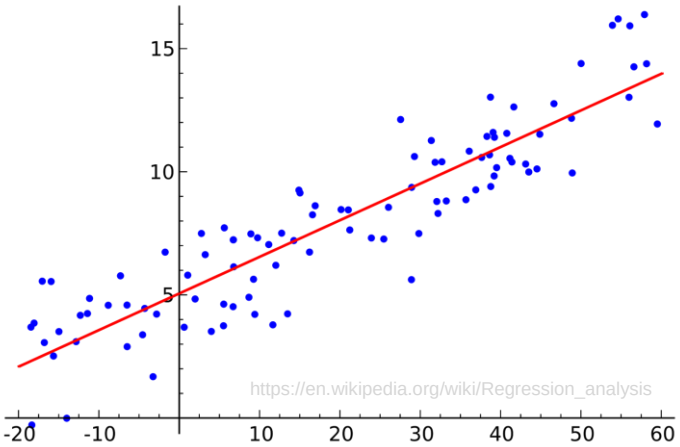
Linear Regression

Regression Analysis

Regression Analysis(회귀분석) 이란?

- Regression Analysis (회귀분석)
  - 관찰된 연속형 변수들에 대해 두 변수 사이의 모형을 구한 뒤 적합도를 특정해 내는 분석 방법

- 기본 동작 원리
  - 데이터들의 특성을 파악
  - 경향성(Tendency) 및 의존성(Dependency)을 수식으로 작성
  - 앞으로 발생할 일을 예측(Prediction)



Regression Analysis 데이터

- Regression Analysis 데이터 특징
  - 분석을 통해 나온 예측값과 실제 데이터 오차는 모든 데이터값(독립변수)에 대하여 동일한 분산을 가지고 있음
  - 데이터의 확률 분포는 정규분포를 이룸
  - 독립변수 상호간에는 상관 관계가 없음(선형적으로 독립)
  - 독립변수와 종속변수 사이에는 상관 관계가 존재(선형관계)

❏ **상관관계** : 두 변수 a, b 가 있을 때 a값이 증가하거나 감소할때 b의 값도 a 값의 영향으로 증가하거나 감소하게 됨
❏ **독립변수** : 다른 변수에 영향을 받지 않는 변수
❏ **종속변수** : 독립변수에 영향을 받아서 변화하는 변수

예) 시험공부를 한 시간의 크기와 시험 결과의 상관관계를 분석
독립변수 : 시험공부를 한 시간
종속변수 : 시험의 결과

선형(Linearity),비선형(Non Linearity) 데이터 모델

- 선형(Linearity) 데이터 모델
  - 독립변수 x와 종속변수 y간의 관계가 1차식(y = ax +b)으로 표현되는 것이 아님
  - 직선(1차식)이 아닐지라도 직선의 특징을 가지는 데이터 모델
  - 중첩의 원리가 적용됨 : 입력값과 출력값이 비례성 및 가산성을 가지게 되어 어느정도 예측이 가능함
    - 두식은 동일한 가산성을 가진 수식이며, 동일한 결과를 가짐

      $y = a(x_1 + x_1)$ 독립변수 x가 개별적으로 더함

      $y = ax_2 + ax_2$ 독립변수 x를 합쳐서 더함

    - y = ax + b 는 독립변수의 회귀계수인 기울기 ‘a’에 관하여 비례성을 가진 수식임

- 비선형(Non Linearity) 데이터 모델
  - 데이터(독립변수, 종속변수)를 변형하더라도 중첩의 원리를 적용한 수식으로 데이터 표현 할 수 없는 모델
  - 비선형 모델 데이터 수식

    $$y = \frac{ax}{b + x}$$

- 선형 회귀 모델은 파라미터(회귀계수)가 선형식으로 표현되는 회귀 모델
- 비선형 모델은 복잡한 패턴을 가지고 있기 때문에 예측이 불가능하며 이런 문제는 Deep Learning을 이용함

Regression Analysis 데이터 모델 분류

- 데이터의 특성에 따른 분류
  - Linear Regression Analysis Model(선형 회귀분석 모델)
  - NonLinear Regression Analysis Model(비선형 회귀분석 모델)

- 독립변수 개수에 따른 분류
  - Simple Regression Analysis Model(단순 회귀분석 모델) : 독립변수 1개
  - Multiple Regression Analysis Model(다중 회귀분석 모델) : 독립변수 2개 이상

- 종속변수 개수에 따른 분류
  - Univariate Regression Analysis Model(단변량 회귀분석 모델) : 종속 변수 1개
  - Multivariate Regression Analysis Model(다변량 회귀분석 모델) : 종속 변수 2개 이상

Regression Analysis 데이터 모델 분류

- 선형 회귀 모델 구분
  - Univariate Simple Linear Regression Analysis Model (단변량 단순 선형 회귀분석 모델)
    : 독립변수 1, 종속변수 1

  - Univariate Multiple Linear Regression Analysis Model(단변량 다중 선형 회귀분석 모델)
    : 독립변수 2개 이상, 종속변수 1개

  - Multivariate Simple Linear Regression Analysis Model(다변량 단순 선형 회귀분석 모델)
    : 독립변수 1개, 종속변수 2개 이상

  - Multivariate Multiple Linear Regression Analysis Model(다변량 다중 선형 회귀분석 모델)
    : 독립변수 2개 이상, 종속변수 2개 이상

