

11주차(3/3)

# MNIST 경사하강법 1

파이썬으로 배우는 기계학습

한동대학교  
김영섭 교수

# MNIST 경사하강법

---

- 학습 목표

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

- 학습 내용

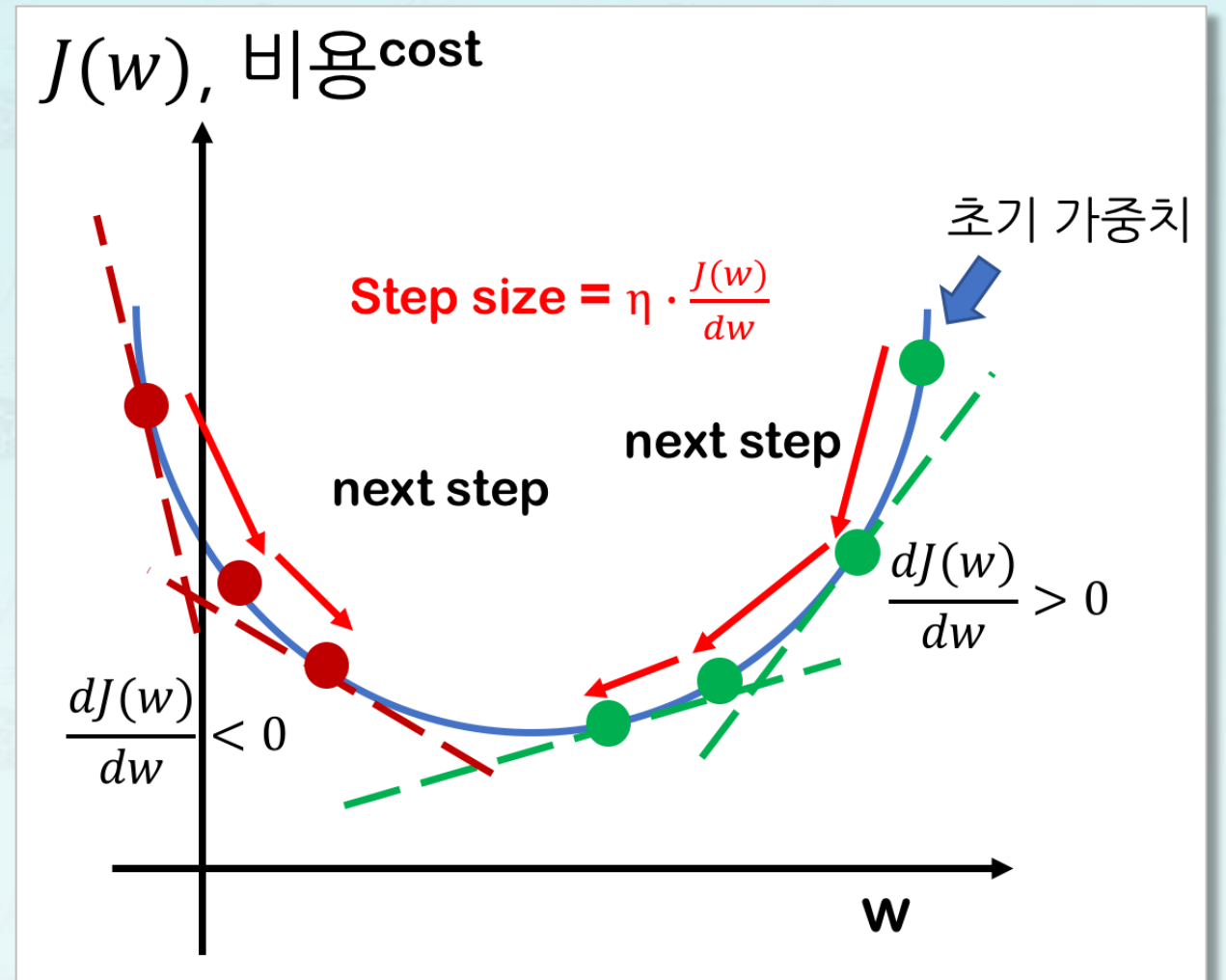
- **MNIST** 자료셋에 대한 다양한 경사하강법 비교하기
- 배치 경사하강법(**Batch GD**)의 정확도 이해하기
- 확률적 경사하강법(**Stochastic GD**)으로 학습하기

