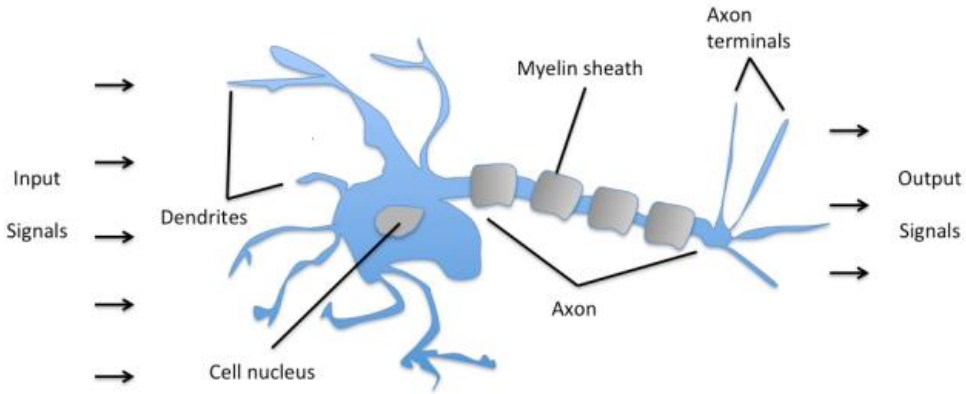
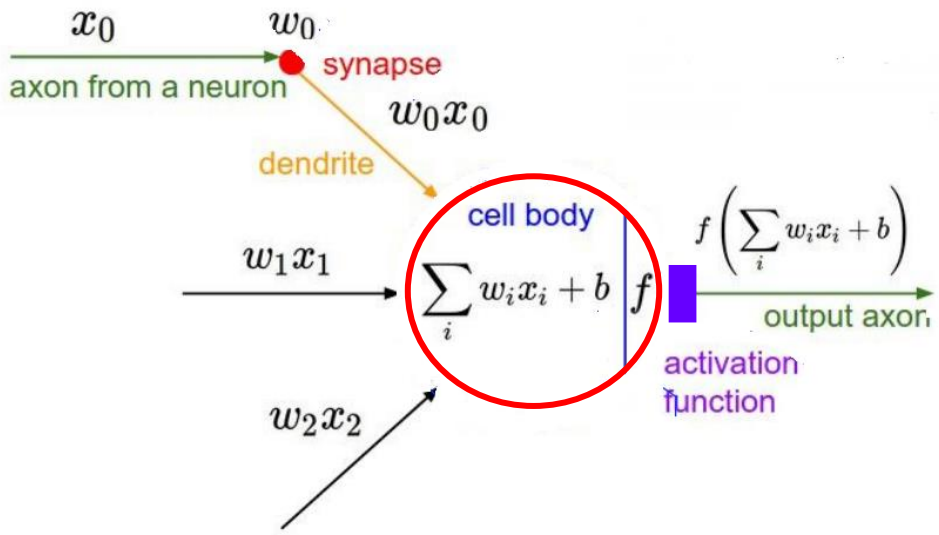
[TensorFlow로 신경망 구현]