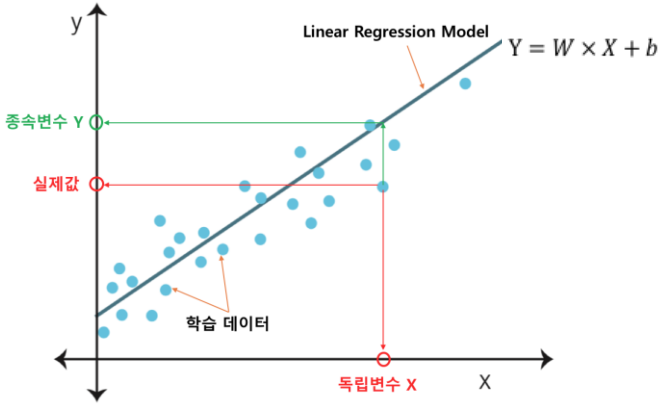
- Linear Regression Analysis를 위하여 데이터들의 특성을 파악하여 선형 회귀 모델을 수식을 만들어야 함

Single Variable Linear Regression

Single Variable Linear Regression

- Linear Regression Analysis 목표
  종속변수 Y(결괏값)와 독립변수 X(입력값)의 선형적 특성을 가지는 상관관계 모델을 생성하여 새로운 독립변수 X에 대한 결과를 예측

- Linear Regression Model
  - 학습 데이터 : 그래프에 표시된 점
  - 학습 데이터의 특성을 대표하는 모델 : 그래프의 직선
  - 독립변수 X(입력값) : 학습데이터의 x 축 값
  - 실제값 : 학습데이터의 y 축 값
  - 종속변수 Y(결괏값 or 예측값 or 가설값) : 학습 모델(직선) 수식에 대입한 독립변수 X의 값



Single Variable Linear Regression

- Linear Regression Model 수식
  - $Y = W \times X + b$
  - 가설 수식이라고도 하며 독립변수 X의 입력값을 넣어 계산되 결괏값은 예측값이라고 함
  - W(Weight) 가중치라고 하며, b(bias)편향이라고 함

Single Variable Linear Regression 실습

- **학습에 필요한 모듈 선언**

```
1  ######################################################################
2  # [학습에 필요한 모듈 선언]
3  ######################################################################
4  import tensorflow as tf
5  import numpy as np
6  from matplotlib import pyplot as plt
7
```

  - tensorflow : 모델을 생성 하기 위한 라이브러리
  - numpy : 행렬이나 일반적인 대규모 다차원 배열을 처리하는 라이브러리
  - matplotlib : 그래프 생성 라이브러리

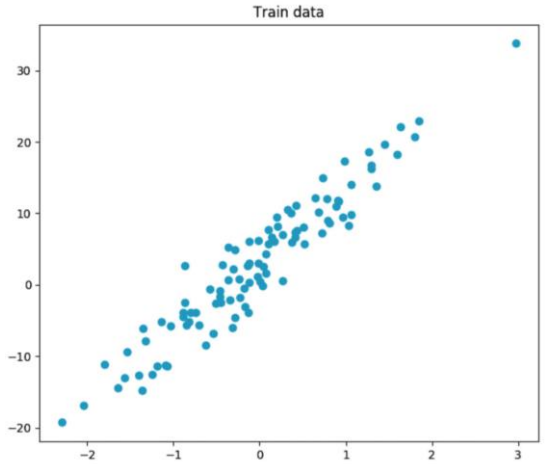
Single Variable Linear Regression 실습

- **환경설정**

```
8   ######################################################################
9   # [환경설정]
10  ######################################################################
11  # 훈련용 데이터 수 선언
12  trainDataNumber = 100
13  # 모델 최적화를 위한 학습률 선언
14  learningRate = 0.01
15  # 총 학습 횟수 선언
16  totalStep = 1001
17
```

  - 학습 데이터를 랜덤하게 생성하여 학습을 하기 위한 훈련용 데이터 수를 지정
  - 학습률은 학습 모델 수식의 W(Weight)와 b(bias)의 최적의 값을 찾기 위한 최적화 함수의 입력 파라미터임
  - 최적의 모델을 만들기 위한 총 학습 횟수 선언함

Single Variable Linear Regression 실습

- **빌드단계 - Step1) 학습 데이터 준비**
  - 모델 학습을 위한 학습데이터 생성
  - numpy.random.normal() 을 이용하여 평균 0, 표준편차 1을 가지는 학습데이터 x(독립변수 x) 100개 생성
  - 학습데이터의 실제 값을 구하기 위하여 y=10x+3+np.random.normal(0.0,3) 식에 학습데이터 x 를 대입하여 계산
  - 


```
18  ##################################################################
19  # [빌드단계]
20  # Step 1) 학습 데이터 준비
21  ##################################################################
22  # 항상 같은 난수를 생성하기 위하여 시드설정
23  np.random.seed(321)
24
25  # 학습 데이터 리스트 선언
26  xTrainData = list()
27  yTrainData = list()
28
29  # 학습 데이터 생성
30  xTrainData = np.random.normal(0.0, 1.0, size=trainDataNumber)
31
32  for x in xTrainData:
33      # y 데이터 생성
34      y = 10 * x + 3 + np.random.normal(0.0, 3)
35      yTrainData.append(y)
36
37  # 학습 데이터 확인
38  plt.plot(xTrainData, yTrainData, 'bo')
39  plt.title("Train data")
40  plt.show()
41
```

