11주차(3/3)

경사하강법 1

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

경사하강법 1

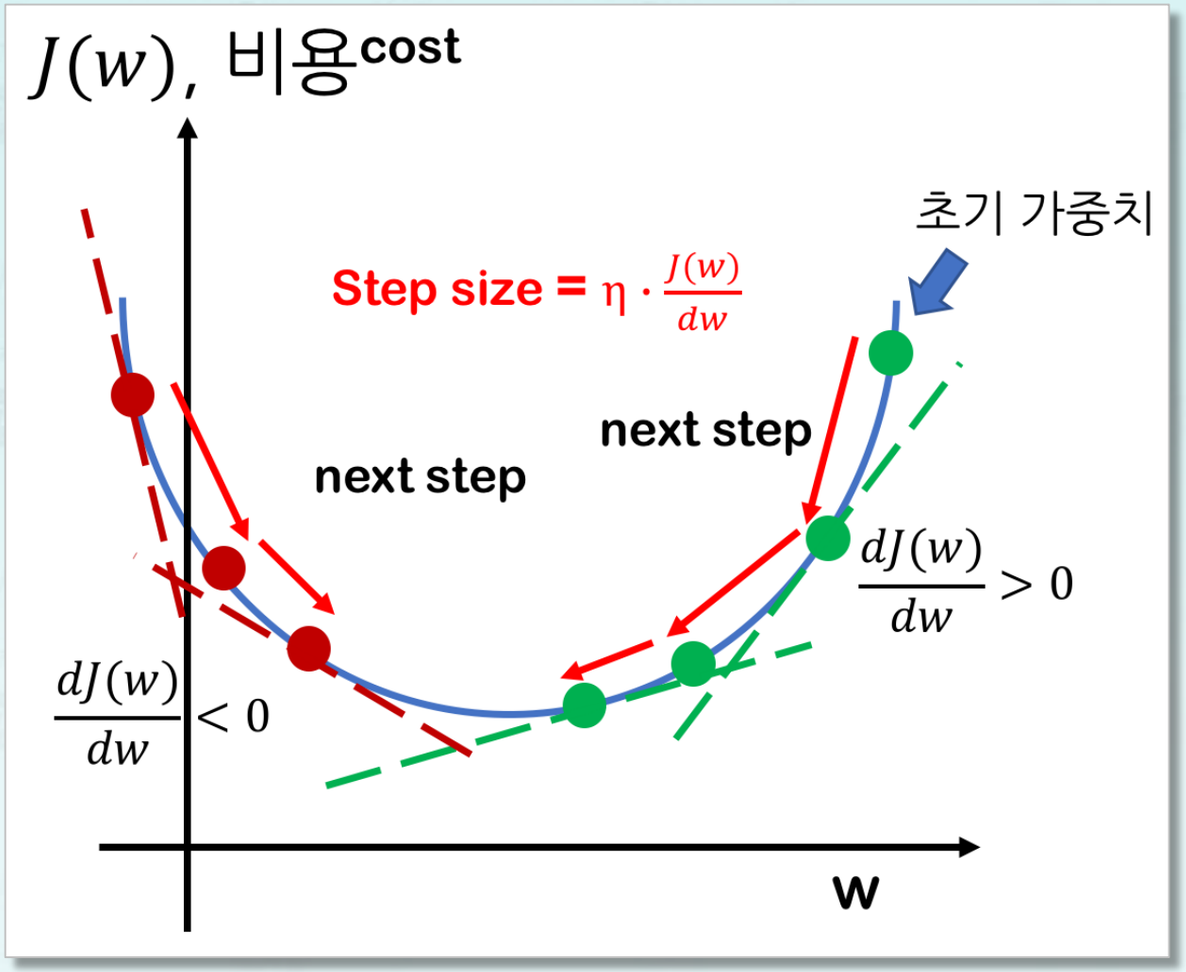
- 학습 목표
  - 배치 경사하강법**(Batch GD)**으로 학습의 정확도를 이해한다.
  - 확률적 경사하강법**(Stochastic GD)**으로 학습의 효율성을 이해한다.

- 학습 내용
  - **MNIST** 자료셋에 대한 다양한 경사하강법 비교하기
  - 배치 경사하강법**(Batch GD)**의 정확도 이해하기
  - 확률적 경사하강법**(Stochastic GD)**으로 학습하기

