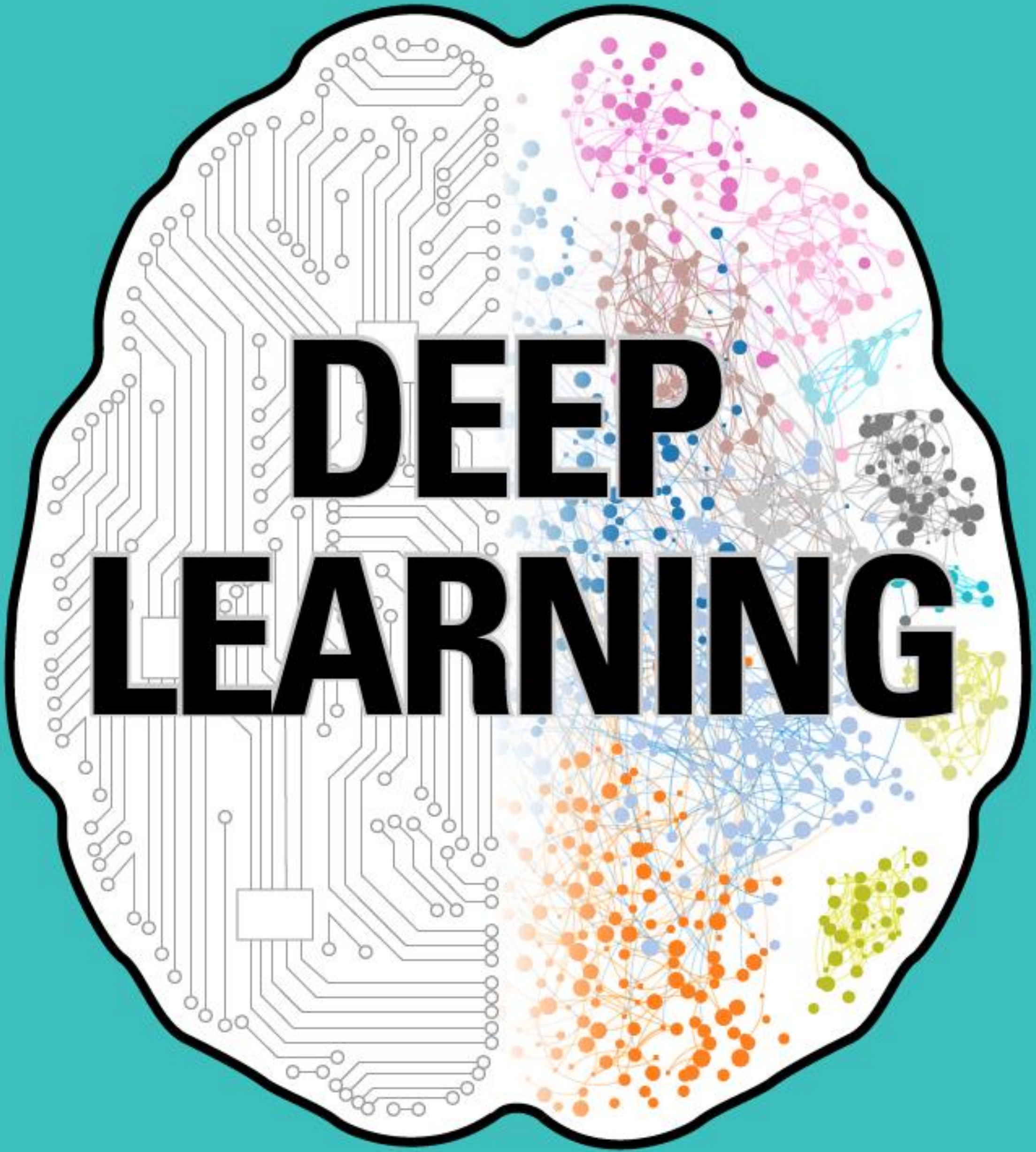
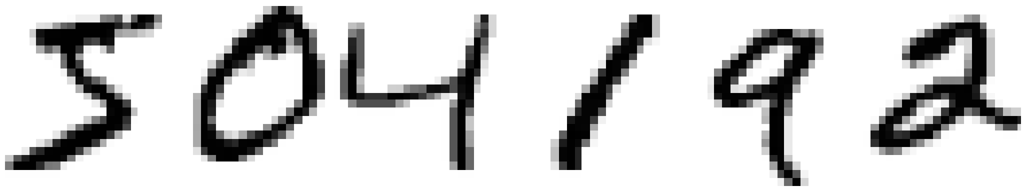



# Neural Networks & Deep Learning

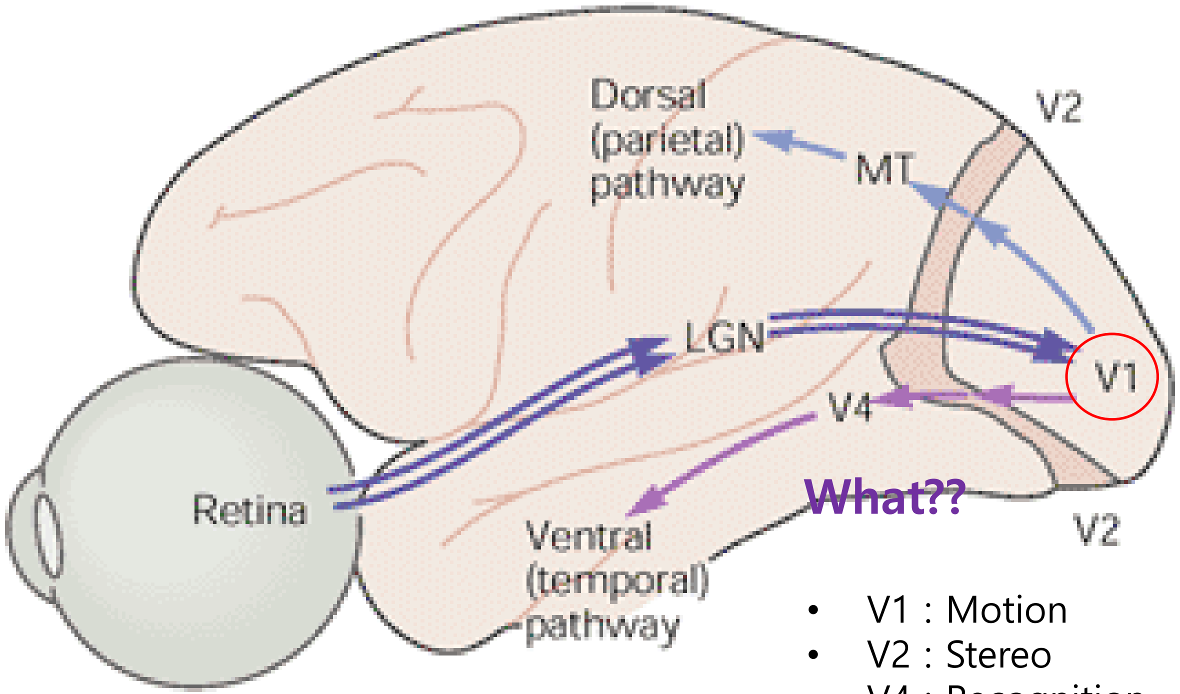


http://neuralnetworksanddeeplearning.com/chap1.html

## [ Brain ]



504192




- 140million neurons
- Tens of billions of connections between them

- V1 : Motion
- V2 : Stereo
- V4 : Recognition
- MT(V5) : Attention

# [ Computer Program ]

(1) Find common features....

