메디치소프트 기술연구소

딥러닝 4일차

Constants

.tf.constant 로 저장된 hello는 상수로 문자열 'HELLO. IT'S ME.'를 가집니다.
이 값은 텐서플로우 내부에서 사용되는 상수입니다. 텐서플로우에서 아직 run을 하지 않은 상태이므로 아직 값을 출력 할 수 없습니다.

```
import tensorflow as tf
hello = tf.constant("HELLO. IT'S ME. ")
print(hello)
sess = tf.Session()
print("OPEN SESSION")
hello_out = sess.run(hello)
print(hello_out)
```

Variables

. 텐서플로우에서의 Variable은 모델링 된 그래프에서 내부적으로 사용되는 변수 입니다. 이 변수값이 사용자가 정의하는 모델에 맞게 최적화 되어 결과를 만들어 냅니다.
. 텐서플로우 Variable을 run하기 위해서는 initialize (초기화) 작업 필수

```
import tensorflow as tf
#matrix 5*2 mean=0.0, std=0.1 graph define
weight = tf.Variable(tf.random_normal([5, 2], stddev=0.1))
print(weight)
sess = tf.Session()
sess.run(weight.initializer)
weight_out = sess.run(weight)
print(weight_out)
```

Variables

. 아래와 같이 Variable의 갯수가 여러개가 된다면 global_Variable_initializer()를 사용하여 초기화

```
weight2 = tf.Variable(tf.random_normal([5, 2], stddev=0.1))
weight3 = tf.Variable(tf.random_normal([5, 2], stddev=0.1))
sess = tf.Session()
sess.run(tf.global_variables_initializer())
weight2_out = sess.run(weight2)
weight3_out = sess.run(weight3)
print(weight2_out)
print(weight3_out)
```

Shape

. 텐서플로우로 정의된 매트릭스의 shape을 알고 싶은 경우 tf.shape(대상)을 실행(eval)하여 확인할 수 있습니다. 텐서 자체를 실행하려면 위에서 실행하였던 것 처럼 Session을 실행하여야 합니다. 또는 Session이 아닌 InteractiveSession()을 이용하면 바로 eval()을 사용할 수 있습니다.

```
import tensorflow as tf
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a*b
# print(c.eval()) => 오류 발생 : 기본 Session이 없기때문에 실행되지 않는다.
with tf.Session():
 print(c.eval())
```

. interactiveSession()을 설정하면 default 세션으로 설정되므로 with 구문으로 Session을 실행

```
sess = tf.InteractiveSession()
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
print(c.eval())
```

Shape, Rank

```python
import tensorflow as tf

sess = tf.InteractiveSession()
t1 = tf.constant([1, 2, 3, 4])
t2 = tf.constant([[1, 2],
                  [3, 4]])
t3 = tf.constant([[[[1, 2, 3, 4],
                    [5, 6, 7, 8],
                    [9, 10, 11, 12]],
                  [[13, 14, 15, 16],
                    [17, 18, 19, 20],
                    [21, 22, 23, 24]]]])
print(tf.shape(t1).eval())
print(tf.shape(t2).eval())
print(tf.shape(t3).eval())
```

1차원 => Rank :1, Shape : 4

2차원 => Rank :2, Shape : (2 , 2)

3차원 => Rank :4, Shape : (1 , 2 , 3 , 4)

```
[
    [
        [
            [1,2,3,4],
            [5,6,7,8],
            [9,10,11,12]
        ],
        [
            [13,14,15,16],
            [17,18,19,20],
            [21,22,23,24]
        ]
    ]
]
```

Matmul VS multipy

```python
import tensorflow as tf

sess = tf.InteractiveSession()
matrix1 = tf.constant([[1., 2.],
                       [3, 5]])
matrix2 = tf.constant([[2.],
                       [2.]])
print(tf.matmul(matrix1, matrix2).eval())
print((matrix1*matrix2).eval())
```

```
[[ 6.]
 [16.]]
[[ 2.  4.]
 [ 6. 10.]]
```

1차원 => Rank :2, Shape : (2, 2)

2차원 => Rank :2, Shape : (2 , 1)

Matmul => Rank :2, Shape : (2, 1)

[1, 2] [3, 5] [2]
[2] => [[6.] , [16.]]

Reduce mean

```
sess = tf.InteractiveSession()

print(tf.reduce_mean([1, 2], axis=0).eval())  # 1
x = [[1., 2.],
     [3., 4.]]
print(tf.reduce_sum(x).eval())          # 10
print(tf.reduce_mean(x, axis=0).eval())# (1+3)/2=2,   (2+4)/2=3
print(tf.reduce_mean(x, axis=1).eval())# (1+2)/2=1.5, (3+4)/2=3.5
print(tf.reduce_sum(x, axis=0).eval())#[4., 6.]
print(tf.reduce_sum(x, axis=1).eval())#[3., 7.]
print(tf.reduce_mean(tf.reduce_sum(x, axis=1)).eval()) #5=(3+7)/2
```

```
1
10.0
[2. 3.]
[1.5 3.5]
[4. 6.]
[3. 7.]
5.0
```

Argmax : 위치

```
import tensorflow as tf
sess = tf.InteractiveSession()
x =[[0, 1, 2],
    [2, 1, 0]]
print(tf.argmax(x, axis=0).eval())
print(tf.argmax(x, axis=1).eval())
```

```
[1 0 0]
[2 0]
[2 0]
```


MEDICISOFT

reshape :

```python
import tensorflow as tf
import numpy as np
sess = tf.InteractiveSession()
t = np.array([[[0, 1, 2],
               [3, 4, 5]],

              [[6, 7, 8],
               [9, 10, 11]]])
c1 = tf.constant([1, 3, 5, 7, 9, 0, 2, 4, 6, 8, 3, 7])
print(t.shape) # (2,2,3)
print(tf.reshape(t,  shape=[-1, 3]).eval())
print(tf.reshape(c1, shape=[2, -1]).eval()) #[[1 3 5 7 9 0] [2 4 6 8 3 7]]
print(tf.reshape(c1, shape=[-1, 3]).eval()) #[[1 3 5] [7 9 0] [2 4 6] [8 3 7]]
```

```
(2, 2, 3)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
[[1 3 5 7 9 0]
 [2 4 6 8 3 7]]
[[1 3 5]
 [7 9 0]
 [2 4 6]
 [8 3 7]]
```