Single Variable Linear Regression 실습

- **빌드단계 - Step2) 모델 생성을 위한 변수 초기화**

```
42  ##################################################################
43  # [빌드단계]
44  # Step 2) 모델 생성을 위한 변수 초기화
45  ##################################################################
46  # Weight 변수 선언
47  W = tf.Variable(tf.random_uniform([1]))
48  # Bias 변수 선언
49  b = tf.Variable(tf.random_uniform([1]))
50
51  # 학습데이터 xTrainData가 들어갈 플레이스 홀더 선언
52  X = tf.placeholder(tf.float32)
53  # 학습데이터 yTrainData가 들어갈 플레이스 홀더 선언
54  Y = tf.placeholder(tf.float32)
55
```

  - Weight, bias 변수를 저장할 Variable 로 선언
  - 학습데이터가 들어갈 placeholder 선언
  - X : 학습데이터 x(독립변수)가 들어감
  - Y : 학습데이터 x 의 실제값(정답데이터)이 들어감

Single Variable Linear Regression 실습

- **빌드단계 - Step3) 학습 모델 그래프 구성**

```
56  ##################################################################
57  # [빌드단계]
58  # Step 3) 학습 모델 그래프 구성
59  ##################################################################
60  # 3-1) 학습데이터를 대표 하는 가설 그래프 선언
61  # 방법1 : 일반 연산기호를 이용하여 가설 수식 작성
62  hypothesis = W * X + b
63  # 방법2 :  tensorflow 함수를 이용하여 가설 수식 작성
64  #hypothesis = tf.add(tf.multiply(W,X),b)
65
66  # 3-2) 비용함수(오차함수,손실함수) 선언
67  costFunction = tf.reduce_mean(tf.square(hypothesis - Y))
68
69  # 3-3) 비용함수의 값이 최소가 되도록 하는 최적화함수 선언
70  optimizer = tf.train.GradientDescentOptimizer(learning_rate=learningRate)
71  train = optimizer.minimize(costFunction)
72
```

  - 학습 데이터의 특성을 대표하는 **가설 수식 작성**
  - 가설 수식에 학습데이터 x 의 값을 입력한 결괏값(예측값)과 실제값의 오차를 계산하는 **비용함수(오차함수, 손실 함수) 선언**
  - 비용함수의 값이 최소가 될 수 있도록 W, b의 최적값을 찾는 **최적화 함수 선언**

Single Variable Linear Regression 실습

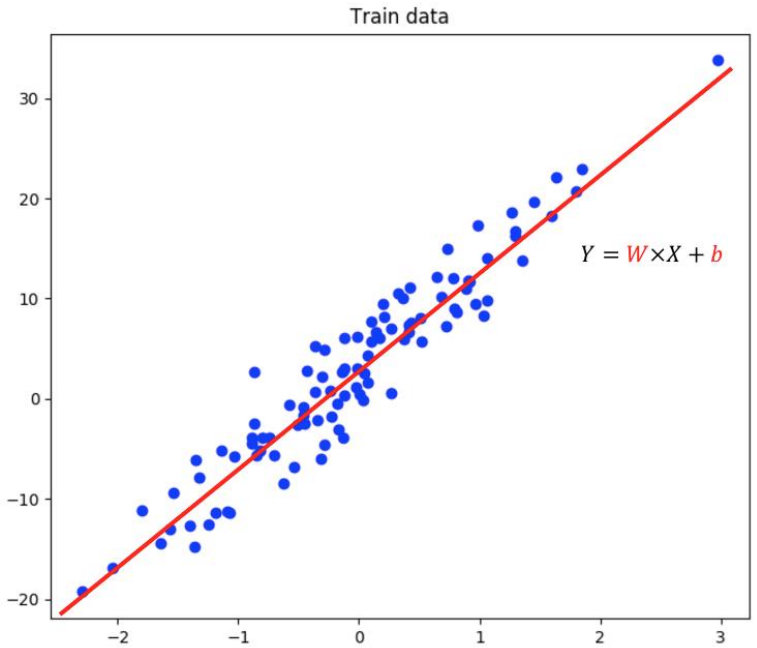
- 가설 그래프 선언
  - **목적** : 학습데이터를 대표하는 가설 그래프 선언
  - 1차 다항식의 직선 형태로 표현

    $Y = W \times X + b$
  - W : Weight, b : Bias
  - 사칙 연산을 이용하여 가설 그래프를 선언 할 수 있음

```
62  hypothesis = W * X + b
```

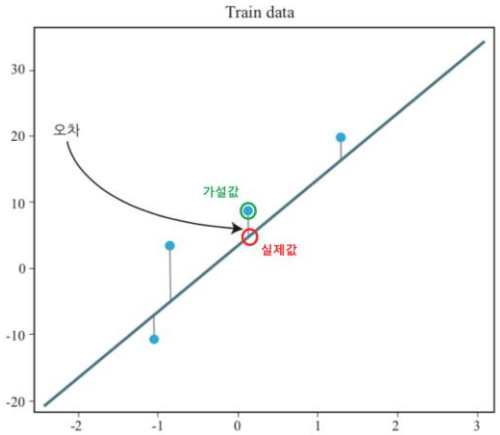
  - tensorflow를 이용하여 동일한 가설 그래프를 선언 할 수

```
64  #hypothesis = tf.add(tf.multiply(W,X),b)
```



