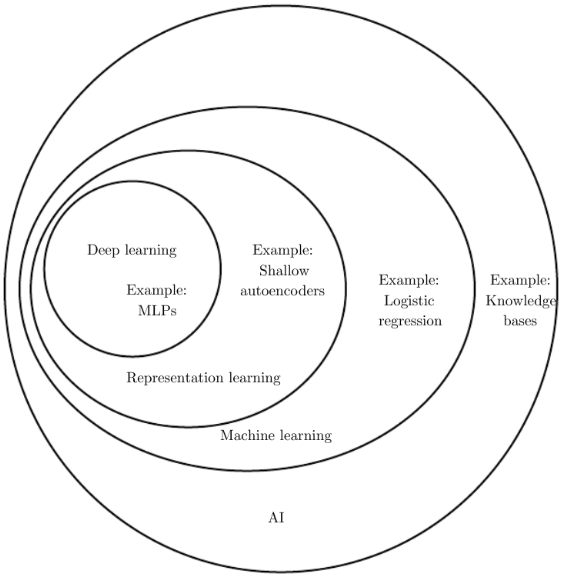
1. Introduction

Machine learning is teaching a computer to perform a task using data or experience.

A computer is said to learn from experience (E) with respect to some task (T) and performance measure (P), If its performance (P) at task (T) improves with experience (E).

Machine learning is a sub field of AI.



Used for:

- Natural language processing (NLP)
- Computer vision

...

There are two main types of machine learning algorithms

1. Supervised Learning - you teach the computer how to perform a task by giving it inputs and the correct output
   - Provide Labelled data

2. Unsupervised Learning - you give the computer only the input and task it to find structure and patterns within the data. (Derives the structure without knowing the effect of the variables)
   - Provide unlabelled data

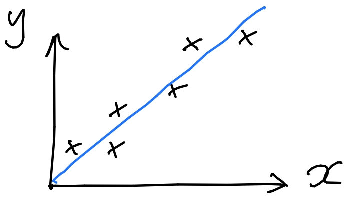
Learning algorithms are like tools and different algorithms should be applied to different problems.

There are two main types of problem

1. Regression
2. Classification

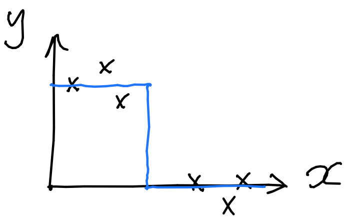
Regression predicts a real valued output → Continuous output

Regression maps an input variable to a continuous function



Classification Predicts an integer output → Discrete output

Classification maps an input variable to a discrete category



2. Model Setup (Supervised)

Use a data set to train and test a learning algorithm.

e.g: Single variable dataset:

x = feature or input variable

y = output

	House Size (m^2)	Value ($)
$x^{(1)}$	50	100,000
$x^{(2)}$	75	200,000
	100	300,000

$\left.\right\}$ $m = 3$ training examples

The 3rd training example $(x^{(3)}, y^{(3)})$

A training set is a list of m training examples

m = number of examples

x = features or input variables

y = output variable

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

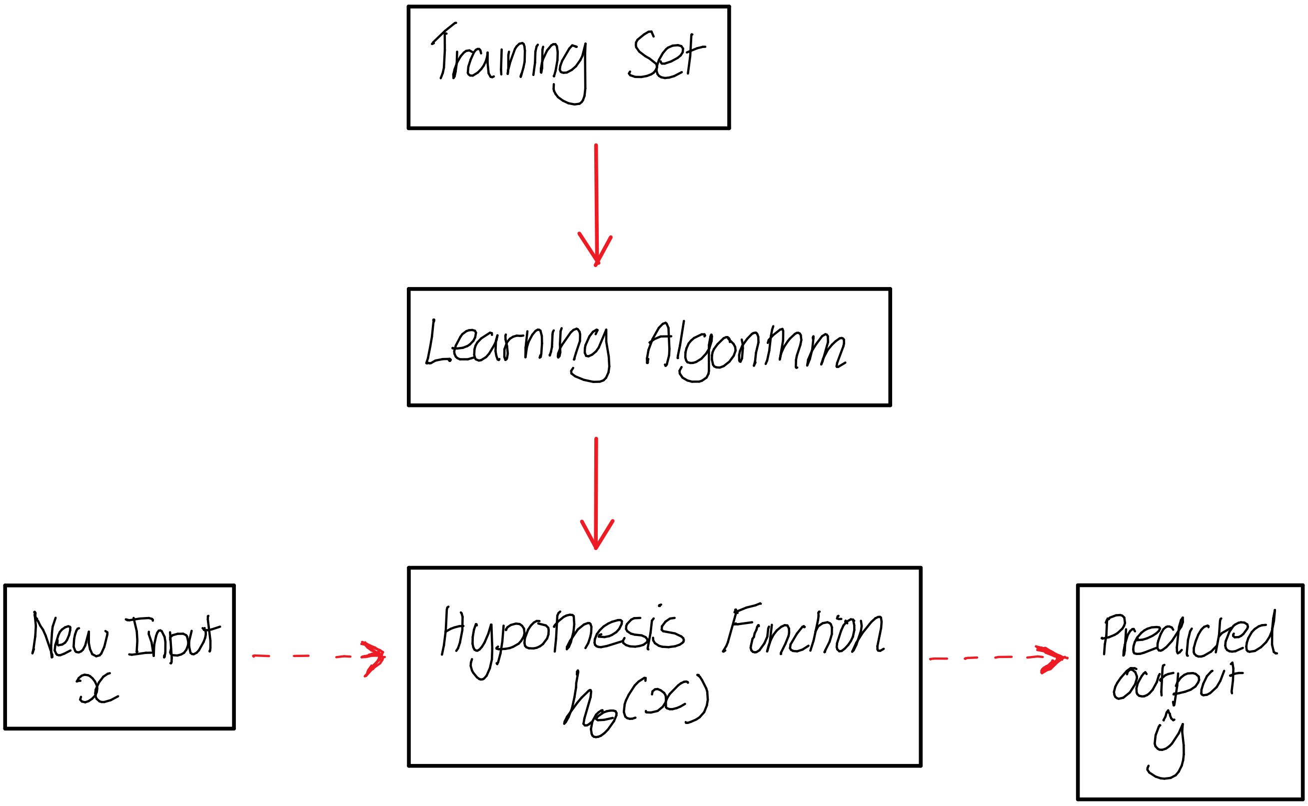
$(x^{(i)}, y^{(i)})$ = the i^{th} training example

e.g $m = 3$ as 3 training examples

x = House size (m^2)

y = House value ($)

$x^{(1)} = 50$, $y^{(1)} = 100\,000$



The learning algorithm uses the training set to fit a function that maps the input to the output

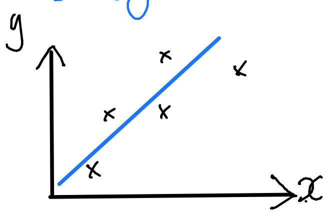
$h_\theta(x)$ is a hypothesis function. It estimates the output for a given input.

↳ need to decide how to represent the hypothesis function $h_\theta(x)$

e.g Single variable linear regression:

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

Predicts y as a linear function of x



θ_0 = intercept
θ_1 = gradient

these parameters are learned/fit during training

if $y = f(x)$ then $h_\theta(x)$ is an estimate of $f(x)$

$$h_\theta(x) = \hat{y} \simeq y = f(x)$$

Formal definition of supervised learning:

Given a training set, learn a function h that maps $x \rightarrow \hat{y}$ so that $\hat{y}$ is a good prediction of y for the corresponding x

The error between the predicted output ($\hat{y}$) and the actual output (y) can be used to define a cost function $J(\theta)$.

