Linear Regression

Adam Carter, EPCC, The University of Edinburgh
a.carter@epcc.ed.ac.uk

1 September 2020

www.archer2.ac.uk

Reusing this material

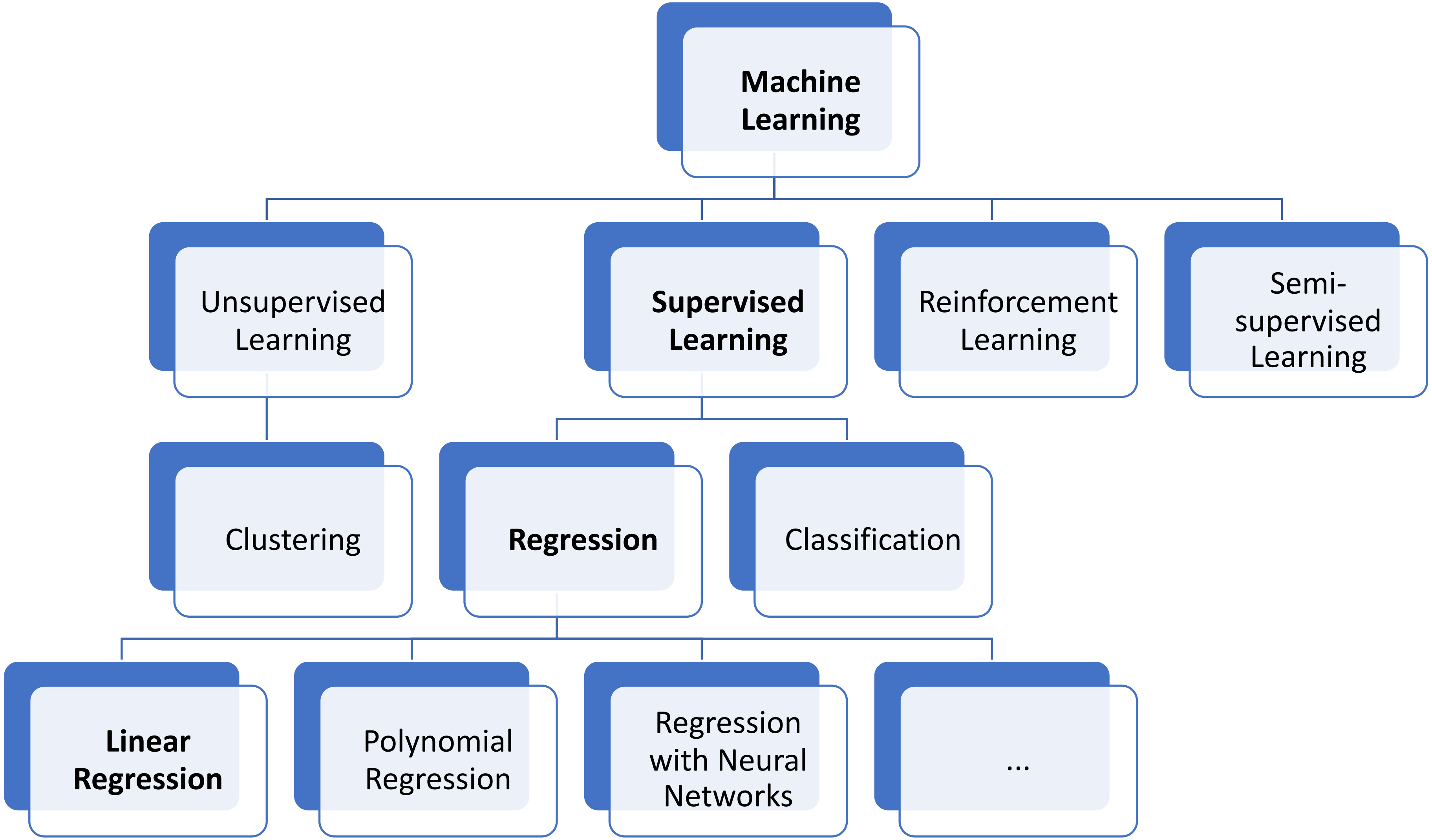
epcc

Partners

Purpose of this Talk

1. Primarily here as a way to introduce concepts in machine learning such as models, optimisation, cost functions, distance metrics, iterative algorithms which appear in many machine learning algorithms
2. …and secondarily to describe how linear regression works
3. Will also introduce a *little* bit of mathematical notation that's often used in the field

The presentation focuses on Linear Regression as an example of machine learning, rather than looking at it from a statistics point of view.

