Deep Learning
CS60010

Abir Das
Assistant Professor
Computer Science and Engineering Department
Indian Institute of Technology Kharagpur

http://cse.iitkgp.ac.in/~adas/

Biological Neural Network

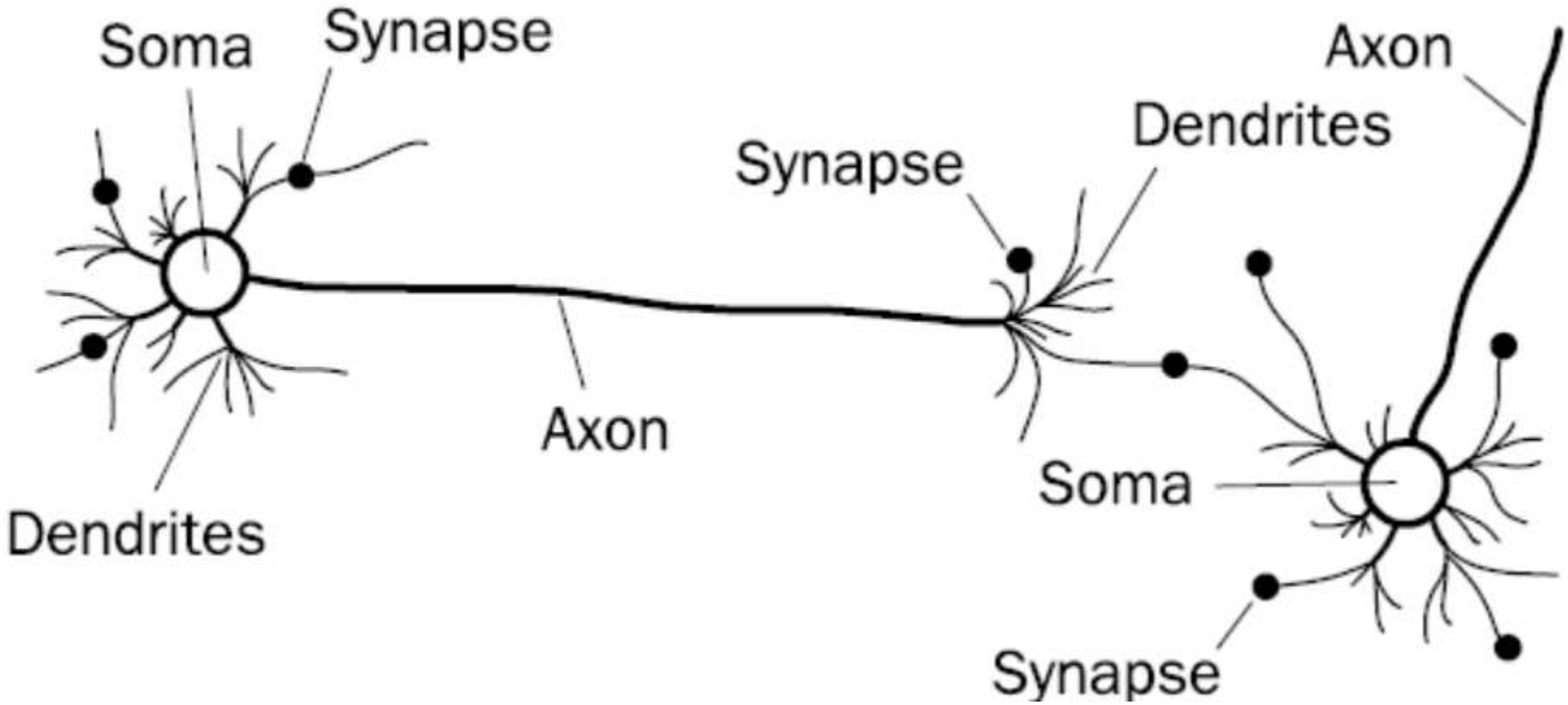
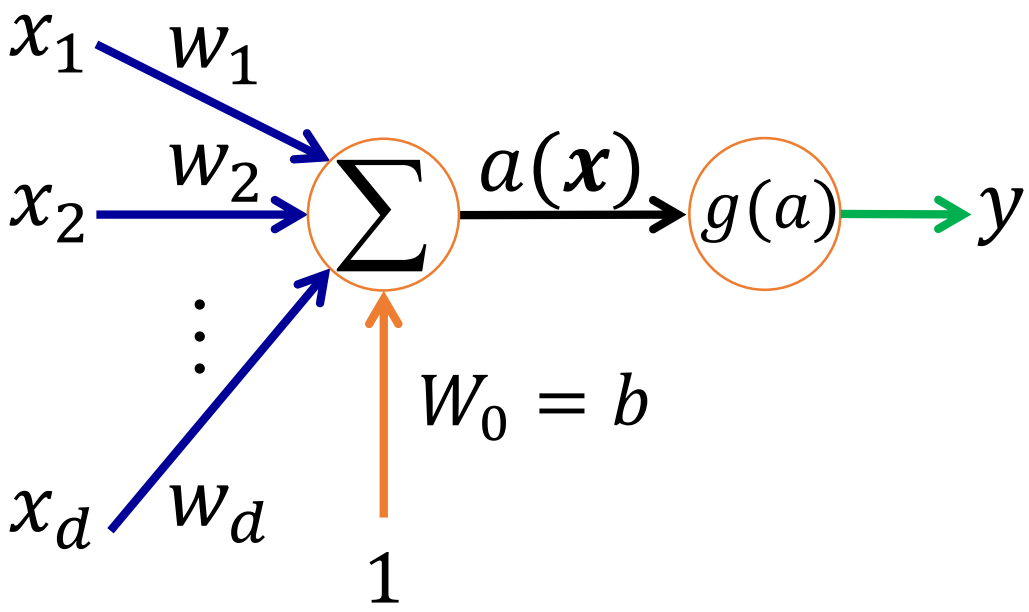


Image courtesy: F. A. Makinde et. al., “Prediction of crude oil viscosity using feed-forward back-propagation neural network (FFBPNN)”." Petroleum & Coal 2012


deep CNN features (~2013) vs. IT neural "features"
Zeiler & Fergus features
features"
xNet features
Dear Computer Vision Researchers,
Thank you !
We are working toward the same goal....
See great related work by:
Kriegeskorte,
Oliva,
Gallant & Malik,
Gardner
...
Controls
Ability to predict IT neurons
(% variance explained)
50
0
HMO features
AlexNet features
Zeiler & Fergus features
N, Majaj N, and DiCarlo JJ. ICLR (2013);
N, Majaj N, and DiCarlo JJ. PLoS Comp Bio (2014)
CVPR
July 21-26 2017
HONOLULU
CVF

Artificial Neuron



$$\boldsymbol{w} = [w_1\ w_2\ \dots w_d]^T \quad \text{and} \quad \boldsymbol{x} = [x_1\ x_2\ \dots x_d]^T$$

$$a(\boldsymbol{x}) = b + \sum_{i=1}^{d} w_i x_i = [\boldsymbol{w}^T b] \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

$$y = g(a(\boldsymbol{x}))$$

Terminologies:-

$\boldsymbol{x}$: input, $\boldsymbol{w}$: weights, $\boldsymbol{b}$: bias

a: pre-activation (input activation)

g: activation function

y: activation (output activation)

Perceptron

The New York Times

Electronic 'Brain' Teaches Itself

JULY 13, 1958

The Navy last week demonstrated the embryo of an electronic computer named the Perceptron which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control." Navy officers demonstrating a preliminary form of the device in Washington said they hesitated to call it a machine because it is so much like a "human being without life."

Dr. Frank Rosenblatt, research psychologist at the Cornell Aeronautical Laboratory, Inc., Buffalo, N. Y., designer of the Perceptron, conducted the demonstration. The machine, he said, would be the first electronic device to think as the recognize the difference between right and left, almost the way a child learns.

When fully developed, the Perceptron will be designed to remember images and information it has perceived itself, whereas ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons, Dr. Rosenblatt said, will be able to recognize people and call out their names. Printed pages, longhand letters and even speech commands are within its reach. Only one more step of development, a difficult step, he said, is needed for the device to hear speech in one language and instantly translate it to speech or writing in another language.

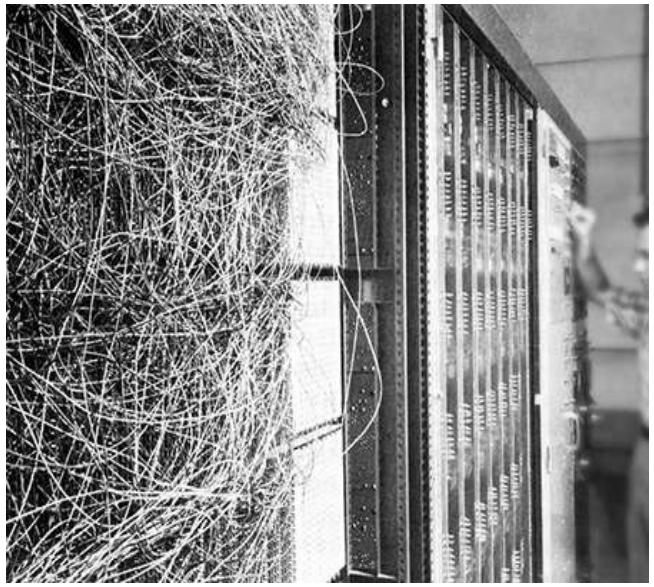
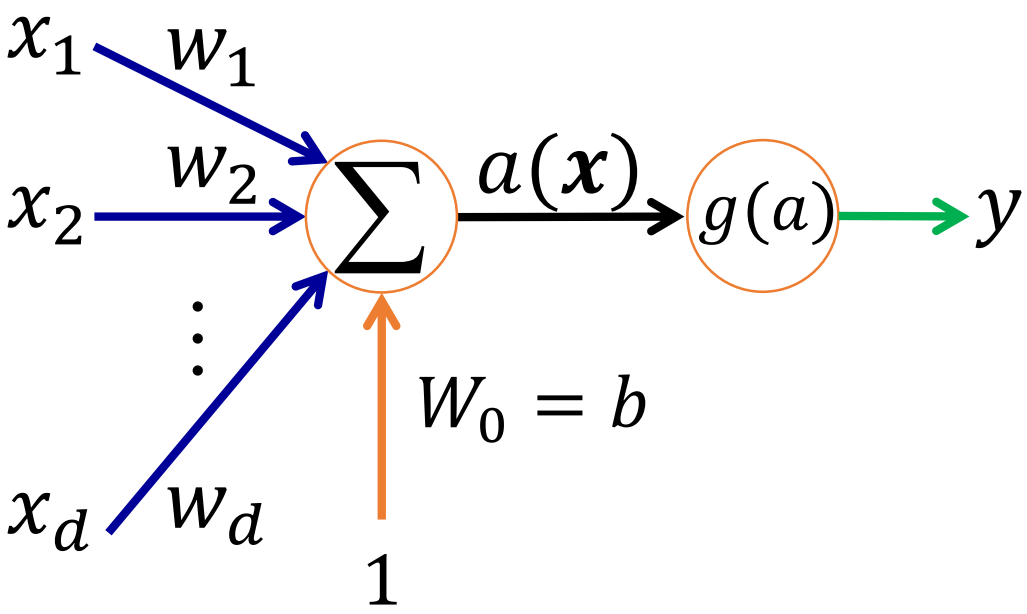


Image courtesy: https://blogs.umass.edu/comphon/2017/06/15/did-frank-rosenblatt-invent-deep-learning-in-1962/

Perceptron

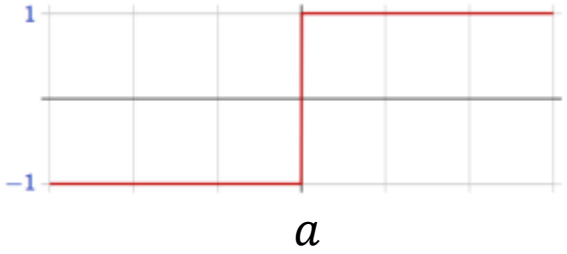


$\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \{0, 1\}$ – Binary Classification

$$g(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases}$$ (Rosenblatt, 1957)

To make things simpler, the response is taken as $y \in \{-1, 1\}$

$$g(a) = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$



The perceptron classification rule, thus, translates to

$$y = \begin{cases} 1, & \boldsymbol{w}^T\boldsymbol{x} + b \geq 0 \\ -1, & \boldsymbol{w}^T\boldsymbol{x} + b < 0 \end{cases}$$

Remember: $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ represents a hyperplane.

Perceptron Learning Algorithm

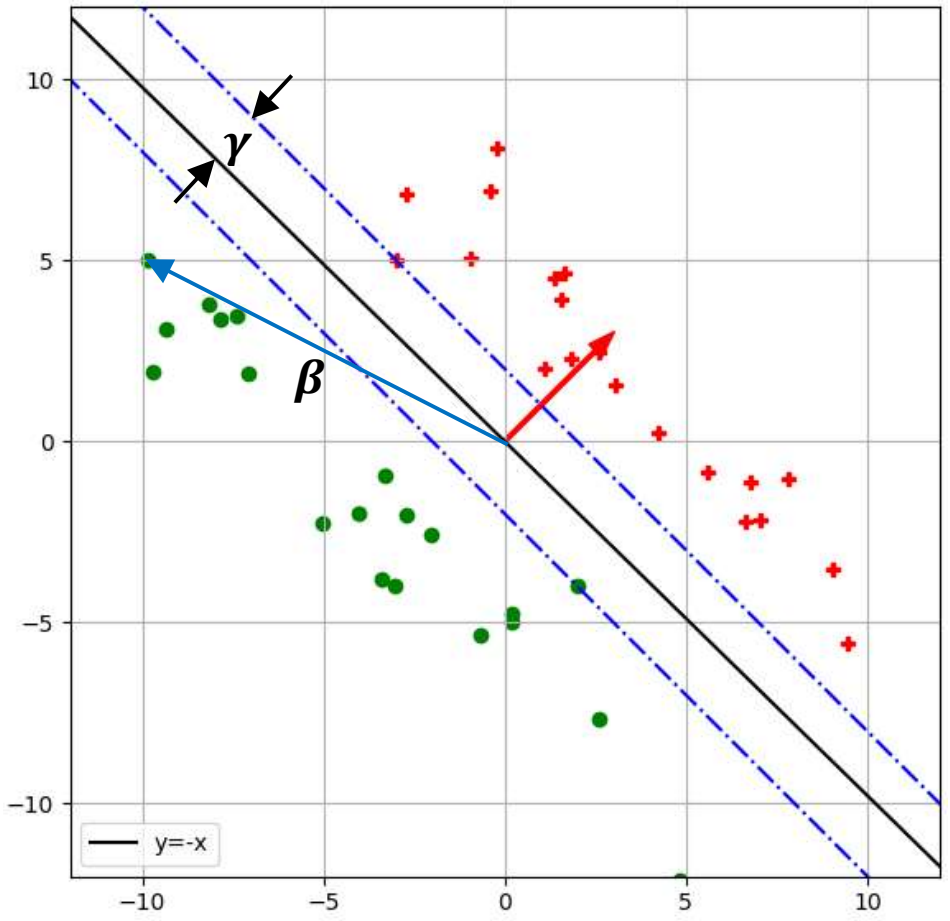
Given a training set: $\{\boldsymbol{x}^{(i)}, y^{(i)} : \boldsymbol{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \{-1, 1\}\}$ for $i = 1, 2, \ldots, N$

1. Start with $k = 1$ and $\boldsymbol{w}^{(k)} = [0\ 0\ \ldots 0]^T$
2. Loop until all examples are correctly classified:

   $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} + y^{(k)}\boldsymbol{x}^{(k)}$ if $\boldsymbol{x}^{(k)}$ is misclassified.

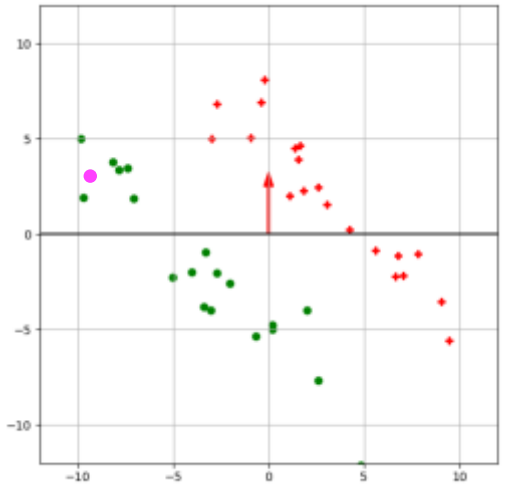
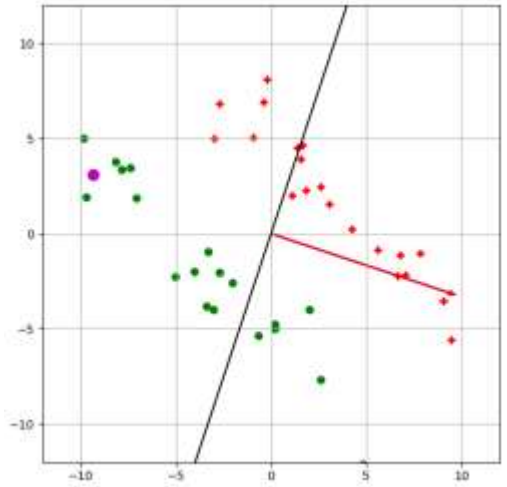
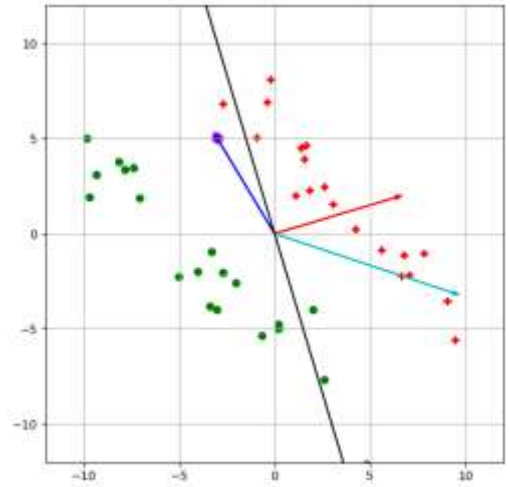
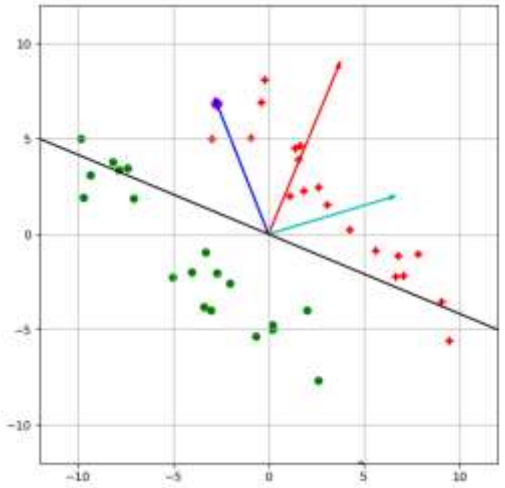
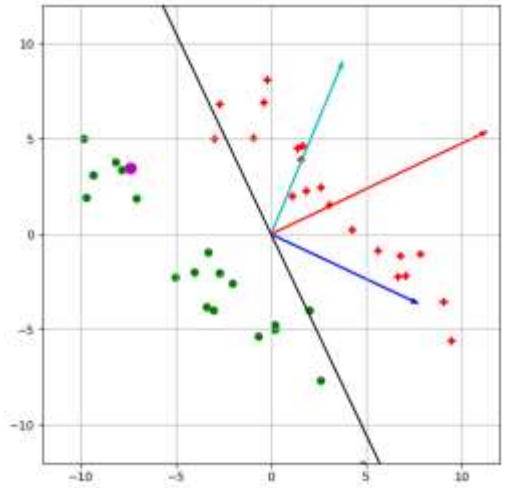
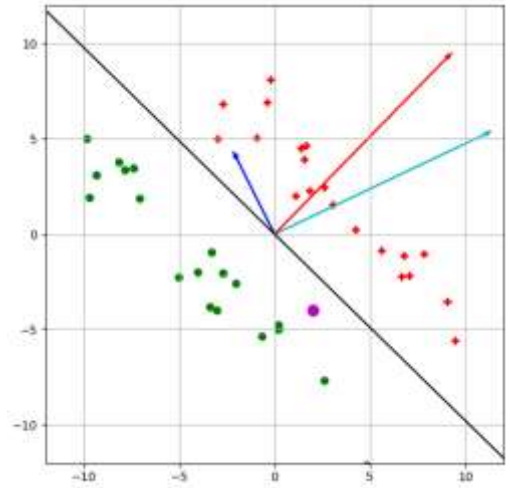
   [This means $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} + \boldsymbol{x}^{(k)}$ or $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \boldsymbol{x}^{(k)}$]

   $k \leftarrow k + 1$ or reset $k = 1$ if $k == N$



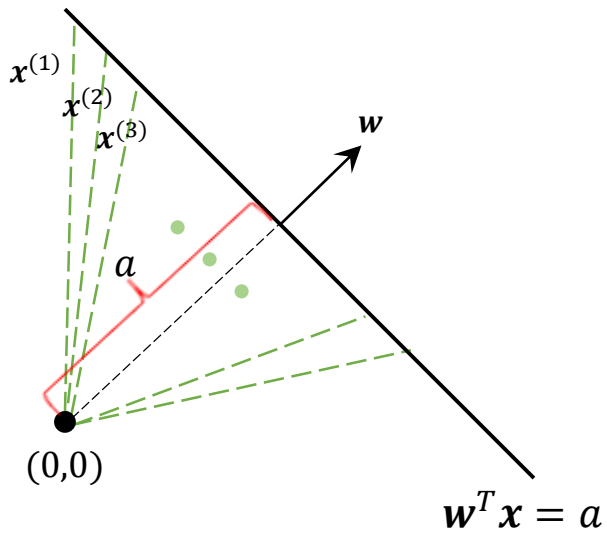
Convergence Theorem:-

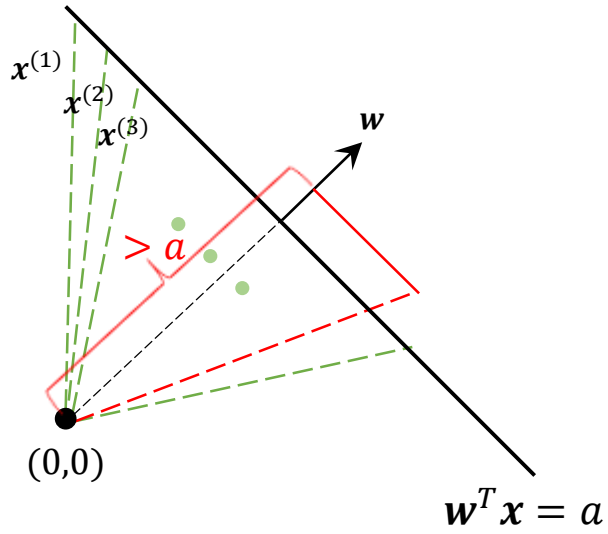
For a finite and linearly separable set of data, the perceptron learning algorithm will find a linear separator in at most $\frac{\beta^2}{\gamma^2}$ iterations where the maximum length of any data point is β and γ is the maximum margin of the linear separators.

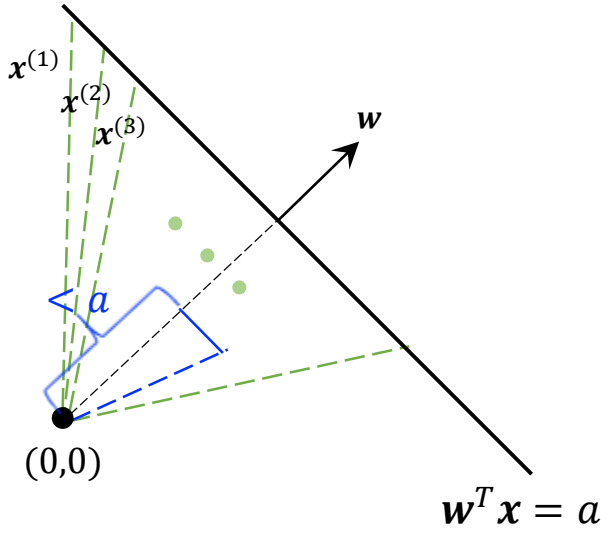







Perceptron Learning Algorithm – Convergence Proof

Some Important Brush-ups:-







For a hyperplane $\boldsymbol{w}^T \boldsymbol{x} = a$, $\boldsymbol{w}$ is the normal vector (often assumed to be unit normal vector) to the hyperplane and a is the distance from the origin.

Thus a hyperplane with normal vector $\mathbf{w}$, can separate two sets of points by examining if $\boldsymbol{w}^T \boldsymbol{x} - a$ is > 0 or < 0. (Remember our notation says –a is nothing but b).

Perceptron Learning Algorithm – Convergence Proof

- Thus a simple classification rule can be $\boldsymbol{w}^T\boldsymbol{x} + b > 0$ or < 0.

- For simplicity, let us assume that the hyperplane passes through origin – This does not affect the problem in our hand as it just shifts the origin. But that means $b = 0$. Thus the classification rule becomes $sgn\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}\right) = y^{(i)} \forall (\boldsymbol{x}^{(i)}, y^{(i)})$.

- There may not exist such hyperplane that can separate two sets of data. If it exists then the data is known to be linearly separable.

- A more formal definition is – Two sets P and N of d dimensional points are said to be linearly separable if there exists $d + 1$ real numbers $\boldsymbol{w} = (w_0, w_1, \cdots, w_{d+1})$ such that every point $\boldsymbol{x} = (1, x_1, x_2, \cdots, x_{d+1}) \in P$, satisfies $sgn(\boldsymbol{w}^T\boldsymbol{x}) = 1$ and every point $\boldsymbol{x} = (1, x_1, x_2, \cdots, x_{d+1}) \in N$, satisfies $sgn(\boldsymbol{w}^T\boldsymbol{x}) = -1$,

Perceptron Learning Algorithm – Convergence Proof

Without loss of generalization we will also assume the following while proving the convergence theorem.

- We normalize all the points $\boldsymbol{x}^{(i)}$ so that $max\ ||\boldsymbol{x}^{(i)}|| = 1$, *i.e.,* maximum length of any data point is 1. Notice that this does not affect the algorithm or the solution as, $sgn\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right) = sgn\left(\boldsymbol{w}^T \frac{\boldsymbol{x}^{(i)}}{||\boldsymbol{x}^{(i)}||}\right)$

- Margin of separation: This is the minimum distance between any data point and the hyperplane $\boldsymbol{w}^T \boldsymbol{x} = 0$. Such a hyperplane is defined to be the *maximum margin separator*.

$$\gamma = \min_{\boldsymbol{x}} |\boldsymbol{w}^T \boldsymbol{x}|$$

Perceptron Learning Algorithm – Convergence Proof

- Let $\boldsymbol{w}^*$ be the normal to the maximum margin linear separator. We want our $\boldsymbol{w}$ in the perceptron algorithm in each step getting closer and closer to this (sort of) ideal $\boldsymbol{w}^*$.
- This means that the inner product between $\boldsymbol{w}$ and $\boldsymbol{w}^*$ should increase with each update.
- Remember the update rule: $\boldsymbol{w}' = \boldsymbol{w} + \boldsymbol{x}^{(i)} y^{(i)}$
- So, $(\boldsymbol{w}')^T \boldsymbol{w}^* = \left(\boldsymbol{w} + \boldsymbol{x}^{(i)}\right)^T \boldsymbol{w}^*$ (assuming $y^{(i)} = +1$)

$$= \boldsymbol{w}^T \boldsymbol{w}^* + \left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{w}^*$$

Previous $\boldsymbol{w}^T \boldsymbol{w}^*$

This is at least γ

- Similarly for a negative training example,

$$(\boldsymbol{w}')^T \boldsymbol{w}^* = \left(\boldsymbol{w} - \boldsymbol{x}^{(i)}\right)^T \boldsymbol{w}^* = \boldsymbol{w}^T \boldsymbol{w}^* - \left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{w}^*$$

$\left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{w}^*$ is negative as the example is negative and thus $-\left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{w}^* \geq \gamma$.

