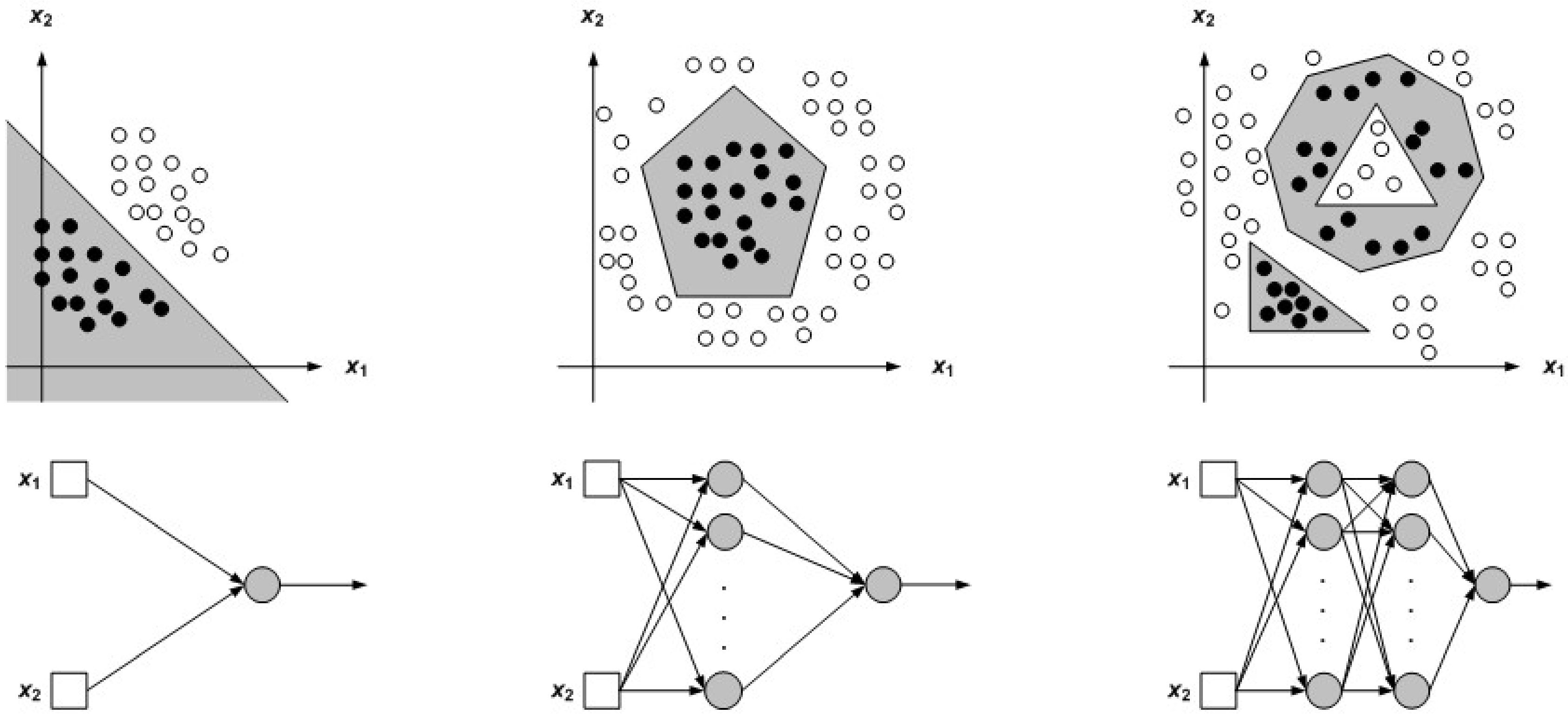
- Tradeoff:
  - More likely to overfit the training data

Layer Number and Size



more neurons = more capacity

Layer Number and Size



Neural Network Architectures

- Single Layer Perceptron
- Multi Layer Perceptron
  - Basic Neural Network Architecture
  - Feed Forward Neural Network
  - Deep Neural Network
- Radial Basis Function Neural Network
- Recurrent Neural Network
- Convolutional Neural Network

