LAB04 심층 신경망 구현하기

- 신경망의 층을 더 추가할 경우, 성능이 향상될 수 있다.

In [2]:

```
import numpy as np
import tensorflow as tf
```

```
C:₩Anaconda3₩lib₩site-packages₩h5py₩__init__.py:36: FutureWarning: Conversion of the
second argument of issubdtype from `float` to `np.floating` is deprecated. In futur
e, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

4-1 신경망의 구성

In [3]:

```
# [특징1, 특징2]
# 특징1, 특징2의 값 이용한 Target의 예측
x_data = np.array( [[0, 0], [1, 0], [1, 1], [0, 0], [0, 0], [0, 1]])

# 원핫인코딩(one-hot encoding)
# 데이터가 가질 수 있는 값들을 일렬로 나열한 배열로 만들고,
# 그중의 표현하려는 값을 인덱스의 원소만 1로 표현하고, 나머지는 0으로 채우는 표기법

# [setosa, versicolor, virginica]
# 다음과 같은 형식을 one-hot 형식의 데이터라고 합니다.
y_data = np.array([
    [1, 0, 0],  # setosa
    [0, 1, 0],  # versicolor
    [0, 0, 1],  # virginica
    [1, 0, 0],
    [1, 0, 0],
    [0, 0, 1]
])
```

In [4]:

```
X = tf.placeholder(tf.float32)   #  X에 들어갈 값(공간)
Y = tf.placeholder(tf.float32)   #  Y에 들어갈 값(공간)
```

신경망의 구성

- 2개의 입력층(특징1, 특징2)
- 10개 뉴런의 은닉층
- 3개 출력층(뉴런 꽃의 종류)

In [5]:

```
# W1 : [2,10] -> [특징, 은닉층의 뉴런 수]
# W2 : [10,3] -> [은닉층의 뉴런 수, 분류 수]
W1 = tf.Variable(tf.random_uniform([2,10],-1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([10,3],-1.0, 1.0))

# 편향
# b1 = [10] -> 은닉층의 뉴런 수
# b2 = [3]  -> 분류 수
b1 = tf.Variable(tf.zeros([10]))
b2 = tf.Variable(tf.zeros([3]))
```

```
WARNING:tensorflow:From C:₩Anaconda3₩lib₩site-packages₩tensorflow₩python₩framework₩o
p_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprec
ated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [6]:

```
# 첫 번째 가중치와 편향, 활성화 함수를 적용
L1 = tf.add(tf.matmul(X, W1), b1)
L1 = tf.nn.relu(L1)
```

In [7]:

```
# 두 번째 가중치와 편향,
model = tf.add(tf.matmul(L1, W2), b2)
```

4-2 Loss 함수 및 최적화 알고리즘

- 손실함수(Loss) : cross_entropy
- 최적화(Optimizer) : AdamOptimizer

In [11]:

```
# 교차 엔트로피 함수 사용
# 최적화 함수 : AdamOptimizer 함수
# AdamOptimizer는 GradientDescentOptimizer보다 보편적으로 성능이 더 좋다.
cost = tf.reduce_mean(
   tf.nn.softmax_cross_entropy_with_logits_v2(labels=Y, logits=model)) # 버전 변경

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

4-3 그래프 실행

In [9]:

```
# 텐서플로 세션 초기화
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

# 레이블 데이터를 이용하여 학습을 진행
for step in range(100):
  sess.run(train_op,feed_dict={X:x_data, Y:y_data})

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

```
10 1.0391954
20 0.87078977
30 0.71131045
40 0.559679
50 0.42707607
60 0.3198483
70 0.23544247
80 0.17264307
90 0.12684555
100 0.0946249
```

In [13]:

```
#########
# 결과 확인
######
# tf.argmax: 예측값과 실제값의 행렬에서
# tf.argmax 를 이용해 가장 큰 값의 위치(인덱스)을 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}))
```

```
예측값: [0 1 2 0 0 2]
실제값: [0 1 2 0 0 2]
정확도: 100.00
```

Summary

- 결과적으로 은닉층의 추가가 정확도의 향상을 가져왔다.

REF

- Tf가 제공하는 다양한 최적화 함수 :
  - https://www.tensorflow.org/api_guides/python/train (https://www.tensorflow.org/api_guides/python/train)

In [ ]: