INTRODUCTION

→ What is Machine Learning ?

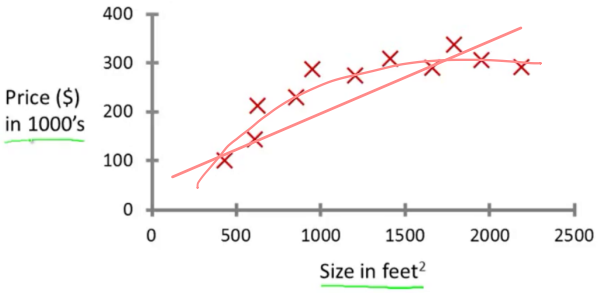
Arthur Samuel (1959) → field of study that gives computers the ability to learn without being explicitly programmed. (checkers game)

Tom Mitchell (1998) → **well posed learning problem.**
A computer program is said to learn from experience E with respect to some task T and performance measure P, if its performance on T, as measured by P, improves with E.

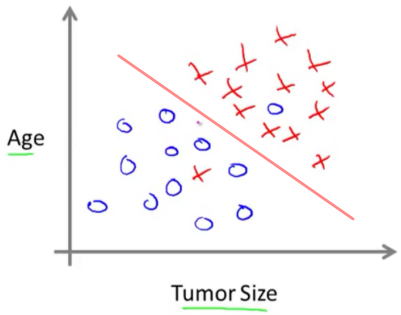
Machine Learning Algorithms:
- **Supervised Learning** → we are going to teach the computer
- **Unsupervised Learning** → the computer learns by itself
- **Others** → reinforcement learning, recommender systems

→ Supervised Learning

'Right answers' given.
We are given a data set and already know what our correct output should look like. → there's a relationship between input and output.



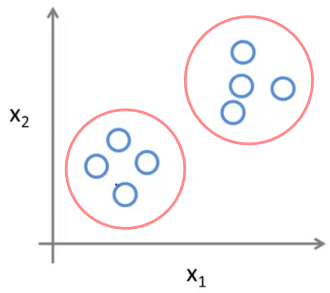
REGRESSION
Continuous valued output (ex: price)



CLASSIFICATION
Discrete valued output (ex: 0 or 1 (or 2, or 3...)

→ Unsupervised learning

Unsupervised learning allows us to approch problems with **little or no idea what our results should be.**
We can derive structure from data where we don't necessarily know the effect of the variables.



CLUSTERING

We can derive structure by clustering the data based on the relationships among the variables in the data.

Applications:

- Organise computing clusters
- Social network analysis
- Market segmentation
- Astronomical data analysis

Cocktail Party Problem:

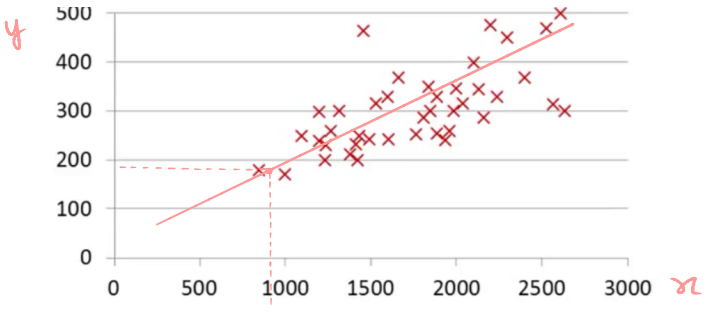
Focus on a specific human voice while filtering out other voices or background noise.

```
[W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

LINEAR REGRESSION WITH ONE VARIABLE

→ Model Representation

Supervised learning regression problem:



if I have a house of size x I should expect to sell it for price y.

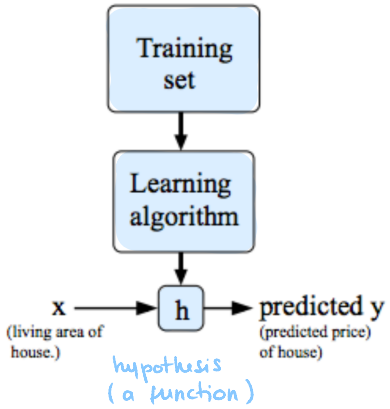
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

$m = 47$

Notation:

- m → number of training examples
- x's → input variable / features
- y's → output variable / "target" variable
- (x,y) → one single training example
- $(x^{(i)},y^{(i)})$ → i^{th} training example

Process:



→ h is a function that maps from x's to y's

→ how do we represent h?

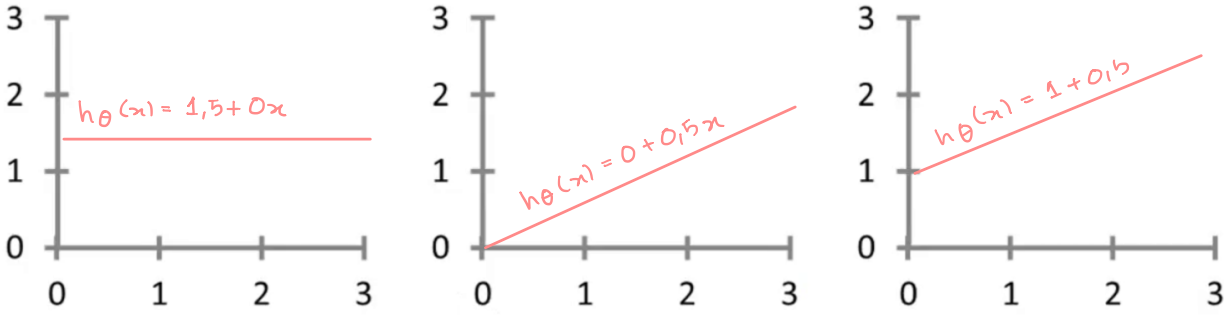
$h_\theta(x) = \theta_0 + \theta_1 x$

shorthand: $h(x)$

→ Cost Function

Parameters

How do we choose θ_0 and θ_1?



$\theta_0 = 1.5$, $\theta_1 = 0$ | $\theta_0 = 0$, $\theta_1 = 0.5$ | $\theta_0 = 1$, $\theta_1 = 0.5$

We choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples!

Cost Function (= squared error function)

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

$h_\theta(x^{(i)})$ → predicted price of a house

$y^{(i)}$ → price the house sold for

$\frac{1}{2m}$ → average

$\sum_{i=1}^{m}$ → sum of training examples

We want to minimise the cost function:

$$\min_{\theta_0, \theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h\theta(x^{(i)}) - y^{(i)})^2$$

→ Cost Function: Intuition I

Simplified Intuition

Hypothesis

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

Parameters

θ_0, θ_1 → θ_1

Cost Function

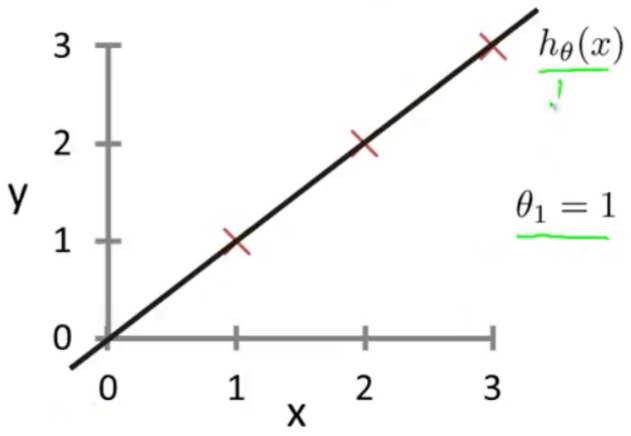
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \rightarrow J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Goal

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

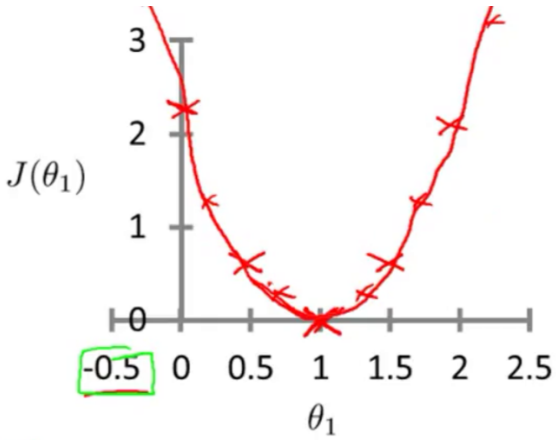
Hypothesis Function $h_\theta(x)$