# 1. Batch GD: 오차함수 $J(w)$

---

# 1. Batch GD: 오차함수 $J(w)$

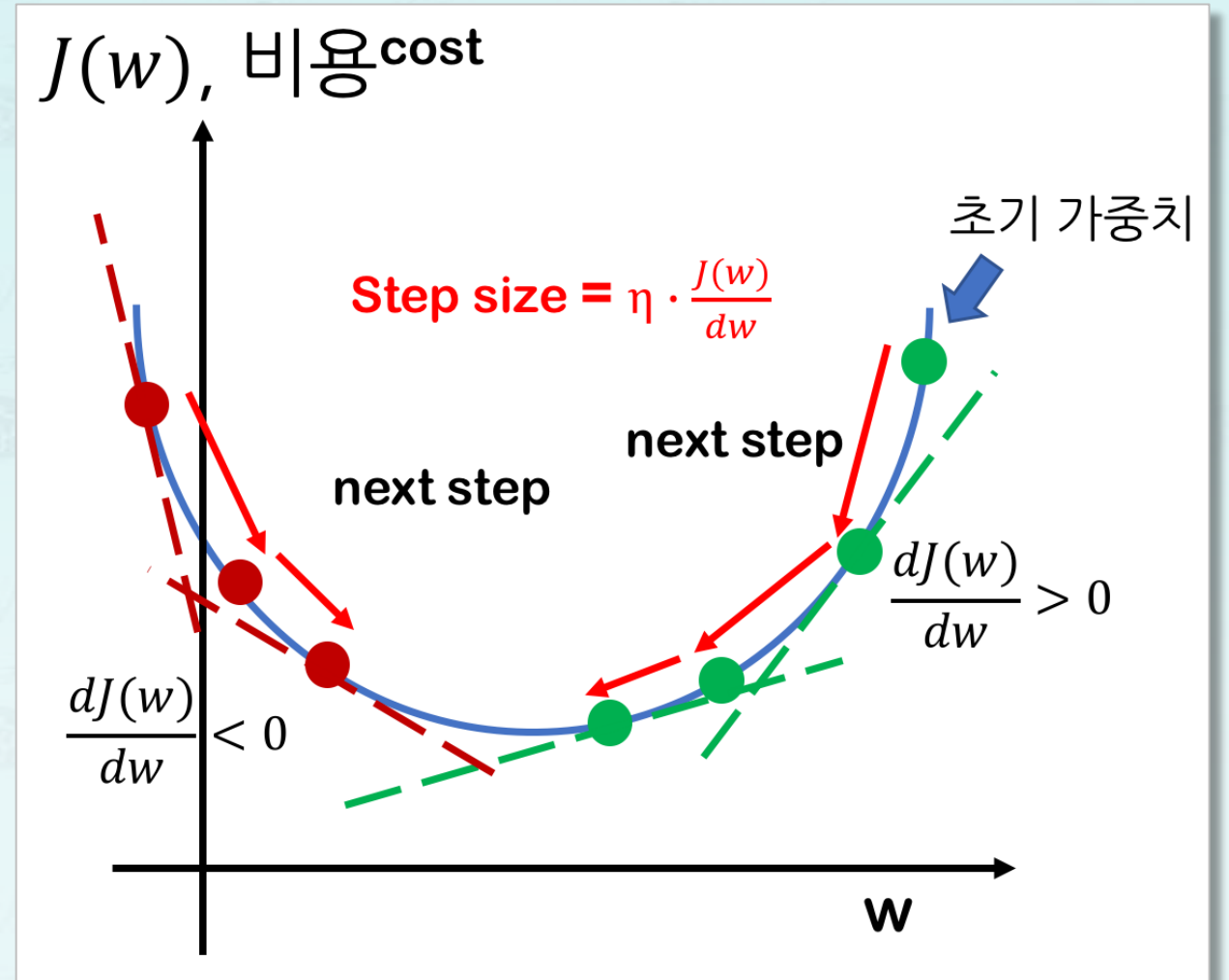
$$\Delta w = -\eta \frac{\partial J(w)}{\partial w_j}$$



