13주차(1/2)

심층 신경망 2

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

심층 신경망 2

- 학습 목표
  - **DeepNeuralNet** 클래스로 배치 경사하강법을 구현한다.
  - 다양한 데이터셋을 이용하여 **DeepNeuralNet** 의 성능을 테스트한다.

- 학습 내용
  - **DeepNeuralNet** 클래스에 배치 경사하강법 구현하기
  - **MNIST-Fashion DataSet** 다루기
  - 고양이 인식 문제 도전하기

1. DNN 배치 경사하강법: 학습 fit() 메소드

```
def fit(self, X, y):
    self.cost_ = []
    self.m_samples = len(y)
    Y = joy.one_hot_encoding(y, self.net_arch[-1])

    for epoch in range(self.epochs):
        A0 = np.array(X, ndmin=2).T
        Y0 = np.array(Y, ndmin=2).T

        Z, A = self.forpass(A0)           # for. pass
        cost = self.backprop(Z, A, Y0)    # back prop
        self.cost_.append(np.sqrt(np.sum(cost * cost)))
    return self
```

1. DNN 배치 경사하강법: 학습 fit() 메소드

```
1  def fit(self, X, y):
2      self.cost_ = []
3      self.m_samples = len(y)
4      Y = joy.one_hot_encoding(y, self.net_arch[-1])
5
6      for epoch in range(self.epochs):
7          A0 = np.array(X, ndmin=2).T
8          Y0 = np.array(Y, ndmin=2).T
9
10         Z, A = self.forpass(A0)           # for. pass
11         cost = self.backprop(Z, A, Y0)    # back prop
12         self.cost_.append(np.sqrt(np.sum(cost * cost)))
13     return self
```

1. DNN 배치 경사하강법: 예측 predict_() 메소드

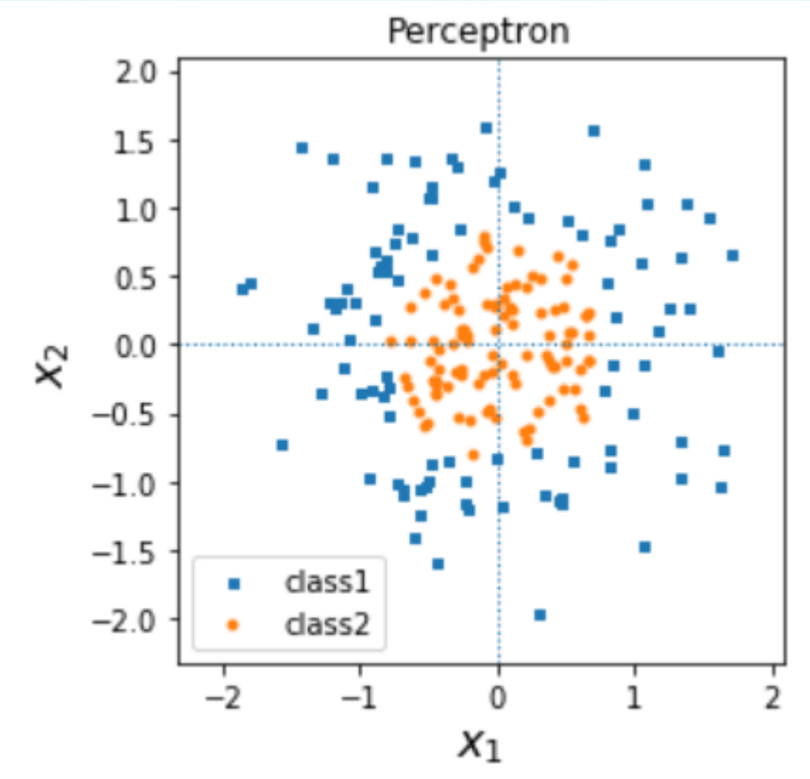
```
1  # invoked by plot_decsion_regions()
2  def predict(self, X):
3      Z, A2 = self.forpass(X)
4      A2 = np.array(A2[len(A2)-1])
5      return A2[-1] > 0.5
6
7  # used by DeepNeuralNet.evaluate()
8  def predict_(self, X):
9      A0 = np.array(X, ndmin=2).T
10     Z, A = self.forpass(A0)
11     return A[-1]
```

1. DNN 배치 경사하강법: 예측 predict_() 메소드

```
1  # invoked by plot_decsion_regions()
2  def predict(self, X):
3      Z, A2 = self.forpass(X)
4      A2 = np.array(A2[len(A2)-1])
5      return A2[-1] > 0.5
6
7  # used by DeepNeuralNet.evaluate()
8  def predict_(self, X):
9      A0 = np.array(X, ndmin=2).T
10     Z, A = self.forpass(A0)
11     return A[-1]
```

2. DNN의 성능: 가우시안 퀀타일즈

```
1  import joy
2  x, y = joy.gaussian_quantiles(random_seed = 0)
3  joy.plot_xyw(x.T, y.squeeze())
```



2. DNN의 성능: 가우시안 퀀타일즈

```
import joy
X, Y = joy.gaussian_quantiles(random_seed=1)
nn1 = DeepNeuralNet_BGD([2, 100, 2],
                        eta=0.2, epochs=2000)
nn1.fit(X.T, Y.flatten())
accuracy = nn1.evaluate(X.T, Y.flatten())
print('Accuracy {}%'.format(accuracy))
```

은닉층 1개

```
import joy
X, Y = joy.gaussian_quantiles(random_seed = 1)
nn2 = DeepNeuralNet_BGD([2, 60, 30, 10, 2],
                                eta=0.2, epochs=2000)
nn2.fit(X.T, Y.flatten())
accuracy = nn2.evaluate(X.T, Y.flatten())
print("Accuracy {}%".format(accuracy))
```

은닉층 3개

2. DNN의 성능: 가우시안 퀀타일즈

```
import joy
X, Y = joy.gaussian_quantiles(random_seed=1)
nn1 = DeepNeuralNet_BGD([2, 100, 2],
                        eta=0.2, epochs=2000)
nn1.fit(X.T, Y.flatten())
accuracy = nn1.evaluate(X.T, Y.flatten())
print('Accuracy {}%'.format(accuracy))
```

```
import joy
X, Y = joy.gaussian_quantiles(random_seed = 1)
nn2 = DeepNeuralNet_BGD([2, 60, 30, 10, 2],
                         eta=0.2, epochs=2000)
nn2.fit(X.T, Y.flatten())
accuracy = nn2.evaluate(X.T, Y.flatten())
print("Accuracy {}%".format(accuracy))
```