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



Cost Function $J(\theta_1)$

(function of parameter θ_1)



Calculating $J(\theta_1)$:

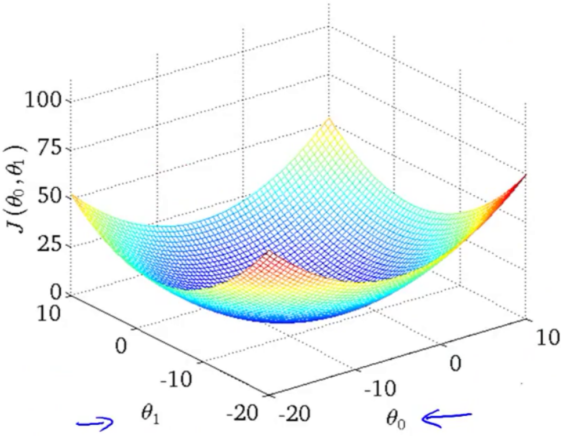
- For $\theta_1 = 0$ → $J(\theta_1) = \frac{1}{2m}(1^2 + 2^2 + 3^2) = \frac{1}{6} \times 14 \simeq 2{,}3$
- For $\theta_1 = 0{,}5$ → $J(\theta_1) = \frac{1}{2m}[(0{,}5-1)^2 + (1-2)^2 + (3{,}5-1{,}5)^2] = \frac{1}{6} \times 3{,}5 = 0{,}58$
- For $\theta_1 = 1$ → $J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\theta_1(x^{(i)}) - y^{(i)})^2 = \frac{1}{2}(0^2 + 0^2 + 0^2) = 0$

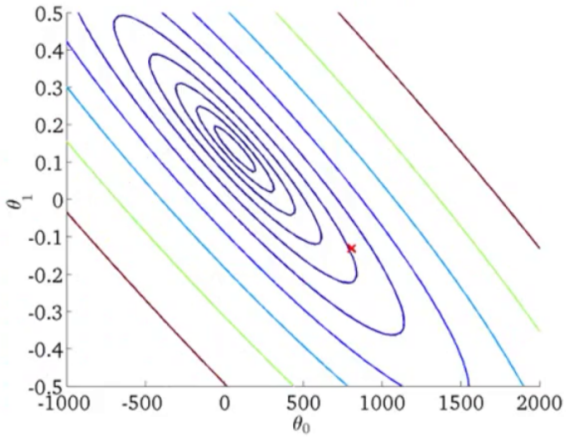
We want to choose the value of θ_1 that minimises $J(\theta_1)$, so $\theta_1 = 1$

→ Cost Function: Intuition II

Cost Function $J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)





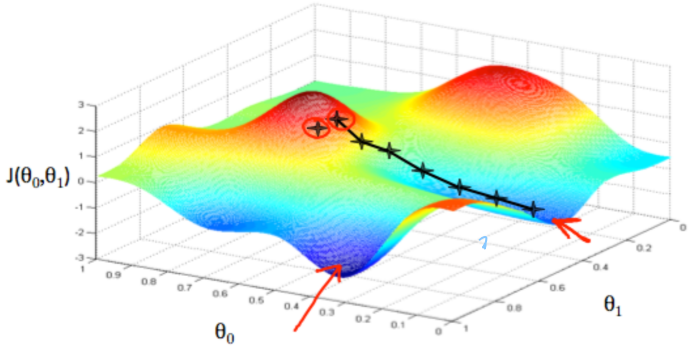
3D SURFACE ⟷ CONTOUR PLOT

→ Gradient Descent

We have some function $J(\theta_0, \theta_1)$ that we want to minimise.

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0 and θ_1 to reduce $J(\theta_0, \theta_1)$ until we reach the minimum.



Gradient Descent Algorithm:

Repeat until convergence {

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

→ derivative term

, for $j=0$ (θ_0) and $j=1$ (θ_1)

}

α: learning rate: controls how big a step we take downhill with gradient descent

We must simultaneously update θ_0 and θ_1

Simultaneous update

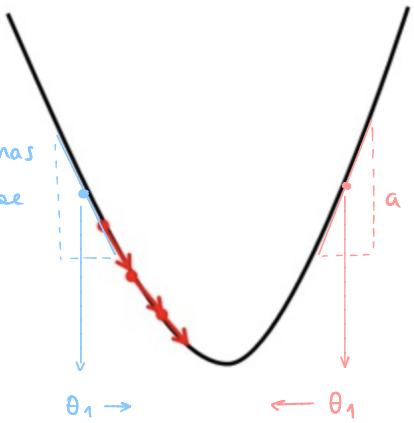
$$\text{temp } 0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$\text{temp } 1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := \text{temp } 0$$

$$\theta_1 := \text{temp } 1$$

→ Gradient Descent : Intuition



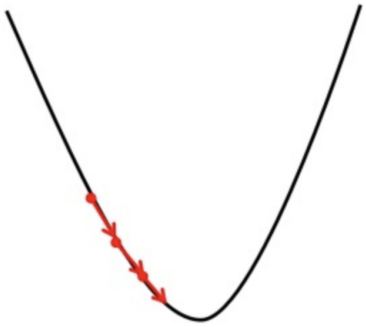
$$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)}$$

the derivative gives us the slope of the tangent to the function on the point.

$\theta_1 \rightarrow \theta_1 - \alpha$. negative number (move to the right)

