#2. Introduction to Deep Learning

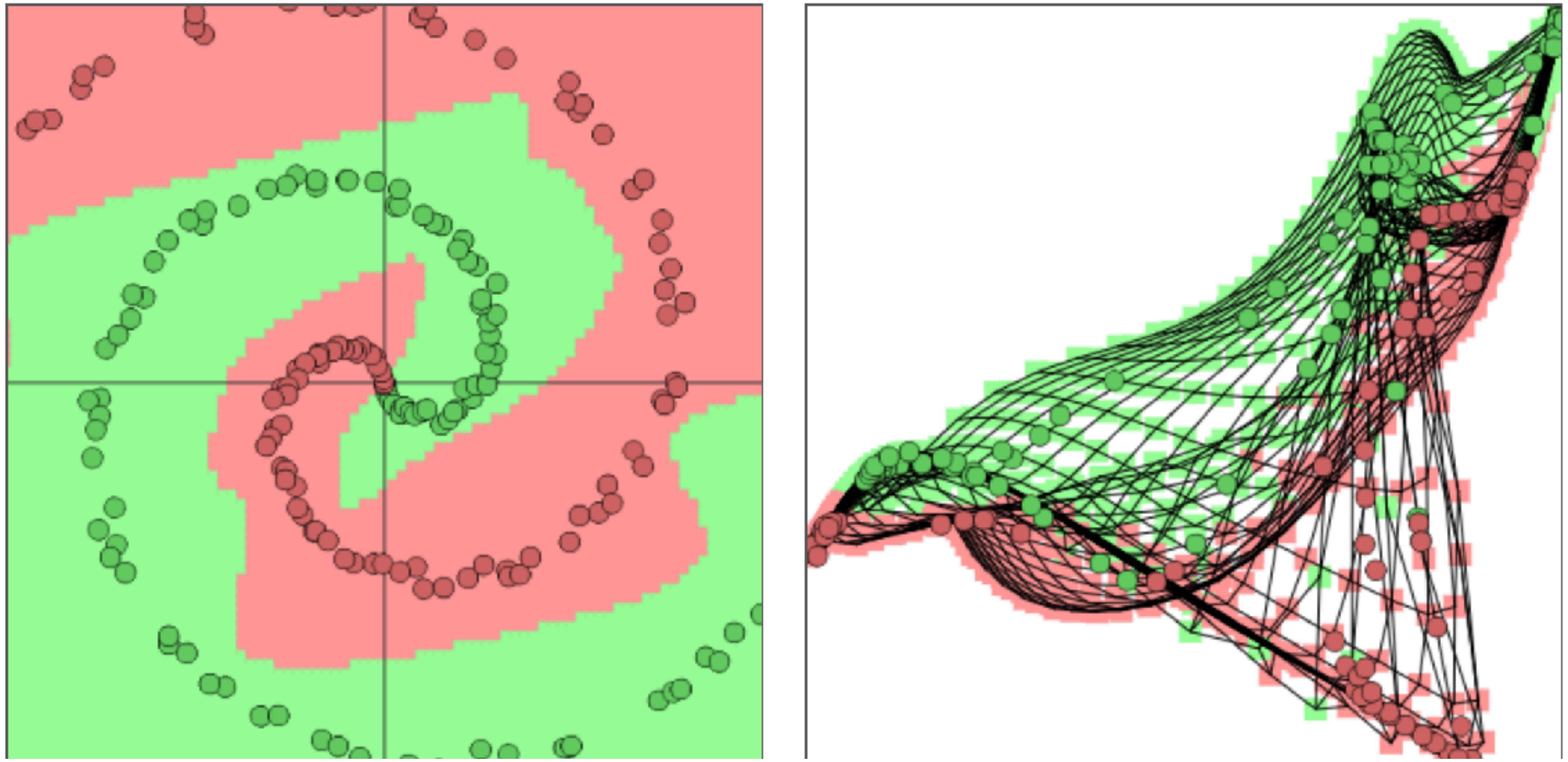
ICEC 2017 - Deep Learning Short Course

Kim Jin Ho

Deep Learning?

> **Deep learning** is the application of artificial neural networks(ANNs) to learning tasks that contain more than one hidden layer. - Wikipedia

> Among the various ways of learning representations, this paper focused on **DEEP LEARNING** methods : those that are formed by the composition of multiple non-linear transformations, with the goal of yielding more abstract - and ultimately more useful - representation.
>
> - LeCun, Y. et. al. Deep learning. *Nature*, *521*(7553), 436-444 (2015).