2. DNN의 성능: 가우시안 퀀타일즈

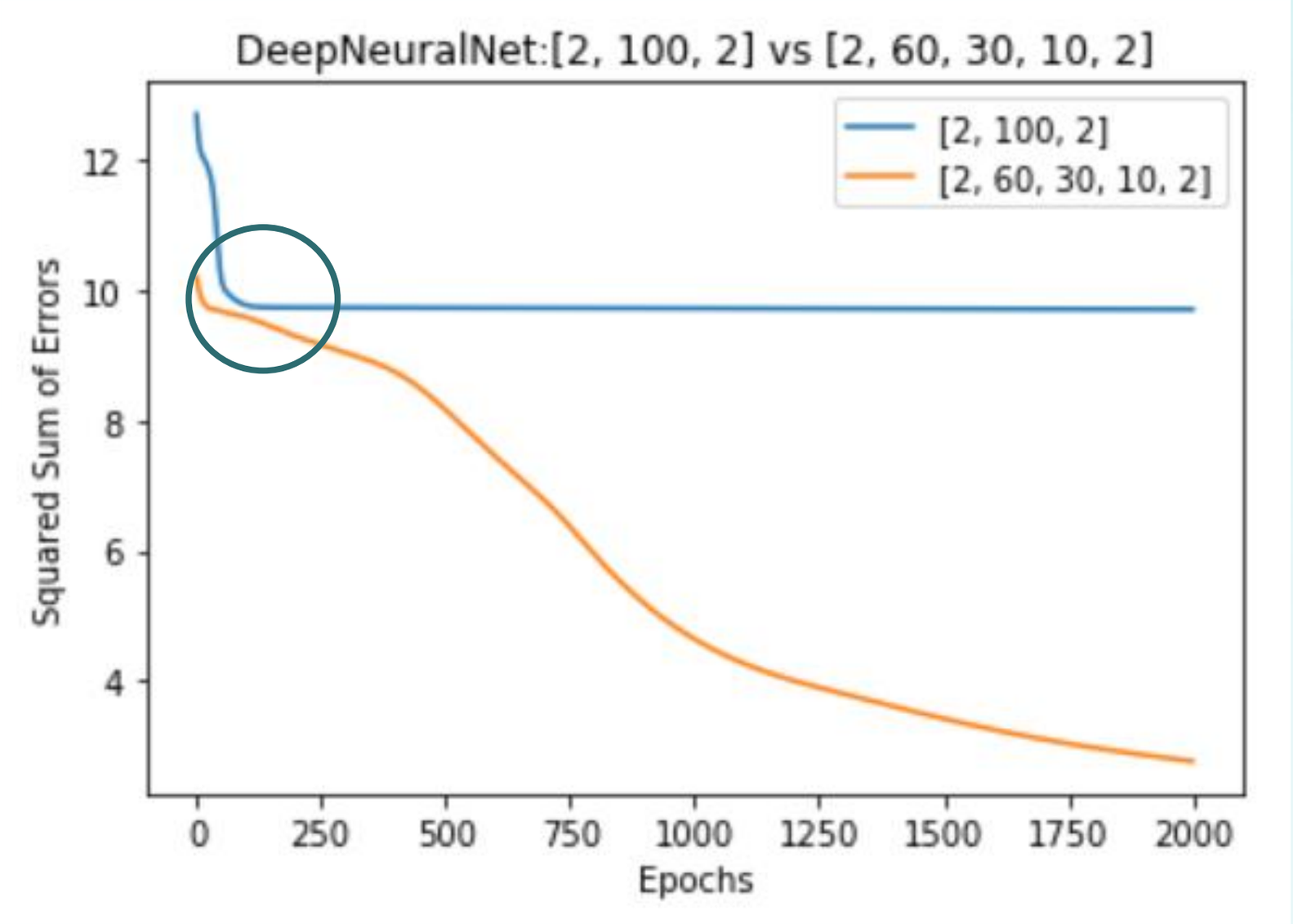
```
import joy
X, Y = joy.gaussian_quantiles(random_seed=1)
nn1 = DeepNeuralNet_BGD([2, 100, 2],
                        eta=0.2, epochs=2000)
nn1.fit(X.T, Y.flatten())
accuracy = nn1.evaluate(X.T, Y.flatten())
print('Accuracy {}%'.format(accuracy))
```

Accuracy 55.50000000000001%

```
import joy
X, Y = joy.gaussian_quantiles(random_seed = 1)
nn2 = DeepNeuralNet_BGD([2, 60, 30, 10, 2],
                              eta=0.2, epochs=2000)
nn2.fit(X.T, Y.flatten())
accuracy = nn2.evaluate(X.T, Y.flatten())
print("Accuracy {}%".format(accuracy))
```

Accuracy 99.5%

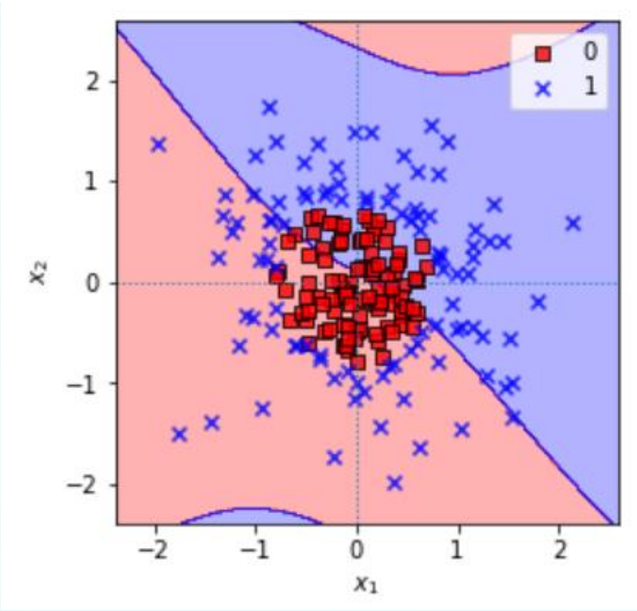
2. DNN의 성능: 가우시안 퀀타일즈



2. DNN의 성능: 가우시안 퀀타일즈

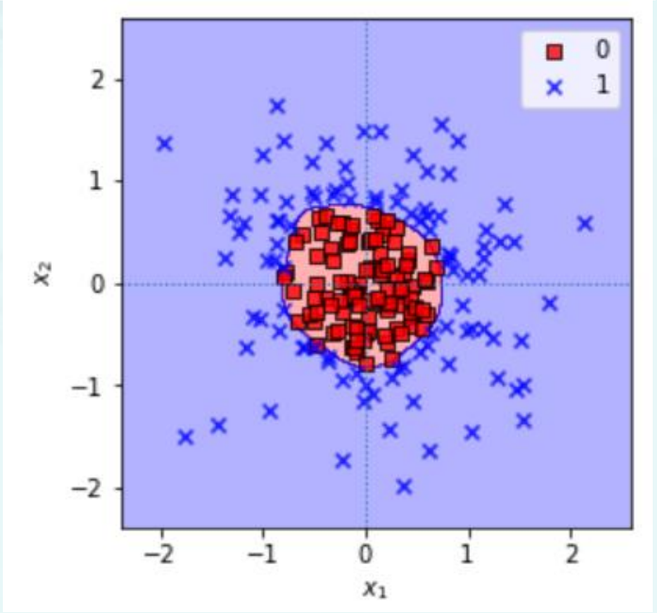
```
import joy
X, Y = joy.gaussian_quantiles(random_seed=1)
nn1 = DeepNeuralNet_BGD([2, 100, 2],
                        eta=0.2, epochs=2000)
nn1.fit(X.T, Y.flatten())
accuracy = nn1.evaluate(X.T, Y.flatten())
print('Accuracy {}%'.format(accuracy))
```

