2장
분류 알고리즘의 기초

Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진

2019.

HJYU@UOS.AC.KR

CHAPTER 2 분류 알고리즘의 기초

2.1 Logistic 회귀를 이용한 이항 분류기 (binary classifier)

2.1.1 확률을 이용한 오차 평가

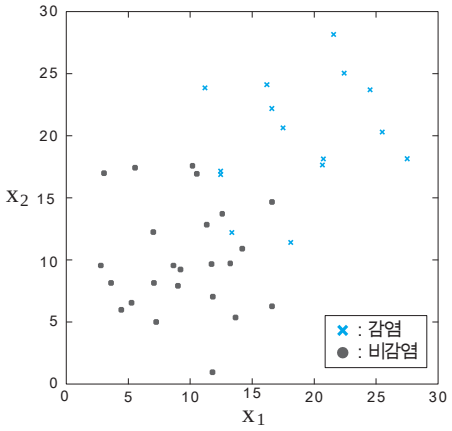
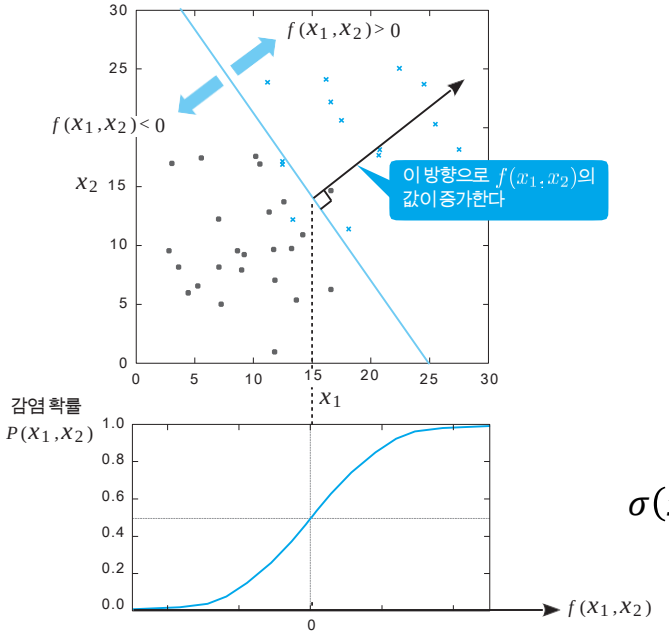


그림 2-2 예비 검사 결과와 실제 감염 상황을 나타낸 데이터



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

그림 2-3 직선을 이용한 분류와 감염 확률로의 변환

review

Binary Classification

- Two classes : $d = 0$, $d = 1$
- 훈련 데이터 $\{(\mathbf{x}_n, \mathbf{d}_n)\}$, $n=1, \ldots, N$ $\boldsymbol{d}_n$: target

$$p(d=1|\mathbf{x}) \approx y(\mathbf{x};\ \mathbf{w})$$

$$p(d|\mathbf{x}) = p(d=1|\mathbf{x})^d\, p(d=0|\mathbf{x})^{1-d} \qquad f(u) = \frac{1}{1+e^{-u}}$$

- Maximum likelihood estimation (최우추정법)

  주어진 데이터를 바르게 예측할 확률을 최대화 하는 것.

$$L(\mathbf{w}) \equiv \prod_{n=1}^{N} p(d_n|\mathbf{x}_n;\mathbf{w}) = \prod_{n=1}^{N} \{y(\mathbf{x}_n;\mathbf{w})\}^{d_n} \{1-y(\mathbf{x}_n;\mathbf{w})\}^{1-d_n}$$

$E = -\log P$

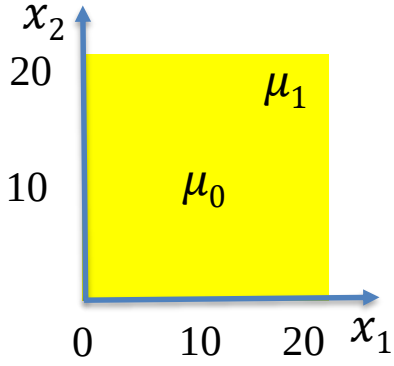
$$E(\mathbf{w}) = -\sum_{n=1}^{N} [d_n \log y(\mathbf{x}_n;\mathbf{w}) + (1-d_n)\log\{1-y(\mathbf{x}_n;\mathbf{w})\}]$$

2.1.2 Maximum likelihood estimation

- Chapter02/Maximum likelihood estimation example.ipynb

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import multivariate_normal, permutation
import pandas as pd
from pandas import DataFrame, Series
```

데이터 생성




```
1.  np.random.seed(20160512) #seed에 따라 동일한 난수 생성
2.  # t=0 (비감염)인 데이터를 난수로 생성
3.  n0, mu0, variance0 = 20, [10, 11], 20     # 20개
4.  data0 = multivariate_normal(mu0, np.eye(2)*variance0 ,n0)
5.  df0 = DataFrame(data0, columns=['x1','x2'])
6.  df0['t'] = 0
7.   # t=1 (감염)인 데이터를 난수로 생성
8.  n1, mu1, variance1 = 15, [18, 20], 22     # 15개
9.  data1 = multivariate_normal(mu1, np.eye(2)*variance1 ,n1)
10. df1 = DataFrame(data1, columns=['x1','x2'])
11. df1['t'] = 1
12. df = pd.concat([df0, df1], ignore_index=True)        # 하나로 모아
13.  # 무작위로 순번을 변경
14. train_set = df.reindex(permutation(df.index)).reset_index(drop=True)
```

```
array([[20., 0.],
       [0., 20.]])
```

DataFrame

- Spread sheet 형식의 2차원 data set

train_set

	x1	x2	t
0	20.729880	18.209359	1
1	16.503919	14.685085	0
2	5.508661	17.426775	0
3	9.167047	9.178837	0
…			

(x1, x2)와 t를 각각 모은 것을 NumPy의 array 오브젝트로 추출해둔다

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ x_{13} & x_{23} \\ \vdots & \vdots \end{pmatrix} \qquad \mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \end{pmatrix}$$

Textbook: Old style

```
train_x = train_set[['x1','x2']].as_matrix()
train_t = train_set['t'].as_matrix().reshape([len(train_set), 1])
```

Panda to numpy