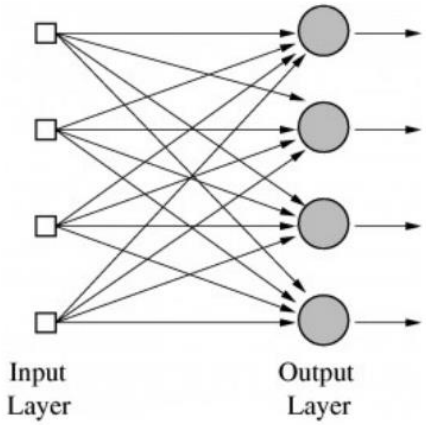
사람의 신경망



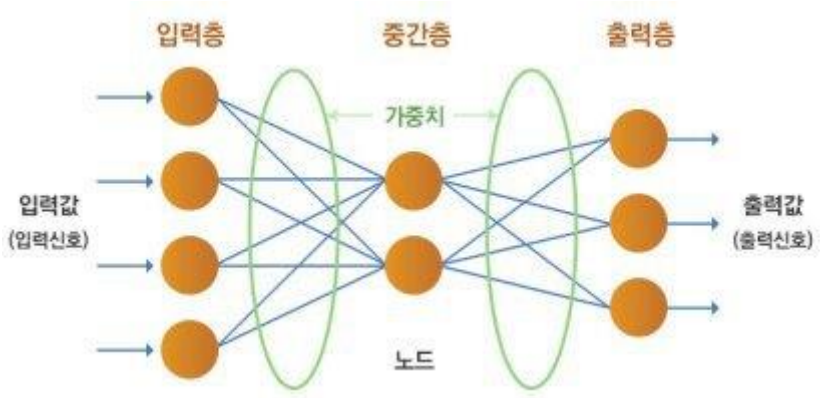
Schematic of a biological neuron.

Neural Network (인공 신경망)





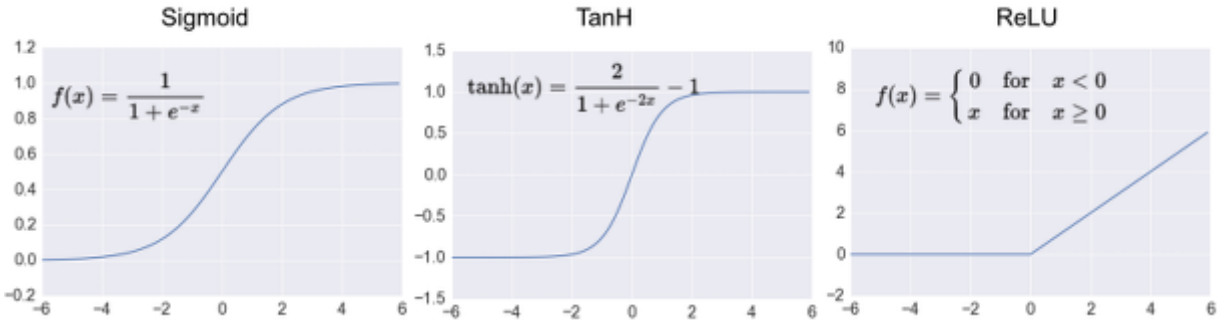
신경망의 구성



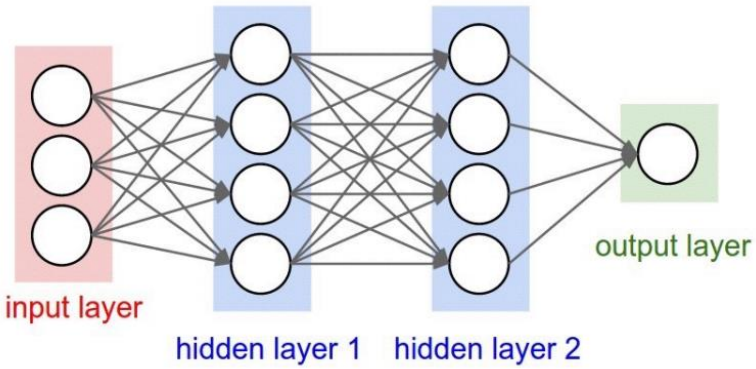
입력값(X)에 가중치(W)를 곱하고 편향(b)를 더한 뒤 활성함수 (Sigmode, ReLU 등)를 거쳐 결과값 y를 만들어내는 것이 인공 신경망의 기본입니다. 원하는 y값을 만들어내기 위해 W와 b 값을 변경해가면서 적절한 값을 찾아내는 최적화 과정을 학습 또는 훈련이라고 합니다.