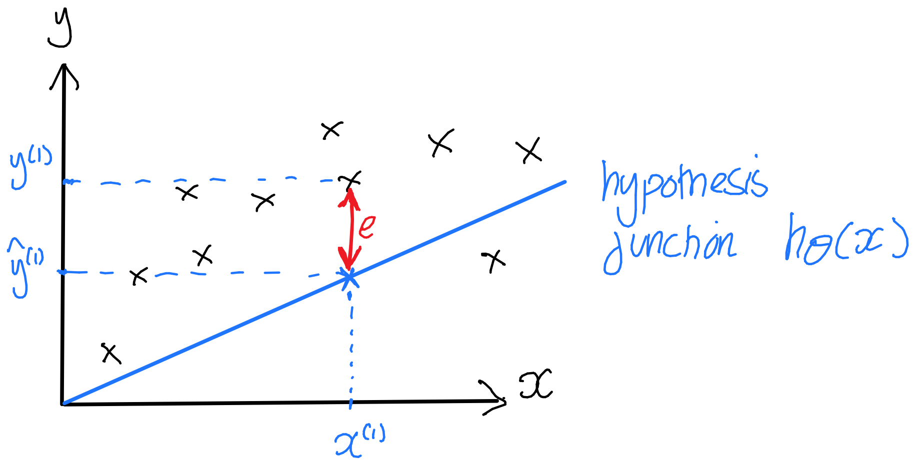
$$\text{error } (e) = \text{Predicted output} - \text{actual output}$$
$$= h_\theta(x) - y$$
$$= \hat{y} - y$$

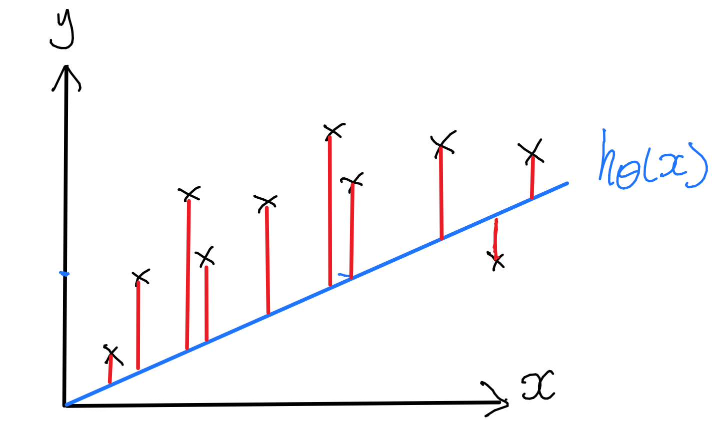
* The cost function $J(\theta)$ can be used to evaluate how well the hypothesis function $h_\theta(x)$ fits the data.

As the error decreases the value of the cost function decreases

$\Rightarrow$ minimise the hypothesis error by minimising the cost function (finding the θ's that minimize $J(\theta)$)

e.g

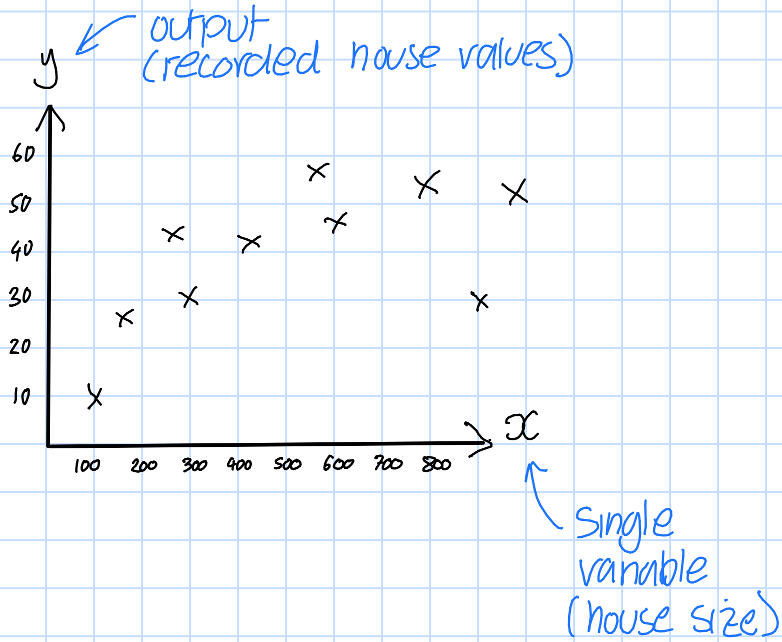




Define cost function to be proportional to the overall error

i.e sum of error^2 etc (mse)

e.g Single variable linear regression:

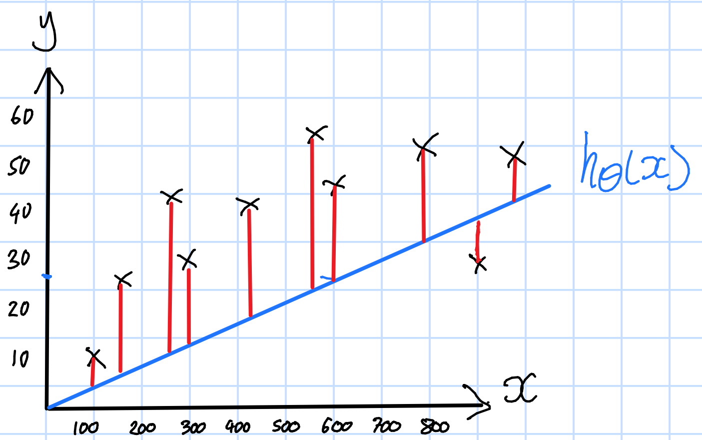


House Size (x)	Value (y)
100	10
150	25
250	45
300	30
410	42
550	57
600	46
790	54
900	30
980	52

From data, hypothesize a linear relationship between x and y

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

Evaluate hypothesis by constructing a cost function based on the error between predicted output and actual output



$$\Rightarrow J(\theta) = \frac{1}{2(10)} \sum_{i=1}^{10} \underbrace{\left(h_\theta(x^{(i)}) - y^{(i)}\right)}_{\text{error}}{}^2$$

Simplifies maths

sum over all training examples

prevents positive and negative errors errors from canceling

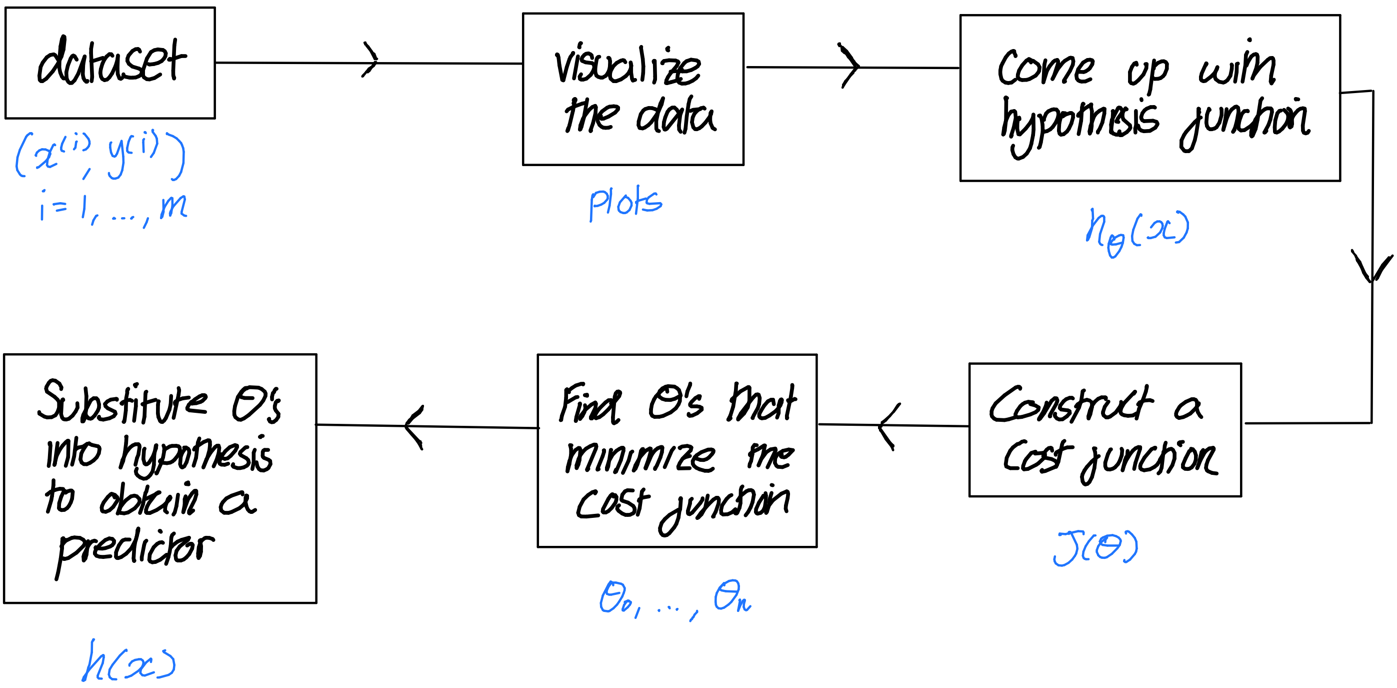
Therefore minimizing the cost function with respect to θ_0 and θ_1 will minimize the total squared error between the predicted output and the actual output

$$\min_{\theta_0,\theta_1} J(\theta) = \min_{\theta_0,\theta_1} \frac{1}{2(10)} \sum_{i=1}^{10} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

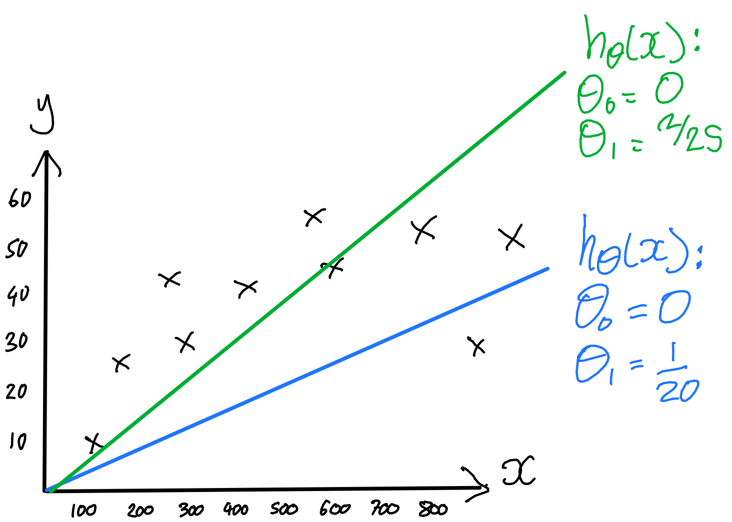
$$\hookrightarrow \sum_{i=1}^{10} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \sum_{i=1}^{10} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

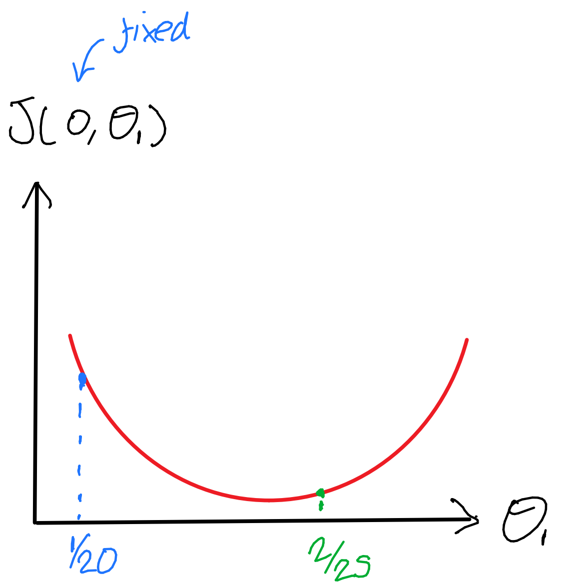
$$= \left[\theta_0 + \theta_1 x^{(1)} - y^{(1)}\right]^2 + \left[\theta_0 + \theta_1 x^{(2)} - y^{(2)}\right]^2 + \dots$$

$$= \left[\theta_0 + \theta_1 (100) - 10\right]^2 + \left[\theta_0 + \theta_1 (150) - 25\right]^2 + \dots$$



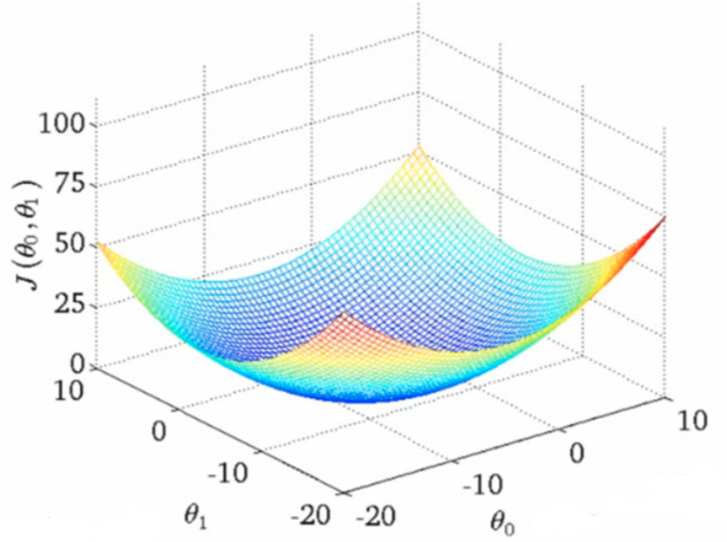
The hypothesis is a function of x and θ where as the cost function is just a function of θ.





If there are n+1 learning parameters $(\theta_0, \theta_1, \ldots, \theta_n)$ then the cost function is n+2 dimensional.

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

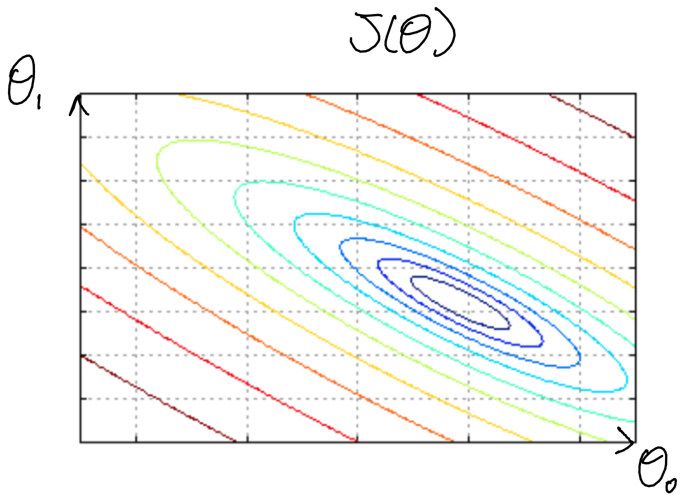


2 learning parameters
$\Rightarrow J(\theta_0, \theta_1)$ is a function of two variables θ_0 and θ_1

* A dimension for each learning parameter and an extra dimension for the function itself

Cost functions are curves or surfaces in one more dimensions than the number of parameters.

Contour plots can be used to visualize certain cost functions.



3. Minimizing Cost functions - Gradient descent

Find a good hypothesis function $h_\theta(x)$ by computing the values of θ that minimize the cost function $J(\theta)$.