MEDICISOFT

Basic derivative

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$ 순간 변화율

$f(x) = 3$

$f(x) = 3$

$f(x) = x$

$f(x) = 2x$ $\quad f(x) = x + x$

$f(x) = x + 3$

$f(x) = 2x$

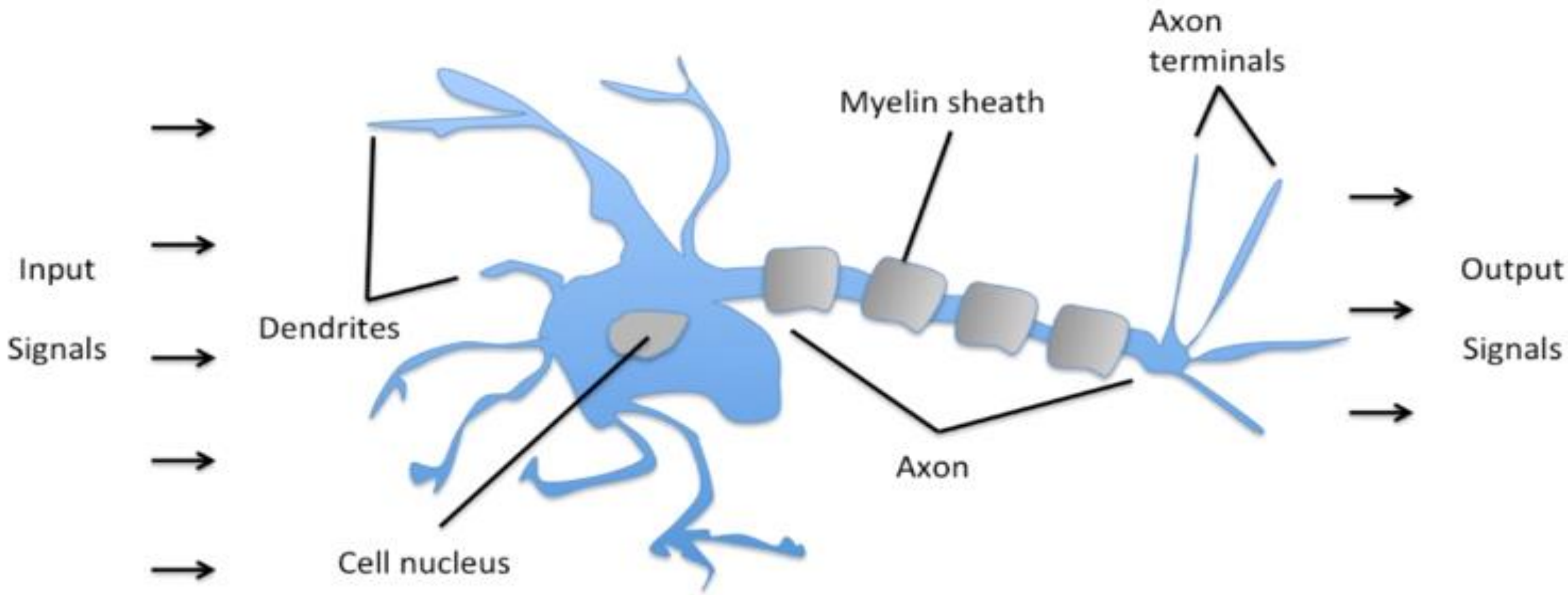
$f(x, y) = xy, \frac{\partial f}{\partial x}$

$f(x, y) = x + y, \frac{\partial f}{\partial x}$

$f(x, y) = x + y, \frac{\partial f}{\partial y}$

$f(x, y) = xy, \frac{\partial f}{\partial y}$

NN



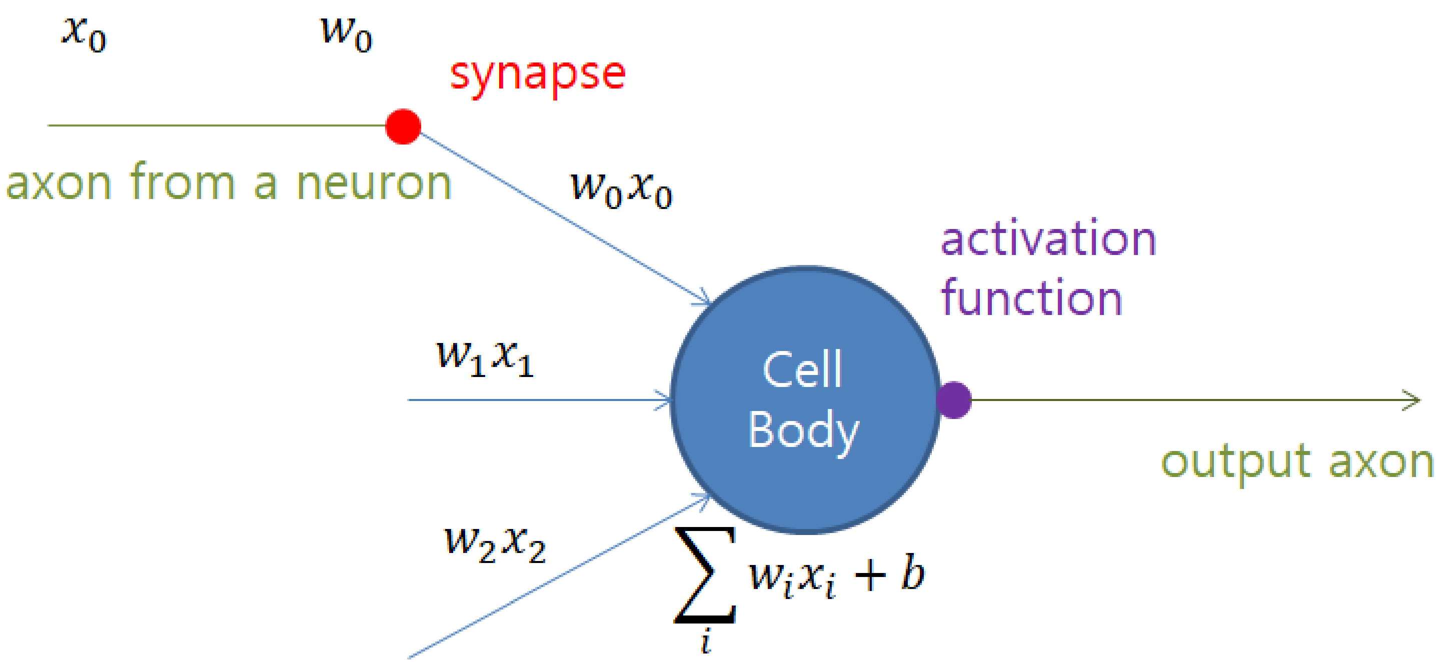
Neuron (출처 : http://sebastianraschka.com)

