11주차(1/3)

로지스틱 회귀 3

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

로지스틱 회귀

- 학습 목표
  - 교차 엔트로피 손실함수를 이해한다.
  - 로지스틱 회귀 신경망의 역전파 계산한다.
  - 소프트맥스 활성화 함수를 이해한다.
  - 로지스틱 회귀 신경망을 구현한다.

- 학습 내용
  - 교차 엔트로피 손실함수와 제곱 합 오차함수의 비교
  - 로지스틱 회귀의 역전파를 행렬로 계산하기
  - 소프트맥스 활성화 함수를 이해하기
  - 로지스틱 회귀 신경망에 구현하여 적용하기

1. 손실함수 비교

- 제곱 합 오차 함수
  - 목적 – 함수를 최소로 하는 값 찾기
- 교차 엔트로피 손실 함수
  - 목적 – 함수를 최소로 하는 값 찾기

1. 손실함수 비교

- 제곱 합 오차 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료 오차의 합이 최소
- 교차 엔트로피 손실 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료의 정확한 분류

1. 손실함수 비교

- 제곱 합 오차 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료 오차의 합이 최소

$$E = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

- 교차 엔트로피 손실 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료의 정확한 분류

$$J = -\sum_{i} y^{(i)} log(\hat{y}^{(i)})$$

1. 손실함수 비교

- 제곱 합 오차 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료 오차의 합이 최소

$$E = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

```python
import numpy as np
def MSEcost(self, A2, Y):
    m = Y.shape[1] # m examples
    E2 = Y - A2
    cost = np.sum(E2 * E2) / m
    return cost
```

- 교차 엔트로피 손실 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료의 정확한 분류

$$J = -\sum_{i} y^{(i)} log(\hat{y}^{(i)})$$

1. 손실함수 비교:

- 제곱 합 오차 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료 오차의 합이 최소

$$E = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

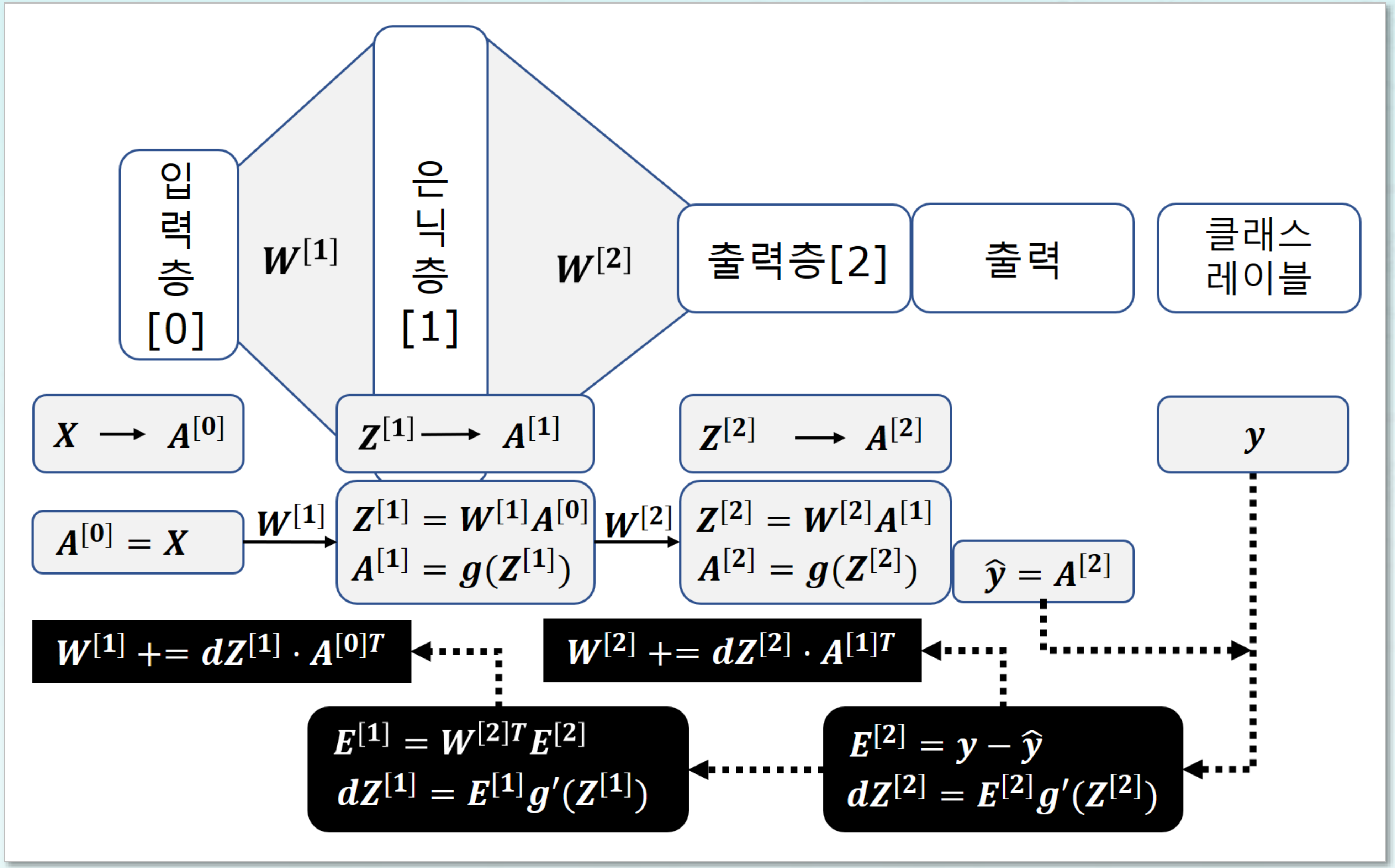
```python
import numpy as np
def MSEcost(self, A2, Y):
    m = Y.shape[1] # m examples
    E2 = Y - A2
    cost = np.sum(E2 * E2) / m
    return cost
```