- So, in both cases $(\boldsymbol{w}')^T \boldsymbol{w}^*$ is increasing by at least γ. Which means length of $\boldsymbol{w}'$ increases with each update → Observation 1

Perceptron Learning Algorithm – Convergence Proof

- length of $\boldsymbol{w}'$ is given by $||\boldsymbol{w}'||^2$. Let us see, how this changes with each update.
- $||\boldsymbol{w}'||^2 = (\boldsymbol{w}')^T\boldsymbol{w}' = \left(\boldsymbol{w} + \boldsymbol{x}^{(i)}y^{(i)}\right)^T\left(\boldsymbol{w} + \boldsymbol{x}^{(i)}y^{(i)}\right) = \left||\boldsymbol{w}|\right|^2 + 2y^{(i)}\left(\boldsymbol{x}^{(i)}\right)^T\boldsymbol{w} + \left(y^{(i)}\right)^2||\boldsymbol{x}^{(i)}||^2$
- Let us consider the case $y^{(i)} = +1$ first.
- $||\boldsymbol{w}'||^2 = \left||\boldsymbol{w}|\right|^2 + 2\left(\boldsymbol{x}^{(i)}\right)^T\boldsymbol{w} + ||\boldsymbol{x}^{(i)}||^2$

This is negative as update occurs only on misclassification and for $y^{(i)} = +1$, misclassification means $\left(\boldsymbol{x}^{(i)}\right)^T\boldsymbol{w} \leq 0$.

This is ≤ 1, as we assumed every point is rescaled to unit ball.

- So, $||\boldsymbol{w}'||^2 \leq \left||\boldsymbol{w}|\right|^2 + 1$

Perceptron Learning Algorithm – Convergence Proof

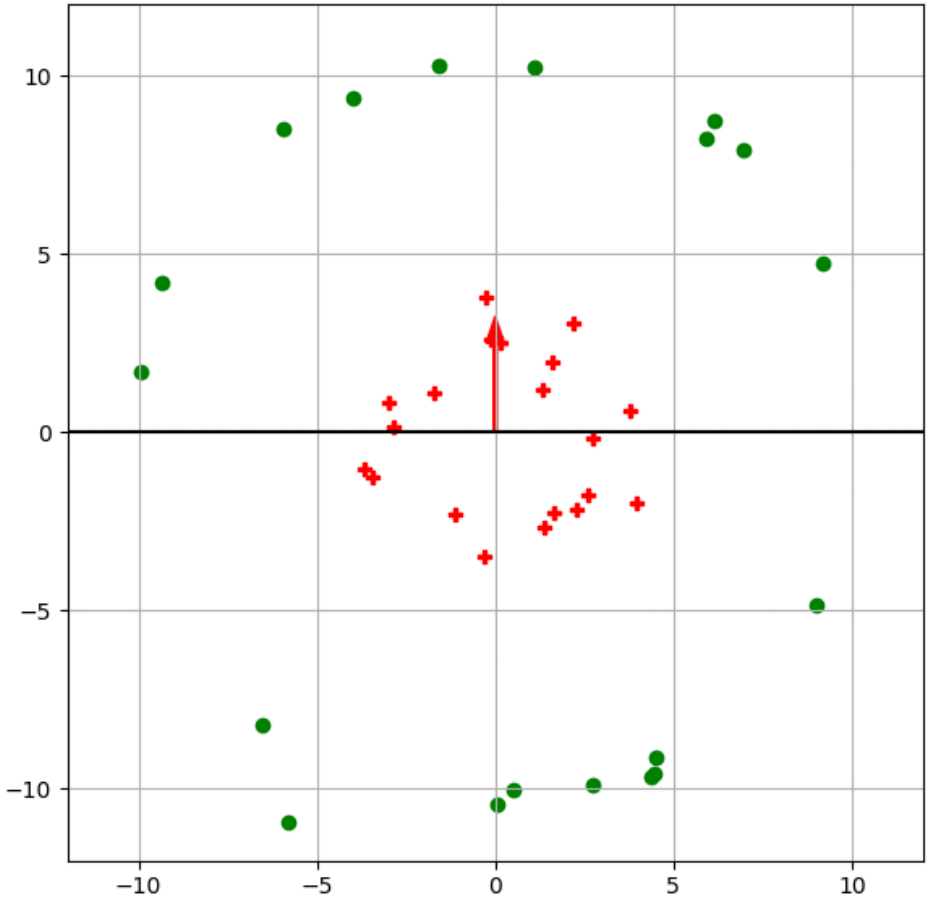
- Let us now consider the case $y^{(i)} = -1$.
- $||\boldsymbol{w}'||^2 = \left|\left|\boldsymbol{w}\right|\right|^2 - 2\left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{w} + ||\boldsymbol{x}^{(i)}||^2$
- Using similar arguments, $\left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{w} \geq 0$, so, $-2\left(\boldsymbol{x}^{(i)}\right)^T \boldsymbol{w} \leq 0$ and also $||\boldsymbol{x}^{(i)}||^2 \leq 1$
- So, here also $||\boldsymbol{w}'||^2 \leq \left|\left|\boldsymbol{w}\right|\right|^2 + 1$
- So, the length of the updated vector is increasing but it is increasing in a controlled way. → Observation 2

Perceptron Learning Algorithm – Convergence Proof

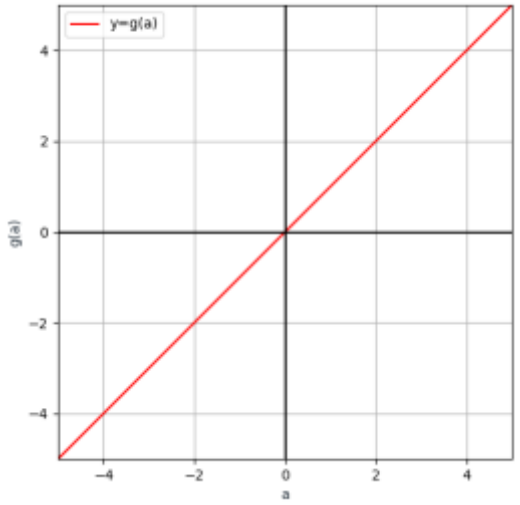
- Say after k iterations, $\boldsymbol{w}$ is denoted as $\boldsymbol{w}_k$.

- So observation 1 implies, $\boldsymbol{w}_k^T \boldsymbol{w}^* \geq k\gamma$ and observation 2 implies $\left|\left|\boldsymbol{w}_k\right|\right|^2 \leq k$ [remember the initial value of $\boldsymbol{w}$ is $\boldsymbol{0}$]

- Now we have to apply the Cauchy-Swartz inequality.
- $\left|\left|\boldsymbol{w}_k\right|\right|\left|\left|\boldsymbol{w}^*\right|\right| \geq \boldsymbol{w}_k^T \boldsymbol{w}^*$, but $\left|\left|\boldsymbol{w}^*\right|\right| = 1$, so,
- $\left|\left|\boldsymbol{w}_k\right|\right| \geq \boldsymbol{w}_k^T \boldsymbol{w}^* \geq k\gamma$ [observation 1]
- This implies $k^2\gamma^2 \leq \left|\left|\boldsymbol{w}_k\right|\right|^2 \leq k$ [observation 2]
- So, $k \leq \frac{1}{\gamma^2}$

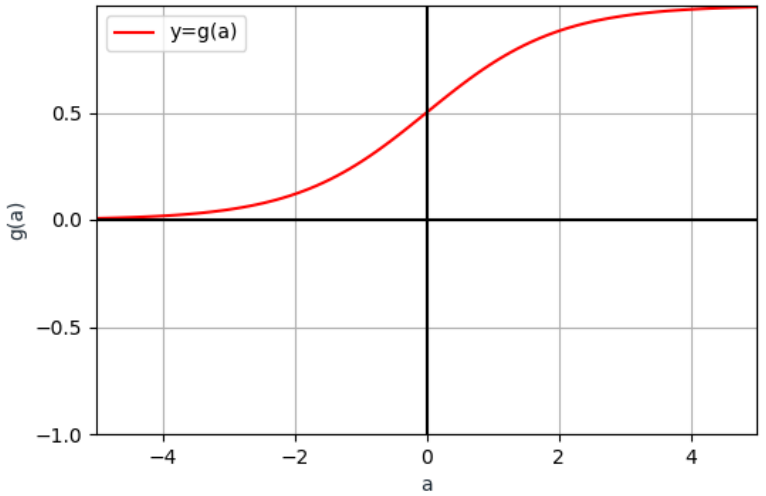

10
5
0
−5
−10
−10
−5
0
5
10

Popular Choices of Activation Functions



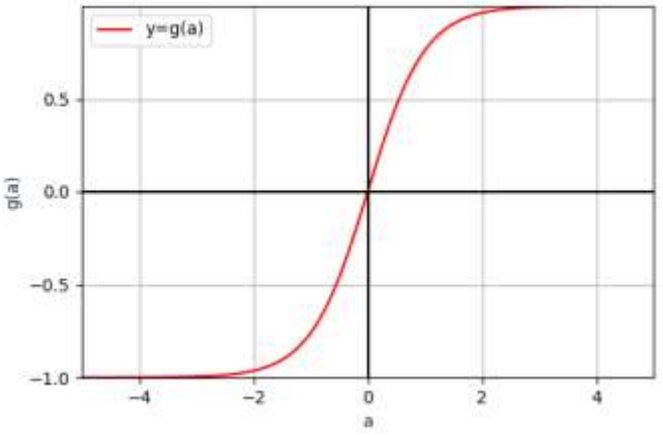
Linear activation function

- $g(a) = a$
- Unbounded
- $g'(a) = 1$



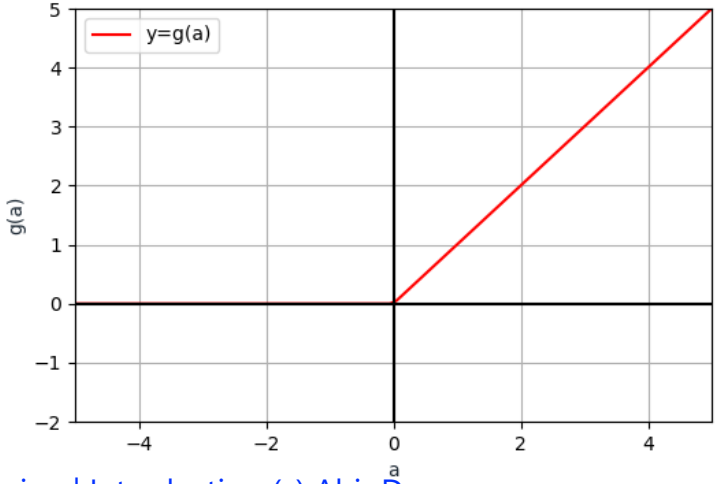
Sigmoid activation function

- $g(a) = \sigma(a) = \frac{1}{1+\exp(-a)}$
- Bounded (0, 1)
- Always positive
- $g'(a) = g(a)(1 - g(a))$



tanh activation function

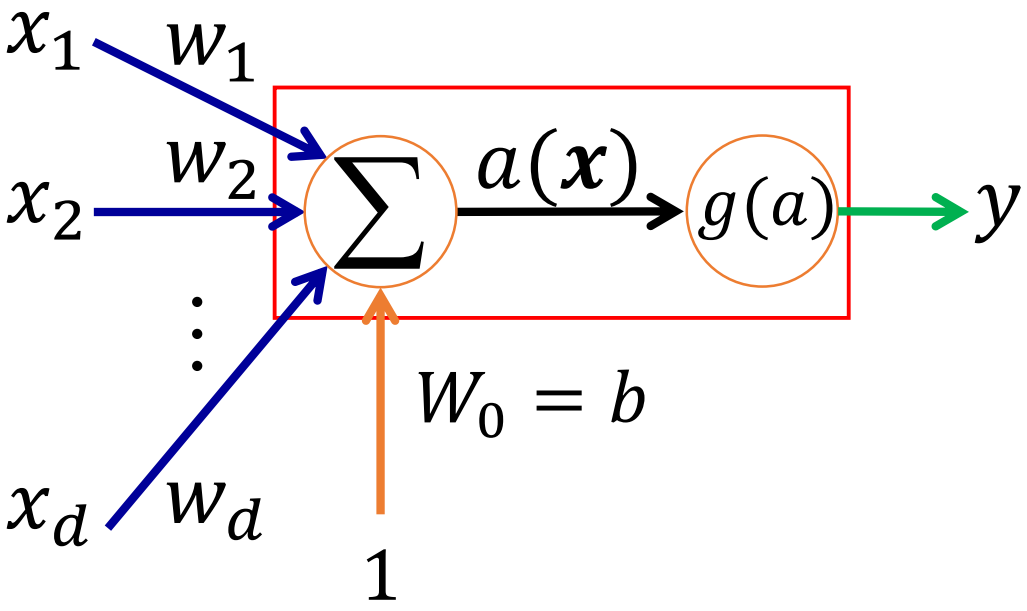
- $g(a) = tanh(a)$
  $= \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$
- Bounded (-1, 1)
- Can be positive or negative
- $g'(a) = 1 - g^2(a)$