Y = Sigmoid(WX + b)

뉴런은 입력이 누적되어 어떤 수준으로 커진 경우에만 출력을 합니다. 즉 입력 값이 어떤 **분계점(threshold)**에 도달해야 출력이 발생합니다. 따라서 뉴런은 미세한 잡음 신호 따위는 전달하지 않고 의미가 있는 신호만 전달하게 된다는 점에서도 직관적으로 이해할 수 있다. 이처럼 입력 신호를 받아 특정 분계점을 넘어서는 경우에 출력 신호를 생성해 주는 함수를 활성화 함수(activation function)라고 합니다. 활성화 함수의 종류는 매우 다양한데 그 중 Sigmoid (Logistic 함수라고도 불림) 는 단순하면서도 전통적으로 많이 쓰여 인공 신경망을 처음 공부할 때는 최고다. 현업에서 많이 쓰는 활성화 함수는 **ReLU**함수다.



뉴런은 한 개의 입력이 아닌, 여러 개의 입력을 받습니다. 값들을 모두 더해서 분계점을 넘으면 작동하게 되는데, 다른 입력 값이 작아도 하나만 커도 작동하게 됩니다. 이러한 생물학적 뉴런을 인공적으로 모델화하려면 뉴런을 여러 계층에 걸쳐 위치시키고, 각각의 뉴런은 직전 계층과 직후 계층에 있는 모든 뉴런들과 상호 연결되어 있는 식으로 표현하면 됩니다. 신경망 층이 둘 이상으로 구성된 딥 러닝이라고 합니다.각각의 인공 뉴런을 노드라고 부릅니다. 그리고 학습을 한다는 것은 연결의 강도를 조절해 나간다는 것입니다. 이 연결을 신경망에서는 가중치라고 부릅니다.



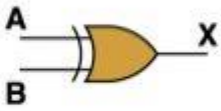
Marvin Minsky, 1969

XOR 문제

Boolean Expression

$X = A \oplus B$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

[이진 분류 문제]

1) 기본신경망으로 구현

(ANN - Artificial Neural Network) - **신경망이 하나만 있는 것**

예)

ts_11_ANN_XOR_BiClassification.py XOR 문제, ANN 로 풀었을 때 문제

```
import tensorflow as tf
import numpy as np

tf.set_random_seed(777)  # for reproducibility
learning_rate = 0.1

x_data = [[0, 0],
          [0, 1],
          [1, 0],
          [1, 1]]
y_data = [[0],
          [1],
          [1],
          [0]]

x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
```

0~1 사이의 숫자로 반환

활성함수에 따라 cost function 이 달라지므로 적합한 cost function 사용해야 합니다.

ts_11_ANN_XOR_BiClassification.py (계속)

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))

train = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={
                  X: x_data, Y: y_data}), sess.run(W))

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy],
                       feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```

정확도 계산

2진 분류라서 True(1) 또는 False(0).으로 답이 나와야 함.
에측값이 0.5보다 크면 Trure,
그렇지 않으면 False로 예측

(출력 결과)

..중략

9900 0.6931472 [[-1.3286063e-07]
 [-1.3267396e-07]]
10000 0.6931472 [[-1.3286063e-07]
 [-1.3267396e-07]]

Hypothesis: [[0.5]
 [0.5]
 [0.5]
 [0.5]]
Correct: [[0.]
 [0.]
 [0.]
 [0.]]
Accuracy: 0.5

이 예제의 데이터 사례는 단순한 ANN 로는 아무리 많이 학습을 하여도 정확도가 매우 낮다.

여러 층의 딥러닝(DNN)으로 개선시킬 수 있다.

[TensorFlow로 Deep learning(DNN) 구현]

[이진 분류 문제]

2) 딥러닝 (DNN) 으로 구현

기존 코드(ts_11_ANN_XOR_BiClassification.py) 에 한 레이어를 더 붙이면 됩니다.

```
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid:   tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
```

출력 개수와 입력 개수, Bais 동일하게 맞추어 주어야 함.

ts_11_ANN_XOR_BiClassification.py 코드의 위 부분을 다음과 같이 변경합니다.

