

# Deep learning

## 3. Neural Networks을 이용한 분류



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

서울시립대학교 컴퓨터과학부 유하진

2019. 11

HJYU@UOS.AC.KR

# 3.1 2층 Neural Networks의 구조

## 3.1.1 2층 Neural Networks을 이용한 이항 분류기

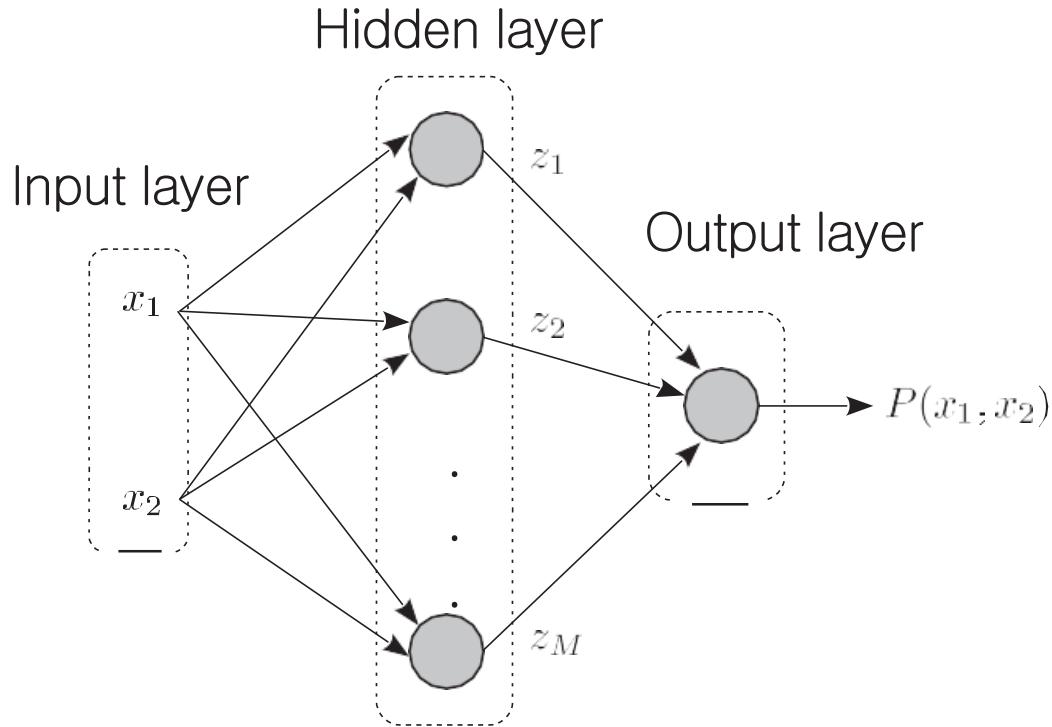
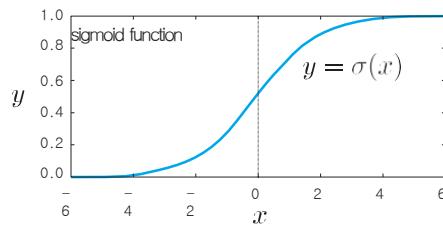


