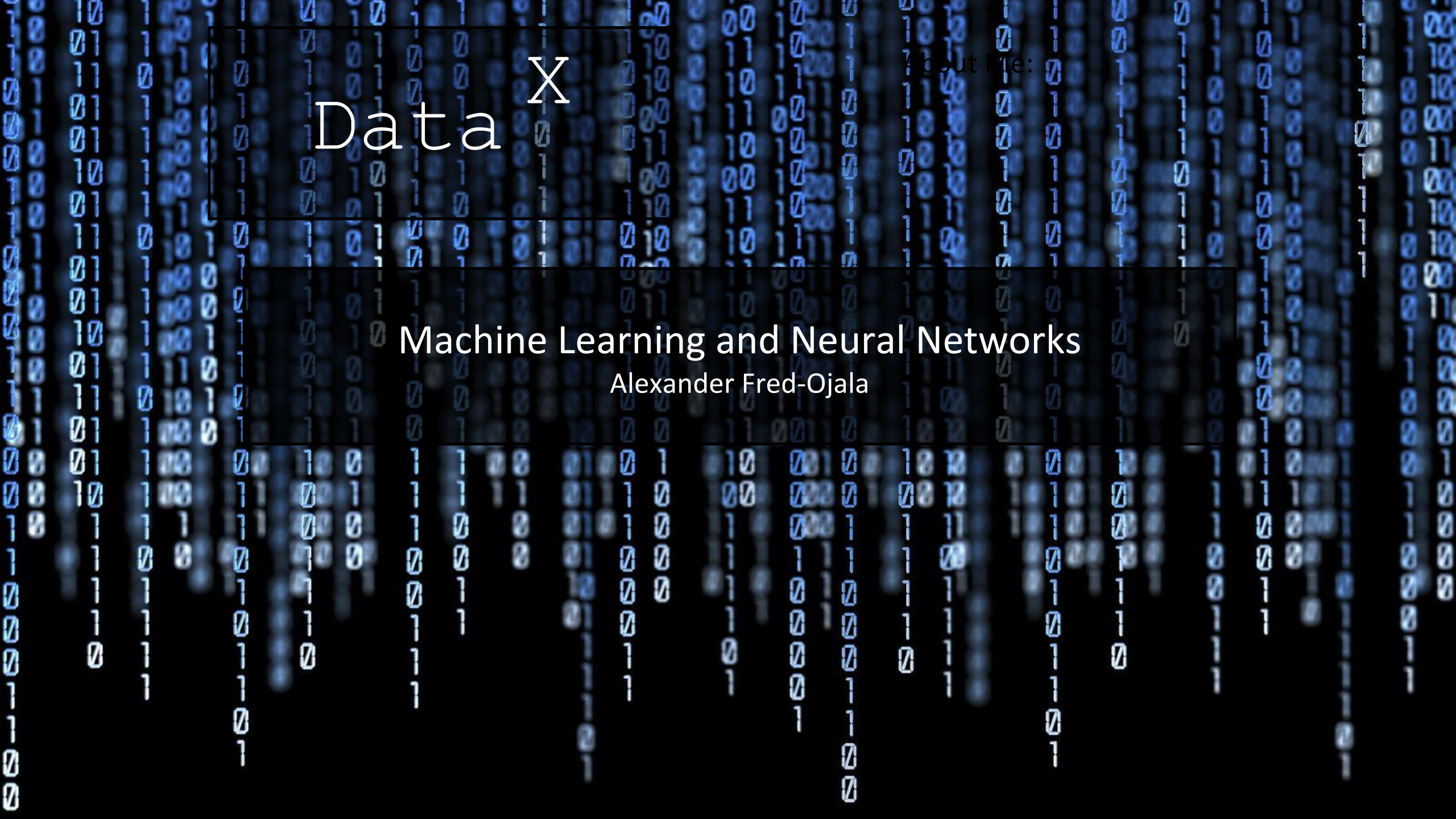



About Me:

# Machine Learning and Neural Networks

Alexander Fred-Ojala

# Outline

1. Deep Learning Use Cases
2. Linear Regression recap
3. Gradient Descent
4. Neural Nets

# Example Use Cases

# Deep Learning: Example Use Cases

**Self-driving cars**

Learns through experiences and replication.

**Healthcare**

Cancer diagnostics, monitoring, personalised medicine.

**Financial Forecasting**

Time series data, recurrent algos

**Sentiment analysis**

From text and images

**Voice Assistants**

Siri, Google Home, Alexa

**Generating images, videos, music, text**

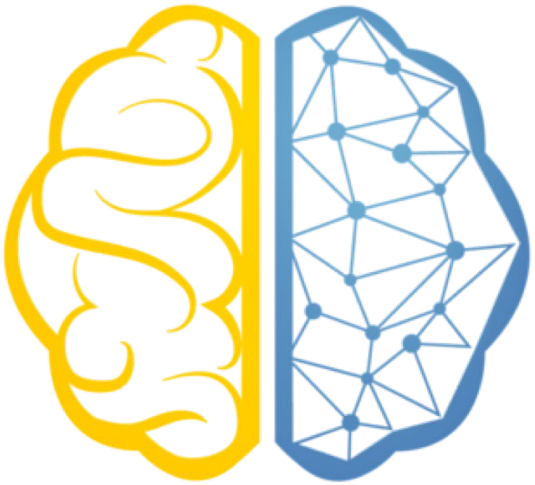
Has become almost indistinguishable from the real

