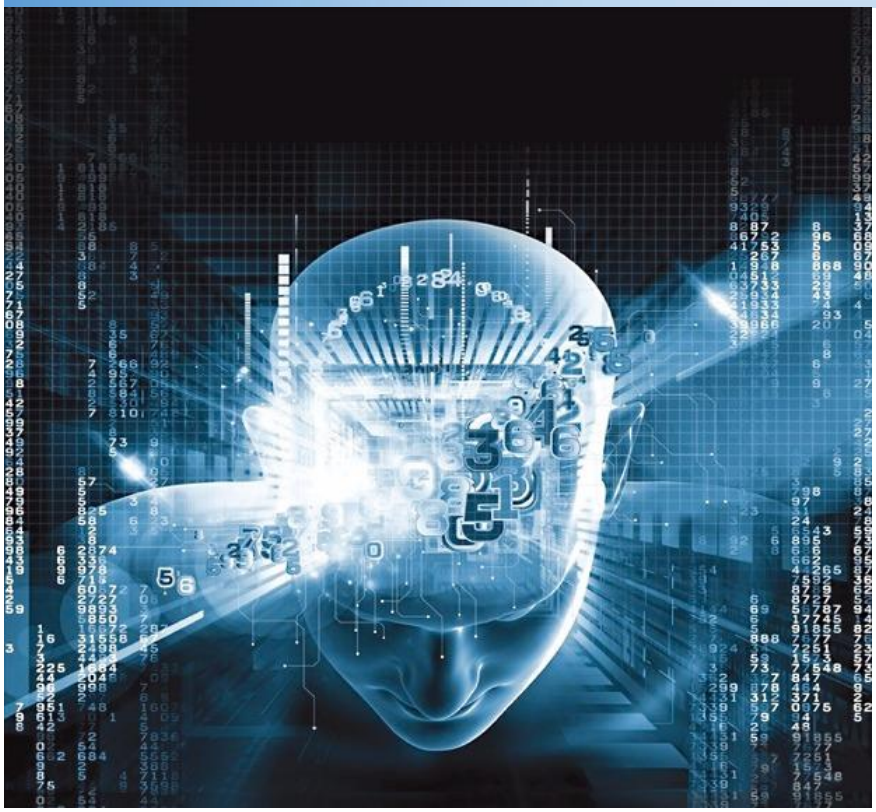
기계학습
텐서플로우 & MNIST



안현식
동명대학교

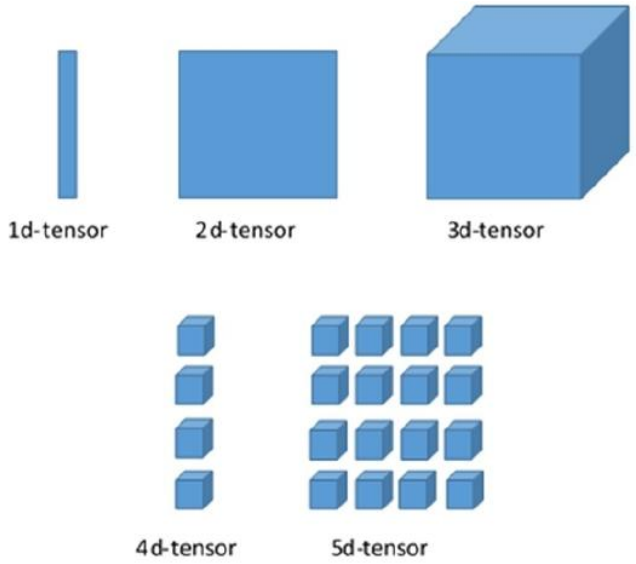
텐서플로우-TensorFlow

텐서플로우란?

- 텐서플로우는 딥러닝을 위해 구글에서 제공하는 프레임워크
- 누구나 사용할수 있는 머신러닝 오픈소스 라이브러리

Tensor?

- Tensor = Multidimensional Arrays = Data
- 딥러닝에서 텐서는 다차원 배열로 나타내는 데이터
- 예를 들어, RGB 이미지는 삼차원 배열로 나타나는 텐서



