# OpenCV

OpenCV의 현재 버전은 3.3.0이다. 아직까지는 python 공식 배포본이 없으나 비공식 배포본이 존재한다.

1. 관리자 모드에서 cmd.exe를 연다. (Anaconda의 경우 Anaconda Prompt를 관리모드에서 연다)

> pip install opencv-contrib-python 🡪 최신 버전 3.3.0.10

확인은 python에서

>>> import cv2

>>> cv2.\_\_version\_\_

주요 참고 사이트는

Opencv.org

https://opencv-python-tutroals.readthedocs.io/en/latest/

# TensorFlow

## 기본사항

### 모든 것은 연산(operation)

TensorFlow는 연산(operation)을 노드(node)로 하는 그래프 형태로 연결하여 구동된다. 이때 그래프의 선을 따라 흐르는 데이터가 텐서(tensor)이다. 따라서 TensorFlow 객체는 모두 연산(operation)을 정의하는 것이며 연산의 결과 값을 보기 위해서는 항상 Session을 통해 구동시켜야 한다

Code 6.1 1-concept.py

**import** tensorflow **as** tf

# tensorflow의 그래프를 구동시키려면 Session 객체가 필요하다.

sess **=** tf**.**Session**()**

# tf.constant(v)는 상수(즉, 계산 과정동안 변경 없는) 연산을 정의하고

hello **=** tf**.**constant**(**'Hello, TensorFlow!'**)** # constant operation

a **=** tf**.**constant**(**2**)** # constant operation

b **=** tf**.**constant**(**3**)** # constant operation

c **=** a**+**b # 사칙연산 역시 연산 add operation

# 따라서 tensorflow 객체를 출력하면, 객체 정보를 출력한다. 값을 출력하는 것이 아니다.

**print(**c**)**

# 실제 값은 계산을 수행해야 한다.

# 즉, session.run(object) 형태로 실행해야 한다.

**print(**sess**.**run**(**hello**))** # Now, I can see the value of c

**print(**sess**.**run**(**c**))** # Now, I can see the value of c

**print(**sess**.**run**(**a**\***c**))**

# 다 쓴 Session 은 닫을 수 있다.

sess**.**close**()**

# 아래와 같이 with 표현으로 많이 쓰기도 한다.

**with** tf**.**Session**()** **as** sess**:**

sess**.**run**(**hello**)**

sess**.**run**(**a**)**

### constant()와 Variable()

tf.constant()는 Session을 통해 계산하는 동한 변화하지 않는 상수값을 지정하는 데 사용하는 반면 tf.Variable()은 변화하는 값을 저장하는데 사용된다. 즉, tf.Variable()은 상태를 저장하고 그 값은 training을 통해 변화가능하다는 의미이다. tf.Variable()을 사용할 때 주의할 점은 반드시 초기화 연산을 통해 초기화를 해야 한다는 점이다.

Code 6.2 1-constantAndVariable.py

import tensorflow as tf

c = tf.constant(3.) # 상수를 3.으로 지정하는 연산

v = tf.Variable(1.) # 변수를 초기값 1.로 지정하는 연산.

# 주의할 점은 변수는 세션이 연산 시작부에

# 초기화하는 연산을 미리 수행해야만 한다.

sess = tf.Session()

init = tf.global\_variables\_initializer() # 모든 변수를 초기화하는 연산

sess.run(init) # 변수 초기화 연산을 수행

# 또는 sess.run(tf.global\_variable\_initializer()) 와 같이 한줄로도 작성 가능

print(sess.run(c))

print(sess.run(v))

### NumPy

tf.constant()와 tf.Variable() 에서 배열 등도 다양하게 지정할 수 있다. 배열일 경우 기본적으로 NumPy의 ndarray를 내부의 데이터인 tensor를 표현하는 데 사용하고 있다.

Code 6.3 3-NumPy.py

import tensorflow as tf

import numpy as np

A = tf.constant([1,2,3])

B = tf.constant(np.arange(0.,np.pi,0.1))

# 내부적으로 ndarray로 변환하므로 아래 두 연산의 결과는 동일하다.

C = tf.Variable(np.array([[1,2],[3,4]]))

D = tf.Variable([[1,2],[3,4]])

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

print(sess.run(A))

print(sess.run(B))

print(sess.run(D))

print(sess.run(C))

### Graph

TensorFlow의 그래프는 연산간의 연결로 이해할 수 있다. 아래 예는 2\*2 행렬, 2\*1 행렬에 대한 연산을 그래프 개념으로 연결한 것이다.

Code 6.4 4-graph.py

import tensorflow as tf

A = tf.constant([[3.,3.],[1,1.5]])

B = tf.constant([[0.1],[0.5]])

C = tf.matmul(A,B)

sess = tf.Session()

print(sess.run(A))

print(sess.run(B))

print(sess.run(C))

### Fetch

Session을 run() 할때 한개 node를 실행할 수도 있지만 여러 개의 node를 리스트로 묶어도 된다. 이것은 multiple tensors(노드내의 값, 상태)를 fetch한다고 한다.

Code 6.5 5-fetch.py

import tensorflow as tf

a = tf.constant([3.0])

b = tf.constant([2.0])

c = a+b

# 단일 노드를 대상으로 실행한 예

with tf.Session() as sess:

r1 = sess.run(a)

r2 = sess.run(c)

rr = [r1,r2]

print(rr)

# 다중 노드를 대상으로 실행한 예

with tf.Session() as sess:

rr = sess.run([a,c])

print(rr)

### Feed

값을 미리 정하지 않고 있다가 Session 실행시 정하는 것을 feed 라고 한다. 이를 위해서는 placeholder로 선언하고 session.run(...,feed\_dict={...}) 형태로 구동시키게 된다.

Code 6.6 6-feed.py

import tensorflow as tf

a = tf.placeholder(tf.float64)

b = tf.placeholder(tf.float64)

c = tf.mul(a,b)

with tf.Session() as session:

print(session.run(a,feed\_dict={a:2}))

print(session.run(b,feed\_dict={b:3}))

print(session.run(c,feed\_dict={a:2,b:3}))

print(session.run([a,b,c],feed\_dict={a:2,b:3}))

# 만약 array 형태라면 placeholder의 두번째 인자에 배열 크기를 지정하면 된다.

matAValue = [[1.,2.,3.],[4,5,6]]

matA = tf.placeholder(tf.float64,shape=(2,3))

with tf.Session() as session:

print(session.run(matA,feed\_dict={matA:matAValue}))

### Traning

만약 어떤 데이터를 가지고 있다고 가정하면, 먼저 그 데이터를 잘 표현할 수 있는 모델을 구성할 수 있다. 그리고 나서 모델 파라미터를 학습(traning)을 통해 구한다. 이제 모델을 예측(prediction)에 사용할 수 있다. 이러한 과정이 Machine Learning에서 이루어지는 전형적인 절차이다.