첫 입력 값의 수

ts_11_DNN_XOR_BiClassification.py

```
W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
```

(출력 결과)

...중략

Hypothesis: [[0.01449734]
 [0.97972405]
 [0.9797362]
 [0.01941727]]

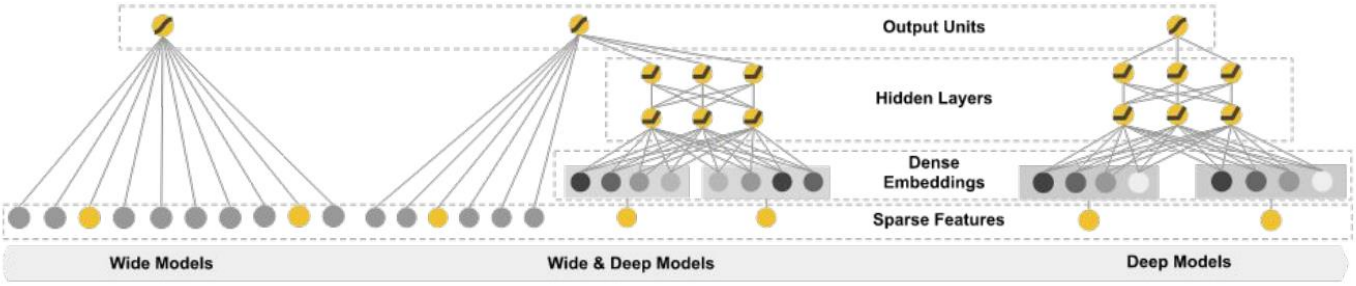
Correct: [[0.]
 [1.]
 [1.]
 [0.]]

Accuracy: 1.0

더 넓게 학습시킬 수 있습니다. (Wide learning)

```
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)
```



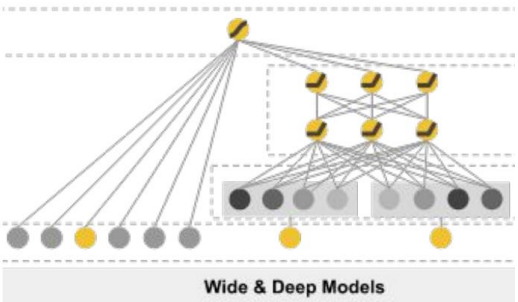
더 깊게 학습시킬 수 있습니다. (Deep learning)

```
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2'
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3'
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')
hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)
```



Wide & Deep 할 수로 더 잘 학습됩니다.

[활성 함수: sigmoid 보다는 ReLu]

XOR 실습 예제의 hidden layer를 9개를 더 늘려봅니다.

```
W1 = tf.Variable(tf.random_normal([2, 5],-1.0,1.0), name='weight1')
W2 = tf.Variable(tf.random_normal([5, 5],-1.0,1.0), name='weight2')
W3 = tf.Variable(tf.random_normal([5, 5],-1.0,1.0), name='weight3')
W4 = tf.Variable(tf.random_normal([5, 5],-1.0,1.0), name='weight4')
W5 = tf.Variable(tf.random_normal([5, 5],-1.0,1.0), name='weight5')
W6 = tf.Variable(tf.random_normal([5, 1],-1.0,1.0), name='weight6')

b1 = tf.Variable(tf.random_normal([5]), name='bias1')
b2 = tf.Variable(tf.random_normal([5]), name='bias2')
b3 = tf.Variable(tf.random_normal([5]), name='bias3')
b4 = tf.Variable(tf.random_normal([5]), name='bias4')
b5 = tf.Variable(tf.random_normal([5]), name='bias5')
b6 = tf.Variable(tf.random_normal([5]), name='bias6')

L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)
L5 = tf.sigmoid(tf.matmul(L4, W5) + b5)
L6 = tf.sigmoid(tf.matmul(L5, W6) + b6)

hypothesis = tf.sigmoid(tf.matmul(L5, W6) + b6)
```