One of the most common ways to minimize a given function is **gradient descent**

Intuition :

1) start at some initial point
($\theta_0 = 0, \theta_1 = 0, \ldots \theta_n = 0$)

2) Take steps to reduce the value of the function

move down the gradient of the function

3) Stop when no more steps will reduce the value of the function

stop at a minimum

Depending on the cost function and the chosen initial point, gradient descent may compute different local minimums

The gradient descent algorithm:

repeat {

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

}

Simultaneously updated

For $j = 0, 1, \ldots, n$

n = number of features

repeat { for $j = 0, \dots, n$

$$\theta_j \boxed{:=} \theta_j - \boxed{\alpha} \boxed{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots \theta_n)}$$

}

Assignment opperator. The value of θ_j is iteratively updated

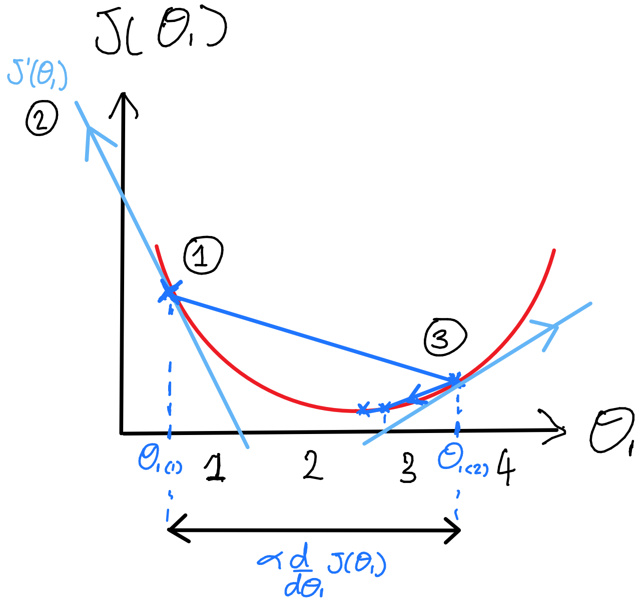
Dictates the step size (Learning rate)

Direction computation. Computes the direction to take a step in by using the gradient of the function.

The parameters must be simultaneously updated.

The learning rate or step factor α does not have to be decreased for gradient descent to converge at a minimum. This is because $\frac{\partial}{\partial \theta_j} J(\theta)$ gets smaller as $J(\theta)$ approaches a minimum.

e.g $J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\theta_1 x^{(i)} - y^{(i)})^2$



① Initial chosen point : $\theta_1 = 0.5$

Choose α : $\alpha = 1$

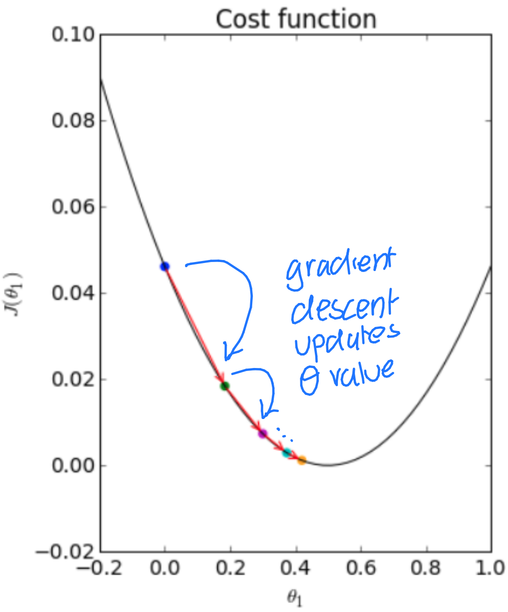
② Compute $\frac{d}{d\theta} J(\theta_1) = J'(\theta_1)$

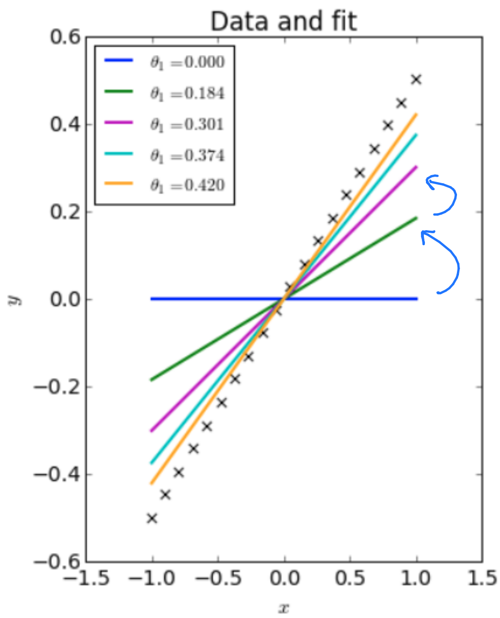
③ Take step $-\alpha J'(\theta)$ down the slope

If the minimum is at a more positive θ_j value (to the right) of the current value for θ_j, then the cost function would have a negative gradient. So the value of θ_j gets increased.

$\theta_j - \alpha$ [negative number] $\Rightarrow$ θ_j + [Positive numer]

$\Rightarrow$ step to right





updated θ values reduce the prediction error ⇒ better fit

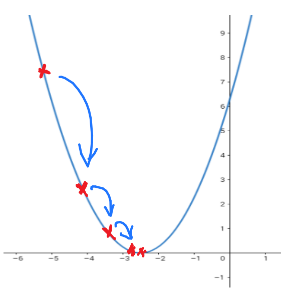
As gradient descent computes the values of θ that minimize the cost function, the overall prediction error decreases and the hypothesis function becomes a better fit for the data.

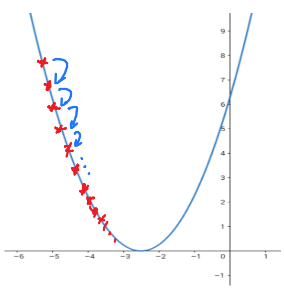
* The choice of learning rate (α) effects Convergence

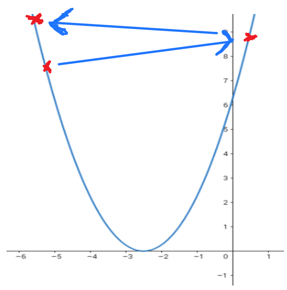
① Good choice of α gives fast convergence to the minimum