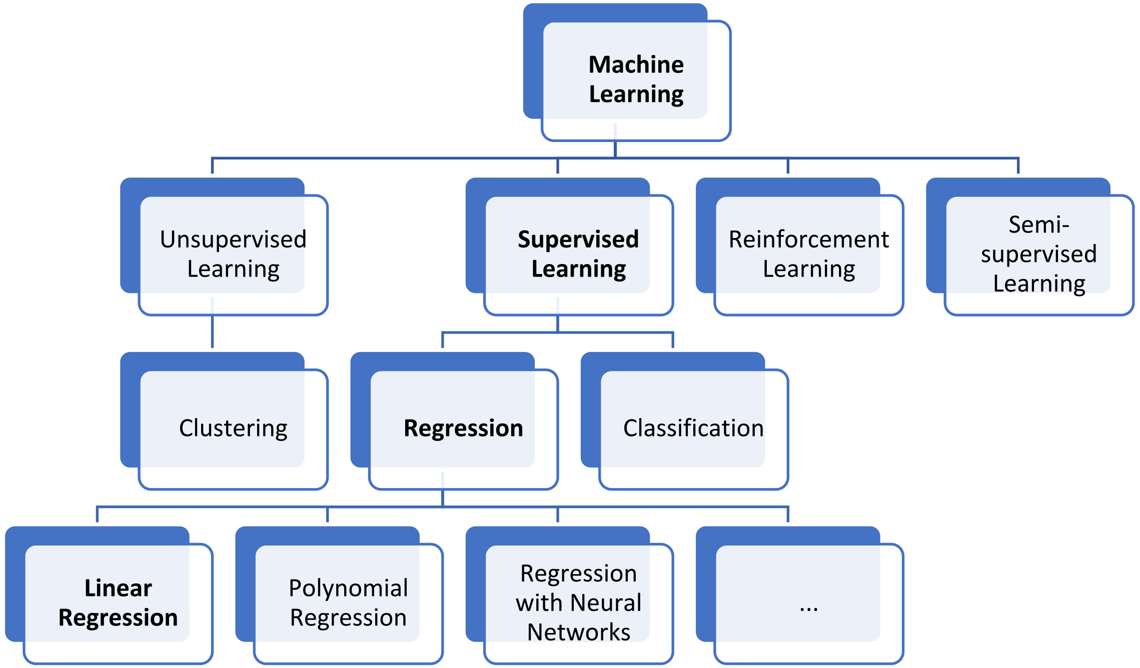

Machine Learning
Unsupervised Learning
Supervised Learning
Reinforcement Learning
Semi-supervised Learning
Clustering
Regression
Classification
Linear Regression
Polynomial Regression
Regression with Neural Networks
...

