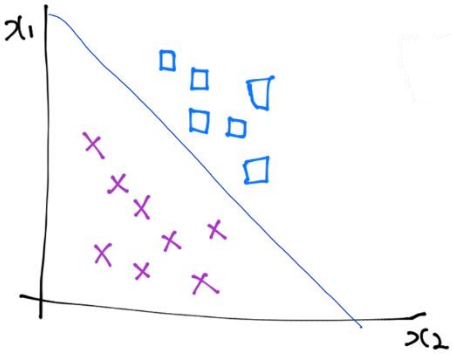
ML lec 06-1

✓ **Sigmoid**(Logistic Classification)의 요약

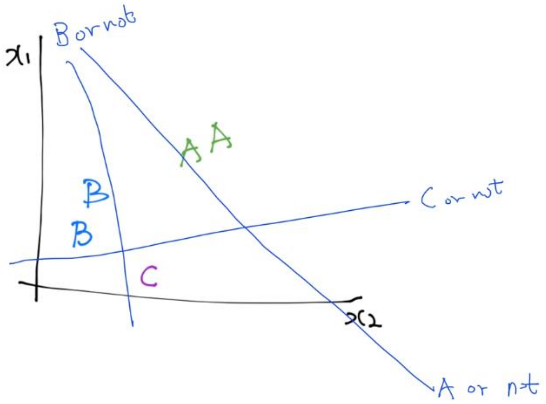
- 보통 Y : 실제값, $\hat{Y}$: 예측값으로 많이 씀
- 위 그림의 직관적 의미는

  어떤 두 가지를 구분하는 선을

  찾아낸다는 것



✓ **Multinomial Classification**



Multinomial은 Binary로 표현가능 ·

- Binary를 Matrix로 모으면 아래 그림처럼 간단히 표현가능하다

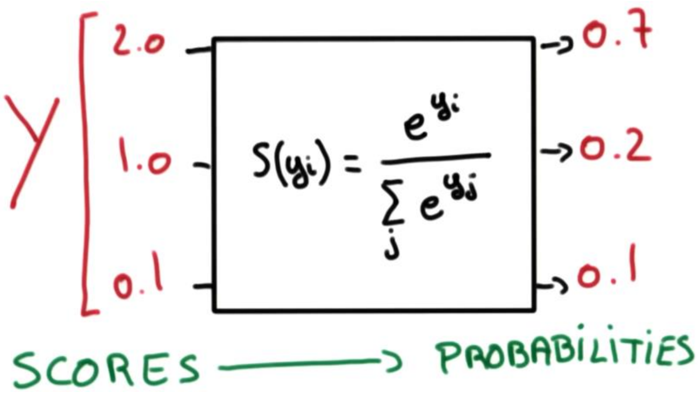
$$\begin{bmatrix} W_{A1} & W_{A2} & W_{A3} \\ W_{B1} & W_{B2} & W_{B3} \\ W_{C1} & W_{C2} & W_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 + W_{A3}x_3 \\ W_{B1}x_1 + W_{B2}x_2 + W_{B3}x_3 \\ W_{C1}x_1 + W_{C2}x_2 + W_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix} \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

ML lec 06-2

✓ **원하던 결과는 0~1의 값이다**

- 결과값을 모두 합했을 때 1이 되게 할 순 없을까?

✓ **Softmax**(hypothesis)**으로 가능하다**



one-hot encoding으로

우세한 값을 1로 표현가능

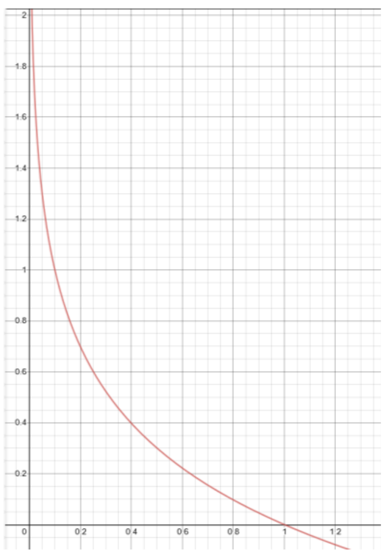
- 이제 필요한 것은 Cost Function

✓ **Cross Entropy가 Softmax의 Cost Function에 적합**

- Cross Entropy : $D(S, L) = -\sum_i L_i \log S_i \quad (S = \hat{Y}, L = Y)$
- $-\log x$그래프의 특성을 이용
- 여러 개의 Training Set도 고려하면

