Artificial Neural Network

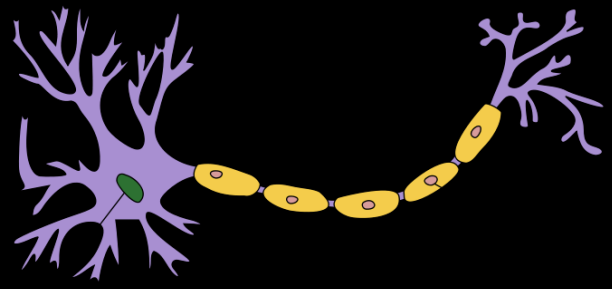
Il-Chul Moon
Dept. of Industrial and Systems Engineering
KAIST

icmoon@kaist.ac.kr

Weekly Objectives

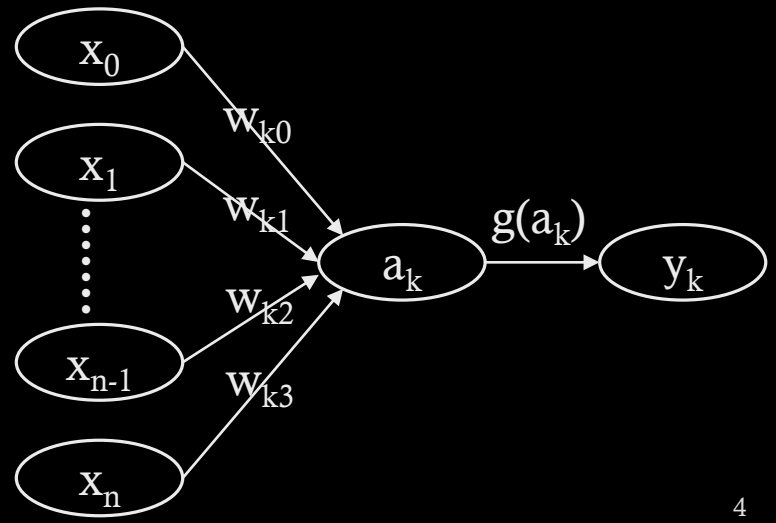
- Learn the perceptron and the neural network
  - Understand the mechanism of single perceptron
  - Know the structure of the single perceptron
  - Understand the roles of activation and pre-activation functions
  - Understand the limitation of the single perceptron and the need of hidden layers
- Know how to infer the parameters of artificial neural network
  - Know the loss function of neural networks
  - Derive the gradients of functions in neural networks
  - Apply the gradient descent algorithm to infer the parameters
  - Understand the back-propagation algorithm to train the neural networks

Perceptron and Neural Network



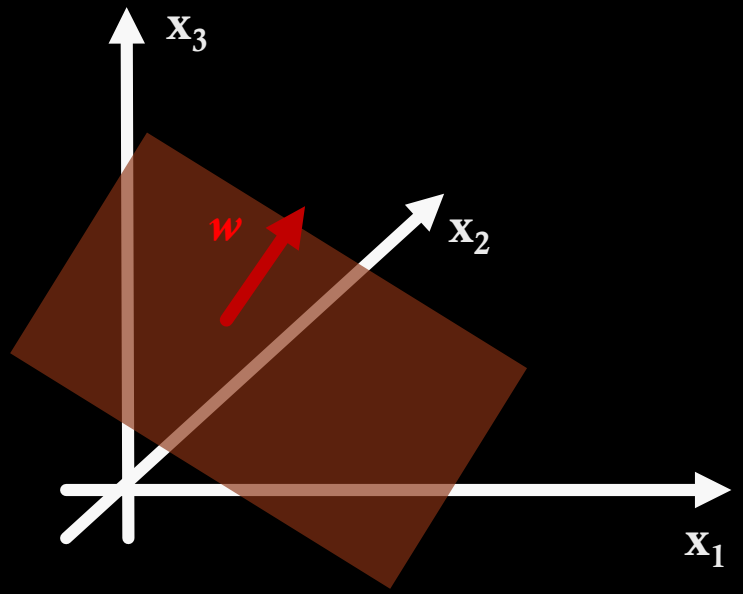
Artificial Neuron

- Neuron
  - Electrically excitable cell that processes and transmits information through electrical and chemical signals
- Artificial neuron
  - Mathematical function that can be activated through processing information
- Neuron input activation
  - $a_k(X) = \sum_{0 \le i \le n} w_{ki} x_i$
  - To represent the constant, or the bias
    - $x_0 = 1$ and $w_{ki} = b_k$
- Neuron output activation
  - Can be a function from diverse choices
  - Sigmoid functions are prominent choice



Neuron Input Activation Function

- Neuron input activation
  - $a_k(X) = \sum_{0 \le i \le n} w_{ki} x_i$
  - To represent the constant, or the bias
    - $x_0 = 1$ and $w_{ki} = b_k$
- $a_k(X) = {w_k}^T X + b_k$
  - Same representation
  - Here, k represents the number of input activation functions
  - Lead to K neurons in the layer
    - $A(X) = WX + B$
  - Each neuron has a different orientation and a different bias



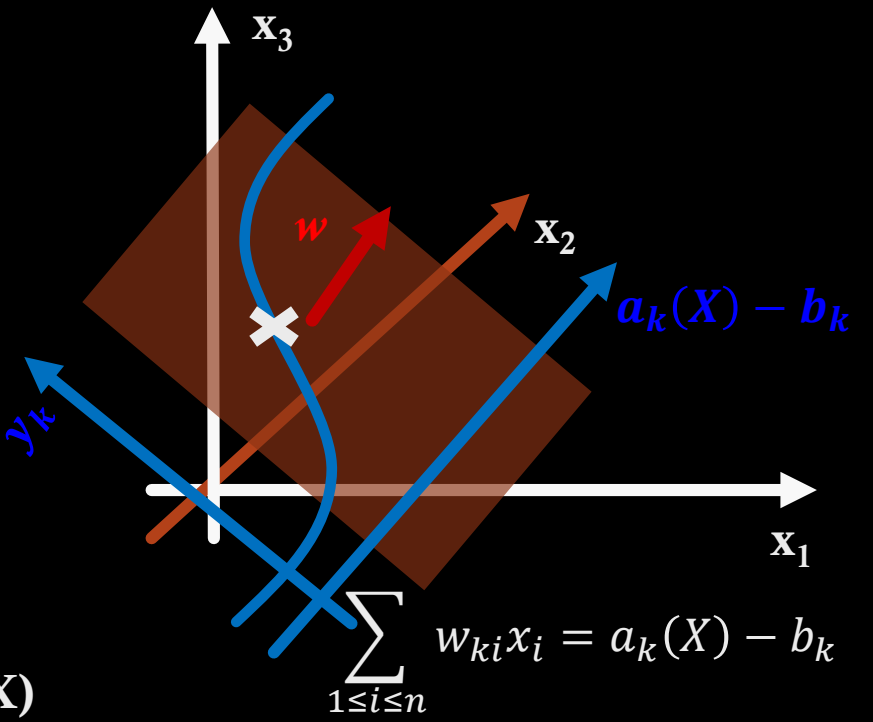
$$\sum_{1 \le i \le n} w_{ki} x_i = a_k(X) - b_k$$

