Week 7(2/3)

Feed-forward Neural Network

Machine Learning with Python

Handong Global University
Prof. Youngsup Kim
idebtor@gmail.com

Feed-forward Neural Network - Example

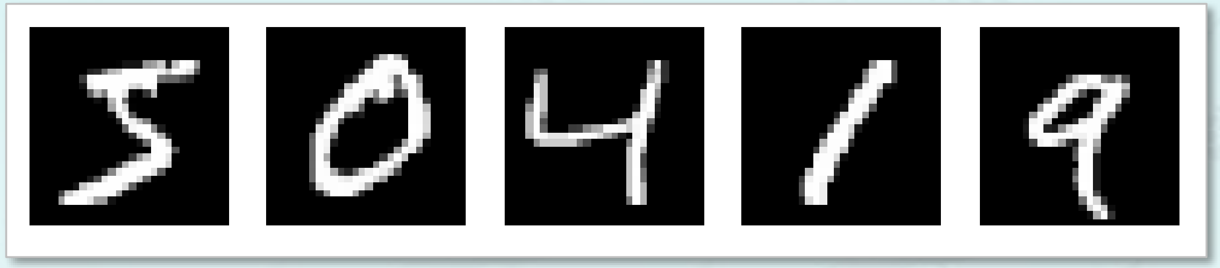
- Objectives
  - Feed-forward Neural Network – Example

- Topics
  - MNIST Dataset
  - Designing Multi-Layer Neural Network
  - Feed-forward Neural Network – Signal Processing Example
  - Feed-forward Neural Network - Example Implementation

1. Multi Layer Neural Network: Input Data

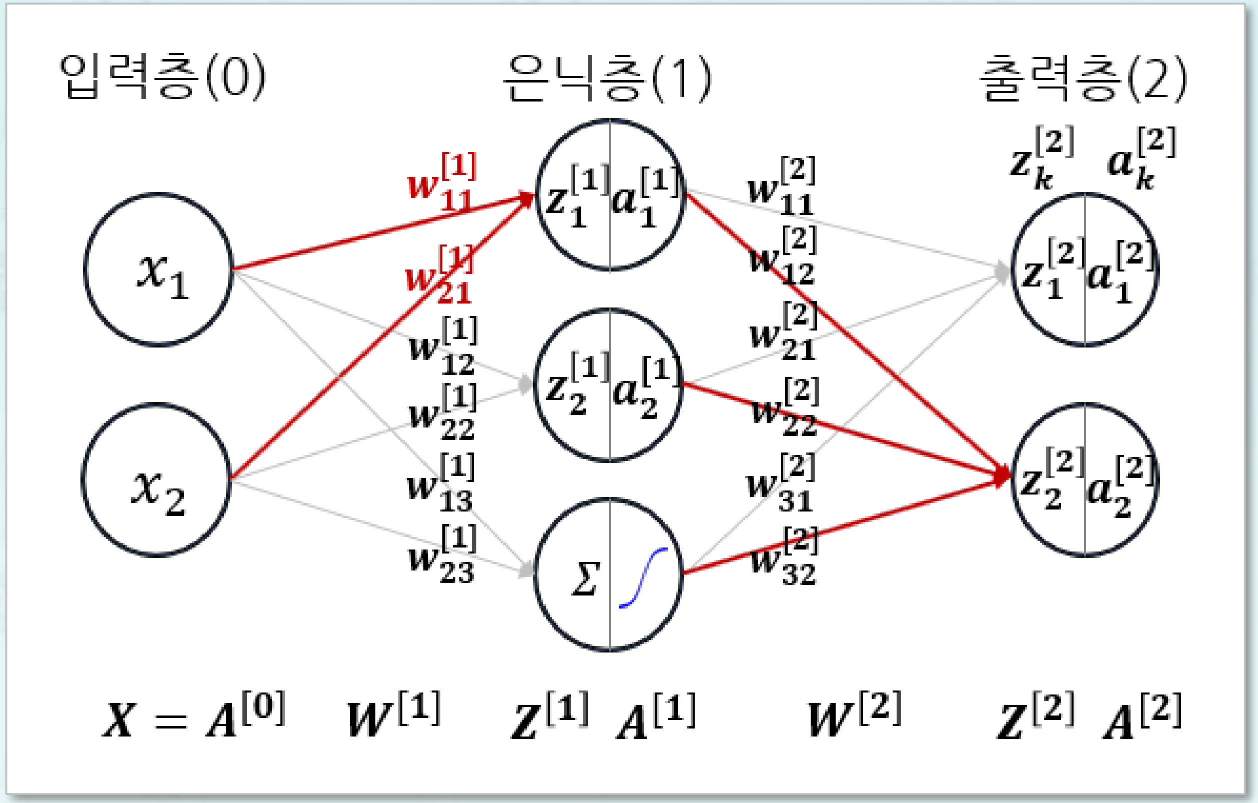
1. Multi Layer Neural Network: Input Data

- **MNIST**



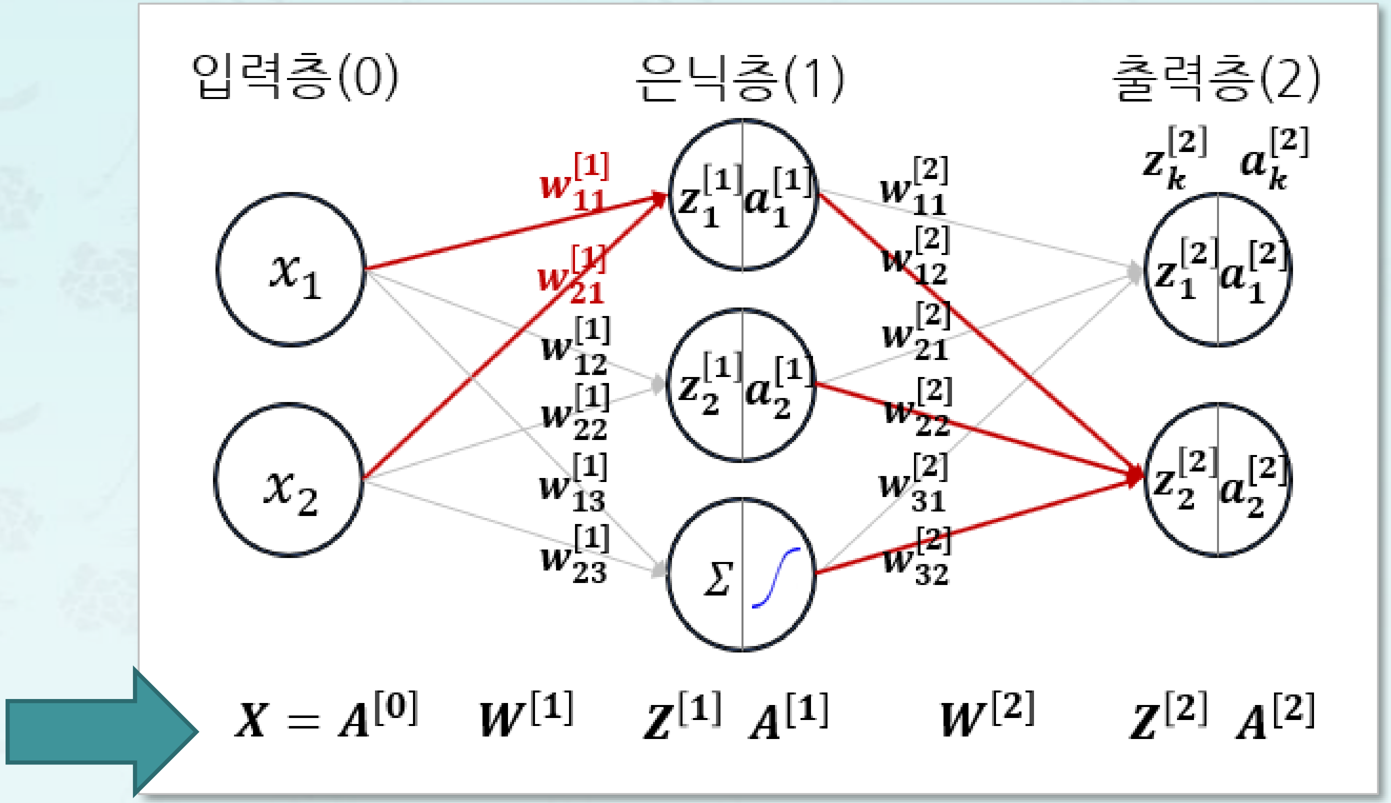
1. Multi Layer Neural Network: Notation(Review)

- **Multi Layer Neural Network**



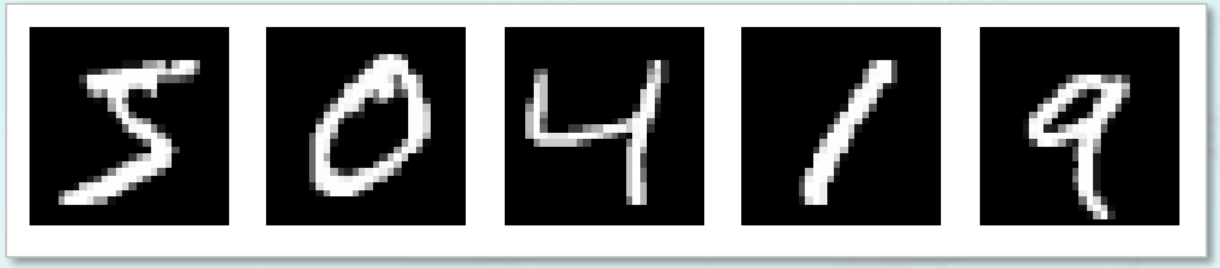
1. Multi Layer Neural Network: Notation(Review)