Cost Function : $\mathbf{L} = \frac{1}{N}\sum_i D(S(wx_i + b), L_i)$

($\mathbf{L}$ = Loss Function, i = Training Set)



ML lab 06-1

✓ Softmax의 구현법

```
hypothesis = tf.nn.softmax(tf.matmul(X, W)+b)
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

✓ arg_max를 사용하여 one-hot encoding하기

```
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9] … )
print(a, sess.run(tf.arg_max(a, 1)))
>>> [[ 1.…e-03    9.…e-01    9.…e-06]] [1]
```

- 끝의 [1]은 one-hot이 적용된 자리를 의미

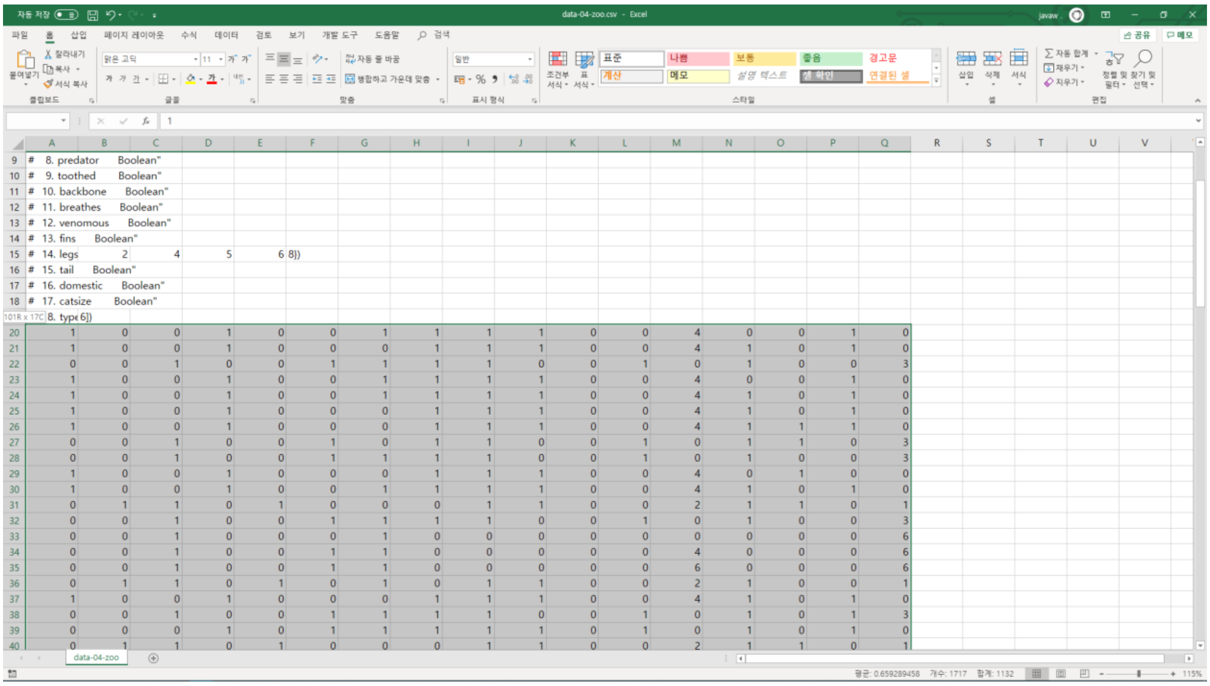
ML lab 06-2

✓ logits(WX+b) = scores

✓ Cross entropy를 조금 더 간단하게

