13주차(2/3)

Deep Neural Network 1

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

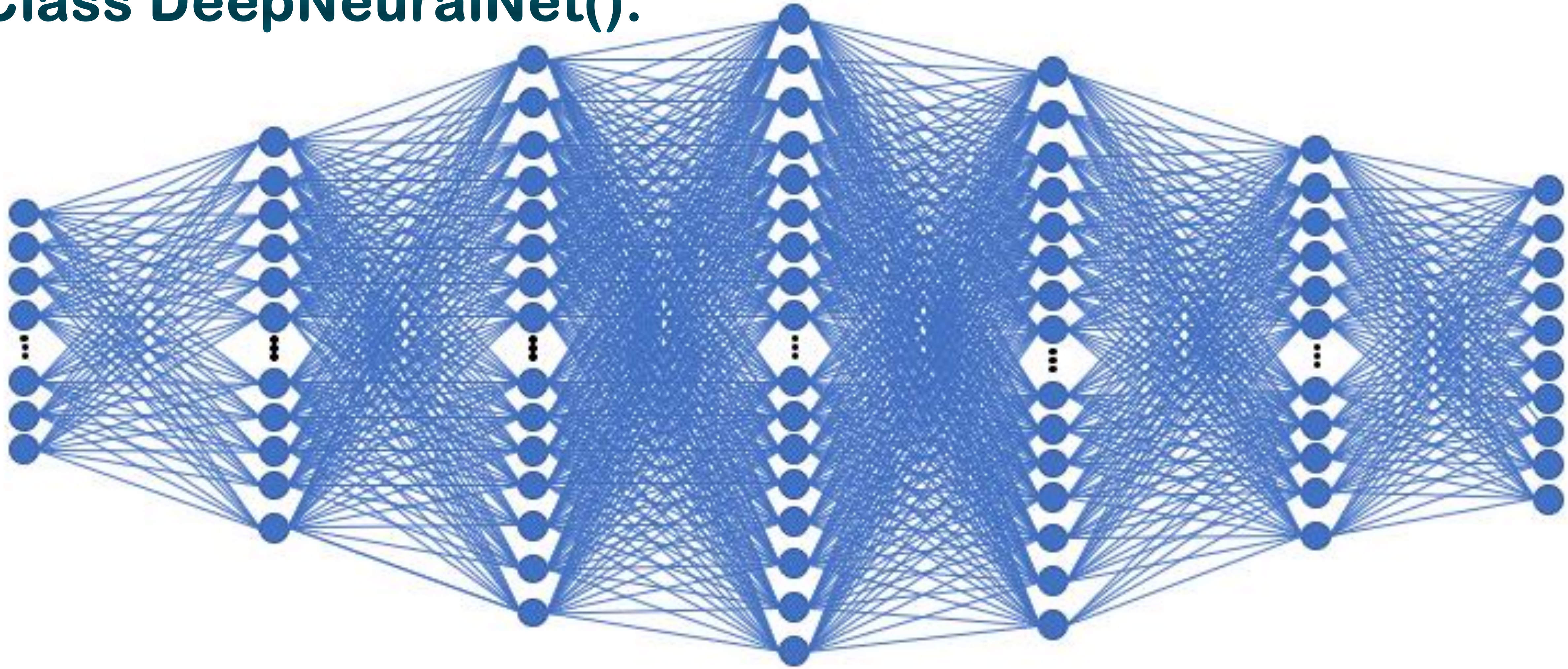
Deep Neural Network 1

- 학습 목표
  - **Deep Neural Network**(심층신경망)을 학습한다.
  - 은닉층의 개수에 따른 성능을 확인한다.

- 학습 내용
  - 심층신경망 이해하기
  - 심층신경망 구현하기
  - 심층신경망 성능확인하기

1. DNN 클래스

Class DeepNeuralNet():