- **Multi Layer Neural Network**

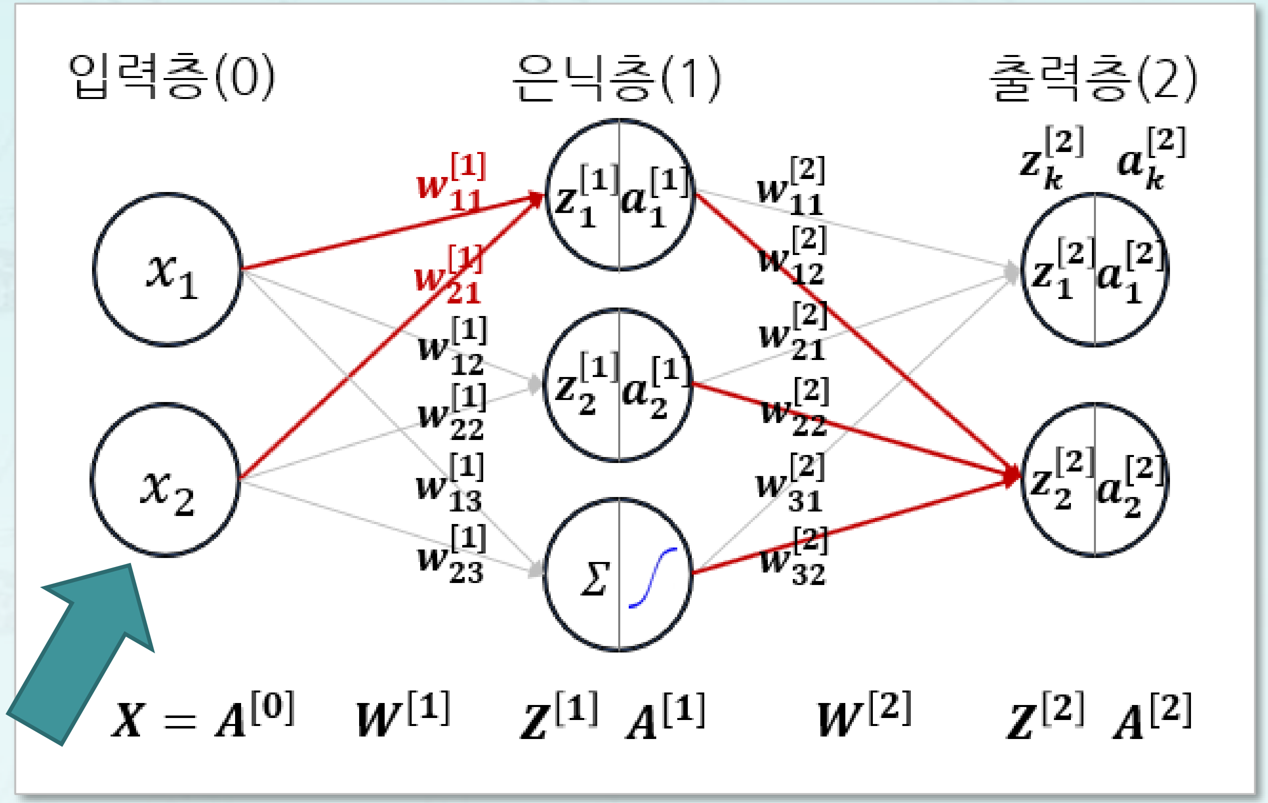


1. Multi Layer Neural Network: Input Data

- **MNIST**

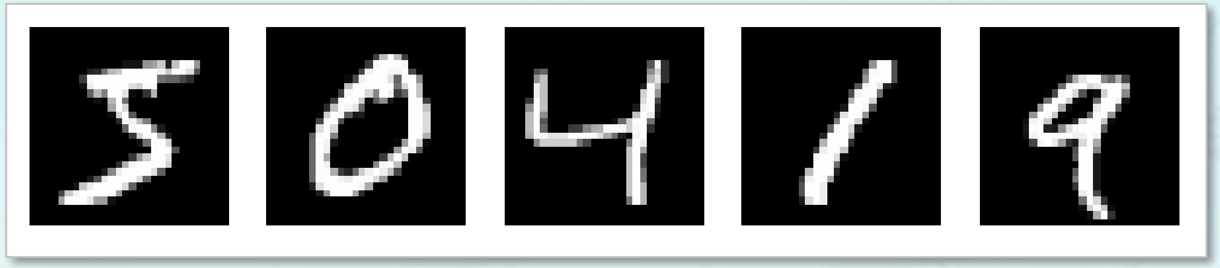


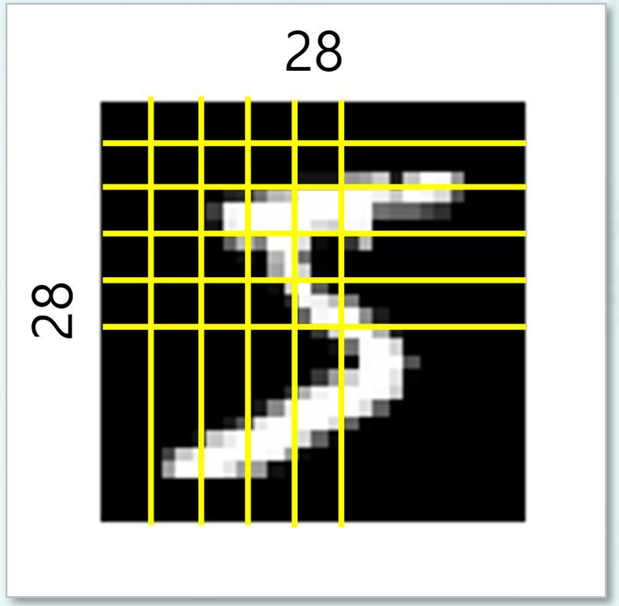
- **Multi Layer Neural Network**



1. Multi Layer Neural Network: Input Data

- **MNIST**
  - 28x28

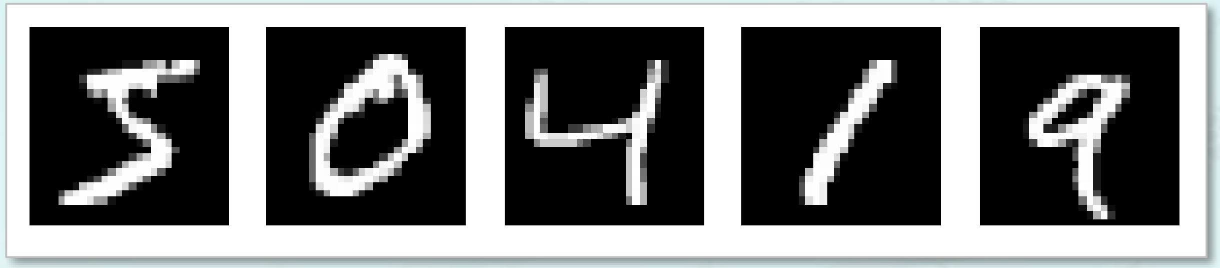


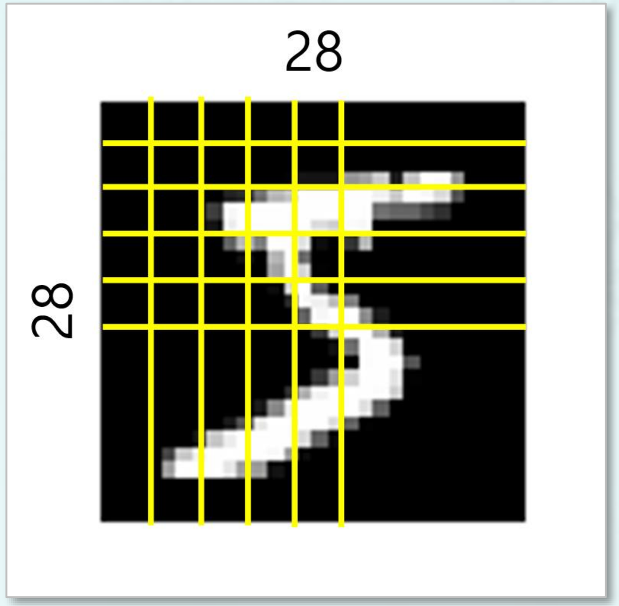


- **Multi Layer Neural Network**

1. Multi Layer Neural Network: Structure

- MNIST
  - 28x28

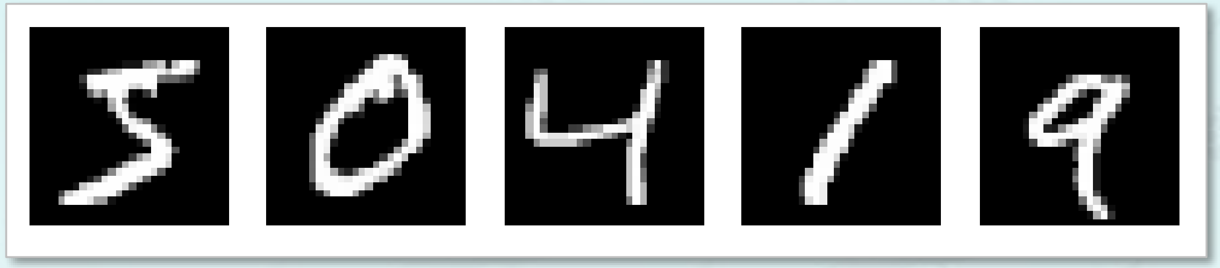


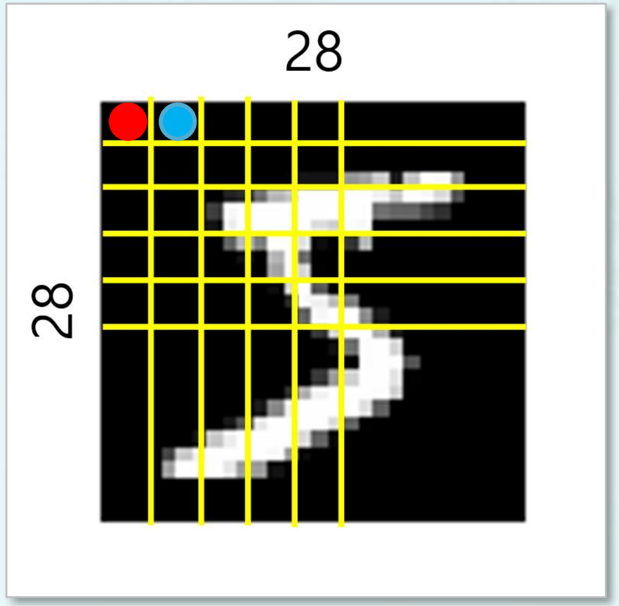


- Input Layer : 784(28x28)

