

MACHINE LEARNING

اَللّٰهُمَّ ارْزُقْنِيْ عِلْمًا نَّافِعًا وَاسِعًا عَمِيْقًا

اَللّٰهُمَّ ارْزُقْنِيْ رِزْقًا وَّاسِعًا حَلَالًا طَيِّبًا
مُّبَارَكًا مِّنْ عِنْدِكَ

WEEK 08

UNIVERIATE LINEAR REGRESSION REVIEW

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

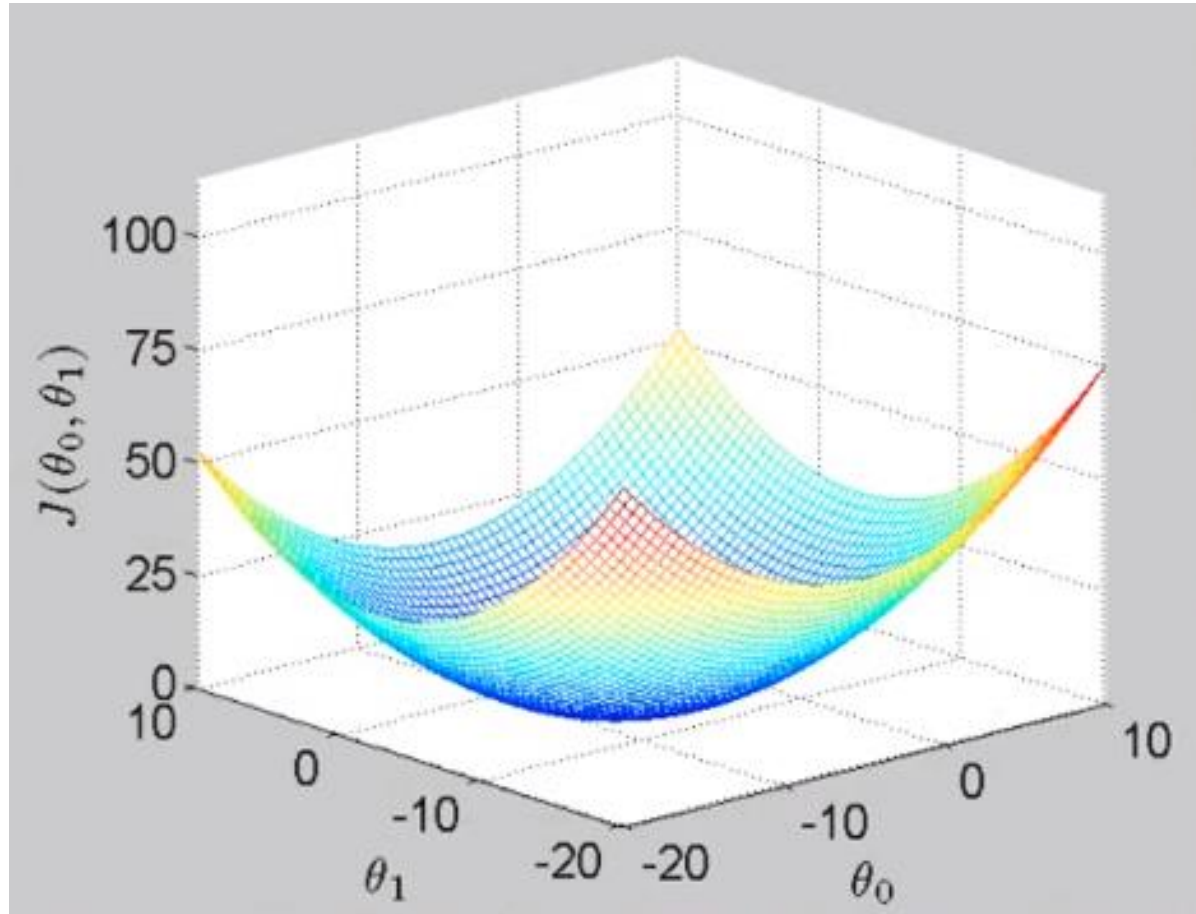
Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$

UNIVERIATE LINEAR REGRESSION REVIEW

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (y^i - (\theta_0 + \theta_1 x^i))^2$$



GRADIENT VECTOR

Scalar-valued multivariable function

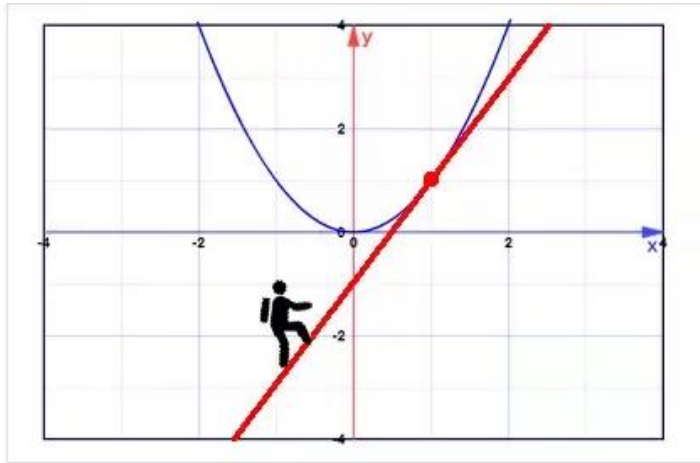
$$\nabla f(x_0, y_0, \dots) =$$

∇f takes the same type of inputs as f

Notation for gradient, called “nabla”.

$$\begin{bmatrix} \frac{\partial f}{\partial x}(x_0, y_0, \dots) \\ \frac{\partial f}{\partial y}(x_0, y_0, \dots) \\ \vdots \end{bmatrix}$$

∇f outputs a vector with all possible partial derivatives of f .



Have some function $J(\theta_0, \theta_1)$

Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum θ_0, θ_1

Gradient descent algorithm

repeat until convergence {
 $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ (for $j = 0$ and $j = 1$)
}

Correct: Simultaneous update

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
 $\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
 $\theta_0 := \text{temp0}$
 $\theta_1 := \text{temp1}$

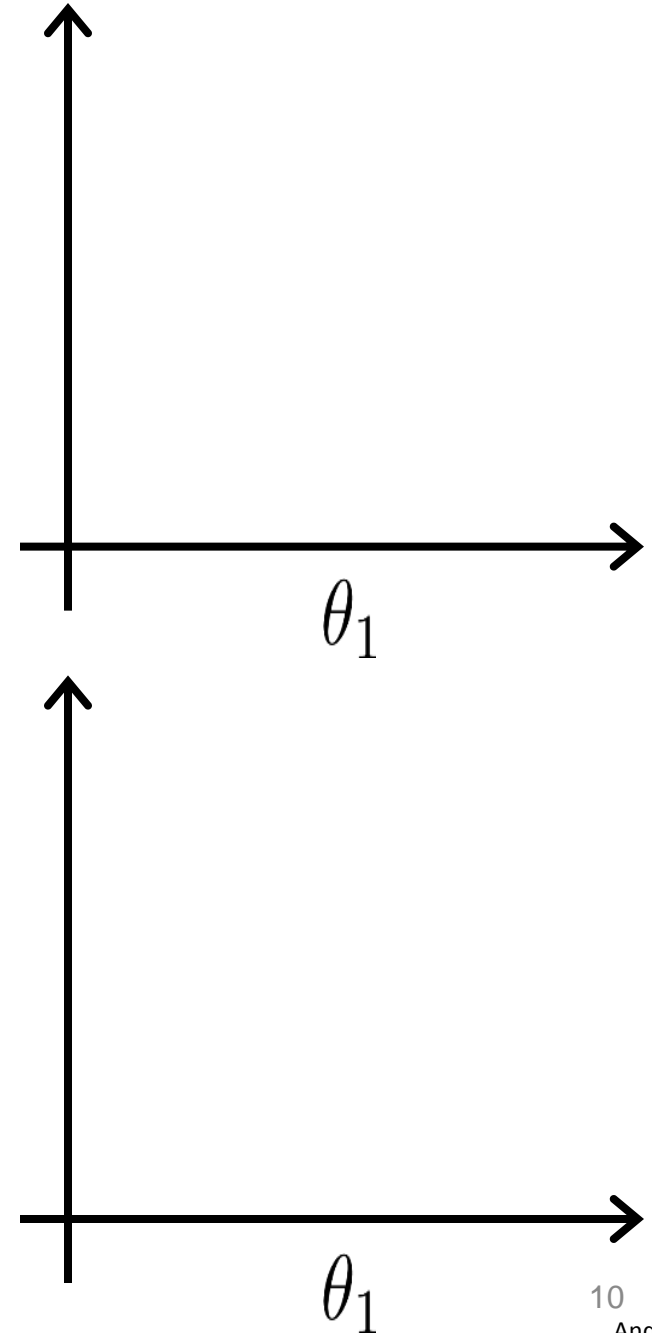
Incorrect:

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
 $\theta_0 := \text{temp0}$
 $\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
 $\theta_1 := \text{temp1}$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Gradient descent algorithm

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

update
and
 θ_0 θ_1
simultaneously

}



MULTIVARIATE LINEAR REGRESSION



Multiple features (variables).

Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...

Notation:
 n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

Hypothesis:

Previously: $h_{\theta}(x) = \theta_0 + \theta_1 x$



Machine Learning

Linear Regression with
multiple variables

GRADIENT DESCENT FOR MULTIPLE VARIABLES

Hypothesis: $h_{\theta}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \dots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

} (simultaneously update for every $j = 0, \dots, n$)

Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update θ_0, θ_1)

}

New algorithm ($n \geq 1$):

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for
 $j = 0, \dots, n$)

}

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

...

SOLVING REGRESSION WITH LINEAR ALGEBRA

STEP 01: ONE PREDICATION VECTOR

LET WE HAVE THETA'S AND
OUR OBJECTIVE IS TO FIND
THE PREDICATION VECTOR
THROUGH LINEAR ALGEBRA

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

Multivariate linear regression.

Dr Awais Hassan UET CS

Andrew NG

22

22

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

Multivariate linear regression.

Dr Awais Hassan UET CS

Andrew NG

23

23

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

Alternatively, we can also write it as

Multivariate linear regression.

Dr Awais Hassan UET CS

Multivariate linear regression.

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

Alternatively, we can also write it as

READING FROM FILE

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12

READING FROM FILE

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12

FINDING PREDICTION VECTOR

1. Read Data from File.
2. Separate X and Y columns
3. Add columns of one as X_0 at zero index of X
4. Generate Random Weights Vector w
5. Take Dot Product of X and w and store result in y_hat
6. Look at the shape of y_hat

READING FROM FILE

```
import numpy as np
import matplotlib.pyplot as plt

#Loading Data
data=np.loadtxt('advertising.csv',delimiter=',',skiprows=1)

#print first column of all rows
print(data[:,1])
#print all columns of first row
print(data[0,:])
```

VIEWING DATA

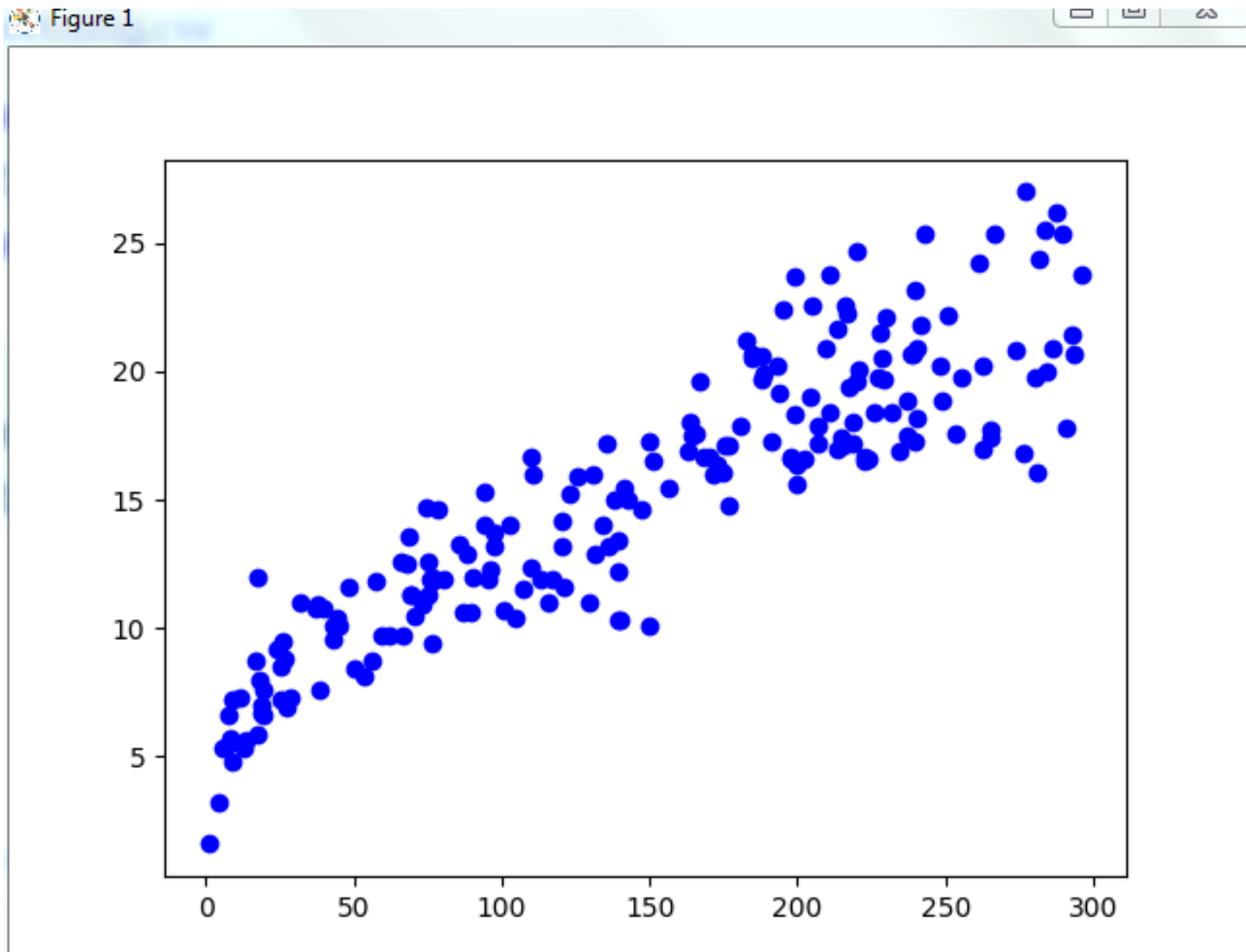
```
#print first column of all rows
print(data[:, :1])
#print all columns of first row
print(data[1, :])
#print all columns of first 20 rows
print(data[:20, :])
#print first two columns of first 20 rows
print(data[:20, :2])
```