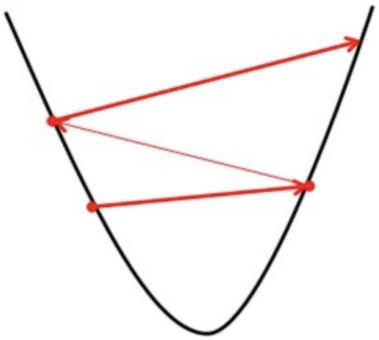
$\theta_1 \rightarrow \theta_1 - \alpha$. positive number (move to the left)

Learning Rate (α)



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

if the α is too small, gradient descent can be very slow.

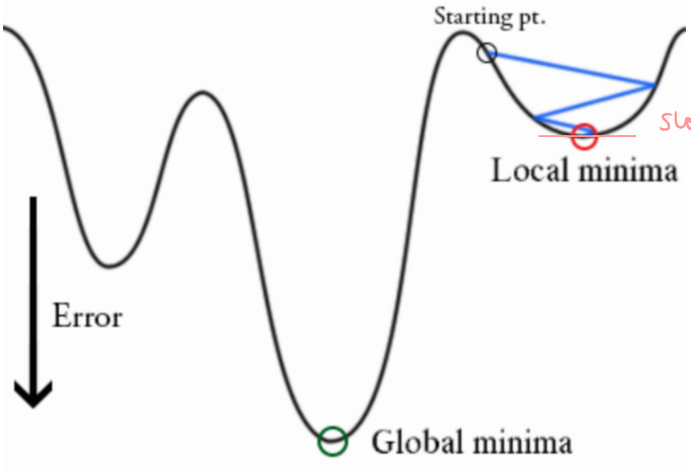


$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

if the α is too large, gradient descent can overshoot the minimum.

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

Starting with θ_1 at a local minimum:



$$\theta_1 := \theta_1 - \alpha \boxed{\frac{\partial}{\partial \theta_1} J(\theta_1)} \rightarrow = 0$$

$$\theta_1 := \theta_1$$

If we start at a local minimum, one step at the gradient descent does absolutely nothing!

→ Gradient Descent for Linear Regression:

Gradient Descent Algorithm:

Repeat until convergence {

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

}

apply here →

Linear Regression Model

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

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

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1) \rightarrow \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) =$$

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

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

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

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

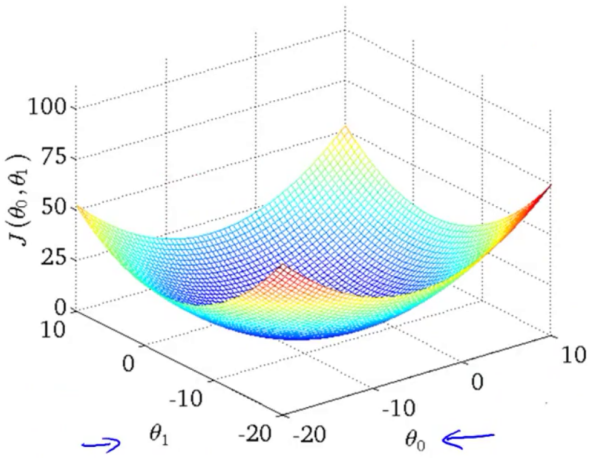
So, the linear regression algorithm is:

Repeat until convergence {

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

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

For linear regression, the cost function will always be bowl-shaped (convex function)



this function does not have local optima except for one global optima. So it will always converge to the global optima.

'Batch' gradient descent:

Batch: each step of gradient descent uses all of the training examples (the entire training set).