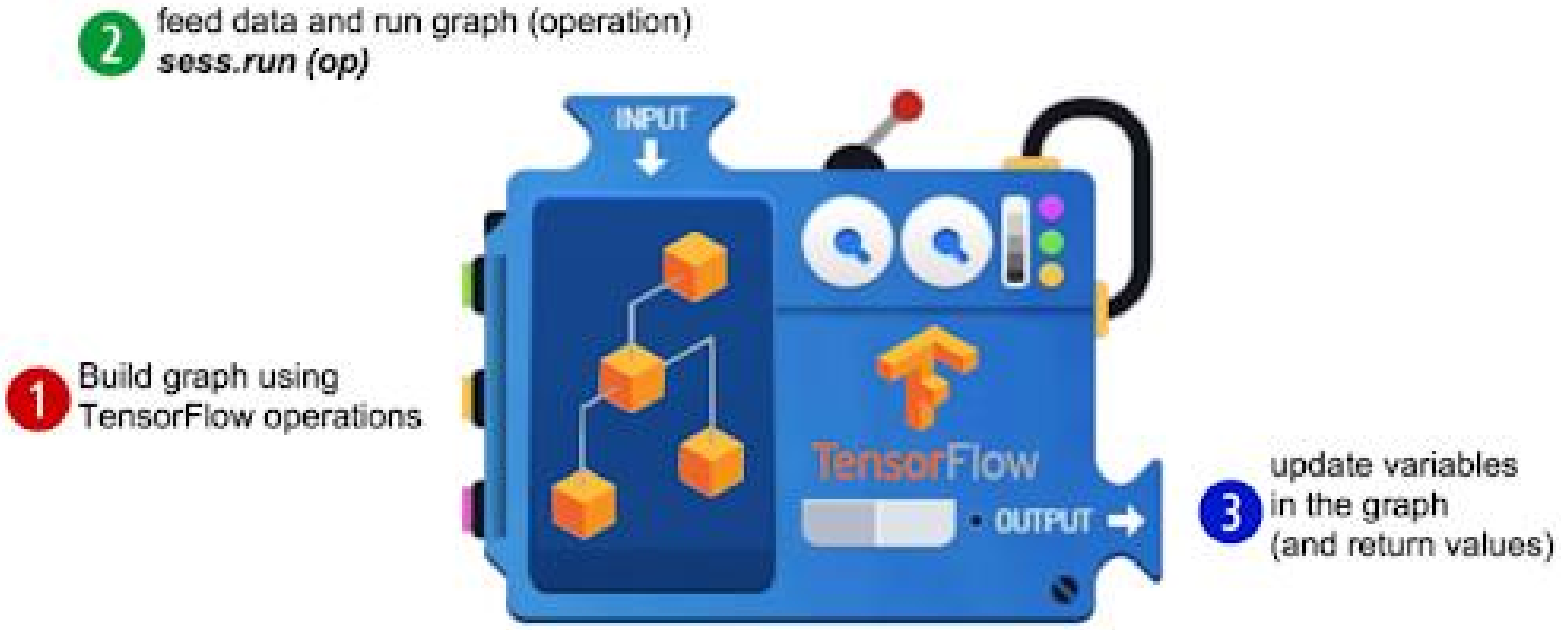
텐서플로우-TensorFlow

Flow?

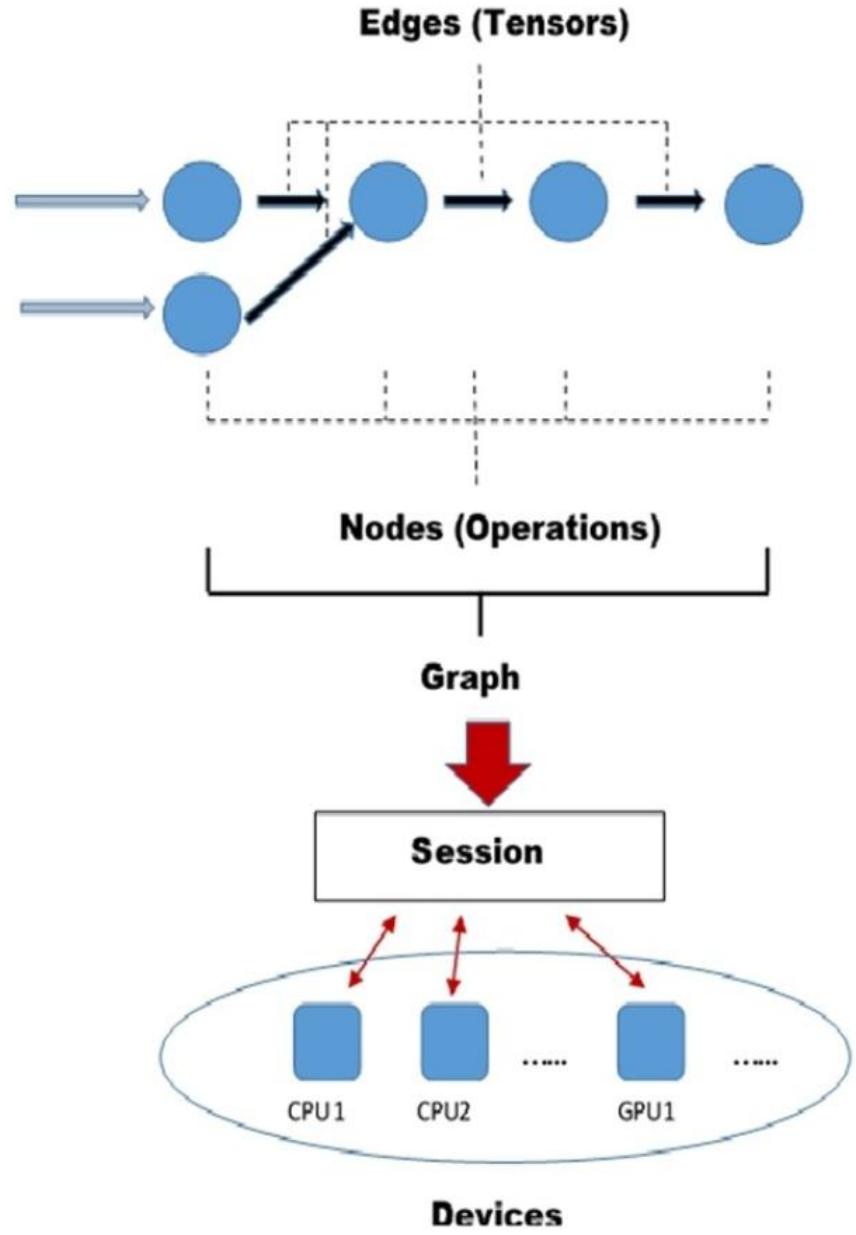
- 플로는 데이터의 흐름
- 텐서플로우에서 계산은 데이터 플로우 그래프(dataflow graph)로 실행
- 그래프 각각의 노드는 수식(operations)을 의미하며 그래프의 간선(edge)는 시스템을 따라 흘러가는 데이터(Tensor)를 의미
- 그래프를 따라 데이터가 노드를 거쳐 흘러가면서 계산을 수행

텐서플로우 처리

1) 그래프를 만든다.
2) Session을 열고 sess.run()으로 그래프를 실행시킨다



텐서플로우 처리


Edges (Tensors)
Nodes (Operations)
Graph
Session
CPU1
CPU2
GPU1
Devices

텐서

- TensorFlow 프로그램은 모든 데이터를 tensor 데이터 구조를 사용해서 표현
- TensorFlow의 tensor는 n-차원 배열 또는 리스트라고 할 수 있음
- 하나의 tensor는 정적 타입과 동적 차원이 있고 그래프의 노드들은 오직 tensor만을 전달

- Rank : 차원 수

Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	`....`

텐서

- Shape : 배열 구조

- Shape는 Python 리스트 / 정수형 튜플 또는 TensorShape class로 표현

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

텐서

- Data types

Data type	Python type	Description
DT_FLOAT	tf.float32	32 비트 부동 소수.
DT_DOUBLE	tf.float64	64 비트 부동 소수.
DT_INT8	tf.int8	8 비트 부호 있는 정수.
DT_INT16	tf.int16	16 비트 부호 있는 정수.
DT_INT32	tf.int32	32 비트 부호 있는 정수.
DT_INT64	tf.int64	64 비트 부호 있는 정수.
DT_UINT8	tf.uint8	8 비트 부호 없는 정수.
DT_STRING	tf.string	가변 길이 바이트 배열. Tensor의 각 원소는 바이트 배열.
DT_BOOL	tf.bool	불리언.
DT_COMPLEX64	tf.complex64	2개의 32 비트 부동 소수로 만든 복소수 : 실수부 + 허수부
DT_COMPLEX128	tf.complex128	2개의 64 비트 부동 소수로 만든 복소수 : 실수부 + 허수부
DT_QINT8	tf.qint8	8 비트 부호 있는 정수로 quantized Ops에서 사용.
DT_QINT32	tf.qint32	32 비트 부호 있는 정수로 quantized Ops에서 사용.
DT_QUINT8	tf.quint8	8 비트 부호 없는 정수로 quantized Ops에서 사용.