- 교차 엔트로피 손실 함수
  - 목적 – 함수를 최소로 하는 값 찾기
  - 방법 – 모든 자료의 정확한 분류

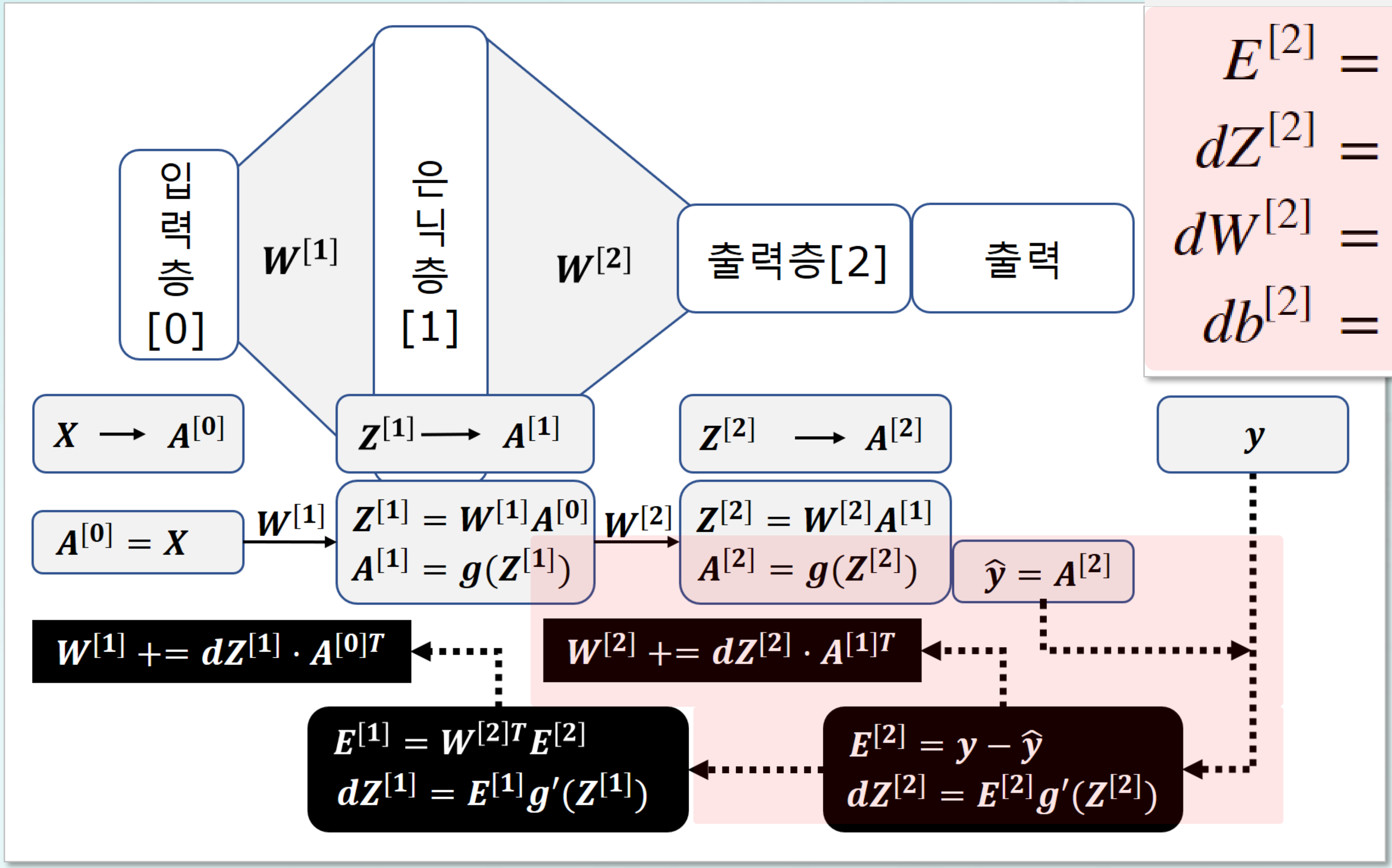
$$J = -\sum_{i} y^{(i)} log(\hat{y}^{(i)})$$

```python
def CEcost(self, A2, Y):
    m = Y.shape[1]  # number of example
    logprobs = np.multiply(Y, np.log(A2))
    cost = -np.sum(logprobs) / m
    cost = np.squeeze(cost)
    return cost
```