**Machine Translation**

From noise generate images, music composition, novels

**Image Recognition**

Identify people etc



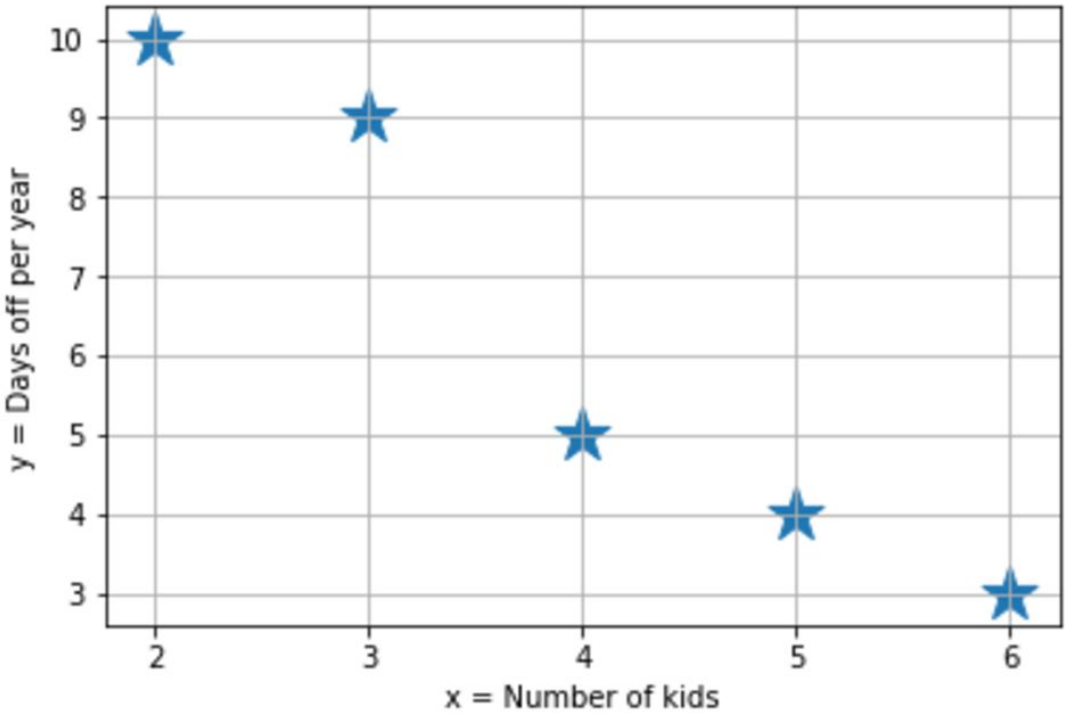
# Recap: Linear Regression

# Recap: Prediction

**Given some data:**

[ (x_1,y_1) , (x_2 , y_2) … (x_m , y_m) ]

| x | y |
|---|---|
| 2 | 10 |
| 4 | 5 |
| 3 | 9 |
| 5 | 4 |
| 6 | 3 |




**Objective:** Be able to predict y given new input x

# Recap: Simple Linear Regression

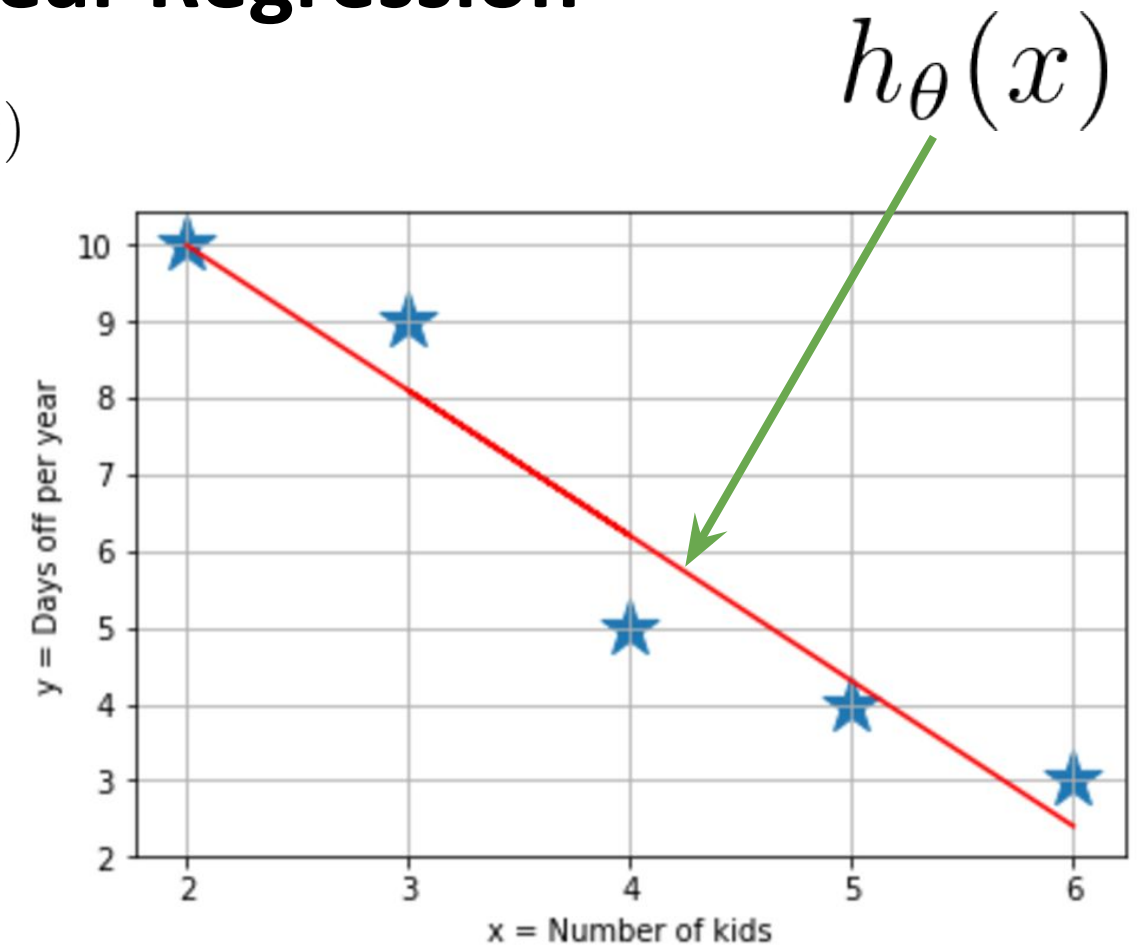
**Simple Linear Regression:** Hypothesis function $h_\theta(x)$