Input의 길이에 따라 신호의 양이 달라(WX) +b

MEDICISOFT

NN

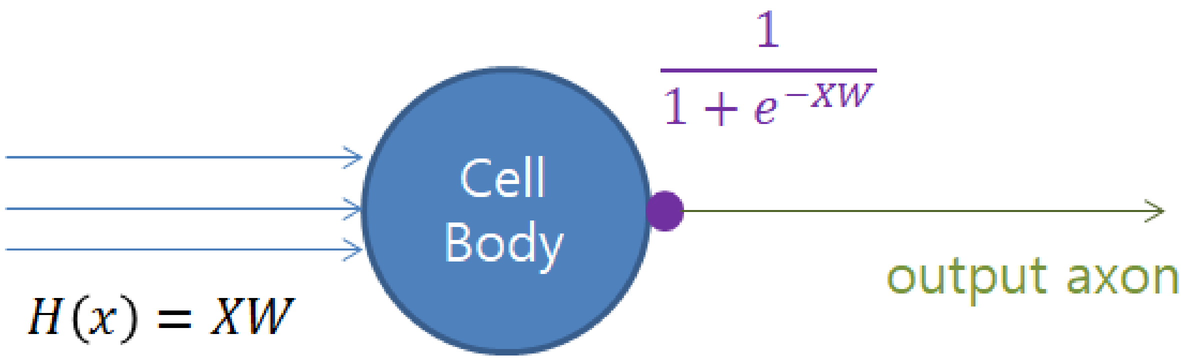
. 뉴런에 있는 각 Dentrites들은 시냅스(Synapse)라고 하는 접점을 통해 외부 뉴런과 연결
. 뉴런은 입력으로 들어오는 여러 개의 신호들을 하나로 합산한 다음 Activation Function을 통해 자신의 출력으로 만들어 낸다. 만들어낸 출력은 다시 다른 뉴런의 입력으로 들어가게 된다



딥러닝(Deep Learning)의 기본을 이루는 뉴런

NN

. 입력값의 행렬인 X와 가중치의 행렬인 W를 곱한 값이 있고 그 값을 시그모이드 함수를 통과

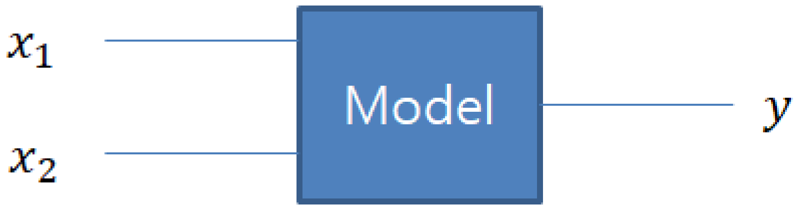


딥러닝(Deep Learning)의 기본을 이루는 뉴런

. Hinton 교수가 CIFAR의 도움을 얻어 2006년 BackPropagation 문제 해결 논문 발표
"뉴럴 네트워크의 초기값이 잘 정해져 있다면, Layer가 많은 뉴럴 네트워크라고 할지라도 잘 학습된다
. 부정적인 이미지 NN이라는 단어 대신 딥러닝이라는 이름으로 Rebranding한 결과 딥러닝 탄생

