Day 3

TensorFlow/NN

Libraries for Deep Learning

Theano

Keras

TensorFlow

TensorFlow

- An open-source library for numerical computation, for which it uses data flow graphs. 데이터플로우그래프를 사용하기 위한 계산 라이브러리
- Developed by Google Brain and actively used at Google both for research and production needs 구글브레인에 위해 개발되어 연구, 생산 목적으로 사용됨
- You can install it with Python pip or conda 피아피나 콘다를 이용하여 설치할수 있음

```
conda install tensorflow
```

Keras

- A powerful easy-to-use Python library for developing and evaluating deep learning models. **딥러닝모델을 개발하고 평가하기 위한 라이브러리**
- Run on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). **텐서플로우, 티아노, CNTK위에서 돌아감**
- You can install it with Python conda or pip **콘다나 피아피를 이용하여 설치할수 있음**

```
pip install keras
```

Theano

- A python library which provides a set of functions for building deep nets that train quickly on our machine.
  딥넷을 개발하기 위한 함수를 제공
- Developed at the University of Montreal, Canada under the leadership of Yoshua Bengio a deep net pioneer. 몬트리올대학에서 개발
- Have to build the deep net from ground up.
  바닥부터 딥넷을 개발해야 함
- TensorFlow and Keras can be used with Theano as backend.
  텐서플로우와 캐라스는 티아노위에서 사용

Intalling TensorFlow

- Download the necessary packages needed for TensorFlow setup. **필요한 패키지를 다운받아 설치함**

```
conda install tensorflow
```

- Then testing if it is working **테스팅으로 버젼을 프린트해봄**

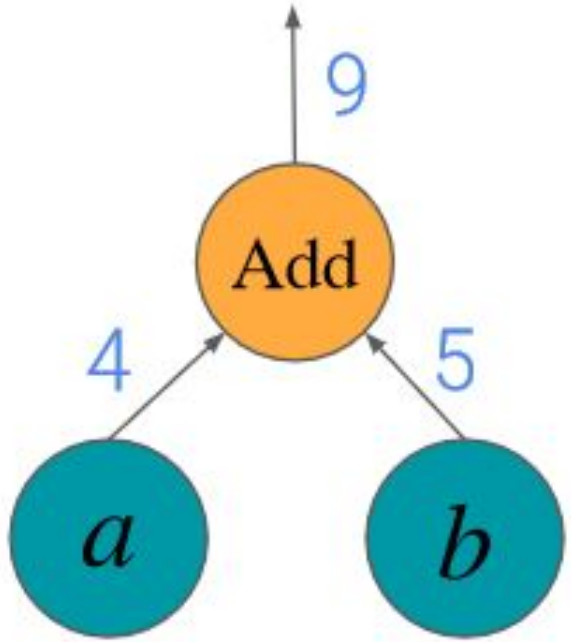
```
import tensorflow as tf
print(tf.__version__)
```

Testing Keras

- Test your installation with https://www.tensorflow.org/tutorials/quickstart/beginner

Tensor

- Defined as multidimensional array or list. **다차원배열이나 리스트로 정의됨**
- Basic data structures in TensorFlow language. **텐서플로우에서 사용하는 기본 데이터구조**
- 0-D tensor: scalar, 1-D tensor: vector, 2-D tensor: matrix, and so on



Data Flow Graph

1 Dimensional Tensor - Vector

```
import numpy as np
tensor_1d = np.array([1.3, 1, 4.0, 23.99])
tensor_1d
tensor_1d[0]
tensor_1d[2]
```

2 Dimensional Tensor - Matrix

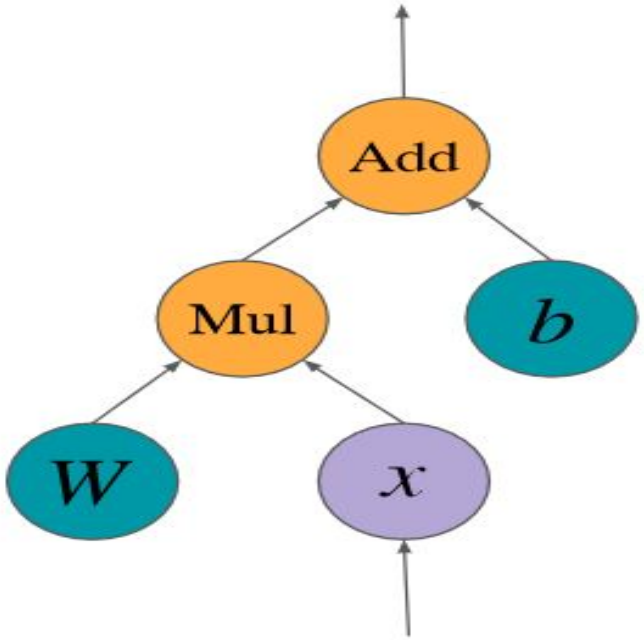
```
import numpy as np

tensor_2d = np.array([(1, 2, 3, 4), (4, 5, 6, 7), (8, 9,
10, 11), (12, 13, 14, 15)])

tensor_2d

tensor_2d[3][2]
```

Data Flow Graph

- Tensor represents the connecting edges in any flow diagram called the Data Flow Graph consisting of variables or constants. 텐서는 데이터플로우그래프에서 연결하는 엣지로 변수, 상수로 구성됨
- Nodes in Data Flow Graph are operators (ops), functions on tensors 노드들은 연산자들로 텐서위의 함수들

Basic Code Structure

- Constants are fixed value tensors - not trainable 상수는 훈련되지 않은 고정된 값의 텐서
- Variables are tensors initialized in a session - trainable
  변수는 훈련될수 있고, 세션에서 초기화됨
- Placeholders are tensors of values that are unknown during the graph construction, but passed as input during a session 입력값



$$\hat{y} = Wx + b$$

variable variable

ops placeholder

Session

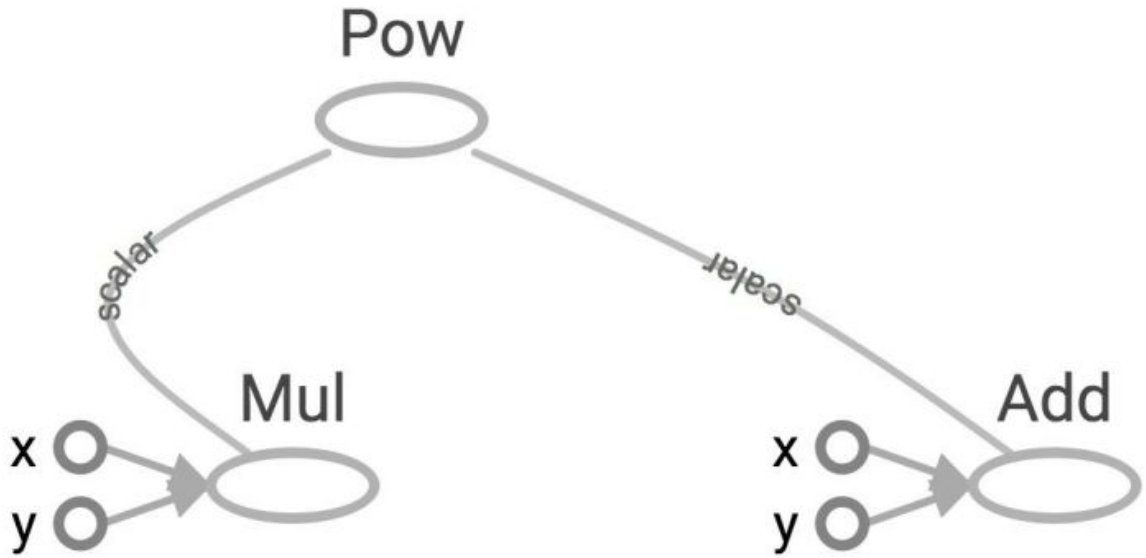
```
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
print(sess.run(a))
sess.close()
```

Create a session, assign it to variable sess so we can call it later

- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated. **연산의 실행과 텐서의 평가를 세션안에 포함시킴**

