# ASIC & AI

***Stage 1:*** AI introduction & algorithms

***Members:*** Jce, Ryan, Taki Lee, Sam Kim, Brian Choi

*Created by:* Tony Do

**4.Oct.2019**

# Content

- Gradient descent
- Perceptron learning algorithm
- Logistic regression
- Softmax regression
- Multilayer neural network and Backpropagation
- Multilayer perceptron
- Support Vector machine

# Gradient descent

- Is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest as defined by the negative of the gradient.

⇒In machine learning we use this algorithm to update the parameters of our model.

- Math, give the cost function:

$$f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

The gradient will be calculated by:

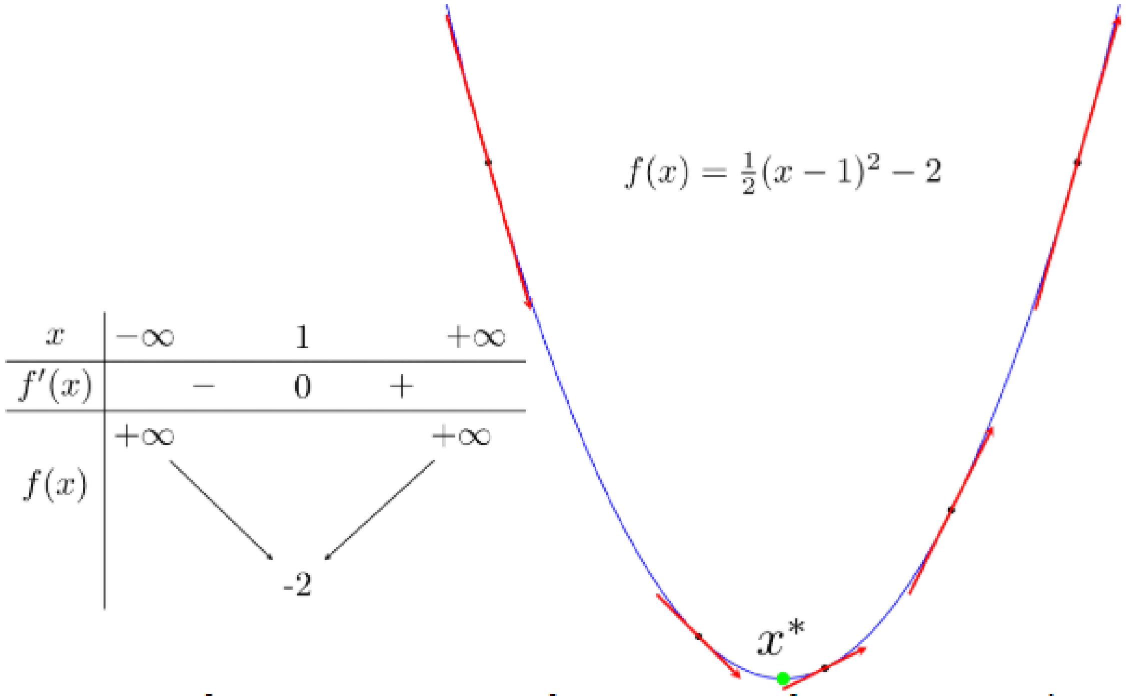
$$f'(m,b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N}\sum -2\,(y_i - (mx_i + b)) \end{bmatrix}$$

=> Solve ***f'(m,b) = 0*** to find out the value of parameters ***m, b***

# Gradient Descent

- Example

<source code>




Graph gradient descent of function f(x)

# Perceptron learning algorithm

- The perceptron is an algorithm for supervised learning of binary classifier. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.
- Definition of a threshold function:

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

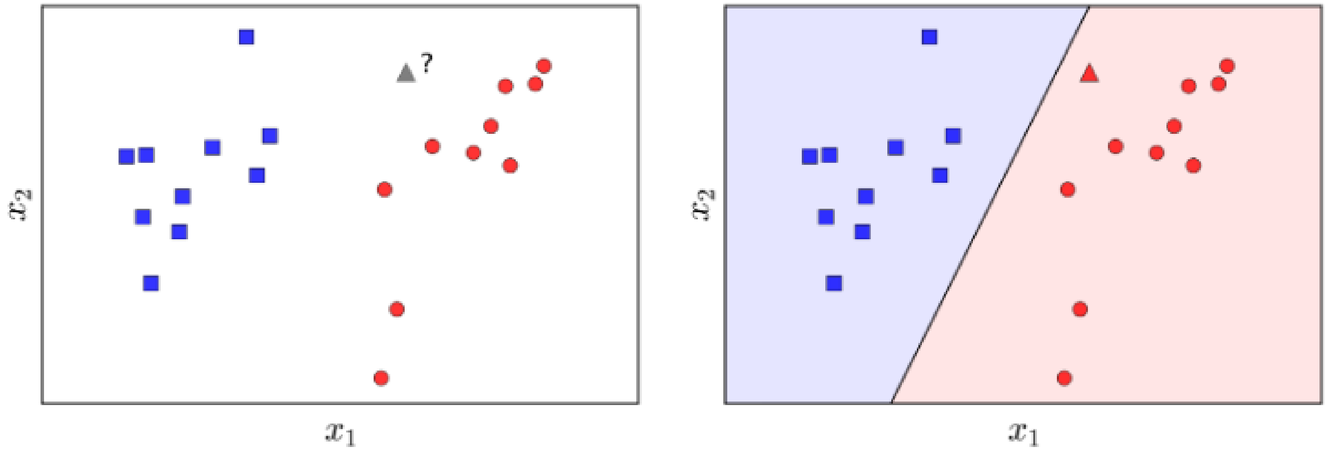
w is a vector of real-valued weights; w.x is the dot product $\sum_{i=1}^{m} w_i x_i$

m is the number of input to the perceptron

b is the bias

=> ***f(x)*** is used to classify x as either a positive or a negative instance.

*Figure*: Binary classification for 2 classes in 2 dimension



# Perceptron learning algorithm

- $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N] \in \mathbb{R}^{\boldsymbol{d \times N}}$ is matrix combine training data, each column $x_i$ is a data point in d dimension space.
- $\boldsymbol{y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N] \in \mathbb{R}^{\boldsymbol{1 \times N}}$ is label of each data point, $y_i$= 1 if $x_i$ belong to 1st class, $y_i$ = -1 if $x_i$ belong to the 2nd class.
- Equation function:

$$f_w(x) = w_1 x_1 + \dots + w_d x_d + w_0 = 0$$

$\Rightarrow f_w(x) = w^T x = 0$

$\Rightarrow$We can define label of input data by:

$$label(x) = \begin{cases} 1 & if\ w^T x \geq 0 \\ -1 & otherwise \end{cases}$$

In other word: $label(x) = \operatorname{sgn}(w^T x)$

# Perceptron learning algorithm

- ***Loss function***

Calculate the number of data point was set in wrong class

$$J_1(w) = \sum_{x_i \in \mathcal{M}} (-y_i \operatorname{sgn}(w^T x_i))$$

$\mathcal{M}$ is group of wrong classify data.

$\Rightarrow$ Find out w satisfy $J_1(w) = 0$

$\Rightarrow$ Apply Gradient Descent algorithm to update w:

$$J(w; x_i; y_i) = -y_i w^T x_i; \qquad \nabla_w J(w; x_i; y_i) = -y_i x_i$$

*Update **w** by:*

$w \leftarrow w - \eta(-y_i x_i) = w + \eta y_i x_i$

$\Rightarrow$ $\eta$ = 1 we have:

$\Rightarrow$ $w_{t+1} = w_t + y_i x_i$

# Perceptron learning algorithm

- 1. at t = 0, chose random vector $w_0$
- 2. at t, if don't have any miss classify data, stop algorithm
- 3. if $x_i$ is a miss classify, update:

$$w_{t+1} = w_t + y_i x_i$$

- 4. change t = t+1 and back to step 2.

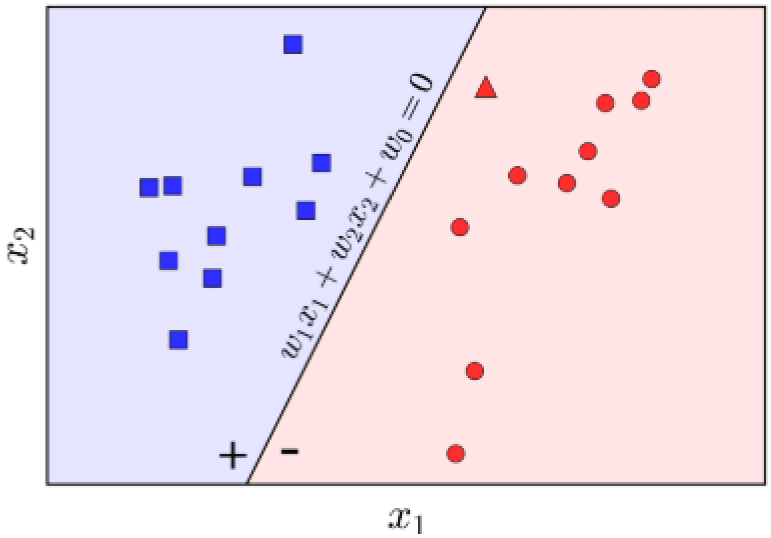


Figure: Boundary equation

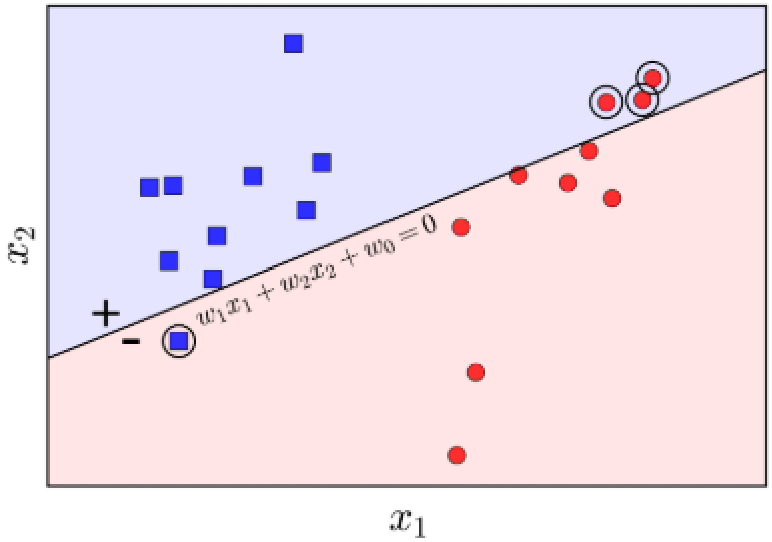


Figure: Random line with misclassified data

# Logistic regression

- ***Recap:***
- Linear regression to predict output y without limited upper or lower.
- PLA (perceptron learning) to predict output y only in 1 or -1

$\Rightarrow$Logistic regression will predict output y in between [0, 1]

- ***Formulation of logistic regression***

$$f(x) = \theta(w^T x)$$

$\theta$ is logistic function

We use sigmoid function to describe logistic regression

$$f(s) = \frac{1}{1 + e^{-s}} \triangleq \sigma(s)$$

***Loss function:***

$$J(w) = -\sum_{i=1}^{N} (y_i log z_i + (1 - y_i) \log(1 - z_i))$$

With:

$$z_i = \mathrm{f}(\mathrm{w}^{\mathrm{T}} \mathrm{x}_{\mathrm{i}})$$
$$P(y_i | x_i; w) = z_i^{y_i} (1 - z_i)^{1 - y_i}$$

# Logistic regression

- Example

(source code)

# Softmax regression

- We need a probability formulation satisfy each input $x, a_i$ show that input belong to class $i$

$\Rightarrow$ Each $a_i$ should be positive and $\sum a_i = 1$

$\Rightarrow$ If $z_i = w_i^T x$ larger, the probability of data fall into class $i$ will be more high

$\Rightarrow$ Function :

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)} \quad , \forall i = 1, 2, \ldots, C$$

$\Rightarrow$ This function was called by ***Softmax*** function.

