

Machine learning

Presented by : Dr. Hanaa Bayomi



Lecture 3 : Linear Regression with multiple Variable

UNI-VARIANT




Size (feet ²)	Price (\$1000)
x	y
2104	460
1416	232
1534	315
852	178
...	...

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

MULTI-VARIABLE

Multiple features (variables).

Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
X_1	X_2	X_3	X_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...

   $m=47$

Notation:

n = number of features

$n=4$

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

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

$X^{(2)} =$

$\begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$

$X_3^{(2)} = 2$

MULTI-VARIABLE

Multiple features (variables).

Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
X_1	X_2	X_3	X_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...

$$X_1^{(4)} = 852$$

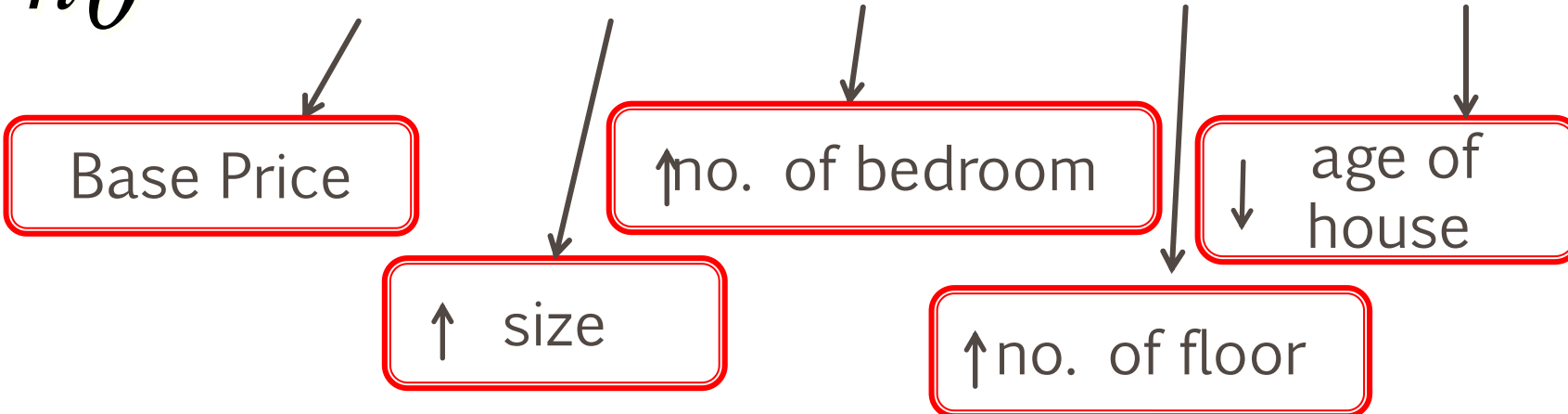
Hypothesis:

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

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

The weights used by the model indicate the effect of each descriptive feature on the predictions returned by the model