Accuracy 55.50000000000001%



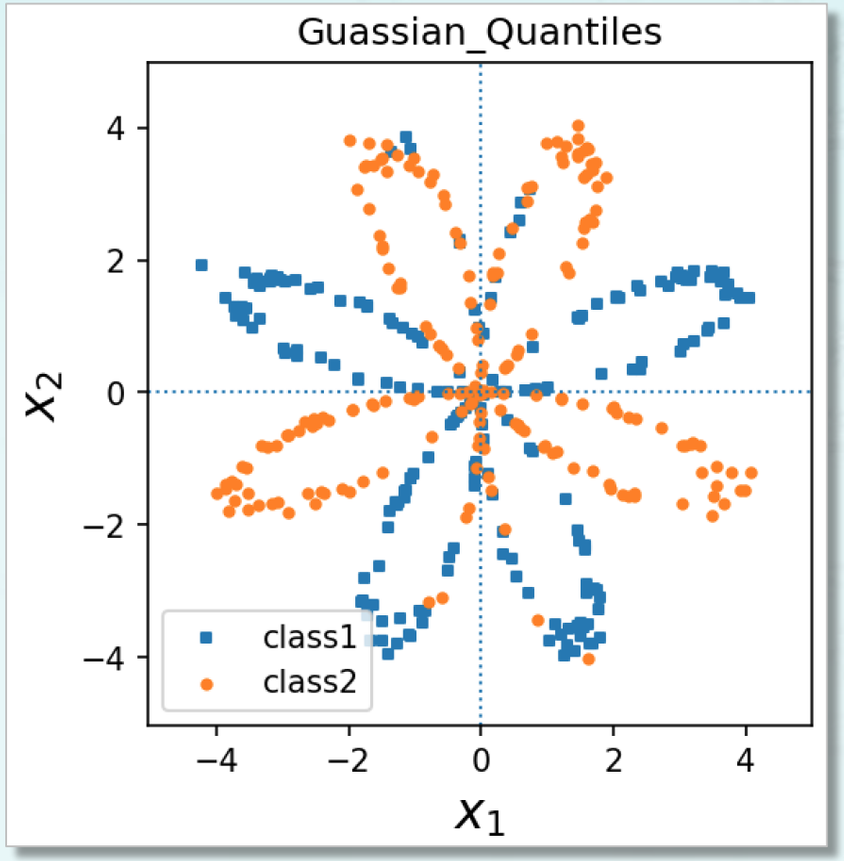

```
import joy
X, Y = joy.gaussian_quantiles(random_seed = 1)
nn2 = DeepNeuralNet_BGD([2, 60, 30, 10, 2],
                        eta=0.2, epochs=2000)
nn2.fit(X.T, Y.flatten())
accuracy = nn2.evaluate(X.T, Y.flatten())
print("Accuracy {}%".format(accuracy))
```

Accuracy 99.5%



2. DNN의 성능: 플라나 데이터

```
1  import joy
2  X, y = joy.planar_data()
3  joy.plot_xyw(X.T, y.squeeze())
```



2. DNN의 성능: 플라나 데이터

```
1  import joy
2  X, y = joy.planar_data()
3  nn1 = DeepNeuralNet_BGD([2, 150, 2],
4                          eta=0.3, epochs=500)
5  nn1.fit(X.T, y.flatten())
6  accuracy = nn1.evaluate(X.T, y.flatten())
7  print("Accuracy {}%".format(accuracy))
```

은닉층 1개

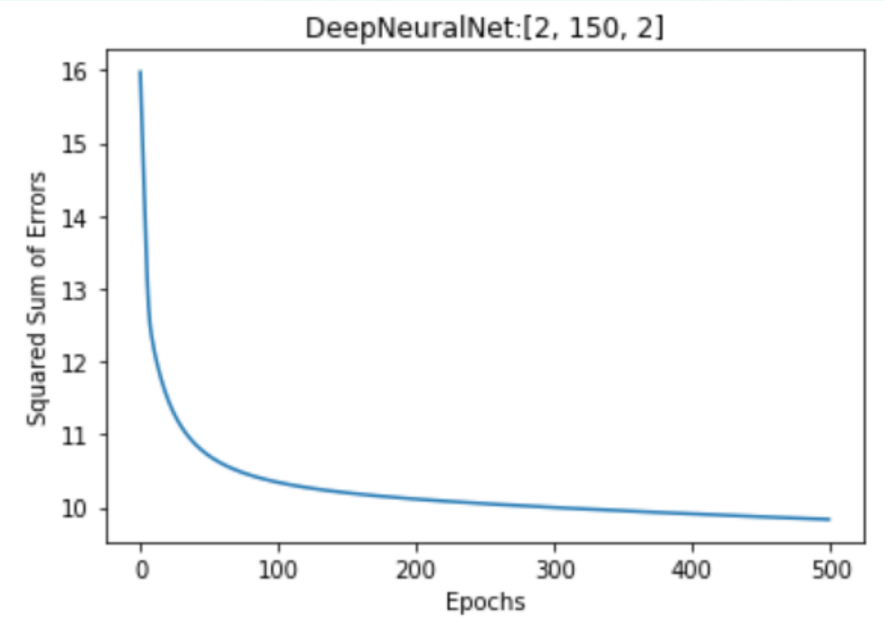
```
1  import joy
2  X, y = joy.planar_data()
3  nn2 = DeepNeuralNet_BGD([2, 100, 30, 20, 2],
4                          eta=0.3, epochs=500)
5  nn2.fit(X.T, y.flatten())
6  accuracy = nn2.evaluate(X.T, y.flatten())
7  print("Accuracy {}%".format(accuracy))
```

은닉층 3개

2. DNN의 성능: 플라나 데이터

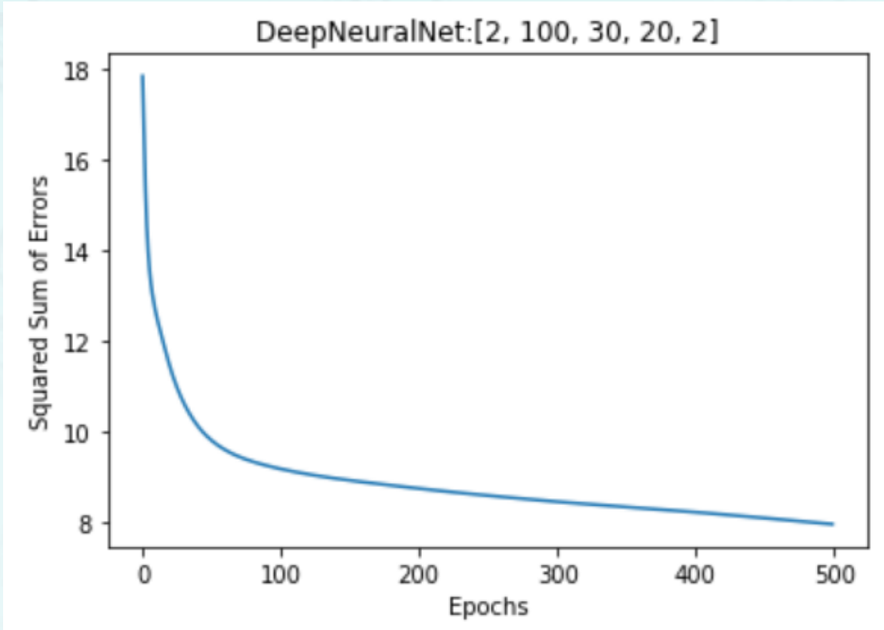
```
1  import joy
2  X, y = joy.planar_data()
3  nn1 = DeepNeuralNet_BGD([2, 150, 2],
4                          eta=0.3, epochs=500)
5  nn1.fit(X.T, y.flatten())
6  accuracy = nn1.evaluate(X.T, y.flatten())
7  print("Accuracy {}%".format(accuracy))
```

