Keras 1장

Neural networks foundations

오세진

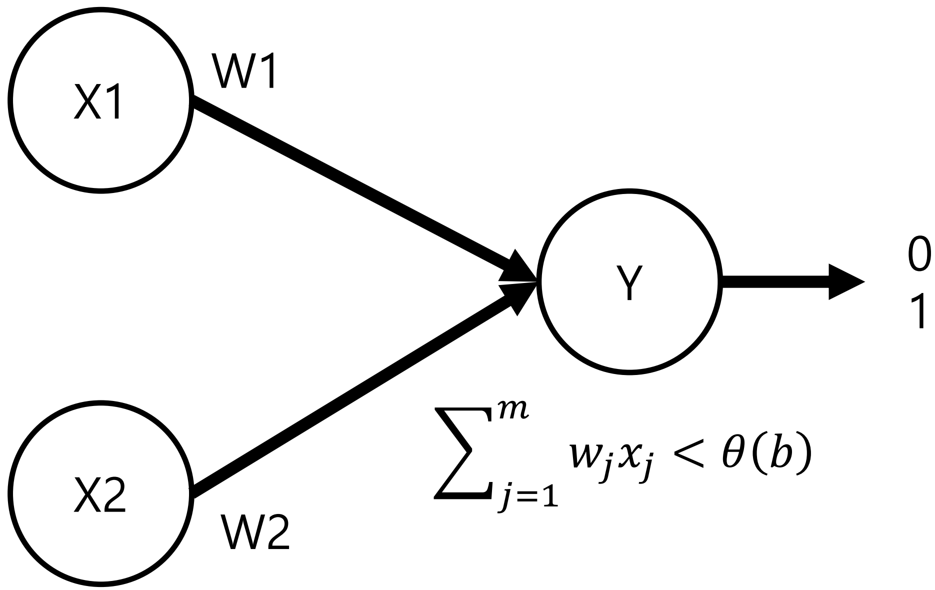
목차

1. Perceptron
    1. The 1st example of
2. Multilayer perceptron – the first example for a network
    1. Problems in training the perceptron and a solution
    2. Activation f(x)
        1. sigmoid, ReLU, others
3. Backpropagation

Perceptron

정의

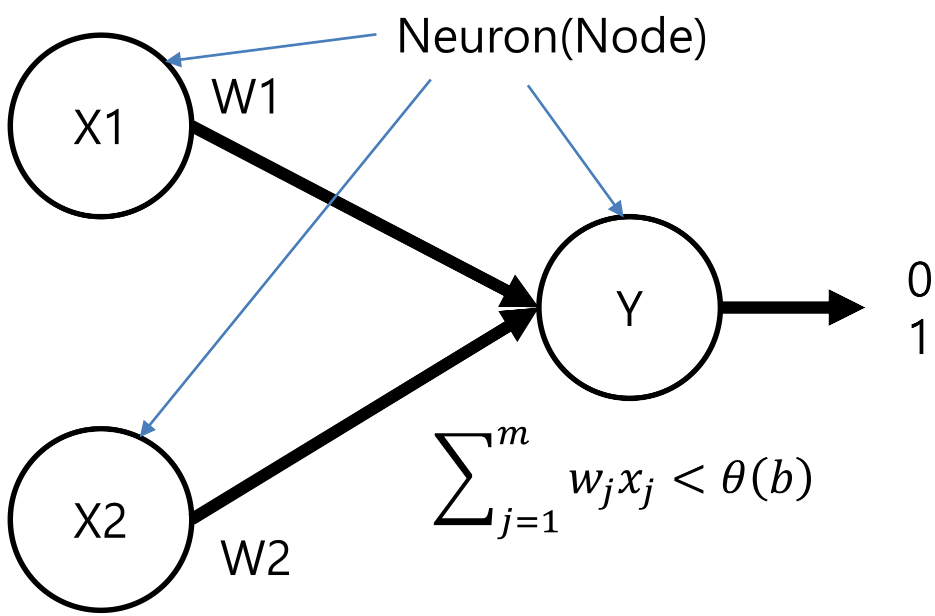
- Simple algorithm which, given an input vector x outputs either 1 (yes) or 0 (no)



Perceptron

정의

- Simple algorithm which, given an input vector x outputs either 1 (yes) or 0 (no)



$$\sum_{j=1}^{m} w_j x_j < \theta(b)$$

Perceptron

정의

- Simple algorithm which, given an input vector x outputs either 1 (yes) or 0 (no)

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

입력별 가중치

$$wx = \sum_{j=1}^{m} w_j x_j$$

b : $bias$

w: $weight$

Perceptron

정의

- Simple algorithm which, given an input vector x outputs either 1 (yes) or 0 (no)

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

입력 벡터

$$\begin{bmatrix} x1 \\ . \\ . \\ . \\ xm \end{bmatrix}$$

$$wx = \sum_{j=1}^{m} w_j x_j$$

b : $bias$

w: $weight$

Perceptron

정의

- Simple algorithm which, given an input vector x outputs either 1 (yes) or 0 (no)