1. Batch GD: 오차함수 *J(w)*

$$\Delta w = -\eta \frac{\partial J(w)}{\partial w_j}$$

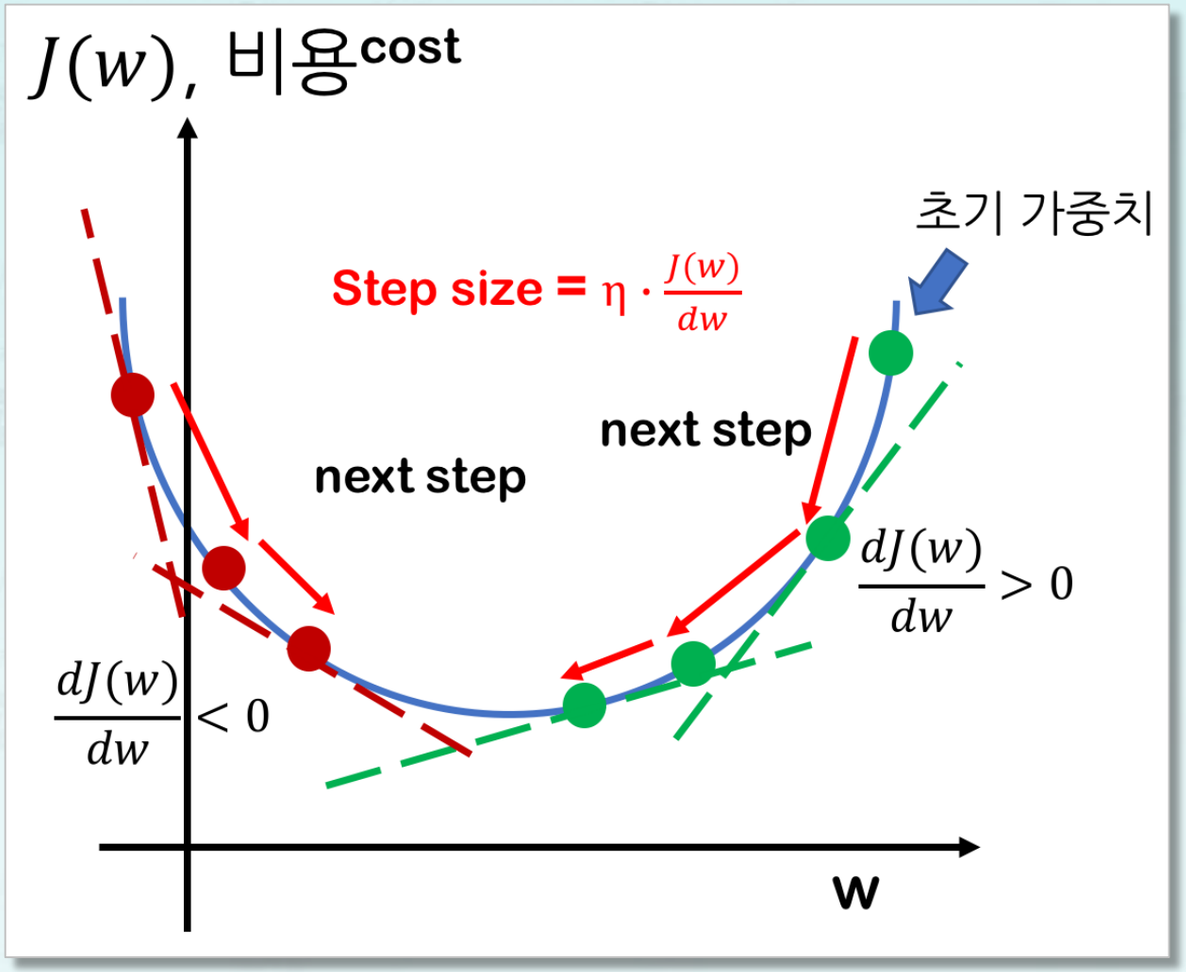
방향과 속도



1. Batch GD: 오차함수 *J(w)*

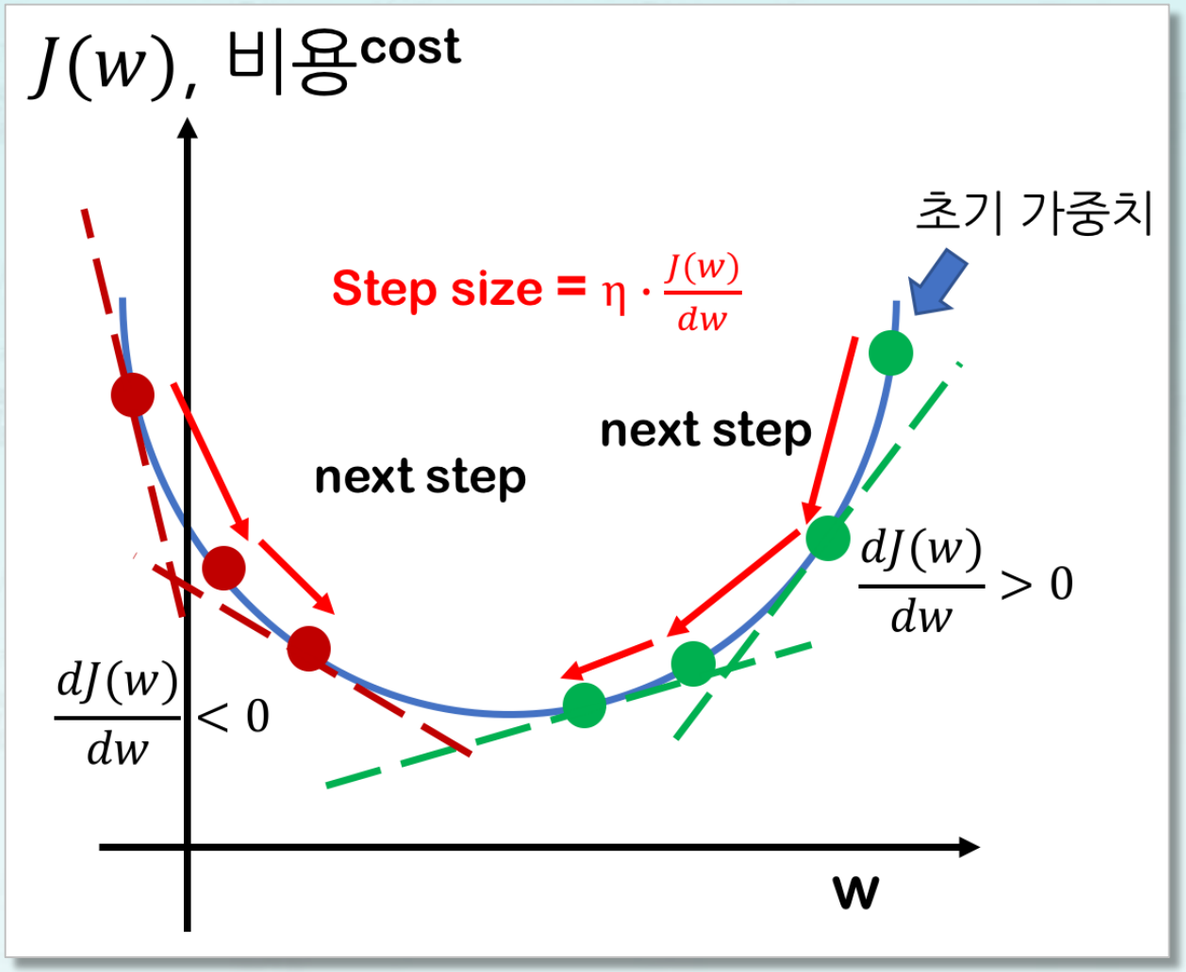
$$\Delta w = -\eta \frac{\partial J(w)}{\partial w_j}$$