"a 9 has a loop at the top, and a vertical stroke in the bottom right"

(2) Express features algorithmically

(0, 0, ..., 54, 120, ..., 0, 0)

(3) Choose decision rules

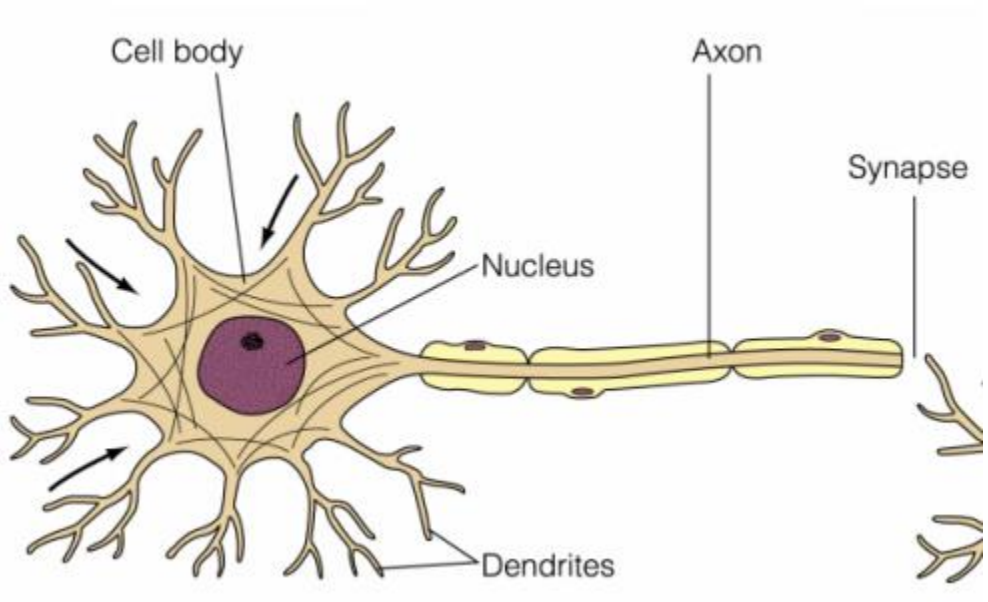

# [ Neural Networks ]

Examples or Training Data

NN uses the examples to automatically infer features and rules for recognizing handwritten digits

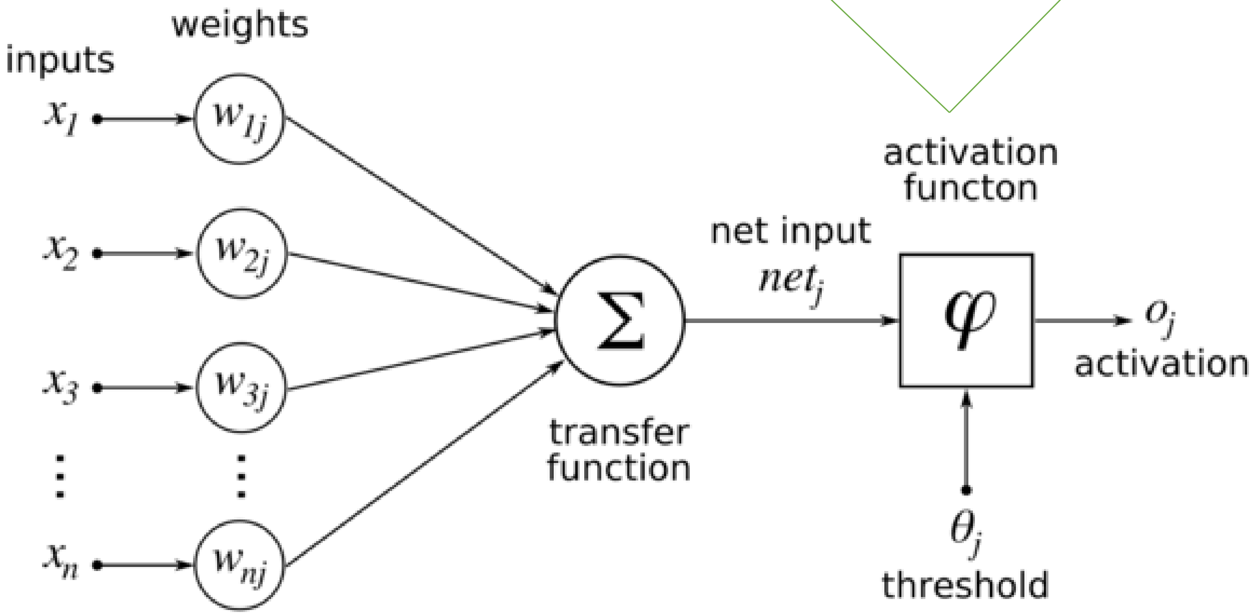
# Artificial Neurons

## Neuron




## Artificial Neuron

Perceptron Sigmoid neuron

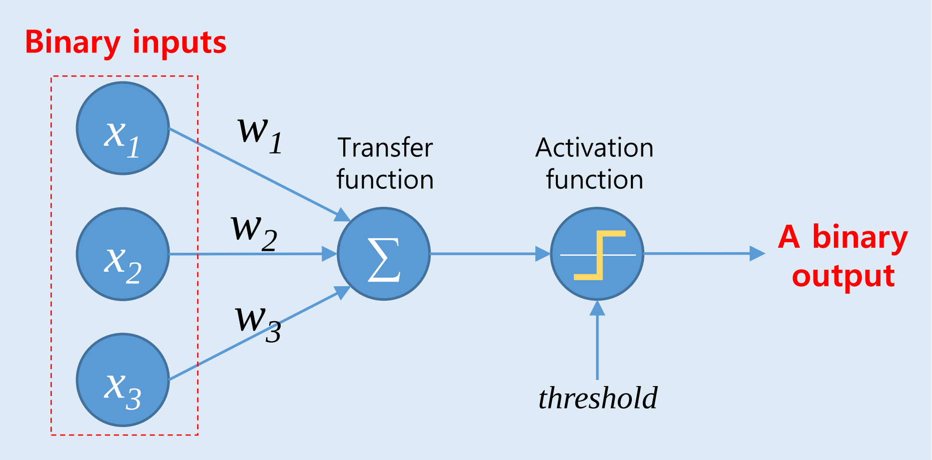


# Perceptrons

$x_1$: Is the weather good?

$x_2$: Does your boyfriend or girlfriend want to accompany you?

$x_3$: Is the festival near public transit?