$$h_{\theta} = 80 + 0.1x_1 + 3x_2 + 0.01x_3 - 2x_4$$



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

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

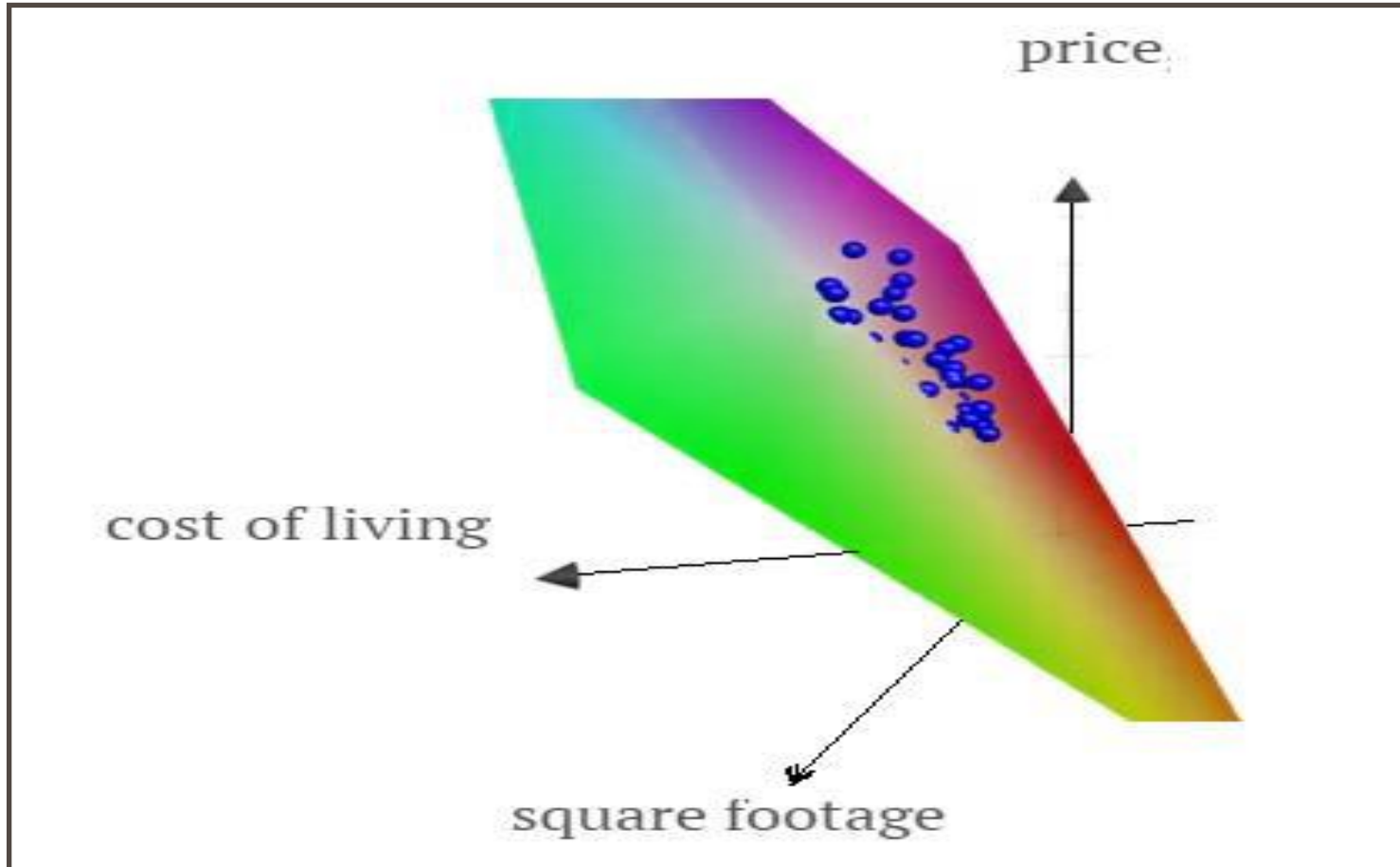
$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \cdot \\ \cdot \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$h_{\theta}(x) = \theta^T X$$

$$[\theta_0 \theta_1 \theta_2 \dots \theta_n] \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

In this case we can again fit a predictor to the data. But instead of drawing a line through the data we have to draw a plane through the data because the function that best predicts the housing price is a function of two variables.



GRADIENT DESCENT FOR MULTIPLE VARIABLE

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 {

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

$$\underbrace{\hspace{10em}}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

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

(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
 $j = 0, \dots, n$)

}

$$x_0 = 1$$

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

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

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

PRACTICAL TIPS FOR GETTING GRADIENT DESCENT TO WORK

1- FEATURE SCALING

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet²)

x_2 = number of bedrooms (1-5)

$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$

$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 \leq 1$$

FEATURE SCALING

More generally, when we're performing feature scaling, what we often want to do is *get every feature into approximately a -1 to +1 range* and concretely, your feature x_0 is always equal to 1. So, that's already in that range,

$$-1 \leq x_i \leq 1$$

$$0 \leq x_1 \leq 3 \quad \checkmark \quad -100 \leq x_3 \leq 100 \quad \times$$

$$-2 \leq x_2 \leq 0.5 \quad \checkmark \quad -0.000001 \leq x_4 \leq 0.000001 \quad \times$$

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean
(Do not apply to $x_0 = 1$).

E.g. $x_1 = \frac{\text{size}-1000}{2000}$

$x_2 = \frac{\text{\#bedrooms}-2}{5}$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

