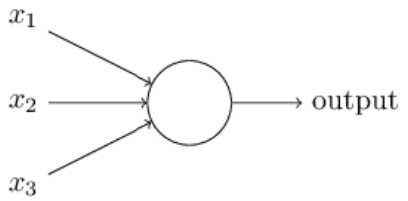
인공지능 실습
(Python)

MLP

로젠블라트의 퍼셉트론

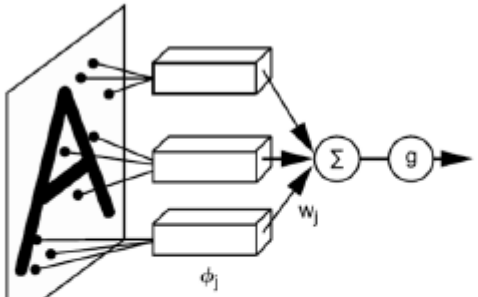
- 다수의 신호를 입력으로 받아 하나의 신호를 출력
- 다중의 입력을 하나의 2진 값으로 출력



퍼셉트론의 학습

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

- 델타 규칙이라는 학습 규칙을 사용
  - 만일 어떤 신경세포의 활성이 다른 신경세포가 잘못된 출력을 내는데 공헌을 했다면, 두 신경 세포간의 연결 가중치를 그것에 비례하게 조절
- 목적 패턴을 이용한 신경망의 학습을 인위로 제어



$\phi_j()$와 $g()$는 고정된 임계값
w_j는 가변 가중치

[그림 13-10] 퍼셉트론

퍼셉트론 신경망에 적용된 학습규칙 → **델타 규칙(Delta Rule)**

델타 규칙

$$w_{ij}^{new} = w_{ij}^{old} + \alpha e_j a_i \quad (13.3)$$

$$e_j = t_j - b_j \quad (13.4)$$

※ w_{ij}^{new} : 신경세포 i, j 사이의 조절된 후 연결 가중치

w_{ij}^{old} : 신경세포 i, j 사이의 조절되기 전 연결 가중치

α : 학습률 ($0 < \alpha \leq 1$)

e_j : 신경세포 j의 오차

a_i : 입력층 신경세포 i의 활성값

t_j : 목적 패턴의 출력층 신경세포 j에 대응하는 성분값

b_j : 출력층 신경세포 j의 활성값

델타 규칙

델타 규칙을 사용하여 신경망을 학습하는 과정을 요약하면 다음과 같다.

① 입력층에 입력 패턴을 제시
② 신경망을 동작
③ 델타 규칙에 의해 연결 가중치를 조절

$$w_{ij}^{new} = w_{ij}^{old} + \alpha e_j a_i \qquad (13.5)$$

④ 신경망이 완전하게 학습될 때까지 과정 ①~③을 입력 패턴에 대해 반복

델타 규칙에 의한 학습 과정은 감독 학습 방법이 사용하는 일반적인 형태의 학습 과정과 동일하다. 단지 과정 ③에서 연결 가중치 조절식만 다를 뿐이다.

델타 규칙에서 학습 완료 정도를 나타내는 오차는 헤브의 규칙에서와 같다. 앞서 (13.4)에서와 같이 신경망의 실제 출력 패턴의 목적 패턴의 차이에 의해 계산된다. 이 부분에서 중요한 것은 "잘못된 출력을 낸 신경세포들의 연결 가중치를 그 정도에 비례하여 조절한다"이다.

MNIST 데이터셋

- 머신러닝의 고전적인 문제
- 학습데이터 60000개, 테스트데이터 10000개
  - 또는 학습데이터 55000개, 검증데이터 5000개
- 0~9까지 필기 숫자들의 그레이스케일 28X28 픽셀 이미지를 보고 판별하는 문제

```
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt

mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

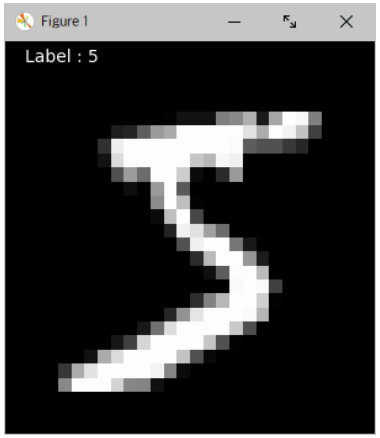
```
2020-06-03 15:02:19.004027: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic
 library cudart64_101.dll
Backend Qt5Agg is interactive backend. Turning interactive mode on.
```

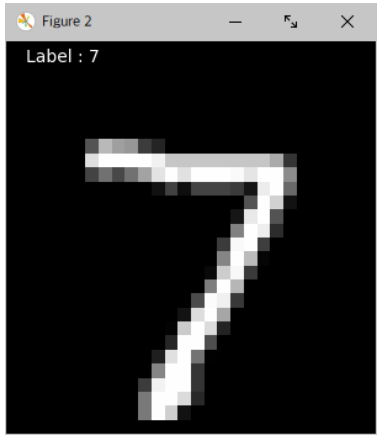
MNIST 샘플 데이터 보기

