10주차(3/3)

XOR 신경망 구현

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

XOR 신경망

- 학습 목표
  - 다층 신경망을 경사하강법과 역전파 알고리즘으로 구현한다.
  - **XOR**로 신경망을 학습하고 테스트한다.

- 학습 내용
  - 객체지향 다층 신경망 구현하기
  - **fit()** 메소드
  - **net_input()** 메소드
  - **predict()** 메소드
  - **XOR** 신경망 학습

기본 메소드: 생성자, 활성화 함수, 활성화 함수 미분

- 클래스
  - 이름: **NeuralNetwork**
  - 생성자: **__init__()**
  - 활성화 함수: **g()**
  - 활성화 함수 미분 : **g_prime()**

```
1  class NeuralNetwork():
2      """ This class implements a multi-perceptron
3          with backpropagation. This handles a simple logics
4          such as OR, AND, NAND, and NOR gates, including XOR.
5      """
6      def __init__(self, net_arch, eta=0.1, epochs=10000,
7                   random_seed=1):
8          self.layers = len(net_arch)
9          self.net_arch = net_arch
10         self.eta = eta
11         self.epochs = epochs
12         self.random_seed = random_seed
13
14     def g(self, x):
15         return 1/(1 + np.exp((-x)))
16
17     def g_prime(self, x):
18         return self.g(x) * (1 - self.g(x))
19
20     def fit(self, X, Y):
```

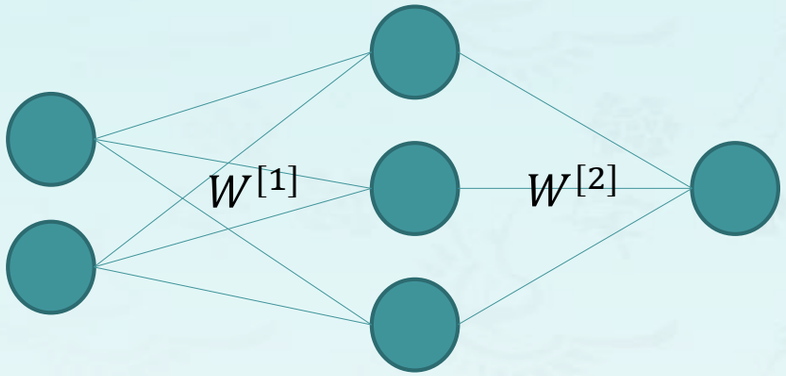
fit() 메소드: 가중치 W_{ij}^{T}

- 클래스
  - 이름: **NeuralNetwork**
  - 생성자: **__init__()**
  - 활성화 함수: **g()**
  - 활성화 함수 미분 : **g_prime()**
  - 학습 메소드: **fit()**

```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 가중치 W_{ij}^{T}

- 클래스
  - 학습 메소드: **fit()**

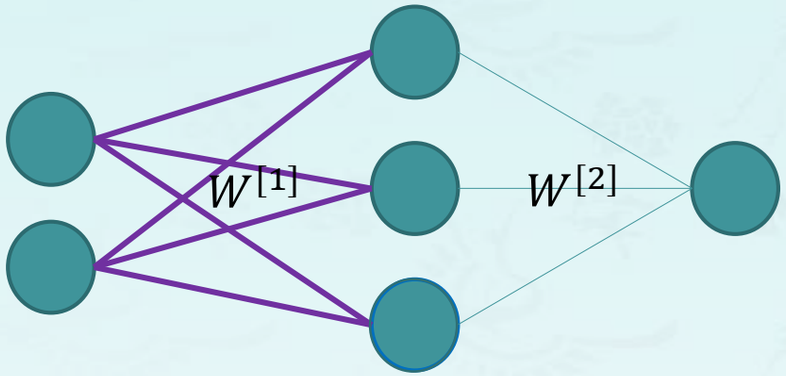



```
self.net_arch = [2, 3, 1]
```

```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 가중치 W_{ij}^{T}

- 클래스
  - 학습 메소드: **fit()**



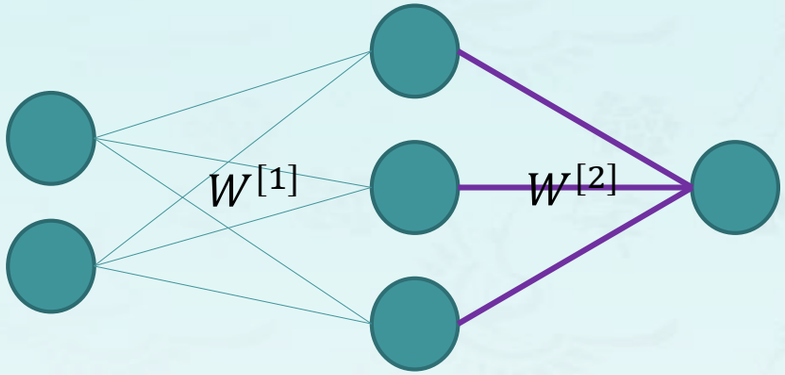

```
self.net_arch = [2, 3, 1]
```

```
W1_shape = (self.net_arch[1], self.net_arch[0])
W2_shape = (self.net_arch[2], self.net_arch[1])
self.W1 = 2*np.random.random(W1_shape) - 1
self.W2 = 2*np.random.random(W2_shape) - 1
```

```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 가중치 W_{ij}^{T}

- 클래스
  - 학습 메소드: **fit()**



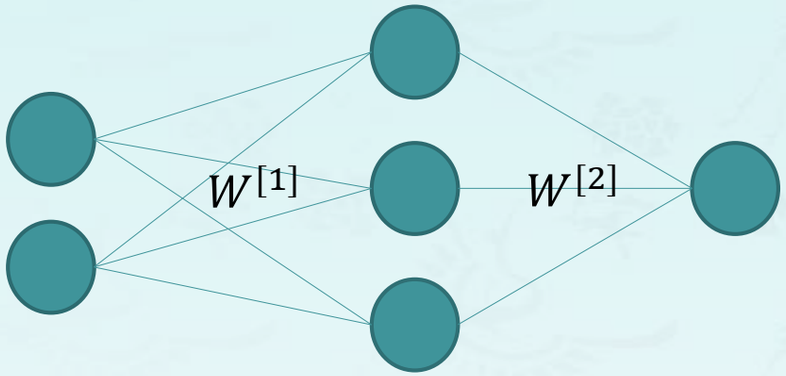