Single Variable Linear Regression 실습

- 비용함수(오차함수, 손실함수) 선언
  - **목적** : 가설 수식에 모든 학습데이터 X값을 입력하여 나온 종속변수 y값(가설값, 예측값, 결과값) 과 학습데이터의 실제값 Y의 오차를 계산하는 수식 작성



  - 평균 제곱 오차(Mean Square Error)를 사용하여 비용함수 수식 작성

$$MSE = \frac{\sum_{i=1}^{m}(h(x_i) - y_i)^2}{m}$$

h(x) : 가설값, **y** : 실제값

Single Variable Linear Regression 실습

- 최적화 함수 선언
  - **목적** : 비용함수의 수식이 최소가 되는 W(Weight), b(Bias) 의 값을 찾는 최적화 함수 선언
  - 최적화 알고리즘 Gradient descent 사용
- 최적화 함수
  - 미분을 이용하여 스스로 최저 비용(오차)을 찾아가게 됨
  - 최적화 함수를 통하여 W, b의 변수를 변화시키게 됨
  - 오차가 최소가 되는 W, b 의 값을 찾아내는 과정을 통하여 최소 비용(오차)를 가지는 모델을 만듬
- Gradient descent 알고리즘의 θ의 변화식

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

θ : Weight, bias 변수
$J(\theta)$: 최적화 시킬 함수(비용 함수)
$\nabla_\theta J(\theta)$: 최적화 시킬 함수의 기울기

  - 최적화 시킬 함수(비용 함수)의 최솟값은 함수의 기울기가 최소가 되는 부분
  - 기울기가 0에 가까워 지는 θ의 값을 찾게 됨

Single Variable Linear Regression 실습

- Linear Regression에서 Gradient descent 알고리즘
  - 최적화 시킬 비용함수 : 가설 함수의 최소 비용(오차) 함수

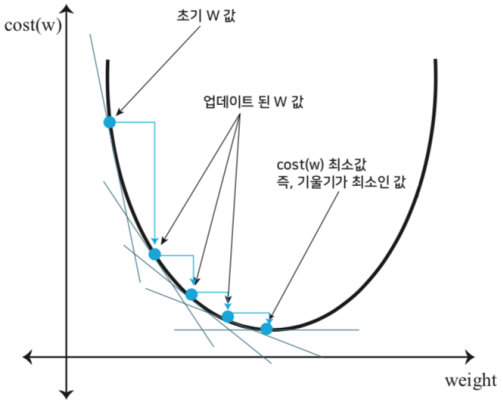
$$H(x) = \mathrm{W} * x + \mathrm{b}$$

$$cost(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(H_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

  - θ 변화식에 최적화 시킬 함수를 대입하여 정리하여 경사를 구하는 수식으로
    (θ 는 W, b 변수이지만 식을 간략하게 표현하기 위하여 영향이 적은 b 는 생략함)

$$\mathrm{W} := \mathrm{W} - \alpha\frac{\partial}{\partial \mathrm{w}}cost(w)$$ α : 학습률

$$\mathrm{W} := \mathrm{W} - \alpha\frac{1}{m}\left(Wx^{(i)} - y^{(i)}\right)x^{(i)}$$



  - 계산된 θ(W, b 변수)의 값을 업데이트 하여 최소 비용(오차)을 구하는 과정을 '머신러닝 모델학습' 이라고 함
  - tensorflow에서는 이러한 최적화 함수를 tf.train 모듈에서 제공함

Single Variable Linear Regression 실습

- **실행단계 - 학습 모델 그래프 실행**

```
73  ##############################################################
74  # [실행단계]
75  # 학습 모델 그래프를 실행
76  ##############################################################
77  # 실행을 위한 세션 선언
78  sess = tf.Session()
79  # 최적화 과정을 통하여 구해질 변수 W, b 초기화
80  sess.run(tf.global_variables_initializer())
81
82  # 비용함수 그래프를 그리기 위한 변수 선언
83  WeightValueList = list()
84  costFunctionValueList = list()
85
```

  - Session 변수(sess)를 선언
  - 최적화 과정에서 계산되는 변수(W, b)의 초기화
  - 모델 학습 결과 확인을 위한 리스트 선언

Single Variable Linear Regression 실습

- **실행단계 - 학습 모델 그래프 실행**
  - totalStep 횟수 만큼 모델이 학습됨
  - sess을 통하여 최적화 함수를 계산하여 학습 결과를 저장하고 출력함
- 학습 조건
  - 학습 데이터수 : 100개
  - 최적화 함수 : Gradient descent 알고리
  - 학습률 : 0.01
  - 학습 횟수 1,001회

```
print("--------------------------------------------------------------------------------")
print("Train(Optimization) Start ")
# totalStep 횟수 만큼 학습
for step in range(totalStep):
    # X, Y에 학습 데이터 입력하여 비용함수, W, b, train을 실행
    cost_val, W_val, b_val, _ = sess.run([costFunction, W, b, train],
                                         feed_dict={X: xTrainData,
                                                    Y: yTrainData})
    # 학습 결과값을 저장
    WeightValueList.append(W_val)
    costFunctionValueList.append(cost_val)
    # 학습 50회 마다 중간 결과 출력
    if step % 50 == 0:
        print("Step : {}, cost : {}, W : {}, b : {}".format(step,
                                                            cost_val,
                                                            W_val,
                                                            b_val))
        # 학습 100회 마다 중간 결과 Fitting Line 추가
        if step % 100 == 0:
            plt.plot(xTrainData,
                     W_val * xTrainData + b_val,
                     label='Step : {}'.format(step),
                     linewidth=0.5)
print("Train Finished")
```