② Too small α gives slow convergence to the minimum

③ Too large α gives slow or No convergence



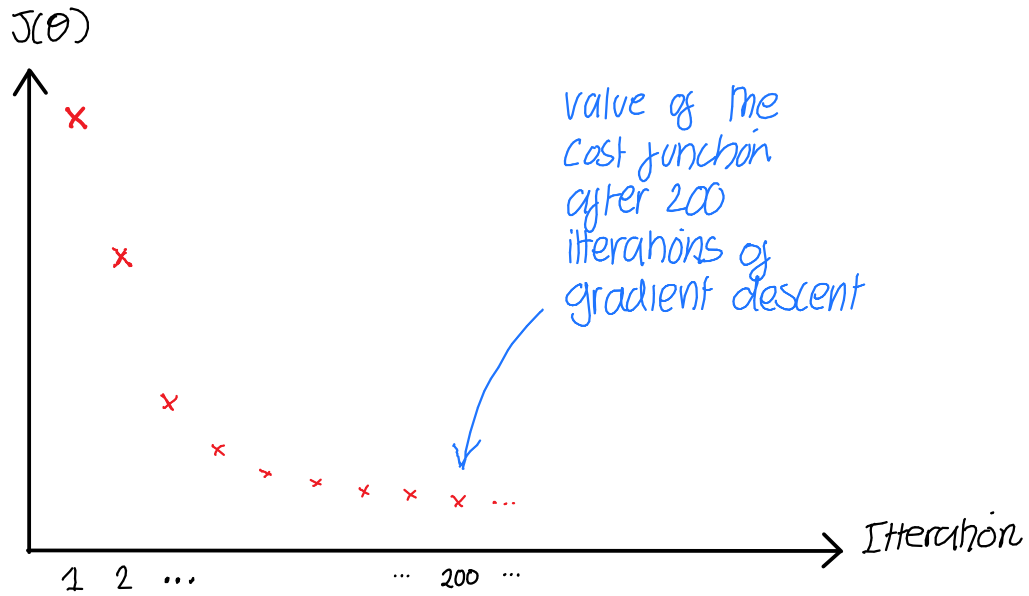




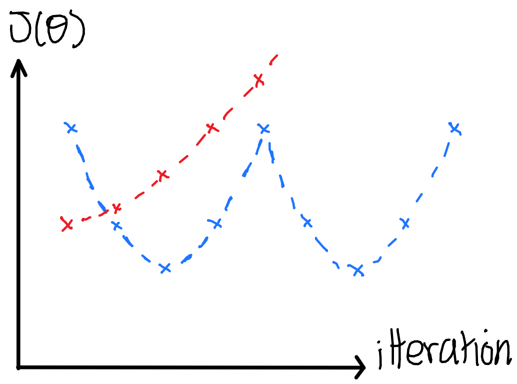
To find a good learning rate:

Plot the value of the cost function after each update iteration. The graph should always be decreasing. If the graph is not decreasing α is too Large. Increase and reduce α by a factor of 3 to find a range of too small to too large. Choose α closest to too large for maximum speed of convergence.

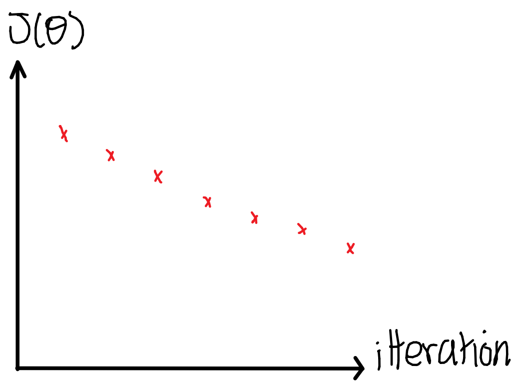
Reasonable choice of α:



α too large:



α too small:



Can use automatic convergence test i.e $J(\theta)$ hasn't decreased by more than ϵ for the past L itterations.
(Can be difficult to choose ϵ and L)

* Try 3 fold increases and decreases based on graph of $J(\theta)$ against number of itterations. Pick value with fastest rate of convergence

Gradient descent for linear regression:

Linear regression hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

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

mean squared error

Gradient descent algorithm: repeat { $\theta_j := \theta_j - \alpha \boxed{\frac{\partial}{\partial\theta_j} J(\theta)}$

}

simultaneous update for $j = 0, \ldots, n$

⇒ Computing the gradient of $J(\theta)$:

$$\frac{\partial}{\partial\theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial\theta_j}\left[\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right] \quad \text{for } j = 0, 1$$

$$= \frac{1}{2m}\sum_{i=1}^{m}(2)(h_\theta(x^{(i)}) - y^{(i)}) \cdot \frac{d}{d\theta_j}(h_\theta(x^{(i)}) - y^{(i)})$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot \frac{d}{d\theta_j} h_\theta(x^{(i)})$$

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

Therefore, the step **direction** to update the values of θ_0 and θ_1 for linear regression is

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

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

Hence the gradient descent algorithm for single variable linear regression is

θ_0 = initial0
θ_1 = initial1

repeat {

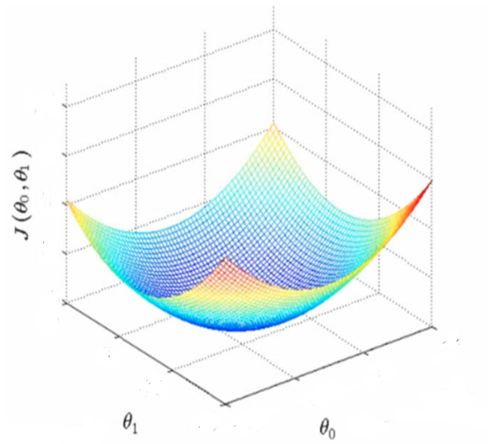
$$temp0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

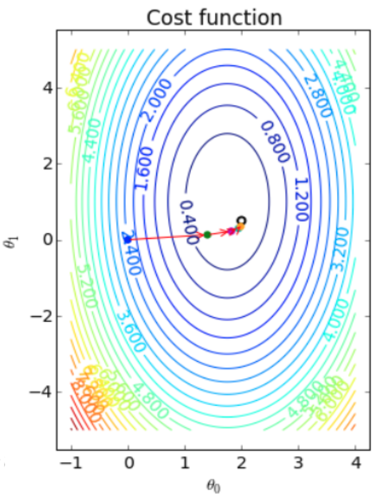
$$temp1 = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) x^{(i)}$$

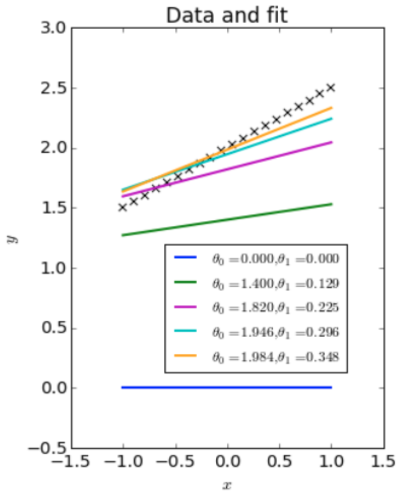
θ_0 = temp0
θ_1 = temp1

} until convergence to local minimum

e.g:







Multivariable Linear Regression:

Predicting a continuous value from **many** features

feature 0	feature 1	feature 2	...	feature n	output
Bias (x_0)	location (x_1)	Age (x_2)	...	House Size (x_n)	Value (y)
1	1000	20		100	10
1	6000	70		150	25
1	70	30		250	$\boxed{45}$
1	800	120		300	30
1	2600	5		410	42
1	⋮	⋮	⋮	550	57
1				600	46
1				790	54
1				900	30
1				980	52

$x^{(3)}$ → feature vector 3

$y^{(3)} \rightarrow \boxed{45}$

$$\vec{x}^{(3)} = \begin{bmatrix} 1 \\ 70 \\ 30 \\ \vdots \\ 250 \end{bmatrix}$$

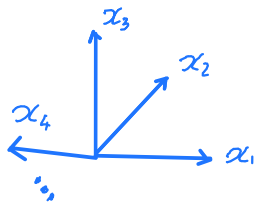
Added bias feature for vectorization

For one learning parameter per input feature the linear regression hypothesis function is

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

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

$$h_\theta(x) = \sum_{k=0}^{n} \theta_k x_k$$



The bias feature ($x_0 = 1$) is added for compactness and improves computational efficiency with use of matricies

Gradient descent for the multivariable linear regression cost function $J(\theta)$ computes the gradient of the multivariable function

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

$$= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot \frac{\partial}{\partial\theta_j} h_\theta(x^{(i)})$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(\left(\sum_{k=0}^{n} \theta_k x_k^{(i)}\right) - y^{(i)}\right) \cdot \frac{\partial}{\partial\theta_j} \sum_{k=0}^{n} \theta_k x_k^{(i)}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left(\left(\sum_{k=0}^{m} \theta_k x_k^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

Sum over all features (with bias feature)

$$\frac{\partial}{\partial\theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[\left(\left(\underbrace{\sum_{k=0}^{n} \theta_k x_k^{(i)}}_{h_\theta(x)}\right) - y^{(i)}\right) x_j^{(i)}\right]$$

