

Introduction to Machine Learning I

Linear and Polynomial Regression,
Gradient Descent, Logistic Regression
and Classification

Dimosthenis Karatzas (dimos@cvc.uab.es)

What fish is this?



Sea Bass
(*lubina*)



Sea Bream
(*dorada*)



What fish is this?



Sea Bass (lubina)



Sea Bass (lubina)



Sea Bream (dorada)



Sea Bass (lubina)



Sea Bream (dorada)



Sea Bream (dorada)

How do you do it?



What fish is this?

Can you create a program that automatically classifies fish?

Data



*Extract
features*

$$\begin{bmatrix} 0.8 \\ 1.2 \\ \vdots \\ 0.2 \\ 1.5 \end{bmatrix}$$

*Translate your
knowledge into code*

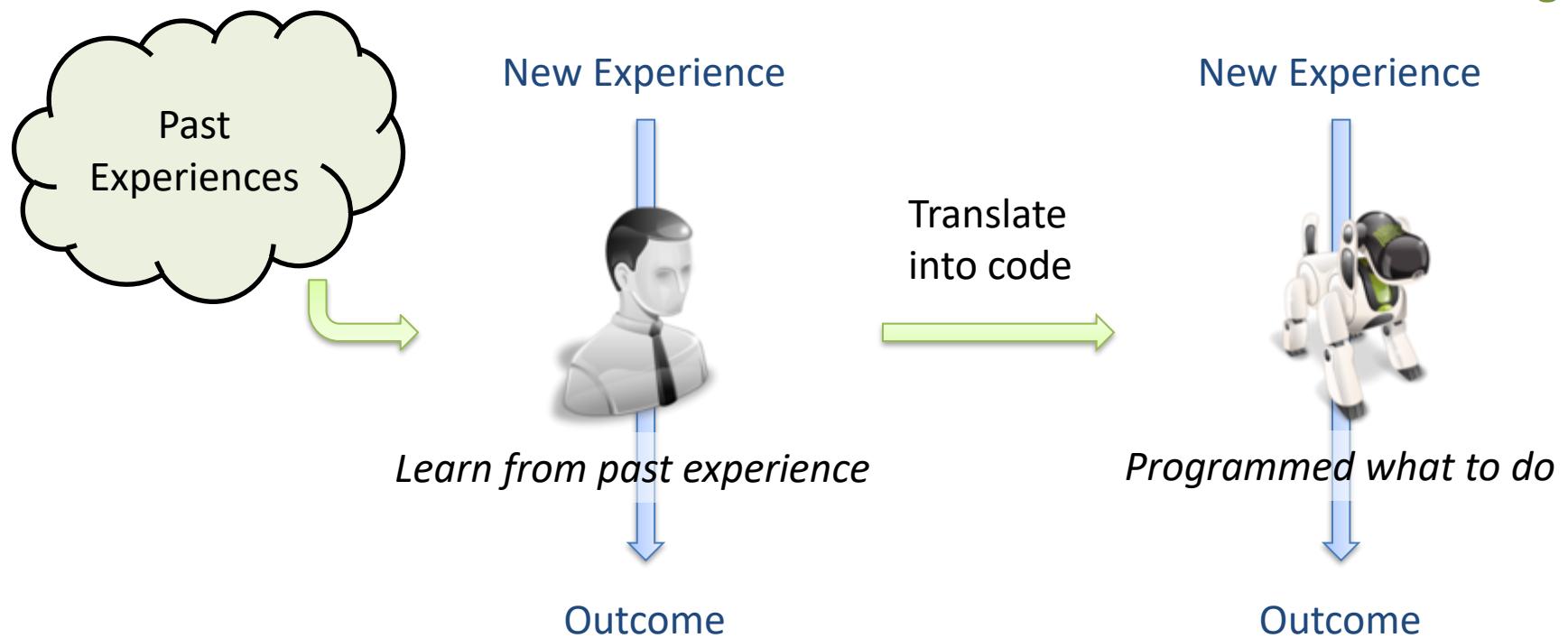
```
if (aspect ratio > 0.7):  
    return "Sea Bass"  
else:  
    return "Sea Bream"
```

{ Sea Bass
Sea Bream }

How to make a machine do intelligent stuff?

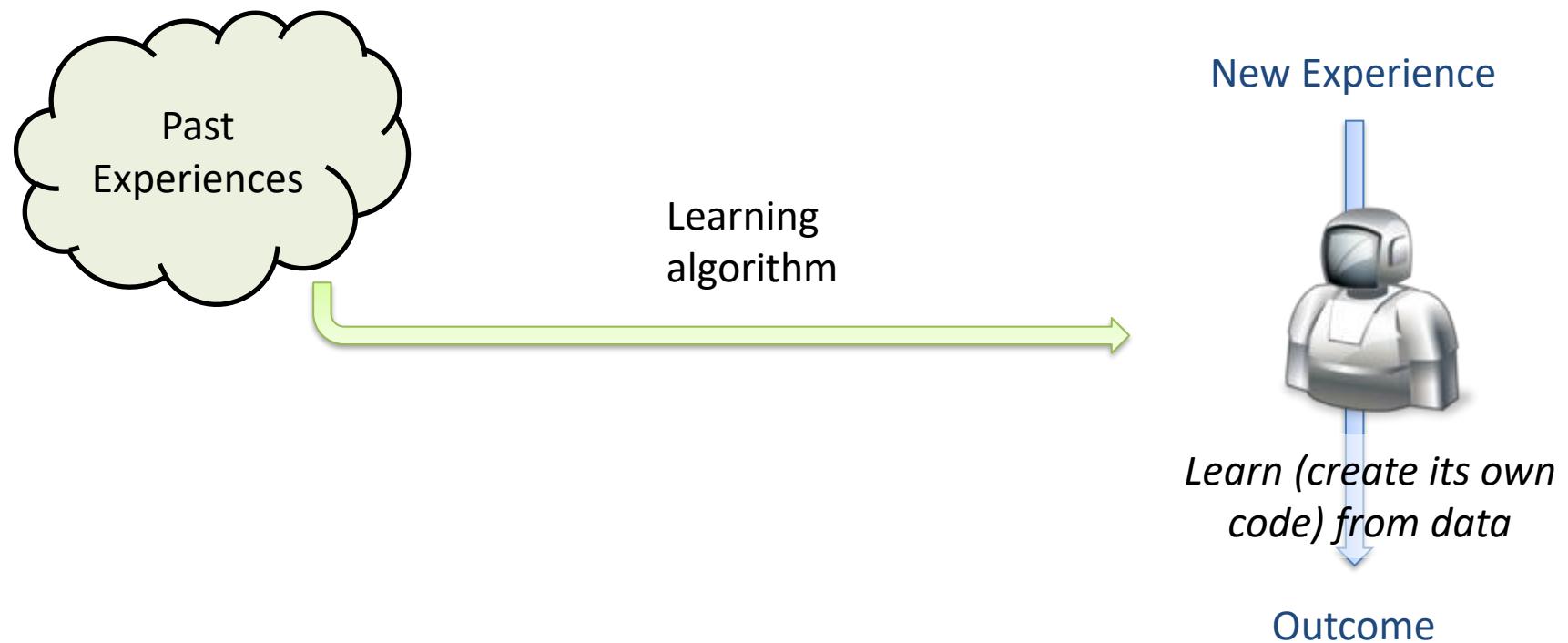
One way to achieve it, is to manually and painstakingly translate our knowledge into code that identifies specific patterns in the data

But, can we get rid of the middleman?

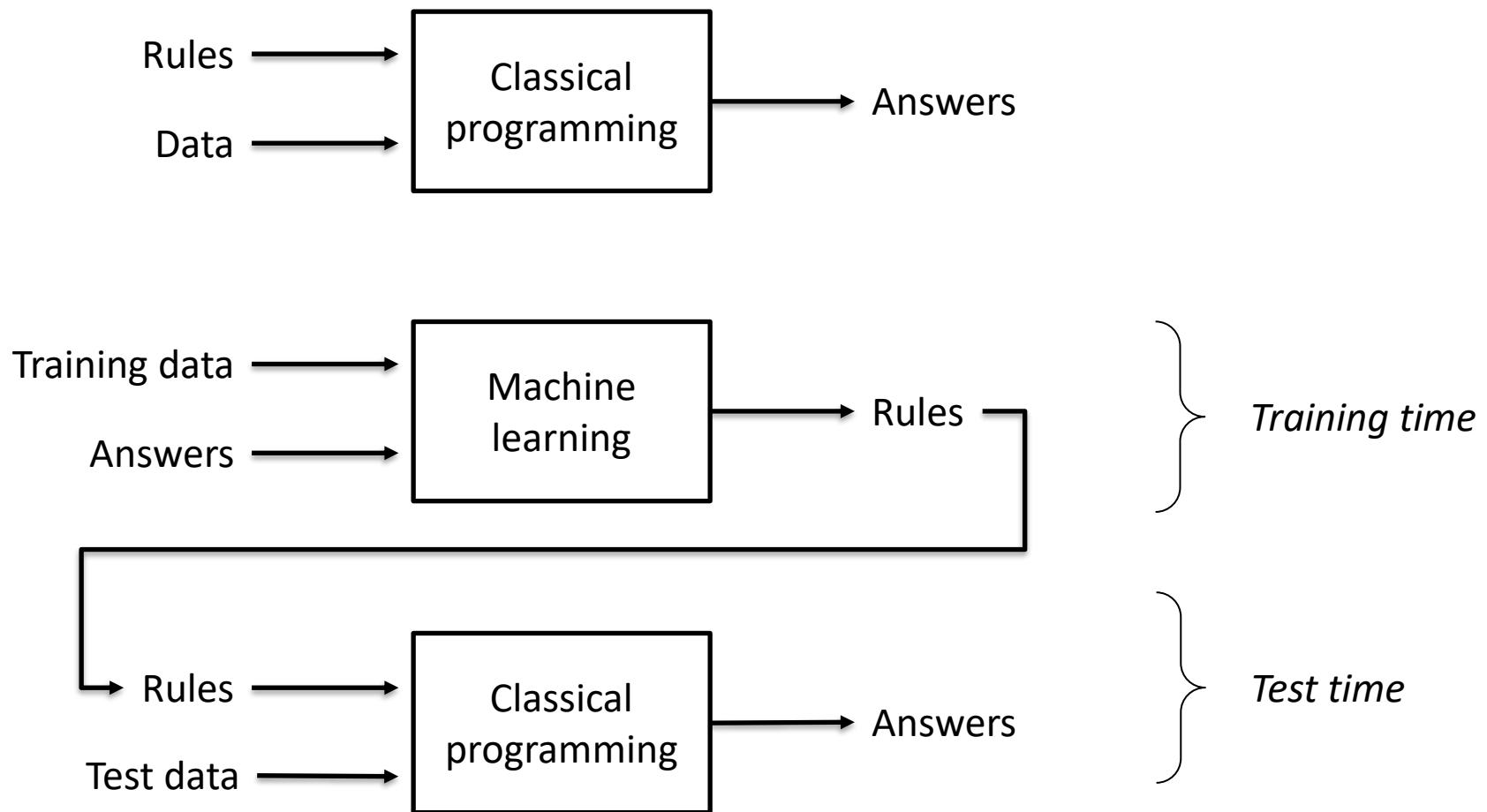


Machine Learning

Machine learning is the process of learning automatically from data the parameters of computational models to recognise useful patterns



Machine learning



What fish is this?

Can you create a program that automatically classifies fish?

Data



*Extract
features*

$$\begin{bmatrix} 0.8 \\ 1.2 \\ \vdots \\ 0.2 \\ 1.5 \end{bmatrix}$$

*Translate your
knowledge into code*

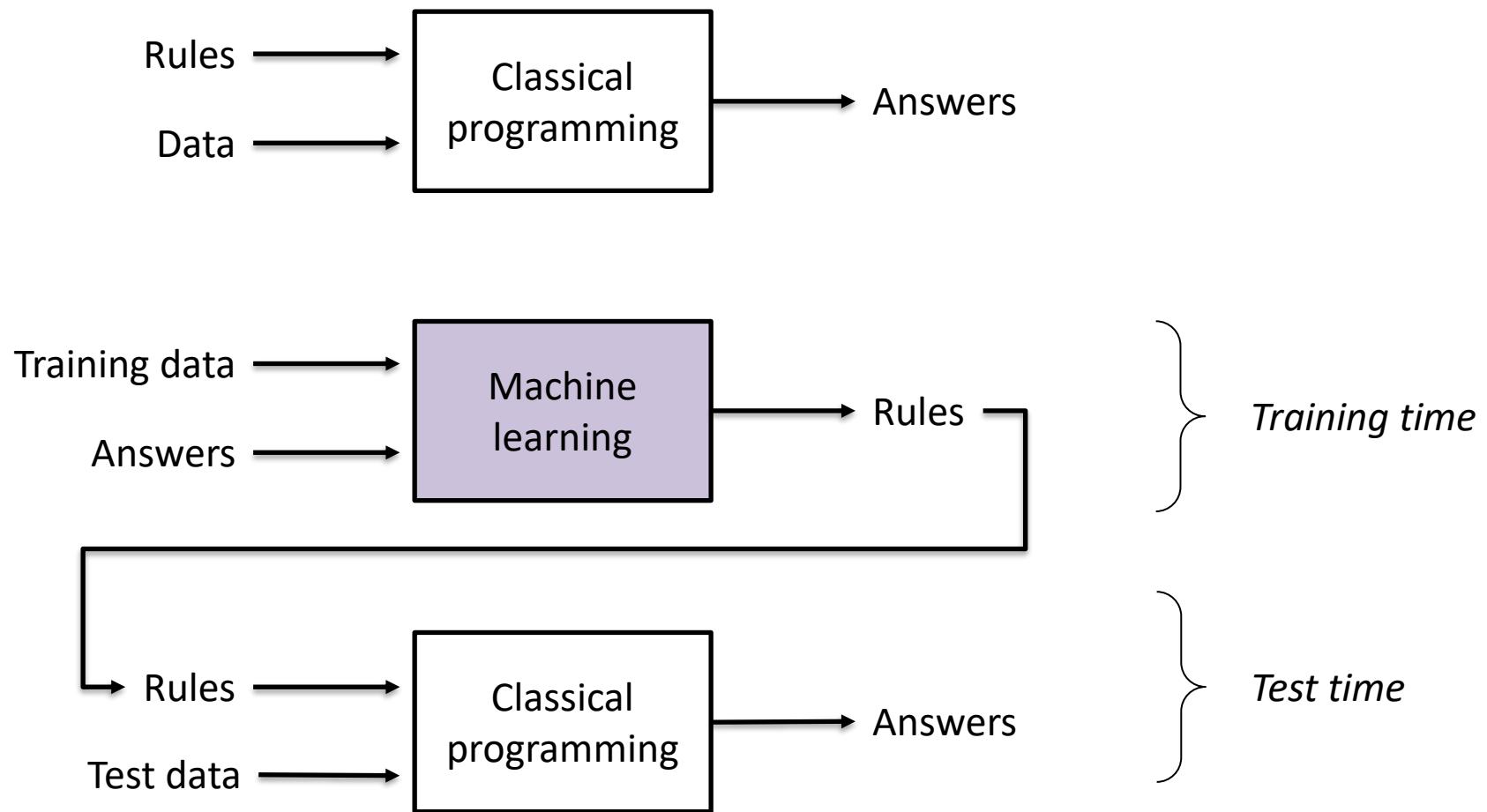
```
if (aspect ratio > 0.7):  
    return "Sea Bass"  
else:  
    return "Sea Bream"
```

{ Sea Bass
Sea Bream }

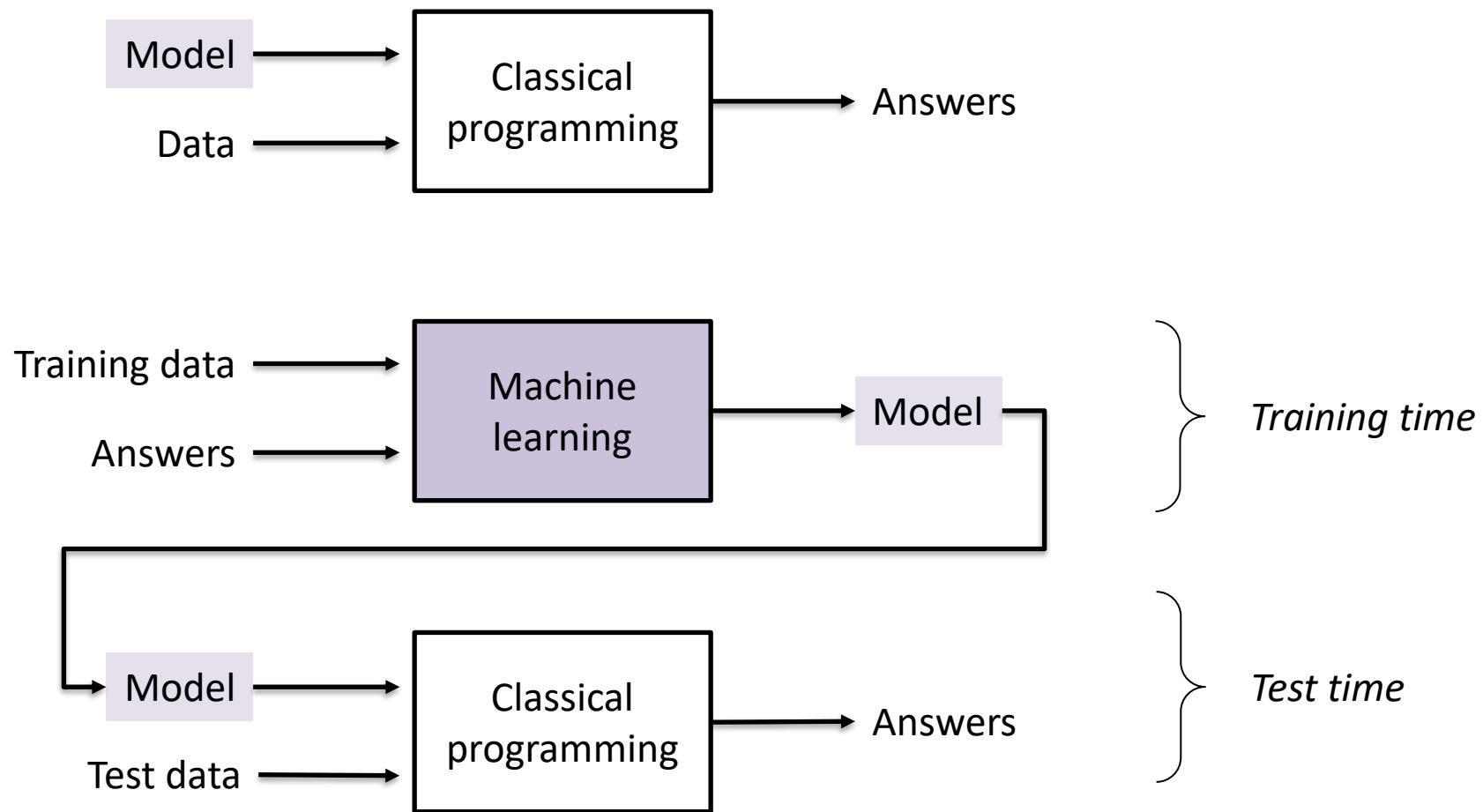
Classical Machine Learning

Deep Learning

Machine learning



Machine learning

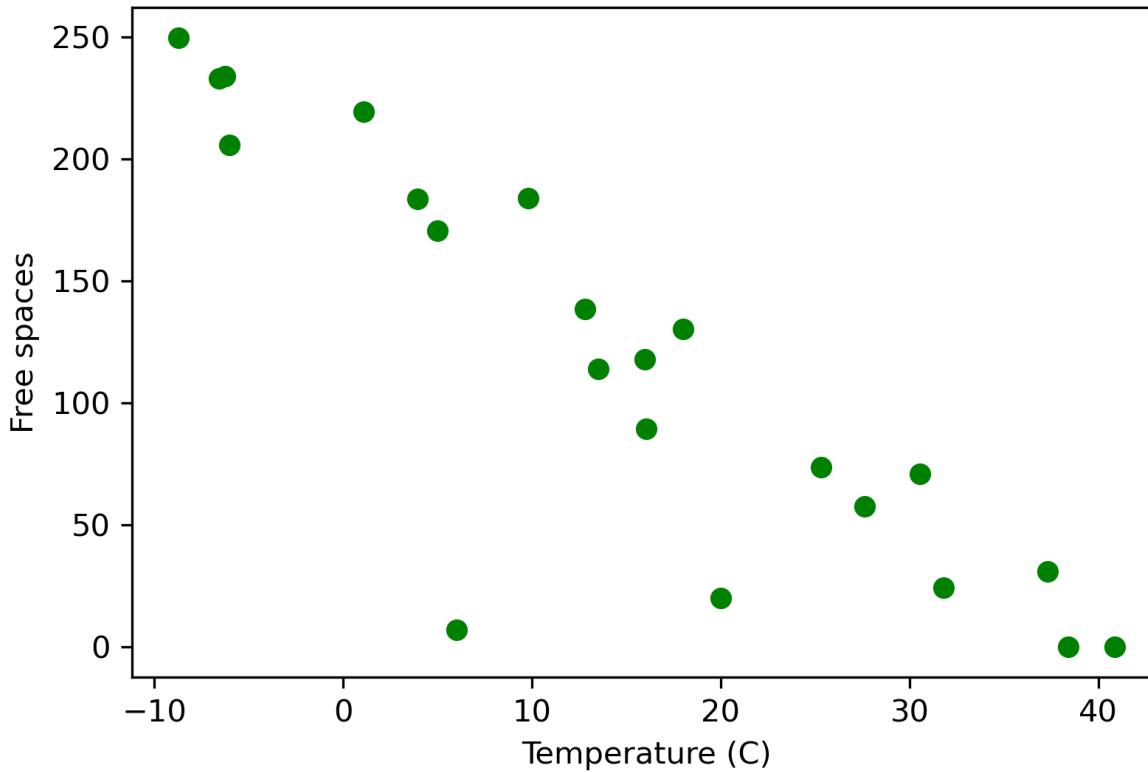


Machine Learning Tribes

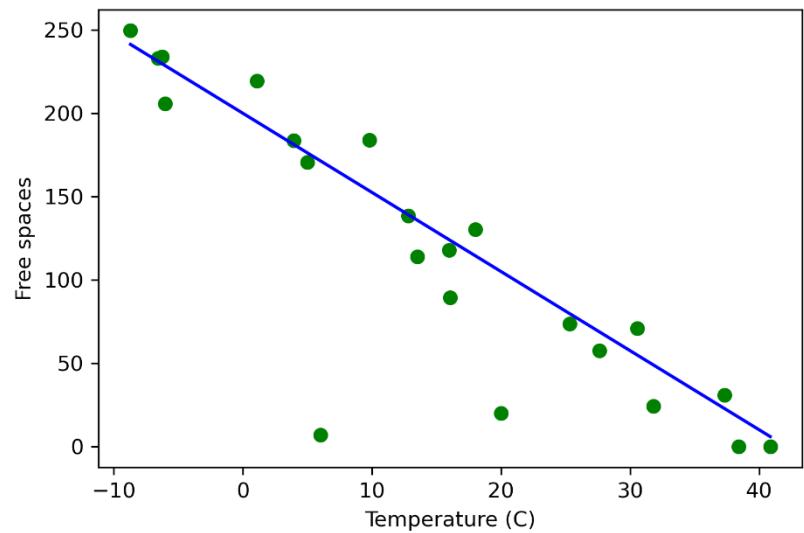
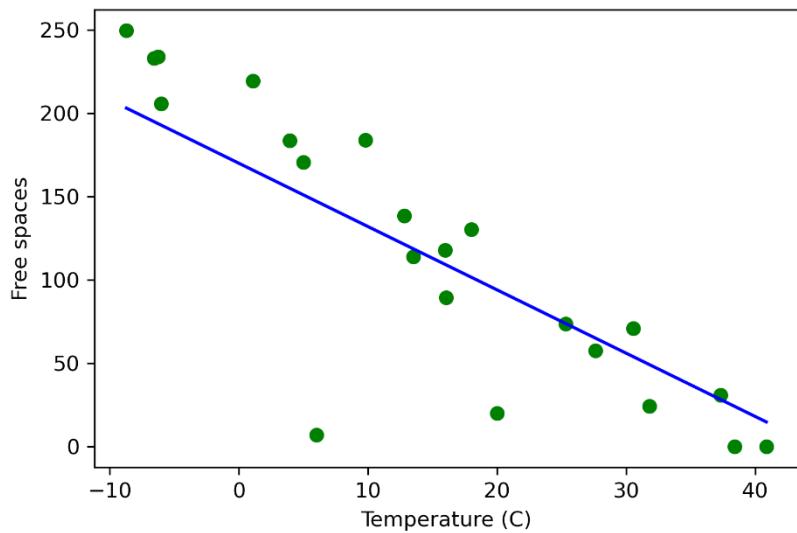
| | Learning algorithm what is in the ML box? | Evaluation how do they measure success? | Representation what do their models look like? |
|---|---|---|--|
| λ | Symbolists Inverse deduction | Accuracy | Logic |
|  | Connectionists Gradient descent | Squared error / cross-entropy loss | Function / Neural Network |
|  | Evolutionaries Genetic search | Fitness | Genetic programs |
| $P()$ | Bayesians Probabilistic inference | Posterior probability | Graphical models |
|  | Analogisers Constrained optimisation | Margin | Support vectors |

Toy Problem: A parking lot

Can you predict how many free spaces there will be in the parking tomorrow by looking at the temperature prediction?



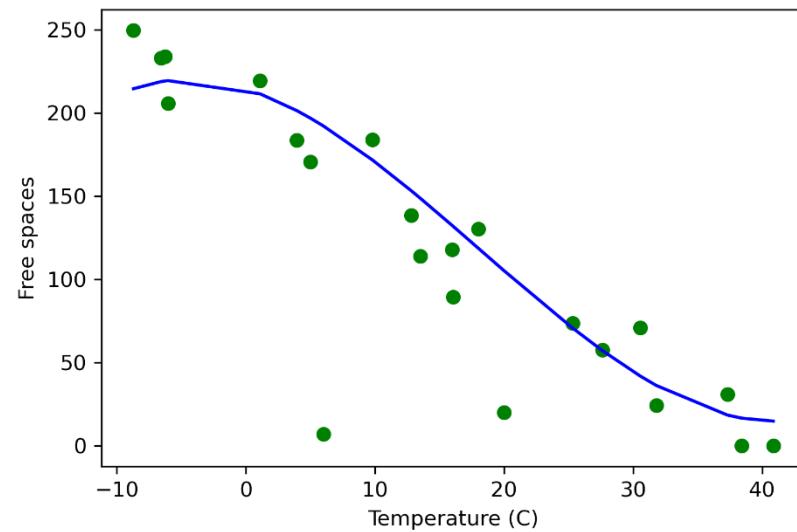
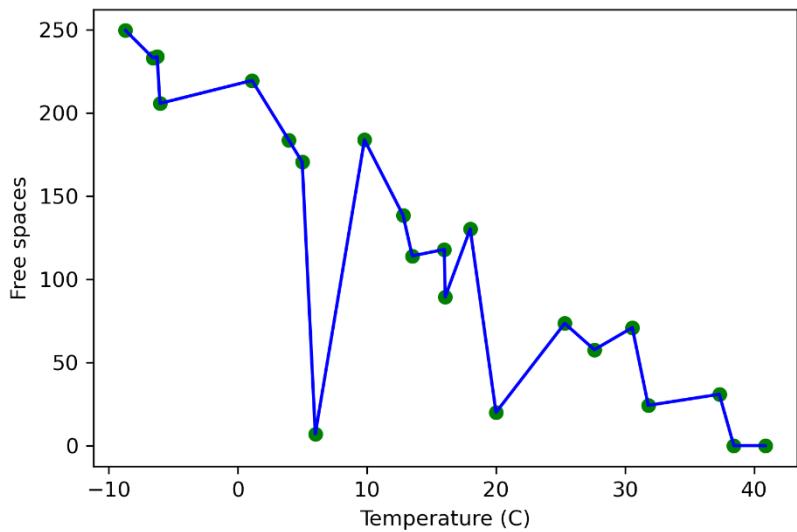
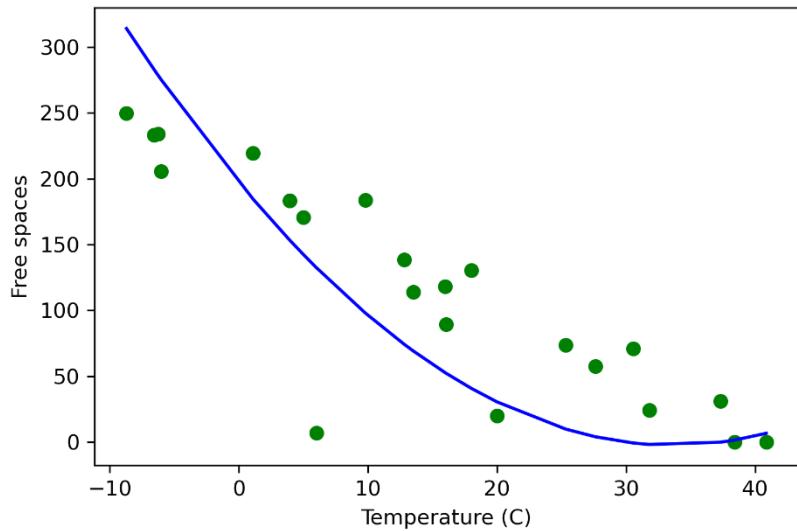
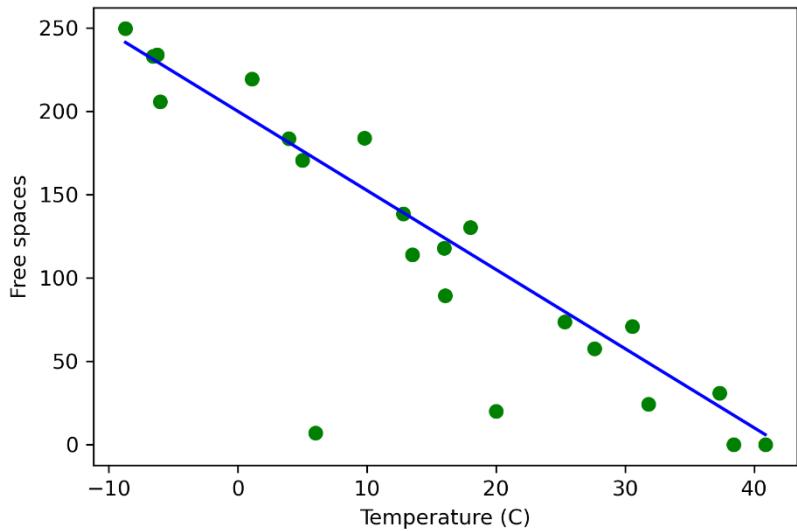
Teaser Problem



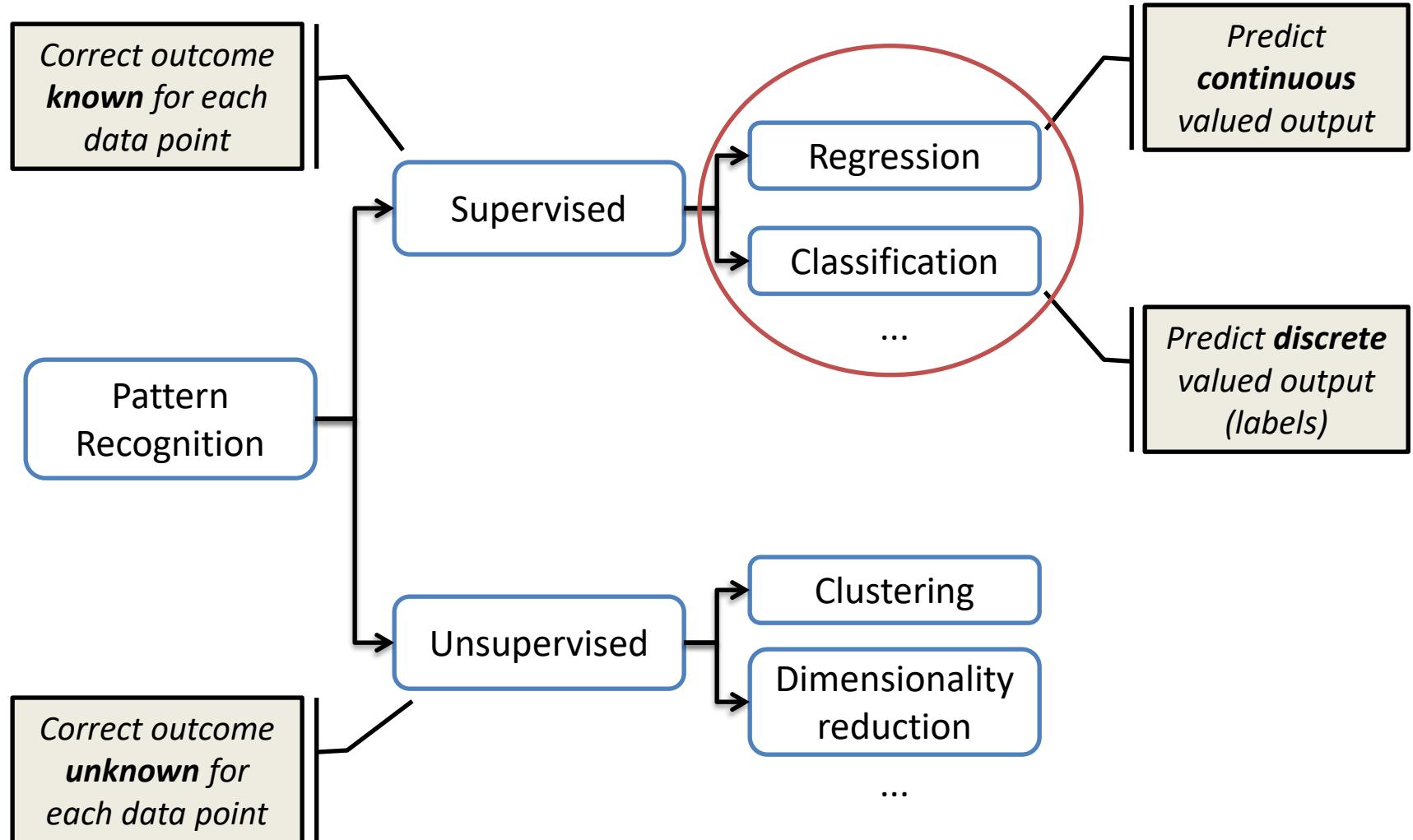
Which of the two is a better function to use?