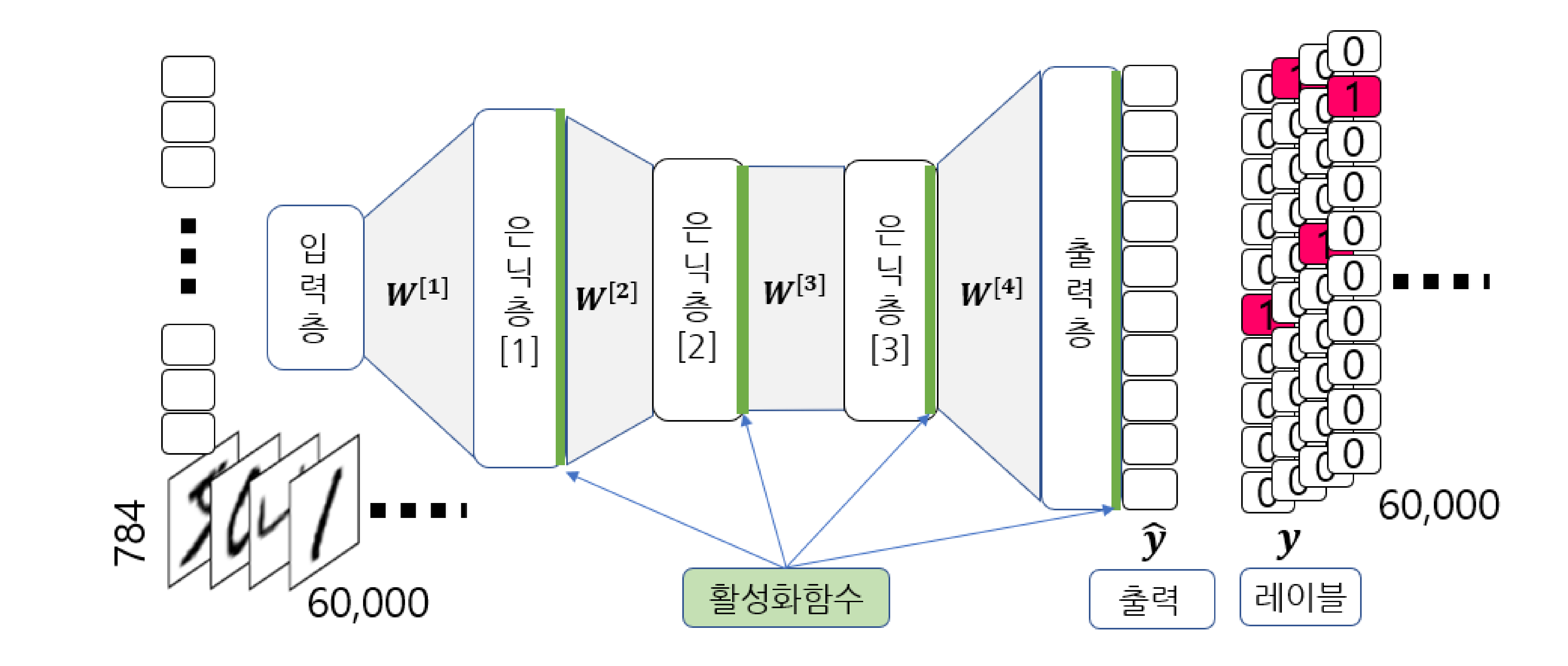


1. DNN 클래스

```python
class DeepNeuralNet():
    """ implements a deep neural net.
        Users may specify any number of layers.
        net_arch -- consists of a number of neurons in each layer
    """
    def __init__(self, net_arch, activate = None,
                      eta = 1.0, epochs = 100, random_seed = 1):
        pass

    def forpass(self, A0):
        pass

    def backprop(self, Z, A, Y):
        pass

    def fit(self, X, y):
        pass

    def predict(self, X):
        pass

    def evaluate(self, Xtest, ytest):
        pass
```

2. DNN 활성화 함수

Class DeepNeuralNet():



2. DNN 활성화 함수: tanh

$$tanh(x) = \frac{1 - e^{-x}}{1 + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1$$

```
def tanh(x):
    return (1.0 - np.exp(-2 * x))/(
            1.0 + np.exp(-2 * x))

def tanh_d(x):
    return (1 + tanh(x)) * (1 - tanh(x))

def sigmoid(x):
    #x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp((-x)))

def sigmoid_d(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return np.maximum(x, 0)

def relu_d(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

2. DNN 활성화 함수: tanh

$$tanh(x) = \frac{1-e^{-x}}{1+e^{-x}} = \frac{2}{1+e^{-2x}} - 1$$

$$\frac{d}{dx}\tanh(x) = 1 - \left(\frac{2}{1+e^{-2x}} - 1\right)^2$$
$$= 1 - \tanh^2(x)$$

```python
def tanh(x):
    return (1.0 - np.exp(-2 * x))/(
            1.0 + np.exp(-2 * x))

def tanh_d(x):
    return (1 + tanh(x)) * (1 - tanh(x))

def sigmoid(x):
    #x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp((-x)))

def sigmoid_d(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return np.maximum(x, 0)

def relu_d(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

2. DNN 활성화 함수: sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+\frac{1}{e^x}}$$

```python
def tanh(x):
    return (1.0 - np.exp(-2 * x))/(
            1.0 + np.exp(-2 * x))

def tanh_d(x):
    return (1 + tanh(x)) * (1 - tanh(x))

def sigmoid(x):
    #x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp((-x)))

def sigmoid_d(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return np.maximum(x, 0)

def relu_d(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

2. DNN 활성화 함수: sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{1}{1+\frac{1}{e^{x}}}$$

$$\frac{d}{dx}\sigma(x) = \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right)$$
$$= \sigma(x)(1-\sigma(x))$$

```python
def tanh(x):
    return (1.0 - np.exp(-2 * x))/(
            1.0 + np.exp(-2 * x))

def tanh_d(x):
    return (1 + tanh(x)) * (1 - tanh(x))

def sigmoid(x):
    #x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp((-x)))

def sigmoid_d(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return np.maximum(x, 0)

def relu_d(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

2. DNN 활성화 함수: ReLU

$$Relu(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & otherwise \end{cases}$$

```python
def tanh(x):
    return (1.0 - np.exp(-2 * x))/(
            1.0 + np.exp(-2 * x))

def tanh_d(x):
    return (1 + tanh(x)) * (1 - tanh(x))

def sigmoid(x):
    #x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp((-x)))

def sigmoid_d(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return np.maximum(x, 0)

def relu_d(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

2. DNN 활성화 함수: ReLU

$$Relu(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & otherwise \end{cases}$$

$$\frac{d}{dx} Relu(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & otherwise \end{cases}$$

```python
def tanh(x):
    return (1.0 - np.exp(-2 * x))/(
            1.0 + np.exp(-2 * x))

def tanh_d(x):
    return (1 + tanh(x)) * (1 - tanh(x))

def sigmoid(x):
    #x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp((-x)))

def sigmoid_d(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return np.maximum(x, 0)

def relu_d(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

2. DNN 활성화 함수

```python
def tanh(x):
    return (1.0 - np.exp(-2 * x))/(
                1.0 + np.exp(-2 * x))

def tanh_d(x):
    return (1 + tanh(x)) * (1 - tanh(x))

def sigmoid(x):
    #x = np.clip(x, -500, 500)
    return 1 / (1 + np.exp((-x)))

def sigmoid_d(x):
    return sigmoid(x) * (1 - sigmoid(x))

def relu(x):
    return np.maximum(x, 0)

def relu_d(x):
    x[x<=0] = 0
    x[x>0] = 1
    return x
```

3. DNN 구현: 생성자

```python
def __init__(self, net_arch, activate = None,
             eta = 1.0, epochs = 100, random_seed = 1):
    self.eta = eta
    self.epochs = epochs
    self.net_arch = net_arch
    self.layers = len(net_arch)
    self.W = []

    self.g       = [lambda x: sigmoid(x)   for _ in range(self.layers)]
    self.g_prime = [lambda x: sigmoid_d(x) for _ in range(self.layers)]

    if activate is not None:
        for i, (g, g_prime) in enumerate(zip(activate[::2], activate[1::2])):
            self.g[i+1] = g
            self.g_prime[i+1] = g_prime

    np.random.seed(random_seed)
    self.W = [[None]]    ## the first W0 is not used.
    for layer in range(self.layers - 1):
        w = 2 * np.random.rand(self.net_arch[layer+1],
                               self.net_arch[layer]) - 1
        self.W.append(w)
```

3. DNN 구현: 생성자

```python
def __init__(self, net_arch, activate = None,
             eta = 1.0, epochs = 100, random_seed = 1):
    self.eta = eta
    self.epochs = epochs
    self.net_arch = net_arch
    self.layers = len(net_arch)
    self.W = []

    self.g       = [lambda x: sigmoid(x)   for _ in range(self.layers)]
    self.g_prime = [lambda x: sigmoid_d(x) for _ in range(self.layers)]

    if activate is not None:
        for i, (g, g_prime) in enumerate(zip(activate[::2], activate[1::2])):
            self.g[i+1] = g
            self.g_prime[i+1] = g_prime

    np.random.seed(random_seed)
    self.W = [[None]]    ## the first W0 is not used.
    for layer in range(self.layers - 1):
        w = 2 * np.random.rand(self.net_arch[layer+1],
                               self.net_arch[layer]) - 1
        self.W.append(w)
```