```
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

cost_i = tf.nn.softmax_corss_entropy_with_logits(logits=logits,
                                                 labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
```

✓ 이것을 이용한 Animal Classification



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
9	# 8. predator	Boolean"															
10	# 9. toothed	Boolean"															
11	# 10. backbone	Boolean"															
12	# 11. breathes	Boolean"															
13	# 12. venomous	Boolean"															
14	# 13. fins	Boolean"															
15	# 14. legs	2	4	5	6	8])											
16	# 15. tail	Boolean"															
17	# 16. domestic	Boolean"															
18	# 17. catsize	Boolean"															
101R x 17C	8. type	6])															
20	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
21	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
22	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
23	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
24	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
25	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
26	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
27	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
28	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
29	1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
30	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
31	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
32	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
33	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
34	0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
35	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
36	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
37	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
38	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	3
39	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1	0
40	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1

- 총 101R × 17C, 마지막 Q열은 y data
- 그런데 tf.one_hot을 사용하게 되면, **차원**이 한 차원 **더 높아지게 된다**.

✓ 그래서 필요한 것이 tf.reshape

```
Y = tf.placeholder(tf.int32, [None, 1])
Y_one_hot = tf.one_hot(Y, nb_classes)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
```

- tf.reshape을 사용하면 차원을 조절할 수 있다
- 다음은 위의 내용을 종합한 Animal Classification의 구현

```
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1] # 전체 행에서 Q열을 제외
y_data = xy[:, [-1]] # 전체 행에서 Q열만 포함

nb_classes = 7 # 다리는 0~6개라서

X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1])
Y_one_hot = tf.one_hot(Y, nb_classes)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])

W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                 labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

with tf.Session() as sess:
    sess.run(tf.global_variable_initializer())

    for step in range(2000):
        sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            loss, acc = sess.run([cost, accuracy], feed_dict={
                                 X: x_data, Y: y_data})
            print("Step: {:>5}¦tLoss: {:.3f}¦tAcc: {:.2%}".format(
                  step, loss, acc))

pred = sess.run(prediction, feed_dict={X: x_data})
for p, y in zip(pred, y_data.flatten()): # flatten: [[1], [0]] -> [1, 0]
    print("[{}] Prediction: {} True Y: {}".
          format(p == int(y), p, int(y)))
```

✓ 결과 : 제대로 작동한다

IPython console

Console 1/A

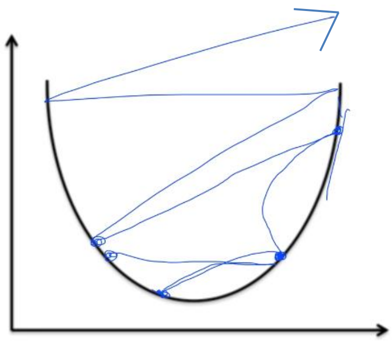
```
In [1]: runfile('D:/Documents/ML/동물 분석/Animal Learning.py', wdir='D:/Documents/ML/동물
분석')
WARNING:tensorflow:From D:/Documents/ML/동물 분석/Animal Learning.py:25:
softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and
will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See tf.nn.softmax_cross_entropy_with_logits_v2.