```
train_x = train_set[['x1','x2']].to_numpy()
train_t = train_set['t'].to_numpy().reshape([len(train_set), 1])
```

[MLE-05] Training Set 데이터에 대해 t=1일 확률을 구하는 계산식 p 준비

1. x = tf.placeholder(tf.float32, [None, 2])
2. w = tf.Variable(tf.zeros([2, 1]))
3. w0 = tf.Variable(tf.zeros([1]))
4. f = tf.matmul(x, w) + w0
5. p = tf.sigmoid(f)

None : 임의의 개수

$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ x_{13} & x_{23} \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + \begin{pmatrix} w_0 \\ w_0 \\ w_0 \\ \vdots \end{pmatrix}$$

각각의 성분에 대하여 sigmoid 계산

$$\begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} \sigma(f_1) \\ \sigma(f_2) \\ \sigma(f_3) \\ \vdots \end{pmatrix}$$

Broadcasting rule: 다차원 리스트에 스칼라 값을 더할 경우 리스트의 각 요소에 동일한 값이 더해짐.

그림 2-5 Broadcasting rule of the tensorflow

(1) 행렬과 스칼라의 덧셈은 각 성분에 대한 덧셈이다

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} + (10) = \begin{pmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{pmatrix}$$

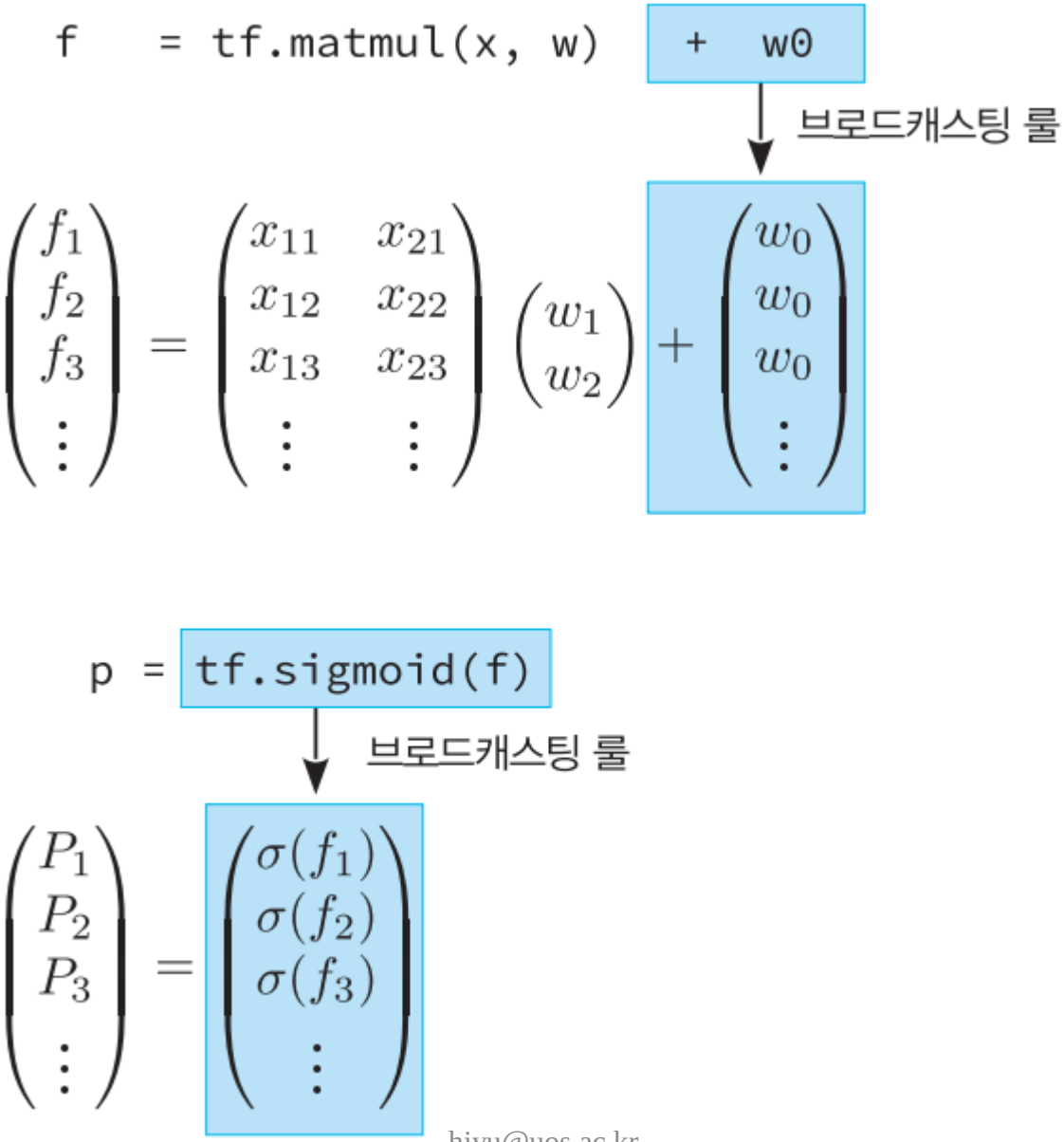
(2) 크기가 같은 행렬의 '*' 연산은 성분별 곱셈이다

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} * \begin{pmatrix} 10 & 100 & 1000 \\ 10 & 100 & 1000 \\ 10 & 100 & 1000 \end{pmatrix} = \begin{pmatrix} 10 & 200 & 3000 \\ 40 & 500 & 6000 \\ 70 & 800 & 9000 \end{pmatrix}$$

(3) 스칼라를 넘겨받는 함수를 행렬에 적용하면 각 성분에 함수가 적용된다

$$\sigma \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} \sigma(1) \\ \sigma(2) \\ \sigma(3) \end{pmatrix}$$

그림 2-6 변수 f와 변수 p가 나타내는 계산식

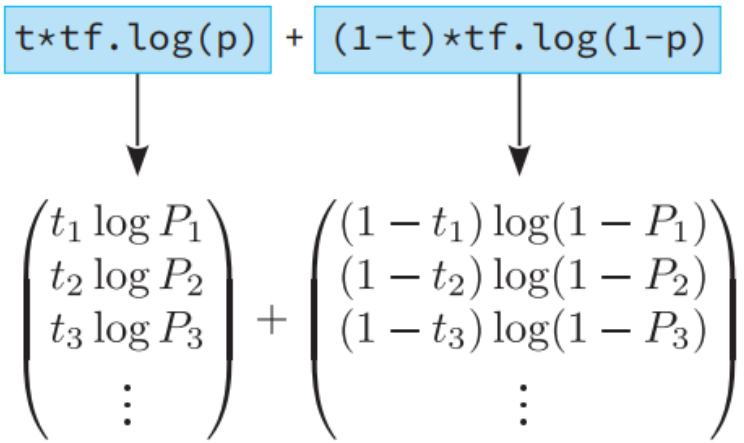


[MLE-06] loss와 training algorithm train_step을 정의