Code 6.7 7-traning.py

import tensorflow as tf

# Data = traing set

xData = [ 0., 0.2, 0.4, 0.7]

yData = [ 0.295 , 0.315, 0.341, 0.369]

# y = W\*X +b (linear model) assumed

# then, traing target are (W,b)

W = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

b = tf.Variable(tf.zeros([1]))

y = W \* xData + b

# Minimize the mean squared errors.

loss = tf.reduce\_mean(tf.square(y - yData)) # loss means "cost" or "objective"

optimizer = tf.train.GradientDescentOptimizer(0.5) # learningRate=0.5

train = optimizer.minimize(loss)

# Before starting, initialize the variables. We will 'run' this first.

init = tf.global\_variables\_initializer()

# Launch the graph.

sess = tf.Session()

sess.run(init)

# Fit the line.

lossVal = []

for step in range(201):

sess.run(train)

if step % 20 == 0:

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

# Learns best fit is W: [ 0.10728782], b: [ 0.2951315]

# view the traing results

import matplotlib.pyplot as plt

WW = sess.run(W)

bb = sess.run(b)

plt.plot(xData,yData,'o',xData,WW\*xData+bb)

## Linear Regression

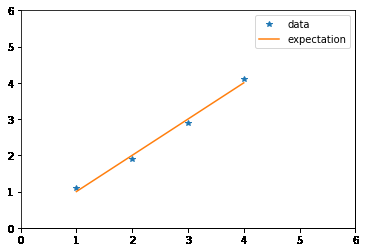
### 기본 개념

다음과 같은 데이터가 주어져 있고, x=2.5 일때 y 값을 예측하는 문제를 생각해 보기로 한다.

x = [ 1., 2., 3., 4.];

y = [ 1.1,1.9,2.9, 4.1];

위 데이터를 그림으로 표현하면 거의 로 표현할 수 있음을 알 수 있으며, 이를 통해 일 때 예측하면 이다.

import matplotlib.pyplot as plt

x = [ 1., 2., 3., 4.];

y = [ 1.1,1.9,2.9, 4.1];

plt.plot(x,y,'\*',label='data')

plt.plot(x,x,'-',label='expectation')

plt.axis([0,6,0,6]); plt.legend()

이와 같이 연속적인 Y값에 대해 적용 가능한 문제는 regression 문제라 하며, 특히 선형모델()로 표현하는 문제를 linear regression 문제라고 한다. Linear regression 문제는 모델 파라미터 와 를 결정하는 문제라고 할 수 있다. 수학적으로 표현하면 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.1) |

여기에서, 는 비용함수(cost fuction) 또는 목적함수(objective function), 손실함수(loss function)이고, 를 모델(model) 또는 가설(hypothesis)이다. 은 데이터의 수이고 위첨자를 각 데이터를 지칭한다. 와 는 최소화문제의 해로 결정되는 모델 파라미터이다. 저자에 따라 모델을 형태로 쓰기도 하는데 는 가중치(weight), b는 bias라고 한다.

ML 에서는 모델 파라미터 를 기존 데이터로부터 결정하는 과정을 traning이라고 부르며, 최적화문제를 푸는 도구를 Optimizer라고 한다. 다음 코드는 간단한 예이다. 실행하면 와 는 각각 1.0, 0.0으로 수렴되며 계산되는 것을 알 수 있다.

Code 6.8 1-LinearRegression.py

import tensorflow as tf

xData = [ 1., 2., 3., 4.];

yData = [ 1.1,1.9,2.9, 4.1];

W = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

b = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

# Linear Model as hypothesis with (W,b) paramters

H = W\*xData+b

# Define objective(=cost=loss)

cost = tf.reduce\_mean(tf.square(H-yData))

# Define train object with some optimizer

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(cost)

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

for step in range(1001):

sess.run(train)

if step % 20 == 0:

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

# Check with graphics

import matplotlib.pyplot as plt

plt.plot(xData,yData,'\*',label='input data')

plt.axis([0.,5.,0.,5.])

plt.plot(xData,WW\*xData+bb,label='trained model')

plt.xlabel('x'); plt.ylabel('y')

plt.legend()

보통 tf.placeholder()를 이용하여 사용하여 Code 6.9처럼 코드를 재사용할 수 있도록 하는 데 다음과 같다.

Code 6.9 2-LinearRegressionPlaceHolder.py

…xData, yData is given

X = tf.placeholder(tf.float32)

Y = tf.placeholder(tf.float32)

…

# Linear Model with (W,b) paramters

H = W\*X+b

# Define objective(=cost=loss)

cost = tf.reduce\_mean(tf.square(H-Y))

…

with tf.Session() as sess:

sess.run(init)

for step in range(1001):

sess.run(train,feed\_dict={X:xData,Y:yData})

if step % 20 == 0:

print(step, sess.run(cost,feed\_dict={X:xData,Y:yData}),sess.run(W))

PlaceHolder를 사용하면 ML 과정을 Code 6.10과 같이 함수 형태로 제작할 수도 있다. 이 코드는 미완성인데 성공적으로 traning이 수행되었는지 검토하는 부분이 없다.

Code 6.10 3-LinearRegressionFunction.py

import tensorflow as tf

def linearRegression(xData,yData):

X = tf.placeholder(tf.float32)

Y = tf.placeholder(tf.float32)

W = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

b = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

# Linear Model with (W,b) paramters

H = W\*X+b

# Define objective(=cost=loss)

cost = tf.reduce\_mean(tf.square(H-Y))

# Alternative .... Define train object with some optimizer

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(cost)

init = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init)

for step in range(1001):

sess.run(train,feed\_dict={X:xData,Y:yData})

# How to check, ...???

WW = sess.run(W)

bb = sess.run(b)

return (WW,bb)

import matplotlib.pyplot as plt

xData = [ 1., 2.,3., 4.];

yData = [ 1.1,1.9,2.9, 4.1];

(WW,bb)=linearRegression(xData,yData)

…

### 알고리즘 분석

Linear Regression 문제에서 다음과 같이 비용함수를 정의하고, 최적화 알고리즘을 선택하였다.

cost = tf.reduce\_mean(tf.square(H-yData))

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(cost)

위 코드에서 GradientDescentOptimizer는 비용함수에 대한 그래디언트를 계산하여 그 값이 줄어드는 방향으로 최적화를 수행하게 된다.