Linear Regression

- Linear Regression predicts a continuous variable from one or more variables

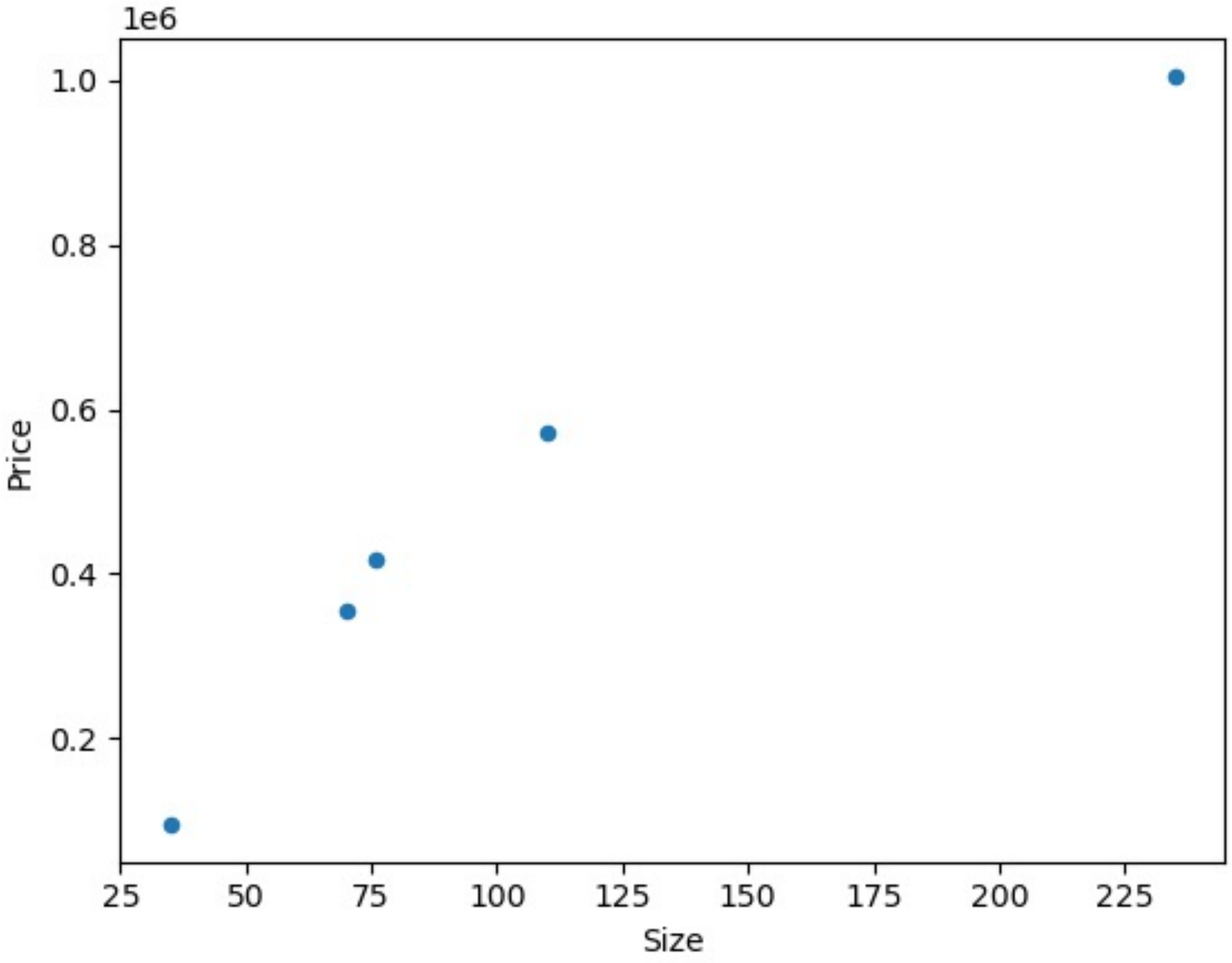
- One of the simplest predictive models

- Existing data is used to create a **linear model**



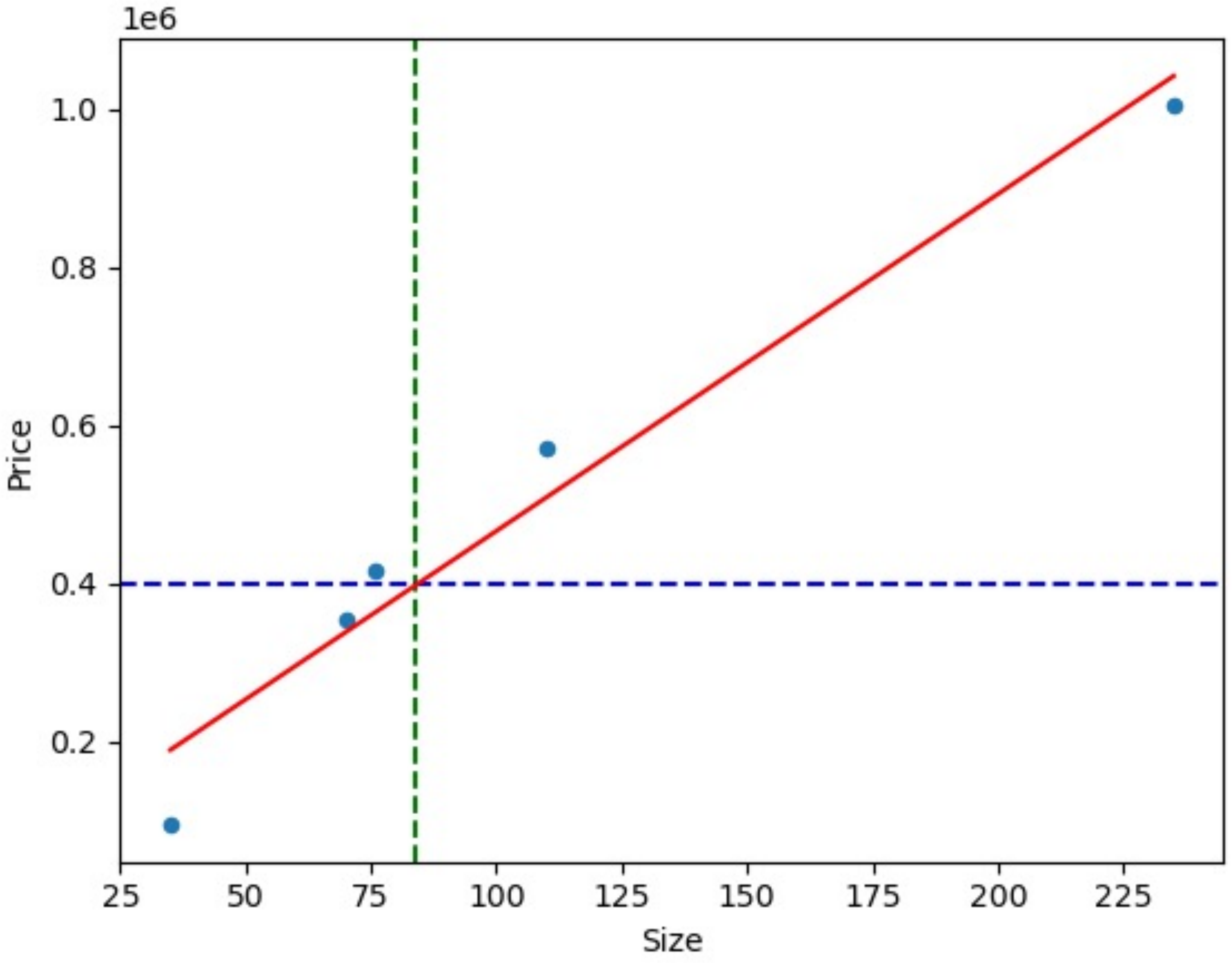
1. Linear Regression - Example: House Prices

