Machine & Deep learning with Tensorflow

HyeJin Sim

August 17, 2019

Contents

1 Machine Learning

- Limitations of explicit programming
  - Spam filter : many rules
  - Automatic driving : too many rules

- Machine learning : "Field of study that gives computers the ability to learn without being explicitly programmed" Arthur Samuel (1959)

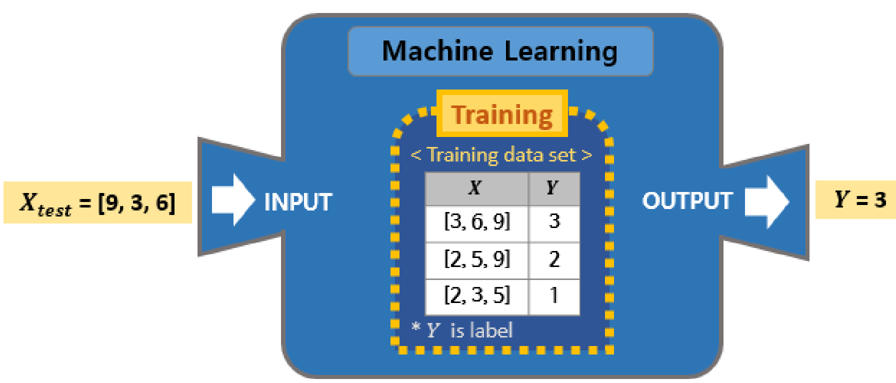
- Method of learning (Supervised and Unsupervised learning)
  1. Supervised learning : learning with labeled examples - training set
     ex) image labeling(cat, dog, car)
  2. Unsupervised learning : un-labeled data ex) Google news grouping, Word clustering
     ⇒ Can't give label because it's already grouping
     ⇒ Self learning !

2 Supervised learning

Most common problem type in Machine Learning

- Image labeling : learning from tagged images
- Email spam filter : learning from labeled (spam or ham) email
- Predicting exam score : learning form previous exam score and time spent

● Training data set



• Types of supervised learning

- Predicting final exam score based on time spent → **"Regression"**
- Pass/non-pass based on time spent → **"Binary Classification"**
- Letter grade (A,B,C,D and E) based on time spent → **"Multi-label Classification"**

3 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs.

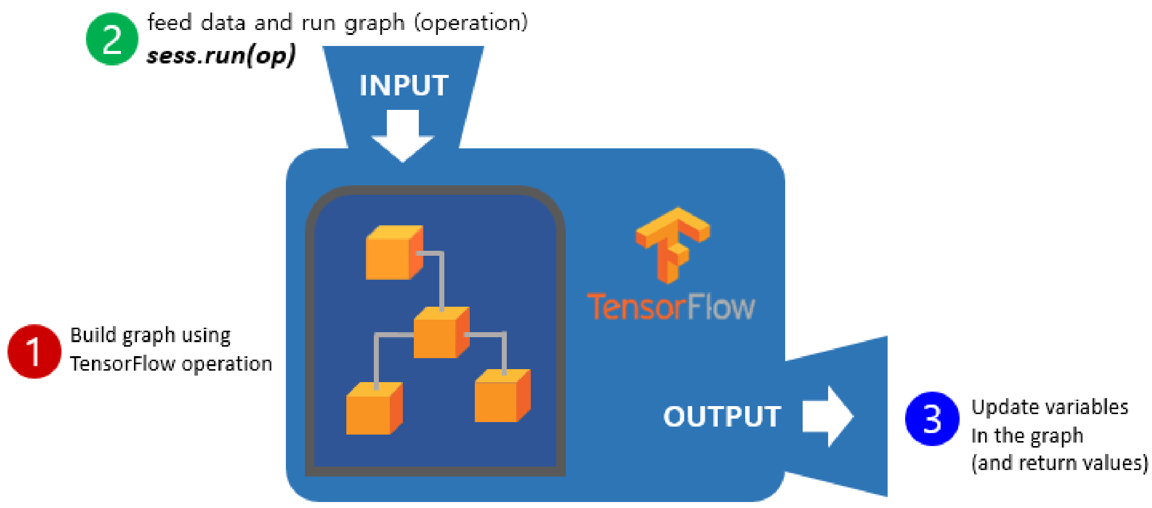
◇ What is Data Flow Grqph?

- Nodes in the graph represent mathematical operations
- Edges represent the multi-dimensional data arrays(=tensors) communicated between them

`tf._version_` : check installation and version

```
import tensorflow as tf

tf._version_
```

3.1 Computational Graph

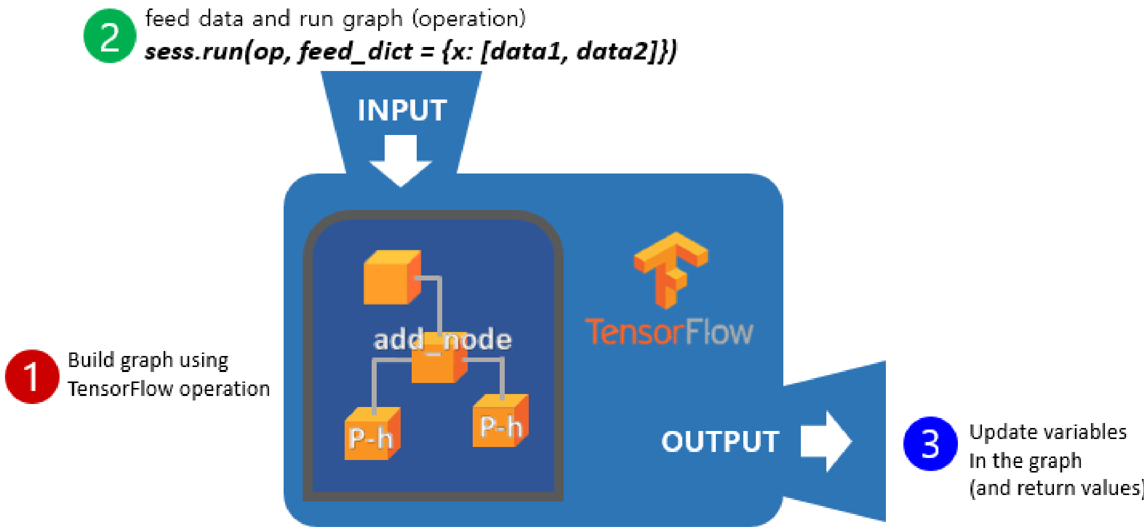


< TensorFlow Mechanics >

MLlab 01 Computational Graph

```python
node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0)   ## also tf.float32 implicitly
node3 = tf.add(node1, node2)

print("node1:", node1, "node2:", node2)
print("node3:", node3)

sess = tf.Session()
print("sess.run(node1, node2):", sess.run([node1, node2]))
print("sess.run(node3):", sess.run(node3))
```