VIEWING DATA: GRAPH

```
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl

#Loading Data
data=np.loadtxt('advertising.csv',delimiter=',',skiprows=1)
#Showing on Graph
pl.scatter(data[:,0],data[:,3],marker='o',c='b')
pl.show()
```

VIEWING DATA: GRAPH

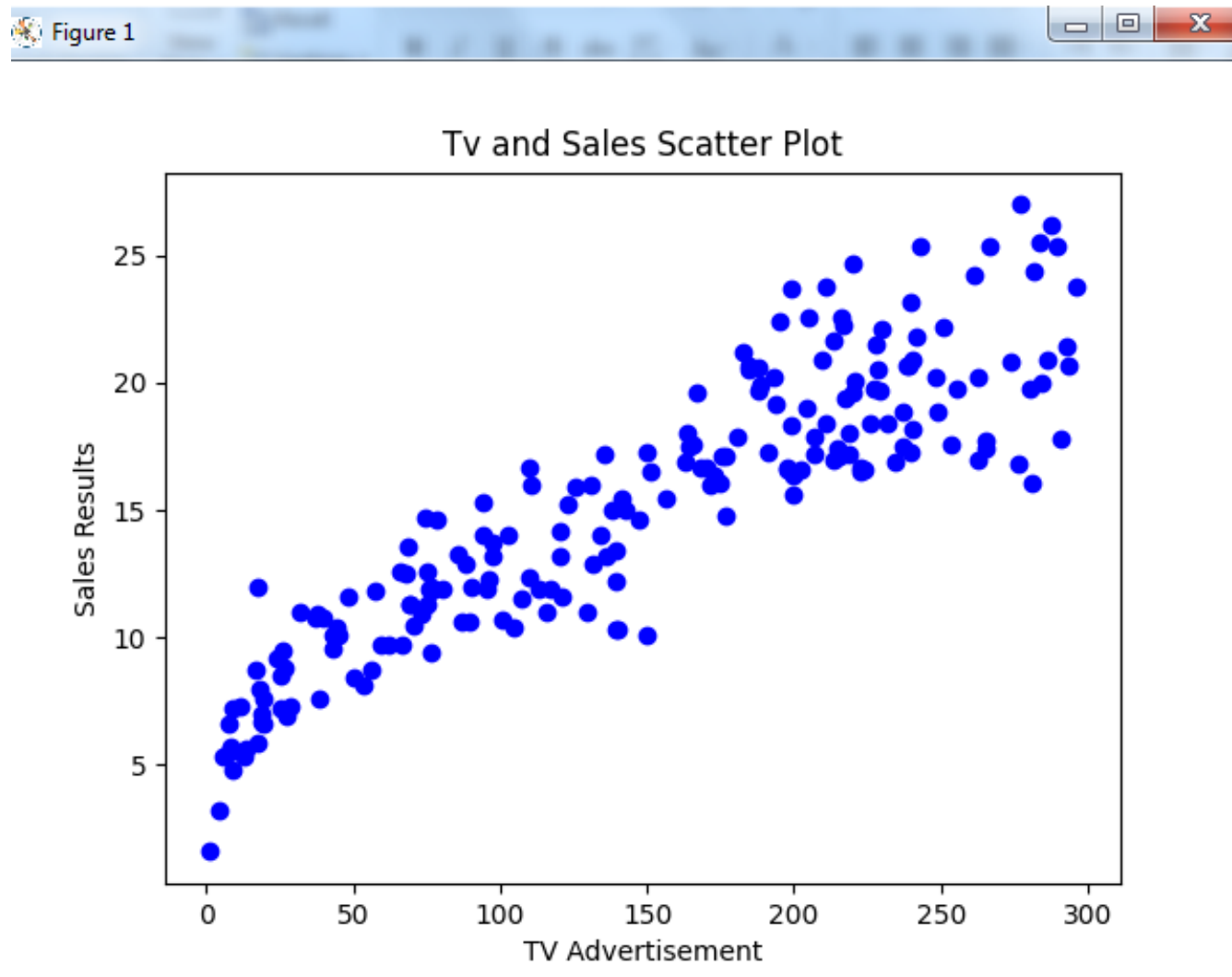


ADDING ANNOTATIONS

```
import numpy as np
import matplotlib.pyplot as plt
import pylab as pl

#Loading Data
data=np.loadtxt('advertising.csv',delimiter=',',skiprows=1)
pl.scatter(data[:,0],data[:,3],marker='o',c='b')
pl.xlabel("TV Advertisement")
pl.ylabel("Sales Results")
pl.title("Tv and Sales Scatter Plot")
pl.show()
```


ADDING ANNOTATIONS



ADDING COLUMNS OF ONES

```
import numpy as np
import pylab as pl
```

```
#Loading Data
```

```
data=np.loadtxt('advertising.csv',delimiter=',',skiprows=1)
```

```
#adding columns of ones at 0 index of data
```

```
data=np.insert(data,0,1,axis=1)
```

```
print(data)
```

SEPARATING X AND Y

```
#Separating X and y
#m are rows and n are columns
m,n=data.shape
X =data[:m,:n-1]
y= data[:m,n-1]
#Make y as column Vector
y=y.reshape(m,1)
```

ADDING COLUMN OF ONES

#adding columns of ones at 0 index of
data

```
X=np.insert(X,0,1,axis=1)
```

ADDING COLUMN OF ONES

#adding columns of ones at 0 index of
data

```
X=np.insert(X,0,1,axis=1)
```

GENERATING WEIGHTS

```
w = np.random.normal(size=(n, 1))
```

MAKING PREDICTIONS

```
#Making Predictions
```

```
y_hat = X.dot(w)
```

COMPLETE ALGORITHM

#Loading Data

```
data=np.loadtxt('ad_small.csv',delimiter=',',skiprows=1)
```

#Separating X and y

m,n=data.shape #m are rows and n are columns

```
X =data[:m,:n-1]
```

```
y= data[:m,n-1]
```

#Make y as column Vector

```
y=y.reshape(m,1)
```

#adding columns of ones at 0 index of data

```
X=np.insert(X,0,1,axis=1)
```

#initalizing weight Vector

```
w = np.random.normal(size=(n, 1))
```

#Making Predictions

```
y_hat = X.dot(w)
```


FINDING THE COST

GET COST

```
def getCost(X,y,w):  
    y_hat = X.dot(w)  
    error = y - y_hat  
    sq_error=np.square(error).sum()  
    cost = 1/2* 1/m * se  
    return cost
```

FINDING THE GRADIENT

Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update θ_0, θ_1)

}

Dr Awais Hassan UET CS

New algorithm ($n \geq 1$):

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for
 $j = 0, \dots, n$)

}

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

...