1. Multi Layer Neural Network: Structure

- **MNIST**
  - 28x28





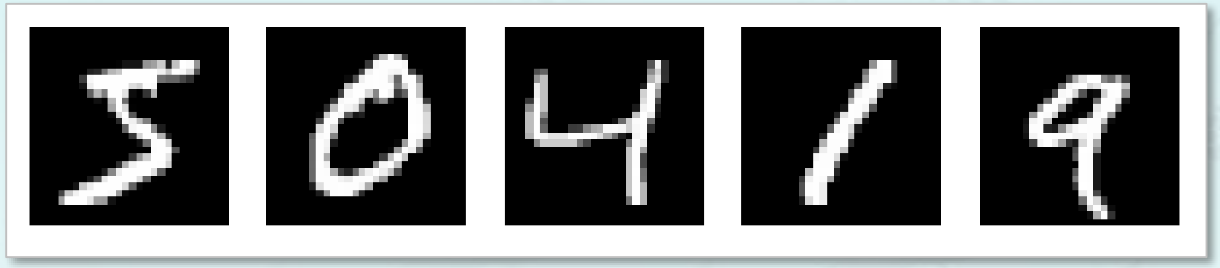
- **Input Layer : 784(28x28)**

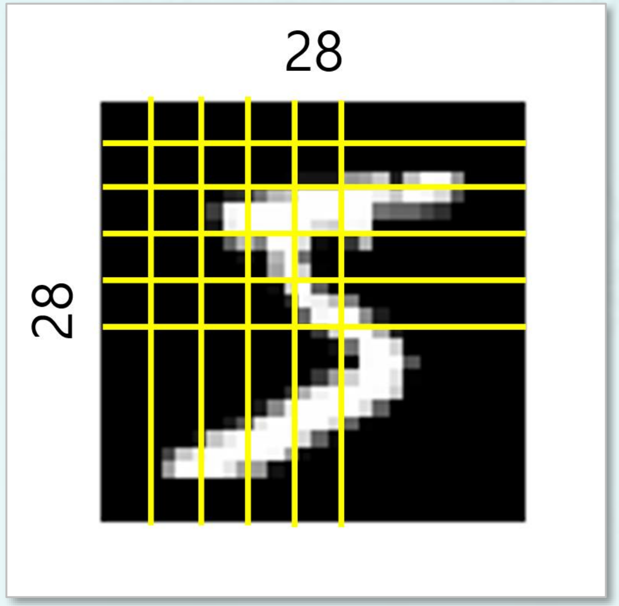
$$\mathbf{X} \in \mathbb{R}^{nxm}$$

$$\mathbf{X} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_{783}^{(0)} \\ x_{784}^{(0)} \end{pmatrix}$$