- ***Loss function***

$$J(W) = \sum_{i=1}^{N} \| a_i - y_i \|_2^2$$

Solve equation by minimize ***J(W)*** with ***SDG*** algorithms:

$$\boldsymbol{W = W + \eta x_i (y_i - a_i)^T}$$

# *Softmax regression*

- Example (source code)

# Multilayer neural network and Backpropagation

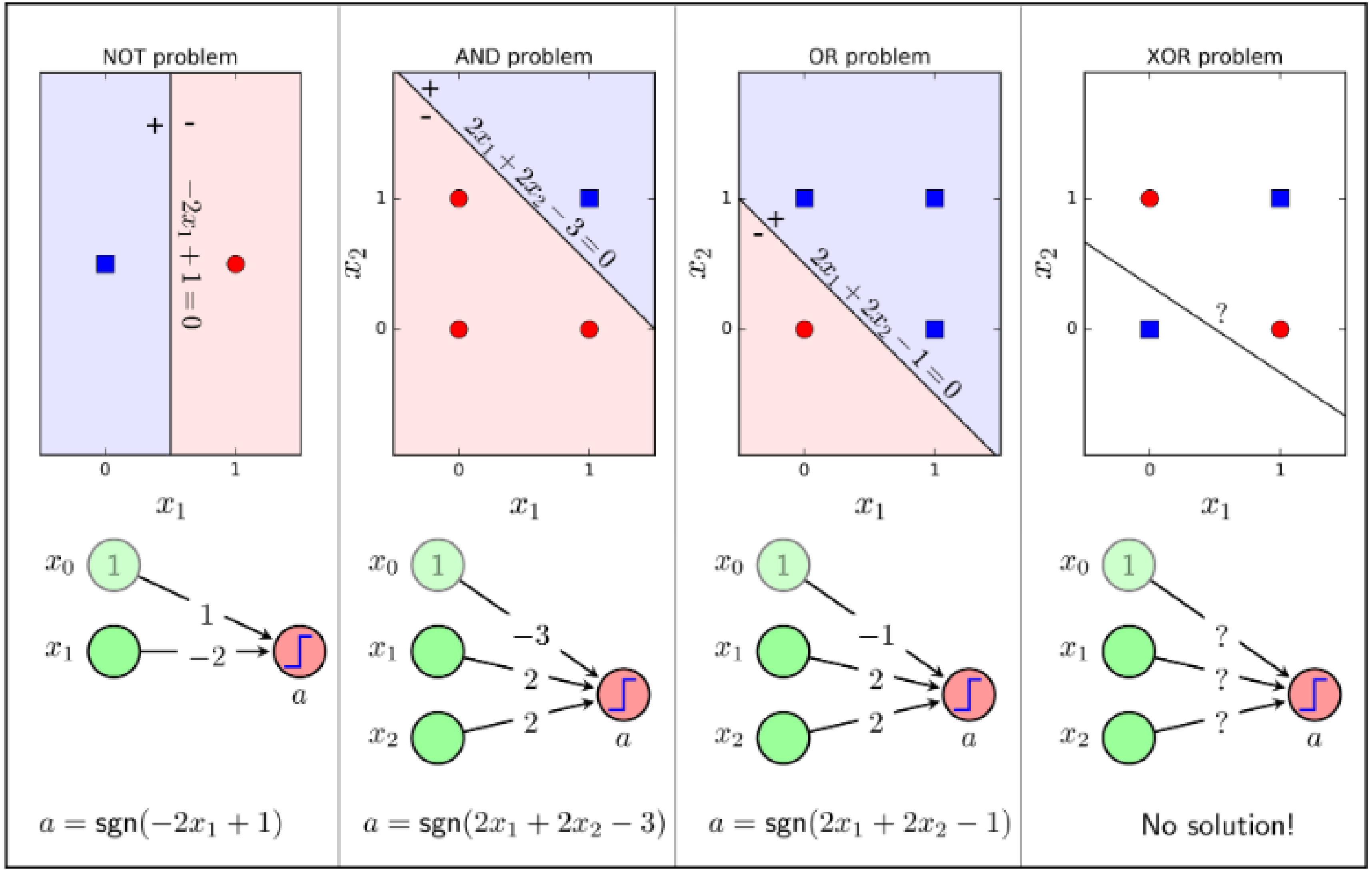



***Figure***: PLA with logic function basic.