Step:     0     Loss: 3.387     Acc: 18.81%
Step:   100     Loss: 0.576     Acc: 85.15%
Step:   200     Loss: 0.370     Acc: 90.10%
Step:   300     Loss: 0.284     Acc: 94.06%
Step:   400     Loss: 0.233     Acc: 95.05%
Step:   500     Loss: 0.197     Acc: 96.04%
Step:   600     Loss: 0.170     Acc: 97.03%
Step:   700     Loss: 0.148     Acc: 97.03%
Step:   800     Loss: 0.131     Acc: 98.02%
Step:   900     Loss: 0.117     Acc: 100.00%
Step:  1000     Loss: 0.106     Acc: 100.00%
Step:  1100     Loss: 0.096     Acc: 100.00%
Step:  1200     Loss: 0.088     Acc: 100.00%
Step:  1300     Loss: 0.081     Acc: 100.00%
Step:  1400     Loss: 0.076     Acc: 100.00%
Step:  1500     Loss: 0.071     Acc: 100.00%
Step:  1600     Loss: 0.066     Acc: 100.00%
Step:  1700     Loss: 0.063     Acc: 100.00%
Step:  1800     Loss: 0.059     Acc: 100.00%
Step:  1900     Loss: 0.056     Acc: 100.00%
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
[True] Prediction: 3 True Y: 3
[True] Prediction: 0 True Y: 0
[True] Prediction: 0 True Y: 0
[True] Prediction: 1 True Y: 1
[True] Prediction: 3 True Y: 3
[True] Prediction: 6 True Y: 6
[True] Prediction: 6 True Y: 6
[True] Prediction: 6 True Y: 6
[True] Prediction: 1 True Y: 1
[True] Prediction: 0 True Y: 0
[True] Prediction: 3 True Y: 3
```

IPython console | History log | Python console

Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8-GUESSED | Line: 1 | Column: 1 | Memory: 81 %

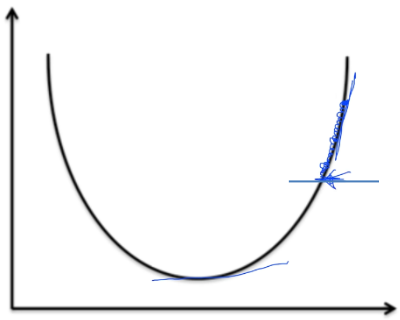
ML lec 07-1

✓ learning_rate이 크다면?



- **Overshooting이 일어남**

✓ learning_rate이 작다면?



- **오래걸리고, local minimum이 생김**

그러므로 조금씩 변경하며 찾아야 함

✓ **Gradient Descent**를 preprocessing(선처리)하는 방법

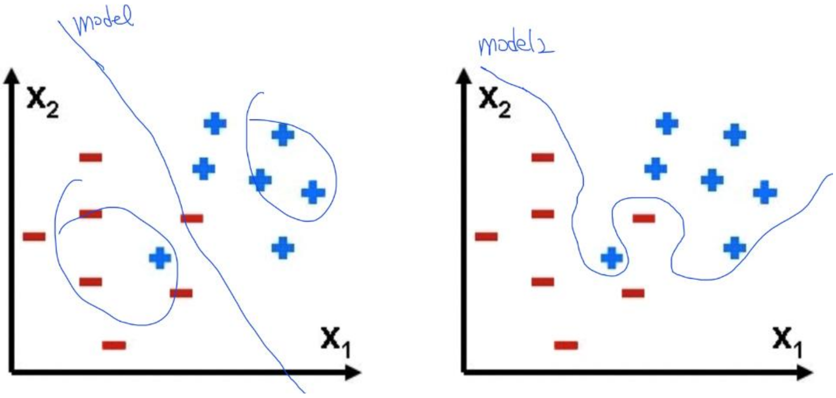
Standardization

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$

```
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
```

✓ **Overfitting?**

- 어떤 모델에 지나치게 최적화되었을 경우 생김(e.g. model2)



- 해결방법
  - Training Data 늘리기
  - 중복된 features 줄이기
  - Regularization

✓ **Regularization**(일반화)**?**

- 구부리지 말고, **좀 펴자**
- 구부린다는 것은 weight이 큰 값을 가진다는 의미
- Cost Function 뒤에 $\lambda \sum w^2$를 추가하면 된다

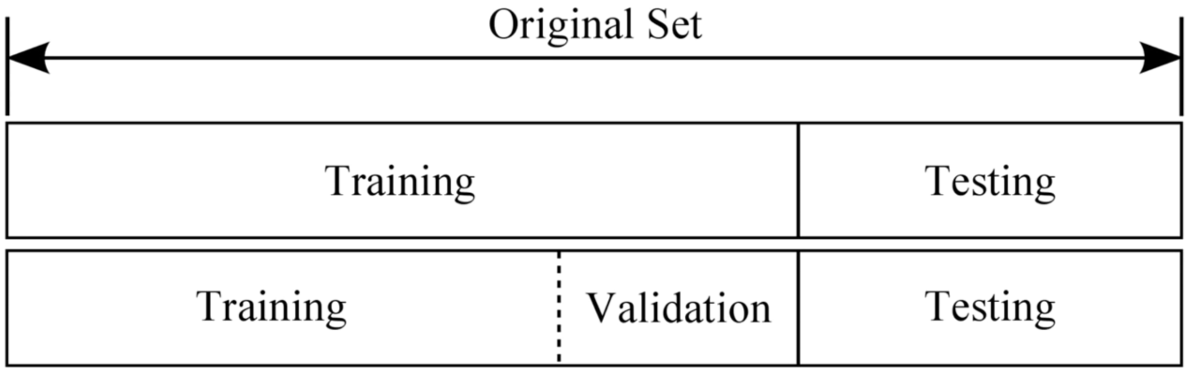
$$\mathbf{L} = \frac{1}{N}\sum_i D(S(wx_i + b), L_i) + \lambda \sum w^2$$

✓ **구현하는 법**