$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

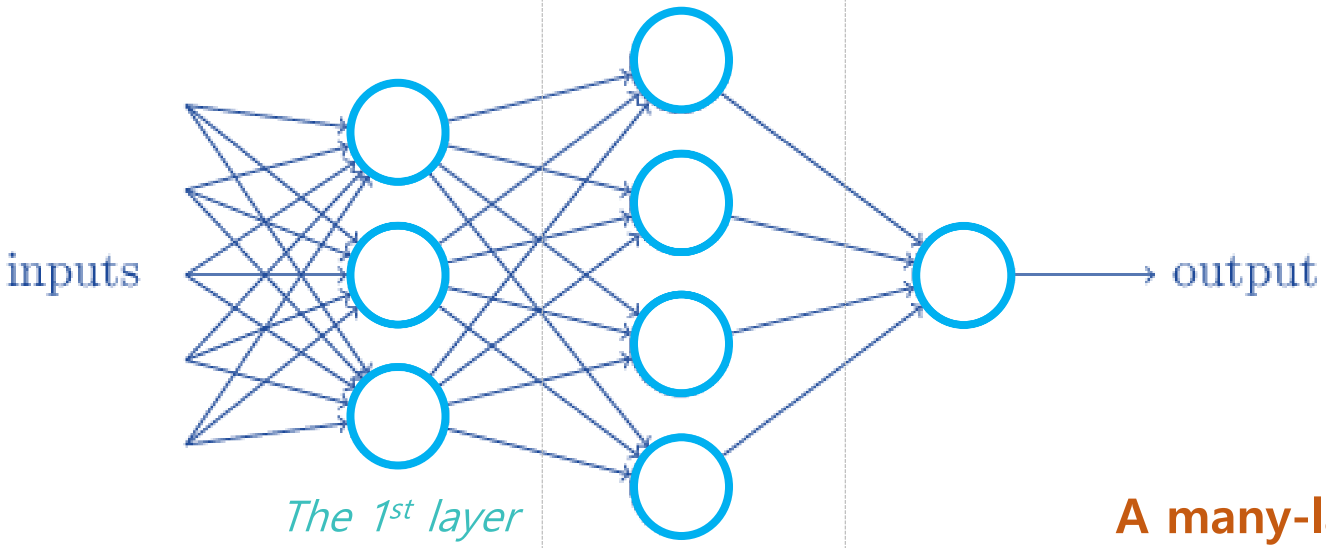
1 : Join the festival
0 : Not join the festival

| | |
|---|---|
| $[w_1, w_2, w_3, threshold]$ = [ 6, 2, 2, 5] | You cares only about the weather !!! |
| $[w_1, w_2, w_3, threshold]$ = [ 3, 2, 2, 5] | All conditions must be satisfied !!! |

- **By varying the weights and the threshold, we can get different models of decision-making.**
- **It seems that a perceptron can weigh up different kinds of evidence in order to make a decision.**

# Perceptrons

How about this??




*The 1st layer*

simple decisions are made

*The 2nd layer*

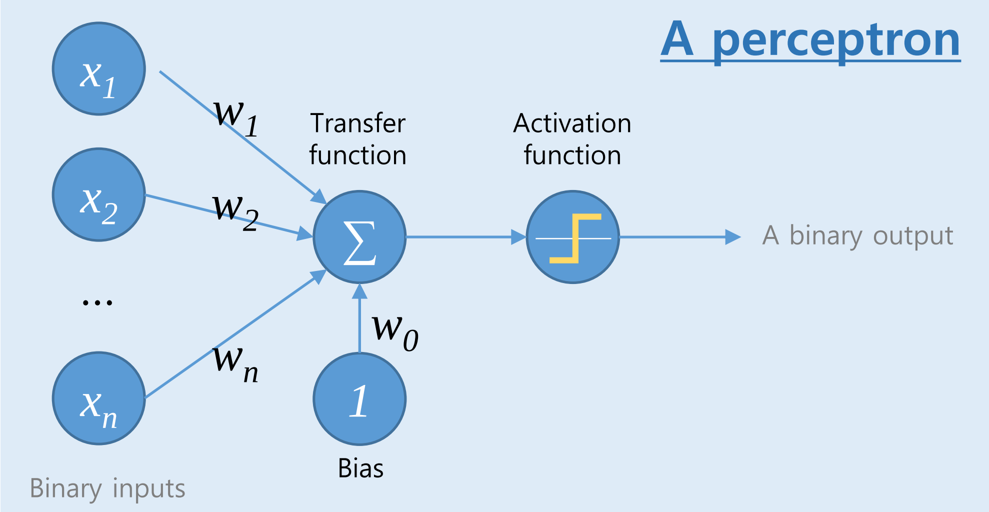
more complex and more complex level decisions are made

*The 3rd layer*

Most complex decision is made

**A many-layer network of perceptrons can engage in sophisticated decision making.**

# Perceptrons




$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

dot-product

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$w=[w_1, w_2, \ldots, w_n]'$
$x=[x_1, x_2, \ldots, x_n]'$
$b = w_0$*-threshold*

*b* is called a perseptron's bias.
The bias is a measure of how easy it is to get the perceptron to *fire* (that's why thresholding is called activiation function in a figure)

# Perceptrons

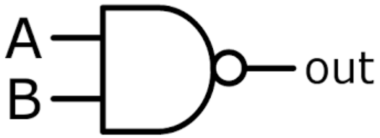
A perceptron can compute the elementary logical functions such as AND, OR, and NAND

## 2bits NAND gate

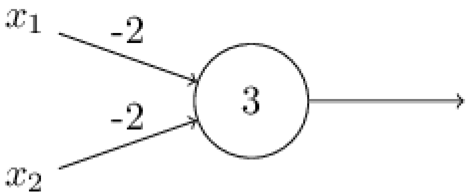
❖ Truth table

| $x_1$ | $x_2$ | *output* |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

❖ Symbol




❖ A perceptron



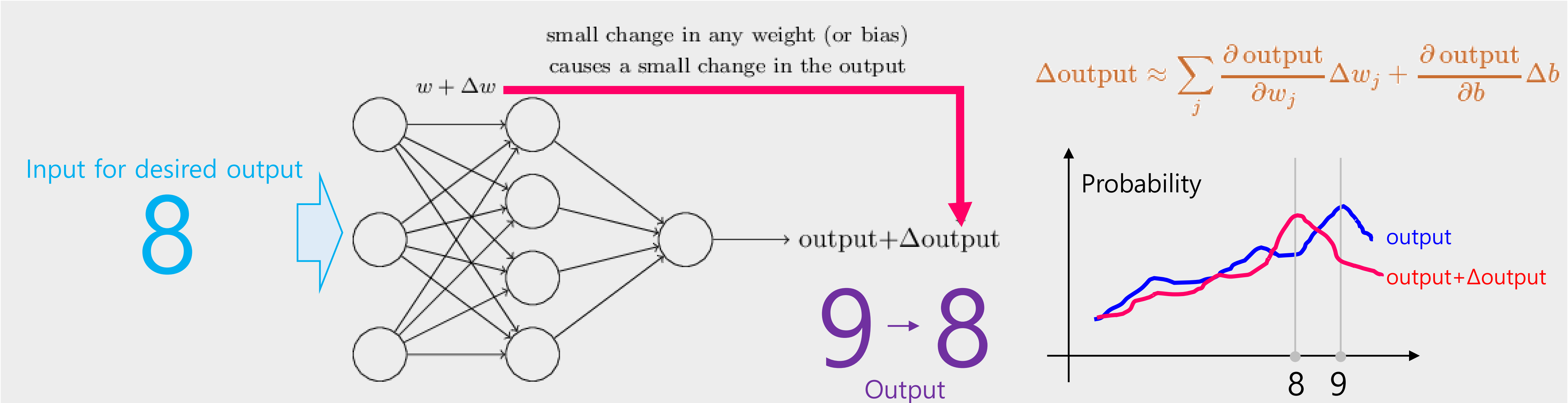

**We can use networks of perceptrons to compute *any* logical function at all.**

**The reason is that the NAND gate is <u>universal</u> for computation, that is, we can build any computation up out of NAND gates.**

# Sigmoid neurons

What for did sigmoid neurons appear?