```
t = tf.placeholder(tf.float32, [None, 1])
loss = -tf.reduce_sum(t*tf.log(p) + (1-t)*tf.log(1-p))
train_step = tf.train.AdamOptimizer().minimize(loss)
```

reduce_sum :
모든 요소 합산

$$\text{loss} = -\sum_{n=1}^{N} [t_n \log P(x_{1n}, x_{2n}) + (1 - t_n) \log \{1 - P(x_{1n}, x_{2n})\}]$$



$$\begin{pmatrix} t_1 \log P_1 \\ t_2 \log P_2 \\ t_3 \log P_3 \\ \vdots \end{pmatrix} + \begin{pmatrix} (1-t_1) \log(1-P_1) \\ (1-t_2) \log(1-P_2) \\ (1-t_3) \log(1-P_3) \\ \vdots \end{pmatrix}$$

그림 2-7 **tf.reduce_sum**
의 인수 부분

[MLE-07] 정답률 accuracy를 정의

```
correct_prediction = tf.equal(tf.sign(p-0.5), tf.sign(t-0.5))
# if p > 0.5 → 1 else 0
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
# cast: Bool → 1,0
# reduce_mean : 각 성분의 평균
```

- [MLE-08] 세션을 준비하고 Variable을 초기화

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

[MLE-09] 경사 하강법에 의한 파라미터 최적화

```
i = 0
for _ in range(20000):
    i += 1
    sess.run(train_step, feed_dict={x:train_x, t:train_t})
    if i % 2000 == 0:
        loss_val, acc_val = sess.run( [loss, accuracy],
                feed_dict={x:train_x, t:train_t})
        print ('Step: %d, Loss: %f, Accuracy: %f'
                % (i, loss_val, acc_val))
```

20000회 반복

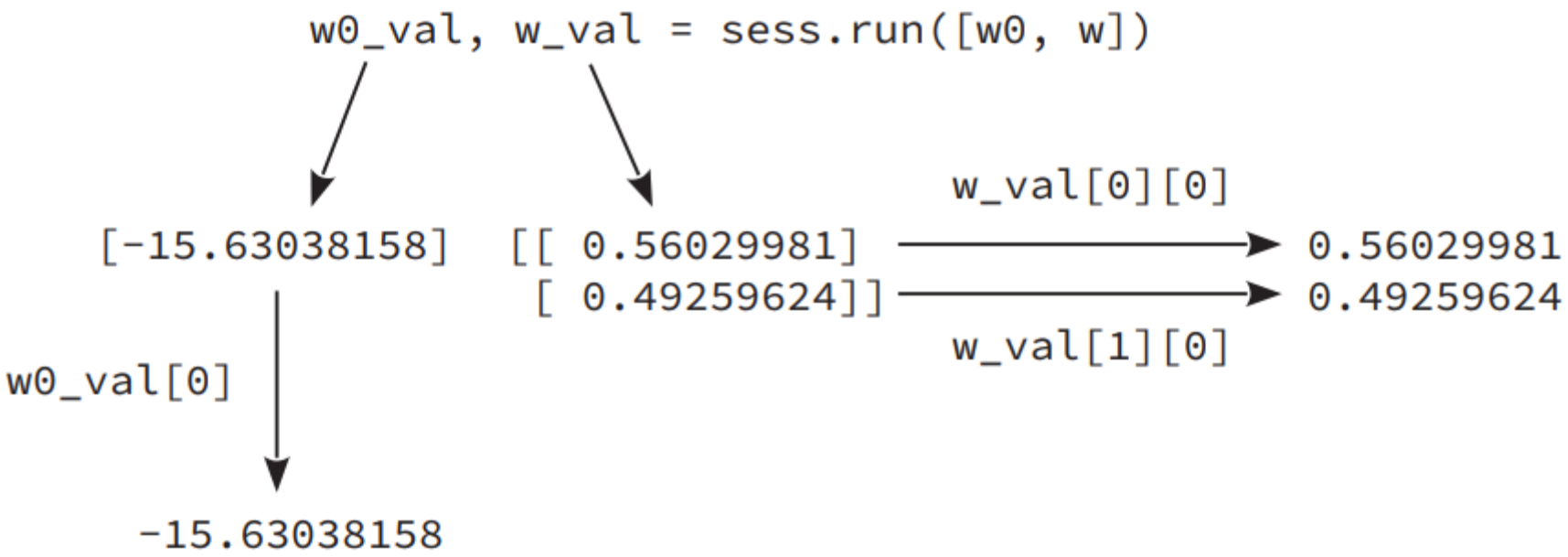
```
Step: 2000, Loss: 15.165894, Accuracy: 0.885714
Step: 4000, Loss: 10.772635, Accuracy: 0.914286
Step: 6000, Loss: 8.197757, Accuracy: 0.971429
Step: 8000, Loss: 6.576121, Accuracy: 0.971429
Step: 10000, Loss: 5.511973, Accuracy: 0.942857
Step: 12000, Loss: 4.798011, Accuracy: 0.942857
Step: 14000, Loss: 4.314180, Accuracy: 0.942857
Step: 16000, Loss: 3.986264, Accuracy: 0.942857
Step: 18000, Loss: 3.766511, Accuracy: 0.942857
Step: 20000, Loss: 3.623064, Accuracy: 0.942857
```

Loss : 감소
Accuracy : ?

[MLE-10] 이 시점의 파라미터 값을 추출

1. w0_val, w_val = sess.run([w0, w])
2. w0_val, w1_val, w2_val = w0_val[0], w_val[0][0], w_val[1][0]
3. print(w0_val, w1_val, w2_val)

- -15.6304 0.5603 0.492596



[MLE-11] 추출한 파라미터 값을 이용해 결과를 그래프로 출력

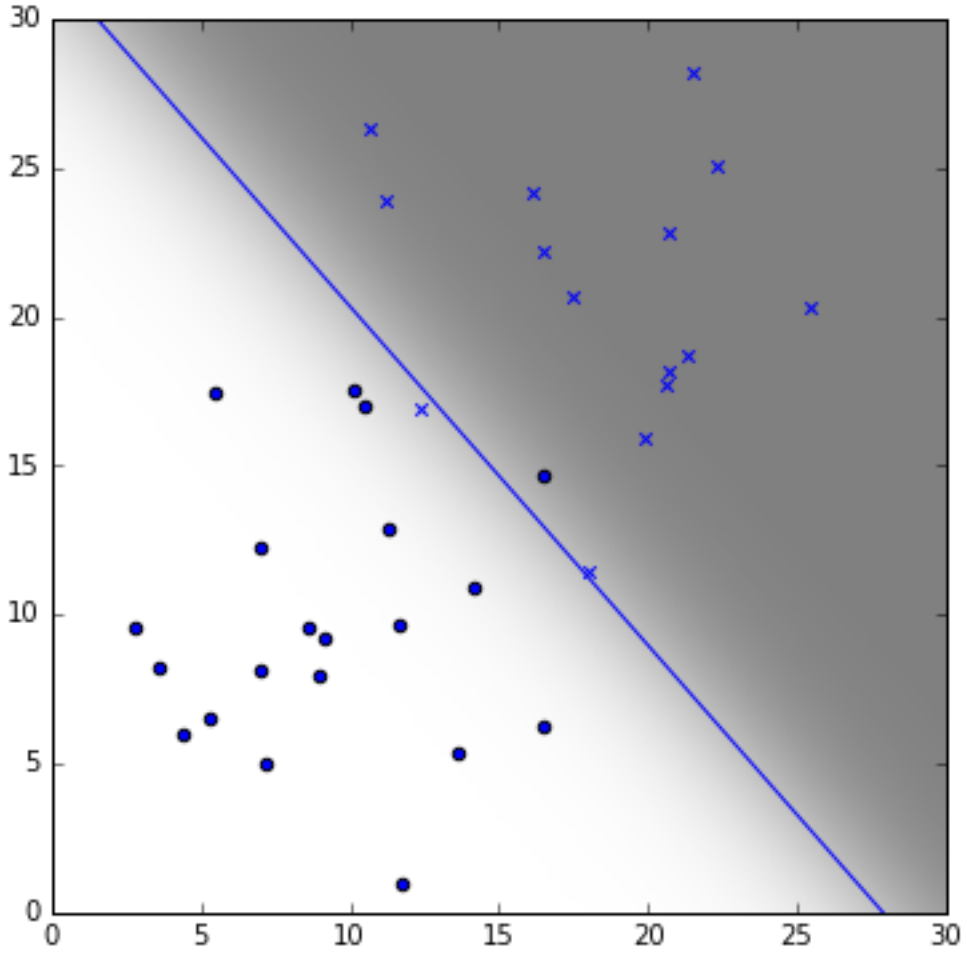
```
1.  train_set0 = train_set[train_set['t']==0]
2.  train_set1 = train_set[train_set['t']==1]
3.   # 산포도
4.  fig = plt.figure(figsize=(6,6))
5.  subplot = fig.add_subplot(1,1,1)
6.  subplot.set_ylim([0,30])
7.  subplot.set_xlim([0,30])
8.  subplot.scatter(train_set1.x1, train_set1.x2, marker='x')
9.  subplot.scatter(train_set0.x1, train_set0.x2, marker='o')
10.  # 직선
11. linex = np.linspace(0,30,10)
12. liney = - (w1_val*linex/w2_val + w0_val/w2_val)
13. subplot.plot(linex, liney)
14.  # 확률의 변화
15. field = [[(1 / (1 + np.exp(-(w0_val + w1_val*x1 + w2_val*x2))))
16.           for x1 in np.linspace(0,30,100)]
17.          for x2 in np.linspace(0,30,100)]
18. subplot.imshow(field, origin='lower', extent=(0,30,0,30),
19.                cmap=plt.cm.gray_r, alpha=0.5)
```

Training set 중에서 t=0, t=1 인 data를 개별로 추출

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$
$$x_2 = -(w_1 x_1 / w_2 + w_0 x_0 / w_2)$$

$$f(u) = \frac{1}{1+e^{-u}}$$

그림 2-9 실행 결과



Accuracy : 100%
는 불가능

2.1.3 Validation set 을 이용한 검증

- Overfitting
  - Training set 에 대한 정답률은 높은 방면,
    미지의 데이터에 대한 정확도는 낮음.
- 해결방법:
  - 일부 데이터를 test용으로 사용 → Validation set

[CAF-02] Training Set 데이터를 준비