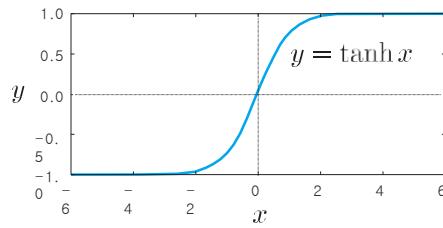
그림 3-2 2층 Neural Networks을 이용한 이항 분류기

# Activation functions

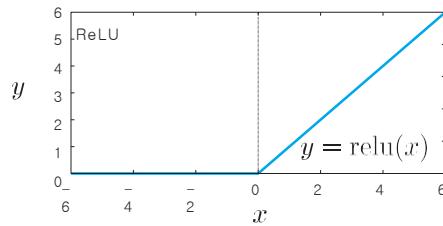
sigmoid function



Hyperbolic tangent



ReLU



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

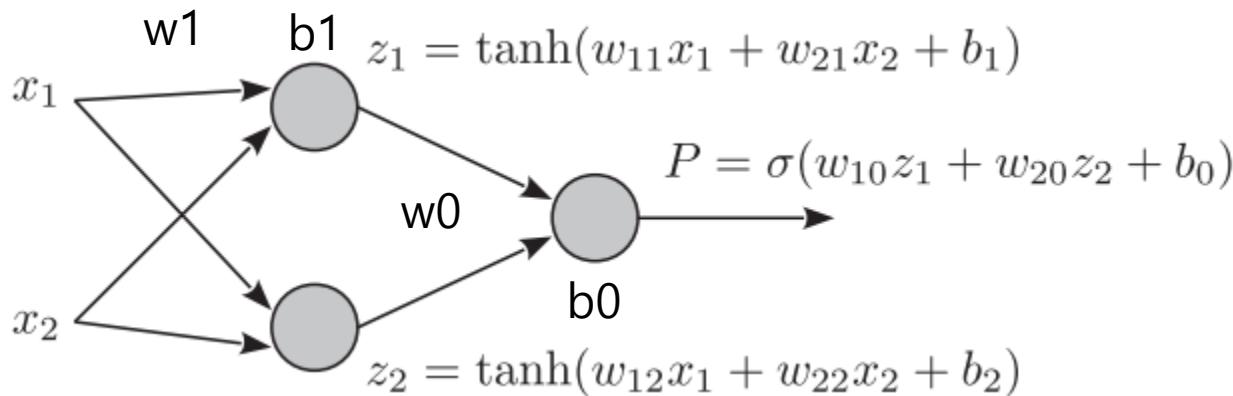
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\text{relu}(x) = \max(0, x)$$

## 3.1.2 Hidden layer의 역할

- Output of the hidden layer

$$\begin{cases} z_1 = \tanh(w_{11}x_1 + w_{21}x_2 + b_1) \\ z_2 = \tanh(w_{12}x_1 + w_{22}x_2 + b_2) \end{cases}$$