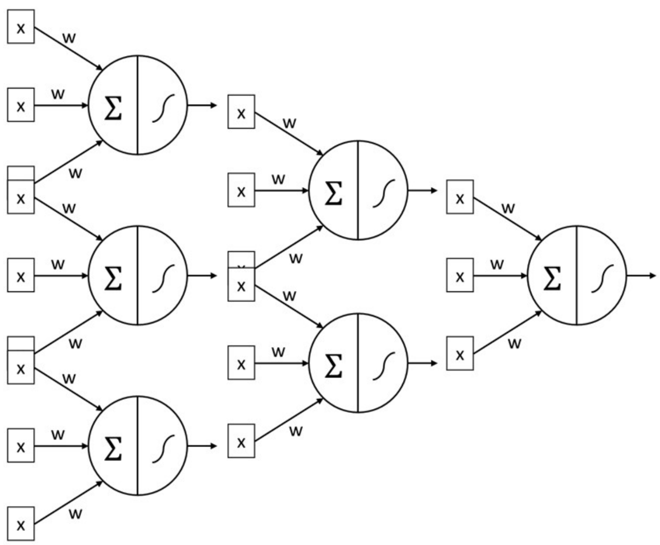
Neural Network Architectures

- Boltzmann Machine
- Hopfield Network
- Deep Belief Network

Activation Functions

Neural Network

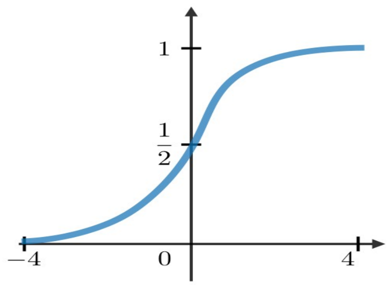
- Neural Network is just a series (stacks) of neuron
- Each connection between neuron is precedented by a non-linear function

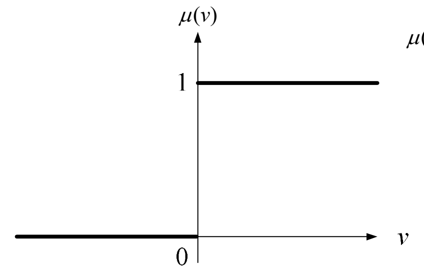


Activations Functions

- **Sigmoid**

$$\frac{1}{1+e^{-(v)}}$$

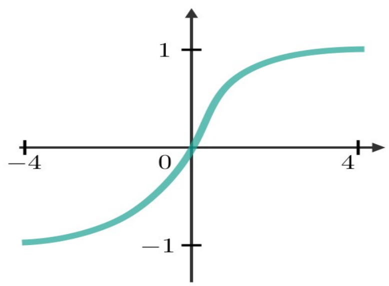


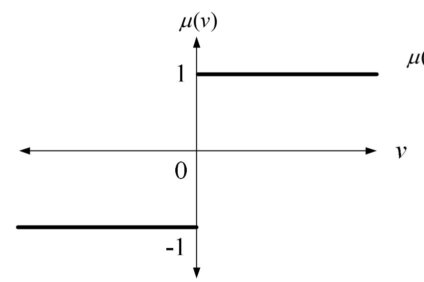


$$\mu(v)=\begin{cases}1 & \text{if } v>0\\ 0 & \text{if } v\le 0\end{cases}$$

- **Tanh**

$$\tanh(x)$$

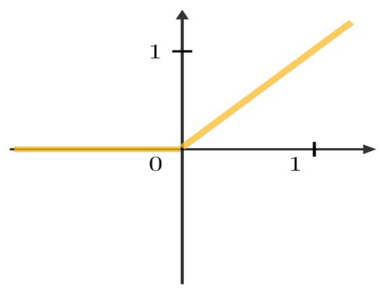


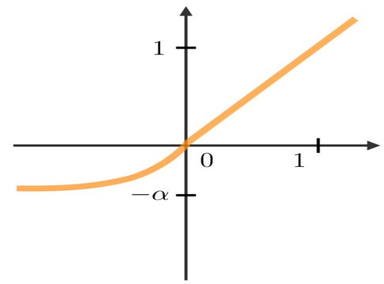


$$\mu(v)=\begin{cases}1 & \text{if } v>0\\ 0 & \text{if } v=0\\ -1 & \text{if } v<0\end{cases}$$