```
n0, mu0, variance0 = 800, [10, 11], 20
data0 = multivariate_normal(mu0, np.eye(2)*variance0 ,n0)
df0 = DataFrame(data0, columns=['x','y'])
df0['t'] = 0

n1, mu1, variance1 = 600, [18, 20], 22
data1 = multivariate_normal(mu1, np.eye(2)*variance1 ,n1)
df1 = DataFrame(data1, columns=['x','y'])
df1['t'] = 1

df = pd.concat([df0, df1], ignore_index=True)
df = df.reindex(permutation(df.index)).reset_index(drop=True)
```

20%의 데이터를 테스트 세트로 분리

```
num_data = int(len(df)*0.8)
train_set = df[:num_data]
test_set = df[num_data:]
```

(x, y)와 t를 각각 모은 것을 NumPy의 array 오브젝트로 추출

```
train_x = train_set[['x','y']].to_numpy()
train_t = train_set['t'].to_numpy().reshape([len(train_set), 1])
test_x = test_set[['x','y']].to_numpy()
test_t = test_set['t'].to_numpy().reshape([len(test_set), 1])
```

Parameter optimization을 2500회 반복하면서 training set와 test set에 대한 **accuracy** 변화를 기록

```
train_accuracy = [ ]
test_accuracy = [ ]
for _ in range(2500):
    sess.run(train_step, feed_dict={x:train_x, t:train_t})
    acc_val = sess.run(accuracy, feed_dict={x:train_x, t:train_t})
    train_accuracy.append(acc_val)
    acc_val = sess.run(accuracy, feed_dict={x:test_x, t:test_t})
    test_accuracy.append(acc_val)
```

결과를 그래프로 출력