```
self.net_arch = [2, 3, 1]
```

```
W1_shape = (self.net_arch[1], self.net_arch[0])
W2_shape = (self.net_arch[2], self.net_arch[1])
self.W1 = 2*np.random.random(W1_shape) - 1
self.W2 = 2*np.random.random(W2_shape) - 1
```

```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 가중치 W_{ij}^{T}

- 클래스
  - 학습 메소드: **fit()**




```
self.net_arch = [2, 3, 1]
```

```
W1_shape = (self.net_arch[1], self.net_arch[0])
W2_shape = (self.net_arch[2], self.net_arch[1])
self.W1 = 2*np.random.random(W1_shape) - 1
self.W2 = 2*np.random.random(W2_shape) - 1
```

```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 오차

- 클래스
  - 이름: **NeuralNetwork**
  - 생성자: **__init__()**
  - 활성화 함수: **g()**
  - 활성화 함수 미분 : **g_prime()**
  - 학습 메소드: **fit()**

```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 순전파

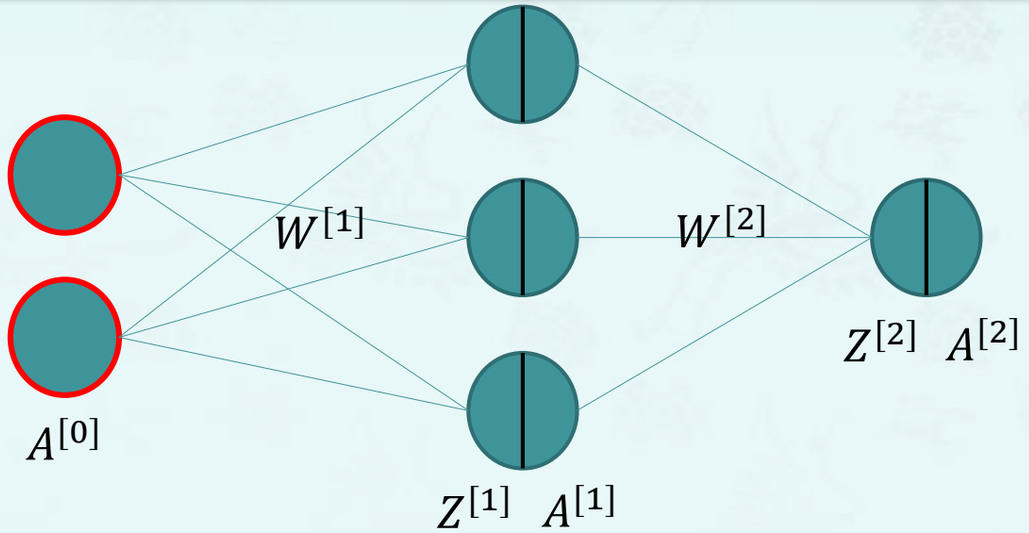
- 클래스
  - 학습 메소드: **fit()**
    - 순전파

```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 순전파

- 클래스
  - 학습 메소드: **fit()**
    - 순전파 : 입력층 ➔ 은닉층

```
A0 = X
Z1 = np.dot(self.W1, A0)
A1 = self.g(Z1)
Z2 = np.dot(self.W2, A1)
A2 = self.g(Z2)
```

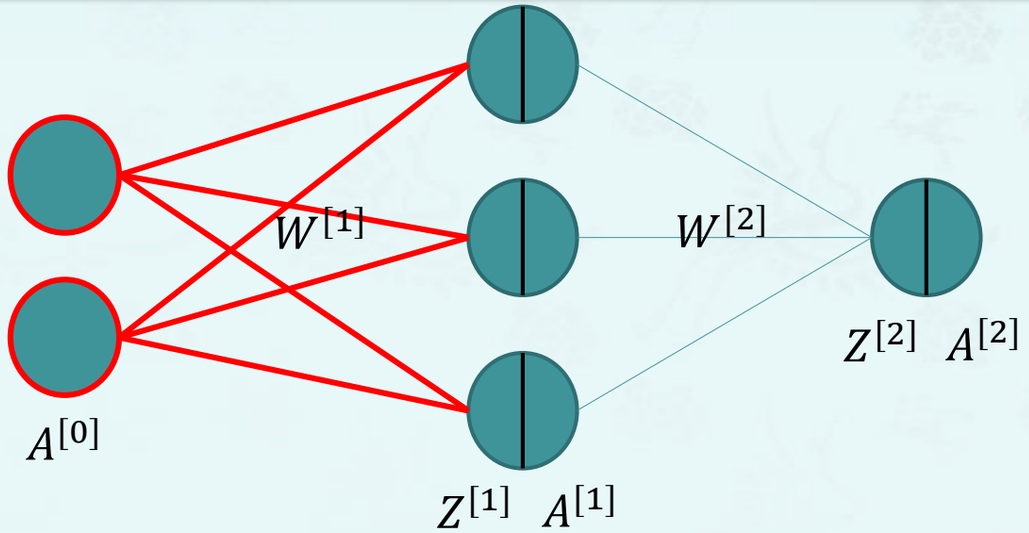



```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 순전파

- 클래스
  - 학습 메소드: **fit()**
    - 순전파 : 입력층 ➔ 은닉층

```
A0 = X
Z1 = np.dot(self.W1, A0)
A1 = self.g(Z1)
Z2 = np.dot(self.W2, A1)
A2 = self.g(Z2)
```