3.2 Placeholder



< TensorFlow Mechanics >

MLlab 01 Placeholder

```python
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

add_node = a + b

print(sess.run(add_node, feed_dict={a: 3, b: 4.5}))
print(sess.run(add_node, feed_dict={a: [1, 3], b: [2, 4]}))
```

4 Simple Linear Regression

- (Linear) Hypothesis

$$H(x) = Wx + b$$

- Cost/Loss function : How fit the line to our (training) data

$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

$$\Rightarrow cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

◇ Goal : Minimize Cost function

$$\min_{W,b} cost(W,b)$$

4.1 Implementation of Simple Linear Regression

MLlab 02 method ① - Assign values

Step1. Build graph

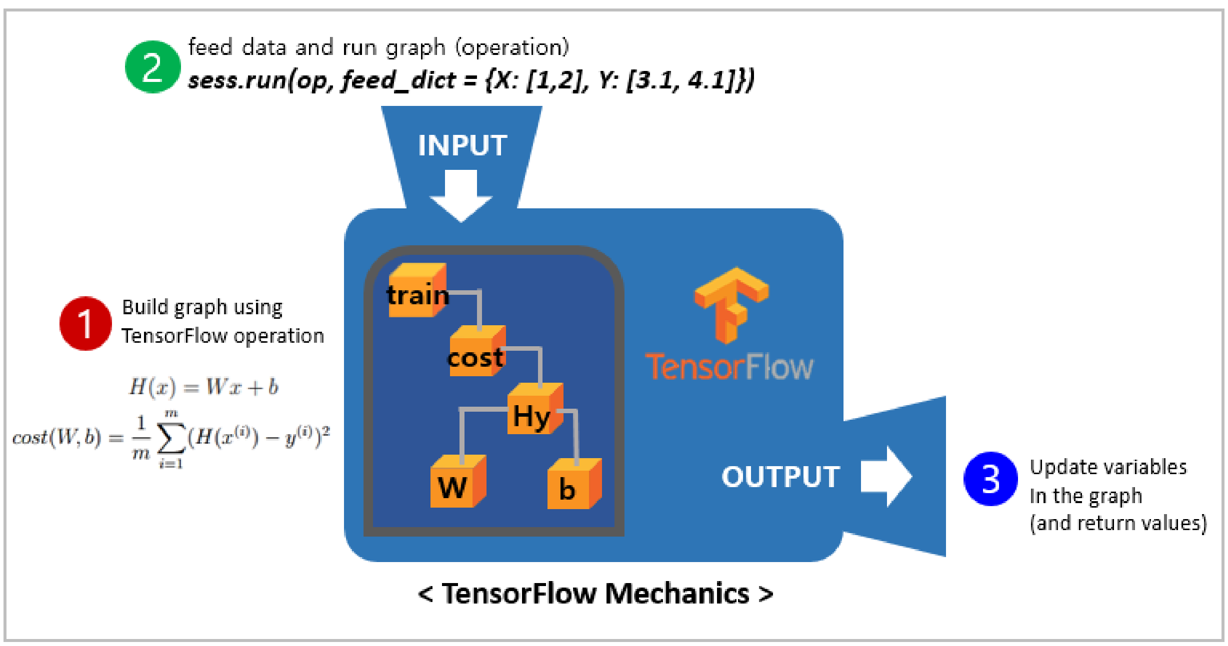
```
# X and Y data
x_train = [1, 2,3]
y_train = [1, 2, 3]

w = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Our hypothesis XW+b
hypothesis = x_train * w + b

# Cost/Loss function
cost = tf.reduce_mean(tf.square(hypothesis - y_train))

# Gradient Descent_Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)
```

Step2. Run & Update graph and get results

```
# Launch the graph in a session
sess = tf.Session()

# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

# Fit the line
for step in range(2001):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(cost), sess.run(w), sess.run(b))
```

MLlab 02 method ② - Use placeholder

```python
# Now we can use X and Y in place of x_data and y_data
## Placeholders for a tensor that will be always fed using feed_dict
X = tf.placeholder(tf.float32, shape=[None])
Y = tf.placeholder(tf.float32, shape=[None])
W = tf.Variable(tf.random_normal([1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Our hypothesis
hypothesis = X * W + b
# Cost function
cost =  tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train = optimizer.minimize(cost)

# Launch the graph in a session
sess = tf.Session()
# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

# Fit the line with new training data
for step in range(2001):
    cost_val, W_val, b_val, _ = \
         sess.run([cost, W, b, train],
                  feed_dict={X: [1, 2, 3, 4, 5], Y: [2.1, 3.1, 4.1, 5.1, 6.1]})
    if step % 20 == 0:
            print(step, cost_val, W_val, b_val)

# Testing our model
print(sess.run(hypothesis, feed_dict={X: [5]}))
print(sess.run(hypothesis, feed_dict={X: [1.5, 3.5]}))
```



4.2 Gradient descent

One of the most popular and widely used method to find the minimum of a function for machine learning algorithms.

- Minimize cost function
- Gradient descent is used many minimization problems
- For a given cost function, $cost(W, b)$, it will find W, b to minimize cost
- It can be applied to more general function : $cost(w_1, w_2, \dots)$

1) How it works?