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

Activation 강도 조절

$$wx = \sum_{j=1}^{m} w_j x_j$$

b : $bias$

w: $weight$

Perceptron

-Simple and multilayer

AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

NAND

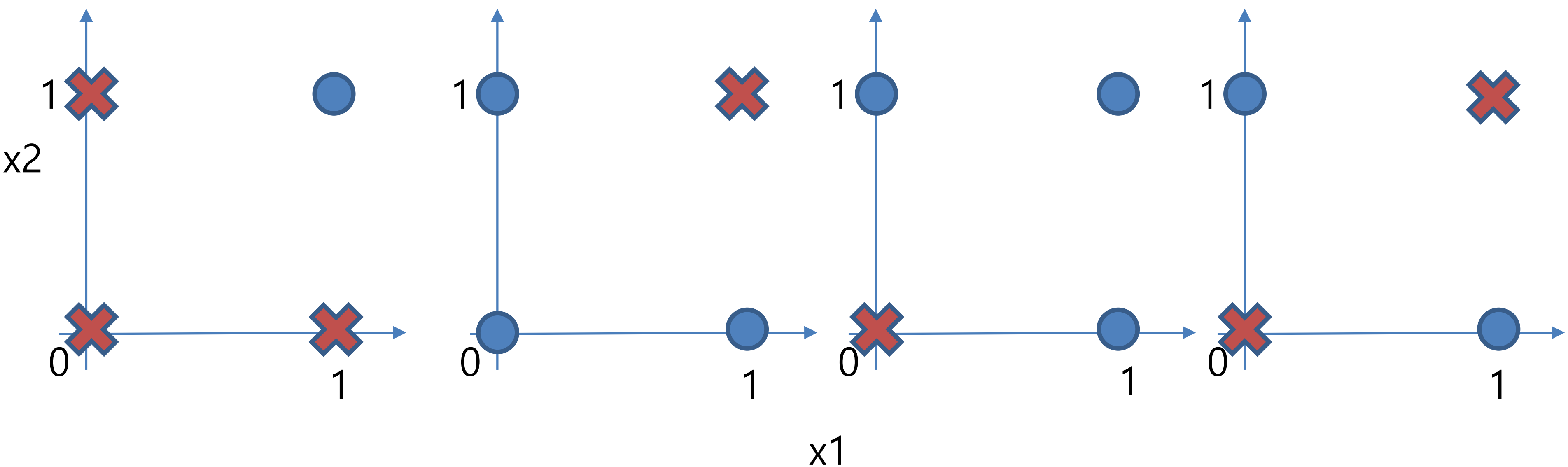
x1	x2	y
0	0	1
0	1	1
1	0	1
1	1	0

OR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron

- Simple and multilayer

Simple

AND

x1	**x2**	**y**
0	0	0
0	1	0
1	0	0
1	1	1

NAND

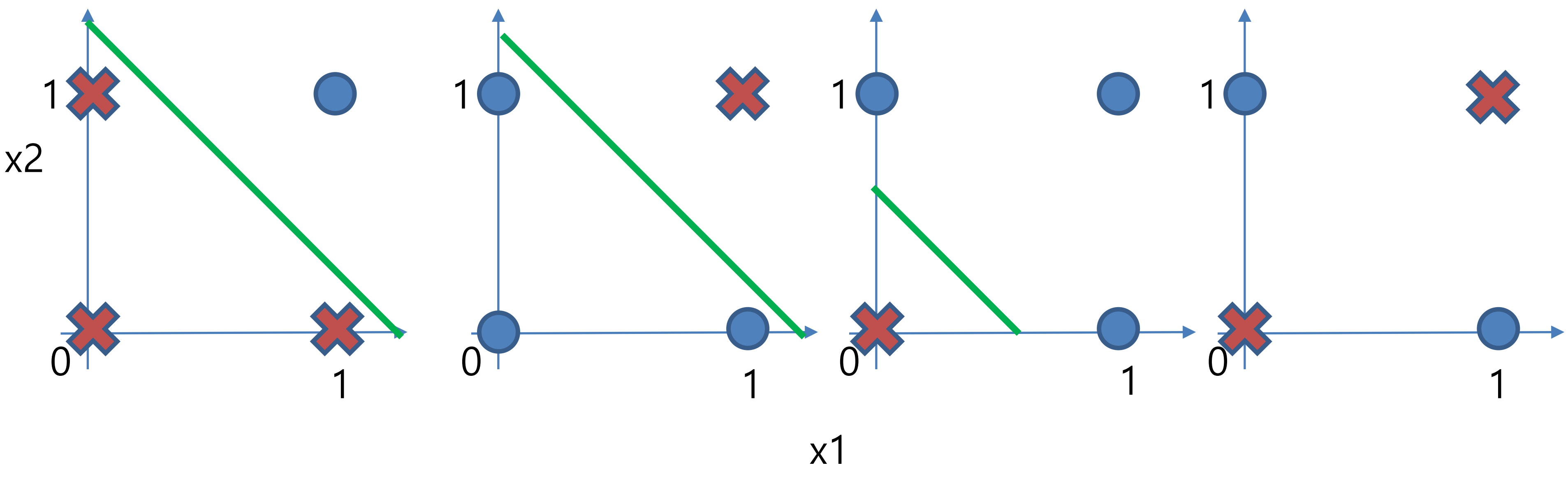
x1	**x2**	**y**
0	0	1
0	1	1
1	0	1
1	1	0

OR

x1	**x2**	**y**
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x1	**x2**	**y**
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron

-Single and multilayer

Simple | Multilayer

AND

x1	**x2**	**y**
0	0	0
0	1	0
1	0	0
1	1	1

NAND

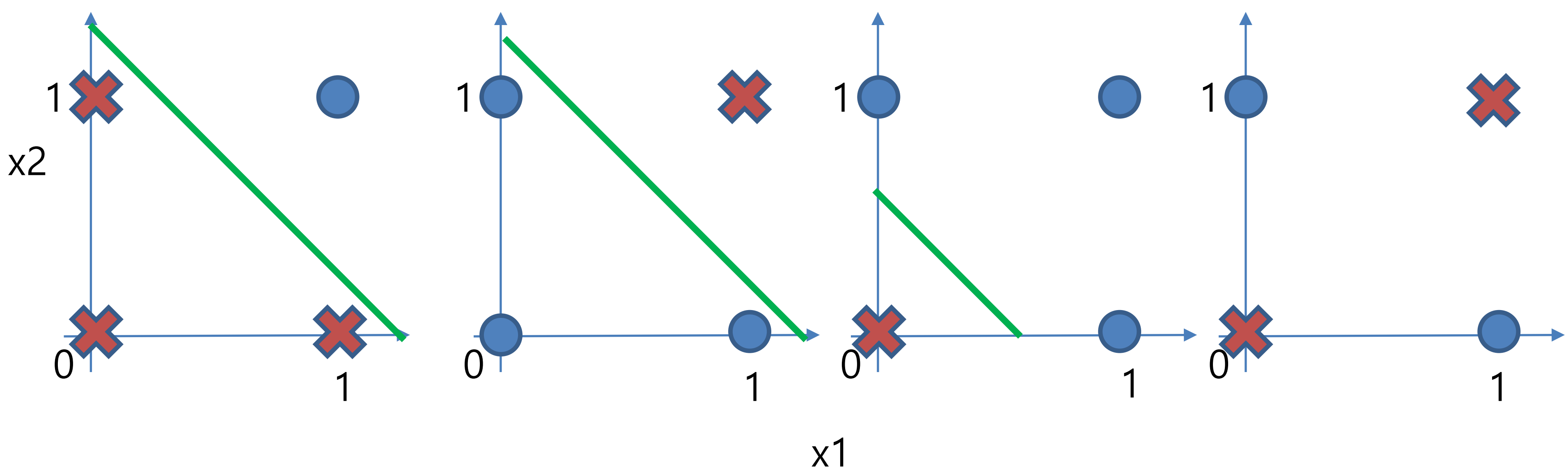
x1	**x2**	**y**
0	0	1
0	1	1
1	0	1
1	1	0

OR

x1	**x2**	**y**
0	0	0
0	1	1
1	0	1
1	1	1

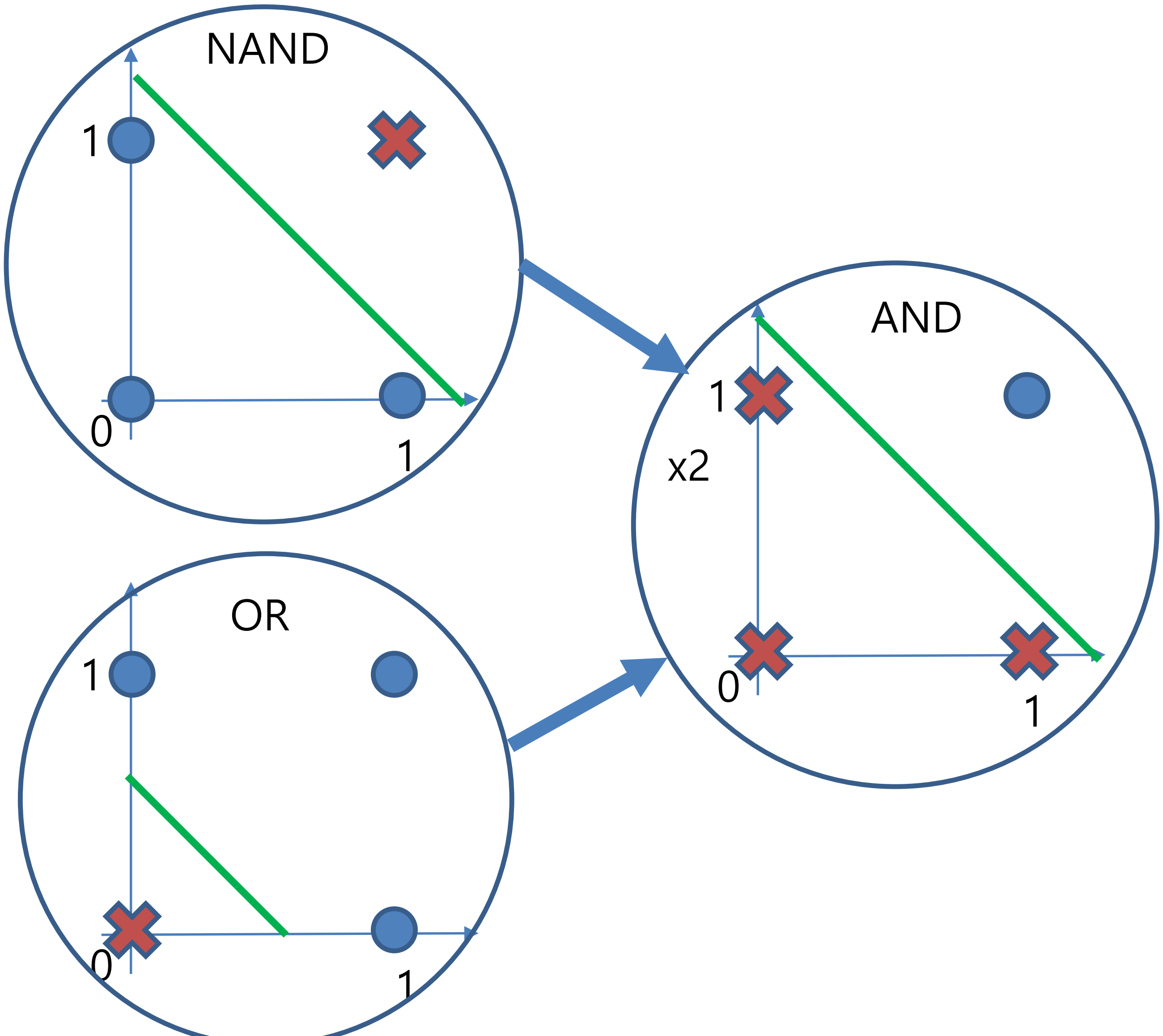
XOR

x1	**x2**	**y**
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron

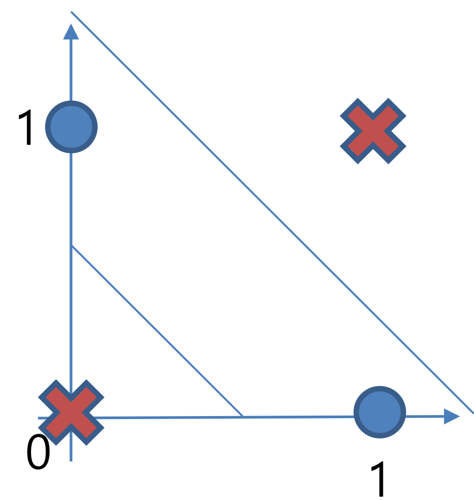
-Single and multilayer



Multilayer

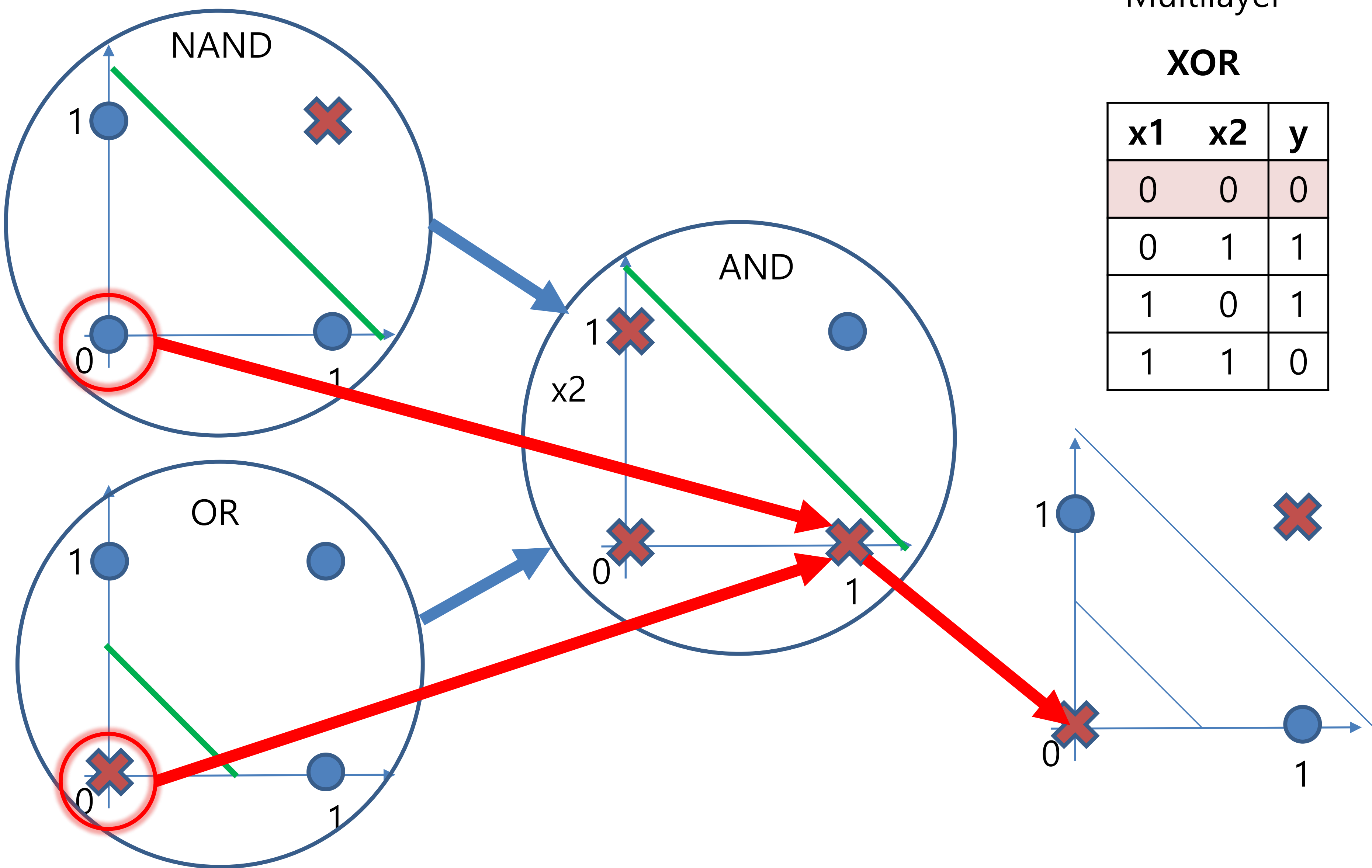
XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