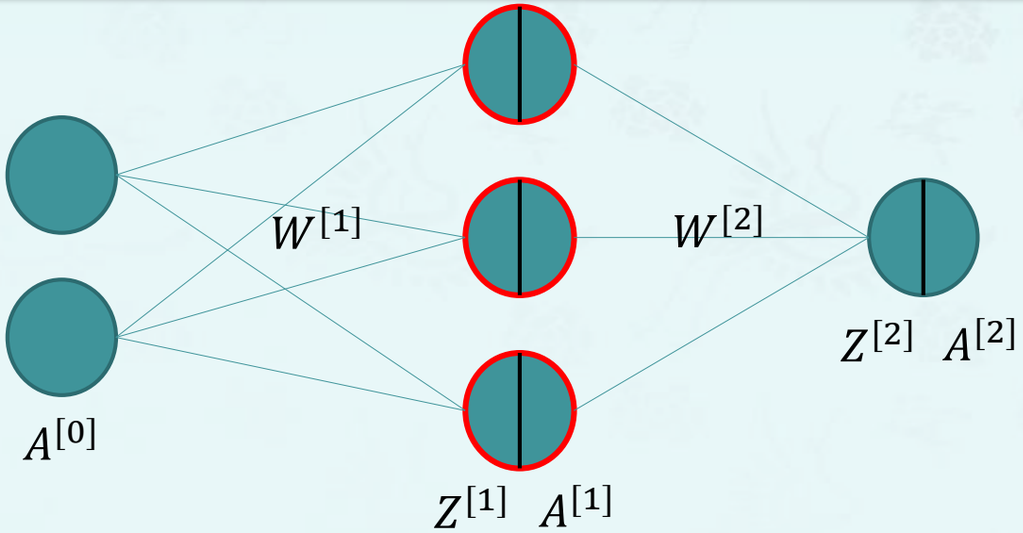



```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 순전파

- 클래스
  - 학습 메소드: **fit()**
    - 순전파 : 입력층 ➔ 은닉층

```
A0 = X
Z1 = np.dot(self.W1, A0)
A1 = self.g(Z1)
Z2 = np.dot(self.W2, A1)
A2 = self.g(Z2)
```

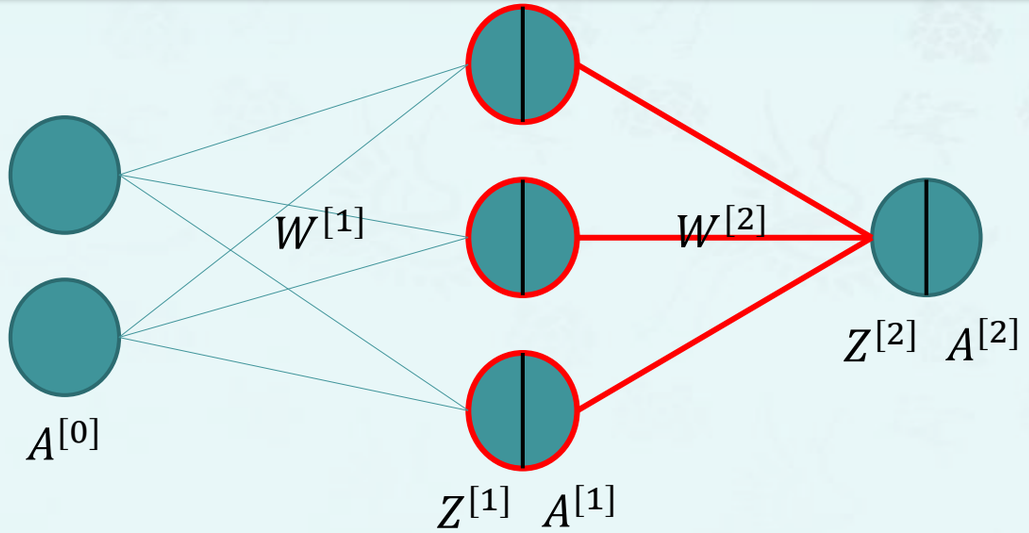



```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 순전파

- 클래스
  - 학습 메소드: **fit()**
    - 순전파 : 은닉층 ➔ 출력층

```
A0 = X
Z1 = np.dot(self.W1, A0)
A1 = self.g(Z1)
Z2 = np.dot(self.W2, A1)
A2 = self.g(Z2)
```

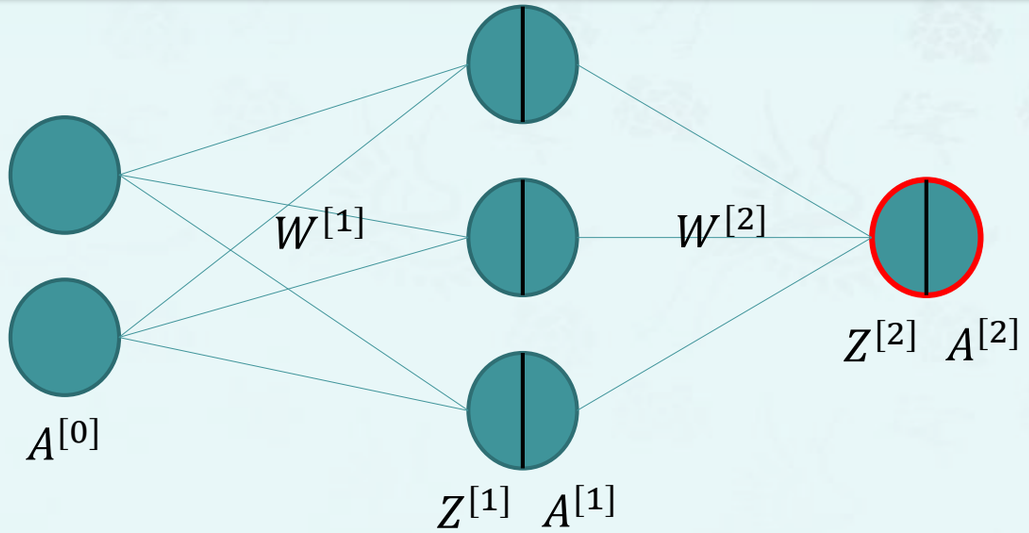



```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 순전파

- 클래스
  - 학습 메소드: **fit()**
    - 순전파 : 은닉층 ➔ 출력층