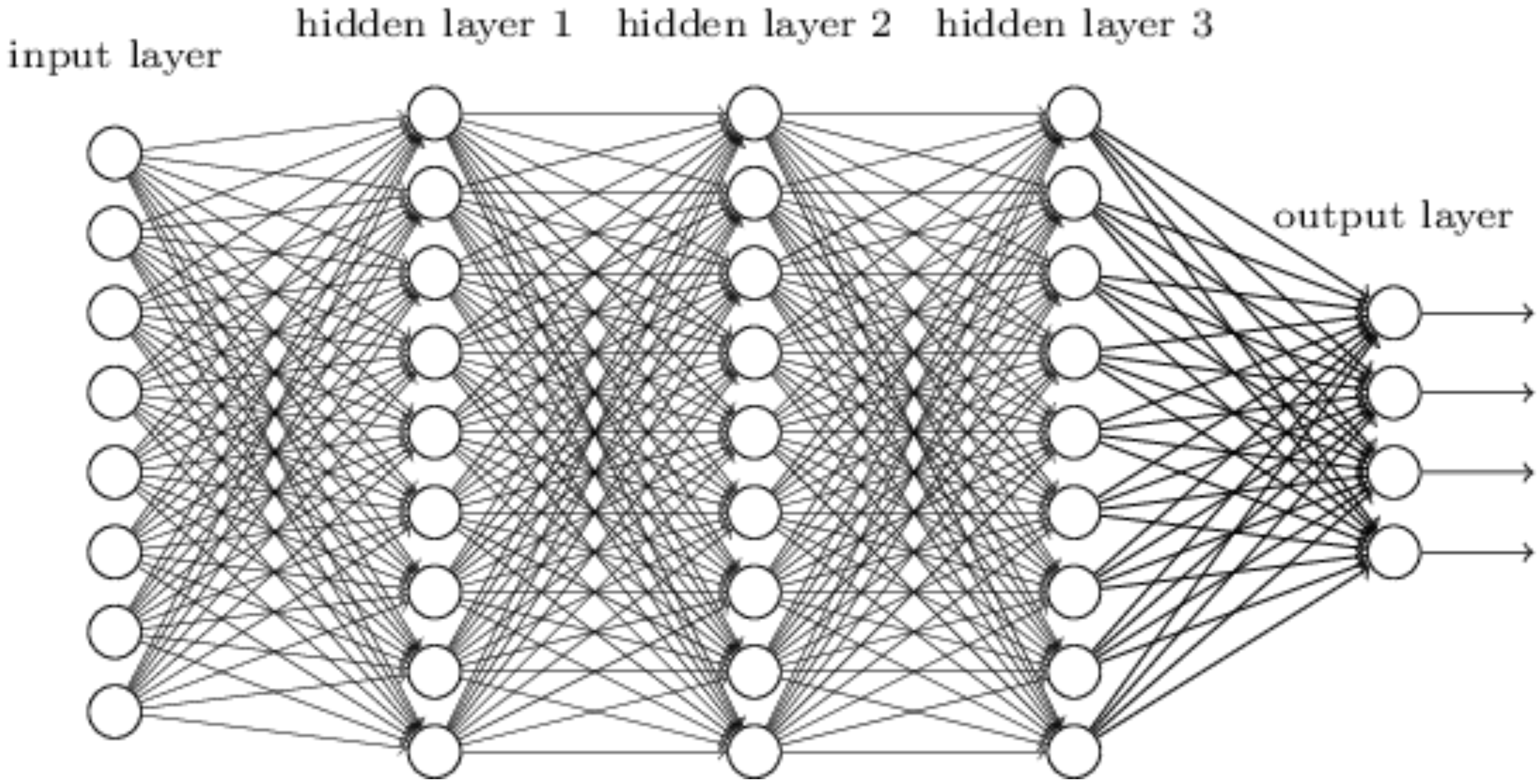


- 딥러닝은 여러 비선형 변환기법의 조합을 통해 높은 수준의 추상화를 시도하는 기계학습 알고리즘의 집합.
    - 추상화 (Abstraction) : 다량의 데이터나 복잡한 자료들 속에서 핵심적인 내용 또는 기능을 요약하는 작업.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

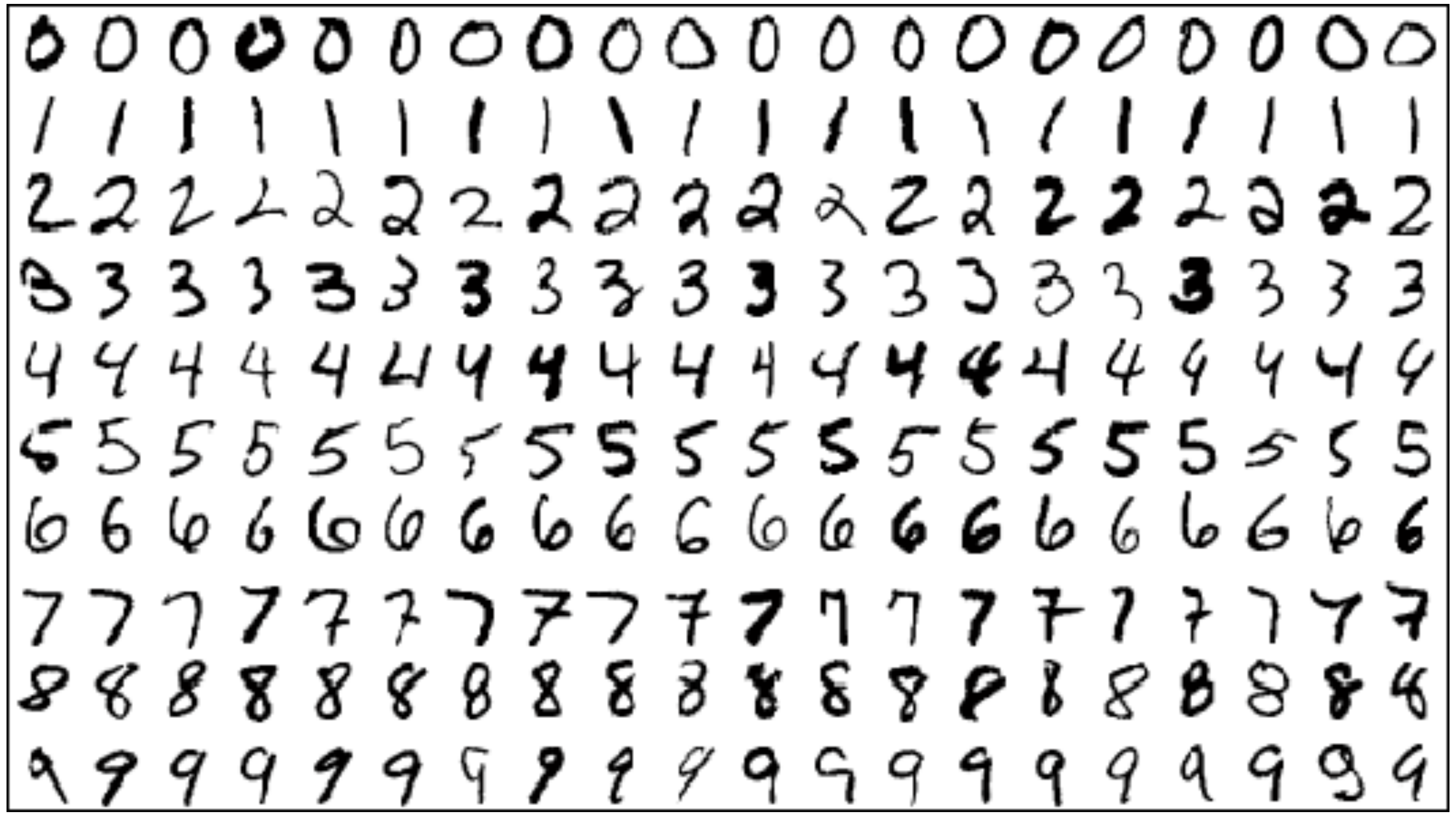
Deep Learning!