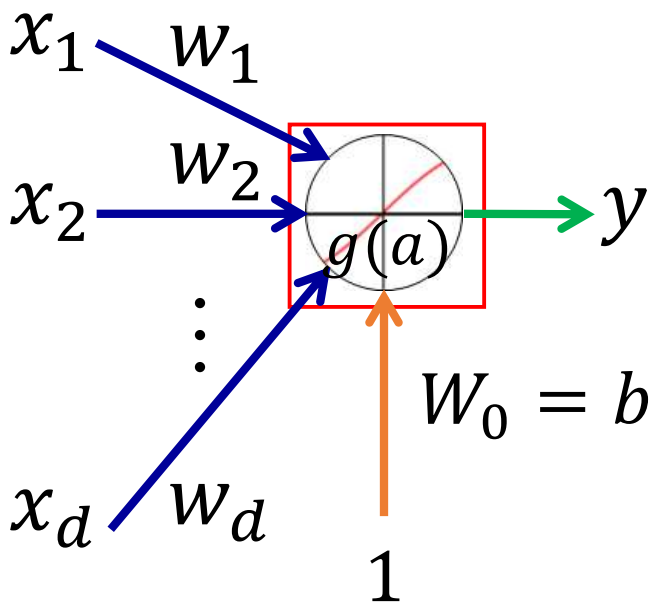


ReLU activation function

- $g(a) = \max(0, a)$
- Bounded below by 0
- But not upper-bounded
- $g'(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases}$

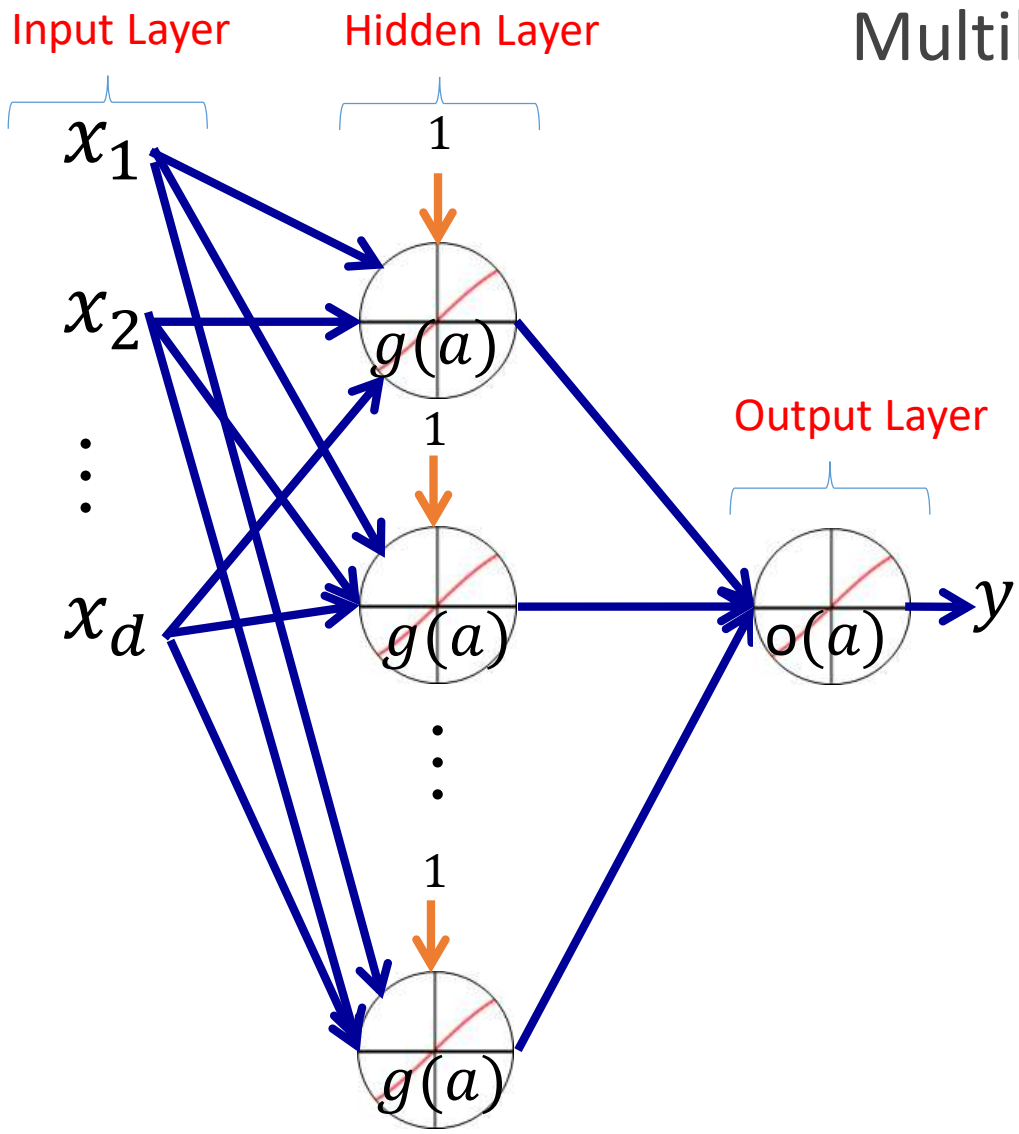
Multilayer Neural Network





A bit of notation change

Multilayer Neural Network

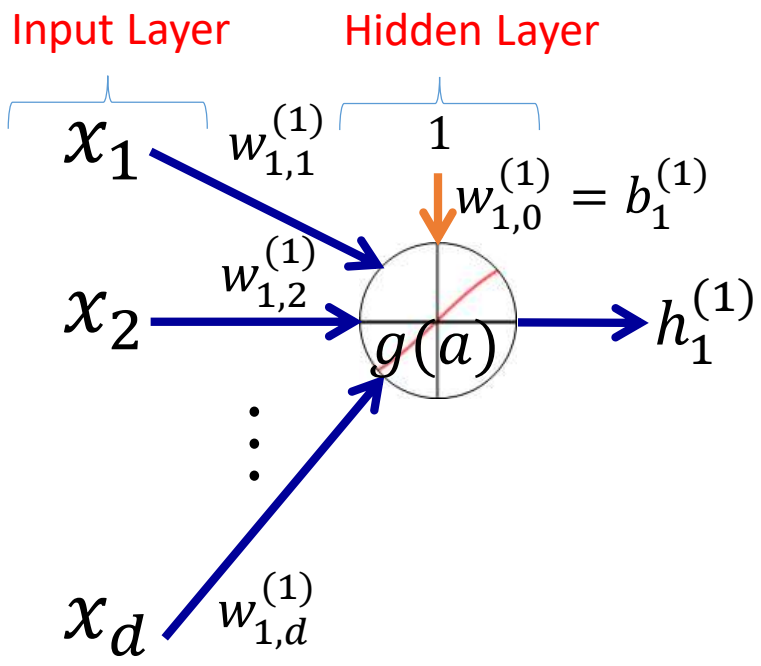


Intuition:-

Hidden Layer: Extracts better representation of the input data

Output layer: Does the classification

Multilayer Neural Network



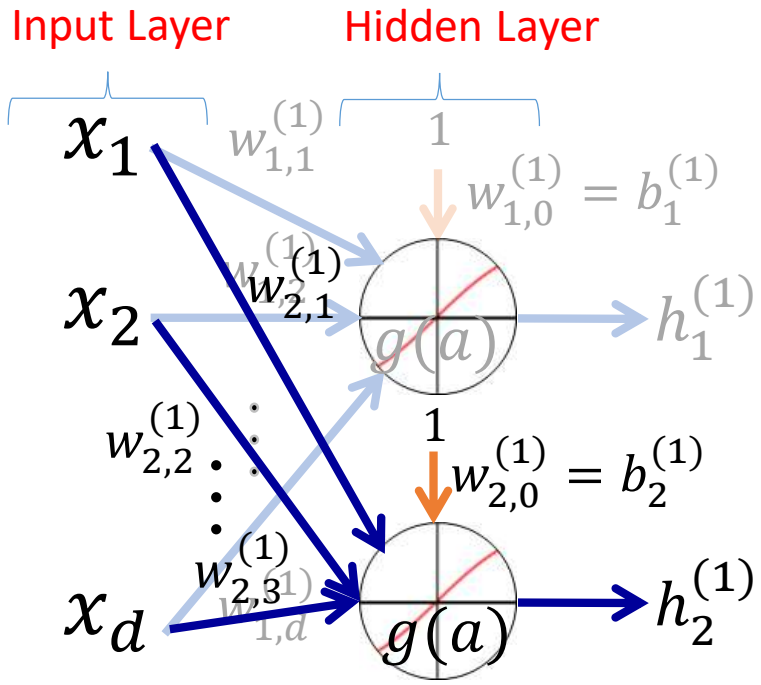
$$a_1^{(1)}(\boldsymbol{x}) = b_1^{(1)} + \sum_{i=1}^{d} w_{1,i}^{(1)} x_i = \left[\boldsymbol{w}_1^{(1)} b_1^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}$$

Row vector $\left[w_{1,1}^{(1)}, w_{1,2}^{(1)}, \cdots, w_{1,d}^{(1)}\right]$

Column vector $[x_1\ x_2\ \ldots x_d]^T$

$$h_1^{(1)} = g\left(a_1^{(1)}(\boldsymbol{x})\right)$$

Multilayer Neural Network



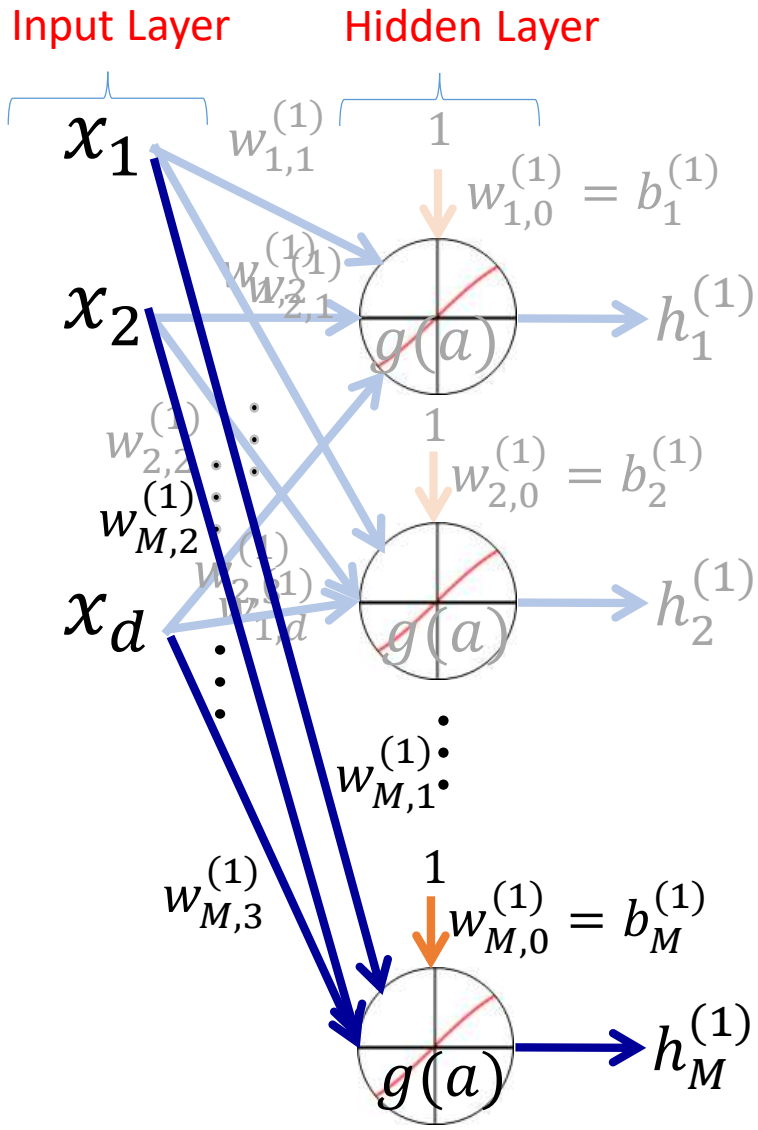
$$a_1^{(1)}(\boldsymbol{x}) = b_1^{(1)} + \sum_{i=1}^{d} w_{1,i}^{(1)}\, x_i = \left[\boldsymbol{w}_1^{(1)} b_1^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}$$

$$h_1^{(1)} = g\left(a_1^{(1)}(\boldsymbol{x})\right)$$

$$a_2^{(1)}(\boldsymbol{x}) = b_2^{(1)} + \sum_{i=1}^{d} w_{2,i}^{(1)}\, x_i = \left[\boldsymbol{w}_2^{(1)} b_2^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}$$

$$h_2^{(1)} = g\left(a_2^{(1)}(\boldsymbol{x})\right)$$

Multilayer Neural Network



$$a_1^{(1)}(\boldsymbol{x}) = b_1^{(1)} + \sum_{i=1}^{d} w_{1,i}^{(1)}\, x_i = \left[\boldsymbol{w}_1^{(1)} b_1^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}$$

$$h_1^{(1)} = g\left(a_1^{(1)}(\boldsymbol{x})\right)$$

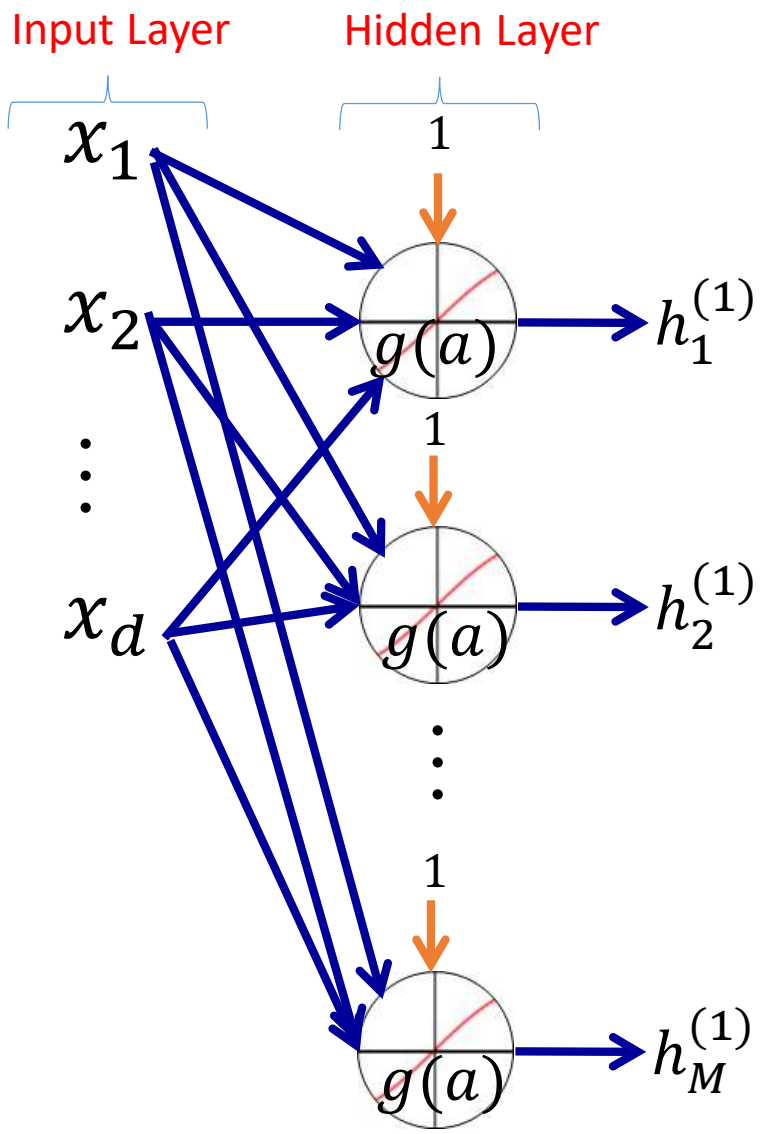
$$a_2^{(1)}(\boldsymbol{x}) = b_2^{(1)} + \sum_{i=1}^{d} w_{2,i}^{(1)}\, x_i = \left[\boldsymbol{w}_2^{(1)} b_2^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}$$

