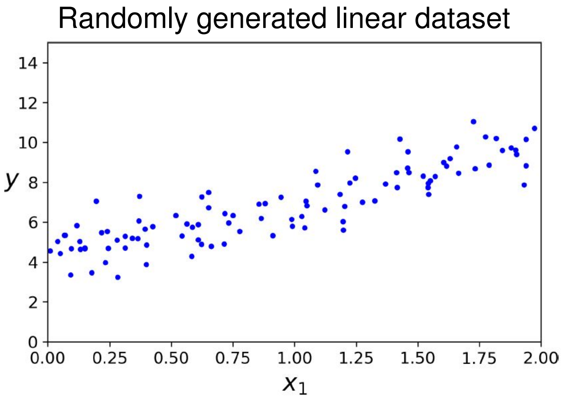
Gradient Descent in Regression and Classification
CS-477 Computer Vision

Dr. Mohsin Kamal
Associate Professor
dr.mohsinkamal@seecs.edu.pk

School of Electrical Engineering and Computer Science (SEECS)
National University of Sciences and Technology (NUST), Pakistan

Model representation



Supervised Learning
Given the "right answer" for each example in the data.

Regression Problem
Predict real-valued output

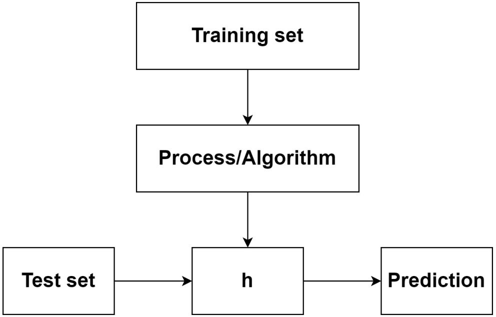
Training set of housing prices

Area in $feet^2$ (x)	**Price (y)**
100	100,000
150	140,000
200	250,000
220	290,000
...	...

Notation:

- m = Number of training examples
- x's = "input" variable / features
- y's = "output" variable / "target" variable

Model representation

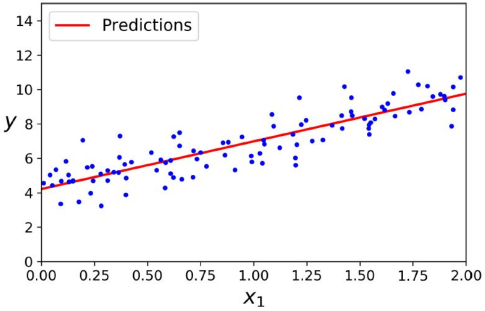


How do we represent h?

Model representation

What to do?

- Training a model means setting its parameters so that the model best fits the training set.
- To train a Linear Regression model, you need to find the value of θ that minimizes the Mean Squared Error.



Cost function

Cost function: It is a function that measures the performance of a Machine Learning model for given data.

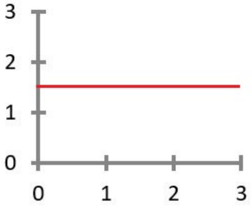
Table 1 : Training set

Area in *feet*^{2} **(x)**	**Price (y)**
100	100,000
150	140,000
200	250,000
220	290,000
...	...

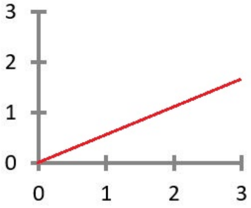
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$
θ's: Parameters
How to choose θ's?

Cost function

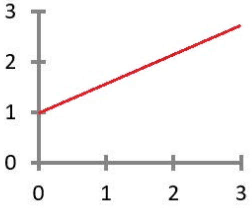
$$h_\theta(x) = \theta_0 + \theta_1 x$$



$\theta_0 = 1.5$
$\theta_1 = 0$

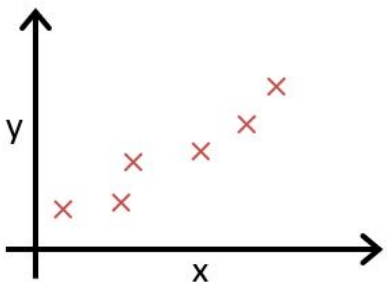


$\theta_0 = 0$
$\theta_1 = 0.5$



$\theta_0 = 1$
$\theta_1 = 0.5$

Cost function



Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

Cost function

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

Simplified

$h_\theta(x) = \theta_1 x$ when $\theta_0 = 0$

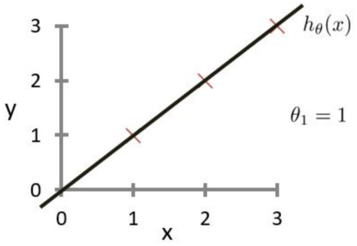
θ_1

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

$\min_{\theta_1} J(\theta_1)$

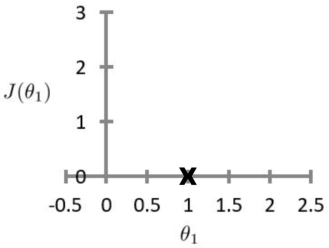
$h_\theta(x)$

(for fixed θ_1, this is a function of x)

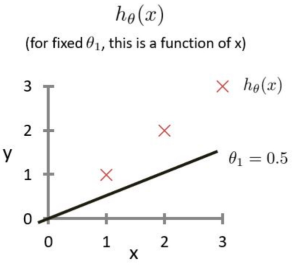


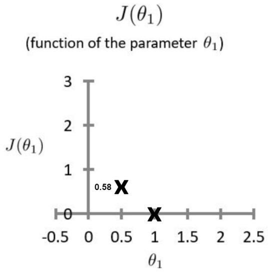
$J(\theta_1)$

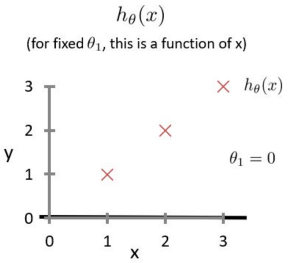
(function of the parameter θ_1)

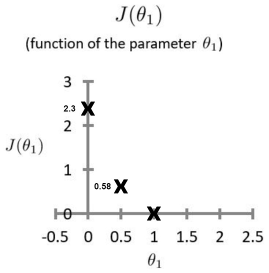


Cost function

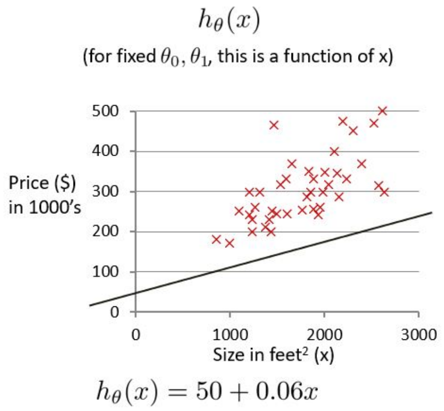


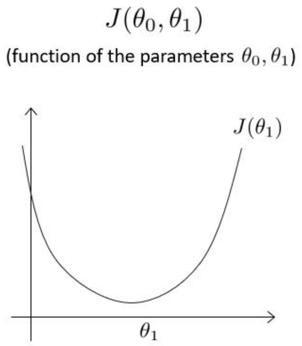






Cost function





Fortunately, the MSE cost function for a Linear Regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve. This implies that there are no local minima, just one global minimum.

Gradient Descent

Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

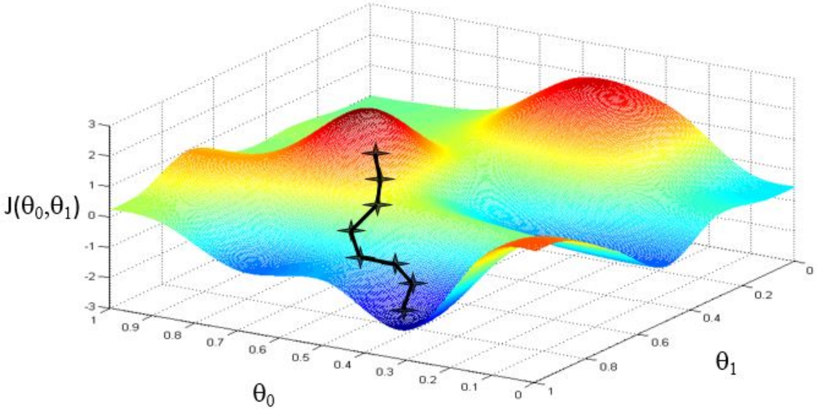
- Have some function $J(\theta_0, \theta_1)$
- Goal: $\min\limits_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

Outline:

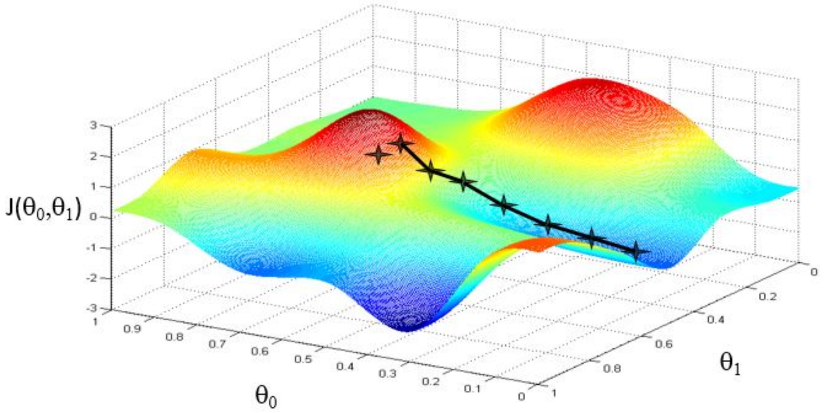
- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at the minimum

Summary: You start by filling θ with random values (this is called random initialization), and then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum

Gradient Descent



Gradient Descent



Gradient descent Algorithms

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Batch gradient descent

repeat until convergence:

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

Correct: Simultaneous update

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

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

$\theta_0 :=$ temp0

$\theta_1 :=$ temp1

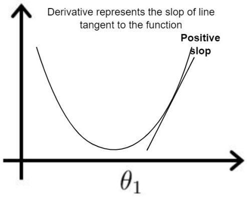
Incorrect

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

$\theta_0 :=$ temp0

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

$\theta_1 :=$ temp1


Derivative represents the slop of line tangent to the function
Positive slop
θ_1

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

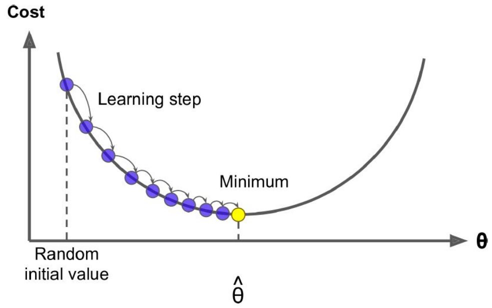
- If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

Batch gradient descent

- Gradient descent can converge to a local minimum, even with the learning rate α fixed.

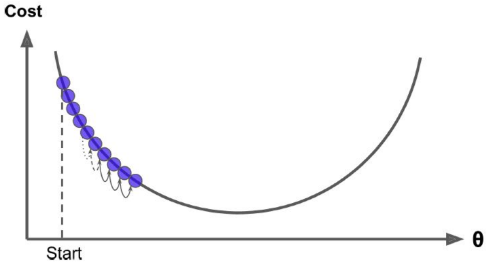
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

- As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



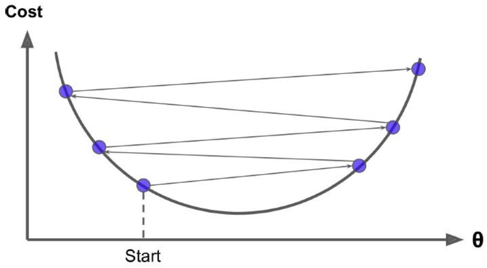
Batch gradient descent

If the **learning rate is too small**, then the algorithm will have to go through many iterations to converge, which will take a long time.



Batch gradient descent

- If the **learning rate is too high**, you might jump across the valley and end up on the other side, possibly even higher up than you were before.
- This might make the algorithm diverge, with larger and larger values, failing to find a good solution.



Batch gradient descent

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

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

$$j = 0 : \frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$$

$$j = 1 : \frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}).x^{(i)}$$

Gradient descent algorithm becomes:

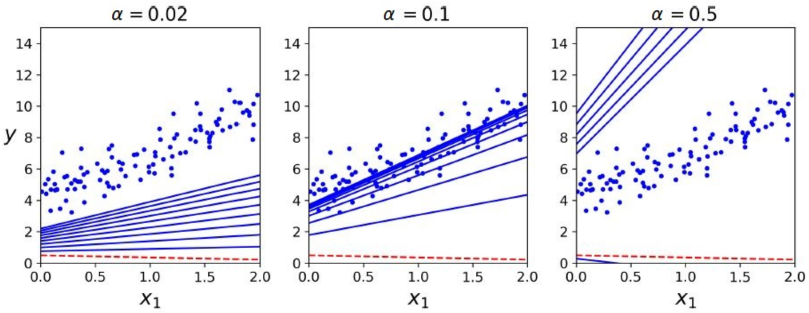
Repeat until convergence {

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

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

}

Batch gradient descent



Stochastic gradient descent

- The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.
- Stochastic Gradient Descent just picks a random instance in the training set at every step and computes the gradients based only on that single instance.
- This makes the algorithm much faster since it has very little data to manipulate at every iteration.

Stochastic gradient descent

- On the other hand, due to its stochastic (i.e., random) nature, this algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.
- Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down. So once the algorithm stops, the final parameter values are good, but not optimal.
- When the cost function is very irregular, this can actually help the algorithm jump out of local minima, so Stochastic Gradient Descent has a better chance of finding the global minimum than Batch Gradient Descent does.

Stochastic gradient descent

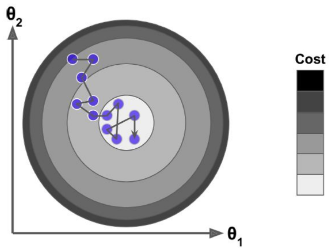


Figure 1 : Stochastic Gradient Descent

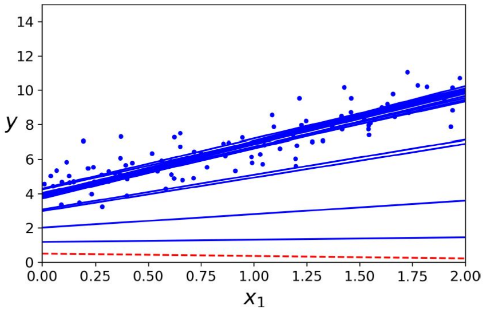


Figure 2 : Stochastic Gradient Descent first 20 steps

Mini-batch gradient descent

- It is quite simple to understand once you know Batch and Stochastic Gradient Descent: at each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Minibatch GD computes the gradients on small random sets of instances called *minibatches*.
- The algorithm's progress in parameter space is less irregular than with SGD, especially with fairly large mini-batches.
- As a result, Mini-batch GD will end up walking around a bit closer to the minimum than SGD.

Mini-batch gradient descent

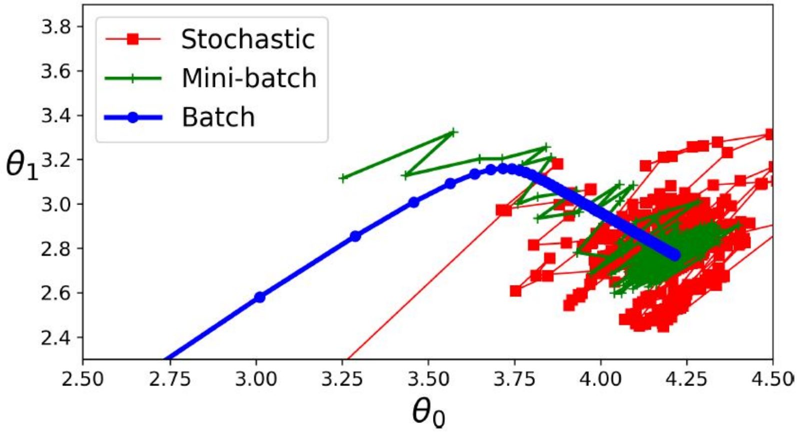


Figure 3 : Gradient Descent paths in parameter space

GD for multiple variables

Size ($feet^2$) x_1	No. of Bedrooms x_2	No. of floors x_3	Age of home x_4	Price ($k) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

Notation:

- m = Number of training examples i.e., 47
- 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
- y's = "output" variable / "target" variable

GD for multiple variables

Hypothesis:

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

How to represent it mathematically?

For convenience of notation, define $x_0 = 1$

let

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \Re^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \Re^{n+1}$$

We can write it in multiplication form as:

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \ldots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$h_\theta(x) = \theta^T x$$

GD for multiple variables

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$
Parameters: θ
Cost function: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$
Gradient descent:
Repeat {
$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$
}
(Simultaneously update for every $j = 0, 1, 2. \dots, n$)

GD for multiple variables

Gradient Descent

Previously (n=1):

Repeat {

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

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

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

}

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

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

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

$\ldots$

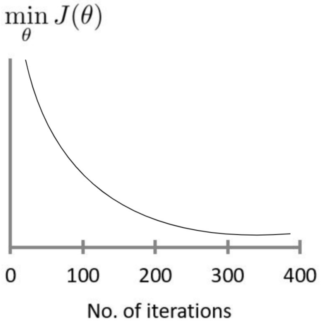
Learning rate

Gradient descent
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
We will learn:

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

Learning rate

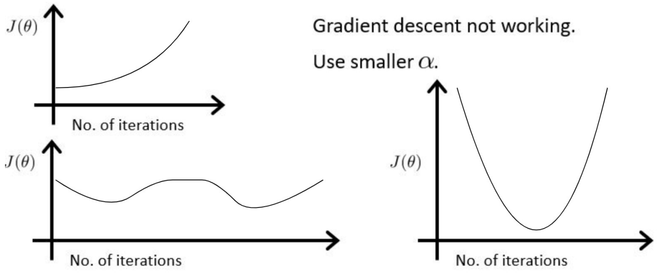
Making sure gradient descent is working correctly.



Example automatic convergence test:
Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

Learning rate

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.

Learning rate

Summary:

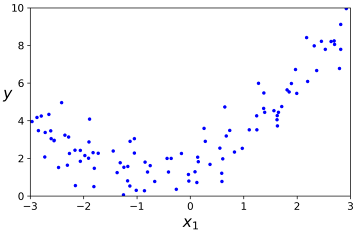
- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

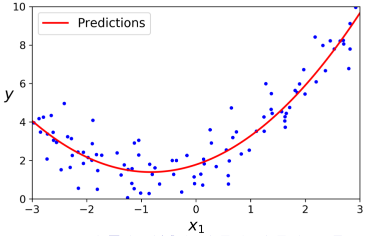
To choose α, try

$$\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$$

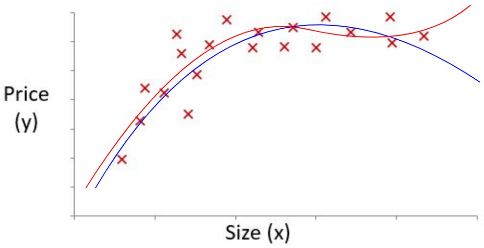
Polynomial regression

- What if your data is actually more complex than a simple straight line?
- Surprisingly, you can actually use a linear model to fit nonlinear data.
- A simple way to do this is to add powers of each feature as new features, then train a linear model on this extended set of features.
- This technique is called **Polynomial Regression.**





Polynomial regression



$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

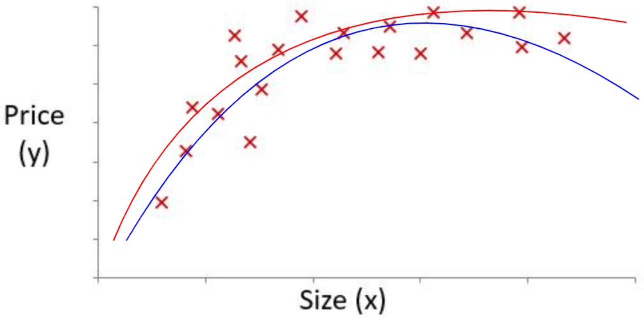
We will require feature scaling because:

$size$: 1 - 1000

$size^2$: 1 - 1000000

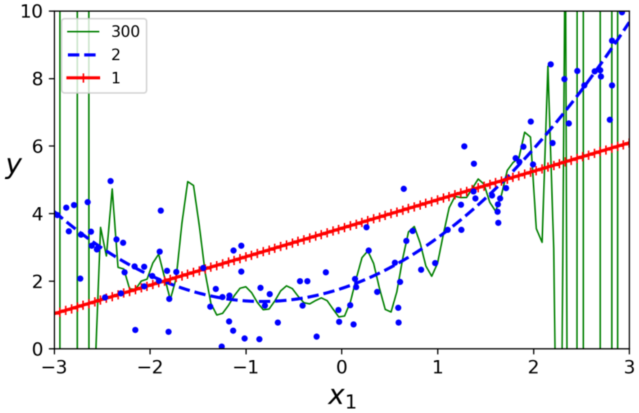
$size^3$: 1 - 10^9

Choice of features



$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$
$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

High-degree Polynomial Regression



Normal Equation

Normal equation: Method to solve for θ analytically.
When to use it?

Intuition: If 1D ($\theta \in \Re$) $J(\theta) = a\theta^2 + b\theta + c$

how to minimize quadratic function?,

$$\frac{d}{d\theta} J(\theta) = \cdots = 0$$

Solve for θ

When $\theta \in \Re^{n+1}$

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

do,

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

Solve for $\theta_0, \theta_1, \cdots, \theta_m$

Normal Equation

Training examples: $m = 4$.

	Size ($feet^2$)	Bedrooms	Floors	Age (years)	Price (k)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

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

Normal Equation

m training examples, *n* features.

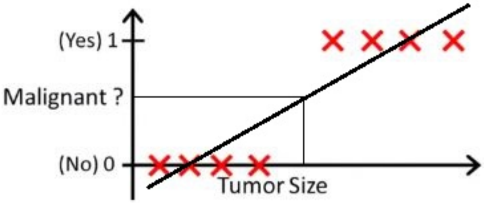
Gradient Decent	**Normal Equation**
Need to choose α	No need to choose α
Needs many iterations	Doesn't need to iterate
Works well even when *n* is large	Need to compute $(X^T X)^{-1}$
	Slow if *n* is very large

- Email: Spam / Not Spam?
- Online Transactions: Fraudulent (Yes / No)?
- Tumor: Malignant / Benign?

In all these examples,

$$y \in \{0, 1\}$$

0: "Negative Class" (e.g., benign tumor)
1: "Positive Class" (e.g., malignant tumor)



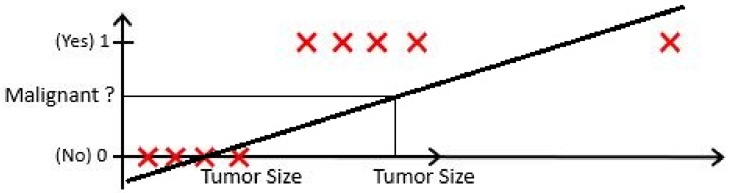
If we apply linear regression algorithm i.e.,

$$h_\theta = \theta^T x$$

then,

Threshold classifier output h_θ at 0.5:

$$\text{If } h_\theta \geq 0.5\text{, predict "}y = 1\text{"}$$
$$\text{If } h_\theta < 0.5\text{, predict "}y = 0\text{"}$$



Will linear regression algorithm apply??

Problems:

- Extreme cases can not be handled
- Bad error function

Classification predicts: $y = 0$ *or* 1
But,
h_θ can be > 1 *or* < 0 when applying linear regression.

Solution:
Logistic Regression gives: $0 \leq h_\theta \leq 1$

Logistic regression model

We want $0 \leq h_\theta \leq 1$
Previously, from linear regression model, we know that

$$h_\theta = \theta^T x \tag{1}$$

modifying equation 1 by introducing **Sigmoid function** or **Logistic function**

$$h_\theta = g(\theta^T x) \tag{2}$$

where,

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

Equation 2 becomes

$$h_\theta = \frac{1}{1 + e^{-\theta^T x}} \tag{4}$$

Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that $y = 1$ on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ tumorSize \end{bmatrix}$

and we get $h_\theta(x) = 0.7$

Tell patient that 70% chance of tumor being malignant

Hypothesis equation can be represented as:

$$h_\theta(x) = P(y = 1|x;\theta) \tag{5}$$

Equation 5 translates as "probability that $y = 1$, given x, parameterized by θ"

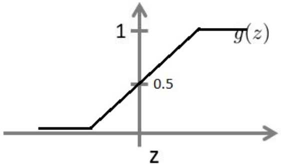
$$P(y = 0|x;\theta) + P(y = 1|x;\theta) = 1 \tag{6}$$

$$P(y = 0|x;\theta) = 1 - P(y = 1|x;\theta) \tag{7}$$

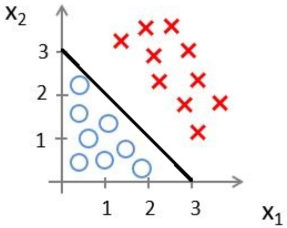
Logistic regression

Referring to equations 2 and 3.

Predict "$y = 1$" if $h_\theta(x) \geq 0.5$	Predict "$y = 0$" if $h_\theta(x) < 0.5$
$g(z) \geq 0.5$	$g(z) < 0.5$
when $z \geq 0$	when $z < 0$
$h_\theta(x) = g(\theta^T x) \geq 0.5$	$h_\theta(x) = g(\theta^T x) < 0.5$
whenever $\theta^T x \geq 0$	whenever $\theta^T x < 0$



Decision boundary



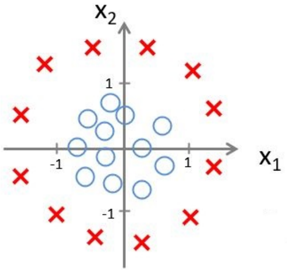
If we have $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
having $\theta_0 = -3,\ \theta_1 = 1$ *and* $\theta_2 = 1$ then,

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$$\implies \quad x_1 + x_2 = 3$$

Also, $x_1 + x_2 < 3$

Non-linear decision boundaries



If we have $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$
having $\theta_0 = -1,\ \theta_1 = 0,\ \theta_2 = 0,\ \theta_3 = 1$ *and* $\theta_4 = 1$ then,

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$$\implies \quad x_1^2 + x_2^2 \geq 1$$

So, $x_1^2 + x_2^2 = 1$ represents the equation on circle with radius 1.

Another equation could be:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \cdots)$$

Cost function (CF)

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix}$ $x_0 = 1, y \in \{0, 1\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

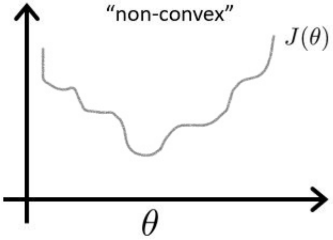
How to choose parameters θ ?

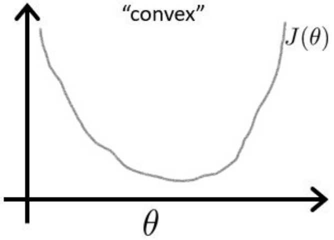
Cost function (CF)

Cost function

Linear regression: $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

$\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

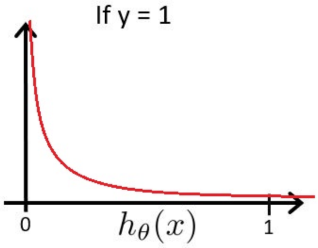




Cost function (CF)

Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Cost $= 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

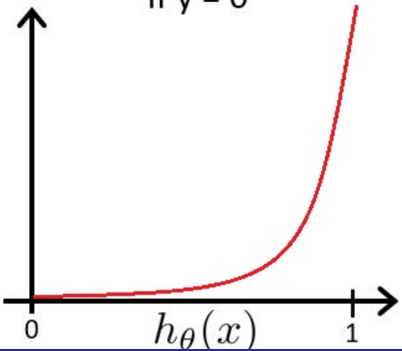
Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x;\theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Cost function (CF)

Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

If y = 0



Simplified CF and GD

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

Simplified CF and GD

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

To fit parameters θ:

$$\min_\theta J(\theta)$$

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Simplified CF and GD

Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

$\}$ (simultaneously update all θ_j)

Simplified CF and GD

Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log (1-h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

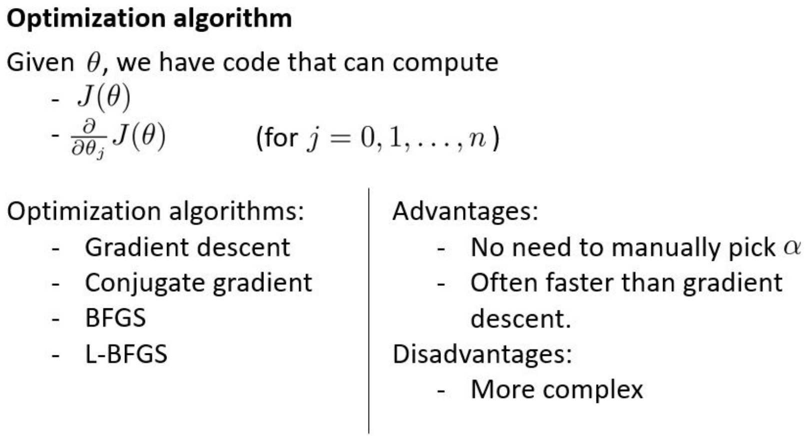
$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$

Algorithm looks identical to linear regression!

Advanced optimization



Optimization algorithm

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial\theta_j} J(\theta)$ (for $j = 0, 1, \ldots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

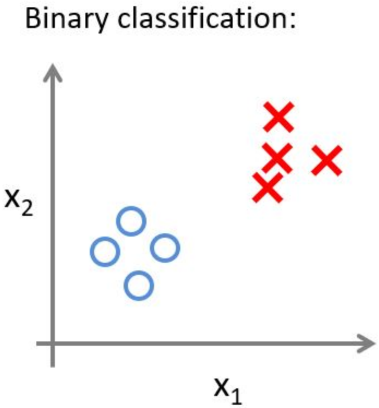
- More complex

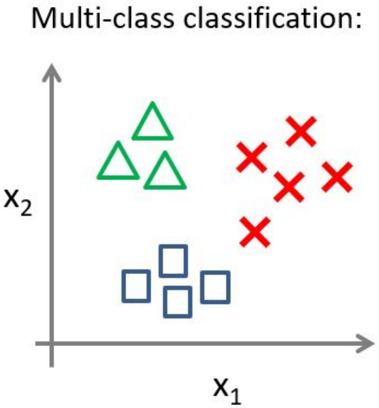
Multiclass

Multiclass classification:

- A classification task with more than two classes.
- Each sample can only be labeled as one class.
- Example 1: Tumor - Benign, stage1, stage2, stage3, stage4.
- Example 2: Weather - Sunny, Cloudy, Rain, Snow.

Multiclass


Binary classification:
x_2
x_1


Multi-class classification:
x_2
x_1

Multiclass

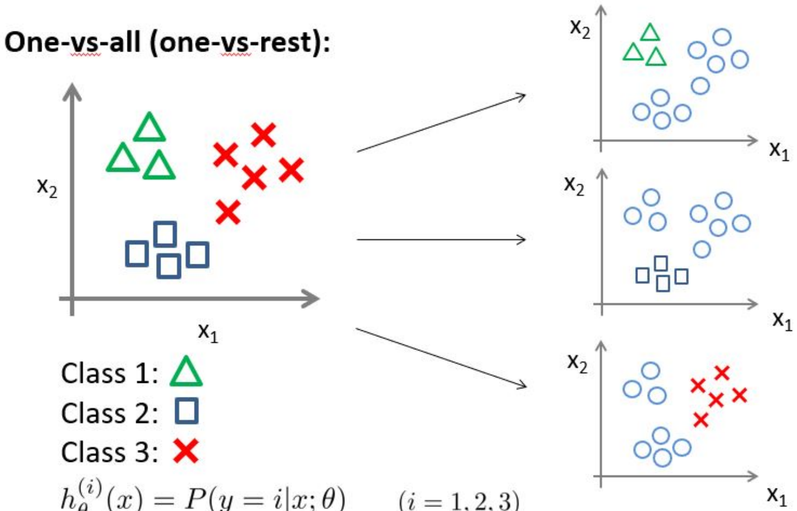
- Some algorithms (such as Random Forest classifiers or naive Bayes classifiers) are capable of handling multiple classes directly.
- Others (such as Support Vector Machine classifiers or Linear classifiers) are strictly binary classifiers.
- However, there are various strategies that you can use to perform multiclass classification using multiple binary classifiers.

Multiclass

- One way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on). Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score. This is called the **one-versus-all (OvA)** strategy (also called **one-versus-the-rest**).
- Another strategy is to train a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on. This is called the **one-versus-one (OvO)** strategy. If there are N classes, you need to train $N \times (N-1)/2$ classifiers.
- Some algorithms (such as Support Vector Machine classifiers) scale poorly with the size of the training set, so for these algorithms OvO is preferred since it is faster to train many classifiers on small training sets than training few classifiers on large training sets. For most binary classification algorithms, however, OvA is preferred.

Multiclass

One-vs-all (one-vs-rest):



Class 1: △

Class 2: □

Class 3: ✗

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$

Multiclass

One-vs-all

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

Another method:

- Softmax

Multiclass

Multilabel Classification
Until now each instance has always been assigned to just one class. In some cases you may want your classifier to output multiple classes for each instance. For example, consider a face-recognition classifier: what should it do if it recognizes several people on the same picture? Of course it should attach one tag per person it recognizes. Say the classifier has been trained to recognize three faces, Alice, Bob, and Charlie; then when it is shown a picture of Alice and Charlie, it should output [1, 0, 1] (meaning "Alice yes, Bob no, Charlie yes"). Such a classification system that outputs multiple binary tags is called a multilabel classification system.