- 비선형 조합기법을 활용한 추상화를 통해 데이터를 잘 분류해 내는 기계학습
  - Multi Layer Perceptron with Non-Linear Activation Function



MNIST

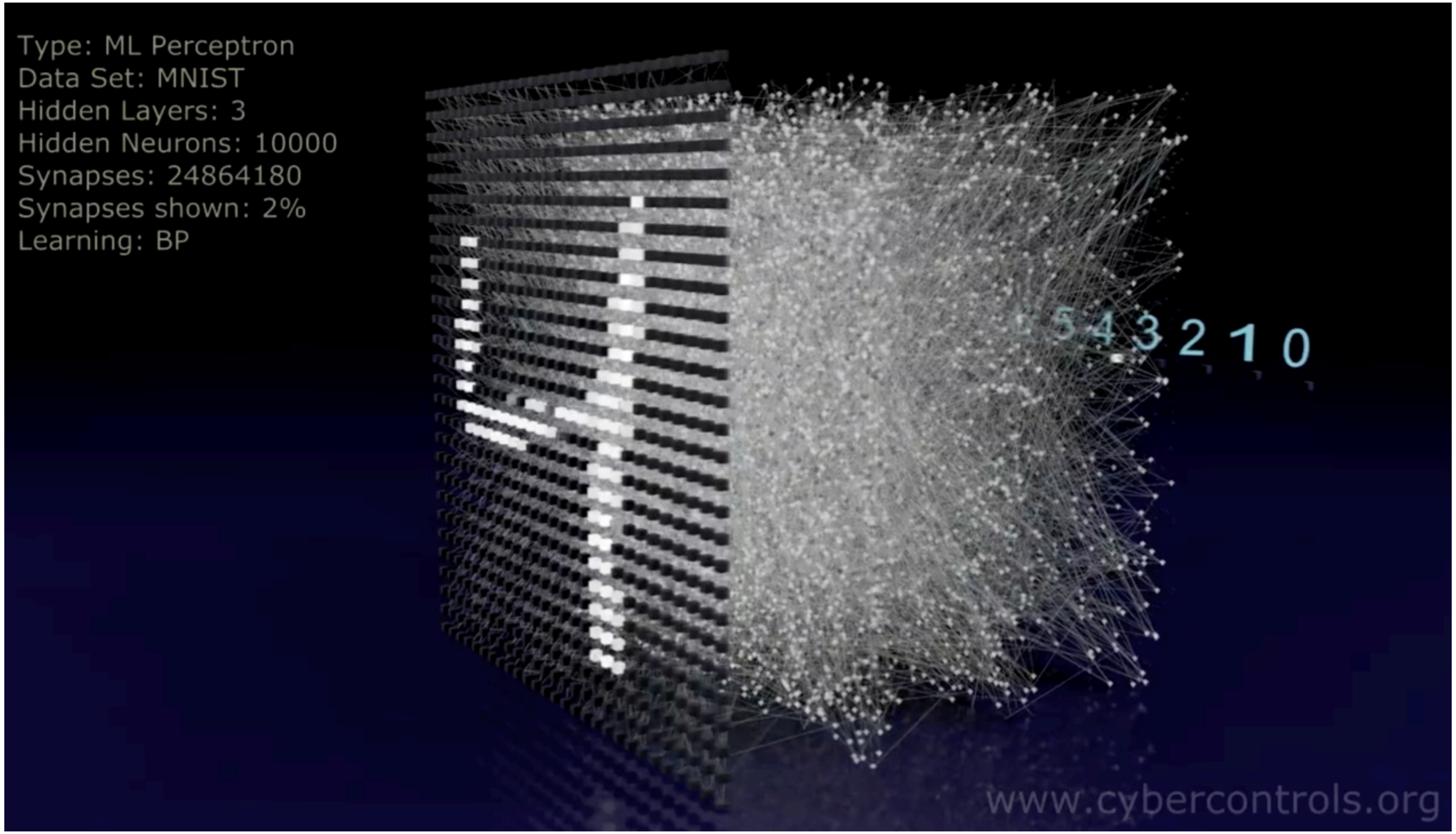
- **<u>Mixed National Institute of Standard and Technology(MNIST)</u>** Database
- 60000개의 트레이닝 데이터와 10000개의 테스트 데이터로 이루어진 Handwritten Digits
- Deep Learning을 포함한 기계학습 분야의 가장 대표적인 예제
  - 어떤 모델이든, 만들었다하면 시험삼아 테스트를 진행하는 가장 기본적인 예제



- 색상 : Gray Scale
- 크기 : 28 by 28 (=784 pixels)

- Test Error Rate (%)
  - Linear Classifier : 12.0 %
  - SVM w/ Gaussian Kernel : 1.4 %
  - Neural Net. : 4.7% ← **Goal! 1%**
  - 35 Conv Net. : 0.23 %

MNIST w/ Deep Learning



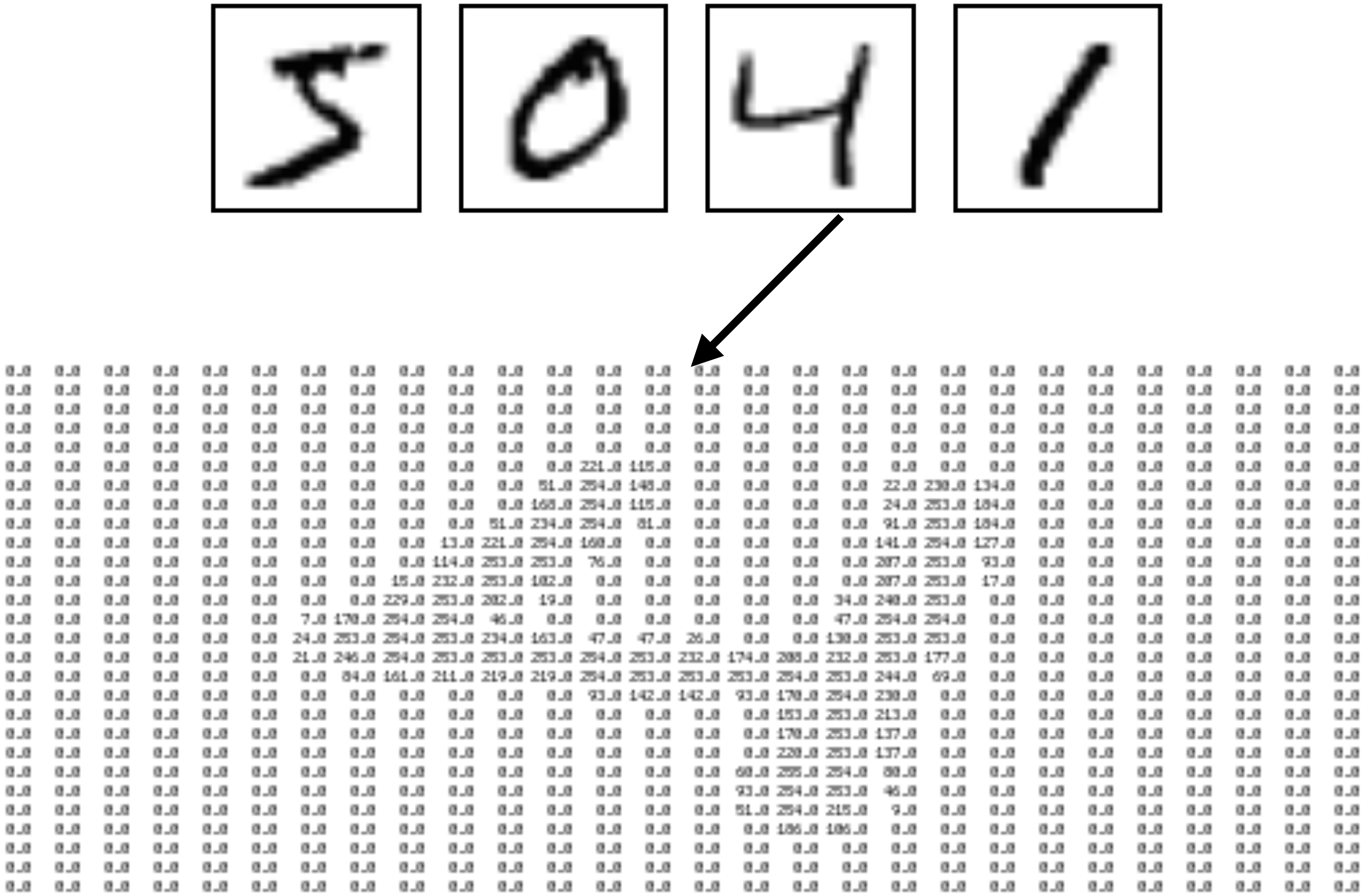
https://www.cybercontrols.org/neuralnetworks