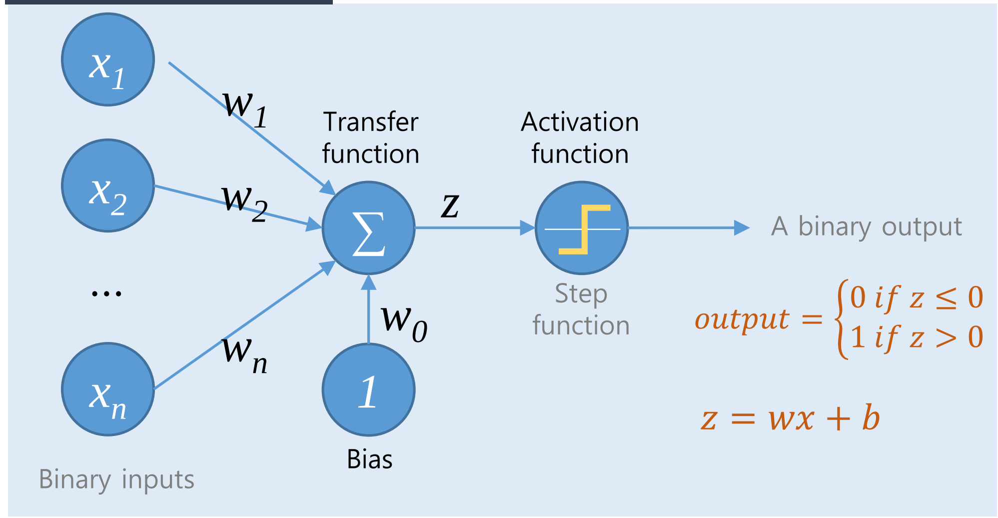
- Learning Algorithms decide weights and biases for each neuron.
- Learning algorithm changes weights/biases in order output to be more likely desired output.
- For the purpose, an assumption that a small change in a weight/bias causes only a small change in output is required.
- But this is not true for perceptron. (Flip between 0 ↔ 1 is possible)

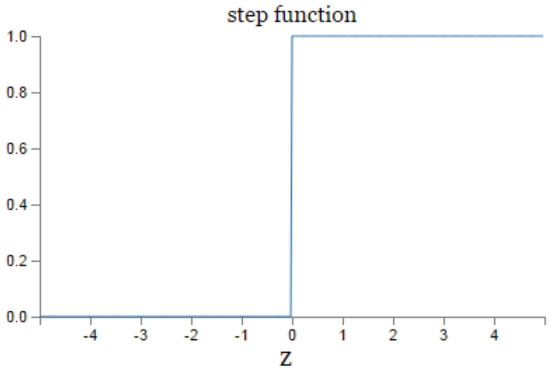


# Sigmoid neurons

A sigmoid neuron is modified version of a perceptron so that small changes in their weights and bias cause only a small change in their output.

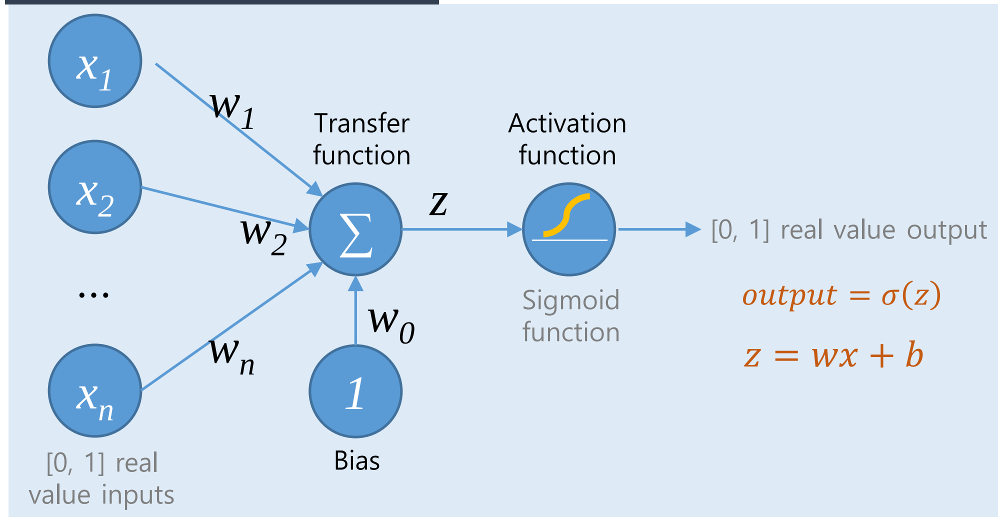
## A perceptron

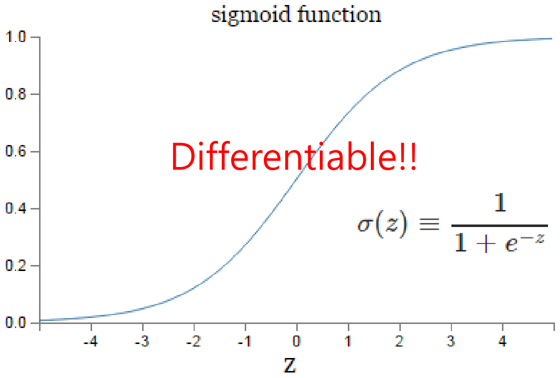



Activation function

## A sigmoid neuron

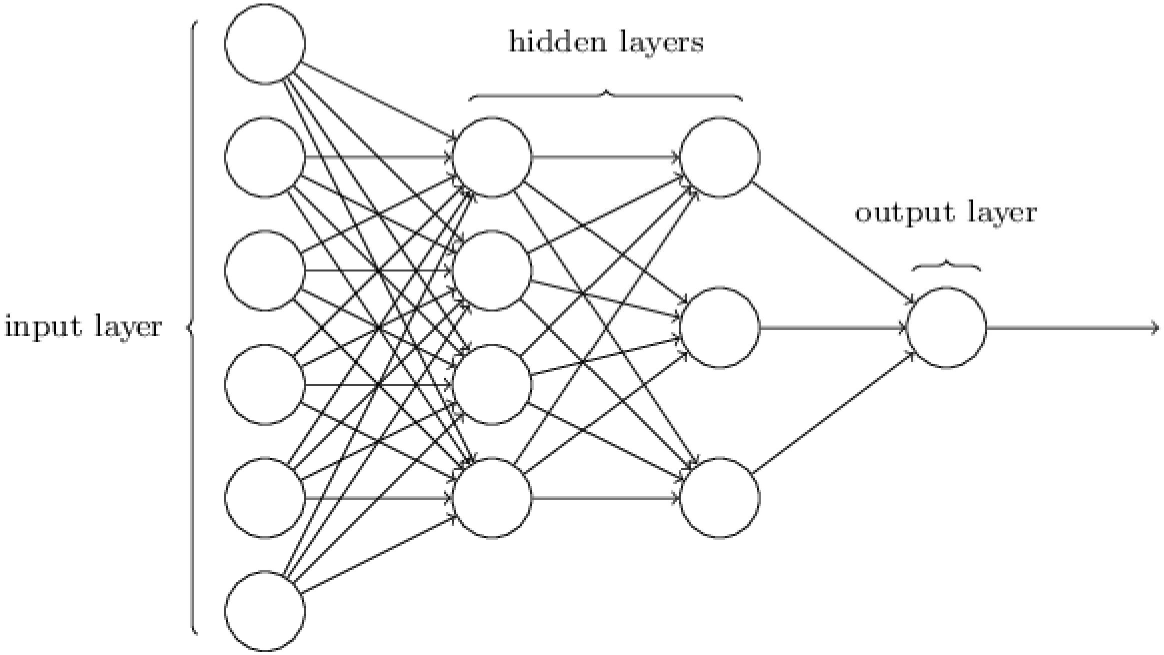



Activation function

# The architecture of neural networks

## Naming




The design of input & output layers is straightforward. In handwritten digit case, input is image pixels, and output is '9' or not.