```
A0 = X
Z1 = np.dot(self.W1, A0)
A1 = self.g(Z1)
Z2 = np.dot(self.W2, A1)
A2 = self.g(Z2)
```




```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 오차

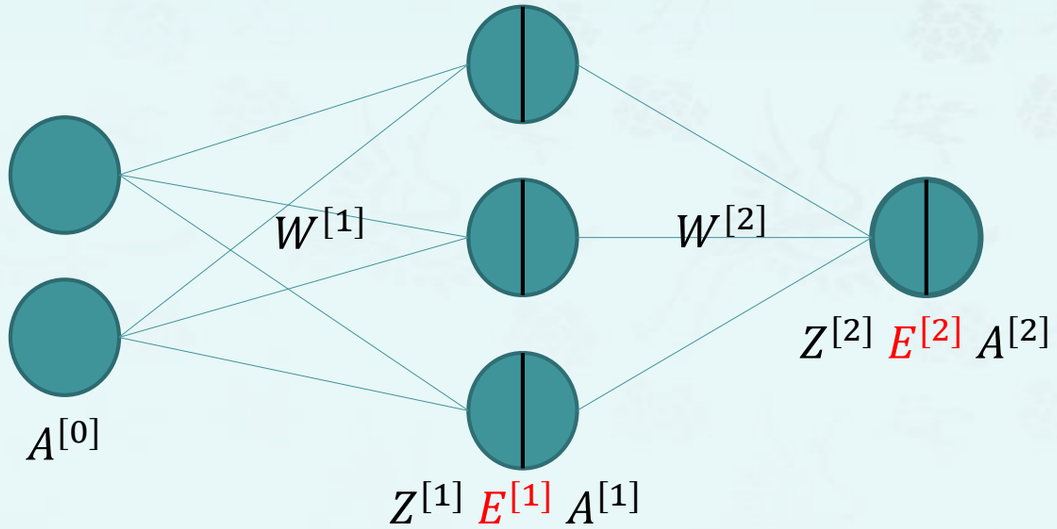
```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 오차

- 역전파 : 오차 계산

$$E^{[2]} = Y - A^{[2]}$$

$$E^{[1]} = W^{[2]\cdot T} \cdot E^{[2]}$$




```python
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 가중치 조정

$$\Delta W^{[2]} = \frac{\partial E}{\partial W^{[2]}} = \underbrace{-E^{[2]} \cdot g'(Z^{[2]})}_{dZ2} \cdot A^{[1]T}$$

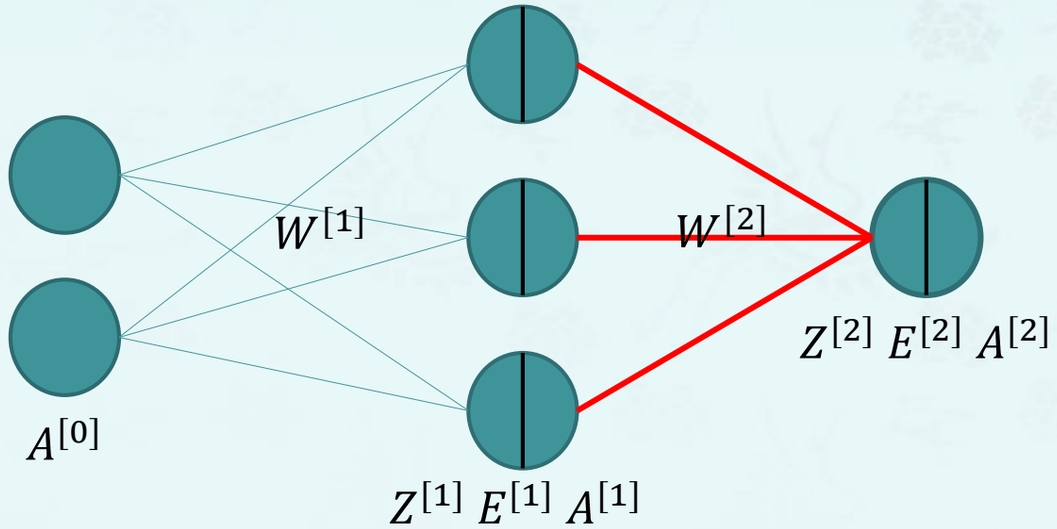
$$\Delta W^{[1]} = \frac{\partial E}{\partial W^{[1]}} = \underbrace{-E^{[1]} \cdot g'(Z^{[1]})}_{dZ1} \cdot A^{[0]T}$$

```python
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

fit() 메소드: 가중치 조정

$$\Delta W^{[2]} = \frac{\partial E}{\partial W^{[2]}} = \underbrace{-E^{[2]} \cdot g'(Z^{[2]})}_{dZ2} \cdot A^{[1]T}$$

$$\Delta W^{[1]} = \frac{\partial E}{\partial W^{[1]}} = -E^{[1]} \cdot g'(Z^{[1]}) \cdot A^{[0]T}$$



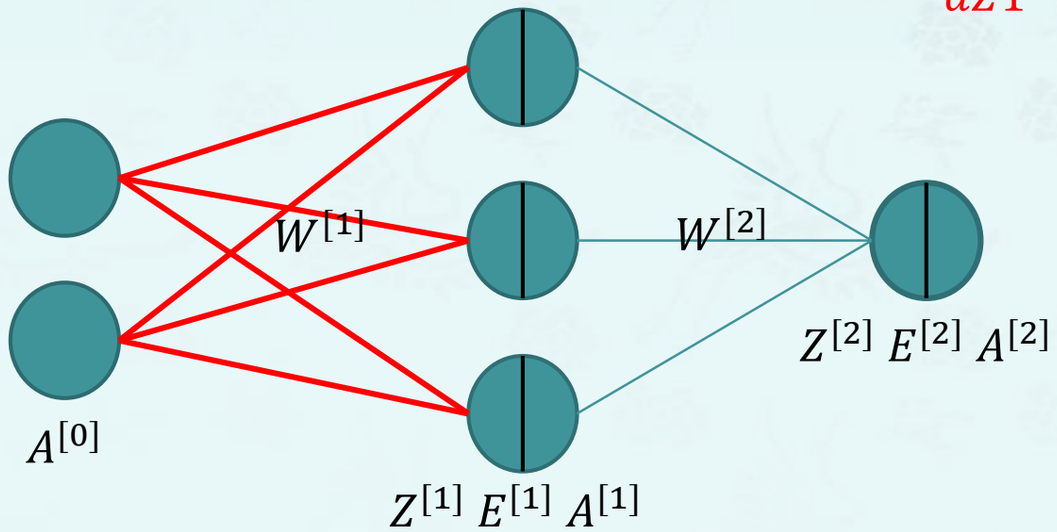