Accuracy 85.5%

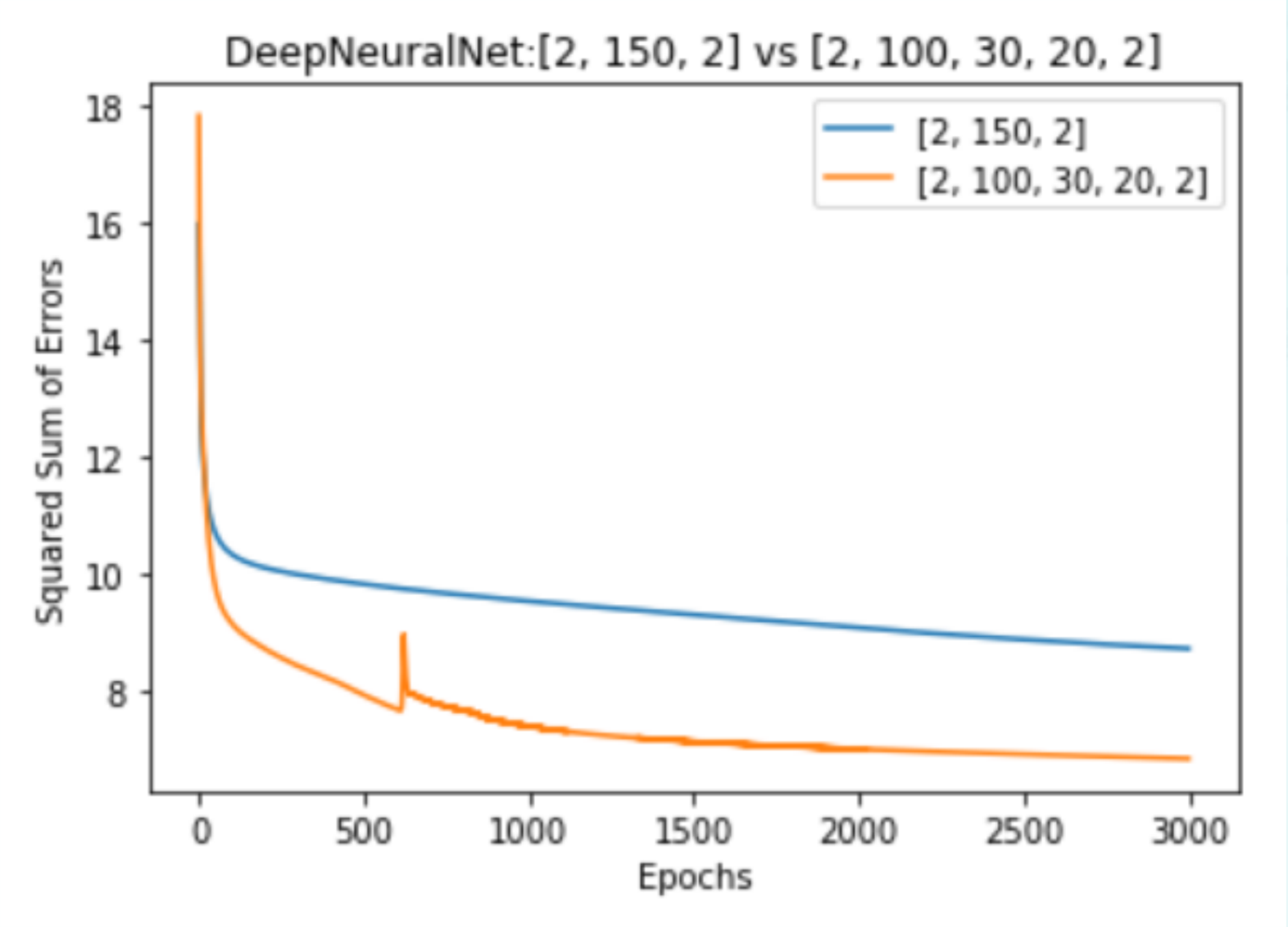



```
1  import joy
2  X, y = joy.planar_data()
3  nn2 = DeepNeuralNet_BGD([2, 100, 30, 20, 2],
4                          eta=0.3, epochs=500)
5  nn2.fit(X.T, y.flatten())
6  accuracy = nn2.evaluate(X.T, y.flatten())
7  print("Accuracy {}%".format(accuracy))
```

Accuracy 87.75%



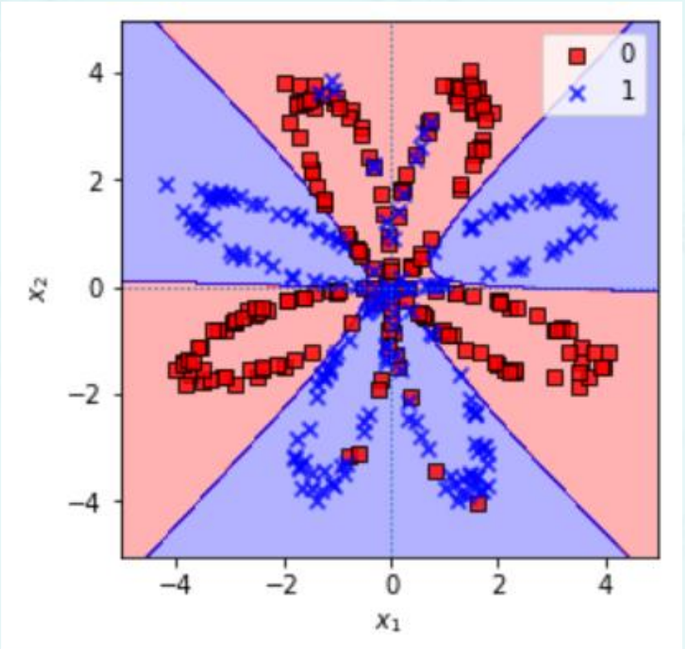
2. DNN의 성능: 플라나 데이터



2. DNN의 성능: 플라나 데이터

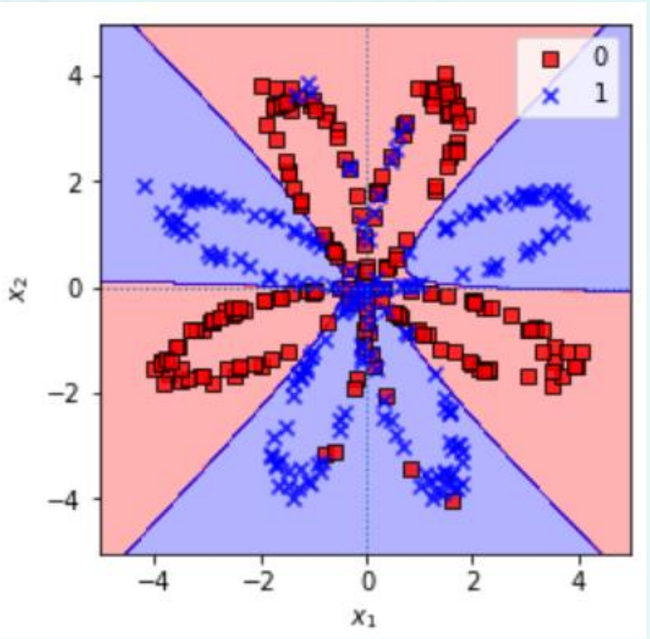
```
1  import joy
2  X, y = joy.planar_data()
3  nn1 = DeepNeuralNet_BGD([2, 150, 2],
4                          eta=0.3, epochs=3000)
5  nn1.fit(X.T, y.flatten())
6  accuracy = nn1.evaluate(X.T, y.flatten())
7  print("Accuracy {}%".format(accuracy))
```

