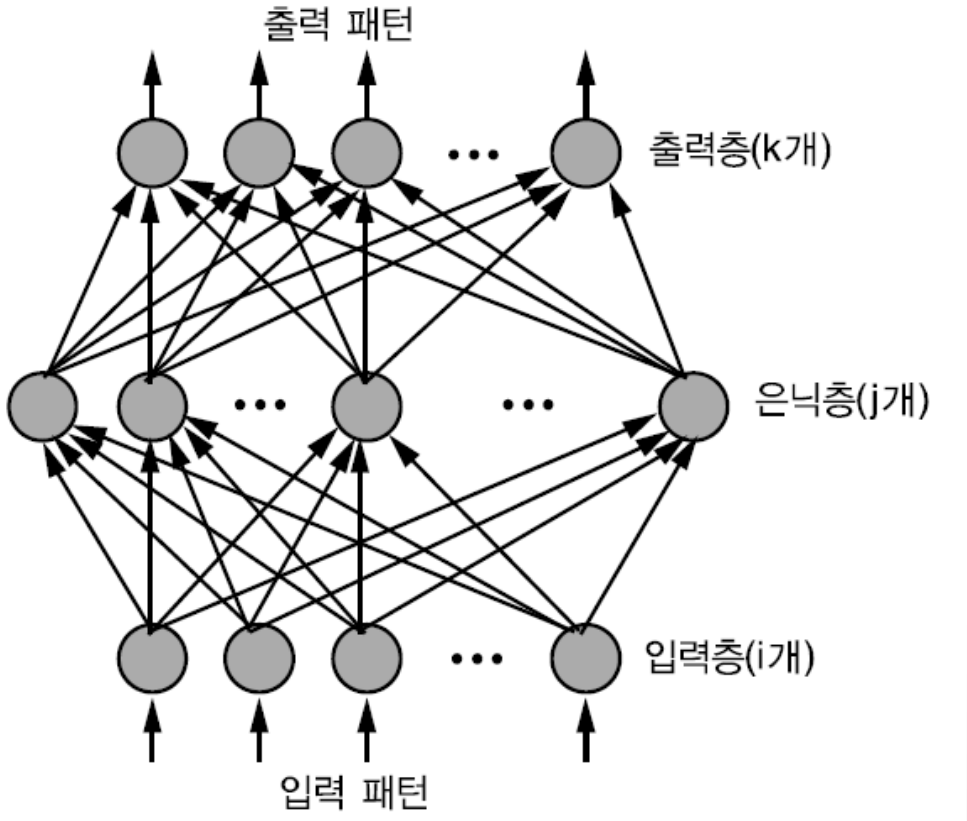
# Python

ANN

# ANN

- ANN

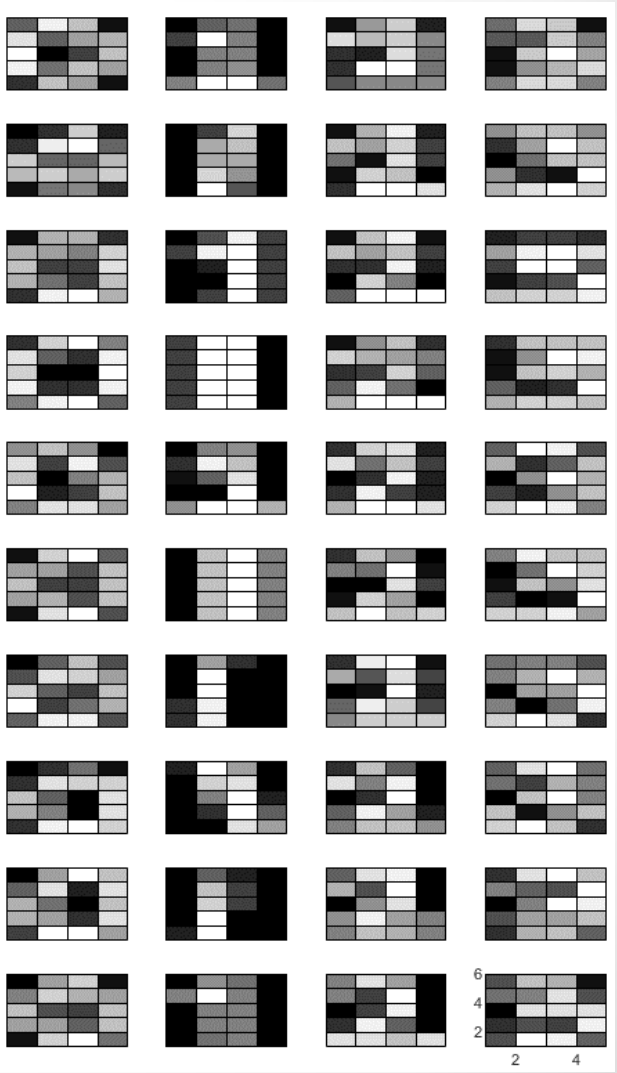


# ANN

- 입력 데이터

```
import numpy as np
import matplotlib.pyplot as plt

data = np.array([[0.1875, 0.75, 0.625, 0.0625, 0.9375, 0.4375, 0.75, 0.625, 1, 0, 0.25, 0.75, 0.875, 0.375, 0.625, 0
                 [0.5, 1, 1, 0.4375, 0, 0.5, 0.5625, 0, 0, 0.5, 0.5, 0, 0.25, 1, 0.5, 0, 0, 0.375, 0.4375, 0],
                 [0.3125, 0.5, 0.5, 0.5, 0.1875, 0.9375, 0.9375, 0.4375, 0.1875, 0.125, 0.8125, 0.4375, 0.8125, 0.68
                 [0.4375, 0.75, 0.75, 0.4375, 0.0625, 0.5, 0.625, 0.75, 0.0625, 0.6875, 0.875, 0.5625, 0.3125, 0.312
                 [0.0625, 0.4375, 0.5, 0.1875, 0.6875, 0.75, 0.625, 0.75, 0.75, 0.375, 0.375, 0.6875, 0.1875, 0.875,
                 [0, 0.75, 0.25, 0, 0, 0.625, 0.4375, 0, 0, 0.5, 0.5, 0, 0, 0.5, 0.5625, 0, 0, 0.1875, 0.625, 0],
                 [0.1875, 1, 1, 0.875, 0.0625, 0.8125, 0.75, 0, 0.4375, 0.0625, 0.875, 0.25, 0.75, 0.625, 0.8125, 0.
                 [0.6875, 0.875, 1, 0.8125, 0.5625, 0.1875, 0.0625, 1, 0, 0.4375, 0.75, 0.75, 0.1875, 0.625, 1, 0.75
                 [0.1875, 0.9375, 1, 0.6875, 0.6875, 0.4375, 0.25, 0.75, 0.75, 0.25, 0.25, 0.875, 0.6875, 0.625, 0.5
                 [0, 0.25, 1, 0.25, 0, 0, 1, 0.25, 0, 0.125, 1, 0.25, 0.25, 0.9375, 1, 0.25, 0, 0.3125, 0.9375, 0.25