2. 로지스틱 회귀 신경망: 역전파



2. 로지스틱 회귀 신경망: 역전파



출력층 역전파 일반공식

$$E^{[2]} = y - \hat{y}$$
$$dZ^{[2]} = E^{[2]}g^{[2]\prime}(Z^{[2]})$$
$$dW^{[2]} = dZ^{[2]}A^{[1]T}$$
$$db^{[2]} = dZ^{[2]}$$

2. 로지스틱 회귀 신경망: 역전파

- 출력층 오차
  - 출력층 -> 은닉층

$$
\begin{aligned}
E^{[2]} &= y - A^{[2]} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
&= E^{[2]} \\
dW^{[2]} &= \frac{\partial E}{\partial W^{[2]}} \\
&= \frac{1}{m} E^{[2]} \cdot A^{[1]T} \\
&= \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \\
db^{[2]} &= \frac{1}{m} np.sum(dZ^{[2]}, axis = 1)
\end{aligned}
$$

출력층 역전파 일반공식

$$
\begin{aligned}
E^{[2]} &= y - \hat{y} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
dW^{[2]} &= dZ^{[2]} A^{[1]T} \\
db^{[2]} &= dZ^{[2]}
\end{aligned}
$$

2. 로지스틱 회귀 신경망: 역전파

- 출력층 오차
  - 출력층 -> 은닉층

$$
\begin{aligned}
E^{[2]} &= y - A^{[2]} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
&= E^{[2]} \\
dW^{[2]} &= \frac{\partial E}{\partial W^{[2]}} \\
&= \frac{1}{m} E^{[2]} \cdot A^{[1]T} \\
&= \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \\
db^{[2]} &= \frac{1}{m} np.\,sum(dZ^{[2]}, axis = 1)
\end{aligned}
$$

출력층 역전파 일반공식

$$
\begin{aligned}
E^{[2]} &= y - \hat{y} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
dW^{[2]} &= dZ^{[2]} A^{[1]T} \\
db^{[2]} &= dZ^{[2]}
\end{aligned}
$$

2. 로지스틱 회귀 신경망: 역전파

- 출력층 오차
  - 출력층 -> 은닉층

$$
\begin{aligned}
E^{[2]} &= y - A^{[2]} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
&= E^{[2]} \\
dW^{[2]} &= \frac{\partial E}{\partial W^{[2]}} \\
&= \frac{1}{m} E^{[2]} \cdot A^{[1]T} \\
&= \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \\
db^{[2]} &= \frac{1}{m} np.sum(dZ^{[2]}, axis = 1)
\end{aligned}
$$

출력층 역전파 일반공식

$$
\begin{aligned}
E^{[2]} &= y - \hat{y} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
dW^{[2]} &= dZ^{[2]} A^{[1]T} \\
db^{[2]} &= dZ^{[2]}
\end{aligned}
$$