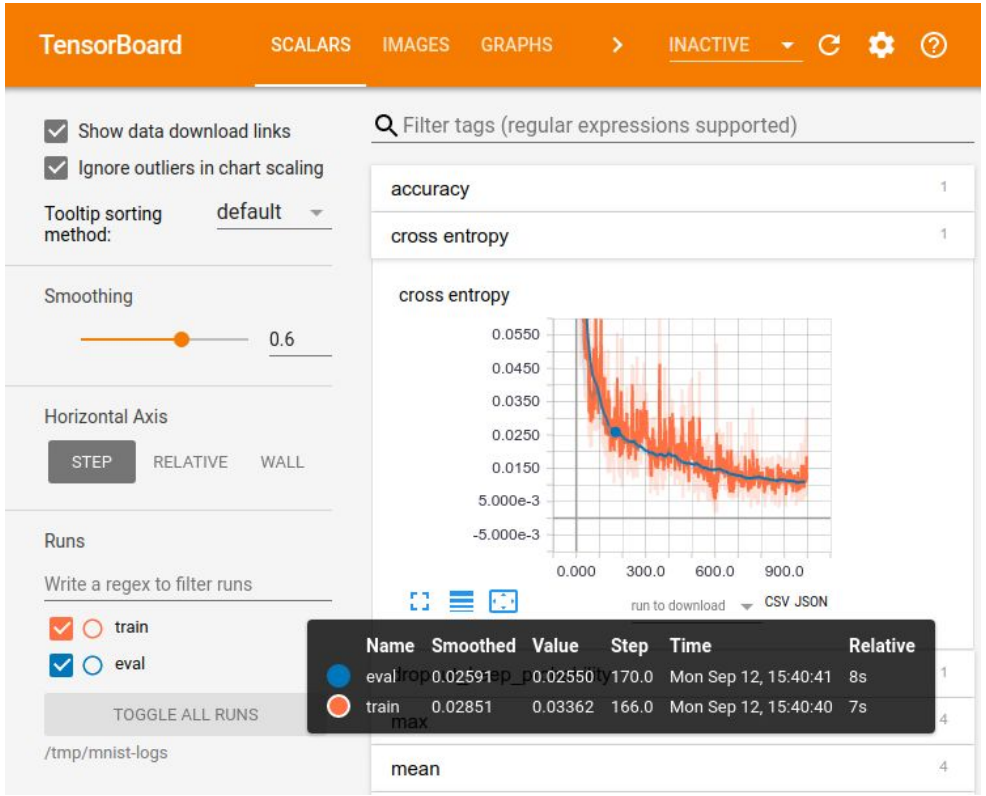
Operators

```
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)
op3 = tf.pow(op2, op1)
with tf.Session() as
sess:
    op3 = sess.run(op3)
```



TensorBoard

- A visualization software that comes with any standard TensorFlow installation
  텐서플로우 설치하면 같이 오는 시각화소프트웨어



Visualizing Graph

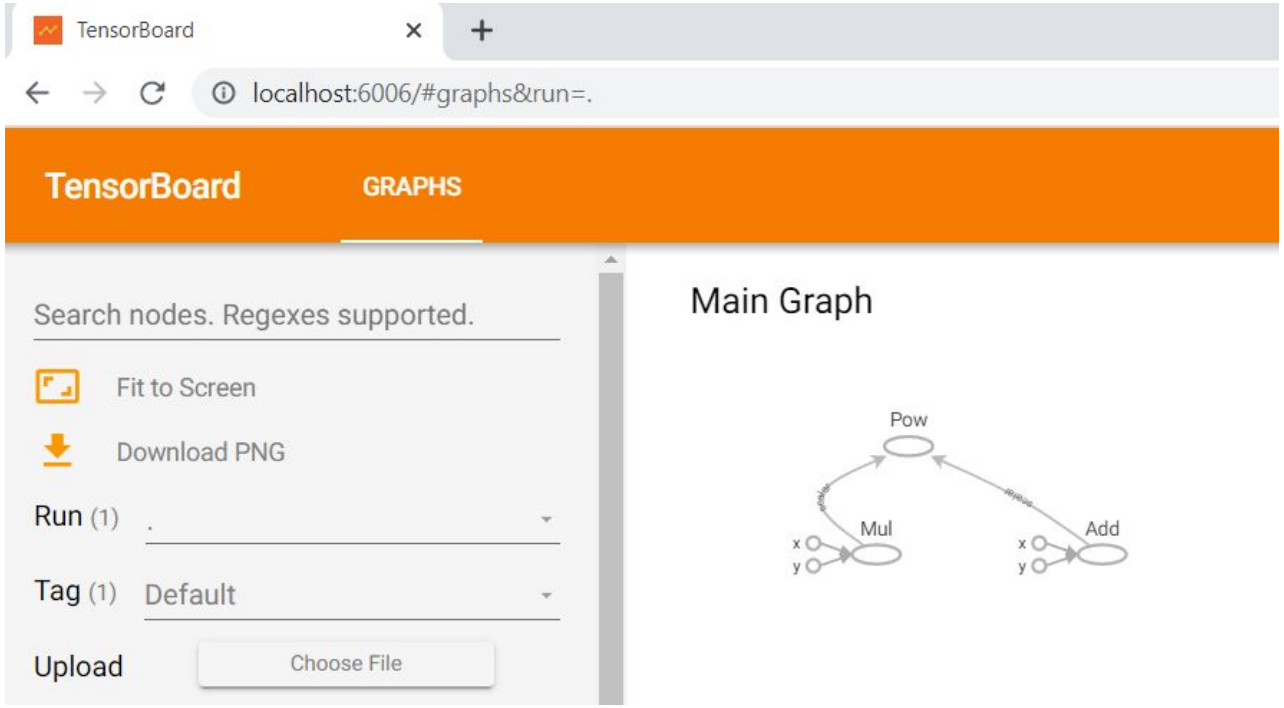
```
Import tensorflow as tf
tf.reset_default_graph()
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)
op3 = tf.pow(op2, op1 ①)
with tf.Session() as sess:
    tf.summary.FileWriter('./graphs', sess.graph)
    op3 = sess.run(op3)
    writer = tf.summary.FileWriter('./graphs', sess.graph)
```

명령프롬프트에서 텐서보드 실행

tensorboard --logdir=./graphs --port 6006

브라우저에 로컬호스트치고 텐서보드 들어감

http://localhost:6006



Exercise #3

- Complete the graph and session tutorial on https://www.easy-tensorflow.com/tf-tutorials/basics/graph-and-session
  - **a와 b에 2, 3을 넣어서 더한후 출력**
  - **x와 y에 2, 3을 넣어서 더하고, 곱하고, 그 결과로 지수를 만들고 x와 더한값을 곱한 결과들을 출력**

Constant

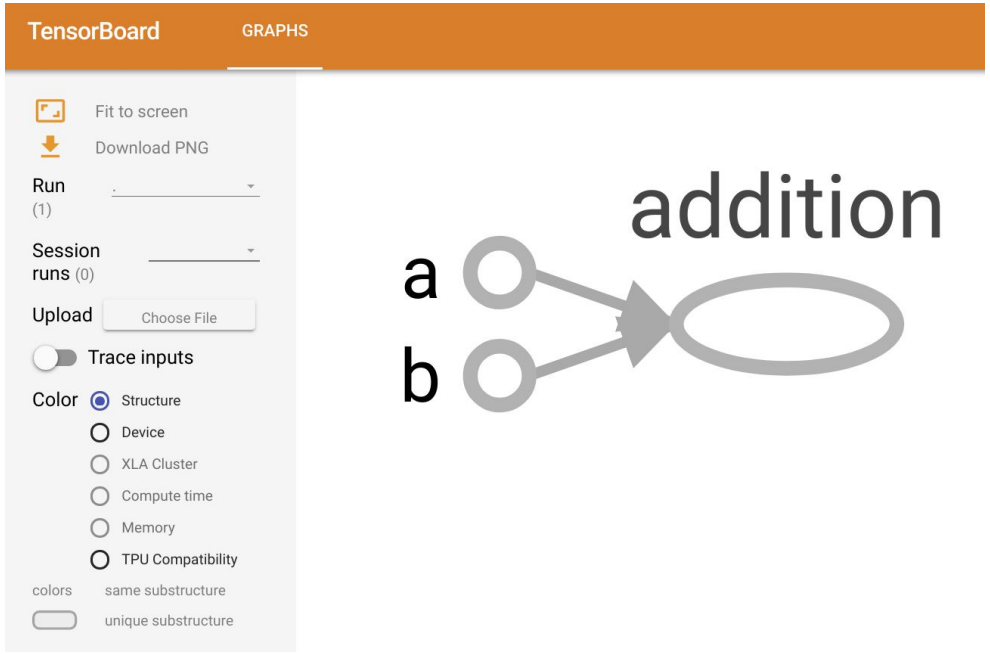
```
import tensorflow as tf

# create graph
a = tf.constant(2)
b = tf.constant(3)
c = a + b

# launch the graph in a session
with tf.Session() as sess:
    print(sess.run(c))
```

```
a = tf.constant(2, name='A')
b = tf.constant(3, name='B')
c = tf.add(a, b, name='Sum')
# launch the graph in a session
with tf.Session() as sess:
    print(sess.run(c))
```

Creates a node that takes value and it does not change. **노드를 만들어서 값을 지정. 변하지 않음**