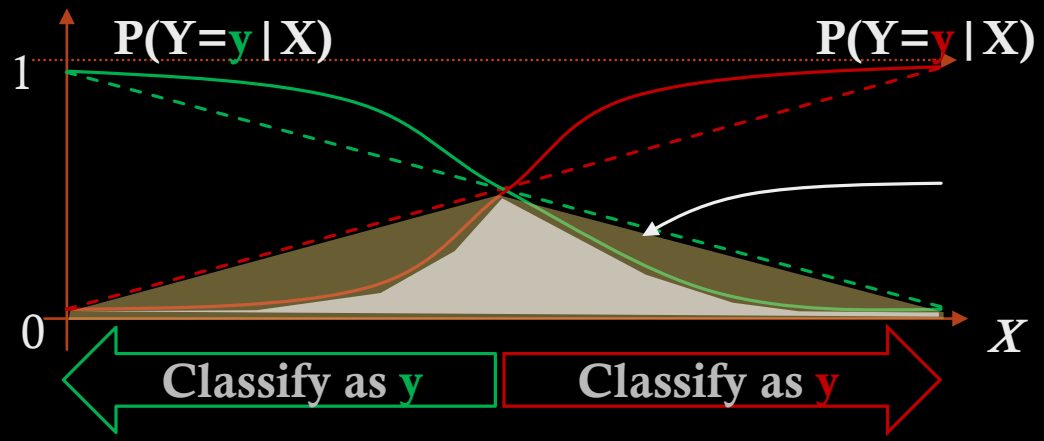
Adjusting the hyperplane orientation

Adjusting the hyperplane location

Neuron Output Activation Function

- $g(a_k(X)) = y_k$
  - Output activation function is typically a single format in a neural network
  - The output activation function can be chosen from many candidates
  - Often to be a sigmoid function
    - Why?





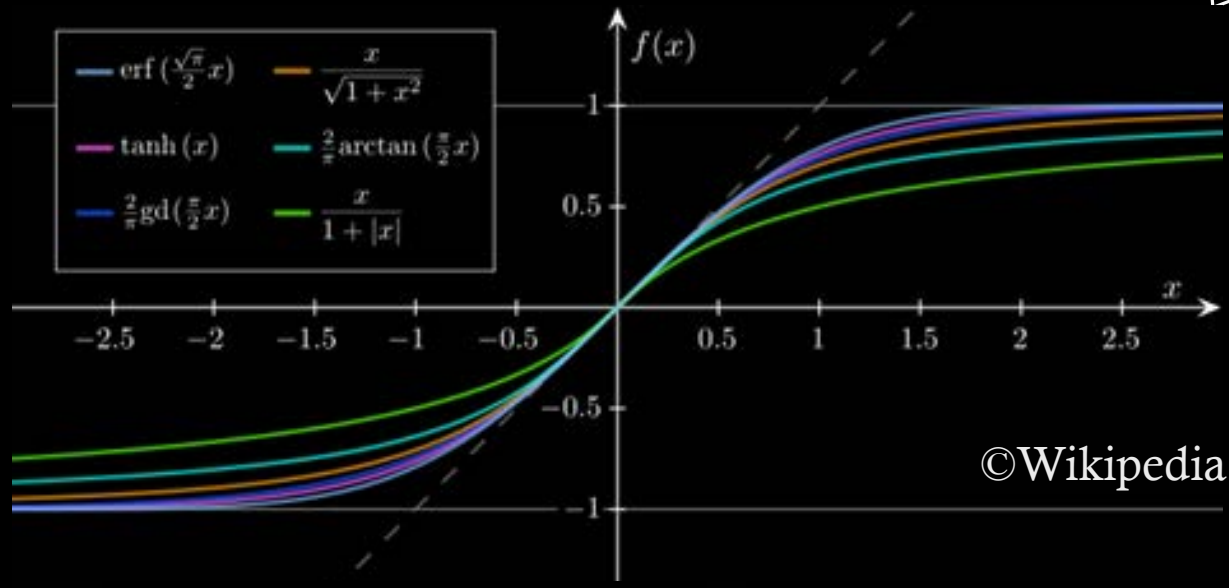
- The sigmoid function shows the S-curve ranged and sharply adapted.
- The output gets different by the orientation of the hyper plane

Neuron Output Activation Function: $g(a_k(X)) = y_k$

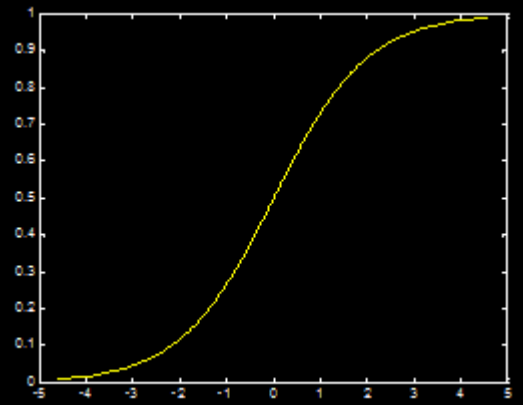
Detour: Logistic function

Many types of sigmoid functions

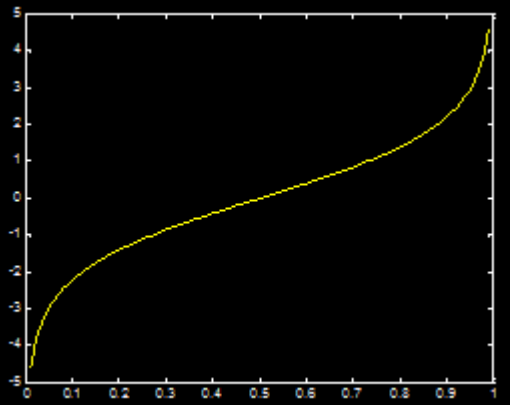
- Sigmoid function is
  - Bounded
  - Differentiable
  - Real function
  - Defined for all real inputs
  - With positive derivative
- Logistic function is
  - $g(x) = \frac{1}{1+e^{-x}}$
  - In relation to the population growth
  - Why is this good?
    - Sigmoid function
    - Particularly, easy to calculate the derivative…