초기 Deep Learning

1
.AND 연산과 OR연산은 Linear모델로 가능
.XOR불가능 => MIT AI Lab의 Marvin Minsky교수 증명

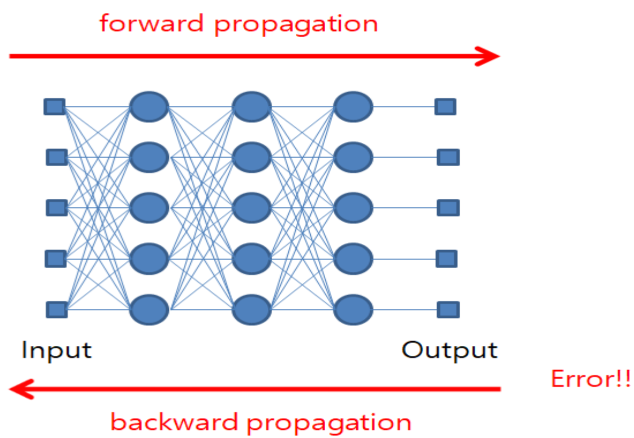


X1과 X2를 입력 받아 Y출력하는 일종의 논리 게이트

2
하나 이상의 레이어(Layer)를 갖는 MLP
(Multi Layer Perceptron)이용 해결

3
.1974,1982년 Paul Werbos, Hinton
Backpropagation 알고리즘 고안

학습 불가능

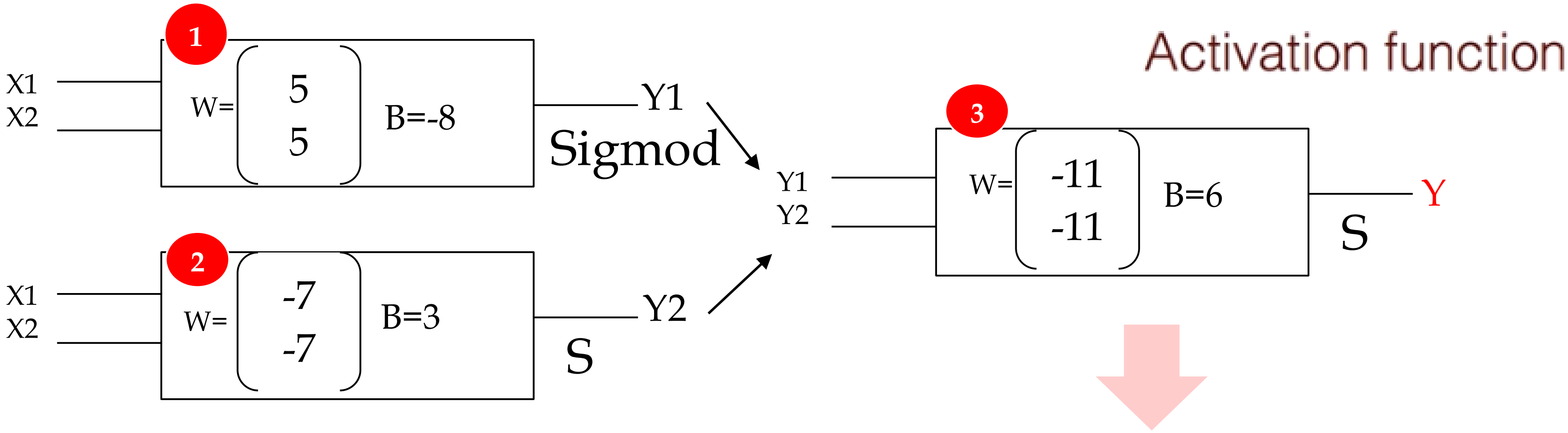


4
FP예측한 결과 값이 틀린 경우, 에러를 다시
반대 방향으로 전파시켜가면서 가중치W 값을
보정(layer가 많을 수록 input전파 안됨)

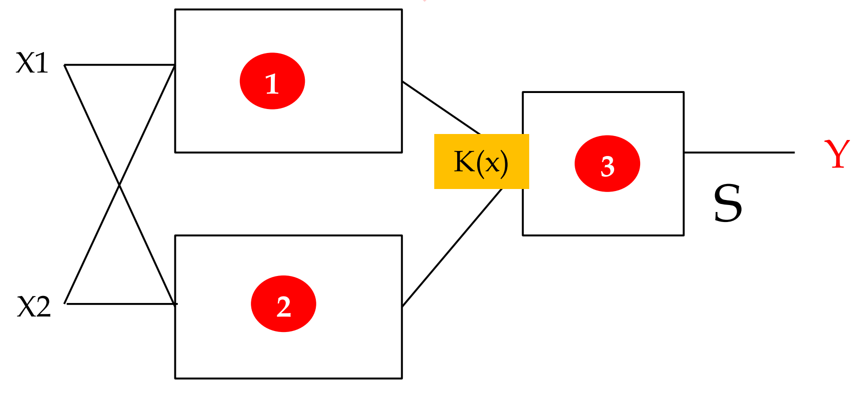
LeCun 교수님 고양이 실험 특정 뉴런만 반응
Convolutional Neural Network
(이미지 부분 잘라서 입력, 나중에 다시 합치는 방법)

5 딥러닝 탄생

Neural Network(XOR)



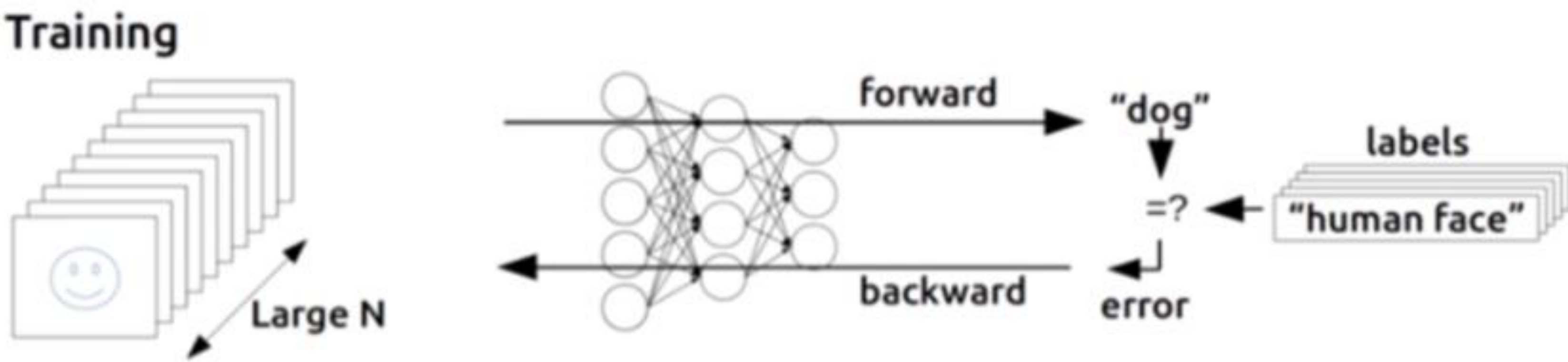
X1	X2	XOR	Y1	Y2	Y
0	0	0 (-)	0(-8)	1(3)	0(-5)
0	1	1 (+)	0(-3)	0(-4)	1(6)
1	0	1 (-)	0(-3)	0(-4)	1(6)
1	1	0 (+)	1(2)	0(-11)	0(-5)



Forward Propagation

```
# NN
K = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(K, W2) + b2)
```

BackPropagation(편미분 Chain Rule)



출처 : https://devblogs.nvidia.com/parallelforall/inference-next-step-gpu-accelerated-deep-learning

