

12주차(1/2)

# 경사하강법 2

파이썬으로 배우는 기계학습

한동대학교  
김영섭 교수

# 경사하강법 2

---

- 학습 목표
  - 미니배치(**Mini-Batch**) 경사하강법을 학습한다.
  - 다양한 경사하강법들의 차이점을 학습한다.
  - 과대적합(**Overfitting**) 원인과 해결 방법을 학습한다.
- 학습 내용
  - 미니배치(**Mini-Batch**) 경사하강법
  - 경사하강법들의 장단점과 학습률
  - 과대적합(**Overfitting**) 원인과 해결 방법

# 1. 경사하강법 비교: 배치 경사하강법

- 모든 샘플의 오차 총합
- 가중치 조정
- 반복으로 오차 줄임

$$J(w) = \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)})^2$$

$$w_{new} = w_{old} + \eta \frac{\partial J(w)}{\partial w_j}$$

$$= w_{old} + \eta \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

# 1. 경사하강법 비교: 배치 경사하강법

- 단점
  - 메모리 문제
  - 훈련 자료가 많아지면 학습 속도가 느려짐
- 장점
  - 안정적인 학습
  - 오차함수의 최소 값으로 수렴할 가능성 높음

$$J(w) = \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)})^2$$

$$w_{new} = w_{old} + \eta \frac{\partial J(w)}{\partial w_j}$$

$$= w_{old} + \eta \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

# 1. 경사하강법 비교: 확률적 경사하강법

- 배치 경사하강법

- 모든 샘플의 오차 총합

$$J(w) = \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)})^2$$

$$w_{new} = w_{old} + \eta \frac{\partial J(w)}{\partial w_j}$$

- 확률적 경사하강법

- 각 샘플의 오차로 가중치 조정
- 메모리 문제 없음
- 수렴 속도가 빠름
- 불안정한 수렴
- 대규모 기계학습에 적합

$$= w_{old} + \eta \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

$$w_{new} = w_{old} + \eta (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

# 1. 경사하강법 비교: 미니 배치 경사하강법

- 배치 경사하강법
  - 모든 샘플의 오차 총합
- 확률적 경사하강법
  - 각 샘플의 오차
- 미니 배치 경사하강법
  - 일정 샘플들(미니 배치)의 오차 총합
  - 미니 배치 크기  
(bs = 8, 16, 32, 64...)
  - GPU/Numpy의 효율적 배열 처리

$$J(w) = \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)})^2$$

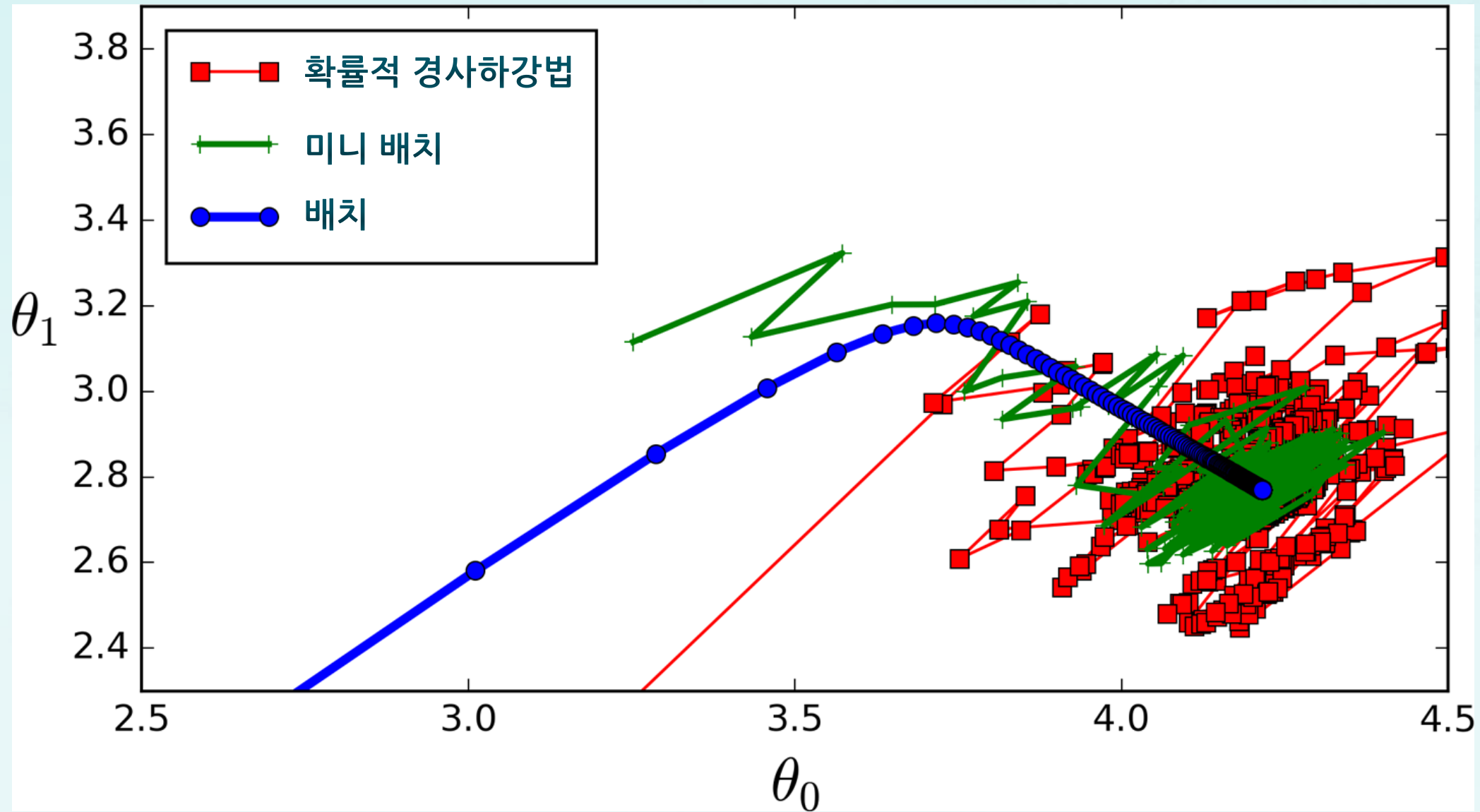
$$w_{new} = w_{old} + \eta \frac{\partial J(w)}{\partial w_j}$$

$$= w_{old} + \eta \frac{1}{m} \sum_i^m (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

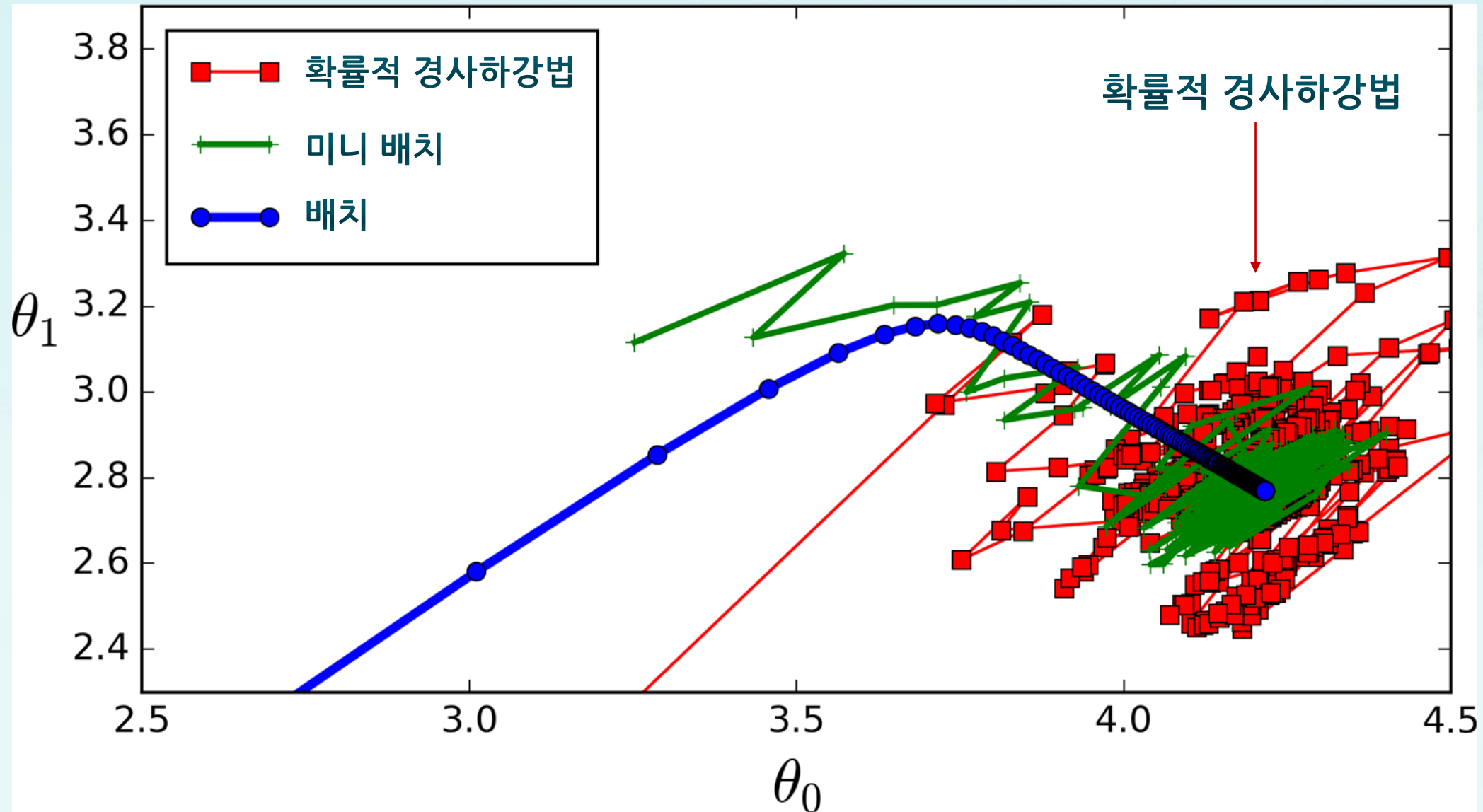
$$w_{new} = w_{old} + \eta (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

$$w_{new} = w_{old} + \eta \frac{1}{bs} \sum_i^{bs} (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

