m → # of training examples

x → input variable / feature

y → output variable / target

(x, y) - one training example

(x^i, y^i) - a specific training example

(x^2, y^2) → 2nd example

Size of House (x)	Price \$ (y)
2104	460
1416	232
1534	315
852	178
⋮	⋮

$m = 47$

Use training set get a hypothesis function, which is created by a learning algorithm,

The hypothesis can then predict an output (y) for a certain input (x).

size of house $x \rightarrow$ hye $\rightarrow$ est. value y

h is represented by a straight line → $h_\theta(x) = \theta_0 + \theta_1 x$

or

$h(x) = \theta_0 + \theta_1 x$

$y = a + bx$ (y-int, slope)

⇒ same as function for a straight line

- This is linear regression with one variable
- The variable is x (Univariate linear regression)

Cost function

- helps us figure out the best possible straight line to fit our data

Hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$ → How do we choose those values??

θ_i's are parameters — Depending on the value, the line will change

① We want to choose values for θ_0 and θ_1 so that $h(x)$ is close to the y for the training examples

② We want to minimize θ_0 and θ_1. The difference between $h_\theta(x)$ and y to be small

$h_\theta(x)$ ↓ the estimate — y ↘ the actual value

so $h_\theta(x) - y$ should be small

$\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ (# training examples) minimize the sum of the squares of $h(x) - y$

$\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ minimize the average of

↘ Take the average

$\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ use constant $\frac{1}{2} \Rightarrow \frac{1}{2m}$

write later with easier

gives same values of θ_0 θ_1 as minimizing the function ??

minimize θ_0 θ_1 $\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

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

↘ find me the values of θ_0 θ_1 that causes the hypothesis function to be minimized.

$\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ is expressed as $J(\theta_0, \theta_1)$ so we are minimizing that $\Rightarrow$ $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

$\frac{1}{2}\bar{x}$ → average of the squares of predicted val. − actual

↓ cost function or Squared error function (most commonly used for linear regression, but there are others)

cost is higher for greater differences

Gradient of Descent

Test hypotheses for different values of Θ (Θ_0 Θ_1) to reach the minimum the quickest as possible

$$\left\{ \Theta_j := \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1) \quad \text{for } j=0 \text{ and } j=1 \right\} \text{ repeat until convergence}$$

Assignment
↓
overwrite Θ_j with new value

$\alpha \rightarrow$ Alpha $\rightarrow$ learning rate. Controls the rate at which we descend (the step we take)

$\frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1) \rightarrow$ derivative term (explained later) $\rightarrow$ It is the slope of the tangent line at a certain point of $J(\Theta_0, \Theta_1)$

Θ_0 and Θ_1 should be updated simultaneously

e.g. ① temp0 := $\Theta_0 - \alpha \frac{\partial}{\partial \Theta_0} J(\Theta_0, \Theta_1)$ ⟩ first calculate both

② temp1 := $\Theta_1 - \alpha \frac{\partial}{\partial \Theta_1} J(\Theta_0, \Theta_1)$ ✱ NOT ①→③→②→④

③ Θ_0 := temp0 ⟩ then assign

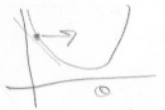
④ Θ_1 := temp1

When the derivative has a positive slope, $\Theta_j := \Theta_j - \alpha$ (pos. num)

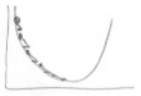
so Θ_j will be updated to a smaller number

When derivative has a negative slope, $\Theta_j := \Theta_j - \alpha$ (neg. num), so $\Theta_j + (\text{num})$

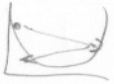
Θ_j will increase



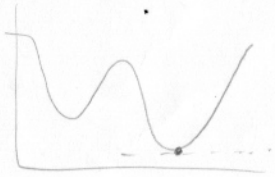
For α, if it's too small, the steps will take a long time



If its too large, it could overshoot the minimum. It could fail to converge of even diverge



Gradient of Descent (con't)



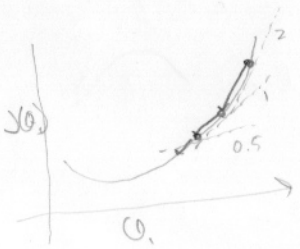
If θ is already at a minimum the slope of the derivative will be 0,

$\theta_j := \theta_j - \alpha \boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$ → is 0

$\theta_j := \theta_j - \alpha \times 0$

So θ will not change!