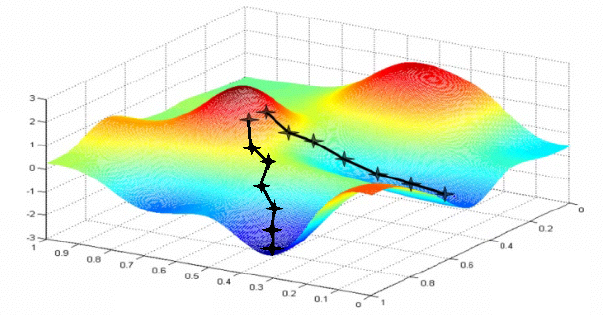


Figure 6.1 Gradient Descent Optimizer

설명의 편의를 위해 bias가 없다고 가정하고, 모델파라미터를 만을 선택하는 경우 다음과 같은 최적화 문제이다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.2) |

Gradient Descent 알고리즘은 각 단계에서 다음과 같이 모델 파라미터 를 갱신하는 것이다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.3) |

위에서 는 learning rate라 불리는 데 너무 큰값을 사용하면 overshooting이 발생하여 수렴하지 않고, 너무 작은 값을 사용하면 수렴이 늦거나 국부 최소해를 찾아가게 된다. 따라서 적당한 값을 사용자가 선택하도록 해야야 한다. Code 6.11은 tensorflow에서 제공하는 optimizer인 GradientDescentOptimizer를 사용하는 것과 식 (6.3)의 제시된 알고리즘을 직접 코딩한 것을 비교한 것이다. 완전히 동일한 결과를 나타낸다.

Code 6.11 3-LinearRegressionManual.py

import tensorflow as tf

xData = [ 1., 2., 3., 4.];

yData = [ 1.1,1.9,2.9, 4.1];

W = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))

# Linear Model as hypothesis with (W,b) paramters

H = W\*xData

# Define objective(=cost=loss)

cost = tf.reduce\_mean(tf.square(H-yData))

init = tf.global\_variables\_initializer()

# Using TensorFlow's Optimizer

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(cost)

with tf.Session() as sess:

sess.run(init)

for step in range(1001):

sess.run(train)

print('Optimizer : W=%f' %sess.run(W))

# Using Manual Optimizer

alpha = 0.1

dHdW = 2.\*tf.reduce\_mean(tf.mul((tf.mul(W, xData) - yData), xData))

newW = W - alpha\*dHdW

update = W.assign(newW)

with tf.Session() as sess:

sess.run(init)

for step in range(1001):

sess.run(update)

print('Manual : W=%f' %sess.run(W))

### General Expression

만약 입력값이 여러 개인 경우(multiple feature) 다음과 같이 확장될 것이다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.4) |

위에서

|  |  |  |
| --- | --- | --- |
|  |  | (6.5) |
|  |  | (6.6) |

입력 변수, 즉 feature의 개수가 *n*이라면 traning 을 통해 결정해야 할 변수는 *n*+1이다. 크기 의 traning set을 적용하면 개의 모델값이 계산될 것이고 대응하는 해가 존재한다. 행렬 표현식으로 표현하면 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.7) |

최소화 함수는 다음과 같이 표현된다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.8) |

Traning set을 Python 코드로 직접 정의하거나, 외부 파일로 저장된 데이터에서 읽어 X와 Y를 구성할 때 식 (6.7)과 식 **오류! 참조 원본을 찾을 수 없습니다.** 중 어떤 형태를 사용하는지에 대해 조심해야 한다. 일반적으로 식 **오류! 참조 원본을 찾을 수 없습니다.**보다는 식 (6.7)이 표현이 간결하여 많이 사용된다.

Code 6.12는 Code 6.9에 대한 행렬표현식을 이용한 코드이다. xData의 첫번째 행이 모두 1.을 지정하였고, yData를 2차원 배열에서 (4,1) 크기가 갖도록 한 점에 주의해야 한다. Placeholder 역시 [None,2]로 2차원 배열로 지정했는데 이때 None은 크기 변경될 수 있는 traing set 의 개수를 의미한다.

Code 6.12 3-LinearRegressionMatrix.py

import tensorflow as tf

# Caution

xData = [[1.,1.],

[1.,2.],

[1.,3.],

[1.,4.]]

yData = [[1.1],

[1.9],

[2.9],

[4.1]]

# NOTE. yData = [1.1,1.9,2.9,4.1] gives errors

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

Y = tf.placeholder(tf.float32)

W = tf.Variable(tf.random\_uniform([2,1], -1.0, 1.0))

H = tf.matmul(X,W)

# Define objective(=cost=loss)

cost = tf.reduce\_mean(tf.square(H-Y))

# Alternative .... Define train object with some optimizer

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(cost)

init = tf.global\_variables\_initializer()

feed = {X:xData,Y:yData}

with tf.Session() as sess:

sess.run(init)

for step in range(1001):

sess.run(train,feed\_dict={X:xData,Y:yData})

if step % 20 == 0:

print(step, sess.run(cost,feed\_dict=feed),sess.run(W))

WW = sess.run(W)

# W = [0 1]

특히 주의할 점은 yData = [1.1,1.9,2.9,4.1] 과 정의하면 오류가 발생한다는 점이다. 이렇게 선언할 경우yData는 내부적으로 (4,) shape을 갖는 ndarray로 변환되는데 이경우 비용함수를 구할 때 H-Y에서 (4,1) - (4,) 의 연산이 (4,4) 행렬로 broadcasting 되기 때문이다.

yData = [1.1,1.9,2.9,4.1]

...

H = tf.matmul(X,W) # (4,1) ndarray

...