Size in m^2 (x)	Price in £ (y)
70	355,000
110	571,000
35	95,500
235	1,005,000
76	417,000
84	?



1. Linear Regression - Example: House Prices

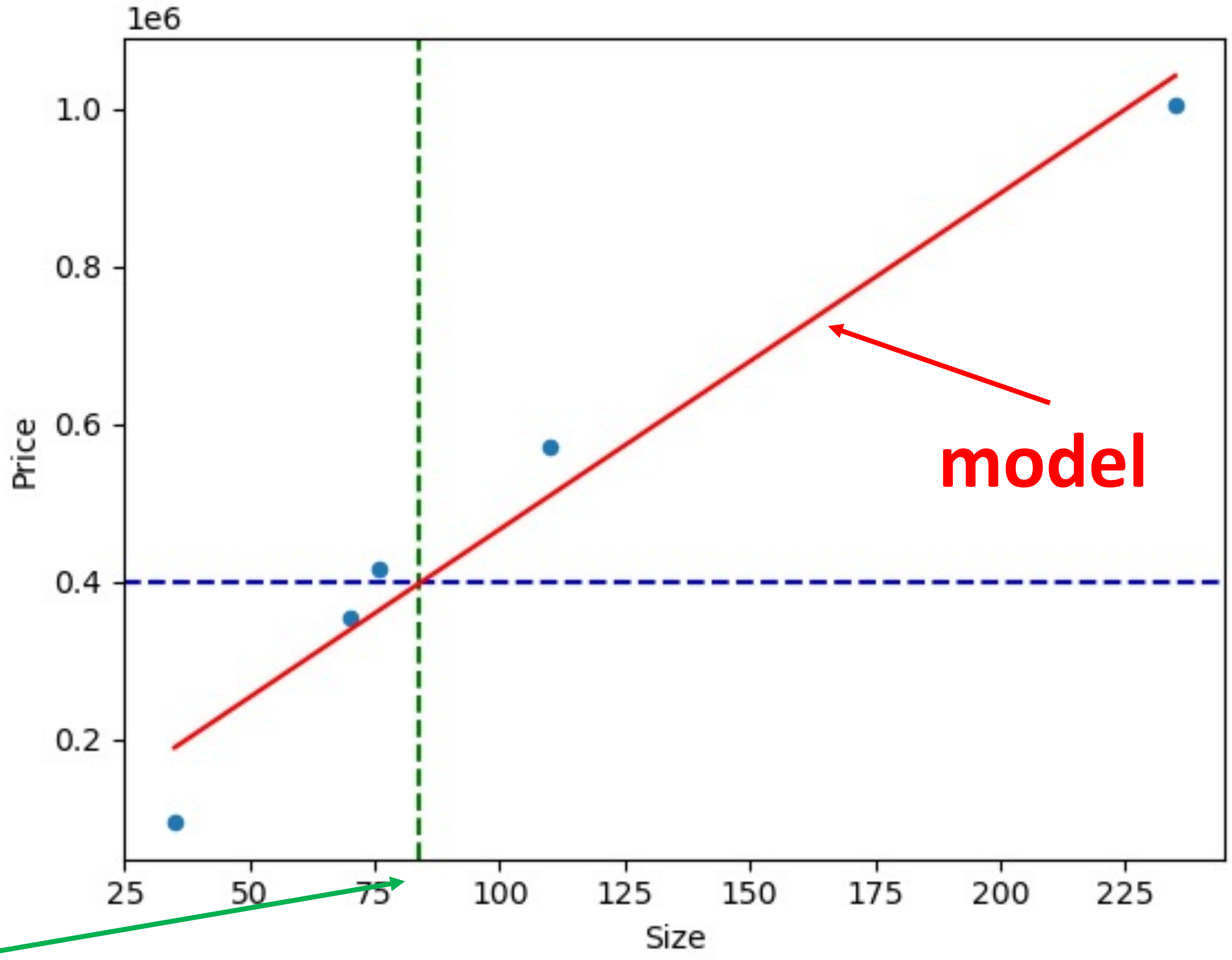
Size in m² (x)	Price in £ (y)
70	355,000
110	571,000
35	95,500
235	1,005,000
76	417,000
84	?



