

## ▼ p91 - 4) 단일 계층 신경망

```

1 import pandas as pd
2 import tensorflow as tf
3 import seaborn as sns
4 import numpy as np
5
6 #

```

```

1 (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
2 X_train = X_train.astype(np.float32).reshape(-1, 28*28) / 255.0
3 X_test = X_test.astype(np.float32).reshape(-1, 28*28) / 255.0
4 y_train = tf.keras.utils.to_categorical(y_train)
5 y_test = tf.keras.utils.to_categorical(y_test)
6 y_train = y_train.astype(np.int32)
7 y_test = y_test.astype(np.int32)
8

```

배치 데이터를 만들기 위해 파이썬 제너레이터 함수를 정의합니다.

- 기존 책에 없던 부분

```

1 def shuffle_batch(X, y, batch_size):
2     rnd_idx = np.random.permutation(len(X))
3     n_batches = len(X) // batch_size
4     for batch_idx in np.array_split(rnd_idx, n_batches):
5         X_batch, y_batch = X[batch_idx], y[batch_idx]
6         yield X_batch, y_batch

```

## ▼ 94p

- get\_shape 함수로 구조를 확인해 보겠습니다
- convert\_to\_tensor 함수를 이용해 텐서로 변환후, get\_shape 함수를 사용함

```

1 tf.convert_to_tensor(X_train).get_shape()
2

```

```
TensorShape([Dimension(60000), Dimension(784)])
```

- [MNIST 바로가기](#)
- [이책의 링크 정리된 네이버블로그](#)
- [박해성님의 블로그](#)

가중치 텐서와 바이어스 텐서를 만듭니다.

```

1
2 W = tf.Variable(tf.zeros([784, 10]))
3 b = tf.Variable(tf.zeros([10]))

```

훈련 이미지 데이터를 넣을 플레이스홀더와 소프트맥스 텐서를 만듭니다.

```

1 x = tf.placeholder("float", [None, 784])
2 y = tf.nn.softmax(tf.matmul(x, W) + b)

```

실제 레이블을 담기위한 텐서와 교차 엔트로피 방식을 이용하는 그래디언트 디센트 방식을 선택합니다.

```

1 y_ = tf.placeholder("float", [None, 10])
2 cross_entropy = -tf.reduce_sum(y_*tf.log(y))
3 train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

```

변수를 초기화하고 세션을 시작합니다.

```
1
2 sess = tf.Session()
3 sess.run(tf.global_variables_initializer())
```

1000의 반복을 수행하고 결과를 출력합니다. 최종 정확도는 91% 정도 입니다.

```
1 for i in range(1000):
2     batch_xs, batch_ys = next(shuffle_batch(X_train, y_train, 100))
3     sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
4     correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
5     accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
6     if i % 100 == 0:
7         print(sess.run(accuracy, feed_dict={x: X_test, y_: y_test}))
```

```
0.4843
0.8975
0.9055
0.9053
0.905
0.9106
0.9124
0.9151
0.9166
0.9163
```

```
1 tf.convert_to_tensor(X_train).get_shape()
2
```

```
TensorShape([Dimension(60000), Dimension(784)])
```

가중치 텐서와 바이어스 텐서를 만듭니다.

```
1
2 W = tf.Variable(tf.zeros([784, 10]))
3 b = tf.Variable(tf.zeros([10]))
```

훈련 이미지 데이터를 넣을 플레이스홀더와 소프트맥스 텐서를 만듭니다.

```
1 x = tf.placeholder("float", [None, 784])
2 y = tf.nn.softmax(tf.matmul(x,W) + b)
```

실제 레이블을 담기위한 텐서와 교차 엔트로피 방식을 이용하는 그라디언트 디센트 방식을 선택합니다.


```
1 y_ = tf.placeholder("float", [None, 10])
2 cross_entropy = -tf.reduce_sum(y_*tf.log(y))
3 train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

변수를 초기화하고 세션을 시작합니다.

```
1 sess = tf.Session()
2 sess.run(tf.global_variables_initializer())
```

1000의 반복을 수행하고 결과를 출력합니다. 최종 정확도는 91% 정도 입니다.

```
1 for i in range(1000):
2     batch_xs, batch_ys = next(shuffle_batch(X_train, y_train, 100))
3     sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
4     correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
5     accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
6     if i % 100 == 0:
7         print(sess.run(accuracy, feed_dict={x: X_test, y_: y_test}))
```

 0.228  
0.8783  
0.8809  
0.9081  
0.9123  
0.9137  
0.9125  
0.9119  
0.9212  
0.9125

1