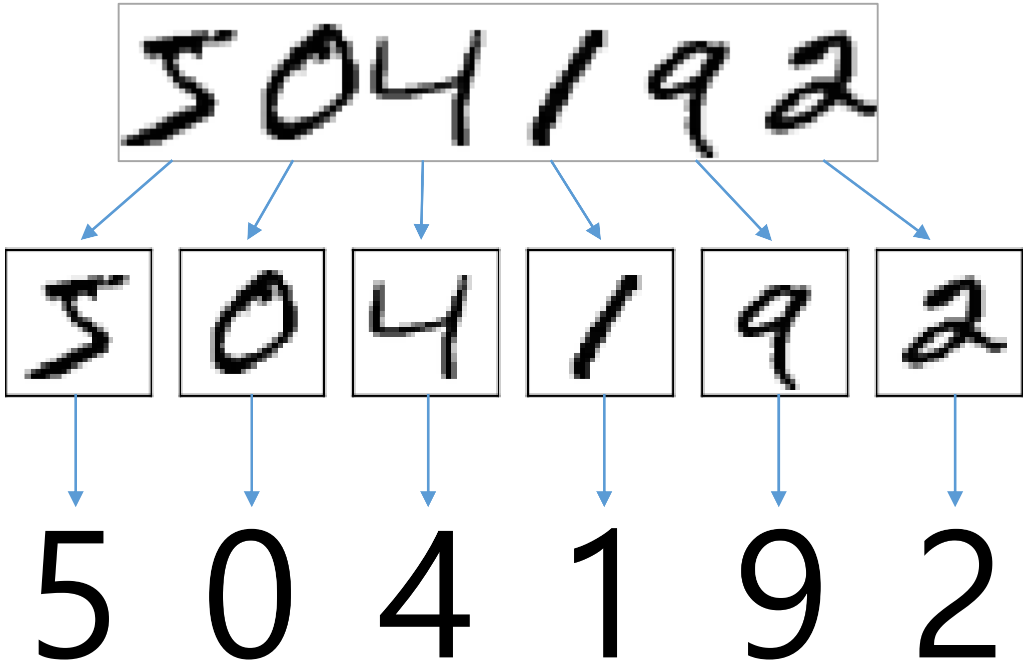
- Input : input image
- Output : 'input image is 9' or 'input image is not 9'

Instead, neural networks researchers have developed many design heuristics for the hidden layers, which help people get the behavior they want out of their nets. (we will see later…)

Multiple layer networks are called multilayer perceptrons (MLPs), despite being made up of sigmoid neurons, not perceptrons.

# A simple network to classify handwritten digits

## Define a problem




*Given an image,*

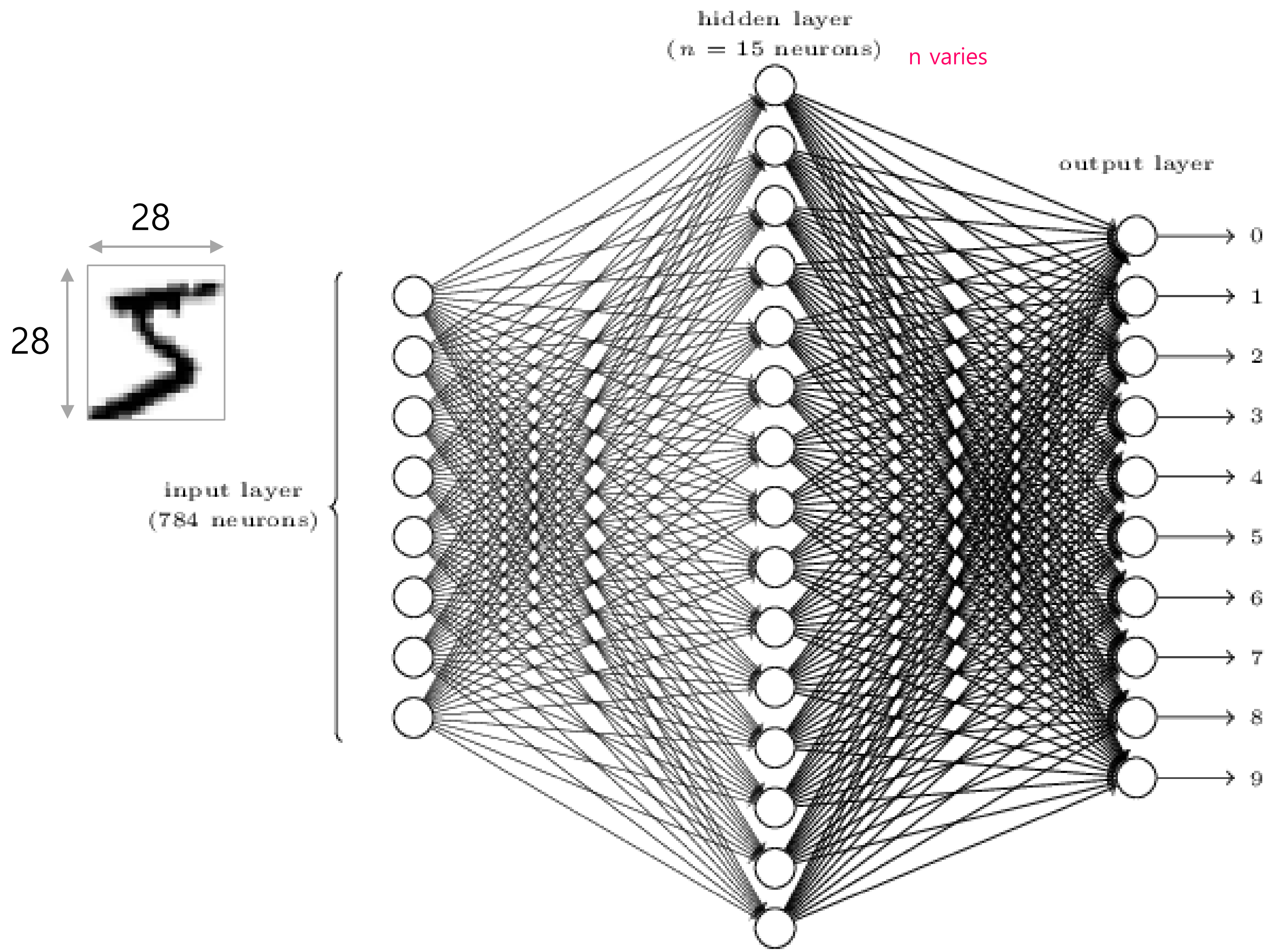
*Step 1 : separate images so that each sub image contains one digit.*
(Segmentation problem)

*Step 2 : classify each image into a digit*
(Classification problem)

We focus on this problem!!

# A simple network to classify handwritten digits

## Proposed network




- This network can be considered as follows.

- Consider output layer as the 2nd hidden layer and add one output layer of choosing a output of the maximum activation function value

An alternative.