©Wikipedia



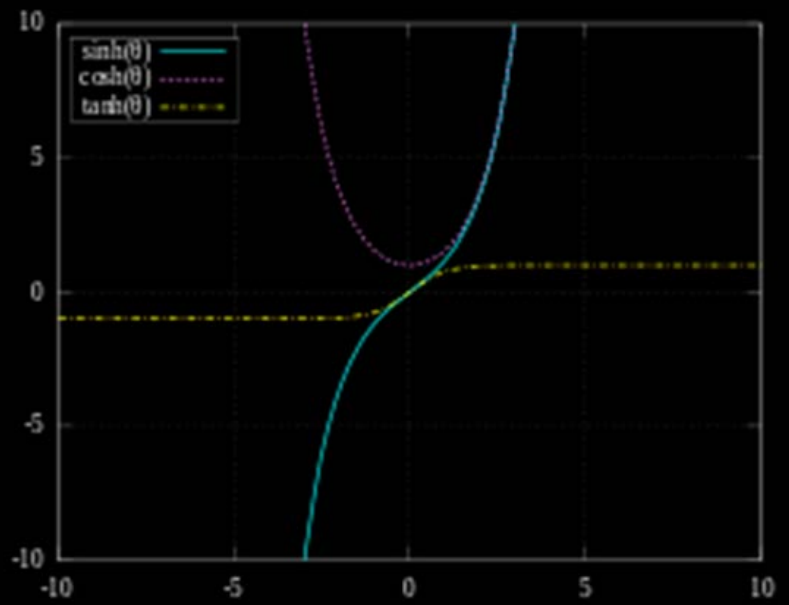
Logistic Function

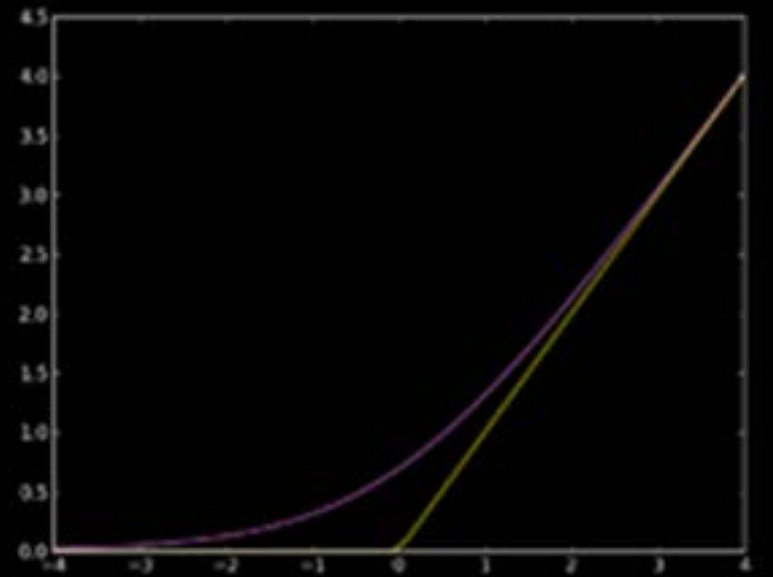


Logit Function, $g(x) = \log(\frac{x}{1-x})$

Hyperbolic Tangent Function and Rectified Linear Function

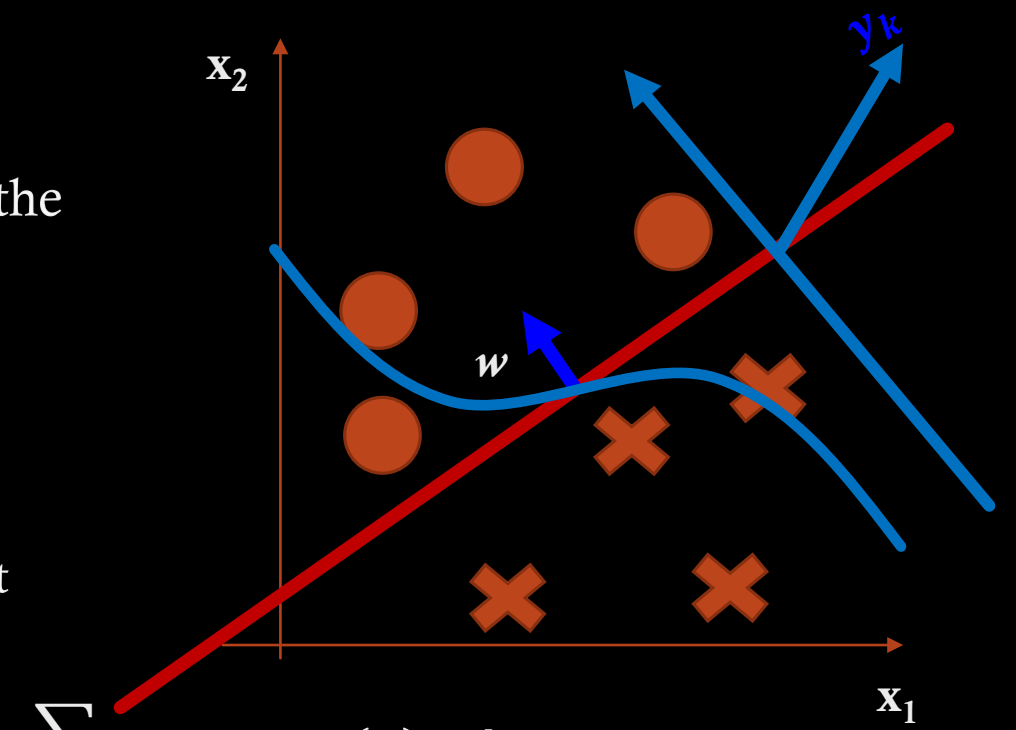
- Hyperbolic tangent function
  - A type of sigmoid function
  - Ranged from -1 to 1
    - Unlike the logistic function from 0 to 1
    - This function can result in negative
  - Strictly increasing
    - Continuous
  - $g(\mathrm{x}) = \tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$
- Rectified linear activation function
  - A.k.a. Rectifier in the neural network field
  - $g(x) = \max(0, x)$
  - Biologically plausible
    - Actual neurons in a brain rarely reach the maximum saturation
  - Smoothed version is
    - Softplus function: $g(x) = \ln(1 + e^x)$
    - Derivation of softplus: $g'(x) = \frac{1}{1 + e^{-x}}$
  - Sparse activation of hidden neurons → Deep neural net





Classification with Neuron

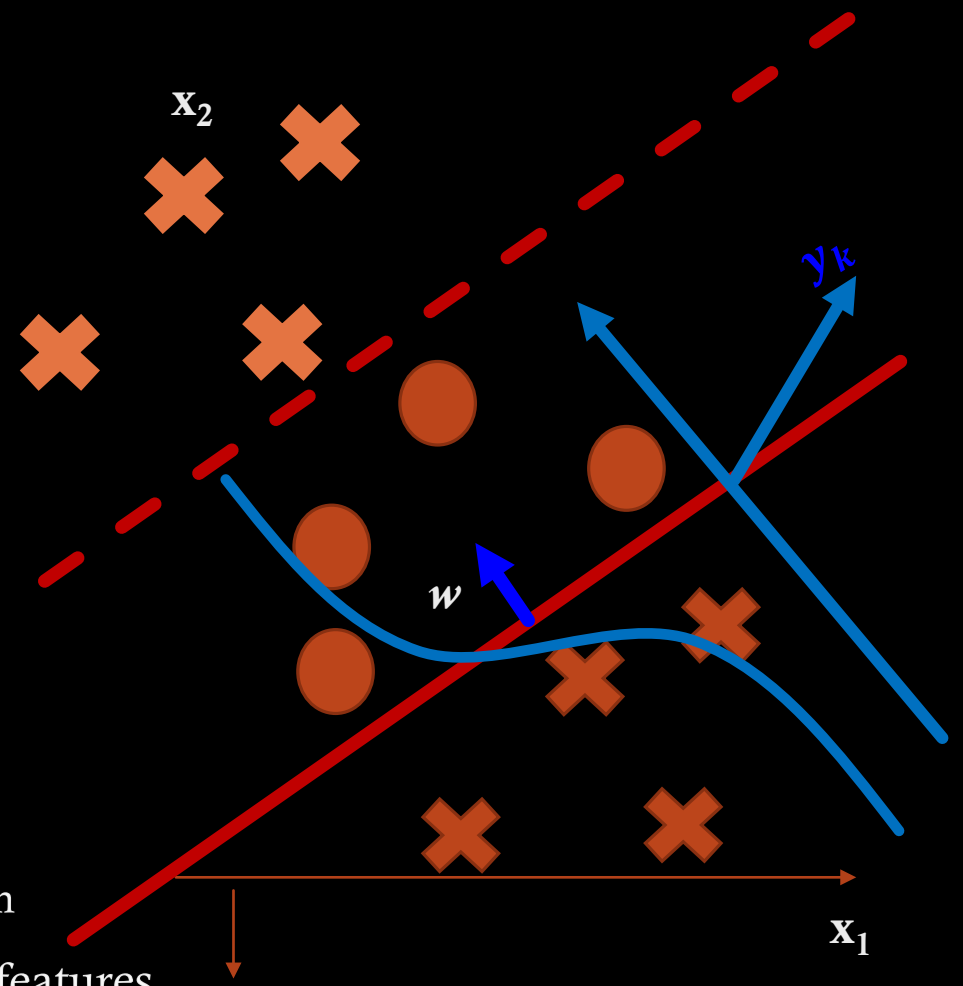
- Neuron can be used to a binary classification with a linear decision boundary
  - Sigmoid function
    - Good analogy to P(y=1 | X)
    - E.x. Logistic regression
- If a neuron output is above than the half of the range, it classifies the instance as positive
  - If we used the logistic function
    - Positive above 0.5
  - If we used the hyperbolic tangent function
    - Positive above 0
- Linear decision boundary