3. DNN 구현: 생성자

```python
def __init__(self, net_arch, activate = None,
             eta = 1.0, epochs = 100, random_seed = 1):
    self.eta = eta
    self.epochs = epochs
    self.net_arch = net_arch
    self.layers = len(net_arch)
    self.W = []

    self.g       = [lambda x: sigmoid(x)   for _ in range(self.layers)]
    self.g_prime = [lambda x: sigmoid_d(x) for _ in range(self.layers)]

    if activate is not None:
        for i, (g, g_prime) in enumerate(zip(activate[::2], activate[1::2])):
            self.g[i+1] = g
            self.g_prime[i+1] = g_prime

    np.random.seed(random_seed)
    self.W = [[None]]    ## the first W0 is not used.
    for layer in range(self.layers - 1):
        w = 2 * np.random.rand(self.net_arch[layer+1],
                               self.net_arch[layer]) - 1
        self.W.append(w)
```

3. DNN 구현: 생성자

```python
def __init__(self, net_arch, activate = None,
                eta = 1.0, epochs = 100, random_seed = 1):
    self.eta = eta
    self.epochs = epochs
    self.net_arch = net_arch
    self.layers = len(net_arch)
    self.W = []

    self.g       = [lambda x: sigmoid(x)   for _ in range(self.layers)]
    self.g_prime = [lambda x: sigmoid_d(x) for _ in range(self.layers)]

    if activate is not None:
        for i, (g, g_prime) in enumerate(zip(activate[::2], activate[1::2])):
            self.g[i+1] = g
            self.g_prime[i+1] = g_prime

    np.random.seed(random_seed)
    self.W = [[None]]    ## the first W0 is not used.
    for layer in range(self.layers - 1):
        w = 2 * np.random.rand(self.net_arch[layer+1],
                               self.net_arch[layer]) - 1
        self.W.append(w)
```

3. DNN 구현: 생성자

```python
def __init__(self, net_arch, activate = None,
             eta = 1.0, epochs = 100, random_seed = 1):
    self.eta = eta
    self.epochs = epochs
    self.net_arch = net_arch
    self.layers = len(net_arch)
    self.W = []

    self.g       = [lambda x: sigmoid(x)   for _ in range(self.layers)]
    self.g_prime = [lambda x: sigmoid_d(x) for _ in range(self.layers)]

    if activate is not None:
        for i, (g, g_prime) in enumerate(zip(activate[::2], activate[1::2])):
            self.g[i+1] = g
            self.g_prime[i+1] = g_prime

    np.random.seed(random_seed)
    self.W = [[None]]    ## the first W0 is not used.
    for layer in range(self.layers - 1):
        w = 2 * np.random.rand(self.net_arch[layer+1],
                               self.net_arch[layer]) - 1
        self.W.append(w)
```

(뒷층 노드 수, 앞층 노드 수)

3. DNN 구현: fit() 메소드

```
def fit(self, X, y):
    self.cost_ = []
    for epoch in range(self.epochs):
        Z, A = self.forpass(X)
        cost = self.backprop(Z, A, y)
        self.cost_.append(np.sqrt(
            np.sum(cost * cost)))
    return self
```

3. DNN 구현: fit() 메소드

```python
def fit(self, X, y):
    self.cost_ = []
    for epoch in range(self.epochs):
        Z, A = self.forpass(X)
        cost = self.backprop(Z, A, y)
        self.cost_.append(np.sqrt(
            np.sum(cost * cost)))
    return self
```

3. DNN 구현: 순전파

```python
def forpass(self, A0):
    Z = [[None]] # Z0 is not used.
    A = []       # A0 = X0 is used.
    A.append(A0)
    for i in range(1, len(self.W)):
        z = np.dot(self.W[i], A[i-1])
        Z.append(z)
        a = self.g[i](z)
        A.append(a)
    return Z, A
```

```python
def fit(self, X, y):
    self.cost_ = []
    for epoch in range(self.epochs):
        Z, A = self.forpass(X)
        cost = self.backprop(Z, A, y)
        self.cost_.append(np.sqrt(
            np.sum(cost * cost)))
    return self
```

$\boldsymbol{X \longrightarrow A^{[0]}}$

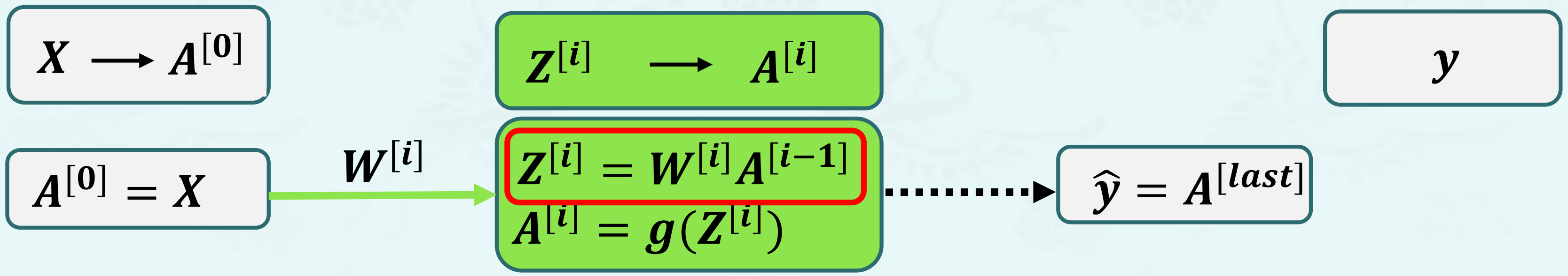
$\boldsymbol{Z^{[i]} \longrightarrow A^{[i]}}$

$\boldsymbol{y}$

$\boldsymbol{A^{[0]} = X} \xrightarrow{\boldsymbol{W^{[i]}}}$ $\boldsymbol{Z^{[i]} = W^{[i]} A^{[i-1]}}$, $\boldsymbol{A^{[i]} = g(Z^{[i]})}$ $\longrightarrow$ $\boldsymbol{\hat{y} = A^{[last]}}$