# 1. 경사하강법 비교: 시각적 비교



# 1. 경사하강법 비교: 학습 스케줄





## 2. 학습 스케줄링

---

- 학습률의 빠른 감소
  - 지역 최소값에서 멈춤
- 학습률의 느린 감소
  - 전역 최소값에 도달하지 못함

## 2. 학습 스케줄링 : 4가지 방법

---

1. 미리 정의된 고정 학습 스케줄링
2. 성능 기반 학습 스케줄링
3. 지수 기반 학습 스케줄링

$$\eta(t) = \eta_0 10^{-t/r}$$

4. 거듭제곱 기반 학습 스케줄링

$$\eta(t) = \eta_0 \left(1 + \frac{t}{r}\right)^{-c}$$

## 2. 학습 스케줄링 : 코딩

```
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      → eta_scheduled = np.linspace(self.eta,
6                                     0.0001, self.epochs)
7      for epoch in range(self.epochs):
8          A0 = np.array(X, ndmin=2).T
9          Y0 = np.array(Y, ndmin=2).T
10
11         Z1, A1, Z2, A2 = self.forpass(A0)
12         E2 = Y0 - A2
13         E1 = np.dot(self.W2.T, E2)
14
15         dZ2 = E2 * self.g_prime(Z2)
16         dZ1 = E1 * self.g_prime(Z1)
17
18      → eta = eta_scheduled[epoch]
19         self.W2 += eta * np.dot(dZ2, A1.T) / m_samples
20         self.W1 += eta * np.dot(dZ1, A0.T) / m_samples
21         self.cost_.append(np.sqrt(np.sum(E2 * E2)))
22      return self
```

### 3. 미니 배치 경사하강법: 코드 설명

- 각 epoch마다
  - 각 미니 배치 크기로 나누어 연산
  - **batch\_size = 8, 16, 32, 64, ...**

```
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 i in range(0, m_samples, self.batch_size):
7             A0 = X[i: i + self.batch_size]
8             Y0 = Y[i: i + self.batch_size]
9             A0 = np.array(X[m], ndmin=2).T
10            Y0 = np.array(Y[m], ndmin=2).T
11            Z1 = np.dot(self.W1, A0)
12            A1 = self.g(Z1)
13            Z2 = np.dot(self.W2, A1)
14            A2 = self.g(Z2)
15
16            E2 = Y0 - A2
17            E1 = np.dot(self.W2.T, E2)
18            dZ2 = E2 * self.g_prime(Z2)
19            dZ1 = E1 * self.g_prime(Z1)
20            dW2 = np.dot(dZ2, A1.T)
21            dW1 = np.dot(dZ1, A0.T)
22            self.W2 += self.eta * dW2/self.batch_size
23            self.W1 += self.eta * dW1/self.batch_size
24            self.cost_.append(np.sqrt(np.sum(E2 * E2))
25                               /self.batch_size)
26     return self
```

### 3. 미니 배치 경사하강법: 코드 설명

- 각 **epoch**마다
  - 각 미니 배치 크기로 나누어 연산
  - **batch\_size = 8, 16, 32, 64, ...**
  - 입력을 **batch\_size**로 슬라이싱

```
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 i in range(0, m_samples, self.batch_size):
7             A0 = X[i: i + self.batch_size]
8             Y0 = Y[i: i + self.batch_size]
9             A0 = np.array(X[m], ndmin=2).T
10            Y0 = np.array(Y[m], ndmin=2).T
11            Z1 = np.dot(self.W1, A0)
12            A1 = self.g(Z1)
13            Z2 = np.dot(self.W2, A1)
14            A2 = self.g(Z2)
15
16            E2 = Y0 - A2
17            E1 = np.dot(self.W2.T, E2)
18            dZ2 = E2 * self.g_prime(Z2)
19            dZ1 = E1 * self.g_prime(Z1)
20            dW2 = np.dot(dZ2, A1.T)
21            dW1 = np.dot(dZ1, A0.T)
22            self.W2 += self.eta * dW2/self.batch_size
23            self.W1 += self.eta * dW1/self.batch_size
24            self.cost_.append(np.sqrt(np.sum(E2 * E2))
25                               /self.batch_size)
26        return self
```

### 3. 미니 배치 경사하강법: 코드 설명


- 각 **epoch**마다
  - 각 미니 배치 크기로 나누어 연산
  - **batch\_size = 8, 16, 32, 64, ...**
  - 입력을 **batch\_size**로 슬라이싱
  - 가중치를 조정함

```
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 i in range(0, m_samples, self.batch_size):
7             A0 = X[i: i + self.batch_size]
8             Y0 = Y[i: i + self.batch_size]
9             A0 = np.array(X[m], ndmin=2).T
10            Y0 = np.array(Y[m], ndmin=2).T
11            Z1 = np.dot(self.W1, A0)
12            A1 = self.g(Z1)
13            Z2 = np.dot(self.W2, A1)
14            A2 = self.g(Z2)
15
16            E2 = Y0 - A2
17            E1 = np.dot(self.W2.T, E2)
18            dZ2 = E2 * self.g_prime(Z2)
19            dZ1 = E1 * self.g_prime(Z1)
20            dW2 = np.dot(dZ2, A1.T)
21            dW1 = np.dot(dZ1, A0.T)
22
23            self.W2 += self.eta * dW2/self.batch_size
24            self.W1 += self.eta * dW1/self.batch_size
25            self.cost_.append(np.sqrt(np.sum(E2 * E2))
26                               /self.batch_size)
27
28 return self
```



### 3. 미니 배치 경사하강법: 학습 결과

- 검증단계
  - 학습자료 1000, 테스트 자료 100



```
1 (X, y), (Xtest, ytest) = joy.load_mnist()
2 nn = MnistMiniBatch(784, 100, 10,
3                     epochs = 20, batch_size = 32)
4 nn.fit(X[:1000], y[:1000])
5 accuracy = nn.evaluate(Xtest[:100], ytest[:100])
6 print('accuracy {}%'.format(accuracy))
```

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

### 3. 미니 배치 경사하강법: 학습 결과

- 검증단계
  - 학습자료 **1000**, 테스트 자료 **100**
  - 정확도: **87.0 %**

```
1 (X, y), (Xtest, ytest) = joy.load_mnist()
2 nn = MnistMiniBatch(784, 100, 10,
3                     epochs = 20, batch_size = 32)
4 nn.fit(X[:1000], y[:1000])
5 accuracy = nn.evaluate(Xtest[:100], ytest[:100])
6 print('accuracy {}%'.format(accuracy))
```



accuracy 87.0%

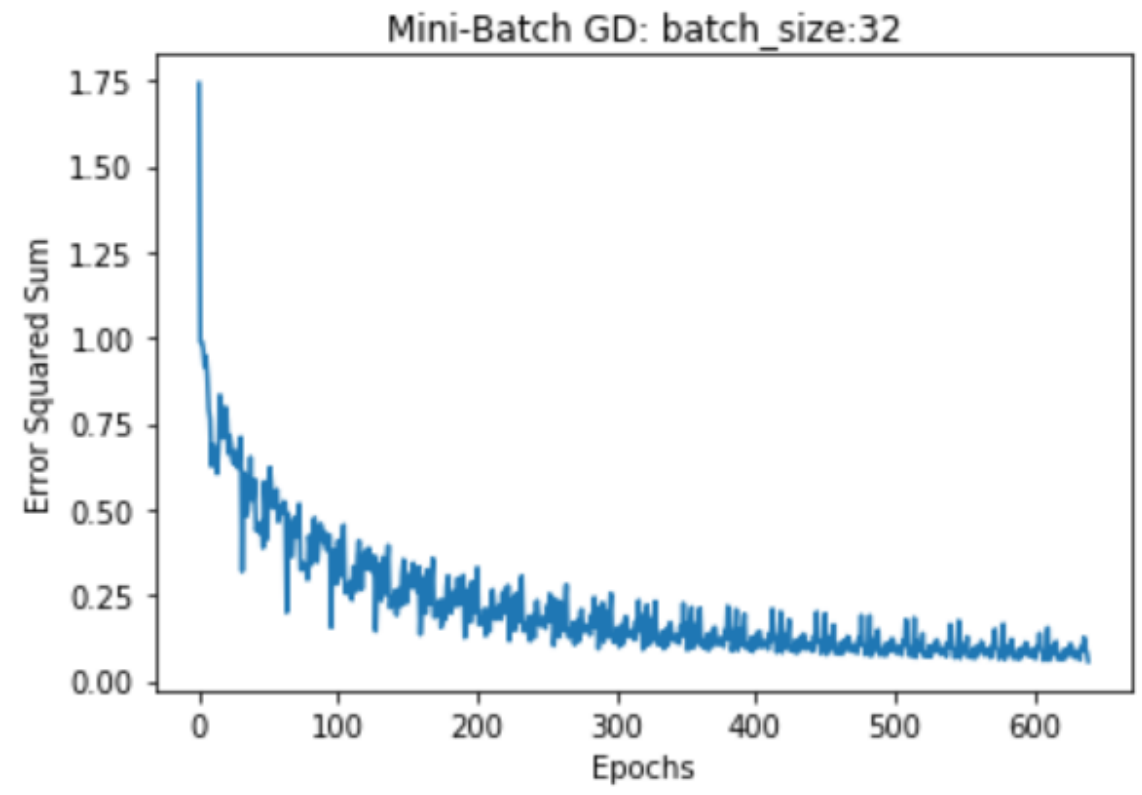
```
75     def evaluate(self, Xtest, ytest):
76         m_samples = len(ytest)
77         scores = 0
78         A2 = self.predict(Xtest)
79         yhat = np.argmax(A2, axis = 0)
80         scores += np.sum(yhat == ytest)
81         return scores/m_samples * 100
```