$$\sum_{1 \le i \le n} w_{ki} x_i = a_k(X) - b_k$$

Problem of Linear Decision Boundary

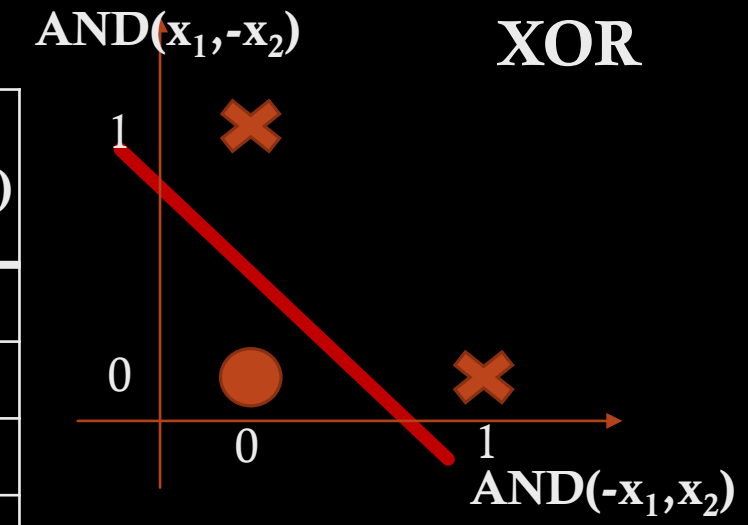
- Consider the sandwich case
  - The orange X marks cannot be classified as the negative by the model limitation
- Many real world applications have non-linear cases
  - Any cases with peak as positive
    - Then, the before and the after around the peak are negative → Non-linear boundary
- Ways to make the boundary non-linear
  - Make the model more complex
    - Change the neuron input activation function
  - Apply the model to the higher dimension features



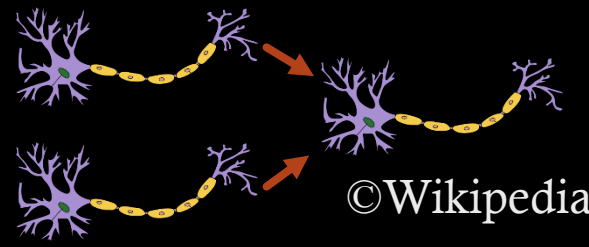
Linear Boundary and Logic Operator

- Ways to make the boundary non-linear
  - Apply the model to the higher dimension features
- XOR(x,y)
  - x==y→Same → Negative
  - x!=y→Different → Positive
  - Clear example of non-linear decision boundary
- What if
  - AND(x,-y) OR AND(-x,y)
  - Equivalent to the linear decision boundary

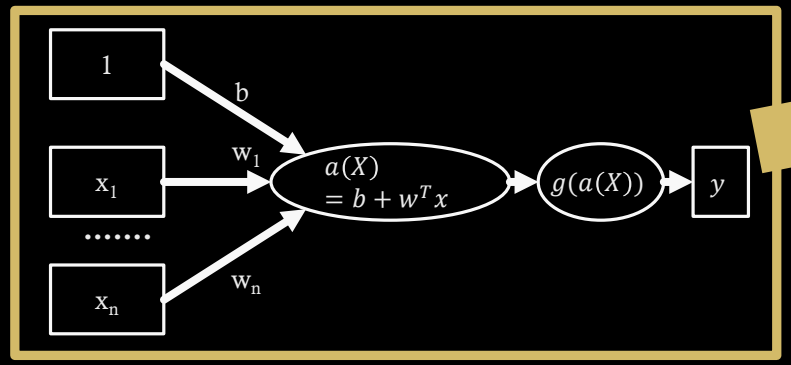
X	Y	XOR	AND (X,-Y)	AND (-X,Y)	AND(x,-y) OR AND(-x,y)
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	0	0

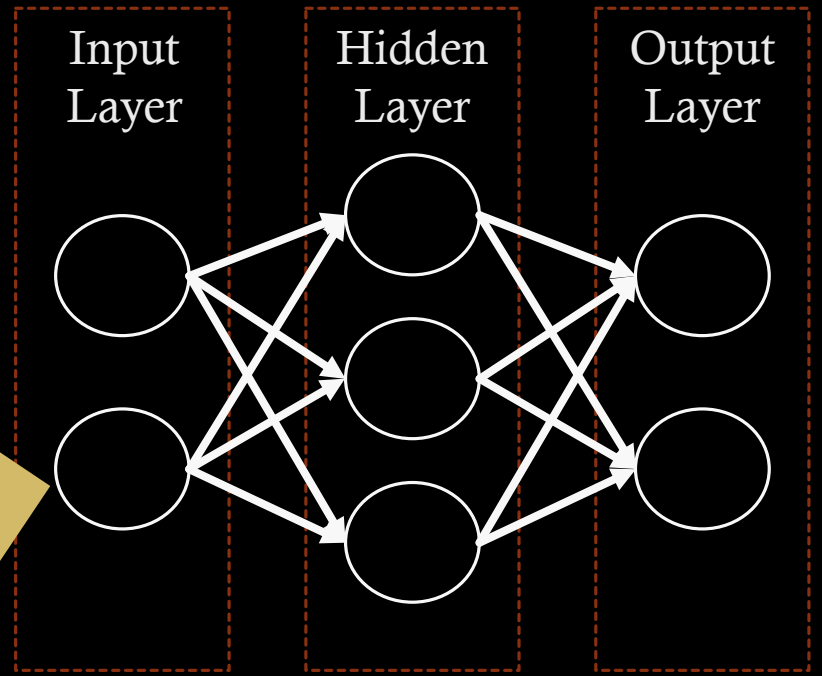


Artificial Neural Network



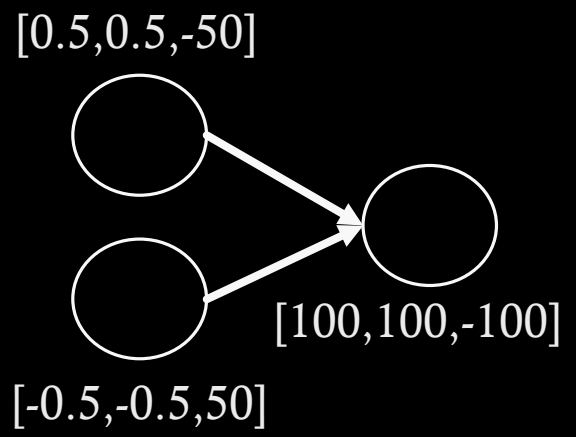
- To overcome the linear decision boundary of a single perceptron, we use multiple perceptrons structured in a certain form.
- Artificial neural network
  - A function approximation method with a collection of interconnected perceptrons
  - Structured by stacking layers
    - Input layer
    - Hidden layer
    - Output layer
  - Can create a multiple output

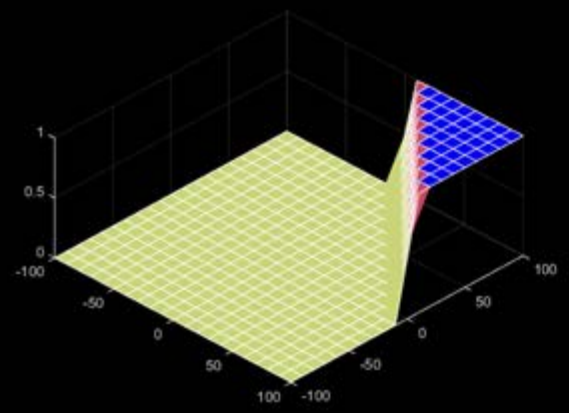


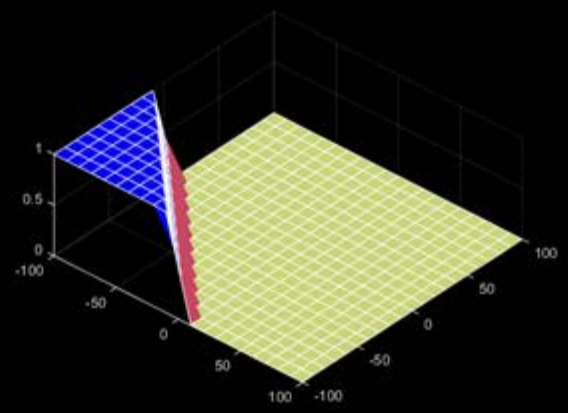


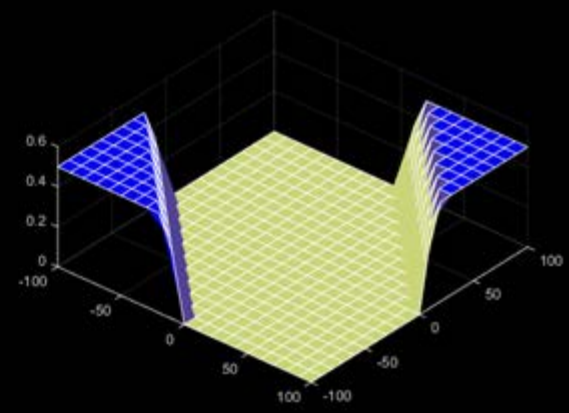
Effects of Multiple Perceptrons

- Consider the following artificial neural network case
  - Three perceptrons
    - Three different sets of parameters, w
      - $[w_1, w_2, b]$
      - Linear pre-activation function parameters and bias
    - Activation function as a sigmoid function
      - Logistic function
  - Finally, we can approximate the nonlinear decision boundary



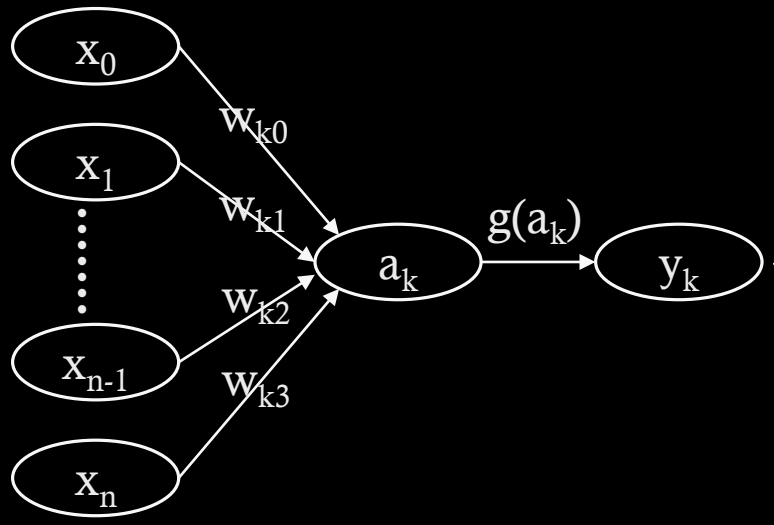


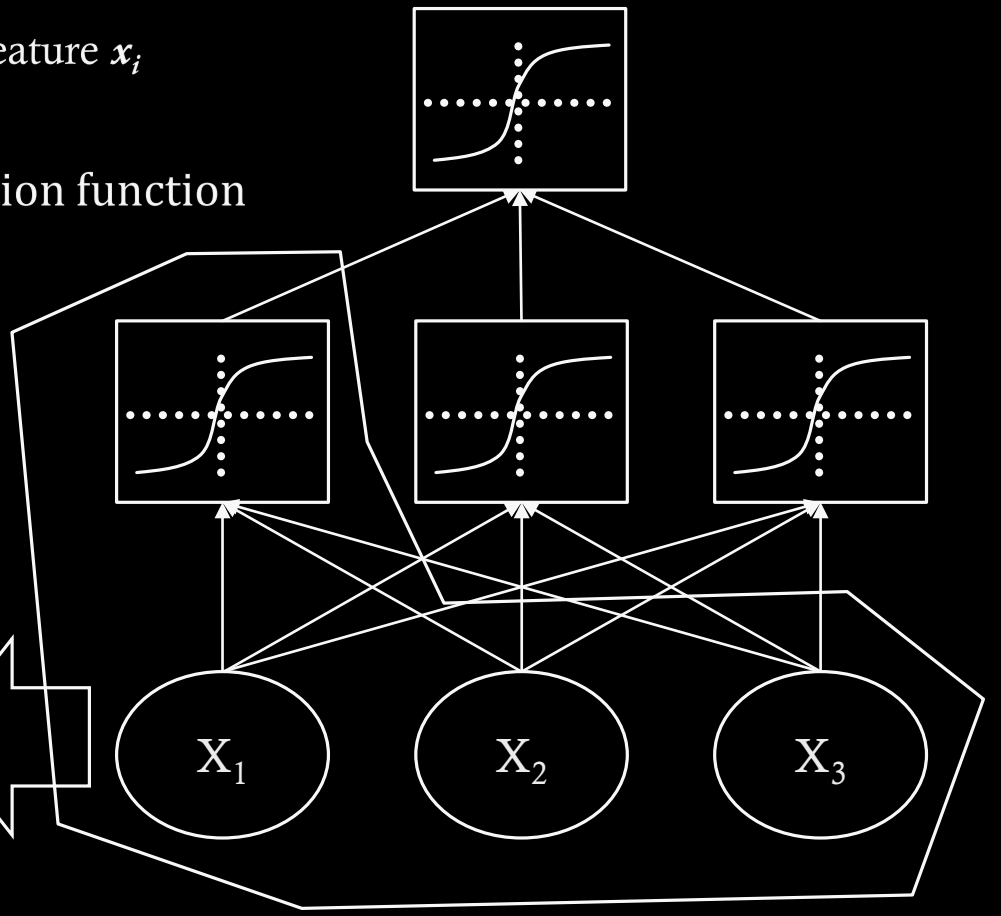




Linking Multiple Neurons

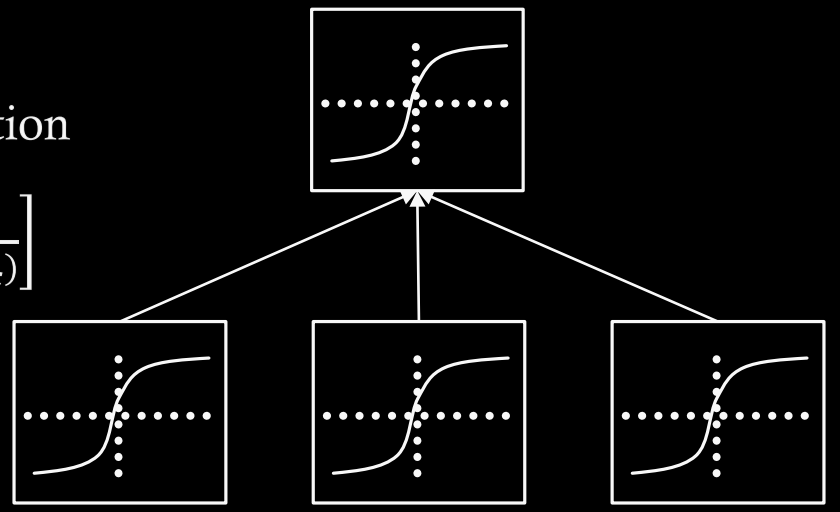
- Single hidden layer neural network
  - Hidden layer neuron input activation function
    - $\mathbf{a}(X) = \boldsymbol{b} + W_{i,j}^{1} X$
    - $W_{i,j}^{1}$: weight of the **first** layer from feature $\boldsymbol{x_i}$ to the hidden unit of $\boldsymbol{j}$
  - Hidden layer neuron output activation function
    - $\mathrm{h}(X) = \mathrm{g}(\mathrm{a}(X))$
  - Output layer activation function
    - $\mathrm{f}(X) = o(b + w\mathrm{h}(X))$