```
fig = plt.figure(figsize=(8,6))
subplot = fig.add_subplot(1,1,1)
subplot.plot(range(len(train_accuracy)), train_accuracy,
        linewidth=2, label='Training set')
subplot.plot(range(len(test_accuracy)), test_accuracy,
        linewidth=2, label='Test set')
subplot.legend(loc='upper left')
plt.show()
```

2.2 소프트맥스 함수와 다항 분류기

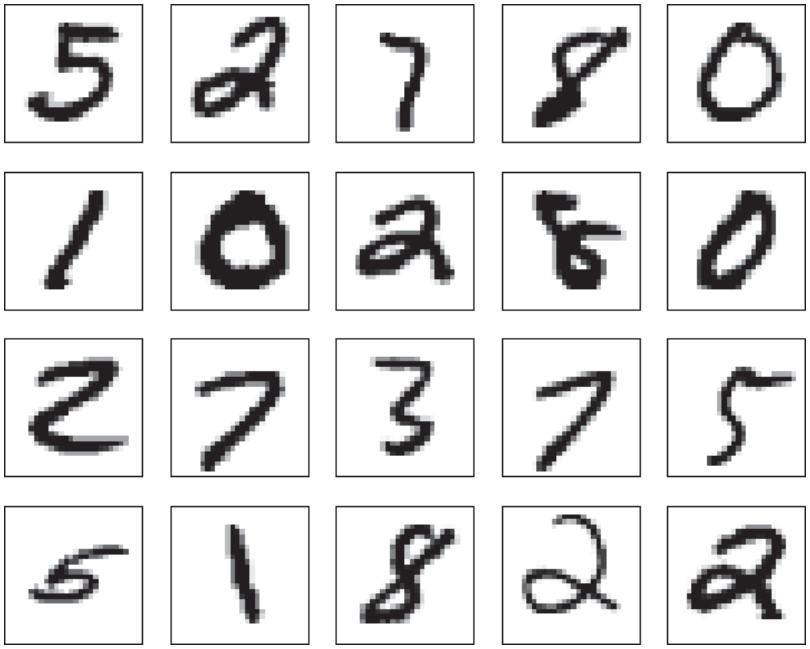


그림 2-11 필기 숫자 이미지 데이터

2.2.1 선형 다항 분류기의 구조

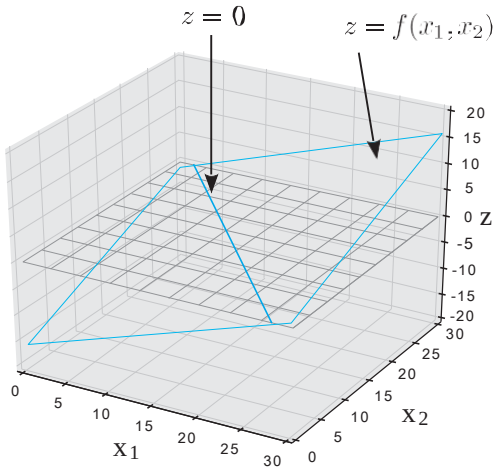


그림 2-12 1차 함수 $f_1(x_1,x_2)$를 3차원 그래프에 표시

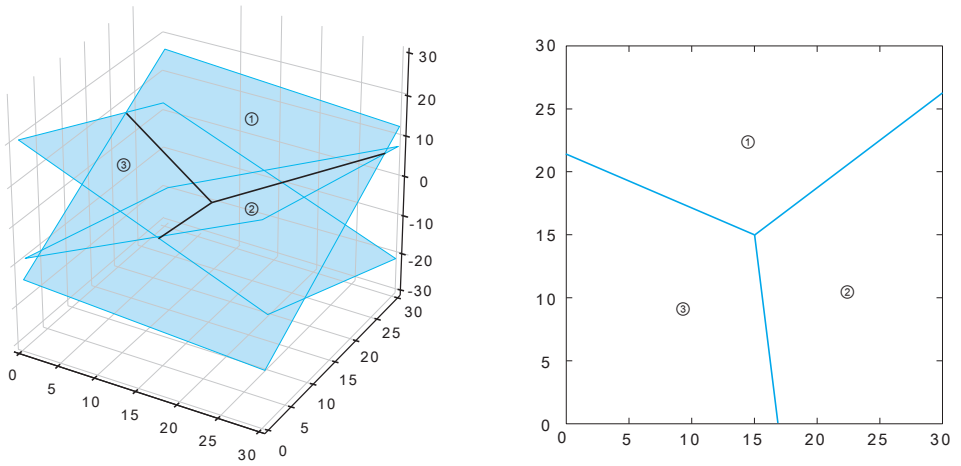
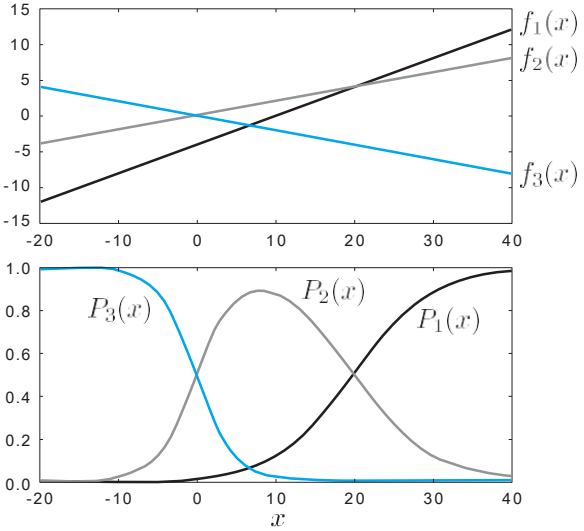


그림 2-13 3장의 판을 이용해 평면을 3분할한 모습

$$f_1(x_1,x_2) = w_{01} + w_{11}x_1 + w_{21}x_2$$
$$f_2(x_1,x_2) = w_{02} + w_{12}x_1 + w_{22}x_2$$
$$f_3(x_1,x_2) = w_{03} + w_{13}x_1 + w_{23}x_2$$

2.2.2 Softmax function을 이용한 확률로의 변환



- Hardmax : 경계선
  - 최대가 되는 곳
- Softmax : 확률

$$P_k(x_1, \ldots, xM) = \frac{e^{f_k\ (x_1,\ldots,xM)}}{\sum_{j=1}^{K} e^{f_j\ (x_1,\ldots,xM)}}$$

점 $(x_1, \ldots, xM)$이 k번째 영역에 있을 확률

$$f_k(x_1,x_2 ,\ldots, x_M) = w_{0k} + w_{1k}x_1 + w_{2k}x_2 + \ldots + w_{Mk}x_M$$

2.3 다항 분류기를 이용한 필기 문자 분류

2.3.1 MNIST 데이터 세트 이용 방법

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
```

[MDS-02] MNIST 데이터 세트를 다운로드해서 오브젝트에 저장한다.

```
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting /tmp/data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

[MDS-03] Training Set에서 10개의 데이터를 추출하고 이미지 데이터와 라벨을 각각의 변수에 저장한다.