# 1. Batch GD: 오차함수 $J(w)$

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

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



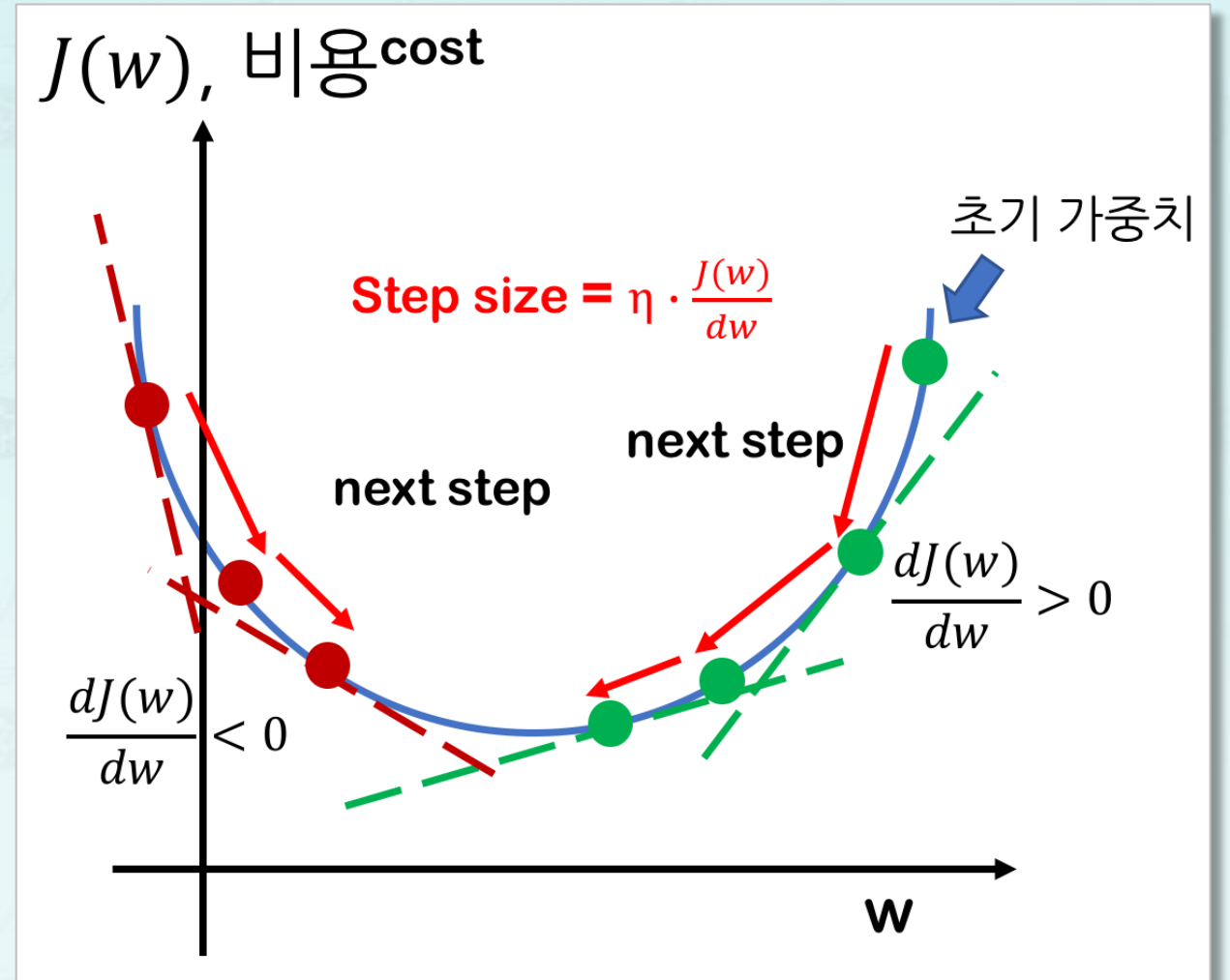
# 1. Batch GD: 오차함수 $J(w)$

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

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

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

- 평균제곱오차(MSE)
  - Mean squared error



# 1. Batch GD: MNIST 학습 코드

```
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: MNIST 학습 코드

```
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: MNIST 학습 코드

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

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

```

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: MNIST 학습 코드

```
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: MNIST 학습 코드

```
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: 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: 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: MNIST 학습 결과

$A2 =$

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

```
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

$A2 =$

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

```
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

$A2 =$

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

↑ 샘플의 수 100  
한 샘플의 예측값 10

```
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:**

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

```
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