How to classify MNIST

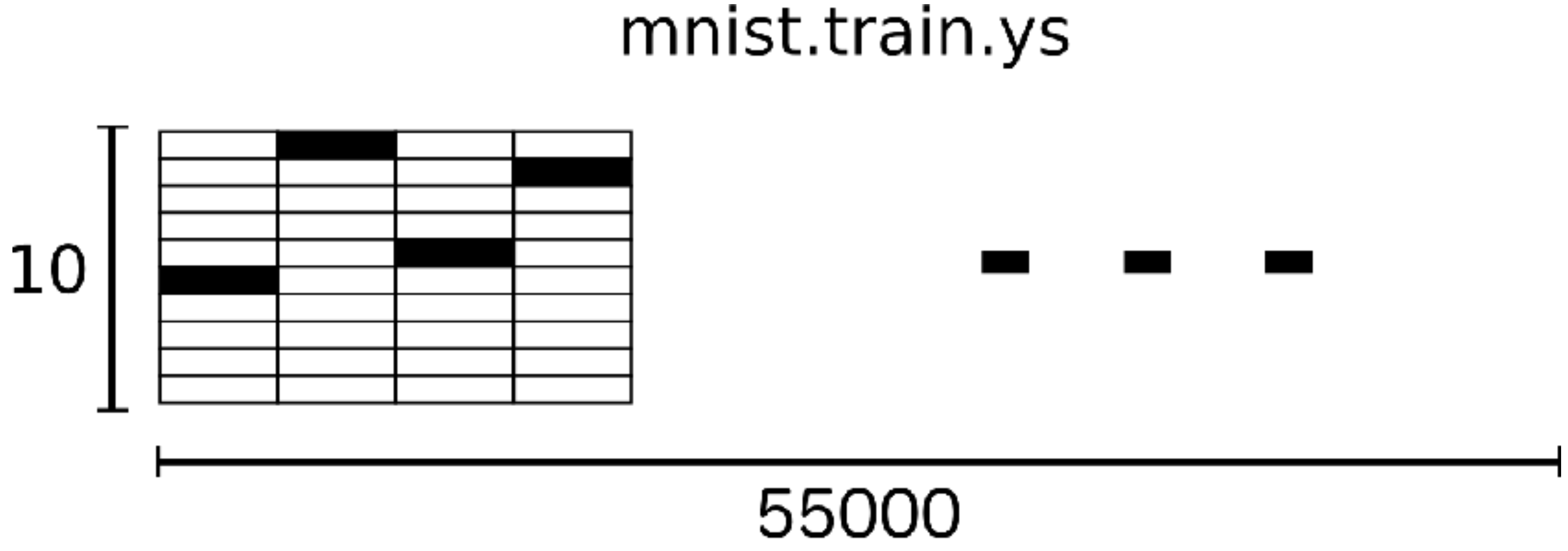
- 어떻게 MNIST를 분류하는 딥러닝 모델을 설계할 수 있을까?
  - Data Preprocessing
    - Input Preprocessing
    - Output Preprocessing (One-Hot Vector Encoding)
  - Neural Network Design (Feed Forward)
    - Hidden Layer w/ Activation Function
    - Output (Softmax Fuction)
  - Backpropagation
  - Evaluation
- 데이터 정제 - 모델 선택 - 평가 Metric설정의 일반적인 Machine Learning의 순서와 일치한다.

Data Preprocessing

- 딥러닝에서 분류를 효율적으로 하기 위해서는 MNIST의 정답(Target)을 어떻게 하는 것이 가장 좋을까?



Data



5 : [0,0,0,0,0,1,0,0,0,0]
0 : [1,0,0,0,0,0,0,0,0,0]
4 : [0,0,0,0,1,0,0,0,0,0]
1 : [0,1,0,0,0,0,0,0,0,0]