# 그림 3-5 은닉 계층의 출력값 변화

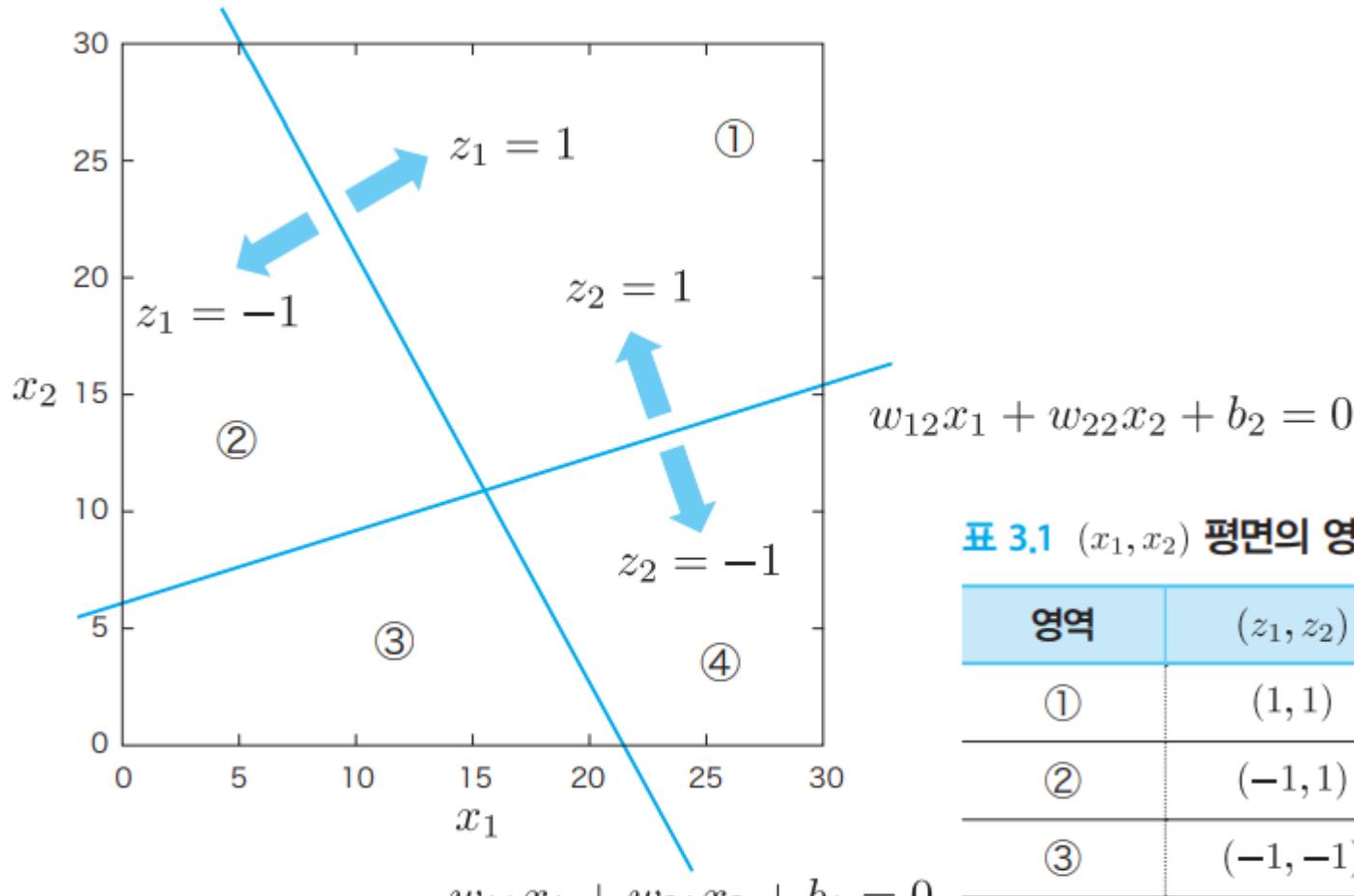


표 3.1  $(x_1, x_2)$  평면의 영역과  $(z_1, z_2)$  값의 대응 관계

영역	$(z_1, z_2)$
①	(1, 1)
②	(-1, 1)
③	(-1, -1)
④	(1, -1)

# Single layer network example.ipynb

## [SNE-01] 모듈을 임포트하고 난수의 시드를 설정

```
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
np.random.seed(20160614)
tf.set_random_seed(20160614)
```

# [SNE-02] Training set data generation

```
def generate_datablock(n, mu, var, t):
    data = multivariate_normal(mu, np.eye(2)*var, n)
    df = DataFrame(data, columns=['x1','x2'])
    df['t'] = t
    return df

# train data
df0 = generate_datablock(15, [7,7], 22, 0)
df1 = generate_datablock(15, [22,7], 22, 0)
df2 = generate_datablock(10, [7,22], 22, 0)
df3 = generate_datablock(25, [20,20], 22, 1)

df = pd.concat([df0, df1, df2, df3], ignore_index=True)
train_set = df.reindex(permutation(df.index)).reset_index(drop=True)
```

[SNE-03] (x1, x2)와 t를 각각 모은 것을 NumPy의 array 오브젝트로 추출해둔다.

```
train_x = train_set[['x1','x2']].as_matrix()
```

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

[SNE-04] 단층 Neural Networks을 이용한 이항 분류기 모델을 정의한다.

```
num_units = 2 # number of units in the hidden layer
```

```
mult = train_x.flatten().mean()
```

Optimization 고속화를 위해

```
x = tf.placeholder(tf.float32, [None, 2])
```

```
w1 = tf.Variable(tf.truncated_normal([2, num_units]))
```

```
b1 = tf.Variable(tf.zeros([num_units]))
```

```
hidden1 = tf.nn.tanh(tf.matmul(x, w1) + b1*mult)
```

Random number: 표준편차  
의 2배가 넘는 값은 제외

$$\mathbf{Z} = \tanh(\mathbf{X}\mathbf{W}_1 \oplus \mathbf{b}_1)$$

```
w0 = tf.Variable(tf.zeros([num_units, 1]))
```

```
b0 = tf.Variable(tf.zeros([1]))
```

```
p = tf.nn.sigmoid(tf.matmul(hidden1, w0) + b0*mult)
```

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

$$\mathbf{Z} = \tanh(\mathbf{X}\mathbf{W}_1 \oplus \mathbf{b}_1)$$

$\oplus$  : Broadcasting rule을 적용한 덧셈

$$\mathbf{Z} = \begin{pmatrix} z_{11} & z_{21} \\ z_{12} & z_{22} \\ z_{13} & z_{23} \\ \vdots & \vdots \end{pmatrix} \quad \mathbf{W}_1 = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \quad \mathbf{b}_1 = (b_1, b_2)$$

# [SNE-05] 오차 함수 loss, Training algorithm train\_step, 정답률 accuracy 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.GradientDescentOptimizer(0.001).minimize(loss)  
correct_prediction = tf.equal(tf.sign(p-0.5), tf.sign(t-0.5))  
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Learning rate

[SNE-06] Session을 준비하고 Variable을 초기화한다.

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

# Parameter optimization 1000회 반복

```
i = 0
```

```
for _ in range(10000):
```

```
    i += 1
```

```
    sess.run(train_step, feed_dict={x:train_x, t:train_t})
```

```
    if i % 100 == 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))
```

```
Step: 100, Loss: 44.921848, Accuracy: 0.430769  
Step: 200, Loss: 39.270321, Accuracy: 0.676923  
Step: 300, Loss: 51.999702, Accuracy: 0.584615  
Step: 400, Loss: 21.701561, Accuracy: 0.907692  
Step: 500, Loss: 12.708739, Accuracy: 0.953846  
Step: 600, Loss: 11.935550, Accuracy: 0.953846  
Step: 700, Loss: 11.454470, Accuracy: 0.953846  
Step: 800, Loss: 10.915851, Accuracy: 0.953846  
Step: 900, Loss: 10.570508, Accuracy: 0.953846  
Step: 1000, Loss: 11.822164, Accuracy: 0.953846
```