Z=H-Y # (4,1) - (4,)는 (4,1) - (1,4)로 취급되어 broadcasting되며 (4,4) 생성

다중입력(multi feature)를 갖는 경우 역시 Code 6.12는에서 입력 데이터를 정의하는 부분과 X placeholder, W의 차원만 재정의하면 된다. Code 6.13은 2개의 입력값을 갖는 경우로 확장한 코드이다. 음영처리된 영역만 변경되었다

Code 6.13 3-LinearRegressionMulti.py

import tensorflow as tf

# Input data

# x1 x2 y

# 1 0 1

# 0 2 2

# 3 0 3

# 0 4 4

# 5 0 5

xData = [[1., 1.,0.],

[1., 0.,2.],

[1., 3.,0.],

[1., 0.,4.],

[1., 5.,0.]]

yData = [[1.],

[2.],

[3.],

[4.],

[5.]]

**X = tf.placeholder(tf.float32,[None,3])**

Y = tf.placeholder(tf.float32)

**W = tf.Variable(tf.random\_uniform([3,1], -1.0, 1.0))**

H = tf.matmul(X,W)

# Define objective(=cost=loss)

cost = tf.reduce\_mean(tf.square(H-Y))

# Alternative .... Define train object with some optimizer

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(cost)

init = tf.global\_variables\_initializer()

feed = {X:xData,Y:yData}

with tf.Session() as sess:

sess.run(init)

for step in range(2001):

sess.run(train,feed\_dict={X:xData,Y:yData})

if step % 20 == 0:

print(step, sess.run(cost,feed\_dict=feed),sess.run(W))

WW = sess.run(W)

# W = [0 1 1]

Traing set이 클 경우 일반적으로 외부 파일로 정의된다. 만약 Code 6.13의 데이터가 train1.txt라는 파일에 정의되어 있다고 가정하기로 한다.

# x0 x1 x2 y

1 1 0 1

1 0 2 2

1 3 0 3

1 0 4 4

1 5 0 5

파일을 읽어들이는 것은 NumPy의 loadtxt() 함수를 이용하면 된다. 이 함수는 ndarry 형태로 읽어 들이게 된다. 실제 코드는 Code 6.14과 같아. 주의할 점은 yData를 구할 때 reshape()을 통해 강제로 (5,1) 형태로 변환해야 한다는 점이다.

Code 6.14 7-LinearRegressionFile.py

# -\*- coding: utf-8 -\*-

import tensorflow as tf

import numpy as np

data = np.loadtxt('train1.txt')

xData = data[:,0:3]

yData = data[:,3].reshape(len(data),1) # Caution

X = tf.placeholder(tf.float32,[None,3])

Y = tf.placeholder(tf.float32)

W = tf.Variable(tf.random\_uniform([3,1], -1.0, 1.0))

H = tf.matmul(X,W)

# Define objective(=cost=loss)

cost = tf.reduce\_mean(tf.square(H-Y))

# Alternative .... Define train object with some optimizer

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(cost)

init = tf.global\_variables\_initializer()

feed = {X:xData,Y:yData}

with tf.Session() as sess:

sess.run(init)

for step in range(2001):

sess.run(train,feed\_dict={X:xData,Y:yData})

if step % 20 == 0:

print(step, sess.run(cost,feed\_dict=feed),sess.run(W))

WW = sess.run(W)

# W = [0 1 1]

## Logistic Regression

Logistic Regression은 ML에서 사용되는 classification 기법 중에 하나이다. 입력 변수가 여러 개인 경우 Linear Regression 역시 가능하다. ML에서는 입력은 feature라고 표현한다. 따라서 multiple feature를 가질때의 Linear Regression이다. 예를 들어 학습에 투자한 시간(x1)과 수업에 참석한 회수(x2)가 있을 때 시험 점수(y)라는 데이터가 있을 수 있다. 이 경우 (x1,x2)에 대해 시험 점수를 회귀(regression)을 통해 예측할 수 있다. 만약 시험 결과가 성공/실패 로 주어지거나 A,B,C,D,E오 같은 등급으로 주어진다면 이 문제는 classification 문제가 되며 이 경우 적용가능한 기법 준 하나가 logistic regression이다.

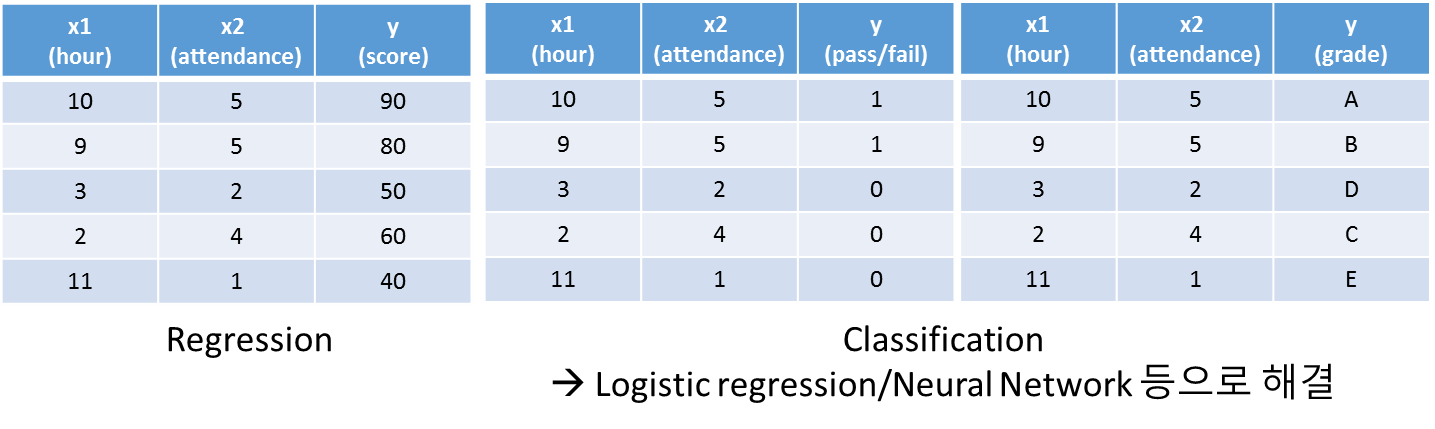


Figure 6.2 Gradient Descent Optimizer

### Binary Logistic Regression

Y의 값이 0 또는 1로 제한되는 문제는 Logistic Regression으로 ML을 수행할 수 있다. Logistic Regression은 Y가 연속(continuous)일 때 사용하는 Linear Regression에서 모델(model, hypothesis)과 비용함수(cost function)을 적절하게 변경하면 된다.