Target

Neural Network

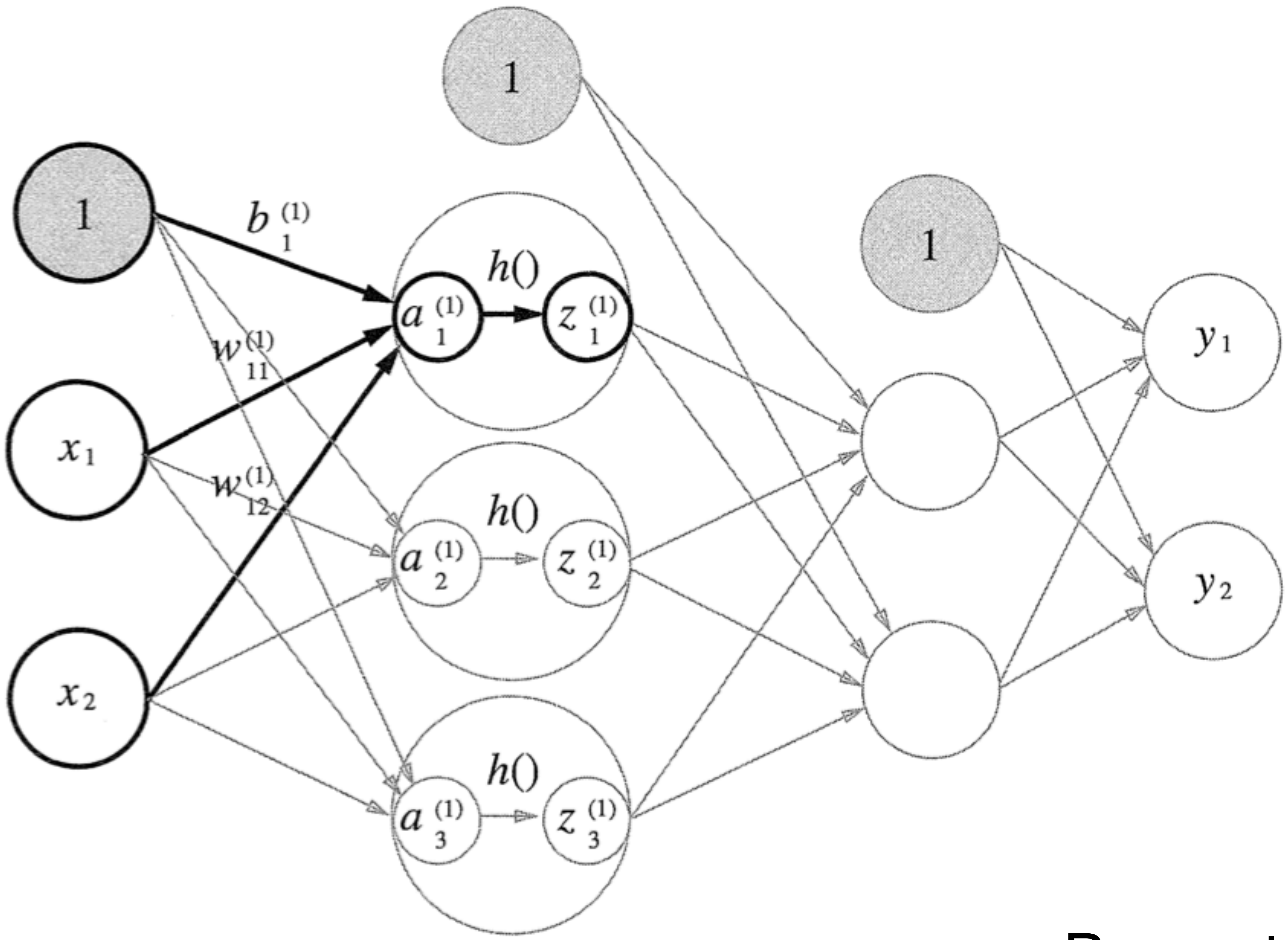
Q. 다음 중 Neural Network를 설계하는데 있어 가장 중요한 것은 무엇인가?

1. 몇 개의 Hidden Layer를 놓을 것인가?
2. 각 Hidden Layer에 속하는 노드의 수는 몇 개로 할 것인가?
3. 각 Hidden Layer에서 어떤 Activation Function을 사용할 것인가?
4. 어떤 구조를 사용할 것인가?

2-layer NN, 300 hidden units, mean square error	none	4.7
2-layer NN, 300 HU, MSE, [distortions]	none	3.6
2-layer NN, 300 HU	deskewing	1.6
2-layer NN, 1000 hidden units	none	4.5
2-layer NN, 1000 HU, [distortions]	none	3.8
3-layer NN, 300+100 hidden units	none	3.05
3-layer NN, 300+100 HU [distortions]	none	2.5
3-layer NN, 500+150 hidden units	none	2.95
3-layer NN, 500+150 HU [distortions]	none	2.45
3-layer NN, 500+300 HU, softmax, cross entropy, weight decay	none	1.53
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6
2-layer NN, 800 HU, cross-entropy [affine distortions]	none	1.1
2-layer NN, 800 HU, MSE [elastic distortions]	none	0.9
2-layer NN, 800 HU, cross-entropy [elastic distortions]	none	0.7

Convolutional net LeNet-1	subsampling to 16x16 pixels	1.7
Convolutional net LeNet-4	none	1.1
Convolutional net LeNet-4 with K-NN instead of last layer	none	1.1
Convolutional net LeNet-4 with local learning instead of last layer	none	1.1
Convolutional net LeNet-5, [no distortions]	none	0.95
Convolutional net LeNet-5, [huge distortions]	none	0.85