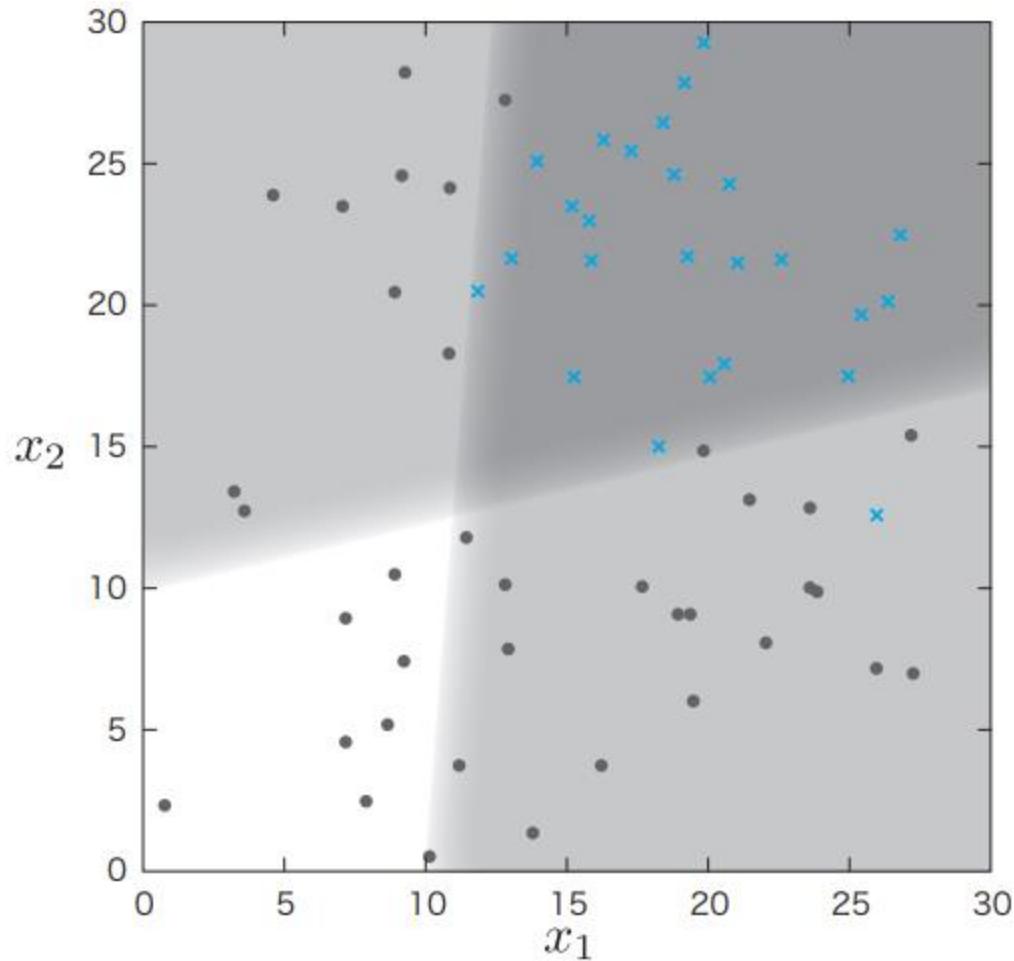
# [SNE-08] 얻어진 확률을 그림에 표시

```
1. train_set1 = train_set[train_set['t']==1]
2. train_set2 = train_set[train_set['t']==0]
3.
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_set2.x1, train_set2.x2, marker='o')
10.
11. locations = []
12. for x2 in np.linspace(0,30,100):
13.     for x1 in np.linspace(0,30,100):
14.         locations.append((x1,x2))
15. p_vals = sess.run(p, feed_dict={x:locations})
16. p_vals = p_vals.reshape((100,100))
17. subplot.imshow(p_vals, origin='lower', extent=(0,30,0,30),
18.                 cmap=plt.cm.gray_r, alpha=0.5)
```

평면을 100x100영역으로 분할.  
각 점을 location에 저장.