Accuracy 85.75%



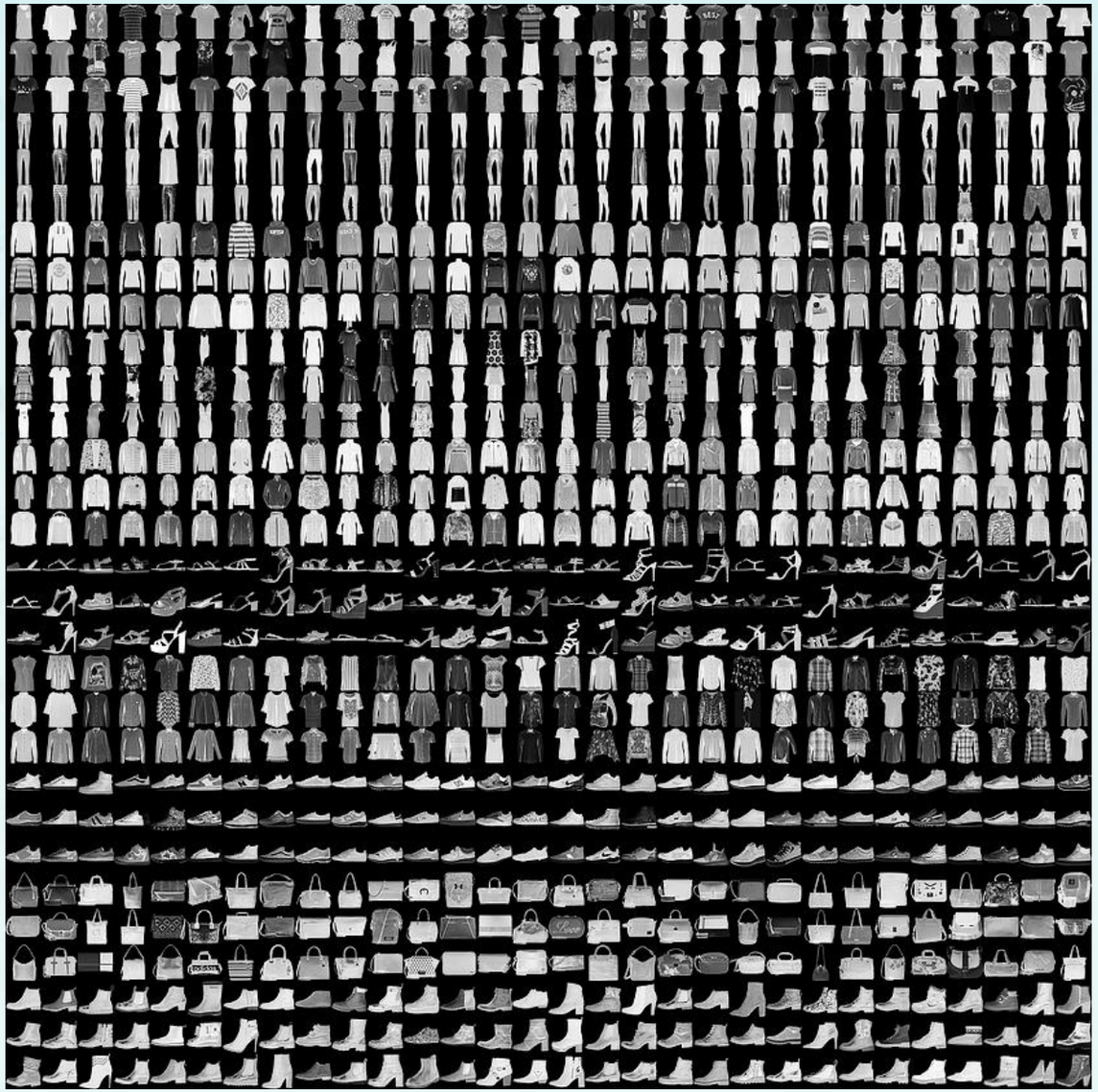

```
1  import joy
2  X, y = joy.planar_data()
3  nn2 = DeepNeuralNet_BGD([2, 100, 30, 20, 2],
4                          eta=0.3, epochs=3000)
5  nn2.fit(X.T, y.flatten())
6  accuracy = nn2.evaluate(X.T, y.flatten())
7  print("Accuracy {}%".format(accuracy))
```

Accuracy 91.0%



3. MNIST-Fashion DataSet: 개요

- 잘란도(**Zalando**)의 패션 작품의 데이터 셋
  - **28x28** 크기의 **grayscale image**
  - **60,000**개의 학습 자료
  - **10,000**개의 테스트 데이터



3. MNIST-Fashion DataSet: 기존 MNIST의 문제점

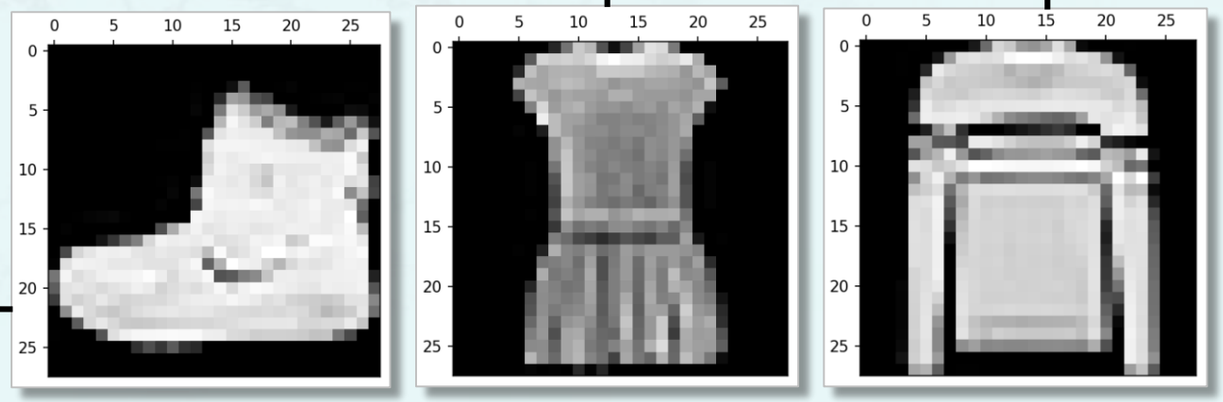
1. 학습하기 매우 쉬움
2. 너무 많이 사용되고 인용됨
3. 최근 영상처리 관련 자료를
   대표 할 수 없음