Visualizing Graph

```
import tensorflow as tf
tf.reset_default_graph()
a = tf.constant(2, name="a")
b = tf.constant(3, name="b")
c = tf.add(a, b,
name="addition")


with tf.Session() as sess:
    writer =
tf.summary.FileWriter('./graphs
', sess.graph)
    print(sess.run(c))
```




```
tensorboard --logdir=./graphs --port 6006
```

Scalar and Matrix Constant

```
s = tf.constant(2.3,
name='scalar',
dtype=tf.float32)

m = tf.constant([[1, 2],
[3, 4]], name='matrix')


# launch the graph in a
session

with tf.Session() as sess:

    print(sess.run(s))

    print(sess.run(m))
```

- Constants can be defined with different types (integer, float, etc.) and shapes (vectors, matrices, etc.)
  정수, 소수등으로 정의할수도 있고, 벡터, 메트릭스등의 모양으로 정의할수도 있음

Exercise #3

```
import tensorflow as tf

import numpy as np


matrix1 = np.array([(2,2,2), (2,2,2),
(2,2,2)], dtype='int32')

matrix2 = np.array([(1,1,1), (1,1,1),
(1,1,1)], dtype='int32')

matrix1 = tf.constant(matrix1)

matrix2 = tf.constant(matrix2)

matrix_product = tf.matmul(matrix1,
matrix2)

matrix_sum = tf.add(matrix1,matrix2)

matrix3 =
np.array([(2,7,2),(1,4,2),(9,0,2)],dtype
= 'float32')
```

```
matrix_det =
tf.matrix_determinant(matrix3)

with tf.Session() as sess:

   result1 =
sess.run(matrix_product)

   result2 = sess.run(matrix_sum)

   result3 = sess.run(matrix_det)

print(result1)

print(result2)

print(result3)
```

- Visualize the graph!

Determinant

- A scalar value that can be computed from the elements of a square matrix and encodes certain properties of the linear transformation described by the matrix.
  정방행렬의 요소들로부터 계산된 값으로 행렬을 변형하는 방법의 하나

$$\det\begin{bmatrix} a & b \\ c & d \end{bmatrix} = a \bullet d - c \bullet b$$

$$\det\begin{bmatrix} 8 & 3 \\ 4 & 2 \end{bmatrix} = 8 \bullet 2 - 4 \bullet 3 = 16 - 12 = 4$$

Examples:

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = 1 \cdot 4 - 2 \cdot 3 = -2$$

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = (1 \cdot 5 \cdot 9 + 2 \cdot 6 \cdot 7 + 3 \cdot 4 \cdot 8) - (3 \cdot 5 \cdot 7 + 2 \cdot 4 \cdot 9 + 1 \cdot 6 \cdot 8) = 0$$

Variables

```
s= tf.Variable(2, name="scalar")

m = tf.Variable([[1, 2], [3, 4]], name="matrix")

W = tf.Variable(tf.zeros([784, 10]))
```

- In-memory buffers containing tensors
  메모리버퍼에 텐서를 저장
- Should be explicitly initialized **초기화 시켜서 사용**

Initialize Variables

```
a = tf.get_variable(name="A",
initializer=tf.constant(2))
b = tf.get_variable(name="B",
initializer=tf.constant(3))
c = tf.add(a, b, name="Add")
# add an Op to initialize global variables
init_op = tf.global_variables_initializer()
# launch the graph in a session
with tf.Session() as sess:
    # run the variable initializer operation
    sess.run(init_op)
    # now let's evaluate their value
    print(sess.run(a))
    print(sess.run(b))
    print(sess.run(c))
```

- First initialize all the variables and then proceed to evaluate them.
  우선 변수를 초기화시켜서 평가함

Exercise #3

- Create the weight and bias matrices for a fully-connected layer with 2 neuron to another layer with 3 neuron. In this scenario, the weight and bias variables must be of size [2, 3] and 3 respectively
  정규분포로부터 2x3 매트릭스를 만들고, 0이 3개있는 벡터를 만들어 프린트하시요.
  - Weights are usually initialized from a normal distribution using tf.truncated_normal_initializer(stddev=0.01).
  - Biases are usually initialized from zeros using tf.zeros_initializer().

Placeholder

```
a = tf.placeholder(tf.float32, shape=[5])

b = tf.placeholder(dtype=tf.float32, shape=None, name=None)

X = tf.placeholder(tf.float32, shape=[None, 784], name='input')

Y = tf.placeholder(tf.float32, shape=[None, 10], name='label')
```

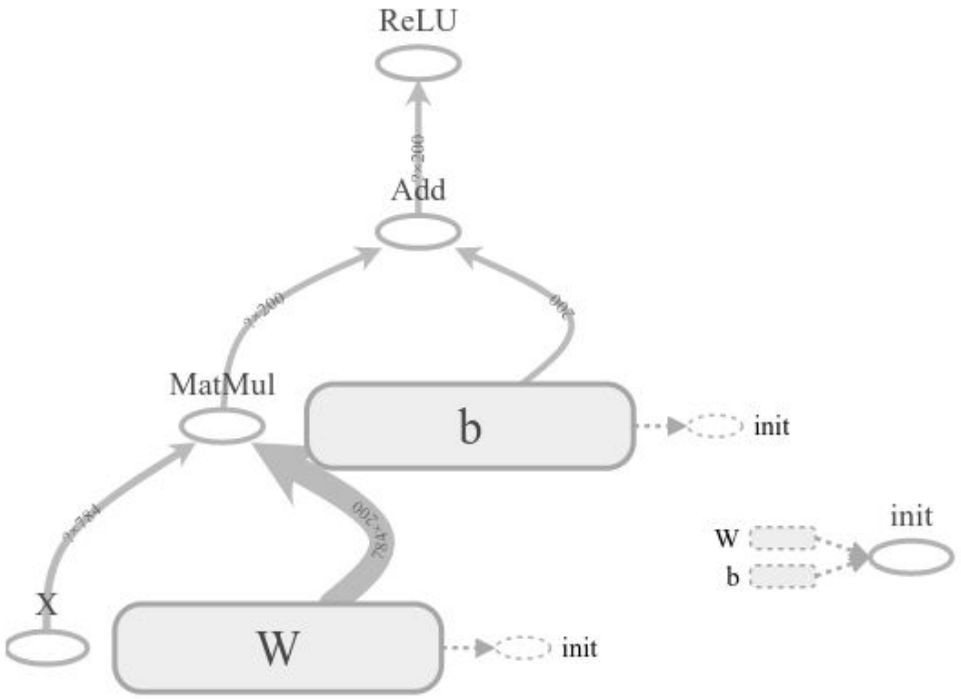
- A variable that we assign data in a future time 나중에 데이터를 할당

Placeholder Example

```
a = tf.constant([5, 5, 5], tf.float32, name='A')
b = tf.placeholder(tf.float32, shape=[3], name='B')
c = tf.add(a, b, name="Add") #상수벡터와 플레이스홀더를 더함

with tf.Session() as sess:
    # create a dictionary:
    d = {b: [1, 2, 3]} #플레이스홀더값을 만듬
    # feed it to the placeholder
    print(sess.run(c, feed_dict=d)) #플레이스홀더값을 이용하여 결과출력
```

Exercise #3

- Create a data flow graph of the neural network look like this



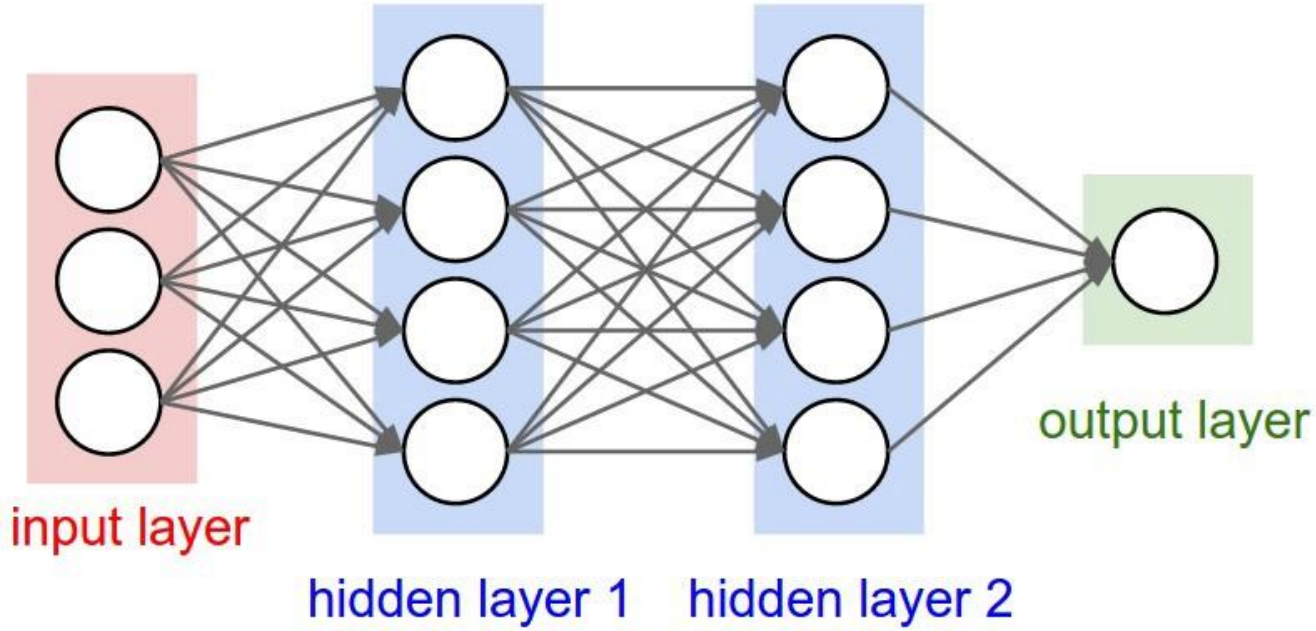
- X – 100 images generated by random pixel values (100x784 matrix using np.random.rand)
- W – 784 x 200 matrix from normal distribution (mean=0, stddev=0.01)
- B – 200 vector with 0
- ReLU – tf.nn.relu()