Single Variable Linear Regression 실습

- **실행단계 - 학습 모델 그래프 실행**

```
110  print("------------------------------------------------------------------------------------")
111  print("[Train Result]")
112  # 최적화가 끝난 학습 모델의 비용함수 값
113  cost_train = sess.run(costFunction, feed_dict={X: xTrainData,
114                                                 Y: yTrainData})
115  # 최적화가 끝난 W, b 변수의 값
116  w_train = sess.run(W)
117  b_train = sess.run(b)
118  print("Train cost : {}, W : {}, b : {}".format(cost_train, w_train, b_train))
119  print("------------------------------------------------------------------------------------")
120  print("[Test Result]")
121  # 테스트를 위하여 x값 선언
122  testXValue = [2.5]
123  # 최적화된 모델에 x에 대한 y 값 계산
124  resultYValue = sess.run(hypothesis, feed_dict={X: testXValue})
125  # 테스트 결과값 출력
126  print("x value is {}, y value is {}".format(testXValue, resultYValue))
127  print("------------------------------------------------------------------------------------")
128
```

  - 모델 학습이 완료 후 비용함수의 값을 계산하고 W, b 변수의 값을 출력
  - 학습 결과 확인을 위하여 X 값에 2.5를 입력하여 예측값을 출력

Single Variable Linear Regression 실습

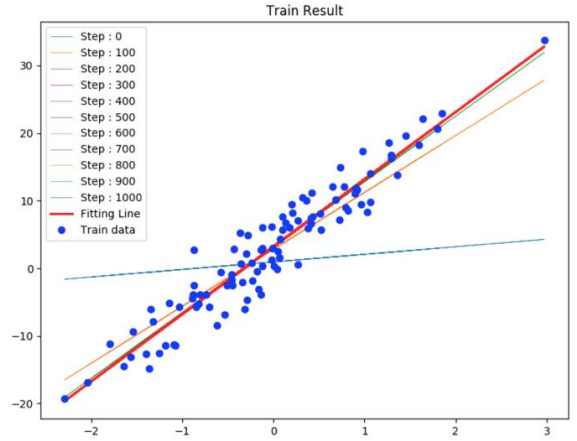
- **실행단계 - 학습 모델 그래프 실행**
  - matplotlib 를 이용하여 학습 결과 시각화

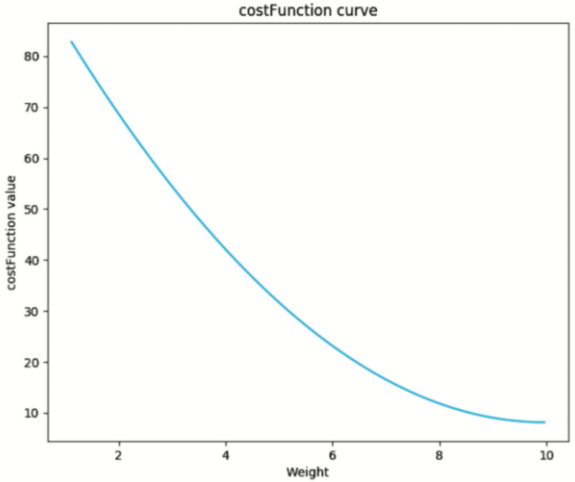
```
129  # matplotlib 를 이용하여 결과를 시각화
130  # 결과 확인 그래프
131  plt.plot(xTrainData,
132           sess.run(W) * xTrainData + sess.run(b),
133           'r',
134           label='Fitting Line',
135           linewidth=2)
136  plt.plot(xTrainData,
137           yTrainData,
138           'bo',
139           label='Train data')
140  plt.legend()
141  plt.title("Train Result")
142  plt.show()
143
144  # 비용함수 최적화 그래프
145  plt.plot(WeightValueList,costFunctionValueList)
146  plt.title("costFunction curve")
147  plt.xlabel("Weight")
148  plt.ylabel("costFunction value")
149  plt.show()
150
151  #세션종료
152  sess.close()
```