1. Multi Layer Neural Network: Structure

- MNIST
  - 28x28

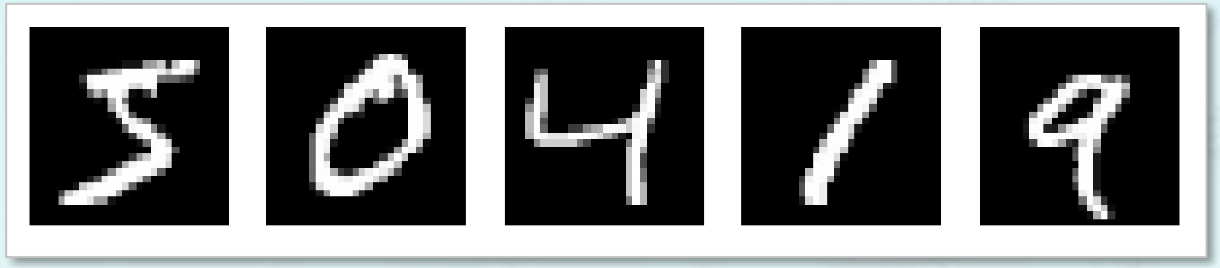


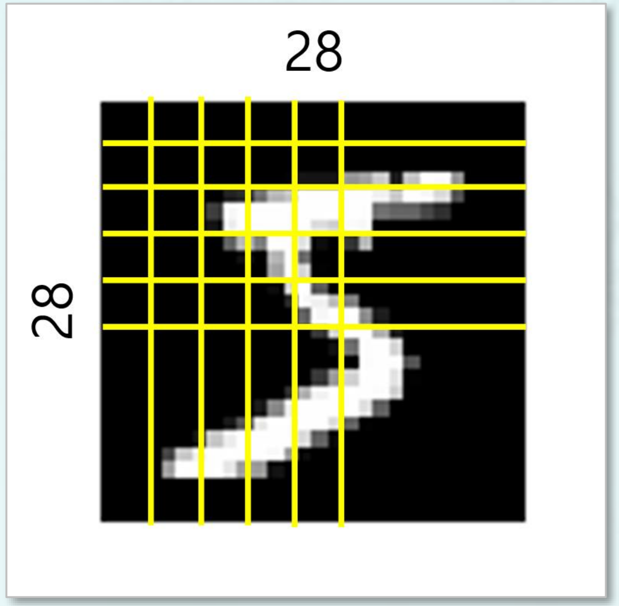


- Input Layer : 784(28x28)
- Hidden Layer : 100

1. Multi Layer Neural Network: Structure

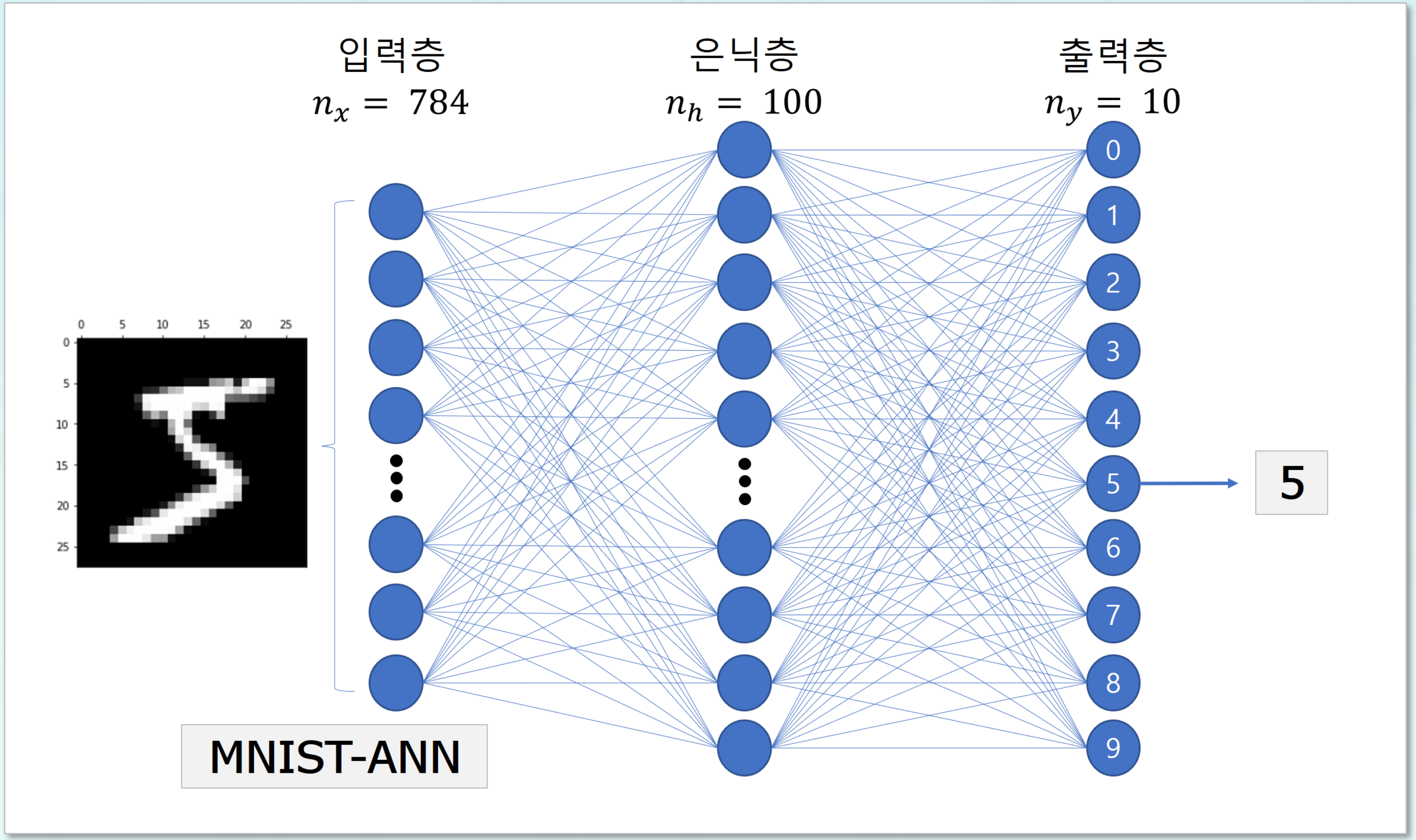
- **MNIST**
  - 28x28





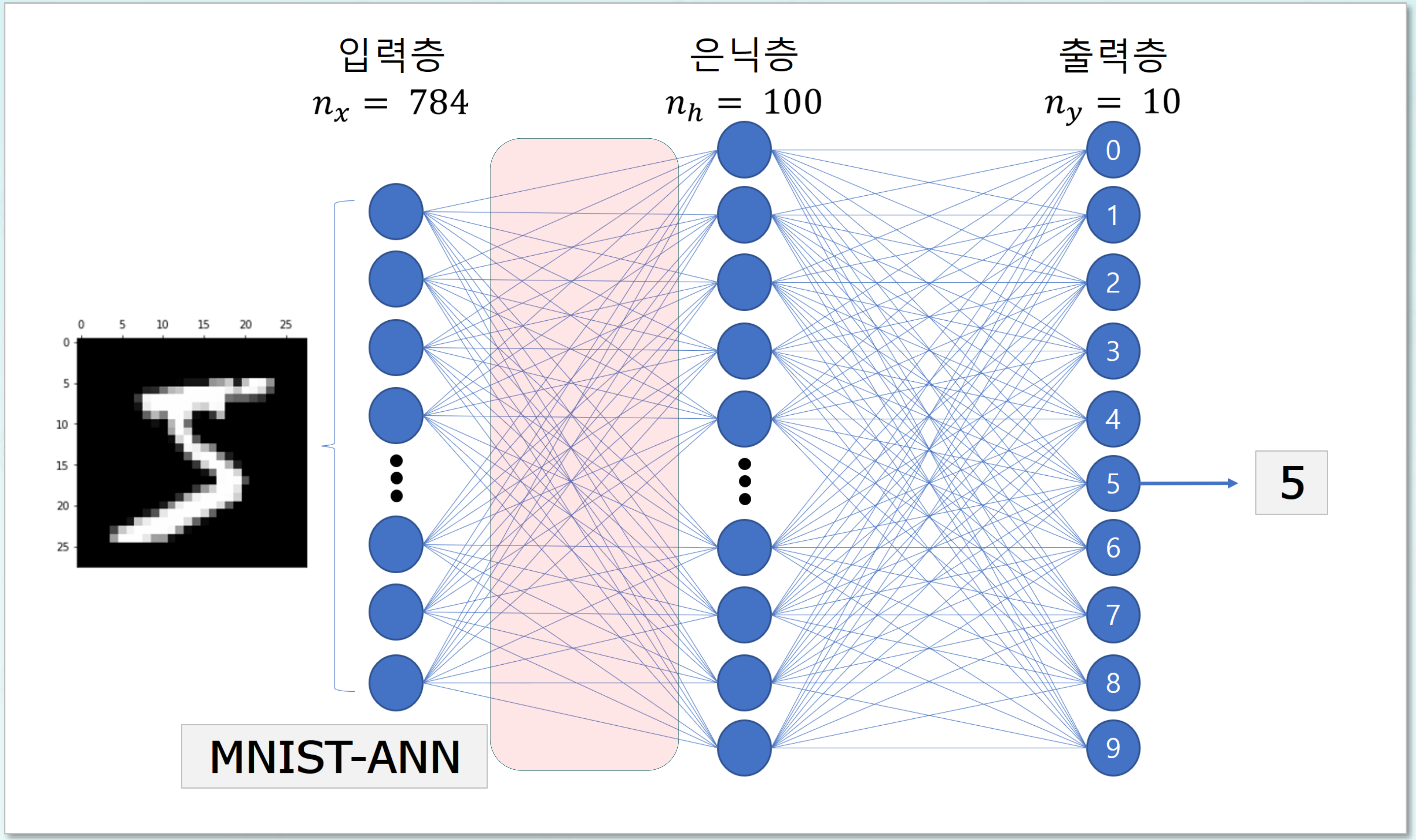
- **Input Layer : 784(28x28)**
- **Hidden Layer : 100**
- **Output Layer Computation : 10**

1. Multi Layer Neural Network: Structure



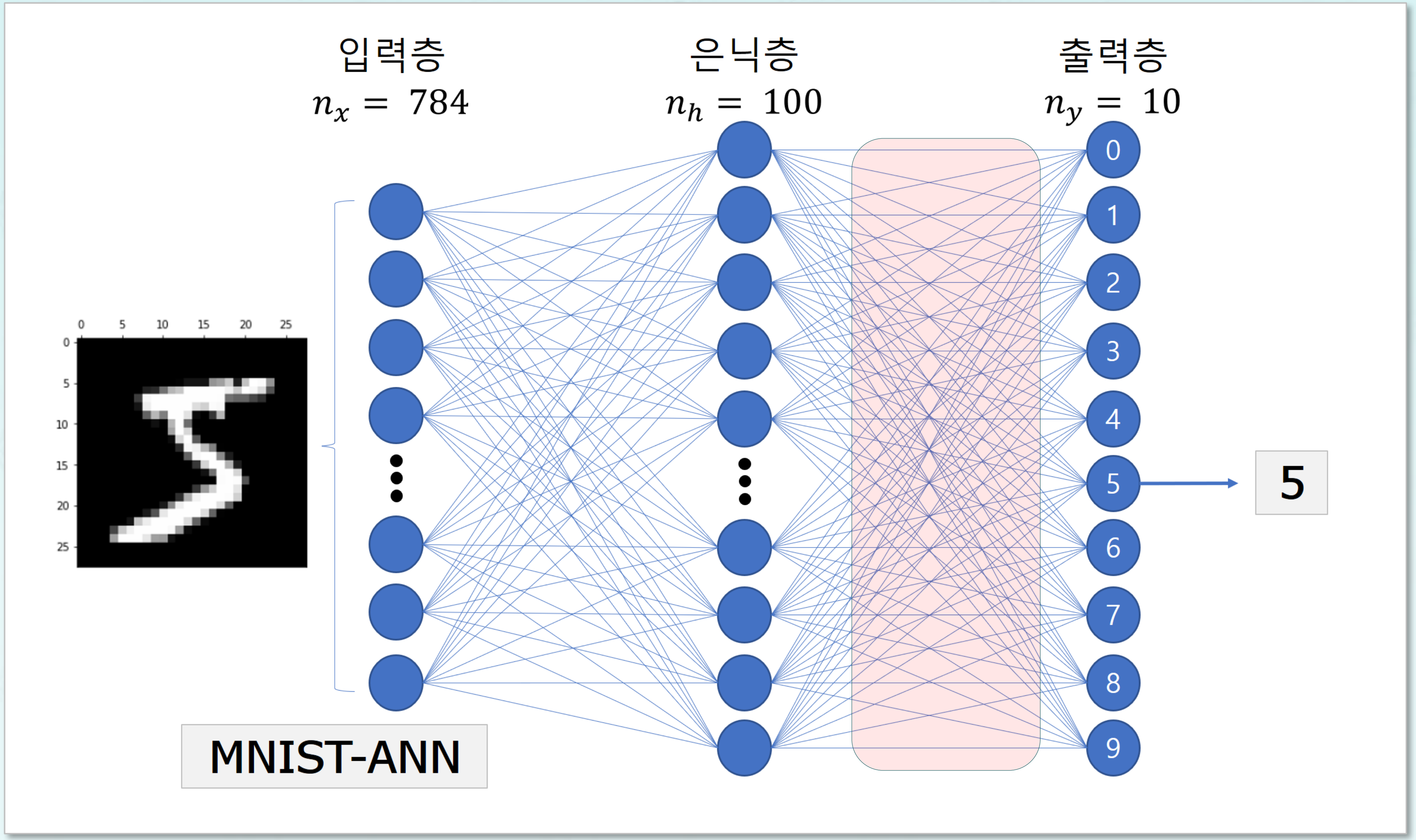
1. Multi Layer Neural Network: Structure

- **Weights**
  - **784x100**



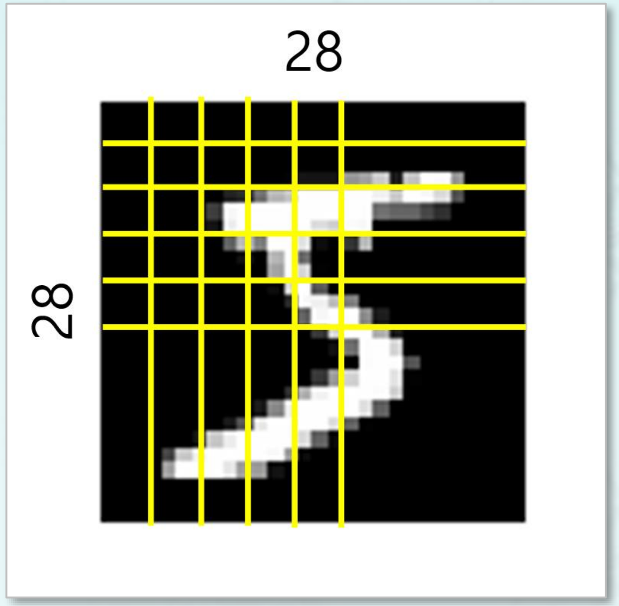
1. Multi Layer Neural Network: Structure