- Start with initial guesses
  - Start at 0 (or any other value)
  - Keeeping changing W and b a little bit to try and reduce $cost(W, b)$
- Each time you change the parameters, you select the gradient which reduces $cost(W, b)$ the most possible
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
  - Where you start can determine which minimum you end up

2) Formal definition

- $cost(W) = \dfrac{1}{2m}\displaystyle\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$
- $W := W - \alpha\dfrac{\partial}{\partial W}cost(W)$ ($\alpha = 0.1;$ learning rate)

$$\Rightarrow \quad W := W - \alpha\frac{\partial}{\partial W}\frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

$$\Rightarrow \quad W := W - \alpha\frac{1}{2m}\sum_{i=1}^{m}2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$\Rightarrow \quad W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)} \quad : \text{“}Gradient\ \ descent\ \ algorithm\text{”}$$

4.3 Gradient descent algorithm

- Simplified Hypothesis

$$H(x) = Wx$$

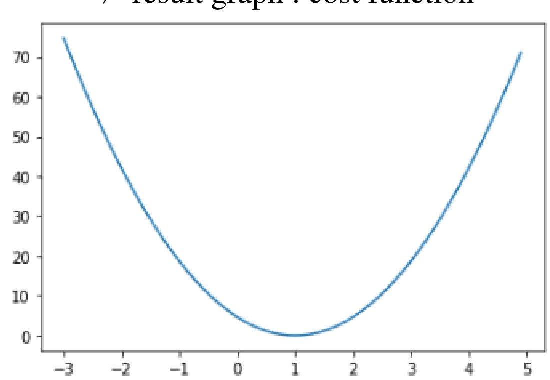
- Cost function

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

MLlab 03 Plot of cost function

```
X = [1, 2, 3]
Y = [1, 2, 3]

W = tf.placeholder(tf.float32)

# Our hypothesis for linear model X*W
hypothesis = X * W

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Launch the graph in a session
sess = tf.Session()

# Initializes global variables in the graph
# (Open session to initialize variables)
sess.run(tf.global_variables_initializer())

# Variables for plotting cost function
W_val = []      # make empty list
cost_val = []

type(W_val)     # see type of variable

for i in range(-30, 50):     # start -30, cuz cannot be incremented by 0.1
    feed_W = i * 0.1         # x-axis : 0.1 increments from -3 to 5
    curr_cost, curr_W = sess.run([cost, W], feed_dict={W: feed_W})
    W_val.append(curr_W)     # accumulate the list data
    cost_val.append(curr_cost)

# Show the cost function
plt.plot(W_val, cost_val)
plt.show()
```

$\Rightarrow$ result graph : cost function



MLlab 03 Minimize cost function

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

W = tf.Variable(tf.random_normal([1]), name='weight')
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

# Our hypothesis for linear model X*W
hypothesis = X * W
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize: Gradient Descent using derivative: W -= learning_rate * derivative
learning_rate = 0.1
gradient = tf.reduce_mean((W * X - Y) * X)
descent = W - learning_rate * gradient
update = W.assign(descent)

# Launch the graph in a session
sess = tf.Session()
# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

for step in range(21):
    sess.run(update, feed_dict={X: x_data, Y: y_data})
    print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W))
```

- **Exercise**: Using `tf.train.GradientDescentOptimizer()`

```python
# tf Graph Input
X = [1, 2, 3]
Y = [1, 2, 3]

# Set wrong model weights
W = tf.Variable(5.0)

# linear model
hypothesis = X * W
# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize: Gradient Descent Magic
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train = optimizer.minimize(cost)

# Launch the graph in a session
sess = tf.Session()
# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

for step in range(100):
    print(step, sess.run(W))
    sess.run(train)
```

MLlab 03 Optional: compute_gradients() and apply_gradients()

```python
# tf Graph Input
X = [1, 2, 3]
Y = [1, 2, 3]

# Set wrong model weights
W = tf.Variable(5.0)

# linear model
hypothesis = X * W

# Manual gradient
gradient = tf.reduce_mean((W * X - Y) * X) * 2

# cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)

# Get gradients
gvs = optimizer.compute_gradients(cost,[W])

# Apply gradients
apply_gradients = optimizer.apply_gradients(gvs)

# Launch the graph in a session
sess = tf.Session()
# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

for step in range(100):
    print(step, sess.run([gradient, W, gvs]))
    sess.run(apply_gradients)
```

5 Multi-variable Linear Regression

- Hypothesis

$$H(x_1, x_2, x_3, \cdots, x_n) = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b$$

- Cost function

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left(H\left(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \cdots, x_n^{(i)}\right) - y^{(i)}\right)^2$$

5.1 Hypothesis using matrix

<Test Scores for General Psychology>

X_1	X_2	X_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

① $H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3$

MLlab 04-1 ① Hypothesis using general tensorflow

```
x1_data = [73., 93., 89., 96., 73.]
x2_data = [80., 88., 91., 98., 66.]
x3_data = [75., 93., 90., 100., 70.]
y_data = [152., 185., 180., 196., 142.]

# placeholders for a tensor that will be always fed.
x1 = tf.placeholder(tf.float32)
x2 = tf.placeholder(tf.float32)
x3 = tf.placeholder(tf.float32)

Y = tf.placeholder(tf.float32)

w1 = tf.Variable(tf.random_normal([1]), name='weight1')
w2 = tf.Variable(tf.random_normal([1]), name='weight2')
w3 = tf.Variable(tf.random_normal([1]), name='weight3')
b = tf.Variable(tf.random_normal([1]), name='bias')

#Hypothesis
hypothesis = x1 * w1 + x2 * w2 + x3 * w3 + b
```

$$② \quad H(X) = XW = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (w_1x_1 + w_2x_2 + w_3x_3)$$

MLlab 04-1 ② Hypothesis using matrix

```
x_data = [[73., 80., 75.], [93., 88., 93.], [89., 91., 90.], [96., 98., 100.], [73., 66., 70.]]
y_data = [[152.], [185.], [180.], [196.], [142.]]

# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([3, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

#Hypothesis
hypothesis = tf.matmul(X, W) + b
```

```
#Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

# Launch the graph in a session
sess = tf.Session()
# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

for step in range(2001):
    cost_val, hy_val, _ = sess.run([cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
    if step % 10 == 0:
        print(step, "Cost:", cost_val, "\nPrediction:\n", hy_val)
```

5.2 Loading data from file

`os.chdir()` : Setting working directory

```
import os

print(os.getcwd())     # print working directory

os.chdir(".\\Data\\") # set w.d.
os.getcwd()            # check working directory
```

✓(disadvantage) Type of all data must be same.

MLlab 04-2 Loading data from file

```python
import numpy as np

xy = np.loadtxt('data-01-test-score.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# Make sure the shape and data are OK
print(x_data.shape, x_data, len(x_data))
print(y_data.shape, y_data)
```

◇ Indexing, Slicing, Iterating

- Arrays can be indexed, sliced, iterated much like lists and other sequence types in Python
- As with Python lists, slicing in `Numpy` can be accomplished with the colon(:) syntax
- Colon instances(:) can be replaced with dots(. . .)

MLlab 04-2 Slicing

```python
nums = range(5)       # range() is a built-in function that creates a list of integers
print(nums)
print(nums[2:4])      # Get a slice form index 2 to 4 (exclusive)
print(nums[2:])       # Get a slice form index 2 to the end
print(nums[:2])       # Get a slice form the start to index 2 (exclusive)
print(nums[:])        # Get a slice of the whole list
print(nums[:-1])      # Slice indices can be negative

nums[2:4] = [8,9]     # Assign a new sublist to a slice
print(nums)
```

range(0, 5)
range(2, 4)
range(2, 5)
range(0, 2)
range(0, 5)
range(0, 4)

TypeError
<ipython-input-21-86ffeac3decc> in <module>
7 print(nums[:-1]) Slice indices can be negative
8
—-> 9 nums[2:4] = [8,9] Assign a new sublist to a slice
10 print(nums)

TypeError: 'range' object does not support item assignment

MLlab 04-2 `np.array()`

```python
import numpy as np

a = np.array([1, 2, 3, 4, 5])
# array([1, 2, 3, 4, 5])

a[1:3]
# array([2, 3])

a[-1]
# 5

a[0:2] = 9
a
# array([9, 9, 3, 4, 5])
```

```python
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
# array([[ 1,  2,  3,  4],
#        [ 5,  6,  7,  8],
#        [ 9, 10, 11, 12]])

b[:, 1]
# array([ 2,  6, 10])

b[-1]
# array([ 9, 10, 11, 12])

b[-1,:]
# array([ 9, 10, 11, 12])

b[-1, ...]
# array([ 9, 10, 11, 12])

b[0:2,:]
# array([[1, 2, 3, 4],
#        [5, 6, 7, 8]])
```

MLlab 04-2 [Full Code] multi-variable linear regression

```python
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility

#==========================================================================================#
# Loading data from file
xy = np.loadtxt('data-01-test-score.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

# Make sure the shape and data are OK
print(x_data.shape, x_data, len(x_data))
print(y_data.shape, y_data)

#==========================================================================================#
# Placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([3, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis
hypothesis = tf.matmul(X, W) + b

# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

# Launch the graph in a session
sess = tf.Session()
# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

#==========================================================================================#
# Set up feed_dict variables inside the loop
for step in range(2001):
    cost_val, hy_val, _ = sess.run([cost, hypothesis, train], feed_dict={X: x_data, Y: y_data})
    if step % 10 == 0:
       print(step, "Cost:", cost_val, "\nPrediction:\n", hy_val)

#==========================================================================================#
# Ask my score
print("Your score will be", sess.run(hypothesis, feed_dict={X: [[100, 70, 101]]}))

print("Other scores will be", sess.run(hypothesis,
                                       feed_dict={X: [[60, 70, 110], [90, 100, 80]]}))
```

5.3 Queue Runners

If the data volume is large, it becomes difficult to load the data into the memory. For example, when the data size is too large, using `Numpy` can cause out of memory problems. In this case, it is more efficient to load only the necessary data.

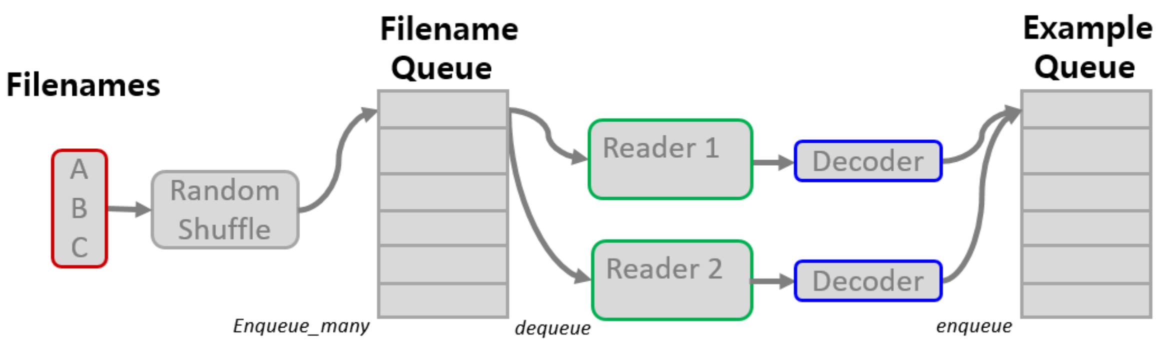
In the previous deep learning model, data was put into `feed_dict` argument in `tf.Session().Run()`. However, this method is slow because it copies data to a single thread.
To compensate for this drawback, TensorFlow has a "Queue Runners" system.

✓ Step of Queue Runners

① Make a list of files.

② Define a reader to read the file.

③ Set the data type with `tf.decode_csv()`. To determine parsing the file's value.



1

```
filename_queue = tf.train.string_input_producer(
   ['data-01-test-score.csv', 'data-01-test-score.csv', ...],
    shuffle=False, name='filename_queue')
```

2

```
reader = tf.TextLineReader()
key, value = reader.read(filename_queue)
```

3

```
record_defaults = [[0.], [0.], [0.], [0.]]
xy = tf.decode_csv(value, record_defaults = record_defaults)
```

MLlab 04-2 [Full Code] multi-variable linear regression using Queue Runners

```python
import tensorflow as tf

#===Queue Runners==========================================================================#
filename_queue = tf.train.string_input_producer(
    ['data-01-test-score.csv'], shuffle=False, name='filename_queue')

reader = tf.TextLineReader()
key, value = reader.read(filename_queue)