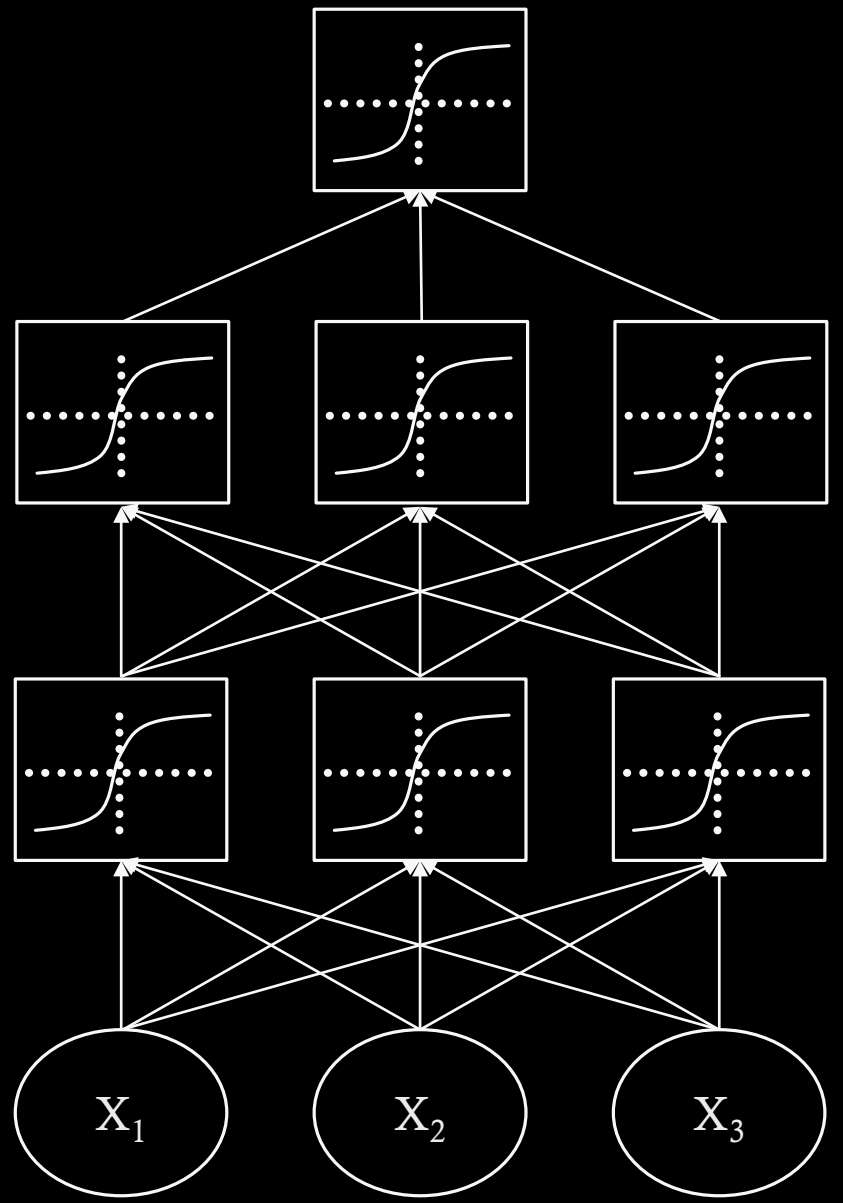
Output Activation Function

- Output layer activation function
  - $f(X) = o(b + wh(X))$
- Sigmoid function is good enough for the binary case
- What if more than the binary case?
  - We can use the softmax activation function
  - $o(a)=\text{softmax}(a)=\left[\frac{\exp(a_1)}{\sum_c \exp(a_c)}, \ldots, \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]$
  - C is the number of classes
  - Strictly positive
  - Sums to one
  - Because of the exponential term
    - Sharp adaptation to a certain decision case



Multiple Layers of Neural Network

- Considering L hidden layers
  - Neuron input activation function for the ***k-th*** layer
    - $a^k(x) = b^k + W^k h^{(k-1)}(x)$
  - Neuron output activation function for the ***k-th*** layer
    - $h^k(x) = g\left(a^k(x)\right)$
  - Output layer activation
    - $h^{L+1}(x) = o\left(a^{L+1}(x)\right) = f(x)$
- Each function is available by choices
  - Sigmoid functions
  - Softmax functions



Training neural network

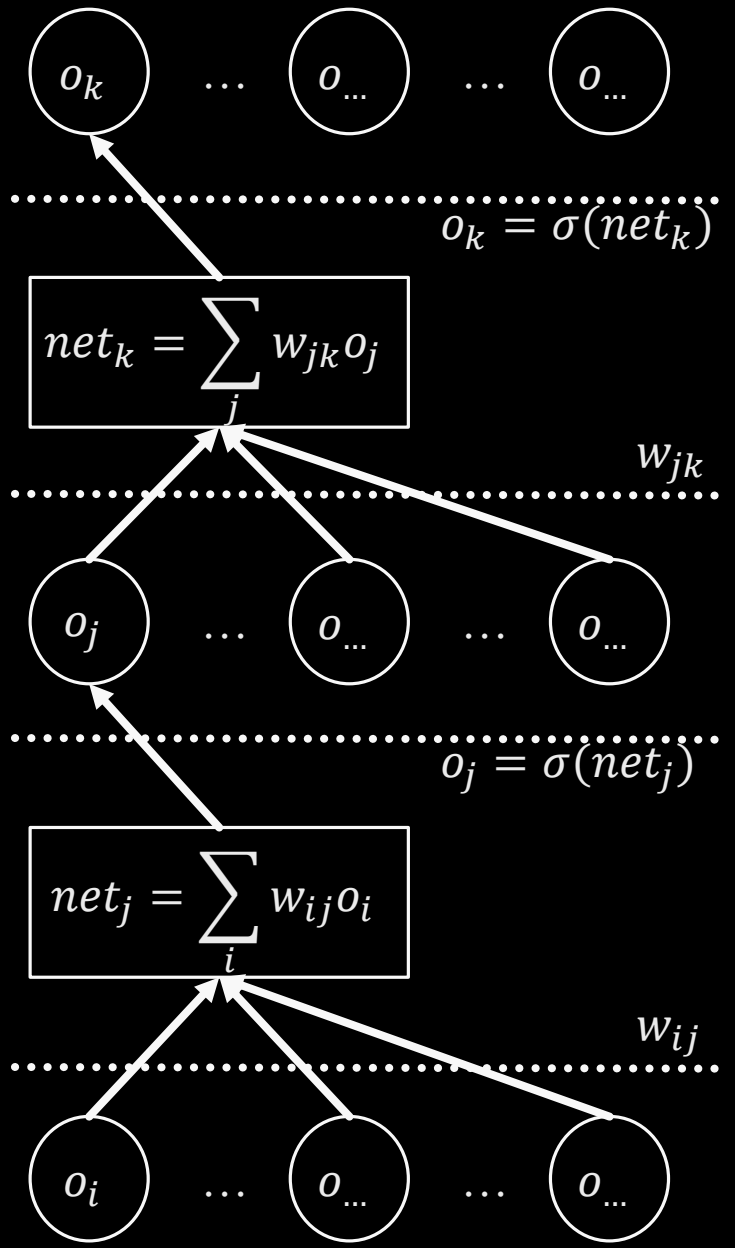
Derivative of Activation Function

- Sigmoid function is often used for the output function
  - Need the derivative for the learning stage
- Logistic function : $f(x) = \frac{1}{1+e^{-x}}$