                 [0.375, 1, 1, 1, 0, 0.8125, 0.5, 0, 0.1875, 0.1875, 0.9375, 0.125, 0.75, 0.625, 0.75, 0.25, 0.0625,
                 [0.75, 0.8125, 0.8125, 0.9375, 0.0625, 0.25, 0.3125, 1, 0.25, 1, 1, 0.375, 0.625, 0.9375, 1, 0.875
                 [0.5, 0.9375, 1, 0.375, 0.9375, 0.1875, 0.1875, 0.9375, 0.8125, 0, 0, 1, 0.875, 0.375, 0.1875, 0.93
                 [0.25, 1, 1, 0, 0.25, 1, 1, 0, 0.25, 1, 1, 0, 0.25, 1, 1, 0, 0.25, 1, 1, 0],
                 [0.6875, 1, 1, 1, 0.375, 0.9375, 0.4375, 0, 0.1875, 0.25, 0.8125, 0.375, 0.8125, 0.6875, 0.625, 0.5
                 [0.6875, 0.8125, 0.8125, 0.75, 0.375, 0.125, 0.1875, 1, 0.0625, 0.75, 0.8125, 0.6875, 0.0625, 0.562
                 [0.5, 0.875, 0.875, 0.625, 1, 0.125, 0.25, 0.75, 0.75, 0, 0.5, 0.6875, 0.875, 0.25, 0.9375, 0.3125
```



# ANN

- Activation Function

$$\frac{1}{1+\exp(-net)}$$

```
def sigmoidFunc(totalInput):
    return 1.0 / (np.ones(totalInput.shape) + np.exp(-1.0 * totalInput))
```

# ANN

- 출력층과 은닉층 간의 연결 강도의 변화

$$f'(net_{pk}) = O_{pk}(1 - O_{pk})$$

$$\delta_{pk} = (t_{pk} - O_{pk}) f'(net_{pk})$$

```
def outputDeltas(output, target):
    #detals, output, target은 모두 벡터
    sigmoidDeriv = output * (np.ones(output.shape) - output)
    return 2 * (target - output) * sigmoidDeriv
```