방향과 스텝 크기



1. Batch GD: 오차함수 *J(w)*

$$\Delta w = -\eta \frac{\partial J(w)}{\partial w_j}$$



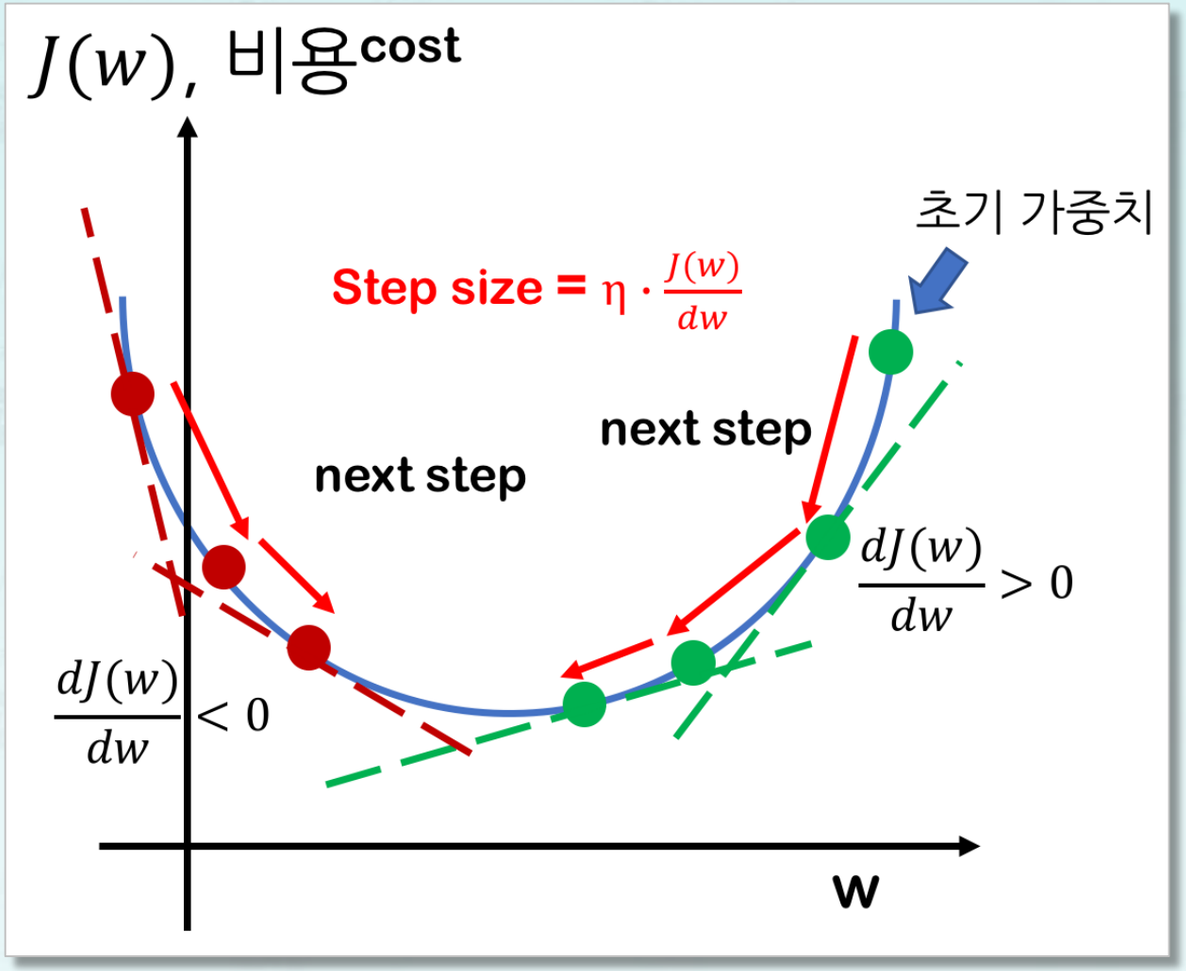
1. Batch GD: 오차함수 *J(w)*

$$\Delta w = -\eta \frac{\partial J(w)}{\partial w_j}$$

$$J(w) = \frac{1}{2}\sum_i \left(y^{(i)} - h\left(z^{(i)}\right)\right)^2$$

$$J(w) = \frac{1}{2m}\sum_i \left(y^{(i)} - h\left(z^{(i)}\right)\right)^2$$

- 평균제곱오차
  - **Mean squared error (MSE)**



1. Batch GD: MNIST 학습 코드

```python
def fit(self, X, y):
    m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    self.cost_ = []
    for epoch in range(self.epochs):
        A0 = np.array(X, ndmin=2).T
        Y0 = np.array(Y, ndmin=2).T

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

        E2 = Y0 - A2
        E1 = np.dot(self.W2.T, E2)
        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)
        dW2 = self.eta * np.dot(dZ2, A1.T)
        dW1 = self.eta * np.dot(dZ1, A0.T)

        self.W2 += dW2 / m_samples
        self.W1 += dW1 / m_samples
        self.cost_.append(np.sqrt(np.sum(E2*E2)))
    return self
```

1. Batch GD: one-hot-encoding

$$Y = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & & 0 \\ 0 & 1 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 1 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 1 & 0 & & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & 0 & & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 0 \\ 4 \\ 1 \\ 9 \\ \vdots \\ \vdots \\ 3 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & & 0 \end{pmatrix}$$

```
1   def fit(self, X, y):
2       m_samples = len(y)
3       Y = joy.one_hot_encoding(y, self.n_y)
4       self.cost_ = []
5       for epoch in range(self.epochs):
6           A0 = np.array(X, ndmin=2).T
7           Y0 = np.array(Y, ndmin=2).T
8
9           Z1 = np.dot(self.W1, A0)
10          A1 = self.g(Z1)
11          Z2 = np.dot(self.W2, A1)
12          A2 = self.g(Z2)
13
14          E2 = Y0 - A2
15          E1 = np.dot(self.W2.T, E2)
16          dZ2 = E2 * self.g_prime(Z2)
17          dZ1 = E1 * self.g_prime(Z1)
18          dW2 = self.eta * np.dot(dZ2, A1.T)
19          dW1 = self.eta * np.dot(dZ1, A0.T)
20
21          self.W2 += dW2 / m_samples
22          self.W1 += dW1 / m_samples
23          self.cost_.append(np.sqrt(np.sum(E2*E2)))
24      return self
```

1. Batch GD: 2차원 배열 및 컬럼 벡터 설정 코드

```
def fit(self, X, y):
    m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    self.cost_ = []
    for epoch in range(self.epochs):
        A0 = np.array(X, ndmin=2).T
        Y0 = np.array(Y, ndmin=2).T

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

        E2 = Y0 - A2
        E1 = np.dot(self.W2.T, E2)
        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)
        dW2 = self.eta * np.dot(dZ2, A1.T)
        dW1 = self.eta * np.dot(dZ1, A0.T)

        self.W2 += dW2 / m_samples
        self.W1 += dW1 / m_samples
        self.cost_.append(np.sqrt(np.sum(E2*E2)))
    return self
```

1. Batch GD: 가중치 조정 코드

```
def fit(self, X, y):
    m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    self.cost_ = []
    for epoch in range(self.epochs):
        A0 = np.array(X, ndmin=2).T
        Y0 = np.array(Y, ndmin=2).T

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

        E2 = Y0 - A2
        E1 = np.dot(self.W2.T, E2)
        dZ2 = E2 * self.g_prime(Z2)
        dZ1 = E1 * self.g_prime(Z1)
        dW2 = self.eta * np.dot(dZ2, A1.T)
        dW1 = self.eta * np.dot(dZ1, A0.T)

        self.W2 += dW2 / m_samples
        self.W1 += dW1 / m_samples
        self.cost_.append(np.sqrt(np.sum(E2*E2)))
    return self
```

1. Batch GD: MNIST 학습 결과

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

1. Batch GD: 학습 상태 확인 코드

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

1. Batch GD: evaluate 메소드

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

```
1  def evaluate(self, Xtest, ytest):
2      m_samples = len(ytest)
3      scores = 0
4      A2 = self.predict(Xtest)
5      yhat = np.argmax(A2, axis = 0)
6      scores += np.sum(yhat == ytest)
7      return scores/m_samples * 100
```

1. Batch GD: evaluate 메소드

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

```
1  def evaluate(self, Xtest, ytest):
2      m_samples = len(ytest)
3      scores = 0
4      A2 = self.predict(Xtest)
5      yhat = np.argmax(A2, axis = 0)
6      scores += np.sum(yhat == ytest)
7      return scores/m_samples * 100
```

1. Batch GD: MNIST 학습 결과

```
(X, y), (Xtest, ytest) = joy.load_mnist()
X, y = X[:1000], y[:1000]
Xtest, ytest = Xtest[:100], ytest[:100]
nn = MnistBGD(784, 100, 10, eta = 0.1,
              epochs = 1000)
nn.fit(X, y)
accuracy = nn.evaluate(Xtest, ytest)
print('Accuracy {}%'.format(accuracy))
```

```
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ \vdots & & & & & \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

샘플의 수 100

한 샘플의 예측값 10

1. Batch GD: MNIST 학습 결과

- A2.shape: (10, 100)
- yhat.shape:
- ytest.shape:

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

```
1  def evaluate(self, Xtest, ytest):
2      m_samples = len(ytest)
3      scores = 0
4      A2 = self.predict(Xtest)
5      yhat = np.argmax(A2, axis = 0)
6      scores += np.sum(yhat == ytest)
7      return scores/m_samples * 100
```

1. Batch GD: MNIST 학습 결과

- A2.shape: (10, 100)
- yhat.shape:
- ytest.shape: (100,)

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ & & & & & \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