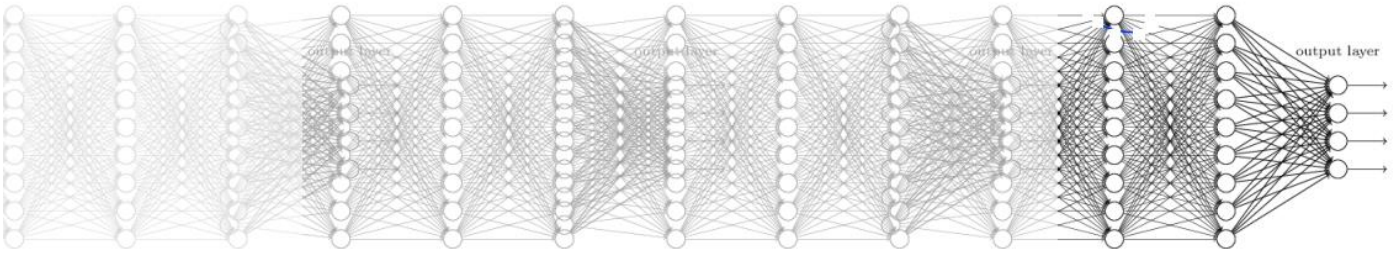
(출력 결과)

```
196000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
198000 [0.69314718, array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32)]
[array([[ 0.49999988],
       [ 0.50000006],
       [ 0.49999982],
       [ 0.5       ]], dtype=float32), array([[ 0.],
       [ 1.],
       [ 0.],
       [ 1.]], dtype=float32)]
Accuracy: 0.5
```

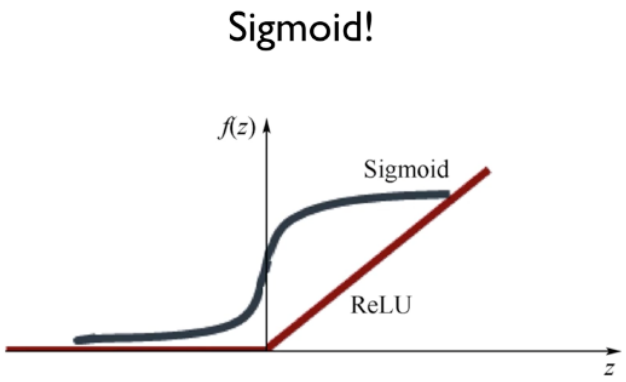
정확도가 많이 실망스럽습니다. 그 이유는?

Sigmoid 함수의 결과는 0~1 사이의 값이고, 그 값이 NN layer지나가면서 사라지는 현상 때문(Vanishing gradient)

(예) Sigmoid(Z) 결과가 0.01 이면 0.01*0.01*0.02 ...)



그래서, 딥러닝에서는 Sigmoid 함수보다 Relu 함수(0~무한대)를 대신 사용합니다.



XOR 실습 예제의 hidden layer 활성 함수를 Relu 함수로 바꾸어 봅니다.

```
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L5 = tf.nn.relu(tf.matmul(L4, W5) + b5)
hypothesis = tf.sigmoid(tf.matmul(L5, W6) + b6)
# 마지막 단은 0~1사이의 값이어야 해서(이진 분류 문제이므로) #Sigmoid 사용
```

[다중 분류 문제]

1) 기본신경망으로 구현

(ANN - Artificial Neural Network) - **신경망이 하나만 있는 것**

예)

ts_12_ANN_MultiClassification.py 다중 분류 모델 구현

```
# 털과 날개가 있는지 없는지에 따라, 포유류인지 조류인지 분류하는 신경망 모델을 만들어봅니다
import tensorflow as tf
import numpy as np

# [털, 날개]
x_data = np.array(
    [[0, 0], [1, 0], [1, 1], [0, 0], [0, 0], [0, 1]])

# [기타, 포유류, 조류]
# 다음과 같은 형식을 one-hot 형식의 데이터라고 합니다.
y_data = np.array([
    [1, 0, 0],  # 기타
    [0, 1, 0],  # 포유류
    [0, 0, 1],  # 조류
    [1, 0, 0],
    [1, 0, 0],
    [0, 0, 1]
])
#########
# 신경망 모델 구성
######
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

# 신경망은 2차원으로 [입력층(특성), 출력층(레이블)] -> [2, 3] 으로 정합니다.
W = tf.Variable(tf.random_uniform([2, 3], -1., 1.))

# 편향을 각각 각 레이어의 아웃풋 갯수로 설정합니다.
# 편향은 아웃풋의 갯수, 즉 최종 결과값의 분류 갯수인 3으로 설정합니다.
b = tf.Variable(tf.zeros([3]))

# 신경망에 가중치 W과 편향 b을 적용합니다
L = tf.add(tf.matmul(X, W), b)
# 가중치와 편향을 이용해 계산한 결과 값에
# 텐서플로우에서 기본적으로 제공하는 활성화 함수인 ReLU 함수를 적용합니다.
L = tf.nn.relu(L)

# 마지막으로 softmax 함수를 이용하여 출력값을 사용하기 쉽게 만듭니다
# softmax 함수는 다음처럼 결과값을 전체합이 1인 확률로 만들어주는 함수입니다.
# 예) [8.04, 2.76, -6.52] -> [0.53 0.24 0.23]
model = tf.nn.softmax(L)
```