# Default values, in cast of empty columns. Also specifies the type of the decoded result.
record_defaults = [[0.], [0.], [0.], [0.]]
xy = tf.decode_csv(value, record_defaults = record_defaults)

# collect batches of csv in
train_x_batch, train_y_batch = tf.train.batch([xy[0:-1], xy[-1:]], batch_size=10)

#=========================================================================================#
# Placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([3, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis
hypothesis = tf.matmul(X, W) + b

# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))

# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

# Launch the graph in a session
sess = tf.Session()
# Initializes global variables in the graph
sess.run(tf.global_variables_initializer())

#===Start populating the filename==========================================================#
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(sess=sess, coord=coord)

for step in range(2001):
    x_batch, y_batch = sess.run([train_x_batch, train_y_batch])
    cost_val, hy_val, _ = sess.run([cost, hypothesis, train],
                                    feed_dict={X: x_batch, Y: y_batch})
    if step % 10 == 0:
        print(step, "Cost:", cost_val, "Prediction:", hy_val)

coord.request_stop()
coord.join(threads)
```

MLlab 04-2 Shuffle Batch

```python
# min_after_dequeue defines how big a buffer we will randomly sample
#   from -- bigger means better shuffling but slower start up and more memory used.
# capacity must be larger than min_after_dequere and the amount larger
#   determines the maximum we will prefetch. Recommendation:
#   min_after_dequere + (num_threads + a small safety margin) * batch_size

batch_size = 10
min_after_dequeue = 10000
capacity = min_after_dequeue + 3 * batch_size

example_batch, label_batch = tf.train.shuffle_batch([example, label],
   batch_size=batch_size, capacity=capacity, min_after_dequeue=min_after_dequeue)
```

✓Summary of Linear Regression

- Hypothesis $H(X) = WX$
- Cost function $cost(W) = \dfrac{1}{m}\sum_{i=1}^{m}(WX - y)^2$
- Gradient descent $W := W - \alpha \dfrac{\partial}{\partial W} cost(W)$

6 Logistic Regression Classification

- Binary Classification $\Rightarrow$ **0 & 1 encoding**
  - Spam Email Detection: Spam(**1**) or Ham(**0**)
  - Facebook feed: show(**1**) or hide(**0**)
  - Credit Card Fraudulent Transaction detection : legitimate(**0**) or fraud(**1**)

6.1 Logistic Hypothesis

- logistic function (sigmoid function) $g(z) = \dfrac{1}{(1 + e^{-z})}$
- let $z = WX,$ then $H(x) = g(z)$

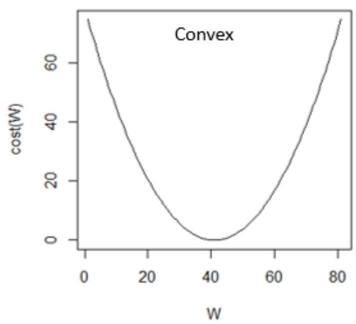
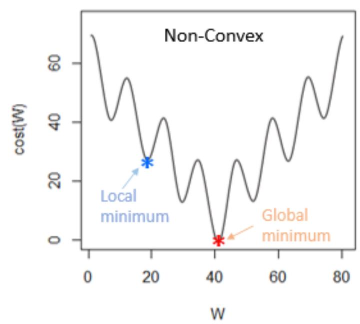
$\Rightarrow$ Logistic Hypothesis

$$H(X) = \frac{1}{1 + e^{-W^T X}} \quad (0 < H(X) < 1)$$

6.2 Cost function

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

- What about the graph of the cost function in logistic regression?

	Linear regression	Logistic regression
Hypothesis	$H(X) = Wx + b$	$H(X) = \frac{1}{1 + e^{-W^T X}}$
Plot of cost function		

⇒ Because the graph of the cost function is **non-convex**, <u>Global Minimum is not found and Local Minimum is found.</u>

⇒ It is difficult to apply the gradient decent algorithm.

- New cost function for logistic

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m} c(H(x), y)$$

$$c(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

$$\Rightarrow \quad c(H(x), y) = -y\log(H(x)) - (1 - y)\log(1 - H(x))$$

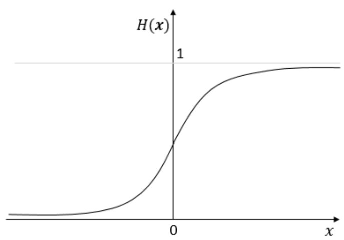
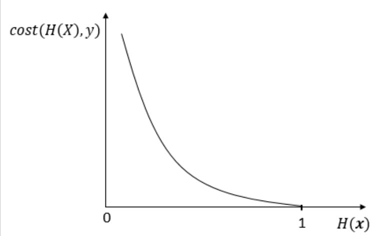
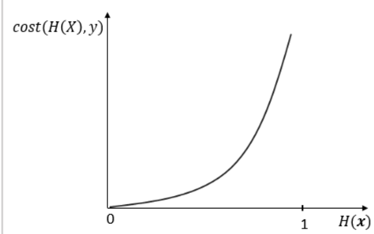
6.3 Minimize cost & Gradient descent

- (Logistic) Hypothesis

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

- (Logistic) Cost function

$$c(H(x), y) = -\frac{1}{m} \sum_{i=1}^{m} \{y \log(H(x)) + (1 - y) \log(1 - H(x))\}$$

Hypothesis	Cost function	
	$if\,(\boldsymbol{y} = \mathbf{1})$	$if\,(\boldsymbol{y} = \mathbf{0})$
		

- Gradient descent algorithm

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

MLlab 05 Gradient descent algorithm

```
# cost function
cost = - tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))

# Minimize
a = tf.Variable(0.1)    # Learning rate; alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
```

```
import tensorflow as tf

#===Training Data===================================================================#
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]   # binary 0,1

# Placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight') # 2 = number of input (X)
                                                              # 1 = number of output (Y)
b = tf.Variable(tf.random_normal([1]), name='bias')

#==================================================================================#
# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W) + b))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

# cost function.
cost = - tf.reduce_mean(Y * tf.log(hypothesis) + (1-Y) * tf.log(1-hypothesis))

# Gradient descent algorithm : Minimize cost.
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).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))