Perceptron

-Single and multilayer



Multilayer

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

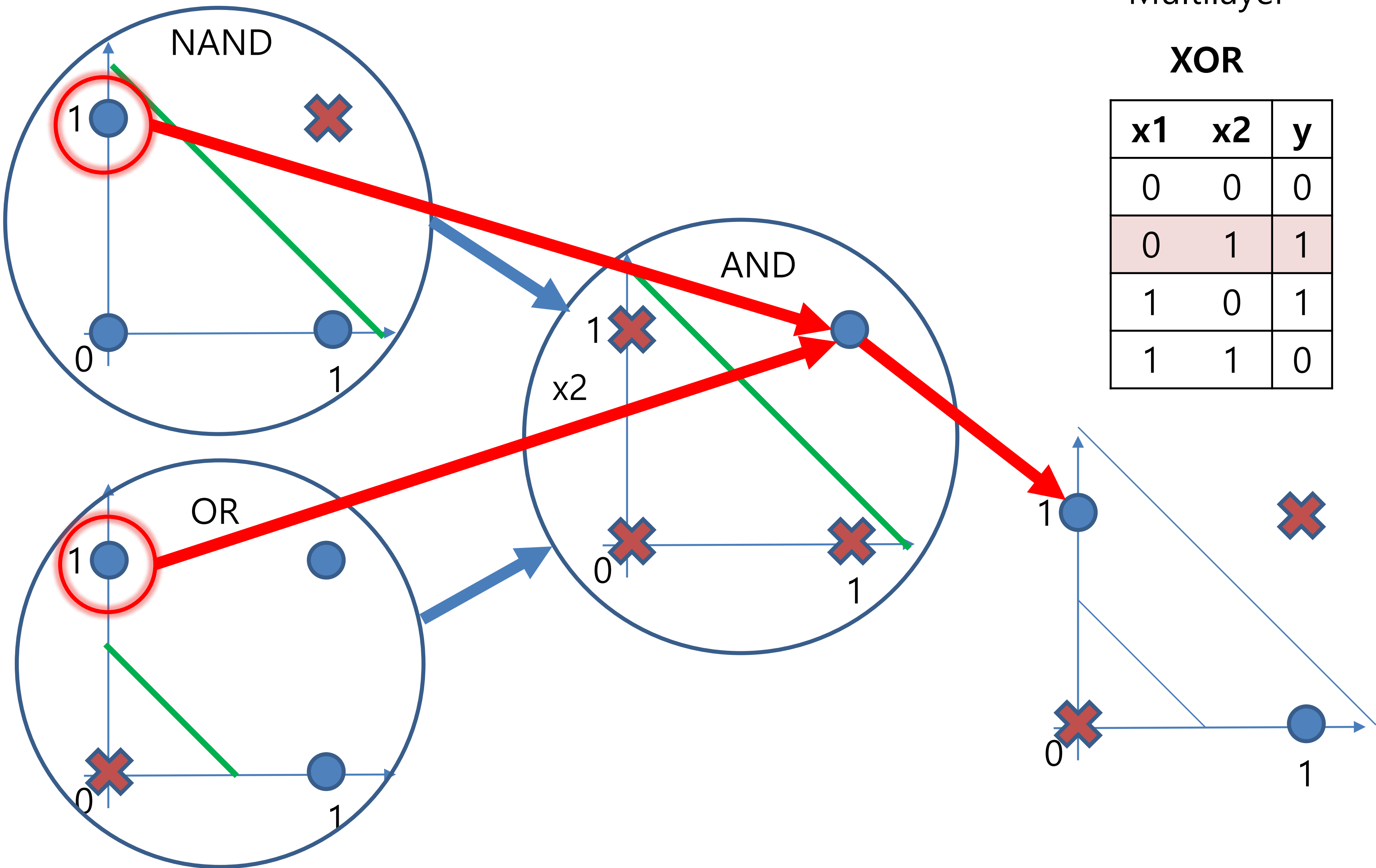
Perceptron

-Single and multilayer

Multilayer

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



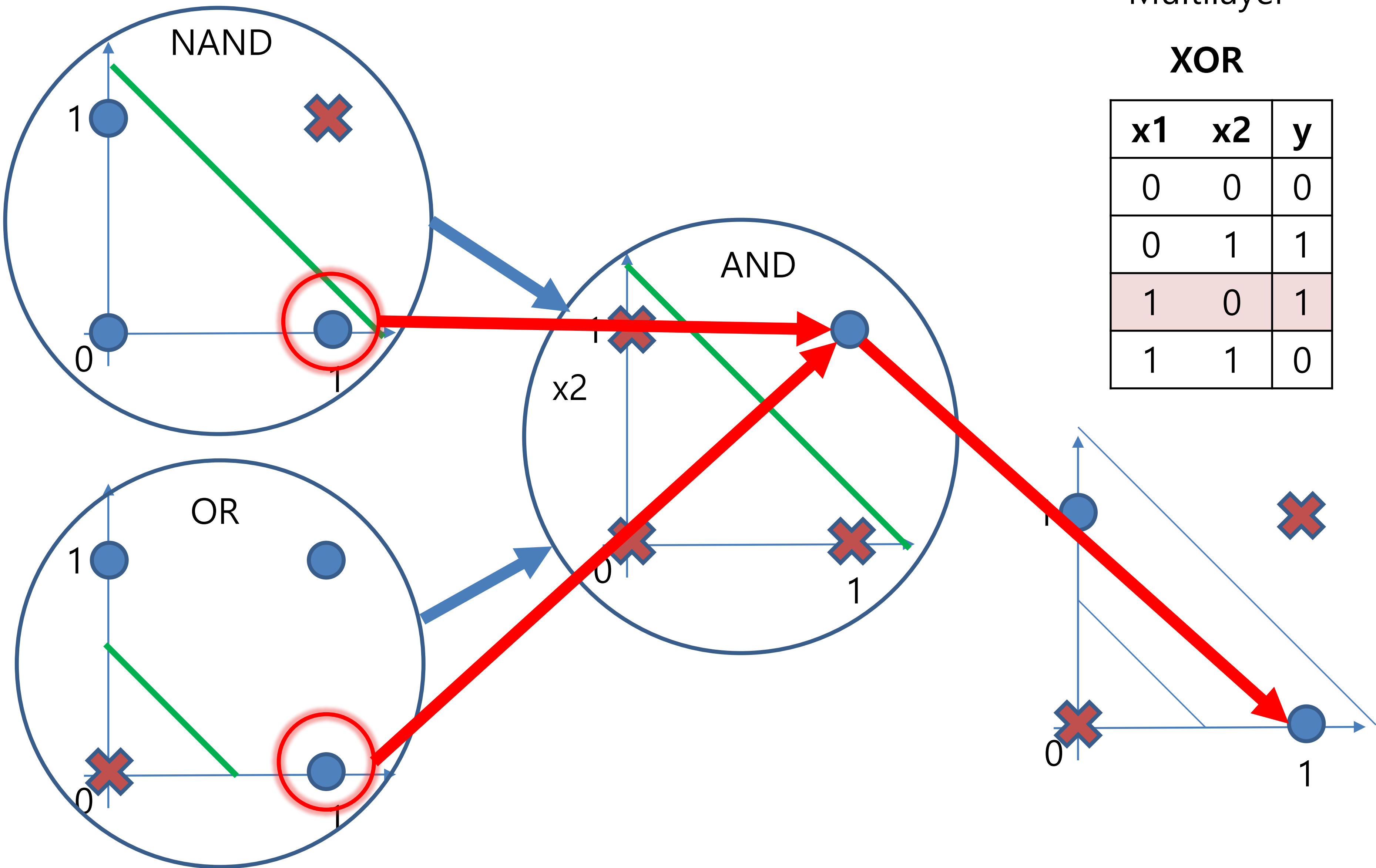