```
l2reg = 0.001 * tf.reduce_sum(tf.square(W))
#λ로 강도 결정가능. 0: 신경안씀, 1: 엄청중요, 0.001: 적당히
```

ML lec 07-2

✓ 정확한 평가를 위해 시험평가 같은 Test set이 필요



✓ 대표적인 예로 **MNIST Dataset**이 있음

ML lab 07-1

✓ Training and Test datasets으로 나누기

```
x_data = [[1, 2, 1], [1, 3, 2], [1, 3, 4], …, [1, 7, 7]]
y_data = [[0, 0, 1], [0, 0, 1], [0, 0, 1], …, [1, 0, 0]]

x_test = [[2, 1, 1], [3, 1, 2], [3, 3, 4]]
y_test = [[0, 0, 1], [0, 0, 1], [0, 0, 1]]
```

✓ test 결과 확인하는 법

```
is_correct = tf.equal(prediction, tf.arg_max(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
…
print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

✓ learning_rate을 크게 할 경우

```
…
optimizer = tf.train.GradientDescentOptimizer(learning_rate=10.0)
                                              .minimize(cost)
…
```

```
>>> 0 68.1359 [[ … … …]
[ … … …]
[ … … …]]
1 inf [[ nan nan nan]
[ nan nan nan]
[ nan nan nan]]
…
Prediction: [0 0 0]
Accuracy: 0.0
```

- 임의의 횟수부터 계산을 포기하게 된다

✓ learning_rate을 작게 할 경우

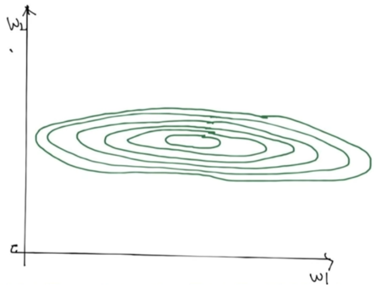
```
…
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-10)
                                              .minimize(cost)
…
```

```
>>> 0 6.94284 [[ … … …]
[ … … …]
[ … … …]]
1 6.94284 [[ … … …]
[ … … …]
[ … … …]]
…
200 6.94284 [[ … … …]
[ … … …]
[ … … …]]
Prediction: [0 0 0]
Accuracy: 0.0
```

- 진도가 안나간다

✓ 또는 data값이 불규칙할 경우

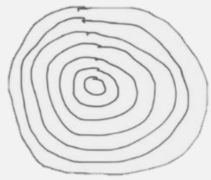
```
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```



- 이때도 마찬가지로 NaN이 나타나게 된다

✓ 이 경우엔 MinMaxScaler()로 해결 가능

```
xy = np.array([[828.659973, 908100, …],
              …
              [809.51001, …]])

xy = MinMaxScaler(xy)
```



ML lab 07-2

✓ Meet MNIST Dataset

Editor - D:₩Documents₩ML₩MNIST₩Number Recognition.py

Number Recognition.py

```python
# ML lab 07-2
import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777) # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data

# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))

# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
num_epochs = 15
batch_size = 100
num_iterations = int(mnist.train.num_examples / batch_size)

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(num_epochs):
        avg_cost = 0

        for i in range(num_iterations):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            _, cost_val = sess.run([train, cost], feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += cost_val / num_iterations

        print("Epoch: {:04d}, Cost: {:.9f}".format(epoch + 1, avg_cost))

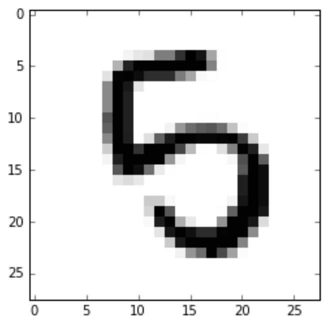
    print("Learning finished")