Gradient of Descent can converge to local minimum even if α is constant, because the slope of the derivative naturally decreases, thus the descent also naturally takes smaller and smaller steps. → No need to decrease α over time



e.g.

$\theta_j := \theta_j - \alpha(2)$

$\theta_j := \theta_j - \alpha(1)$

$\theta_j := \theta_j - \alpha(0.5)$

} Decreases gradually become smaller

Gradient of Descent for Linear Regression

- putting together Gradient of Descent with our hypothesis and Cost function

First we need to figure out the partial derivative term

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

① ↑ cost function

② ↑ hypothesis

③ $\frac{\partial}{\partial \theta_j} \times \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$

④ θ_0, $j=0$: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \rightarrow \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$

θ_1, $j=1$: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \rightarrow \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \times x^{(i)}$

↑ Getting from ③ to ④ requires multivariate calculus but don't need to worry now

Gradient descent algorithm for Linear Regression (cont'd)

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)}_{\text{derivative } \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)}$$

$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}}_{\text{derivative } \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$$

}] update θ_0 and θ_1 simultaneously!

For Linear Regression, the cost function is always a bowl-shaped function (convex function)

→ No local optimum except for the global optimum

Using this to get the optimum hypothesis is called "Batch" Gradient Descent because it looks at the entire batch of training examples at each step.

(some algorithms use a sample of batches)

Multivariate linear regression

- Predict price with multiple features, e.g. size of house, # bedrooms, # floors, ~~Age~~, etc

Size	# Bed	# floor	Age	$
2104	5	1	45	460
1416	3	2	40	232
⋮	⋮	⋮	⋮	
x_1	x_2	x_3	x_4	y

y the output

n = # of features

m = # of examples

$x^{(i)}$ = input (eg. vector) of all features for the i^{th} training set → i = index of set

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

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix} \qquad x_3^{(2)} = 3$$

Hypothesis = $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$ → For convenience, define an extra feature $x_0 = 1$ so $x_0^{(i)} = 1$ → $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ (features)

Example $h_\theta(x) = 80 + 0.1 x_1 + 0.01 x_2 + 3 x_3 - 2 x_4$ (size, #bedrooms, #Floor, Age)

With the convenience feature, we can do matrix calculations!

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

(always 1, so same as above hypothesis)

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

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$ (parameters)

$$\underbrace{[\theta_0\ \theta_1\ \cdots\ \theta_n]}_{\theta^T} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

θ^T: $1 \times (n+1)$ matrix or row matrix

→ that is the inner product between feature vector and parameter vector

Gradient of Descent for Multiple variables

- Basically the same as for one variable, except we have to repeat for "n" features!

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$ ← this is one for x_0

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

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

⋮

}

⇓

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad \text{for } j := 0 \ldots n$$

}

Getting Gradient of Descent to Work well

- speed Gradient Descent up by making each input variable roughly in the same range (θ descends quickly on small ranges but slowly for large ranges)

  E.g. size of house 0 – 2000 ✱ will slow down grad. des.

  but

  rooms 1 – 5

- Ideally, modify input variables so they range:

  $-1 \leq x_{(i)} \leq 1$ or $-0.5 \leq x_{(i)} \leq 0.5$

  ✱ As long as it's close or small range, doesn't have to be exact

- Can use feature scaling or mean normalization

✱ Feature scaling

- Dividing input values by the range (max.val – min.val)

E.g. $x_1 = \frac{\text{size}}{2000}$ ends up $0 \geq x \geq 1$

$x_2 \quad \frac{\#\text{ bedrooms}}{5}$

✱ Mean Normalization

- subtracting Average for an input value resulting in a new average of just 0 for input variable.

Replace x_i with $x_i - \mu_i$ ← Average value

(Do not apply to x_0)

Combine

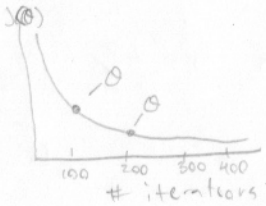
$$x_1 \rightarrow \frac{x_1 - \mu_1}{s_1}$$

μ_1 ← Average value of x in training set

s_1 ← range (max – min) or Standard Deviation

Making sure Gradient of Descent is working properly

- Look at the cost $\min_{\theta} J(\theta)$ after several iterations
- $J(\theta)$ should decrease after every iteration



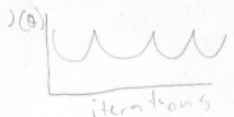
hard to predict how many iterations it will take. 30 - 3000 in 3 will?