- **Weights**
  - **784x100**
  - **100x10**



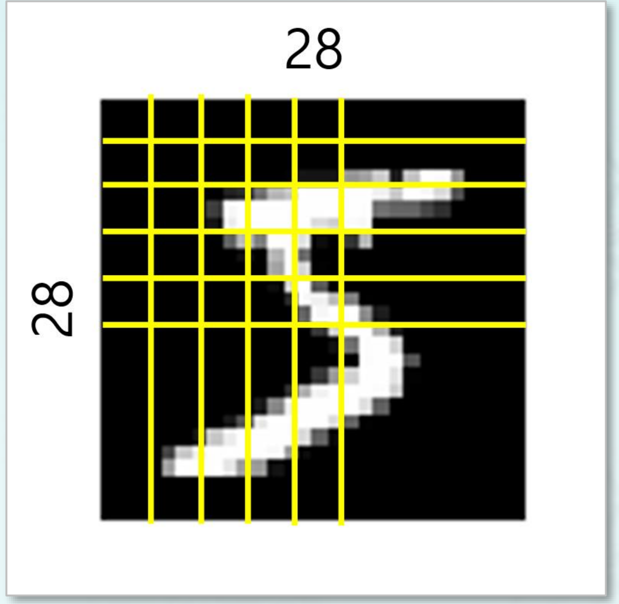
2. ANN Implementation: Input Dataset

- X^{nxm}
- n = 784, m = 1



2. ANN Implementation: Input Dataset

- X^{nxm}
- n = 784, m = 1

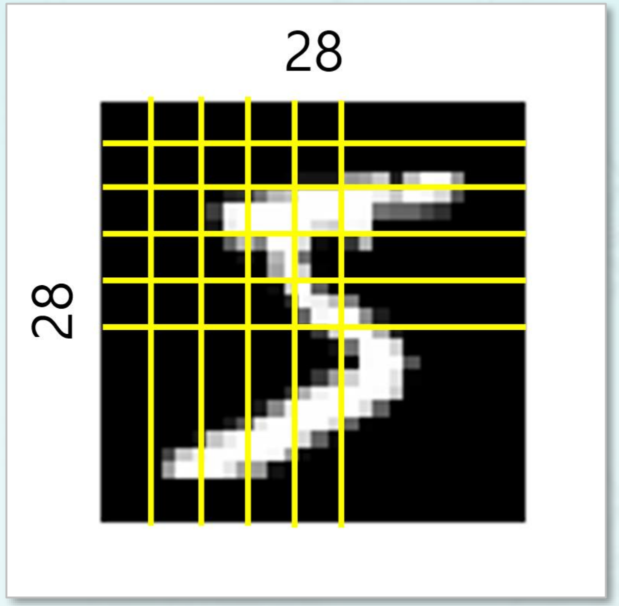


$$\mathbf{X} \in \mathbb{R}^{nxm}$$

$$\mathbf{X} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_{783}^{(0)} \\ x_{784}^{(0)} \end{pmatrix}$$

2. ANN Implementation: Input Dataset

- X^{nxm}
- n = 784, m = 1



$$\mathbf{X} \in \mathbb{R}^{nxm}$$

$$\mathbf{X} = \begin{pmatrix} x_{\boxed{1}}^{(0)} \\ x_{\boxed{2}}^{(0)} \\ \vdots \\ x_{\boxed{783}}^{(0)} \\ x_{\boxed{784}}^{(0)} \end{pmatrix}$$

2. ANN Implementation: Reusing Weights

- Preprocessed Weights
- 96% Accuracy

2. ANN Implementation: Reusing Weights

- **$W^{[1]}$ = 100x784**

$$W^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & W_1^{(783)} & w_1^{(784)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & W_2^{(783)} & w_2^{(784)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{100}^{(1)} & w_{100}^{(2)} & \cdots & w_{100}^{(783)} & w_{100}^{(784)} \end{pmatrix}$$

2. ANN Implementation: Reusing Weights

- $W^{[1]}$ = 100x784

$$W^{[1]} = \begin{pmatrix} w_1^{(1)} & \boxed{w_1^{(2)}} & \cdots & W_1^{(783)} & w_1^{(784)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & W_2^{(783)} & w_2^{(784)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{100}^{(1)} & w_{100}^{(2)} & \cdots & w_{100}^{(783)} & w_{100}^{(784)} \end{pmatrix}$$

```
!type data/w_xh.txt
```

```
!cat data/w_xh.txt
```

```
[-1.65955991e-01  4.40648987e-01
 -7.06488218e-01 -8.15322810e-01
 -2.06465052e-01  7.76334680e-02
 -5.92088802e-01  7.54060585e-01
 -1.65390395e-01  1.17379657e-01
```

2. ANN Implementation: Hidden Layer Computation

2. ANN Implementation: Hidden Layer Computation

$$
\begin{aligned}
\mathbf{Z}^{[1]} &= W^{[1]} A^{[0]} \\
&= \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\
&= \begin{pmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{pmatrix}
\end{aligned}
$$

2. ANN Implementation: Hidden Layer Computation

$$\mathbf{Z}^{[1]} = W^{[1]}A^{[0]}$$
$$= \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$
$$= \begin{pmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{pmatrix}$$

$$\mathbf{W}^{[1]}\mathbf{X}^{[0]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & w_1^{(783)} & w_1^{(784)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & w_2^{(783} & w_2^{(784)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{100}^{(1)} & w_{100}^{(2)} & \cdots & w_{100}^{(783)} & w_{100}^{(784)} \end{pmatrix} \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_{783}^{(0)} \\ x_{784}^{(0)} \end{pmatrix}$$

$$\mathbf{A}^{[1]} = sigmoid(\mathbf{Z}^{[1]})$$

2. ANN Implementation: Output Layer Computation

- $A^{[1]}$: 100x1

$$W^{[2]}A^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & w_1^{(99)} & w_1^{(100)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & w_2^{(99)} & w_2^{(100)} \\ \vdots & \vdots & \vdots & \vdots & \\ w_{10}^{(1)} & w_{10}^{(2)} & \cdots & w_{10}^{(99)} & w_{10}^{(100)} \end{pmatrix} \begin{pmatrix} \boxed{\begin{matrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{99}^{(1)} \\ a_{100}^{(1)} \end{matrix}} \end{pmatrix}$$

2. ANN Implementation: Output Layer Computation

- $A^{[1]}$: 100x1
- $W^{[2]}$: 10x100

$$W^{[2]}A^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & w_1^{(99)} & w_1^{(100)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & w_2^{(99)} & w_2^{(100)} \\ \vdots & \vdots & \vdots & \vdots & \\ w_{10}^{(1)} & w_{10}^{(2)} & \cdots & w_{10}^{(99)} & w_{10}^{(100)} \end{pmatrix} \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{99}^{(1)} \\ a_{100}^{(1)} \end{pmatrix}$$

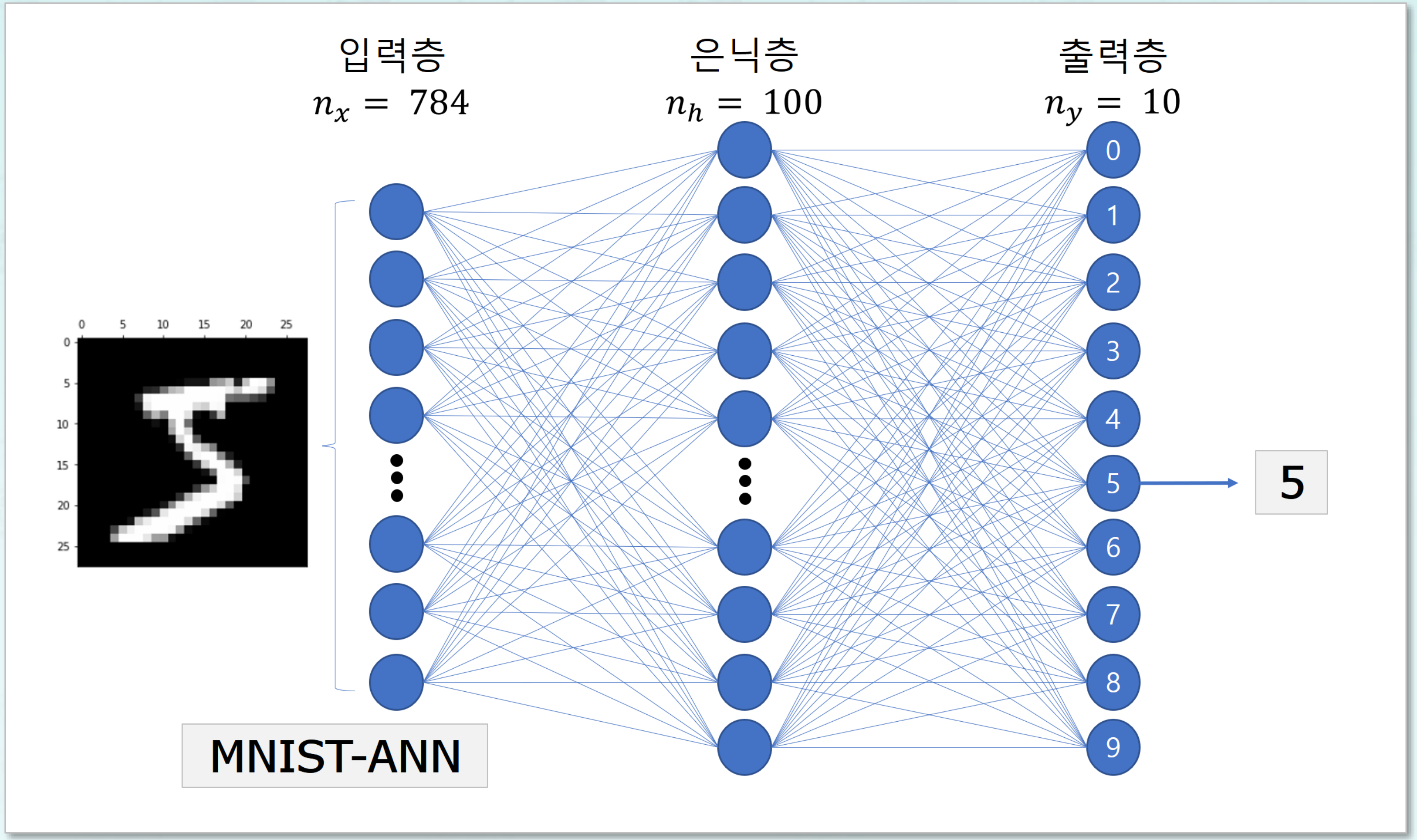
2. ANN Implementation: Output Layer Computation