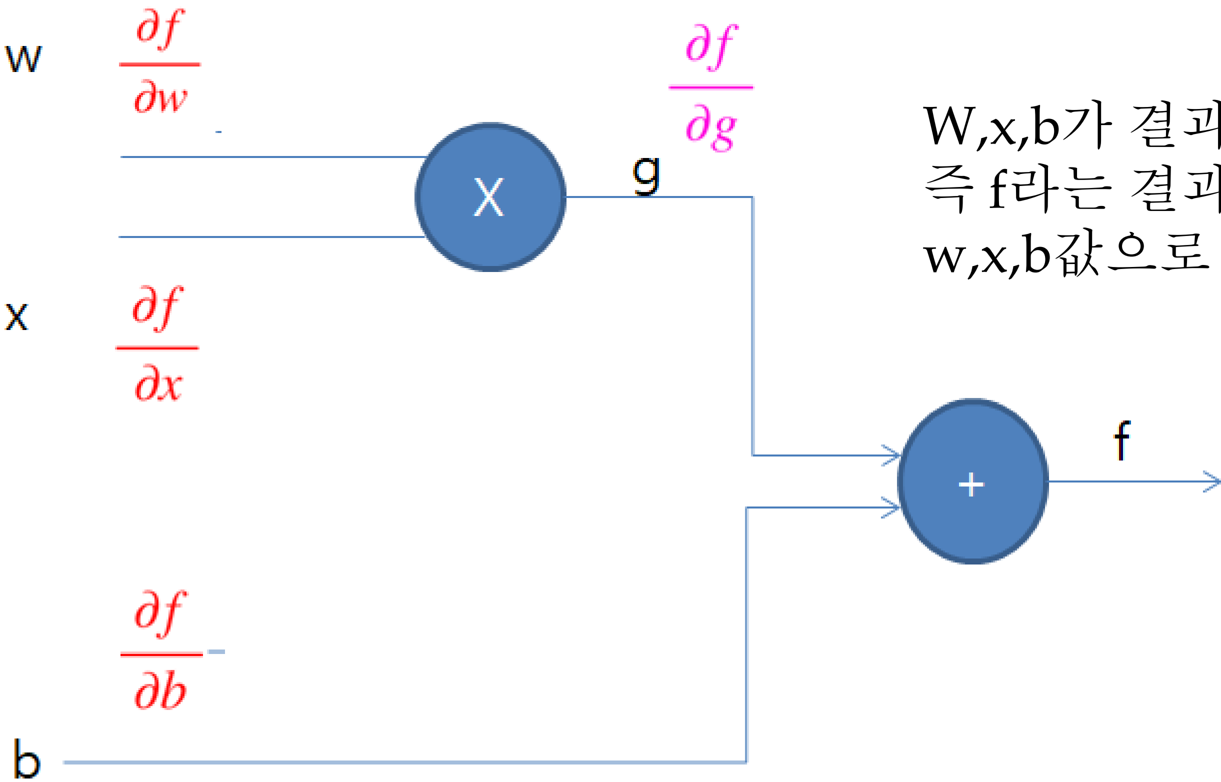
.여러 레이어로 구성된 신경망을 학습하는 딥러닝은 Backprogation.가장 중요한 법칙은 편미분의 Chain Rule

$$f(g(x)),$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial x}$$

BackPropagation(편미분 Chain Rule)

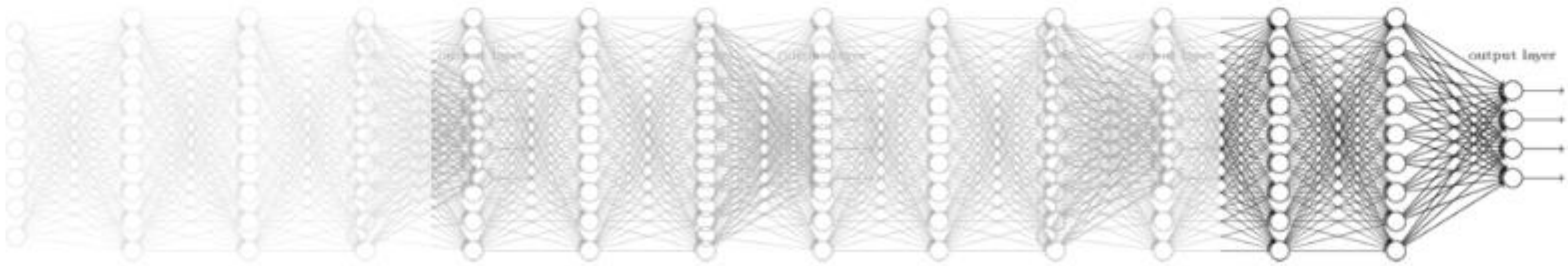
$$f = wx + b, \qquad g = wx, \qquad f = g + b$$



W,x,b가 결과값이 f에 미치는 영향도
즉 f라는 결과값을 각 입력값인
w,x,b값으로 미분한 값을 알아야 한다.

Backpropagation

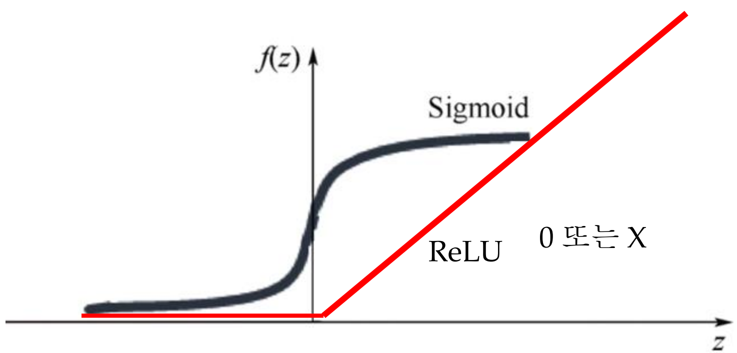
Vanishing gradient 경사 기울기 사라짐)



Sigmoid!

1보다 작아서 output이
너무 작아진다.