```
1  def evaluate(self, Xtest, ytest):
2      m_samples = len(ytest)
3      scores = 0
4      A2 = self.predict(Xtest)
5      yhat = np.argmax(A2, axis = 0)
6      scores += np.sum(yhat == ytest)
7      return scores/m_samples * 100
```

1. Batch GD: MNIST 학습 결과

- A2.shape: (10, 100)
- yhat.shape:
- ytest.shape: (100,)

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ & & & & & \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

```
1  def evaluate(self, Xtest, ytest):
2      m_samples = len(ytest)
3      scores = 0
4      A2 = self.predict(Xtest)
5      yhat = np.argmax(A2, axis = 0)
6      scores += np.sum(yhat == ytest)
7      return scores/m_samples * 100
```

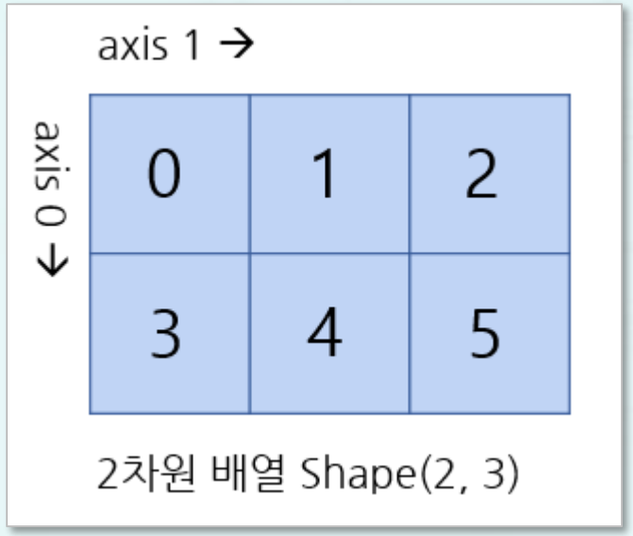
1. Batch GD: MNIST 학습 결과

```
(X, y), (Xtest, ytest) = joy.load_mnist()
X, y = X[:1000], y[:1000]
Xtest, ytest = Xtest[:100], ytest[:100]
nn = MnistBGD(784, 100, 10, eta = 0.1,
              epochs = 1000)
nn.fit(X, y)
accuracy = nn.evaluate(Xtest, ytest)
print('Accuracy {}%'.format(accuracy))
```

```
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

[복습] 배열의 축 (Axis) 다루기

- **2차원 배열 Axis**




```
print('sum(axis=0):', np.sum(a, axis=0))
```

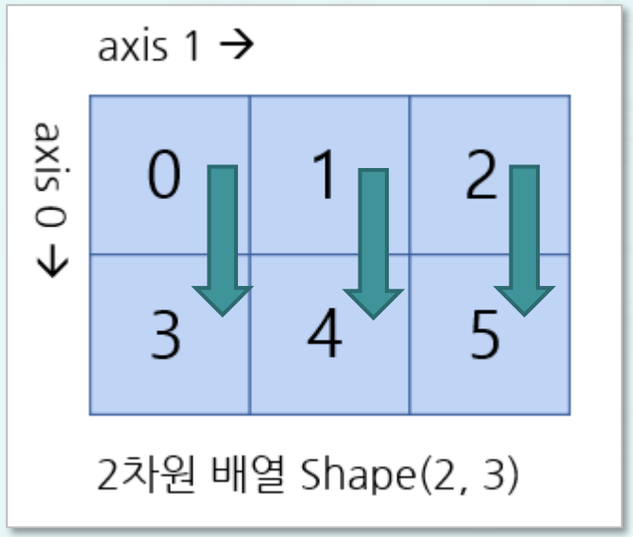
```
sum(axis=0):
```

np.sum(a, axis = 0) = ? **(3, 12)**

(3, 5, 7)

[복습] 배열의 축 (Axis) 다루기

- **2차원 배열 Axis**




```
print('sum(axis=0):', np.sum(a, axis=0))
```

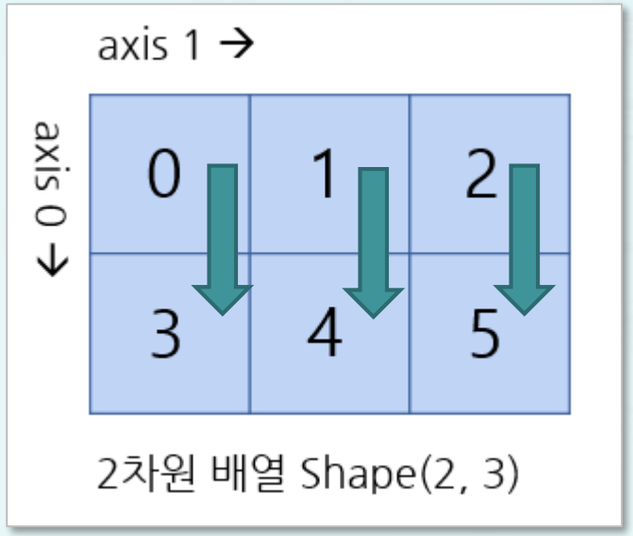
```
sum(axis=0):
```

np.sum(a, axis = 0) = ? **(3, 12)**

(3, 5, 7)

[복습] 배열의 축 (Axis) 다루기

- **2차원 배열 Axis**




```
print('sum(axis=0):', np.sum(a, axis=0))
```

```
sum(axis=0): [3 5 7]
```

np.sum(a, axis = 0) = (3, 5, 7)

1. Batch GD: MNIST 학습 결과

- A2.shape: (10, 100)
- yhat.shape:
- ytest.shape: (100,)

axis = 0

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ \vdots & & & & & \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

yhat =

```
1  (X, y), (Xtest, ytest) = joy.load_mnist()
2  X, y = X[:1000], y[:1000]
3  Xtest, ytest = Xtest[:100], ytest[:100]
4  nn = MnistBGD(784, 100, 10, eta = 0.1,
5                epochs = 1000)
6  nn.fit(X, y)
7  accuracy = nn.evaluate(Xtest, ytest)
8  print('Accuracy {}%'.format(accuracy))
```

```
1  def evaluate(self, Xtest, ytest):
2      m_samples = len(ytest)
3      scores = 0
4      A2 = self.predict(Xtest)
5      yhat = np.argmax(A2, axis = 0)
6      scores += np.sum(yhat == ytest)
7      return scores/m_samples * 100
```