- $A^{[1]}$: 100x1
- $W^{[2]}$: 10x100

$$W^{[2]}A^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & w_1^{(99)} & w_1^{(100)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & w_2^{(99)} & w_2^{(100)} \\ \vdots & \vdots & \vdots & \vdots & \\ w_{10}^{(1)} & w_{10}^{(2)} & \cdots & w_{10}^{(99)} & w_{10}^{(100)} \end{pmatrix} \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{99}^{(1)} \\ a_{100}^{(1)} \end{pmatrix}$$

$$\mathbf{A}^{[2]} = sigmoid(\mathbf{Z}^{[2]})$$

2. ANN Implementation: Structure



2. ANN Implementation

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

2. ANN Implementation:

- **Library**

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

2. ANN Implementation:

- **Library**

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

2. ANN Implementation:

- **Activation function**

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

2. ANN Implementation:

- **Input Dataset Loading**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

2. ANN Implementation:

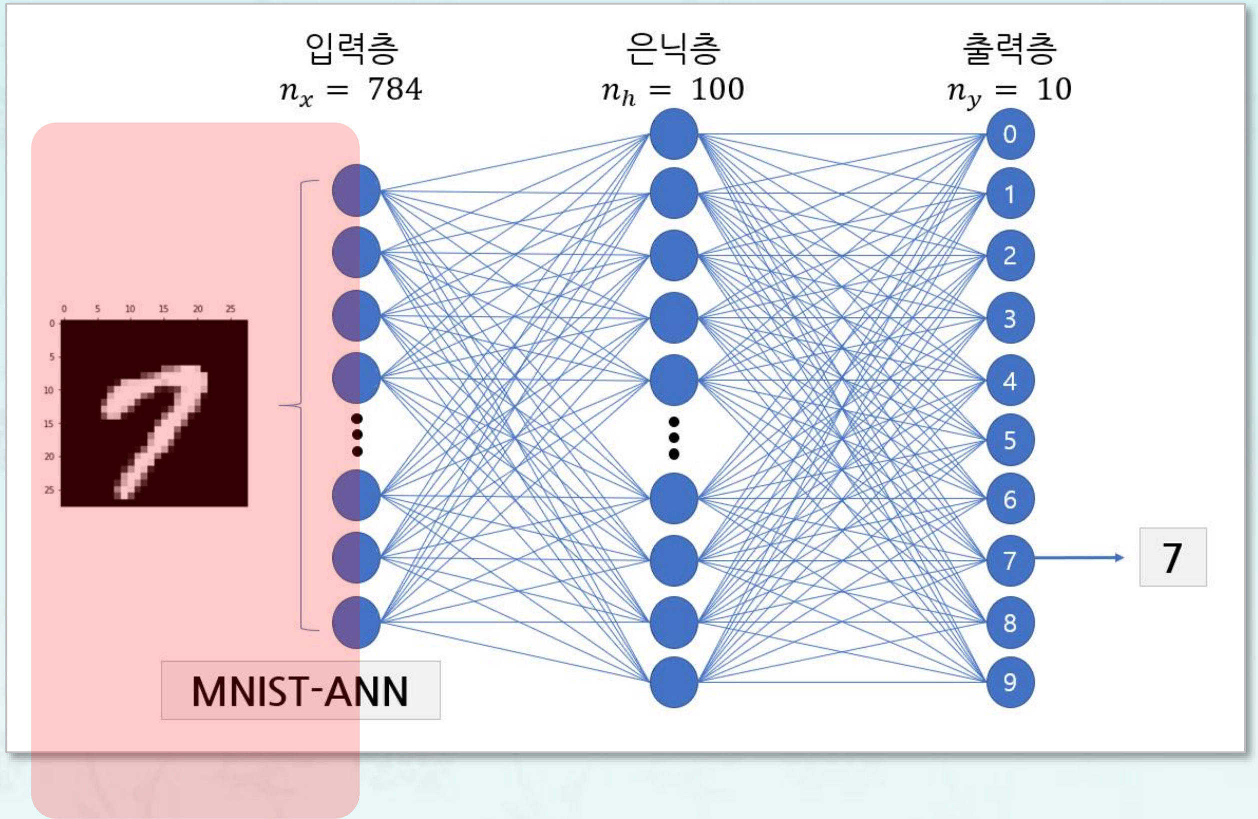
- **Input Dataset Loading**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```



2. ANN Implementation:

- **Reusing Weights**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

$$W^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & W_1^{(783)} & w_1^{(784)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & W_2^{(783)} & w_2^{(784)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{100}^{(1)} & w_{100}^{(2)} & \cdots & w_{100}^{(783)} & w_{100}^{(784)} \end{pmatrix}$$

2. ANN Implementation:

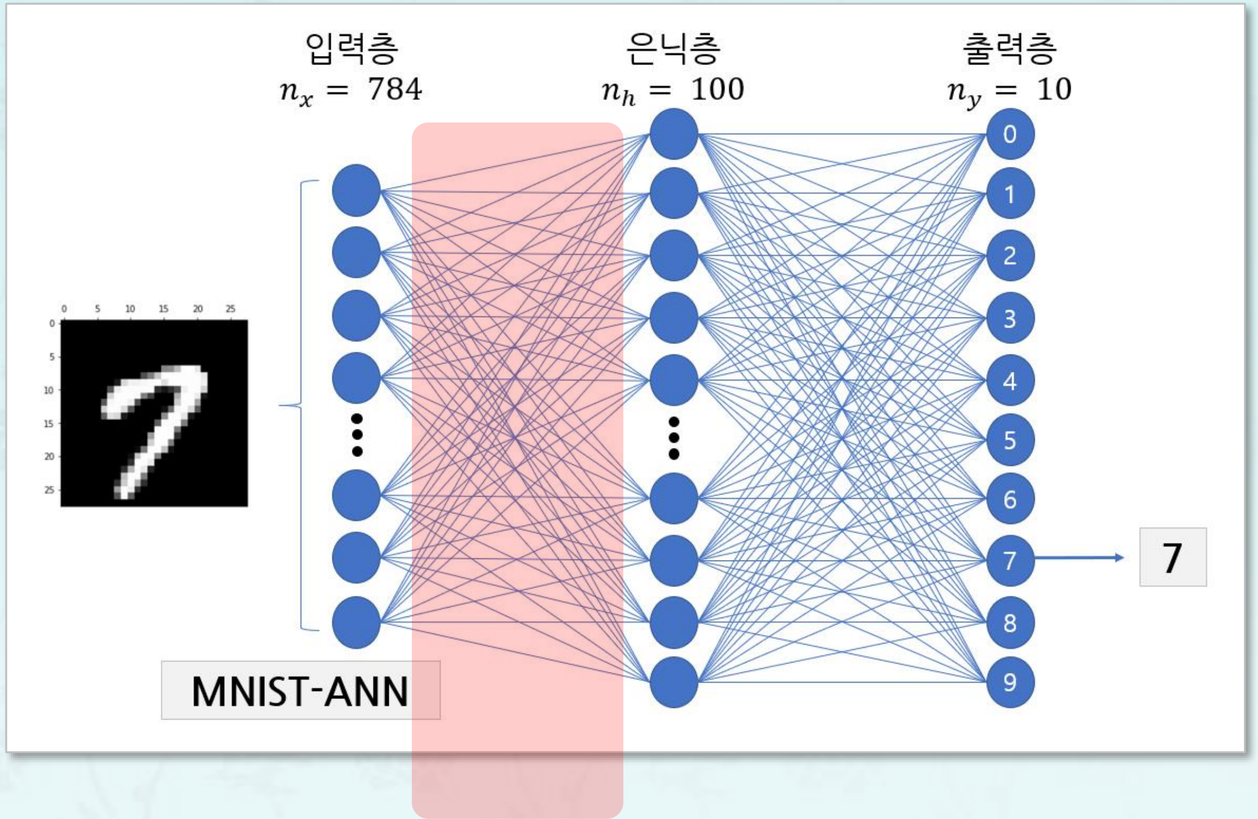
- **Reusing Weights**

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```



2. ANN Implementation:

- **Computation**

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

$$W^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & W_1^{(783)} & w_1^{(784)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & W_2^{(783)} & w_2^{(784)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{100}^{(1)} & w_{100}^{(2)} & \cdots & w_{100}^{(783)} & w_{100}^{(784)} \end{pmatrix}$$