1. Linear Regression - Example: House Prices

training set

Size in m² (x)	Price in £ (y)
70	355,000
110	571,000
35	95,500
235	1,005,000
76	417,000
84	?



unseen data

Linear Regression: Supervised Learning

epcc

Training Set

Size in m^2 (x)	Price in £ (y)
70	355,000
110	571,000
35	95,500
235	1,005,000
76	417,000

Learning Algorithm

(Linear Regression)

Model

$Price = 4261.61 \times Size + 40378.5$

$Price$ =

84

Apply Model

$Size$ =

398353.8

Notation

Size in m² (x)	Price in £ (y)
70	355,000
110	571,000
35	95,500
235	1,005,000
76	417,000
84	?

Training set
m=5

Here, the price is a function of one **feature**, so *n*=1

m	number of training examples
n	number of features
x	input variable (or feature)
y	output (or target) variable (or feature)
(x, y)	one training example (one row)
$(x^{(i)}, y^{(i)})$	i^{th} training example (i^{th} row)
$h_\theta(x)$	hypothesis function
θ_j (where $j = 0, \dots, n$)	parameters

So, for simple linear regression as we have here: $h_\theta(x) = \theta_0 + \theta_1 x$

where θ_0 is a constant

Hypothesis Function – A Simple Example

- We choose our **hypothesis function** to be a linear function with one variable x

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

  - That is, the equation of a straight line, with y-intercept θ_0 and slope θ_1
- Once we find the optimal parameters, this function will represent the model.
- Models like this are sometimes referred to as **parameterised models**
  - Most machine learning models are some kind of parameterised model. We choose the general form of the model, and the computer *learns* the parameters.
  - We have two parameters here. Deep neural networks can have millions…

Cost Function – A Simple Example

epcc

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

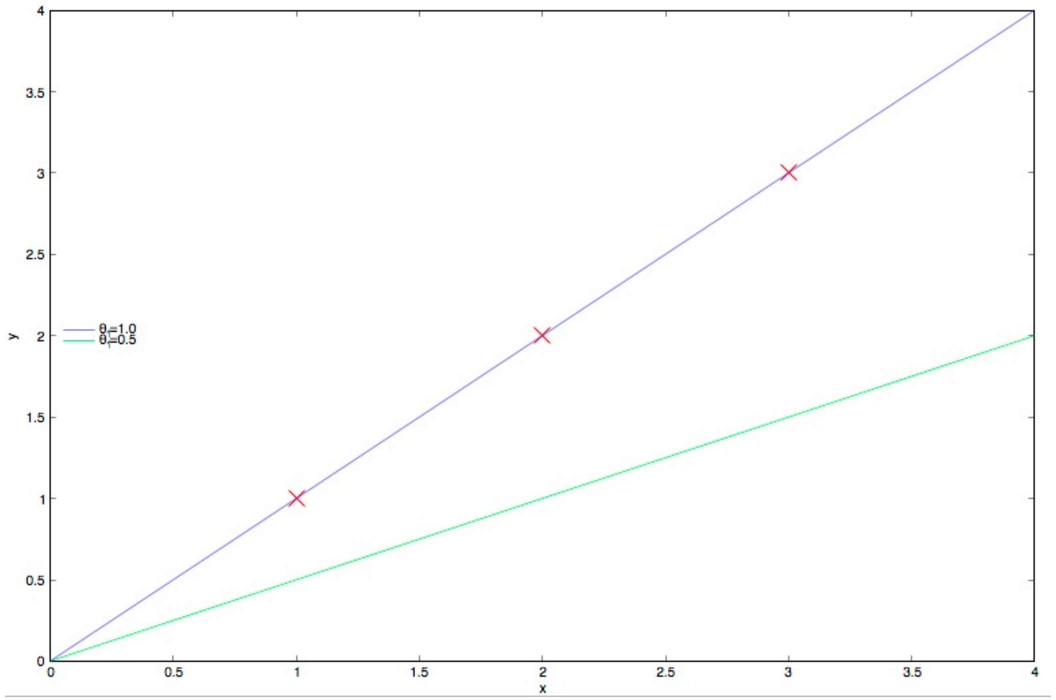
- So, given our hypothesis function, how do we choose θs?
- We want $h_\theta(x)$ as close to y as possible
  - => We want $(h_\theta(x) - y)$ to be as small as possible
- We define a **cost function**, $J(\theta_0, \theta_1)$, which is a measure of how far away the hypothesis function is from the measured values of the target variable in the training set:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

- Note that the we can choose the cost function here. Simple linear regression usually implies minimising the **mean square error** which gives the expression above.
- Finally, we **minimise the cost function**, that is, we find the values of θ_0, θ_1 for which J is smallest.