1. Batch GD: MNIST 학습 결과

- A2.shape: (10, 100)
- yhat.shape: (100,)
- ytest.shape: (100,)

```
(X, y), (Xtest, ytest) = joy.load_mnist()
X, y = X[:1000], y[:1000]
Xtest, ytest = Xtest[:100], ytest[:100]
nn = MnistBGD(784, 100, 10, eta = 0.1,
              epochs = 1000)
nn.fit(X, y)
accuracy = nn.evaluate(Xtest, ytest)
print('Accuracy {}%'.format(accuracy))
```

```
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

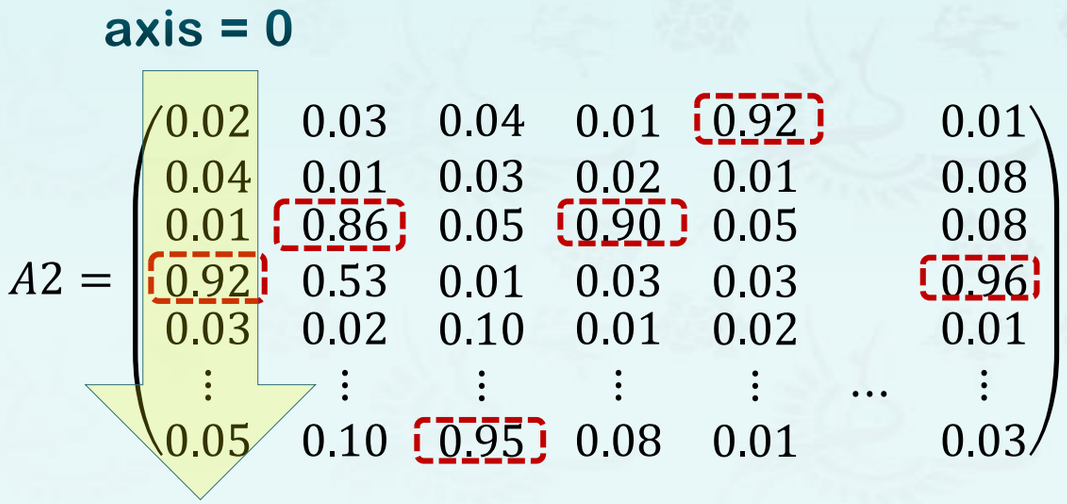
axis = 0

$$A2 = \begin{pmatrix} 0.02 & 0.03 & 0.04 & 0.01 & 0.92 & 0.01 \\ 0.04 & 0.01 & 0.03 & 0.02 & 0.01 & 0.08 \\ 0.01 & 0.86 & 0.05 & 0.90 & 0.05 & 0.08 \\ 0.92 & 0.53 & 0.01 & 0.03 & 0.03 & 0.96 \\ 0.03 & 0.02 & 0.10 & 0.01 & 0.02 & 0.01 \\ \vdots & & & & & \\ 0.05 & 0.10 & 0.95 & 0.08 & 0.01 & 0.03 \end{pmatrix}$$

yhat =

1. Batch GD: MNIST 학습 결과

- A2.shape: (10, 100)
- yhat.shape: (100,)
- ytest.shape: (100,)



yhat = [3, 2, 10, 2, 0, ... 3]

```python
(X, y), (Xtest, ytest) = joy.load_mnist()
X, y = X[:1000], y[:1000]
Xtest, ytest = Xtest[:100], ytest[:100]
nn = MnistBGD(784, 100, 10, eta = 0.1,
              epochs = 1000)
nn.fit(X, y)
accuracy = nn.evaluate(Xtest, ytest)
print('Accuracy {}%'.format(accuracy))
```

```python
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

1. Batch GD: MNIST 학습 결과

```
(X, y), (Xtest, ytest) = joy.load_mnist()
X, y = X[:1000], y[:1000]
Xtest, ytest = Xtest[:100], ytest[:100]
nn = MnistBGD(784, 100, 10, eta = 0.1,
              epochs = 1000)
nn.fit(X, y)
accuracy = nn.evaluate(Xtest, ytest)
print('Accuracy {}%'.format(accuracy))
```

```
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

1. Batch GD: MNIST 학습 결과

```
(X, y), (Xtest, ytest) = joy.load_mnist()
X, y = X[:1000], y[:1000]
Xtest, ytest = Xtest[:100], ytest[:100]
nn = MnistBGD(784, 100, 10, eta = 0.1,
              epochs = 1000)
nn.fit(X, y)
accuracy = nn.evaluate(Xtest, ytest)
print('Accuracy {}%'.format(accuracy))
```

```
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

1. Batch GD: MNIST 학습 결과

```
(X, y), (Xtest, ytest) = joy.load_mnist()
X, y = X[:1000], y[:1000]
Xtest, ytest = Xtest[:100], ytest[:100]
nn = MnistBGD(784, 100, 10, eta = 0.1,
              epochs = 1000)
nn.fit(X, y)
accuracy = nn.evaluate(Xtest, ytest)
print('Accuracy {}%'.format(accuracy))
```

```
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

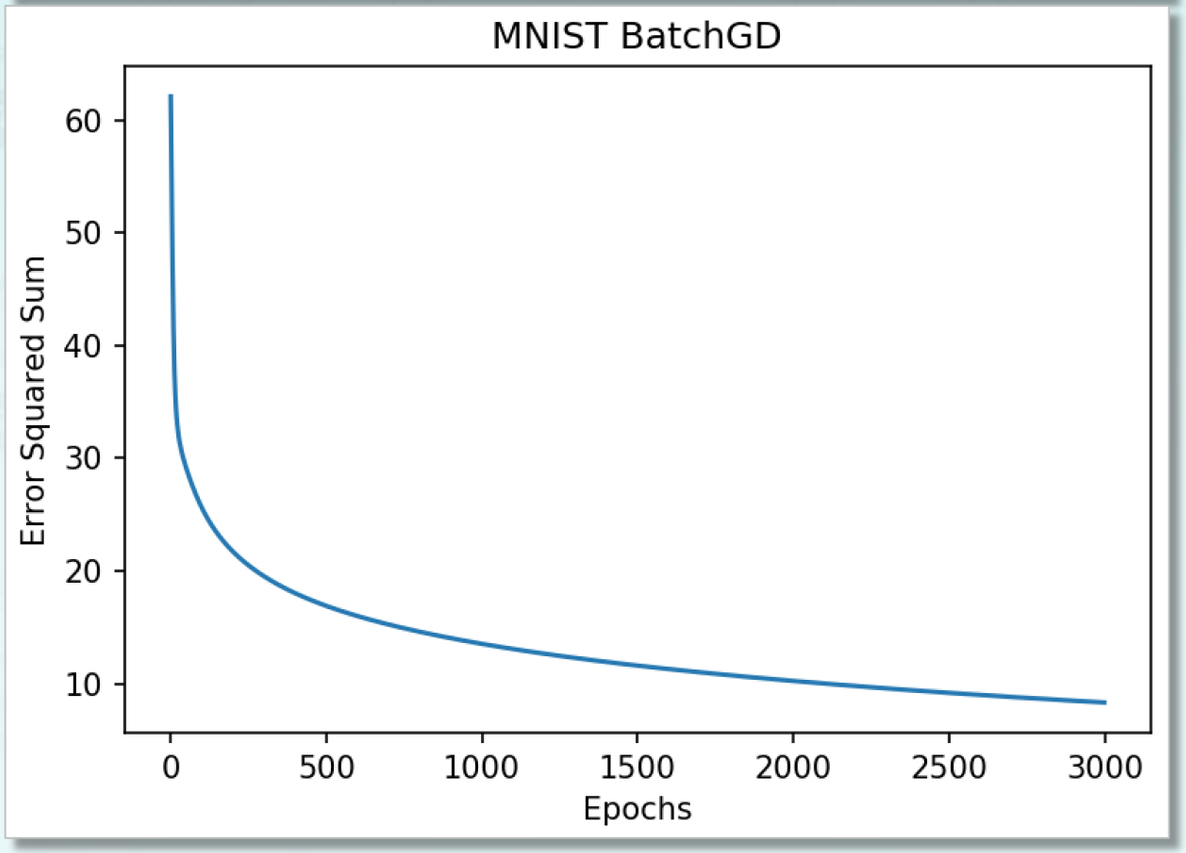
```
Accuracy 85.0%
```

1. Batch GD: MNIST 학습 결과