2. ANN Implementation:

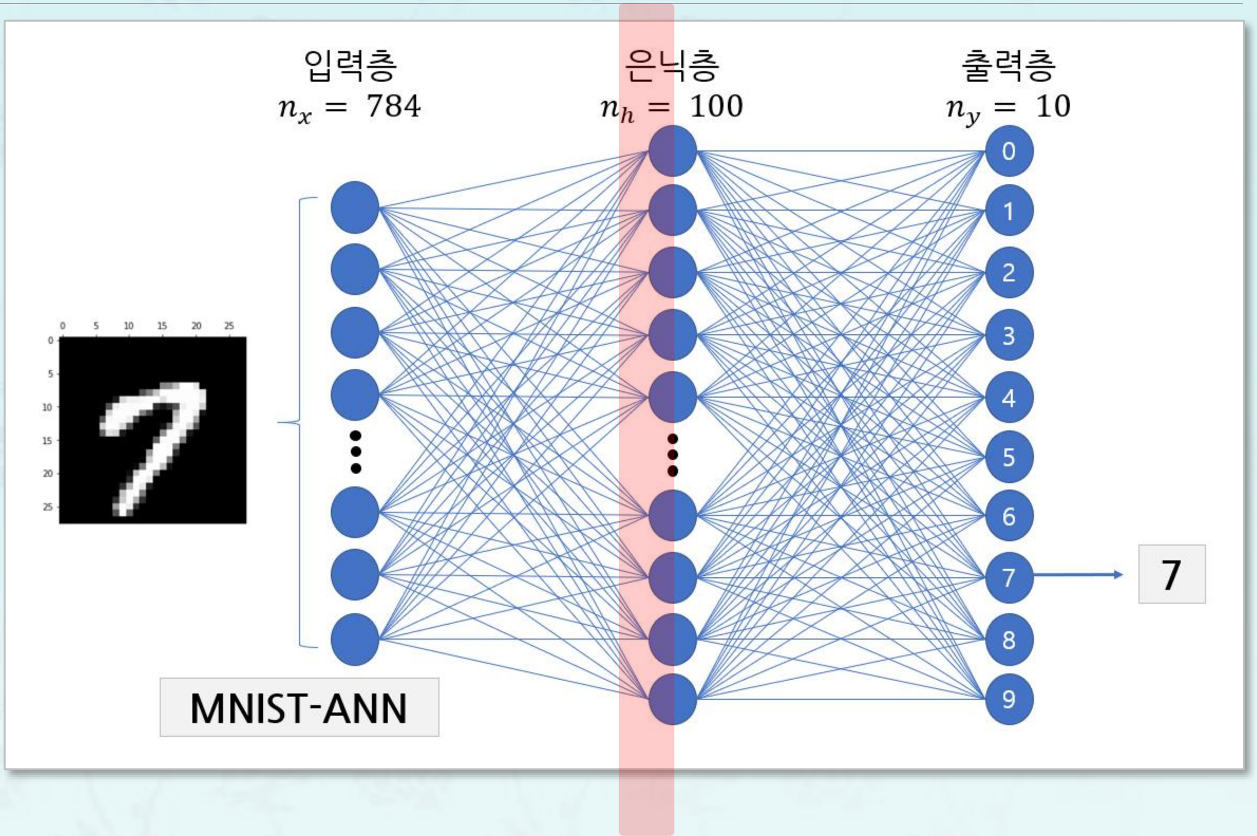
- **Computation**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```



2. ANN Implementation:

Hidden Layer Computation

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

$$W^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & W_1^{(783)} & w_1^{(784)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & W_2^{(783)} & w_2^{(784)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{100}^{(1)} & w_{100}^{(2)} & \cdots & w_{100}^{(783)} & w_{100}^{(784)} \end{pmatrix}$$

$$\mathbf{A}^{[1]} = sigmoid(\mathbf{Z}^{[1]})$$

2. ANN Implementation:

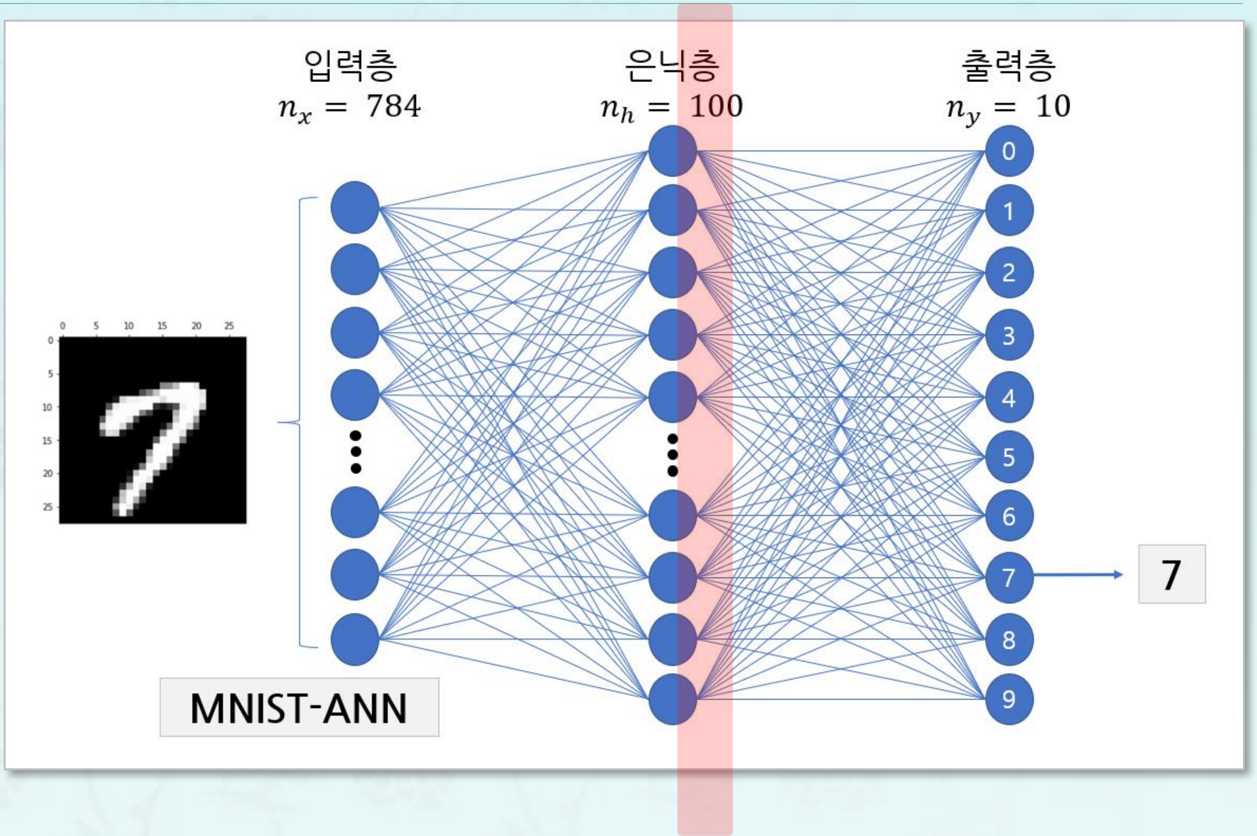
- **Hidden Layer Computation**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```



2. ANN Implementation:

▪ Reusing Weights

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

$$W^{[2]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & W_1^{(99)} & w_1^{(100)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & W_2^{(99)} & w_2^{(100)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{100}^{(1)} & w_{100}^{(2)} & \cdots & w_{100}^{(99)} & w_{100}^{(100)} \end{pmatrix}$$

2. ANN Implementation

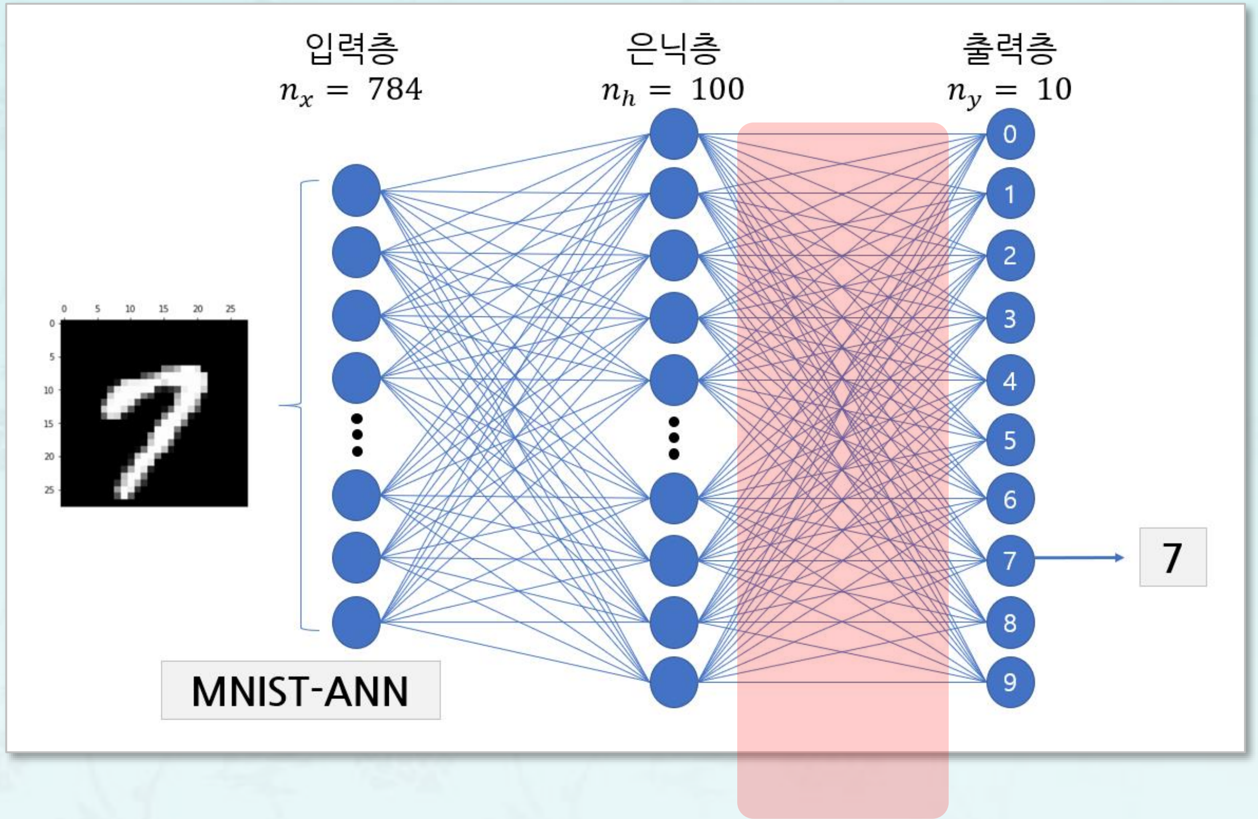
- **Reusing Weights**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```