  $\frac{d}{dx} f(x) = \frac{d}{dx}(1 + e^{-x})^{-1} = -1 \times (1 + e^{-x})^{-2} \times e^{-x} \times -1$
  $= e^{-x}(1 + e^{-x})^{-2} = \frac{1}{1+e^{-x}} \times \frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}} \times \frac{1-1+e^{-x}}{1+e^{-x}} = f(x)(1 - f(x))$
- The derivative of the logistic function can be expressed with the logistic function, again.
  - Simplicity in the mathematical notations
- When the logistic function is used as an activation function
  - The output can be plugged into the gradient function
- Similarly,
  - The derivative of $f(x) = \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \rightarrow \frac{d}{dx} f(x) = 1 - f^2(x)$
- ReLU
  - Use the sub-gradient method,
  - $f(x) - f(x_0) \geq c(x - x_0) \rightarrow c \in \left[\lim_{x \to x_0^-} \frac{f(x)-f(x_0)}{x-x_0}, \lim_{x \to x_0^+} \frac{f(x)-f(x_0)}{x-x_0}\right]$
  - Need to pick a candidate of c from the range

Sample Neural Network

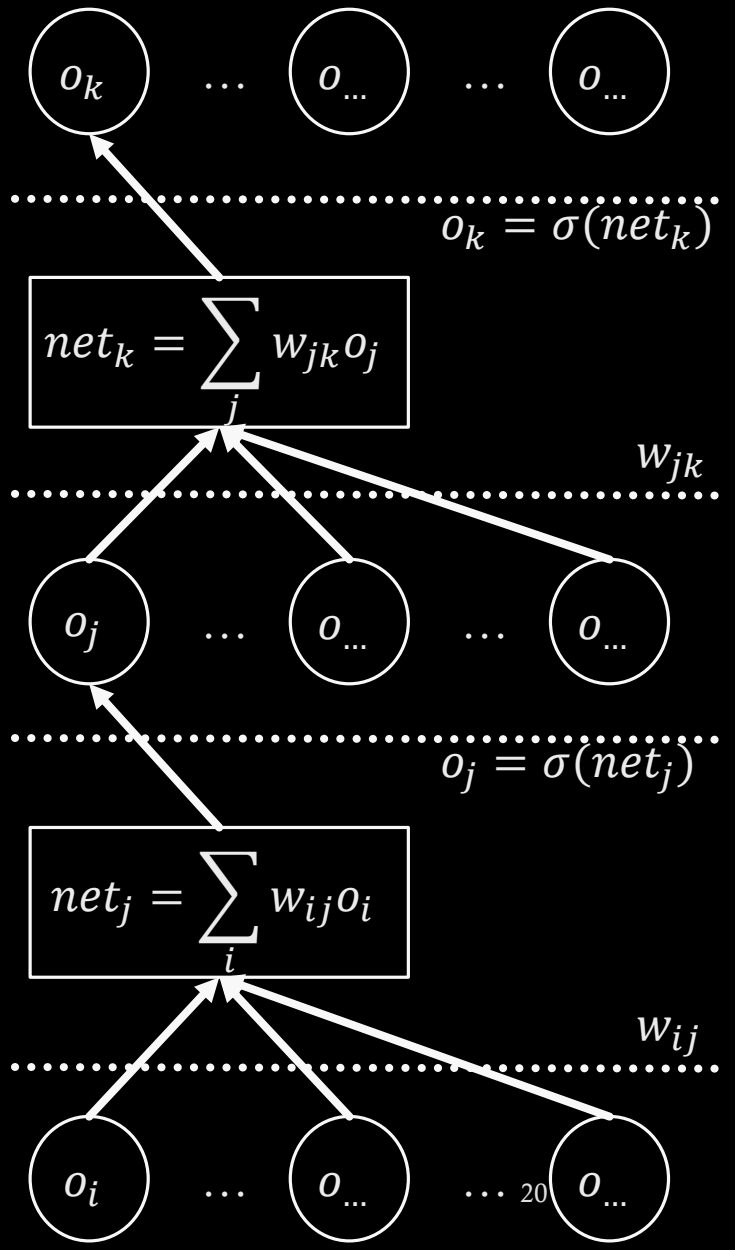
- Two hidden layers
  - net_k, net_j
- σ is an activation function
  - Here, we use the logistic function
  - $\frac{d}{dx}\sigma(net_j) = \sigma(net_j)\left(1 - \sigma(net_j)\right)$
    $= o_j(1 - o_j)$
- Definition of the loss function
  - $E = \frac{1}{2}(\sum_k (t_k - o_k)^2)$
- Parameter learnings
  - w_{jk}, w_{ij}
  - Updating the weights requires the gradient of $\frac{\partial E}{\partial w_{jk}}$, $\frac{\partial E}{\partial w_{ij}}$
- E and w_{jk}, w_{ij} are not directly related
  → Need to use the chain-rule of derivatives



$\frac{d}{dx}\sigma(net_j) = o_j(1 - o_j)$

Learning Single Depth

- $E = \frac{1}{2}(\sum_k (t_k - o_k)^2)$
- $\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}}$
  $= -(t_k - o_k)o_k(1 - o_k)o_j$
- $w_{jk}^{t+1} \leftarrow w_{jk}^t - \eta\frac{\partial E}{\partial w_{jk}}$
  $= w_{jk}^t - \eta\left(-(t_k - o_k)o_k(1 - o_k)o_j\right)$
  $= w_{jk}^t - \eta\left(-\delta_k o_j\right) = w_{jk}^t + \eta\delta_k o_j$
  - Gradient descent
  - η is the learning rate
  - Delta signal : $\delta_k = (t_k - o_k)o_k(1 - o_k)$
- So called, “delta rule”
  - $\Delta w_{jk} = \eta\delta_k o_j = \eta(t_k - y_k)\sigma'(h_k)x_j$



$$\frac{d}{dx}\sigma(net_j) = o_j(1 - o_j)$$

Learning Two Depth

- $E = \frac{1}{2}(\sum_k (t_k - o_k)^2)$
- $\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}} = -(t_k - o_k)o_k(1 - o_k)o_j$
  - Delta signal : $\delta_k = (t_k - o_k)o_k(1 - o_k)$
- $\frac{\partial E}{\partial w_{ij}} = \sum_k \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k}\frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$

$$= \sum_k -(t_k - o_k)o_k(1 - o_k)\frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

$$= \sum_k -\delta_k w_{jk} \times o_j(1 - o_j) \times o_i$$

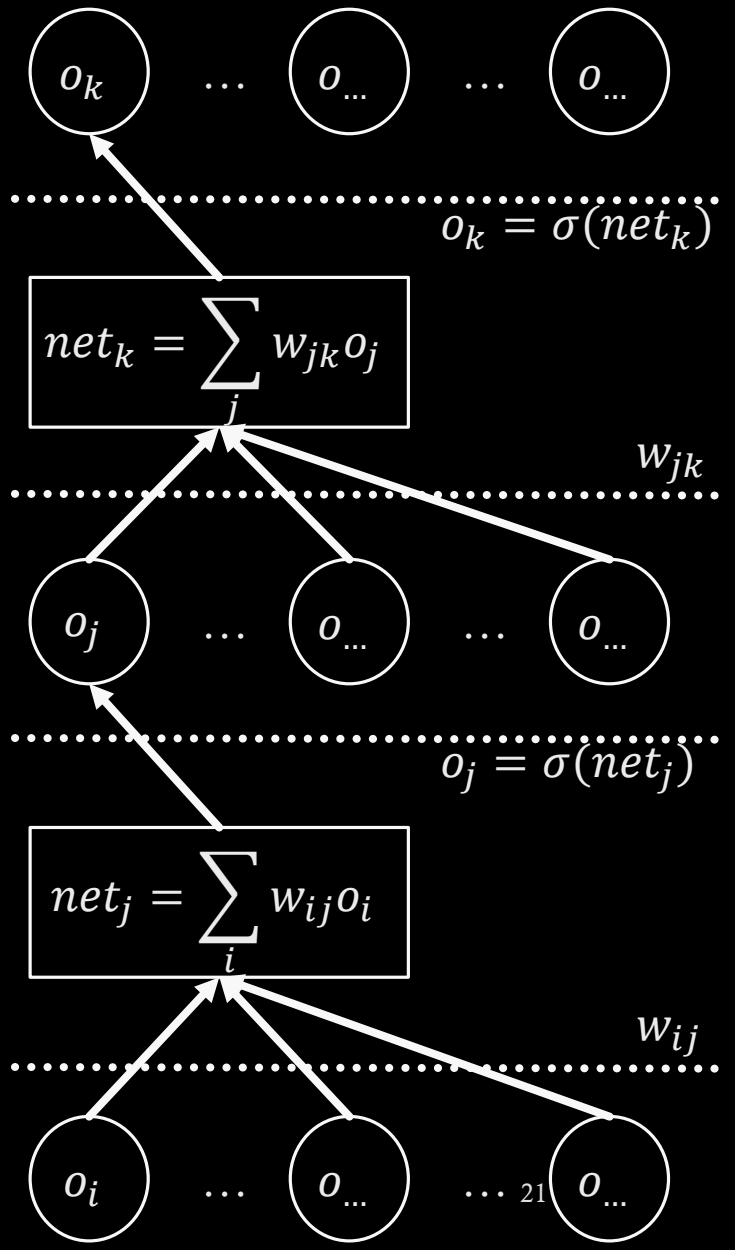
$$= -o_i o_j(1 - o_j)\sum_k \delta_k w_{jk}$$