#===Train the model==================================================================#
# Launch the graph.
with tf.Session() as sess:
    # Initializes TensorFlow vaiables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0:
            print(step, "Cost:", cost_val)
    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy],
                       feed_dict={X: x_data, Y: y_data})
     print("\nHypothesis: ", h, "\nCorrect (Y): ", c, "\nAccuracy: ", a)
```

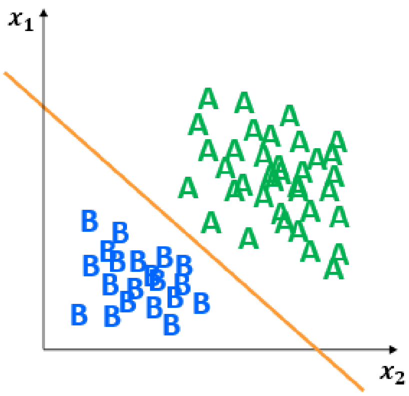
• **Exercise**: : Classifying Diabetes.

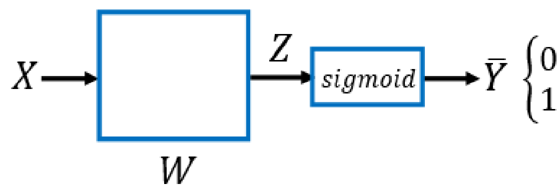
```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility

#===Load Data======================================================================#
xy = np.loadtxt('data-03-diabetes.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

# Placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 8])
Y = tf.placeholder(tf.float32, shape=[None, 1])

W = tf.Variable(tf.random_normal([8, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
```

7 Softmax classification

7.1 Multi-nomial classification

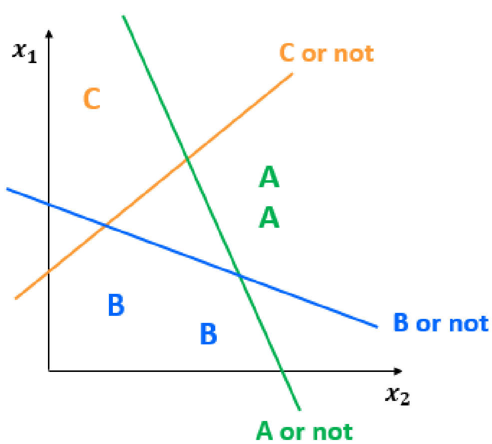
- Binary classification
  ⇒ Same as finding a line that divides into two (0 or 1).

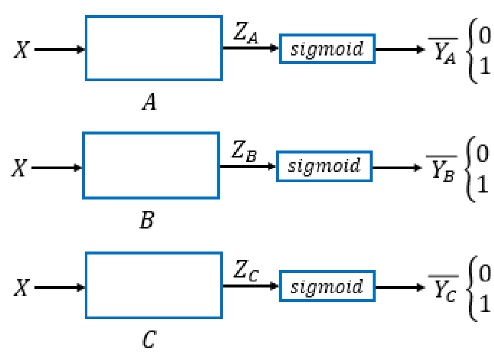




- Multi-nomial classification

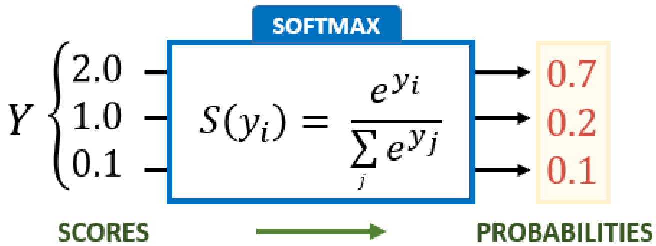
X_1 (hours)	X_2 (attendance)	Y (grade)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C





$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} Z_A \\ Z_B \\ Z_C \end{bmatrix} \rightarrow \boxed{sigmoid} \rightarrow \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix} = \begin{bmatrix} \mathbf{0} < H_A(X) < \mathbf{1} \\ \mathbf{0} < H_B(X) < \mathbf{1} \\ \mathbf{0} < H_C(X) < \mathbf{1} \end{bmatrix}$$

7.2 Softmax



1. Like "sigmoid", it changes the result of the score to a value between 0 and 1.
2. It makes a total sum of the results = 1.

⇒ For example, if the result is $\overline{y_A}$ = 0.7, this means that there is a 70% chance that A can be.

- one-hot encoding 을 이용해서, 가장 높은 확률의 선택지를 1.0으로, 나머지를 0.0으로 대치.
- one-hot encoding is argmax of tensorflow

⇒ 예측모델 완성!

- now 예측값과 실제값의 차이가 얼마인지 확인하는 Cost 함수를 완성해야됨

7.3 Cost function

- Cross-Entropy

$$D(S, L) = -\sum_i L_i \log(S_i) = \sum_i L_i(-\log(S_i))$$

$$(S = \overline{Y} \textit{ is predicted value, } L = Y \textit{ is true value})$$

✓ Why using this cost function?
If the predicted value is the same as the true value, the cost value will be small.
When different, the cost value will be large.

- Logistic cost **vs** Cross-entropy cost ?
  - Logistic cost
$$c(H(x), y) = -y\log(H(x)) - (1-y)\log(1-H(x))$$
$$(H(x) \textit{ is predicted value, } y \textit{ is true value})$$
  - Cross entropy cost
$$D(S, L) = -\sum_i L_i \log(S_i) = \sum_i L_i(-\log(S_i))$$
$$(S \textit{ is predicted value}, L \textit{ is true value})$$

⇒ Let predicted value is $\overline{Y}$ and true value is L,
then it can see Logistic cost and Cross entropy cost are the same.

$$cost(\overline{Y}, L) = -\sum_i L_i \log(\overline{Y}_i) = -L\log(\overline{Y}) - (1-L)\log(1-\overline{Y})$$

7.4 Cross-Entropy Cost function

$$\begin{aligned}\mathcal{L} = cost(\overline{Y}, L) &= \frac{1}{m}\sum_{i=1} D(S_i, L_i)\\ &= \frac{1}{m}\sum_{i=1} D(Softmax(z_i), L_i)\\ &= \frac{1}{m}\sum_{i=1} D(Softmax(wx_i + b), L_i)\end{aligned}$$

($z_i = wx_i + b$, $\overline{Y} = S_i = Softmax(z_i)$ is predicted value, *i is training set*)

7.5 Gradient descent

Find W that minimizes the cost function $\mathcal{L}(W)$. (Same as for logistic)

$$W := W - \alpha\frac{\partial}{\partial W}\mathcal{L}(W)$$

7.6 Implementation of Softmax Classifier

MLlab 06-1 Softmax Classifier