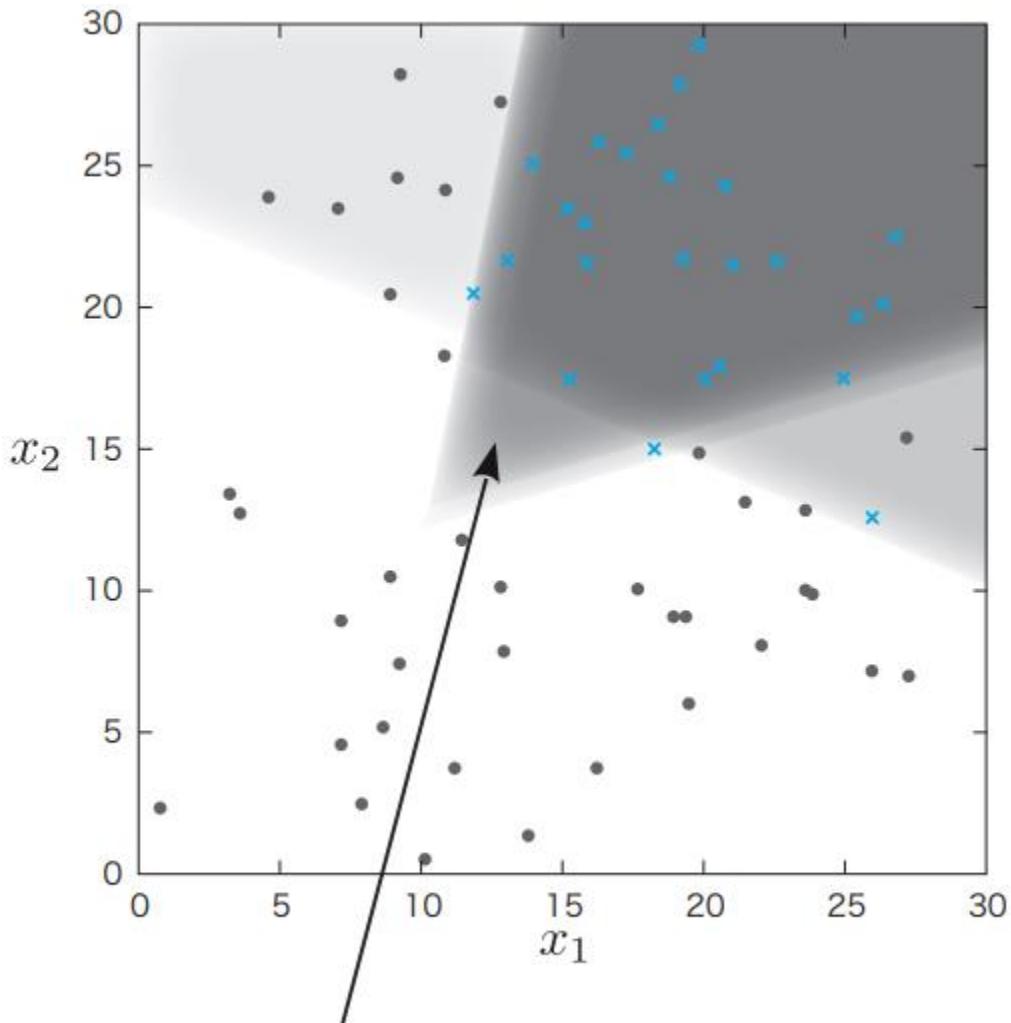
각 점에 대하여 확률 p 계산

# 학습된 확률



### 3.1.3 노드 개수와 활성화 함수 변경에 따른 효과

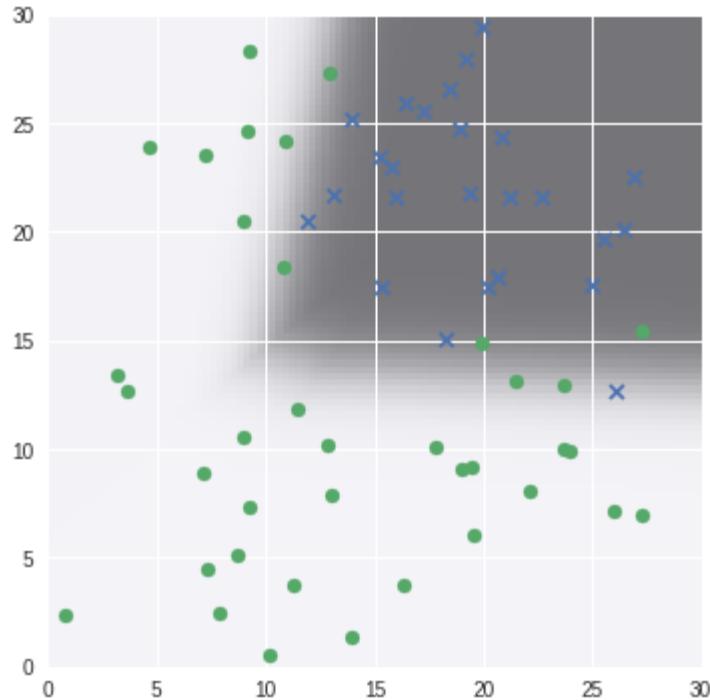
- num\_units = 4
- train\_step =  
`tf.train.GradientDescentOptimizer(0.0005).minimize(loss)`
- for \_ in range(4000):
- 오차함수의 형상이 복잡해  
졌을 것으로 생각하고 보다  
정교하게 학습