텐서플로우 처리순서

- 텐서플로우는 그래프 빌드(building)와 실행(session)에 의해 처리

```
import tensorflow as tf
x = tf.constant(1,name='x')
y = tf.Variable(x+5,name='y')
model = tf.global_variables_initializer()
with tf.Session() as session:
    session.run(model)
    print(session.run(y))
```

1. tensorflow 모듈을 가져 와서 tf 호출
2. x라는 상수 값을 만들고 숫자 값 1을 지정
3. y라는 변수를 만들고 방정식 x + 5로 정의
4. global_variables_initializer로 변수 초기화
5. 값을 계산하기 위한 세션 만들기
6. 4에서 만든 모델 실행
7. 변수 y 만 실행하고 현재 값을 출력

텐서플로우 처리순서

```
import tensorflow as tf
a = tf.constant(2)
b = tf.constant(3)

# Launch the default graph.
with tf.Session() as sess:
    print("a=2, b=3")
    print("Addition: %i" % sess.run(a+b))
    print("Multiplication: {}".format(sess.run(a*b)))
```

텐서플로우 기초

Placeholder

함수에서 파라미터를 미리 placeholder 로 type 지정해줌

실행하는 시점에서 파라미터의 값을 지정해줄 수 있음

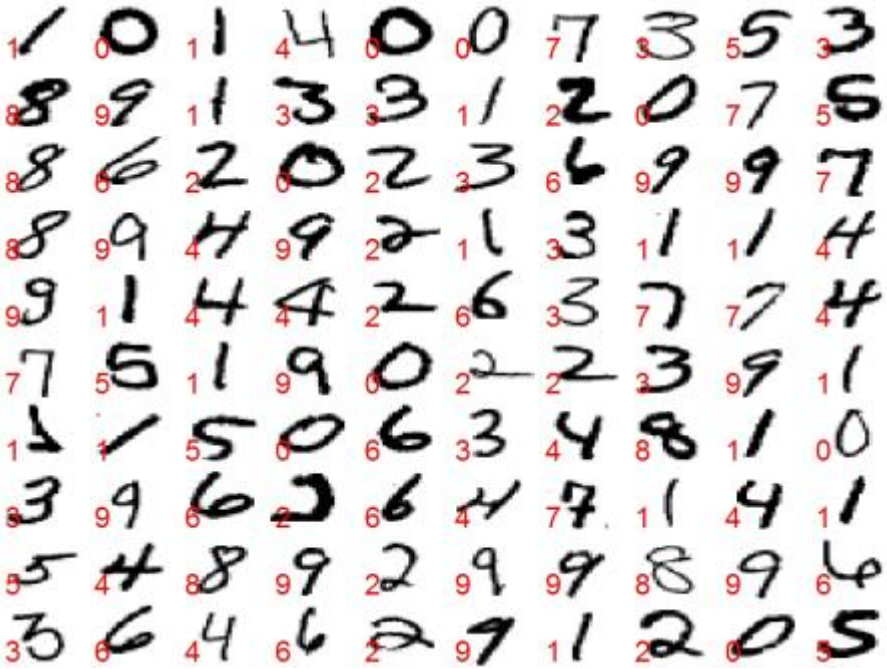
```
import tensorflow as tf
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)
# Define some operations
add = tf.add(a, b)
mul = tf.multiply(a, b)
# Launch the default graph.
with tf.Session() as sess:
    # Run every operations with variable input
    print("Addition : %i" % sess.run(add, feed_dict={a: 2, b: 3}))
    print("Muliplication : {}".format(sess.run(mul, feed_dict={a: 2, b: 3})))
```

```
placeholder의 전달 파라미터.
placeholder(
    dtype,
    shape=None,
    name=None
)
dtype : 데이터 타입을 의미하며 반드시 입력
shape : 입력 데이터의 형태로서 상수 값, 다차원 배열 등 ( 디폴트 None 지정 )
name : 해당 placeholder의 이름 ( 디폴트 None 지정 )
```

```
a=2, b=3
Addition: 5
Multiplication: 6
```

MNIST 데이터셋

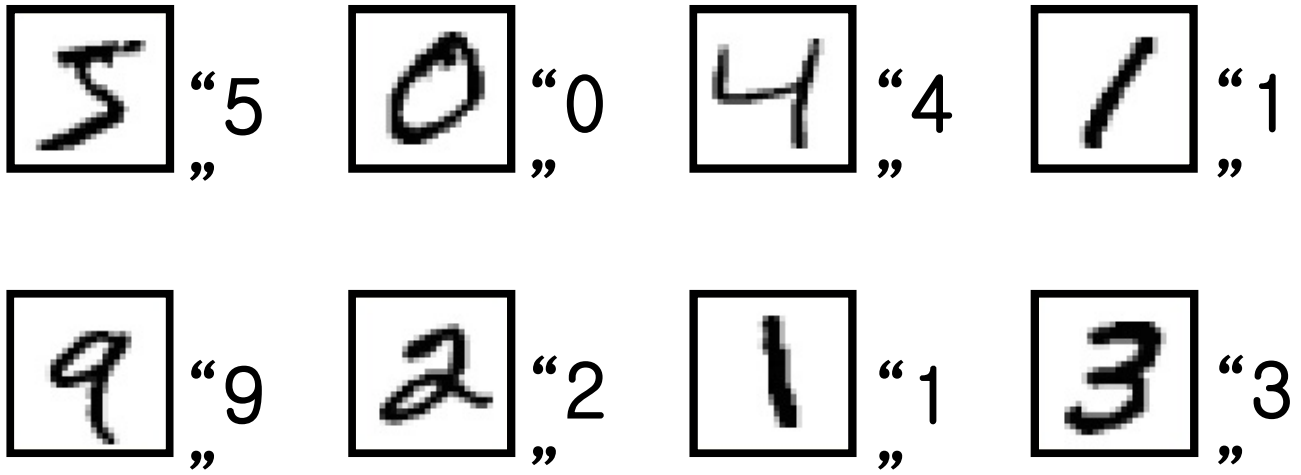
- MNIST 데이터베이스는 미국표준국(NIST)에서 수집한 필기 숫자 데이터베이스로, 훈련집합 60,000자, 테스트집합 10,000자를 제공한다. http://yann.lecun.com/exdb/mnist에 접속하면 무료로 내려받을 수 있으며, 1988년부터 시작한 인식률 경쟁 기록도 볼 수 있다. 2017년 8월 기준으로는 [Ciresan2012] 논문이 0.23%의 오류율로 최고 자리를 차지하고 있다. 테스트집합에 있는 10,000개 샘플에서 단지 23개만 틀린 것이다.