Feed Forward



1. Input과 Weight, Bias의 합

$$a_1^{(1)} = x_1 w_{11}^{(1)} + x_2 w_{12}^{(1)} + b_1^{(1)}$$

$$a_i^{(l)} = b_i^{(l)} + \sum_{j=1} x_j w_{ij}^{(l)} + x_j w_{ij}^{(l)}$$

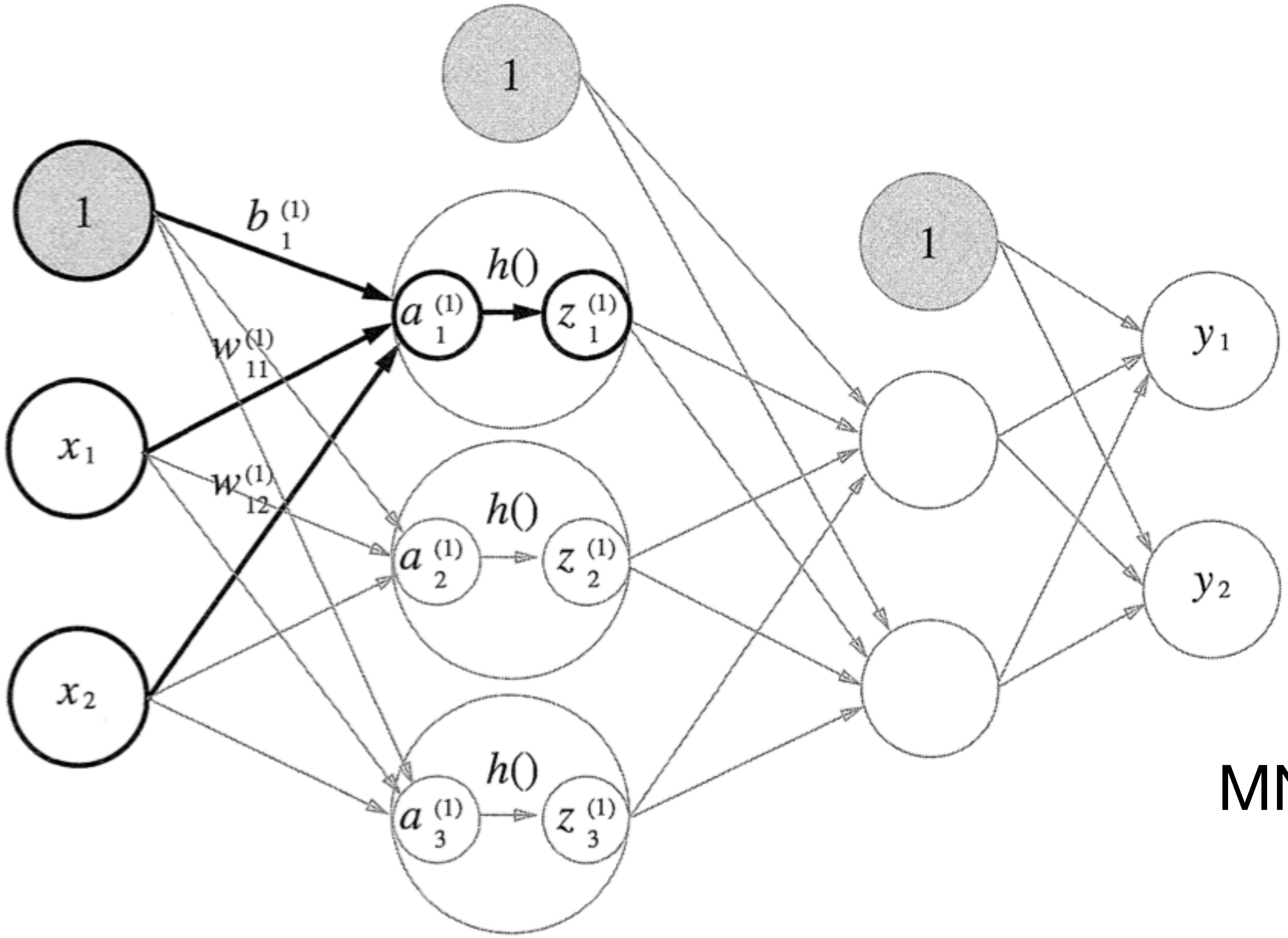
2. Activation Function

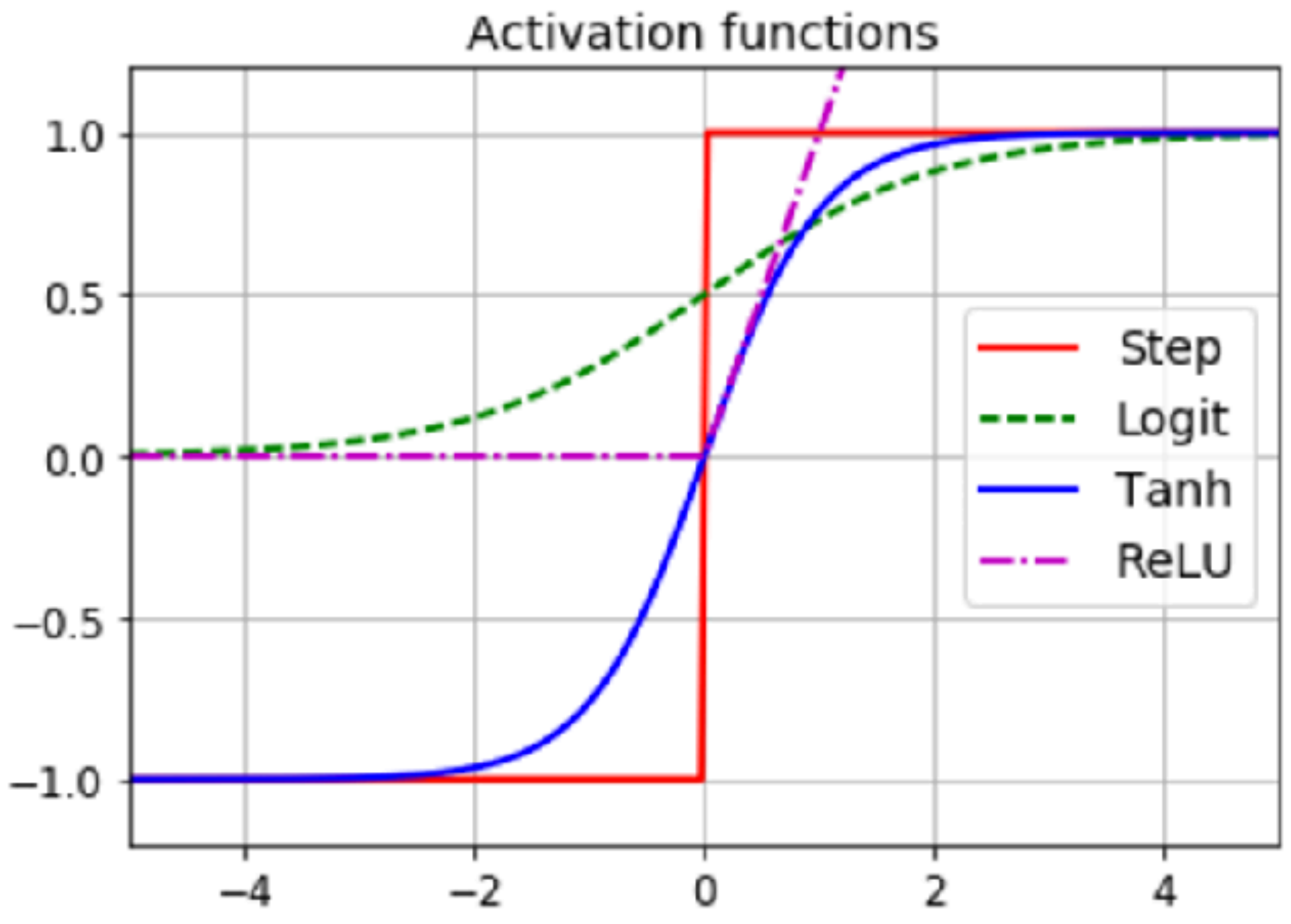
$$z_i^{(l)} = h(a_i^{(l)})$$

3. Next Layer

Perceptron에서의 Feed Forward와 다르지 않다!