Figure 6.3(a)과 같은 변수가 1개인 데이터가 있다고 가정하기로 한다. Figure 6.3(b)는 linear regression을 적용한 것이다. 찾아낸 모델 함수에서 성공과 실패를 0.5를 기준으로 한다면 5 근처의 데이터에 대해 잘못된 추정이 됨을 알 수 있다. 이러한 이유로 Figure 6.3(c)와 같이 0과 1사이에서 정의되는 새로운 지수함수를 정의하게 된다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.9) |

또는 다음과 같이 표기하기도 한다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.10) |

위에서 를 Sigmoid function 또는 Logistic Function이라고 한다. Logistic regression에서는 항상 label 값이 0 또는 1이다(). 이때 모델함수 는 일 확률을 의미한다. 수식으로는 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.11) |

Figure 6.4는 logistic regression을 그래프 표현방식으로 나타낸 것이다

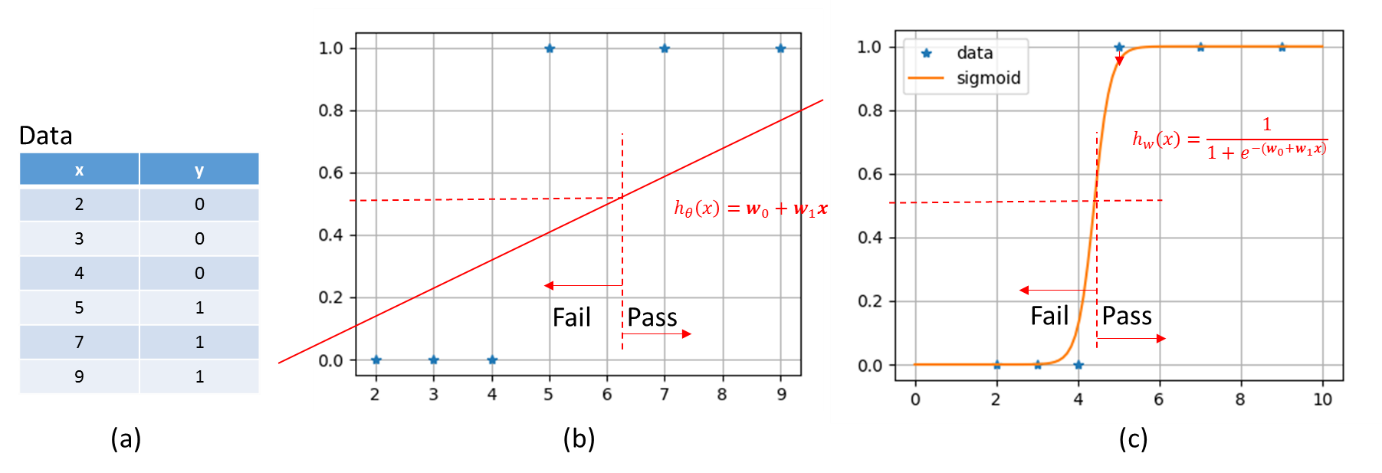


Figure 6.3 Binary classisfication; (a) dataset (b) linear regression, and (c) logistic regression

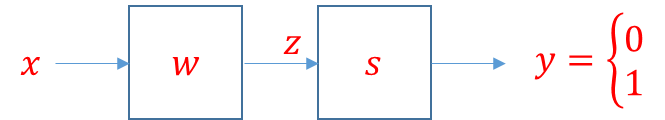


Figure 6.4 Graph representation of Logistic regression

Logistic regression에서는 training의 대상인 비용함수를 최적화 알고리즘의 안정성(예를 들어 steepest descent에서 local minima에 빠지지 않도록)을 이유로 다음과 같은 대수함수를 사용한다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.12) |

위에서 c() 함수는 y가 0일 때 모델이 0을 예측하면 0을 1을 예측하면 무한대를 선택한다. 반대로 y가1이면 1을 예측하면 0을, 0을 예측하면 무한대가 되는 함수이다. 보다 간결하게 쓰면 또는 이므로 다음과 같이 쓸 수 있다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.13) |

결과적으로 식 (6.12)는 Figure 6.5 에 나타낸 것과 같이 비용함수를 계산하게 된다.