MNIST 데이터셋

Training Data

- Preparing training data: images and their labels

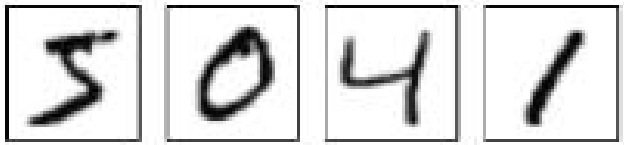


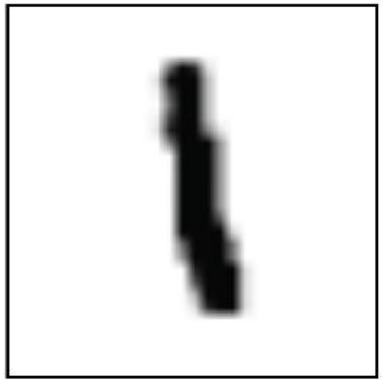
Using the training data to find the network parameters.

MNIST 데이터셋

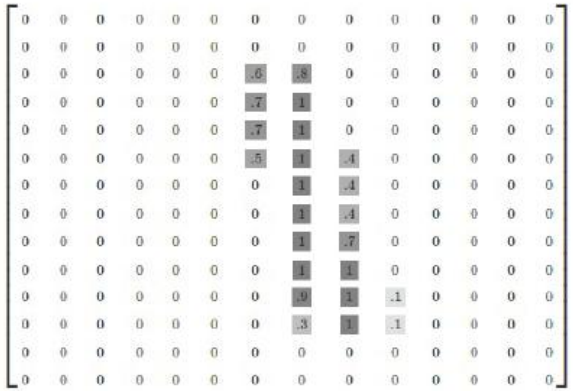
- 55,000개의 학습 데이터(mnist.train), 10,000개의 테스트 데이터

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```





≈

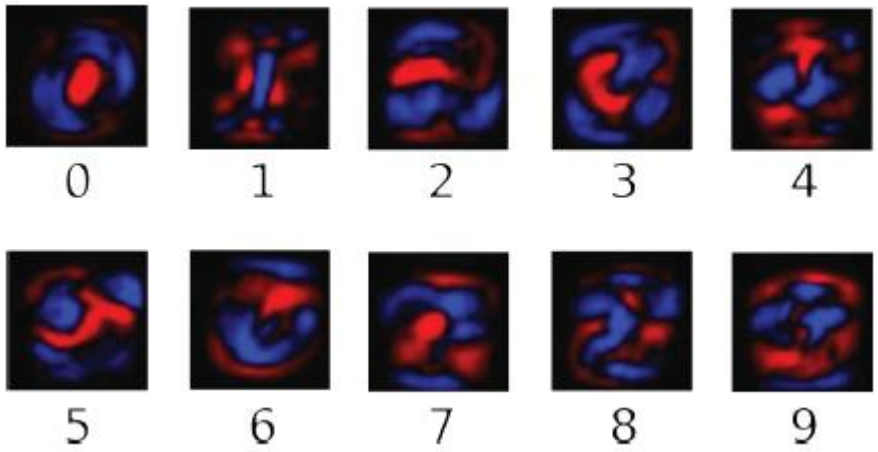


이 배열을 펼쳐서 28x28 = 784 개의 벡터 생성
mnist.train.images 는 [55000, 784] 의 형태를 가진 텐서

파일	목적
`train-images-idx3-ubyte.gz`	학습 셋 이미지 - 55000개의 트레이닝 이미지, 5000개의 검증 이미지
`train-labels-idx1-ubyte.gz`	이미지와 매칭되는 학습 셋 레이블
`t10k-images-idx3-ubyte.gz`	테스트 셋 이미지 - 10000개의 이미지
`t10k-labels-idx1-ubyte.gz`	이미지와 매칭되는 테스트 셋 레이블

소프트맥스 회귀 (softmax regression)

서로 다른 여러 항목 중 하나일 확률을 계산



학습한 가중치

$$\text{evidence}_i = \sum_j W_{i,\,j} x_j + b_i$$

$$y = \text{softmax}(\text{evidence})$$

$$\text{softmax}(x) = \text{normalize}(\exp(x))$$

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Softmax: 계산한 선형 함수를 10가지 경우에 대한 확률 분포로 변환
(입력값을 지수화한 뒤 정규화 하는 과정,
모두 합하면 1이 되는 확률 분포로 정규화)