2. 로지스틱 회귀 신경망: 역전파

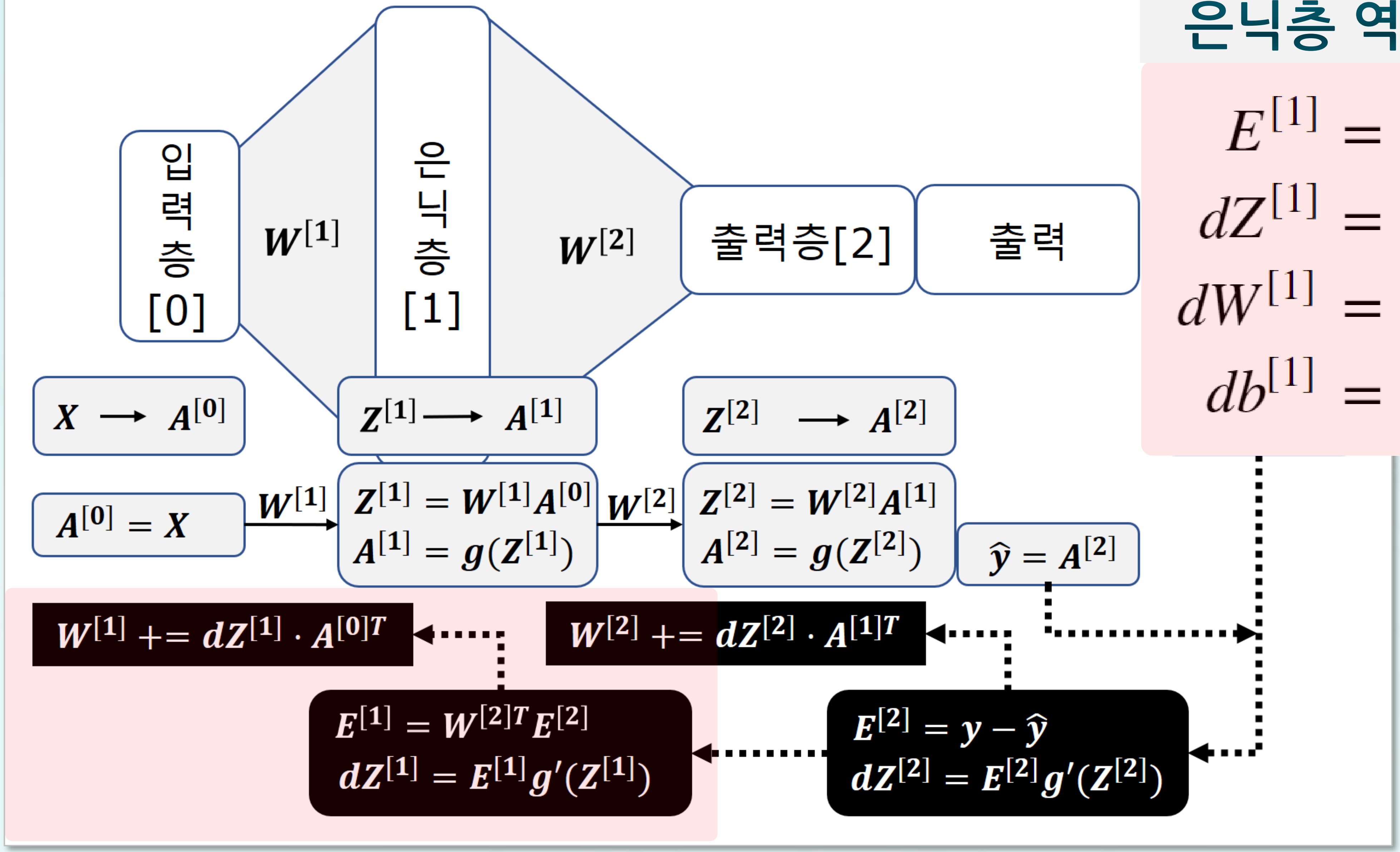
- 출력층 오차
  - 출력층 -> 은닉층

$$
\begin{aligned}
E^{[2]} &= y - A^{[2]} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
&= E^{[2]} \\
dW^{[2]} &= \frac{\partial E}{\partial W^{[2]}} \\
&= \frac{1}{m} E^{[2]} \cdot A^{[1]T} \\
&= \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \\
db^{[2]} &= \frac{1}{m} np.sum(E^{[2]}, axis = 1)
\end{aligned}
$$

출력층 역전파 일반공식

$$
\begin{aligned}
E^{[2]} &= y - \hat{y} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
dW^{[2]} &= dZ^{[2]} A^{[1]T} \\
db^{[2]} &= dZ^{[2]}
\end{aligned}
$$

2. 로지스틱 회귀 신경망: 역전파



은닉층 역전파 일반공식

$$E^{[1]} = W^{[2]T}E^{[2]}$$
$$dZ^{[1]} = E^{[1]}g^{[1]\prime}(Z^{[1]})$$
$$dW^{[1]} = dZ^{[1]}A^{[0]T}$$
$$db^{[1]} = dZ^{[1]}$$