$$h_2^{(1)} = g\left(a_2^{(1)}(\boldsymbol{x})\right)$$

$$a_M^{(1)}(\boldsymbol{x}) = b_M^{(1)} + \sum_{i=1}^{d} w_{M,i}^{(1)}\, x_i = \left[\boldsymbol{w}_M^{(1)} b_2^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}$$

$$h_M^{(1)} = g\left(a_M^{(1)}(\boldsymbol{x})\right)$$

Multilayer Neural Network



$$a_1^{(1)}(\boldsymbol{x}) = \left[\boldsymbol{w}_1^{(1)} b_1^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}$$
$$a_2^{(1)}(\boldsymbol{x}) = \left[\boldsymbol{w}_2^{(1)} b_2^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}$$
$$\vdots$$
$$a_M^{(1)}(\boldsymbol{x}) = \left[\boldsymbol{w}_M^{(1)} b_2^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}$$

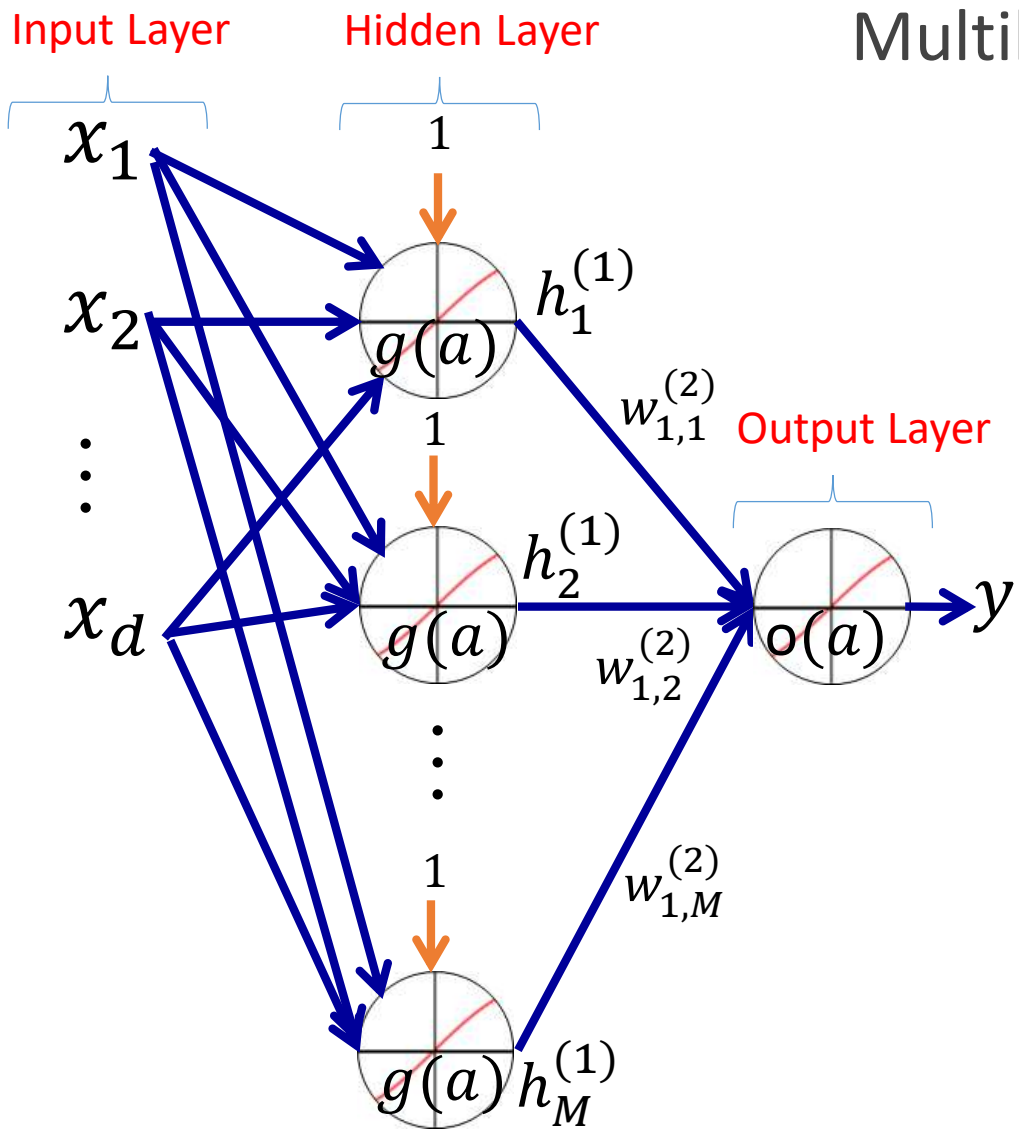
$$\begin{bmatrix}a_1^{(1)}(\boldsymbol{x})\\a_2^{(1)}(\boldsymbol{x})\\\vdots\\a_M^{(1)}(\boldsymbol{x})\end{bmatrix} = \begin{bmatrix}\boldsymbol{w}_1^{(1)} & b_1^{(1)}\\\boldsymbol{w}_2^{(1)} & b_2^{(1)}\\\vdots & \vdots\\\boldsymbol{w}_M^{(1)} & b_M^{(1)}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}$$

$$\boldsymbol{a}^{(1)}(\boldsymbol{x}) = \left[\boldsymbol{W}^{(1)}\boldsymbol{b}^{(1)}\right]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}$$

Column vector — Matrix — Column vector — Column vector

Hidden Layer Activation $\begin{bmatrix}h_1^{(1)}\\h_2^{(1)}\\\vdots\\h_M^{(1)}\end{bmatrix} = \begin{bmatrix}g\left(a_1^{(1)}\right)\\g\left(a_2^{(1)}\right)\\\vdots\\g\left(a_M^{(1)}\right)\end{bmatrix}$ $\boldsymbol{h}^{(1)} = \boldsymbol{g}\left(\boldsymbol{a}^{(1)}\right)$

Multilayer Neural Network



Output Layer Pre-activation

$$a_1^{(2)}(\boldsymbol{h}^{(1)}) = \left[\boldsymbol{w}_1^{(2)} b_1^{(2)}\right]\begin{bmatrix}\boldsymbol{h}^{(1)} \\ 1\end{bmatrix}$$

Output Layer Activation

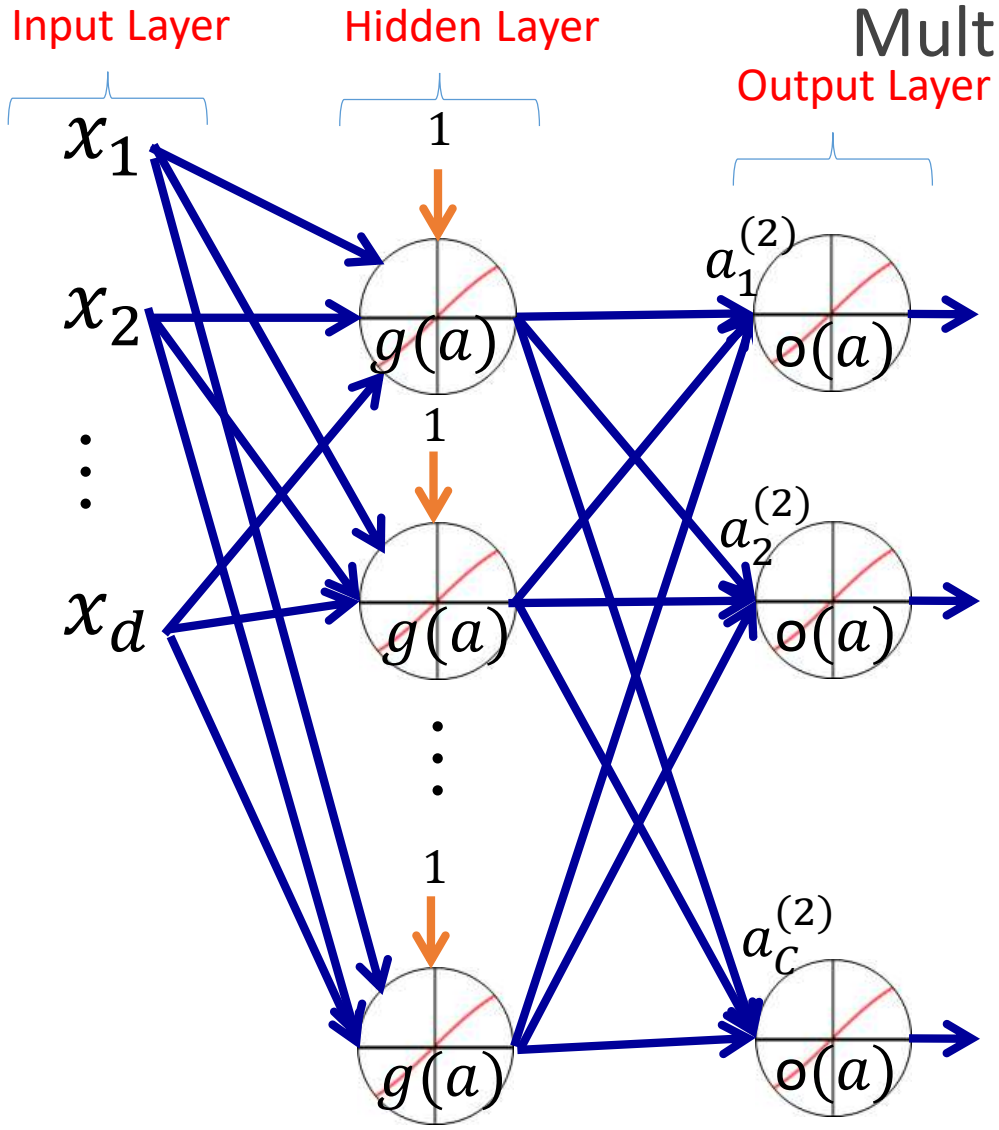
$$y = o\left(a_1^{(2)}(\boldsymbol{h}^{(1)})\right)$$

o can have many options

- sigmoid
- tanh
- Relu etc.

One special is softmax function

Multilayer Neural Network



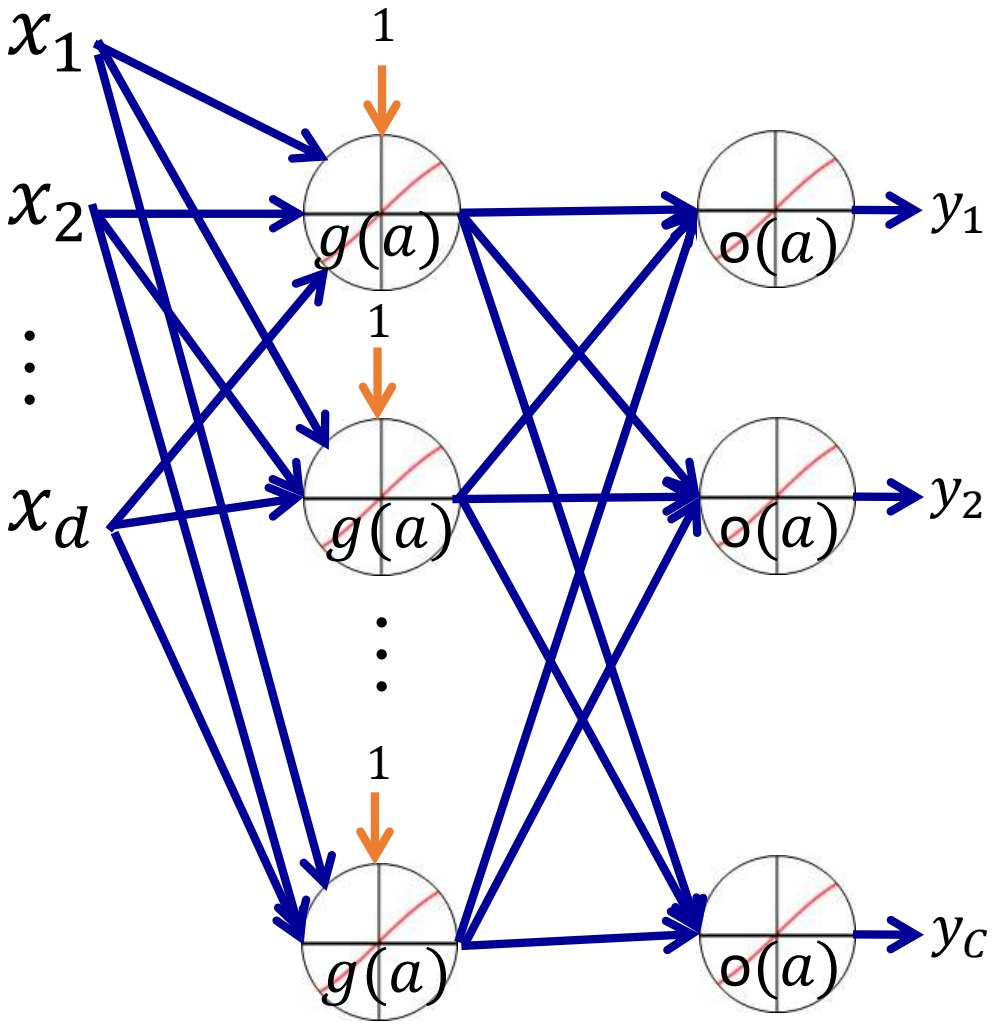
$$y_1 = o_1(\boldsymbol{a}^{(2)}) = \frac{\exp(a_1^{(2)})}{\sum_c \exp(a_c^{(2)})}$$

$$y_2 = o_2(\boldsymbol{a}^{(2)}) = \frac{\exp(a_2^{(2)})}{\sum_c \exp(a_c^{(2)})}$$

$$y_C = o_C(\boldsymbol{a}^{(2)}) = \frac{\exp(a_C^{(2)})}{\sum_c \exp(a_c^{(2)})}$$

Softmax function:
Exponentiate each component of output activation vector and elementwise divide by the sum of the exponentials.