$$x_1 = \frac{x_1 - \mu_1}{R_1}$$

Average of x_1 in the
training set

Range

2- LEARNING RATE

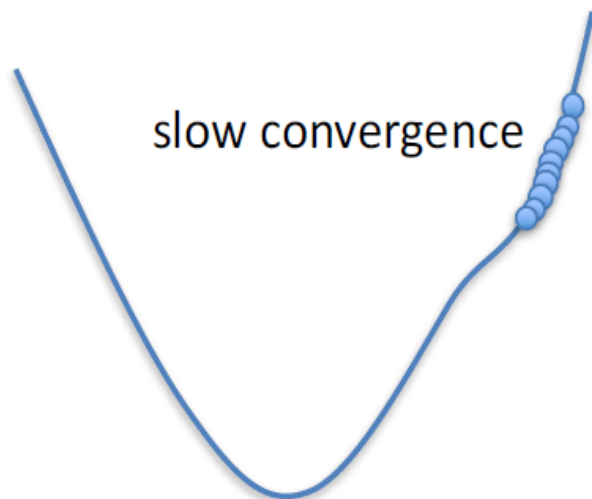
Gradient descent

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

- “Debugging”: How to make sure gradient descent is working correctly.
- How to choose learning rate α .

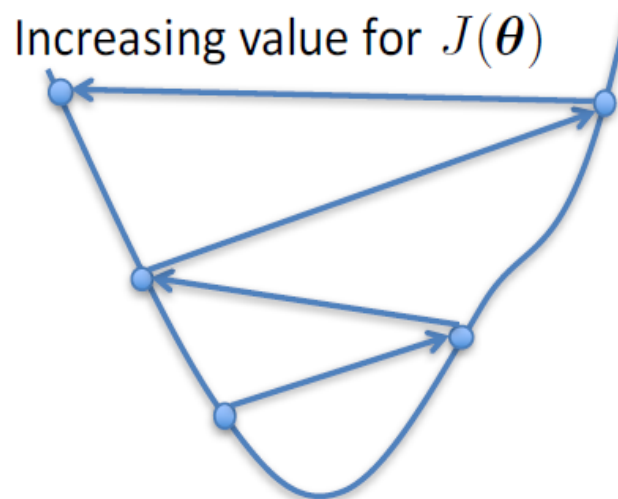
Choosing α

α too small



slow convergence