Single Variable Linear Regression 실습

- 결과 확인

```
--------------------------------------------------------------------------------
Train(Optimization) Start
Step : 0, cost : 94.79385375976562, W : [0.47239307], b : [0.63599426]
Step : 50, cost : 23.25467300415039, W : [5.9996414], b : [2.1302087]
Step : 100, cost : 10.796128273010254, W : [8.307192], b : [2.7502053]
Step : 150, cost : 8.626452445983887, W : [9.270514], b : [3.0076535]
Step : 200, cost : 8.2485990524292, W : [9.672644], b : [3.114627]
Step : 250, cost : 8.182793617248535, W : [9.840506], b : [3.1591008]
Step : 300, cost : 8.171333312988281, W : [9.910573], b : [3.1776006]
Step : 350, cost : 8.16933822631836, W : [9.939816], b : [3.1852984]
Step : 400, cost : 8.168990135192871, W : [9.952025], b : [3.1885042]
Step : 450, cost : 8.168930053710938, W : [9.95712], b : [3.1898384]
Step : 500, cost : 8.168920516967773, W : [9.9592495], b : [3.190394]
Step : 550, cost : 8.168917655944824, W : [9.960135], b : [3.1906257]
Step : 600, cost : 8.168917655944824, W : [9.9605055], b : [3.1907222]
Step : 650, cost : 8.168917655944824, W : [9.960658], b : [3.1907623]
Step : 700, cost : 8.168917655944824, W : [9.960721], b : [3.1907785]
Step : 750, cost : 8.168917655944824, W : [9.960744], b : [3.1907847]
Step : 800, cost : 8.168917655944824, W : [9.960744], b : [3.1907847]
Step : 850, cost : 8.168917655944824, W : [9.960744], b : [3.1907847]
Step : 900, cost : 8.168917655944824, W : [9.960744], b : [3.1907847]
Step : 950, cost : 8.168917655944824, W : [9.960744], b : [3.1907847]
Step : 1000, cost : 8.168917655944824, W : [9.960744], b : [3.1907847]
Train Finished
--------------------------------------------------------------------------------
[Train Result]
Train cost : 8.168917655944824, W : [9.960744], b : [3.1907847]
--------------------------------------------------------------------------------
[Test Result]
x value is [2.5], y value is [28.092644]
--------------------------------------------------------------------------------
```





Multi Variable Linear Regression

Multi Variable Linear Regression 실습

- **Multi Variable Linear Regression**
  - 목표 : 독립변수 2개, 종속 변수 1개를 가지는 Linear Regression 모델 학습
  - 직접 생성한 학습데이터를 이용하여 모델 학습

Multi Variable Linear Regression 실습

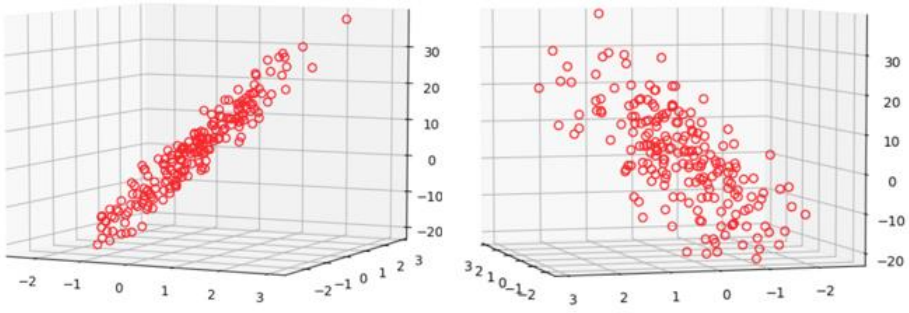
- **학습에 필요한 모듈 선언 및 환경설정**

```
#################################################################
# [학습에 필요한 모듈 선언]
#################################################################
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
# 3차원 공간에서 그래프 출력
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

#################################################################
# [환경설정]
#################################################################
# 학습 데이터 수 선언
trainDataNumber = 200
# 모델 최적화를 위한 학습률 선언
learningRate = 0.01
# 총 학습 횟수 선언
totalStep = 1001

```

  - tensorflow, numpy, matplotlib 라이브러리 사용
  - 훈련용 데이터수, 학습률, 총 학습 횟수 선언
  - 환경설정 값을 수정하여 다양한 조건의 학습을 할 수 있음

Multi Variable Linear Regression 실습

- **빌드단계 - Step1) 학습 데이터 준비**
  - 모델 학습을 위한 학습 데이터 생성
  - 독립변수 X 2개, 실제값 1개
  - y=10 * x1 + 5.5 * x2 + 3 + np.random.normal(0.0, 3)
    식에 학습데이터를 대입하여 실제 값 계산
  - 학습데이터 분포 그래프



```
21  ##############################################################
22  # [빌드단계]
23  # Step 1) 학습 데이터 준비
24  ##############################################################
25  # 항상 같은 난수를 생성하기 위하여 시드설정
26  np.random.seed(321)
27
28  # 학습 데이터 리스트 선언
29  x1TrainData = list()
30  x2TrainData = list()
31  yTrainData = list()
32
33  # 학습 데이터 생성
34  x1TrainData = np.random.normal(0.0, 1.0, size=trainDataNumber)
35  x2TrainData = np.random.normal(0.0, 1.0, size=trainDataNumber)
36
37  for i in range(0, trainDataNumber):
38      # y데이터 생성
39      x1 = x1TrainData[i]
40      x2 = x2TrainData[i]
41      y = 10 * x1 + 5.5 * x2 + 3 + np.random.normal(0.0, 3)
42      yTrainData.append(y)
43
44  # 학습 데이터 확인
45  fig = plt.figure()
46  ax = fig.add_subplot(111, projection='3d')
47  ax.plot(x1TrainData,
48          x2TrainData,
49          yTrainData,
50          linestyle="none",
51          marker="o",
52          mfc="none",
53          markeredgecolor="red")
54  plt.show()
55
```

Multi Variable Linear Regression 실습

- **빌드단계 - Step2) 모델 생성을 위한 변수 초기화**

```
56  ##################################################################
57  # [빌드단계]
58  # Step 2) 모델 생성을 위한 변수 초기화
59  ##################################################################
60  # Weight 변수 선언
61  W1 = tf.Variable(tf.random_uniform([1]))
62  W2 = tf.Variable(tf.random_uniform([1]))
63  # Bias 변수 선언
64  b = tf.Variable(tf.random_uniform([1]))
65
66  # 학습 데이터 x1TrainData, x2TrainData가 들어갈 플레이스 홀더 선언
67  X1 = tf.placeholder(tf.float32)
68  X2 = tf.placeholder(tf.float32)
69  # 학습 데이터 yTrainData가 들어갈 플레이스 홀더 선언
70  Y = tf.placeholder(tf.float32)
71
```