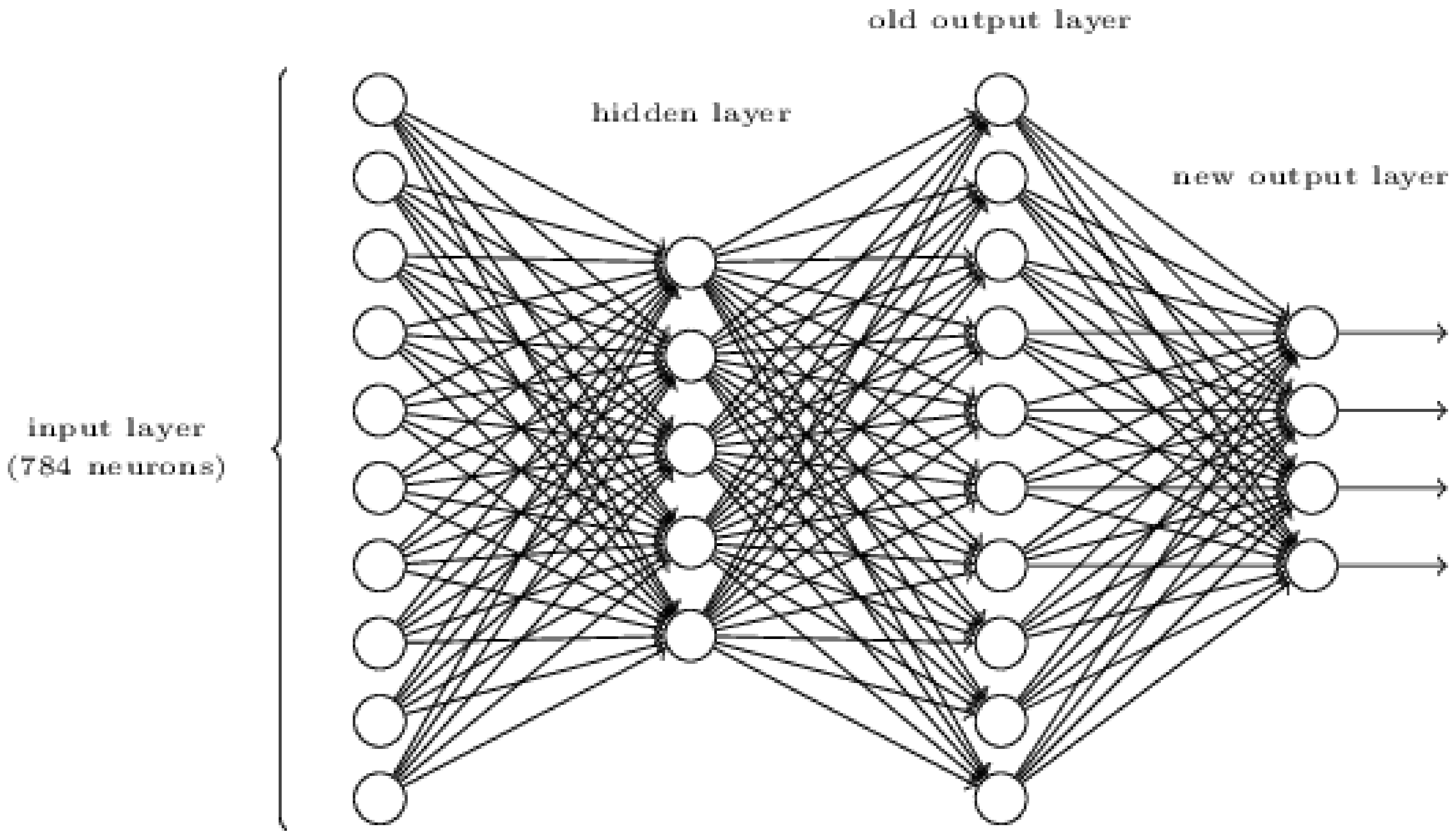
- Output layer is changed by 4 neurons where each neuron gives binary value. $2^4 = 16$ can cover the representation of from 0 to 9

- But we tested 2 models, it turns out that a model with output layer of 10 neurons performs better

# A simple network to classify handwritten digits

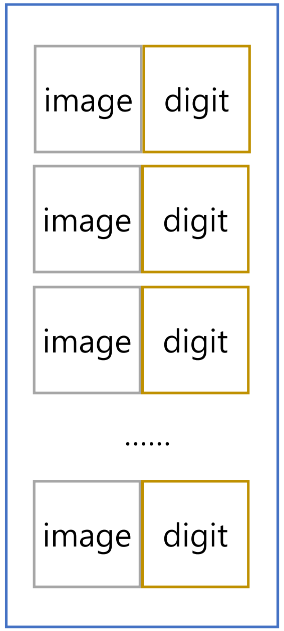
**But we can design like this…**

Output layer design



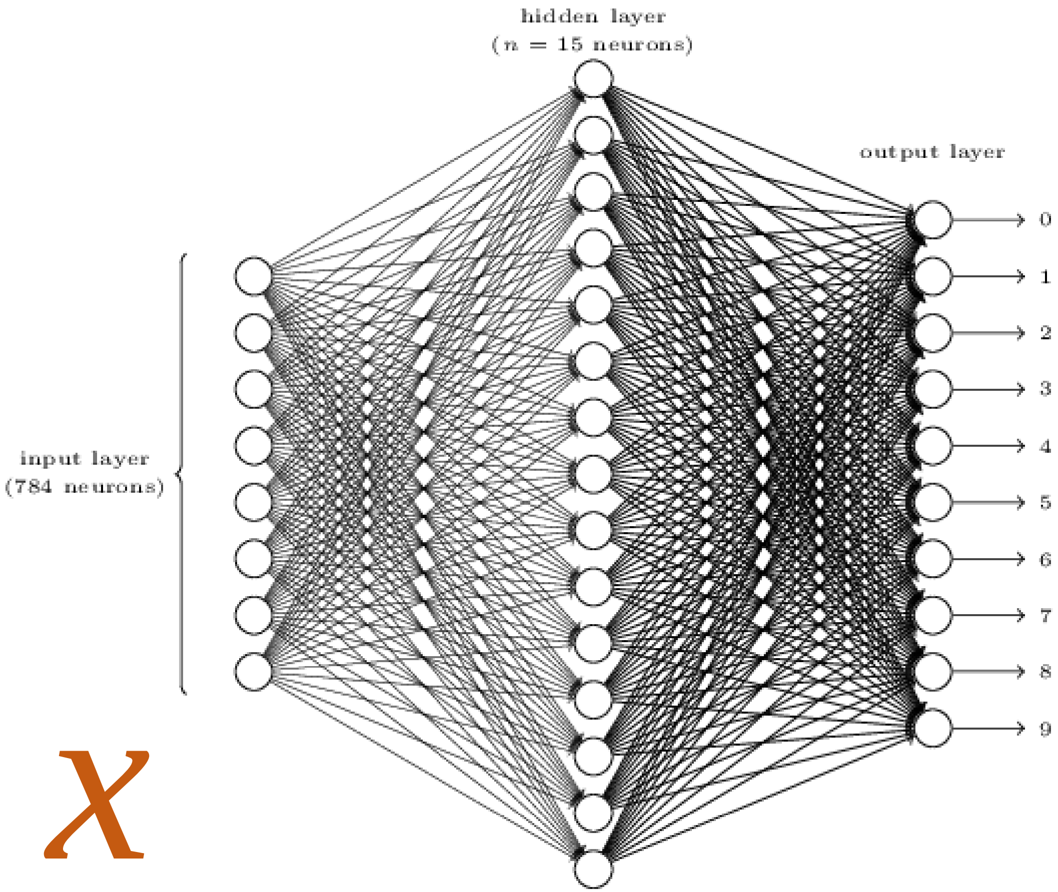
# Learning with gradient descent

## Notation




Dataset for training
= training DB
= a collection of examples

How to train a NN with a dataset?
: how to finds weights and biases?