$$\hat{y} = f(x, \theta) = h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

$$x = \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$$ **x is given input**

**Objective:** fit the best linear function to the training data, i.e. find optimal parameters θ

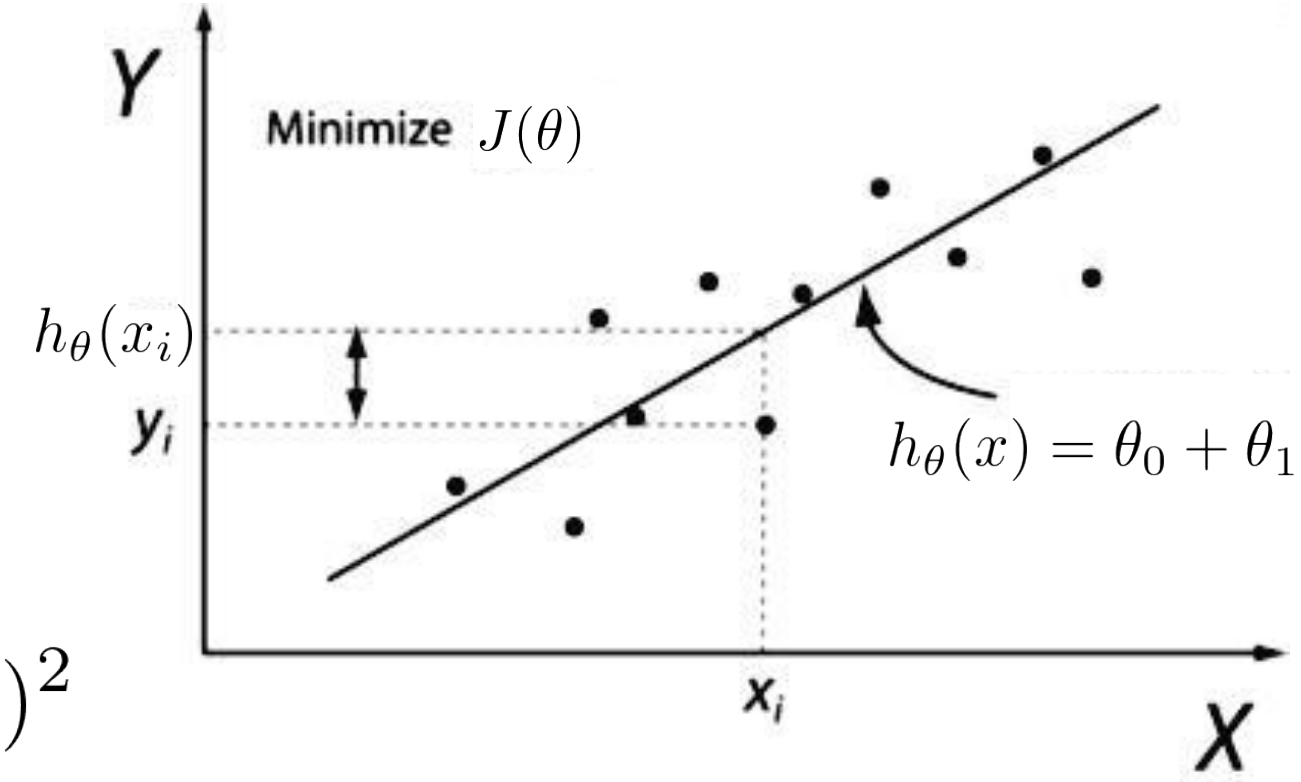
$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$



# Cost function, Mean Squared Error (MSE)

**Simple Linear Regression**

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1$$




**Cost function:**

Measures how good our predictions are (MSE)

