2. MLP

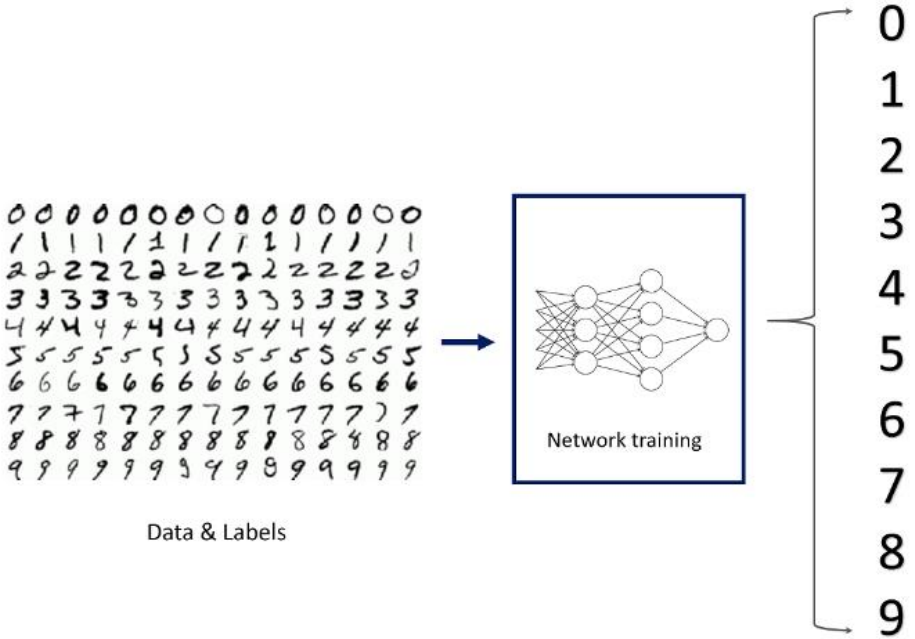
Multi-layer perceptrons

AILAB

Hanyang Univ.

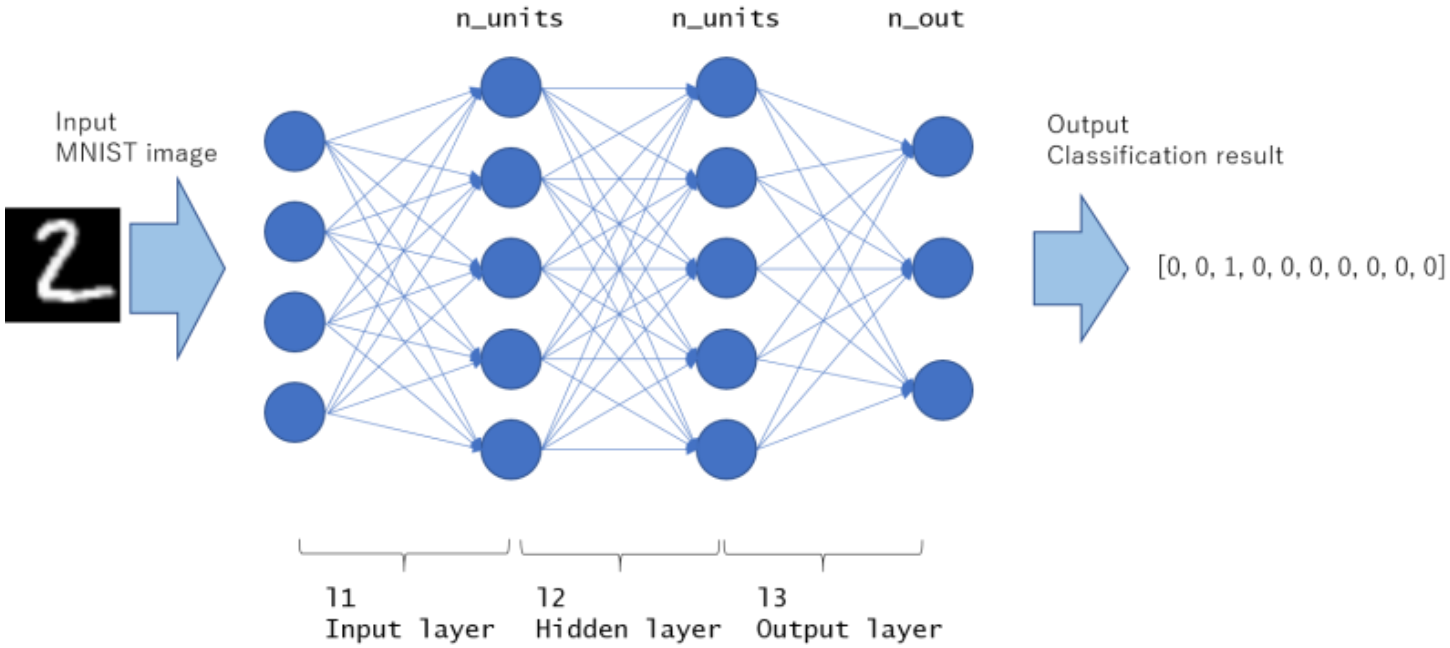
오늘 실습 내용

- **Linear regression**
- **Single Layer Perceptron**
- **Multi Layer Perceptron to solve XOR**



MNIST Dataset and Number Classification [1]

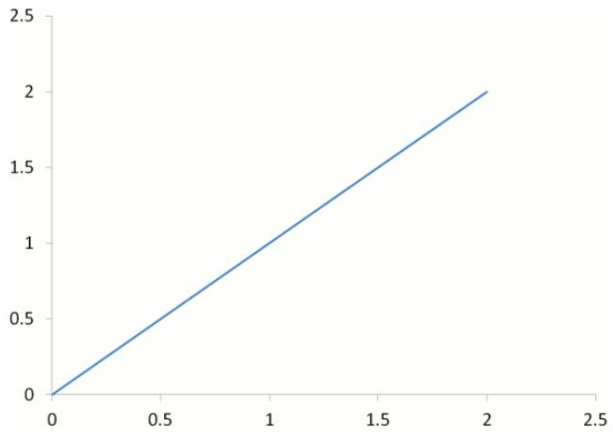
신경망모델의 구성


n_units
n_units
n_out
Input
MNIST image
Output
Classification result
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
l1
Input layer
l2
Hidden layer
l3
Output layer

신경망모델 학습 프로세스

- 데이터 processing
- model 디자인
  - layer 종류, 개수 및 뉴런 개수 설정
  - 각 layer 마다의 activation function 설정
- Loss function 설정
- Optimizer 설정
- 학습

Linear Regression 실습



Input	Output
1	1
2	2
3	3

Linear Regression 실습

```python
import tensorflow as tf

x_data = [1, 2, 3]
y_data = [1, 2, 3]

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")

hypothesis = W * X + b

cost = tf.reduce_mean(tf.square(hypothesis - Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = optimizer.minimize(cost)

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

    for step in range(30):
        _, cost_val = sess.run([train_op, cost], feed_dict={X : x_data, Y : y_data})

        print(step, cost_val, sess.run(W), sess.run(b))

    print("X : 6, Y : ", sess.run(hypothesis, feed_dict={X: 6}))
    print("X : 2.7, Y : ", sess.run(hypothesis, feed_dict={X : 2.7}))
```

Linear Regression 실습

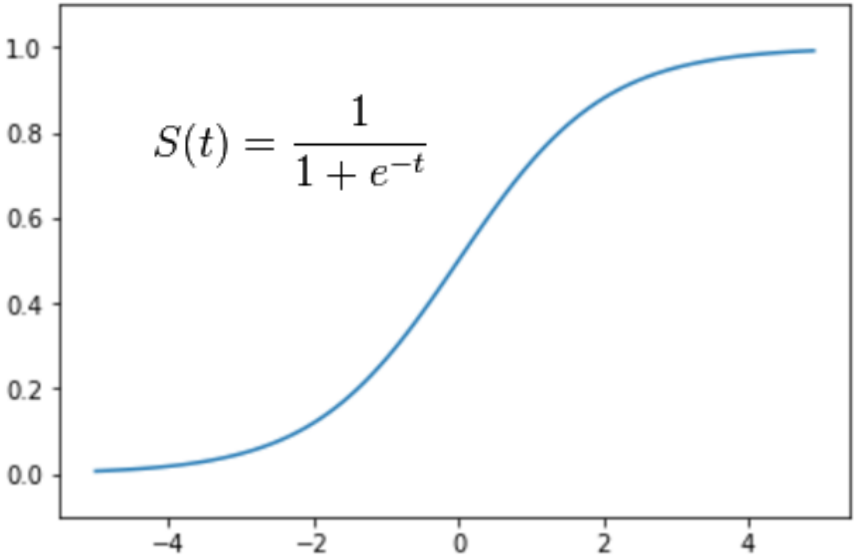
```python
import tensorflow as tf

x_data = [1, 2, 3]
y_data = [1, 2, 3]

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")

hypothesis = W * X + b

cost = tf.reduce_mean(tf.square(hypothesis - Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = optimizer.minimize(cost)

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

    for step in range(30):
        _, cost_val = sess.run([train_op, cost], feed_dict={X : x_data, Y : y_data})

        print(step, cost_val, sess.run(W), sess.run(b))

    print("X : 6, Y : ", sess.run(hypothesis, feed_dict={X: 6}))
    print("X : 2.7, Y : ", sess.run(hypothesis, feed_dict={X : 2.7}))
```

활성화 함수(Activation Function)

- 개념 : 입력 신호의 총합을 출력 신호로 변환하는 함수
- 종류
  - Step function
  - Sigmoid function
  - ReLU fuction
  - Softmax

주로 쓰는 활성함수(activation function)

```
# before_layer -> activation function 거치기 전
# after_layer -> activation function 거친 후

# sigmoid 함수
after_layer = tf.sigmoid(before_layer)

# relu 함수
after_layer = tf.nn.relu(before_layer)

# softmax 함수
after_layer = tf.nn.softmax(before_layer)
```



Sigmoid_Function(시그모이드 함수)

주로 쓰는 활성함수(activation function)

#before_layer -> activation function 거치기 전

#after_layer -> activation function 거친 후

sigmoid 함수

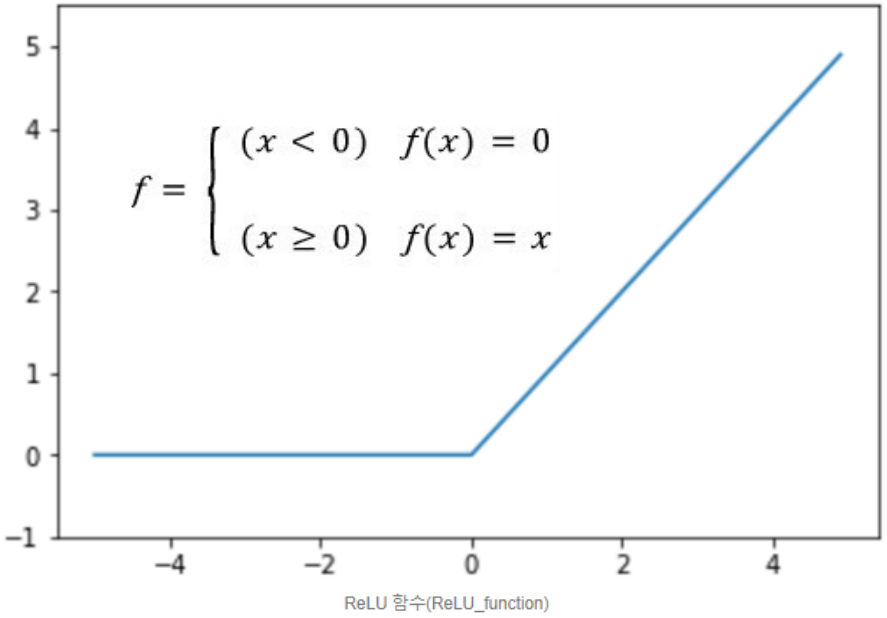
```
after_layer = tf.sigmoid(before_layer)
```

relu 함수

```
after_layer = tf.nn.relu(before_layer)
```

softmax 함수

```
after_layer = tf.nn.softmax(before_layer)
```



ReLU 함수(ReLU_function)

주로 쓰는 활성함수(activation function)

#before_layer -> activation function 거치기 전

#after_layer -> activation function 거친 후

sigmoid 함수

```
after_layer = tf.sigmoid(before_layer)
```

relu 함수

```
after_layer = tf.nn.relu(before_layer)
```

softmax 함수

```
after_layer = tf.nn.softmax(before_layer)
```

$$softmax(x) = \frac{x_i}{\sum_{j=0}^{k} e^{x_j}} \ (i = 0, 1, \ldots k)$$

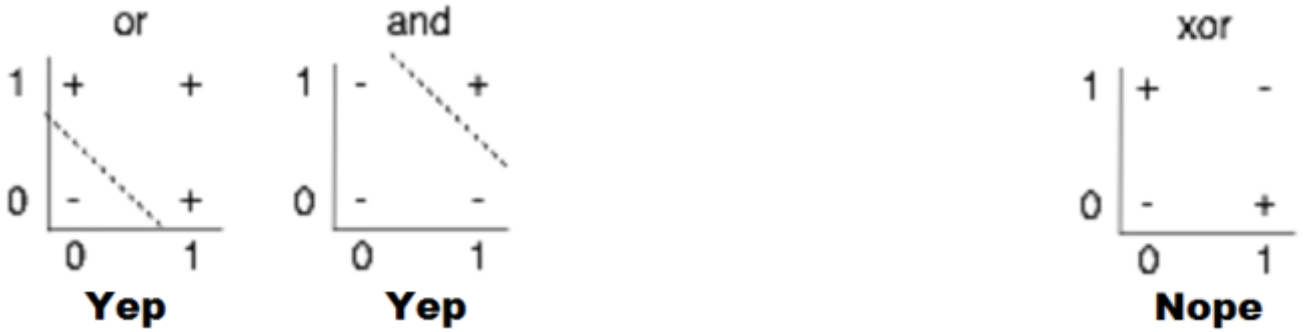
주로 쓰는 손실함수(cost function)

```
# reduce_mean 으로 평균 손실값을 구함
# tf.square 로 거리의 제곱을 손실함수로 적용
cost = tf.reduce_mean(tf.square(Y – model))

# tensorflow가 기본 제공하는 cross entropy 함수를 손실함수로 적용
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model, labels=Y))
```

주로 쓰는 최적화함수(optimizer)

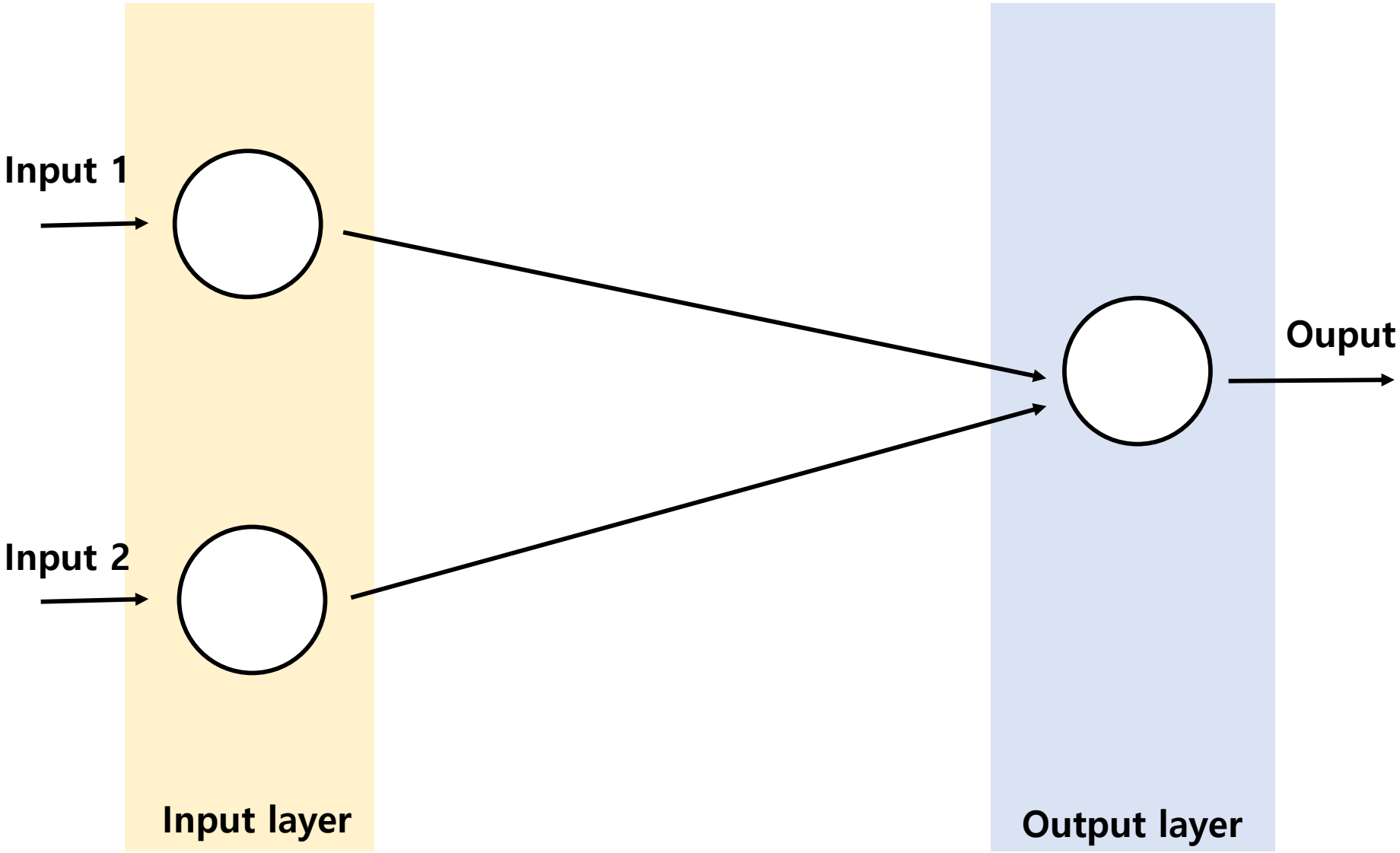
```
# GradientDescentOptimizer를 사용해서 손실값을 최소화하는 최적화 수행
# 0.001 은 learning rate
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001).minimize(cost)
```

AND 문제



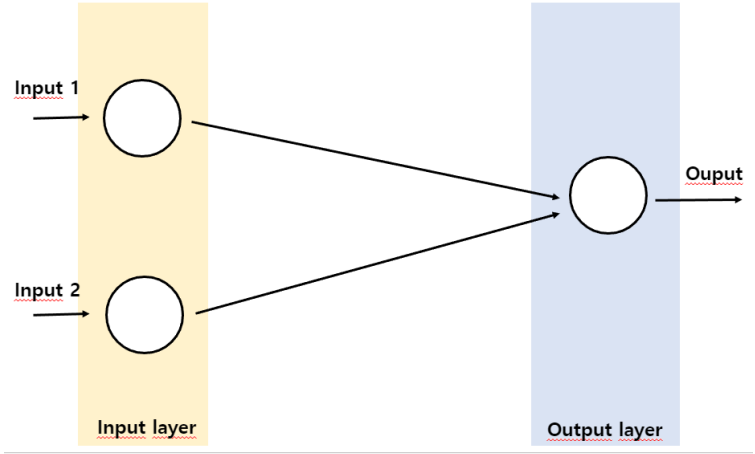
Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Single Layer Perceptron 구성



Output Layer 구성

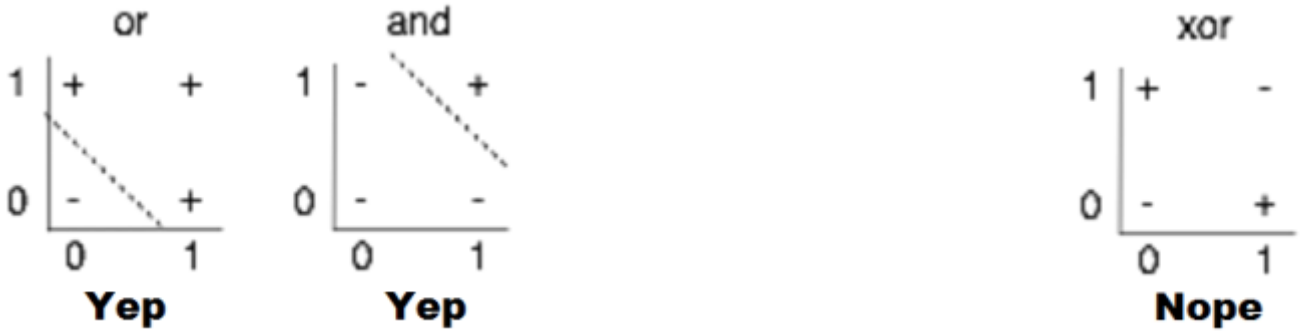
```
# -1.0 ~ 1.0 사이의 임의의 값으로 가중치 초기화
W = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))
 # 편향(bias) 초기화
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
# 입력값 벡터X와 가중치행렬 W을 행렬곱한 후 bias를 더함, 활성화 함수로 sigmoid 사용
logits = tf.add(tf.matmul(X, W), b)
output = tf.nn.sigmoid(logits)
```



Single Layer Perceptron 실습

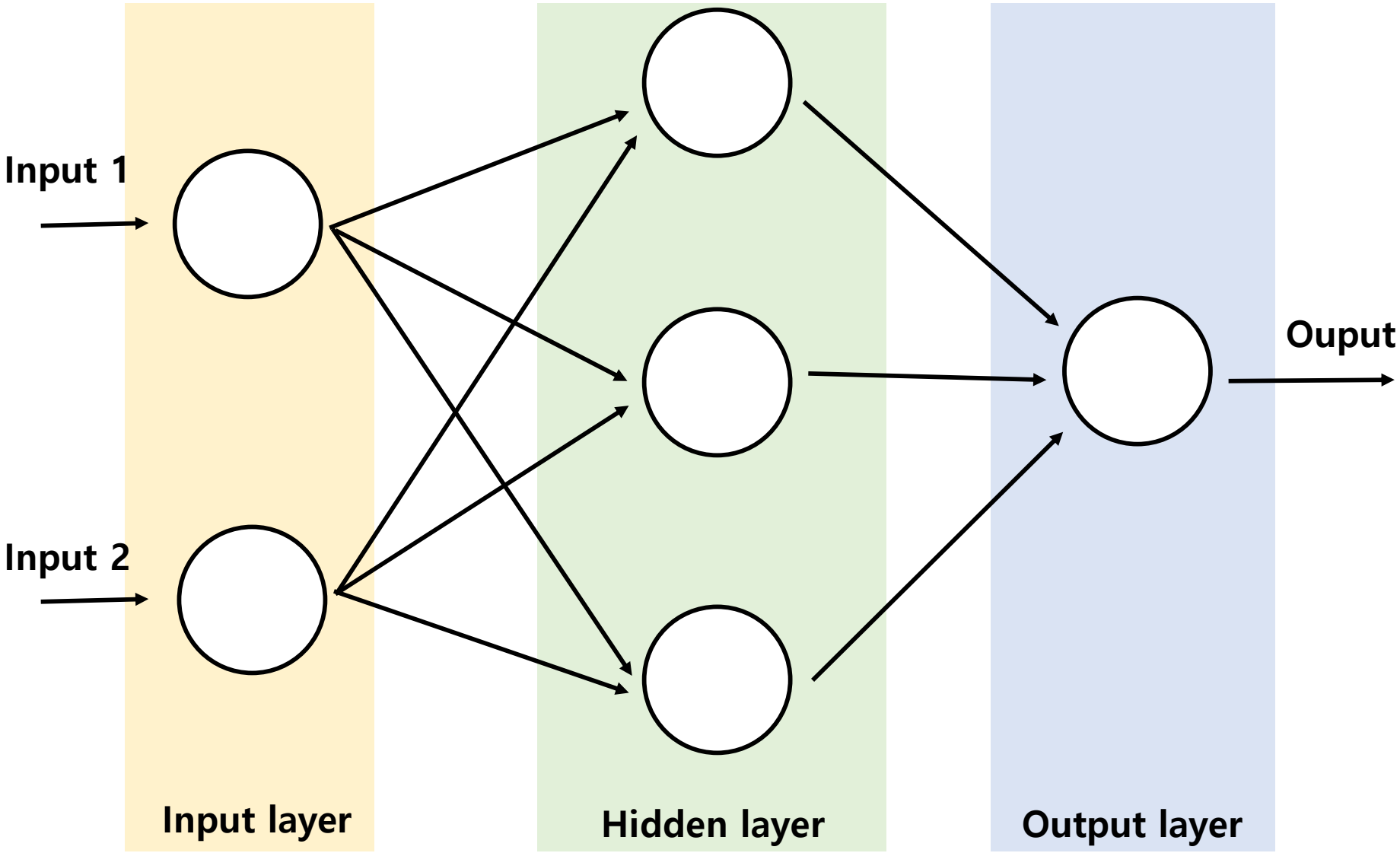
```python
import tensorflow as tf

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


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

W = tf.Variable(tf.random_uniform([2, 1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

logits = tf.add(tf.matmul(X, W), b)
output = tf.nn.sigmoid(logits)

cost = tf.reduce_mean(tf.square(output - Y))
opt = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = opt.minimize(cost)


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

    for step in range(100):
        for x, y in zip(x_data, y_data):
            _, cost_val = sess.run([train_op, cost], feed_dict={X:[x], Y:[y]})
        print(step, cost_val, sess.run(W), sess.run(b))

    print(sess.run(output, feed_dict={X:x_data}))
```