Perceptron

-Single and multilayer

Multilayer

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



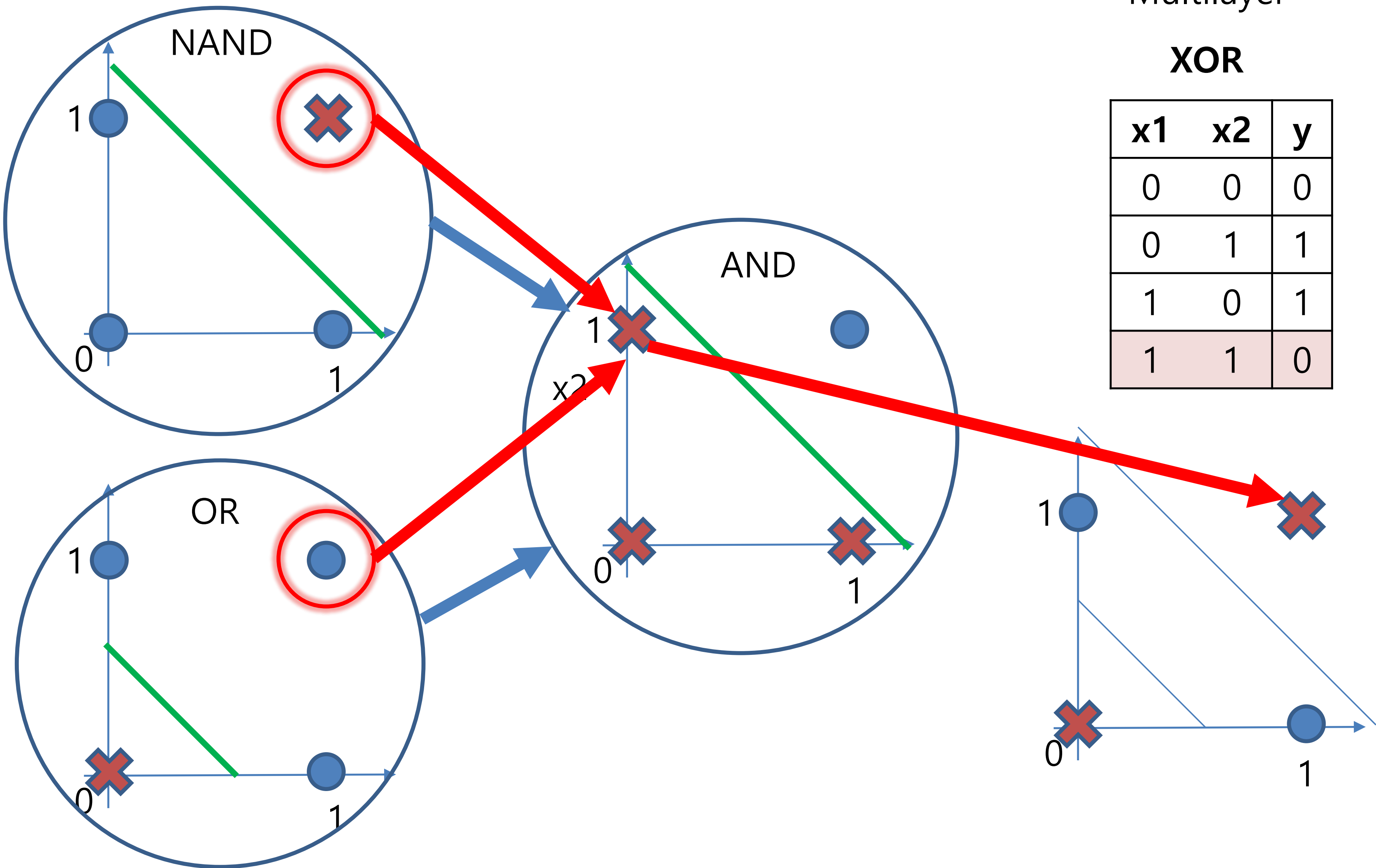
Perceptron

-Single and multilayer

Multilayer

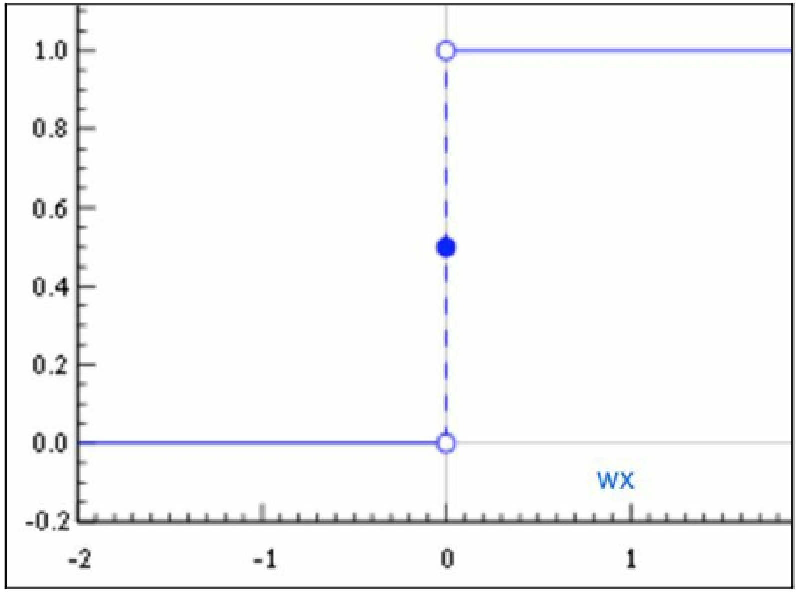
XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