Gradient Descent

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

Gradient Descent

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

Gradient Descent

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

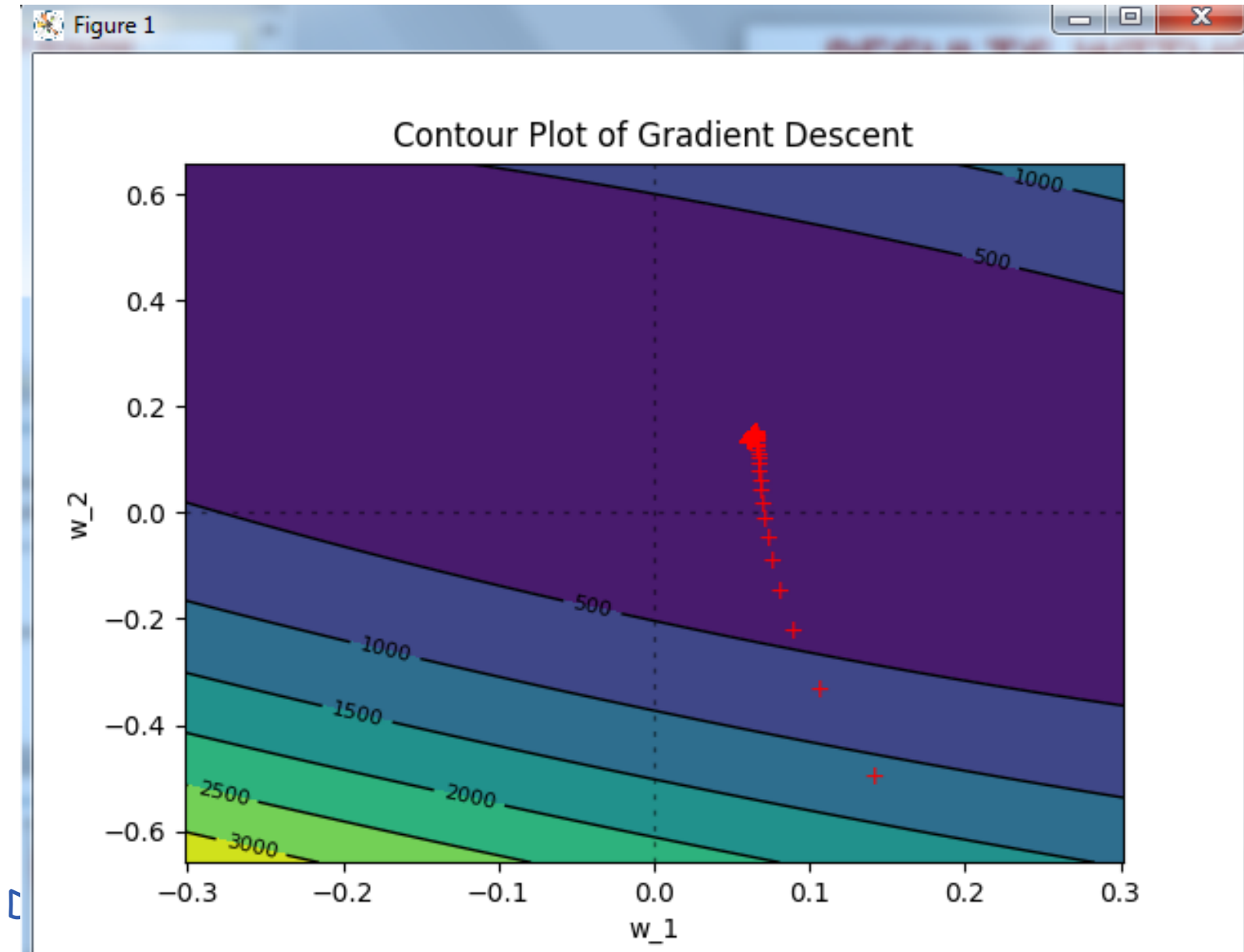
Finding Gradient Vector for given w

```
def getGradient(X,y,w):  
    y_hat = X.dot(w)  
    error = y_hat - y  
    m,n=X.shape  
    gradient =(1.0/m) * (error.T.dot(X))  
    return gradient.T
```