Teaser Problem



Pattern recognition algorithms



Accompanying Problems

Problems are provided as Python Notebooks

Objectives:

- To have a practical implementation of all the algorithms we discuss in the theory class
- To obtain some minimum hands-on experience with the gradient descent algorithm applied to regression and classification
- To refresh key python procedures and programming practices for dealing with data, visualisation, reporting results etc
- To apply the aforementioned to a simple computer vision problem

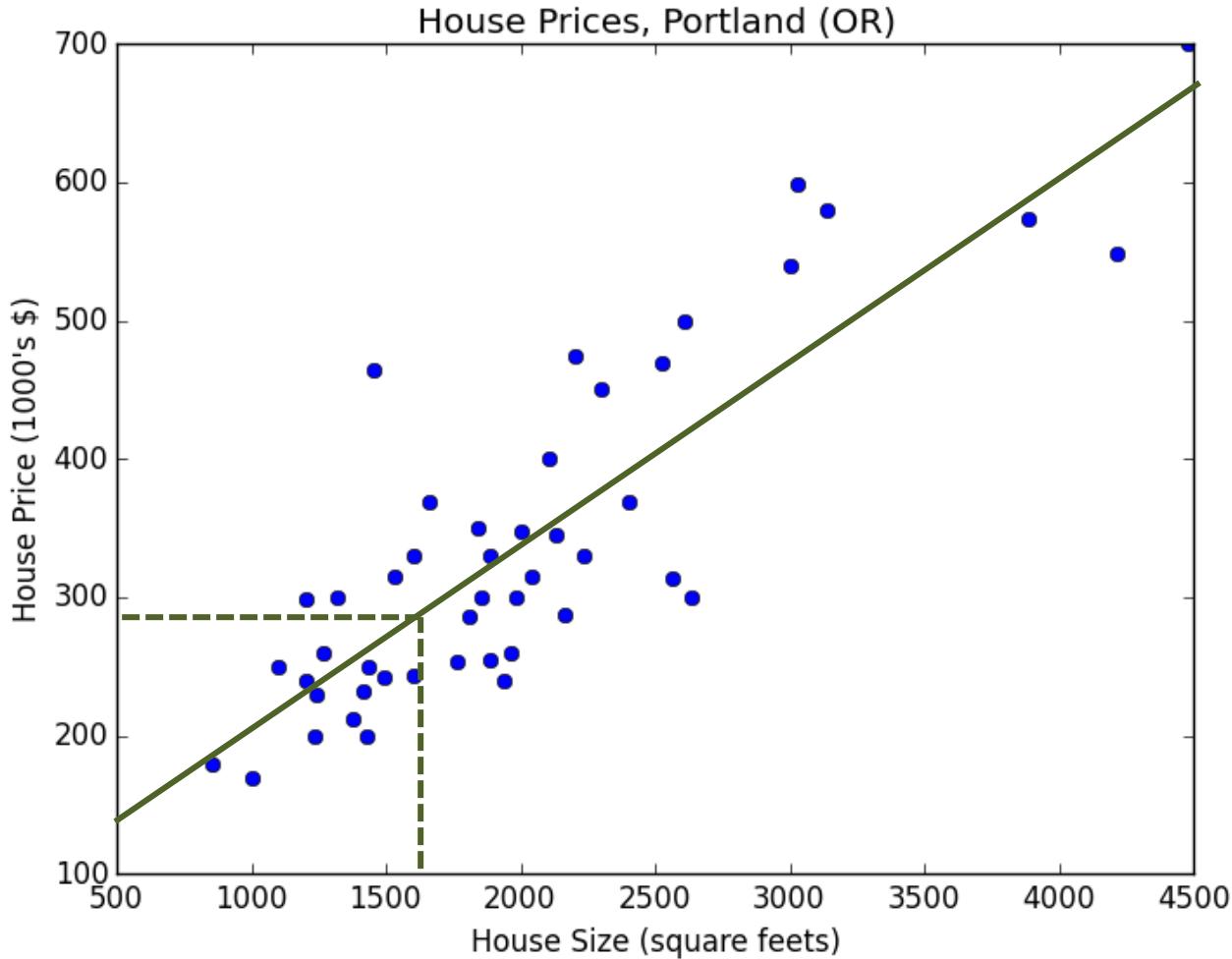
Problems are **optional** – but **highly recommended** to do

The material in the problems is exam material

Check instructions and download the notebooks from M3 shared space

LINEAR REGRESSION AND GRADIENT DESCENT

The right price



Supervised Learning
“right answers” given

Regression: Predict continuous
valued output (price)

Training Set

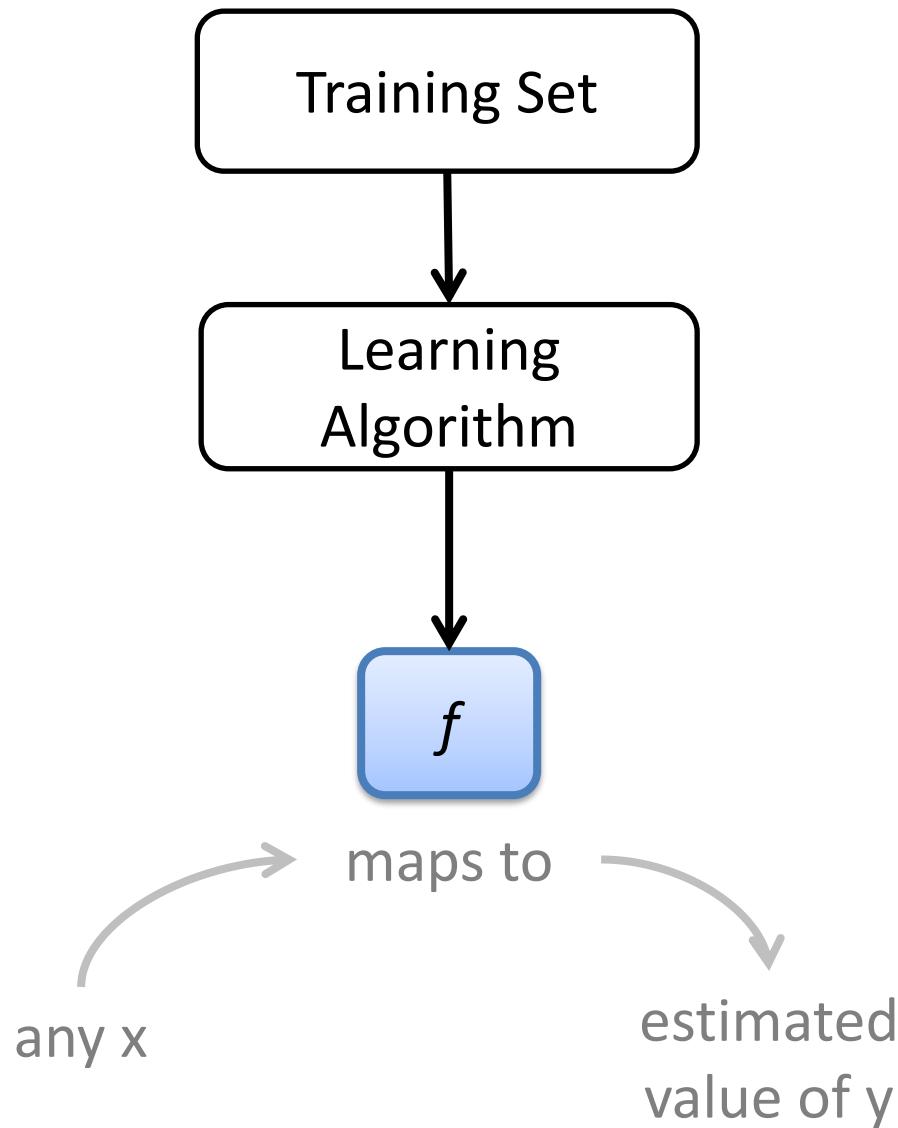
| Size in feet ² (x) | Price (\$) in 1000's (y) | |
|-------------------------------|--------------------------|--|
| 2104 | 460 | |
| 1416 | 232 | |
| 1534 | 315 | |
| 852 | 178 | |
| ... | ... | |

m

Notation

| | |
|----------------------|-----------------------------------|
| m | number of training examples |
| x | Input variable / features |
| y | Output variable / target variable |
| (x, y) | one training example |
| $(x^{(i)}, y^{(i)})$ | i^{th} training example |

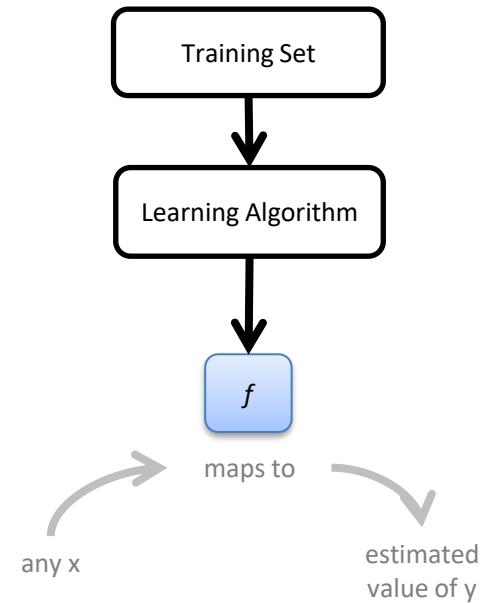
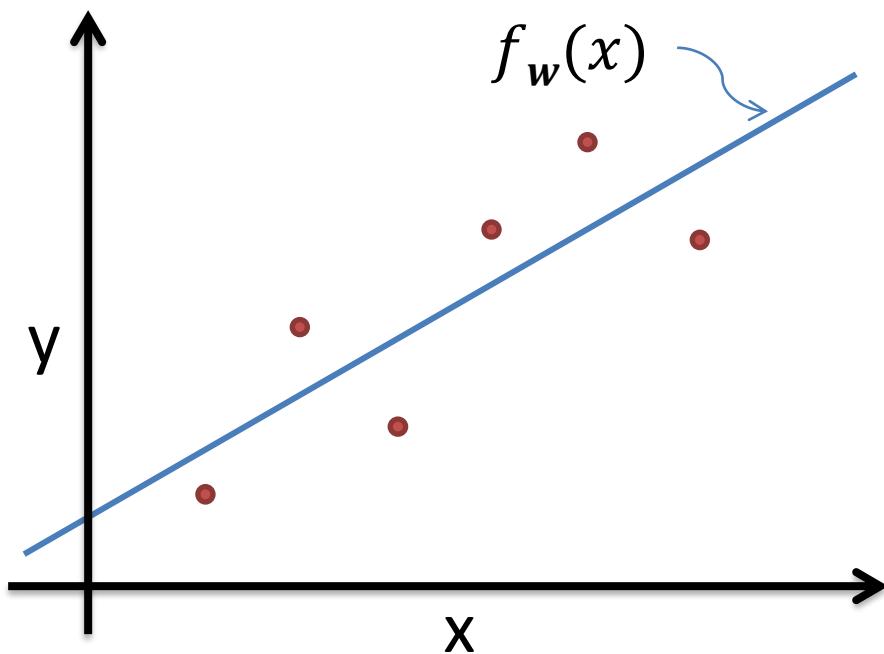
The Process



Model Representation

To start, we will represent f as a linear function of one variable:

$$\hat{y} = f_w(x) = w_0 + w_1 x$$



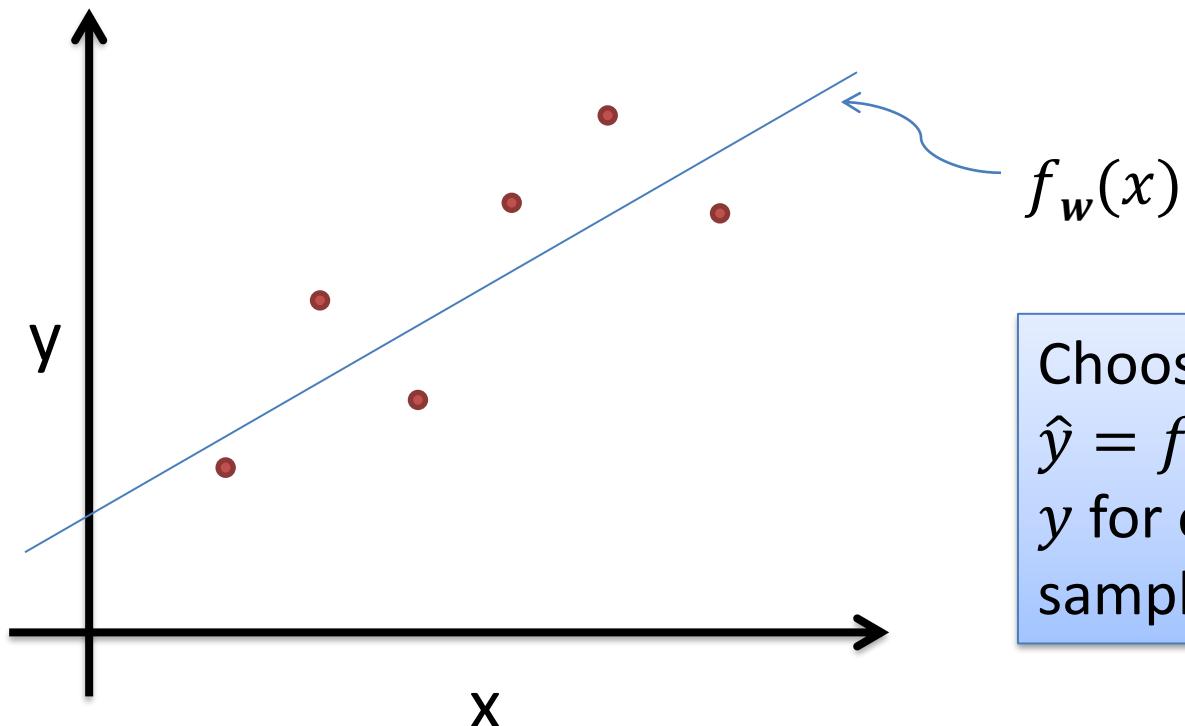
Updated notation

| | |
|----------------------|---|
| m | number of training examples |
| x | Input variable / features |
| y | Target variable |
| \hat{y} | Predicted output |
| (x, y) | one training example |
| $(x^{(i)}, y^{(i)})$ | i^{th} training example (out of m) |
| $f_w(x)$ | hypothesis |
| w_j | j^{th} parameter |

Model representation for univariate linear regression:

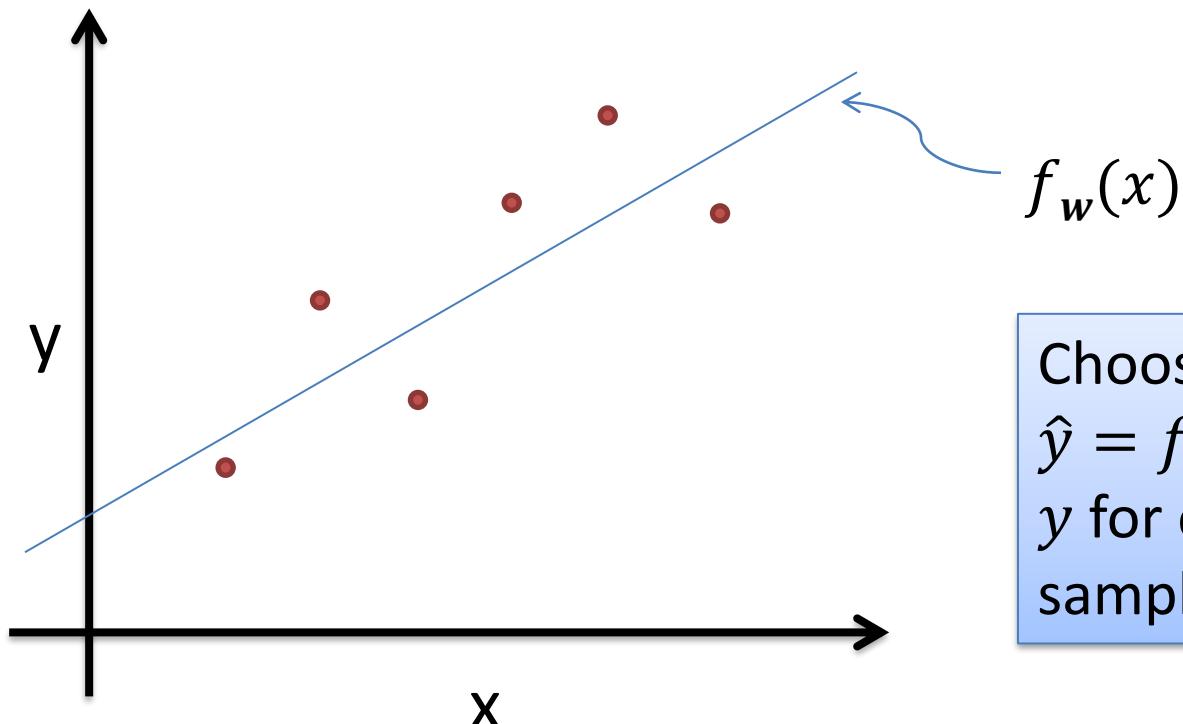
$$\hat{y} = f_w(x) = w_0 + w_1 x$$

The Cost (Loss) Function



Choose w_0, w_1 so that
 $\hat{y} = f_w(x)$ is close to
 y for our training
samples (x, y)

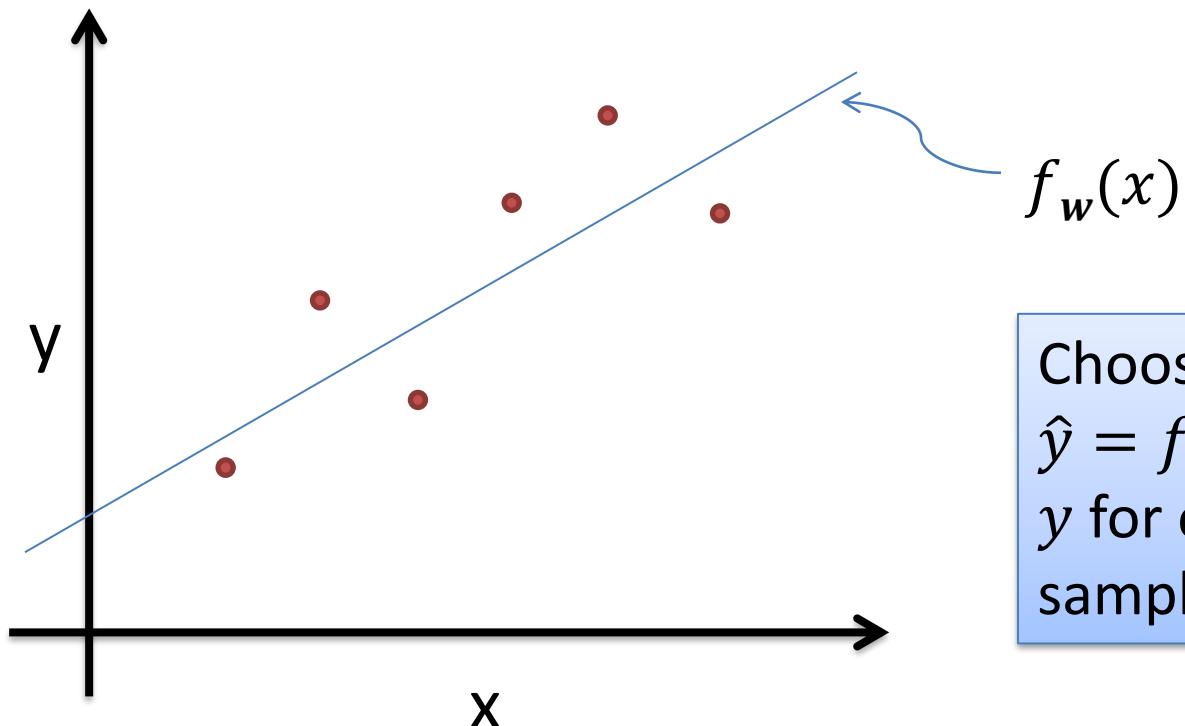
The Cost (Loss) Function



Choose w_0, w_1 so that
 $\hat{y} = f_w(x)$ is close to
 y for our training
samples (x, y)

$$\underset{w_0, w_1}{\text{minimise}} \underbrace{\left(\frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2 \right)}_{J(w_0, w_1)}$$

The Cost (Loss) Function



Choose w_0, w_1 so that
 $\hat{y} = f_w(x)$ is close to
 y for our training
samples (x, y)

Mean
Squared
Error (MSE)

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2$$

Insights on the cost function

Hypothesis: $f_w(x) = w_0 + w_1x$ → $f_w(x) = w_1x$

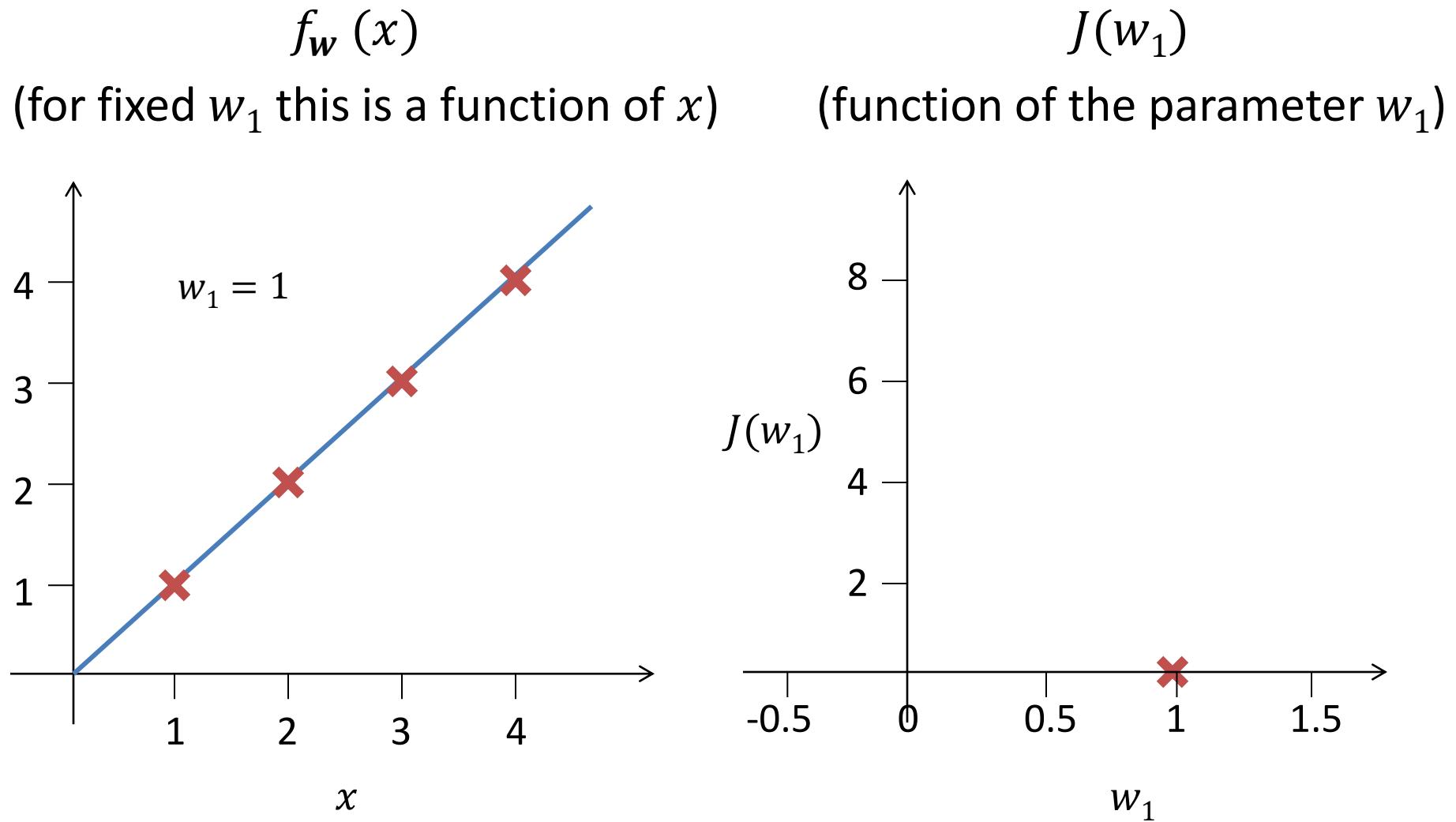
Parameters: w_0, w_1 → w_1

Cost Function:

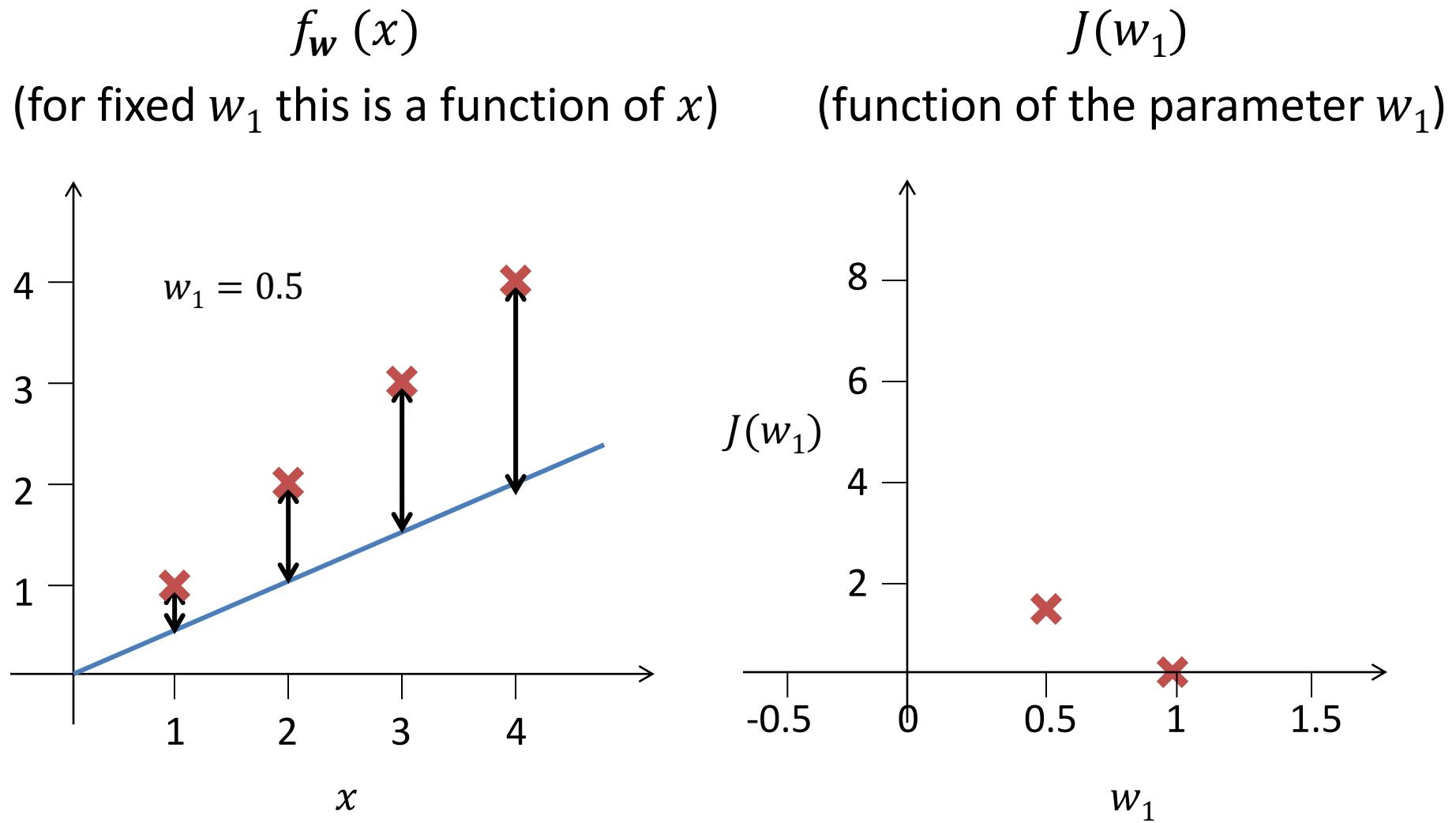
$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2 \rightarrow J(w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2$$

Goal: $\underset{w_0, w_1}{\text{minimise}}(J(w_0, w_1)) \rightarrow \underset{w_1}{\text{minimise}}(J(w_1))$