ReLU: Rectified Linear Unit



L1 = tf.sigmoid(tf.matmul(X, W1) + b1)

L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

Neural Network

```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

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)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).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([W1, W2]))

    # 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)
```

NN for MNIST

```python
import tensorflow as tf
import random
# import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3
```

NN for MNIST

```python
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())
# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
print('Learning Finished!')
# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels}))
```

NN for MNIST

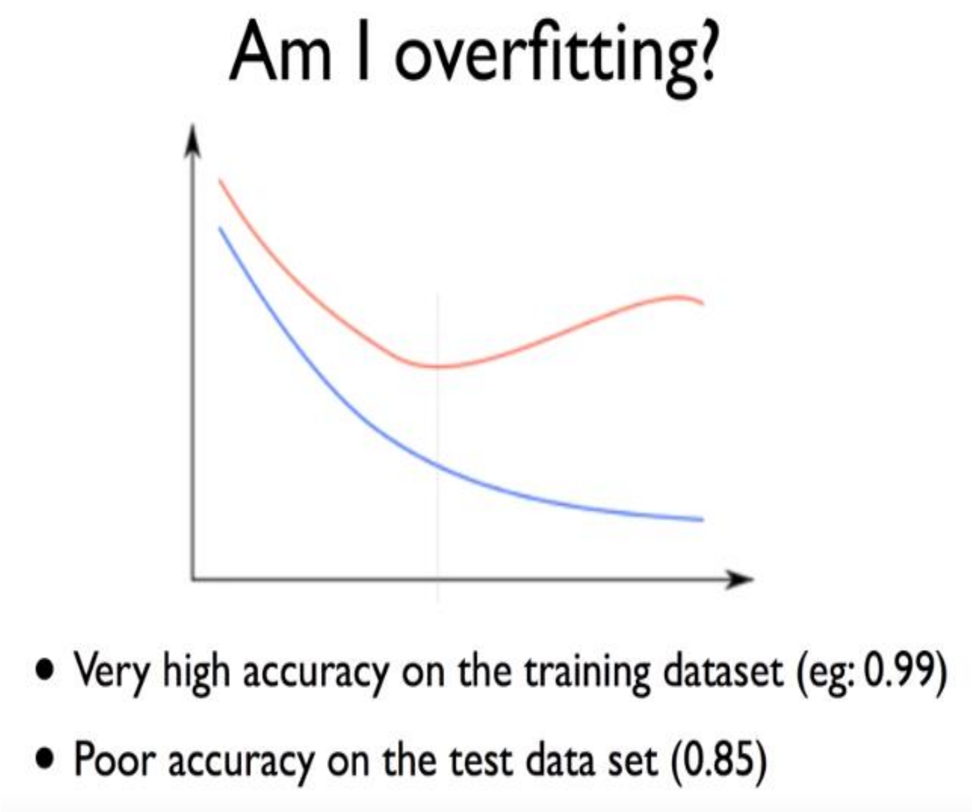
```python
# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

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

NN for MNIST

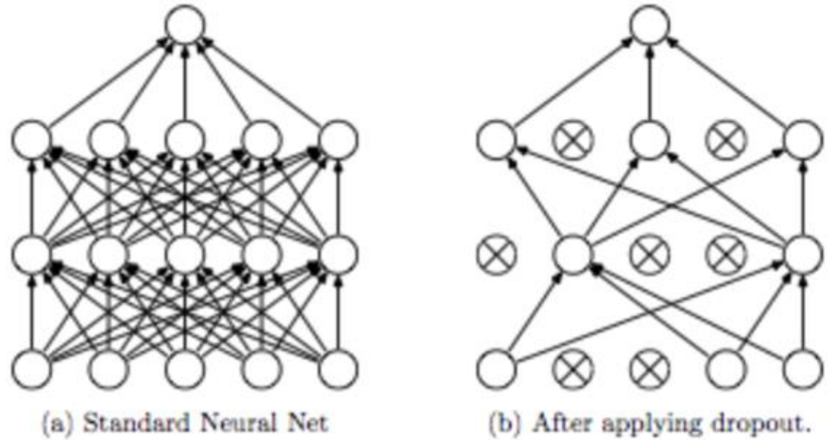
```python
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
  logits=hypothesis, labels=Y))
optimizer =
tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
Epoch: 0008 cost = 5.189753455
Epoch: 0009 cost = 3.917619447
Epoch: 0010 cost = 2.949852954
Epoch: 0011 cost = 2.278957298
Epoch: 0012 cost = 1.682389740
Epoch: 0013 cost = 1.177988671
Epoch: 0014 cost = 1.107972809
Epoch: 0015 cost = 0.806155598
Learning Finished!
Accuracy: 0.9464
Label:  [7]
Prediction:  [7]
```

overfitting



Droput : A Simple Way to Prevent Neural Networks from Overfitting(Srivastava et al.2014)



Forces the network to have a redundant representation.

Dropout

```
dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L1 = tf.nn.dropout(_L1, dropout_rate)
```

```
TRAIN:
  sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys,
  dropout_rate: 0.7})


EVALUATION:
  print "Accuracy:", accuracy.eval({X: mnist.test.images, Y:
  mnist.test.labels, dropout_rate: 1})
```

Optimizers

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

- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer

```
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                                                  logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

https://www.tensorflow.org/api_guides/python/train

Xavier for MNIST

```python
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581
W1 = tf.get_variable("W1", shape=[784, 256],
                     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[256, 256],
                     initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[256, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
  logits=hypothesis, labels=Y))
optimizer =
tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
Epoch: 0007 cost = 0.025337304
Epoch: 0008 cost = 0.020022087
Epoch: 0009 cost = 0.018914942
Epoch: 0010 cost = 0.016427606
Epoch: 0011 cost = 0.012441785
Epoch: 0012 cost = 0.013542771
Epoch: 0013 cost = 0.009965794
Epoch: 0014 cost = 0.008924985
Epoch: 0015 cost = 0.009013314
Learning Finished!
Accuracy: 0.9752
Label:  [0]
Prediction:  [0]
```