2. 로지스틱 회귀 신경망: 역전파

- 은닉층 오차
  - 은닉층 -> 입력층

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

$$dZ^{[1]} = E^{[1]} * g^{[1]\prime}(Z^{[1]})$$
$$= E^{[1]} * (1 - tanh^2(Z^{[1]}))$$
$$= E^{[1]} * (1 - A^{[1]2})$$
$$dW^{[1]} = dZ^{[1]} \cdot A^{[0]T}$$
$$db^{[1]} = np.sum(dZ^{[1]}, axis = 1)$$

은닉층 역전파 일반공식

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

$$dZ^{[1]} = E^{[1]} g^{[1]\prime}(Z^{[1]})$$
$$dW^{[1]} = dZ^{[1]} A^{[0]T}$$
$$db^{[1]} = dZ^{[1]}$$

2. 로지스틱 회귀 신경망: 역전파

- 은닉층 오차
  - 은닉층 -> 입력층

$$
\begin{aligned}
E^{[1]} &= W^{[2]T} E^{[2]} \\
dZ^{[1]} &= E^{[1]} * g^{[1]\prime}(Z^{[1]}) \\
&= E^{[1]} * (1 - tanh^2(Z^{[1]})) \\
&= E^{[1]} * (1 - A^{[1]2}) \\
dW^{[1]} &= dZ^{[1]} \cdot A^{[0]T} \\
db^{[1]} &= np.sum(dZ^{[1]}, axis = 1)
\end{aligned}
$$

$$
\begin{aligned}
g'(Z^{[1]}) &= tanh'(Z^{[1]}) \\
&= 1 - tanh^2(Z^{[1]})
\end{aligned}
$$

여기서 * 는 원소 별 곱셈을 의미합니다.

2. 로지스틱 회귀 신경망: 역전파

- 은닉층 오차
  - 은닉층 -> 입력층

$$
\begin{aligned}
E^{[1]} &= W^{[2]T} E^{[2]} \\
dZ^{[1]} &= E^{[1]} * g^{[1]\prime}(Z^{[1]}) \\
&= E^{[1]} * (1 - tanh^2(Z^{[1]})) \\
&= E^{[1]} * (1 - A^{[1]2}) \\
dW^{[1]} &= dZ^{[1]} \cdot A^{[0]T} \\
db^{[1]} &= np.sum(dZ^{[1]}, axis = 1)
\end{aligned}
$$

$$A^{[1]} = g(Z^{[1]}) = tanh(Z^{[1]})$$

여기서 * 는 원소 별 곱셈을 의미합니다.

2. 로지스틱 회귀 신경망: 역전파

- 은닉층 오차
  - 은닉층 -> 입력층

$$
\begin{aligned}
E^{[1]} &= W^{[2]T} E^{[2]} \\
dZ^{[1]} &= E^{[1]} * g^{[1]\prime}(Z^{[1]}) \\
&= E^{[1]} * (1 - tanh^2(Z^{[1]})) \\
&= E^{[1]} * (1 - A^{[1]2}) \\
dW^{[1]} &= dZ^{[1]} \cdot A^{[0]T} \\
db^{[1]} &= np.sum(dZ^{[1]}, axis = 1)
\end{aligned}
$$

은닉층 역전파 일반공식

$$
\begin{aligned}
E^{[1]} &= W^{[2]T} E^{[2]} \\
dZ^{[1]} &= E^{[1]} g^{[1]\prime}(Z^{[1]}) \\
dW^{[1]} &= dZ^{[1]} A^{[0]T} \\
db^{[1]} &= dZ^{[1]}
\end{aligned}
$$

3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화
  - 상대적 비교

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화
  - 상대적 비교
  - 분류

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화
  - 상대적 비교
  - 분류

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

- 시그모이드
  - 강아지**(0.9)** 고양이**(0.8)** 호랑이**(0.7)**

3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화
  - 상대적 비교
  - 분류

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

- 시그모이드
  - 강아지**(0.9)** 고양이**(0.8)** 호랑이**(0.7)**
  - 강아지**(0.1)** 고양이**(0.3)** 호랑이**(0.5)**

3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화
  - 상대적 비교
  - 분류

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

- 시그모이드
  - 강아지**(0.9)** 고양이**(0.8)** 호랑이**(0.7)**
  - 강아지**(0.1)** 고양이**(0.3)** 호랑이**(0.5)**