Training a Neural Network – Loss Function



Probability that $\boldsymbol{x}$ belongs to class 1

Probability that $\boldsymbol{x}$ belongs to class 2

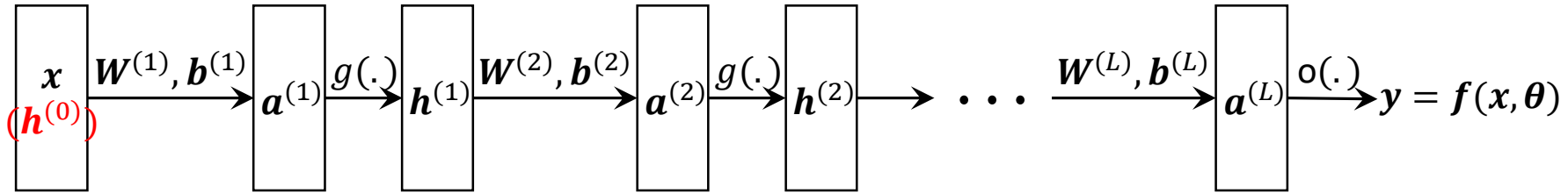
Probability that $\boldsymbol{x}$ belongs to class C

So we can aim to maximize the probability corresponding to the correct class for any example $\boldsymbol{x}$

$$\max \boldsymbol{y}_c$$
$$\equiv \max (log\, \boldsymbol{y}_c)$$
$$\equiv \min (-log\, \boldsymbol{y}_c)$$

Can be equivalently expressed as $-\sum_i \mathbb{I}(i = c)\log(\boldsymbol{y}_i)$ known as cross-entropy loss

Forward Pass in a Nutshell



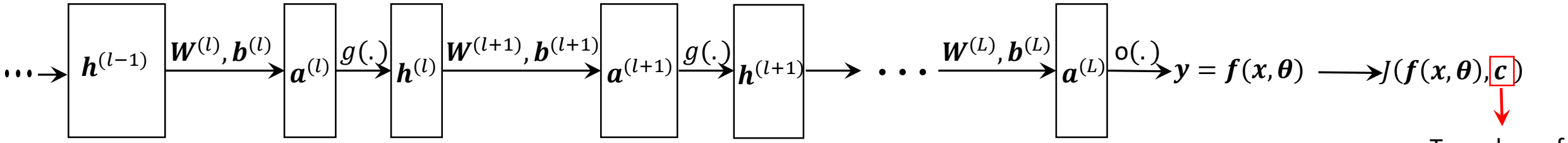
$\boldsymbol{\theta}$ is the collection of all learnable parameters i.e., all $\boldsymbol{W}$ and $\boldsymbol{b}$

Hidden layer pre-activation:
For $l = 1, \dots, L$; $\boldsymbol{a}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}$

Hidden layer activation:
For $l = 1, \dots, L-1$; $\boldsymbol{h}^{(l)} = g(\boldsymbol{a}^{(l)})$

Output layer activation:
For $l = L$; $\boldsymbol{h}^{(L)} = o(\boldsymbol{a}^{(L)}) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$

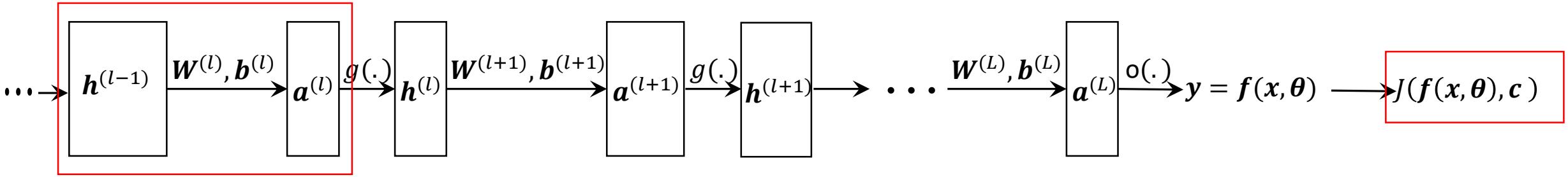
Backpropagation



For gradient descent we need to compute $\frac{\partial J}{\partial \boldsymbol{W}_{i,j}^{(l)}}$ and $\frac{\partial J}{\partial \boldsymbol{b}_{i}^{(l)}}$ $\forall i, j, l$

Backpropagation consists of intelligent use of the chain rule of differentiation for computation of these gradients

Backpropagation



$W_{i,j}^{(l)}$ influences J through $a_i^{(l)}$. That means J is a function of $a_i^{(l)}$ and $a_i^{(l)}$, in turn, is a function of $W_{i,j}^{(l)}$

$$\frac{\partial J}{\partial W_{i,j}^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial W_{i,j}^{(l)}}$$

$$a_i^{(l)} = \sum_j W_{i,j}^{(l)} h_j^{(l-1)} + b_i^{(l)} = W_{i,1}^{(l)} h_1^{(l-1)} + W_{i,2}^{(l)} h_2^{(l-1)} + \cdots + W_{i,j}^{(l)} h_j^{(l-1)} + \cdots + b_i^{(l)}$$

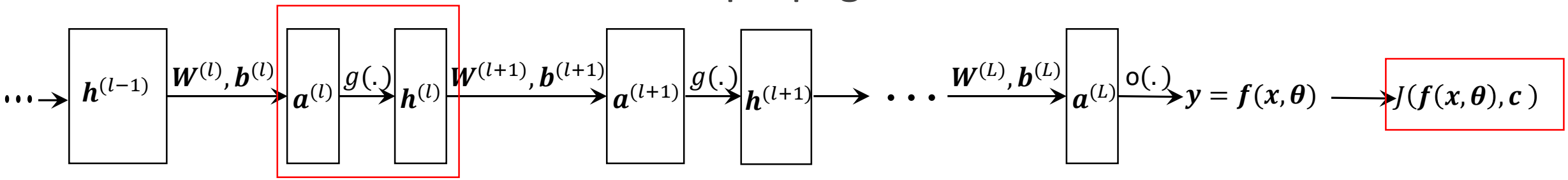
$$\Rightarrow \frac{\partial J}{\partial W_{i,j}^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial W_{i,j}^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}} h_j^{(l-1)} \quad \ldots (1)$$

And similarly $\frac{\partial J}{\partial b_i^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial b_i^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}}$ … (2)

We know $h_j^{(l-1)}$ from forward pass,

so we need $\frac{\partial J}{\partial a_i^{(l)}}$

Backpropagation

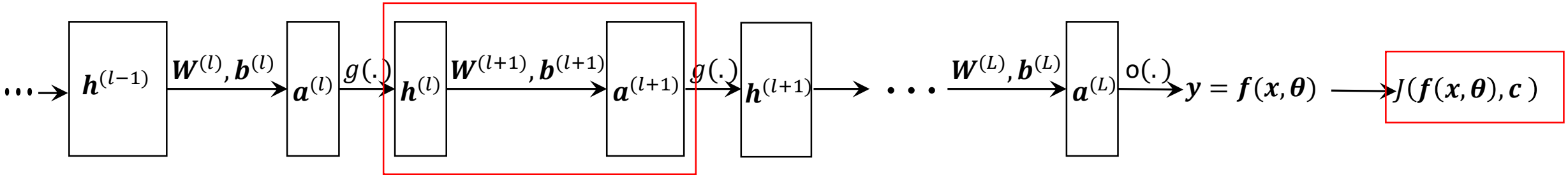


$\boldsymbol{a}_i^{(l)}$ influences J through $\boldsymbol{h}_i^{(l)}$. That means J is a function of $\boldsymbol{h}_i^{(l)}$ and $\boldsymbol{h}_i^{(l)}$, in turn, is a function of $\boldsymbol{a}_i^{(l)}$

$$\frac{\partial J}{\partial \boldsymbol{a}_i^{(l)}} = \frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}} \frac{\partial \boldsymbol{h}_i^{(l)}}{\partial \boldsymbol{a}_i^{(l)}} = \frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}} g'\left(\boldsymbol{a}_i^{(l)}\right) \qquad \text{... (3)}$$

We know $g'\left(\boldsymbol{a}_i^{(l)}\right)$ from forward pass, so we need $\frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}}$

Backpropagation



$\boldsymbol{h}_i^{(l)}$ influences J through $\boldsymbol{a}_{1:M_{l+1}}^{(l+1)}$. That means J is a function of $\boldsymbol{a}_{1:M_{l+1}}^{(l+1)}$ and each of $\boldsymbol{a}_{1:M_{l+1}}^{(l+1)}$, in turn, is a function of $\boldsymbol{h}_i^{(l)}$

Number of neurons in hidden layer $(l+1)$

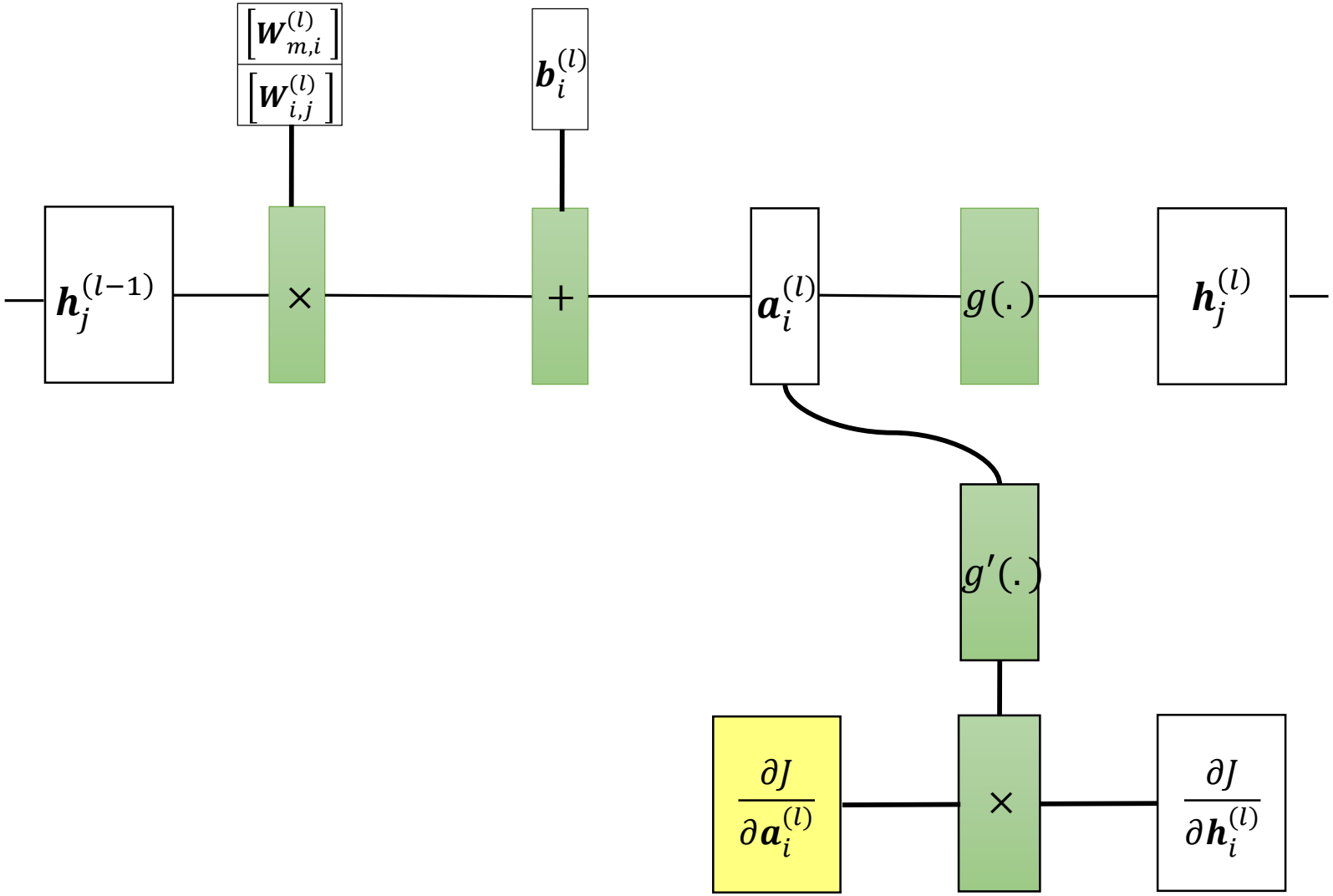
$$\frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}} = \sum_{m=1}^{M_{l+1}} \frac{\partial J}{\partial \boldsymbol{a}_m^{(l+1)}} \frac{\partial \boldsymbol{a}_m^{(l+1)}}{\partial \boldsymbol{h}_i^{(l)}}$$