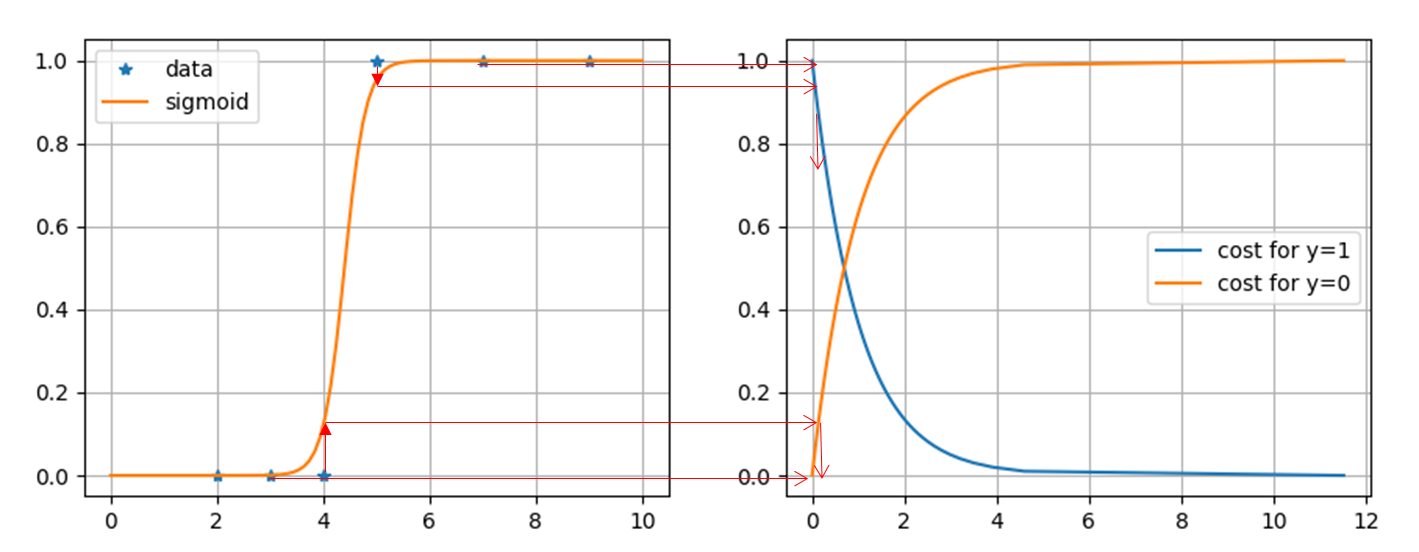


Figure 6.5 Binary classisfication by logistic regression

행렬표현식으로 표현할 경우 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.14) |

|  |  |  |
| --- | --- | --- |
|  |  | (6.15) |

Code 6.15 2-LogisticRegression.py

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

# data = np.loadtxt('BinaryClassification.txt',dtype='float32')

data = np.loadtxt('PassFail.txt',dtype='float32')

(ndata,nv) = data.shape

nv = nv-1

xData = data[:,0:nv];

yData = data[:,nv].reshape(ndata,1)

X = tf.placeholder(tf.float32,[None,nv])

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

W = tf.Variable(tf.random\_uniform([nv,1], -1.0, 1.0))

z = tf.matmul(X,W)

H = 1./(1.+tf.exp(-z))

f = tf.reduce\_mean(-Y\*tf.log(H) - (1-Y)\*tf.log(1-H))

optimizer = tf.train.GradientDescentOptimizer(0.1) # learning rate

train = optimizer.minimize(f)

init = tf.global\_variables\_initializer()

sess= tf.Session()

sess.run(init)

steps = []

objs = []

feed = {X:xData,Y:yData}

for step in range(4001):

sess.run(train,feed\_dict=feed)

objs.append(sess.run(f,feed\_dict=feed)); steps.append(step)

if step % 20 == 0:

print(step,sess.run(f,feed\_dict=feed),sess.run(W))

#4000 0.329686 [[-7.37101936]

# [ 0.52562273]

# [ 1.29136634]]

### Decision Boundary

Y의 값이 0 또는 1로 제한되는 문제는 Logistic Regression으로 ML을 수행할 수 있다. Logistic Regression은 Y가 연속(continuous)일 때 사용하는 Linear Regression에서 모델(model, hypothesis)과 비용함수(cost function)을 적절하게 변경하면 된다.

Logistic Regression을 수행할 경우 feature(X) 측면에서 보면 그림과 같이 구분한 것이 된다.

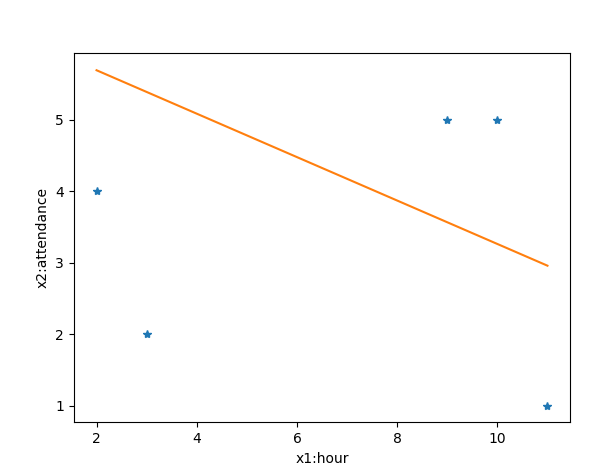


Figure 6.6 Descision boundary

### Multinomial Logistic Regression : Softmax

Multinomial Logistic Regression 문제는 binary logistic regression의 문제의 조합으로 생각할 수 있다. Figure 6.7과 같이 세가지 분류로 레이블(Y값)이 주어지는 경우, 즉 Y={0,1,2} 인 경우 binary logisitic regression을 3번 적용하고 이때 가장 큰 확률로부터 예측이 가능하다.

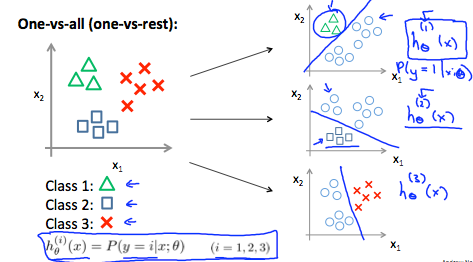


Figure 6.7 Multinomial Logistic Regression (Softmax) (source : https://www.coursera.org/learn/machine-learning/supplement/HuE6M/multiclass-classification-one-vs-all)

입력 feature가 개이고, 개로 분류하는 문제를 생각해 보기로 한다. 이 경우 레이블은 로 정의된다. 따라서 모델함수는 각각의 레이블에 대해 정의가능하며 이때 가장 큰 확률로 선택이 가능할 것이다.

|  |  |  |
| --- | --- | --- |
| Given: |  | (6.16) |
|  |  |  |
| Model |  | (6.17) |
| Prediction |  | (6.18) |

위 식에서 결정해야할 모델 파라미터는 개이다. Traing set의 크기가 *m*일 때 비용함수는 다음과 같이 정의된다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.19) |

위에서 은 indicator function인데 1{a true statement} = 1이고, 1{a false statement} = 0인 함수이다. 즉, 은 이면 1이고 그렇지 않으면 0이다.