2. ANN Implementation:

Hidden Layer Computation

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

$$W^{[2]}A^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & w_1^{(99)} & w_1^{(100)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & w_2^{(99)} & w_2^{(100)} \\ \vdots & \vdots & \vdots & \vdots & \\ w_{10}^{(1)} & w_{10}^{(2)} & \cdots & w_{10}^{(99)} & w_{10}^{(100)} \end{pmatrix} \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{99}^{(1)} \\ a_{100}^{(1)} \end{pmatrix}$$

2. ANN Implementation:

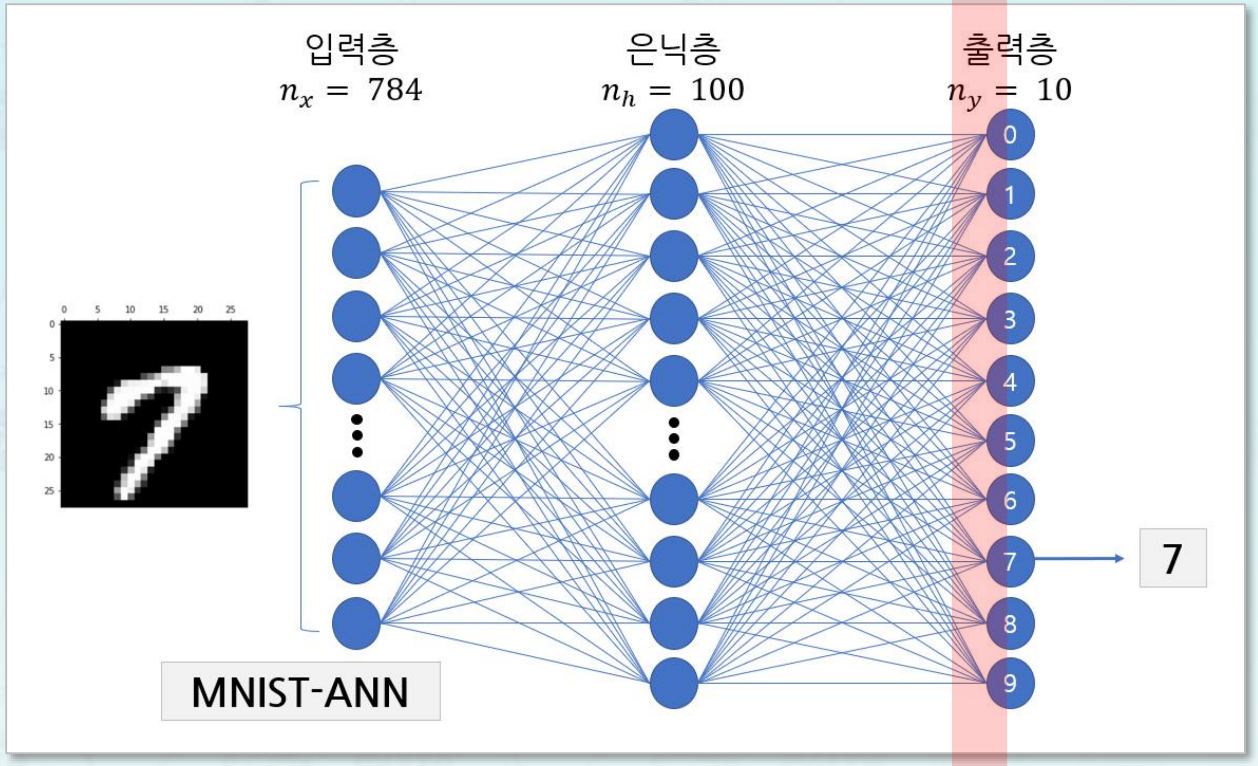
- **Hidden Layer Computation**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```



2. ANN Implementation:

- **Output Layer Computation**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

$$W^{[2]}A^{[1]} = \begin{pmatrix} w_1^{(1)} & w_1^{(2)} & \cdots & w_1^{(99)} & w_1^{(100)} \\ w_2^{(1)} & w_2^{(2)} & \cdots & w_2^{(99)} & w_2^{(100)} \\ \vdots & \vdots & \vdots & \vdots & \\ w_{10}^{(1)} & w_{10}^{(2)} & \cdots & w_{10}^{(99)} & w_{10}^{(100)} \end{pmatrix} \begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_{99}^{(1)} \\ a_{100}^{(1)} \end{pmatrix}$$

$$\mathbf{A}^{[2]} = sigmoid(\mathbf{Z}^{[2]})$$

2. ANN Implementation:

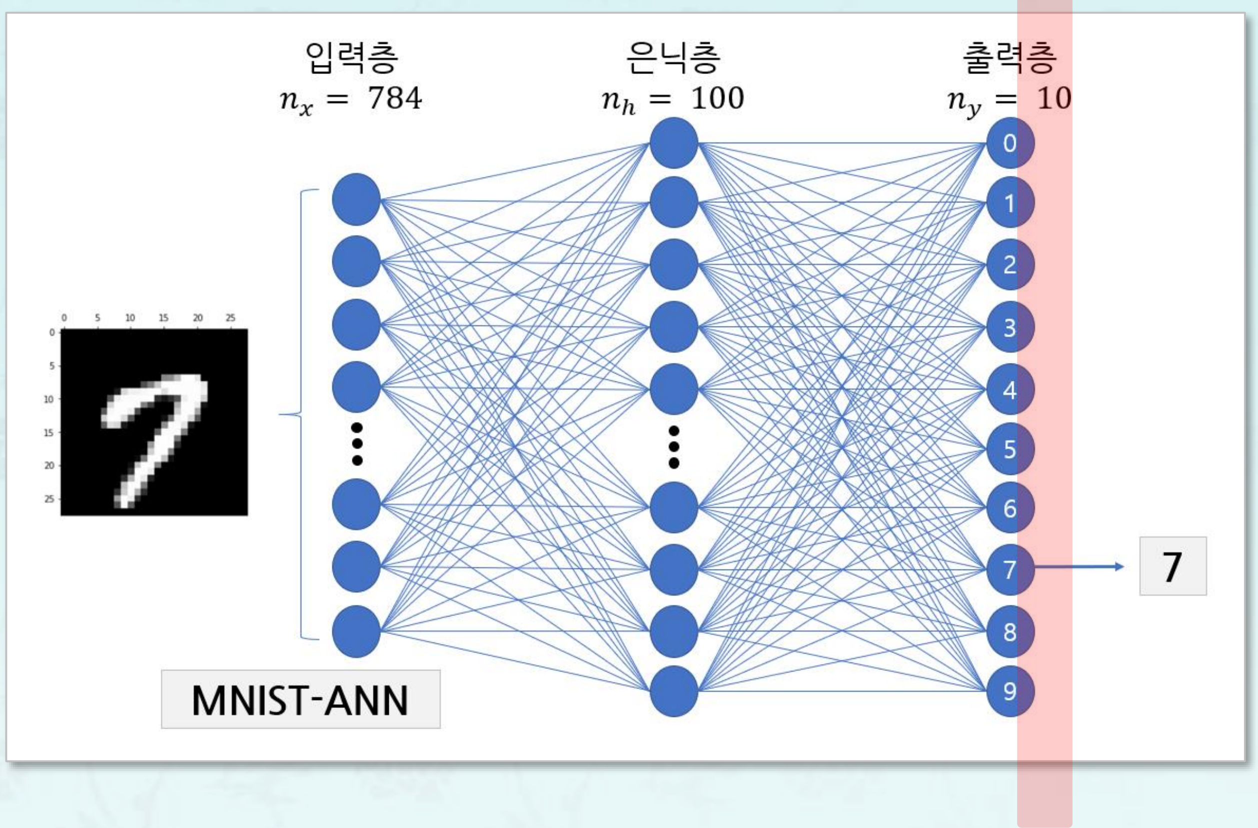
- **Output Layer Computation**

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```



2. ANN Implementation:

- **Prediction**

```python
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```python
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

```
image: 7
predict: [0.
          0.002
          0.001
          0.
          0.
          0.001
          0.
          0.979
          0.006
          0.003]
```

2. ANN Implementation:

- **Prediction**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

```
image: 7
predict: [0.
          0.002
          0.001
          0.
          0.
          0.001
          0.
          0.979
          0.006
          0.003]
```

2. ANN Implementation:

- **Prediction**

```
import joy
import numpy as np
g = lambda x : 1 / (1 + np.exp(-x))
```

```
(X, y) = joy.load_mnist_num(7)
W1 = joy.load_mnist_weight('data/w_xh.weights')
Z1 = np.dot(W1, X)
A1 = g(Z1)

W2 = joy.load_mnist_weight('data/w_hy.weights')
Z2 = np.dot(W2, A1)
yhat = g(Z2)

print('image:', y)
print('predict:', np.round_(yhat, 3))
```

```
image: 7
predict: [0.
          0.002
          0.001
          0.
          0.
          0.001
          0.
          0.979
          0.006
          0.003]
```

0
1
2
3
4
5
6
7
8
9

Feed-forward Neural Network - Example

- Summary
  - Understanding Feed-forward Neural Network – Example
  - Understanding Features from data
  - Feed-forward Neural Network - Example Modeling
  - Reusing Weights

- Next
  - 7-3 Adaline Gradient Descent

7주차(2/3)

Feed-forward Neural Network

Machine Learning with Python

Handong Global University
Prof. Youngsup Kim
idebtor@gmail.com

여러분 곁에 항상 열려 있는 K-MOOC 강의실에서 만나 뵙기를 바랍니다.