이 부분은  $t = 1$ 인 확률이 줄어들었다

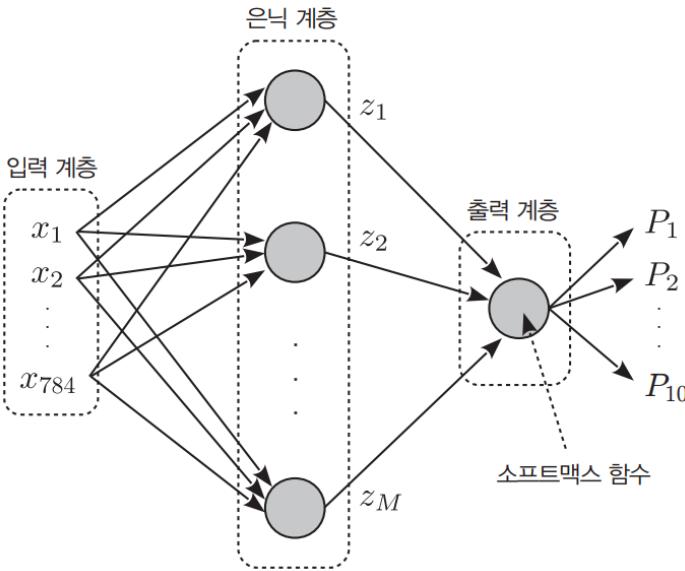
# 그림 3-8 Activation function으로 ReLU를 이용한 효과

- hidden1 = tf.nn.relu(tf.matmul(x,w1) + b1\*mult)



## 3.2 단층 Neural Networks을 이용한 필기 문자 분류

- 3.2.1 단층 Neural Networks을 이용한 다항 분류기



$$f_k(z_1, \dots, z_M) = w_{1k}^{(0)} z_1 + \dots + w_{Mk}^{(0)} z_M + b_k^{(0)} \quad (k = 1, \dots, 10)$$

$$P_k = \frac{e^{f_k}}{\sum_{k'=1}^{10} e^{f_{k'}}} \quad (k = 1, \dots, 10)$$

그림 3-9 MNIST data set을 분류하는 단층 Neural Networks

- [MSL-01] 필요한 모듈을 임포트하고 난수의 시드를 설정한다.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
np.random.seed(20160612)
tf.set_random_seed(20160612)
```

## [MSL-02] MNIST data set를 준비한다.

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

### [MSL-03] 단층 Neural Networks를 이용한 확률 p의 계산식을 준비

```
num_units = 1024
```

```
x = tf.placeholder(tf.float32, [None, 784])
```

```
w1 = tf.Variable(tf.truncated_normal([784, num_units]))
```

```
b1 = tf.Variable(tf.zeros([num_units]))
```

```
hidden1 = tf.nn.relu(tf.matmul(x, w1) + b1)
```

```
w0 = tf.Variable(tf.zeros([num_units, 10]))
```

```
b0 = tf.Variable(tf.zeros([10]))
```

```
p = tf.nn.softmax(tf.matmul(hidden1, w0) + b0)
```

### [MSL-04] 오차 함수 loss, Training 알고리즘 train\_step, 정답률 accuracy를 정의

```
t = tf.placeholder(tf.float32, [None, 10])
```

```
loss = -tf.reduce_sum(t * tf.log(p))
```

```
train_step = tf.train.AdamOptimizer().minimize(loss)
```

```
correct_prediction = tf.equal(tf.argmax(p, 1), tf.argmax(t, 1))
```

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

# Optimization

```
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))
```

# '0' ~ '9'의 숫자에 대해 정답과 오답 예를 3개씩 출력

```
images, labels = mnist.test.images, mnist.test.labels
p_val = sess.run(p, feed_dict={x:images, t: labels})