```
1  plt.plot(range(len(nn.cost_)), nn.cost_)
2  plt.title('MNIST BatchGD')
3  plt.xlabel('Epochs')
4  plt.ylabel('Error Squared Sum')
5  plt.show()
```

1. Batch GD: MNIST 학습 결과

```
1  plt.plot(range(len(nn.cost_)), nn.cost_)
2  plt.title('MNIST BatchGD')
3  plt.xlabel('Epochs')
4  plt.ylabel('Error Squared Sum')
5  plt.show()
```



1. Batch GD: 단점

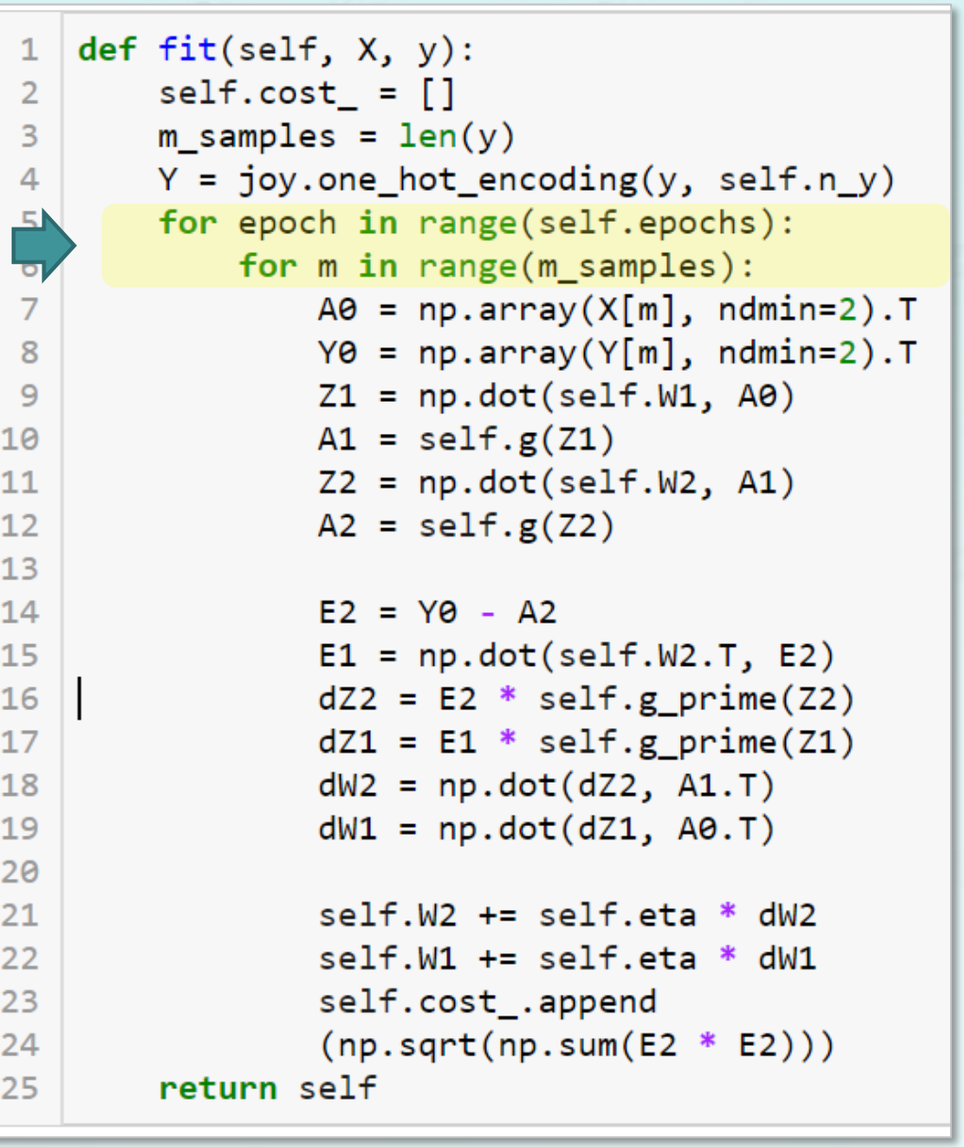
$$w_j := w_j + \eta \sum_{i}^{m} \left(y^{(i)} - h\left(z^{(i)}\right) \right) x_j^{(i)}$$

1. Batch GD: 단점

$$w_j := w_j + \eta \sum_i^m \left(y^{(i)} - h\left(z^{(i)}\right)\right) x_j^{(i)}$$

$$w_j := w_j + \eta\left(y^{(i)} - h(z^{(i)})\right)x_j^{(i)}$$

2. Stochastic GD: 학습 코드



```
def fit(self, X, y):
    self.cost_ = []
    m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.n_y)
    for epoch in range(self.epochs):
        for m in range(m_samples):
            A0 = np.array(X[m], ndmin=2).T
            Y0 = np.array(Y[m], ndmin=2).T
            Z1 = np.dot(self.W1, A0)
            A1 = self.g(Z1)
            Z2 = np.dot(self.W2, A1)
            A2 = self.g(Z2)

            E2 = Y0 - A2
            E1 = np.dot(self.W2.T, E2)
            dZ2 = E2 * self.g_prime(Z2)
            dZ1 = E1 * self.g_prime(Z1)
            dW2 = np.dot(dZ2, A1.T)
            dW1 = np.dot(dZ1, A0.T)

            self.W2 += self.eta * dW2
            self.W1 += self.eta * dW1
            self.cost_.append
            (np.sqrt(np.sum(E2 * E2)))
    return self
```

2. Stochastic GD: 학습 코드

```
1  def fit(self, X, y):
2      self.cost_ = []
3      m_samples = len(y)
4      Y = joy.one_hot_encoding(y, self.n_y)
5      for epoch in range(self.epochs):
6          for m in range(m_samples):
7              A0 = np.array(X[m], ndmin=2).T
8              Y0 = np.array(Y[m], ndmin=2).T
9              Z1 = np.dot(self.W1, A0)
10             A1 = self.g(Z1)
11             Z2 = np.dot(self.W2, A1)
12             A2 = self.g(Z2)
13
14             E2 = Y0 - A2
15             E1 = np.dot(self.W2.T, E2)
16             dZ2 = E2 * self.g_prime(Z2)
17             dZ1 = E1 * self.g_prime(Z1)
18             dW2 = np.dot(dZ2, A1.T)
19             dW1 = np.dot(dZ1, A0.T)
20
21             self.W2 += self.eta * dW2
22             self.W1 += self.eta * dW1
23             self.cost_.append
24             (np.sqrt(np.sum(E2 * E2)))
25     return self
```

2. Stochastic GD: 학습 코드