```
import tensorflow as tf
tf.set_random_seed(777)  # for reproducibility

#===Training Data=================================================================#
x_data = [[1, 2, 1, 1],
          [2, 1, 3, 2],
          [3, 1, 3, 4],
          [4, 1, 5, 5],
          [1, 7, 5, 5],
          [1, 2, 5, 6],
          [1, 6, 6, 6],
          [1, 7, 7, 7]]

# one-hot-encoding: y=[2,2,2,1,1,1,0,0]
y_data = [[0, 0, 1],
          [0, 0, 1],
          [0, 0, 1],
          [0, 1, 0],
          [0, 1, 0],
          [0, 1, 0],
          [1, 0, 0],
          [1, 0, 0]]

X = tf.placeholder("float", [None, 4])
Y = tf.placeholder("float", [None, 3])

nb_classes = 3

W = tf.Variable(tf.random_normal([4, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

#================================================================================#
# Hypothesis using softmax
# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_mean(Y*tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

#===Train the model==============================================================#
# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(2001):
            _, cost_val = sess.run([optimizer, cost], feed_dict={X: x_data, Y: y_data})
            if step % 200 == 0:
                print(step, cost_val)
```

MLlab 06-1 Testing & One-hot encoding

```python
#===Testing & One-hot encoding=====================================================#
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
        print(a, sess.run(tf.argmax(a, 1)))

    print('--------------')
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        b = sess.run(hypothesis, feed_dict={X: [[1, 3, 4, 3]]})
        print(b, sess.run(tf.argmax(b, 1)))

    print('--------------')
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        c = sess.run(hypothesis, feed_dict={X: [[1, 1, 0, 1]]})
        print(c, sess.run(tf.argmax(c, 1)))

    print('--------------')
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        all = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9],
                                                  [1, 3, 4, 3],
                                                  [1, 1, 0, 1]]})
        print(all, sess.run(tf.argmax(all, 1)))
```

7.7 Fancy Softmax Classifier

- `softmax_cross_entropy_with_logit()`

```python
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

## Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_mean(Y*tf.log(hypothesis), axis=1))

cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
                                                 labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
```

- `tf.one_hot()` and `tf.reshape()`

```python
Y = tf.placeholder(tf.int32, [None, 1])  # 0 ~ 6, shape=(?, 1)
Y_one_hot = tf.one_hot(Y, nb_classes)    # one hot shape=(?, 1, 7)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])  # shape=(?, 7)
```

MLlab 06-2 Exercise: Predicting animal type based on various features

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # for reproducibility

#===Predicting animal type based on various features================================#
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

nb_classes = 7  # 0 ~ 6

X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1])  # 0 ~ 6

# change to one-hot
Y_one_hot = tf.one_hot(Y, nb_classes)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])

W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')

#==================================================================================#
# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)

# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)

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

#===Train the model=================================================================#
prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(2001):
        _, cost_val, acc_val = sess.run([optimizer, cost, accuracy], feed_dict={X: x_data, Y: y_data})

        if step % 100 == 0:
            print("Step: {:5}\tCost: {:.3f}\tAcc: {:.2%}".format(step, cost_val, acc_val))
```

8 Application & Tip

8.1 Learning rate

- Large learning rate : Overshooting.
- Small learning rate : It takes too long and trapping in local minimum.

⇒ Try several learning rates. at first, try α=0.01 !

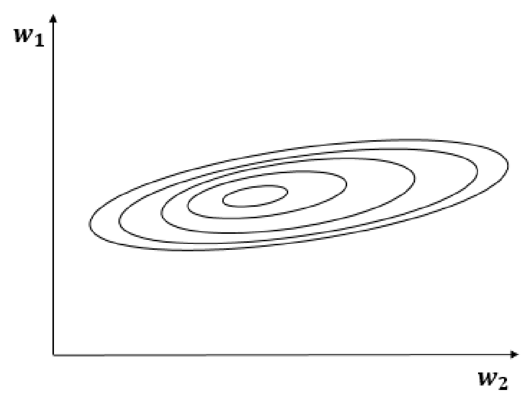
✓ Observe the cost function.

✓ Check it goes down in a reasonable rate.

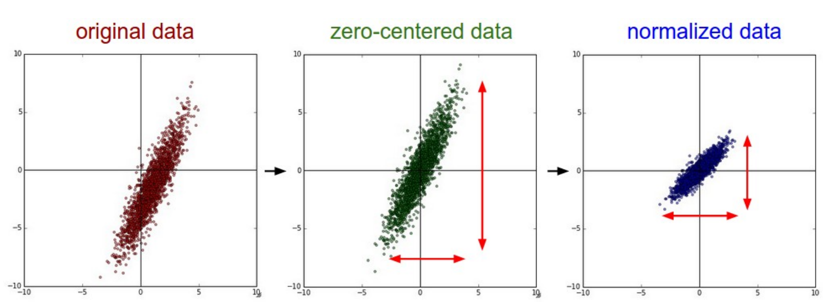
8.2 Data Preprocessing

Preprocessing for gradient descent

X_1	X_2	Y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C



x1 변수는 10보다 작은 숫자, x2 변수는 -5000에서 9000까지의 숫자라면 진짜 동그랗게 생긴 원의 모양이 아니라 한쪽으로 길게 늘어진 타원 모양이 된다. 이렇게 된다면 수평으로 이동할 때와 수직으로 이동할 때 엄청난 불균형이 발생하게 되어 gradient descent 알고리듬을 적용하기 어려운 상황이 될 수 있다.
등고선으로 표현할 때, 가장 좋은 형태는 완벽하게 둥근 원(circle)이다. 수평과 수직으로 동일한 범위를 갖게 만들면 가장 이상적인 원이 된다. gradient descent 알고리듬을 적용하기 전에 preprocessing 작업으로 데이터의 범위를 제한할 수 있다.