Insights on the cost function



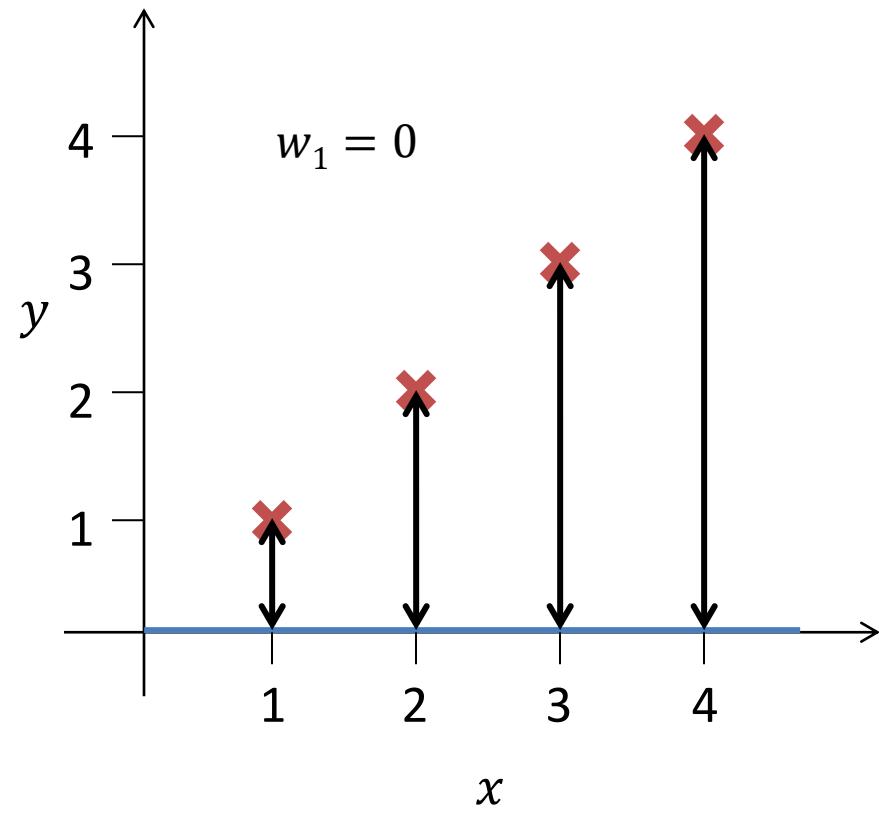
Insights on the cost function



Insights on the cost function

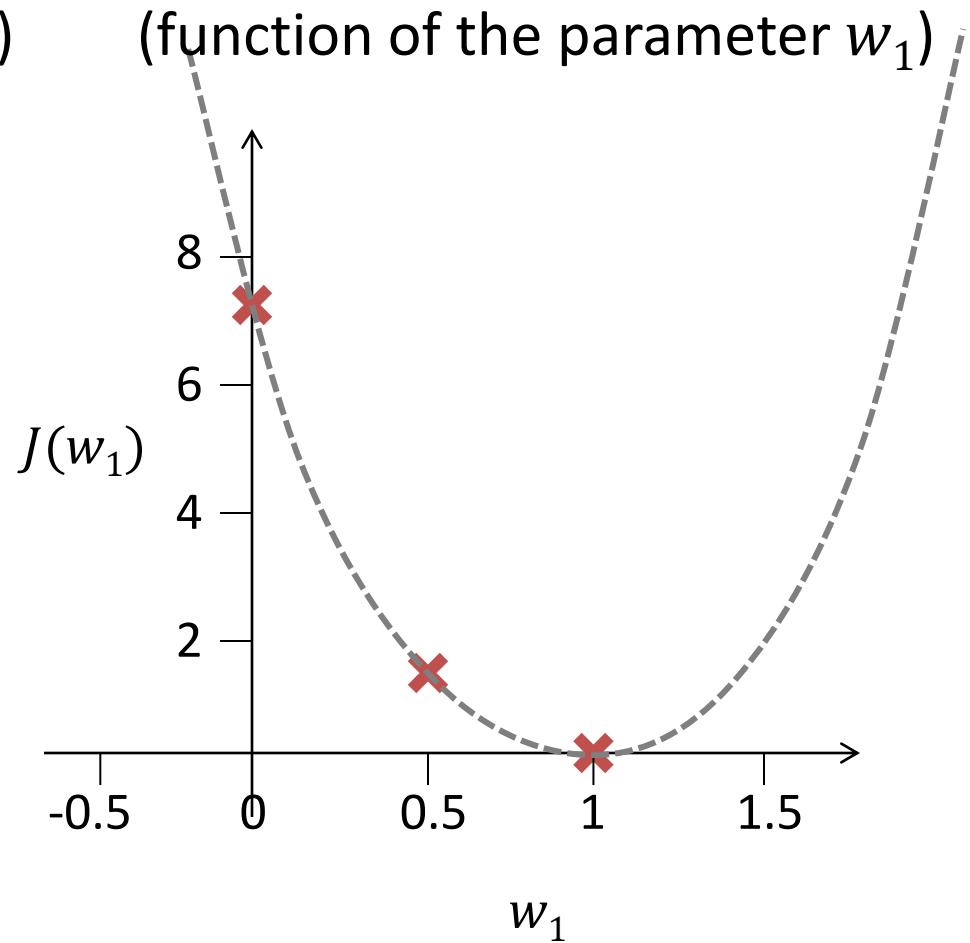
$f_w(x)$

(for fixed w_1 this is a function of x)

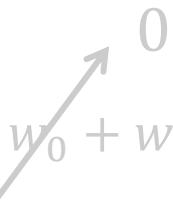


$J(w_1)$

(function of the parameter w_1)



Insights on the cost function

Hypothesis: $f_w(x) = w_0 + w_1x$  $\rightarrow f_w(x) = w_1x$

Parameters: w_0, w_1 $\rightarrow w_1$

Cost Function:

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2 \rightarrow J(w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2$$

Goal: $\underset{w_0, w_1}{\text{minimise}}(J(w_0, w_1))$ $\rightarrow \underset{w_1}{\text{minimise}}(J(w_1))$

Insights on the cost function

Hypothesis: $f_w(x) = w_0 + w_1x$

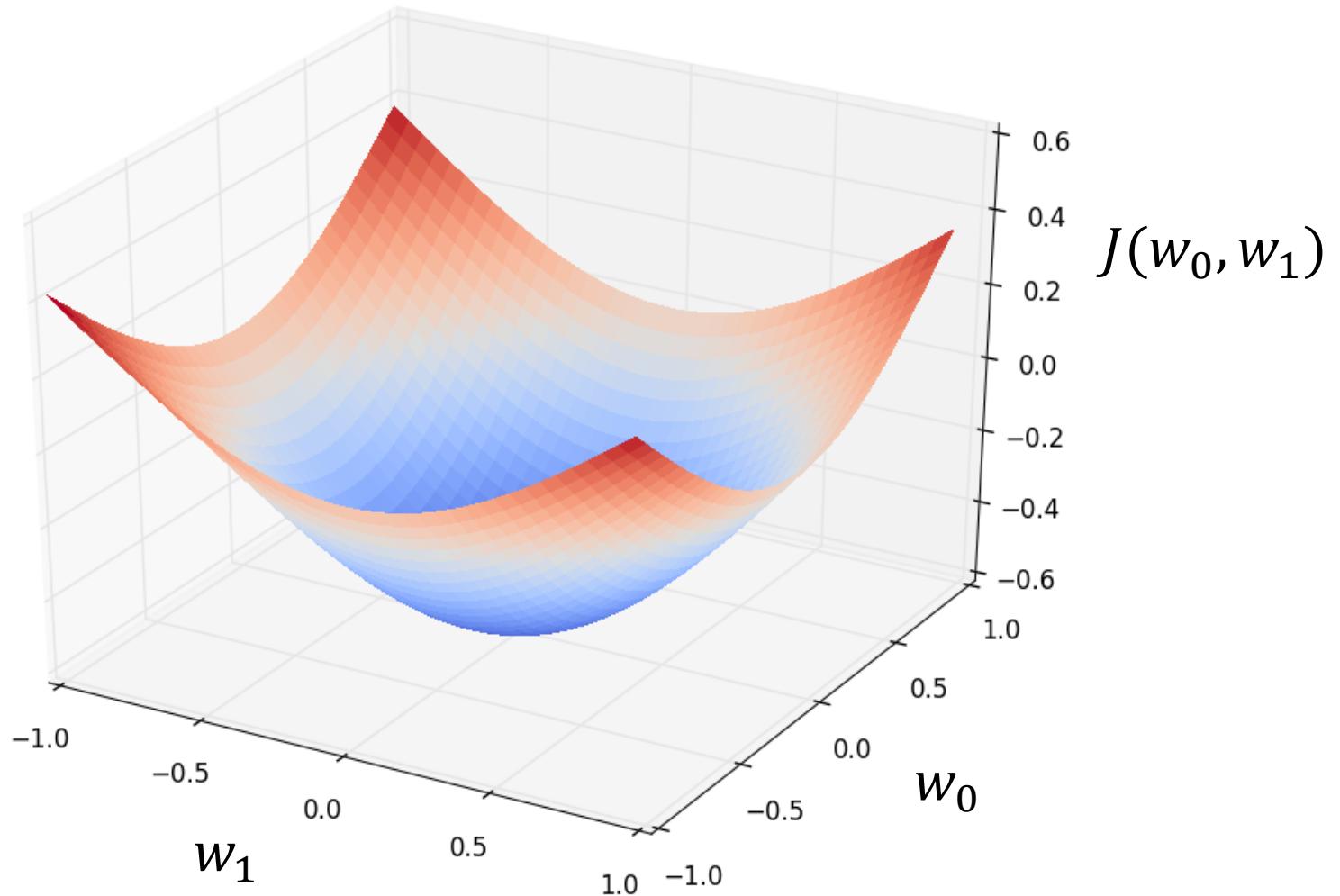
Parameters: w_0, w_1

Cost Function:

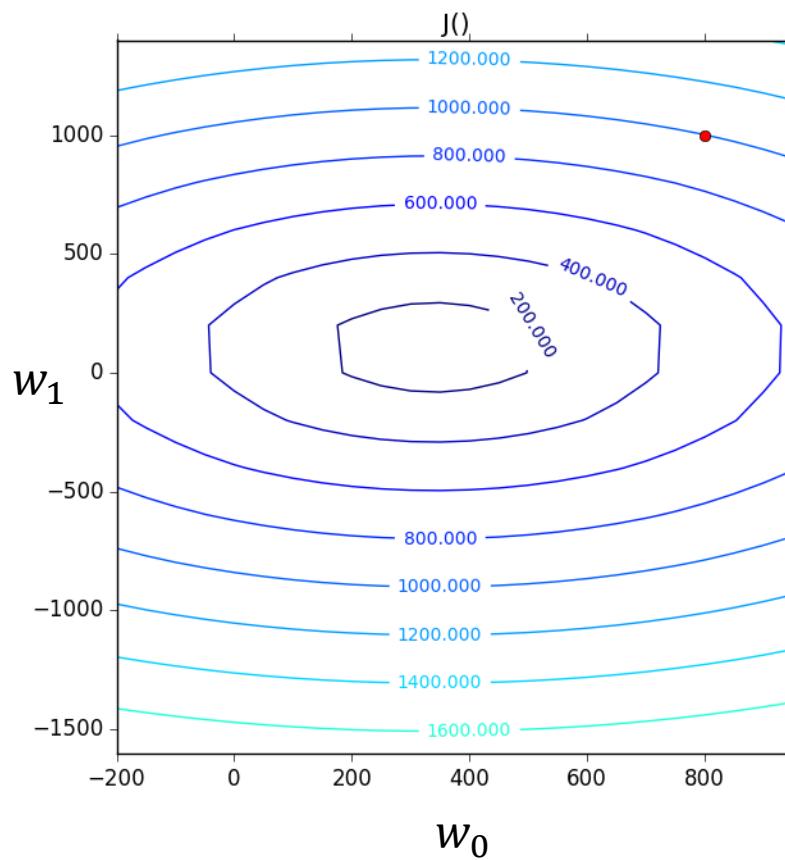
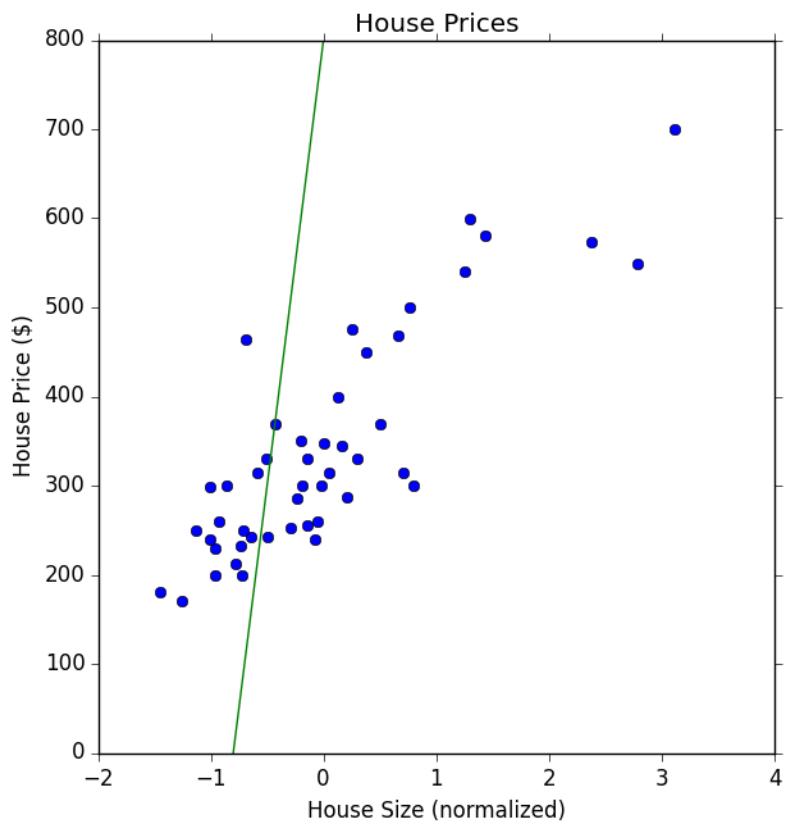
$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2$$

Goal: $\underset{w_0, w_1}{\text{minimise}}(J(w_0, w_1))$

Minimizing over two parameters

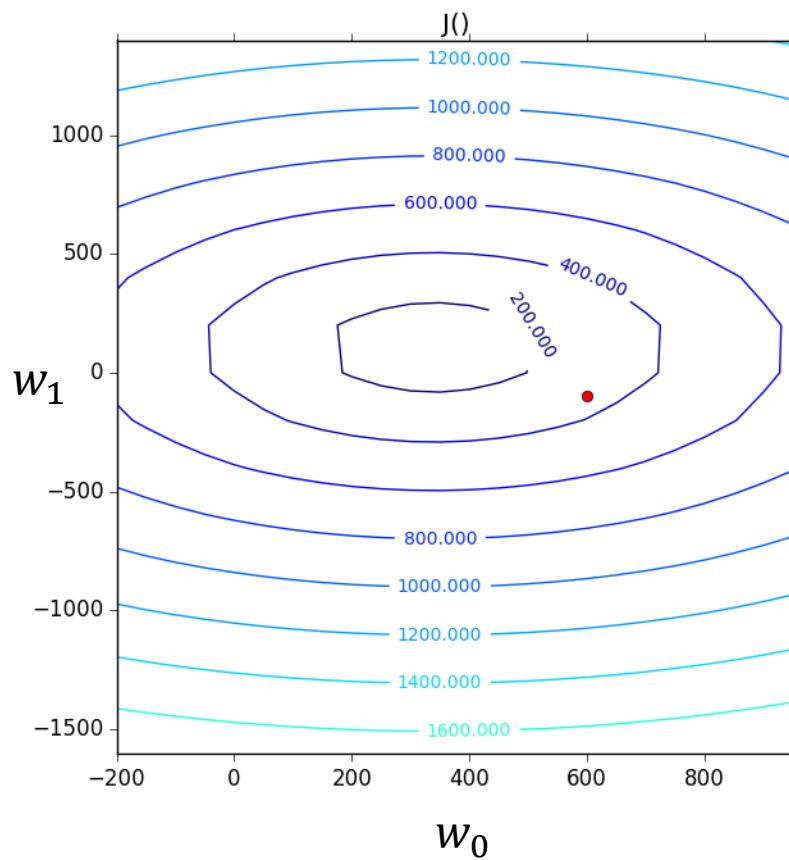
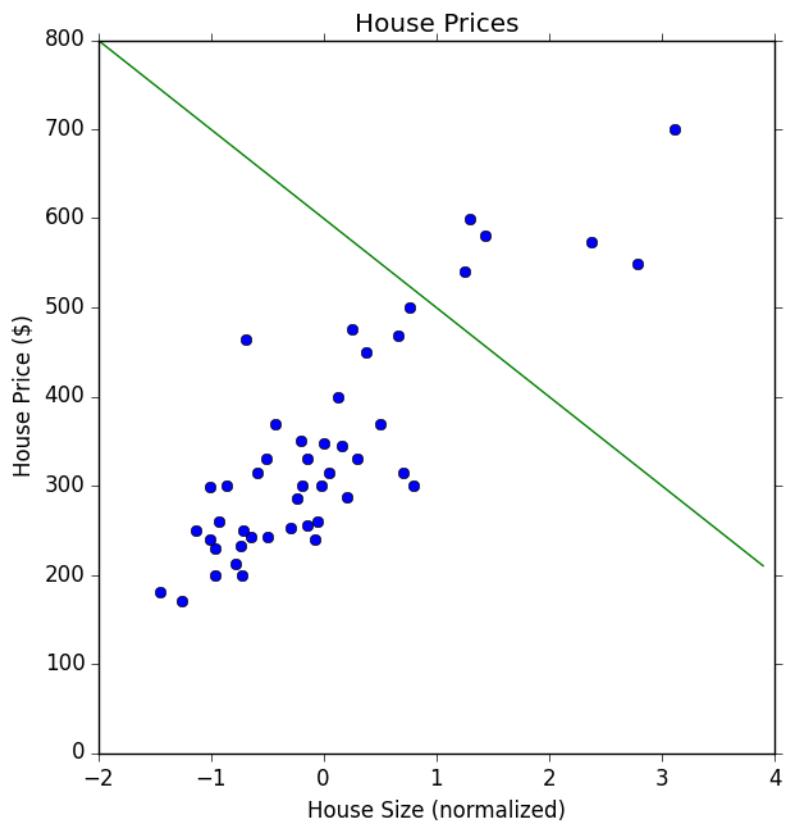


Cost of different hypotheses



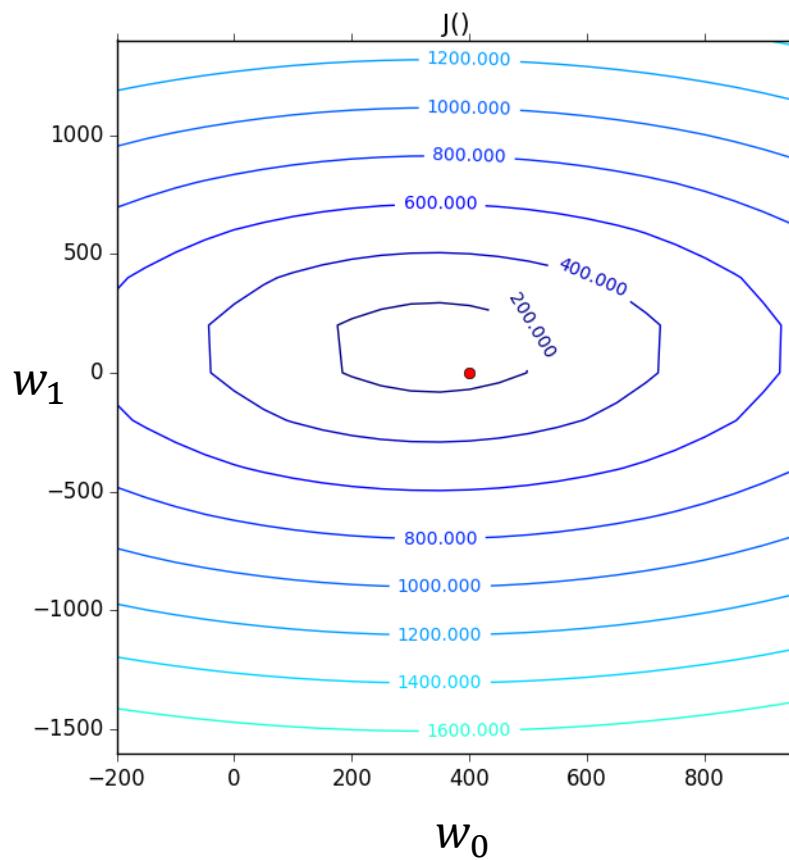
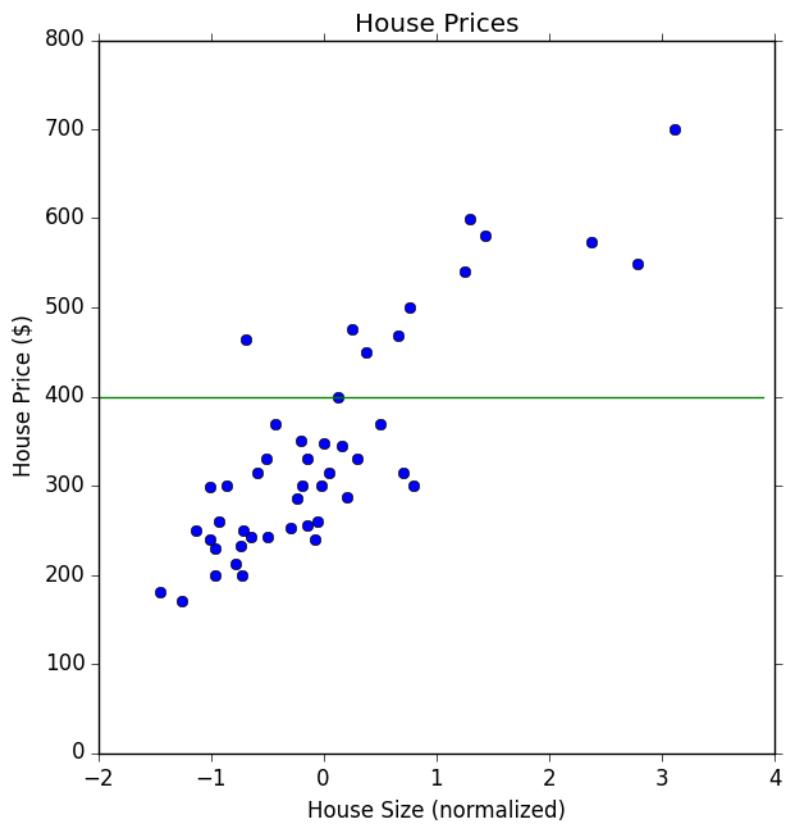
$$w_0 = 800, w_1 = 1000$$

Cost of different hypotheses



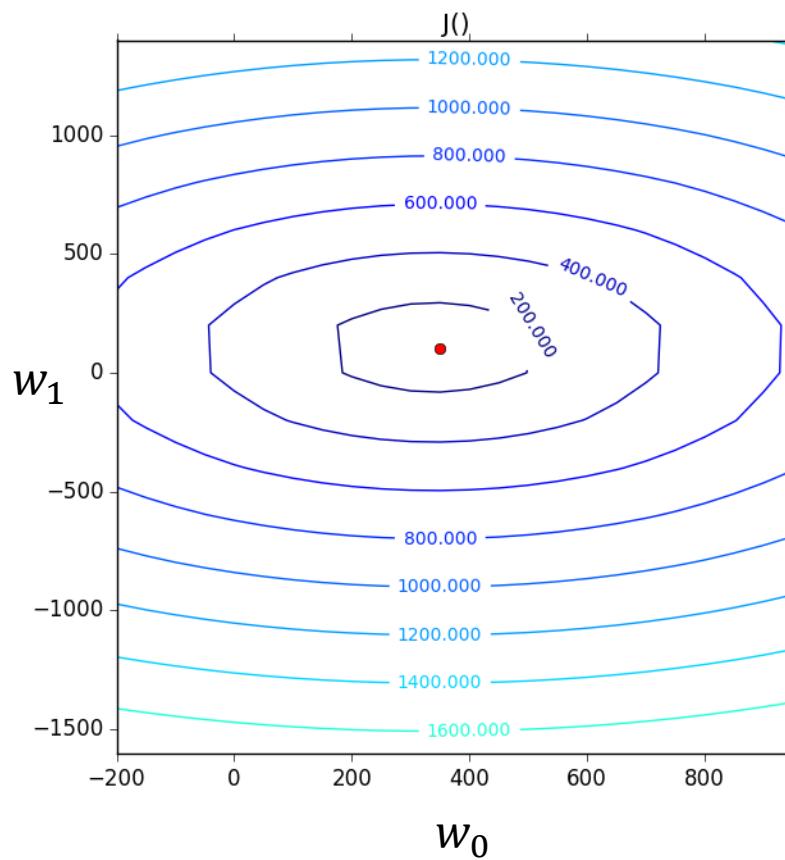
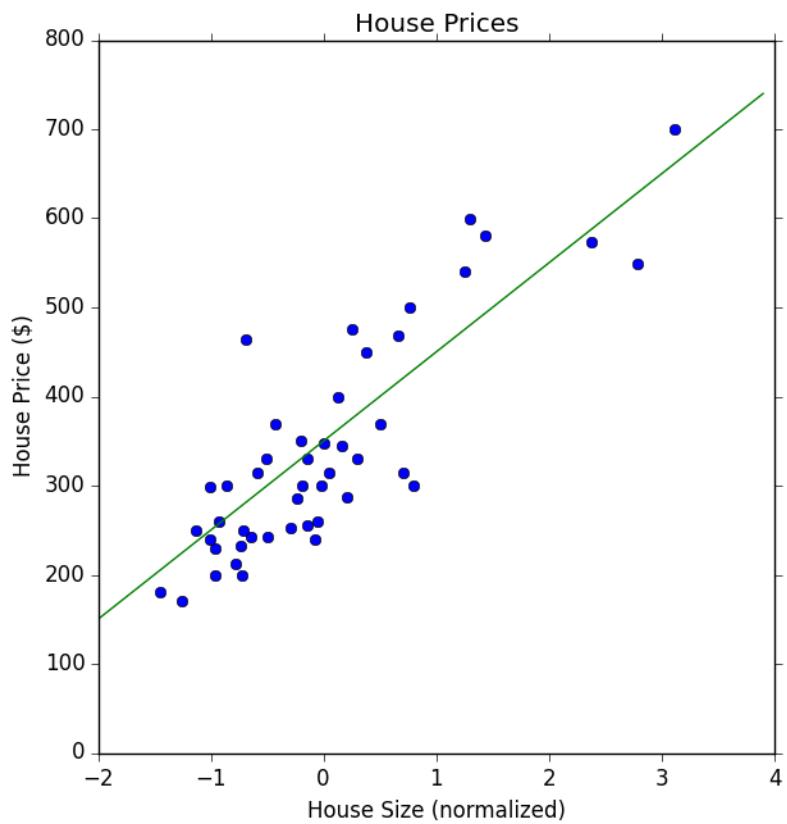
$$w_0 = 600, w_1 = -100$$

Cost of different hypotheses



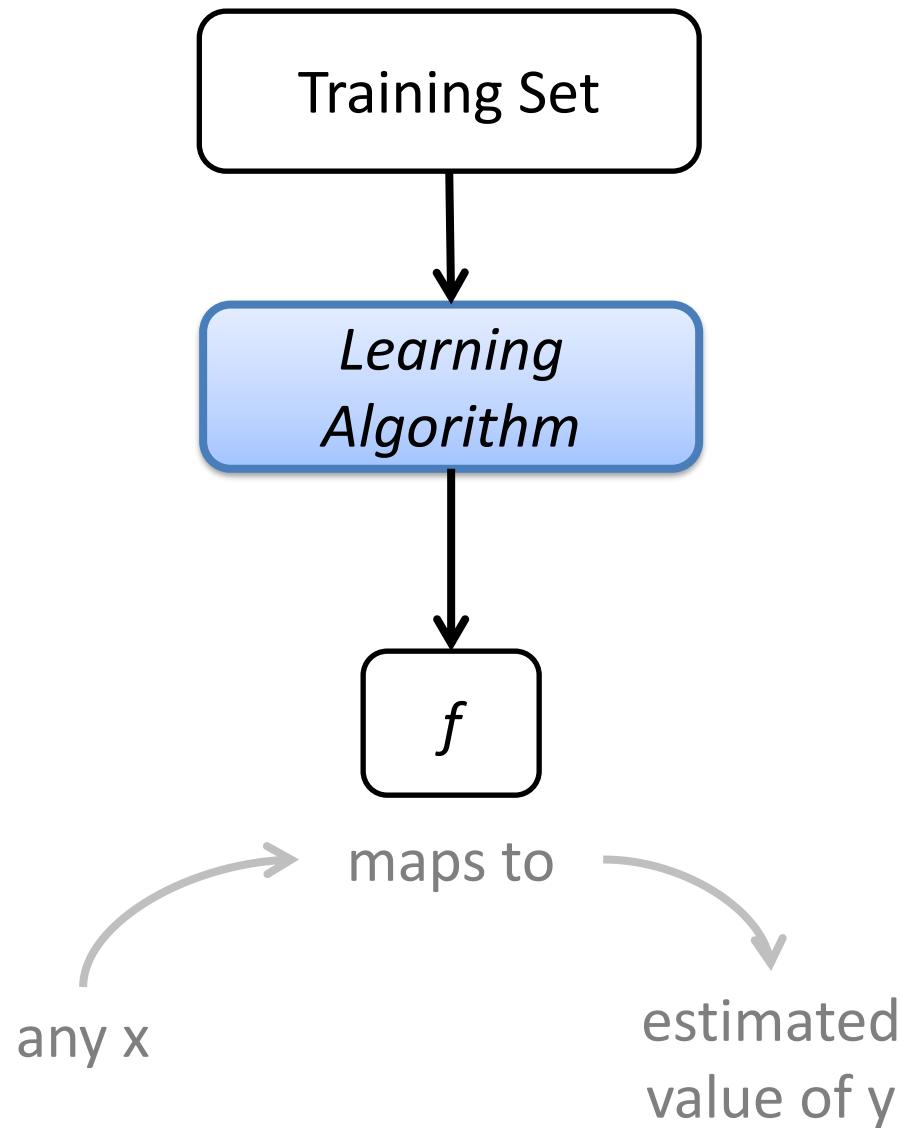
$$w_0 = 400, w_1 = 0$$

Cost of different hypotheses

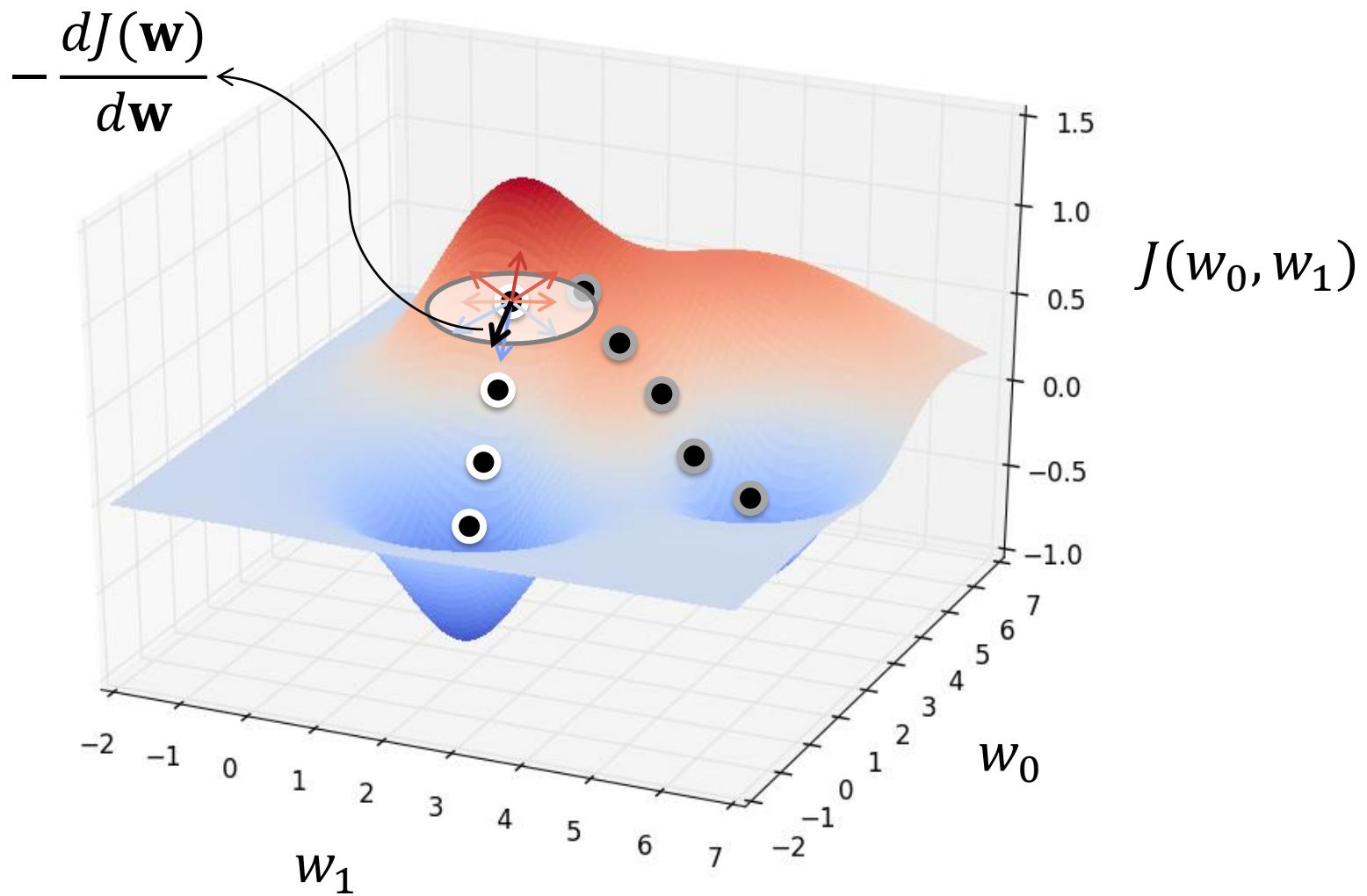


$$w_0 = 374, w_1 = 100$$

The Process



Gradient Descent Algorithm



Gradient Descent Algorithm

Repeat until convergence

{

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1)$$

, for ($j = 0$ and $j = 1$)