Neural Network
기계학습과 인지과학에서
생물학의 신경망(동물의
중추신경계중 특히 뇌)에서
영감을 얻은 통계학적 학습
알고리즘

Components of Neural Networks

- An input layer, x
- An arbitrary amount of hidden layers
- An output layer, ŷ

Each column is a layer. The first layer of your data is the input layer. Then, all the layers between the input layer and the output layer are the hidden layers



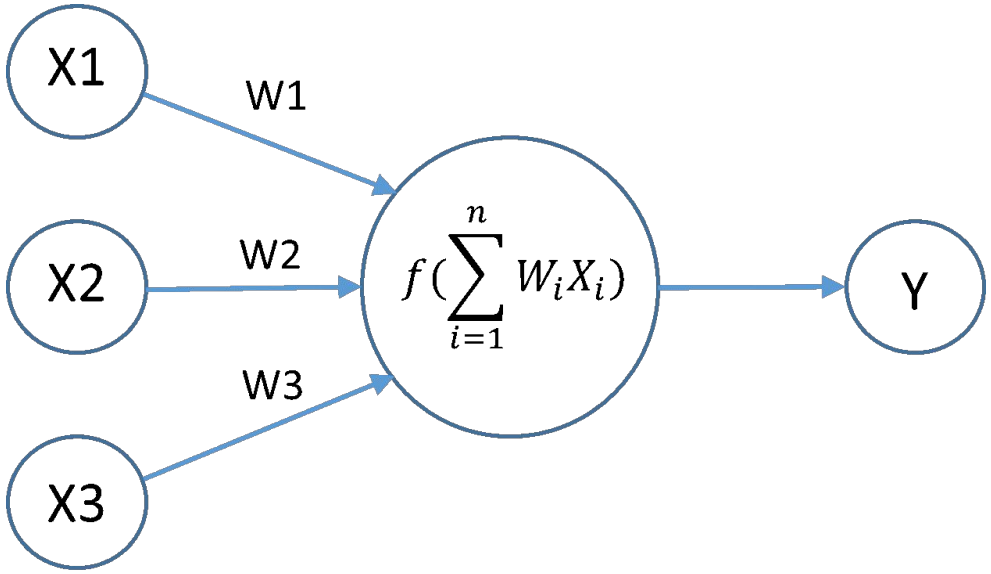
Neural Network Model

```
from sklearn.neural_network import MLPClassifier

mlp =
MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=500)
mlp.fit(x_train,y_train)
```

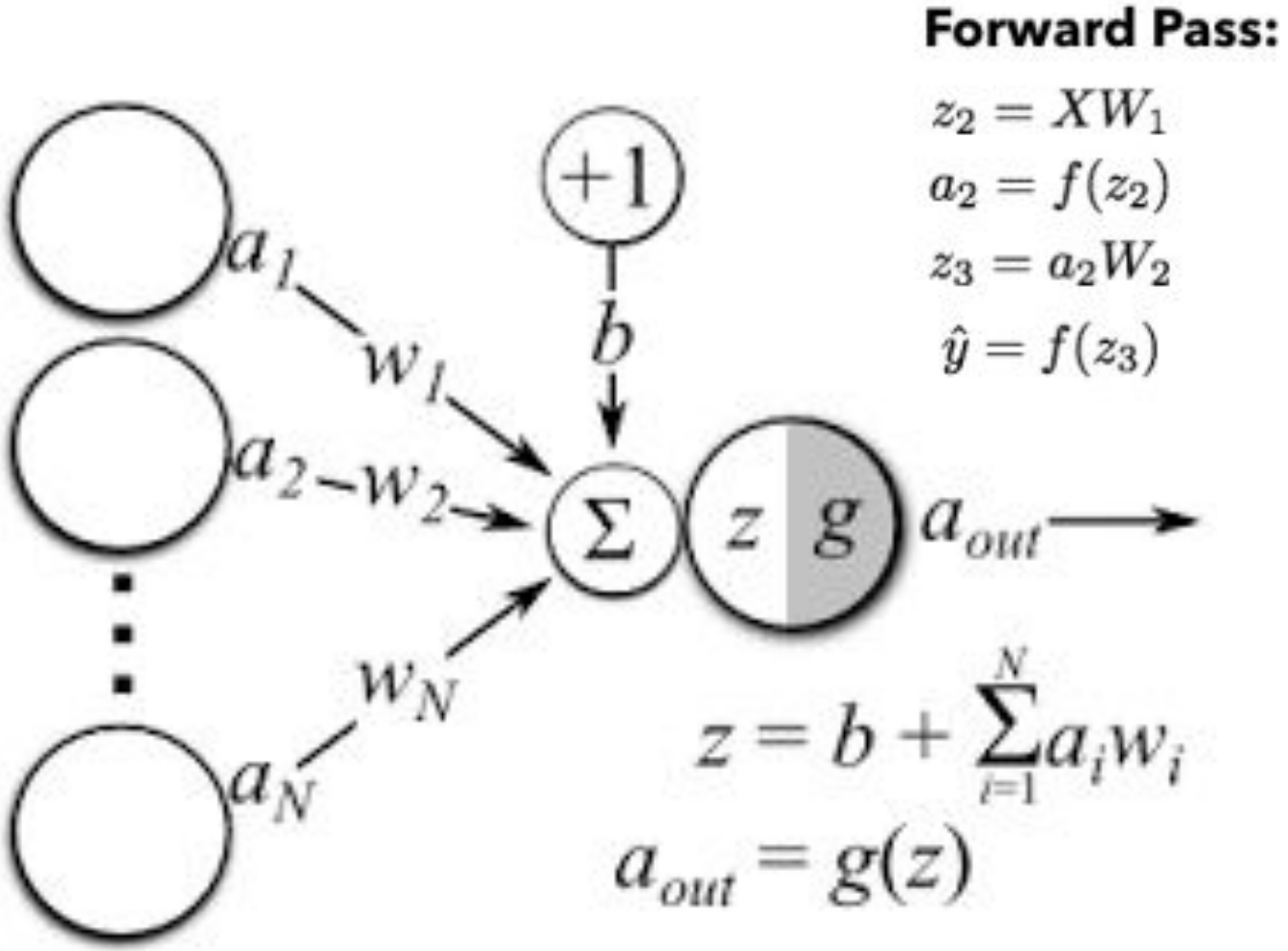
Components of Neural Networks (cont.)