Deep NN for MNIST

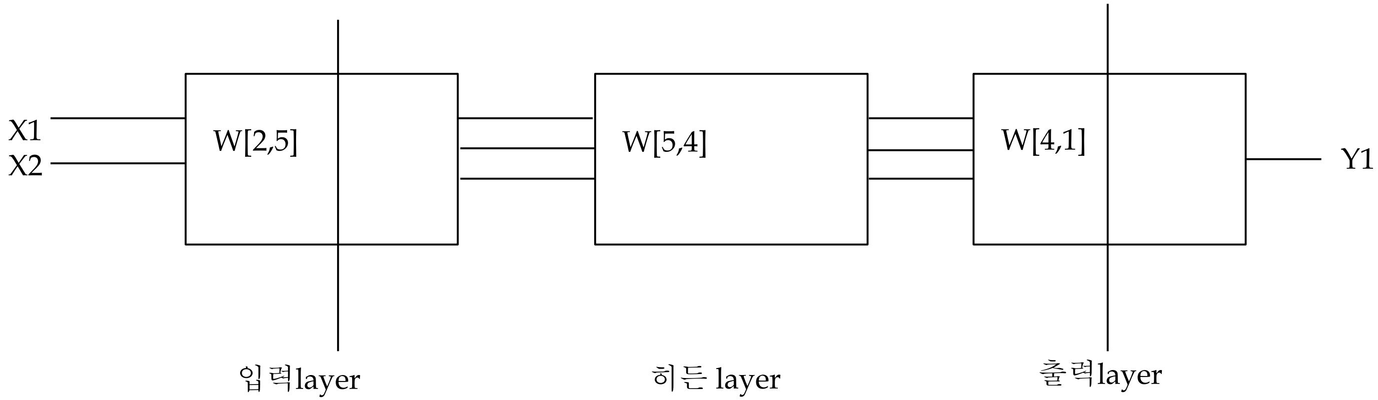
```python
W1 = tf.get_variable("W1", shape=[784, 512],
    initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)

W4 = tf.get_variable("W4", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)

W5 = tf.get_variable("W5", shape=[512, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
Epoch: 0008 cost = 0.026992815
Epoch: 0009 cost = 0.020625100
Epoch: 0010 cost = 0.021516134
Epoch: 0011 cost = 0.020800157
Epoch: 0012 cost = 0.018889848
Epoch: 0013 cost = 0.015781448
Epoch: 0014 cost = 0.015059421
Epoch: 0015 cost = 0.015016575
Learning Finished!
Accuracy: 0.979
Label:  [4]
Prediction:  [4]

Process finished with exit code 0
```


MEDICISOFT

Deep & Wide

```
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```

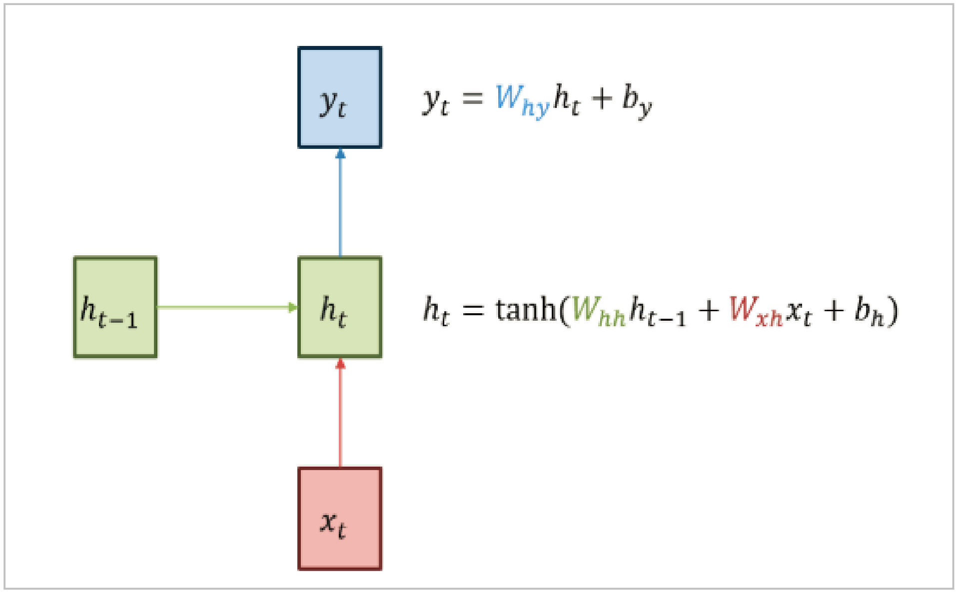


Dropout for MNIST

```
# dropout (keep_prob) rate  0.7 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

W1 = tf.get_variable("W1", shape=[784, 512])
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

W2 = tf.get_variable("W2", shape=[512, 512])
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
…
# train my model
for epoch in range(training_epochs):
    ...
  for i in range(total_batch):
      batch_xs, batch_ys = mnist.train.next_batch(batch_size)
      feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
      c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
      avg_cost += c / total_batch