Feed Forward

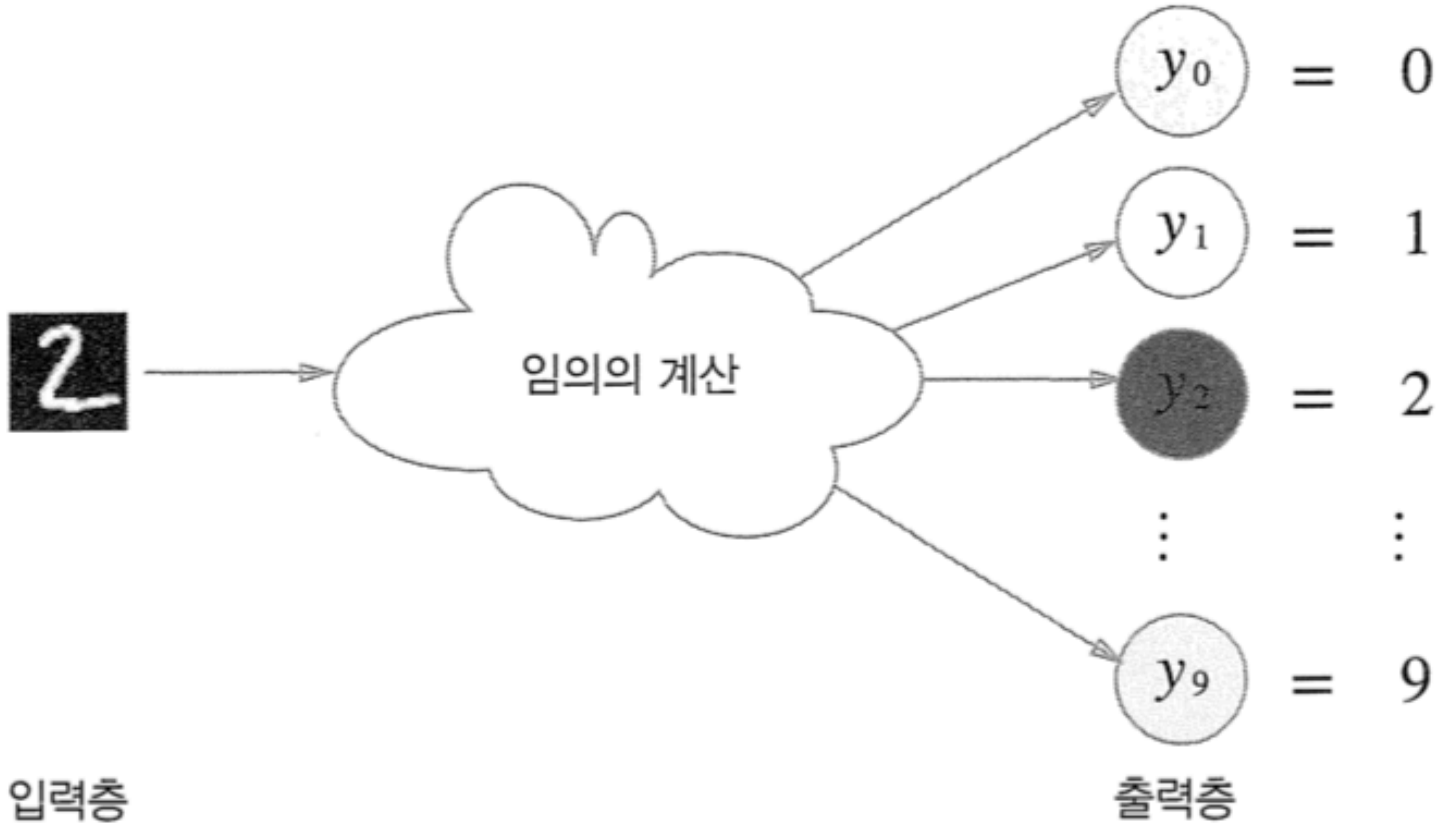




MNIST분류에 있어 많은 경우에 ReLU를 쓴다.
왜 그럴까?

Hint: MNIST의 Input은 784개이다.

Output Layer



- 결과를 어떻게 표현하는 것이 가장 좋을까?

  1. 정답 = 1 : [0,   1,0,0,0,0,0,   0,0,0]
  2. 정답 = 1 : [0,0.7,0,0,0,0,0,0.3,0,0]

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

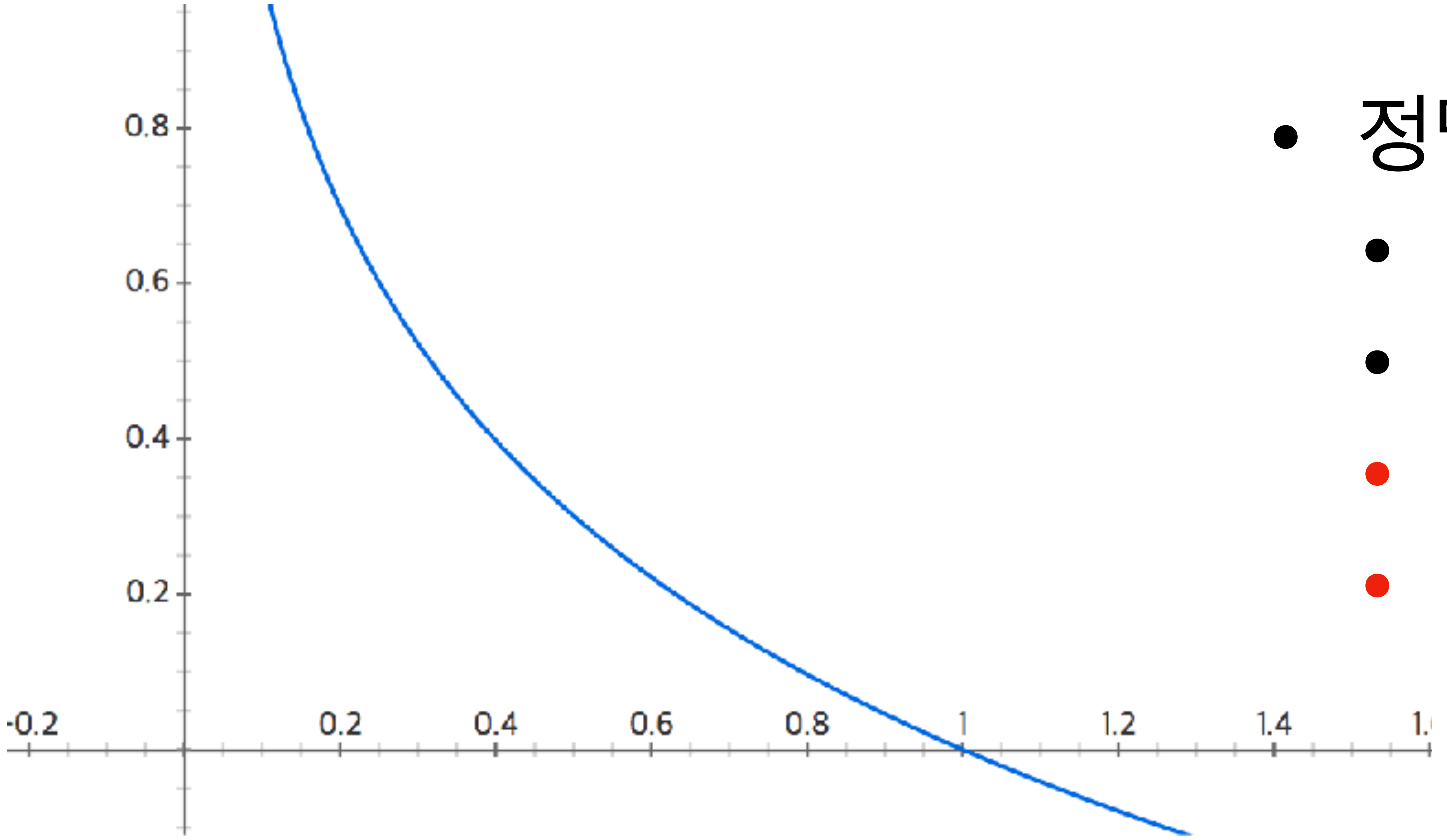
Softmax Function

- 왜 단순 확률이 아닌 exp(지수)함수의 합일까?
  - Cross-Entropy Cost Function

Cross-Entropy Cost Function

- Cross Entropy는 서로 다른 두 확률분포를 비교하여 무질서한 정도를 측정한 값.