  - 학습데이터 X1, X2 에 대한 W(Weight) 변수 2개 선언, b(bias)변수 선언
  - 학습데이터에 대한 입력공간을 placeholder 로 선언

Multi Variable Linear Regression 실습

- **빌드단계 - Step3) 학습 모델 그래프 구성**

```
72  ##############################################################
73  # [빌드단계]
74  # Step 3) 학습 모델 그래프 구성
75  ##############################################################
76  # 3-1) 학습 데이터를 대표 하는 가설 그래프 선언
77  hypothesis = W1 * X1 + W2 * X2 + b
78
79  # 3-2) 비용함수(오차함수,손실함수) 선언
80  costFunction = tf.reduce_mean(tf.square(hypothesis - Y))
81
82  # 3-3) 비용함수의 값이 최소가 되도록 하는 최적화함수 선언
83  optimizer = tf.train.GradientDescentOptimizer(learning_rate=learningRate)
84  train = optimizer.minimize(costFunction)
85
```

  - 학습 데이터의 특성을 대표하는 **가설 수식 작성**
  - 가설 수식에 학습데이터 x 의 값을 입력한 결괏값(예측값)과 실제값의 오차를 계산하는 **비용함수(오차함수, 손실 함수) 선언**
  - 비용함수의 값이 최소가 될 수 있도록 W, b의 최적값을 찾는 **최적화 함수 선언**

Multi Variable Linear Regression 실습

- 종속 변수의 수가 2개 이상으로 많이 들어 나면 가설 수식을 행렬 곱셈(Matrix Multiplication)과 Broadcasting 으로 간단하게 작성 가능

▪ Matrix Multiplication

tensorflow에서는 행렬 곱셈은 tf.matmul() 함수를 사용합니다.
다음 수식은 행렬 곱셈의 과정입니다.

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix} \times \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

$$=$$

$$\begin{pmatrix} x_{11} \times w_{11} + x_{12} \times b_{21} + x_{13} \times b_{31} & x_{11} \times w_{12} + x_{12} \times w_{22} + x_{13} \times w_{32} \\ x_{21} \times w_{11} + x_{22} \times b_{21} + x_{23} \times b_{31} & x_{21} \times w_{12} + x_{22} \times w_{22} + x_{23} \times w_{32} \\ x_{31} \times w_{11} + x_{32} \times b_{21} + x_{33} \times b_{31} & x_{31} \times w_{12} + x_{33} \times w_{32} + x_{33} \times w_{32} \end{pmatrix}$$

행렬 x는 3x3, 행렬 w 차원은 3x2이며 두 행렬을 곱하면 3x2 차원 행렬이 출력됩니다. 두 행렬의 곱을 tensorflow로 표현하면 tf.matmul(x,w)으로 표현됩니다.

▪ Broadcasting

행렬의 연산은 행렬의 차원이 맞아야 연습이 가능합니다. 예를 들어 3x3 행렬과 3x2 행렬은 연산이 가능하지만, 3x2 행렬과 1x1 행렬은 차원이 다르기 때문에 연산을 할 수 없습니다. Broadcasting은 행렬 연산(덧셈, 뺄셈, 곱셈)에서 차원이 맞지 않으면 행렬을 자동으로 늘려서 차원을 맞춰 주는 개념입니다. 반대로 차원을 줄이는 것은 불가능합니다.

$$\begin{pmatrix} \boldsymbol{xw_{11}} & \boldsymbol{xw_{12}} \\ \boldsymbol{xw_{21}} & \boldsymbol{xw_{22}} \\ \boldsymbol{xw_{31}} & \boldsymbol{xw_{32}} \end{pmatrix} + (\boldsymbol{b}) \rightarrow \text{행렬 연산 불가능}$$

Broadcasting 적용 되면 1x1인 b 행렬이 xw 행렬과 같은 차원으로 늘어납니다.

$$\begin{pmatrix} \boldsymbol{xw_{11}} & \boldsymbol{xw_{12}} \\ \boldsymbol{xw_{21}} & \boldsymbol{xw_{22}} \\ \boldsymbol{xw_{31}} & \boldsymbol{xw_{32}} \end{pmatrix} + \begin{pmatrix} \boldsymbol{b} & \boldsymbol{b} \\ \boldsymbol{b} & \boldsymbol{b} \\ \boldsymbol{b} & \boldsymbol{b} \end{pmatrix}$$

$$=$$

$$\begin{pmatrix} xw_{11} + b & xw_{12} + b \\ xw_{21} + b & xw_{22} + b \\ xw_{31} + b & xw_{32} + b \end{pmatrix}$$

Matrix Multiplication와 Broadcasting을 이용하게 되면 독립변수, 종속변수의 수와 상관없이 간단하게 가설 수식을 선언할 수 있습니다.