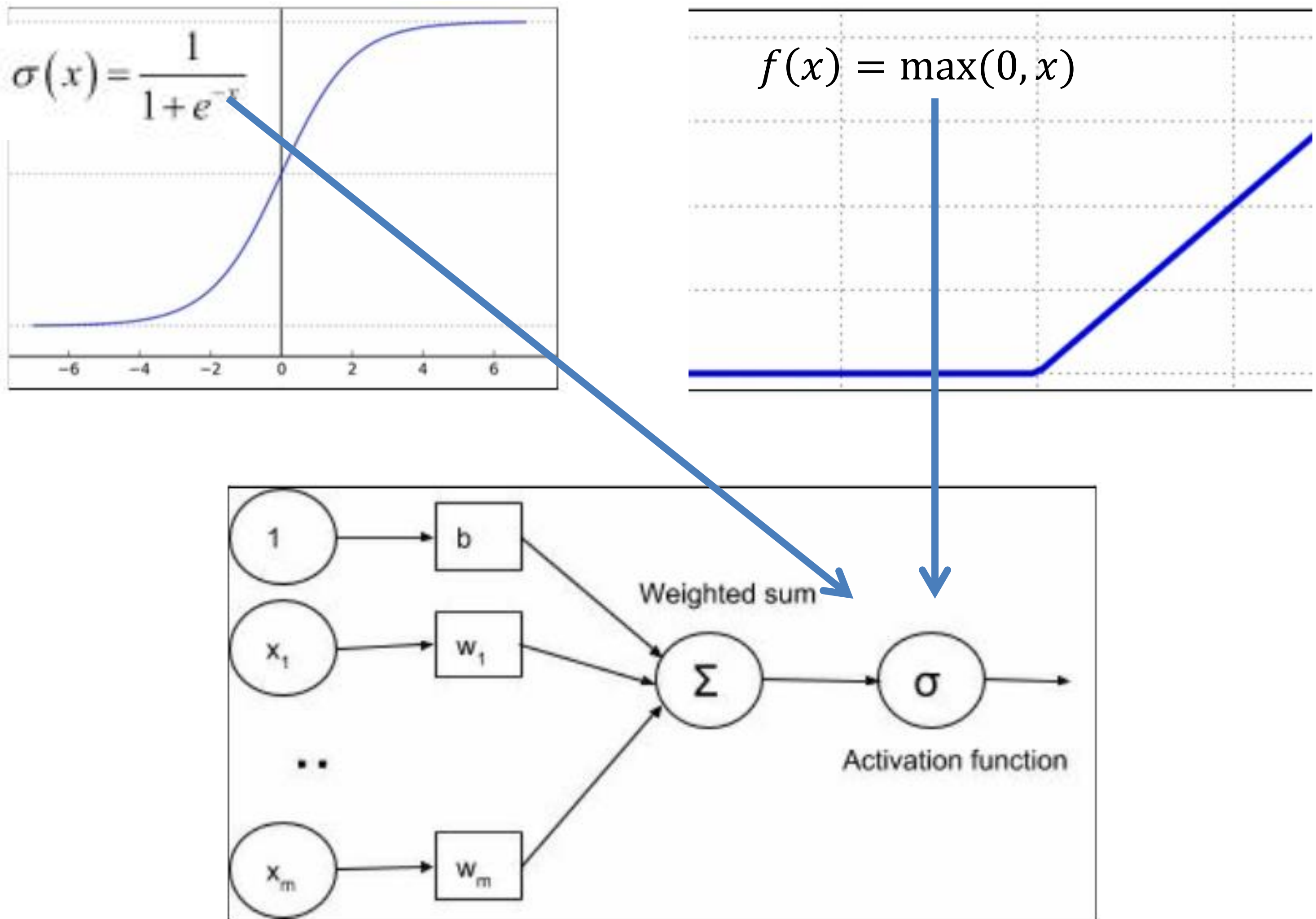
Problems in training the perceptron and a solution

- So far, we need to adjust parameters for single neuron (weight and bias)
- Ideally, providing a set of training set and let the computer adjust the weight and the bias such a way that the errors produced in the output are minimized
- A big output jump cannot progressively learn without knowing whether improving (Perceptron only returns 0 or 1, Need to use nonlinear f(x))
- Stacking layers is useless (ex. $h(x)=ax$, $y(x)=h(h(h(x)))=a^3x=bx$



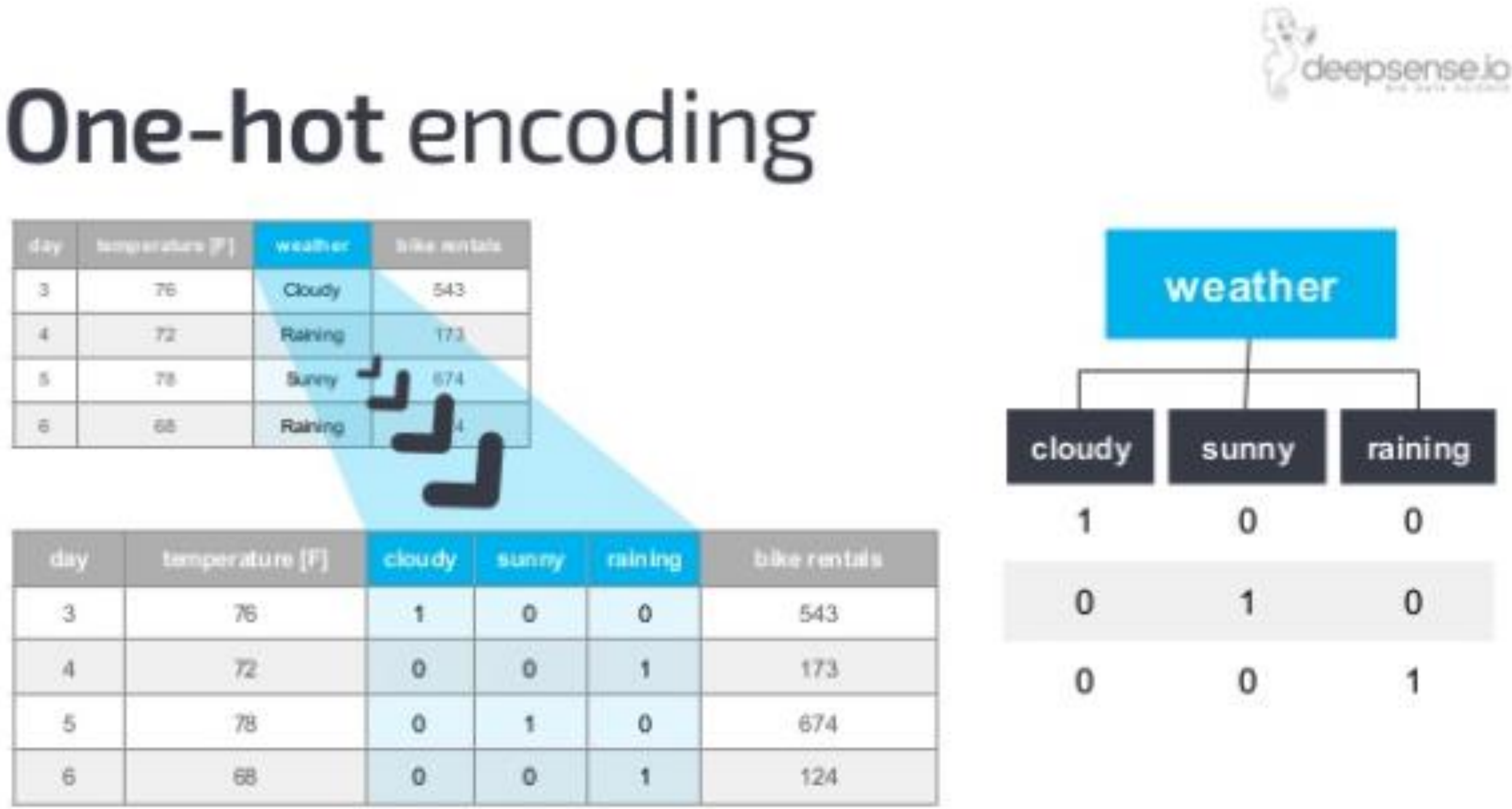
Activation F(x)
-Sigmoid and ReLU

- A neuron can use the sigmoid for computing the nonlinear function
- ReLU : Rectified linear unit



One-hot encoding

-Transform categorical into numerical variables



http://angeliot.blogspot.kr/2016/11/one-hot-encoding.html

One-hot encoding

-Transform categorical into numerical variables
- Usually in bioinformatics……

	A	B	C	D	E	F	G	H	I
1	sym	KATOIII	SKGT2	NCC59	AGS	SNU638	NUGC3	IM95	YCC3
2	A1CF	1	1	0	0	0	0	0	0
3	A2ML1	0	0	1	1	0	0	0	0
4	AACS	0	0	1	1	0	0	0	0
5	AADAC	0	0	0	0	1	0	0	0
6	AADACL4	0	1	0	0	0	0	0	0
7	AADAT	0	0	0	0	0	1	0	0
8	AAR2	0	0	1	0	0	0	1	1
9	AARS	0	0	0	0	0	0	0	1
10	AASDH	0	0	0	0	0	0	0	0
11	AATK	0	0	0	0	0	0	0	0
12	ABAT	0	0	0	0	0	0	0	0
13	ABCA10	0	0	1	0	0	0	0	0
14	ABCA12	0	0	1	1	1	0	1	0
15	ABCA13	0	0	0	1	0	0	0	0

1 = 돌연변이
0 = 돌연변이 없음.