# ANN

- 은닉층과 입력층 간의 연결 강도의 변화

$$f'(net_{pj}) = O_{pj}(1 - O_{pj})$$

$$\delta_{pj} = f'(net_{pj}) \sum_{k} \delta_{pk} W_{kj}$$

```
def hiddenDeltas(outputDeltas, hiddenOutputs, outputWeights):
    # deltas, outputs은 열벡터.
    # output Weights는 hidden->output 가중치행렬(각 행은 출력 유닛의 가중치 벡터)
    sigmoidDeriv = hiddenOutputs * (np.ones(hiddenOutputs.shape) - hiddenOutputs)
    return (np.dot(outputWeights.T, outputDeltas.T)) * sigmoidDeriv
```

# ANN

- 파라미터 초기화

```
patterns = data.T
nPats = data.shape[0]
nTrainingPats = 20
nTestPats = 20
nInputs = data.shape[1]
nHidden = 10
nOutputs = 4

hiddenWeights = 0.5 * (np.random.rand(nHidden, nInputs+1) - np.ones((nHidden, nInputs+1)) * 0.5)
outputWeights = 0.5 * (np.random.rand(nOutputs, nHidden+1) - np.ones((nOutputs, nHidden+1)) * 0.5)

input = patterns
target = np.zeros((nOutputs, nPats))
classNum = 0
eta = 0.1
NEpochs = 1000

for pat in range(0, nPats, 1):
    target[classNum, pat] = 1
    classNum = classNum + 1
    if classNum >= nOutputs:
        classNum = 0

ErrorsLastNEpochs = np.zeros((1, NEpochs))
TestErrorsLastNEpochs = np.zeros((1, NEpochs))
```

# ANN

- 전방향 패스 & 역방향 패스

```
for epoch in range(0, NEpochs, 1):
    sumSqrError = 0.0
    sumSqrTestError = 0.0
    outputWGrad = np.zeros(outputWeights.shape)
    hiddenWGrad = np.zeros(hiddenWeights.shape)

    for pat in range(0, nTrainingPats, 1):
        # 전방향패스(pass)
        inp = np.hstack([input[:, pat], np.array([1])])
        hiddenNetInputs = np.dot(hiddenWeights, inp)
        hiddenStates = sigmoidFunc(hiddenNetInputs)
        hidStatesBias = np.hstack([hiddenStates, np.array([1])])
        outputNetInputs = np.dot(outputWeights, hidStatesBias)
        outputStates = sigmoidFunc(outputNetInputs)

        #역방향 패스(pass)
        targetStates = target[:, pat]
        error = outputStates - targetStates
        sumSqrError = sumSqrError + np.dot(error, error)
        outputDel = outputDeltas(outputStates, targetStates)
        outputWGrad = outputWGrad + np.dot(np.array([outputDel]).T, np.array([hidStatesBias]))
        hiddenDel = hiddenDeltas(outputDel, hidStatesBias, outputWeights)
        hiddenDelArray = np.array([hiddenDel])
        hiddenWGrad = hiddenWGrad + np.dot(hiddenDelArray[:, 0:nHidden].T, np.array([inp]))
```

# ANN

- 연결 가중치 갱신

```
outputWChange = eta * outputWGrad
outputWeights = outputWeights + outputWChange
hiddenWChange = eta * hiddenWGrad
hiddenWeights = hiddenWeights + hiddenWChange
```

# ANN

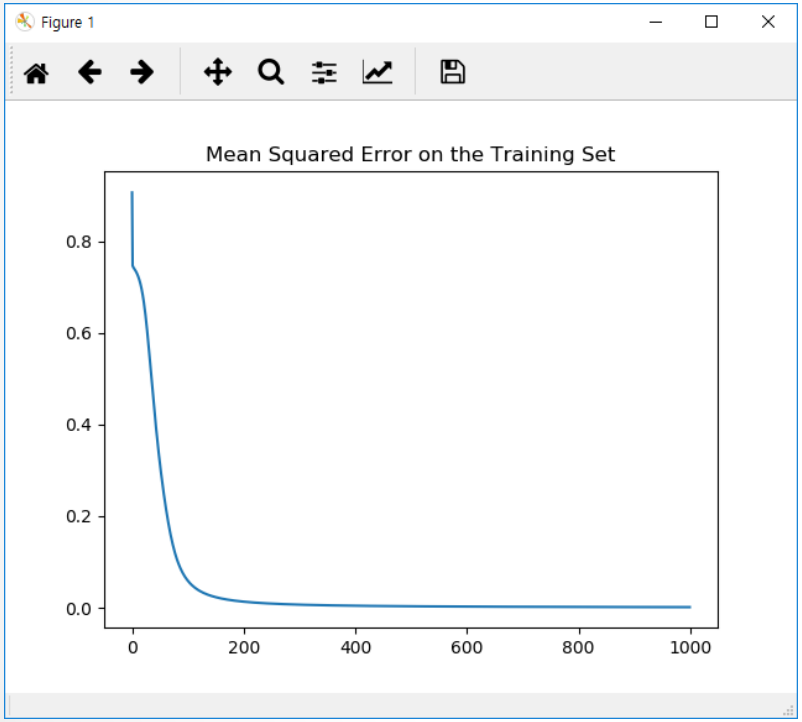


Figure 1
Mean Squared Error on the Training Set
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000