Multi Variable Linear Regression 실습

- **실행단계 - 학습 모델 그래프 실행**
  - Session 변수(sess) 선언
  - 최적화 과정에서 계산되는 변수(W, b)의 초기화
  - 결과 시각화를 위한 그래프 선언

```
86  ##################################################################
87  # [실행단계]
88  # 학습 모델 그래프를 실행
89  ##################################################################
90  # 실행을 위한 세션 선언
91  sess = tf.Session()
92  # 최적화 과정을 통하여 구해질 변수 W,b 초기화
93  sess.run(tf.global_variables_initializer())
94
95  # 학습 데이터와 학습 결과를 matplotlib를 이용하여 결과 시각화
96  fig = plt.figure()
97  ax = fig.add_subplot(111, projection='3d')
98  ax.plot(x1TrainData,
99          x2TrainData,
100         yTrainData,
101         linestyle="none",
102         marker="o",
103         mfc="none",
104         markeredgecolor="red")
105
106 Xs = np.arange(min(x1TrainData), max(x1TrainData), 0.05)
107 Ys = np.arange(min(x2TrainData), max(x2TrainData), 0.05)
108 Xs, Ys = np.meshgrid(Xs, Ys)
109
```

Multi Variable Linear Regression 실습

- **실행단계 - 학습 모델 그래프 실행**
  - totalStep 횟수 만큼 모델이 학습됨
  - sess을 통하여 최적화 함수를 계산하여 학습 결과를 저장하고 출력함
- 학습 조건
  - 학습 데이터수 : 100개
  - 최적화 함수 : Gradient descent 알고리즘
  - 학습률 : 0.01
  - 학습 횟수 1,001회

```
print("-------------------------------------------------------------------------------")
print("Train(Optimization) Start")
# totalStep 횟수 만큼 학습
for step in range(totalStep):
    # X, Y에 학습데이터 입력하여 비용함수, W, b, train을 실행
    cost_val, W1_val, W2_val, b_val, _ = sess.run([costFunction, W1, W2, b, train],
                                                  feed_dict={X1: x1TrainData,
                                                             X2: x2TrainData,
                                                             Y: yTrainData})
    # 학습 50회 마다 중간 결과 출력
    if step % 50 == 0:
        print("Step : {}, cost : {}, W1 : {}, W2 : {}, b : {}".format(step,
                                                                      cost_val,
                                                                      W1_val,
                                                                      W2_val,
                                                                      b_val))
        # 학습 단계 중간결과 Fitting Surface 추가
        if step % 100 == 0:
            ax.plot_surface(Xs,
                            Ys,
                            W1_val * Xs + W2_val * Ys + b_val,
                            rstride=4,
                            cstride=4,
                            alpha=0.2,
                            cmap=cm.jet)
print("Train Finished")
print("-------------------------------------------------------------------------------")
# 결과 확인 그래프
plt.show()

#세션종료
sess.close()
```

Multi Variable Linear Regression 실습

- **결과 확인**

```
Train(Optimization) Start
Step : 0, cost : 135.67445373535156, W1 : [0.40116325], W2 : [0.58713955], b : [0.52120167]
Step : 50, cost : 27.280153274536133, W1 : [6.2737875], W2 : [3.982579], b : [2.1778917]
Step : 100, cost : 11.898796081542969, W1 : [8.582164], W2 : [5.1055026], b : [2.762643]
Step : 150, cost : 9.669946670532227, W1 : [9.494222], W2 : [5.4696126], b : [2.9672873]
Step : 200, cost : 9.339807510375977, W1 : [9.856352], W2 : [5.584425], b : [3.0381536]
Step : 250, cost : 9.289814949035645, W1 : [10.000801], W2 : [5.619148], b : [3.0623662]
Step : 300, cost : 9.282081604003906, W1 : [10.058669], W2 : [5.6289535], b : [3.0704947]
Step : 350, cost : 9.280861854553223, W1 : [10.0819435], W2 : [5.6313806], b : [3.07316]
Step : 400, cost : 9.28066635131836, W1 : [10.091336], W2 : [5.631803], b : [3.0740042]
Step : 450, cost : 9.280634880065918, W1 : [10.09514], W2 : [5.6317697], b : [3.0742583]
Step : 500, cost : 9.280630111694336, W1 : [10.096688], W2 : [5.631682], b : [3.0743284]
Step : 550, cost : 9.280628204345703, W1 : [10.097318], W2 : [5.631621], b : [3.0743444]
Step : 600, cost : 9.280628204345703, W1 : [10.097575], W2 : [5.6315866], b : [3.0743444]
Step : 650, cost : 9.280627250671387, W1 : [10.09768], W2 : [5.6315646], b : [3.0743444]
Step : 700, cost : 9.280628204345703, W1 : [10.097727], W2 : [5.631562], b : [3.0743444]
Step : 750, cost : 9.280628204345703, W1 : [10.097727], W2 : [5.631562], b : [3.0743444]
Step : 800, cost : 9.280628204345703, W1 : [10.097727], W2 : [5.631562], b : [3.0743444]
Step : 850, cost : 9.280628204345703, W1 : [10.097727], W2 : [5.631562], b : [3.0743444]
Step : 900, cost : 9.280628204345703, W1 : [10.097727], W2 : [5.631562], b : [3.0743444]
Step : 950, cost : 9.280628204345703, W1 : [10.097727], W2 : [5.631562], b : [3.0743444]
Step : 1000, cost : 9.280628204345703, W1 : [10.097727], W2 : [5.631562], b : [3.0743444]
Train Finished
```