```
1  def fit(self, X, y):
2      self.cost_ = []
3      m_samples = len(y)
4      Y = joy.one_hot_encoding(y, self.n_y)
5      for epoch in range(self.epochs):
6          for m in range(m_samples):
7              A0 = np.array(X[m], ndmin=2).T
8              Y0 = np.array(Y[m], ndmin=2).T
9              Z1 = np.dot(self.W1, A0)
10             A1 = self.g(Z1)
11             Z2 = np.dot(self.W2, A1)
12             A2 = self.g(Z2)
13
14             E2 = Y0 - A2
15             E1 = np.dot(self.W2.T, E2)
16             dZ2 = E2 * self.g_prime(Z2)
17             dZ1 = E1 * self.g_prime(Z1)
18             dW2 = np.dot(dZ2, A1.T)
19             dW1 = np.dot(dZ1, A0.T)
20
21             self.W2 += self.eta * dW2
22             self.W1 += self.eta * dW1
23             self.cost_.append
24             (np.sqrt(np.sum(E2 * E2)))
25     return self
```

2. Stochastic GD: 학습 결과

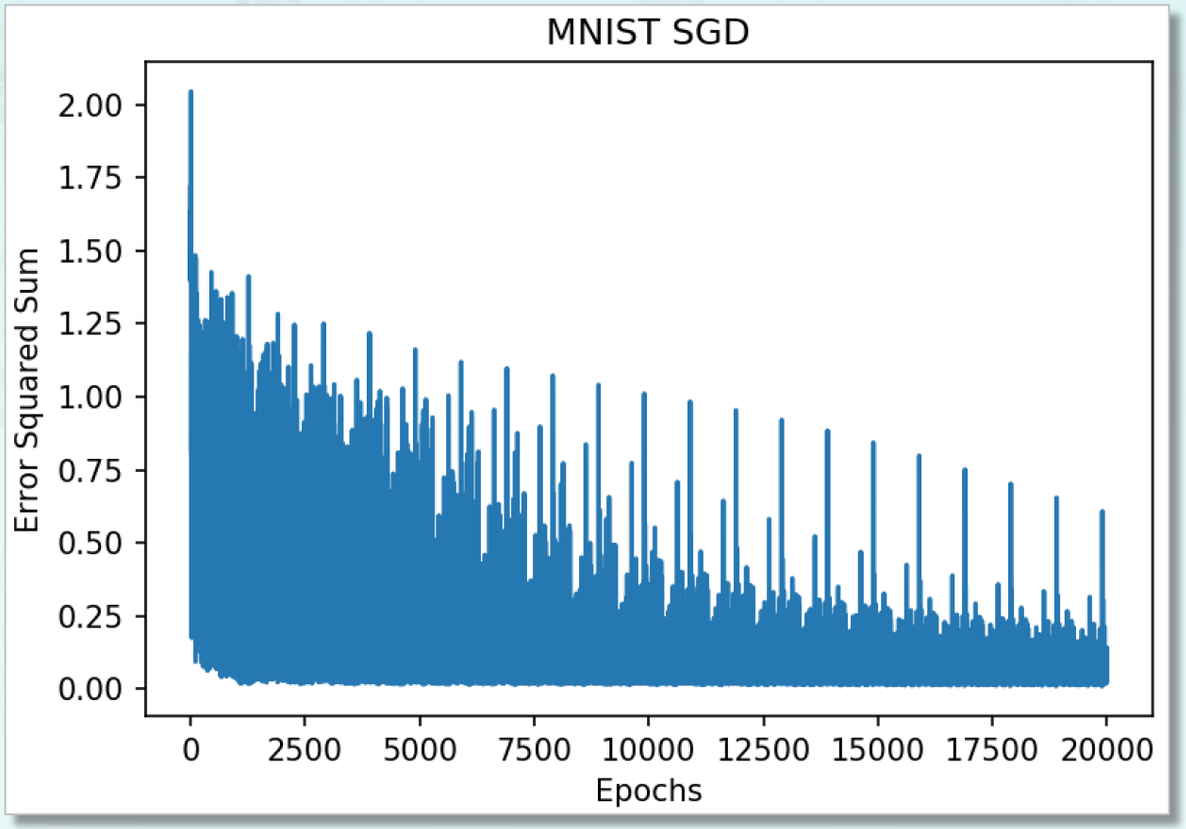
```
import joy
(X, y), (Xtest, ytest) = joy.load_mnist()

nn = MnistSGD([784, 100, 10], eta=0.1,
              epochs=20)
nn.fit(X[:1000], y[:1000])
acy = nn.evaluate(Xtest[:200], ytest[:200])
print('Accuracy {}%'.format(acy))
```

```
def evaluate(self, Xtest, ytest):
    m_samples = len(ytest)
    scores = 0
    A2 = self.predict(Xtest)
    yhat = np.argmax(A2, axis = 0)
    scores += np.sum(yhat == ytest)
    return scores/m_samples * 100
```