Gradient Descent Algorithm

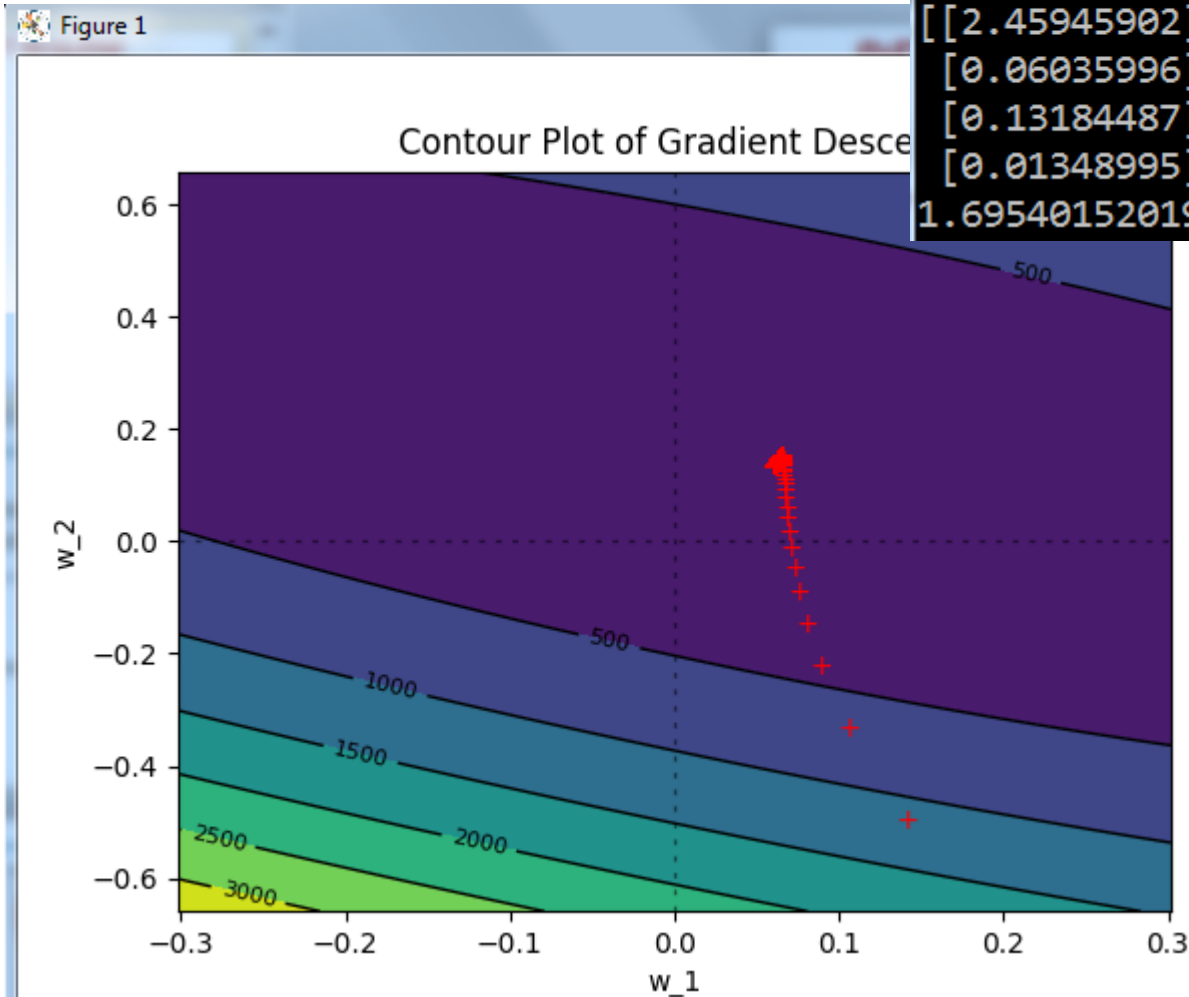
```
gv_history=[]
cost_history =[]
def gradientDescent(X,y,w,lr,threshold):
    oldCost=getCost(X, y, w)
    gv_history.append(w)
    cost_history.append(oldCost)
    newCost=0
    while (abs(oldCost-newCost) >threshold):
        gv = getGradient(X, y, w)
        #update weight code
        w= w - lr * gv
        oldCost=newCost
        newCost=getCost(X, y, w)
        gv_history.append(w)
        cost_history.append(newCost)
    return w, newCost
```

RESULTS WITHOUT SCALING LEARNING RATE IS 0.00001



RESULTS WITHOUT SCALING LEARNING RATE IS 0.00001

```
C:\Users\Awais\Google Drive\Teaching\Mach  
ariate_GradientDescent>python MLGD.py  
iterations= 416101  
[[2.45945902]  
 [0.06035996]  
 [0.13184487]  
 [0.01348995]]  
1.6954015201946353
```



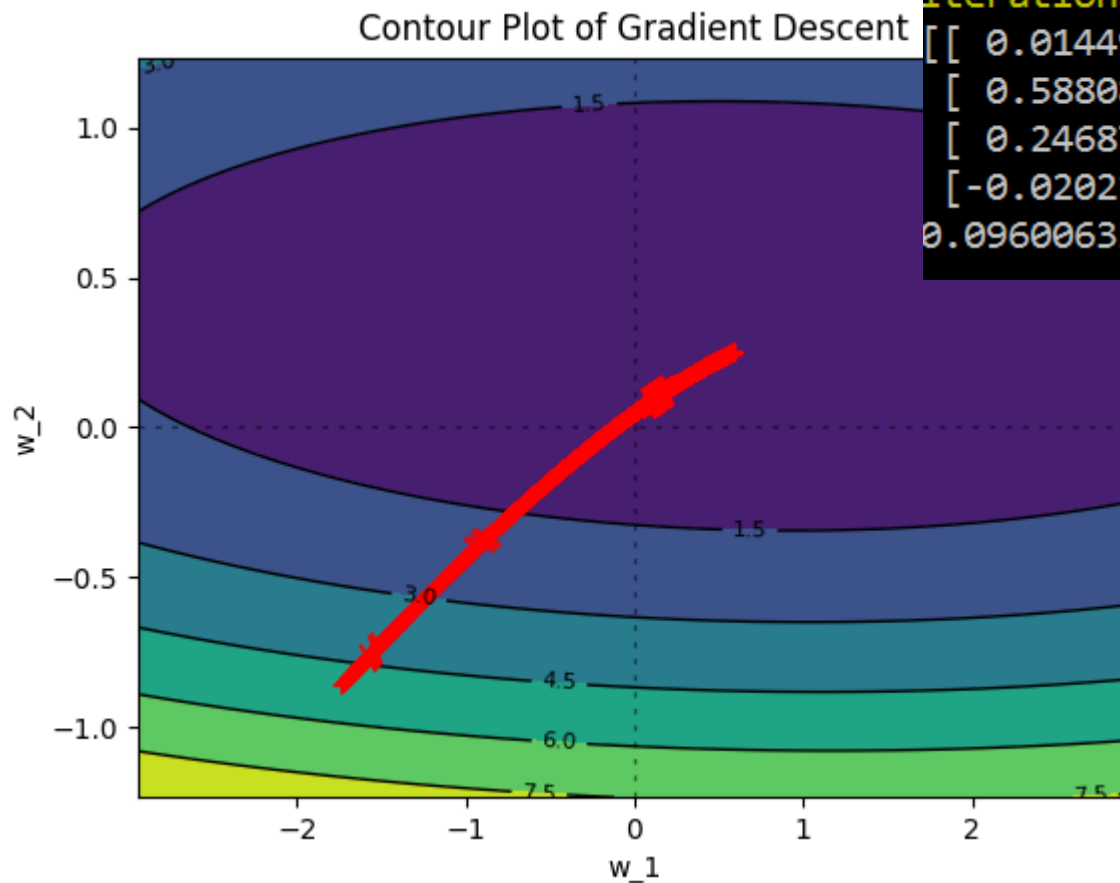
SCALING AND LEARNING RATE

- If we do not scale we need to take α very small because the gradient descent could diverge easily.