```
plt.rcParams['toolbar'] = 'None'

fg1 = plt.figure(1, figsize=(3, 3))
fg1.canvas.window().statusBar().setVisible(False)
ax1 = fg1.add_axes([0, 0, 1, 1])
ax1.imshow(train_images[0], cmap='gray', aspect='auto')
ax1.axis('off')
ax1.text(1, 1, "Label : {}".format(train_labels[0]), fontsize=10, color='white')

fg2 = plt.figure(2, figsize=(3, 3))
fg2.canvas.window().statusBar().setVisible(False)
ax2 = fg2.add_axes([0, 0, 1, 1])
ax2.imshow(test_images[0], cmap='gray', aspect='auto')
ax2.axis('off')
ax2.text(1, 1, "Label : {}".format(test_labels[0]), fontsize=10, color='white')
```

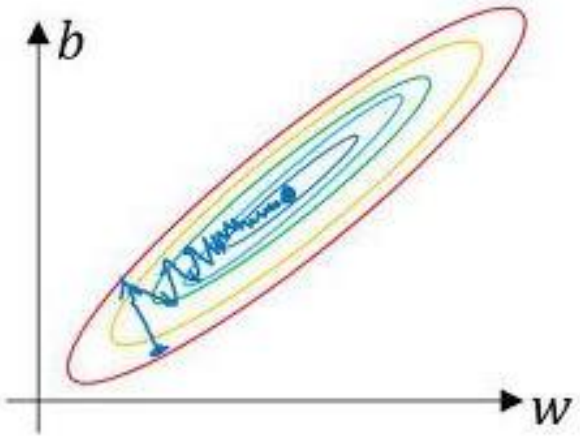


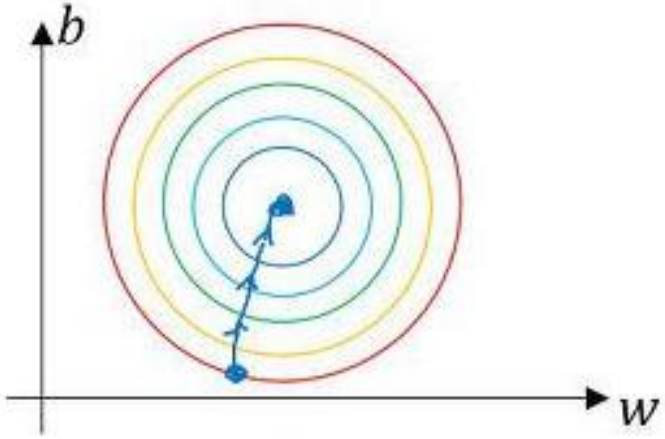


데이터 정규화(normalize)

- 머신러닝 알고리즘은 데이터가 가진 특징(feature)들을 비교하여 데이터의 패턴을 찾음
- 데이터가 가진 특징들의 스케일이 심하게 차이가 나는 경우 문제

```
#%% normalize
train_input = 정규화된 train_images
test_input = 정규화된 test_images
```





One-hot encoding/vector

- 데이터를 쉽게 중복 없이 표현할 때 사용하는 형식
- discrete representation

```
#%% target to one-hot vector
train_target = keras.utils.to_categorical(train_labels)
test_target = keras.utils.to_categorical(test_labels)
```

	0	1	2	3	4	5	6	7	8	9
0	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000
1	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2	0.00000	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000
3	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000
5	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
6	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
7	0.00000	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
8	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

MLP 구현

분류를 위한 모델 (Multi Layer Perceptron)

```
#%% define network
model = keras.Sequential(
    layers=[
        keras.layers.Flatten(input_shape=(28, 28), name='Input'),
        keras.layers.Dense(128, activation='sigmoid', name='Hidden_1'),
        keras.layers.Dense(128, activation='sigmoid', name='Hidden_2'),
        keras.layers.Dense(128, activation='sigmoid', name='Hidden_3'),
        keras.layers.Dense(10, activation='sigmoid', name='Output')
    ],
    name="MNIST_Classifer"
)
model.summary()
```

```
Model: "MNIST_Classifer"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input (Flatten)              (None, 784)               0
_________________________________________________________________
Hidden_1 (Dense)             (None, 128)               100480
_________________________________________________________________
Hidden_2 (Dense)             (None, 128)               16512
_________________________________________________________________
Hidden_3 (Dense)             (None, 128)               16512
_________________________________________________________________
Output (Dense)               (None, 10)                1290
=================================================================
Total params: 134,794
Trainable params: 134,794
Non-trainable params: 0
_________________________________________________________________
```

분류를 위한 모델 (Multi Layer Perceptron)

- 학습 방법 : 확률적 경사 하강법
- 에러(손실) : 평균제곱오차
- 성능 평가 척도 : 분류 정확도

```
model.compile(optimizer=keras.optimizers.SGD(lr=0.2),
              loss='mean_squared_error',
              metrics=['accuracy'])
```

분류기 학습

- 학습 반복 횟수(epoch) : 20회

```
#%% fit model to train data
model.fit(train_input, train_target, epochs=20, verbose=2)
```

```
Epoch 17/20
60000/60000 - 2s - loss: 0.0414 - accuracy: 0.7585
Epoch 18/20
60000/60000 - 2s - loss: 0.0384 - accuracy: 0.7837
Epoch 19/20
60000/60000 - 2s - loss: 0.0357 - accuracy: 0.8023
Epoch 20/20
60000/60000 - 2s - loss: 0.0331 - accuracy: 0.8173
```

학습한 모델 테스트

- 테스트 정확도와 학습 정확도의 비교
- 오버피팅 / 언더피팅

```
10000/10000 - 0s - loss: 0.0374 - accuracy: 0.7817

Test accuracy: 0.7817
```

```
#%% evaluate model by test data
test_loss, test_acc = model.evaluate(test_input, test_target, verbose=2)
print('\nTest accuracy:', test_acc)

predictions = model.predict(test_input)
```

샘플 데이터 테스트

```
sample_pred = model.predict(test_sample)
sample_result = np.argmax(sample_pred)
```