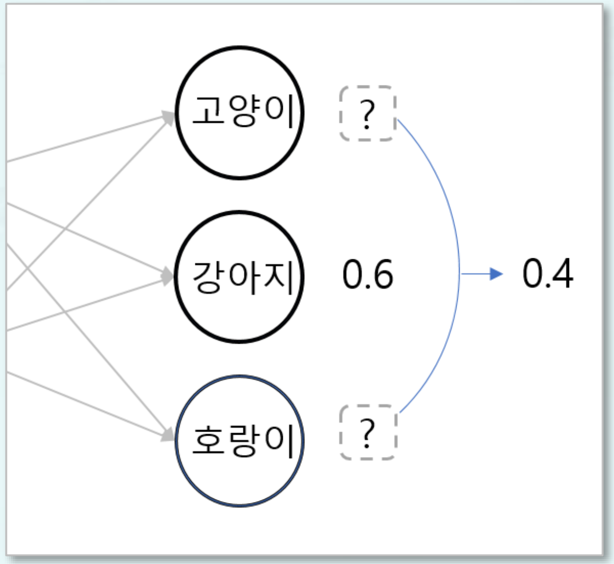
- 소프트 맥스
  - 강아지**(0.6)** 고양이**(0.2)** 호랑이**(0.1)**
  - 강아지**(0.0)** 고양이**(0.2)** 호랑이**(0.6)**

3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화
  - 상대적 비교
  - 분류

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

- 시그모이드
  - 강아지**(0.9)** 고양이**(0.8)** 호랑이**(0.7)**
  - 강아지**(0.1)** 고양이**(0.3)** 호랑이**(0.5)**
- 소프트 맥스
  - 강아지**(0.6)** 고양이**(0.2)** 호랑이**(0.1)**
  - 강아지**(0.0)** 고양이**(0.2)** 호랑이**(0.6)**



3. 활성화 함수: 소프트 맥스

- 소프트 맥스
  - 정규화
  - 상대적 비교
  - 분류

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad j = 1, \ldots, K$$

```
def softmax(self, a):
    exp_a = np.exp(a - np.max(a))
    return exp_a / np.sum(exp_a)
```

- 시그모이드
  - 강아지**(0.9)** 고양이**(0.8)** 호랑이**(0.7)**
  - 강아지**(0.1)** 고양이**(0.3)** 호랑이**(0.5)**
- 소프트 맥스
  - 강아지**(0.6)** 고양이**(0.2)** 호랑이**(0.1)**
  - 강아지**(0.0)** 고양이**(0.2)** 호랑이**(0.6)**

4. 로지스틱 회귀: 구현 – 순전파

- 순전파: **forpass()**
  - 은닉층: **sigmoid()**
  - 출력층: **softmax()**

```
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                   # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

```
def forpass(self, A0):
    Z1 = np.dot(self.W1, A0) + self.b1
    A1 = self.g(Z1)
    Z2 = np.dot(self.W2, A1) + self.b2
    A2 = self.softmax(Z2)
    return Z1, A1, Z2, A2
```

4. 로지스틱 회귀: 구현 – 교차 엔트로피

- 손실함수: **CEcost()**
  - 교차 엔트로피

$$J = -\sum_i y^{(i)} log(\hat{y}^{(i)})$$

```python
def CEcost(self, A2, Y):
    m = Y.shape[1]   # number of example
    logprobs = np.multiply(Y, np.log(A2))
    cost = -np.sum(logprobs) / m
    cost = np.squeeze(cost)
    return cost
```

```python
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

4. 로지스틱 회귀: 구현 – 역전파

- 출력층에서 은닉층 역전파

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

$$\begin{aligned} dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\ &= E^{[2]} \end{aligned}$$

$$\begin{aligned} dW^{[2]} &= \frac{\partial E}{\partial W^{[2]}} \\ &= \frac{1}{m} E^{[2]} \cdot A^{[1]T} \\ &= \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \end{aligned}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1)$$

```python
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                     # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

4. 로지스틱 회귀: 구현 – 역전파

- 출력층에서 은닉층 역전파

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

$$dZ^{[2]} = E^{[2]} g^{[2]\prime}(Z^{[2]}) = E^{[2]}$$

$$dW^{[2]} = \frac{\partial E}{\partial W^{[2]}} = \frac{1}{m} E^{[2]} \cdot A^{[1]T} = \frac{1}{m} dZ^{[2]} \cdot A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1)$$

```python
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                  # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

4. 로지스틱 회귀: 구현 – 역전파

- 출력층에서 은닉층 역전파

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

$$dZ^{[2]} = E^{[2]} g^{[2]\prime}(Z^{[2]})$$

$$= E^{[2]}$$

$$dW^{[2]} = \frac{\partial E}{\partial W^{[2]}}$$

$$= \frac{1}{m} E^{[2]} \cdot A^{[1]T}$$

$$= \frac{1}{m} dZ^{[2]} \cdot A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1)$$

```python
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                      # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

4. 로지스틱 회귀: 구현 – 역전파

- 출력층에서 은닉층 역전파

$$
\begin{aligned}
E^{[2]} &= y - A^{[2]} \\
dZ^{[2]} &= E^{[2]} g^{[2]\prime}(Z^{[2]}) \\
&= E^{[2]} \\
dW^{[2]} &= \frac{\partial E}{\partial W^{[2]}} \\
&= \frac{1}{m} E^{[2]} \cdot A^{[1]T} \\
&= \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \\
db^{[2]} &= \frac{1}{m} np.sum(dZ^{[2]}, axis = 1)
\end{aligned}
$$