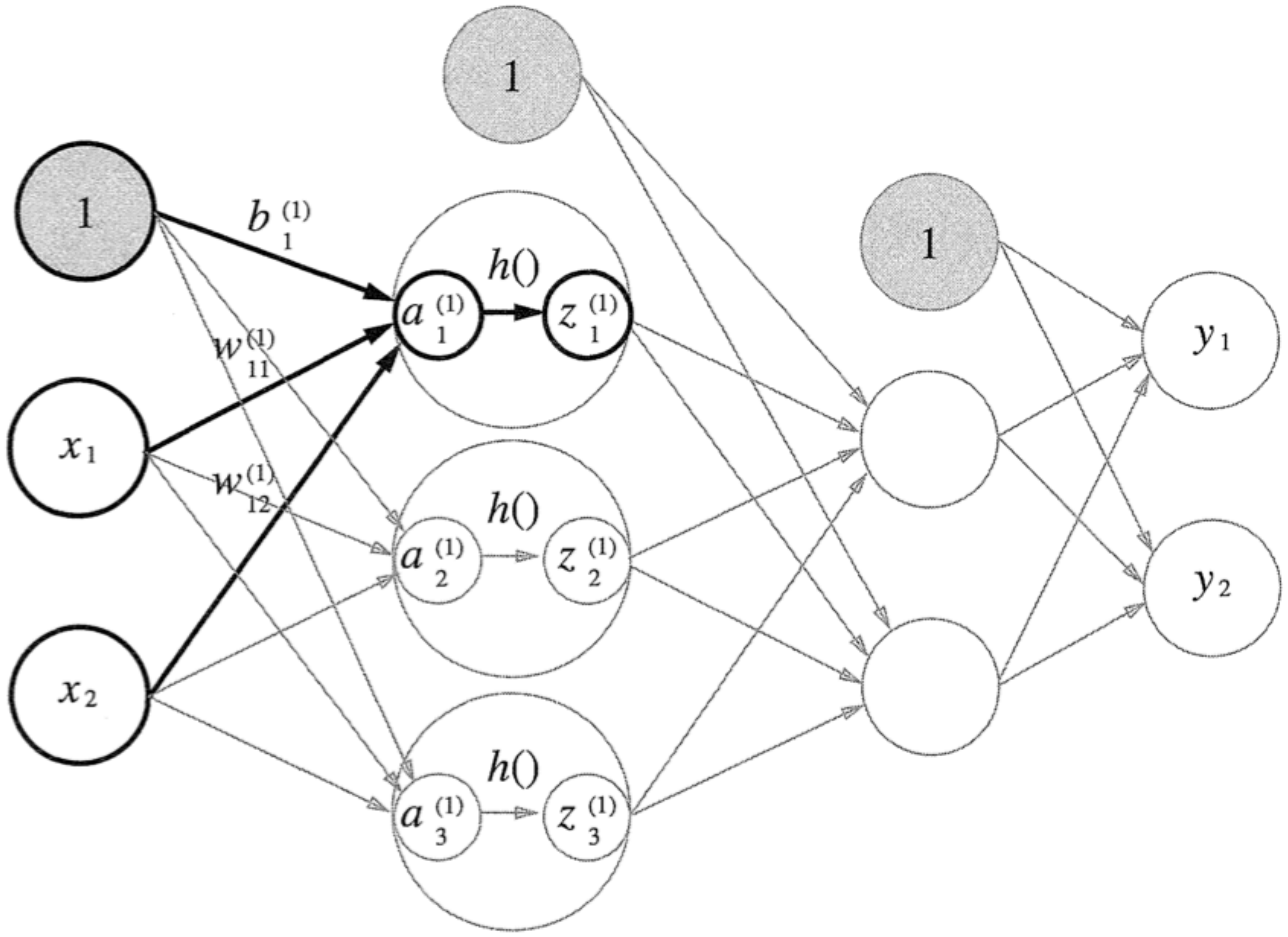
$$H(p,q) = -\sum_x p(x)\log q(x)$$



Log함수의 그래프

- 정답과 모델에 의해 도출된 결과 값은, (ex. 3)
  - 정답 : [0,0,0,1,0,0,0,0,0,0]
  - 결과 : [0,0.1,0.2,0.5,0,0,0,0,0,0.2]
  - 두 개의 확률 분포를 비교하는 것이다!
  - 많이 다르면 큰 값을 Error!, 조금 다르면 작은 값을 Error!

Back Propagation



결국 Delta Rule과 같은 방법!

$$\Delta w_{ij} = \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial h_j}\frac{\partial h_j}{w_{ij}}$$

$$E = \frac{1}{2}(T-O)^2 \qquad \frac{\partial E}{\partial O} = (T-O)$$

$$O = g(h_j) \qquad \frac{\partial O}{\partial h_j} = g'(h_j)$$

$$h_j = \sum w_{ij} x_i \qquad \frac{\partial h_j}{\partial w_{ij}} = x_i$$

$$\Delta w_{ij} = \alpha (T-O) g'(h_j) x_i$$

계산.. 해야합니까?

```python
learning_rate = 0.01

with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)
```

- 아니요, TensorFlow에서는 이 모든 계산을 한,두 줄로 해결해줍니다.
- 다만, 원리는 기억해 두시길 바랍니다.
  - Convolution Neural Network, Recurrent Neural Network 모두 이러한 계산을 사용합니다.

$$\Delta w_{ij} = \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O}\frac{\partial O}{\partial h_j}\frac{\partial h_j}{w_{ij}}$$

$$\Delta w_{ij} = \alpha(T - O)g'(h_j)x_i$$

DNN with Python

```
0 Train accuracy: 0.94 Test accuracy: 0.9101
1 Train accuracy: 0.9 Test accuracy: 0.9302
2 Train accuracy: 0.96 Test accuracy: 0.9394
3 Train accuracy: 0.9 Test accuracy: 0.9453
4 Train accuracy: 0.98 Test accuracy: 0.9503
```

```
Predicted classes: [7 2 1 0 4 1 4 9 6 9 0 6 9 0 1 5 9 7 3 4]
Actual classes:    [7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4]
```