3. MNIST-Fashion DataSet: 데이터

```
import joy
import matplotlib.pyplot as plt
(X, y), (Xt, yt) = joy.load_fashion_mnist(
                normalize=False, flatten=False)
joy.show_mnist(X[0], savefig='Fashion-MNIST1')
joy.show_mnist(X[3], savefig='Fashion-MNIST2')
joy.show_mnist(X[5], savefig='Fashion-MNIST3')
```

3. MNIST-Fashion DataSet: 클래스 레이블

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



3. MNIST-Fashion DataSet: 학습 정확성 평가

```
(X, y), (Xtest, ytest) = joy.load_fashion_mnist()
nn = DeepNeuralNet_BGD([784, 100, 10],
                       eta = 0.1, epochs = 1000)
nn.fit(X[:6000], y[:6000])
accuracy = nn.evaluate(Xtest[:1000], ytest[:1000])
print('Accuracy {}%'.format(accuracy))
```

은닉층 1개

```
(X, y), (Xtest, ytest) = joy.load_fashion_mnist()
nn2 = DeepNeuralNet_BGD([784, 200, 100, 50, 10],
                        eta = 0.1, epochs = 1000)
nn2.fit(X[:6000], y[:6000])
accuracy = nn2.evaluate(Xtest[:1000], ytest[:1000])
print('Accuracy {}%'.format(accuracy))
```

은닉층 3개

3. MNIST-Fashion DataSet: 학습 정확성 평가

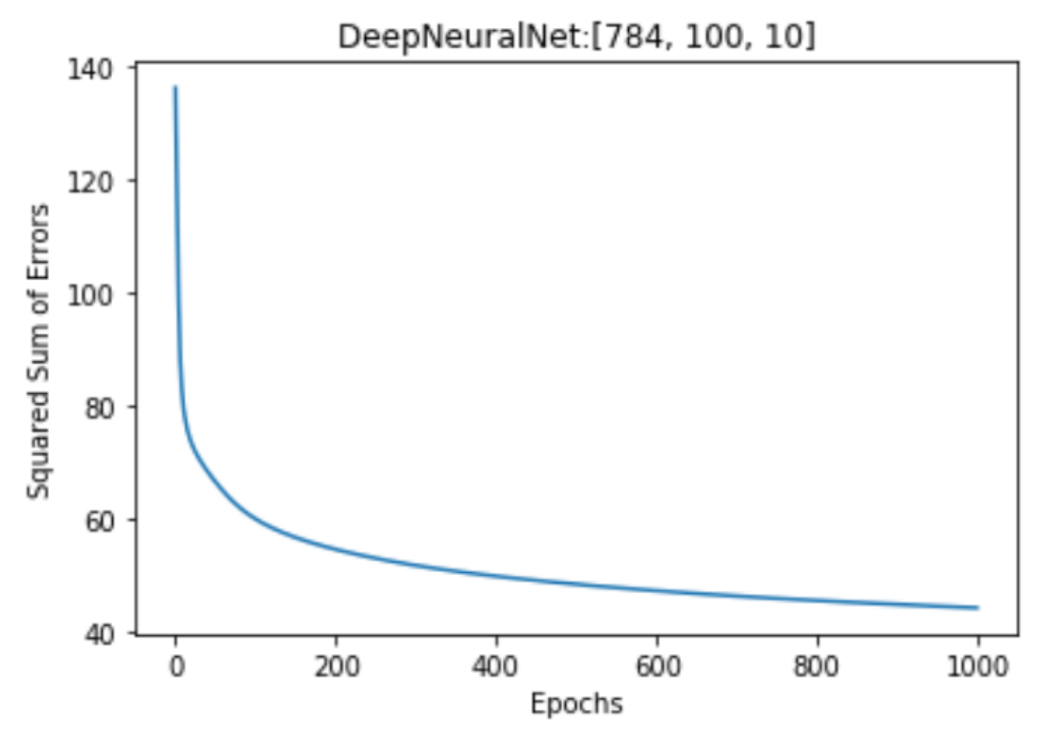
```
(X, y), (Xtest, ytest) = joy.load_fashion_mnist()
nn = DeepNeuralNet_BGD([784, 100, 10],
                       eta = 0.1, epochs = 1000)
nn.fit(X[:6000], y[:6000])
accuracy = nn.evaluate(Xtest[:1000], ytest[:1000])
print('Accuracy {}%'.format(accuracy))
```

```
(X, y), (Xtest, ytest) = joy.load_fashion_mnist()
nn2 = DeepNeuralNet_BGD([784, 200, 100, 50, 10],
                        eta = 0.1, epochs = 1000)
nn2.fit(X[:6000], y[:6000])
accuracy = nn2.evaluate(Xtest[:1000], ytest[:1000])
print('Accuracy {}%'.format(accuracy))
```

3. MNIST-Fashion DataSet: 학습 정확성 평가

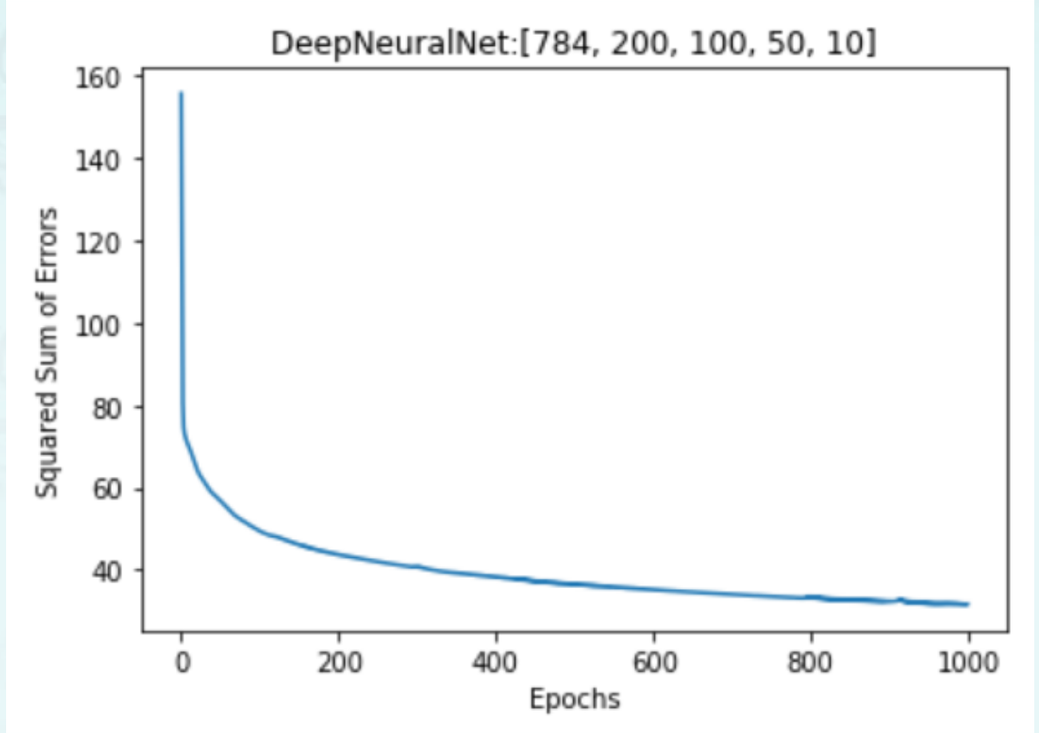
```
(X, y), (Xtest, ytest) = joy.load_fashion_mnist()
nn = DeepNeuralNet_BGD([784, 100, 10],
                       eta = 0.1, epochs = 1000)
nn.fit(X[:6000], y[:6000])
accuracy = nn.evaluate(Xtest[:1000], ytest[:1000])
print('Accuracy {}%'.format(accuracy))
```

```
Accuracy 75.6%
```