```python
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                    # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

4. 로지스틱 회귀: 구현 – 역전파

- 은닉층에서 입력층 역전파

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

$$\begin{aligned} dZ^{[1]} &= E^{[1]} * g^{[1]'}(Z^{[1]}) \\ &= E^{[1]} * (1 - tanh^2(Z^{[1]})) \\ &= E^{[1]} * (1 - A^{[1]2}) \end{aligned}$$

$$dW^{[1]} = dZ^{[1]} \cdot A^{[0]T}$$

$$db^{[1]} = np.sum(dZ^{[1]}, axis = 1)$$

```
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                    # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

4. 로지스틱 회귀: 구현

- 은닉층에서 입력층 역전파

```python
#sigmoid
def g(self, x):
    x = np.clip(x, -500, 500)
    return 1.0/(1.0 + np.exp(-x))
def g_prime(self, x):
    return self.g(x) * (1 - self.g(x))
```

```python
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for sample in range(self.m_samples):
            A0 = np.array(X[sample], ndmin=2).T
            Y0 = np.array(Y[sample], ndmin=2).T
            Z1, A1, Z2, A2 = self.forpass(A0)
            cost = self.CEcost(A2, Y0) #cross-entropy
            self.cost_.append(cost)

            E2 = Y0 - A2                  # Backprop
            dZ2 = E2
            dW2 = np.dot(dZ2, A1.T) / self.m_samples
            db2 = np.sum(dZ2, axis=1,
                         keepdims=True)/self.m_samples
            E1 = np.dot(self.W2.T, E2)
            dZ1 = E1 * self.g_prime(Z1)  #sigmoid
            #dZ1 = E1 * (1 - np.power(A1, 2)) #tanh
            dW1 = np.dot(dZ1, A0.T)
            db1 = np.sum(dZ1, axis=1, keepdims=True)
            self.W1 += self.eta * dW1
            self.b1 += self.eta * db1
            self.W2 += self.eta * dW2
            self.b2 += self.eta * db2
    return self
```

4. 로지스틱 회귀: 실행

```
import joy
(X, y), (Xtest, ytest) = joy.load_mnist()
self_accuracy = []
test_accuracy = []
epoch_list = np.arange(1, 31)
for e in epoch_list:
    nn = joy.LogisticNeuron_stochastic(784, 100, 10,
                                        eta = 0.2, epochs = e)
    nn.fit(X, y)
    self_accuracy.append(nn.evaluate(X, y))
    test_accuracy.append(nn.evaluate(Xtest, ytest))
```

LogisticNeuron_stochastic()는 학습자료를 하나씩 입력받아 손실을 계산하고 가중치를 조정하는 학습방법을 사용하였습니다. 이를 확률적 경사하강법이라고 하며, 다음**(11-3)**에 다룰 예정입니다.

4. 로지스틱 회귀: 실행