}

Learning rate: How *big* steps do you want to take. A *positive* number

Partial derivative: how *steep* it is at a particular direction

Gradient Descent Algorithm

Repeat until convergence

{

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1)$$

, for ($j = 0$ and $j = 1$)

}

Incorrect – sequential update

$$w_0 \leftarrow w_0 - \alpha \frac{\partial}{\partial w_0} J(w_0, w_1)$$

$$w_1 \leftarrow w_1 - \alpha \frac{\partial}{\partial w_1} J(w_0, w_1)$$

Correct – simultaneous update

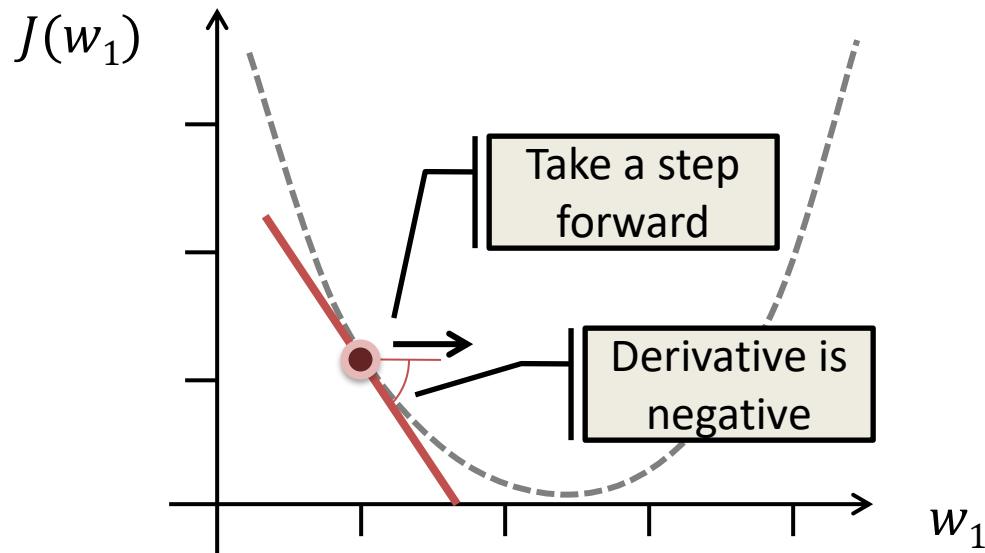
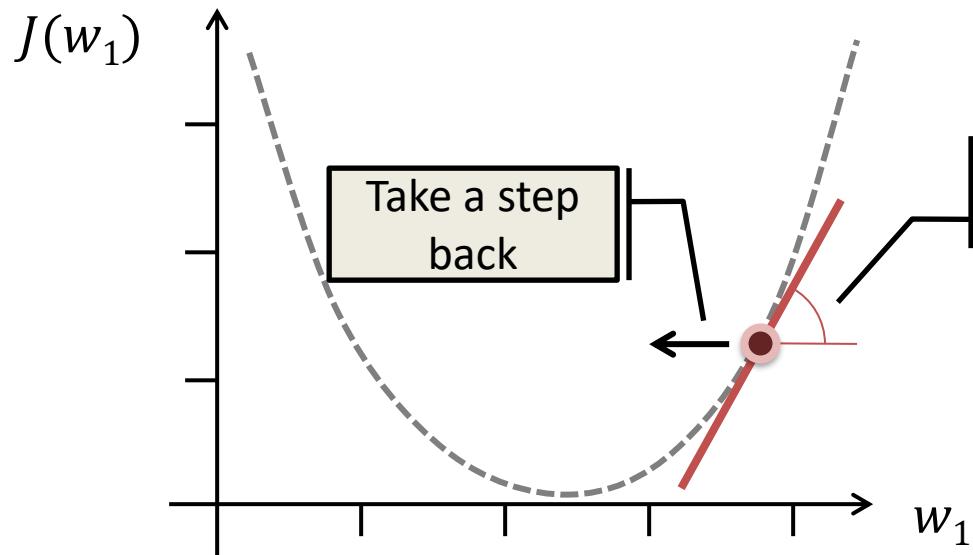
$$\text{Temp0} \leftarrow w_0 - \alpha \frac{\partial}{\partial w_0} J(w_0, w_1)$$

$$\text{Temp1} \leftarrow w_1 - \alpha \frac{\partial}{\partial w_1} J(w_0, w_1)$$

$$w_0 \leftarrow \text{Temp0}$$

$$w_1 \leftarrow \text{Temp1}$$

The derivative – which way

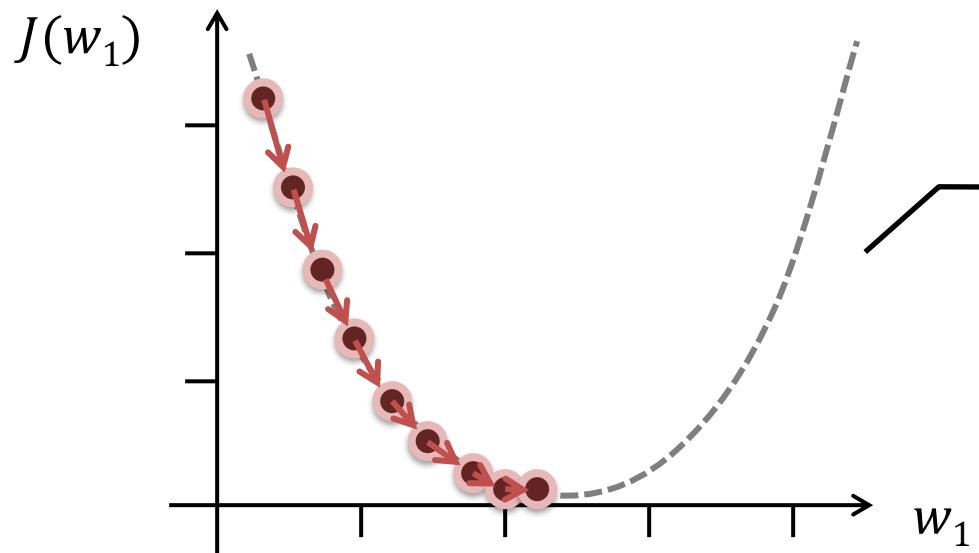


Let's maintain w_0 fixed to simplify this example. Our step change becomes:

$$w_1 \leftarrow w_1 - \alpha \frac{d}{dw_1} J(w_1)$$

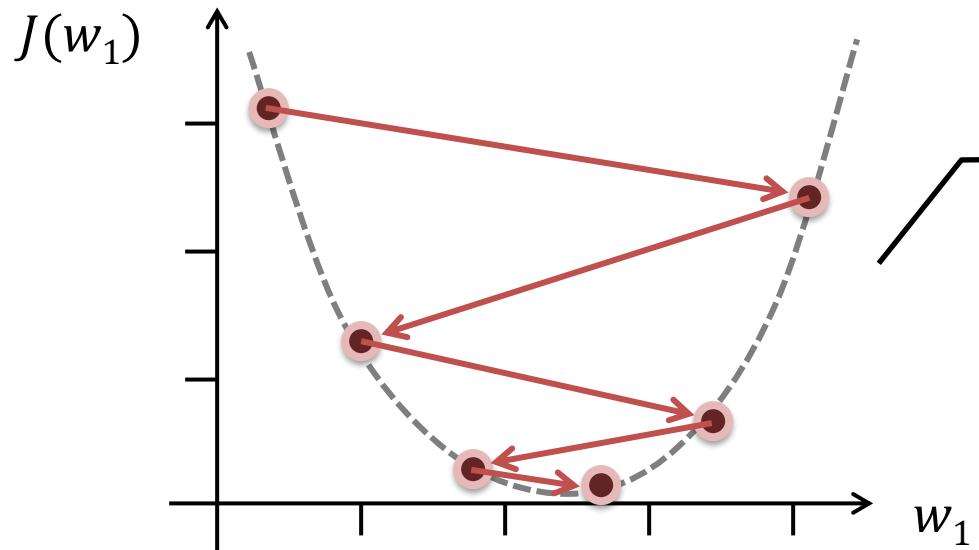
Remember, α is a **positive** number

The learning rate – how fast



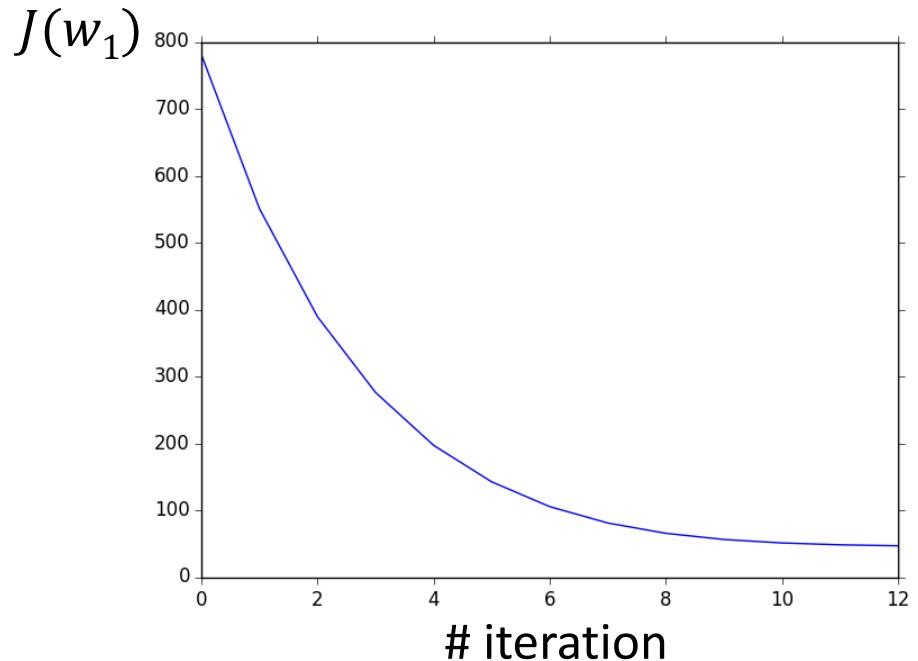
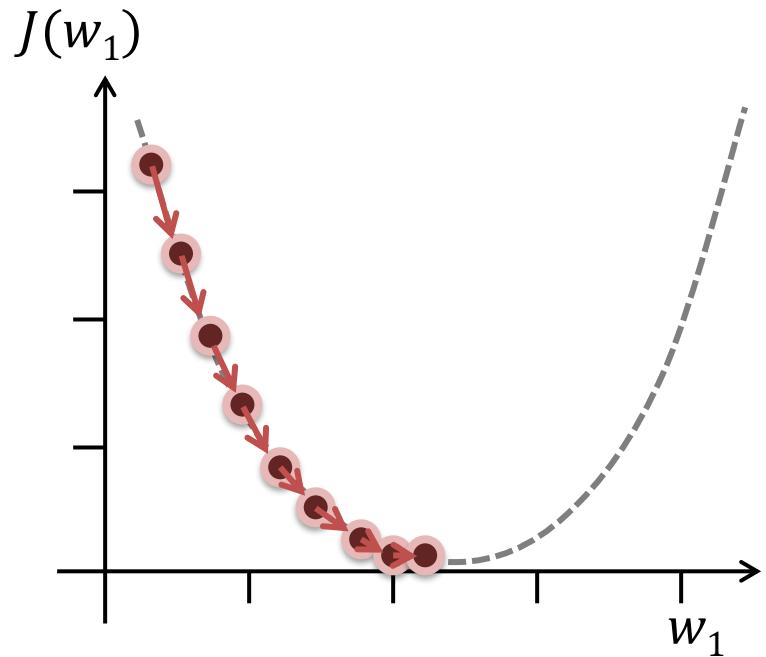
If the learning rate α is too small, it takes many steps to converge

$$w_1 \leftarrow w_1 - \alpha \frac{d}{dw_1} J(w_1)$$



If the learning rate α is large gradient descent might overshoot the minimum. It may still converge, or it may diverge if α is too large

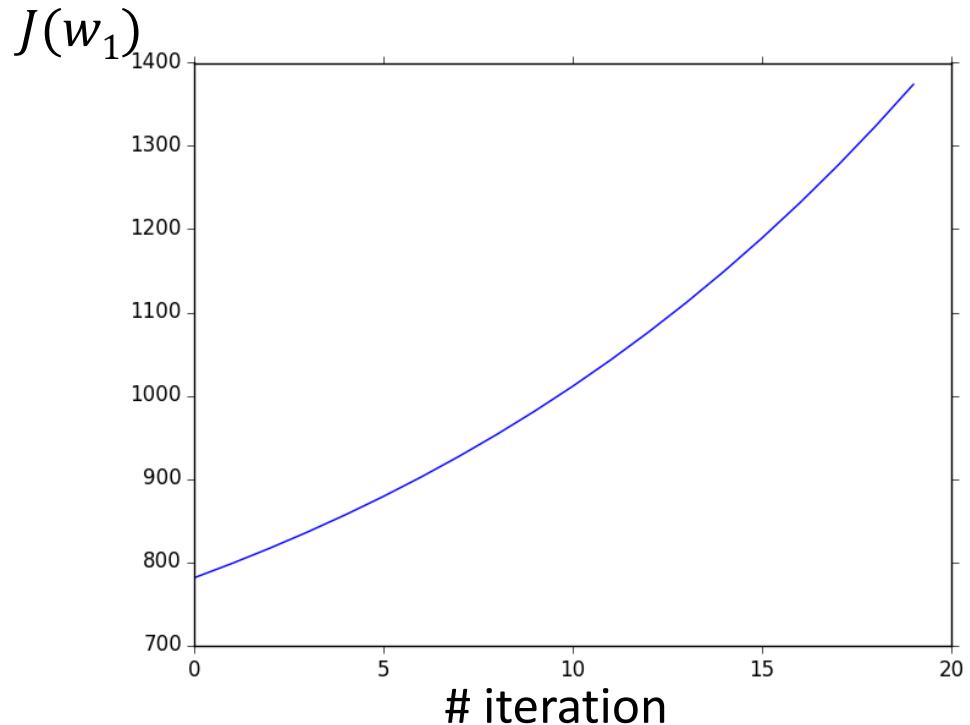
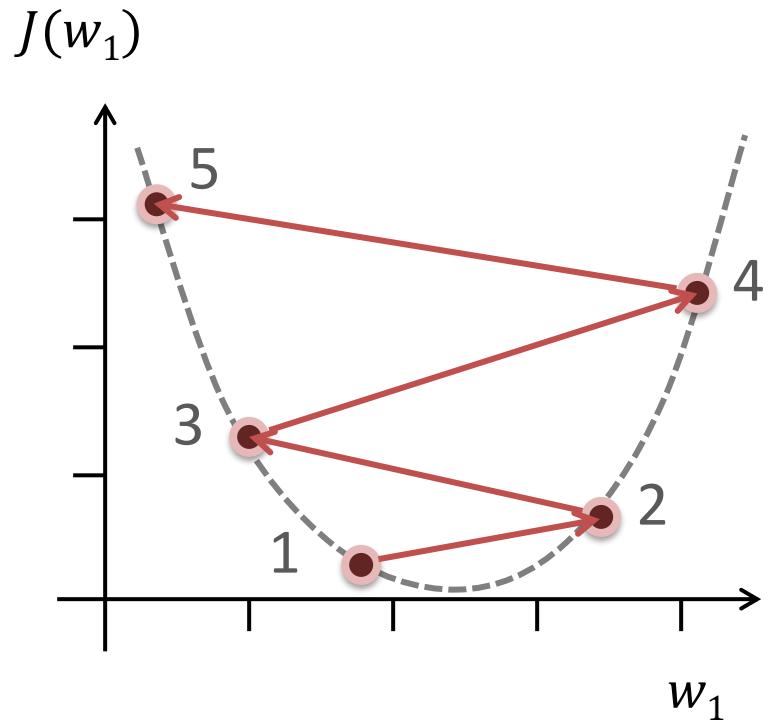
Choosing the learning rate



Plotting the evolution of the cost function is a good sanity check to verify that the learning rate is correctly set.

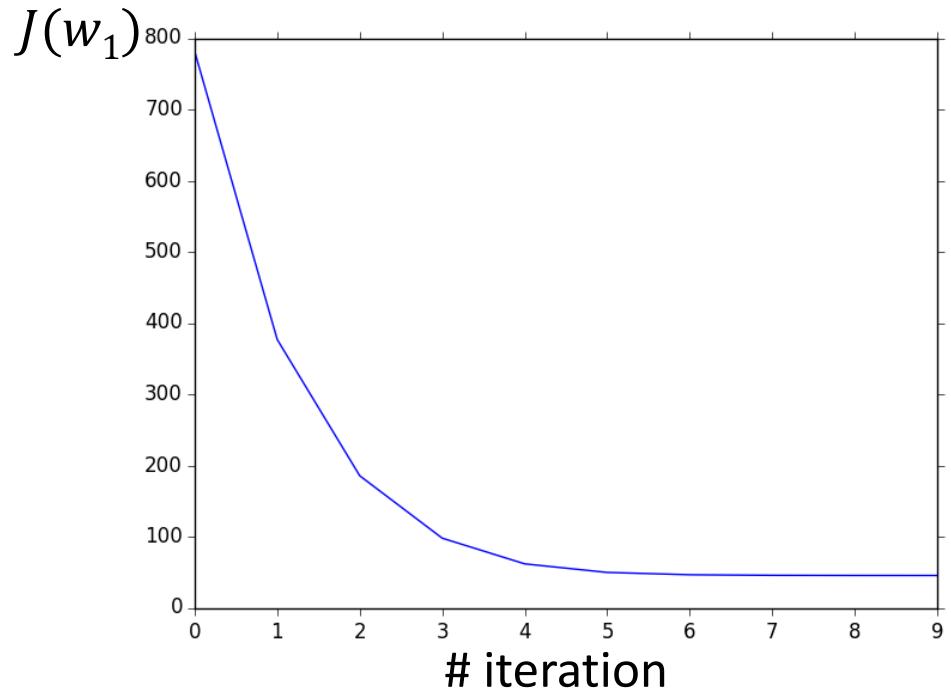
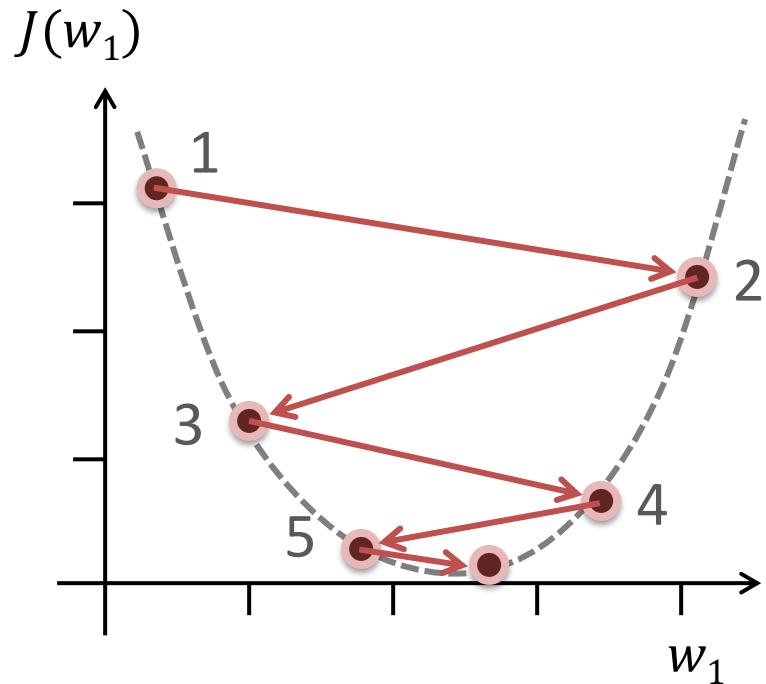
Expected behaviour: cost function yields smaller values in every iteration and converges

Choosing the learning rate



If the learning rate is very high, the algorithm overshoots the minimum and diverges

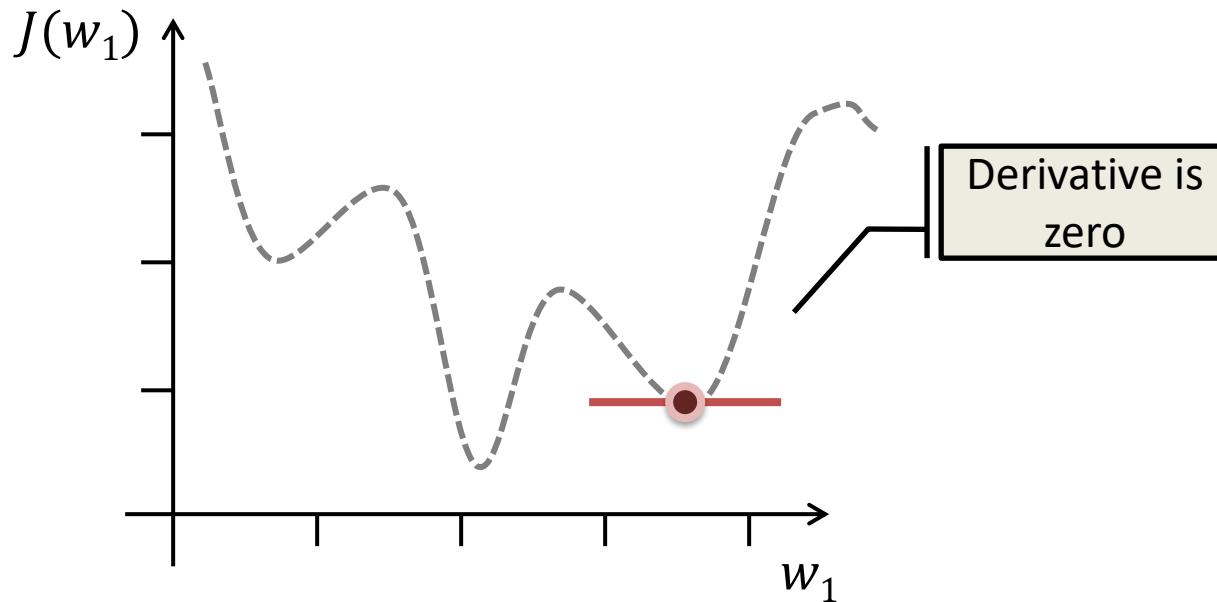
Choosing the learning rate



But, overshooting the minimum does not necessarily mean that the algorithm will diverge

Termination condition: if $J(w)$ decreases by less than ϵ in one iteration

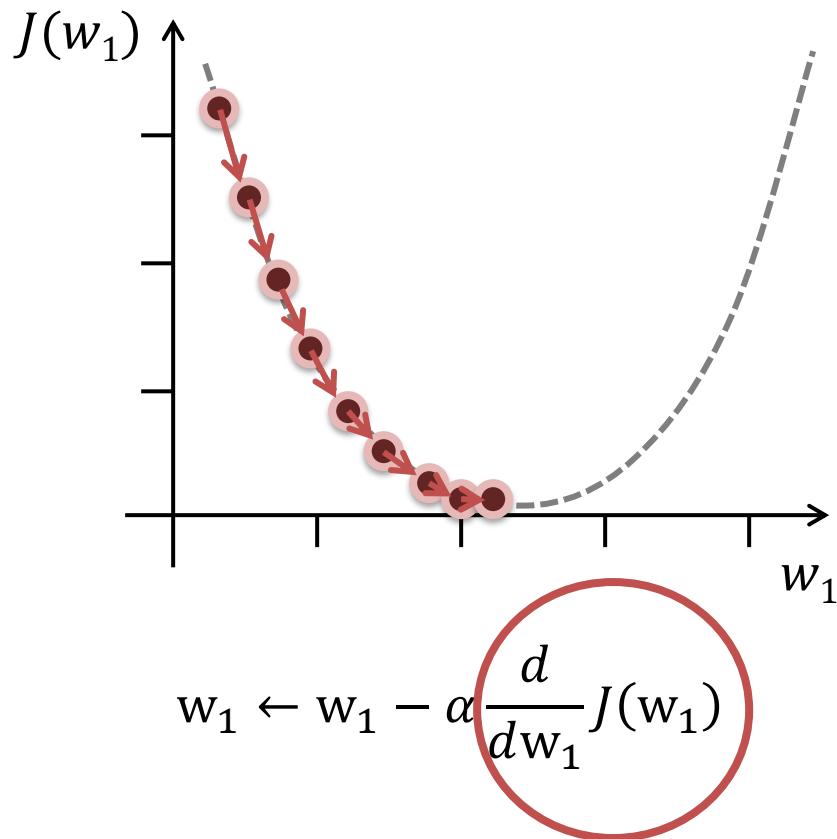
Converging



What happens when we reach a local minimum?

$$w_1 \leftarrow w_1 - \alpha \frac{d}{dw_1} J(w_1)$$

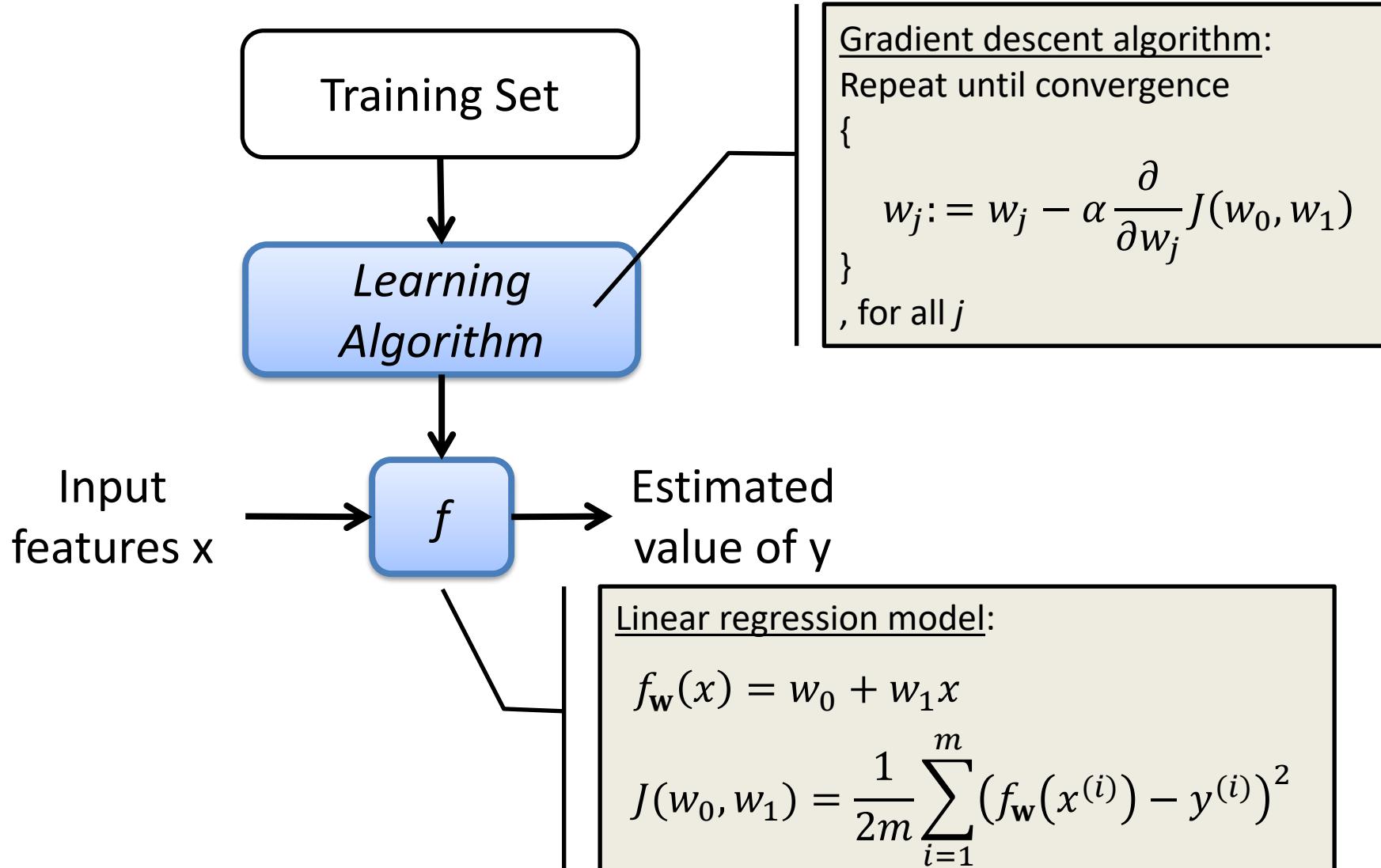
Converging



As we approach a local minimum the derivative approaches slowly zero, so gradient descent will **automatically take smaller steps.**

Gradient descent can converge to a local minimum, even with the learning rate α fixed - no need to decrease α over time.