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
    X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
```

RNN(Recurrent Neural Network)

. Hidden Node가 방향을 가진 엣지로 연결돼 순환구조를 이루는(Directed Cycle) 인공신경망의 한 종류
. 음성, 문자 등 순차적으로 등장하는 데이터에 대한 처리에 적합한 모델
. Sequence 길이에 관계없이 input과 output을 받아들일 수 있는 네트워크 구조



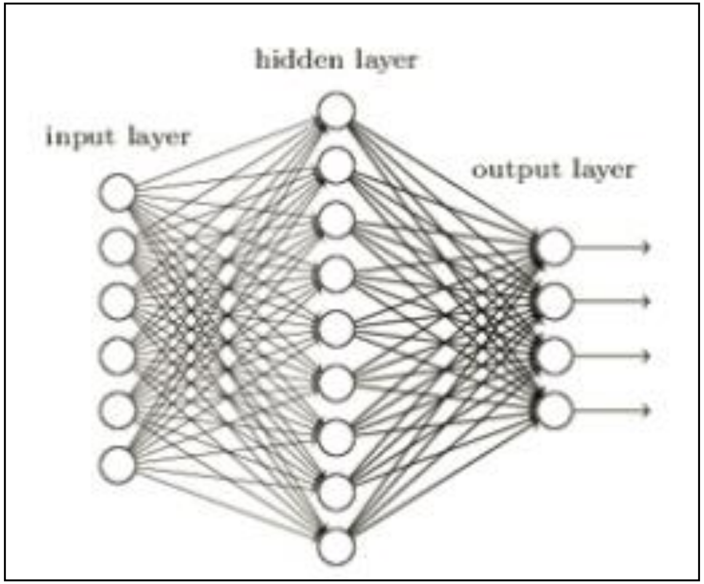
.그림 RNN의 구조

. 녹색 박스: hidden state
. 빨간 박스: input x
. 파란 박스: output y

- 현재 상태의 hidden state ht는 직전 시점의 hidden state ht-1를 받아서 갱신
- 현재 상태의 output Yt는 Ht를 전달받아 갱신되는 구조

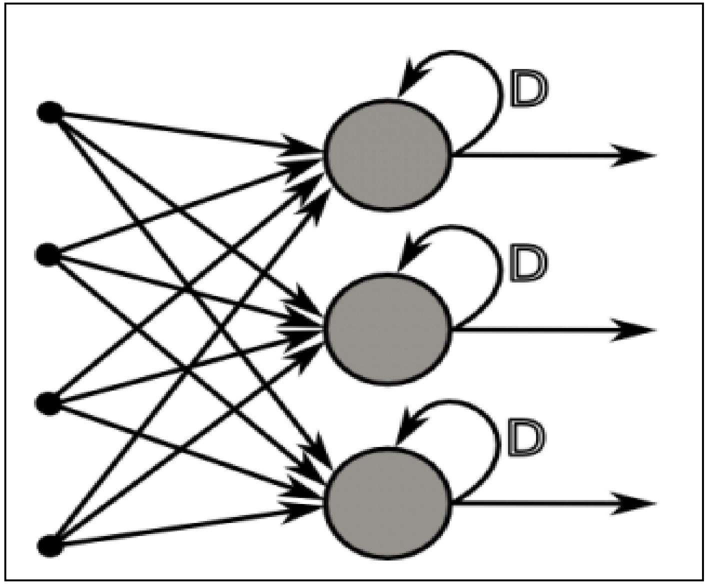
※ hidden state의 활성함수(activation function)은 비선형 함수인 하이퍼볼릭탄젠트(tanh)이다.

RNN(Recurrent Neural Network)



[그림] 기존의 신경망 구조

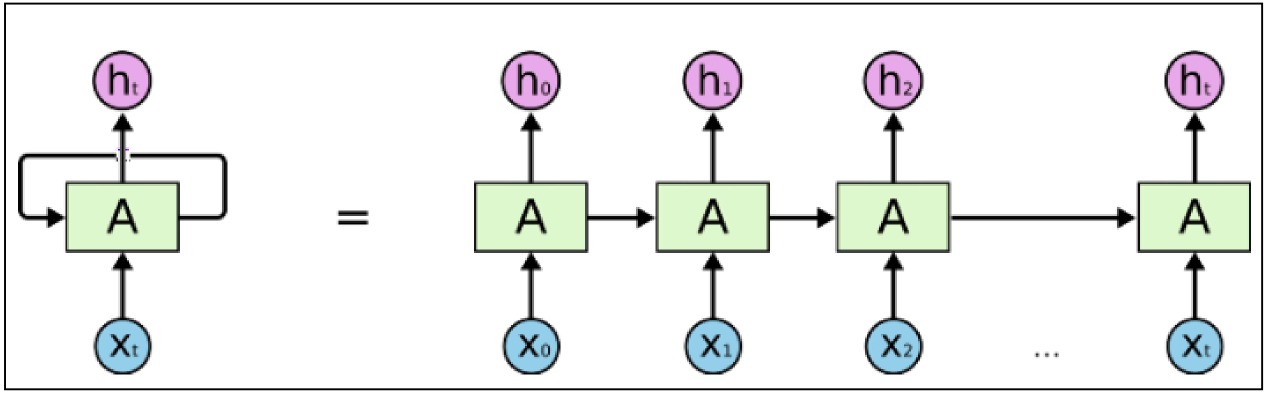
VS



[그림] RNN 신경망 구조

Sequence Data

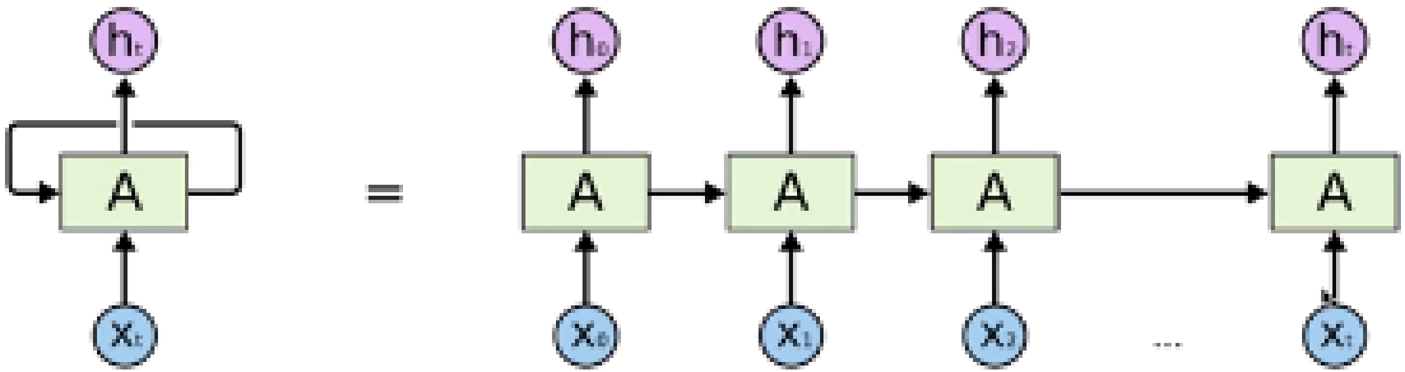
. We don’t understand one word only
. We understand based on the previous words + this word(time series)
. NN/CNN cannot do this



[그림] RNN layer 한 개의 모양

(Vanilla)Recurrent Neural Network

Character-level language model example
Vocabulary:[h,e,l,o]
Example training sequence: “hello”



Input layer:	1 0 0 0	0 1 0 0	0 0 1 0	0 0 1 0
Input chars:	“h”	“e”	“l”	“l”