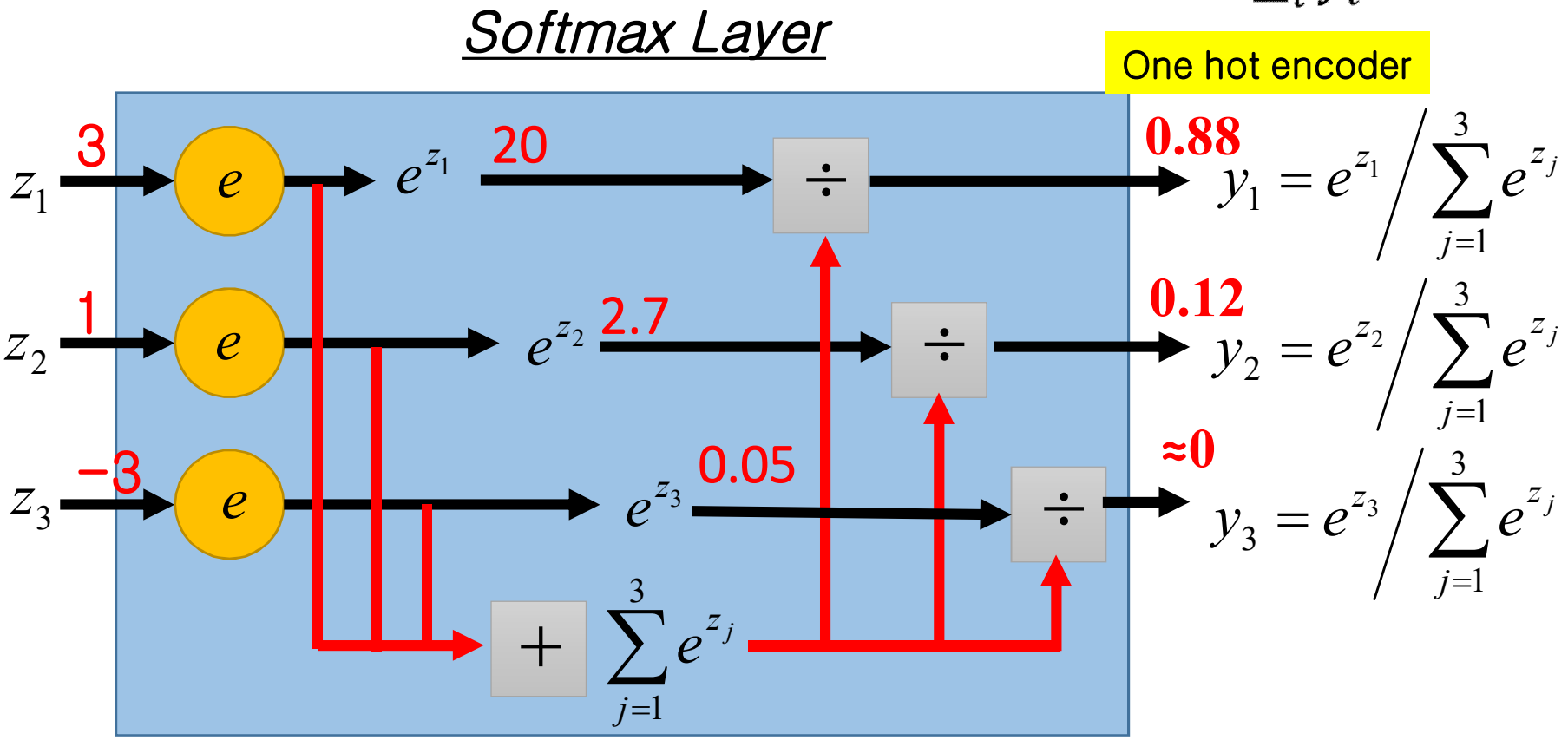
Softmax

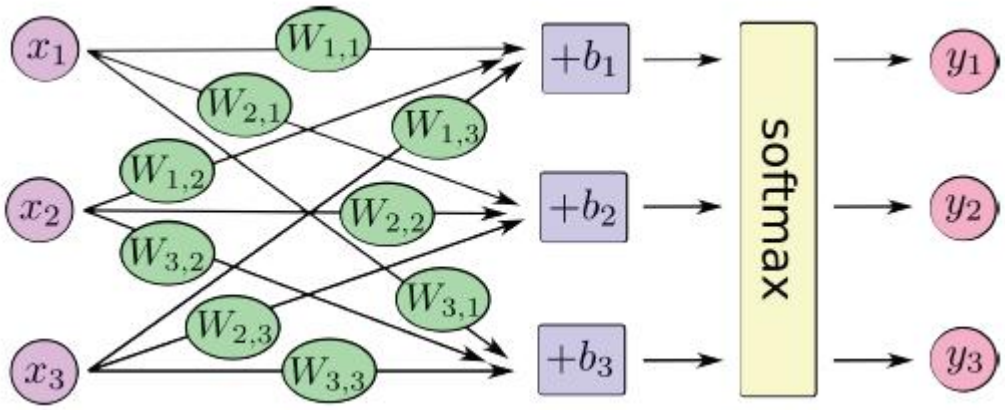
$$f(\vec{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \quad \text{for } i = 1, \ldots, K$$

- Softmax layer as the output layer

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$





$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$y = \text{softmax}(Wx + b)$$

회귀 구현하기

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x,W)+b)
```

학습-크로스 엔트로피

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

```
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y),
reduction_indices=[1]))
train = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

학습 비율 0.5로 경사 하강법(gradient descent algorithm)을 적용하여 크로스 엔트로피를 최소화

```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
```

변수들 초기화
세션에서 모델 실행

```
for i in range(1000):
batch_xs, batch_ys = mnist.train.next_batch(100)
sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

100개씩 배치로 실행

모델 평가

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

결과확인

부동소수점 값으로 변환한 후 평균 계산

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_:
mnist.test.labels}))
```

전체 데이터를 대상으로 정확도를 계산

예측

```
r = 0; # 0~9999
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(tf.argmax(y, 1), feed_dict={x:
mnist.test.images[r:r + 1]}))
print(mnist.test.images[r:r + 1])
print(mnist.test.labels[r:r + 1])
```