ts_12_ANN_MultiClassification.py (계속)

```
# 신경망을 최적화하기 위한 비용 함수를 작성합니다.
# 각 개별 결과에 대한 합을 구한 뒤 평균을 내는 방식을 사용합니다.
# 전체 합이 아닌, 개별 결과를 구한 뒤 평균을 내는 방식을 사용하기 위해 axis 옵션을 사용합니다
# axis 옵션이 없으면 -1.09 처럼 총합인 스칼라값으로 출력됩니다.
#        Y         model         Y * tf.log(model)   reduce_sum(axis=1)
# 예) [[1 0 0]  [[0.1 0.7 0.2]  -> [[-1.0  0    0]  -> [-1.0, -0.09]
#     [0 1 0]]  [0.2 0.8 0.0]]     [ 0   -0.09 0]]
# 즉, 이것은 예측값과 실제값 사이의 확률 분포의 차이를 비용으로 계산한 것이며,
# 이것을 Cross-Entropy 라고 합니다.
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(model), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train_op = optimizer.minimize(cost)


#########
# 신경망 모델 학습
######
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for step in range(100):
    sess.run(train_op, feed_dict={X: x_data, Y: y_data})

    if (step + 1) % 10 == 0:
        print(step + 1, sess.run(cost, feed_dict={X: x_data, Y: y_data}))


#########
# 결과 확인
# 0: 기타 1: 포유류, 2: 조류
######
# tf.argmax: 예측값과 실제값의 행렬에서 tf.argmax 를 이용해 가장 큰 값을 가져옵니다.
# 예) [[0 1 0] [1 0 0]] -> [1 0]
#    [[0.2 0.7 0.1] [0.9 0.1 0.]] -> [1 0]
prediction = tf.argmax(model, 1)
target = tf.argmax(Y, 1)
print('예측값:', sess.run(prediction, feed_dict={X: x_data}))
print('실제값:', sess.run(target, feed_dict={Y: y_data}))

is_correct = tf.equal(prediction, target)
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도: %.2f' % sess.run(accuracy * 100, feed_dict={X: x_data, Y: y_data}))
```

(출력 결과) ----------

10 1.3573036

20 1.3478512

30 1.3385199

```
40 1.3293072
50 1.3202103
60 1.3112272
70 1.3023556
80 1.2935935
90 1.2849394
100 1.2763915
예측값: [0 1 1 0 0 1]
실제값: [0 1 2 0 0 2]
정확도: 66.67----------
```

정확도가 매우 실망스럽습니다. 그 이유는 신경망이 한 층 밖에 안되기 때문입니다. 층을 하나만 더 늘리면 쉽게 해결이 됩니다. 다음 예제에서 신경망 층이 둘이상으로 구성된 딥 러닝을 구현해 보겠습니다.