Gradient Descent for Linear Regression



The derivatives of the square error function

$$\begin{aligned}\frac{\partial}{\partial w_j} J(w_0, w_1) &= \frac{\partial}{\partial w_j} \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2 \\ &= \frac{\partial}{\partial w_j} \frac{1}{2m} \sum_{i=1}^m (w_0 + w_1 x^{(i)} - y^{(i)})^2\end{aligned}$$

$$\begin{aligned}j = 0: \frac{\partial}{\partial w_0} J(w_0, w_1) &= \frac{1}{m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)}) \\ j = 1: \frac{\partial}{\partial w_1} J(w_0, w_1) &= \frac{1}{m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)}) \cdot x^{(i)}\end{aligned}$$

Gradient Descent Algorithm

Repeat until convergence

{

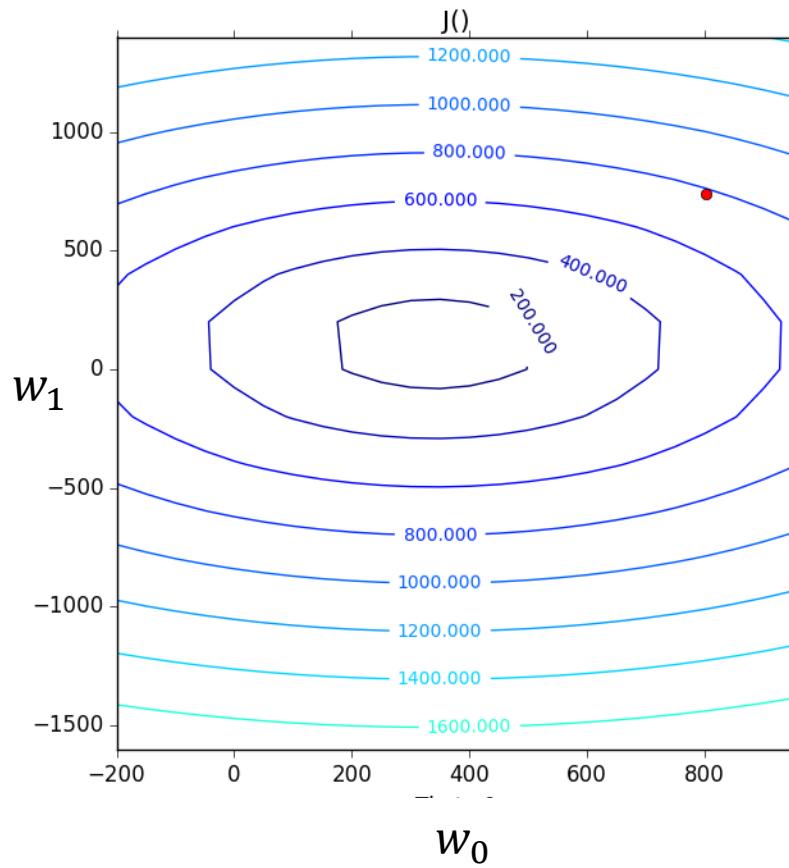
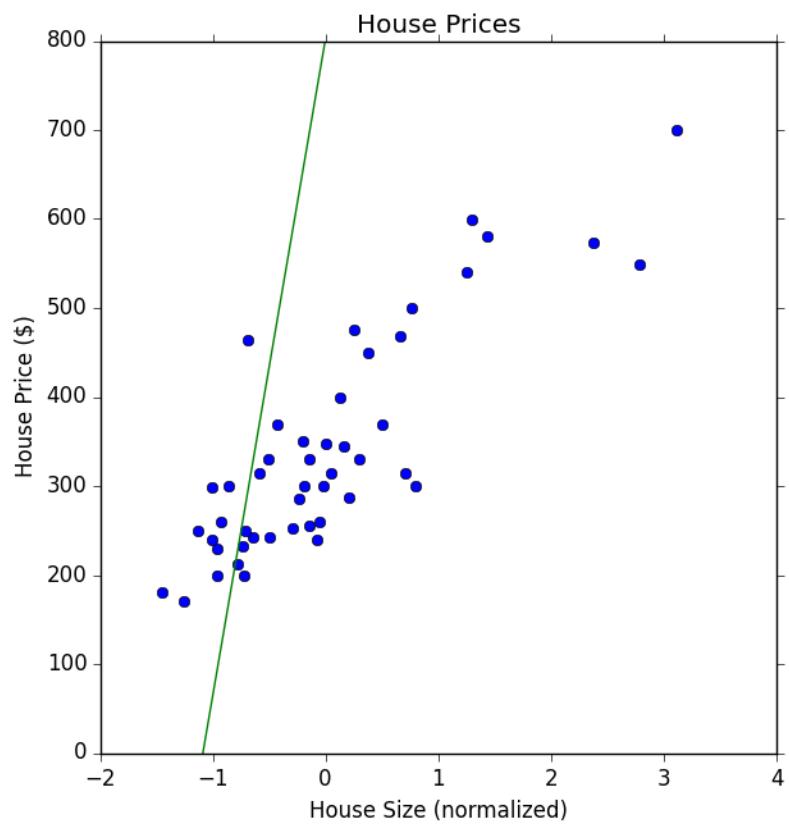
$$w_0 := w_0 - \alpha \frac{\partial}{\partial w_0} J(w_0, w_1) = w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})$$

$$w_1 := w_1 - \alpha \frac{\partial}{\partial w_1} J(w_0, w_1) = w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

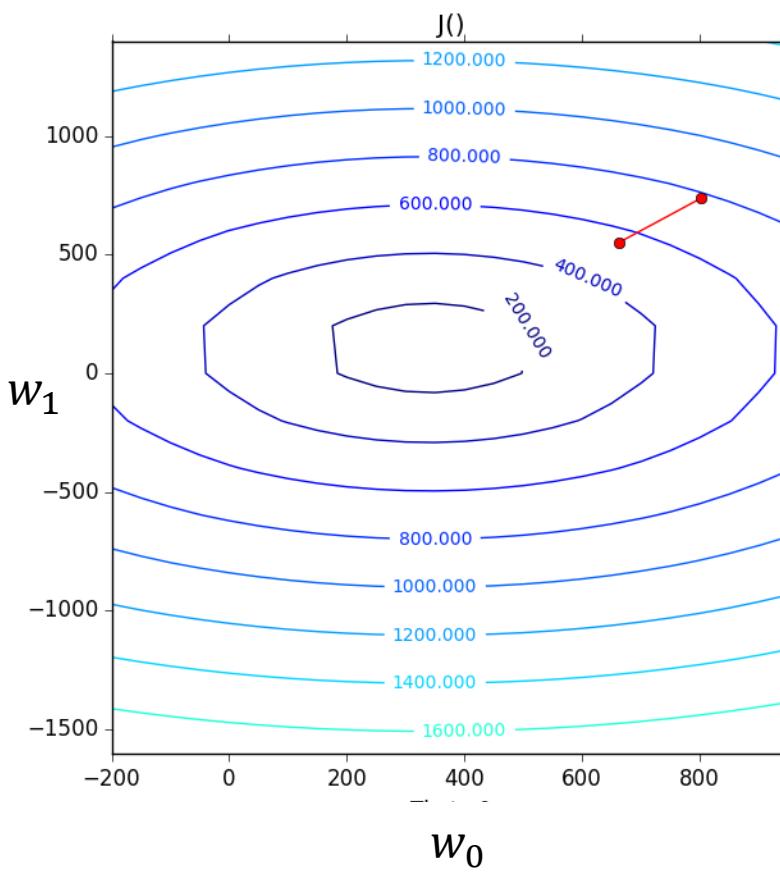
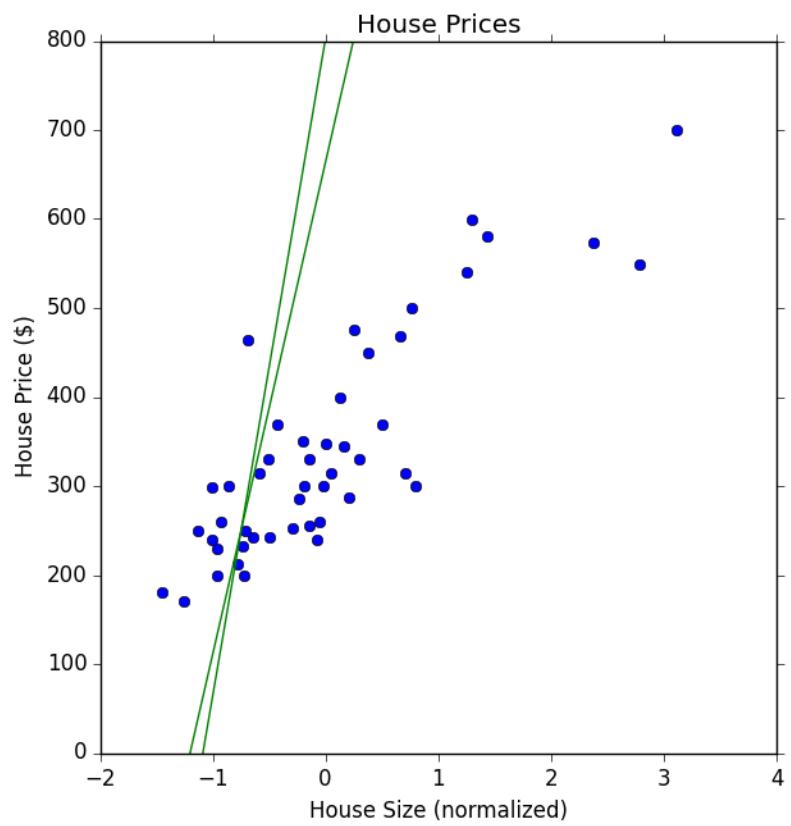
}

Where w_0 and w_1 should be updated simultaneously

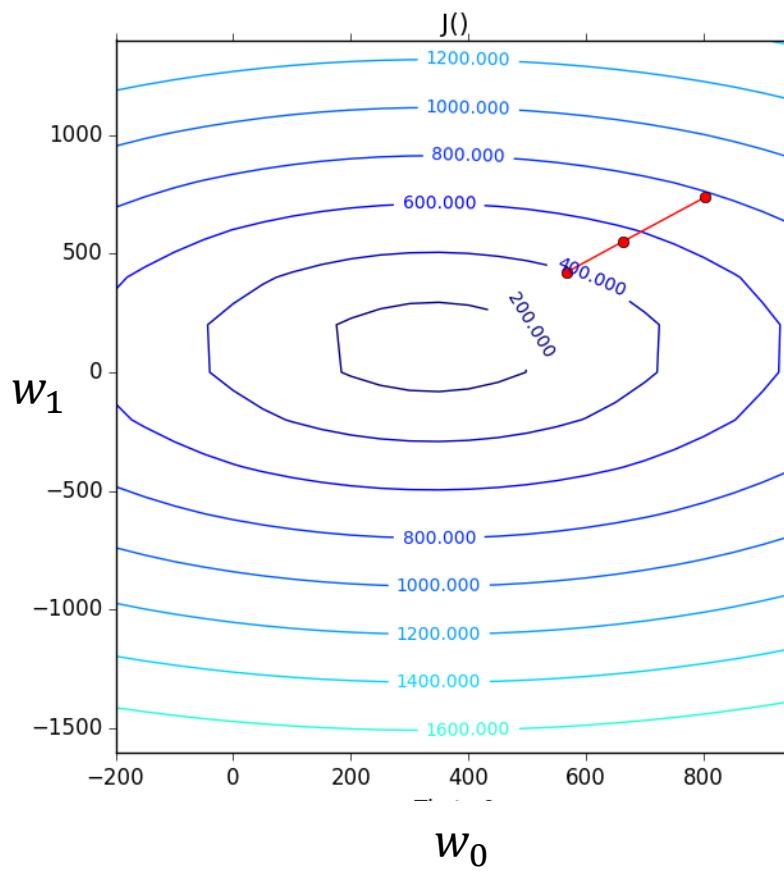
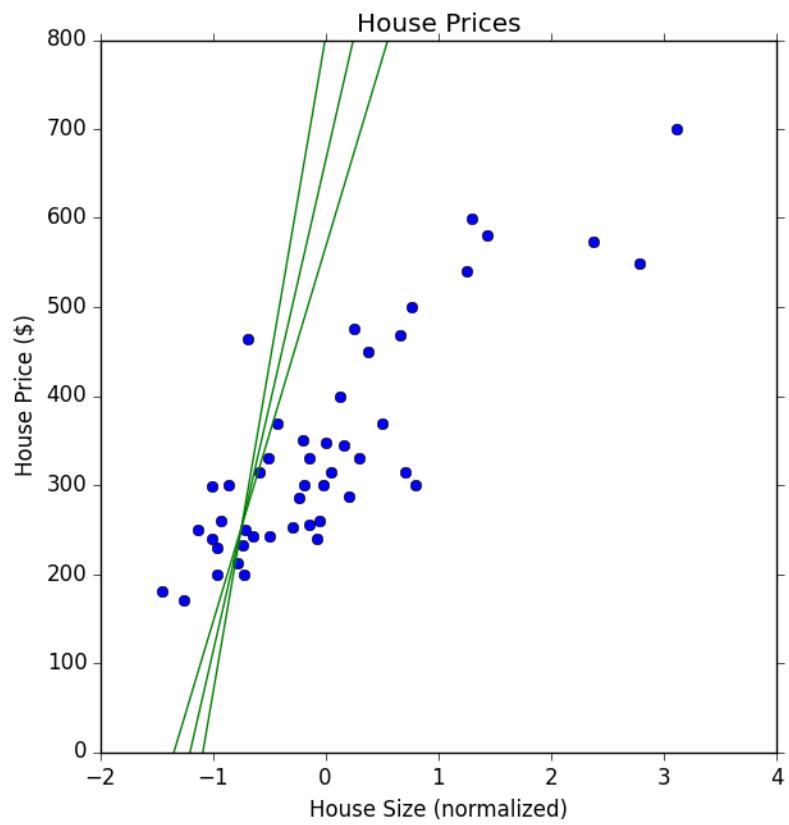
Step by Step



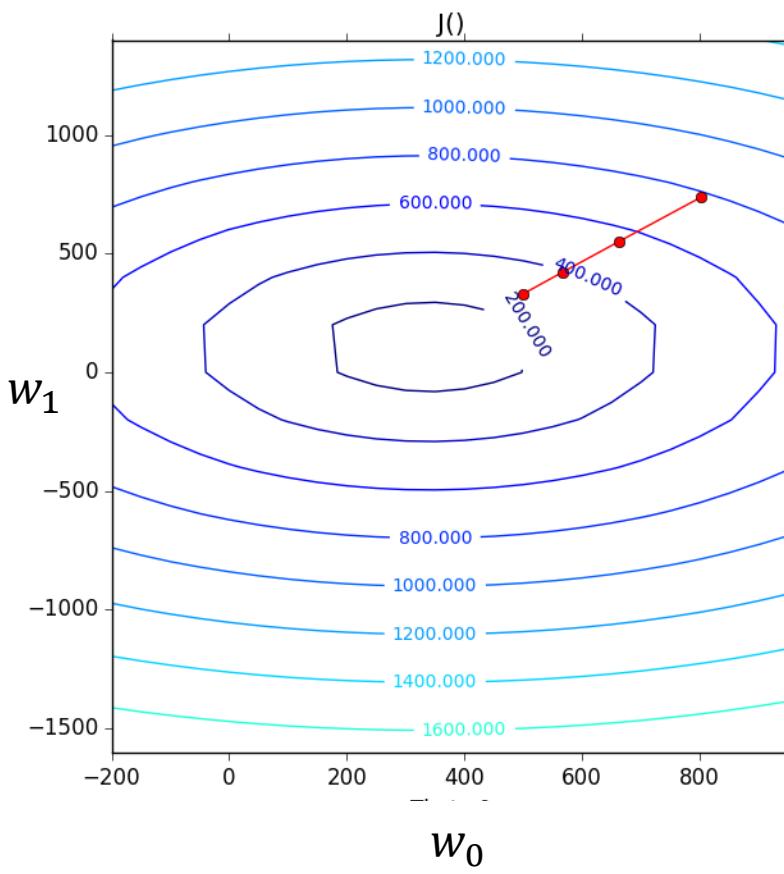
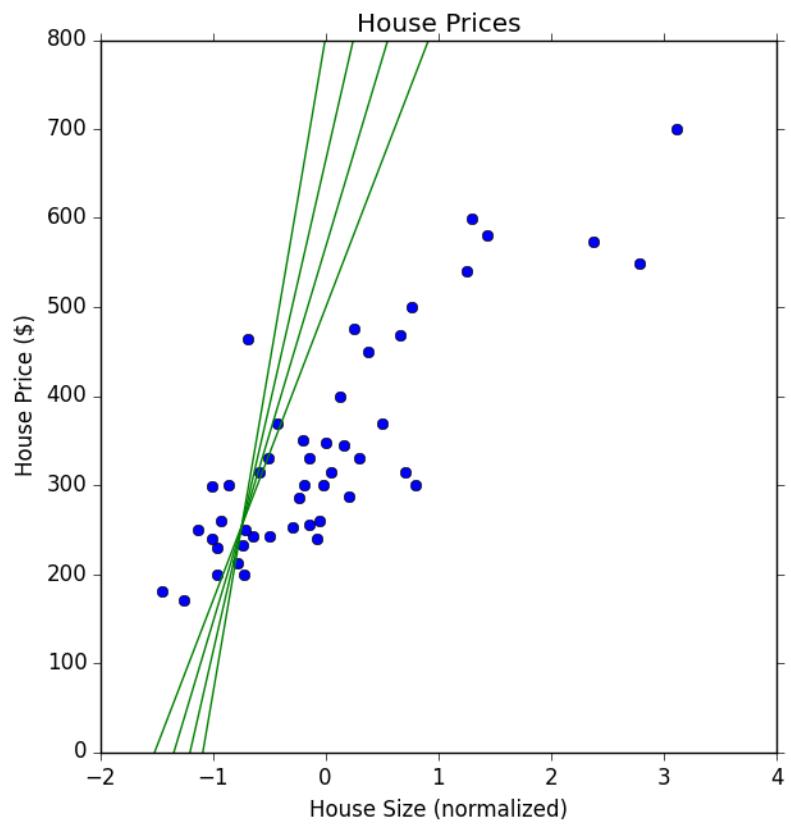
Step by Step



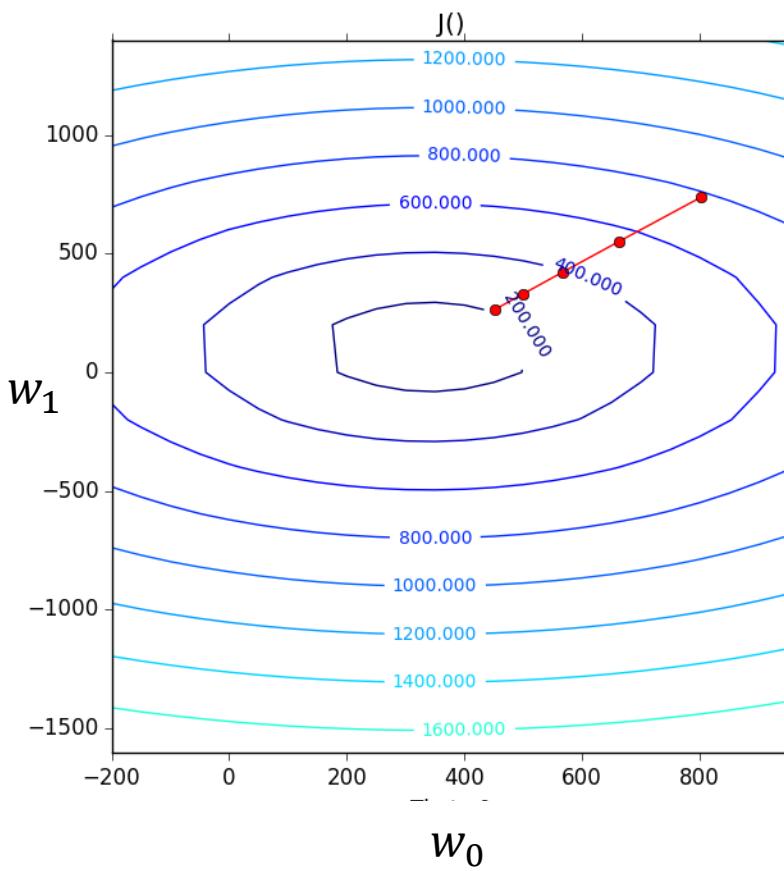
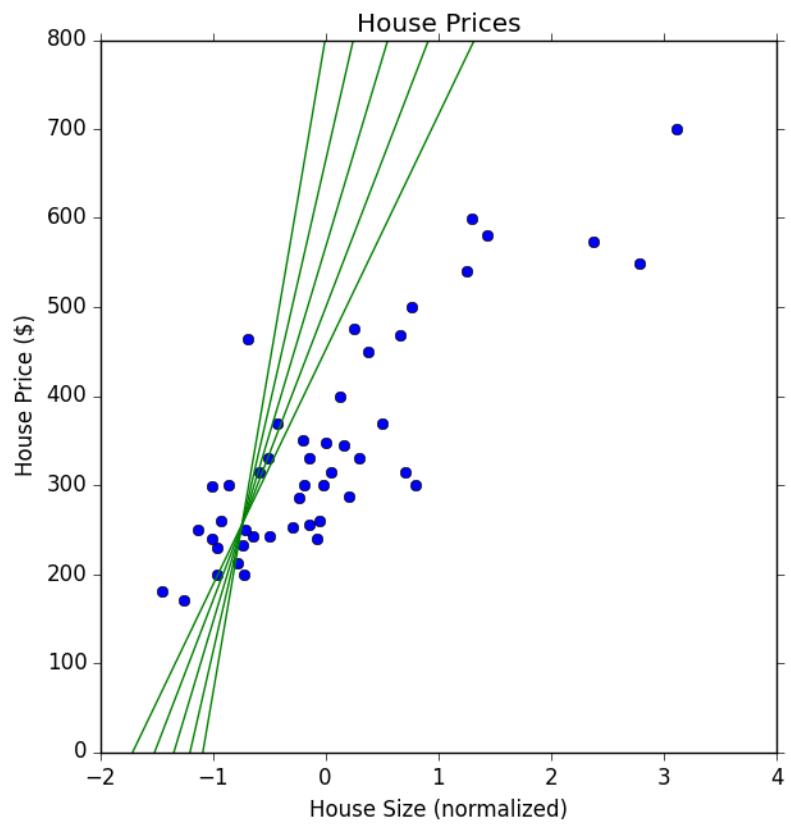
Step by Step



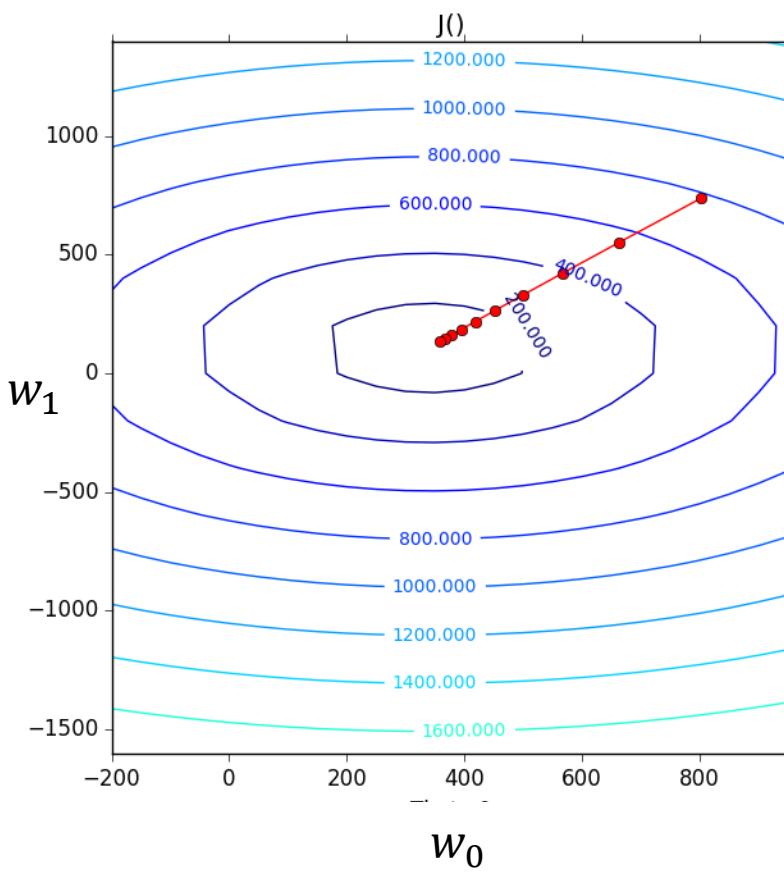
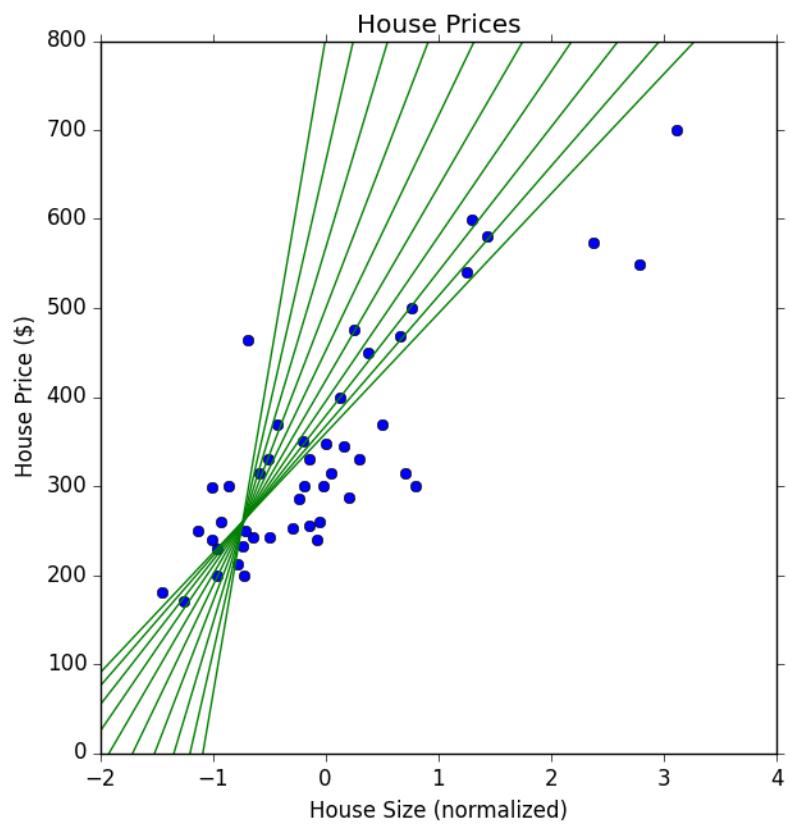
Step by Step



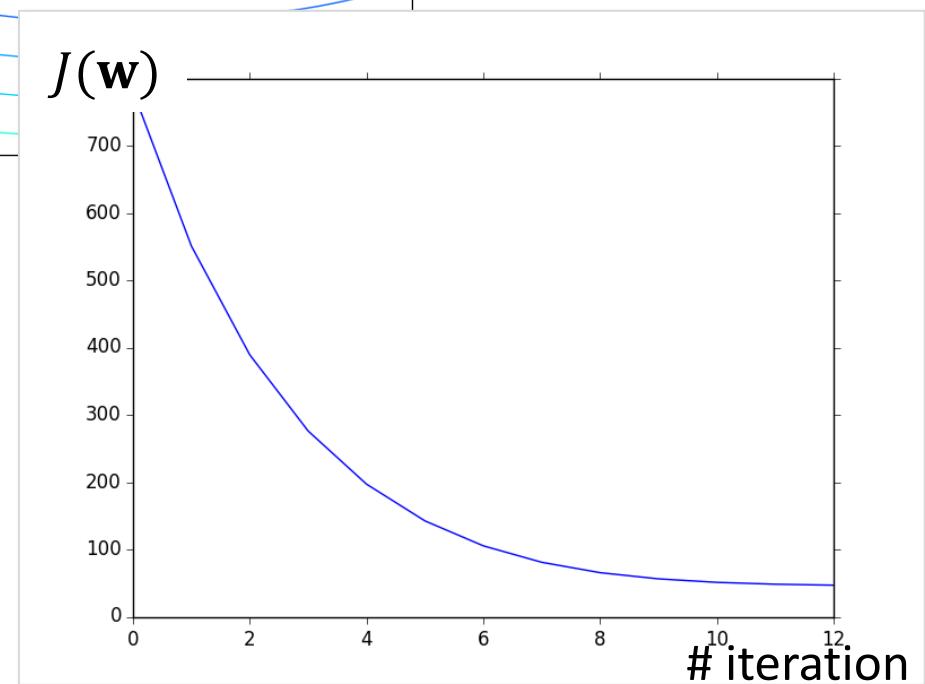
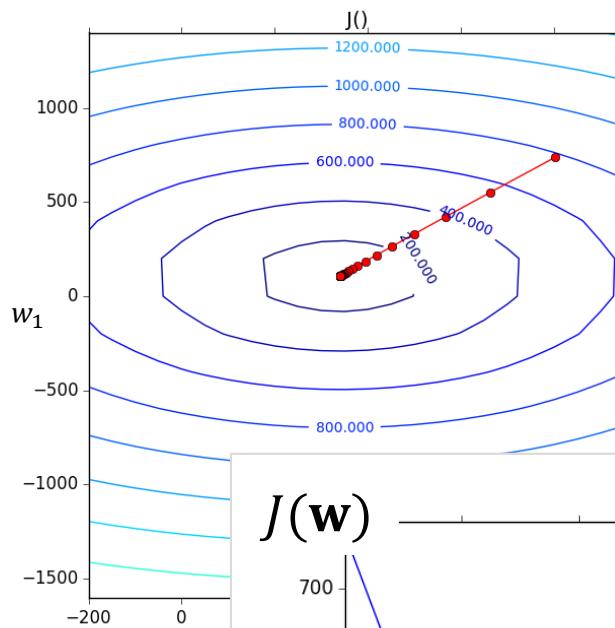
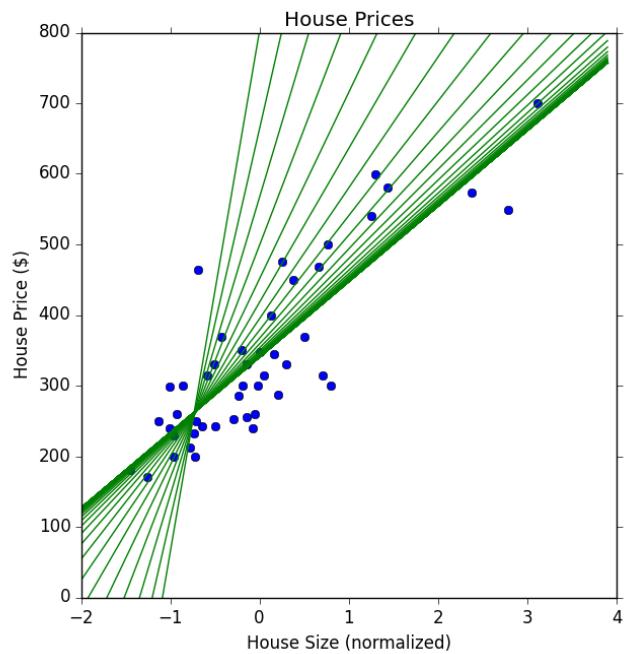
Step by Step



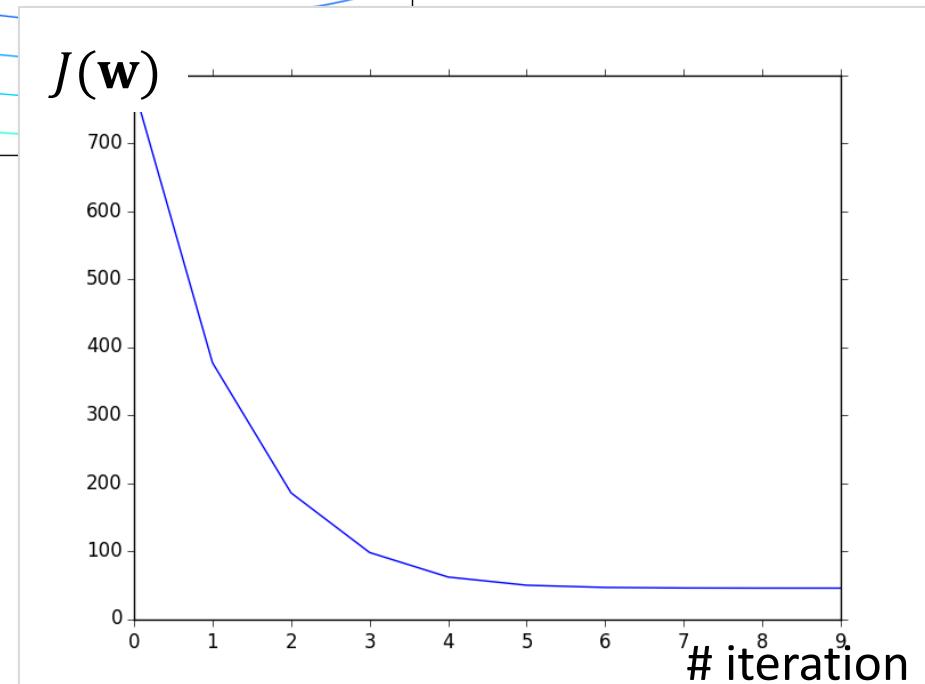
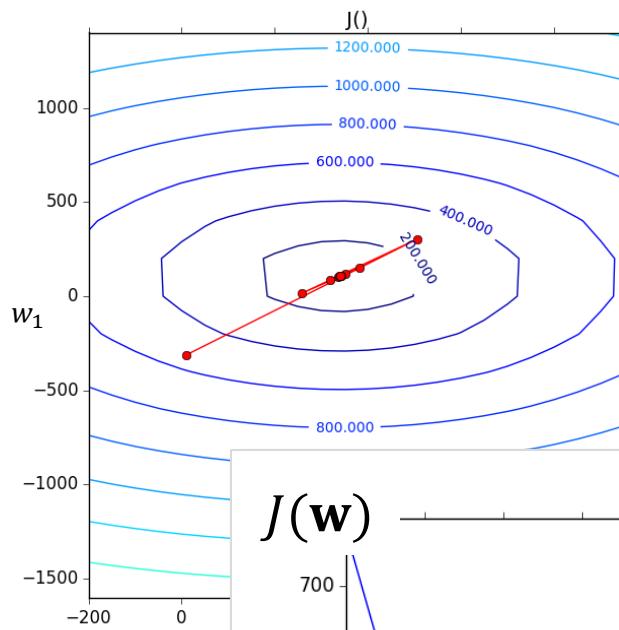
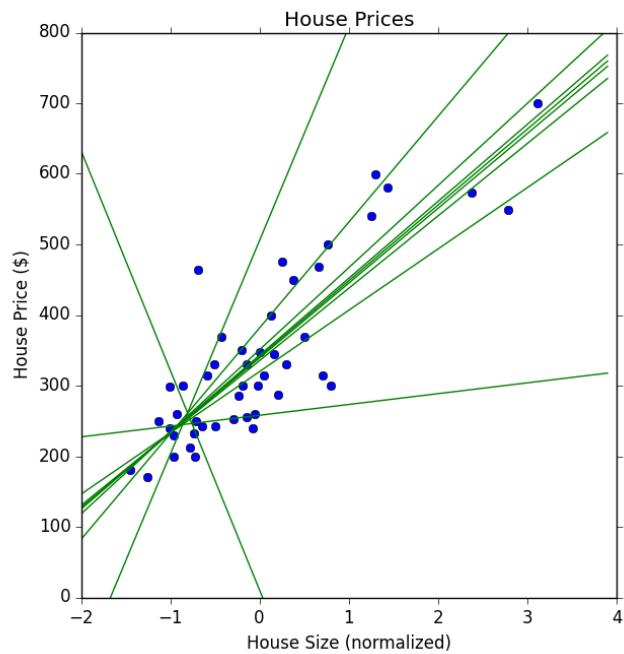
Step by Step