The cost function – simplified example

- Imagine a case where our training set values fall exactly on the line $y = x$.
- Assume $\theta_0 = 0, h_\theta(x) = \theta_1 x$
  - Here the hypothesis function h is passing through (0,0)
  - The choice for θ_1 controls the slope of the straight line
- Let's consider the case where we initially choose $\theta_1 = 0.5$
  - $J(\theta_1 = 0.5) = \frac{1}{6}\left((0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right) = 3.5$
  - So, for our initial guess, we get a value of 3.5 for the cost function
  - We can clearly do better, so how do we make our next guess?



x	$y, h(\Theta_1 = 1.0)$	$h(\Theta_1 = 0.5)$
1	1	0.5
2	2	1.0
3	3	1.5

A simple example (continued)

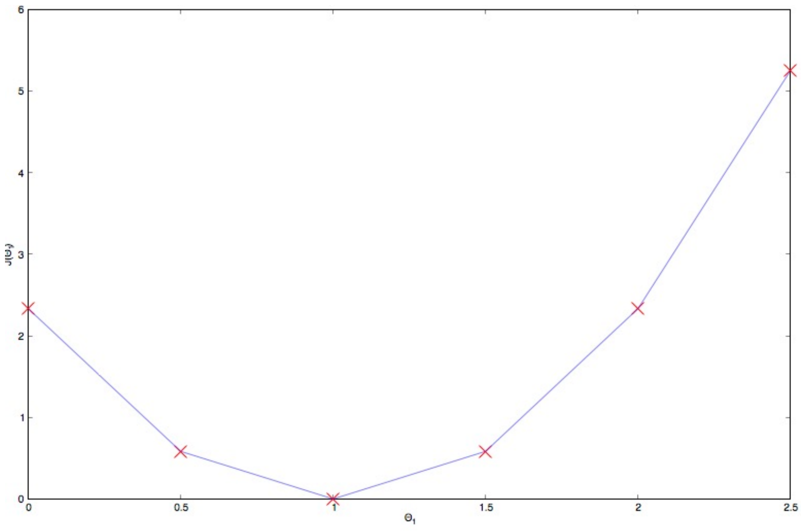
- For this specific training set, let's see how $J(\theta_1)$ varies as a function of θ_1…
- In our case the training set target variables y are given by the equation $y = x$ so we can substitute this value for y and we can substitute $h_\theta(x) = \theta_1 x$ into the usual expression for the cost function

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

$$= \frac{1}{6}\sum_{i=1}^{3}\left(\theta_1 x^{(i)} - x^{(i)}\right)^2$$

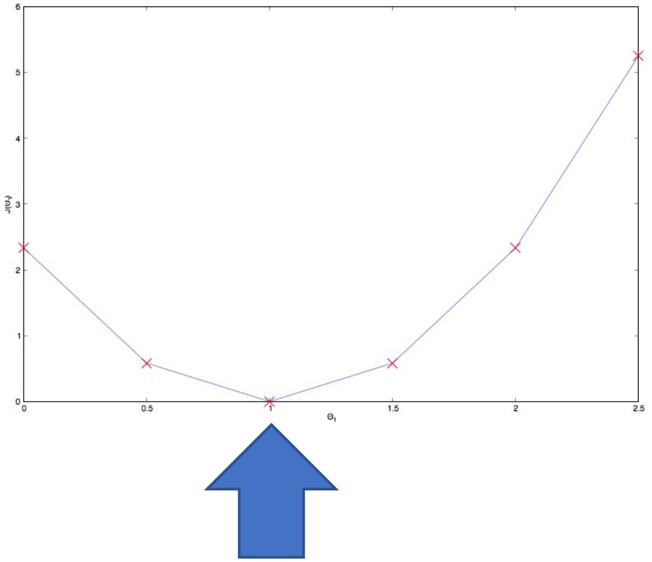
$$= \frac{1}{6}(\theta_1 - 1)^2\sum_{i=1}^{3}\left(x^{(i)}\right)^2$$