# Multilayer neural network and backpropagation

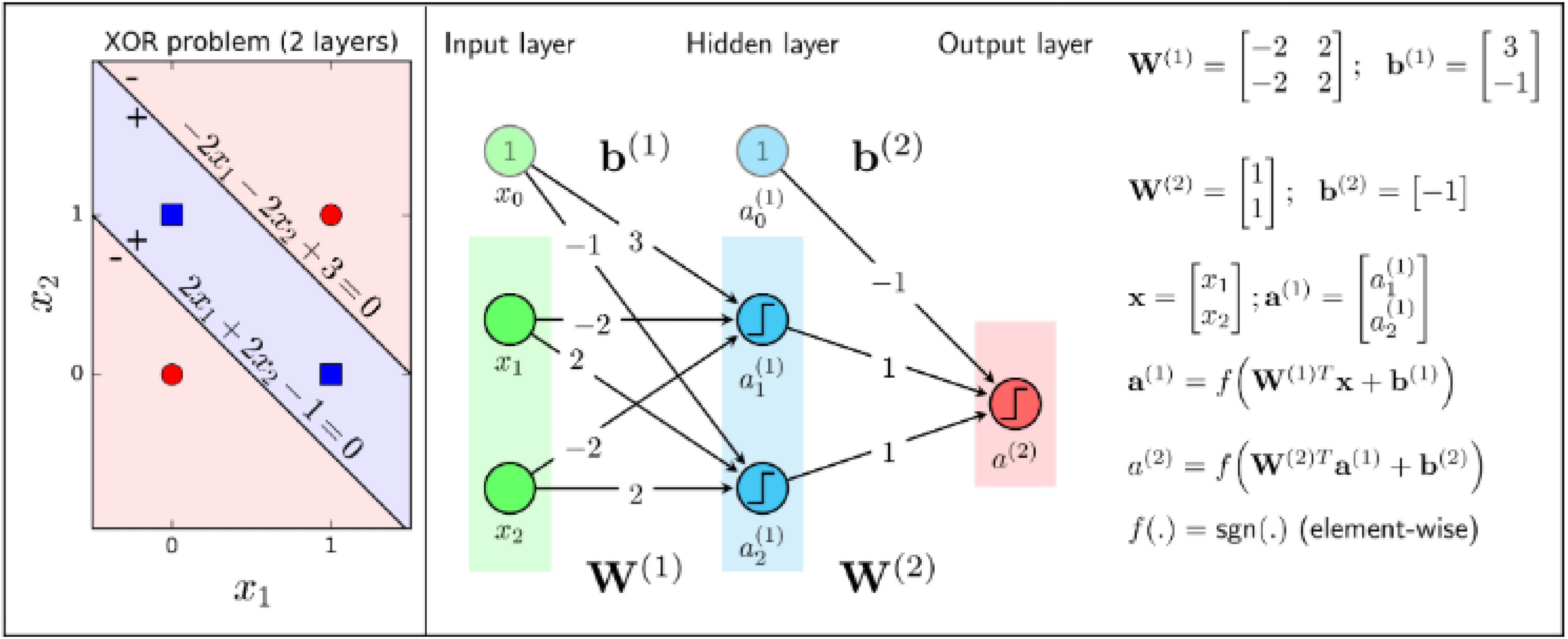



***Figure***: Multilayer Perceptron expression of XOR function

# Multilayer neural network and backpropagation

- **Layers**

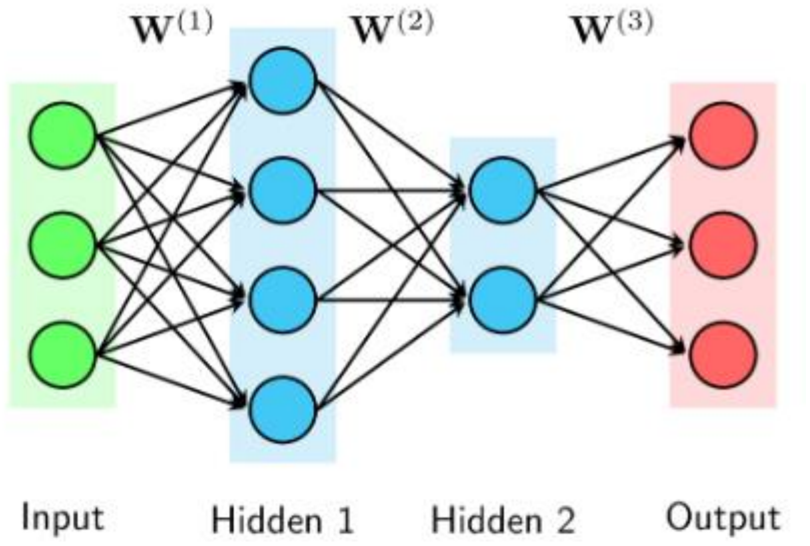



***Figure***: MLP with 2 hidden layers (biases were hided)

- **Weights and biases: W, b**
- **Activation functions**

Function output of each unit:

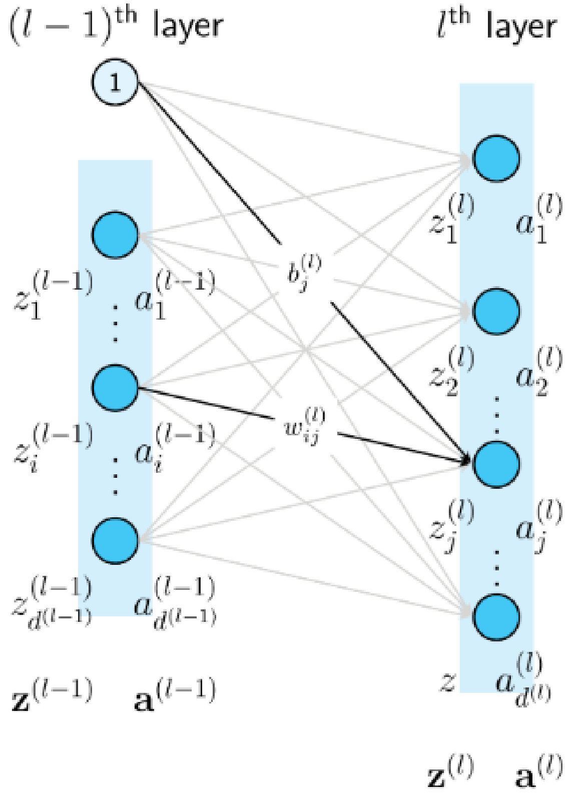
$$a_i^l = f(w_i^{(l)T} a^{l-1} + b_i^l)$$

Vector:

$$a^{(l)} = f(W^{(l)T} a^{(l-1)} + b^{(l)})$$

- **Units**

Each node in circle in layer we call is a unit.




$$\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$$

$$\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)} \times 1}$$

$$z_j^{(l)} = \mathbf{w}_j^{(l)T} \mathbf{a}^{(l-1)} + b_j^{(l)}$$

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)T} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

***Figure***: Symbols uses in MLP

# Multilayer neural network and backpropagation

- ***Backpropagation***

For optimization of loss function with using Gradient Descent algorithm for Weights and bias

$$\frac{\partial J}{\partial W^{(l)}}\ ;\ \frac{\partial J}{\partial b^{(l)}}\ , l = 1, 2, \ldots, L$$