```
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
```

# 배열의 축 Axis 다루기

- 2차원 배열 Axis

axis 1 →

axis 0 →	0	1	2
	3	4	5

2차원 배열 Shape(2, 3)

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

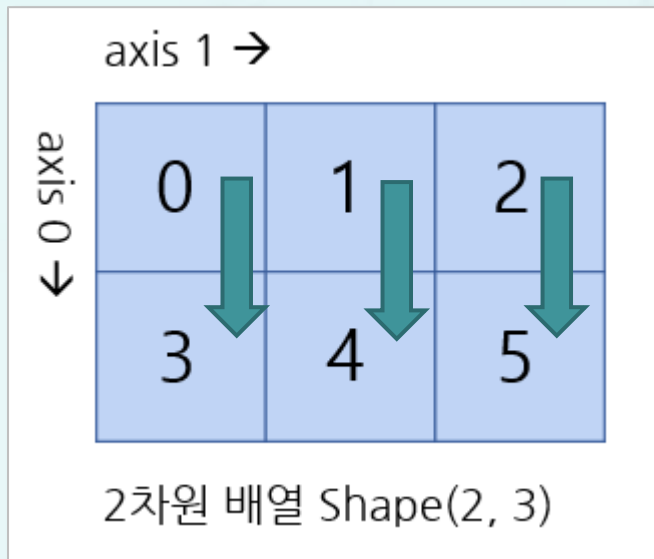
```
sum(axis=0):
```

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

**(3, 5, 7)**

# 배열의 축 Axis 다루기

- 2차원 배열 Axis



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

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

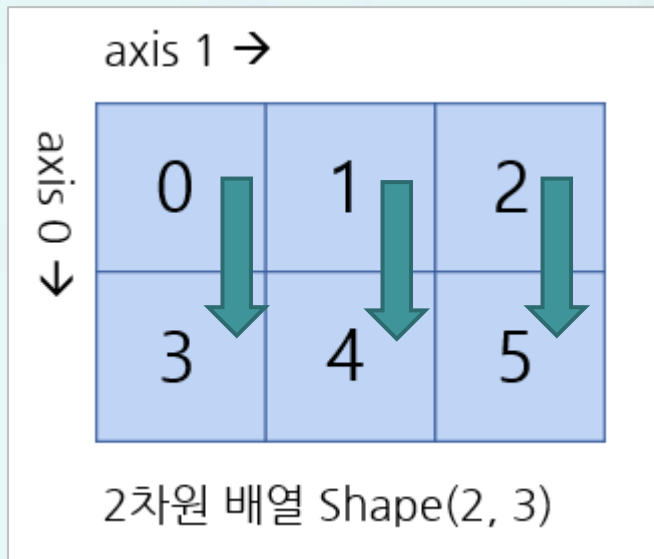
**np.sum(a, axis = 0) = ? (3, 12)**

**(3, 5, 7)**



# 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

$A2 =$

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

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, )`

axis = 0

$A2 =$

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

`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, )`

axis = 0

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

`yhat = [3, 2, 9, 2, 0, ... 3]`

```
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

```
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

```
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

```
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
```