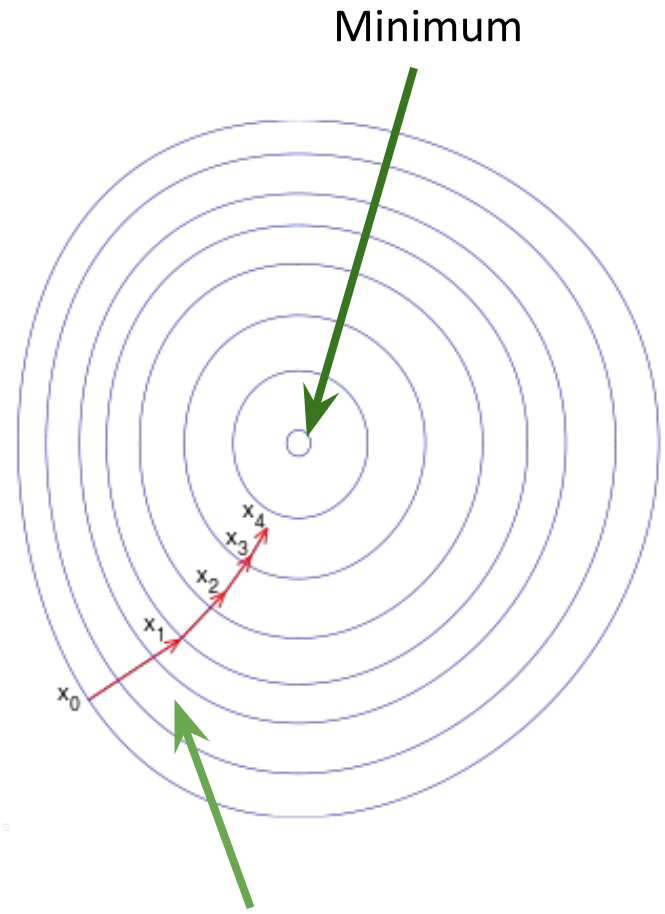
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

# Gradient Descent

# Introducing Gradient Descent

*Gradient descent is a an iterative optimization algorithm for finding the minimum of a function.*

*To reach minima one takes steps proportional to the negative gradient (or approximate gradient) of the function at the current point.*




Step sizes & neg. gradient directions

# Introducing Gradient Descent

Alternative way of minimizing the cost function:

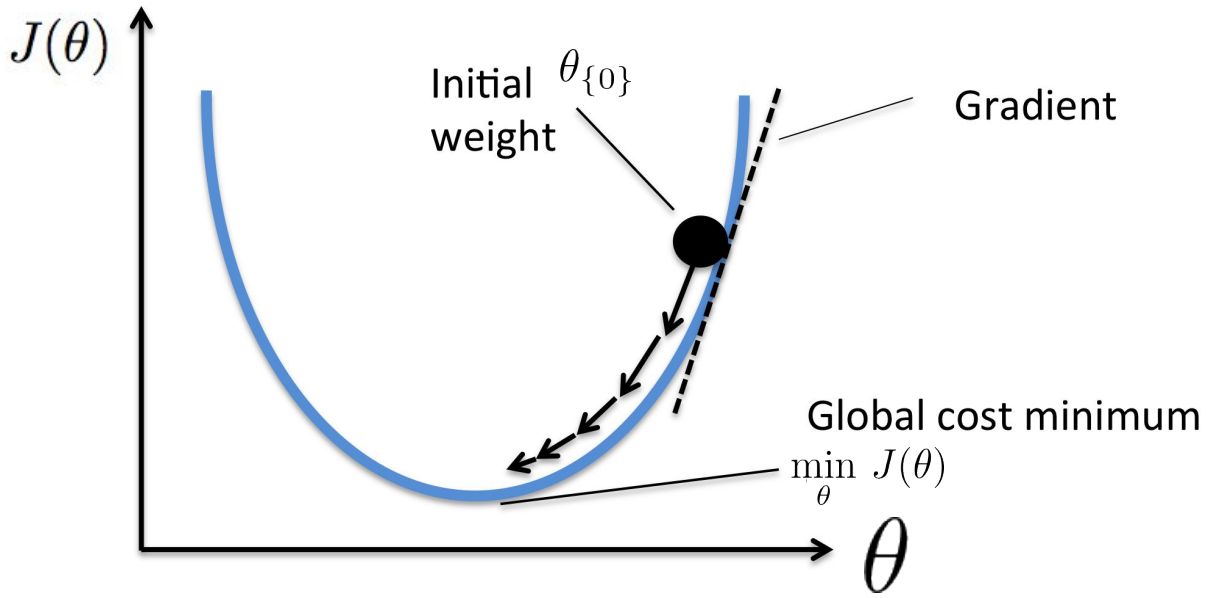
$$J(\theta) = \frac{1}{m}(X\theta - y)^T(X\theta - y)$$

- ***Will always converge because J(θ) is convex***

- Start with / initialize $\theta_0, \theta_1$ . E.g. $(\theta_0, \theta_1) = (0, 0)$
- Keep changing $\theta_0, \theta_1$ to reduce $J(\theta_0, \theta_1)$,

## Illustration of Gradient Descent

for one parameter θ




Source: https://sebastianraschka.com

# Gradient Descent Algorithm: Linear Regression

1. **Calculate the partial derivative** $\frac{\partial}{\partial \theta_j} J(\theta)$ for all j
2. Form the **update rule** for every parameter:

$$\theta_{j,iter+1} := \theta_{j,iter} - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_{j,iter} - \alpha/m \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

3. **Choose a step size/ *learning rate*** $\alpha$ *(*often between 10^-6 and 10^2 -- not too big, then divergence).
4. **Update all the parameters** $\theta_0 \ldots \theta_n$ **by feeding in all training samples in X** ("batch" Gradient descent)
5. Stop when the error has converged.