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}}$$

$$= e_j^{(L)} a_i^{(L-1)}$$

$$e_j^{(l)} = \left(w_j^{(l+1)} e^{(l+1)}\right) f'\left(z_j^{(l)}\right)$$

$$\frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)}$$

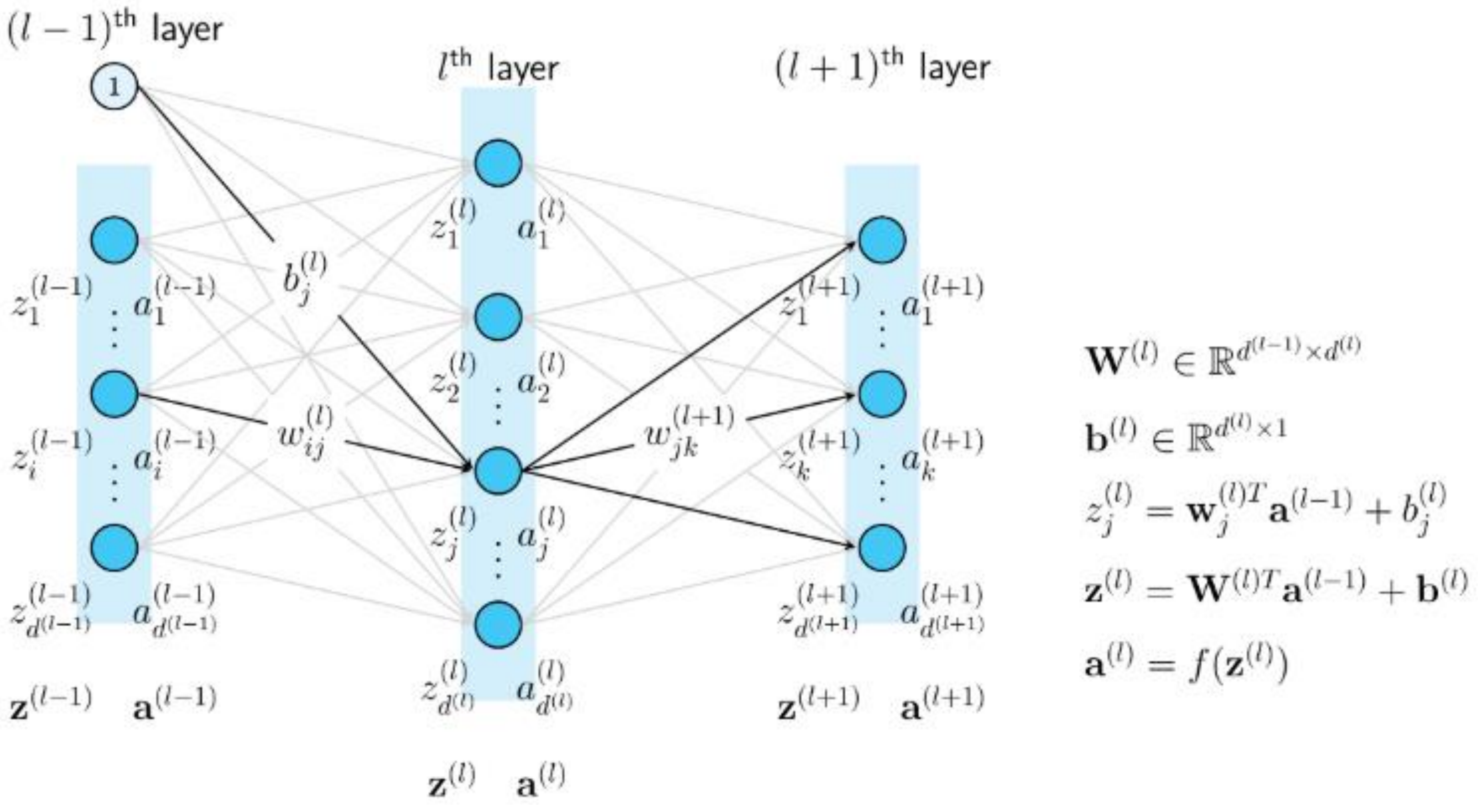



Figure: Illustrates method of calculating backpropagation from the last layer

# Multilayer neural network and backpropagation

- Example (source code)

# Schedule and plan

- *Time:* 30/Sept – 30/Nov
- *Machine learning:* 1/Oct – 18/Oct
- *Deep learning:* 21/Oct – 29/Nov
- 2 times per week, 2 hours per time.

| Time | Contents |
|---|---|
| 1 – 4/Oct | AI Introduction & requirements & applications<br>Linear regression<br>Overfitting<br>K-nearest neighbors<br>K-mean clustering<br>Naïve Bayes classifier<br>Gradient descent |
| 7 – 11/Oct | Perceptron learning algorithm<br>Logistic regression<br>Softmax regression<br>Multilayer neural network and Backpropagation<br>Multilayer perceptron<br>Support Vector machine |

| Time | Contents |
|---|---|
| 14 – 18/Oct | Deep neural network<br>Convolutional neural network: faster R-CNN, SSD (Single Shot MultiBox Detector, YOLO (You Look Only Once) |
| 21 – 25/Oct | Deep neural network<br>Convolutional neural network: faster R-CNN, SSD (Single Shot MultiBox Detector, YOLO (You Look Only Once) |
| 28 – 1/Nov | Deep neural network<br>Convolutional neural network: faster R-CNN, SSD (Single Shot MultiBox Detector, YOLO (You Look Only Once) |
| 4 – 8/Nov | Convolutional neural network: faster R-CNN, SSD (Single Shot MultiBox Detector, YOLO (You Look Only Once) |
| 11 – 15/Nov | Time Series: seq-to-seq modeling, RNN, LSTM, GRU<br>Time Series: seq-to-seq modeling, RNN, LSTM, GRU<br>Introduction to Reinforcement learning |

ASICLAND
(주)에이직랜드

# THANK YOU SO MUCH!