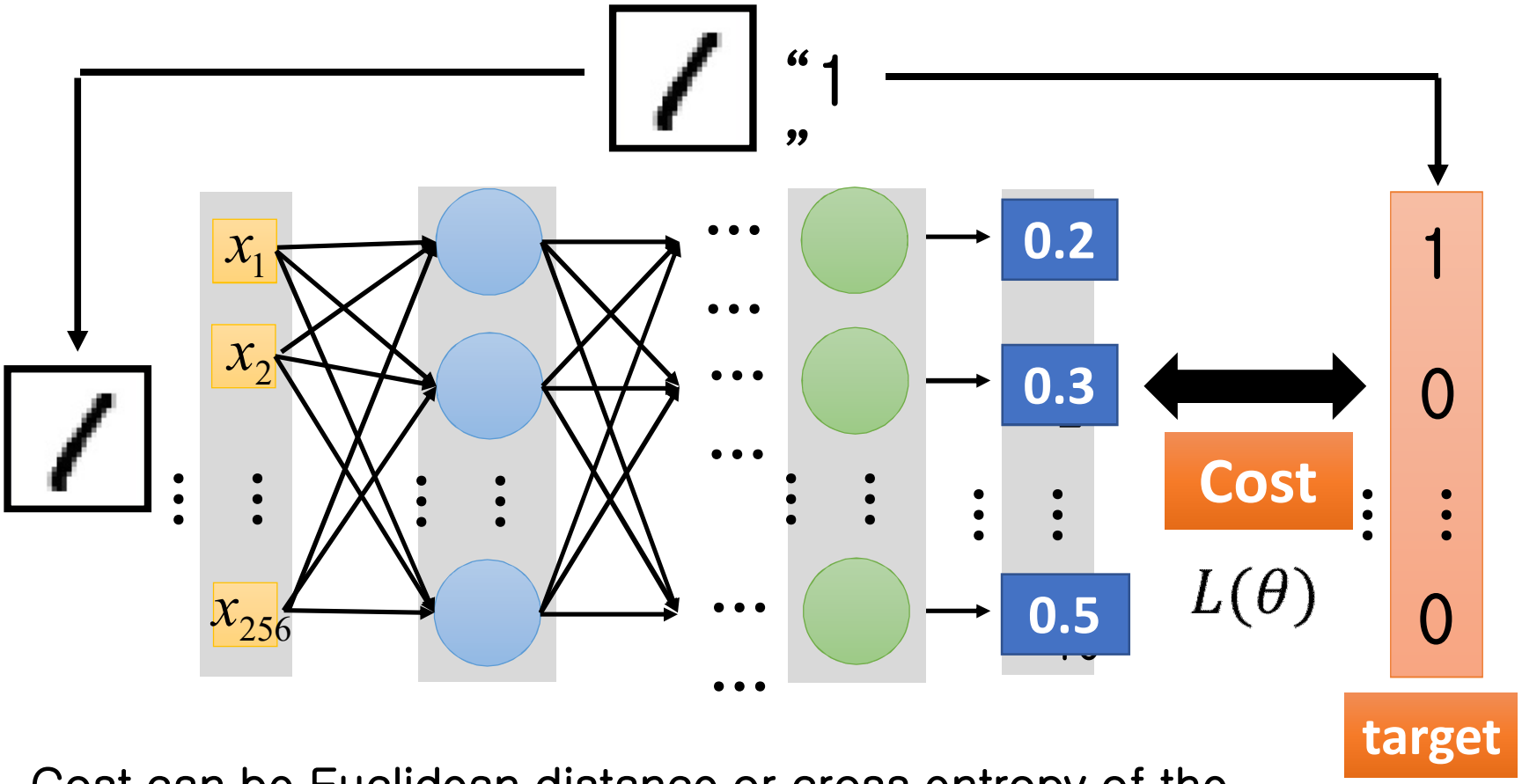
그래픽 보기

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

MNIST 다층 NN

Cost

Given a set of network parameters θ, each example has a cost value.