α too large



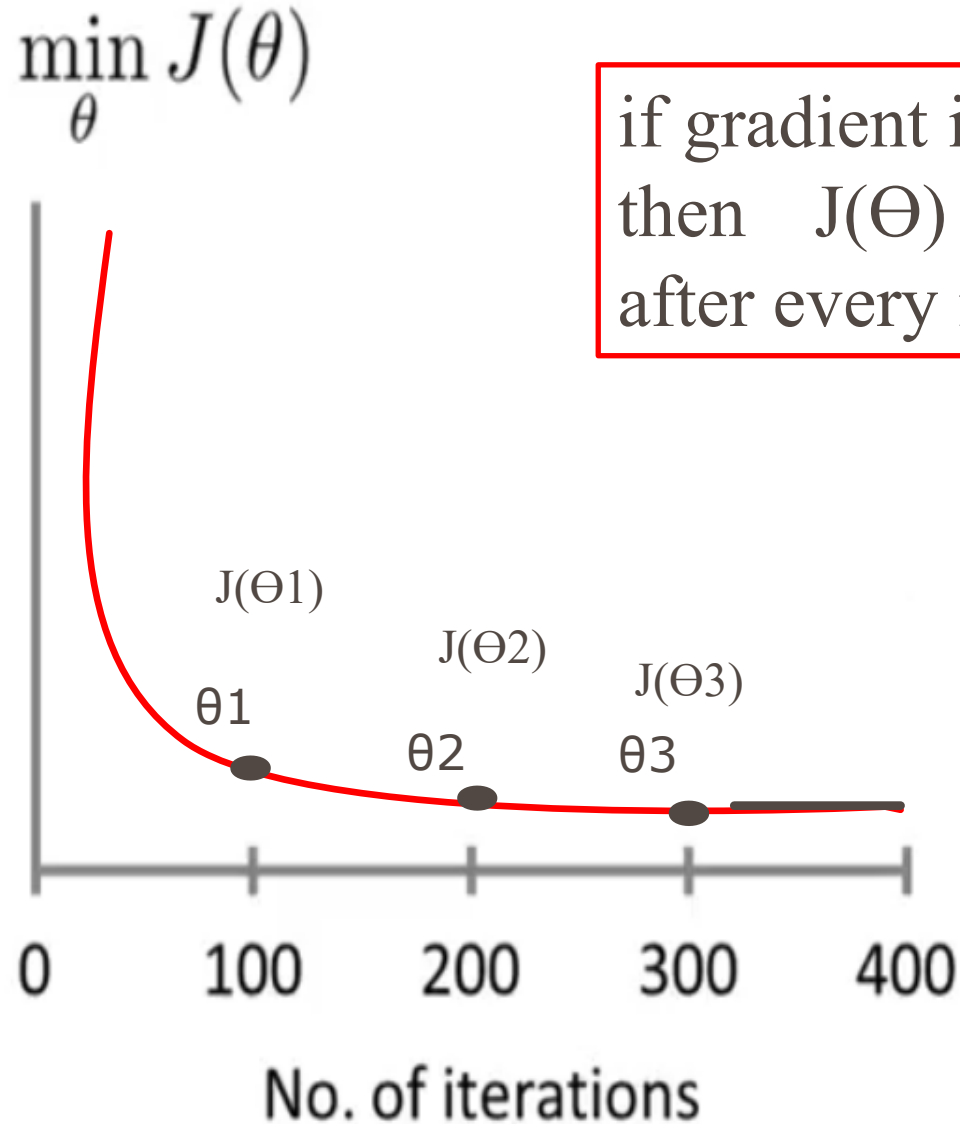
Increasing value for $J(\theta)$

- May overshoot the minimum
- May fail to converge
- May even diverge

To see if gradient descent is working, print out $J(\theta)$ each iteration

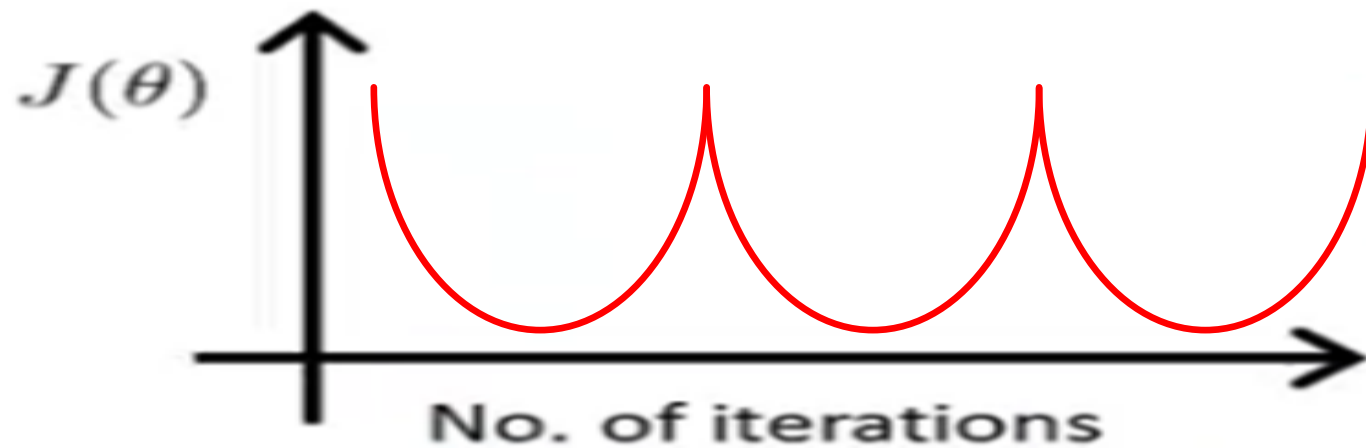
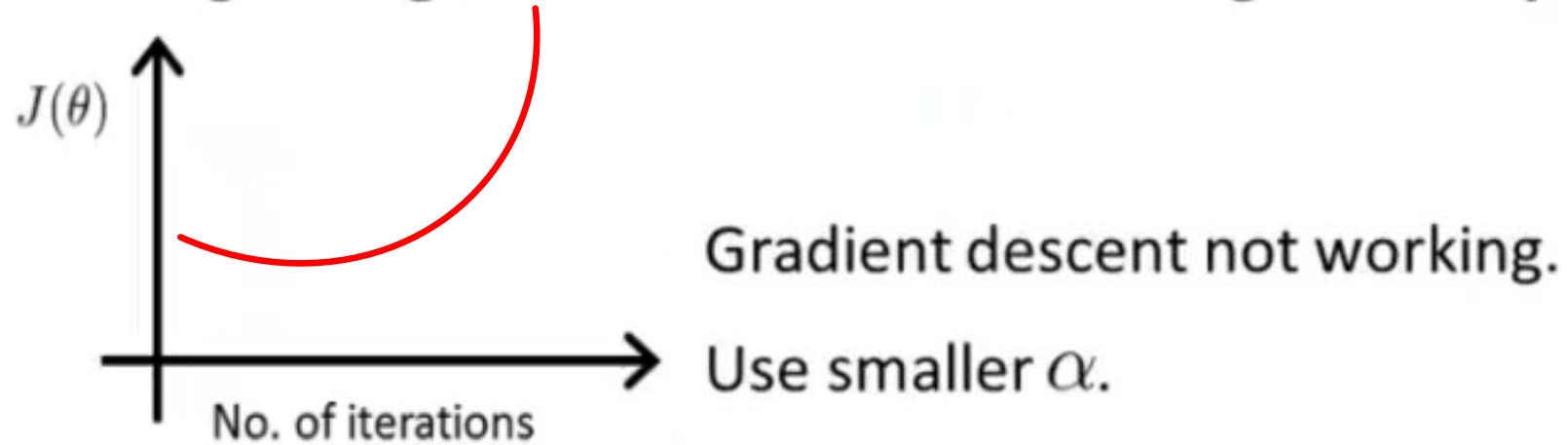
- The value should decrease at each iteration
- If it doesn't, adjust α

Making sure gradient descent is working correctly.



if gradient is working properly then $J(\Theta)$ should decrease after every iteration.

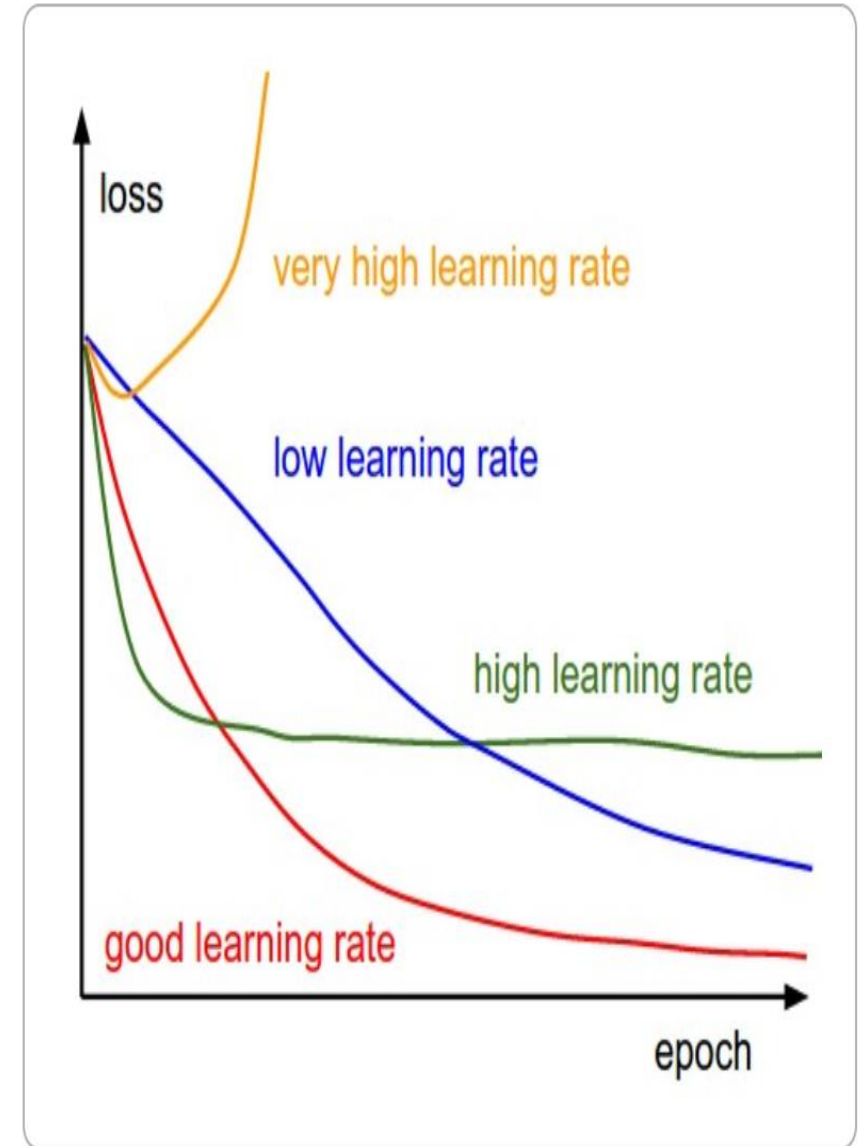
Making sure gradient descent is working correctly.



- For sufficiently small α , $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

The learning rate α vs the cost function $J(\theta_1)$

- The **yellow** plot shows the **divergence** of the algorithm when the learning rate is really high where in the learning steps overshoot.
- The **green** plot shows the case where learning rate is not as large as the previous case but is high enough that the steps keep **oscillating** at a point which is not the minima.
- The **red** plot would be the **optimum curve** for the cost drop as it drops steeply initially and then saturates very close to the optimum value.
- The **blue** plot is the least value of α and **converges very slowly** as the steps taken by the algorithm during update steps are very small.



Effect of alpha on convergence

Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

In order to choose optimum value of α run the algorithm with different values like:

To choose α , try

$\dots, 0.001, \mathbf{0.003}, 0.01, \mathbf{0.03}, 0.1, \mathbf{0.3}, 1, \dots$

plot the learning curve to understand whether the value should be increased or decreased.

POLYNOMIAL REGRESSION

POLYNOMIAL REGRESSION

- Our hypothesis function need not be linear (a straight line) if that does not fit the data well.
- We can **change the behavior or curve** of our hypothesis function by making it a *quadratic* , *cubic* or *square root* function or any other form.

■ For example, if our hypothesis function is $h_{\theta} = \theta_0 x_0 + \theta_1 x_1$

■ then we can ***create additional features based on x_1*** ,to get

the quadratic function $h_{\theta} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2$

or the cubic function $h_{\theta} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

■ In the cubic version, we have created new features x_2 , x_3

$$x_2 = x_1^2$$

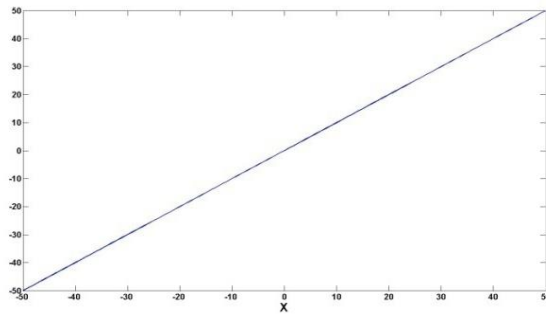
$$x_3 = x_1^3$$

SHAPE AND COEFFICIENT SIGN

The sign of the coefficient for the highest order regressor determines the direction of the curvature

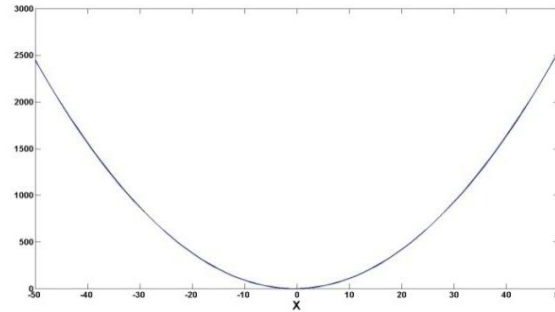
Linear

$$Y' = 0 + 1X$$



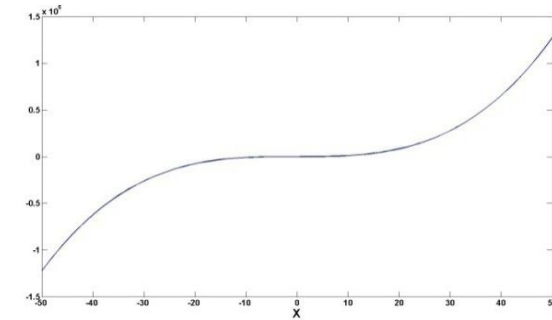
Quadratic

$$Y' = 0 + 1X + 1X^2$$

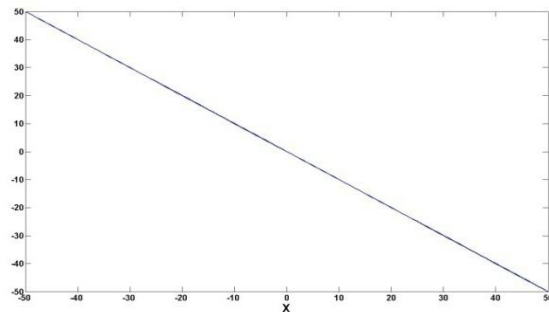


Cubic

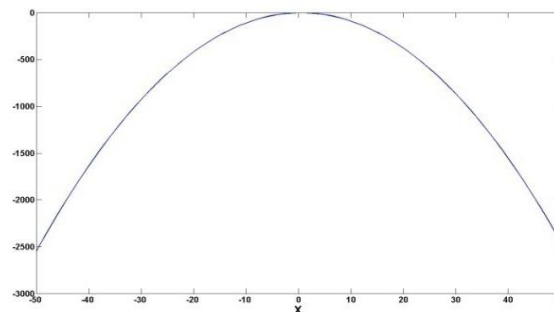
$$Y' = 0 + 1X + 1X^2 + 1X^3$$



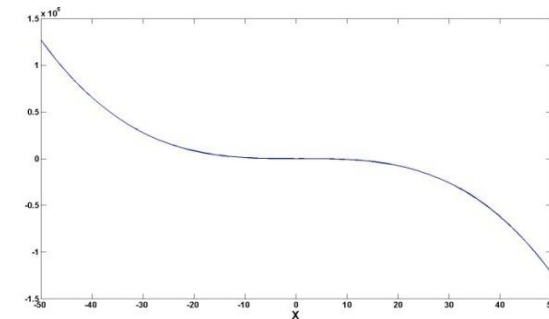
$$Y' = 0 + -1X$$



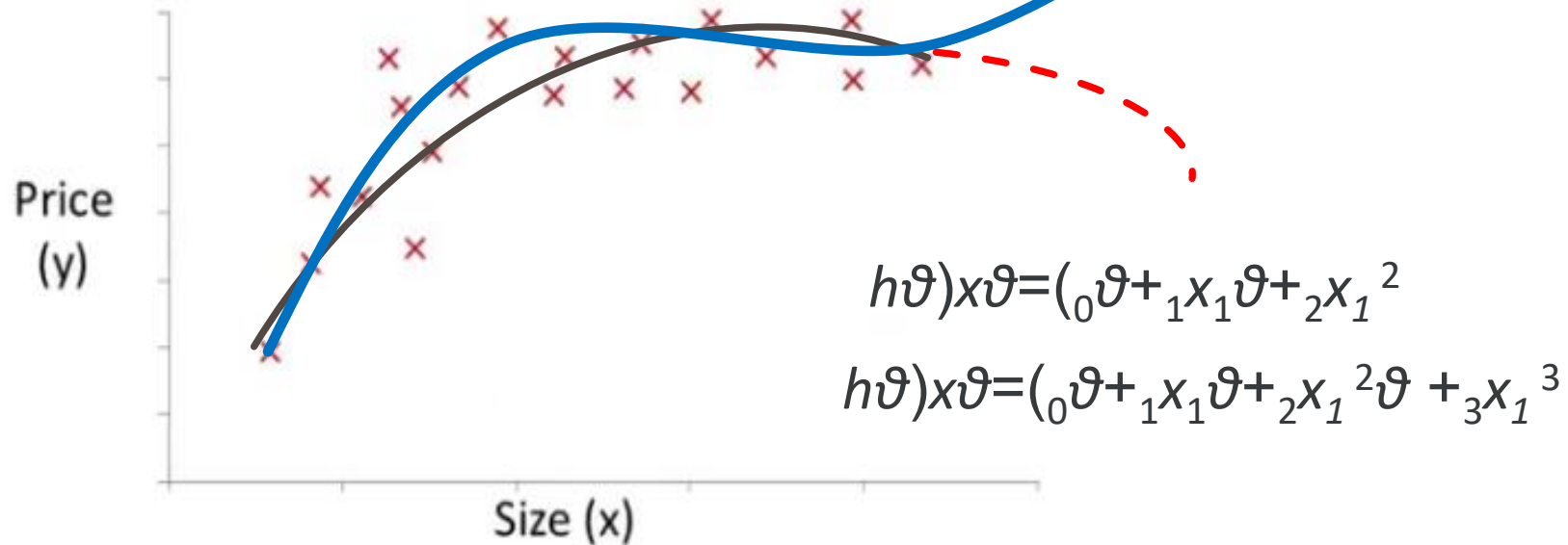
$$Y' = 0 + 1X + -1X^2$$



$$Y' = 0 + 1X + 1X^2 + -1X^3$$



Polynomial regression



$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

$$x_1 = (size)$$

$$x_2 = (size)^2$$

$$x_3 = (size)^3$$

Range



Size	1-1000
size ²	1-1000 000
size ³	1- 1000 000 000

OVER FITTING AND UNDER FITTING

$$h_{\theta} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3 + \dots + \theta_n x_1^n$$

As the *polynomial order N increases*, the functions $h_{\theta}(x)$ are able to capture increasingly complex behavior.

Generalization in Machine Learning

- The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen. “**learning general concepts from specific examples**”
- There is a terminology used in machine learning when we talk about how well a machine learning model learns and generalizes to new data, namely *overfitting and underfitting*.
- Overfitting and underfitting are the *two biggest causes for poor performance of machine learning algorithms*.

OVER FITTING AND UNDER FITTING

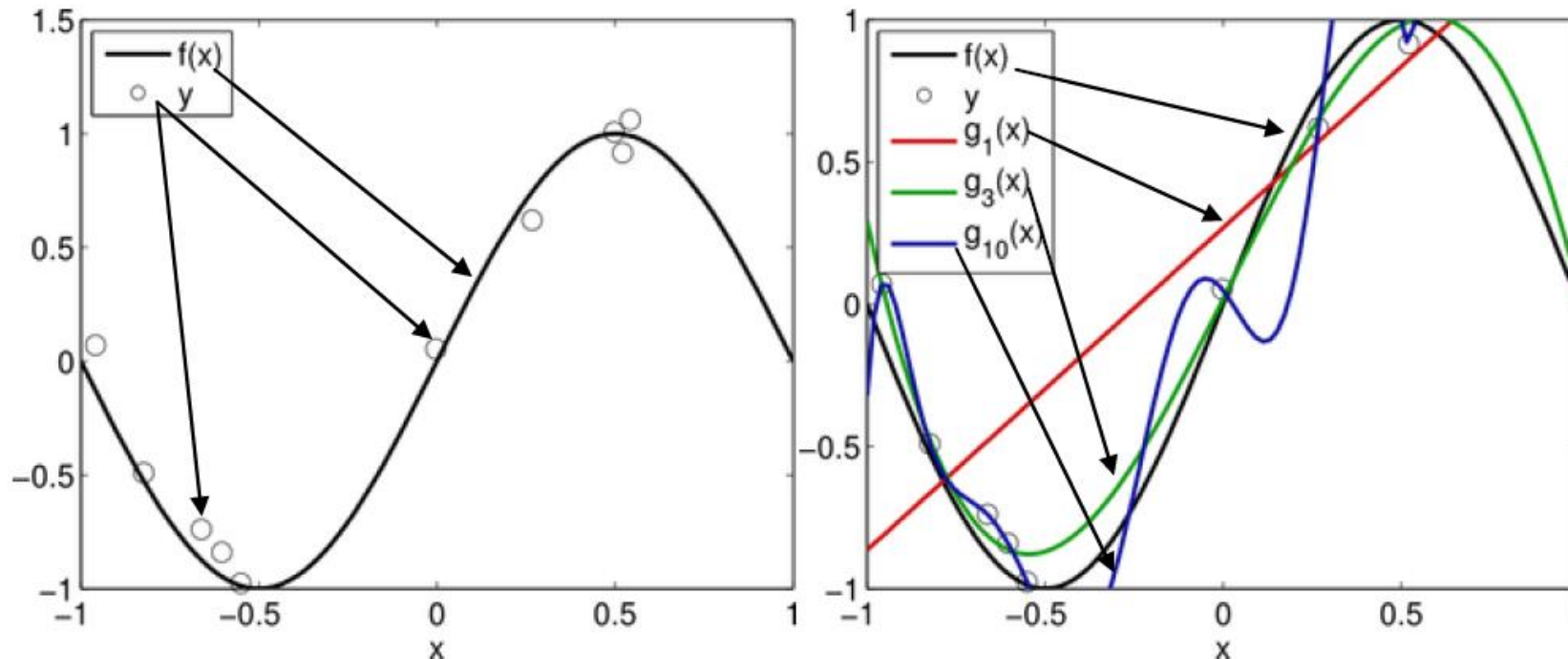
Over-fitting

- Overfitting refers to a model that models the training data too well.
- Overfitting happens when a model learns *the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data*. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.
- *decision trees* are a machine learning algorithm that is very flexible and is subject to overfitting training data.

OVER FITTING AND UNDER FITTING

Under-fitting

- Underfitting refers to a model that can neither model the training data nor generalize to new data.
- An underfitting machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.



OVER FITTING AND UNDER FITTING

A Good Fit in Machine Learning

- Ideally, you want to select a model at the *sweet spot* between underfitting and overfitting.
- The sweet spot is *the point just before the error on the test dataset starts to increase* where **the model has good skill on both the training dataset and the unseen test dataset.**
- Both overfitting and underfitting can lead to poor model performance. But by far the most common problem in applied machine learning is **overfitting**.
- in order to limit overfitting by using a resampling technique (k-fold cross validation) to estimate model accuracy.
- **k-fold cross validation** allows you to train and test your model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.