```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 가중치 조정

$$\Delta W^{[2]} = \frac{\partial E}{\partial W^{[2]}} = -E^{[2]} \cdot g'(Z^{[2]}) \cdot A^{[1]T}$$

$$\Delta W^{[1]} = \frac{\partial E}{\partial W^{[1]}} = -E^{[1]} \cdot g'(Z^{[1]}) \cdot A^{[0]T}$$

$dZ1$




```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 가중치 조정

```
1  def fit(self, X, Y):
2      np.random.seed(self.random_seed)
3      W1_shape = (self.net_arch[1], self.net_arch[0])
4      W2_shape = (self.net_arch[2], self.net_arch[1])
5      self.W1 = 2*np.random.random(W1_shape) - 1
6      self.W2 = 2*np.random.random(W2_shape) - 1
7
8      self.cost_ = []
9
10     for _ in range(self.epochs):
11         A0 = X
12         Z1 = np.dot(self.W1, A0)
13         A1 = self.g(Z1)
14         Z2 = np.dot(self.W2, A1)
15         A2 = self.g(Z2)
16
17         E2 = Y - A2
18         E1 = np.dot(self.W2.T, E2)
19
20         dZ2 = E2 * self.g_prime(Z2)
21         dZ1 = E1 * self.g_prime(Z1)
22
23         self.W2 +=  np.dot(dZ2, A1.T)
24         self.W1 +=  np.dot(dZ1, A0.T)
25         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
26     return self
```

fit() 메소드: 오차

- 클래스
  - 이름: **NeuralNetwork**
  - 생성자: **__init__()**
  - 활성화 함수: **g()**
  - 활성화 함수 미분 : **g_prime()**
  - 학습 메소드: **fit()**

```
def fit(self, X, Y):
    np.random.seed(self.random_seed)
    W1_shape = (self.net_arch[1], self.net_arch[0])
    W2_shape = (self.net_arch[2], self.net_arch[1])
    self.W1 = 2*np.random.random(W1_shape) - 1
    self.W2 = 2*np.random.random(W2_shape) - 1

    self.cost_ = []

    for _ in range(self.epochs):
        A0 = X
        Z1 = np.dot(self.W1, A0)
        A1 = self.g(Z1)
        Z2 = np.dot(self.W2, A1)
        A2 = self.g(Z2)

        E2 = Y - A2
        E1 = np.dot(self.W2.T, E2)

        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)

        self.W2 +=  np.dot(dZ2, A1.T)
        self.W1 +=  np.dot(dZ1, A0.T)
        self.cost_.append(np.sqrt(np.sum(E2 * E2)))
    return self
```

기본 메소드: 순입력

- 클래스
  - 이름: **NeuralNetwork**
  - 생성자: **__init__()**
  - 활성화 함수: **g()**
  - 활성화 함수 미분 : **g_prime()**
  - 학습 메소드: **fit()**
  - 순입력: **net_input()**

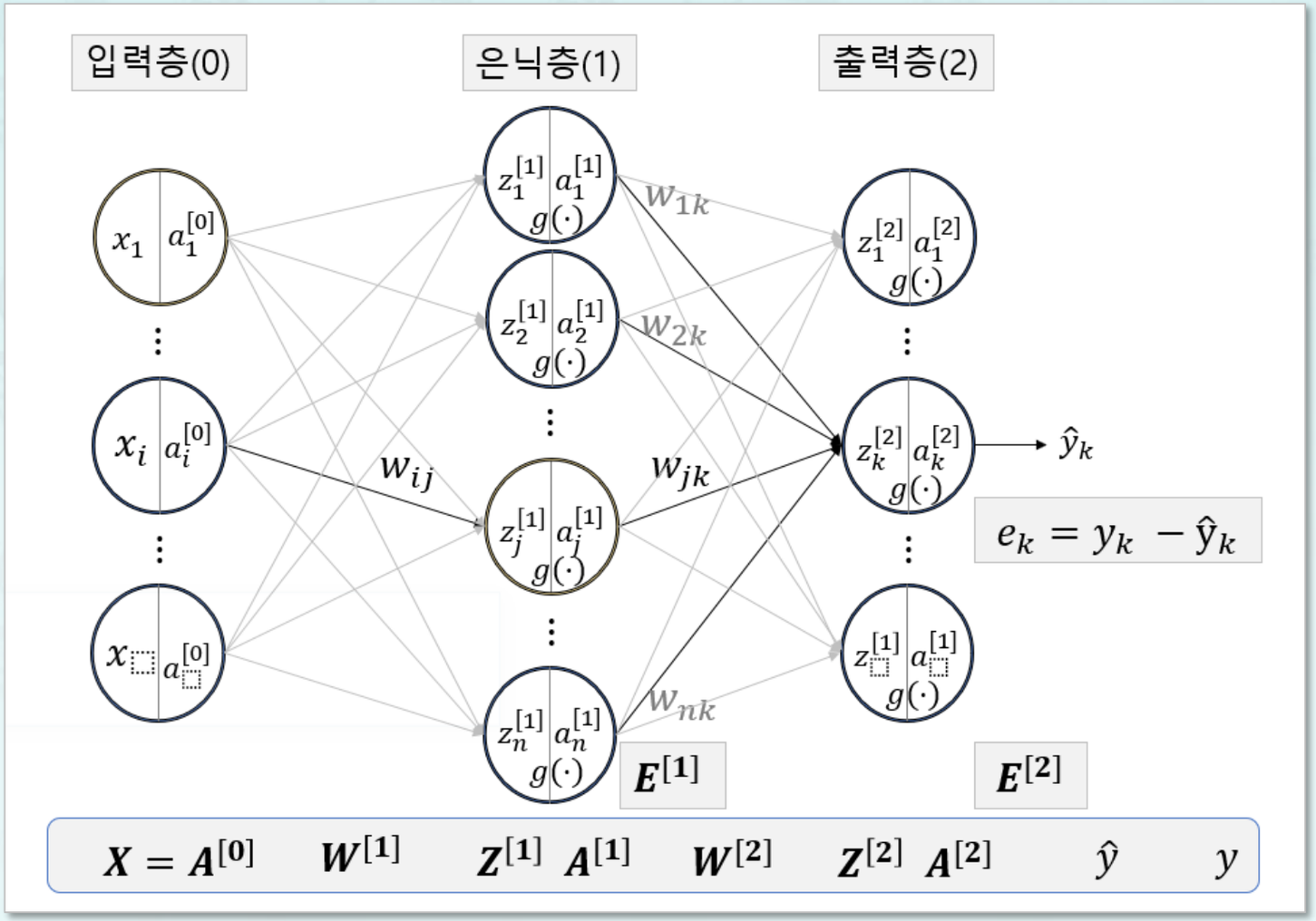
```
44                self.W1 +=  np.dot(dZ1, A0.T)
45                self.cost_.append(np.sqrt(
46                    np.sum(E2 * E2)))
47            return self
48
49        def net_input(self, X):
50            if X.shape[0] == self.w.shape[0]:
51                return np.dot(X, self.w)
52            else:
53                return np.dot(X, self.w[1:]) + self.w[0]
54
55        def predict(self, X):
56            Z1 = np.dot(self.W1, X)
57            A1 = self.g(Z1)
58            Z2 = np.dot(self.W2, A1)
59            A2 = self.g(Z2)
60            return A2
```

기본 메소드: 예측

- 클래스
  - 이름: **NeuralNetwork**
  - 생성자: **__init__()**
  - 활성화 함수: **g()**
  - 활성화 함수 미분 : **g_prime()**
  - 학습 메소드: **fit()**
  - 순입력: **net_input()**
  - 예측: **predict()**