fig = plt.figure(figsize=(8,15))
for i in range(10):
    c = 1
    for (image, label, pred) in zip(images, labels, p_val):
        prediction, actual = np.argmax(pred), np.argmax(label)
        if prediction != i:
            continue
        if (c < 4 and i == actual) or (c >= 4 and i != actual):
            subplot = fig.add_subplot(10,6,i*6+c)
            subplot.set_xticks([])
            subplot.set_yticks([])
            subplot.set_title('%d / %d' % (prediction, actual))
            subplot.imshow(image.reshape((28,28)), vmin=0, vmax=1,
                           cmap=plt.cm.gray_r, interpolation="nearest")
        c += 1
    if c > 6:
        break
```

# 3.3 다층 Neural Networks으로의 확장

## 3.3.1 다층 Neural Networks의 효과

Linear / non-linear

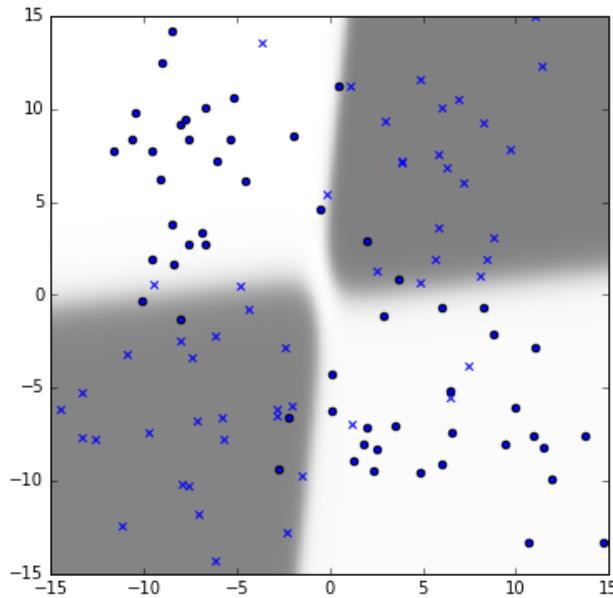


Fig 3-18

# 출력 계층을 확장한 Neural Networks

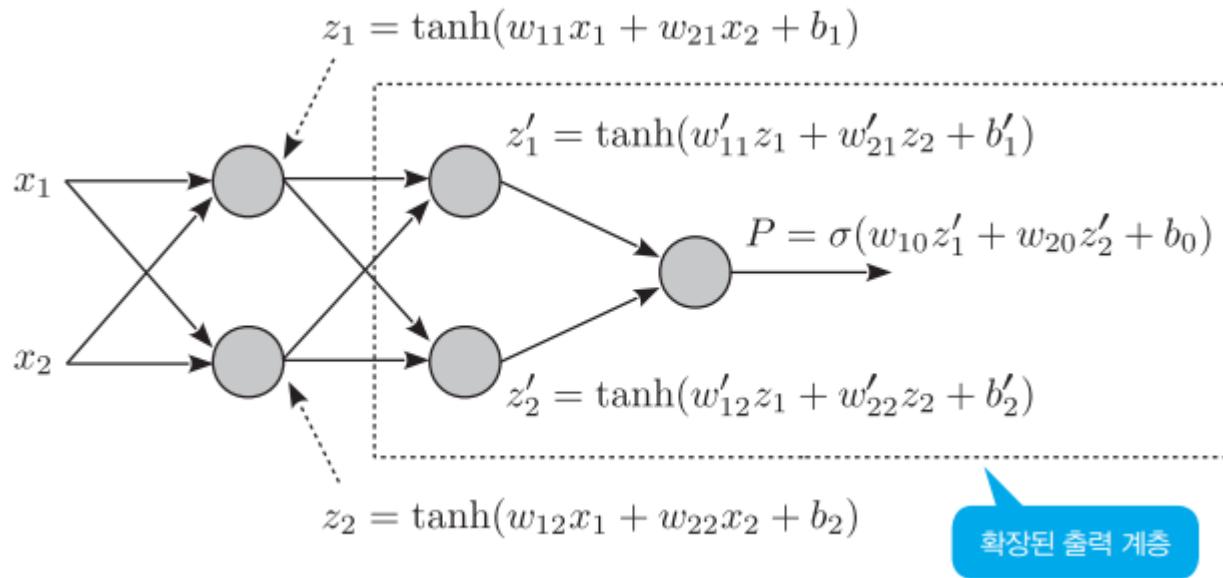


그림 3-17 출력 계층을 확장한 신경망

# [DNE-02] Training set data generation

```
def generate_datablock(n, mu, var, t):  
    data = multivariate_normal(mu, np.eye(2)*var, n)  
    df = DataFrame(data, columns=['x1','x2'])  
    df['t'] = t  
    return df
```

```
df0 = generate_datablock(30, [-7,-7], 18, 1)  
df1 = generate_datablock(30, [-7,7], 18, 0)          Data 전체의 평균 ≈ 0  
df2 = generate_datablock(30, [7,-7], 18, 0)  
df3 = generate_datablock(30, [7,7], 18, 1)  
df = pd.concat([df0, df1, df2, df3], ignore_index=True)  
train_set = df.reindex(permutation(df.index)).reset_index(drop=True)
```

## [DNE-04] 2계층 Neural Networks을 이용한 이항 분류기 모델을 정의

```
num_units1 = 2 # first hidden layer
```

```
num_units2 = 2 # second hidden layer
```

```
x = tf.placeholder(tf.float32, [None, 2])
```

```
w1 = tf.Variable(tf.truncated_normal([2, num_units1]))
```

```
b1 = tf.Variable(tf.zeros([num_units1]))
```

```
hidden1 = tf.nn.tanh(tf.matmul(x, w1) + b1)
```

```
w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
```

```
b2 = tf.Variable(tf.zeros([num_units2]))
```

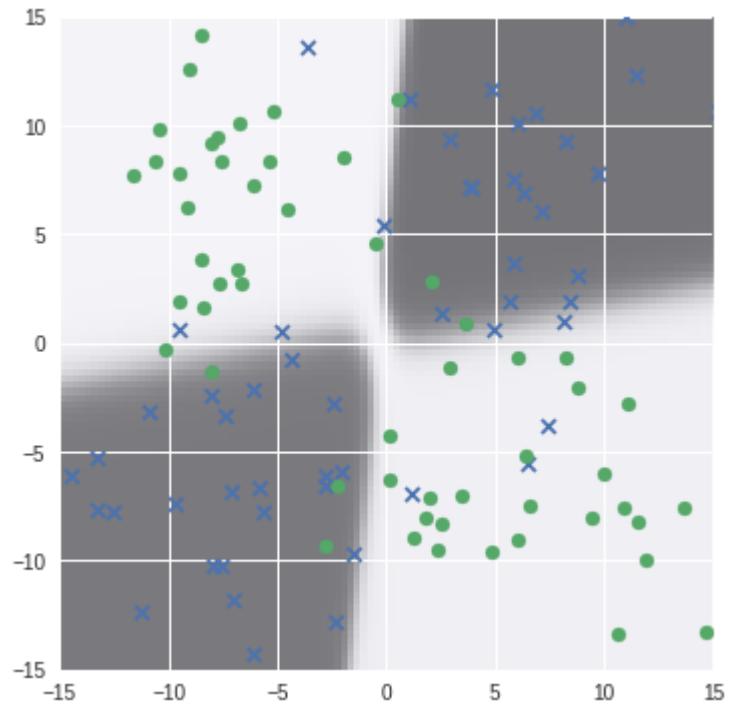
```
hidden2 = tf.nn.tanh(tf.matmul(hidden1, w2) + b2)
```

```
w0 = tf.Variable(tf.zeros([num_units2, 1]))
```

```
b0 = tf.Variable(tf.zeros([1]))
```

```
p = tf.nn.sigmoid(tf.matmul(hidden2, w0) + b0)
```

# Classification result



### 3.3.3 parameter가 극솟값으로 수렴하는 예

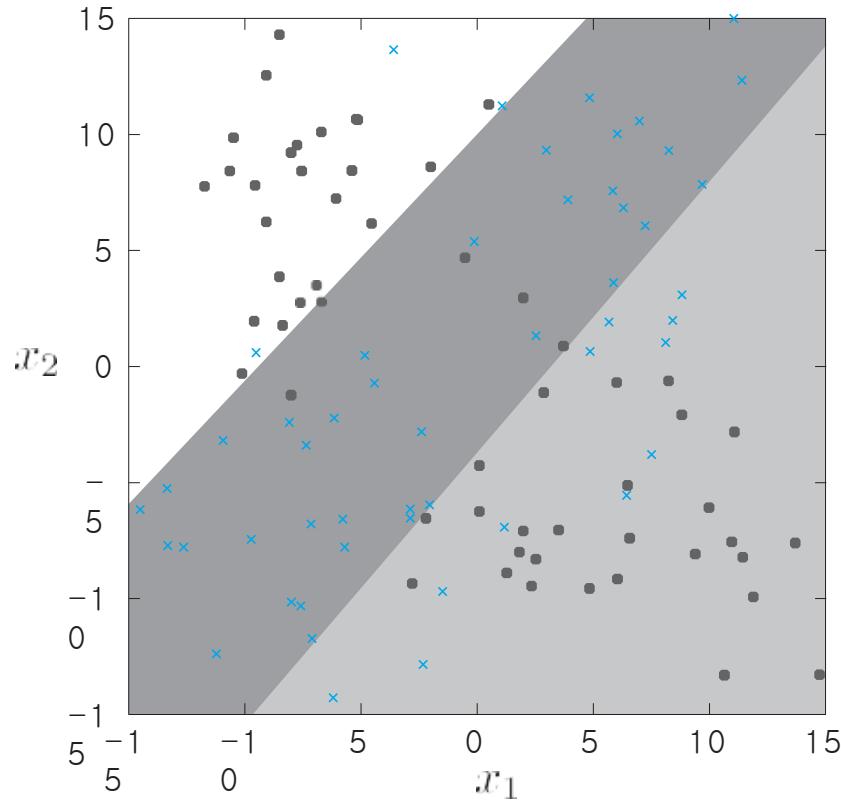


그림 3-24 오차 함수의 극솟값으로 parameter가 수렴한 예

# 그림 3-25 Loss가 갑자기 감소하는 예