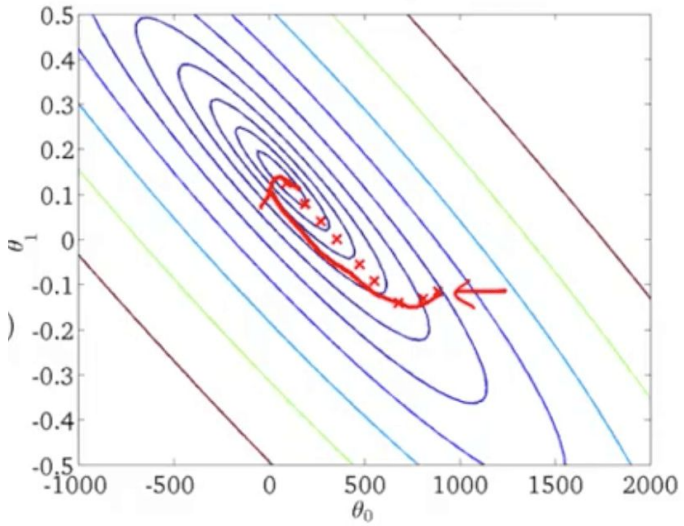
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}



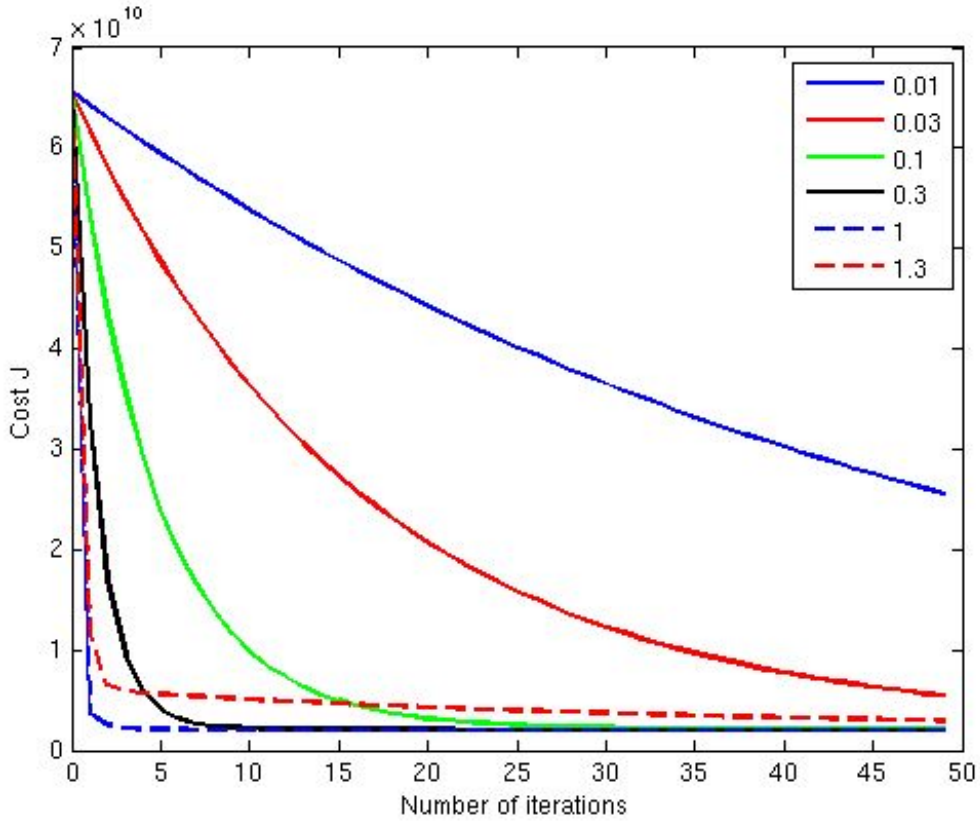

Source:Ritchie Ng

# Gradient Descent Tips

## Monitor convergence

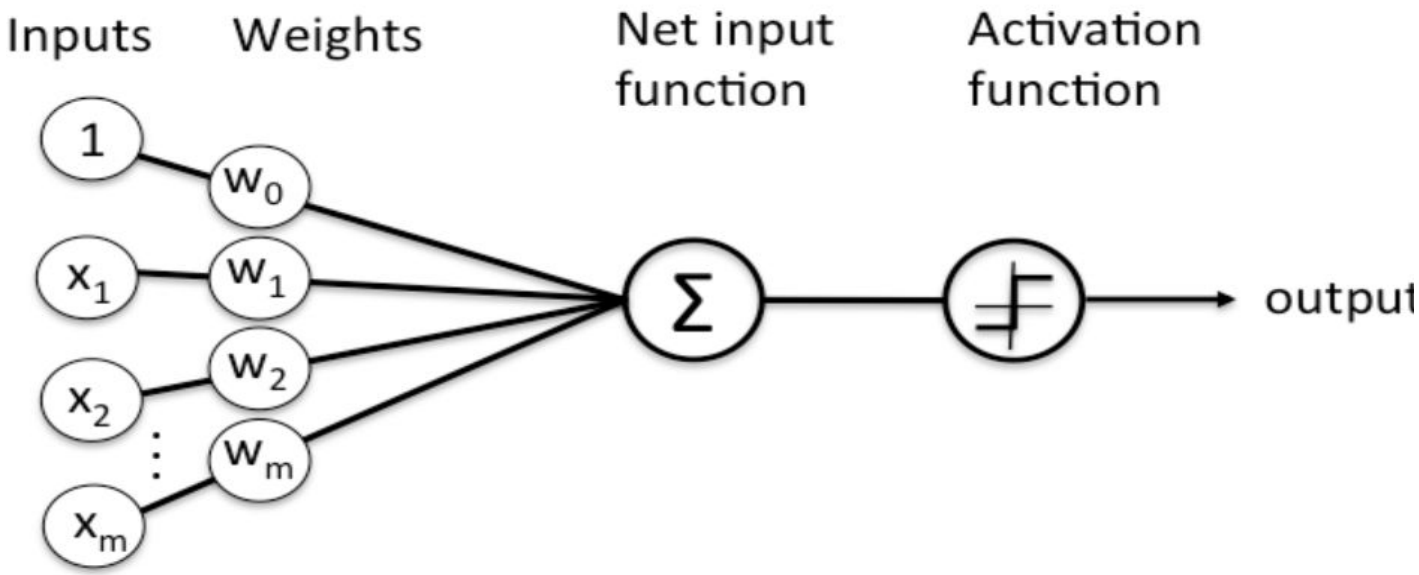
**Plot the value of the error function $J(\theta)$ at every iteration.**