Accuracy 85.0%

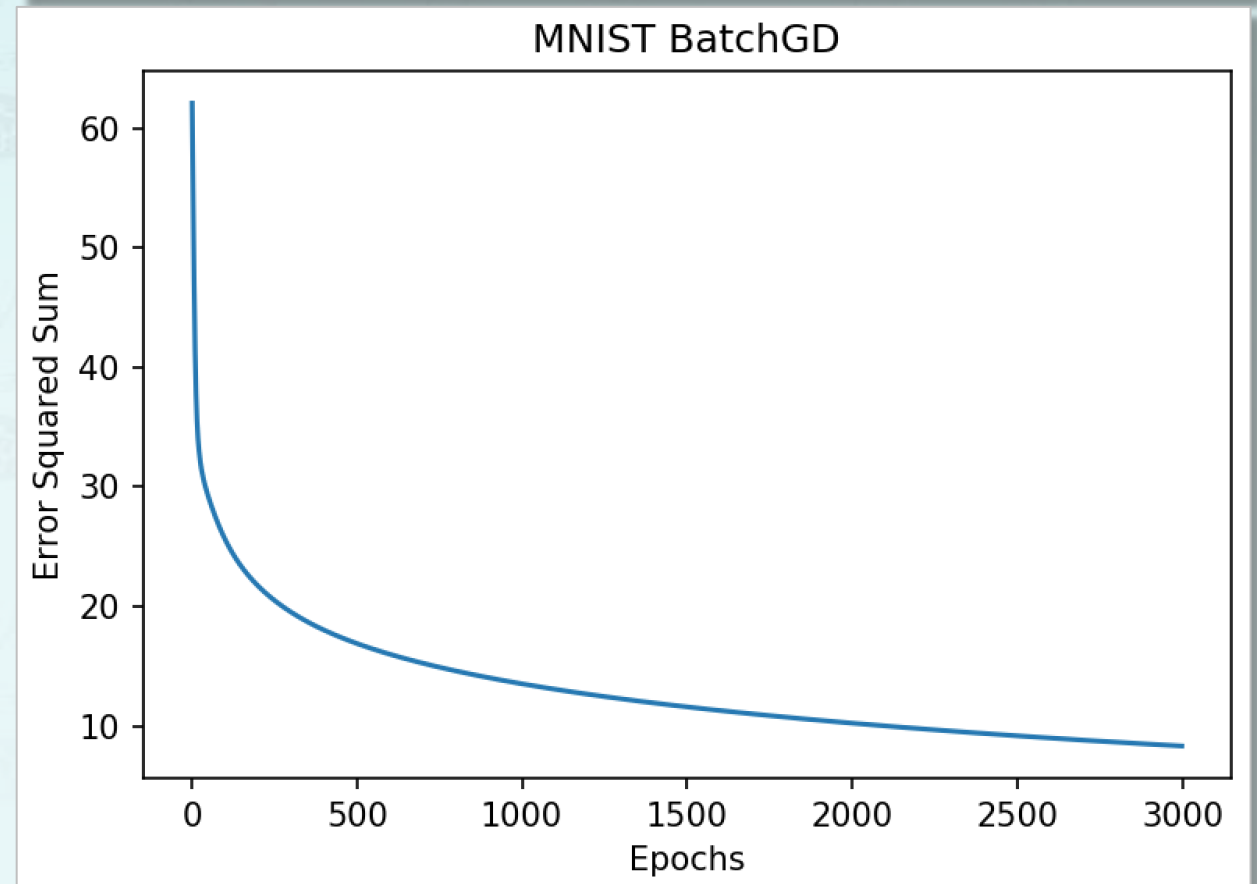
# 1. Batch GD: MNIST 학습 결과

```
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

---

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



# 1. Batch GD: 단점

---


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

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