You can also use automatic convergence test
e.g. Declare convergence if $J(\theta)$ decreases less than 10^{-3} in one iteration

Not Working:

use smaller α

* If α is small enough, should decrease on every iteration. But too small is slow.



also use smaller α

Choosing α

Try: ..., 0.001, 0.01, 0.1, 1 ...

0.003 0.03 0.3

Find value too small, too large

Features & Polynomial Regression

- We can combine features into one

e.g. House frontage $\times$ depth $=$ area
x_1 x_2 x_3

- Hypothesis does not need to be linear!

→ Can change the hypothesis function by making it a quadratic, cubic, or square root function

E.g. If original hypothesis is

$h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1

Quadratic
↳ $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$

Cubic
↳ $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

Sq. Rt
↳ $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

* Feature Scaling is important here

because e.g. $x_1 \to 1$-1000 range, $x_1^2 \to 1$-1,000,000 range,

$x_1^3 \to 1$-10^9 range

Normal Equation — an alternative to Gradient Descent

- solves for θ analytically without taking several steps
- Runs in one step!
- Does not need Feature scaling!

(Extra) x_0	(x_1) Size	(x_2) Bedrooms	(x_3) Floors	(x_4) Age	Value (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

Create Matrix with inputs ($m \times (n+1)$ matrix)

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

Create vector with outputs (n-dimensional vector)

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

m traing ex, n features

Gradient Desc.
- Need to choose α
- Need many iterations
- Works well even w/ a large n

Normal Eq.
- No need for α
- No iterations
- Need to compute $(X^T X)^{-1}$
  - ↳ Slow if n is very large (over 10,000)

↓ Does not work for more complex algorithms

$$\theta = (X^T X)^{-1} X^T y$$ gives you θ that minimizes cost

X^T transpose of X

$(X^T X)^{-1}$ → inverse of that matrix

Octave: `pinv(X'*X)*X'*y`
- `pinv` ← takes inverse of matrix
- `'` → transpose

Constructing X

each training example looks like $x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$

eg. x^1 above $= \begin{bmatrix} 1 \\ 2104 \\ 5 \\ 1 \\ 45 \end{bmatrix}$ $\Rightarrow X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \end{bmatrix}$

each example is transposed into a row of X

making $m \times (n+1)$ matrix

$m = 4$

midterm exam (x_1)	$(\text{midterm})^2$ (x_2)	final (y)
89	7921	96
72	5184	74
94	8836	87
69	4761	78

Use polynomial regression so the model fits $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$, where x_1 is midterm and x_2 is midterm squared. Want to use feature scaling and mean normalization

What is the normalized feature $x_2^{(2)}$?

$$\frac{(x_2^{(2)} - \text{mean of all } x_2\text{'s})}{\text{range of } x_2} = \frac{4761 - 6675.5}{4075} = -0.37$$

(mean of all x_2's: $(7921 + 5184 + 8836 + 4761)/4$)

(range of x_2: $8836 - 4761$)

Normal Equation — an alternative to Gradient Descent

- solves for θ analytically without taking several steps
- Runs in one step!
- Does not need Feature scaling!

(Extra) x_0	(x_1) Size	(x_2) Bedrooms	(x_3) Floors	(x_4) Age	Value (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

Create Matrix with inputs ($m \times (n+1)$ matrix)

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

Create vector with outputs (n-dimensional vector)

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

m traing ex, n features

Gradient Desc.
- Need to choose α
- Need many iterations
- Works well even w/ a large n

Normal Eq.
- No need for α
- No iterations
- Need to compute $(X^T X)^{-1}$
  - ↳ Slow if n is very large (over 10,000)

↓ Does not work for more complex algorithms

$$\theta = (X^T X)^{-1} X^T y$$ gives you θ that minimizes cost

X^T transpose of X

$(X^T X)^{-1}$ → inverse of that matrix

Octave: `pinv(X'*X)*X'*y`
- `pinv` ← takes inverse of matrix
- `'` → transpose

Constructing X

each training example looks like $x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$

eg. x^1 above $= \begin{bmatrix} 1 \\ 2104 \\ 5 \\ 1 \\ 45 \end{bmatrix}$ $\Rightarrow X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \end{bmatrix}$

each example is transposed into a row of X

making $m \times (n+1)$ matrix