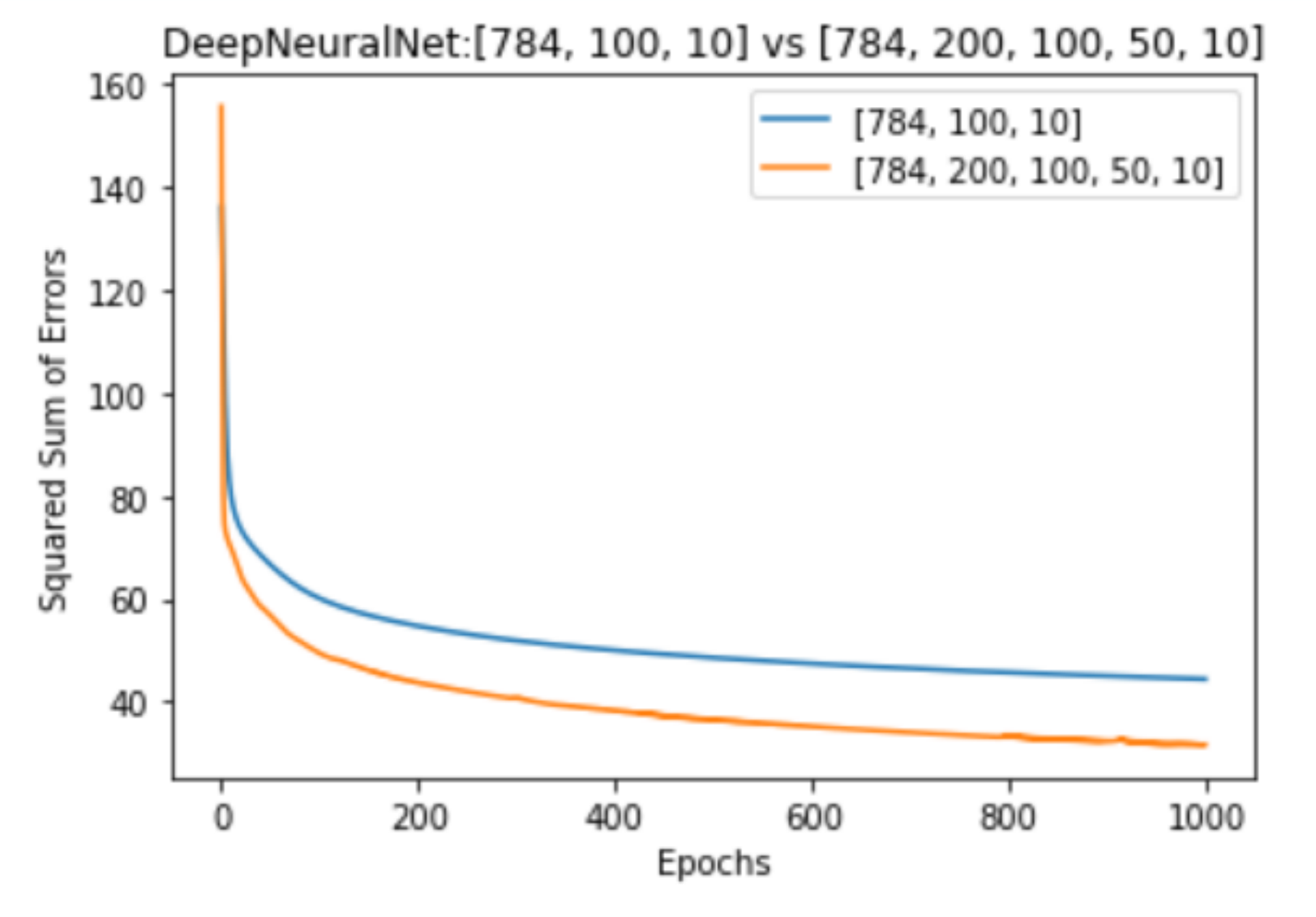



```
(X, y), (Xtest, ytest) = joy.load_fashion_mnist()
nn2 = DeepNeuralNet_BGD([784, 200, 100, 50, 10],
                        eta = 0.1, epochs = 1000)
nn2.fit(X[:6000], y[:6000])
accuracy = nn2.evaluate(Xtest[:1000], ytest[:1000])
print('Accuracy {}%'.format(accuracy))
```

```
Accuracy 83.2%
```