```
44                self.W1 +=  np.dot(dZ1, A0.T)
45                self.cost_.append(np.sqrt(
46                    np.sum(E2 * E2)))
47            return self
48
49        def net_input(self, X):
50            if X.shape[0] == self.w.shape[0]:
51                return np.dot(X, self.w)
52            else:
53                return np.dot(X, self.w[1:]) + self.w[0]
54
55        def predict(self, X):
56            Z1 = np.dot(self.W1, X)
57            A1 = self.g(Z1)
58            Z2 = np.dot(self.W2, A1)
59            A2 = self.g(Z2)
60            return A2
```

XOR 신경망 학습


입력층(0)
은닉층(1)
출력층(2)
x_1 $a_1^{[0]}$
x_i $a_i^{[0]}$
$x_{\square}$ $a_{\square}^{[0]}$
$z_1^{[1]}$ $a_1^{[1]}$ $g(\cdot)$
$z_2^{[1]}$ $a_2^{[1]}$ $g(\cdot)$
$z_j^{[1]}$ $a_j^{[1]}$ $g(\cdot)$
$z_n^{[1]}$ $a_n^{[1]}$ $g(\cdot)$
$z_1^{[2]}$ $a_1^{[2]}$ $g(\cdot)$
$z_k^{[2]}$ $a_k^{[2]}$ $g(\cdot)$
$z_{\square}^{[1]}$ $a_{\square}^{[1]}$ $g(\cdot)$
w_{1k}
w_{2k}
w_{ij}
w_{jk}
w_{nk}
$\hat{y}_k$
$e_k = y_k - \hat{y}_k$
$\boldsymbol{E}^{[1]}$
$\boldsymbol{E}^{[2]}$
$\boldsymbol{X} = \boldsymbol{A}^{[0]}$ $\boldsymbol{W}^{[1]}$ $\boldsymbol{Z}^{[1]}$ $\boldsymbol{A}^{[1]}$ $\boldsymbol{W}^{[2]}$ $\boldsymbol{Z}^{[2]}$ $\boldsymbol{A}^{[2]}$ $\hat{y}$ y

XOR 신경망 학습 결과: 은닉층 노드 3개

```
nn = NeuralNetwork(net_arch=[2, 3, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
     ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

XOR 신경망 학습 결과: 은닉층 노드 3개

```
nn = NeuralNetwork(net_arch=[2, 3, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
print("Final prediction of all")
A2 = nn.predict(X)
for x, yhat in zip(X.T, A2.T):
    print(x, np.round(yhat, 3))
```

XOR 신경망 학습 결과: 은닉층 노드 3개

```
nn = NeuralNetwork(net_arch=[2, 3, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
print("Final prediction of all")
A2 = nn.predict(X)
for x, yhat in zip(X.T, A2.T):
    print(x, np.round(yhat, 3))
```

```
Final prediction of all
[0 0] [ 0.048]
[0 1] [ 0.955]
[1 0] [ 0.499]
[1 1] [ 0.501]
```