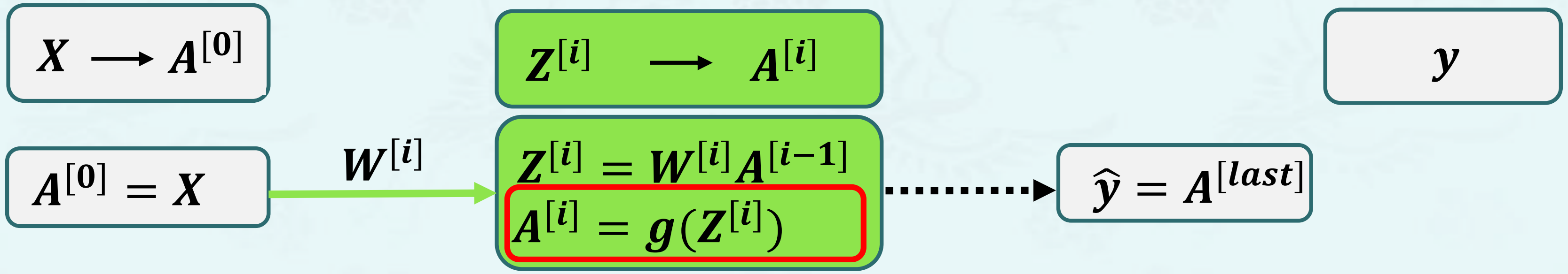
3. DNN 구현: 순전파

```python
def forpass(self, A0):
    Z = [[None]] # Z0 is not used.
    A = []         # A0 = X0 is used.
    A.append(A0)
    for i in range(1, len(self.W)):
        z = np.dot(self.W[i], A[i-1])
        Z.append(z)
        a = self.g[i](z)
        A.append(a)
    return Z, A
```



3. DNN 구현: 순전파

```python
def forpass(self, A0):
    Z = [[None]] # Z0 is not used.
    A = []       # A0 = X0 is used.
    A.append(A0)
    for i in range(1, len(self.W)):
        z = np.dot(self.W[i], A[i-1])
        Z.append(z)
        a = self.g[i](z)
        A.append(a)
    return Z, A
```



3. DNN 구현: fit() 메소드

```python
def fit(self, X, y):
    self.cost_ = []
    for epoch in range(self.epochs):
        Z, A = self.forpass(X)
        cost = self.backprop(Z, A, y)
        self.cost_.append(np.sqrt(
            np.sum(cost * cost)))
    return self
```

3. DNN 구현: 역전파

```python
def backprop(self, Z, A, Y):
    E  = [None for x in range(self.layers)]
    dZ = [None for x in range(self.layers)]

    ll = self.layers - 1
    error = Y - A[ll]
    E[ll] = error
    dZ[ll] = error * self.g_prime[ll](Z[ll])

    for i in range(self.layers-2, 0, -1):
        E[i]  = np.dot(self.W[i+1].T, E[i+1])
        dZ[i] = E[i] * self.g_prime[i](Z[i])

    m = Y.shape[0] # number of samples
    for i in range(ll, 0, -1):
        self.W[i] += self.eta * np.dot(dZ[i], A[i-1].T) / m
    return error
```

y

$Z^{[i]} = W^{[i]} A^{[i-1]}$

$A^{[i]} = g(Z^{[i]})$

$\hat{y} = A^{[last]}$

$W^{[last]} \mathrel{+}= dZ^{[last]} \cdot A^{[last-1]T}$

$W^{[i]} \mathrel{+}= dZ^{[i]} \cdot A^{[i-1]T}$

$E^{[last]} = y - \hat{y}$

$dZ^{[last]} = E^{[last]} g'(Z^{[last]})$

$E^{[i]} = W^{[i+1]T} E^{[i+1]}$

$dZ^{[i]} = E^{[i]} g'(Z^{[i]})$

3. DNN 구현: 역전파

```python
def backprop(self, Z, A, Y):
    E  = [None for x in range(self.layers)]
    dZ = [None for x in range(self.layers)]

    ll = self.layers - 1
    error = Y - A[ll]
    E[ll] = error
    dZ[ll] = error * self.g_prime[ll](Z[ll])

    for i in range(self.layers-2, 0, -1):
        E[i]  = np.dot(self.W[i+1].T, E[i+1])
        dZ[i] = E[i] * self.g_prime[i](Z[i])

    m = Y.shape[0] # number of samples
    for i in range(ll, 0, -1):
        self.W[i] += self.eta * np.dot(dZ[i], A[i-1].T) / m
    return error
```

```python
def fit(self, X, y):
    self.cost_ = []
    for epoch in range(self.epochs):
        Z, A = self.forpass(X)
        cost = self.backprop(Z, A, y)
        self.cost_.append(np.sqrt(np.sum(cost * cost)))
    return self
```

$$y$$

$$Z^{[i]} = W^{[i]} A^{[i-1]}$$
$$A^{[i]} = g(Z^{[i]})$$

$$\hat{y} = A^{[last]}$$

$$W^{[last]} += dZ^{[last]} \cdot A^{[last-1]T}$$

$$W^{[i]} += dZ^{[i]} \cdot A^{[i-1]T}$$

$$E^{[last]} = y - \hat{y}$$
$$dZ^{[last]} = E^{[last]} g'(Z^{[last]})$$

$$E^{[i]} = W^{[i+1]T} E^{[i+1]}$$
$$dZ^{[i]} = E^{[i]} g'(Z^{[i]})$$

3. DNN 구현: 역전파

```python
def backprop(self, Z, A, Y):
    E  = [None for x in range(self.layers)]
    dZ = [None for x in range(self.layers)]

    ll = self.layers - 1
    error = Y - A[ll]
    E[ll] = error
    dZ[ll] = error * self.g_prime[ll](Z[ll])

    for i in range(self.layers-2, 0, -1):
        E[i]  = np.dot(self.W[i+1].T, E[i+1])
        dZ[i] = E[i] * self.g_prime[i](Z[i])

    m = Y.shape[0] # number of samples
    for i in range(ll, 0, -1):
        self.W[i] += self.eta * np.dot(dZ[i], A[i-1].T) / m
    return error
```