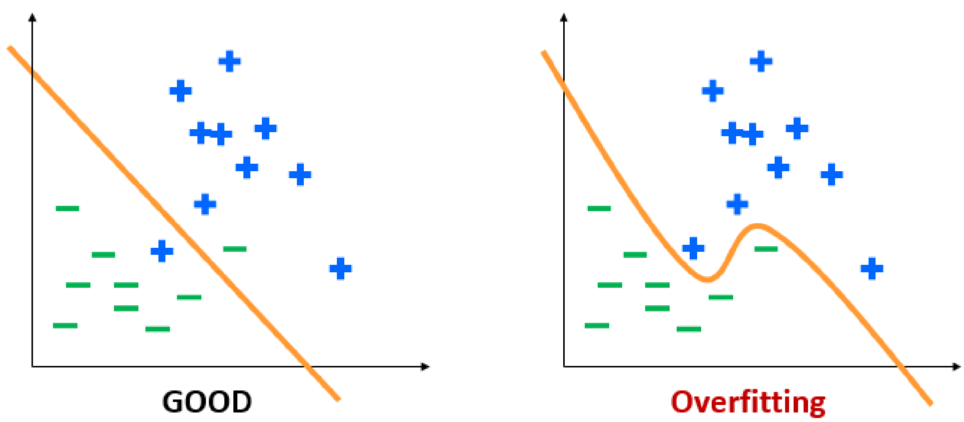


- Normalization
  수식 : (요소값 - 최소값) / (최대값 - 최소값)
  설명 : 전체 구간을 0 100으로 설정하여 데이터를 관찰하는 방법으로, 특정 데이터의 위치를 확인할 수 있게 해줌
  sol of code: xy = MinMaxScaler(xy)

- Standardization
  수식 : (요소값 - 평균) / 표준편차
  설명 : 평균까지의 거리로, 2개 이상의 대상이 단위가 다를 때, 대상 데이터를 같은 기준으로 볼 수 있게 해줌

8.3 Overfitting

very good for training data set, but Not good at test data set or in real use.



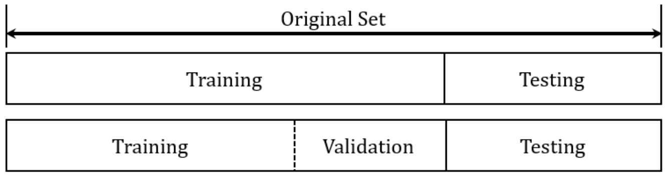
$\Rightarrow$ Solution

- ✓ More training data!
- ✓ Reduce the number of duplicate features(x).
- ✓ **Regularization** : Let's not have too big numbers in the weight.
  (λ is regularization strength.)

$$\mathcal{L} = \frac{1}{m}\sum_{i=1} D(S(wx_i + b), L_i) + \lambda \sum_{i=1} {w_i}^2$$

```
# Regularization
lambda = 0.001    # regularization strength
l2reg = lambda * tf.reduce_mean(tf.square(W))
```

8.4 Training & Validation & Test datasets



MLlab 07-1 Training and Test datasets

```python
import tensorflow as tf
tf.set_random_seed(777)  # for reproducibility

#===Training Data========================================================#
x_data = [[1, 2, 1], [1, 3, 2], [1, 3, 4], [1, 5, 5],
          [1, 7, 5], [1, 2, 5], [1, 6, 6], [1, 7, 7]]

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

#===Test Data============================================================#
# Evaluation our model using this test dataset
x_test = [[2, 1, 1], [3, 1, 2], [3, 3, 4]]

y_test = [[0, 0, 1], [0, 0, 1], [0, 0, 1]]

#========================================================================#
X = tf.placeholder("float", [None, 3])
Y = tf.placeholder("float", [None, 3])

W = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.random_normal([3]))

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
# Try to change learning_rate to small numbers
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

#========================================================================#
# Correct prediction Test model
prediction = tf.argmax(hypothesis, 1)
is_correct = tf.equal(prediction, tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(201):
        cost_val, W_val, _ = sess.run([cost, W, optimizer],
                              feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, W_val)

    # predict
    print("Prediction:", sess.run(prediction, feed_dict={X: x_test}))
    # Calculate the accuracy
    print("Accuracy: ", sess.run(accuracy, feed_dict={X: x_test, Y: y_test}))
```

8.5 Exercise: MNIST Dataset

MLlab 07-2 Check MNIST Data

① Load 'MNIST Data' from library

```
from tensorflow.examples.tutorials.mnist import input_data
# Check out http://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

② $28 \times 28 \times 1$ image

```
# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])

# 0 ~ 9 digits recognition => 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])
```

. source : [THE MNIST DATABASE] http://yann.lecun.com/exdb/mnist/

- Training epoch / batch
  - one **epoch** : one forward pass and one backward pass of all the training examples.
  - **batch size** : the number of training examples in one forward/backward pass.
  - number of **iterations** : number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass

✓ Example.
if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

MLlab 07-2 [Full Code] MNIST Dataset

```python
import tensorflow as tf
tf.set_random_seed(777)  # for reproducibility

#=== Load MNIST Data from html ====================================================#
from tensorflow.examples.tutorials.mnist import input_data
# Check out http://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

#=== MNIST Data ==================================================================#
nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])

# 0 ~ 9 digits recognition => 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))

#=== Softmax! ====================================================================#
# Hypothesis using softmax
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

# Cross-entropy cost
cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

#=== Test model ==================================================================#
is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))

# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

#================================================================================#
# Parameters
training_epochs = 15
batch_size = 100

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    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)
            c, _ = sess.run([cost, optimizer], feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += c / total_batch

        print('Epoch:', '%04d' % (epoch +1), 'cost =', '{:.9f}'.format(avg_cost))

#=== Report results on test dataset ==============================================#
        # Test the model using "test sets".
        print("Accuracy: ", accuracy.eval(session=sess,
           feed_dict={X: mnist.test.images, Y: mnist.test.labels}))
```

MLlab 07-2 Exercise: Sample image show and prediction

```python
import matplotlib.pyplot as plt
import random

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)    # select random number.

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

    print("Lable:", 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()
```