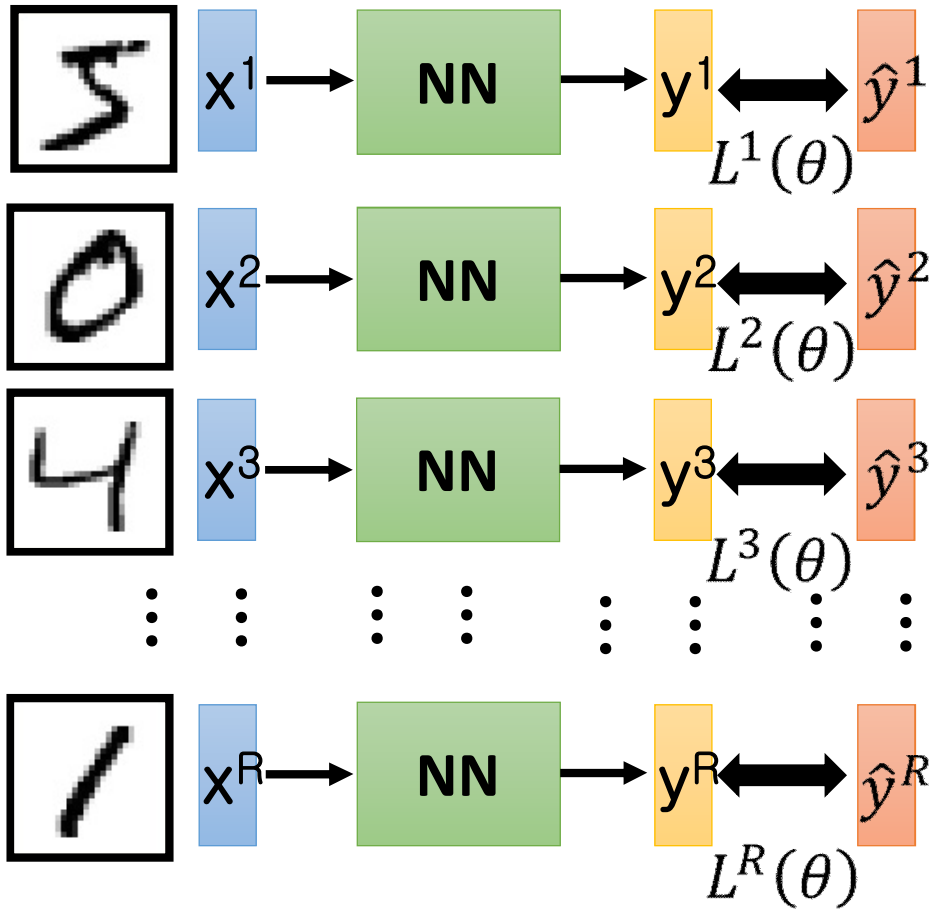


Cost can be Euclidean distance or cross entropy of the network output and target

MNIST 다층 NN

Total Cost

For all training data …



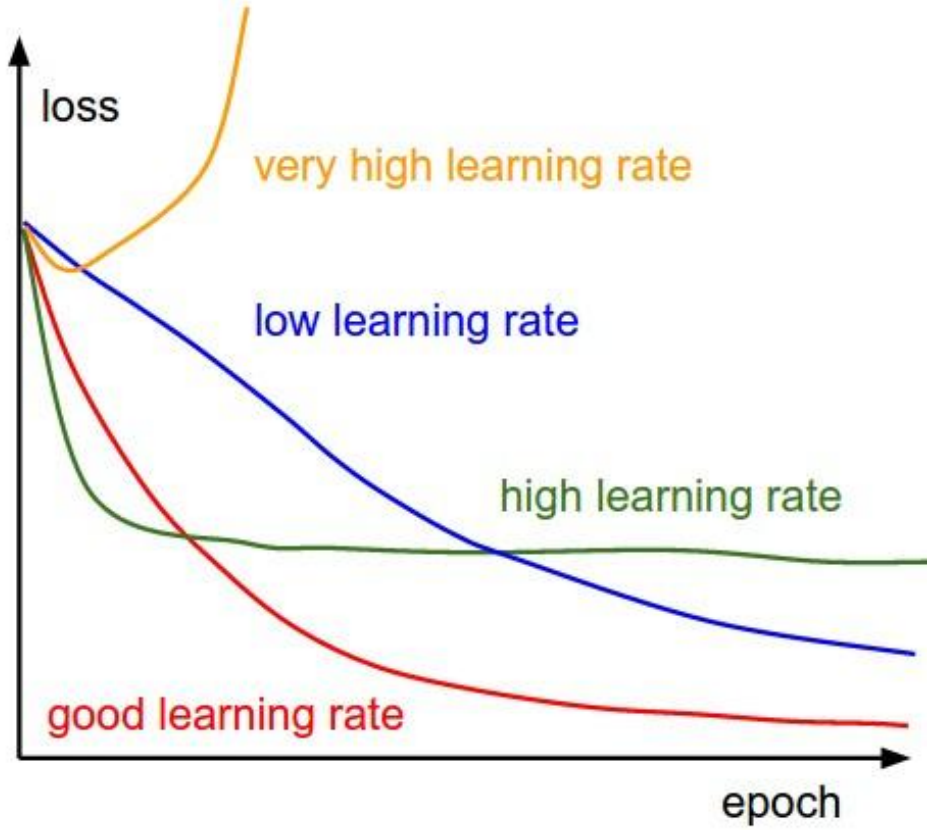
Total Cost:

$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

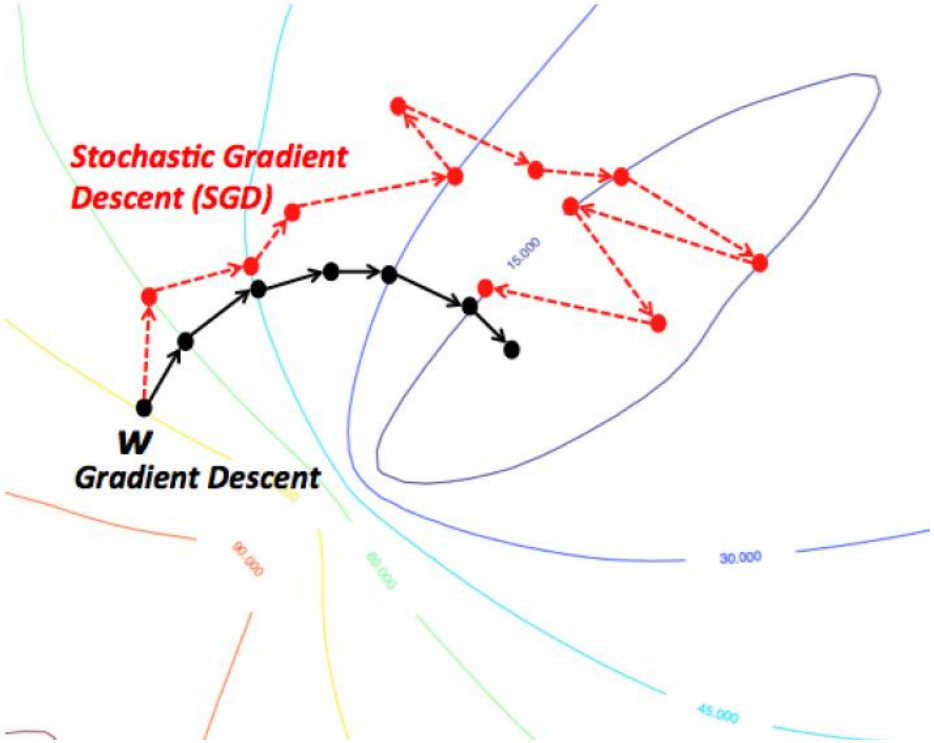
How bad the network parameters θ is on this task

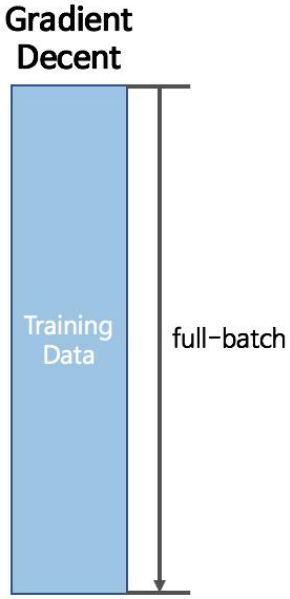
Find the network parameters θ^* that minimize this value

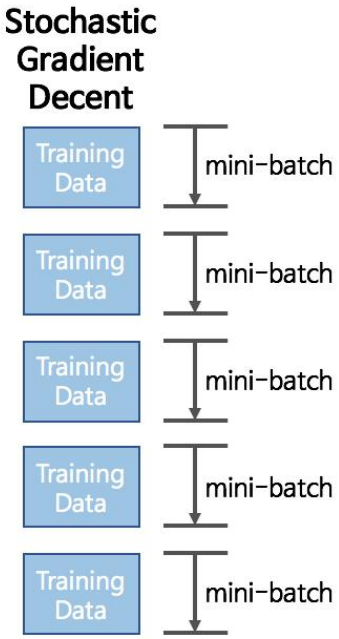
Optimizer


loss
very high learning rate
low learning rate
high learning rate
good learning rate
epoch