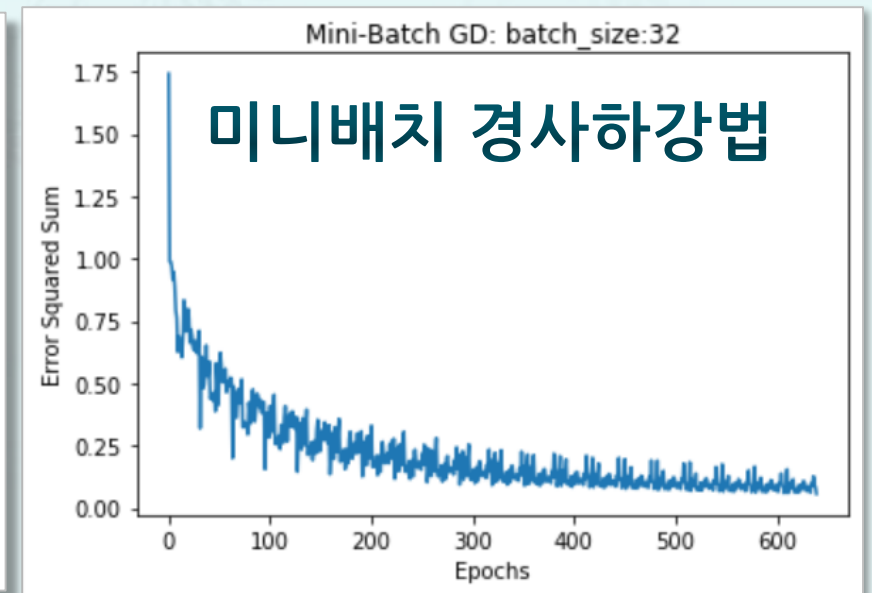
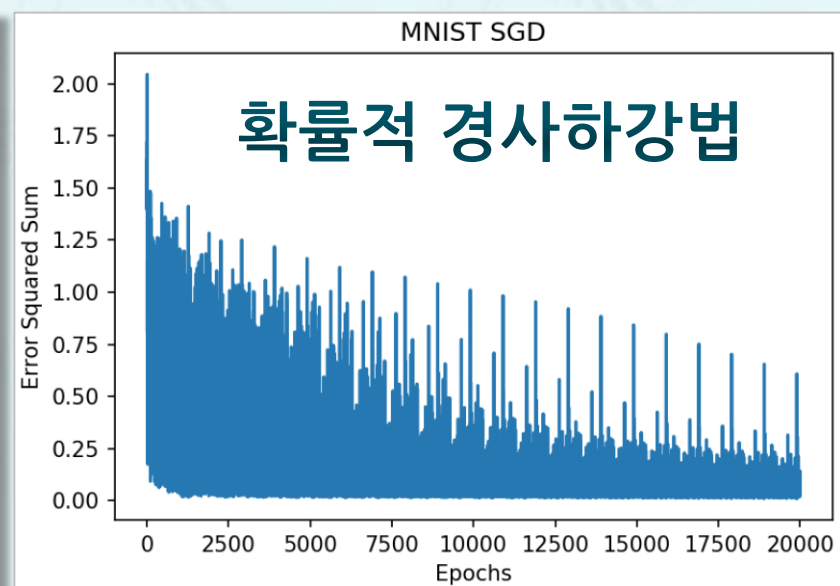
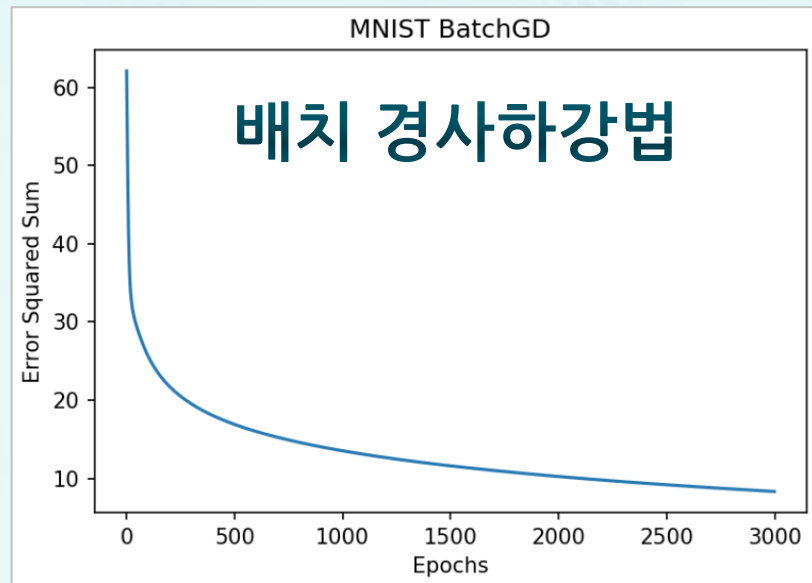
### 3. 미니 배치 경사하강법: 학습 결과 그래프