SCALING AND LEARNING RATE

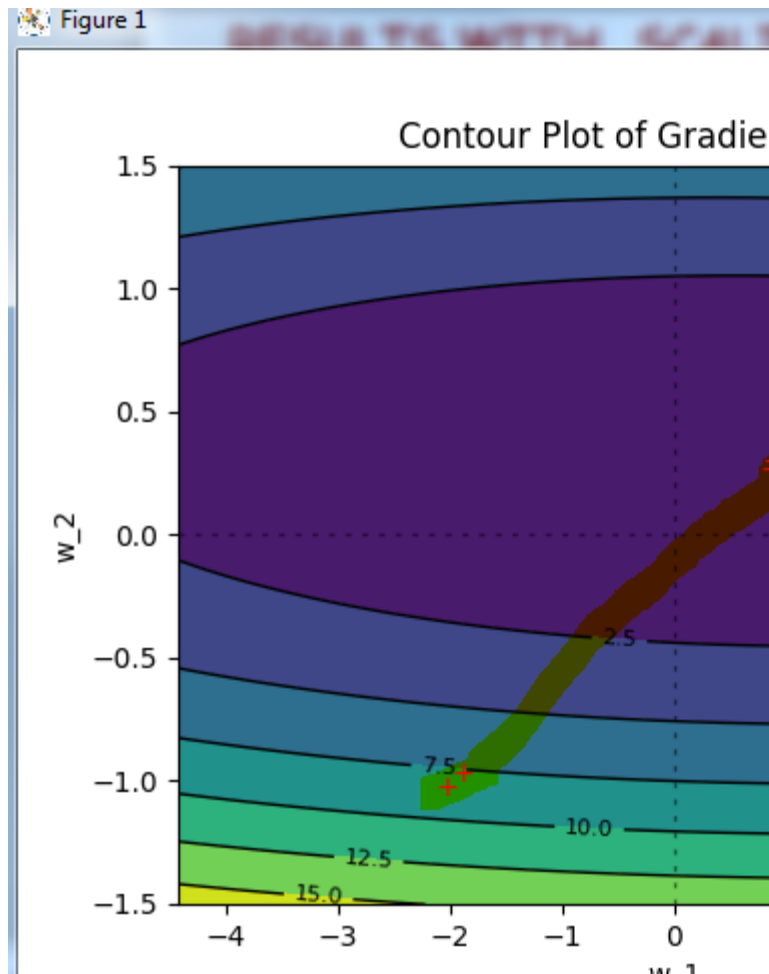
- We can scale the data, and then we have higher learning rate that help to converge faster.

RESULTS WITH SCALING LEARNING RATE IS 0.00001



```
C:\Users\Awais\Google Drive\Teaching\Ma  
ariate_GradientDescent>python MLGD.py  
iterations= 211701  
[[ 0.01449425]  
 [ 0.58808901]  
 [ 0.24687692]  
 [-0.02021414]]  
0.09600635606010982
```

RESULTS WITH SCALING LEARNING RATE IS 0.05



```
C:\Users\Awais\Google Drive\Teaching\Machine Learning\Bivariate_GradientDescent>python MLGD.py  
iterations= 301  
[[2.80481015e+00]  
 [8.84650041e-01]  
 [3.00595096e-01]  
 [1.44330372e-03]]  
0.048704357317603635
```

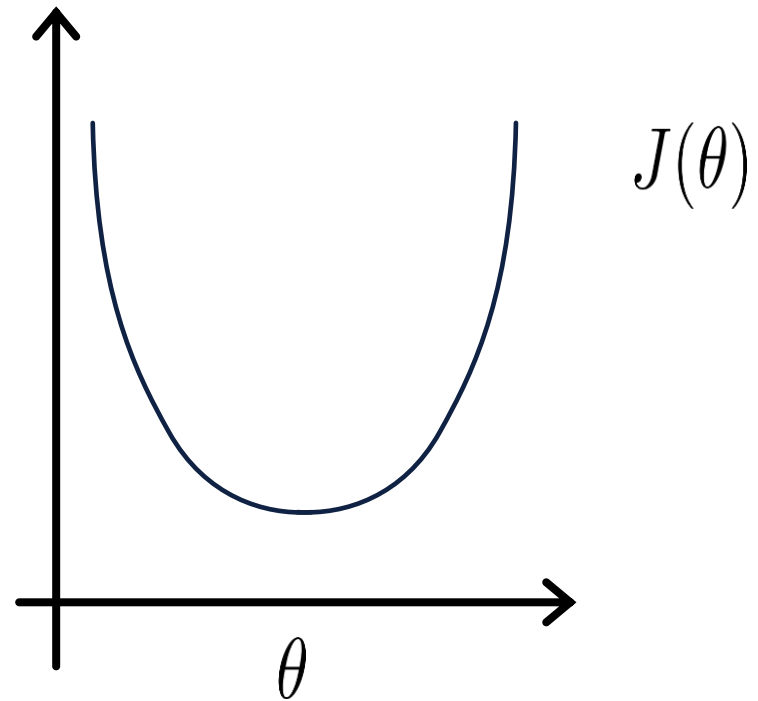


Machine Learning

Linear Regression with multiple variables

NORMAL EQUATION

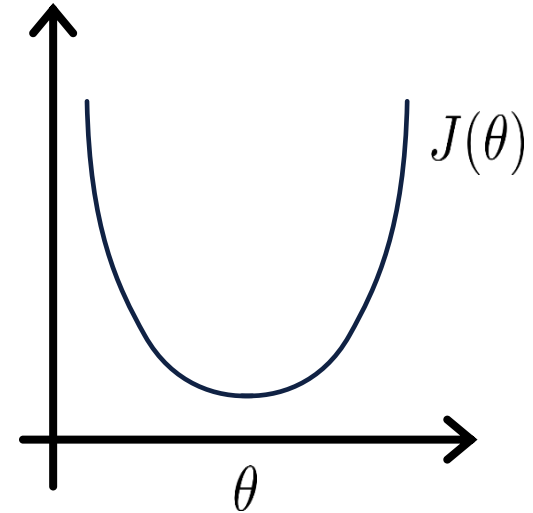
Gradient Descent



Normal equation: Method to solve for θ analytically.

Intuition: If 1D ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$



$$\theta \in \mathbb{R}^{n+1} \quad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \dots = 0 \quad (\text{for every } j)$$

Solve for $\theta_0, \theta_1, \dots, \theta_n$

Examples: $m = 5$.

x_0	x_1	x_2	x_3	x_4	y
	Size (feet²)	Number of bedroom s	Number of floors	Age of home (years)	Price (\$1000)
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178
1	3000	4	1	38	540

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ 1 & 3000 & 4 & 1 & 38 \end{bmatrix}$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ 540 \end{bmatrix}$$

$$\Theta = (X^T X)^{-1} X^T y$$

m training examples, n features.

Gradient Descent

- Need to choose α .
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α .
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large.

m training examples, n features.

Gradient Descent

- Need to choose α .
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α .
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$
- Slow if n is very large.



Machine Learning

Linear Regression with multiple variables NORMAL EQUATION AND NON- INVERTIBILITY (OPTIONAL)

Normal equation

$$\theta = (X^T X)^{-1} X^T y$$

- What if $X^T X$ is non-invertible? (singular/degenerate)
- Octave: `pinv(X' * X) * X' * y`

What if $X^T X$ is non-invertible?

- Redundant features (linearly dependent).

E.g. $x_1 = \begin{matrix} \text{size in feet}^2 \\ \text{size in m}^2 \end{matrix}$
 $x_2 =$


- Too many features (e.g. $m \leq n$).
 - Delete some features, or use regularization.