The gradient for learning parameter j

Sum over all training examples

The value of feature j in input example i

e.g:

$$x^{(3)} = \begin{bmatrix} 1 \\ 70 \\ \boxed{30} \\ \vdots \\ 250 \end{bmatrix} \qquad x_3^{(3)} = 30$$

Bias (x_0)	location (x_1)	Age (x_2)	...	House Size (x_n)	Value (y)
1	1000	20		100	10
1	6000	70		150	25
1	70	30		250	45
1	800	120		300	30
1	2600	5		410	42
1	⋮	⋮	⋮	550	57
1				600	46
1				790	54
1				900	30
1				980	52

m = number of training examples (= 10)

Bias feature

n = number of features

feature vectors are vectors containing the values for each feature. Feature vectors are column vectors

Hence the gradient descent algorithm for multivariable linear regression is

repeat {

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

}

For $j = 0, 1, \dots, n$ ← One learning parameter per input feature + bias

Simultaneously updated

Matrix implementation:

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

$$= \sum_{k=0}^{n} \theta_k x_k$$

Can be implemented by vector multiplication

$$\begin{bmatrix} a & b & \cdots & z \end{bmatrix} \begin{bmatrix} A \\ B \\ \vdots \\ Z \end{bmatrix} = aA + bB + cC + \dots + zZ$$

$$= \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} = \vec{x}^T \vec{\theta}$$

i^{th} feature vector $\vec{x}^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$ Parameter vector $\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$

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

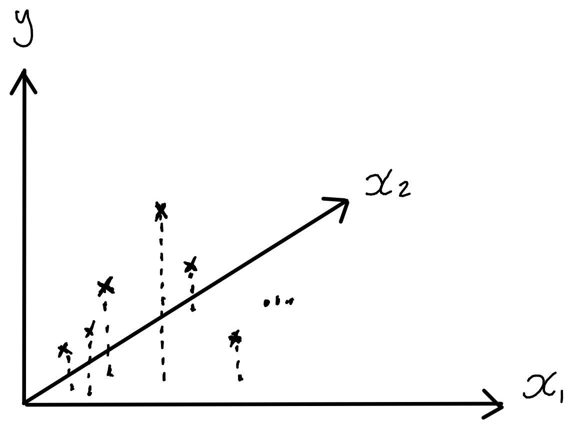
Cost function $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\vec{x}^{(i)T} \vec{\theta} - y^{(i)})^2$

Choosing appropriate features:

Features can be ignored, manipulated or used to create new features.

e.g.:

x_0	length (x_1)	width (x_2)	value (y)
1	30	30	1000
1	⋮	⋮	⋮
⋮			



One learning parameter per input feature

Default Hypothesis: $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots$

If intuition tells you the output value is dependent on the area, then a new area variable should be created.

$$\underbrace{x_3}_{\text{area variable}} = \underbrace{x_1}_{\text{length}} \cdot \underbrace{x_2}_{\text{width}}$$

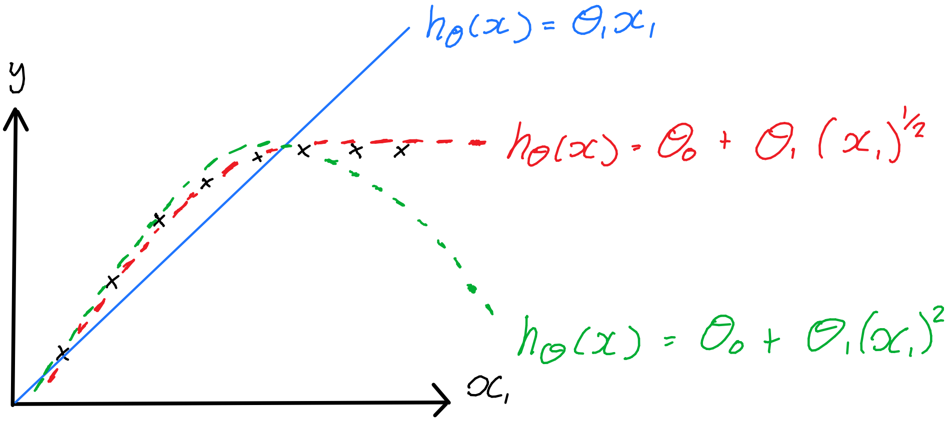
$\Rightarrow$ Hypothesis function can be **chosen** as:

① $h_\theta(x) = \theta_0 x_0 + \theta_3 x_3$

② $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

Individual features can also be manipulated so that the hypothesis function is a better match for the data

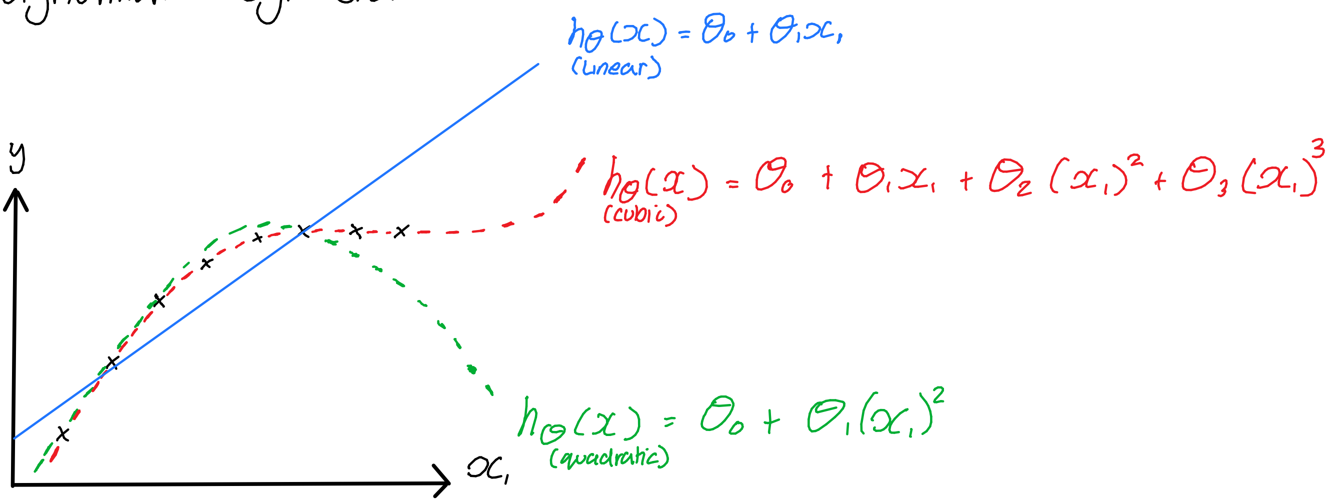
e.g:



⇒ **Multiple** features can be combined into into **one** feature ($x_4 = x_1 x_2 x_3$ or $x_3 = x_1/x_2$ etc ...)

The same feature can also be used in different ways ($x_1 := \sqrt{x_1}$ or $x_1 := (x_1)^3$ etc...)

Polynomial Regression:



⇒ Standard form for polynomial regression hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 (x)^2 + \theta_3 (x)^3 + \dots + \theta_k (x)^k$$

To implement polynomial regression create new features with the manipulated values.

e.g: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 (x_1)^2 + \theta_3 (x_1)^3$ **define:** $x_2 = (x_1)^2$, $x_3 = (x_1)^3$

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

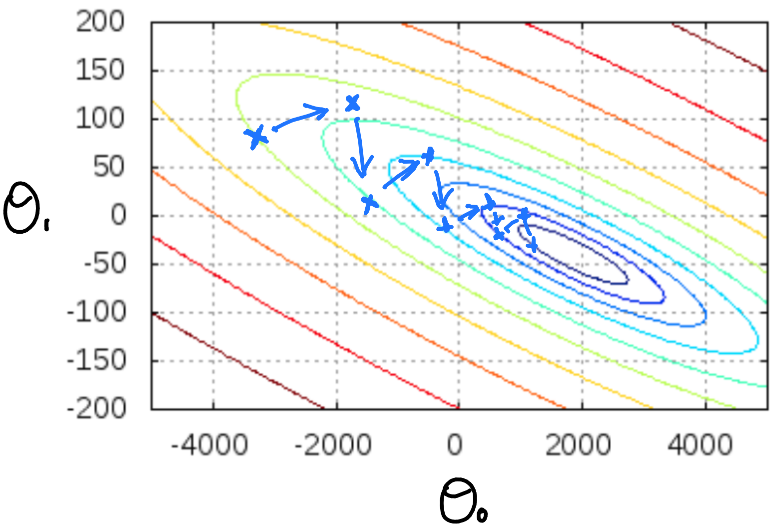
== Multi-variable linear regression

* ⇒ Plot the data and use intuition to choose the function that best fits the data, then choose/create features to match the function.