XOR 신경망 학습 결과: 은닉층 노드 3개

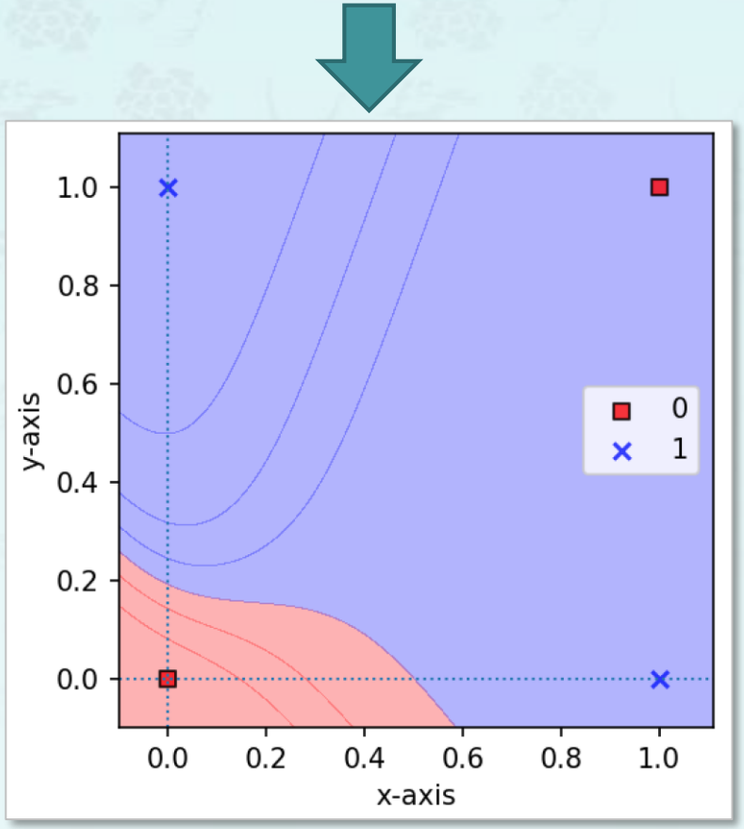
```
nn = NeuralNetwork(net_arch=[2, 3, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
joy.plot_decision_regions(X.T, Y,
                  lambda z : nn.predict(z.T))
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend(loc='best')
plt.show()
```

XOR 신경망 학습 결과: 은닉층 노드 3개

```python
nn = NeuralNetwork(net_arch=[2, 3, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```python
joy.plot_decision_regions(X.T, Y,
                          lambda z : nn.predict(z.T))
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend(loc='best')
plt.show()
```



XOR 신경망 학습 결과: 은닉층 노드 3개

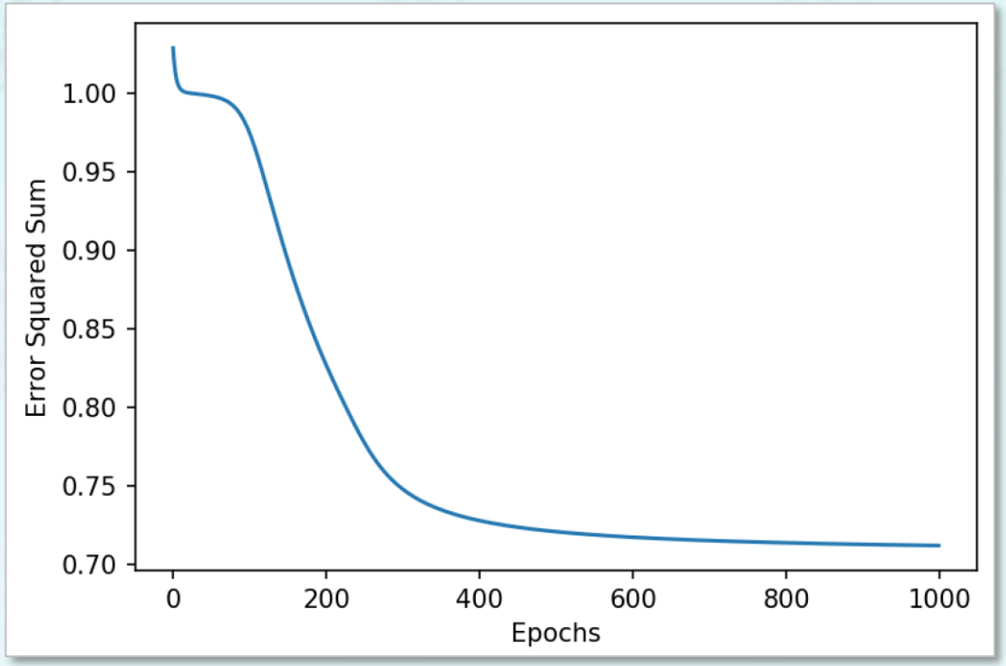
```
nn = NeuralNetwork(net_arch=[2, 3, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
plt.plot(range(len(nn.cost_)), nn.cost_,
         marker='o')
plt.xlabel('Epochs')
plt.ylabel('Error Squared Sum')
plt.show()
```

XOR 신경망 학습 결과: 은닉층 노드 3개

```
nn = NeuralNetwork(net_arch=[2, 3, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
     ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
plt.plot(range(len(nn.cost_)), nn.cost_,
         marker='o')
plt.xlabel('Epochs')
plt.ylabel('Error Squared Sum')
plt.show()
```



XOR 신경망 학습 결과: 은닉층 노드 4개

```
nn = NeuralNetwork(net_arch=[2, 4, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
print("Final prediction of all")
A2 = nn.predict(X)
for x, yhat in zip(X.T, A2.T):
    print(x, np.round(yhat, 3))
```

XOR 신경망 학습 결과: 은닉층 노드 4개

```
nn = NeuralNetwork(net_arch=[2, 4, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
print("Final prediction of all")
A2 = nn.predict(X)
for x, yhat in zip(X.T, A2.T):
    print(x, np.round(yhat, 3))
```

```
Final prediction of all
[0 0] [ 0.077]
[0 1] [ 0.935]
[1 0] [ 0.94]
[1 1] [ 0.043]
```