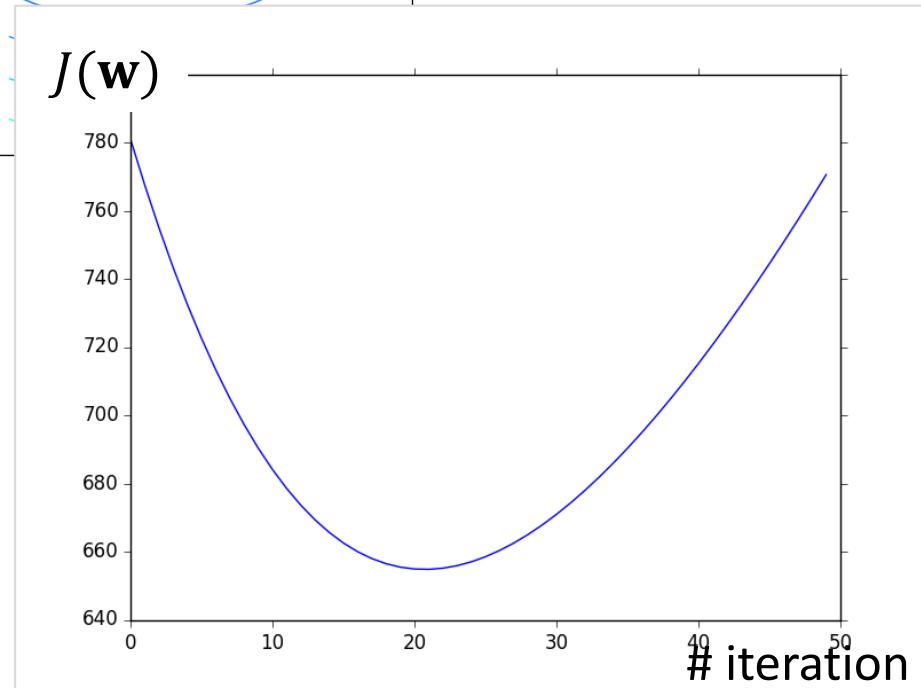
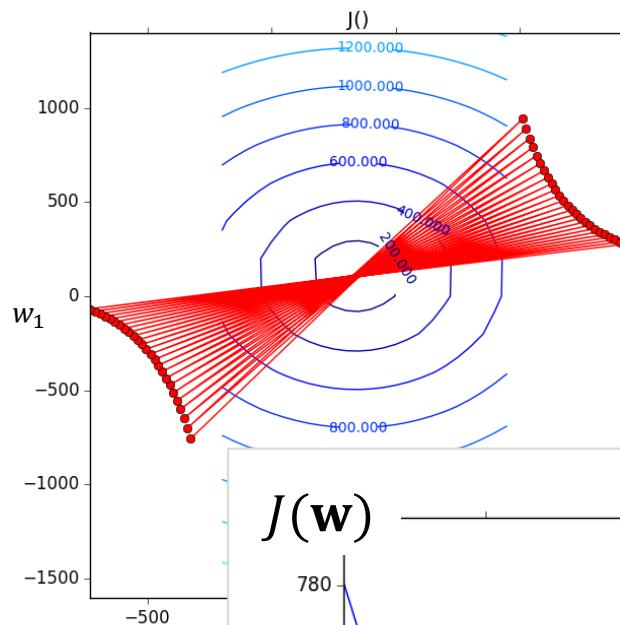
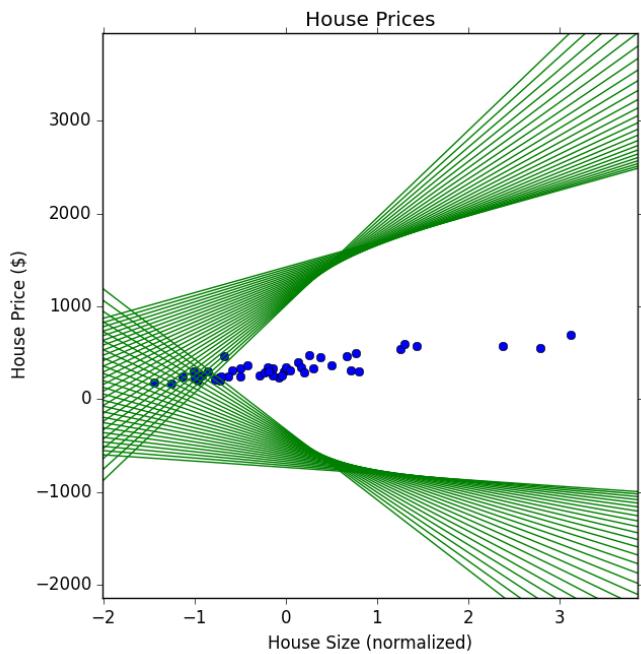
A look at the cost function



A look at the cost function



A look at the cost function



MULTIPLE REGRESSION

One Feature Scenario

| Size in feet ² (x) | Price (\$) in 1000's (y) |
|-----------------------------------|------------------------------|
| 2104 | 460 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ... | ... |

m

- m number of training examples
 x Input variable / features
 y Output variable / target variable
 (x, y) one training example
 $(x^{(i)}, y^{(i)})$ i^{th} training example

Multiple Feature Scenario

| Size (feet ²) (x_1) | Number of bedrooms (x_2) | Number of floors (x_3) | Age of home (years) (x_4) | Price (\$1000) (y) |
|---|------------------------------------|----------------------------------|-------------------------------------|------------------------------|
| 2104 | 5 | 1 | 45 | 460 |
| 1416 | 3 | 2 | 40 | 232 |
| 1534 | 3 | 2 | 30 | 315 |
| ... | ... | ... | ... | ... |

m

m

number of training examples

n

number of features

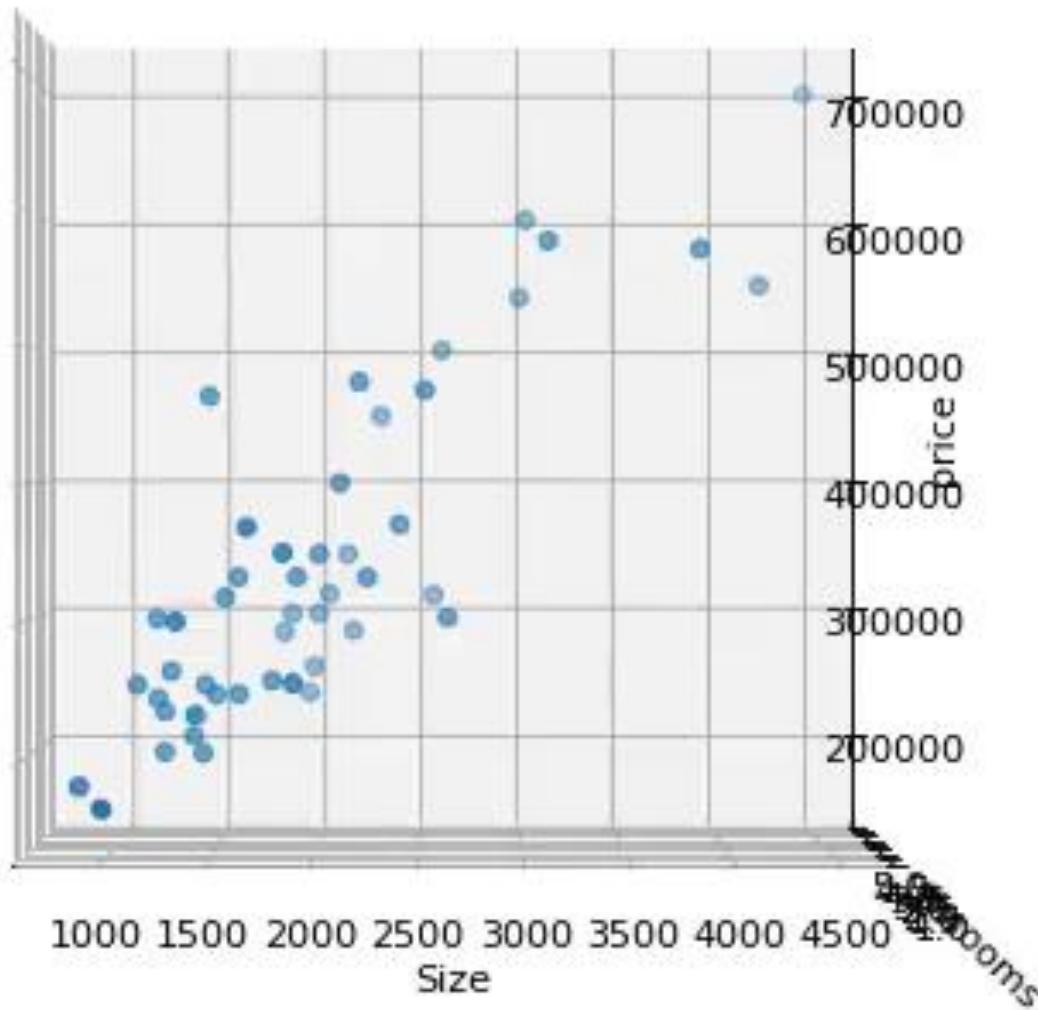
$x_j^{(i)}$

Value of feature j in i^{th} training example

$y^{(i)}$

Output variable / target variable

Multiple Feature Scenario



Hypothesis Representation

We represent f as a linear function
of **multiple** variables:

$$f_{\mathbf{w}}(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

For convenience of notation we
introduce $x_0 = 1$:

$$f_{\mathbf{w}}(x) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Multiple Variables Hypothesis

$$f_{\mathbf{w}}(x) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \mathbf{x}^T \mathbf{w}$$

where:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} \quad \left. \right\} n+1$$
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad \left. \right\} n+1$$

Multiple features

| (x_0) | Size (feet ²) (x_1) | Number of bedrooms (x_2) | Number of floors (x_3) | Age of home (years) (x_4) | Price (\$1000) (y) |
|---------|--------------------------------------|----------------------------------|--------------------------------|-----------------------------------|----------------------------|
| 1 | 2104 | 5 | 1 | 45 | 460 |
| 1 | 1416 | 3 | 2 | 40 | 232 |
| 1 | 1534 | 3 | 2 | 30 | 315 |
| ... | ... | ... | ... | ... | ... |

\brace{m}

$\brace{n+1}$

Design matrix

$$\mathbf{X} = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \brace{m}$$

$$\mathbf{y} = \begin{bmatrix} 460 \\ 232 \\ 315 \\ \vdots \end{bmatrix} \brace{m}$$

$\brace{n+1}$

$\brace{1}$

Linear Regression Revisited

$n=1$

$n \geq 1$

Hypothesis:

$$f_w(x) = w_0 + w_1 x$$

$$f_w(\mathbf{x}) = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

Parameters:

$$w_0, w_1$$

$$w_0, w_1, \dots, w_n$$

Cost Function:

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2$$

$$J(w_0, w_1, \dots, w_n) = \frac{1}{2m} \sum_{i=1}^m (f_w(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{w_0, w_1}{\text{minimise}}(J(w_0, w_1))$$

$$\underset{w_0, w_1, \dots, w_n}{\text{minimise}}(J(w_0, w_1, \dots, w_n))$$

Linear Regression Revisited

$n=1$

$n \geq 1$

Hypothesis:

$$f_w(x) = w_0 + w_1 x$$

$$f_w(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

Parameters:

$$w_0, w_1$$

$$\mathbf{w}$$

Cost Function:

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2$$

$$J(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^m (f_w(\mathbf{x}^{(i)}) - y^{(i)})^2$$

Goal:

$$\underset{w_0, w_1}{\text{minimise}}(J(w_0, w_1))$$

$$\underset{\mathbf{w}}{\text{minimise}}(J(\mathbf{w}))$$

Gradient Descent Revisited

$n=1$

Repeat until convergence

{

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$x^{(i)} \equiv x_1^{(i)}$$

(simultaneously for $j=0$ and $j=1$)

}

$n \geq 1$

Repeat until convergence

{

$$x_0^{(i)} = 1$$

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w(\mathbf{x}^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w(\mathbf{x}^{(i)}) - y^{(i)}) x_1^{(i)}$$

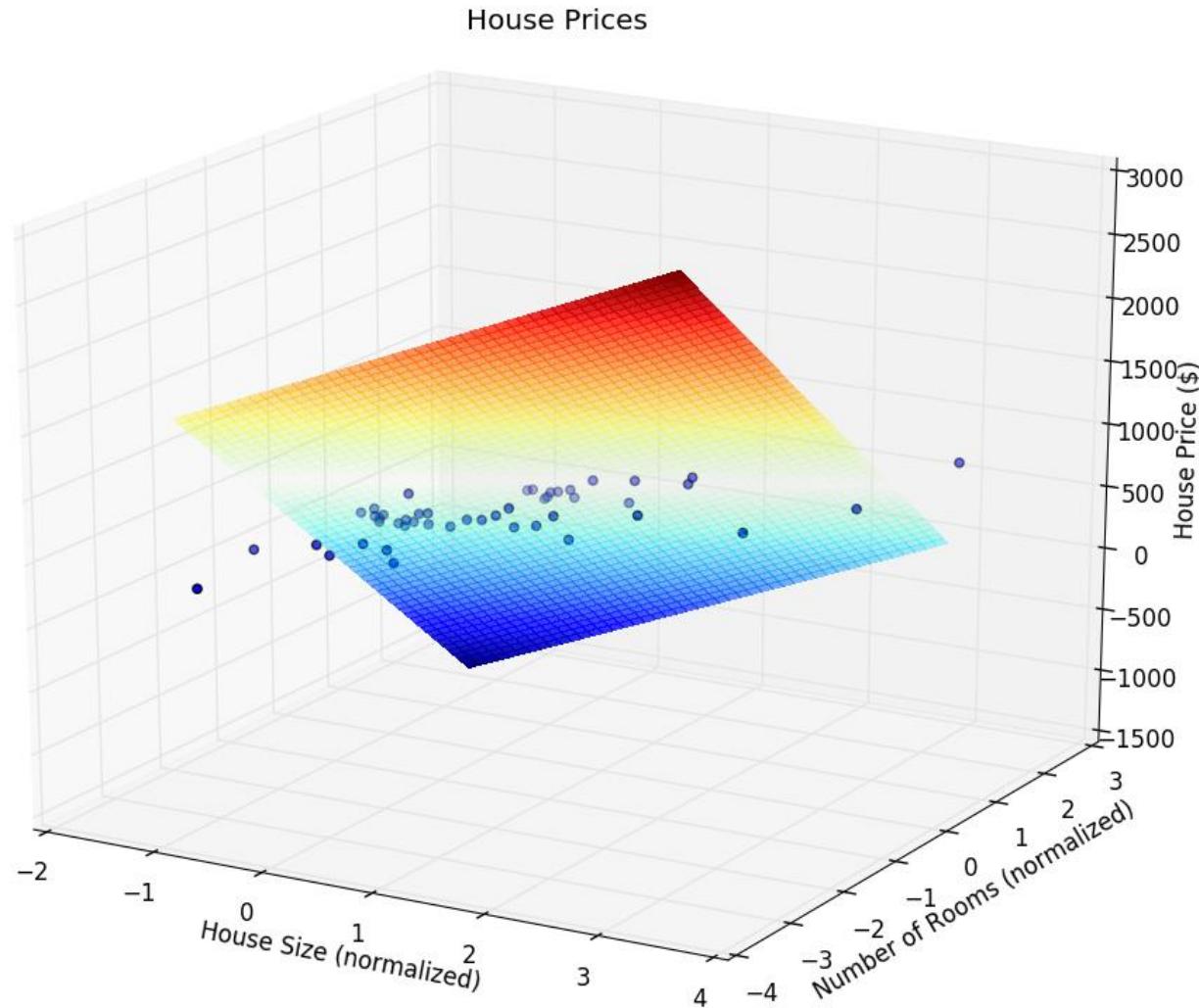
:

$$w_n := w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_w(\mathbf{x}^{(i)}) - y^{(i)}) x_n^{(i)}$$

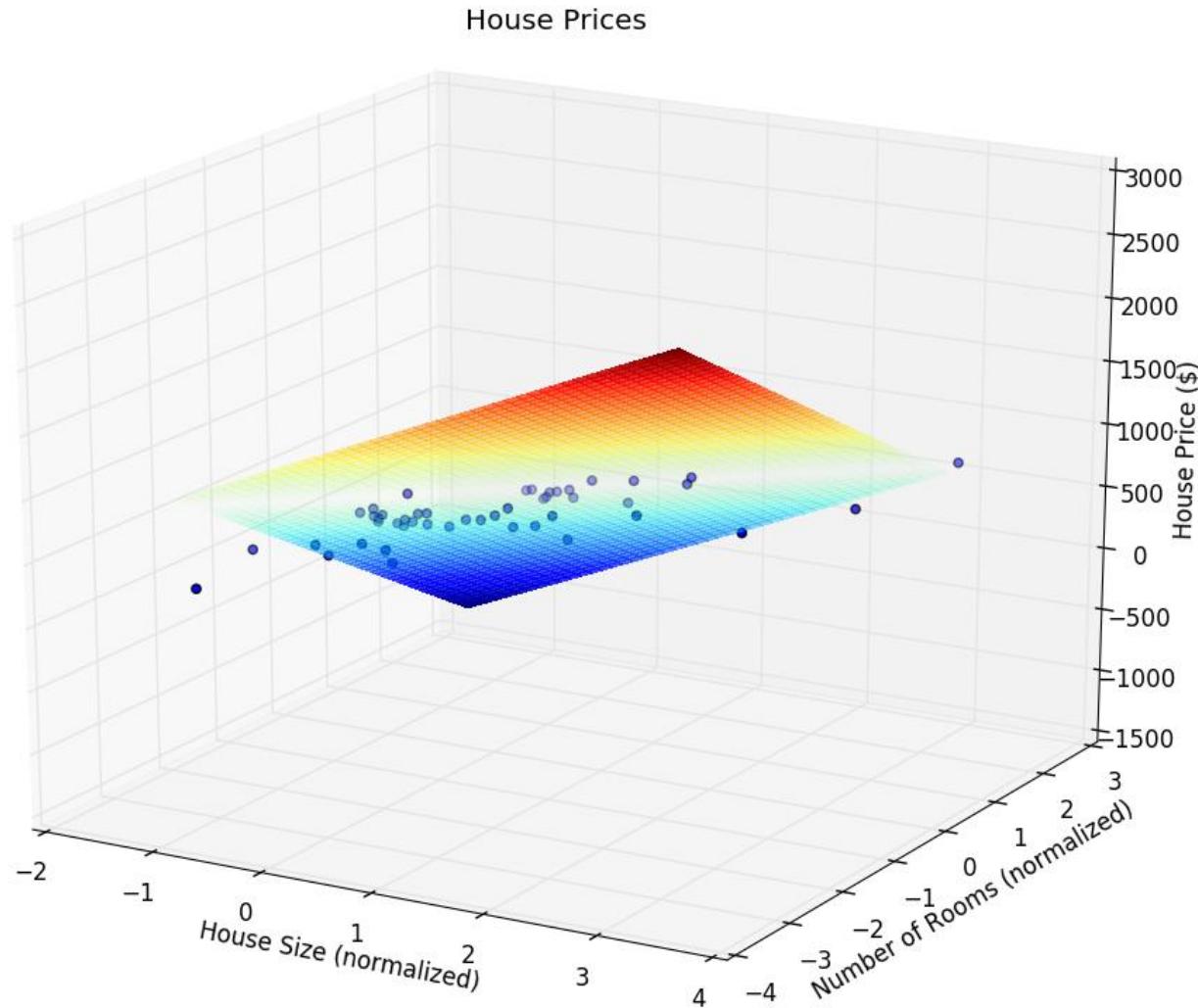
(simultaneously for all j)

}

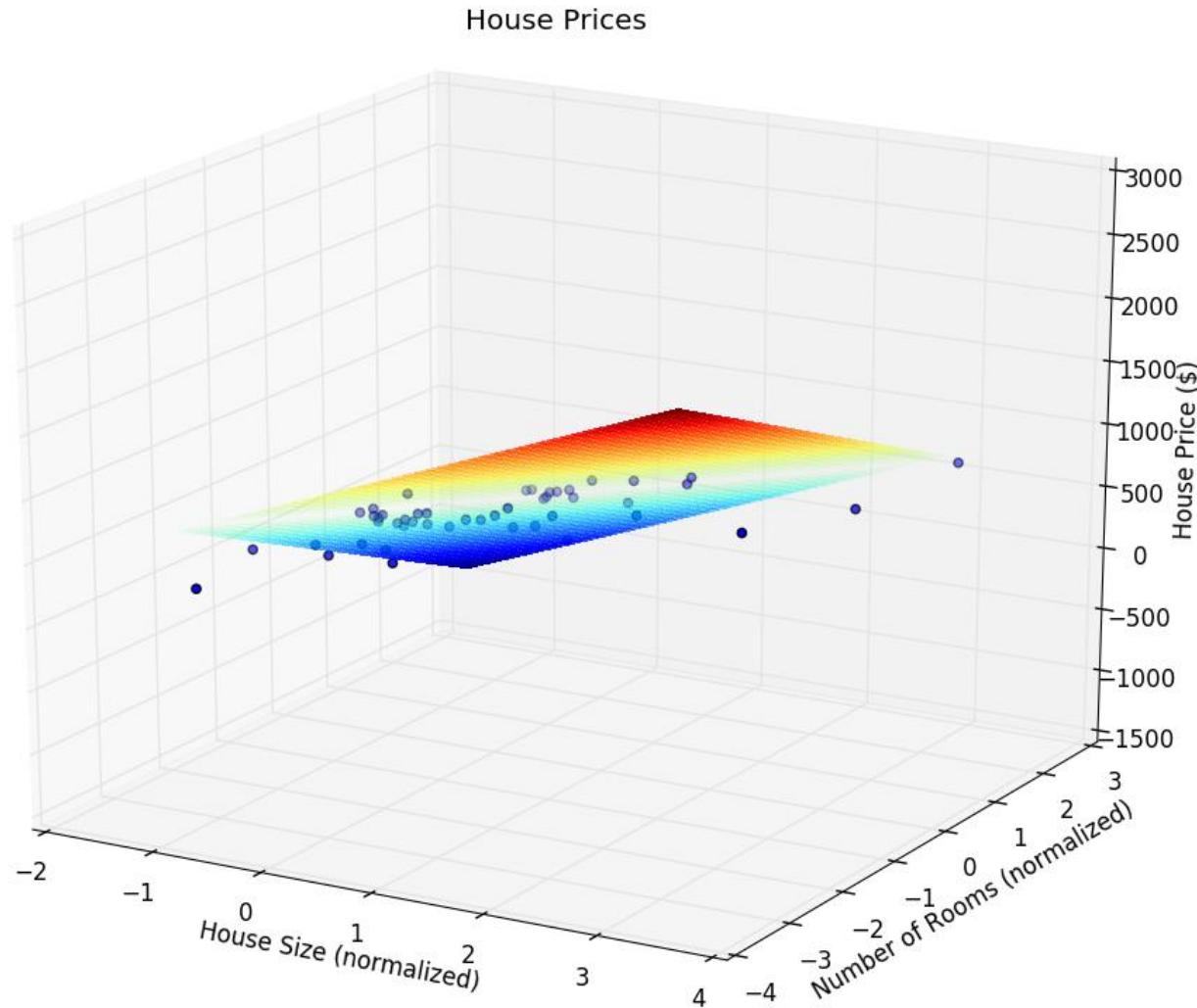
Multiple Regression Example



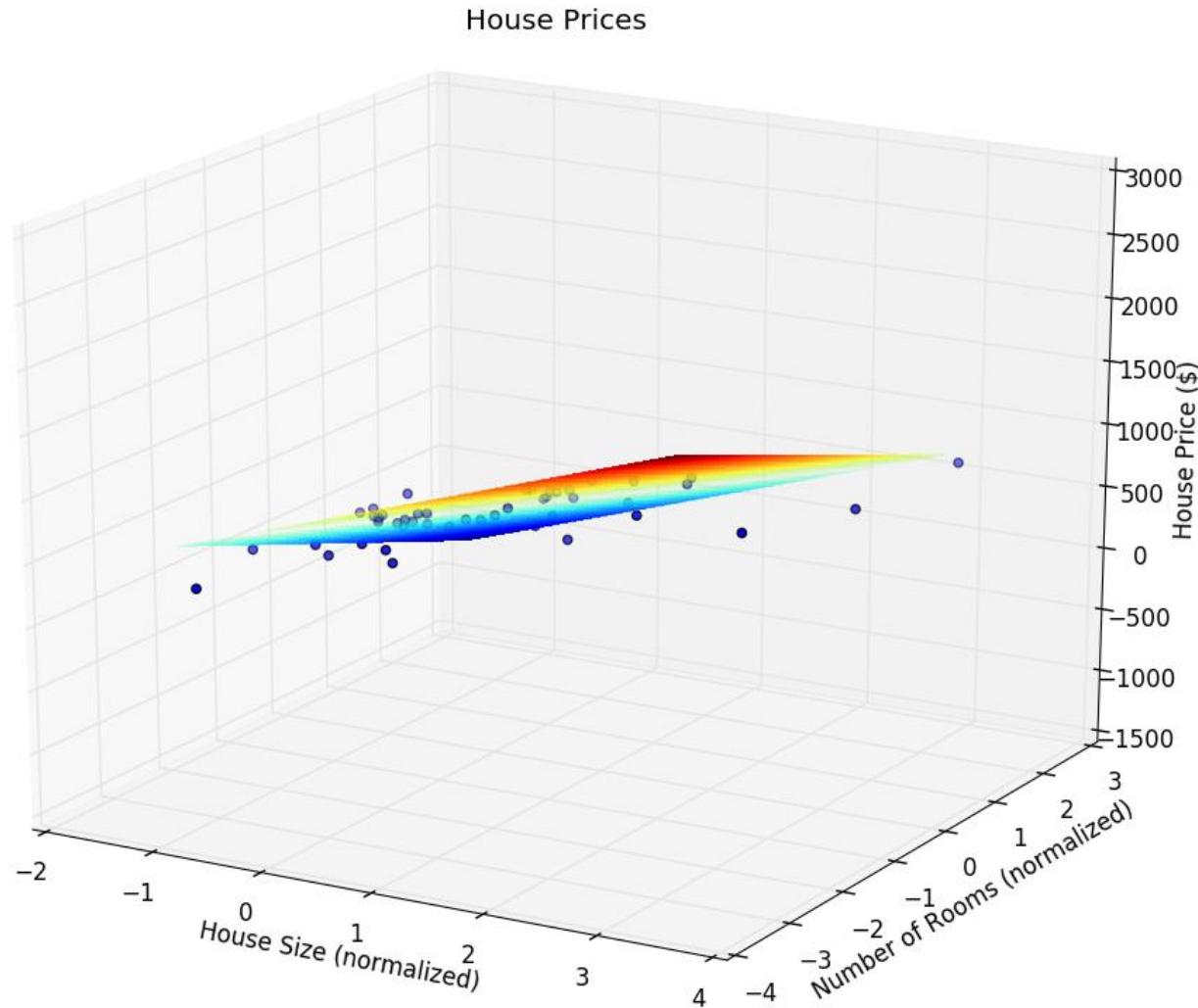
Multiple Regression Example



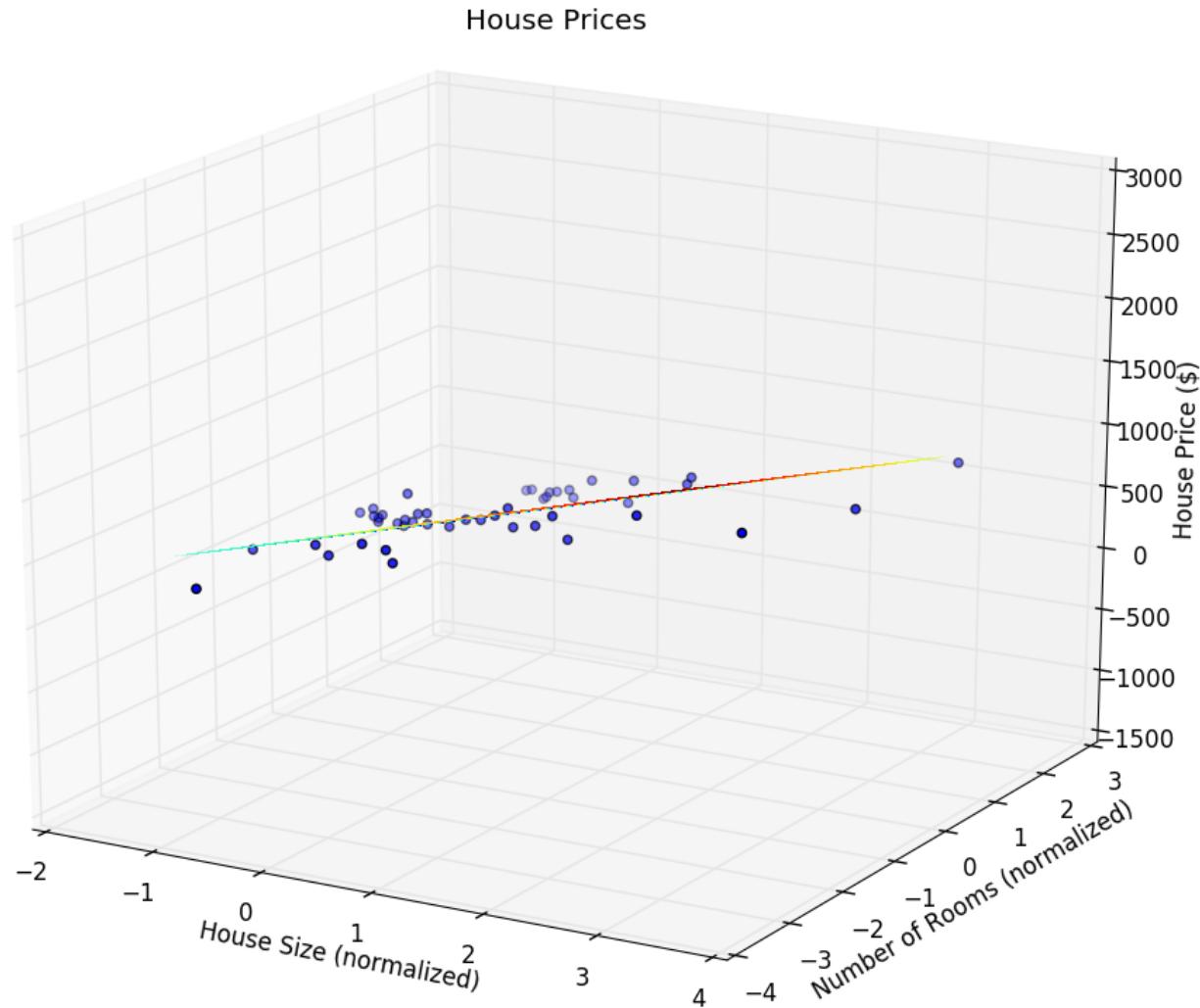
Multiple Regression Example



Multiple Regression Example



Multiple Regression Example



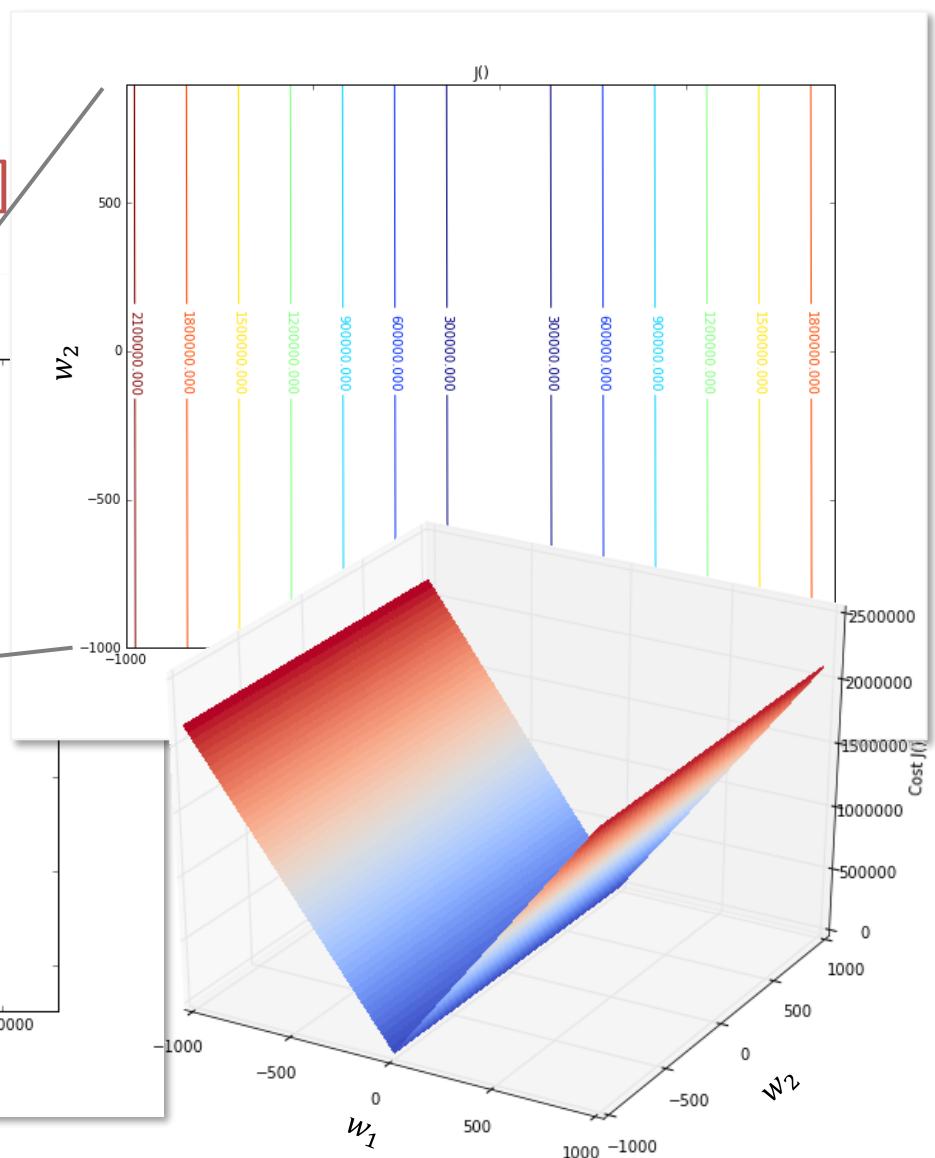
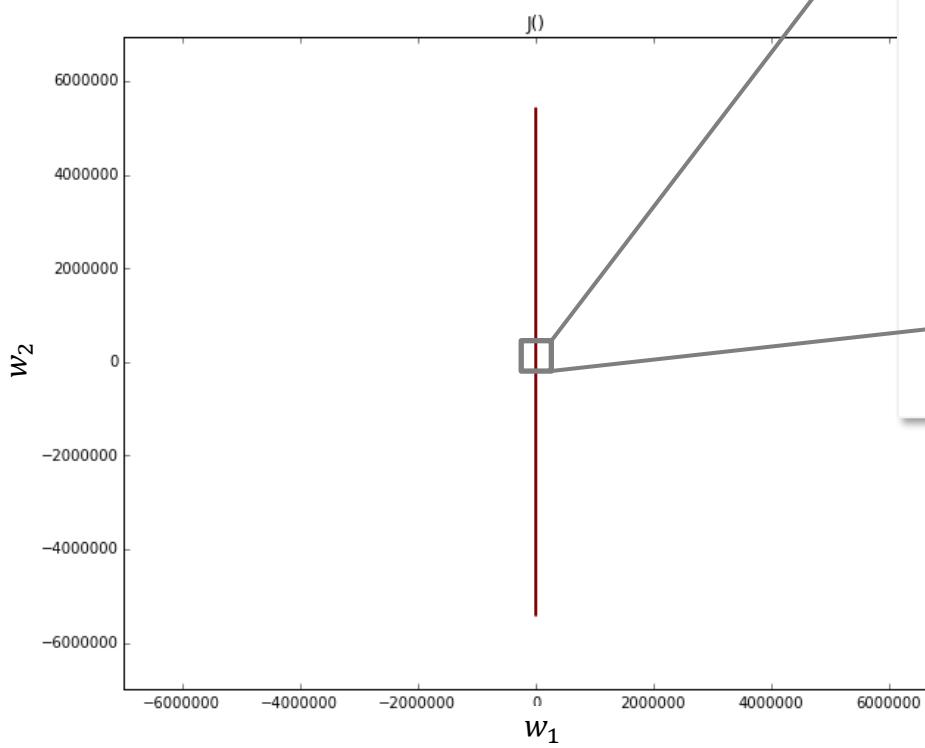
FEATURE SCALING