$$= \frac{1}{6}(\theta_1 - 1)^2(1^2 + 2^2 + 3^2)$$



Optimization Algorithms

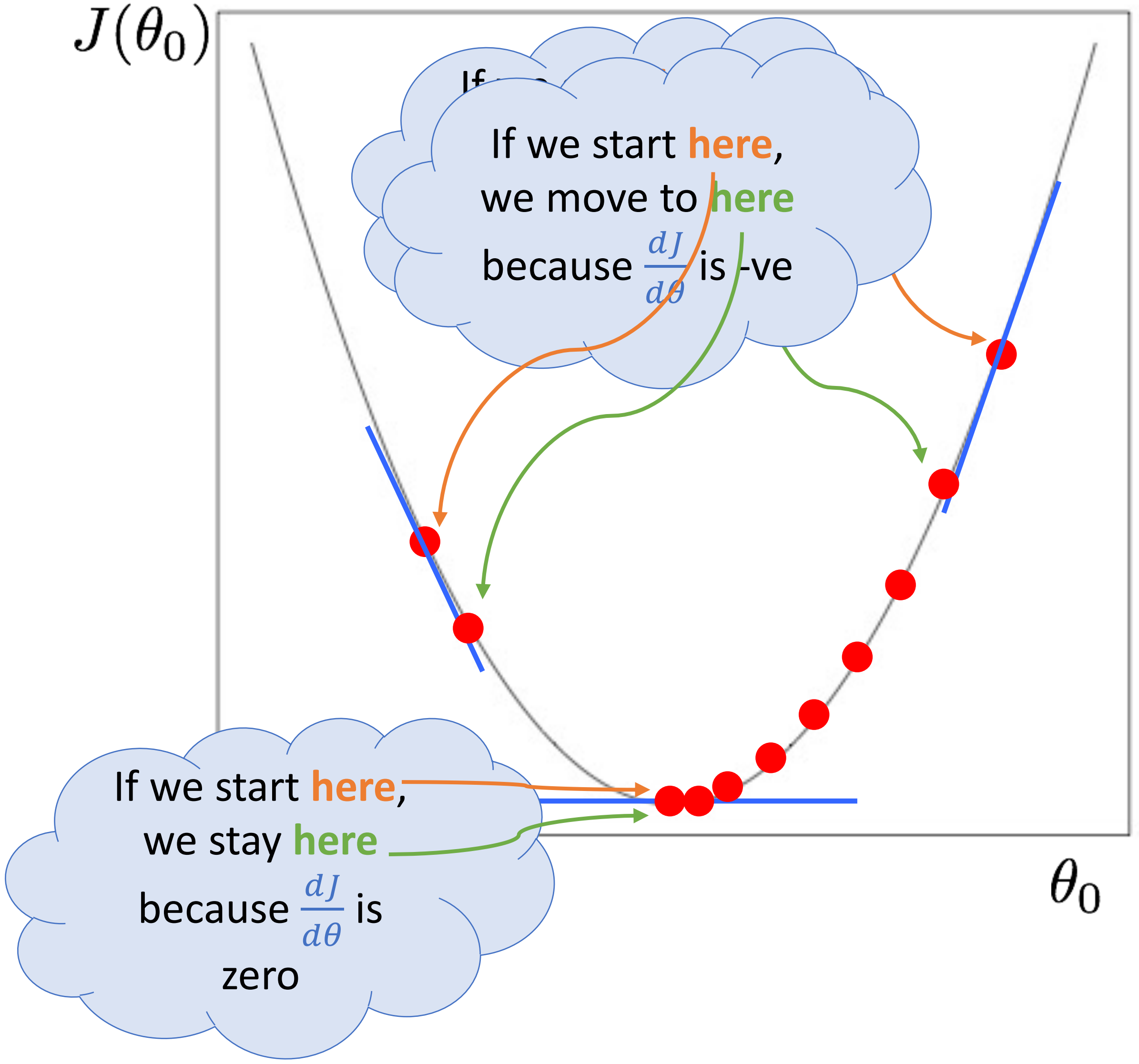
- Given that we have a way to calculate our cost function for any value of our parameters, how do we **minimise** the cost function?
- Many optimization algorithms exist
- Most common approach is to use a version of the iterative **gradient descent algorithm** such as:
  - **Batch Gradient Descent** (using all m training examples)
  - **Stochastic Gradient Descent** (use one example in each iteration)
  - **Mini-batch Gradient Descent** (use b examples (where $1 \leq b \leq m$) in each iteration)



Optimising the Cost Function

- Iterative methods like gradient descent are common in machine learning algorithms including, for example, neural networks
- In some rare cases – and linear regression is an example of one of these – it’s possible to find an exact solution analytically
- For linear regression, the parameters can be found exactly using the so-called **normal equation**