$$y$$

$$Z^{[i]} = W^{[i]} A^{[i-1]}$$
$$A^{[i]} = g(Z^{[i]})$$

$$\hat{y} = A^{[last]}$$

$$W^{[last]} \mathrel{+}= dZ^{[last]} \cdot A^{[last-1]T}$$

$$W^{[i]} \mathrel{+}= dZ^{[i]} \cdot A^{[i-1]T}$$

$$E^{[last]} = y - \hat{y}$$
$$dZ^{[last]} = E^{[last]} g'(Z^{[last]})$$

$$E^{[i]} = W^{[i+1]T} E^{[i+1]}$$
$$dZ^{[i]} = E^{[i]} g'(Z^{[i]})$$

3. DNN 구현: 역전파

```python
def backprop(self, Z, A, Y):
    E  = [None for x in range(self.layers)]
    dZ = [None for x in range(self.layers)]

    ll = self.layers - 1
    error = Y - A[ll]
    E[ll] = error
    dZ[ll] = error * self.g_prime[ll](Z[ll])

    for i in range(self.layers-2, 0, -1):
        E[i]  = np.dot(self.W[i+1].T, E[i+1])
        dZ[i] = E[i] * self.g_prime[i](Z[i])

    m = Y.shape[0] # number of samples
    for i in range(ll, 0, -1):
        self.W[i] += self.eta * np.dot(dZ[i], A[i-1].T) / m
    return error
```

$$y$$

$$Z^{[i]} = W^{[i]} A^{[i-1]}$$
$$A^{[i]} = g(Z^{[i]})$$

$$\hat{y} = A^{[last]}$$

$$W^{[last]} += dZ^{[last]} \cdot A^{[last-1]T}$$

$$W^{[i]} += dZ^{[i]} \cdot A^{[i-1]T}$$

$$E^{[last]} = y - \hat{y}$$
$$dZ^{[last]} = E^{[last]} g'(Z^{[last]})$$

$$E^{[i]} = W^{[i+1]T} E^{[i+1]}$$
$$dZ^{[i]} = E^{[i]} g'(Z^{[i]})$$

3. DNN 구현: fit() 메소드

```python
def fit(self, X, y):
    self.cost_ = []
    for epoch in range(self.epochs):
        Z, A = self.forpass(X)
        cost = self.backprop(Z, A, y)
        self.cost_.append(np.sqrt(
            np.sum(cost * cost)))
    return self
```

4. DNN 학습 결과: XOR 테스트

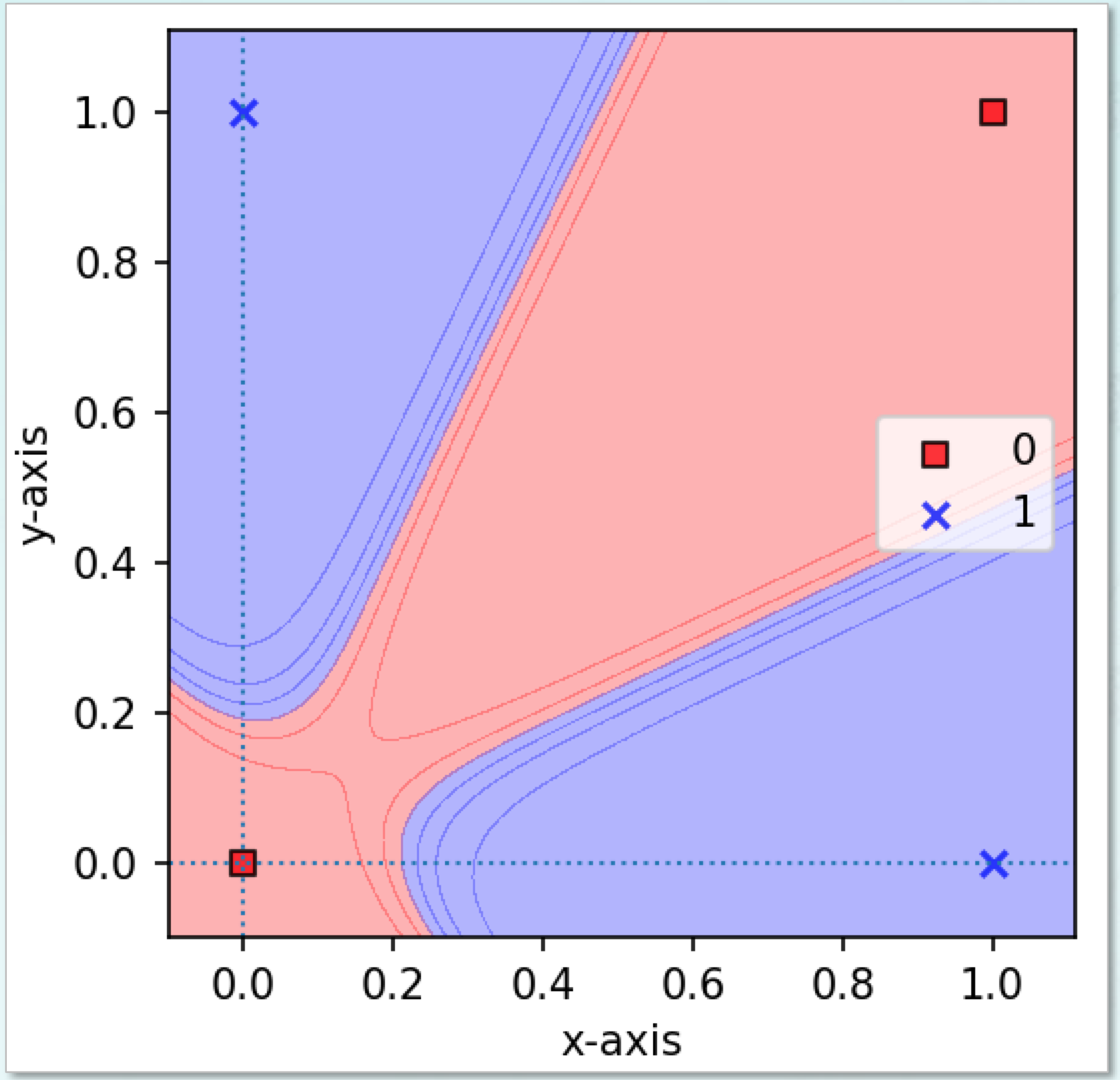
```python
import joy
X = np.array([ [0, 0, 1, 1], [0, 1, 0, 1] ])
y = np.array([0, 1, 1, 0])
dnn = DeepNeuralNet([2, 4, 2, 1], eta = 0.5, epochs = 5000).fit(X, y)

joy.plot_decision_regions(X.T, y, dnn)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend(loc='best')
plt.show()
```