```
Accuracy 87.0%
```

2. Stochastic GD: 학습 결과

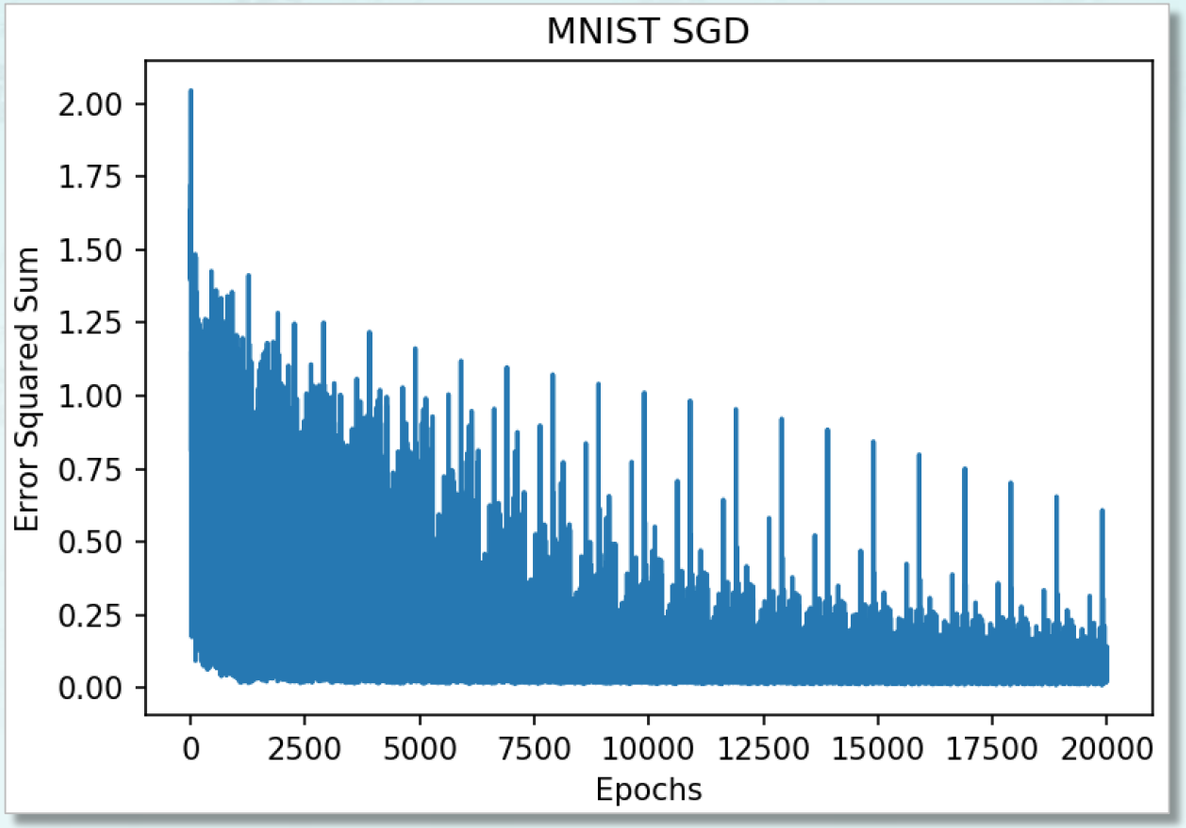
```
1  plt.plot(range(len(nn.cost_)), nn.cost_)
2  plt.title('MNIST SGD')
3  plt.xlabel('Epochs')
4  plt.ylabel('Error Squared Sum')
5  plt.show()
```



2. Stochastic GD: 학습 결과

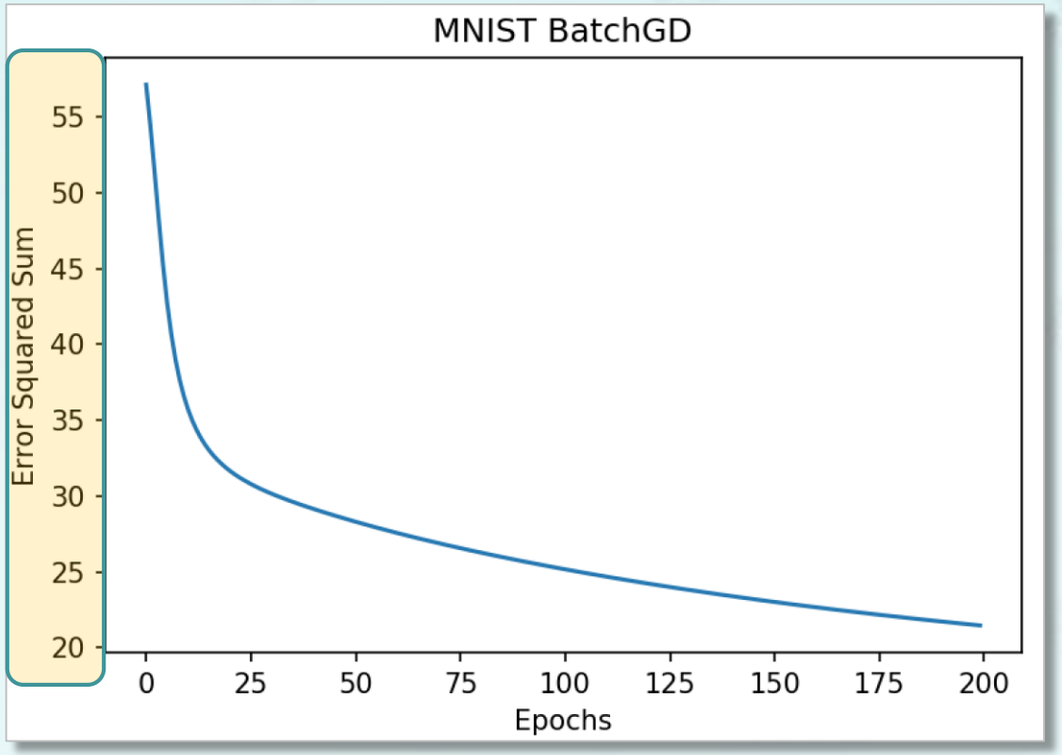
- $epochs \times number\ of\ samples$
  - $20 \times 1000 = 20{,}000$

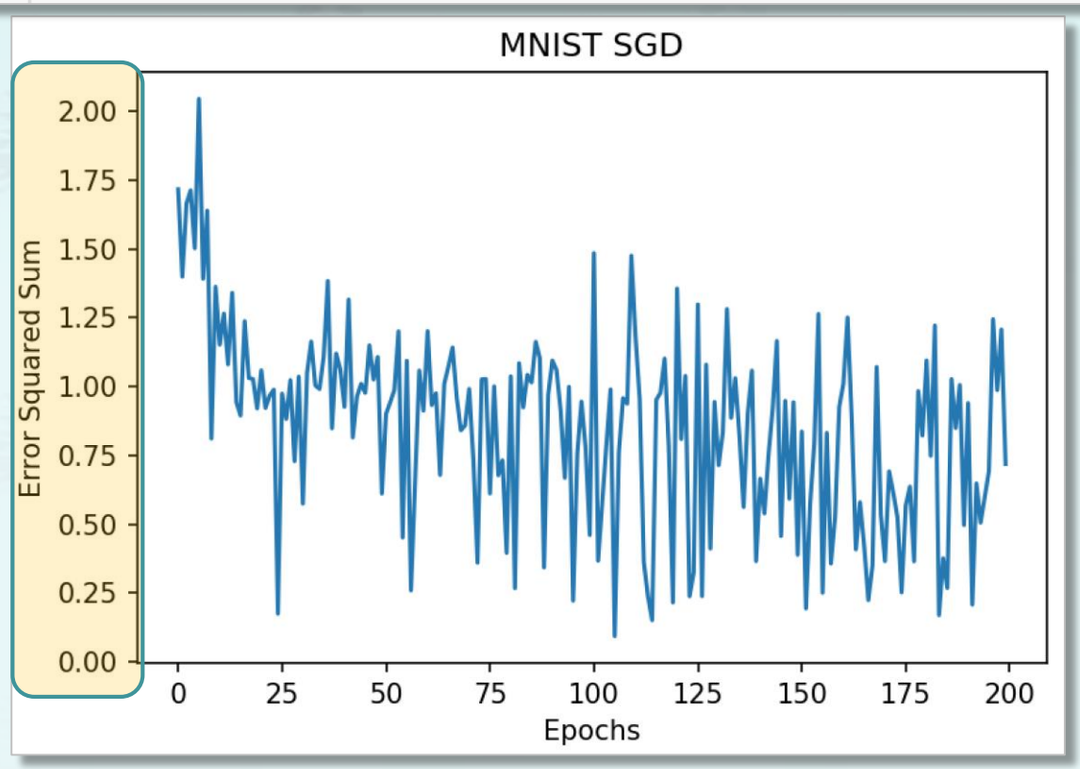
```
1  plt.plot(range(len(nn.cost_)), nn.cost_)
2  plt.title('MNIST SGD')
3  plt.xlabel('Epochs')
4  plt.ylabel('Error Squared Sum')
5  plt.show()
```



2. Stochastic GD: 학습 결과

```
plt.plot(range(len(nn.cost_[:200])),
         nn.cost_[:200])
plt.title('MNIST SGD')
plt.xlabel('Epochs')
plt.ylabel('Error Squared Sum')
plt.show()
```





Batch GD, Stochastic GD

- 학습 정리
  - 배치 경사하강법**(Batch GD)** 이해
  - 확률적 경사하강법**(Stochastic GD)** 이해
  - 두 알고리즘의 차이점