## 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 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

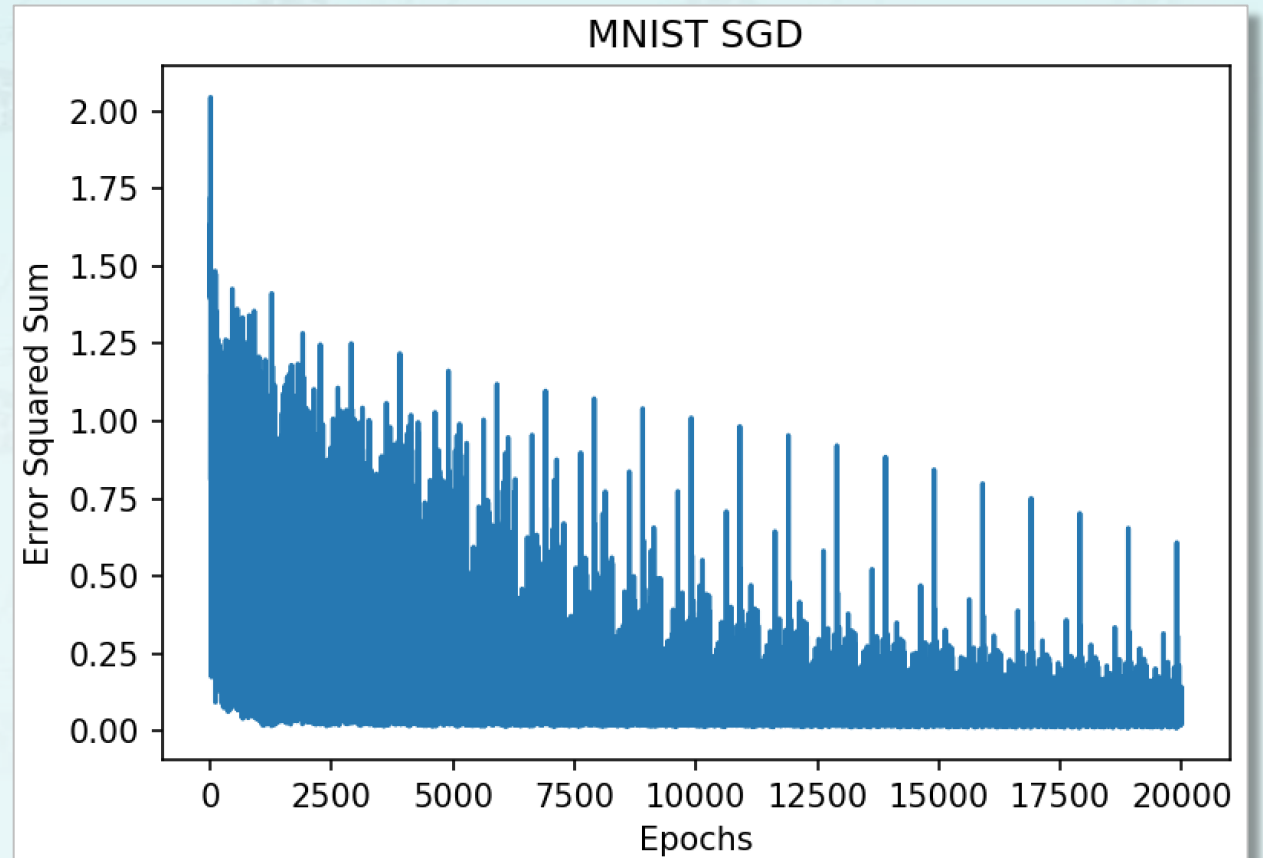
```
1 import joy
2 (X, y), (Xtest, ytest) = joy.load_mnist()
3
4 nn = MnistSGD([784, 100, 10], eta=0.1,
5               epochs=20)
6 nn.fit(X[:1000], y[:1000])
7 acy = nn.evaluate(Xtest[:200], ytest[:200])
8 print('Accuracy {}'.format(acy))
```