[입력층, 은닉층, 은닉층, 출력층]의 노드 수

4. DNN 학습 결과: XOR 테스트

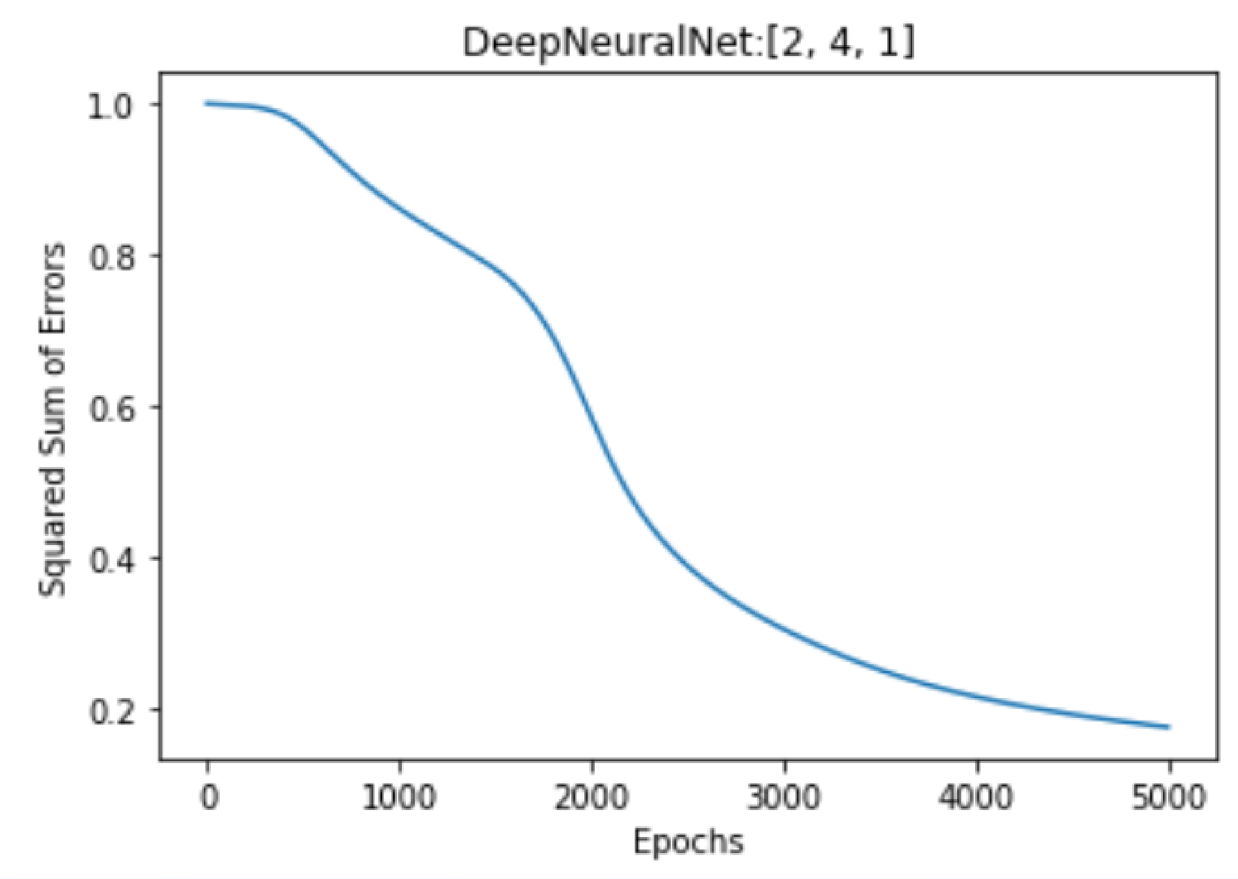
```python
import joy
X = np.array([ [0, 0, 1, 1], [0, 1, 0, 1] ])
y = np.array([0, 1, 1, 0])
dnn = DeepNeuralNet([2, 4, 2, 1], eta = 0.5, epochs = 5000).fit(X, y)

joy.plot_decision_regions(X.T, y, dnn)
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend(loc='best')
plt.show()
```

4. DNN 학습 결과: XOR 테스트



4. DNN 학습 결과: XOR 테스트

```
dnn = DeepNeuralNet([2, 4, 1],
                    eta = 0.5, epochs = 5000).fit(X, y)
plt.plot(range(len(dnn.cost_)), dnn.cost_)
plt.xlabel('Epochs')
plt.ylabel('Squared Sum of Errors')
plt.title('DeepNeuralNet:{}'.format(dnn.net_a
plt.show()
```



4. DNN 학습 결과: XOR 테스트

```
dnn1 = DeepNeuralNet([2,4,1], eta = 0.5, epochs = 5000).fit(X, y)
g = [sigmoid, sigmoid_d, sigmoid, sigmoid_d, sigmoid, sigmoid_d]
dnn2 = DeepNeuralNet([2,4,2,1], activate=g, eta = 0.5, epochs = 5000).fit(X, y)
plt.plot(range(len(dnn1.cost_)), dnn1.cost_, label='{}'.format(dnn1.net_arch))
plt.plot(range(len(dnn2.cost_)), dnn2.cost_, label='{}'.format(dnn2.net_arch))
plt.title('DeepNeuralNet:{} vs {}'.format(dnn1.net_arch, dnn2.net_arch))
plt.xlabel('Epochs')
plt.ylabel('Squared Sum of Errors')
plt.legend(loc='best')
plt.show()
```