과제 - XOR 문제

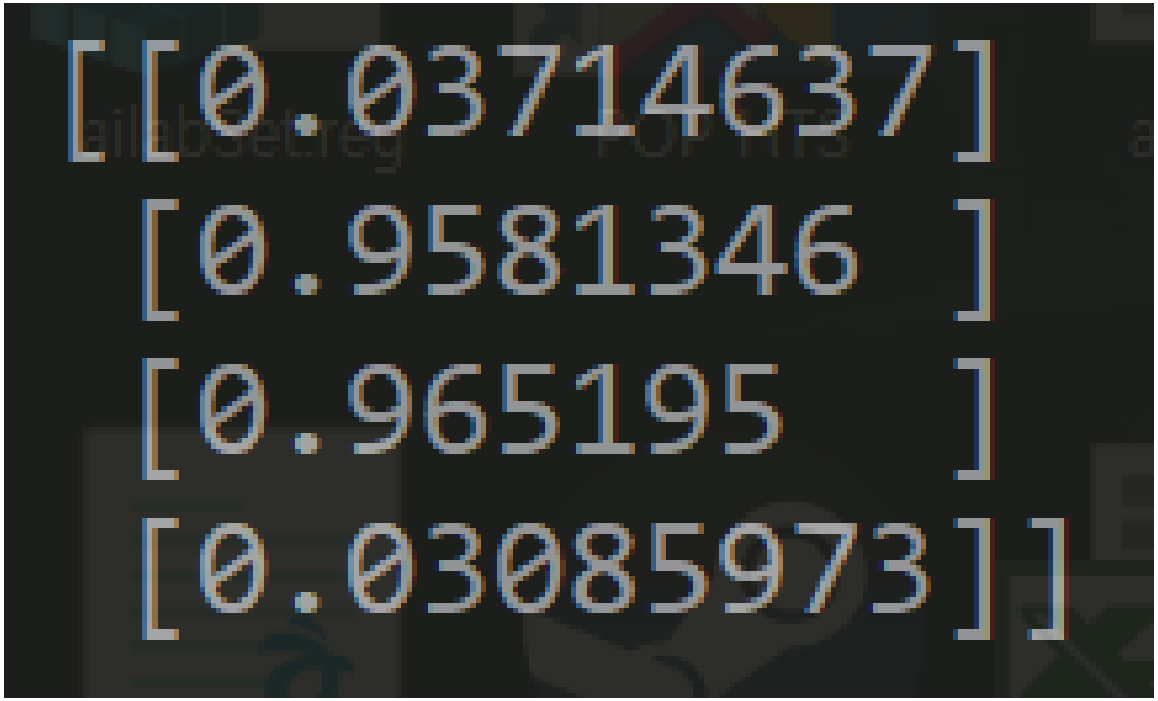


Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Multi Layer Perceptron 구성



과제 - 결과



과제

- 소스와 결과 캡쳐 GitLab에 제출
- 과제 기한 : **다음주 수요일 23:59** 까지
- 수업시간에 한 경우 바로 검사받고 **GitLab**에 제출
- GitLab 관련 사용법은 첨부 파일 확인