- 검증단계
  - 학습자료 **1000**, 테스트 자료 **100**
  - 정확도: **87.0 %**

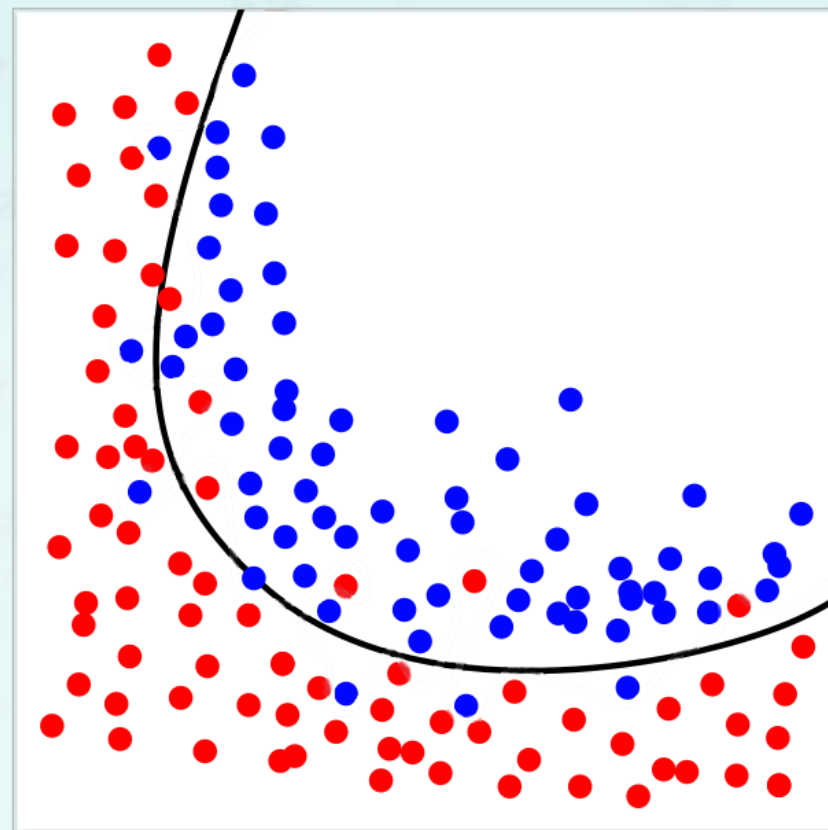
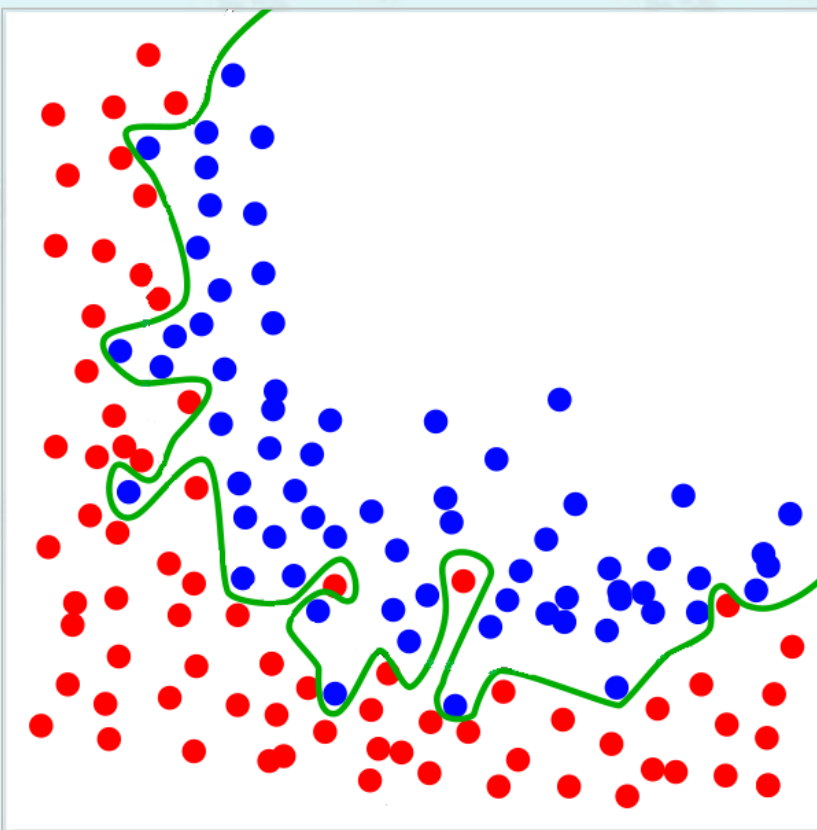
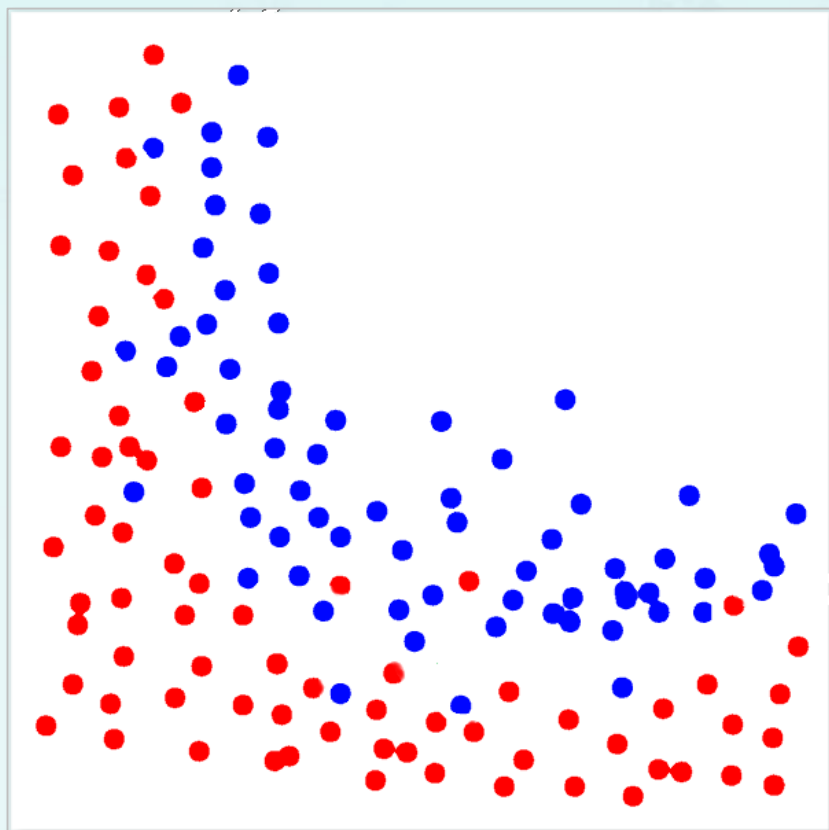
```
1 plt.plot(range(len(nn.cost_)), nn.cost_)
2 plt.xlabel('Epochs')
3 plt.ylabel('Error Squared Sum')
4 plt.title('Mini-Batch GD: batch_size:{}'.format(nn.batch_size))
5
6 plt.show()
```



### 3. 세 종류의 경사하강법 실행 비교 : 학습 결과그래프



## 4. 과대적합: 개념 설명




## 4. 과대적합: 원인 – 반복 횟수 증가

---

- 실험: 학습 자료 **vs** 테스트 자료
  - 은닉층 노드의 수: **100**개로 고정
  - 반복 횟수에 따른 정확도 비교

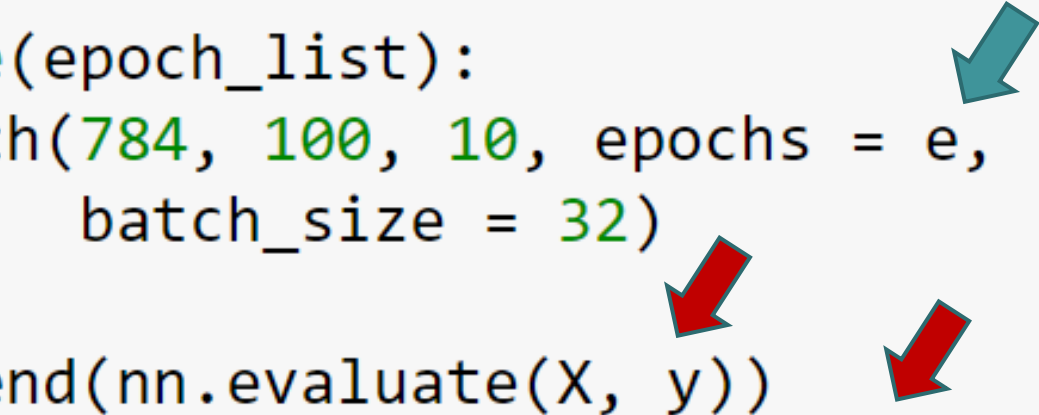
## 4. 과대적합: 코드 - 반복횟수



```
1 (X, y), (Xtest, ytest) = joy.load_mnist()
2 epoch_list = [i for i in np.arange(1, 31, 2)] + \
3               [i for i in np.arange(40, 101, 20)]
4 self_accuracy = []
5 test_accuracy = []
6 for i, e in enumerate(epoch_list):
7     nn = MnistMiniBatch(784, 100, 10, epochs = e,
8                         | batch_size = 32)
9     nn.fit(X, y)
10    self_accuracy.append(nn.evaluate(X, y))
11    test_accuracy.append(nn.evaluate(Xtest, ytest))
```

## 4. 과대적합: 코드 - 정확도

```
1 (X, y), (Xtest, ytest) = joy.load_mnist()
2 epoch_list = [i for i in np.arange(1, 31, 2)] + \
3               [i for i in np.arange(40, 101, 20)]
4 self_accuracy = []
5 test_accuracy = []
6 for i, e in enumerate(epoch_list):
7     nn = MnistMiniBatch(784, 100, 10, epochs = e,
8                         | batch_size = 32)
9     nn.fit(X, y)
10    self_accuracy.append(nn.evaluate(X, y))
11    test_accuracy.append(nn.evaluate(Xtest, ytest))
```

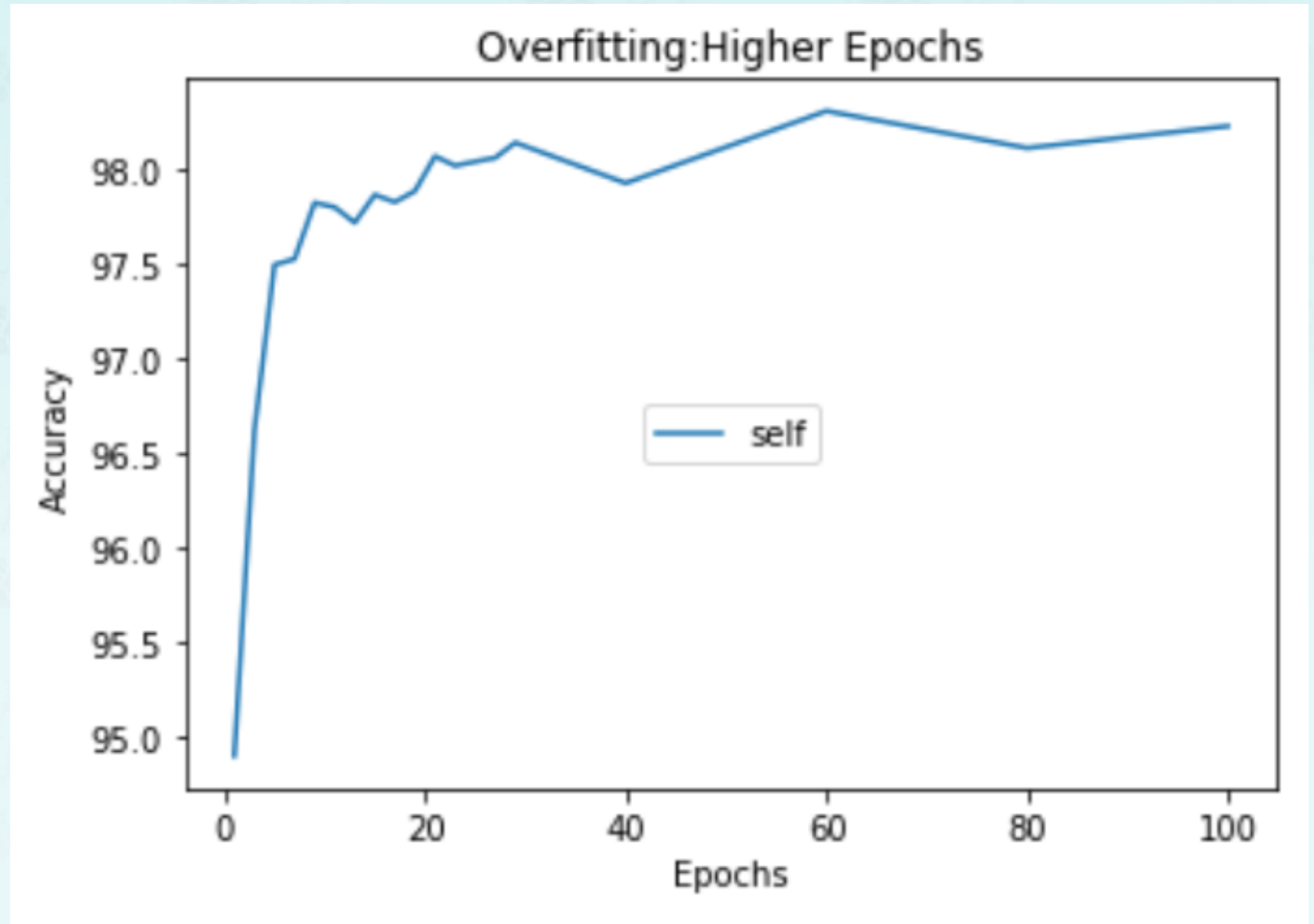


## 4. 과대적합: 시각화 코드

---

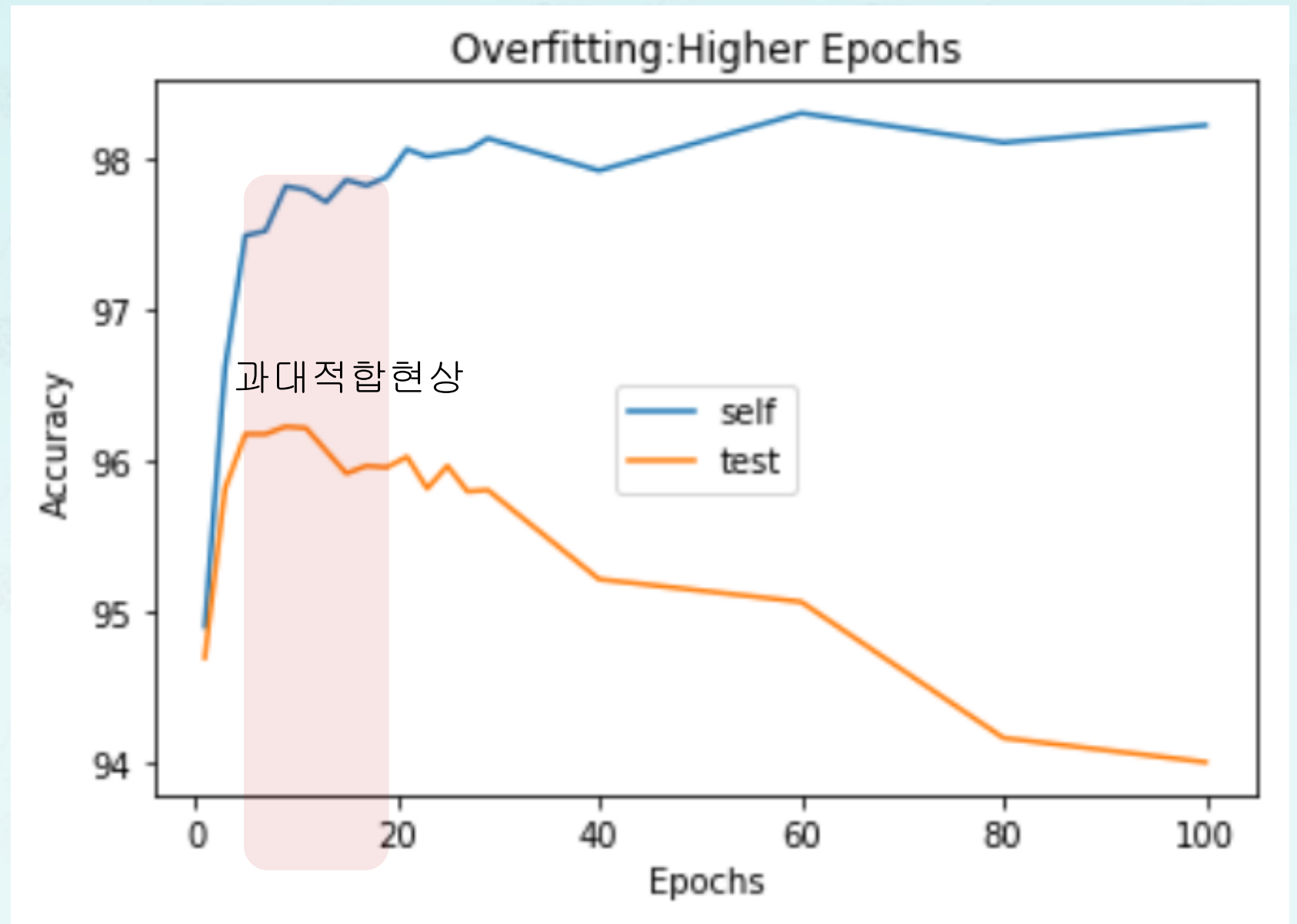
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(epoch_list, self_accuracy, label='self')
plt.plot(epoch_list, test_accuracy, label='test')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Overfitting: Higher Epochs'|
          .format(nn.batch_size))
plt.legend(loc='center')
plt.show()
```

## 4. 과대적합: 정확도 결과 확인 그래프





## 4. 과대적합: 정확도 결과 비교 그래프

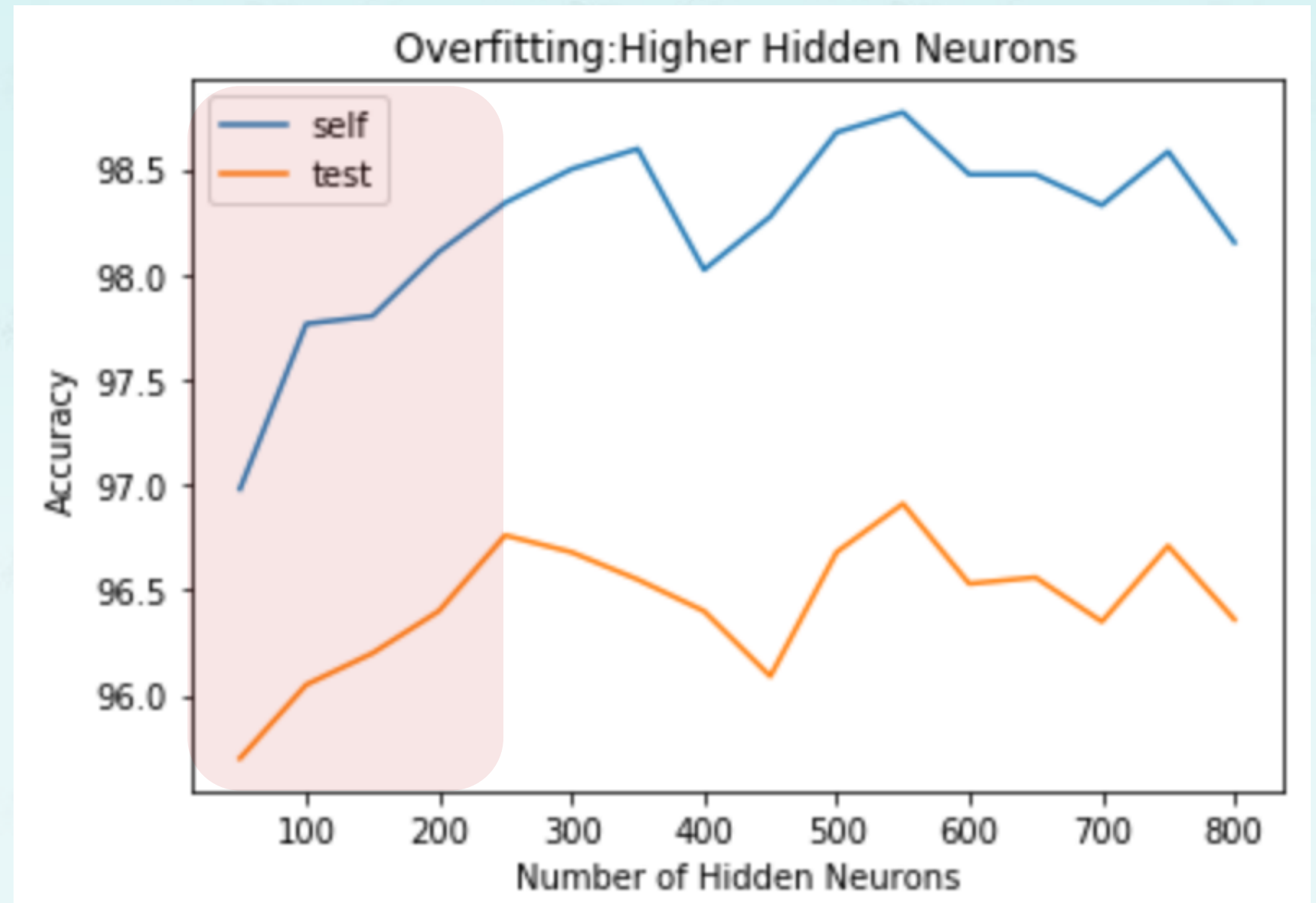


## 4. 과대적합: 원인 - 은닉층 노드의 수 증가

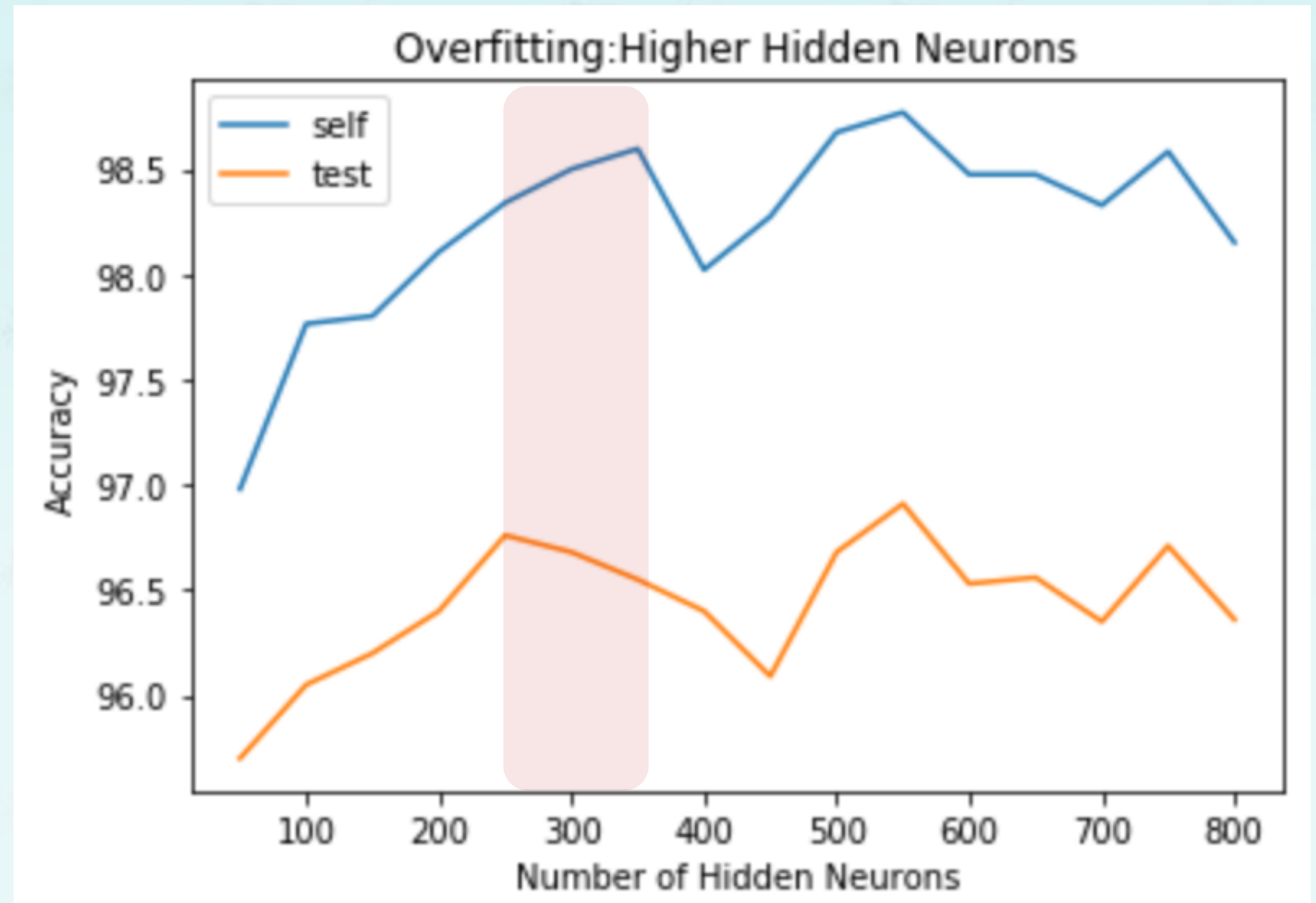
- 실험: 학습 자료 **vs** 테스트 자료
  - 반복 횟수를 **(10)**으로 고정
  - 은닉층 노드의 수에 따른 정확도 비교