  - Delta signal : $\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{jk}$
- $w_{ij}^{t+1} \leftarrow w_{ij}^t - \eta\frac{\partial E}{\partial w_{ij}}$

$$= w_{ij}^t - \eta(-(t_k - o_k)o_k(1 - o_k)o_j)$$

$$= w_{ij}^t - \eta(-o_i\delta_j) = w_{ij}^t + \eta o_i \delta_j$$

- Relation between δ_j and δ_k
  - Calculate δ_k → Calculate δ_j
  - Delta signal is coming back from the output : Backpropagation



Backpropagation

- Backpropagation
  - Do
    - For training examples, x

      // forward pass of the neural net.

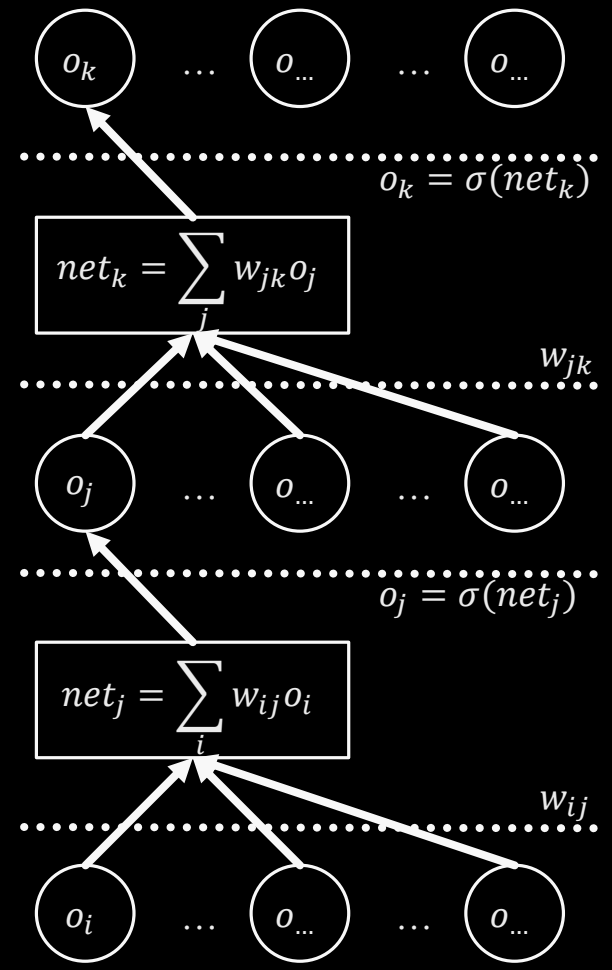
      - $o_k = f(x; w)$
      - $E = \frac{1}{2}(\sum_k (t_k - o_k)^2)$

      // backward pass of the neural net.

      - Calculate $\delta_k = (t_k - o_k) o_k (1 - o_k)$
      - For the backward-pass from the top to the bottom
        - Calculate $\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{jk}$

      // weight update

      - Update w_{jk} with $w_{jk}^{t+1} \leftarrow w_{jk}^t + \eta \delta_k o_j$
      - For the backward-pass from the top to the bottom
        - Update $w_{ij}^{t+1} \leftarrow w_{ij}^t + \eta o_i \delta_j$
  - Until converges



Stochastic Gradient Descent

- Algorithm that performs updates after each example
  - Initialize $\boldsymbol{w}$
  - For N iterations
    - For each training example $\left(\mathbf{x}^{(n)}, t^{(n)}\right)$
      - $\Delta = \nabla_{\boldsymbol{w}} E\left(f\left(\mathbf{x}^{(n)}; \boldsymbol{w}\right), t^{(n)}\right)$
      - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\Delta$
- Gradient descent vs. Stochastic gradient descent
  - Gradient descent
    - $w \leftarrow w - \eta\nabla_{\boldsymbol{w}} E(f(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{t}) = w - \eta\frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{w}} E\left(f\left(\mathbf{x}^{(n)}; \boldsymbol{w}\right), t^{(n)}\right)$
    - Compute the gradient with all instances
  - Stochastic gradient descent
    - $w \leftarrow w - \eta\nabla_{\boldsymbol{w}} E\left(f\left(\mathbf{x}^{(n)}; \boldsymbol{w}\right), t^{(n)}\right)$
    - Compute the gradient with each instance
    - Approximation of the true gradient with all instances
  - Mini-batch : using a part of instances

Weight Initialization

- For bias
  - Initialize all to 0
- For weights
  - Can't initialize weights to 0 with tanh activation
    - We can show that all gradients would then be 0 (saddle point)
  - Can't initialize weights to the same value
    - We can show that all hidden units in a layer will always behave the same
    - Need to break symmetry
  - Recipe: sample $W_{i,j}^{(k)}$ from $U[-b, b]$ where $b = \frac{\sqrt{6}}{\sqrt{H_k + H_k - 1}}$
    - H_k : size of $\mathbf{h}^{(k)}(\mathbf{x})$
    - The idea is to sample around 0 but break symmetry
    - Other values of b could work well (not an exact science) [1]

[1] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *International conference on artificial intelligence and statistics*. 2010.

Universal Approximation Theorem

- We are only going to introduce and understand its meaning
  - No proof

- Universal approximation theorem
  - Assumption
    - Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing function.
    - Let I_m be the m-dimensional unit hybercube, $[0,1]^m$
    - Let $C(I_m)$ be the space of continuous functions on I_m
  - Claim
    - Given any function $f \in C(I_m)$ and $\varepsilon > 0$,
    - There exists
      - Integer N
      - Real constants $v_i, b_i \in R$
      - Real vectors $w_i \in R^m, i = 1, \dots, N$
    - Such that
      - $F(x) = \sum_{i=1}^{N} v_i \varphi(w_i^T x + b_i)$ as an approximate realization of the function f
      - $|F(x) - f(x)| < \varepsilon$
    - For all $x \in I_m$

Problem of Artificial Neural Network

- Problem of gradient method
  - Vanilla version of gradient method
  - Random initialization?
    → Potentially very long learning process

    → Risk of local optima and saddle points
- Problem of back-propagation
  - What-if many layers?
    - Multiplication of many gradients → Getting close to zero
  - Computation time
    - A single logistic regression → Multiple logistic regression with interactions
    - Long training time
- Problem of structure
  - Is feasible and meaningful to fully connect the neurons?

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.