```python
import joy
(X, y), (Xtest, ytest) = joy.load_mnist()
self_accuracy = []
test_accuracy = []
epoch_list = np.arange(1, 31)
for e in epoch_list:
    nn = joy.LogisticNeuron_stochastic(784, 100, 10,
                                        eta = 0.2, epochs = e)
    nn.fit(X, y)
    self_accuracy.append(nn.evaluate(X, y))
    test_accuracy.append(nn.evaluate(Xtest, ytest))
```

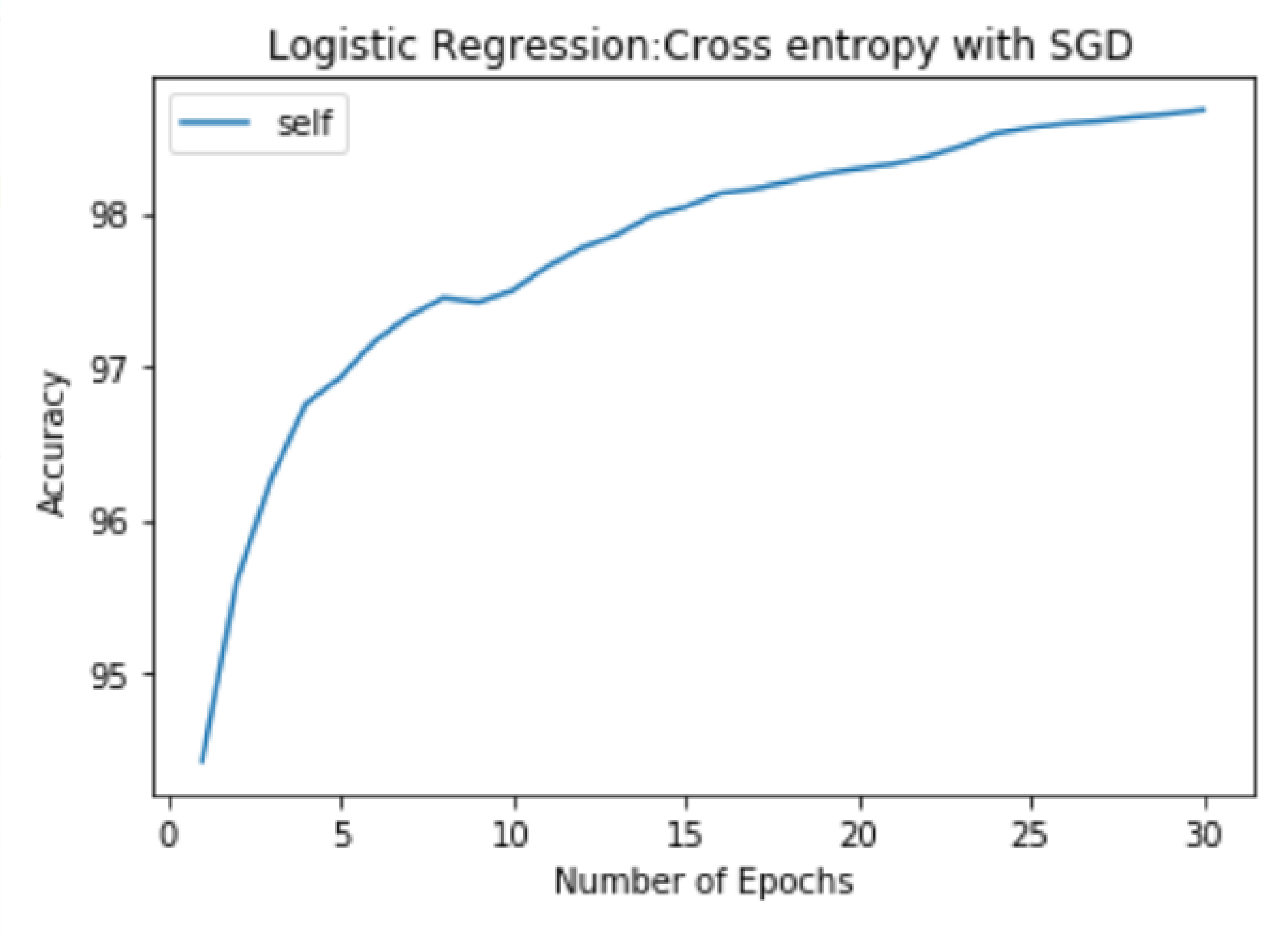
LogisticNeuron_stochastic()는 학습자료를 하나씩 입력받아 손실을 계산하고 가중치를 조정하는 학습방법을 사용하였습니다. 이를 확률적 경사하강법이라고 하며, 다음**(11-3)**에 다룰 예정입니다.

4. 로지스틱 회귀: 실행

```
plt.plot(epoch_list, self_accuracy, label='self')
plt.plot(epoch_list, test_accuracy, label='test')
plt.xlabel('Number of Epochs')
plt.ylabel('Accuracy')
plt.title('Logistic Regressin:Cross entropy with SGD')
plt.legend(loc='best')
plt.show()
```

4. 로지스틱 회귀: 실행

```
plt.plot(epoch_list, se
plt.plot(epoch_list, te
plt.xlabel('Number of E
plt.ylabel('Accuracy')
plt.title('Logistic Reg
plt.legend(loc='best')
plt.show()
```



4. 로지스틱 회귀: 실행

```
plt.plot(epoch_list, se
plt.plot(epoch_list, te
plt.xlabel('Number of E
plt.ylabel('Accuracy')
plt.title('Logistic Reg
plt.legend(loc='best')
plt.show()
```

로지스틱 회귀

- 학습 정리
  - 교차 엔트로피 손실함수의 이해하기
  - 로지스틱 회귀의 역전파를 행렬로 계산하기
  - 소프트맥스 활성화 함수 이해하기
  - 로지스틱 회귀 신경망 구현하기

- 차시 예고
  - **11-2 MNIST Dataset**

10주차(3/3)

로지스틱 회귀 3

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

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