- A set of weights and biases between each layer, W and b
  레이어사이에 가중치와 바이어스가 있음
- A choice of activation function for each hidden layer, σ
  각 히든레이어에 엑티베이션 함수가 있음

The circles are neurons or nodes, with their functions on the data and the lines/edges connecting them are the weights/information being passed along



Output Calculations



Receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification

입력을 받아 계산한후 출력을 함. 분류문제에 적합

Prediction

```
pred = mlp.predict(x_test)
r2_score(y_test, mlp.predict(x_test))
```

```
pred
```

```
import mglearn
```

```
mglearn.plots.plot_2d_separator(mlp, x_train, fill=True,
alpha=.3)
```

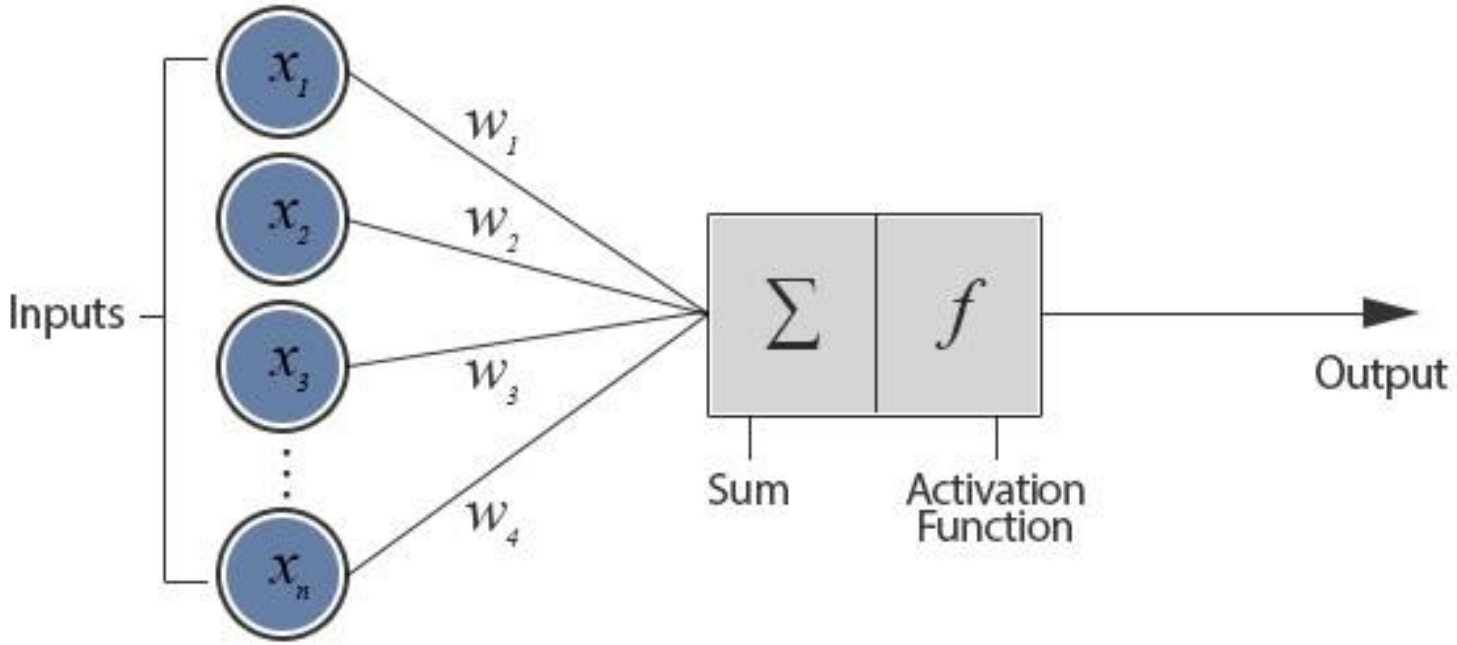
```
mglearn.discrete_scatter(x_train[:,0], x_train[:,1],
y_train)
```

Predictions and Evaluation

```
pred = mlp.predict(x_test)

pred

r2_score(y_test, mlp.predict(x_test))

accuracy_score(y_test, mlp.predict(x_test))

cross_val_score(mlp,x_train, y_train, cv=10,
scoring='accuracy')

confusion_matrix(y_test, mlp.predict(x_test))

print(classification_report(y_test, mlp.predict(x_test)))
```

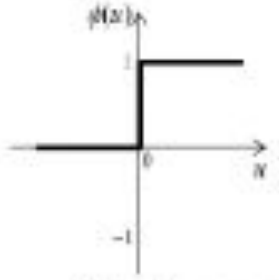
Activation Function (활성함수)

- **output Y를 만들기 위해 은닉된 부분의 합을 가중하는 값.** e.g., sigmoid, sign activation function



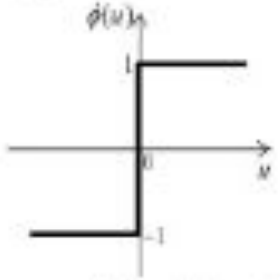
Activation Functions

step function



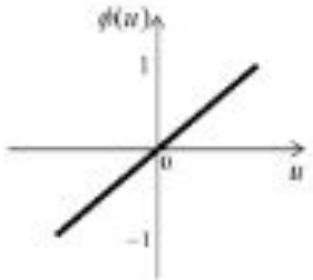
$$\phi_{step}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

sign function

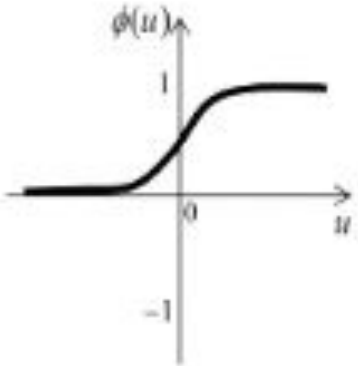


$$\phi_{sign}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

identity function

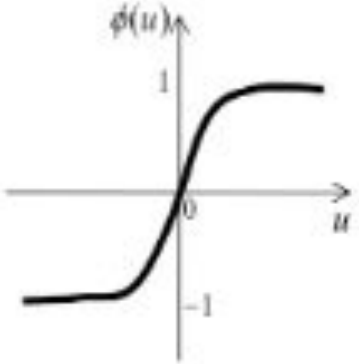


$$\phi_{id}(u) = u$$



$$\phi_{sig}(u) = \frac{1}{1+e^{-u}}$$

sigmoid function



$$\phi_h = \frac{e^u - 1}{e^u + 1}$$

hyper tangent function

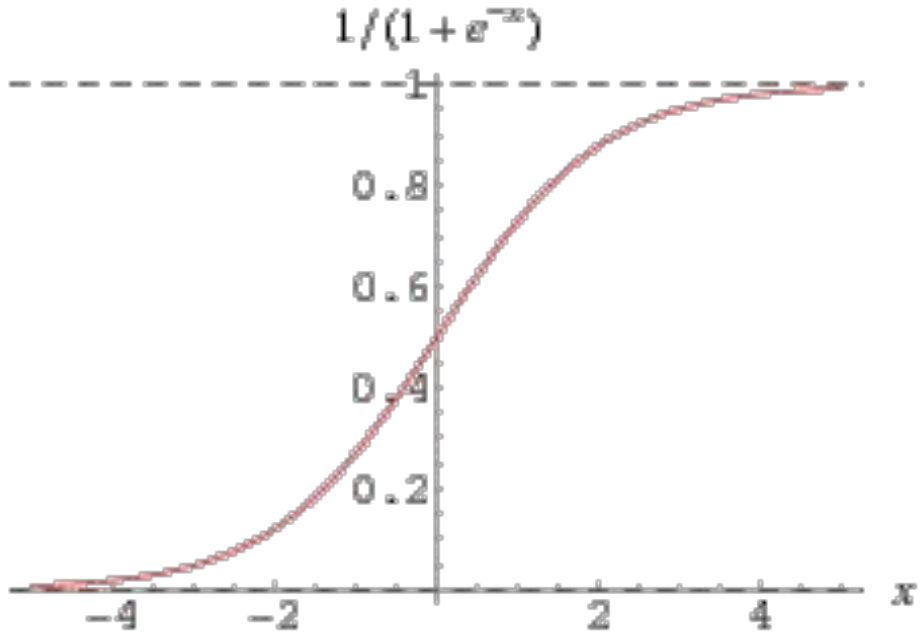
Can capture non-linearity in the network so it would be capable of learning more complex pattern

비선형성을 처리하여 더 복잡한 패턴도 배울수 있음

Activation: Sigmoid

- A mathematical function having a characteristic "S"-shaped curve or sigmoid curve
  s자형곡선을 그리는 함수들
- Also called the sigmoidal curve or logistic function **시그모이드 또는 로지스틱 함수라고도 함**

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



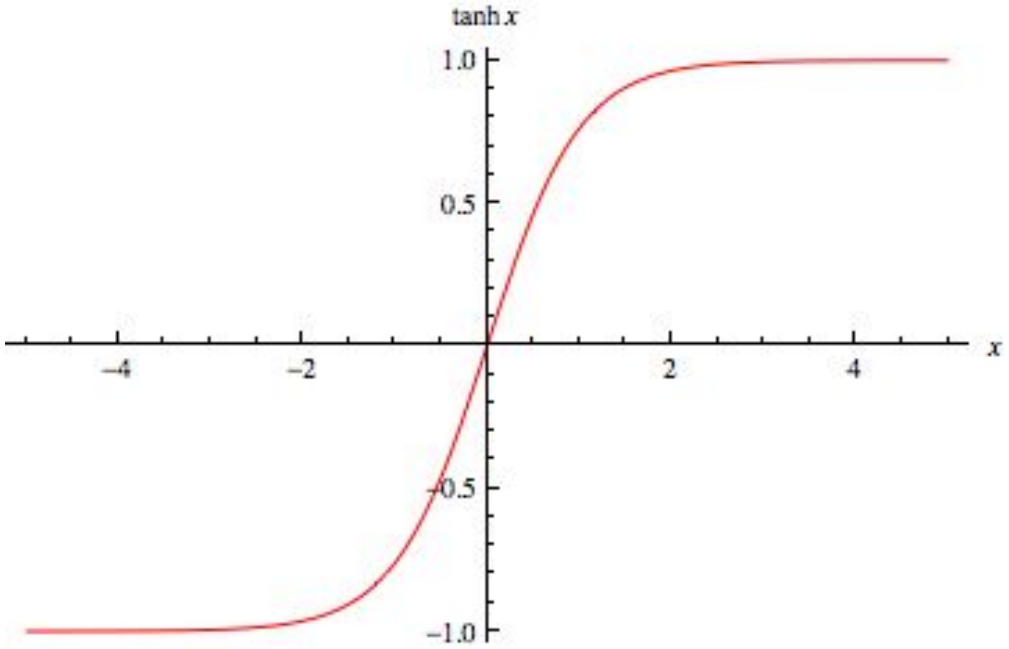
Sigmoid Code

```
import numpy as np


def sigmoid(x, derivative=False):
    if (derivative == True):
        return x * (1-x)
    return 1 / (1+np.exp(-x))
sigmoid(3)
sigmoid(1)
```

Activation: tanh

- Logistic sigmoid but better **시그모이드 종류이나 더 성과가 좋음**
- The range of the tanh function is from (-1 to 1). **-1에서 1사이**

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



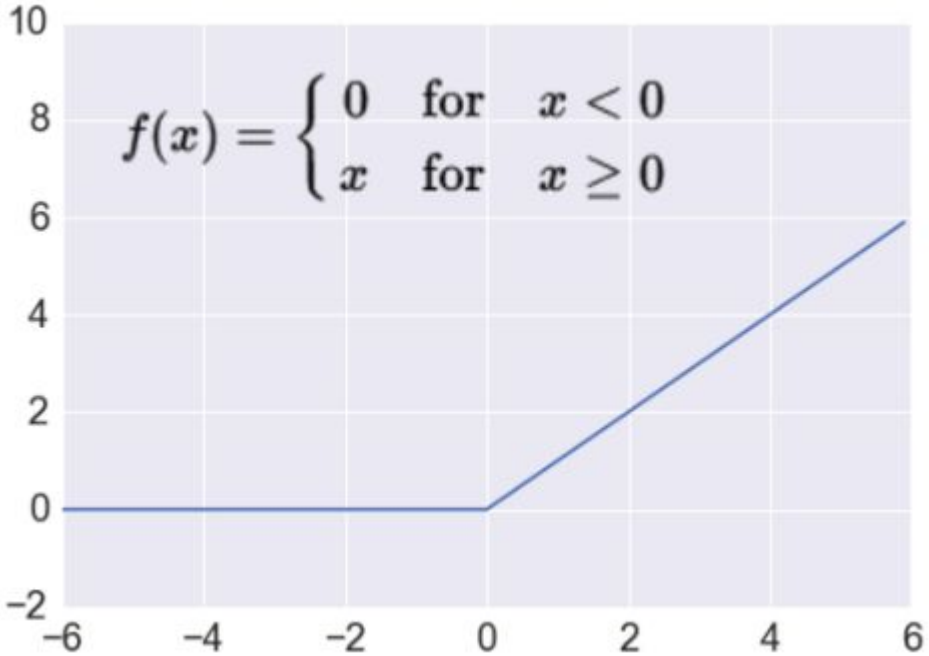
Tanh Code

```
def tanh(x, derivative=False):
    if (derivative == True):
        return (1 == (x **2))
    return np.tanh(x)
tanh(3)
tanh(1)
```

Activation: ReLU

- Most Deep Networks use ReLU nowadays
  딥러닝 네트워크에서는 거의 렐루함수를 사용
- Trains much faster 훈련속도가 더 빠름
- Less expensive operations 계산이 적음
- Prevents the gradient vanishing problem

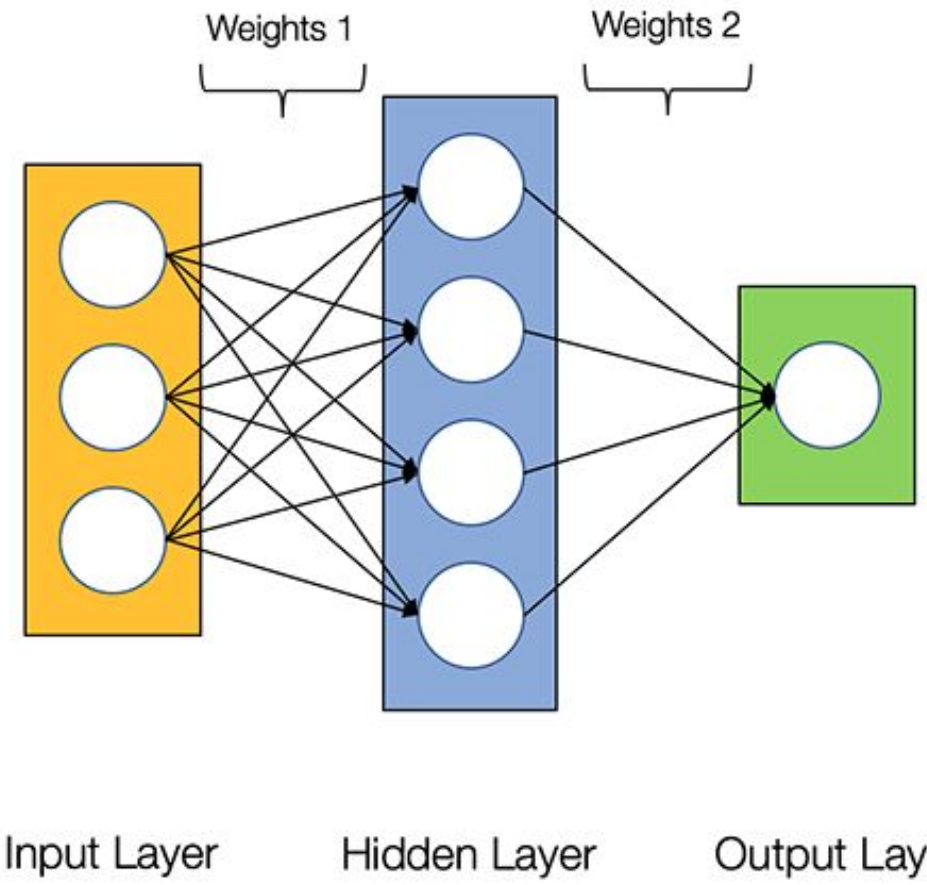
$$f(x) = \max(0, x)$$



$R^n \rightarrow R^n_+$

Takes a real-valued number and thresholds it at zero 실수를 가져다가 영보다 큰수로 만듬

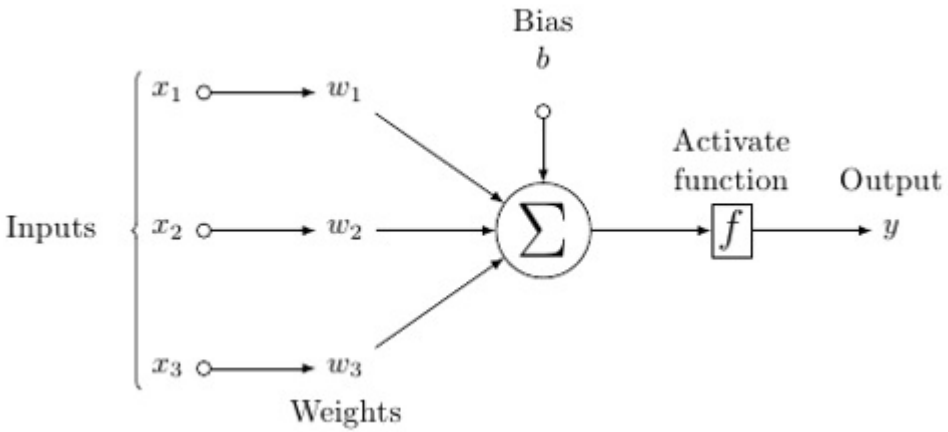
Architecture of 2-Layer Neural Network



The input layer is typically excluded when counting the number of layers in a Neural Network

입력레이어는 일반적으로 카운팅하는데서 빠짐

Weights ad Biases



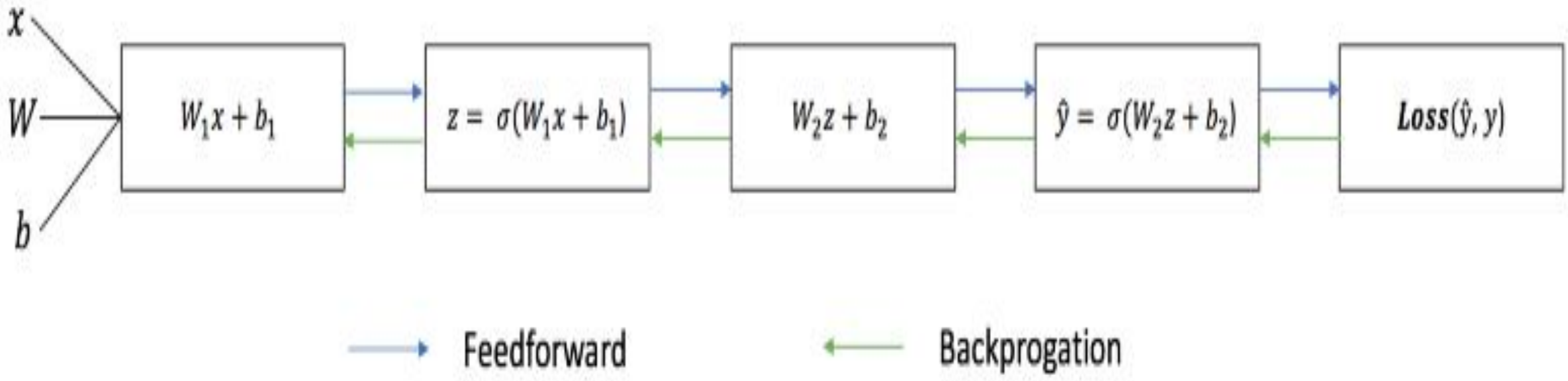
$$\hat{y} = \sigma(W_2 \sigma(W_1 x + b_1) + b_2)$$

- The weights W and the biases b are the only variables that affects the output ŷ. 예측값에 영향을 미치는 두 변수
- The right values for the weights and biases determines the strength of the predictions 가중치와 바이어스의 값에 따라 예측의 정확도가 결정됨

Training Neural Network

- The process of fine-tuning the weights and biases from the input data 학습을 통해 가중치와 바이어스의 값을 조정함
- Each iteration of the training process consists of the following steps:
  - Calculating the predicted output ŷ, known as feedforward 처음 계산할때는 포워드
  - Updating the weights and biases, known as backpropagation 가중치와 바이어스정할때는 백워드

Sequential Graph

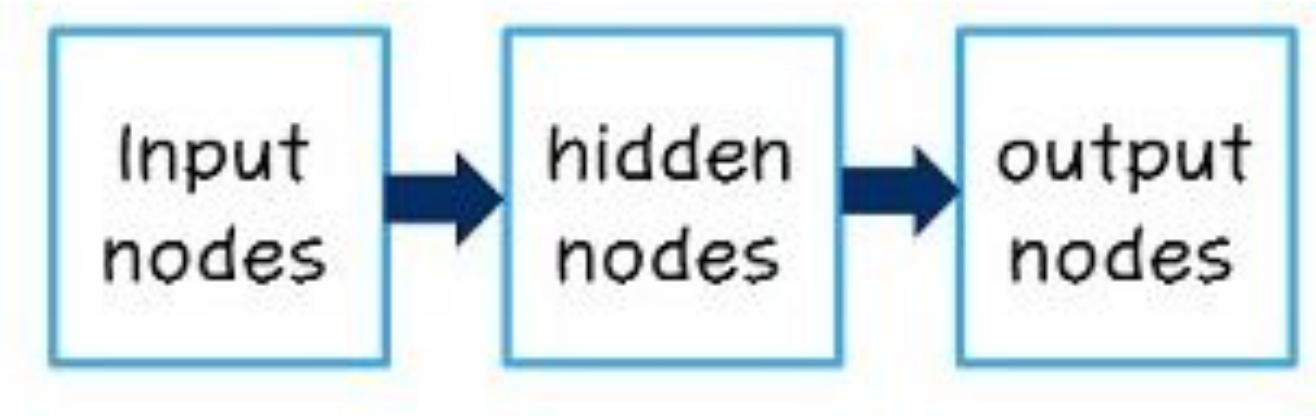


포워드와 백프로퍼게이션을 반복하며 최적의 모델을 찾아냄

Feedforward

- Just simple calculus **(input에서 hidden으로 hidden에서 output으로 쭉 한 방향으로 진행)**

$$\hat{y} = \sigma(W_2 \sigma(W_1 x + b_1) + b_2)$$



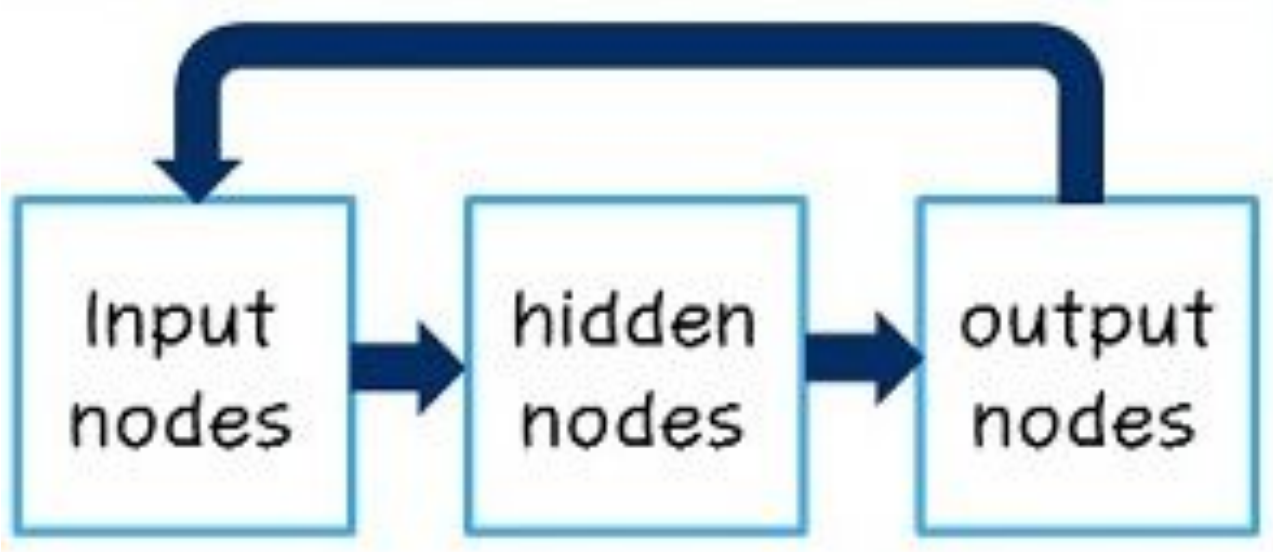
Loss Function

$$Sum - of - Squares\ Error = \sum_{i=1}^{n} (y - \hat{y})^2$$

- Evaluate the "goodness" of the predictions (i.e. how far off are the predictions) **예측이 얼마나 정확한지 보여주는 함수**
- Can use a simple sum-of-sqaures error (SSE) **에러의 제곱의 합을 사용**
- The sum-of-squares error is simply the sum of the difference between each predicted value and the actual value. The difference is squared so that we measure the absolute value of the difference **에러는 예측값과 실제값의 차이**
- Our goal in training is to find the best set of weights and biases that minimizes the loss function **목적함수, 즉 에러를 최소화하는 것이 훈련의 목적**