    # Test the model using test sets
    print(
        "Accuracy: ",
        accuracy.eval(
            session=sess, feed_dict={X: mnist.test.images, Y: mnist.test.labels}
        ),
    )

    # Get one and predict
    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label: ", sess.run(tf.argmax(mnist.test.labels[r : r + 1], 1)))
    print(
        "Prediction: ",
        sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r : r + 1]}),
    )

    plt.imshow(
        mnist.test.images[r : r + 1].reshape(28, 28),
        cmap="Greys",
        interpolation="nearest",
    )
    plt.show()
```

```
In [1]: runfile('D:/Documents/ML/MNIST/Number Recognition.py',
wdir='D:/Documents/ML/MNIST')
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Epoch: 0001, Cost: 2.826302660
Epoch: 0002, Cost: 1.061668948
Epoch: 0003, Cost: 0.838061307
Epoch: 0004, Cost: 0.733232732
Epoch: 0005, Cost: 0.669279880
Epoch: 0006, Cost: 0.624611828
Epoch: 0007, Cost: 0.591160339
Epoch: 0008, Cost: 0.563868978
Epoch: 0009, Cost: 0.541745167
Epoch: 0010, Cost: 0.522673571
Epoch: 0011, Cost: 0.506782322
Epoch: 0012, Cost: 0.492447640
Epoch: 0013, Cost: 0.479955830
Epoch: 0014, Cost: 0.468893666
Epoch: 0015, Cost: 0.458703479
Learning finished
Accuracy:  0.8951
Label:   [5]
Prediction:   [6]
```

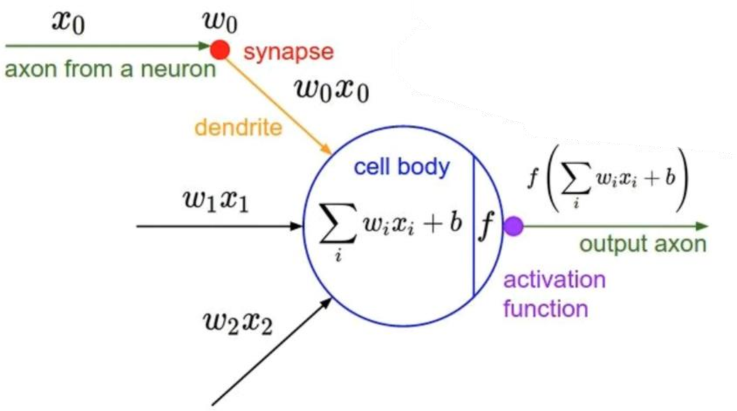


- 아직은 분발이 필요해보인다

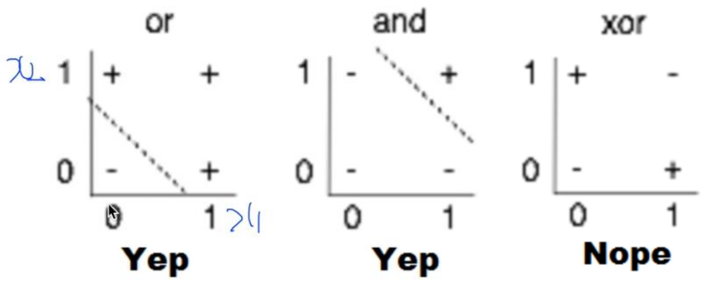