*Check that the error becomes smaller. Plot for different learning rates to find the best one.*



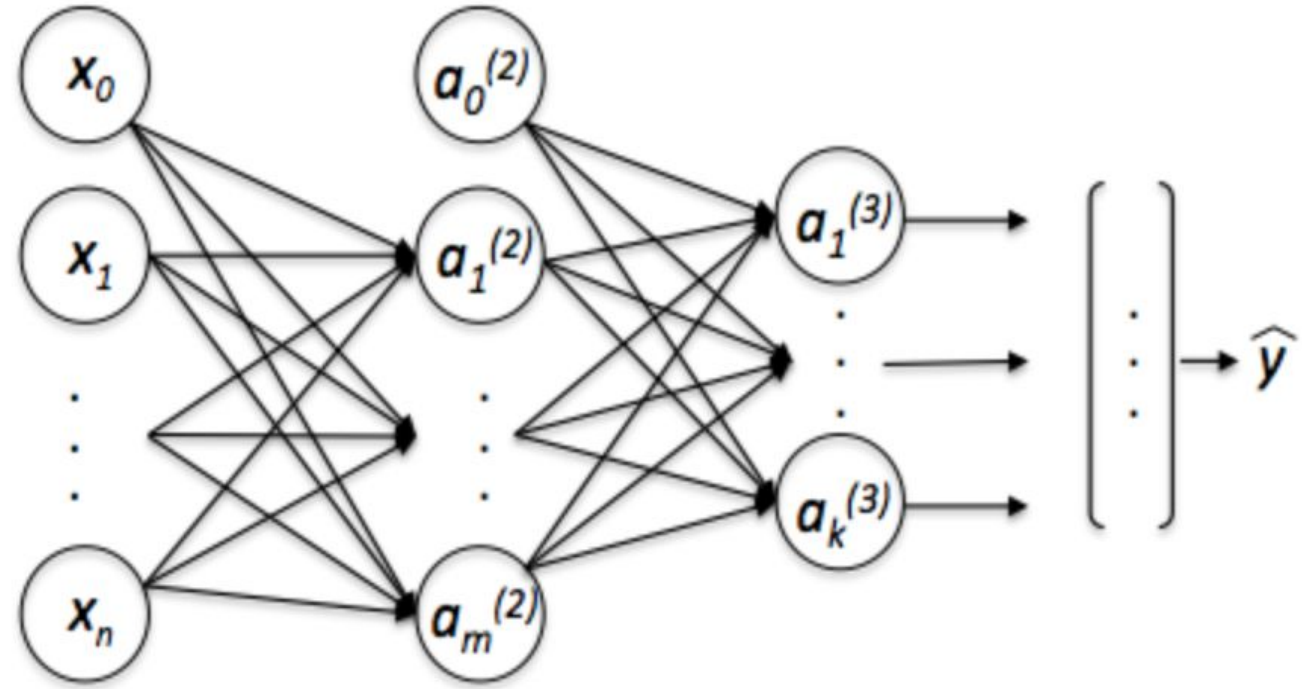
# Neural Networks

# Basic Perceptron:



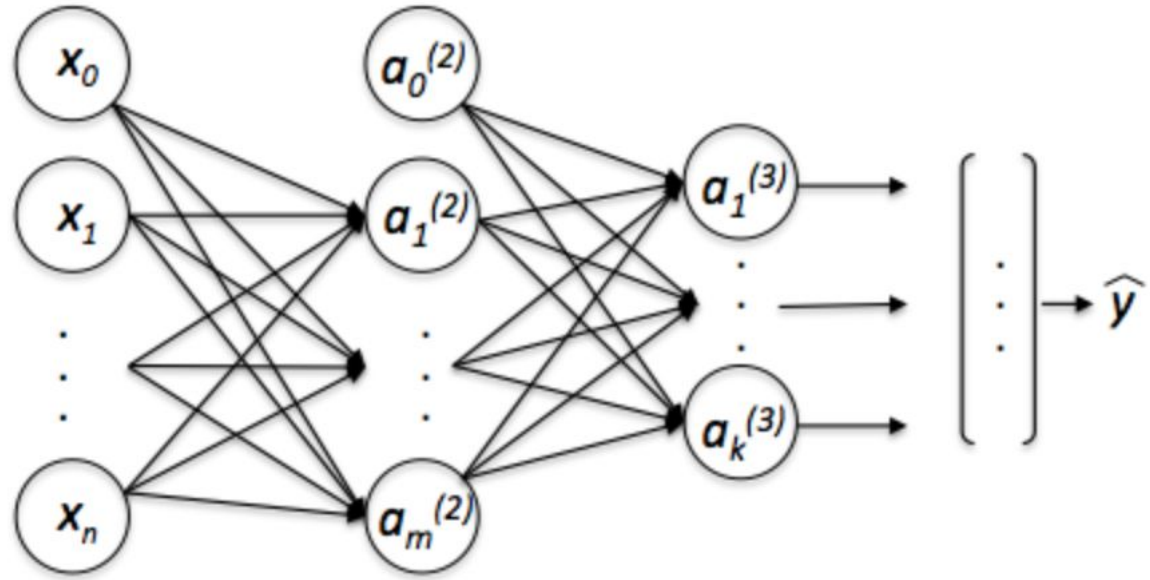

**Schematic of Rosenblatt's perceptron.**

# Neural Network

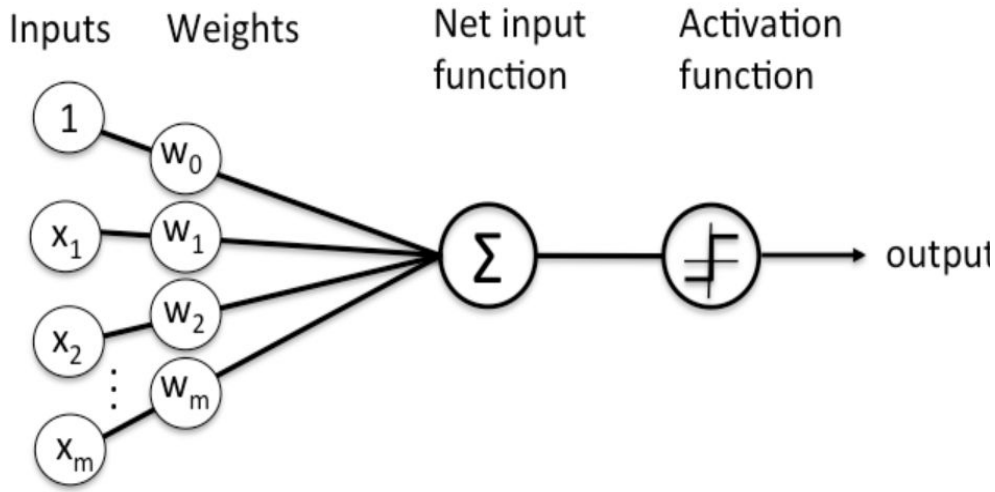



**Schematic of a multi-layer perceptron.**

# Compare the Complexity

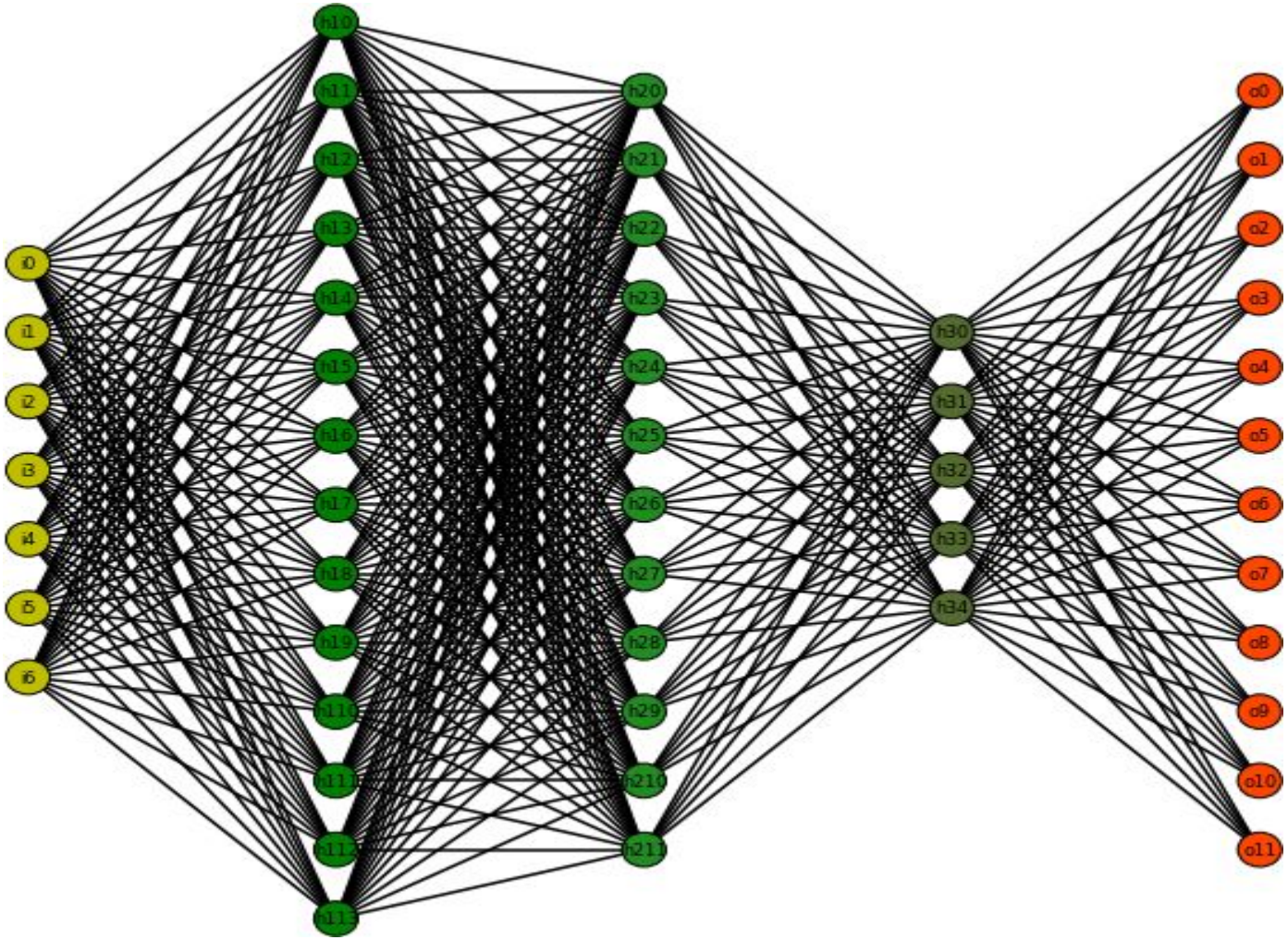



**Schematic of a multi-layer perceptron.**




**Schematic of Rosenblatt's perceptron.**

# What Dense Neural Networks actually look like



# How it works?

How does the network know the strength of connections between neurons?

It learns them!

1. We start with random weights.
2. Input a set of features.
3. Calculate the output.
4. Calculate the  Loss  wrt to actual output value in the data.
5. Find the gradient of the Cost function.
6. The gradients are pushed back into the network and used for adjusting the weights - **Backpropagation**