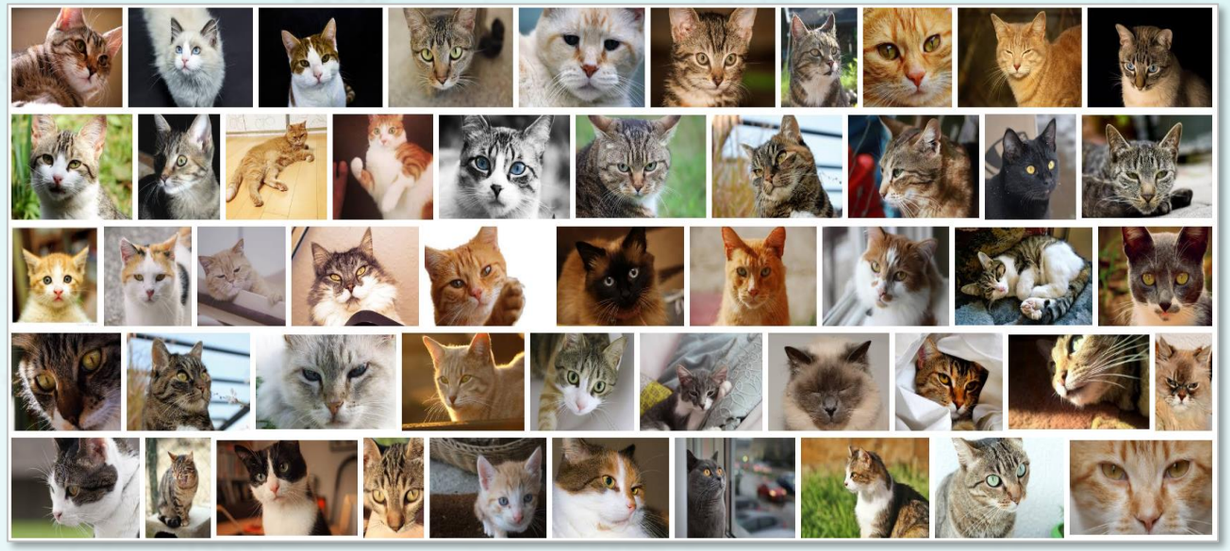


3. MNIST-Fashion DataSet: 학습 정확성 평가



4. 고양이 DataSet: 컴퓨터 vs 사람

3.14159 26535
x 2.71828 18284
x 1.41421 35623
x 1.61803 39887



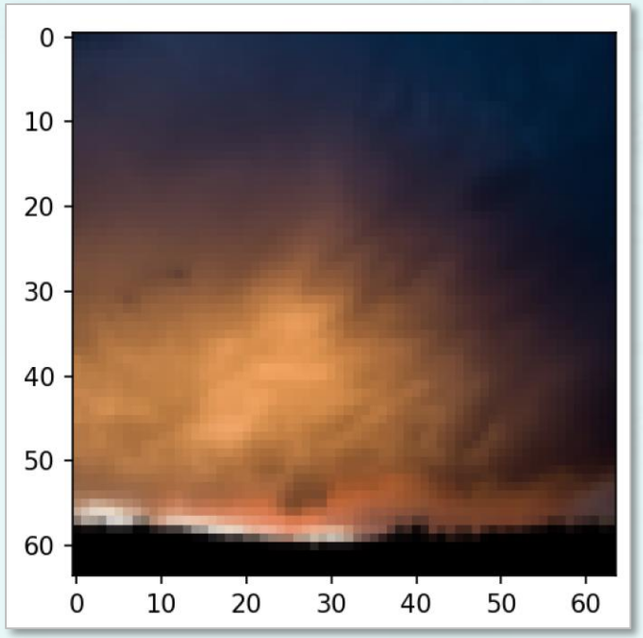
4. 고양이 DataSet: 코드 설명

```
1  import joy
2  X, y, Xtest, ytest, classes = joy.load_cat_data()
3  plt.imshow(X[0])
```

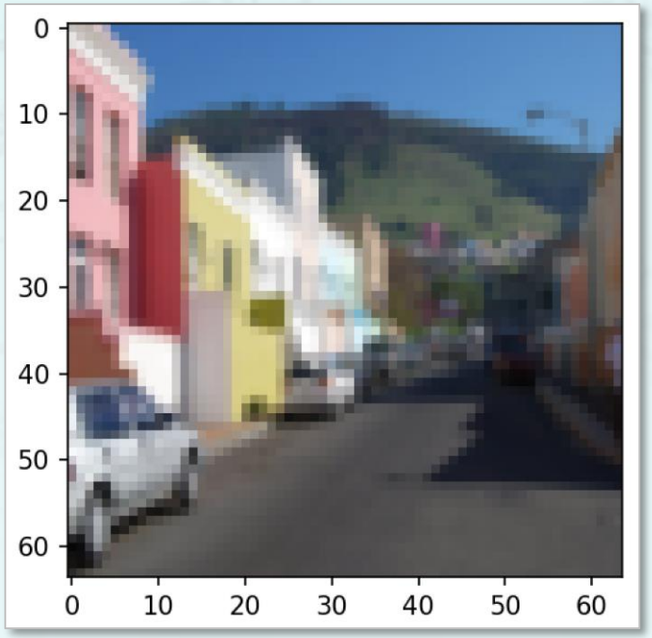
4. 고양이 DataSet: 코드 설명

```
1  import joy
2  X, y, Xtest, ytest, classes = joy.load_cat_data()
3  plt.imshow(X[0])
```

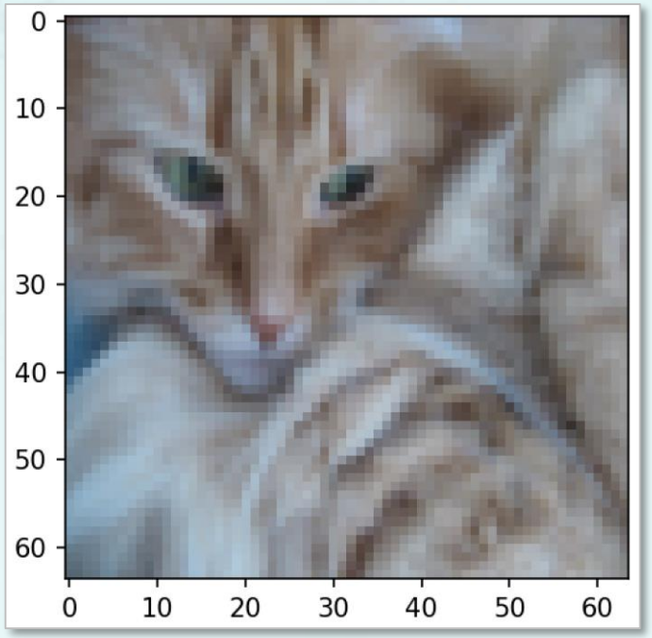
X[0]



X[1]



X[2]



4. 고양이 DataSet: 코드 설명

```
1  print("Number of train examples:", X.shape[0])
2  print("Number of test  examples:", Xtest.shape[0])
3  print("One image size:({}, {}, {})".
4        format(X.shape[1], X.shape[1], 3))
5  print("train X shape:", X.shape)
6  print("train y shape:", y.shape)
7  print("test  X shape:", Xtest.shape)
8  print("test  y shape:", ytest.shape)
```

```
Number of train examples: 209
Number of test  examples: 50
One image size:(64, 64, 3)
train X shape: (209, 64, 64, 3)
train y shape: (1, 209)
test  X shape: (50, 64, 64, 3)
test  y shape: (1, 50)
```

4. 고양이 DataSet: 코드 설명

```
X_flatten = X.reshape(X.shape[0], -1)
Xtest_flatten = Xtest.reshape(Xtest.shape[0], -1)
X = X_flatten/255.
Xtest = Xtest_flatten/255.
print ("train X shape:", X.shape)
print ("test  X shape:", Xtest.shape)
```

```
train X shape: (209, 12288)
test  X shape: (50, 12288)
```

4. 고양이 DataSet: 모델 학습

```
import joy
dnn = DeepNeuralNet_BGD([12288, 100, 50, 2],
                        eta = 0.3, epochs = 164)
dnn.fit(X, y.flatten())
self = dnn.evaluate(X, y.flatten())
print('Accuracy self: {}%'.
      format(np.round(self, 2)))
```

4. 고양이 DataSet: 학습 정확도 결과

```
import joy
dnn = DeepNeuralNet_BGD([12288, 100, 50, 2],
                        eta = 0.3, epochs = 164)
dnn.fit(X, y.flatten())
self = dnn.evaluate(X, y.flatten())
print('Accuracy self: {}%'.
      format(np.round(self, 2)))
```

```
Accuracy self: 97.61%
```

4. 고양이 DataSet: DNN 성능 검증

```
import joy
dnn = DeepNeuralNet_BGD([12288, 100, 50, 2],
                        eta = 0.3, epochs = 178)
dnn.fit(X, y.flatten())
test = dnn.evaluate(Xtest, ytest.flatten())
print('Accuracy self: {}%'.
      format(np.round(test, 2)))
```

4. 고양이 DataSet: 테스트 자료 정확도 결과

```
import joy
dnn = DeepNeuralNet_BGD([12288, 100, 50, 2],
                        eta = 0.3, epochs = 178)
dnn.fit(X, y.flatten())
test = dnn.evaluate(Xtest, ytest.flatten())
print('Accuracy self: {}%'.
      format(np.round(test, 2)))
```

```
Accuracy self: 80.0%
```

심층 신경망 2

- 학습 정리
  - 다층 신경망 **DeepNeuralNet** 성능 테스트
  - **MNIST-Fashion** 데이터 셋
  - 고양이와 고양이가 아닌 데이터 분류