Objective f(x)

- Optimizer that optimize parameters
- Loss f(x), Cost f(x)

MSE
Mean squared error between the predictions and the true values

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(\Upsilon - Y)^2$$

Binary cross-entropy
Binary logarithmic loss. Predicting p while the target is t
Suitable for binary labels prediction $0 < p < 1, t \in (0,1)$

$$-t\log(p) - (1-t)log(1-p)$$

Categorical cross-entropy
Multicalss logarithmic loss. If the target is t(i,j) and the prediction is p(i,j)

$$L_i = -\Sigma_j t_{i,j}\log(p_{i,j})$$

Performance

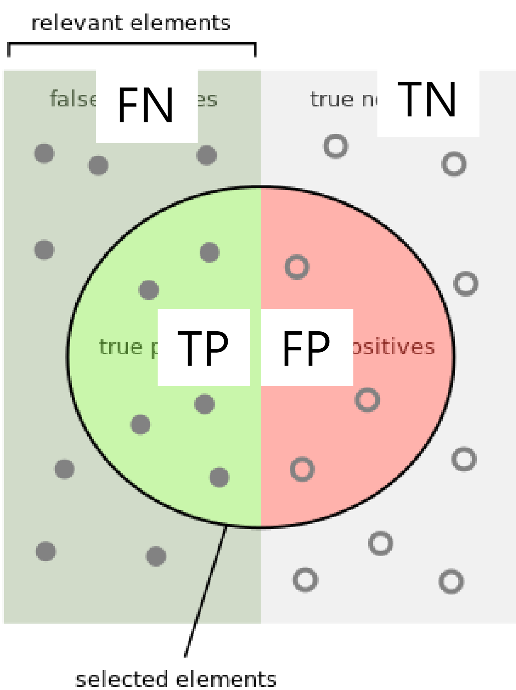
Accuracy

The proportion of correct predictions with respect to the target (Average of precision)

$$\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$$

Precision & Recall

How many selected items are relevant for multilabel classification



Key Metrics:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

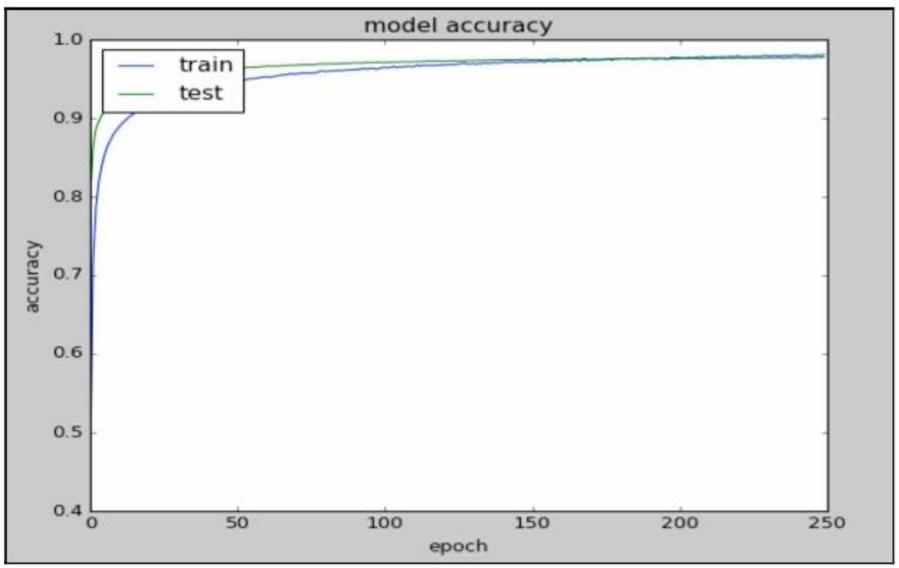
https://en.wikipedia.org/wiki/Precision_and_recall

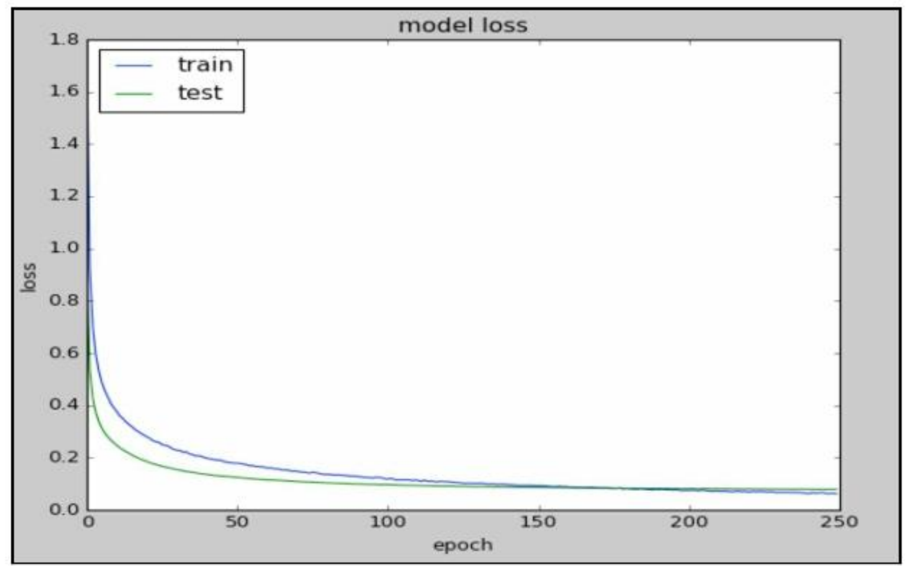
Performance

Epochs
N of times the model is exposed to the training set.

Batch_size
N of training instances observed before the optimizer performs weight update

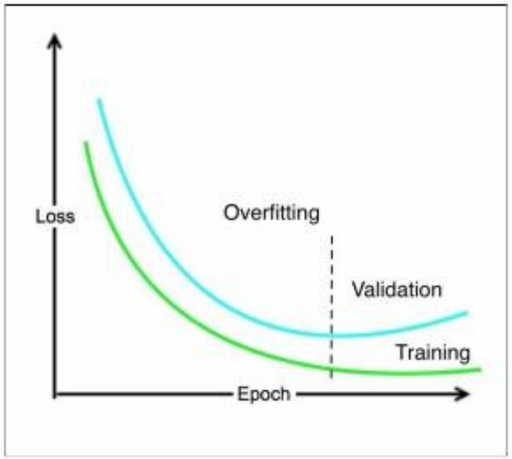
- Useful to observe how accuracy increases on training and test sets