[딥러닝 구현]

[다중 분류 문제]

2) 딥러닝 (DNN) 으로 구현

기존 코드에 한 레이어를 더 붙이면 됩니다.

ts_12_DNN_MultiClassification.py 신경망 층이 둘이상으로 구성된 딥 러닝을 구현 .

```
# 털과 날개가 있는지 없는지에 따라, 포유류인지 조류인지 분류하는 신경망 모델을 만들어봅니다.
# 신경망의 레이어를 여러개로 구성하여 말로만 듣던 딥러닝을 구성해 봅시다!
import tensorflow as tf
import numpy as np

# [털, 날개]
x_data = np.array(
    [[0, 0], [1, 0], [1, 1], [0, 0], [0, 0], [0, 1]])

# [기타, 포유류, 조류]
y_data = np.array([
    [1, 0, 0],  # 기타
    [0, 1, 0],  # 포유류
    [0, 0, 1],  # 조류
    [1, 0, 0],
    [1, 0, 0],
    [0, 0, 1]
])

#########
# 신경망 모델 구성
######
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

# 첫번째 가중치의 차원은 [특성, 히든 레이어의 뉴런갯수] -> [2, 10] 으로 정합니다.
W1 = tf.Variable(tf.random_uniform([2, 10], -1., 1.))
# 두번째 가중치의 차원을 [첫번째 히든 레이어의 뉴런 갯수, 분류 갯수] -> [10, 3] 으로 정합니다.
W2 = tf.Variable(tf.random_uniform([10, 3], -1., 1.))

# 편향을 각각 각 레이어의 아웃풋 갯수로 설정합니다.
# b1 은 히든 레이어의 뉴런 갯수로, b2 는 최종 결과값 즉, 분류 갯수인 3으로 설정합니다.
b1 = tf.Variable(tf.zeros([10]))
b2 = tf.Variable(tf.zeros([3]))

# 신경망의 히든 레이어에 가중치 W1과 편향 b1을 적용합니다
L1 = tf.add(tf.matmul(X, W1), b1)
L1 = tf.nn.relu(L1)
```

```
# 최종적인 아웃풋을 계산합니다.
# 히든레이어에 두번째 가중치 W2와 편향 b2를 적용하여 3개의 출력값을 만들어냅니다.
model = tf.add(tf.matmul(L1, W2), b2)
model = tf.nn.softmax(model)
# 텐서플로우에서 기본적으로 제공되는 크로스 엔트로피 함수를 이용해
# 복잡한 수식을 사용하지 않고도 최적화를 위한 비용 함수를 다음처럼 간단하게 적용할 수 있습니다.
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(model), axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train_op = optimizer.minimize(cost)
```

ts_12_DNN_MultiClassification.py (계속)

```
##########
# 신경망 모델 학습
######
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for step in range(100):
    sess.run(train_op, feed_dict={X: x_data, Y: y_data})

    if (step + 1) % 10 == 0:
        print(step + 1, sess.run(cost, feed_dict={X: x_data, Y: y_data}))


##########
# 결과 확인
# 0: 기타 1: 포유류, 2: 조류
######
prediction = tf.argmax(model, 1)
target = tf.argmax(Y, 1)
print('예측값:', sess.run(prediction, feed_dict={X: x_data}))
print('실제값:', sess.run(target, feed_dict={Y: y_data}))

is_correct = tf.equal(prediction, target)
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도: %.2f' % sess.run(accuracy * 100, feed_dict={X: x_data, Y: y_data}))
```

(출력결과) ----------

10 0.76973885

20 0.5981427

30 0.47975895

40 0.3927695

50 0.323996

60 0.27004

70 0.22719681

80 0.19288574

90 0.16592874

100 0.14441738

예측값: [0 1 2 0 0 2]

실제값: [0 1 2 0 0 2]

정확도: 100.00----------

****

* 어떤 활성함수를 써야 할까?

Hidden layer 의 활성함수는 Relu Output layer 의 활성함수는 이진 분류의 경우 Sigmoid, 다중 분류는 :Softmax

* 어떤 Cost function을 써야 할까?