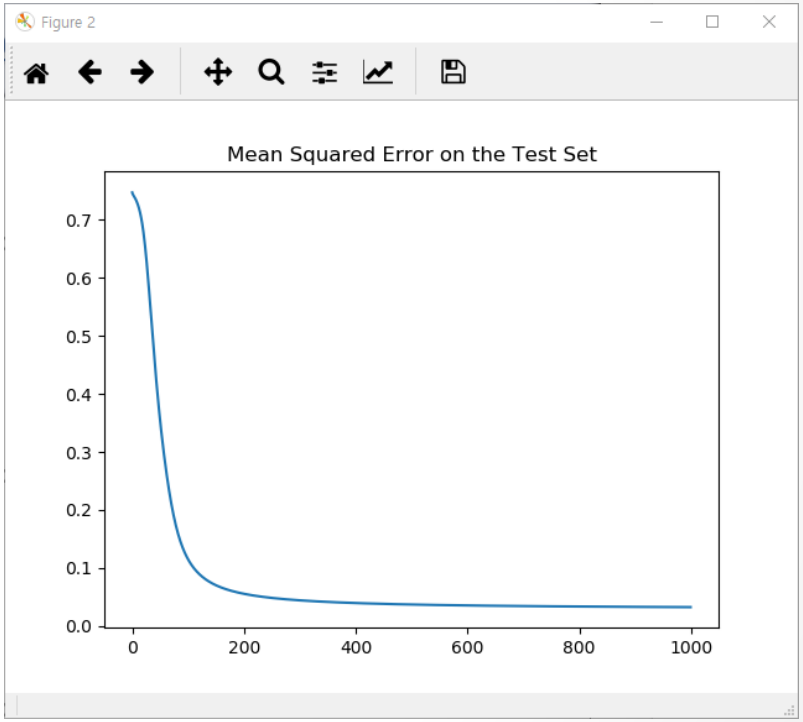


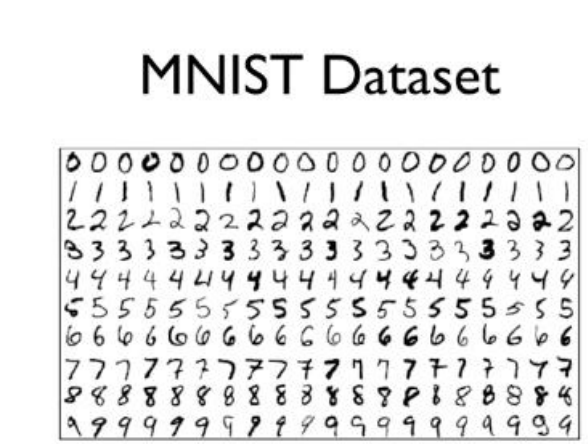
Figure 2
Mean Squared Error on the Test Set
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
200
400
600
800
1000

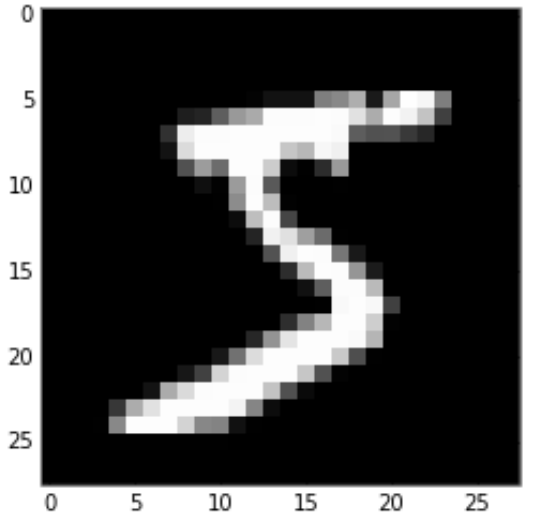
# ANN

- Exercise
  - 교재를 참고하여 성능 테스트와 결과를 출력하는 코드를 완성하시오.

# ANN

- MNIST 데이터셋
  - mnist?
    - 손글씨 숫자 이미지 집합
    - 0 ~ 9까지의 숫자 이미지로 구성
    - 훈련 이미지 60,000장(학습), 시험 이미지 10,000장(분류)



# ANN

```
import numpy as np
from mnist import load_mnist
from PIL import Image

def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True,
normalize=False)

img = x_train[0]
label = t_train[0]
print(label)  # 5

print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)

img_show(img)
```

# ANN

- Exercise
  - Mnist 데이터셋을 이용하여 신경망을 학습을 하시오.