Feature ranges – the problem

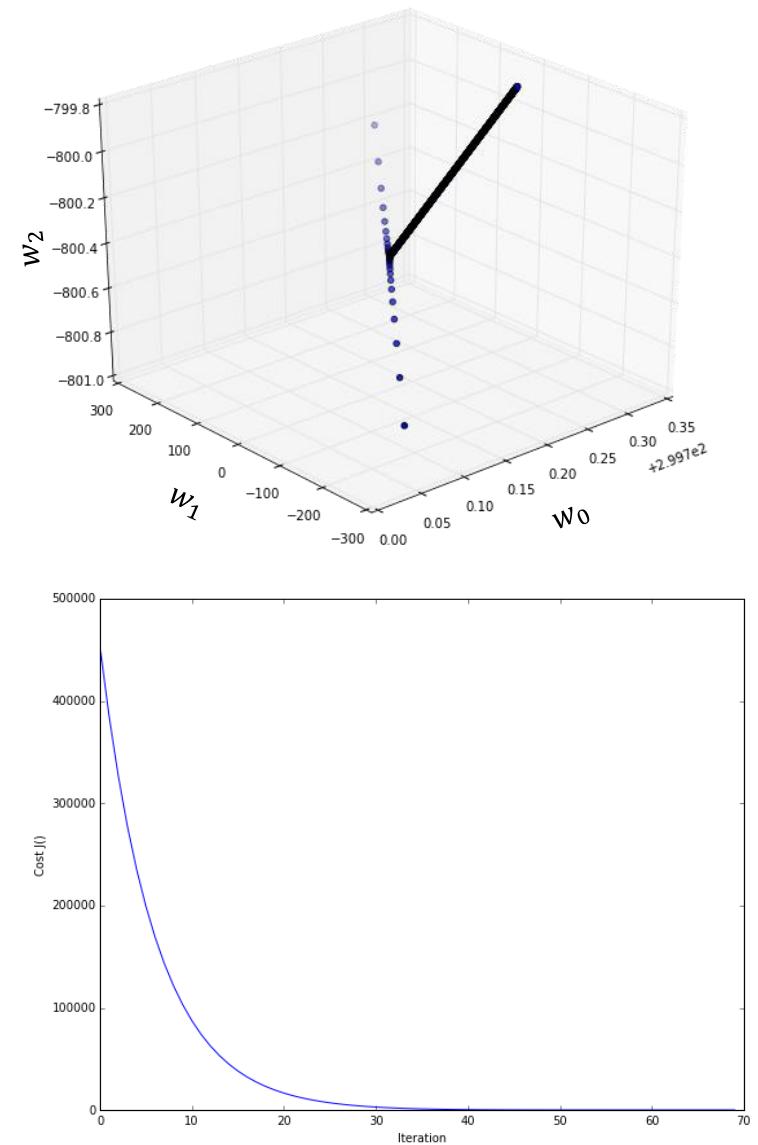
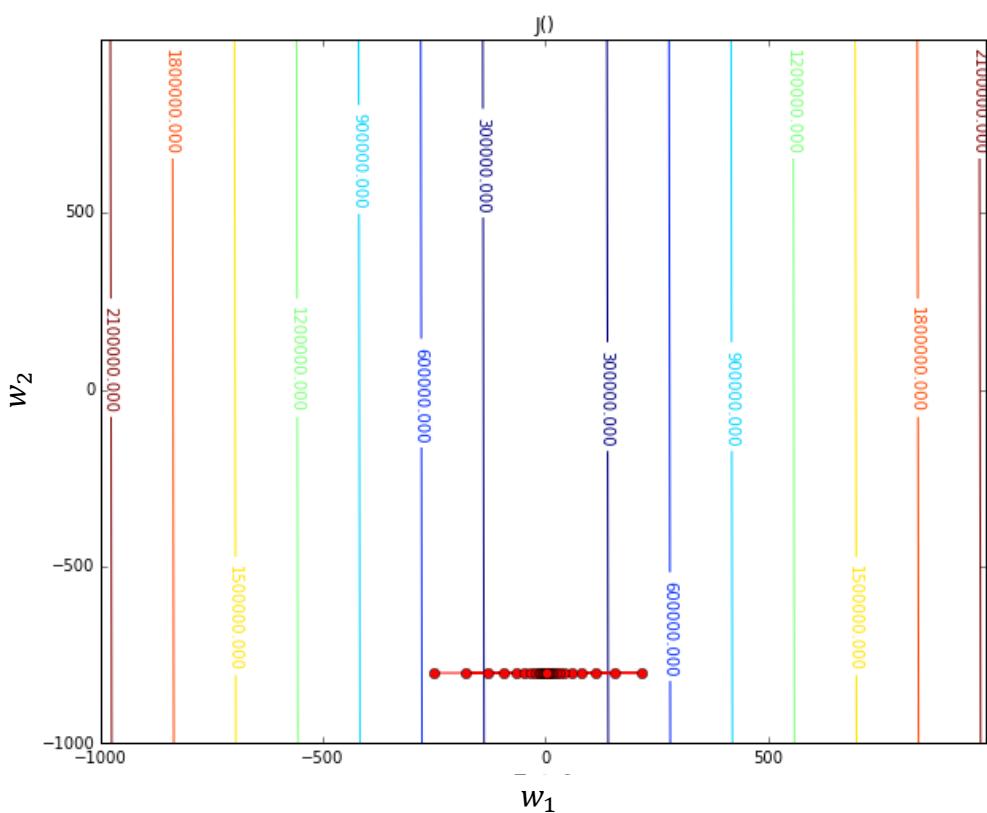
$$f_w(x) = w_0x_0 + w_1x_1 + w_2x_2$$

x_1 = house size (feet²) [800-5000]

x_2 = # bedrooms [1-5]



Feature ranges – the problem



Standardisation

Aim: get every feature x_j into approximately a $-1 \leq x_j \leq 1$ range

Mean normalization:

- Subtract from each feature x_j the feature mean (μ_j) to make features have zero mean
- Divide by the standard deviation (s_j)
- Do not apply to x_0 !!!

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$$

$$\mu_j = \bar{x}_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

$$s_j = \sqrt{\frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2}$$

Feature scaling – standardisation

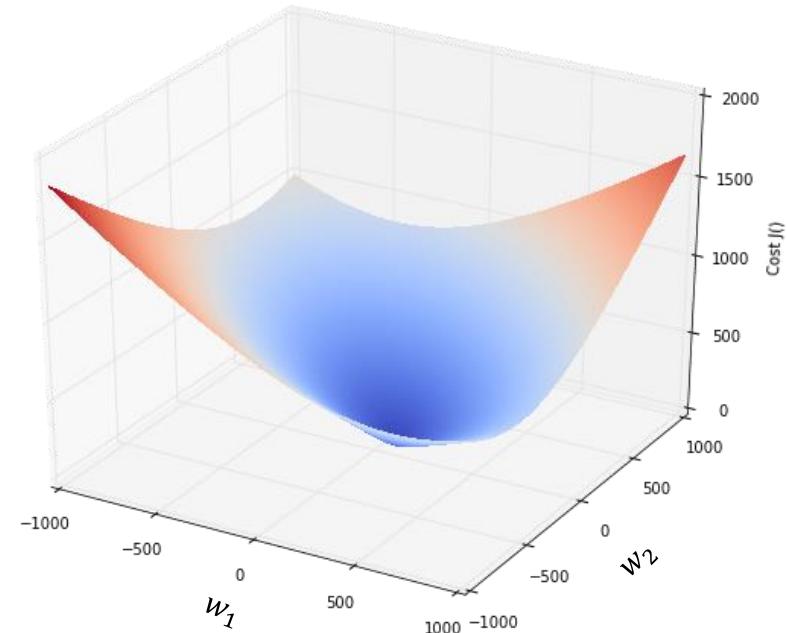
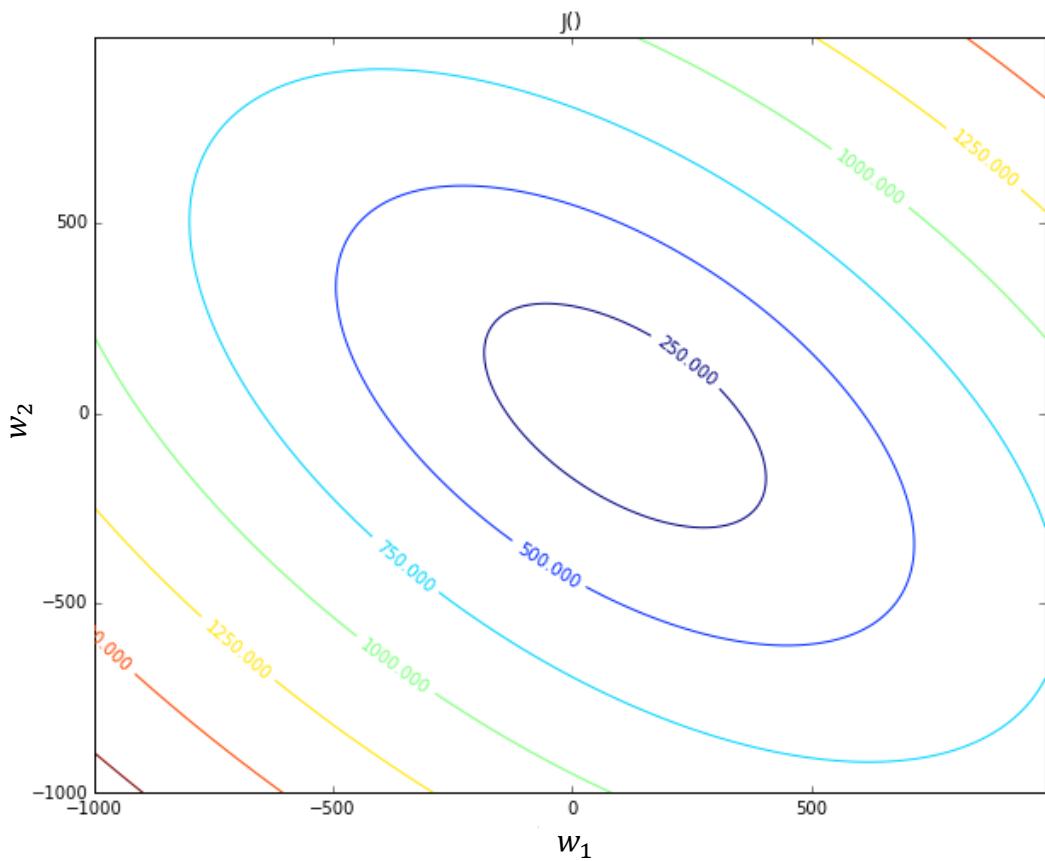
house size (feet²) [800, 5000] → [-1.44, 3.12]

$$x_1^{(i)} = \frac{x_1^{(i)} - \mu_1}{s_1} = \frac{x_1^{(i)} - 2000.68}{794.7}$$

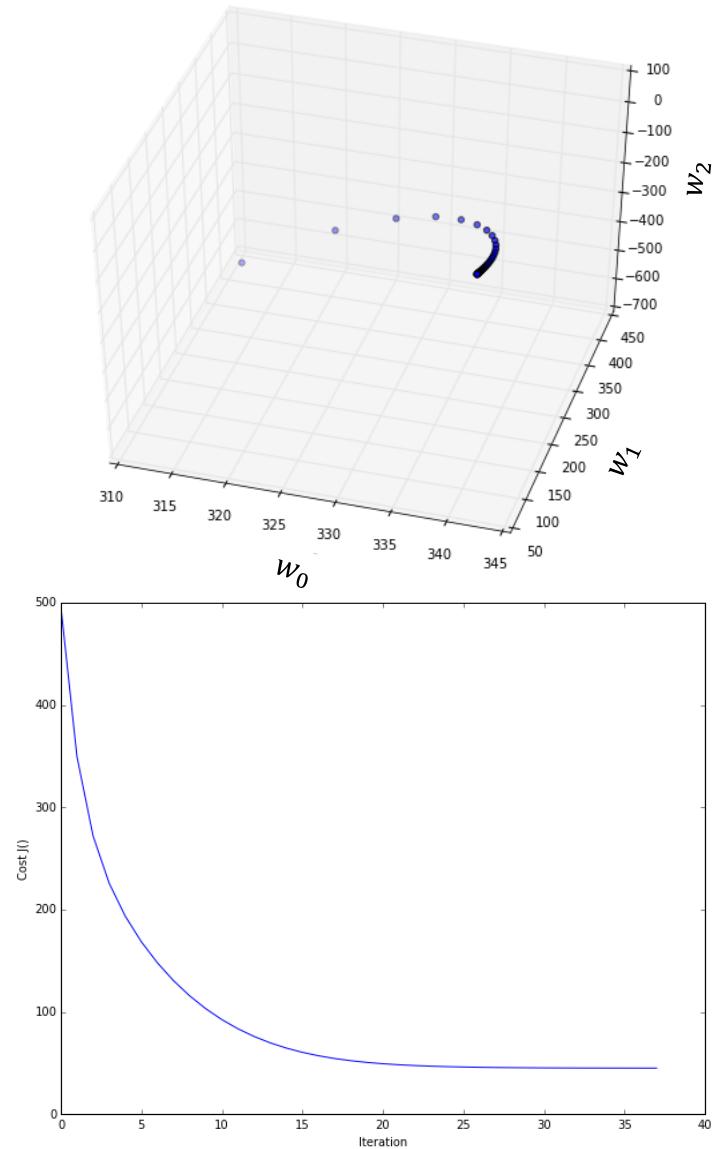
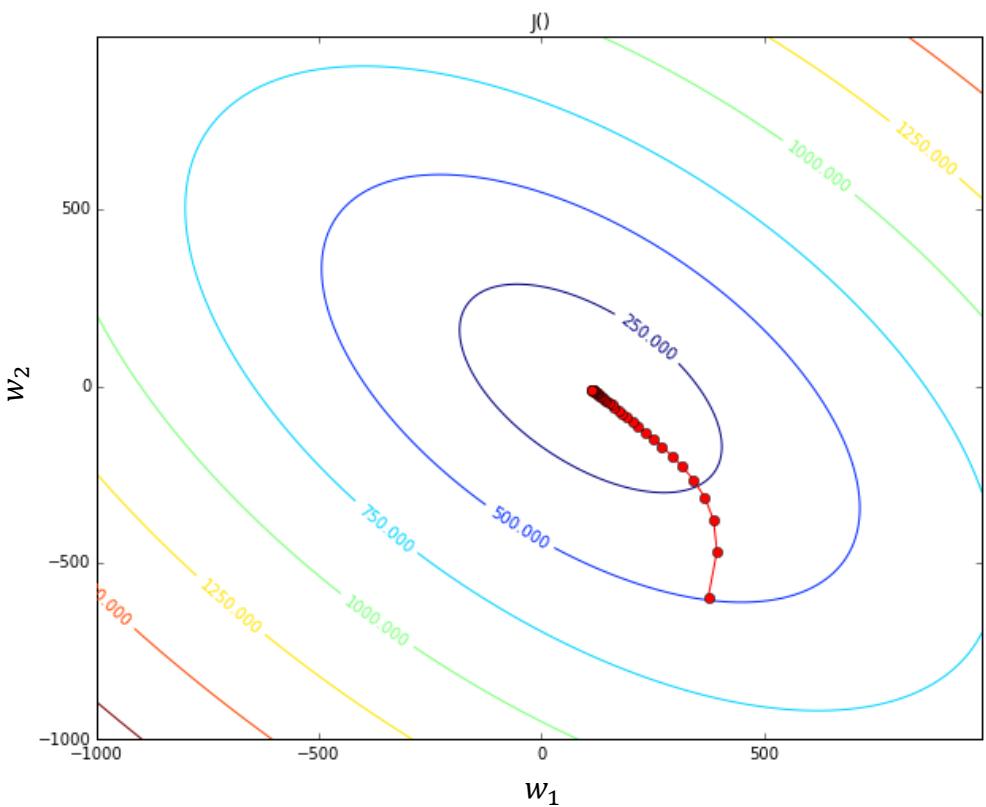
of bedrooms [1-5] → [-2.85, 2.40]

$$x_2^{(i)} = \frac{x_2^{(i)} - \mu_2}{s_2} = \frac{x_2^{(i)} - 3.17}{0.76}$$

Feature scaling – standardisation



Feature scaling – standardisation

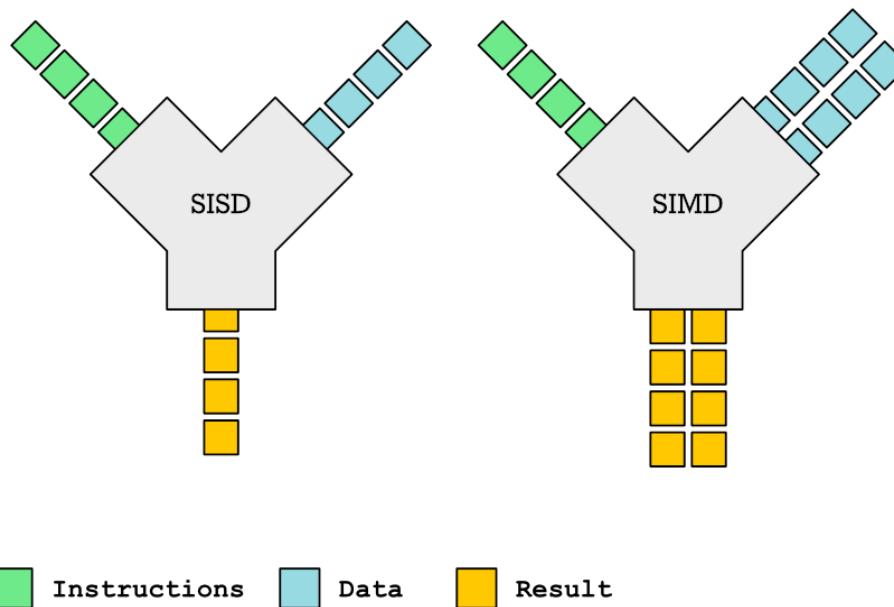


VECTORISING OPERATIONS

Parallel Processing

When possible, it is important to vectorise data, and take advantage of the SIMD (Single Instruction, Multiple Data) capabilities of CPUs and GPUs

E.g. NumPy is linked to specific low-level libraries (BLAS, LAPACK) that leverage the SIMD instructions of your processor



Vectorising the hypothesis function

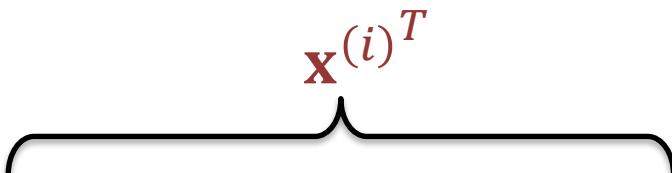
$$f_{\mathbf{w}}(\mathbf{x}) = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \mathbf{x}^T \mathbf{w}$$

where:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} \quad \brace{n+1}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad \brace{n+1}$$

Vectorising the hypothesis function

$$f_{\mathbf{w}}(\mathbf{x}^{(i)}) = \mathbf{x}^{(i)T} \mathbf{w} = [x_0^{(i)} \quad x_1^{(i)} \quad x_2^{(i)} \quad \dots \quad x_n^{(i)}] \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$


One operation - Multiple samples

$$f_{\mathbf{w}}(\mathbf{x}^{(i)}) = f_{\mathbf{w}}^{(i)} = \mathbf{x}^{(i)T} \mathbf{w} = [- \quad \mathbf{x}^{(i)} \quad -] \begin{bmatrix} | \\ \mathbf{w} \\ | \end{bmatrix}$$

$$\mathbf{f} = \begin{bmatrix} f_{\mathbf{w}}^{(1)} \\ f_{\mathbf{w}}^{(2)} \\ \vdots \\ f_{\mathbf{w}}^{(m)} \end{bmatrix} = \mathbf{X}\mathbf{w} = \begin{bmatrix} - & \mathbf{x}^{(1)} & - \\ - & \mathbf{x}^{(2)} & - \\ \vdots & & \\ - & \mathbf{x}^{(m)} & - \end{bmatrix} \begin{bmatrix} | \\ \mathbf{w} \\ | \end{bmatrix}$$

Design matrix
(#samples \times #features)



Gradient Descent Revisited

Repeat until convergence

{

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)}$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_1^{(i)}$$

⋮

$$w_n := w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_n^{(i)}$$

(simultaneously for all weights)

}

Gradient descent – one weight

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)}$$

$$\sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)} = (f_w^{(1)} - y^{(1)}) x_0^{(1)} + (f_w^{(2)} - y^{(2)}) x_0^{(2)} + \dots + (f_w^{(m)} - y^{(m)}) x_0^{(m)}$$

$$\sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)} = [x_0^{(1)} \quad x_0^{(2)} \quad \dots \quad x_0^{(m)}] \begin{bmatrix} f_w^{(1)} - y^{(1)} \\ f_w^{(2)} - y^{(2)} \\ \vdots \\ f_w^{(m)} - y^{(m)} \end{bmatrix}$$

Gradient Descent – Multiple weights

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)}$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_1^{(i)}$$

⋮

$$w_n := w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_n^{(i)}$$

$$\begin{bmatrix} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)} \\ \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_n^{(i)} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} f_w^{(1)} - y^{(1)} \\ f_w^{(2)} - y^{(2)} \\ \vdots \\ f_w^{(m)} - y^{(m)} \end{bmatrix} = \mathbf{X}^T (\mathbf{f} - \mathbf{y})$$

Design matrix transposed
 (#features × #samples)

Gradient Descent – Multiple weights

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)}$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_1^{(i)}$$

⋮

$$w_n := w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_n^{(i)}$$

$$\left[\begin{array}{c} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)} \\ \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_n^{(i)} \end{array} \right] = \left[\begin{array}{cccc} | & | & | & | \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \dots & \mathbf{x}^{(m)} \\ | & | & & | \end{array} \right] \left[\begin{array}{c} f_w^{(1)} - y^{(1)} \\ f_w^{(2)} - y^{(2)} \\ \vdots \\ f_w^{(m)} - y^{(m)} \end{array} \right] = \mathbf{X}^T (\mathbf{f} - \mathbf{y})$$

Design matrix transposed
(#features × #samples)

Gradient Descent in one line

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)}$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_1^{(i)}$$

⋮

$$w_n := w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_n^{(i)}$$

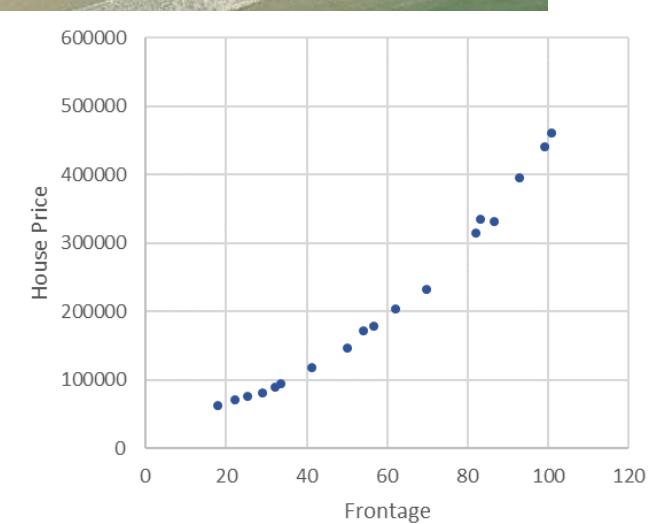
$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} := \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} - \alpha \frac{1}{m} \begin{bmatrix} \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_0^{(i)} \\ \vdots \\ \sum_{i=1}^m (f_w^{(i)} - y^{(i)}) x_1^{(i)} \end{bmatrix}$$

$$\mathbf{w} := \mathbf{w} - \alpha \frac{1}{m} \mathbf{X}^T (\mathbf{X} \mathbf{w} - \mathbf{y})$$

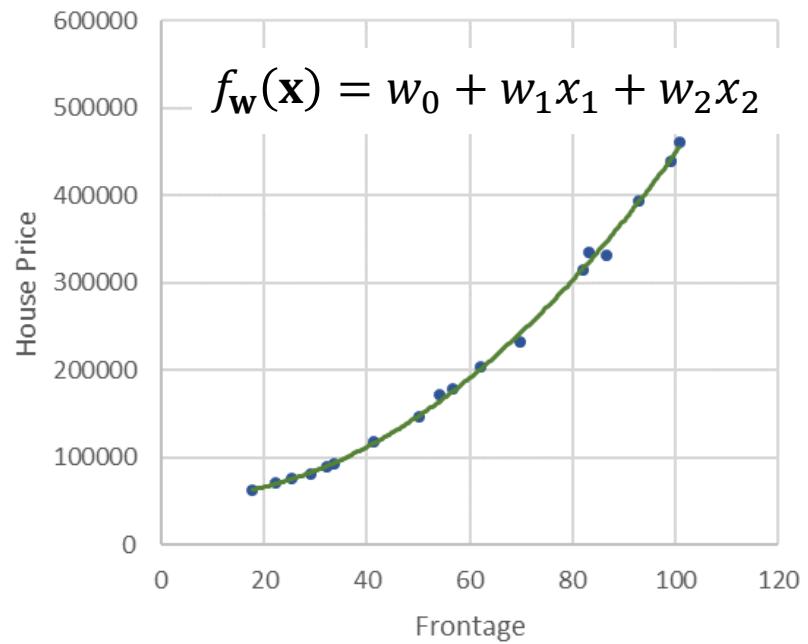
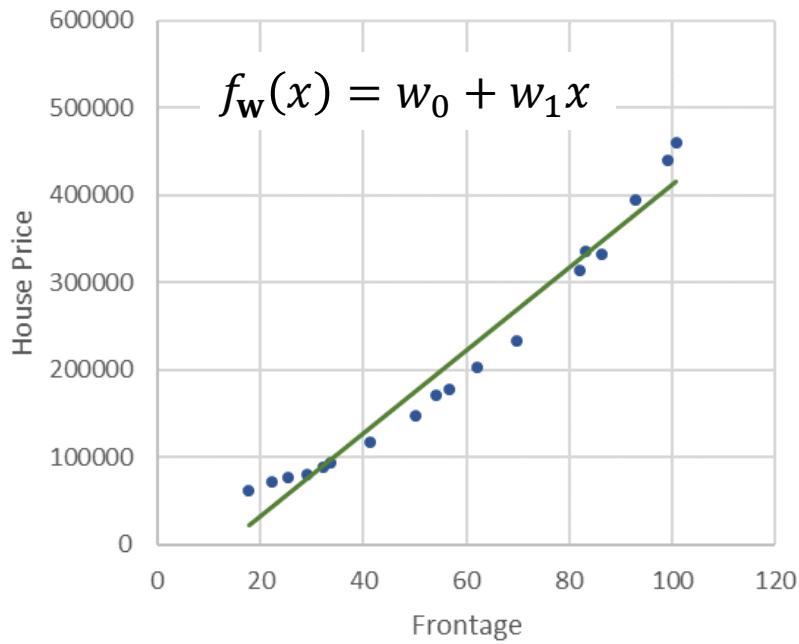
```
w = w - alpha * np.dot(X.T, (np.dot(X, w) - y)) / m
```