행렬 표현식으로 표현한다면 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.20) |
|  |  | (6.21) |
|  |  | (6.22) |
|  |  | (6.23) |
|  |  | (6.24) |

위에서 Y는 one-hot-encoding 기법을 이용해 표현한 것이다. 즉, Y = [0,1,2] 는 Y = [ [1 0 0 ],[0 1 0], [0 0 1]] 등으로 표현하는 것이다

Multinomial logistic regression은 binary logistic regression의 단순한 확장이다. 즉, 만약 k=2인 경우이고, 로 분류한다면 다음과 같이 쓸 수 있다.

|  |  |  |
| --- | --- | --- |
| Given: |  | (6.25) |
|  |  |  |
| Model: |  | (6.26) |
| Cost: |  | (6.27) |

위에서 모델함수는 y=1일 확률은 다음과 같다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.28) |

만약 이고, 한 개의 y=1일 1개의 모델함수를 정의하면 모델 및 비용함수는 다음과 같이 binomial logistic regression과 같아진다.

|  |  |  |
| --- | --- | --- |
|  |  | (6.29) |

|  |  |  |
| --- | --- | --- |
|  |  | (6.30) |

### TensorFlow의 함수들

Tensorflow에서 주로 사용하는 함수로 다음의 함수가 있다.

reduce\_mean(input\_tensor, axis=None, keep\_dims=False, name=None)

reduce\_sum(input\_tensor, axis=None, keep\_dims=False, name=None)

reduce의 의미는 입력되는 텐서의 차원이 줄어든다는 의미이다. axis가 None이며 tensor의 모든 요소에 대해 합을 구하거나 평균을 구한다. axis에 0,1,… 등의 숫자를 사용하면 주어진 값의 차원에 따라 연산을 수행한다. Keep\_dims는 입력 텐서의 차원을 유지할 것인가를 결정한다.

>>> x = [[1.,1.], # (2,2) 2-dimensional array

[2.,2.]]

>>> session = tf.Session()

>>> session.run( tf.reduce\_mean(x) ) # return scalar

1.5

>>> session.run( tf.reduce\_mean(x,0) ) # return 1-d array with (2,)

array([ 1.5, 1.5], dtype=float32)

>>> session.run( tf.reduce\_mean(x,1) ) # return 1-d array with (2,)

array([ 1., 2.], dtype=float32)

>>> session.run( tf.reduce\_mean(x,0,True) ) # return 2-d array with (1,2)

array([[ 1.5, 1.5]], dtype=float32)

>>> session.run( tf.reduce\_mean(x,1,True) ) # return 2-d array with (1,2)

array([[ 1.],

[ 2.]], dtype=float32

또 다른 함수는 softmax() 이다.

softmax(logits, dim=-1, name=None)

이 함수는 풀어쓰면 다음과 같다.

softmax = exp(logits)/ reduce\_sum(exp(logits), axis=-1, keep\_dims=True)

### Implementation of Softmax

실제 Index function을 사용하는 대신 one-hot-encoding이라는 기법을 사용한다.

로 다음과 같이 행렬 표현식으로 표현할 수 있다.

Code 6.16 4-Softmax.py

# -\*- coding: utf-8 -\*-

import tensorflow as tf

import numpy as np

xy = np.loadtxt('MultiClassification.txt',dtype='float32')

xData = xy[:,0:3]; # (8,3)

yData = xy[:,3:]; # (8,3)

#X = np.array(xData)

#Y = np.array(yData)

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

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

#W = tf.Variable(tf.zeros([3,3]))

W = tf.Variable(tf.random\_uniform([3,3], -1.0, 1.0))

H = tf.nn.softmax(tf.matmul(X,W)) # (8,3)\*(3,3) -> (8,3)

cost = tf.reduce\_mean(- tf.reduce\_sum(Y\*tf.log(H),axis=1) ) # (8,3) -> (8,)

learning\_rate = 0.01

optimizer = tf.train.GradientDescentOptimizer(learning\_rate) # learning rate

train = optimizer.minimize(cost)

init = tf.global\_variables\_initializer()

session = tf.Session()

session.run(init)

for step in range(2001):

session.run(train ,feed\_dict={X:xData,Y:yData})

if step % 20 == 0:

print(step,session.run(cost,feed\_dict={X:xData,Y:yData}),session.run(W))

a = session.run(H,feed\_dict={X:[[1,11,7]]})

print("a : ", a,session.run(tf.arg\_max(a,1)))

b = session.run(H, feed\_dict={X: [[1, 3, 4]]})

print("b :", b, session.run(tf.arg\_max(b, 1)))

c = session.run(H, feed\_dict={X: [[1, 1, 0]]})

print("c :", c, session.run(tf.arg\_max(c, 1)))

## Tips

### Learning rate

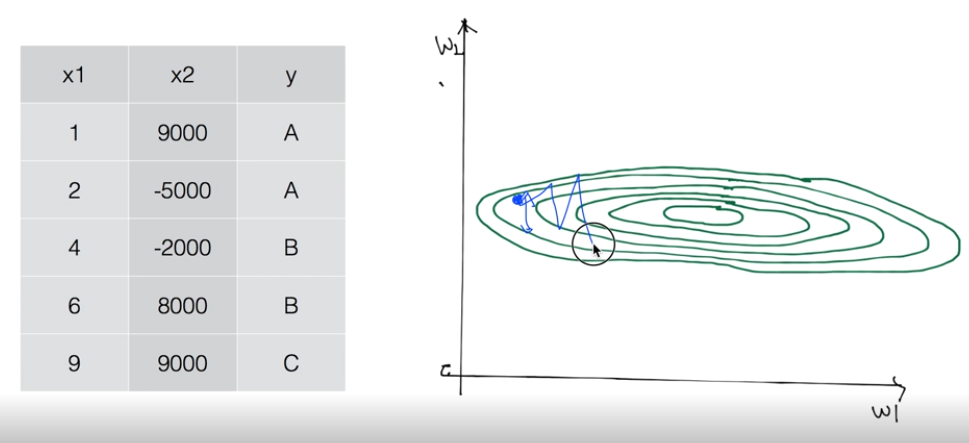
Learning rate가 너무 크면 수렴이 안되고 발산함. nan이 나오는 데 이를 overshooting이라고 함.

너무 작으며 계산 시간이 오래 걸리고, local minimum에 빠질 수 있음. 코스트 함수를 관찰하여 너무 작게 줄어들면 크게 할 것.

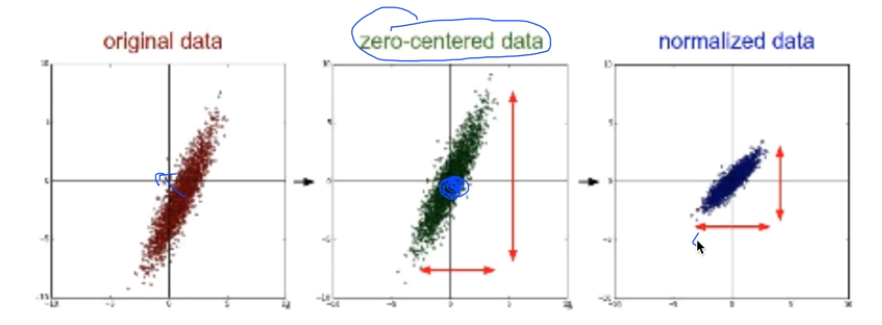
일반적으로 0.01 정도로 시작하여 발산하면 작게, 너무 늦게 수렴하면 줄이는 방법

### Data preprocessing

데이터의 order가 너무 차이나면 normalize 등의 사전 작업이 필요



위에서 x1와 x2는 order가 너무 차이남.