```
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
```

Accuracy 87.0%

## 2. Stochastic GD: 학습 결과

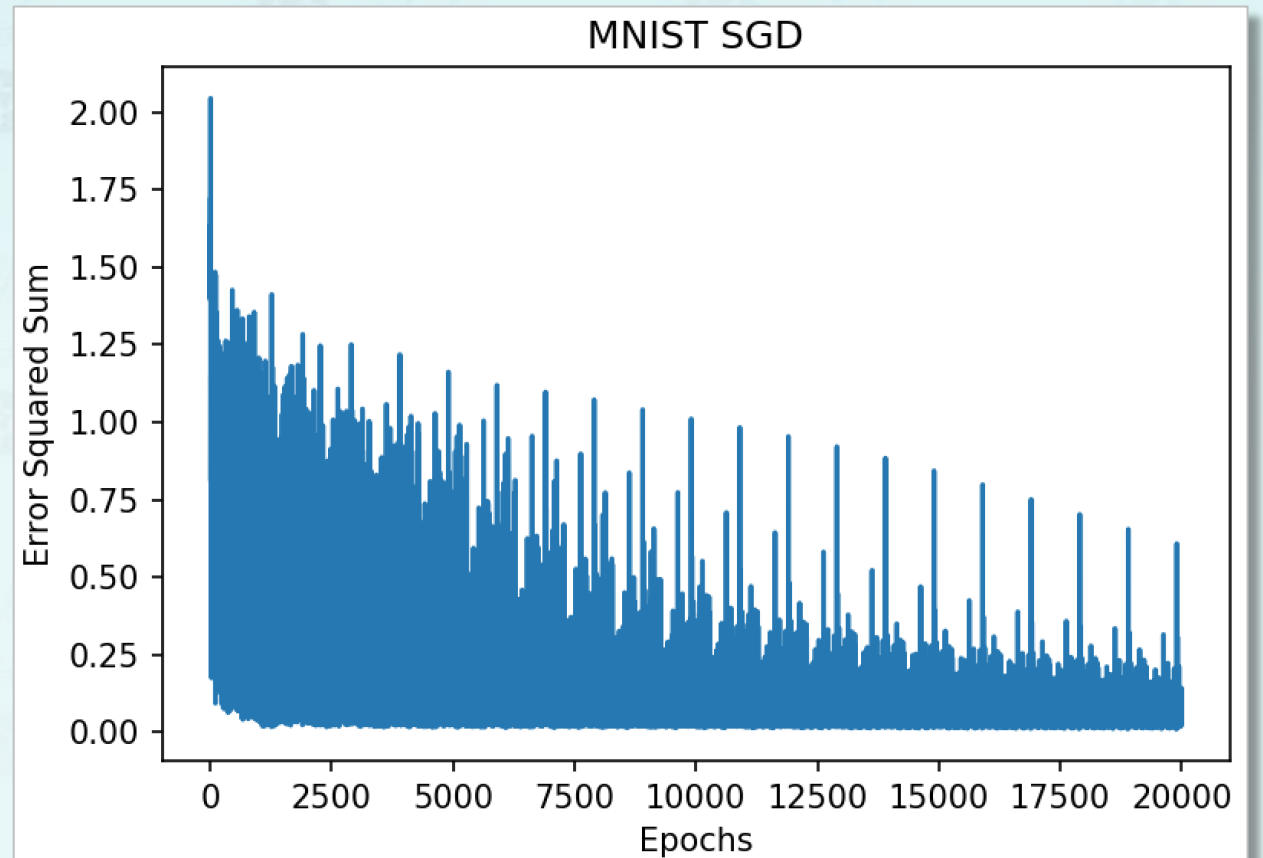
```
1 plt.plot(range(len(nn.cost_)), nn.cost_)
2 plt.title('MNIST SGD')
3 plt.xlabel('Epochs')
4 plt.ylabel('Error Squared Sum')
5 plt.show()
```



## 2. Stochastic GD: 학습 결과

- *epochs* × *number of samples*
  - $20 \times 1000 = 20000$

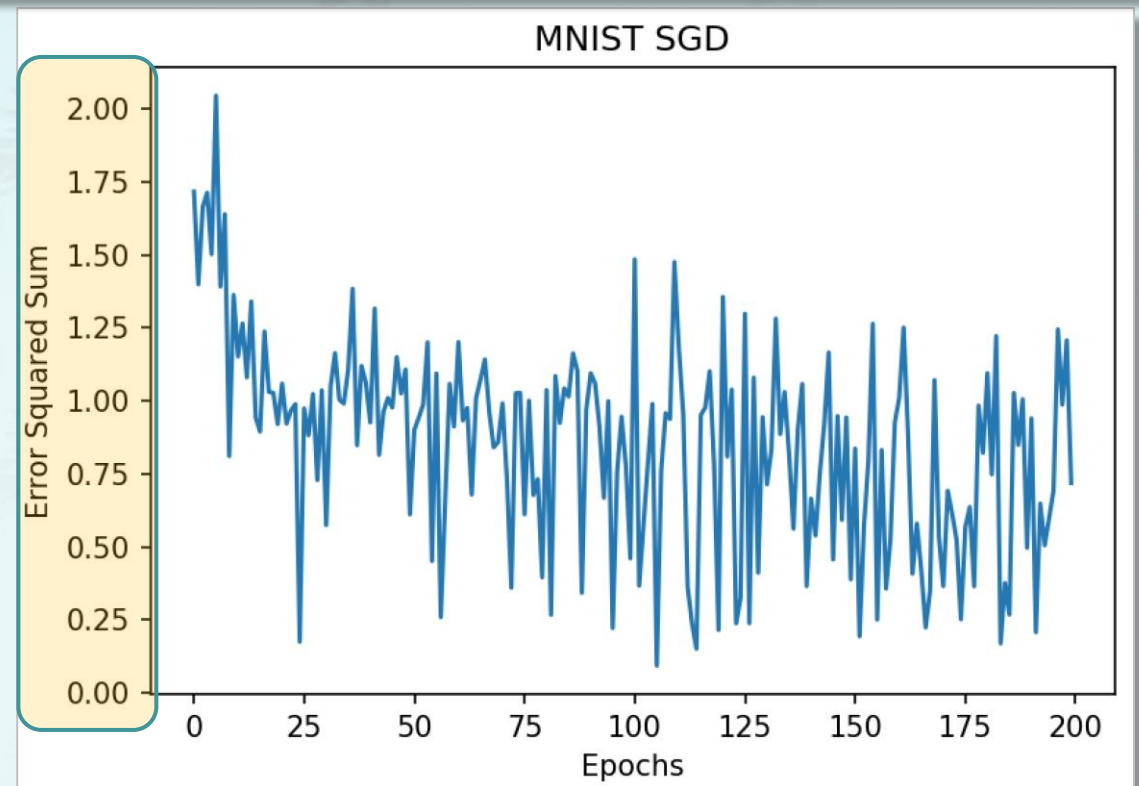
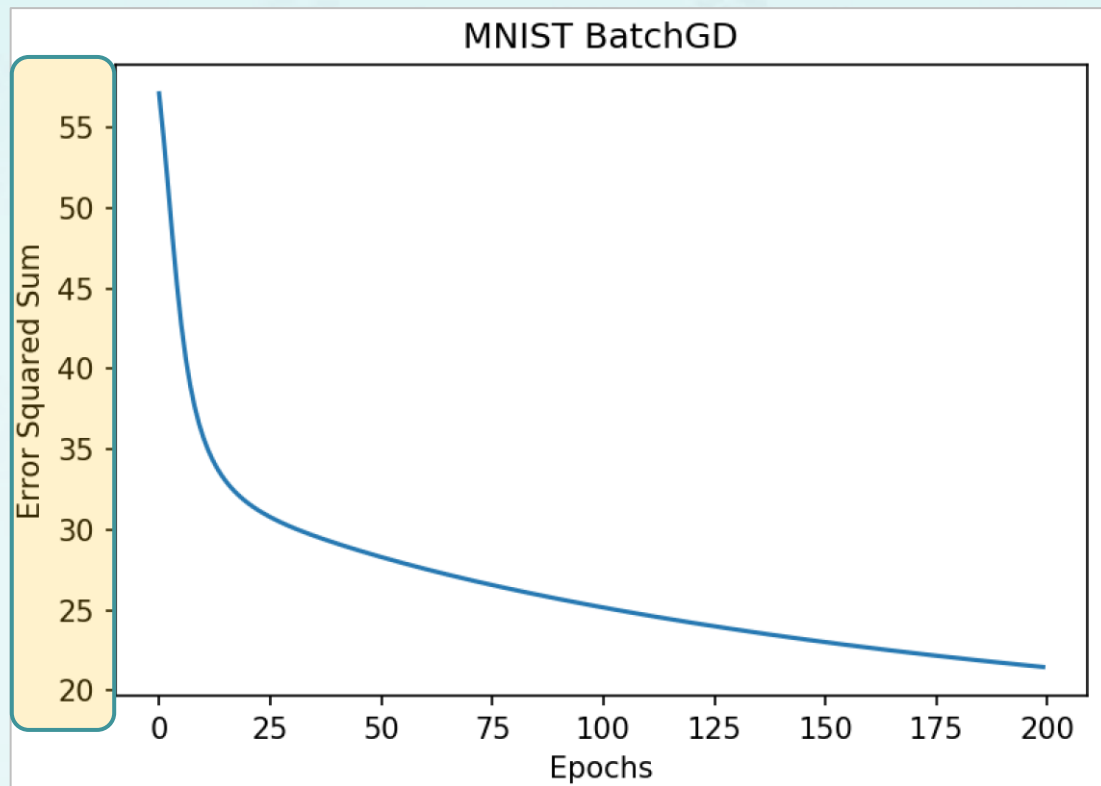
```
1 plt.plot(range(len(nn.cost_)), nn.cost_)
2 plt.title('MNIST SGD')
3 plt.xlabel('Epochs')
4 plt.ylabel('Error Squared Sum')
5 plt.show()
```





## 2. Stochastic GD: 학습 결과

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





# Batch GD, Stochastic GD

---

- 학습 정리
  - 배치 경사하강법(**Batch GD**)으로 학습의 정확도를 이해하기
  - 확률적 경사하강법(**Stochastic GD**)으로 학습의 효율성을 이해하기
- 차시 예고
  - **12-1 MNIST 경사하강법 2**

11주차(3/3)

# MNIST 경사하강법 1

파이썬으로 배우는 기계학습

한동대학교  
김영섭 교수