```
1 (X, y), (Xtest, ytest) = joy.load_mnist()
2 n_h_list = np.linspace(50, 800, 16, dtype=int)
3 self_accuracy = []
4 test_accuracy = []
5 for n_h in n_h_list:
6     nn = MnistMiniBatch(784, n_h, 10,
7                           epochs = 10, batch_size = 32)
8     nn.fit(X, y)
9     self_accuracy.append(nn.evaluate(X, y))
10    test_accuracy.append(nn.evaluate(Xtest, ytest))
```

## 4. 과대적합:정확도 결과 비교 그래프



## 4. 과대적합: 정확도 결과 비교 그래프



## 4. 과대적합: 해결 방법

---

- 조기 종료
- 데이터 증식
- 드롭아웃

## 5. 과대적합 해결 방법 : 조기 종료

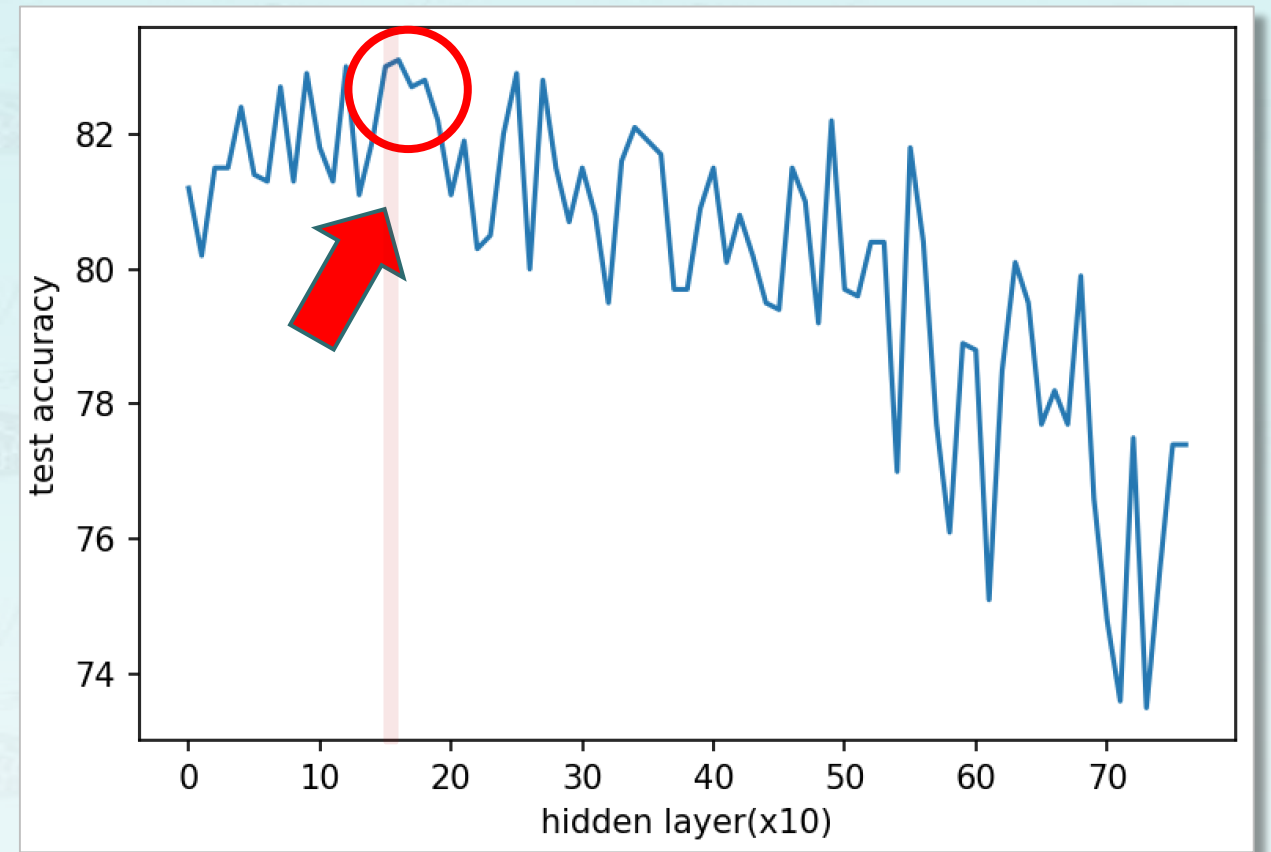
---

- 영어 : **early stopping**
- 배치 경사하강법
  - 조기 종료점 찾기 용이
- 미니 배치, 확률적 경사하강법
  - 조기 종료점 찾기 어려움
- 실험:
  - 은닉층 노드 **40 → 800**

## 5. 과대적합 해결 방법 : 조기 종료 코드

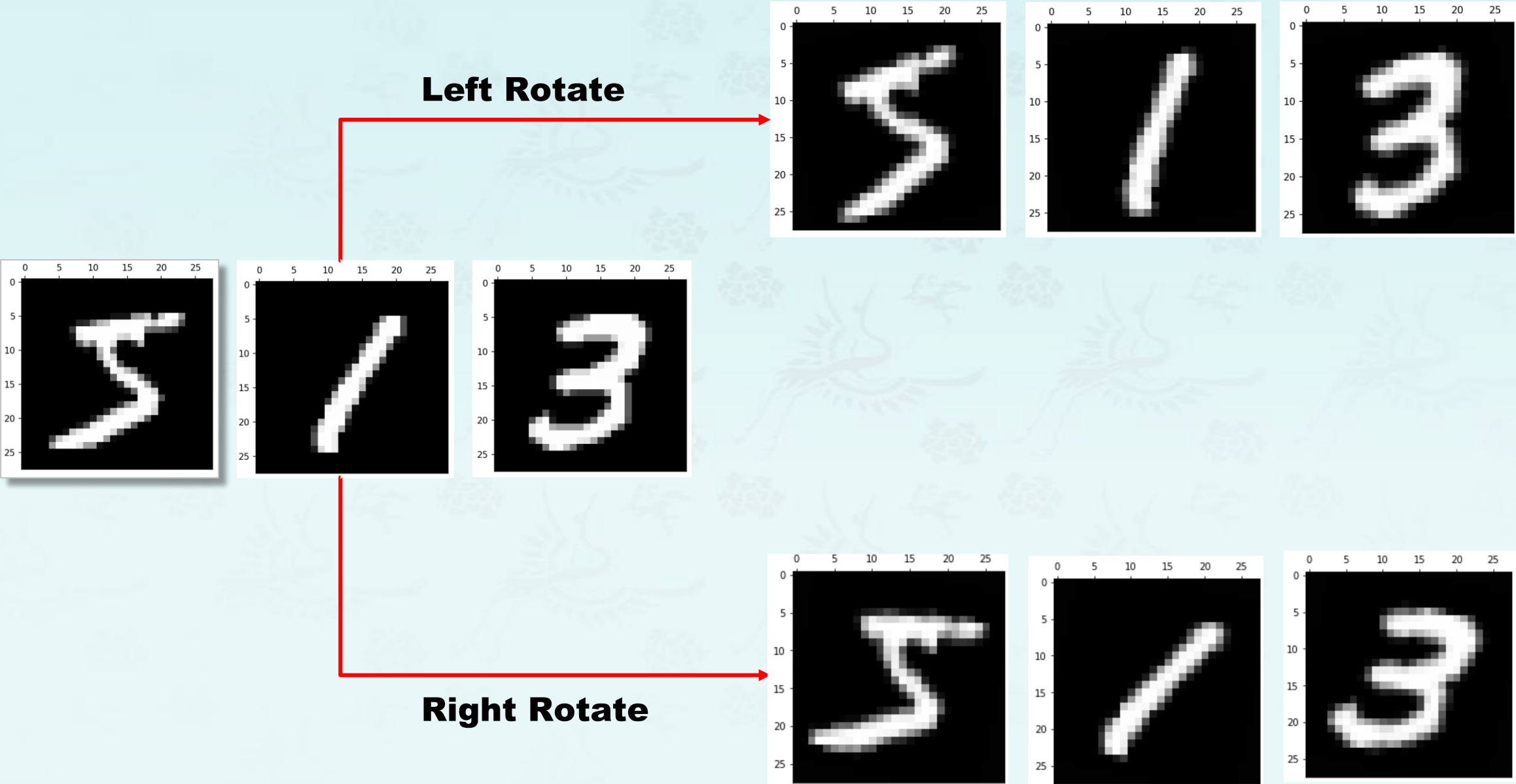
```
1 import joy
2 (X, y), (Xtest, ytest) = joy.load_mnist()
3
4 trainlist = []
5 testlist = []
6 w1 = []
7 w2 = []
8 for h1 in range(40, 810, 10):
9     nn = MnistMiniBatchGD([784, h1, 10], epochs = 20,
10                           batch_size = 32)
11     nn.fit(X[:1000], y[:1000])