1) 활성함수 Sigmoid 일때 Cost function

```
# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))
```

2) 활성함수 Softmax 일때 Cost function

```
model = tf.nn.softmax(L)
```
#(L 은 Hidden layer)

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(model), axis=1))
```

또는 위의 코드 대신 아래와 같이 쓸 수 있다. (아래 코드에서 L1 은 Hidden layer)

```
model = tf.add(tf.matmul(L1, W2), b2)

cost = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits_v2(labels=Y, logits=model))
```

[다중 분류 문제]

MNIST 예)

딥러닝 (DNN) 으로 구현

MNIST 손글씨 데이터 셋으로 신경망 학습 예제 연습해보겠습니다.
손으로 쓴 숫자 이미지를 모아 놓은 데이터셋으로 0~9 까지의 숫자를 28 * 28 픽셀 크기의 이미지로 구성해 놓은 것입니다.
http://yann.lecun.com/exdb/mnist/

ts_13_DNN_MNIST.py

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data
# 텐서플로우에 기본 내장된 mnist 모듈을 이용하여 데이터를 로드합니다.
# 지정한 폴더에 MNIST 데이터가 없는 경우 자동으로 데이터를 다운로드합니다.
# one_hot 옵션은 레이블을 동물 분류 예제에서 보았던 one_hot 방식의 데이터로 만들어줍니다.
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
#########
# 신경망 모델 구성
######
# 입력 값의 차원은 [배치크기, 특성값] 으로 되어 있습니다.
# 손글씨 이미지는 28x28 픽셀로 이루어져 있고, 이를 784개의 특성값으로 정합니다.
X = tf.placeholder(tf.float32, [None, 784])
# 결과는 0~9 의 10 가지 분류를 가집니다.
Y = tf.placeholder(tf.float32, [None, 10])

# 신경망의 레이어는 다음처럼 구성합니다.
# 784(입력 특성값)
#   -> 256 (히든레이어 뉴런 갯수) -> 256 (히든레이어 뉴런 갯수)
#   -> 10 (결과값 0~9 분류)
W1 = tf.Variable(tf.random_normal([784, 256], stddev=0.01))
# 입력값에 가중치를 곱하고 ReLU 함수를 이용하여 레이어를 만듭니다.
L1 = tf.nn.relu(tf.matmul(X, W1))

W2 = tf.Variable(tf.random_normal([256, 256], stddev=0.01))
# L1 레이어의 출력값에 가중치를 곱하고 ReLU 함수를 이용하여 레이어를 만듭니다.
L2 = tf.nn.relu(tf.matmul(L1, W2))

W3 = tf.Variable(tf.random_normal([256, 10], stddev=0.01))
# 최종 모델의 출력값은 W3 변수를 곱해 10개의 분류를 가지게 됩니다.
model = tf.matmul(L2, W3)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

```
#########
# 신경망 모델 학습
######
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)
```

ts_13_DNN_MNIST.py (계속)

```
for epoch in range(15):
    total_cost = 0

    for i in range(total_batch):
        # 텐서플로우의 mnist 모델의 next_batch 함수를 이용해
        # 지정한 크기만큼 학습할 데이터를 가져옵니다.
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)

        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs, Y: batch_ys})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.3f}'.format(total_cost / total_batch))

print('최적화 완료!')

#########
# 결과 확인
######
# model 로 예측한 값과 실제 레이블인 Y의 값을 비교합니다.
# tf.argmax 함수를 이용해 예측한 값에서 가장 큰 값을 예측한 레이블이라고 평가합니다.
# 예) [0.1 0 0 0.7 0 0.2 0 0 0 0] -> 3
is_correct = tf.equal(tf.argmax(model, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도:', sess.run(accuracy,
                        feed_dict={X: mnist.test.images,
                                   Y: mnist.test.labels}))
```

(출력 결과)----------

```
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 Avg. cost = 0.403
Epoch: 0002 Avg. cost = 0.154
Epoch: 0003 Avg. cost = 0.098
Epoch: 0004 Avg. cost = 0.068
Epoch: 0005 Avg. cost = 0.051
```

```
Epoch: 0006 Avg. cost = 0.038
... 중략
Epoch: 0012 Avg. cost = 0.014
Epoch: 0013 Avg. cost = 0.012
Epoch: 0014 Avg. cost = 0.011
Epoch: 0015 Avg. cost = 0.010
최적화 완료!
정확도: 0.9828
```