XOR 신경망 학습 결과: 은닉층 노드 4개

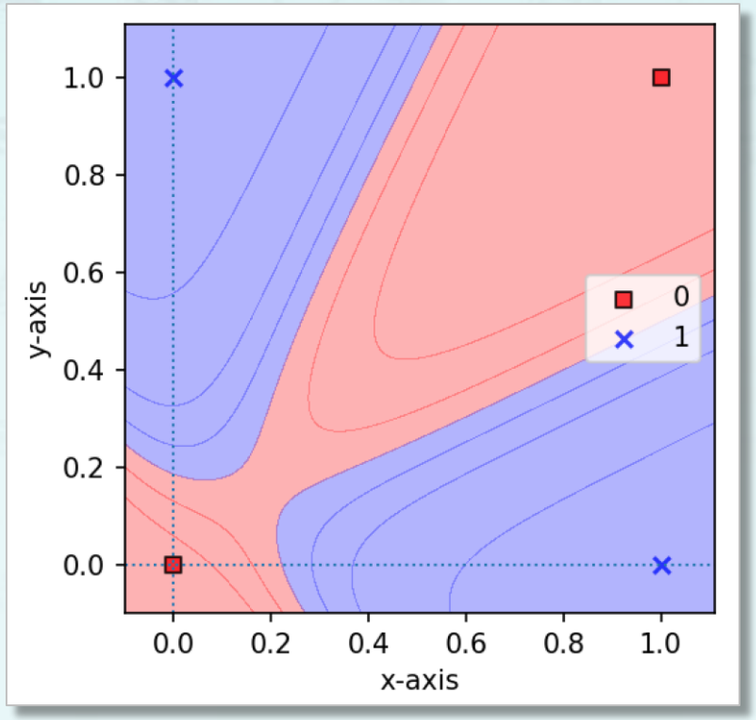
```
nn = NeuralNetwork(net_arch=[2, 4, 1],
                   epochs=1000)
X = np.array([
      [0, 0, 1, 1],
      [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
joy.plot_decision_regions(X.T, Y,
                  lambda z : nn.predict(z.T))
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend(loc='best')
plt.show()
```

XOR 신경망 학습 결과: 은닉층 노드 4개

```python
nn = NeuralNetwork(net_arch=[2, 4, 1],
                   epochs=1000)
X = np.array([
      [0, 0, 1, 1],
      [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```python
joy.plot_decision_regions(X.T, Y,
                          lambda z : nn.predict(z.T))
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend(loc='best')
plt.show()
```



XOR 신경망 학습 결과: 은닉층 노드 4개

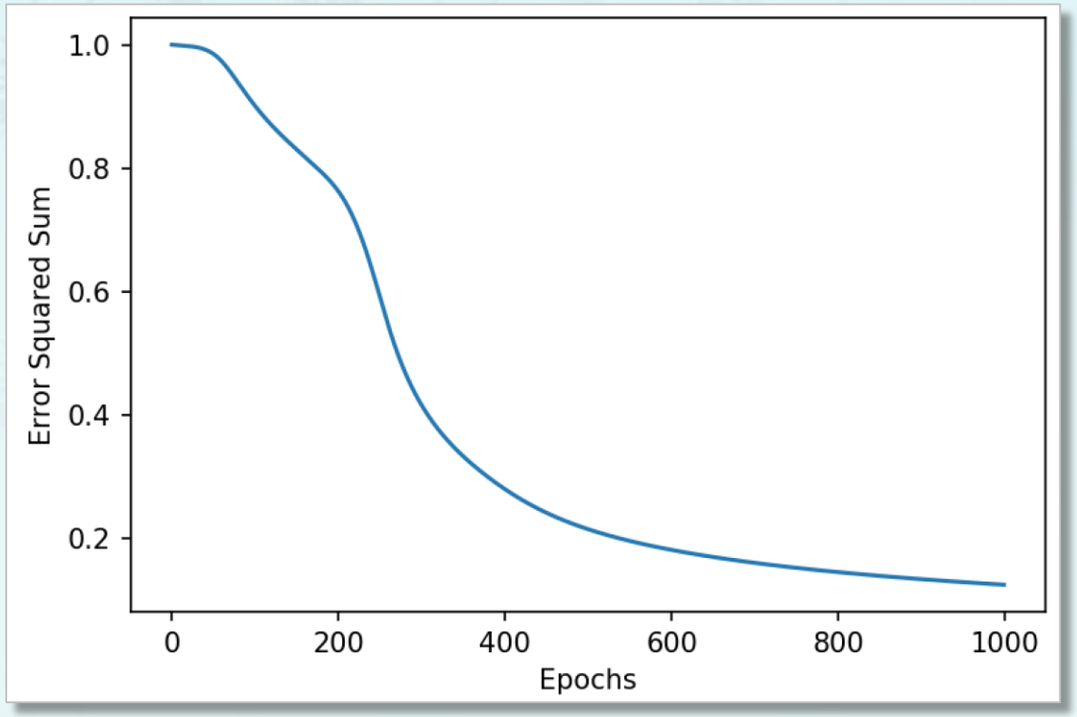
```
plt.plot(range(len(nn.cost_)), nn.cost_,
         marker='o')
plt.xlabel('Epochs')
plt.ylabel('Error Squared Sum')
plt.show()
```

```
nn = NeuralNetwork(net_arch=[2, 4, 1],
                   epochs=1000)
X = np.array([
       [0, 0, 1, 1],
       [0, 1, 0, 1]
     ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

XOR 신경망 학습 결과: 은닉층 노드 4개

```
nn = NeuralNetwork(net_arch=[2, 4, 1],
                   epochs=1000)
X = np.array([
      [0, 0, 1, 1],
      [0, 1, 0, 1]
    ])
Y = np.array([0, 1, 1, 0])
nn.fit(X, Y)
```

```
plt.plot(range(len(nn.cost_)), nn.cost_,
         marker='o')
plt.xlabel('Epochs')
plt.ylabel('Error Squared Sum')
plt.show()
```



XOR 신경망

- 학습 정리
  - **XOR** 신경망을 코드를 이해한다.
  - **XOR** 신경망의 은닉층의 갯수에 따른 결과를 확인한다.

- **10-1** 다층 신경망의 행렬 모델링

10주차(3/3)

XOR 신경망 구현

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수