12     trainning = nn.evaluate(X[:1000], y[:1000])
13     test = nn.evaluate(Xtest[:1000], ytest[:1000])
14
15     trainlist.append(round(trainning, 2))
16     testlist.append(round(test, 2))
17     w1.append(nn.W1)
18     w2.append(nn.W2)
```

## 5. 과대적합 해결 방법 : 조기 종료 그래프

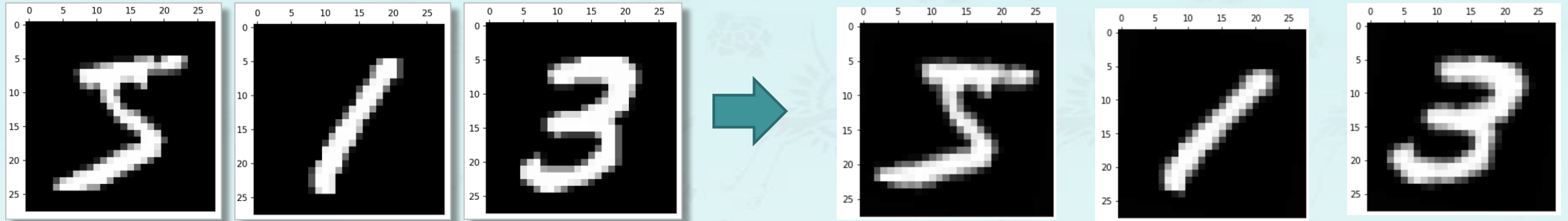




# 5. 과대적합 해결 방법 : 데이터 증식



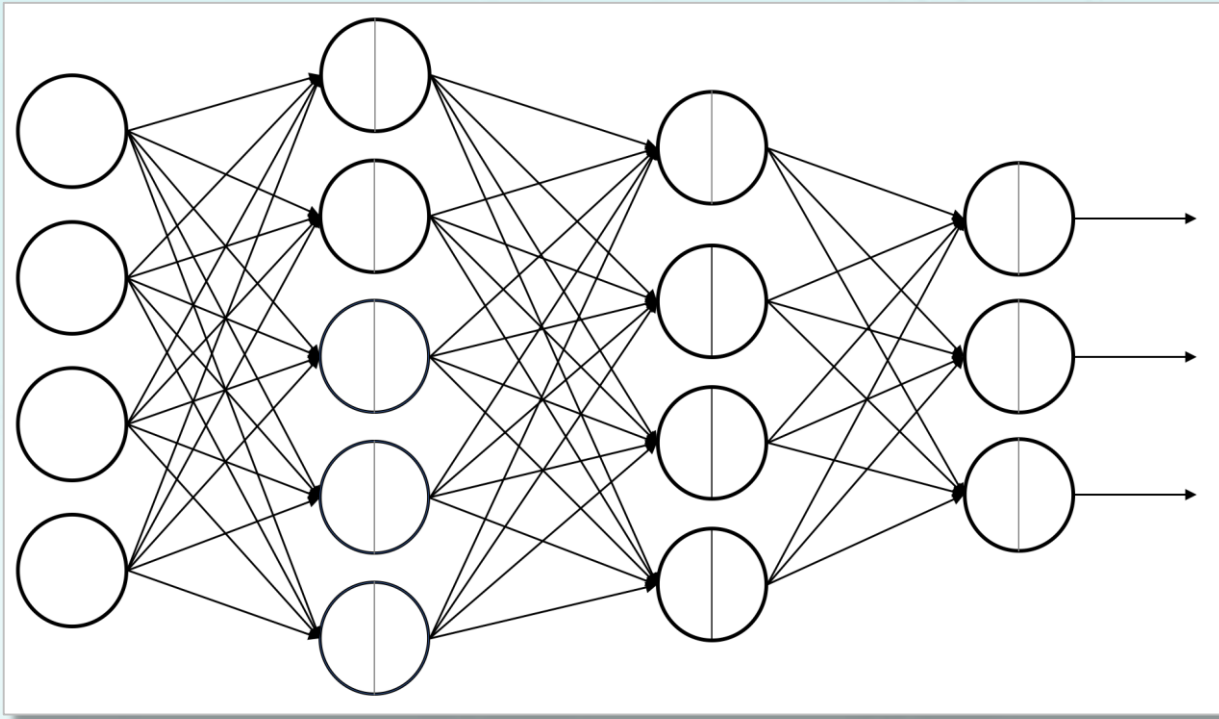
## 5. 과대적합 해결 방법 : 데이터 증식 코드



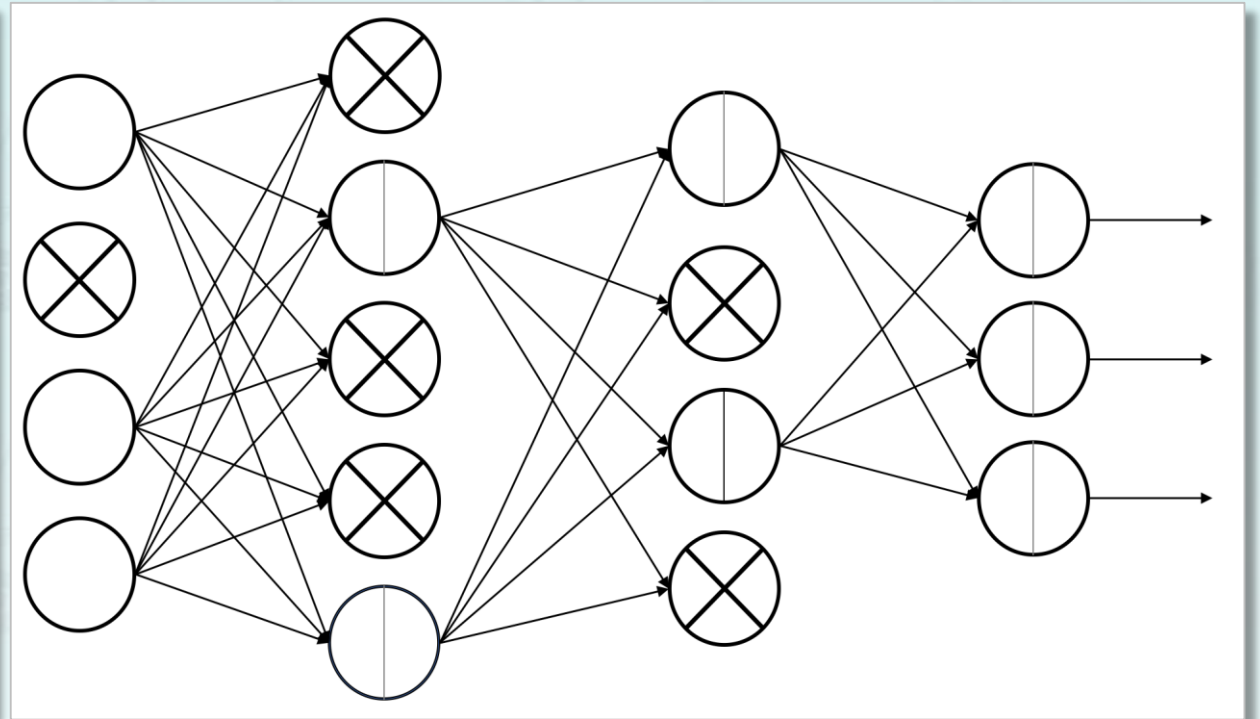
```
1 import joy
2 import scipy
3 (X, y), (Xtest, ytest) = joy.load_mnist()
4 for idx in range(0, 3):
5     Xr = X[idx].reshape(28, 28)
6     joy.show_mnist(Xr, savefig='Xr_rotate')
7     Xr = scipy.ndimage.rotate(Xr, 12.0, cval=0.01,
8                               order=1, reshape=False)
9     joy.show_mnist(Xr, savefig='Xr_rotate_right')
```

## 5. 과대적합 해결 방법 : 드롭아웃

- 기본 다층 인공신경망

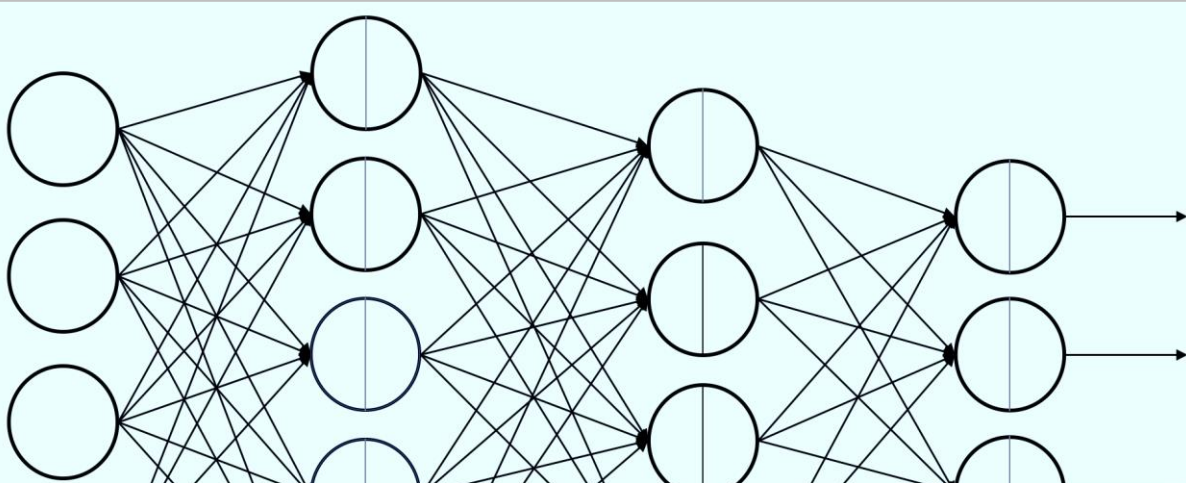


- 드롭아웃 적용한 다층 인공신경망



## 5. 과대적합 해결 방법 : 드롭아웃 결과

### ■ 기본 다층 인공신경망

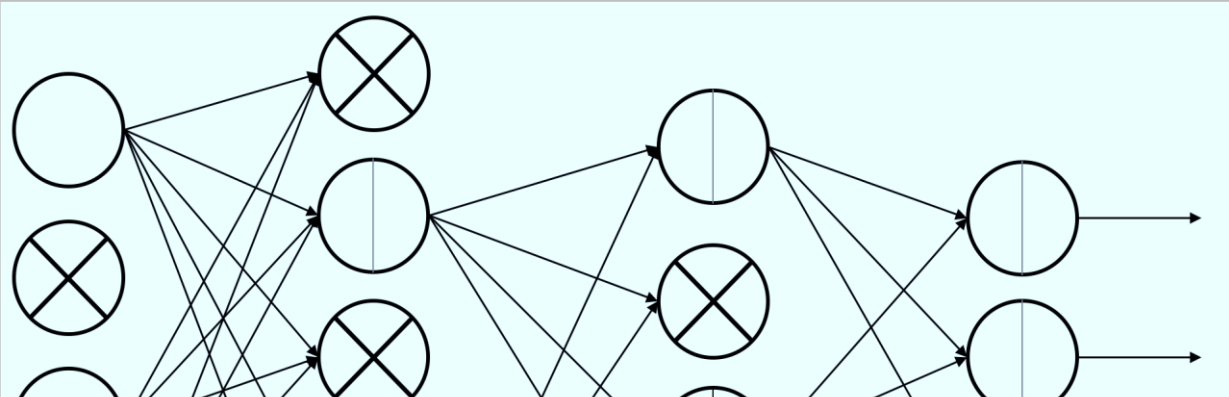