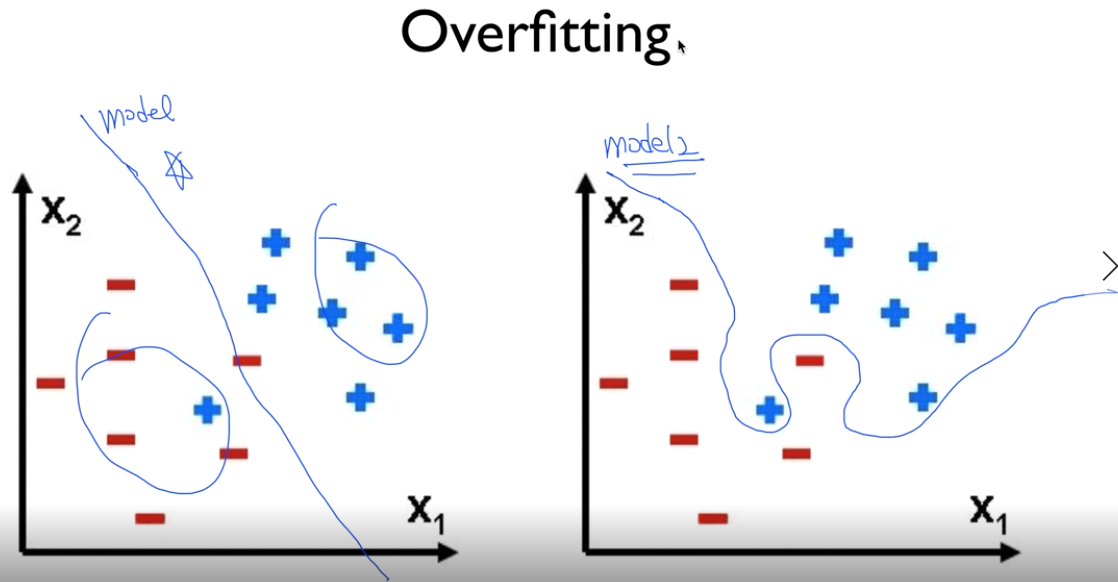


일반적으로 zero center(값이 중심에 있도록), normalize(어떤 범위안에 있도록) 등의 기법을 사용함.

x\_std[:,0] = (X[:,0]-X[:,0].mean()) / X[:,0].std()

### Overfitting

너무 학습데이터에 잘맞는 training을 시켰지만 실제로는 잘 예측을 못하는 것을 의미

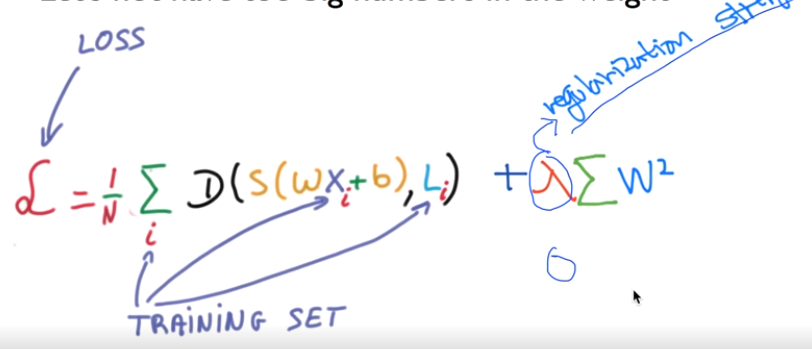


해결책

-training set이 크면 좋음.

- feature의 수를 줄인다(특히 중복되는 것을 제거)

- Regularization : 일반화시킨다는 의미. Weight의 값의 차이가 너무 나지 않도록 하자. 수학적으로 penalty 를 적용하는 것



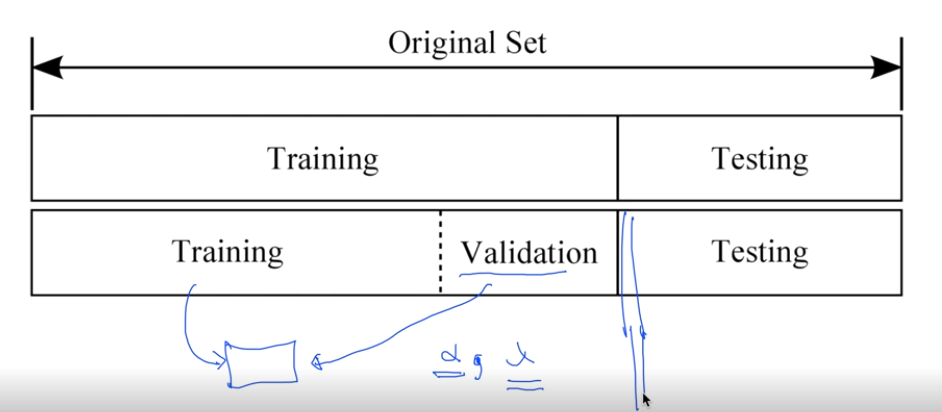
L2reg = 0.001\*tf.reduce\_sum(tf.square(W))

이후 코스트에 더해서 사용함.

### Training set과 test sets

보통 데이터의 70%를 training set, 30%를 test set으로 사용하여 performance를 evaluation 한다. 또는 validation set을 추가로 둘수 있다.

Validation set은 , 를 튜닝할 필요가 있을 때 사용



### Online learning

Training set이 너무 큼. 이를 메모리에 올리기 힘든 경우. 데이터를 잘라서 각각을 학습시킴. 이전 학습데이터가 남아 있어야 함.

예. 100만개의 데이터🡪 10만개 추가 🡪 추가 학습

### Accuracy

이미지 인식분야는 요즘 95% 이상

## TensorBoard 사용법

## LAB : Softmax 를 이용한 이미지 분석

## Neural Network

## Deep Learning