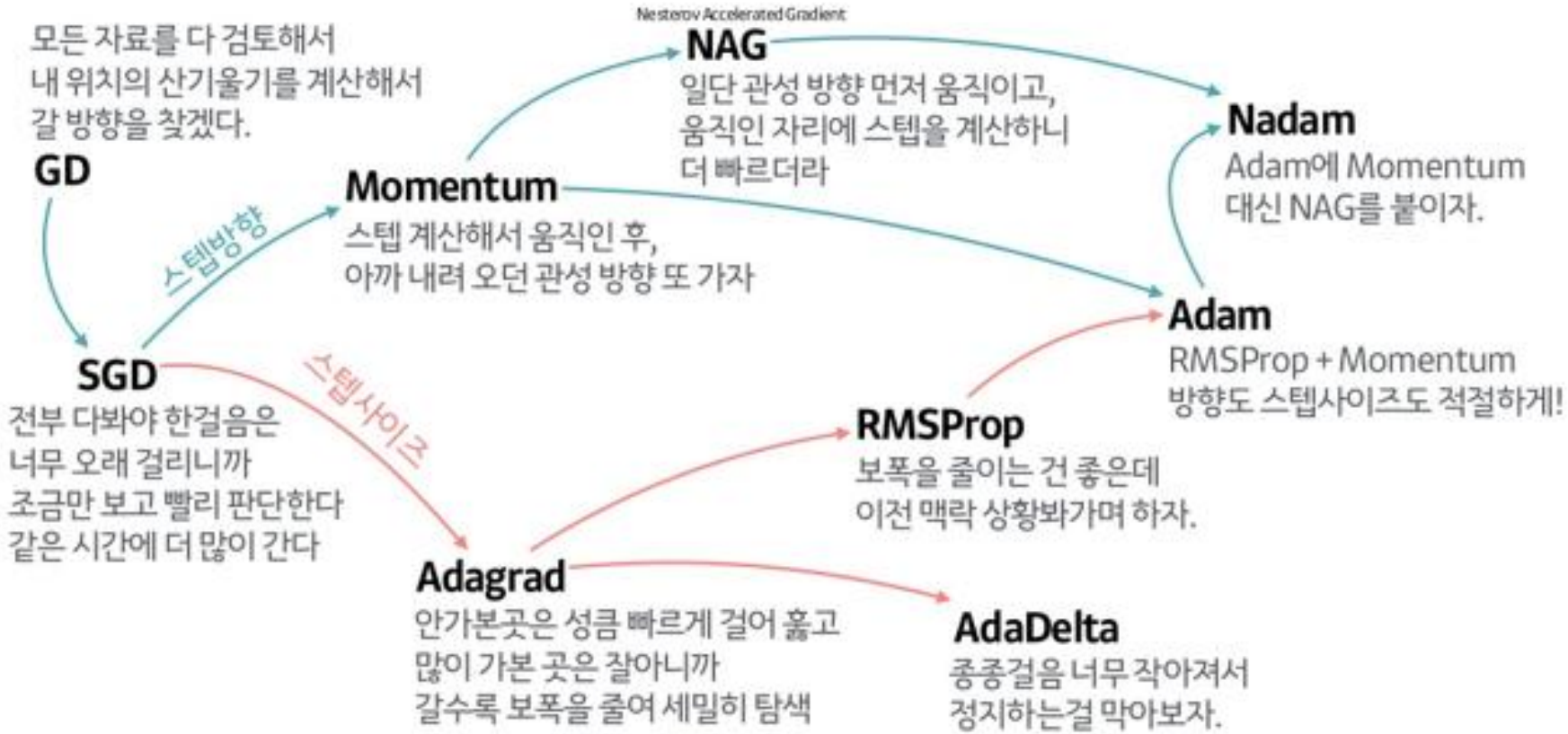
Stochastic Gradient Descent (SGD)







Optimizer



https://seamless.tistory.com/38

MNIST NN Softmax & cross entropy

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x,W)+b)
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y), reduction_indices=[1]))
train = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

for I in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train, feed_dict={x: batch_xs, y_: batch_ys})

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
r=7; # 0~9999
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(tf.argmax(y, 1), feed_dict={x: mnist.test.images[r:r + 1]}))
print(mnist.test.labels[r:r + 1])
plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

Accuracy= 0.9198

MNIST 다층 NN

https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-10-2-mnist_nn.py

```
# Lab 10 MNIST and NN
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data
tf.set_random_seed(777)  # reproducibility
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# 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
# 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)
```

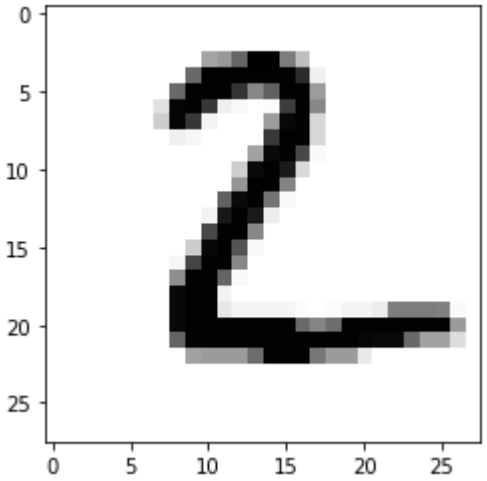
```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# 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}))

# 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]}))
```

Adam Optimizer(Adaptive Moment estimation)

Accuracy: 0.9427

MNIST 다층 NN

```
Epoch: 0001 cost = 142.398641642
Epoch: 0002 cost = 38.787511930
Epoch: 0003 cost = 24.244184681
Epoch: 0004 cost = 16.871073616
Epoch: 0005 cost = 12.309403554
Epoch: 0006 cost = 9.203625508
Epoch: 0007 cost = 6.847962012
Epoch: 0008 cost = 5.082311801
Epoch: 0009 cost = 3.821389055
Epoch: 0010 cost = 2.852849876
Epoch: 0011 cost = 2.121125133
Epoch: 0012 cost = 1.600571488
Epoch: 0013 cost = 1.218156629
Epoch: 0014 cost = 1.021714679
Epoch: 0015 cost = 0.783299278
Learning Finished!
Accuracy: 0.9427
Label:  [2]
Prediction:  [2]
```



Accuracy: 0.9427