```
1 import joy
2 (X, y), (Xtest, ytest) = joy.load_mnist()
3 nn = MnistMiniBatchGD(784, 600, 10, epochs = 40)
4 nn.fit(X[:3000], y[:3000])
5 self = nn.evaluate(X[:3000], y[:3000])
6 test = nn.evaluate(Xtest[:1000], ytest[:1000])
7 print('MNIST self accuracy {}'.format(self))
8 print('MNIST test accuracy {}'.format(test))
```

MNIST self accuracy 100.0%  
MNIST test accuracy 90.0%

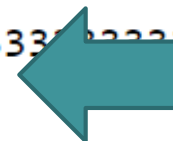


### ■ 드롭아웃 적용한 다층 인공신경망



```
1 import joy
2 (X, y), (Xtest, ytest) = joy.load_mnist()
3 nn = MnistMiniBatchGD_Dropout(784, 600, 10,
4                               epochs = 40, dropout_ratio = 0.5)
5 nn.fit(X[:3000], y[:3000])
6 self = nn.evaluate(X[:3000], y[:3000])
7 test = nn.evaluate(Xtest[:1000], ytest[:1000])
8 print('MNIST self accuracy {}'.format(self))
9 print('MNIST test accuracy {}'.format(test))
```

MNIST self accuracy 99.73333333333333%  
MNIST test accuracy 91.7%



## 5. 과대적합 해결 방법 : 드롭아웃

---

- 2012년 제프리 힌튼 교수 제안
- 간단한 알고리즘
- 학습과정에서 일부 뉴론들 제외
- 성능 향상: **1~2 %**
- 드롭아웃 확률: **50%** 로 시작함


## 5. 과대적합 해결 방법 : 드롭아웃 코드

```
1  def forpass(self, A0, train=True):
2      Z1 = np.dot(self.W1, A0)
3      A1 = self.g(Z1)
4
5      # Dropout
6      if train:
7          self.drop_units = \
8              np.random.rand(*A1.shape) > self.dropout_ratio
9          A1 = A1 * self.drop_units / self.dropout_ratio
10
11     Z2 = np.dot(self.W2, A1)
12     A2 = self.g(Z2)
13     return Z1, A1, Z2, A2
```





## 5. 과대적합 해결 방법 : 드롭아웃 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 i in range(0, m_samples, self.batch_size):
7             A0 = X[i: i + self.batch_size].T
8             Y0 = Y[i: i + self.batch_size].T
9             Z1, A1, Z2, A2 = self.forpass(A0)
10
11             E2 = Y0 - A2
12             E1 = np.dot(self.W2.T, E2)
13             dZ2 = E2 * self.g_prime(Z2)
14             dZ1 = E1 * self.g_prime(Z1)
15
16              dZ1 = dZ1 * self.drop_units
17
18             self.W2 += self.eta * np.dot(dZ2, A1.T)
19             self.W1 += self.eta * np.dot(dZ1, A0.T)
20             self.cost_.append(np.sqrt(np.sum(E2 * E2)
21                                     /self.batch_size))
22     return self
```

## 5. 과대적합 해결 방법 : 드롭아웃 코드

- `predict()`

```
1  def predict(self, X):  
2      A0 = np.array(X, ndmin=2).T  
3      Z1, A1, Z2, A2 = self.forpass(A0, train=False)  
4      return A2
```





# 경사하강법 2

---

- 학습 정리
  - 미니배치(**Mini-Batch**) 경사하강법
  - 학습 스케줄링
  - 과대적합의 원인과 해결 방법
    - 조기 종료
    - 데디터 증식
    - 드롭아웃