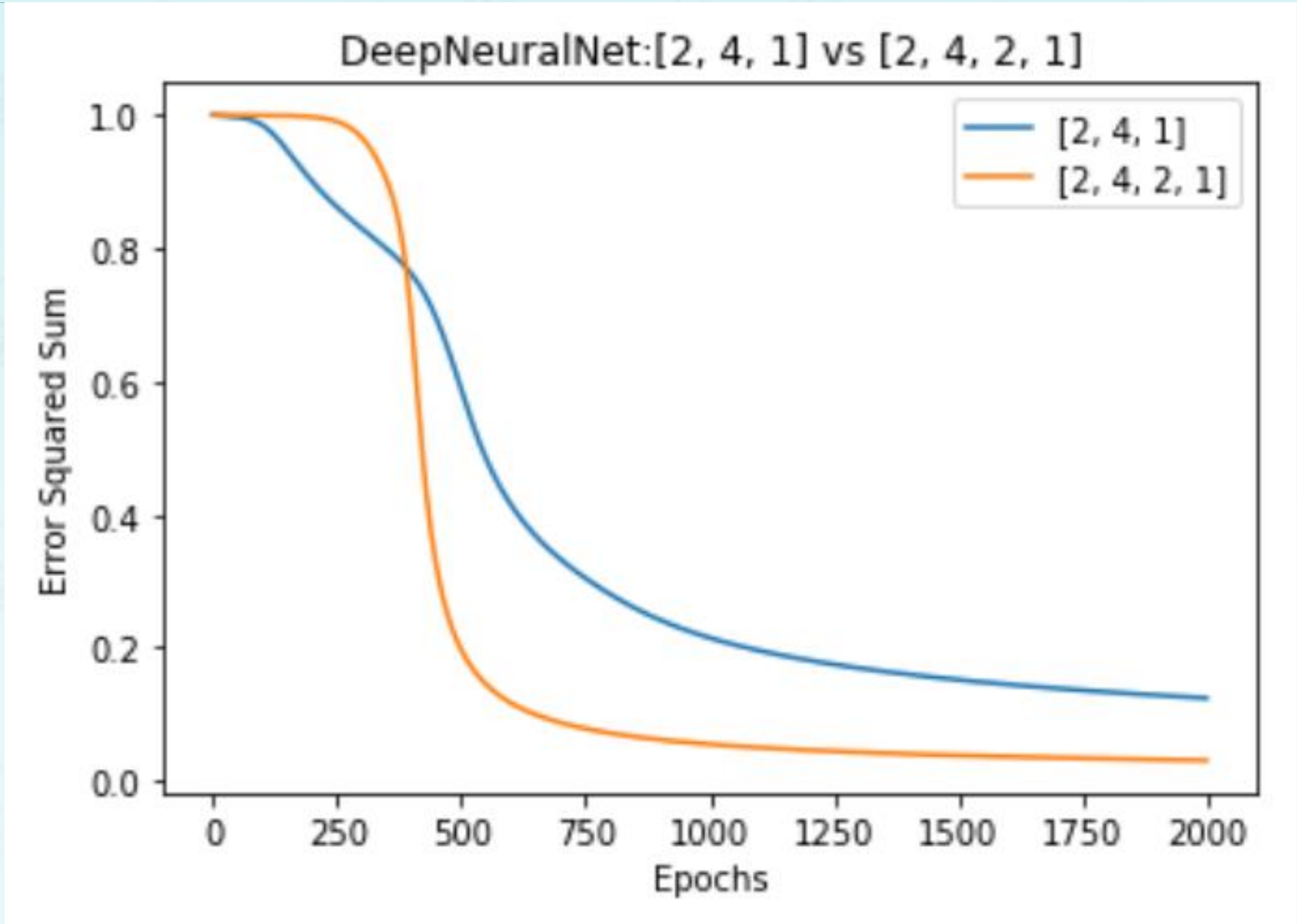
4. DNN 학습 결과: XOR 테스트

```
dnn1 = DeepNeuralNet([2,4,1], eta = 0.5, epochs = 5000).fit(X, y)
g = [sigmoid, sigmoid_d, sigmoid, sigmoid_d, sigmoid, sigmoid_d]
dnn2 = DeepNeuralNet([2,4,2,1], activate=g, eta = 0.5, epochs = 5000).fit(X, y)
plt.plot(range(len(dnn1.cost_)), dnn1.cost_, label='{}'.format(dnn1.net_arch))
plt.plot(range(len(dnn2.cost_)), dnn2.cost_, label='{}'.format(dnn2.net_arch))
plt.title('DeepNeuralNet:{} vs {}'.format(dnn1.net_arch, dnn2.net_arch))
plt.xlabel('Epochs')
plt.ylabel('Squared Sum of Errors')
plt.legend(loc='best')
plt.show()
```

4. DNN 학습 결과: XOR 테스트

```
dnn1 = DeepNeuralNet([2,4,1], eta = 0.5, epochs = 5000).fit(X, y)
g = [sigmoid, sigmoid_d, sigmoid, sigmoid_d, sigmoid, sigmoid_d]
dnn2 = DeepNeuralNet([2,4,2,1], activate=g, eta = 0.5, epochs = 5000).fit(X, y)
plt.plot(range(len(dnn1.cost_)), dnn1.cost_, label='{}'.format(dnn1.net_arch))
plt.plot(range(len(dnn2.cost_)), dnn2.cost_, label='{}'.format(dnn2.net_arch))
plt.title('DeepNeuralNet:{} vs {}'.format(dnn1.net_arch, dnn2.net_arch))
plt.xlabel('Epochs')
plt.ylabel('Squared Sum of Errors')
plt.legend(loc='best')
plt.show()
```