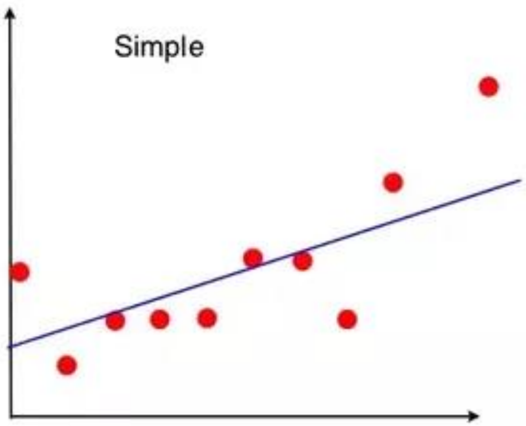


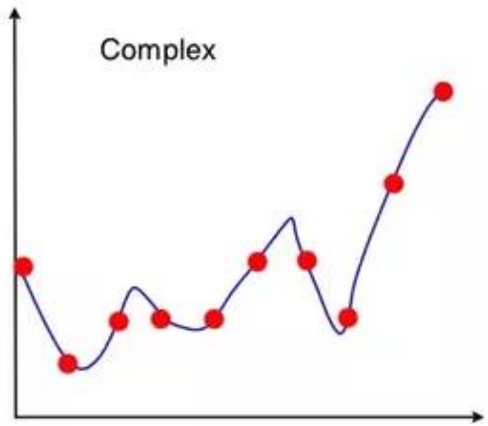


Adopting regularization for avoiding overfitting

Overfitting







- To solve overfitting problems, need to capture the complexity of a model.
- Choose the simplest model that has the minimum N of nonzero weights.

$$\min : \{loss(Training\ Data|Model)\} + \lambda * complexity(Model)$$

λ : Strigency parameter, Increasing lambda will reduce weight of fittures.

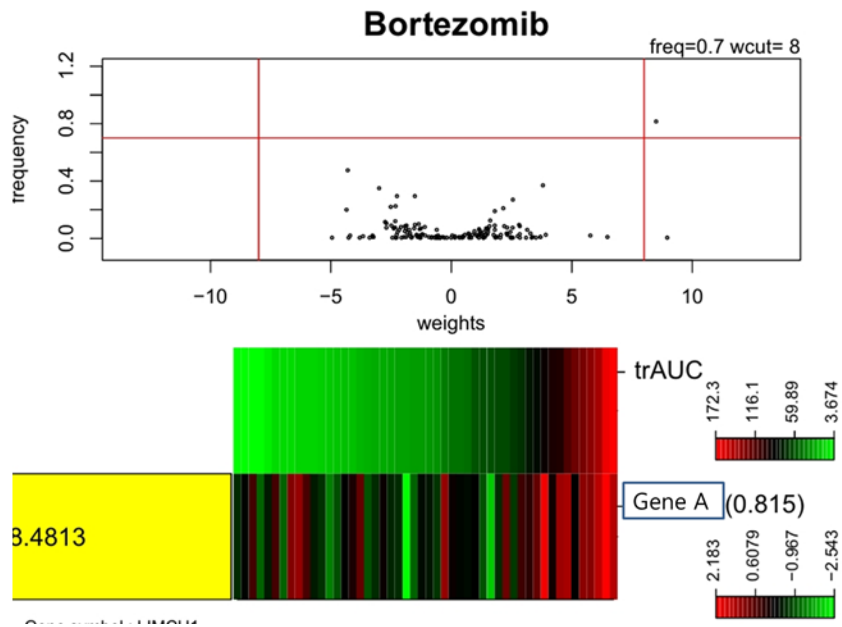
Adopting regularization for avoiding overfitting

Regularization

Lasso (L1 regularization)

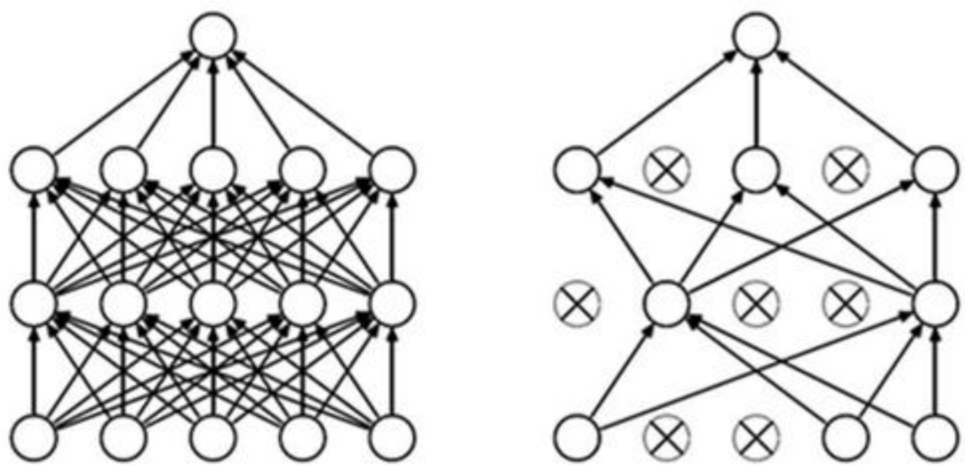
Ridge (L2 regularization)

Elastic net (Lasso+Ridge)



Dropout

Randomly training some of neuron in input or hidden layer



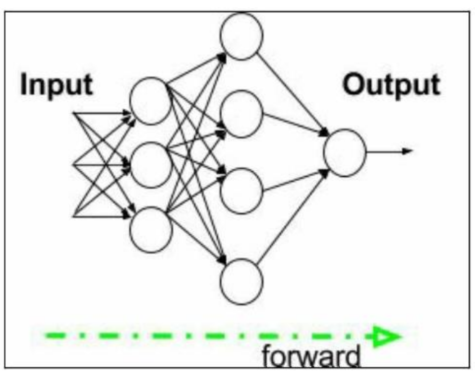
(a) Standard Neural Net

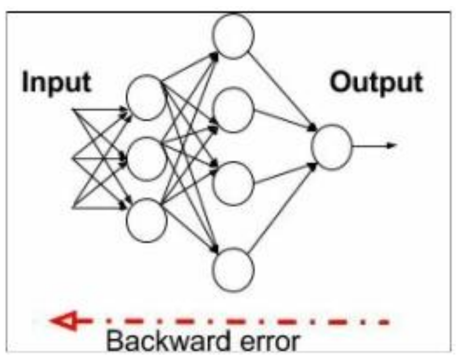
(b) After applying dropout.

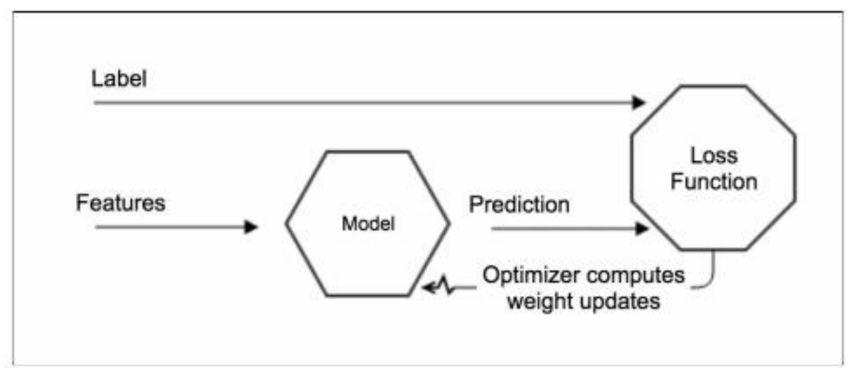
A practical overview of backpropagation

- **Multilayer perceptrons learn from training data via backpropagation**
- **A way of progressively correcting mistakes as soon as they are detected**

1. Starting all neuron with random weight
2. Activated for each input in the training set -> propagated forward
3. Calculate the error made in prediction
4. Backtracking, to propagate the error back and apply optimizer
5. Repeating the process and reduce error







끝