Remember If, $u = f(x, y)$, *where* $x = \phi(t), y = \psi(t)$, then $\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial t}$

$$\boldsymbol{a}_m^{(l+1)} = \sum_j \boldsymbol{W}_{m,j}^{(l+1)} \boldsymbol{h}_j^{(l)} + \boldsymbol{b}_m^{(l+1)} = \boldsymbol{W}_{m,1}^{(l+1)} \boldsymbol{h}_1^{(l)} + \boldsymbol{W}_{m,2}^{(l+1)} \boldsymbol{h}_2^{(l)} + \cdots + \boldsymbol{W}_{m,i}^{(l+1)} \boldsymbol{h}_i^{(l)} + \cdots + \boldsymbol{b}_m^{(l+1)}$$

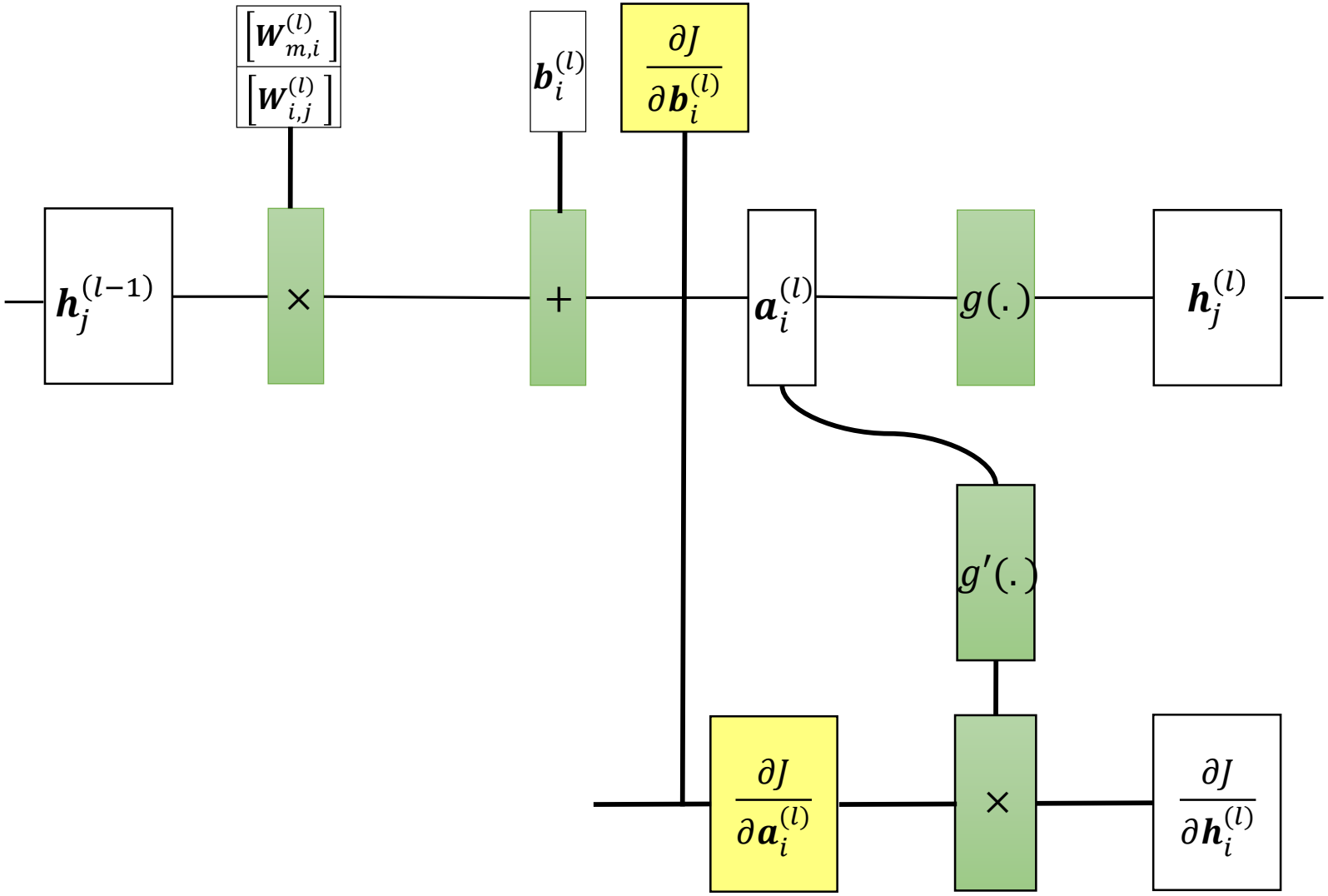
$$\Rightarrow \frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}} = \sum_{m=1}^{M_{l+1}} \frac{\partial J}{\partial \boldsymbol{a}_m^{(l+1)}} \boldsymbol{W}_{m,i}^{(l+1)} \qquad \ldots (4)$$

Backpropagation



$$\frac{\partial J}{\partial \boldsymbol{a}_i^{(l)}} = \frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}} g'\left(\boldsymbol{a}_i^{(l)}\right) \quad \text{... (3)}$$

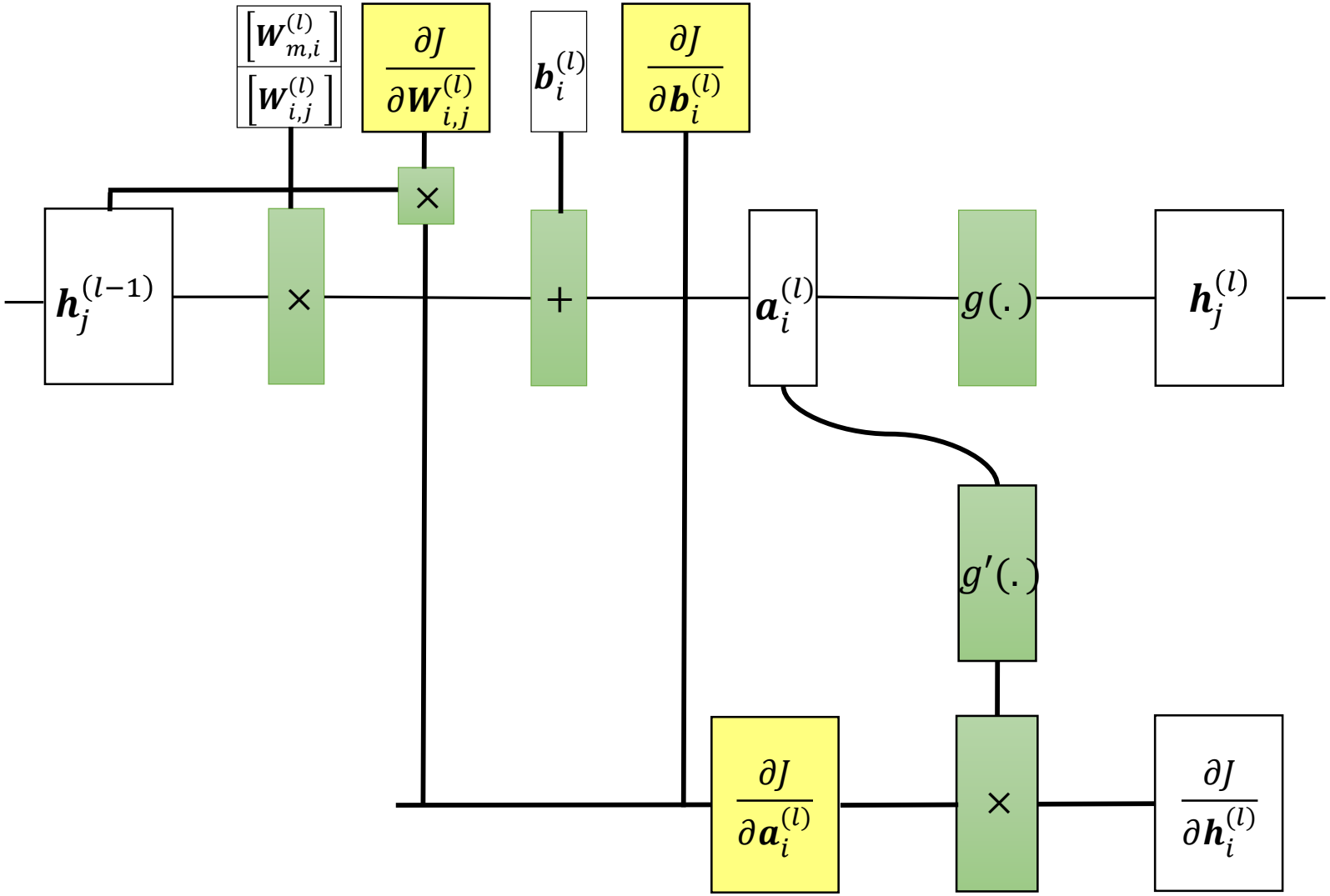
Backpropagation



$$\frac{\partial J}{\partial \boldsymbol{a}_i^{(l)}} = \frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}} g'\left(\boldsymbol{a}_i^{(l)}\right) \quad \text{... (3)}$$

$$\frac{\partial J}{\partial \boldsymbol{b}_i^{(l)}} = \frac{\partial J}{\partial \boldsymbol{a}_i^{(l)}} \quad \text{... (2)}$$

Backpropagation

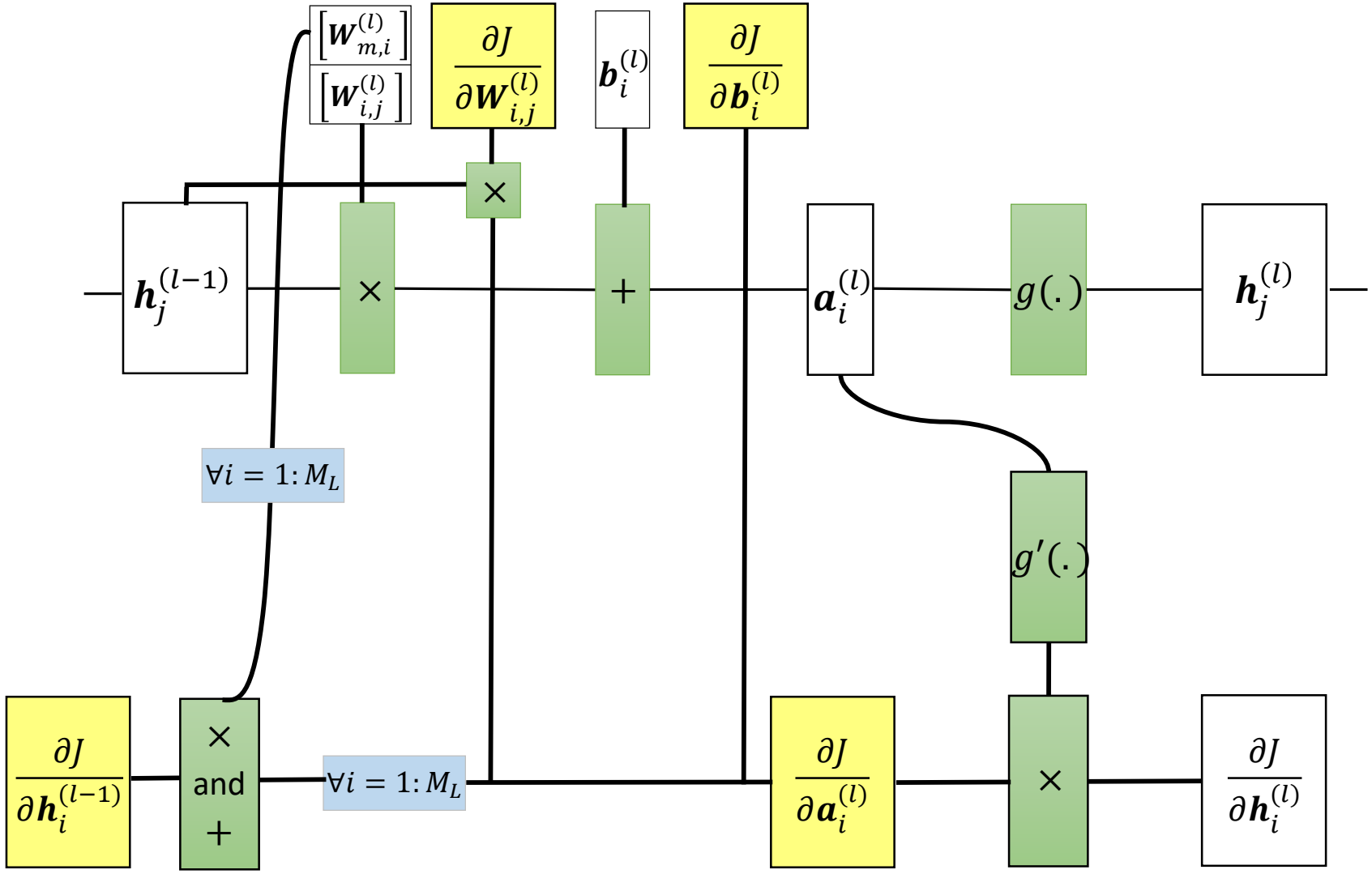


$$\frac{\partial J}{\partial a_i^{(l)}} = \frac{\partial J}{\partial h_i^{(l)}} g'\left(a_i^{(l)}\right) \quad \text{... (3)}$$

$$\frac{\partial J}{\partial b_i^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}} \quad \text{... (2)}$$

$$\frac{\partial J}{\partial W_{i,j}^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}} h_j^{(l-1)} \quad \text{... (1)}$$