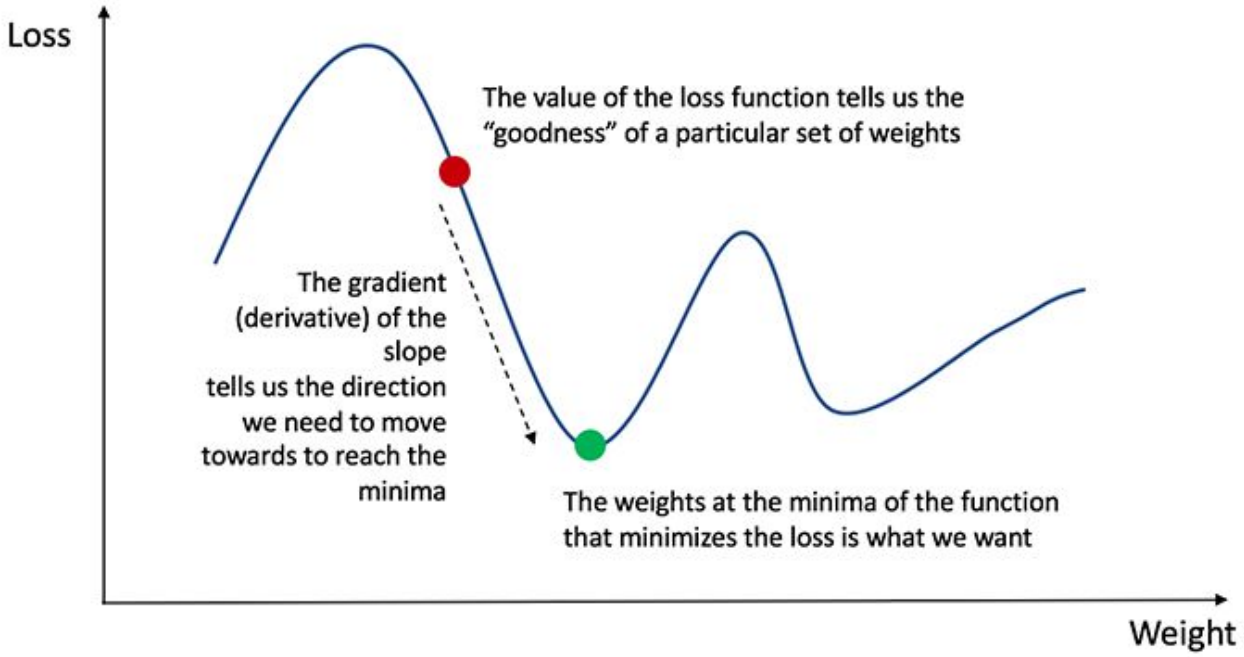
Backpropagation (역전파)

- Find a way to propagate the error back, and to update our weights and biases
  가중치의 에러를 줄이는 방향으로 과정을 계속 반복
- Need to know the derivative of the loss function with respect to the weights and biases 로스함수에서 가중치와 바이어스의 변화의 정도를 보여줘야 함

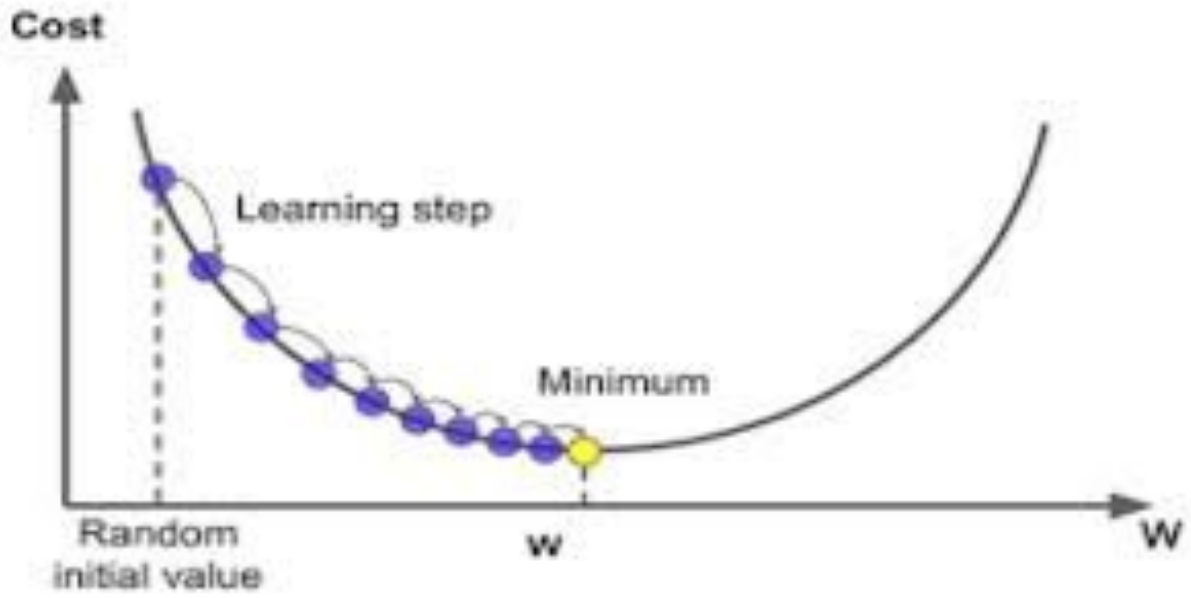


Gradient Descent

- Simply update the weights and biases by increasing/reducing the loss function 로스함수의 증가/감소를 이용하여 가중치와 바이어스를 조정
- Adjusts weights according to the error they caused 에러를 최소화하는 방향으로 가중치를 조정, 즉 기울기의 경사가 최소화를 위한 방향을 정해줌



Learning Rate



- Too small, then converge very slowly 너무 작으면 수렴되는 데 오래걸림
- Too big, then overshoot and even diverge 너무 크면 최소점을 지나치거나 갈라짐

Chain Rule

- Cannot directly calculate the derivative of the loss function with respect to the weights and biases, so need the chain rule
  직접적 계산이 어려우므로 체인룰을 사용함

$$Loss(y,\hat{y}) = \sum_{i=1}^{n}(y - \hat{y})^2$$

$$\frac{\partial\, Loss(y,\hat{y})}{\partial W} = \frac{\partial Loss(y,\hat{y})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial W} \qquad \text{where } z = Wx + b$$

$$= 2(y - \hat{y}) * \text{derivative of sigmoid function} * x$$

$$= 2(y - \hat{y}) * z(1\text{-}z) * x$$

Exercise #3

- Use College Data Set which has several features of a college and a categorical column and build a predictive model to determine whether or not the School is Public or Private 컬리지 데이터셋은 컬리지에 대한 속성과 범주형 변수를 갖는다. 그 학교가 공립인지 사립인지 결정하는 예측모델을 만드시요
  - Normalize before building model 정규화 시킨후 모델을 만드시요.
  - Train and test split 훈련데이터와 테스트데이터로 나누시요.

Mean Removal

- It involves removing the mean from each feature so that it is centered on **zero.**
  평균을 제거해서 0이 중앙값이 되게 함
- Removes any bias from the features. **각 속성의 바이어스를 제거함**

```
import numpy as np
from sklearn.preprocessing import scale
input_data = np.array([[3, -1.5, 3, -6.4], [0, 3, -1.3, 4.1], [1, 2.3, -2.9, -4.3]])
data_standardized = scale(input_data)
```

Scaling

- The values of every feature in a data point can vary between random values. Scales them to match specified rules. **특정룰에 맞춰서 스케일링**

```
from sklearn.preprocessing import MinMaxScaler
data_scaler = MinMaxScaler(feature_range = (0, 1))
data_scaled = data_scaler.fit_transform(input_data)
```

Normalization

- Involves adjusting the values in the feature vector so as to measure them on a common scale. Adjusts so that they sum up to 1. **조정해서 합이 1이 되게 함**

```
from sklearn.preprocessing import normalize

data_normalized = normalize(input_data, norm  = 'l1') #l1
uses absolute value, l2 uses square root
```

Binarization

- Used to convert a numerical feature vector into a Boolean vector. **2개값의 벡터로 전환**

```
from sklearn.preprocessing import Binarizer
data_binarized =
Binarizer(threshold=1.4).transform(input_data)
```

Standardization

- Transforms attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. **평균이 0이고 표준편차가 1인 숫자로 표준화**

```
from sklearn.preprocessing import
StandardScaler
scalar = StandardScaler()

input_standard = scaler.transform(input_data)
```

Loss Function and Optimizer

```
loss = tf.reduce_mean(tf.square(y_pred-y))

optimizer =
tf.train.GradientDescentOptimizer(learning_rate=0.1)

train_op = optimizer.minimize(loss)
```

Exercise #3

- The Boston dataset is a collection of data about housing values in the suburbs of Boston. Use the Boston dataset and predict the median value of owner-occupied homes (medv) using all the other continuous variables available. **보스톤데이터셋은 보스톤교외의 집값에 대한 데이터를 포함한다. 보스톤데이터를 이용하여 집값의 중위수를 예측하시요.**

Exercise #3

- Build a prediction model for AirPassengers Monthly Airline Passenger Numbers 1949-1960. 항공기승객의 수를 예측하는 모델을 만드시요.
- Build a prediction model for Breast cancer (load_breast_cancer() in sklearn.datasets) 유방암을 예측하는 모델을 만드시요

Exercise #3

- Use College Data Set and build a predictive model using tensorflow. **텐서플로우를 이용해서 컬리지데이터의 에러가 가장 작은 가중치를 찾으시요**
  - 데이터를 넣을 때 넘파이 어레이를 list로 바꿔서 입력하시요.
  - Refer to https://towardsdatascience.com/a-line-by-line-laymans-guide-to-linear-regression-using-tensorflow-3c0392aa9e1f
- Compare this result with statsmodel. **Statsmodel에 있는 회귀분석 결과와 비교하시요**
- Compare this result with sklearn.LinearRegression. **Sklearn에 있는 LinearRegression결과와 비교하시요**