There is an algorithm to automatically choose a function that fits the data and get matching features

Feature scaling:

Unscaled features can cause a slow rate of convergence.

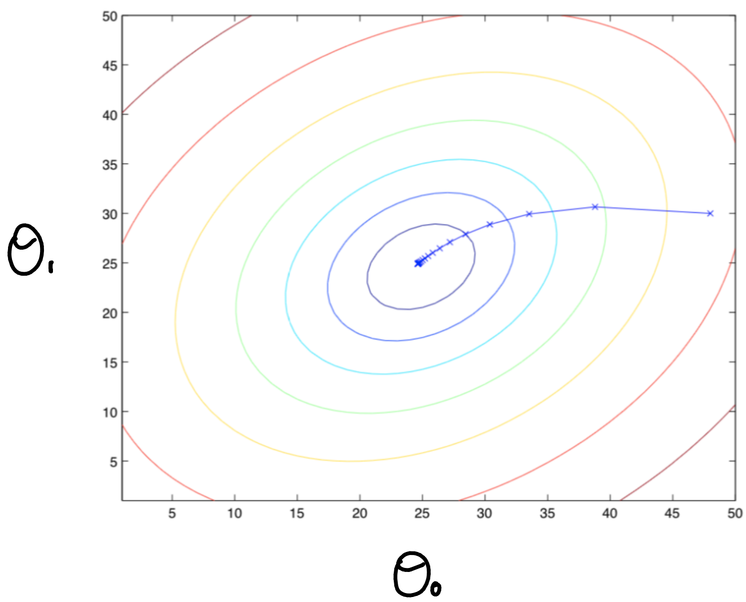


* Feature scaling in polynomial regression is very important as:

if x_1 range = 1 - 10 then:

$(x_1)^2$ range = 1 - 100
$(x_1)^3$ range = 1 - 1000
$(x_1)^4$ range = 1 - 10,000

Make sure different features take on a similar range of values.



Aim to get every feature to be in the range

$-1 \leq x_j \leq 1$

To within a factor of 3 is okay

-3 to +3

$-\frac{1}{3}$ to $+\frac{1}{3}$

Mean normalization $x_j := \dfrac{x_j - \mu_{x_j}}{\sigma_{x_j}}$

4. Minimizing Cost functions - Normal equation

Gradient descent is one way of minimizing a cost function. It iteratively moves down the functions gradient to find the values of θ that minimize the cost function.

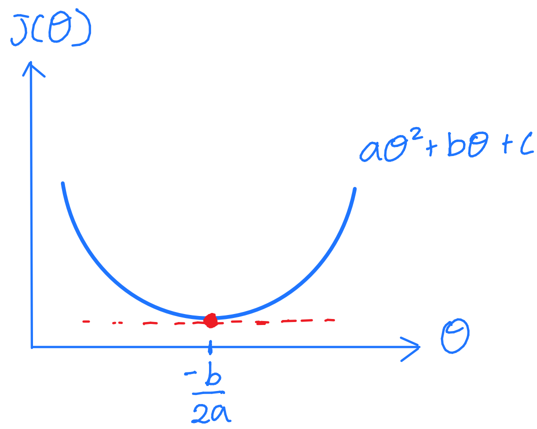
The normal equation finds the minimizing parameters (θ) by "solving" the cost function. It computes the values of θ such that the cost function is at a point where the gradient is 0 (stationary point). All minimum and maximum points are stationary points.
$\Rightarrow$ can find the **global minimum** by finding the stationary values of θ that give the smallest value for the cost function.

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

find point where the gradient is equal to 0

$$\Rightarrow \frac{d}{d\theta} J(\theta) = 2a\theta + b = 0$$

$$\theta = \frac{-b}{2a}$$



For multiple parameters, partially differentiate with respect to each parameter to find the stationary points.

$$J(\theta) = J(\theta_0, \theta_1, \dots, \theta_n)$$

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

$$\underbrace{\frac{\partial}{\partial \theta_j} J(\theta) = \dots = 0}_{\text{Solve this equation to find the minimum point}}$$

$\Rightarrow$ find $\theta_0, \theta_1, \dots, \theta_n$ such that:

$$\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} = 0$$

e.g normal equation matrix implementation:

Bias (x_0)	location (x_1)	Age (x_2)	...	House Size (x_n)	Value (y)
1	1000	20		100	10
1	6000	70		150	25
1	70	30		250	45
1	800	120		300	30
1	2600	5		410	42
1	⋮	⋮	⋮	550	57
1				600	46
1				790	54
1				900	30
1	3000	20		980	52

$(\vec{x}^{(1)})^T \rightarrow$ $\quad\quad \leftarrow \vec{y}$

Construct a matrix X so that each **row** of X is a feature vector

$$X = \begin{bmatrix} 1 & 1000 & 20 & \cdots & 100 \\ 1 & 6000 & 70 & \cdots & 150 \\ 1 & 70 & 30 & \cdots & 250 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 3000 & 20 & & 980 \end{bmatrix} \begin{matrix} \leftarrow (\vec{x}^{(1)})^T \\ \\ \leftarrow (\vec{x}^{(2)})^T \\ \\ \\ \end{matrix}$$

$$= \underbrace{\begin{bmatrix} \leftarrow (\vec{x}^{(1)})^T \rightarrow \\ \leftarrow (\vec{x}^{(2)})^T \rightarrow \\ \leftarrow (x^{(3)})^T \rightarrow \\ \vdots \\ \leftarrow (\vec{x}^{(m)})^T \rightarrow \end{bmatrix}}_{n+1} \updownarrow m \ (=10) \qquad \vec{y} = \underbrace{\begin{bmatrix} 10 \\ 25 \\ 45 \\ \vdots \\ 52 \end{bmatrix}}_{1} \updownarrow m$$

$\vec{x} \in \mathbb{R}^{n+1}$ $\quad$ $X \in \mathbb{R}^{m \times (n+1)}$ $\leftarrow$ X is a $m \times (n+1)$ matrix

$\vec{y} \in \mathbb{R}^{m}$ $\quad$ X is known as the ***design matrix***

$\Rightarrow$ Vectorize and solve the equation $\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} = 0$

To solve for the minimum, that is find the parameter vector $\vec{\theta}$ that minimizes $J(\vec{\theta})$, compute:

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

Where X is the design matrix:

$$X = \begin{bmatrix} \longleftarrow (\vec{x}^{(1)})^T \longrightarrow \\ \longleftarrow (\vec{x}^{(2)})^T \longrightarrow \\ \longleftarrow (x^{(3)})^T \longrightarrow \\ \vdots \\ \longleftarrow (\vec{x}^{(m)})^T \longrightarrow \end{bmatrix}$$

$m \times (n+1)$

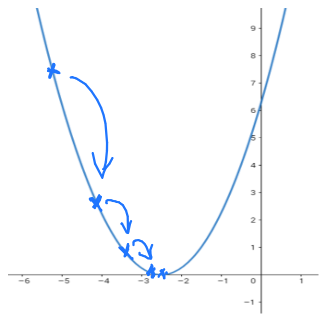
$\vec{y}$ is the output vector:

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

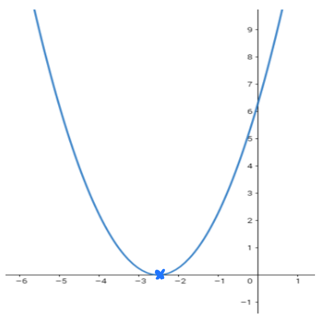
$m \times 1$

Where gradient descent finds $\vec{\theta}$ iteratively, the **normal equation** finds $\vec{\theta}$ **in one shot** (by solving $\frac{\partial}{\partial \theta_j} J(\theta) = 0$)

gradient descent



normal equation



The **normal equation** does **not require feature scaling**

If the design matrix is non-invertible then it's likley that:

① There are redundant/linearly dependent features e.g $x_2 = 2x_1$

$\Rightarrow$ delete redundant features

② Too many features compared to training examples $(n >> m)$

$\Rightarrow$ delete features or regularize

5. Comparing cost minimization by gradient descent and normal equation

Gradient descent	Normal equation
- need to choose learning rate α	+ no α
- need to implement feature scaling	+ no need for feature scaling
- iterative process	+ Solves in one shot
+ Computationally efficient for large numer of learning parameters $O(kn^2)$ ⇒ works well on more complex cost functions.	- Computationally expensive. Slow for large number of learning parameters $\sim O(n^3)$ As need to compute $(X^TX)^{-1}$. if X^TX is non-invertible then use **pevsodo inverse.**
+ Can be used for all cost functions	- Cannot be used for some cost functions.
* use if $n >> 10{,}000$	* use if $n \leq 10{,}000$

Vectorization - hypothesis function

Vectorize to take advantage of advanced linear algebra routines and numerical computation methods.
$\Rightarrow$ vectorized implementations are much **more** efficient.

Predictions: $\vec{h}_\theta(x) = X * \vec{\theta} = \vec{\hat{y}}$

$$= \begin{bmatrix} \leftarrow (\vec{x}^{(1)})^T \rightarrow \\ \leftarrow (\vec{x}^{(2)})^T \rightarrow \\ \leftarrow (x^{(3)})^T \rightarrow \\ \vdots \\ \leftarrow (\vec{x}^{(m)})^T \rightarrow \end{bmatrix} \underbrace{\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \vdots \\ \theta_n \end{bmatrix}}_{\vec{\theta}}$$

$$\vec{h}_\theta(x) = \begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ \vdots \\ h_\theta(x^{(m)}) \end{bmatrix} = \begin{bmatrix} (\vec{x}^{(1)})^T \vec{\theta} \\ (\vec{x}^{(2)})^T \vec{\theta} \\ (x^{(3)})^T \vec{\theta} \\ \vdots \\ (\vec{x}^{(m)})^T \vec{\theta} \end{bmatrix} \begin{array}{l} \leftarrow \hat{y}_1 \\ \leftarrow \hat{y}_2 \\ \vdots \\ \\ \leftarrow \hat{y}_n \end{array}$$

Predictions for each training example.

Vectorization - Gradient descent

Since the gradient descent algorithm for Linear regression is:
(mse cost function)

repeat {
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$
} for $j = 1, \dots, n$

update simultaneously

learning parameter updates can be vectorized.

e.g {

$$\text{temp0} = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\text{temp1} = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_1^{(i)}$$

$$\text{temp2} = \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_2^{(i)}$$

$\theta_0 = \text{temp0}$
$\theta_1 = \text{temp1}$
$\theta_2 = \text{temp2}$

}

can be vectorized and computed in a single line:

$$\{ \ \vec{\theta} = \vec{\theta} - \alpha \boxed{\vec{\delta}} \ \}$$

$\vec{\theta} \in \mathbb{R}^{n+1}$
$\alpha \in \mathbb{R}^{n}$
$\hat{\delta} \in \mathbb{R}^{n+1}$

$$\vec{\delta} = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\ \frac{1}{m}\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_1^{(i)} \\ \frac{1}{m}\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_2^{(i)} \end{bmatrix} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \vec{x}^{(i)}$$

feature vector i
$\vec{x}^{(i)} \in \mathbb{R}^{n+1}$

$$\vec{x}^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$$

① $\frac{1}{m} \sum_{i=1}^{m} \underbrace{\left(h_\theta(x^{(i)}) - y^{(i)} \right)}_{i^{th} \text{ error}} \vec{x}^{(i)}$

② $h_\theta(x^{(i)}) = \sum_{k=0}^{n} \theta_k x_k^{(i)} = (\vec{x}^{(i)})^T \vec{\theta}$

from ② make a column vector containing all errors. (A prediction error for each example) →

$$\vec{e} = X\vec{\theta} - \vec{y}$$

predictions. one prediction per example

Prediction value 1

from ①, each error value multiplies/scales its corresponding training example/feature vector →

want: $e_1 \vec{x}^{(1)} + e_2 \vec{x}^{(2)} + \dots$

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

$= \delta =$ gradient
$\Rightarrow$ setting to 0 would result in the normal equation

$$\boxed{\Rightarrow \vec{\delta} = \frac{1}{m} X^T (X\vec{\theta} - \vec{y})}$$

Summary

The two main types of learning Algorithms:
- Supervised
- Unsupervised

The two main classes of problem:
- Regression
- Classification

definitions:

m = number of training examples
n = number of features
$x^{(i)}$ = input features for the i^{th} training example
$y^{(i)}$ = output for the i^{th} training example
$\theta_{(j)}$ = The learning parameter for the j^{th} feature

$\vec{x}^{(i)}$ = input feature vector for i^{th} training example
$\vec{y}$ = output feature vector
X = The design matrix (rows are the input feature vectors)
$\vec{\theta}$ = The vector of learning parameters

Training data → hypothesis function $h_\theta(x)$ → Cost function $J(\theta)$ → minimize $J(\theta)$

hypothesis function $h_\theta(x)$:
- Linear regression (Multivariable + Polynomial)

Cost function $J(\theta)$:
- mse

minimize $J(\theta)$:
- gradient descent
- normal equation

↓

Subsitute θ's into hypothesis function forming a predictor
- test and validate

→ (back to hypothesis function)

linear regression fits a curve to a dataset by finding the curve parameters that minimize the total error.

hypothesis for linear regression:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n = \sum_{k=0}^{n} \theta_k x_k$$ ← multivariable + polynomial (manipulations are redefined to new variables)

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1$$ ← Single variable linear regression

bias feature (x_0)

gradient descent minimizes $J(\theta)$ by taking steps down the gradient of the function.

θ_j = initial value

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

Simultaneously update for $j = 0, \ldots, n$

learning rate / step factor. choice effects convergence
($\times \div$ 3 till find fast convergence)

For linear regression with mse cost function:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

gradient descent requires feature scaling for fast convergence

mean normalization: $\frac{x_j - \mu_j}{\sigma_j}$

range scaling: $\frac{x_j}{\max x_j - \min x_j}$

Vectorized MSE cost calculation: $\frac{1}{m} (X\vec{\theta} - \vec{y})^T (X\vec{\theta} - \vec{y})$

Vectorized gradient descent:

$X^T(X\vec{\theta} - \vec{y})$ computes the gradient of $J(\theta)$. Solving $X^T(X\vec{\theta} - \vec{y}) = 0$ finds the minimum in one shot

$\vec{\theta}$ = initial vector

repeat {
$$\vec{\theta} = \vec{\theta} - \alpha \frac{1}{m} X^T (X\vec{\theta} - \vec{y})$$
}

The normal equation:

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

$n \ll 10{,}000$ use normal equation
$n \gg 10{,}000$ use gradient descent