Backpropagation



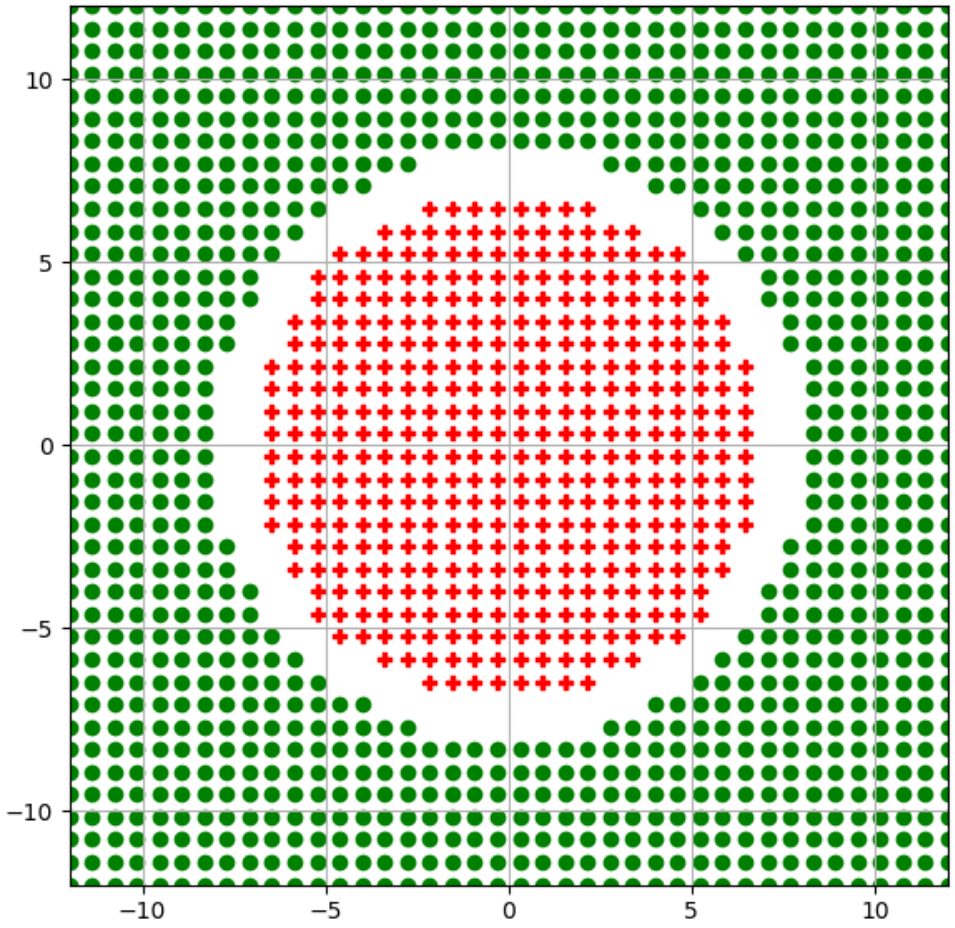
$$\frac{\partial J}{\partial \boldsymbol{a}_i^{(l)}} = \frac{\partial J}{\partial \boldsymbol{h}_i^{(l)}} g'\left(\boldsymbol{a}_i^{(l)}\right) \quad \text{... (3)}$$

$$\frac{\partial J}{\partial \boldsymbol{b}_i^{(l)}} = \frac{\partial J}{\partial \boldsymbol{a}_i^{(l)}} \quad \text{... (2)}$$

$$\frac{\partial J}{\partial \boldsymbol{W}_{i,j}^{(l)}} = \frac{\partial J}{\partial \boldsymbol{a}_i^{(l)}} \boldsymbol{h}_j^{(l-1)} \quad \text{... (1)}$$

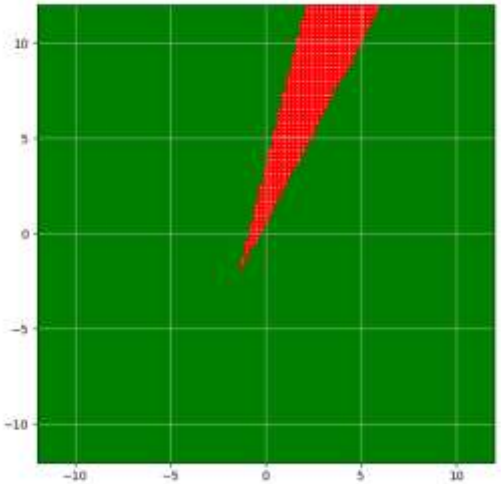
$$\frac{\partial J}{\partial \boldsymbol{h}_i^{(l-1)}} = \sum_{m=1}^{M_l} \frac{\partial J}{\partial \boldsymbol{a}_m^{(l)}} \boldsymbol{W}_{m,i}^{(l)} \quad \text{... (4)}$$

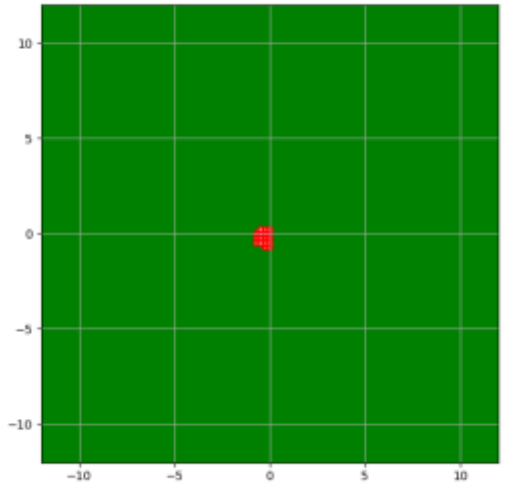
Training Multilayer Neural Network for non-linearly Separable Data

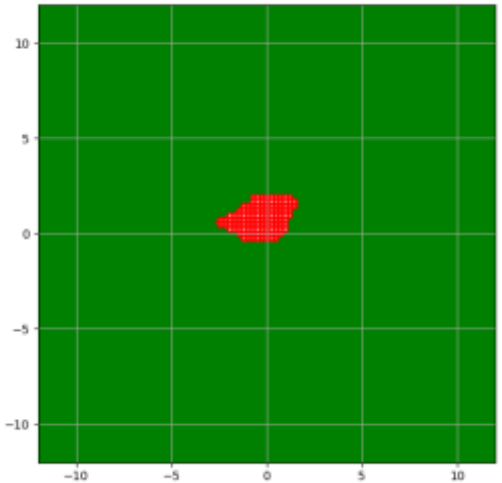


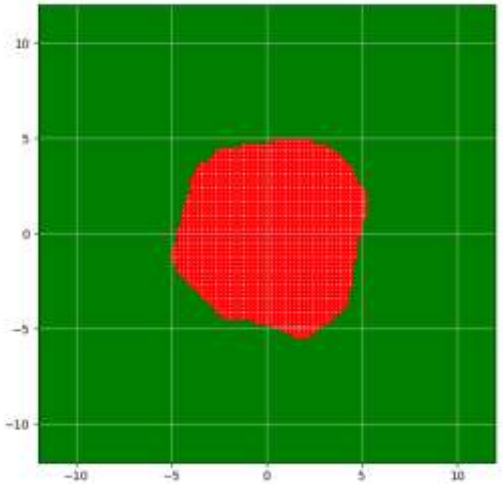
Training Data

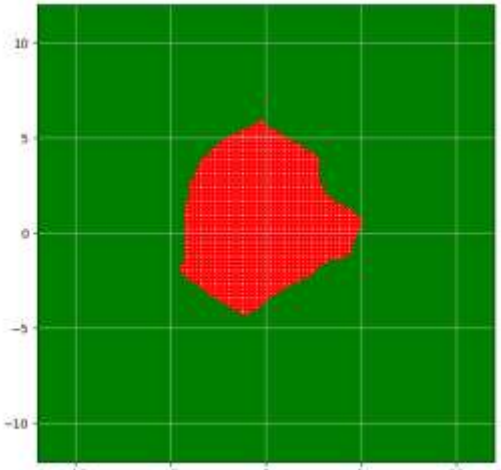
Learned Decision Boundary with Single Hidden Layer

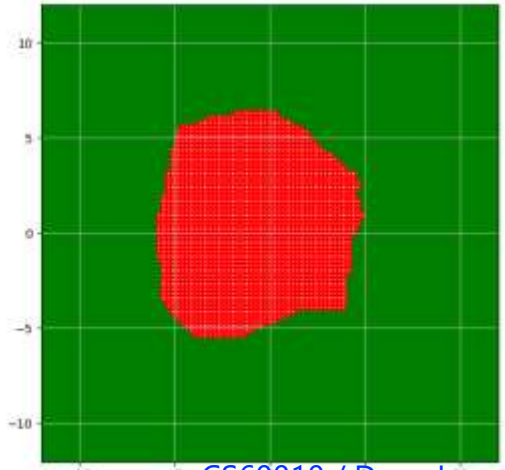

# of hidden neurons = 2

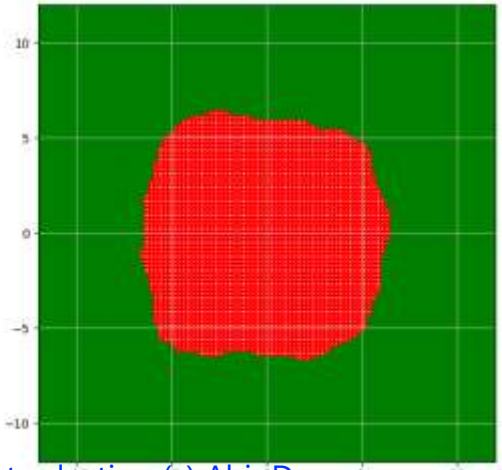

# of hidden neurons = 4

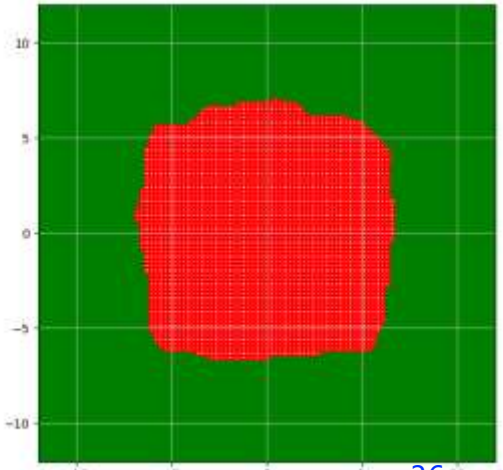

# of hidden neurons = 8


# of hidden neurons = 16

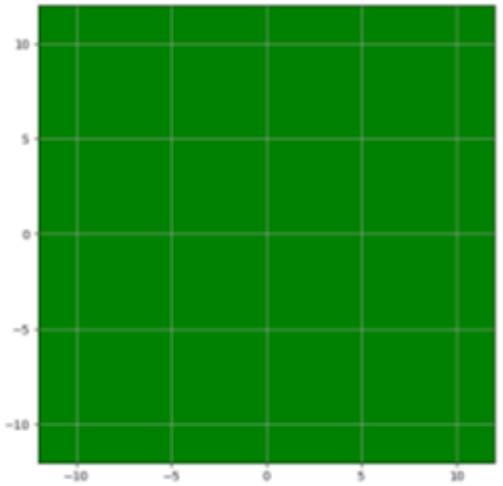

# of hidden neurons = 32


# of hidden neurons = 64

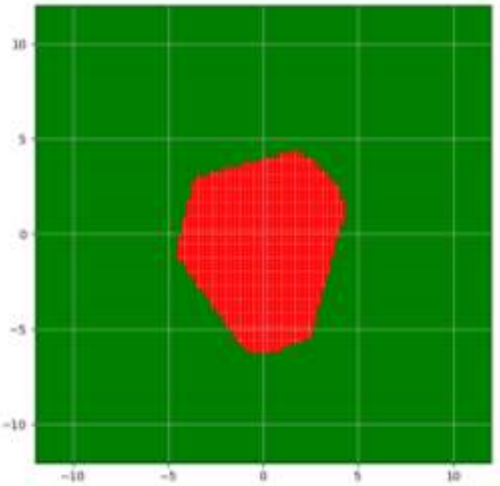

# of hidden neurons = 128


# of hidden neurons = 256

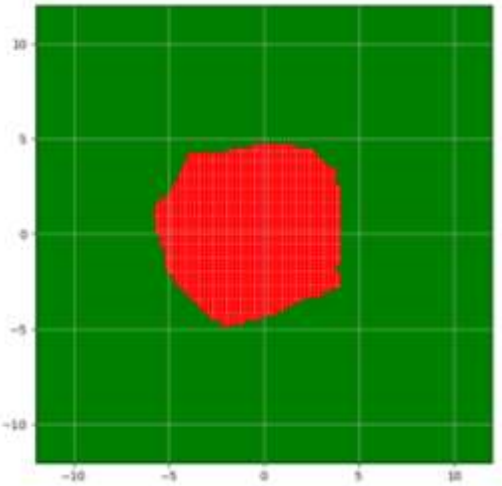
Learned Decision Boundary with Two Hidden Layers



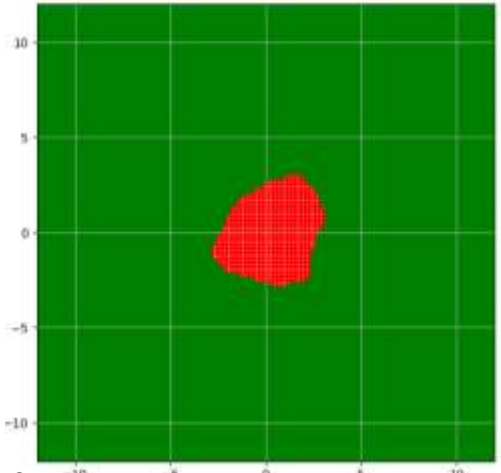
# of neurons in each hidden layer = 2



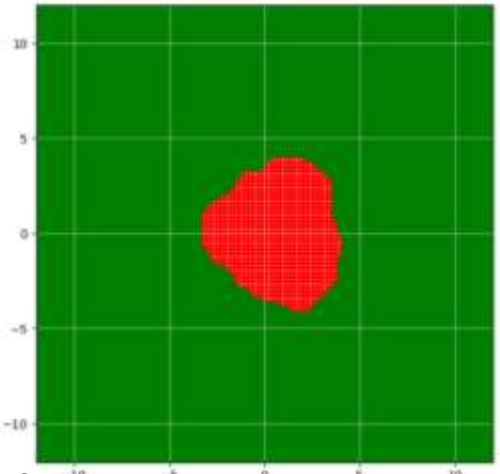
# of neurons in each hidden layer = 4



# of neurons in each hidden layer = 8



# of neurons in each hidden layer = 16



# of neurons in each hidden layer = 32