```
images, labels = mnist.train.next_batch(10)
```

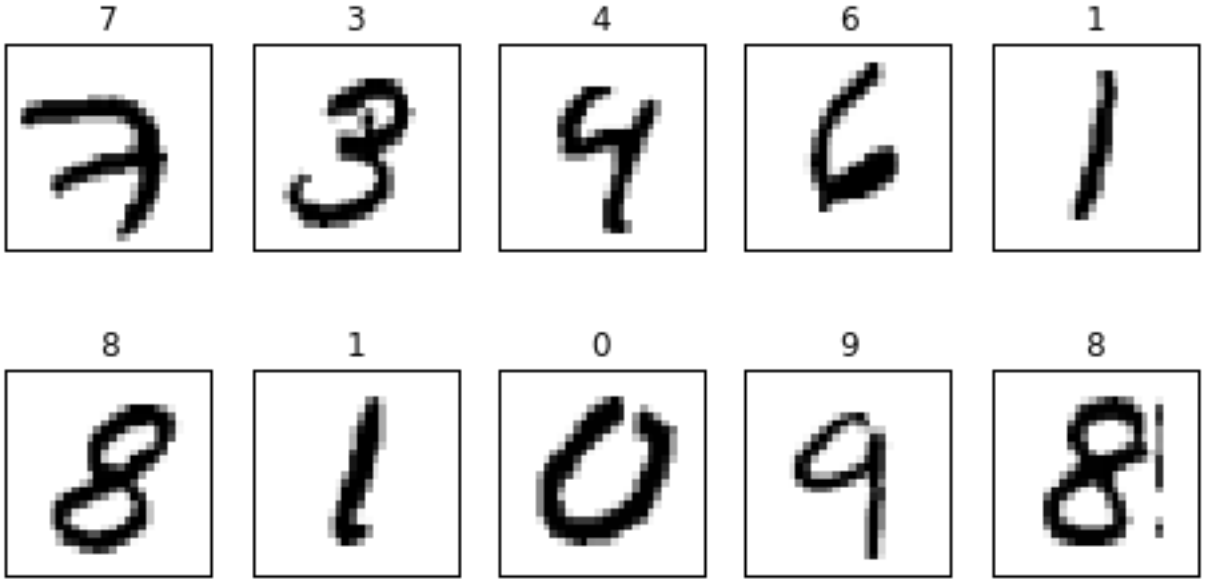
[MDS-05] 해당 라벨을 확인한다. 첫 번째 요소를 0번째로 볼 때 앞에서부터 7번째 요소가 1로 되어 있으므로, 숫자 '7'의 이미지라는 것을 나타낸다.

```
print labels[0]
[ 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
One-hot vector, one-hot encoding, 1-of-K vector
k 번째 요소만 1로 되어 있는 vector
```

[MDS-06] 이미지 데이터를 실제 이미지로 출력해본다.

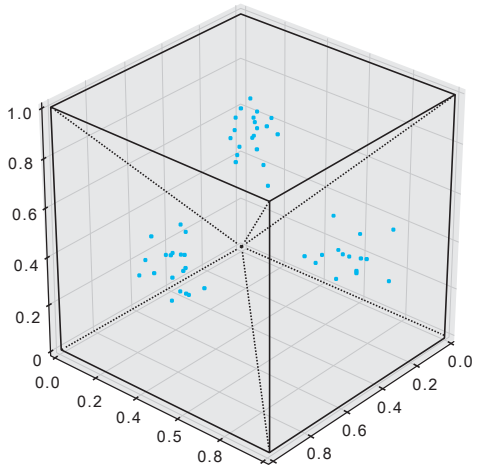
```
fig = plt.figure(figsize=(8,4))
for c, (image, label) in enumerate(zip(images, labels)):
    subplot = fig.add_subplot(2,5,c+1)
    subplot.set_xticks([])
    subplot.set_yticks([])
    subplot.set_title('%d' % np.argmax(label))
    subplot.imshow(image.reshape((28,28)), vmin=0, vmax=1,
          cmap=plt.cm.gray_r, interpolation="nearest")
```

- vmin=0, vmax=1 : 밝기의 최소, 최대
- interpolation="nearest" : image를 부드럽게 출력


7
3
4
6
1
8
1
0
9
8

2.3.2 Image data의 classification algorithm

- Image : 784 dimensional vector (28 x 28)
- 동일한 숫자의 image는 784차원 공간상에서 서로 가까운 위치에 존재



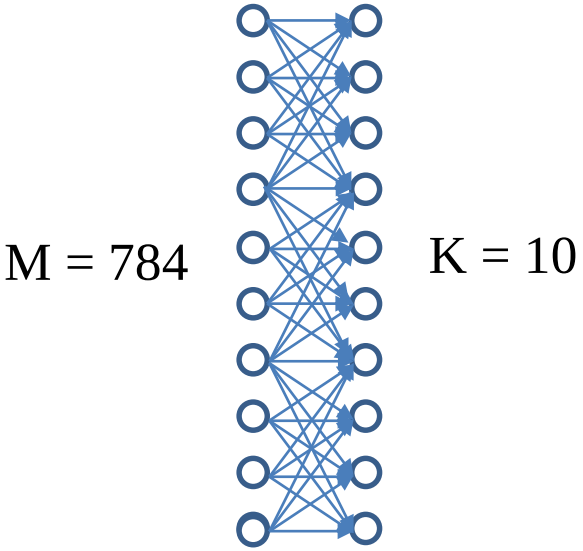
다항 분류기의 1차 함수를 행렬로 정리한 모습

$$\mathbf{F} = \mathbf{XW} \oplus \mathbf{w}$$

$$\begin{pmatrix} f_1(\mathbf{x}_1) & f_2(\mathbf{x}_1) & \cdots & f_K(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_K(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(\mathbf{x}_N) & f_2(\mathbf{x}_N) & \cdots & f_K(\mathbf{x}_N) \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{M1} \\ x_{12} & x_{22} & \cdots & x_{M2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1N} & x_{2N} & \cdots & x_{MN} \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1K} \\ w_{21} & w_{22} & \cdots & w_{2K} \\ \vdots & \vdots & \vdots & \vdots \\ w_{M1} & w_{M2} & \cdots & w_{MK} \end{pmatrix} + \begin{pmatrix} w_{01} & w_{02} & \cdots & w_{0K} \\ w_{01} & w_{02} & \cdots & w_{0K} \\ \vdots & \vdots & \vdots & \vdots \\ w_{01} & w_{02} & \cdots & w_{0K} \end{pmatrix}$$

$f_k(x_1, x_2, \ldots, x_M) = w_{0k} + w_{1k}x_1 + w_{2k}x_2 + \ldots + w_{Mk}x_M$

- M = 784 dimension
- K = 10 classes
- W = M x K
- w : 10 dimensional vector



2.3.3 텐서플로를 이용한 트레이닝 실행

MNIST softmax estimation.ipynb

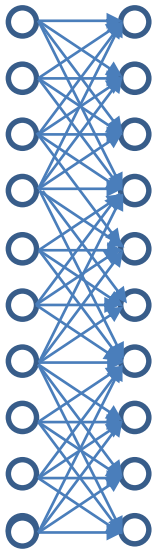
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
np.random.seed(20160604)
```

[MSE-02] MNIST 데이터 세트를 준비한다.

```
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

[MSE-03] 소프트맥스 함수에 의한 확률 p 계산식을 준비한다.

```
x = tf.placeholder(tf.float32, [None, 784])
w = tf.Variable(tf.zeros([784, 10]))
w0 = tf.Variable(tf.zeros([10]))       # bias
f = tf.matmul(x, w) + w0
p = tf.nn.softmax(f)
```



[MSE-04] 오차 함수 loss와 트레이닝 알고리즘 train_step을 준비한다.