TIME COMPLEXITY

Calling n the number of observations and p the number of weights, the overall complexity should be $n^2p + p^3$.

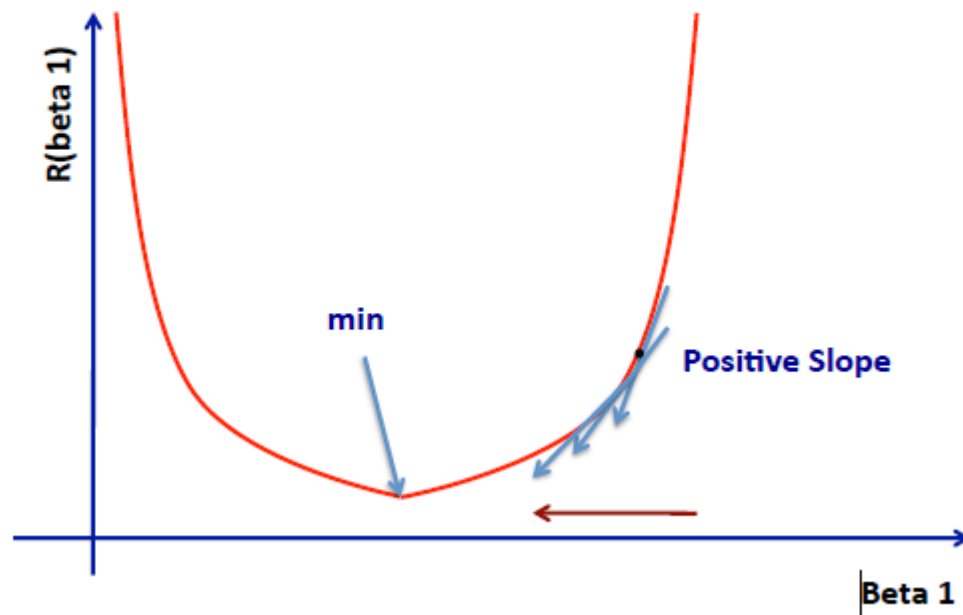
Indeed, when performing a linear regression you are doing matrices multiplication whose complexity is n^2p (when evaluating $X'X$) and inverting the resulting matrix. It is now a square matrix with p rows, the complexity for matrix inversion usually is p^3 (though it can be lowered).

Hence a theoretical complexity : $n^2p + p^3$.

- 
- Also problem with the analytical solution is that it can stuck into the local minima in case of complex risk function.

CONCLUSION

Gradient descent



CONCLUSION

Pros and Cons

Analytical approach: Normal Equation

- + No need to specify a convergence rate or iterate.
 - Works only if $X^T X$ is invertible
 - Very slow if d is large $O(d^3)$ to compute $(X^T X)^{-1}$

Iterative approach: Gradient Descent

- + Effective and efficient even in high dimensions.
 - Iterative (sometimes need many iterations to converge).
 - Needs to choose the rate α .

CONCLUSION

Practical considerations

1. **Scaling**: Bring your features to a similar scale.

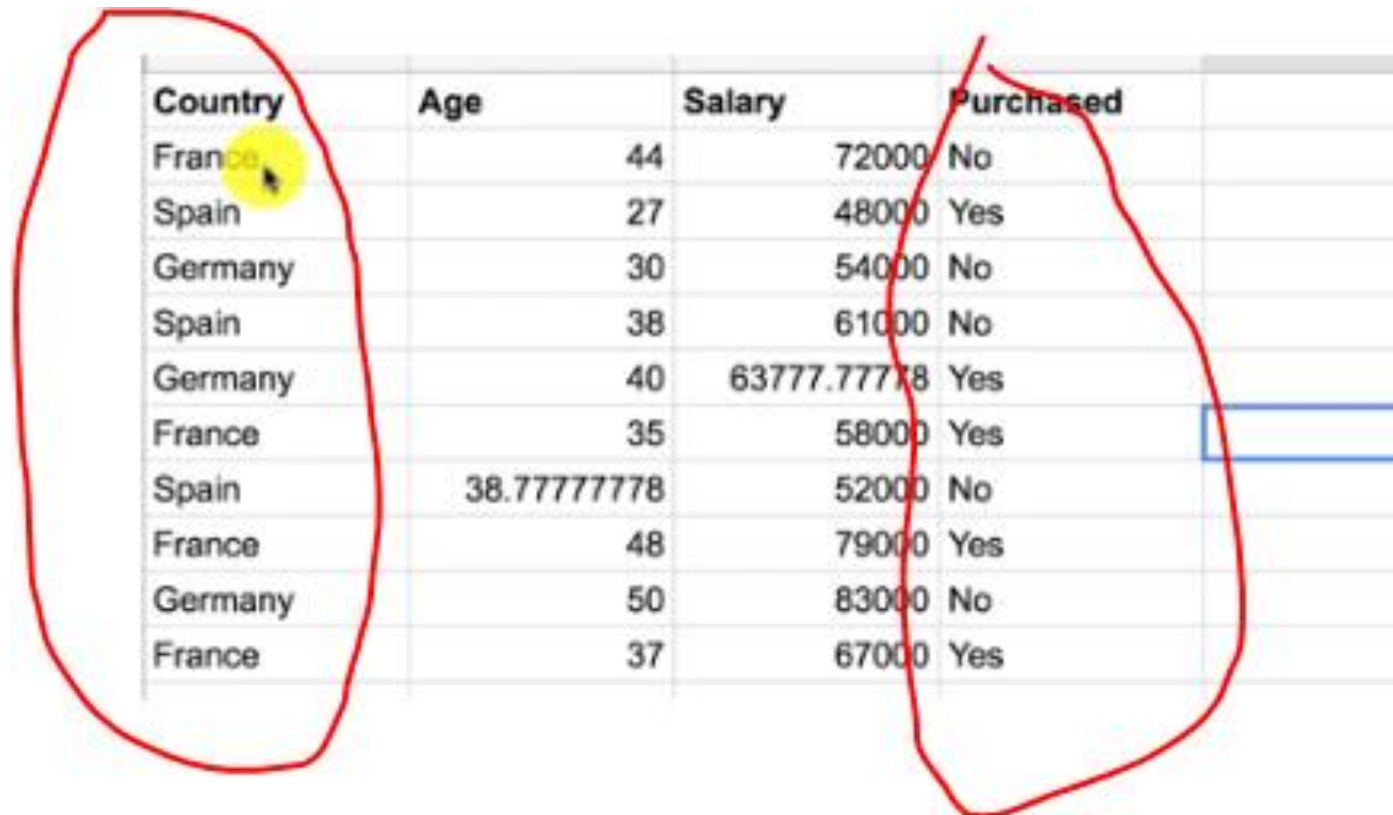
$$x_i := \frac{x_i - \mu_i}{stddev(x_i)}$$

2. **Learning rate**: Don't use a rate that is too small or too large.
3. **R should decrease** after each iteration.
4. **Declare convergence** if it start decreasing by less ϵ
5. If $X^T X$ is not **invertible**?
 - (a) Too many features as compared to the number of examples (e.g., 50 examples and 500 features)
 - (b) Features linearly dependent: e.g., weight in pounds and in kilo.