$x$

Input : a vector of a gray-image
Where each element ranges 0~1.
(normalized by 255)

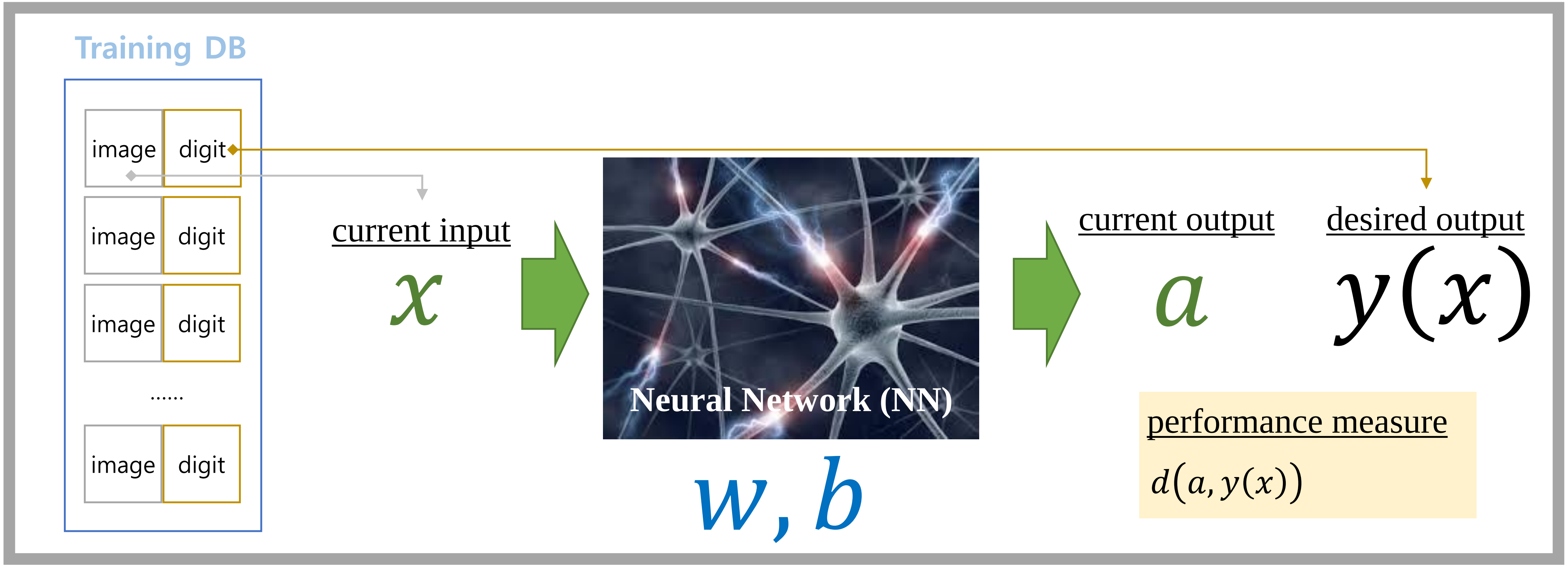
$w$ : all weight, $b$ : all biases

$y(x)$

Desired Output : a vector in 10-dim.
If answer is ‘6’, $y(x) = (0,0,0,0,0,0,1,0,0,0)^T$

# Learning with gradient descent

## Learning/Training Process




NN parameters : weights & biases

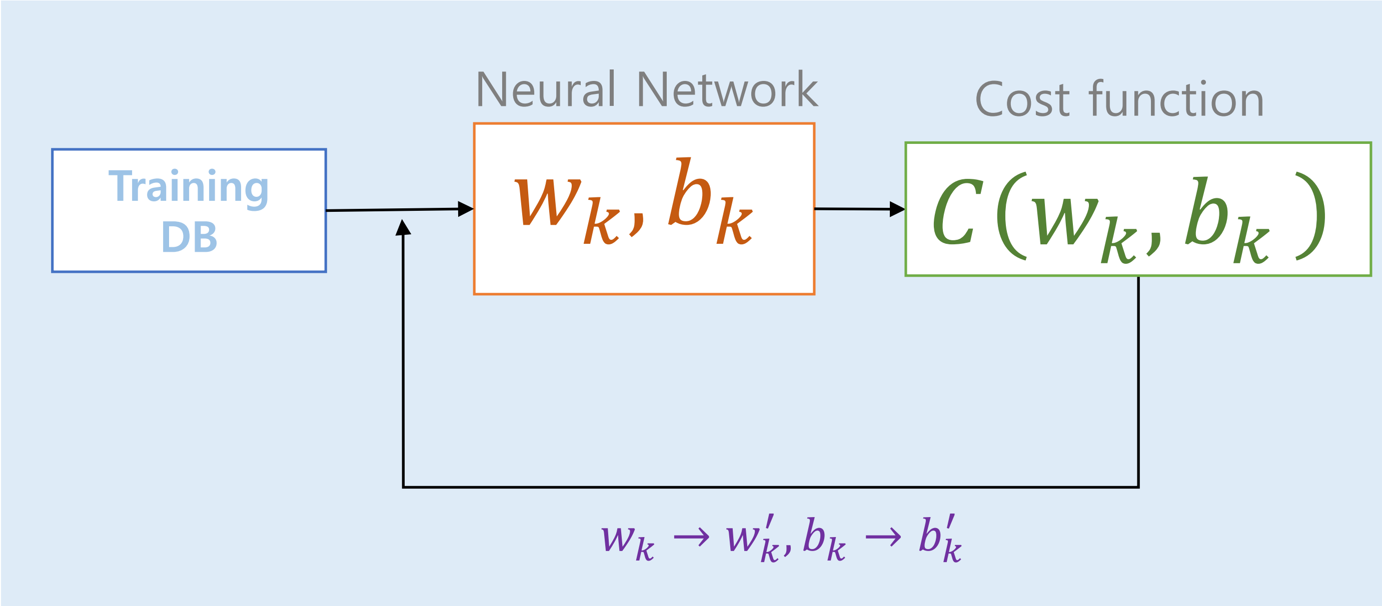
cost/loss/objective function

$$C(w,b) = \underbrace{\sum d(a, y(x))}_{\text{sum for all examples}}$$

If d(.) indicates the difference between ‘a’ and ‘y(x)’ with non-negative value, learning/training aims at C(w,b) = 0 or minimize C(w,b)

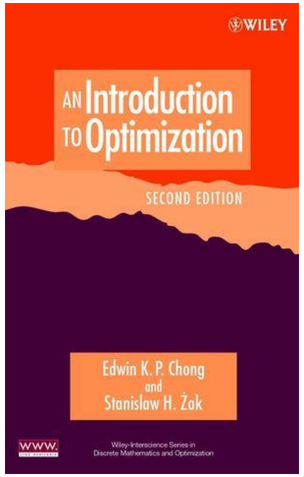
# Learning with gradient descent

## Optimization Problem




Cost function must be a smooth function of the weights and biases for the ease of mathematical analysis.