```
t = tf.placeholder(tf.float32, [None, 10])
loss = -tf.reduce_sum(t * tf.log(p))
train_step = tf.train.AdamOptimizer().minimize(loss)
```

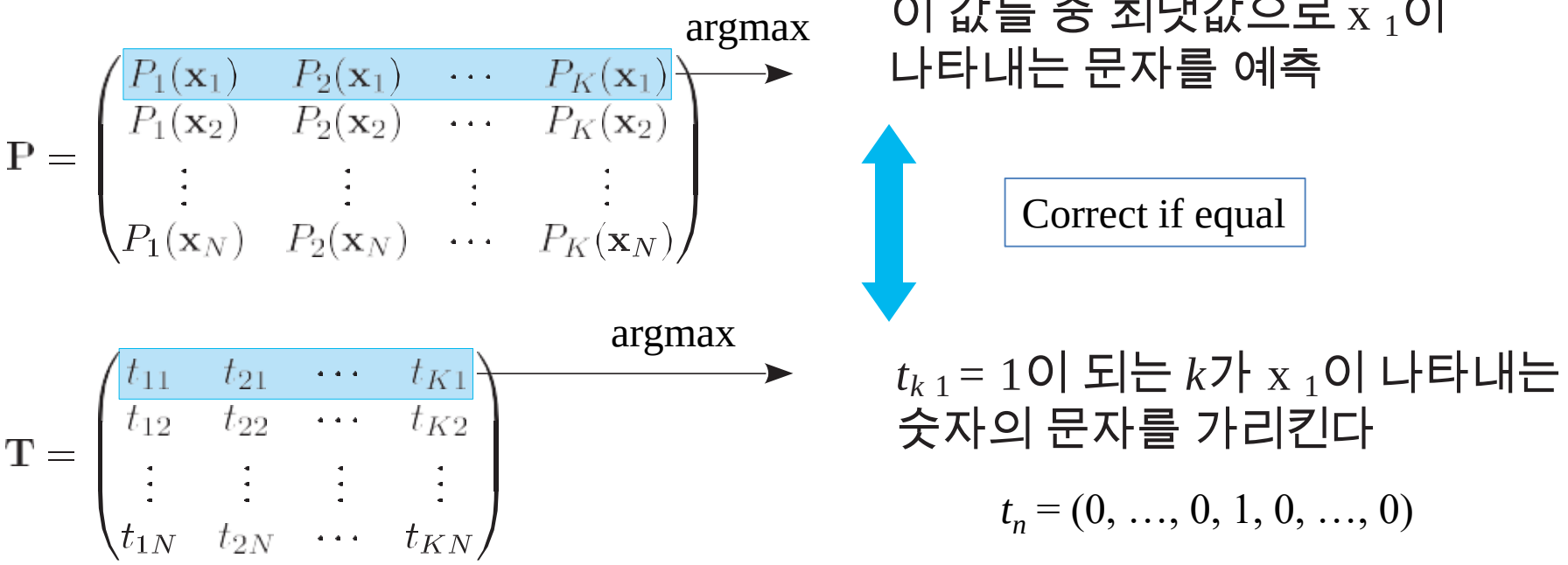
1: 가로 방향
0: 세로 방향

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} d_{nk} \log y_k(\mathbf{x}_n; \mathbf{w})$$

[MSE-05] 정답률 accuracy를 정의한다

```
correct_prediction = tf.equal(tf.argmax(p, 1), tf.argmax(t, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

- tf.cast : Bool 값을 1,0으로 변환



$$\mathbf{P} = \begin{pmatrix} P_1(\mathbf{x}_1) & P_2(\mathbf{x}_1) & \cdots & P_K(\mathbf{x}_1) \\ P_1(\mathbf{x}_2) & P_2(\mathbf{x}_2) & \cdots & P_K(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ P_1(\mathbf{x}_N) & P_2(\mathbf{x}_N) & \cdots & P_K(\mathbf{x}_N) \end{pmatrix}$$

$$\mathbf{T} = \begin{pmatrix} t_{11} & t_{21} & \cdots & t_{K1} \\ t_{12} & t_{22} & \cdots & t_{K2} \\ \vdots & \vdots & \vdots & \vdots \\ t_{1N} & t_{2N} & \cdots & t_{KN} \end{pmatrix}$$

그림 2-21 확률이 최대가 되는 문자로 예측을 실행

np.argmax 함수의 활용 예

$$\mathbf{M} = \begin{pmatrix} 0 & 20 & 40 & 60 \\ 60 & 0 & 20 & 40 \\ 40 & 60 & 0 & 20 \end{pmatrix}$$

가로 방향으로 검색 : $\text{np.argmax}(\mathbf{M}, 1) = (3, 0, 1)$

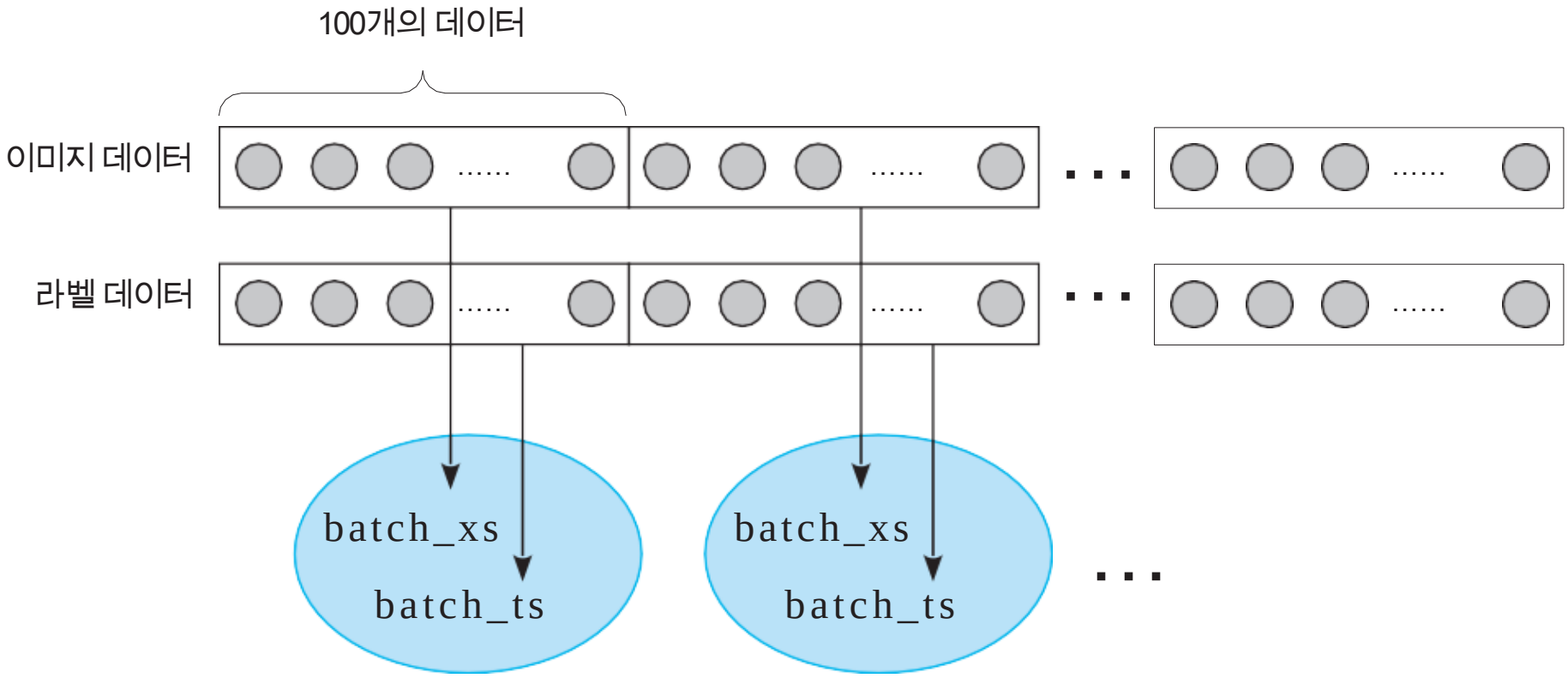
세로 방향으로 검색 : $\text{np.argmax}(\mathbf{M}, 0) = (1, 2, 0, 0)$