7. The whole process is repeated again till we train a model of acceptable performance.

# Implementation Example

# Recap: Multiple Linear Regression

**Multiple Linear Regression:** $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = \theta^T X$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ . \\ . \\ . \\ \theta_n \end{bmatrix}$$ is the parameter vector and

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & . & . & . & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & . & . & . & x_n^{(2)} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_0^{(m)} & x_1^{(m)} & . & . & . & x_n^{(m)} \end{bmatrix}$$ is the feature vector and

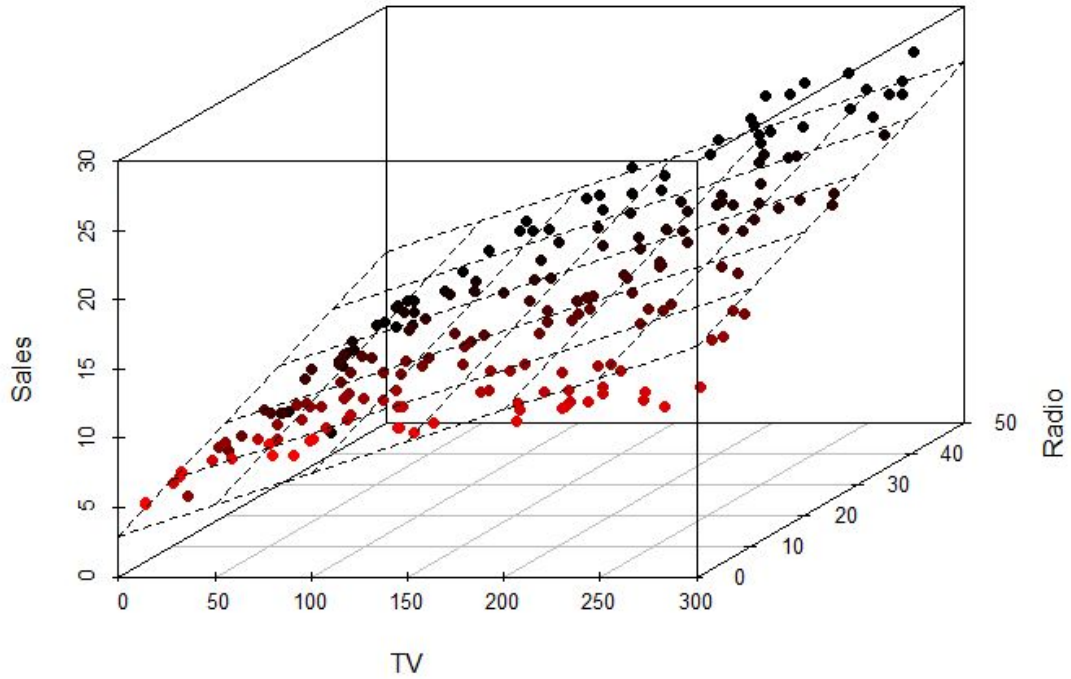
$$h_\theta(X) = \begin{bmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ . \\ . \\ h_\theta(x^{(m)}) \end{bmatrix}$$ is the hypotheses vector

**Example of multiple linear regression** (2 features)

x_1 = TV advertising

x_2 = Radio advertising

y = Sales




Source: 3.bp.blogspot.com/

# Minimize cost function

**Optimal model parameters minimizes J(θ)**

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

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

$$\min_\theta J(\theta) \implies \nabla_\theta J(\theta) = 0$$

$$\frac{\partial J}{\partial \theta} = 2X^T X\theta - 2X^T y = 0$$

Minimize by taking the derivative w.r.t. θ = 0

## Normal equation for Linear Regression

Closed form, analytical solution.

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

### Pros:

- Finds optimum in one calculation
- Really quick for small data sets

### Cons:

- $O(n^2 m)$ complexity, to calculate $(X^T X)$ and. O(n^3) to calculate $(X^T X)^{-1}$
- $(X^T X)^{-1}$ might not be invertible, ie singular (can be solved by using the pseduoinverse)

# Gradient Descent

## Pros

- **Will always converge** if learning rate $\alpha$ is chosen correctly
- **Fast** (time complexity is $O(m)$)
- Supports out of sample training (stochastic / mini batch G.D.)

## Cons

- **We have to choose learning rate** and initialize model parameters
- **Often takes many iterations**