4. DNN 학습 결과: XOR 테스트

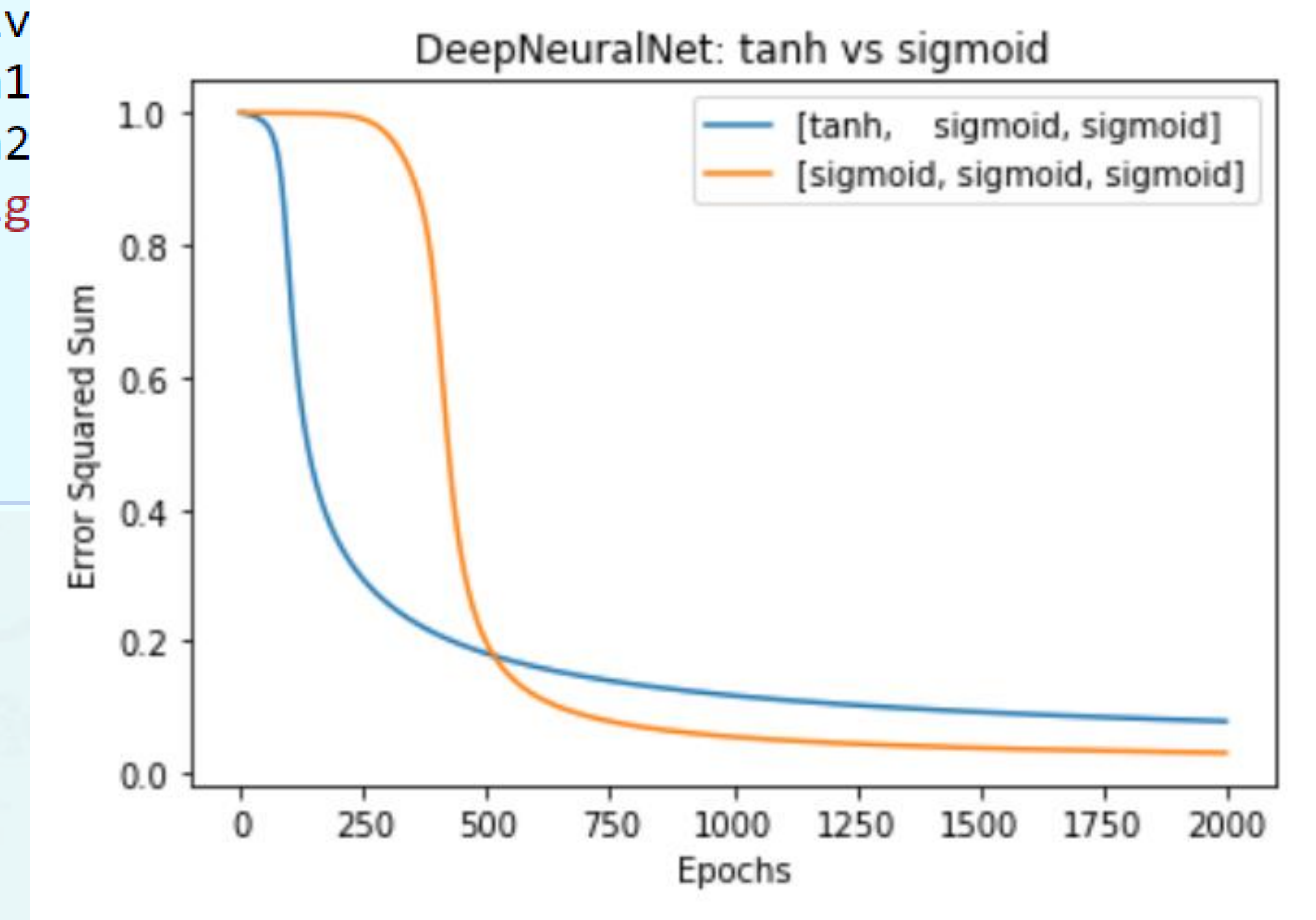


DNN 학습: 각 층별로 활성화 함수 지정

```
g1 = [tanh, tanh_d, sigmoid, sigmoid_d, sigmoid, sigmoid_d]
dnn1 = DeepNeuralNet([2,4,2,1], activate=g1, eta = 0.5, epochs = 5000).fit(X, y)
g2 = [sigmoid, sigmoid_d, sigmoid, sigmoid_d, sigmoid, sigmoid_d]
dnn2 = DeepNeuralNet([2,4,2,1], activate=g2, eta = 0.5, epochs = 5000).fit(X, y)
plt.plot(range(len(dnn1.cost_)), dnn1.cost_, label='[tanh,    sigmoid, sigmoid]')
plt.plot(range(len(dnn2.cost_)), dnn2.cost_, label='[sigmoid, sigmoid, sigmoid]')
plt.title('DeepNeuralNet: tanh vs sigmoid')
plt.xlabel('Epochs')
plt.ylabel('Squared Sum of Errors')
plt.legend(loc='best')
plt.show()
```

DNN 학습: 각 층별로 활성화 함수 지정

```
g1 = [tanh, tanh_d, sigmoid, sigmoid_d, sigmoid, sigmoid_d]
dnn1 = DeepNeuralNet([2,4,2,1], activate=g1, eta = 0.5, epochs = 5000).fit(X, y)
g2 = [sigmoid, sigmoid_d, sigmoid, sigmoid_d, sigmoid, sigmoid_d]
dnn2 = DeepNeuralNet([2,4,2,1], activ
plt.plot(range(len(dnn1.cost_)), dnn1
plt.plot(range(len(dnn2.cost_)), dnn2
plt.title('DeepNeuralNet: tanh vs sig
plt.xlabel('Epochs')
plt.ylabel('Squared Sum of Errors')
plt.legend(loc='best')
plt.show()
```



Deep Neural Network 1

- 학습 정리
  - 심층 신경망인 **DeepNeuralNet** 클래스 구현하기
  - 심층 신경망의 은닉층 갯수에 따른 성능을 확인하기

- **13-1 Deep Neural Network 2**

13주차(2/3)

Deep Neural Network 1

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

$X \longrightarrow A^{[0]}$

$A^{[0]} = X$

$Z^{[i-1]} \longrightarrow A^{[i-1]}$

$$Z^{[i-1]} = W^{[i-1]} A^{[i-2]}$$
$$A^{[i-1]} = g(Z^{[i-1]})$$

$W^{[i]}$

$Z^{[i]} \longrightarrow A^{[i]}$

$$Z^{[i]} = W^{[i]} A^{[i-1]}$$
$$A^{[i]} = g(Z^{[i]})$$

$\hat{y} = A^{[last]}$

y

$$E^{[last]} = y - \hat{y}$$
$$dZ^{[last]} = E^{[last]} g'(Z^{[last]})$$

$$W^{[last]} += dZ^{[last]} \cdot A^{[last-1]T}$$

$$E^{[i]} = W^{[i+1]T} E^{[i+1]}$$
$$dZ^{[i]} = E^{[i]} g'(Z^{[i]})$$

$$W^{[i]} += dZ^{[i]} \cdot A^{[i-1]T}$$

MNIST-Deep-Net

- 학습 목표
  - **Batch Gradient**와 **Stochastic-GD, Mini-Batch-GD**를 적용한다.
  - **Dropout**을 이용하여 과대적합을 피하는 방법을 학습한다.
  - **MNIST-Fashion DataSet**을 이용하여 **Deep Neural Network**를 테스트한다.

- 학습 내용
  - **Batch Gradient, Stochastic-GD, Mini-Batch-GD**
  - **Dropout**
  - **MNIST-Fashion DataSet**

Mini-Batch-GD: 개념 설명

- 배치 경사하강법
  - 모든 샘플들의 오차 총합
- 확률적 경사하강법
  - 각각의 샘플들의 오차
- 미니 배치 경사하강법
  - 특정 샘플들의 오차 총합

- 장점 :
  - **Python**이나 수치 계산 라이브러리가 대부분 큰 배열을 효율적으로 처리
  - 자료 전송에서의 다량 전달로 인해 병목 현상 사라짐

Deep Neural Network

- 학습 정리
  - **Batch Gradient**와 **Stochastic-GD, Mini-Batch-GD**를 적용하기.
  - **Dropout**을 이용하여 과대적합을 피하는 방법을 학습하기.
  - **MNIST-Fashion DataSet**을 이용하여 **Deep Neural Network**를 테스트하기.

- 차시 예고
  - **12-3 Deep Neural Network 2**