Gradient descent with one parameter

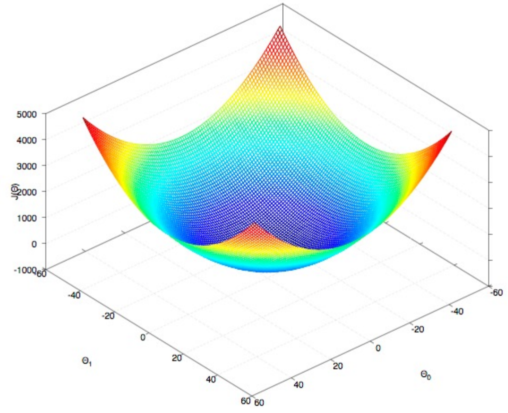


$$\theta_0 := \theta_0 - \alpha \frac{\mathrm{d}}{\mathrm{d}\theta_0} J(\theta_0)$$

- In machine learning, α is known as the **learning rate**
- α is sometimes referred to as a **hyperparameter**
  - larger α leads to faster learning, but *can* let you “overshoot” the minimum
  - smaller α leads to slower learning, but makes it easier to converge

More generally…

- Linear regression involves solving for multiple dimensions (multiple features) to minimise $J(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$
- We repeat the following step, until convergence:
  - $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \forall j, \alpha > 0$
  - We simultaneously update for all θ=$(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$
  - Learning rate α (how big a step you take)
    - α too small: slow process until convergence to minimum
    - α too large: overshoot minimum, get further and further away
- Since $J(\theta)$ is always convex there are no local minima



Gradient Descent – for Linear Regression

- To perform this update, we need $\frac{\partial J}{\partial \theta_j}$ for all j
- In the general case:

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

- This can be evaluated for all n, but let's consider the simpler case where $n = 2$:

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \left[\frac{1}{2m} \sum \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)^2 \right]$$

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

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

…putting these back into the update equation

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

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

- Substituting back into the gradient descent update function, $\theta_j \coloneqq \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$, gives:

$$\theta_0 \coloneqq \theta_0 - \alpha\frac{1}{m}\sum\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\theta_0 \coloneqq \theta_0 - \alpha\frac{1}{m}\sum\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)}$$

</maths>

for just now, at least…

How to choose learning rate α

- Cost function $J(\boldsymbol{\theta})$ should decrease after each iteration
  - Number of iterations till convergence can vary a lot
    - Plot cost function against number of iterations
    - Or: Automatic convergence test – declare convergence if cost function decreases by less than 10^{-3} in one iteration
  - If it doesn't converge
    - Use smaller α
    - Most common cause for increasing cost function is α being too large
  - Try some values for α, e.g. 0.001, 0.01, 0.1, 1,… and plot $J(\boldsymbol{\theta})$ against number of iterations
    - Find an α which is too large and one which is too small

Feature Scaling

- E.g. 'House prices'
  - We had $n = 1$, x corresponding to the size of the house
  - May want to add more features/variables to predict price better

Size $(m^2)/x_1$	No. bedrooms$/x_2$	Age of house (yrs)$/x_3$
70	2	10
110	5	54
35	1	2
235	6	107
76	3	34

- When you have more than one feature θ $(n > 1)$
  - Ensure all features are on a similar scale
    - Faster convergence

Normal Equation

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

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}_{(n+1)\times 1} \qquad X = \begin{bmatrix} --- & (x^{(1)})^T & --- \\ --- & (x^{(2)})^T & --- \\ & \vdots & \\ --- & (x^{(m)})^T & --- \end{bmatrix}_{m\times(n+1)} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{m\times 1}$$

Normal Equation	Gradient Descent
Computes parameters analytically	Many iterations
No need to choose learning rate α	Need to choose α
Slow for large n,cost $\approx \mathcal{O}(n^3)$	Works well for large n
Doesn't work for more sophisticated algorithms,	
e.g. Logistic Regression	

Multivariable Linear Regression

- For the extended house price example, $n = 3$:

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}_{4\times1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}_{4\times1} \qquad h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

- For general n, for convenience, we define $x_0 = 1$:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}_{(n+1)\times1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}_{(n+1)\times1} \qquad h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

- Treating x and θ as $(n+1)\times1$ matrices (as shown above), we can write:

$$h_\theta(x) = \underbrace{\theta^T}_{1\times(n+1)} \underbrace{x}_{(n+1)\times1}$$

Annotations in green show the size and shape of the object

Polynomial Regression

- To get a better model, you can create new features
  - e.g., a non-linear model, or higher order polynomials
    - For house price example, something that scales with x_1^2 (quadratic model)
    - Or $$h_\Theta(x) = \Theta_0 + \Theta_1 x + \Theta_2 x^2 + \Theta_3 x^3 \ldots$$
    - Then feature scaling becomes even more important