- **ReLU**

$$\max(0,x)$$





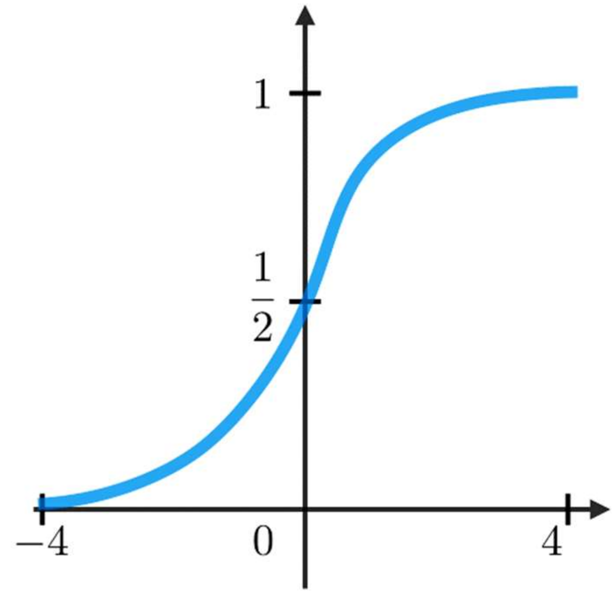
Sigmoid Function

- Forward function

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

- Backward function

$$\boxed{\sigma'^{(x)} = \sigma^{(x)} - \sigma(x)^2}$$



- Squashes numbers to range [0, 1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

Tanh Function

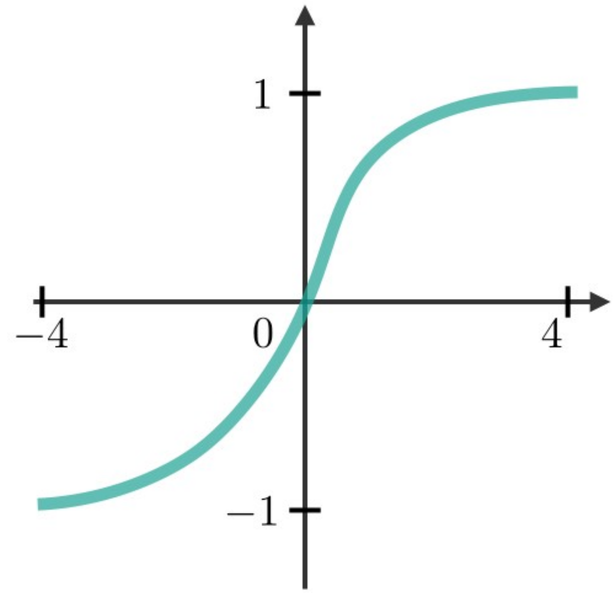
- Forward function

$$\boxed{f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \tanh(x)}$$

- Backward function

$$\boxed{f'^{(x)} = 1 - \tanh^2(x)}$$

- Squashes numbers to range [-1, 1]



[LeCun et al., 1991]

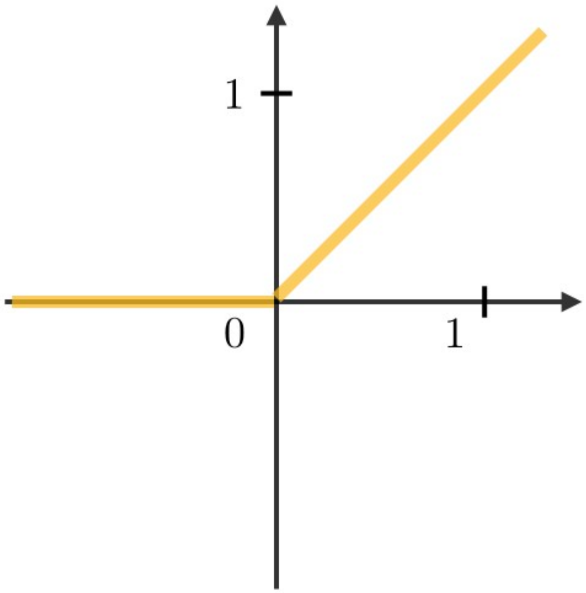
Rectified Linear Unit

- Forward function

$$\boxed{f(x) = \max(0, x)}$$

- Backward function

$$\boxed{f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}}$$



- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice

[Krizhevsky et al., 2012]

Why use Activation Function?

- Without activation function, the neural network will be just a single linear sandwich
- The capacity is the same as just a linear classifier

$$x * w_1 = a * w_2 = y \quad \equiv \quad x * w_1 * w_2 = y$$

Question?

Next Agenda

- Neural Network Training
- Gradient Descent
- Backpropagation
- Advanced Techniques

Next Week


Fakultas Informatika
School of Computing
Telkom University

THANK YOU