**HOW????**
**→ Optimization problem**
**→ Find a set of parameters minimizing or maximizing a function**
**→ a huge research area…**



# Learning with gradient descent

## Gradient descent

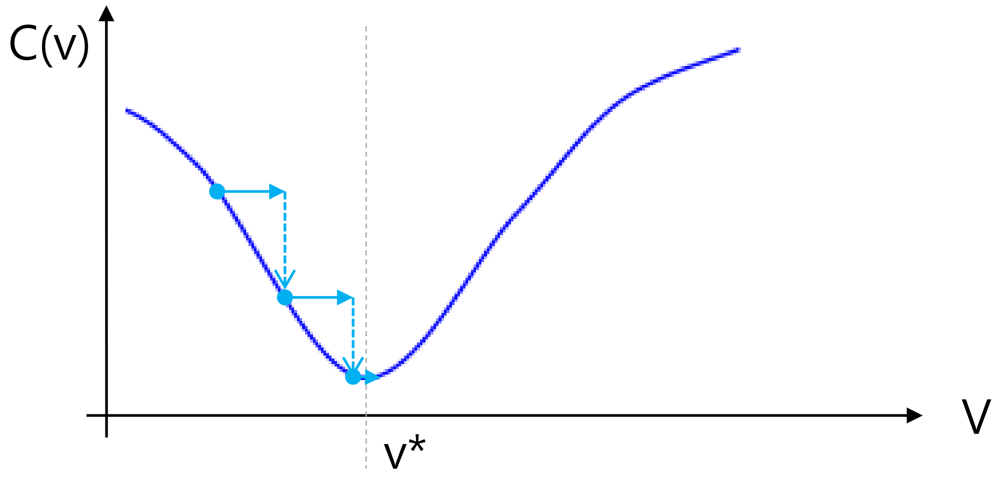
**An optimization problem**

$$v^* = \min C(v)$$
$$\text{for } v \in V$$

- $v^*$ : a global minimizer
- $V$ : search space

| Questions | Strategies |
|---|---|
| How to update v → v + Δv | Only if C(v+Δv) < C(v) |
| When we stop to search?? | If C(v+Δv) == C(v) |

We could find a global minimizer via mathematical analysis for the simple form of C(v). However, C(v) in NN or deep NN is almost impossible.
For this reason, optimization theory is a good solution.



# Learning with gradient descent

## Gradient descent

$$\text{Find } \Delta v \text{ so that } \Delta C = C(v+\Delta v) - C(v) < 0$$

Taylor Expansion : A function can be expressed with a finite number of terms of Taylor Series

1. $C(v+\Delta v) = C(v) + \nabla C \cdot \Delta v$ + second derivative + third derivative + …

The more terms are used, the wider range can be expressed with little error

2. $C(v+\Delta v) = C(v) + \nabla C \cdot \Delta v$ very locally covering expression
→ GD requires the more iteration…

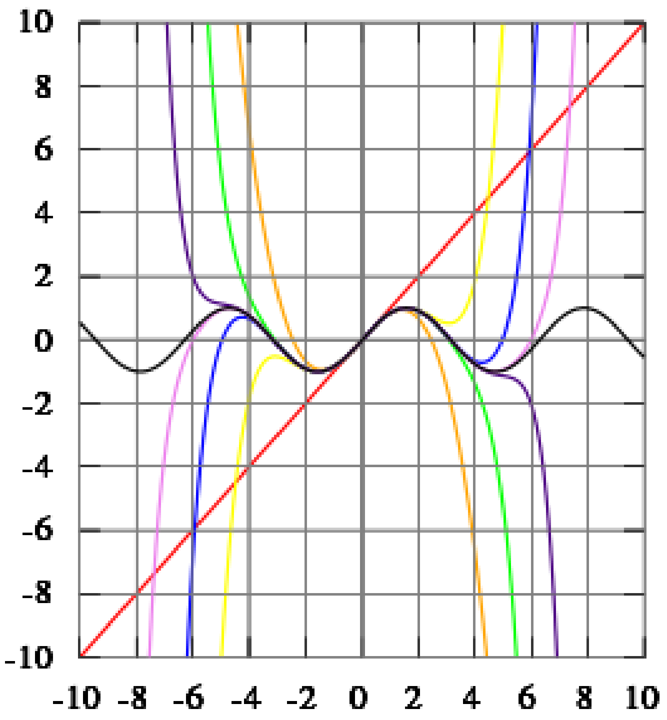
3. $\Delta C = C(v+\Delta v) - C(v) = \nabla C \cdot \Delta v$

Where $\nabla C$ is gradient of C and indicates the steepest increasing direction of C

4. If $\Delta v = -\eta \nabla C$, then $\Delta C = -\eta \|\nabla C\|^2 < 0$

Where $\eta > 0$, and called learning rate

As the degree of the Taylor polynomial rises, it approaches the correct function. This image shows sin(*x*) and its Taylor approximations, polynomials of degree 1, 3, 5, 7, 9, 11 and 13.



# Learning with gradient descent

## A problem of gradient descent in a NN

Training DB

Neural Network

$w_k, b_k$

Cost function

$C(w_k, b_k)$

$C(w_k, b_k) = \sum C_x(w_k, b_k)$ → The value of cost function (training error) is sum of error for each example 'x'

$\nabla C(w_k, b_k) = \sum \nabla C_x(w_k, b_k)$ → Gradient is also sum of gradient for each example

$$w_k \rightarrow w_k' = w_k - \eta \nabla_{w_k} C(w_k, b_k)$$
$$b_k \rightarrow b_k' = b_k - \eta \nabla_{b_k} C(w_k, b_k)$$

When the number of training inputs is very large this can take a long time, and learning thus occurs slowly!!!

**<u>Stochastic Gradient Descent</u> (SGD)**

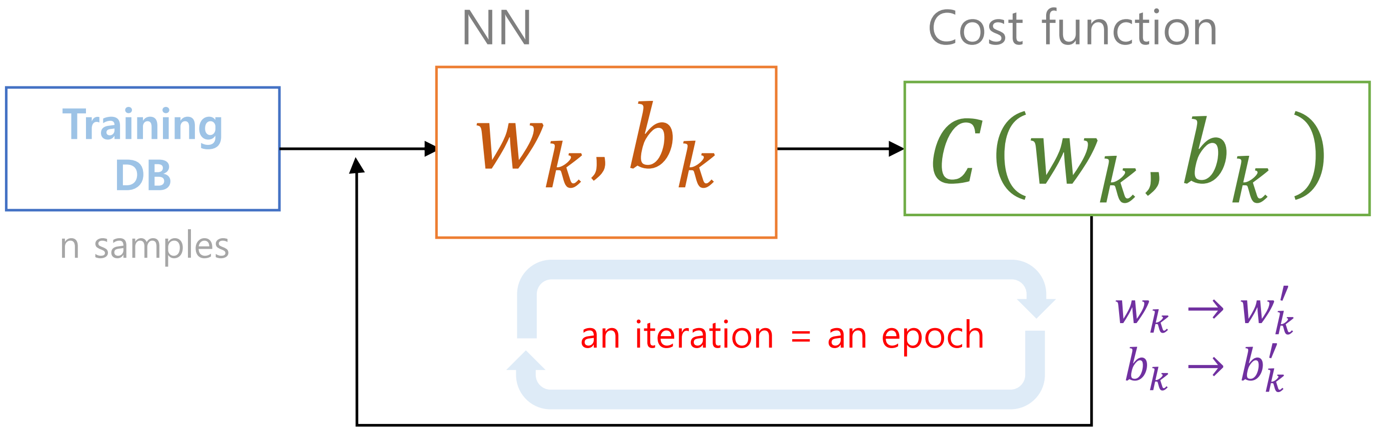
- Variance caused by random sampling is reduced by normalization.
- Learning rate compensates normalization effect.

**∇C is computed not for all samples but for a small sample of randomly chosen training inputs**

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

# Learning with gradient descent

## Summary




**Stochastic Gradient Descent**

$$w_k \to w'_k = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

$$\nabla C \approx \frac{1}{m}\sum_{j=1}^{m} \nabla C_{X_j}$$

m<n samples

- A set of samples used in SGD is called 'mini-batch'
- 1/m is for scaling of the cost function, especially useful when training DB changes in real-time
- If m==1, it is called online, or incremental learning

[ Remind ]

- In a learning process, gradient of cost function is crucial element.
- That's why sigmoid neurons are more preferred to perceptrons.

# Nice References for Beginner

https://www.facebook.com/groups/TensorFlowKR/permalink/490430184631378/