} 최대 요소의 인덱스

```
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
i = 0
for _ in range(2000):
    i += 1
    batch_xs, batch_ts = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, t: batch_ts})
    if i % 100 == 0:
        loss_val, acc_val = sess.run([loss, accuracy],
            feed_dict={x:mnist.test.images, t: mnist.test.labels})
        print ('Step: %d, Loss: %f, Accuracy: %f' % (i, loss_val,
            acc_val))
```

```
Step: 100, Loss: 7747.077148, Accuracy: 0.848400
Step: 200, Loss: 5439.362305, Accuracy: 0.879900
Step: 300, Loss: 4556.467285, Accuracy: 0.890900
Step: 400, Loss: 4132.035156, Accuracy: 0.896100
Step: 500, Loss: 3836.139160, Accuracy: 0.902600
Step: 600, Loss: 3646.572510, Accuracy: 0.903900
...
Step: 1800, Loss: 2902.116699, Accuracy: 0.919000
Step: 1900, Loss: 2870.737061, Accuracy: 0.920000
Step: 2000, Loss: 2857.827881, Accuracy: 0.921100
```

미니 배치에 의한 파라미터 수정



1회째의 파라미터 수정　　2회째의 파라미터 수정

Mini batch

- mnist.train.next_batch(100)
- 데이터를 어디까지 추출했는지를 기억해 두고 호출할 때마다 다음 데이터를 추출
- Stochastic gradient descent

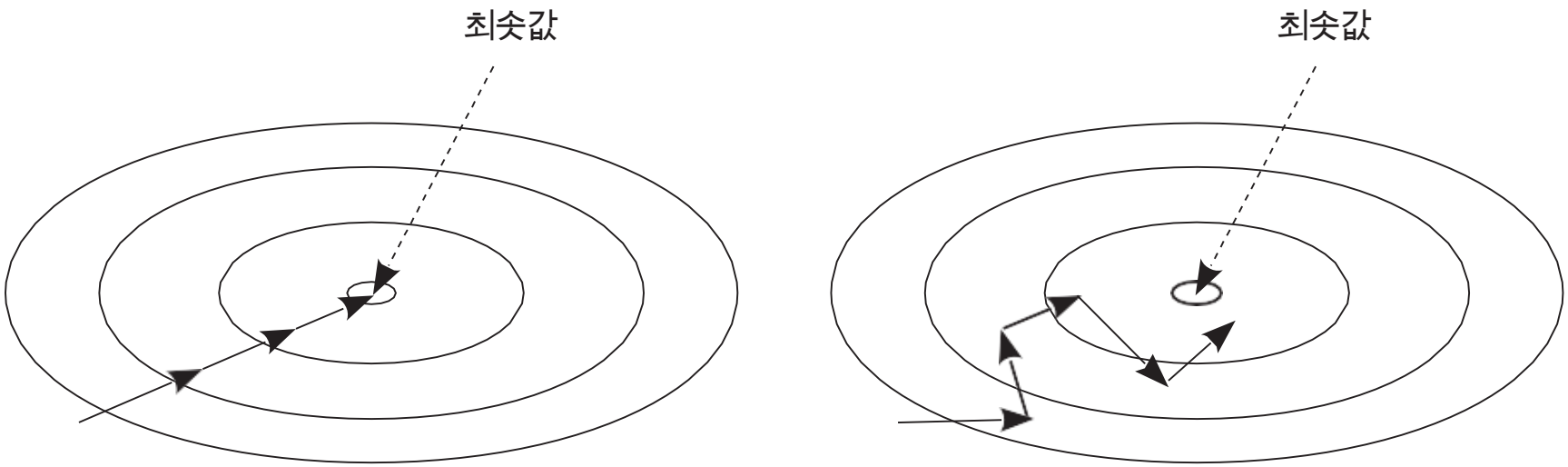
정답과 오답 예를 3개씩 출력

```
1.  images, labels = mnist.test.images, mnist.test.labels
2.  p_val = sess.run(p, feed_dict={x:images, t: labels})
3.
4.  fig = plt.figure(figsize=(8,15))
5.  for i in range(10):
6.      c = 1
7.      for (image, label, pred) in zip(images, labels, p_val):
8.          prediction, actual = np.argmax(pred), np.argmax(label)
9.          if prediction != i:
10.             continue
11.         if (c < 4 and i == actual) or (c >= 4 and i != actual):
12.             subplot = fig.add_subplot(10,6,i*6+c)
13.             subplot.set_xticks([])
14.             subplot.set_yticks([])
15.             subplot.set_title('%d / %d' % (prediction, actual))
16.             subplot.imshow(image.reshape((28,28)), vmin=0, vmax=1, cmap=plt.cm.gray_r,
17.                 interpolation="nearest")
18.         c += 1
19.         if c > 6:
20.             break
```

- Prediction/actual

2.3.4 Mini batch and stochastic gradient descent

- Stochastic gradient descent
  - 기울기 벡터가 정확하게 계산되지 않음
  - Avoid local minimum



모든 데이터를 사용한 경사 하강법 미니 배치에 의한 확률적 경사 하강법

그림 2-24 Stochastic gradient descent 방법으로 최솟값으로 향하는 모습

최솟값과 극솟값을 갖는 오차 함수의 예