DATA PREPROCESSING

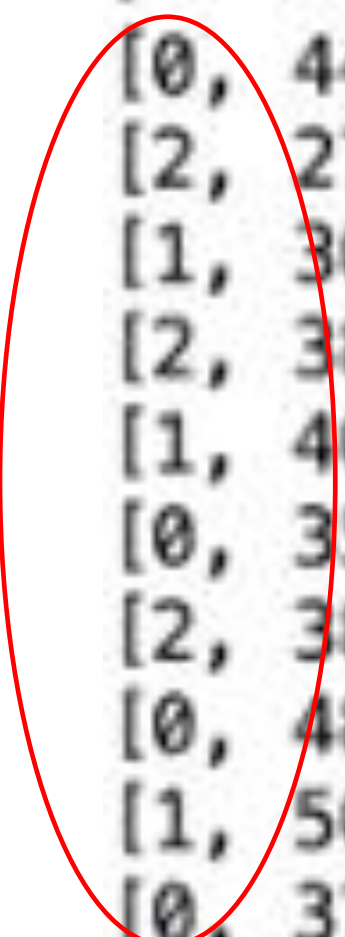
UDEMY NEURAL NETWORK A-Z

CATEGORICAL DATA (CONVERT INTO NUMBERS)



Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40	63777.77778	Yes
France	35	58000	Yes
Spain	38.77777778	52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes

ONE WAY WE GIVE NUMBERS TO COUNTRY IS



[0,	44.0,	72000.0]
[2,	27.0,	48000.0]
[1,	30.0,	54000.0]
[2,	38.0,	61000.0]
[1,	40.0,	63777.77]
[0,	35.0,	58000.0]
[2,	38.77777777777777]	
[0,	48.0,	79000.0]
[1,	50.0,	83000.0]
[0,	37.0,	67000.0]

- What's wrong with the presentation ?
- In this case, algorithm can think that the value of a country is better than others.
- What is the solution

DUMMY VARIABLES (HOT ENCODER)

Country		France	Germany	Spain
France		1	0	0
Spain		0	0	1
Germany		0	1	0
Spain		0	0	1
Germany		0	1	0
France		1	0	0
Spain		0	0	1
France		1	0	0
Germany		0	1	0
France		1	0	0



DUMMY VARIABLE TRAP


- Can we improve previous version ?
- We do not need three variables to represent the three states they can be represented by two variables.

DUMMY VARIABLES (HOT ENCODER)

Country		France	Germany	Spain
France		1	0	0
Spain		0	0	1
Germany		0	1	0
Spain		0	0	1
Germany		0	1	0
France		1	0	0
Spain		0	0	1
France		1	0	0
Germany		0	1	0
France		1	0	0

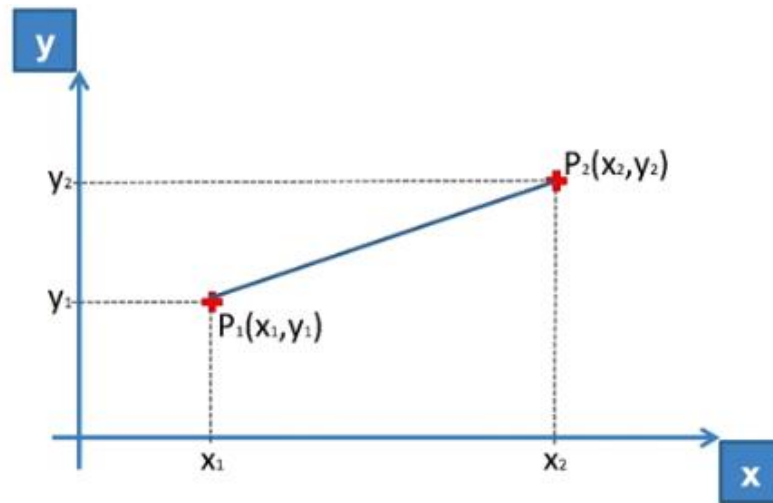
WHAT IS WRONG WITH AGE AND SALARY COLUMN

	A	B	C	D
1	Country	Age	Salary	Purchased
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	38	61000	No
6	Germany	40	63777.77778	Yes
7	France	35	58000	Yes
8	Spain	38.77777778	52000	No
9	France	48	79000	Yes
10	Germany	50	83000	No
11	France	37	67000	Yes
12				

- 
- As age and salary are not at same scale.
 - So when we take Euclidean distance that salary column shall have more effect on final decision. Infact age column shall be dominated by the other salary column

Country	Age	Salary	Purchased				
France	44	72000	No				
Spain	27	48000	Yes				
Germany	30	54000	No				
Spain	38	61000	No		31000	961000000	
Germany	40	63777.77778	Yes				
France	35	58000	Yes		21	441	
Spain	38.77777778	52000	No				
France	48	79000	Yes				
Germany	50	83000	No				
France	37	67000	Yes				

Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	38	61000	No
Germany	40	63777.77778	Yes
France	35	58000	Yes
Spain	38.77777778	52000	No
France	48	79000	Yes
Germany	50	83000	No
France	37	67000	Yes




$$\text{Euclidean Distance between } P_1 \text{ and } P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

STANDARDIZATION

- The result of standardization (Z-Score Normalization) is that the features will be rescaled so that they'll have the properties of a standard normal distribution with $\mu=0$ and $\sigma=1$
- where μ is the mean (average) and σ is the standard deviation from the mean;


NORMALIZATION

- An alternative approach to standardization is the normalization. In this approach, the data is scaled to a fixed range - usually 0 to 1.

- 
- The cost of Normalization in contrast to Standardization is that we will end up with smaller standard deviations, which can suppress the effect of outliers.

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

- **Correlation:** A measure used to represent how strongly two random variables are related.
- **Covariance** provides a measure of the strength of the **correlation** between two or more sets of random variables.

- 
- A positive **covariance** means that asset returns move together while a negative **covariance** means returns move inversely.



- The average of each input variable over the training set should be close to zero
- Scale input variable so that their covariance are about the same
- Input variables should be uncorrelated if possible.

ASSIGNMENT 08

- Assignment 08 is Replica of Assignment 07 but you need to complete the algorithm using Vector and Matrices (using Linear Algebra as we have discussed)
- For your convenience Assignment 07 is given in next slides

ASSIGNMENT 07

- Extend the GradientDescent Algorithm for Multivariate Linear Regression.
- You need to report following convergence Results for both with Normalization and without Normalization

ASSIGNMENT 07

- All Convergence Values (Command Prompt as shown in slides)
- All Theta Graphs (Separate for each theta)
- Surface Graph (use any two theta values) with the Gradient Descent Calculated values of the thetas
- Report the alpha rate, initial theta values along with your results.
- Algorithm (.py file)