ML lec 08-1, 2

✓ **시초 : tninking machine을 향한 갈망**

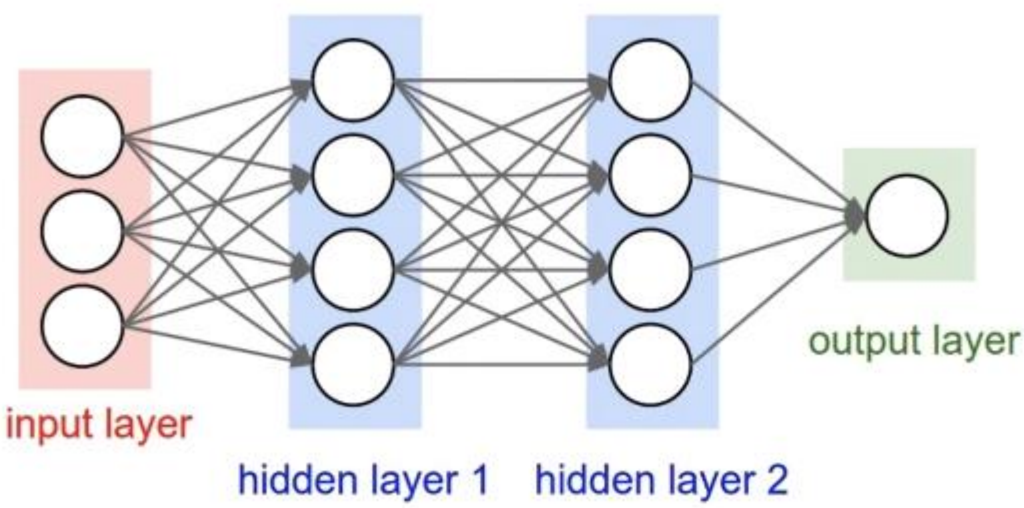
- 뇌의 Neuron을 연구하여 기반을 잡음



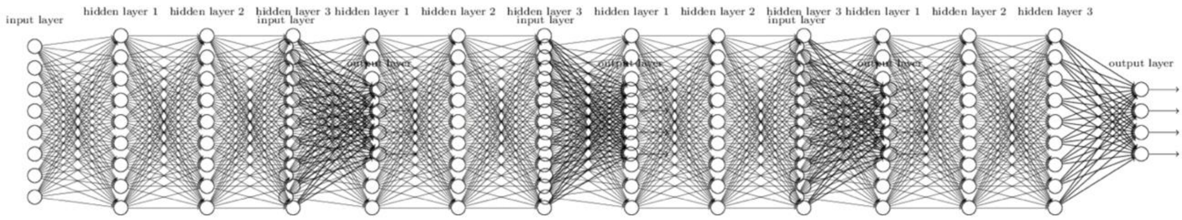
✓ 하지만 or, and와는 달리 xor은 구현이 안됨



- 불가능하다는 현실직시



- ✓ **Backpropagation**의 탄생
- ✓ **Neural Networks**의 탄생
  - 하지만 너무 복잡해서 다시 침체기에 들어감



- ✓ 연구가 진행되며 Deep Learning이란 개념으로 탈바꿈
- ✓ 활발하게 연구가 이루어져 오차율을 급격하게 낮추게 됨
- ✓ Facebook, Google, Netflix 등 이미 많이 사용되고 있음
- ✓ 왜 배워야 하는가?
  - 범용성이 넓고, 정확하며 정보가 많아 배우기 간편하기 때문

ML lab 08

✓ 기초 이론 수업

- a.ndim : a의 rank를 반환
- rank : 바깥 대괄호의 개수로 판별 가능
- shpe :
  - (a, b, …, n)일 때 n은 가장 안쪽 대괄호에 있는 수의 개수
  - n-1은 그 다음 대괄호쌍의 개수
  - 반복
- axis : 가장 바깥 ~ 가장 안쪽 : 0 ~ n
- Matmul VS Multiply
  - 일반적인 차원끼리의 곱셈은 matmul
  - Multiply를 할 경우 Broadcasting이 일어남
  - Broadcasting을 이용하여 rank가 달라도 곱셈 가능
- reduce_mean(float, axis)
- argmax(a, axis)
- reshape(a, shape=[-1, n])
  - squeeze([[0], [1], [2]]) -> [0, 1, 2]
  - expand_dims([0, 1], 1) -> [[0],

```
                                   [1]] # 차원 추가
```

- one_hot([…], depth) 이후에 reshape해야함
- cast([...], format)
- stack([a, b, c], axis) : 쌓기 -> [[a],

```
                                   [b],
                                   [c]]
```

- ones and zeros like
  - ones_like(a) : 모두 1로 바꿈
  - zeros_like(a) : 모두 0으로 바꿈
- ```
  for x, y in zip([1, 2, 3], [4, 5, 6]) :
      print(x, y)
  # for 루프를 통해 복수 개의 텐서를 한 방에 출력
  ```

  -> 1 4

  2 5

  3 6
  ```