POLYNOMIAL REGRESSION

Selling parcels



Polynomial models for house prices

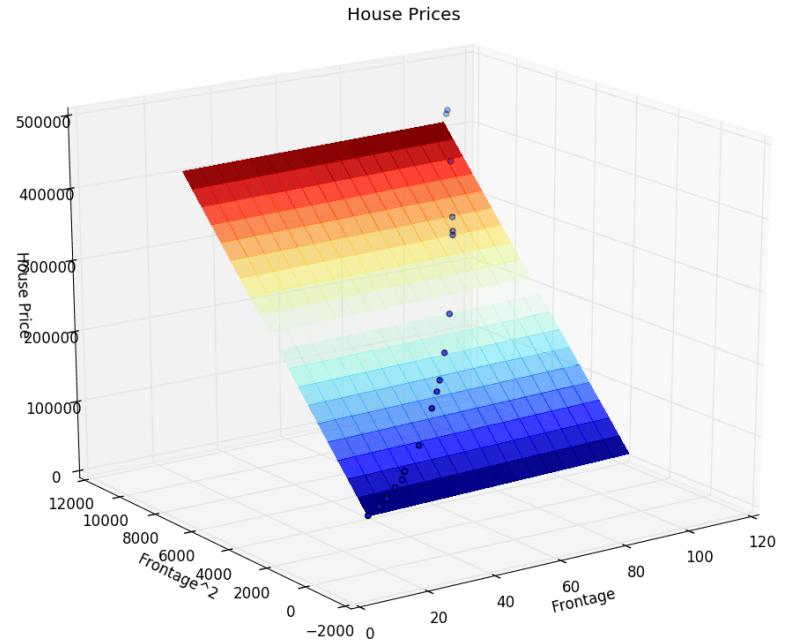
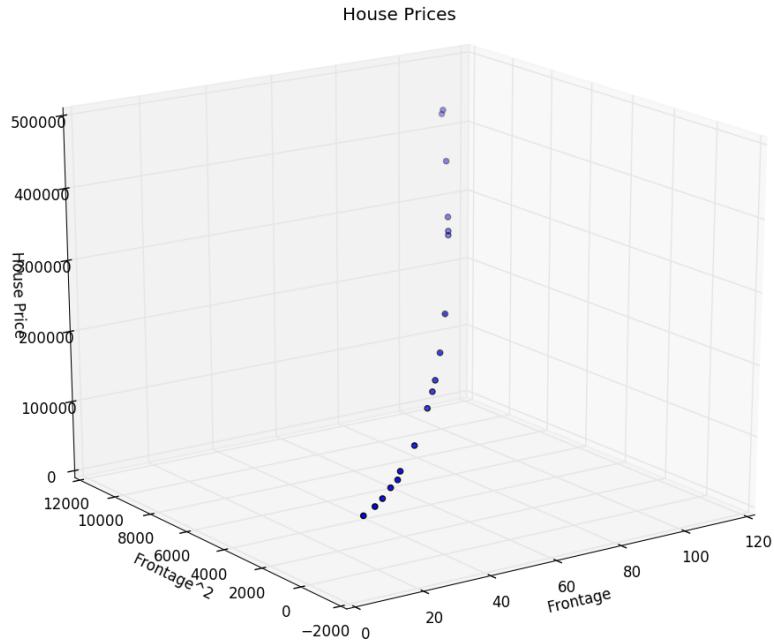


$$f_{\mathbf{w}}(\text{frontage}) = w_0 + w_1(\text{frontage}) + w_2(\text{frontage}^2)$$

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$$

$$\begin{aligned}x_1 &= \text{frontage} \\x_2 &= \text{frontage}^2\end{aligned}$$

Polynomial models for house prices

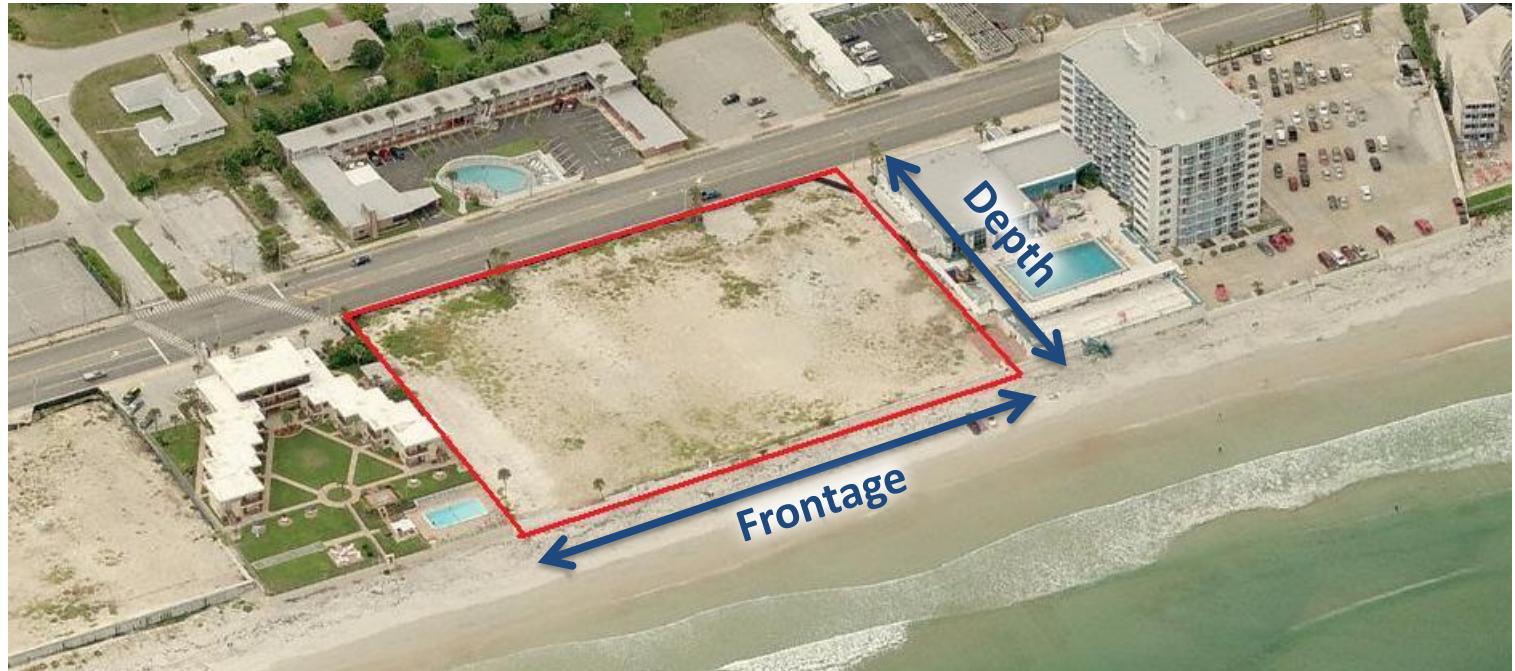


$$f_{\mathbf{w}}(\text{frontage}) = w_0 + w_1(\text{frontage}) + w_2(\text{frontage}^2)$$

$$f_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2$$

$$\begin{aligned}x_1 &= \text{frontage} \\x_2 &= \text{frontage}^2\end{aligned}$$

Creating new features



$$x_1 = \text{Frontage}$$

$$x_2 = \text{Depth}$$

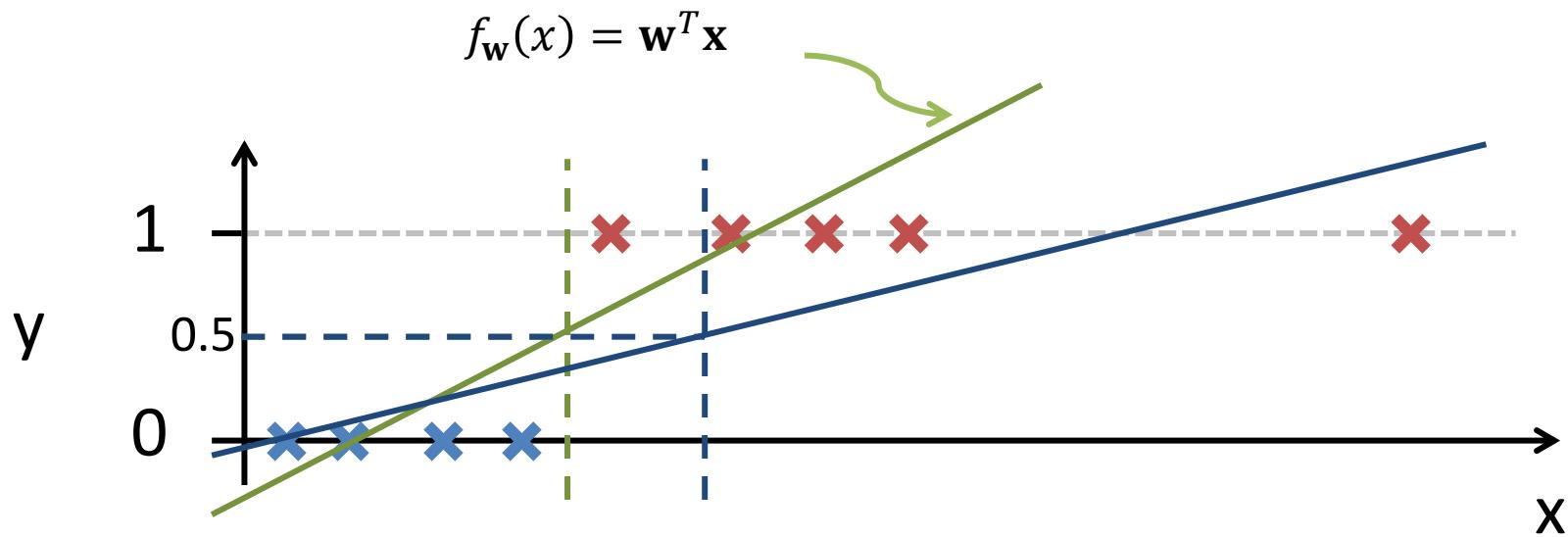
$$x_3 = \text{Area} = \text{Frontage} \times \text{Depth} = x_1 x_2$$

LOGISTIC REGRESSION

And how to use it to do classification

Classification

Classification: Predict discrete valued output (e.g. 0 or 1)

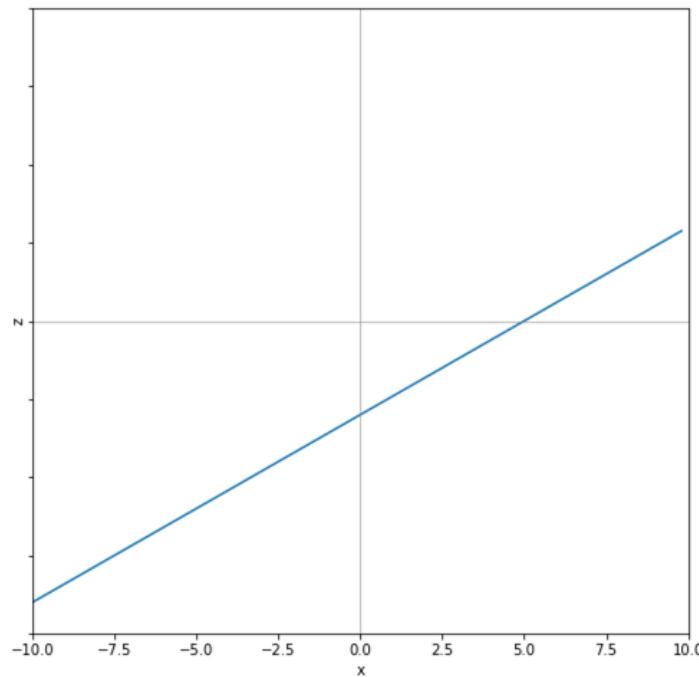
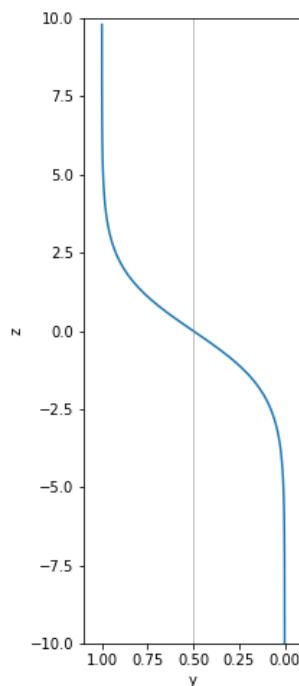
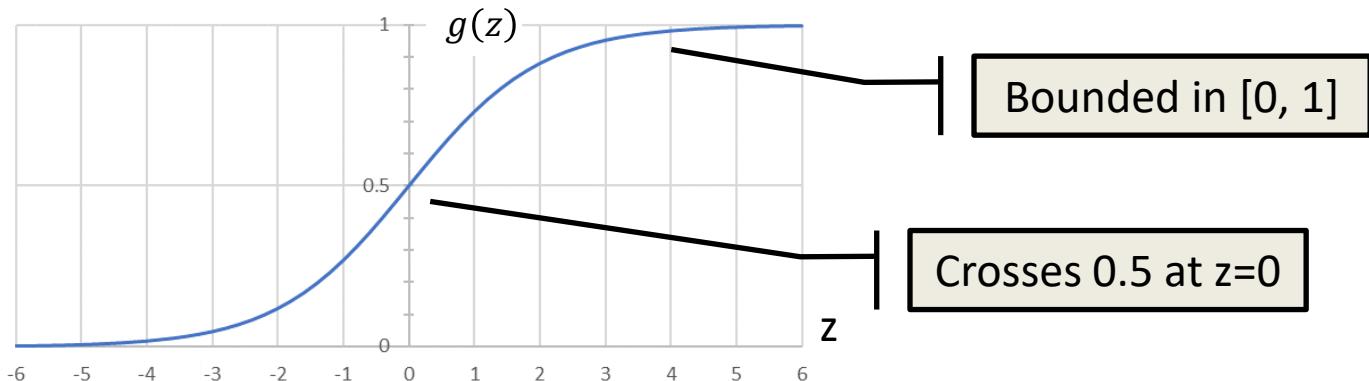


Idea: do linear regression and then **threshold** the prediction at 0.5 so that:

- if $f_w(\mathbf{x}) \geq 0.5$, then class = 1
- if $f_w(\mathbf{x}) < 0.5$, then class = 0

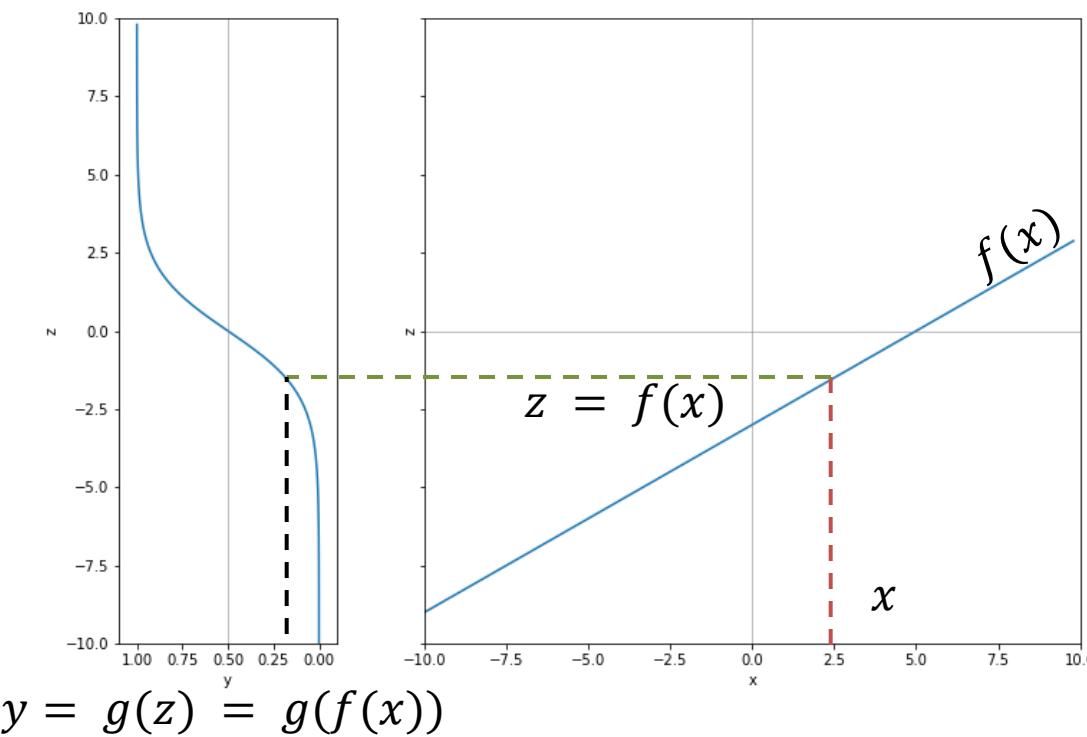
Logistic (Sigmoid) Function

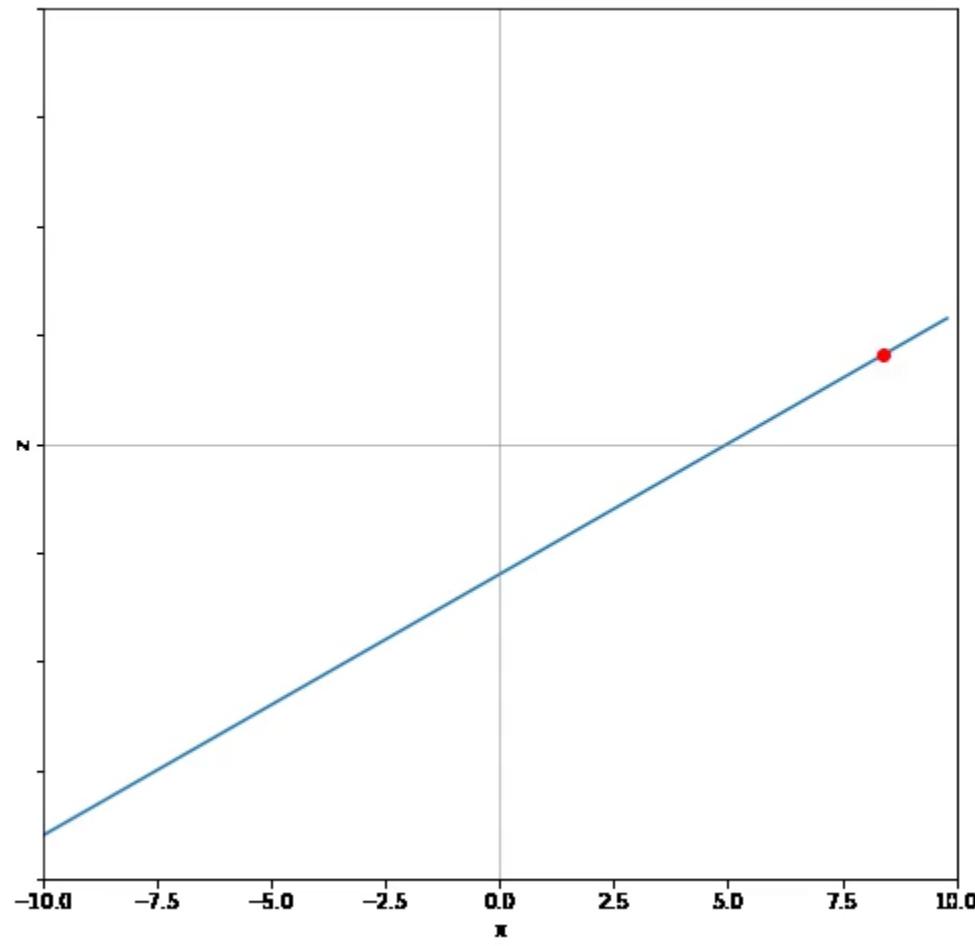
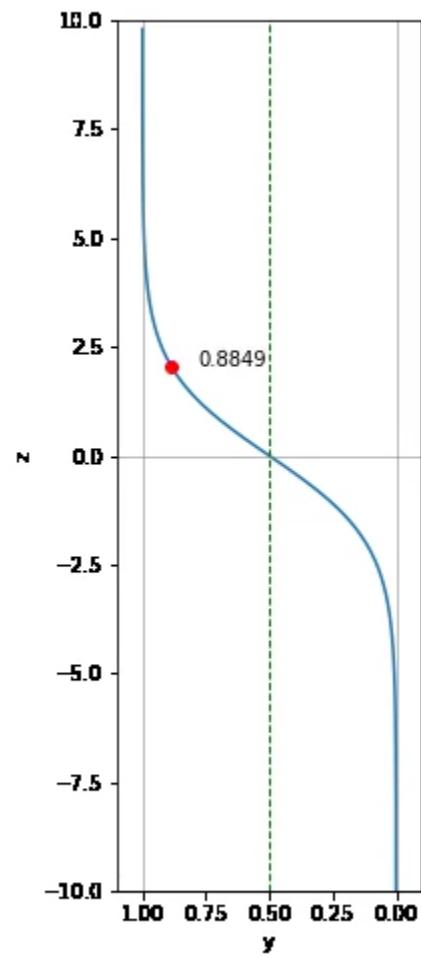
$$g(z) = \frac{1}{1 + e^{-z}}$$



Playing with the Logistic

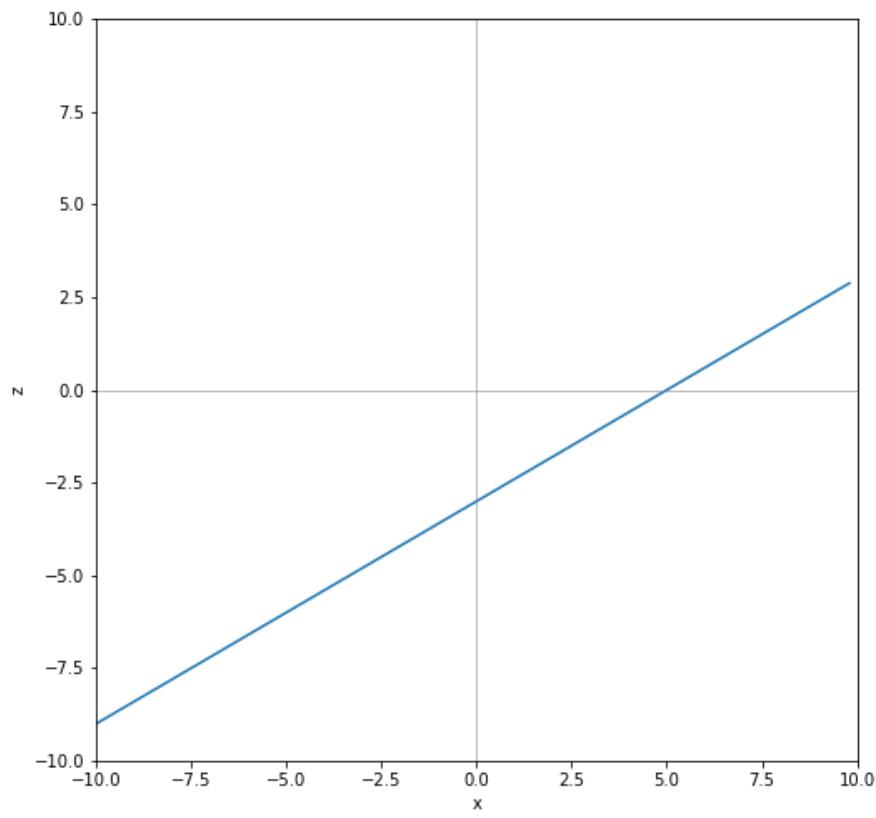
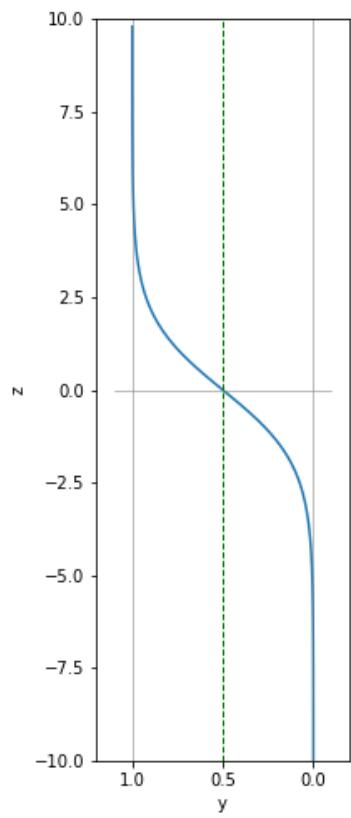
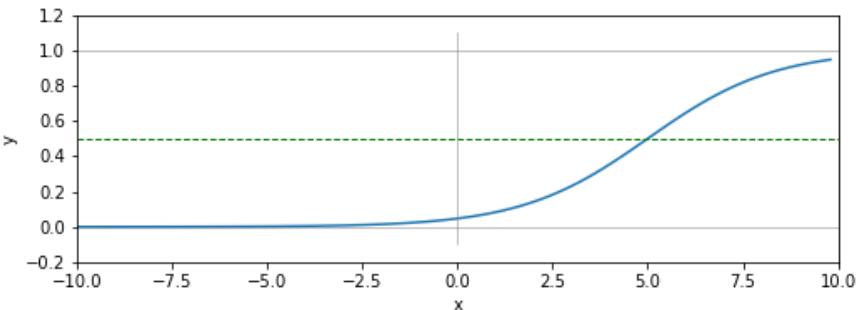
$$g(z) = \frac{1}{1 + e^{-z}}$$
$$z = f(x) = w_0 + w_1 x$$

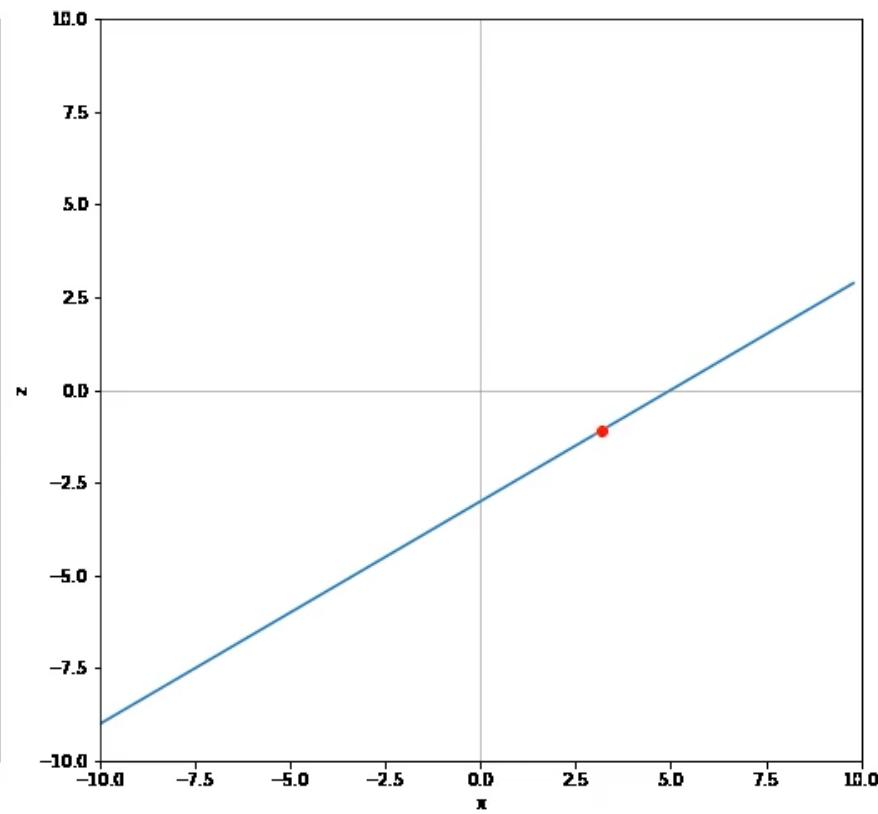
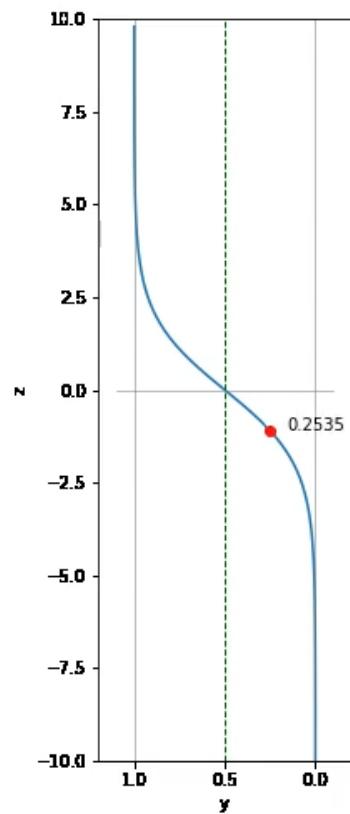
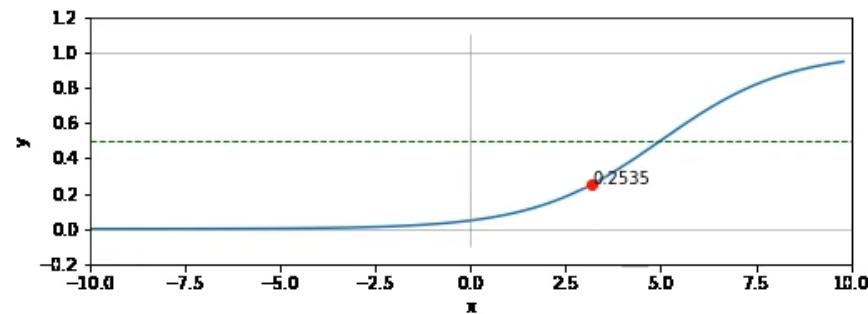


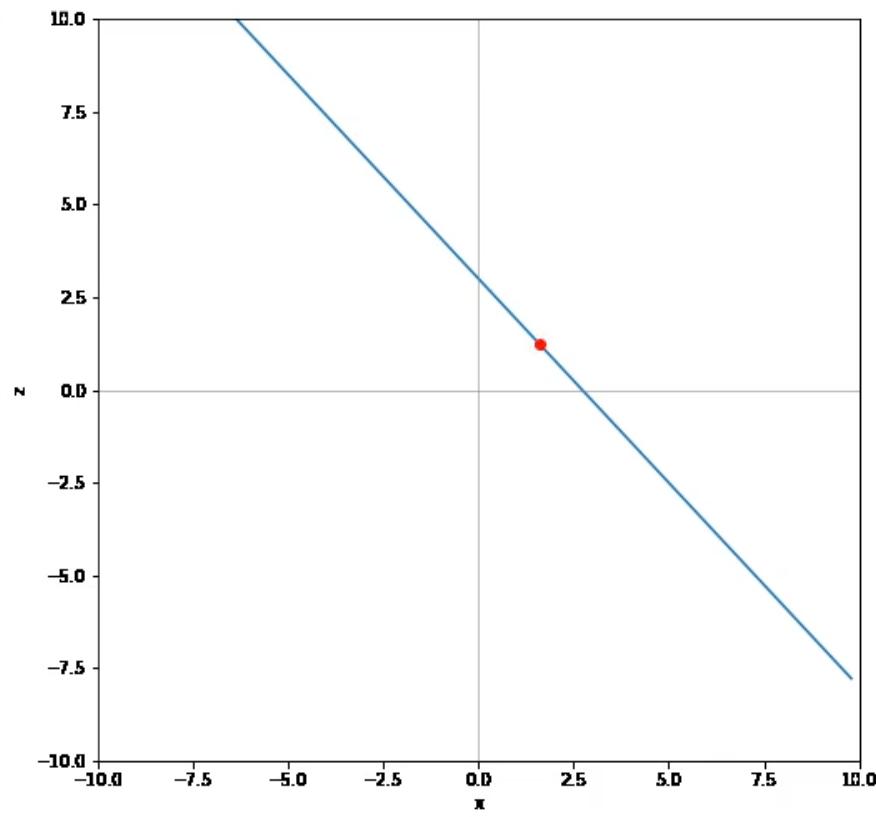
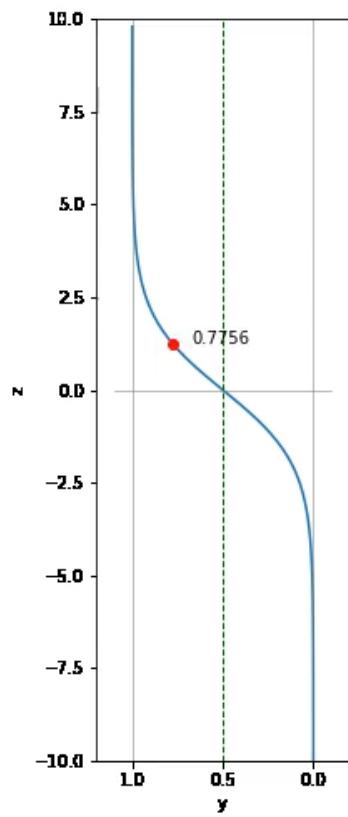
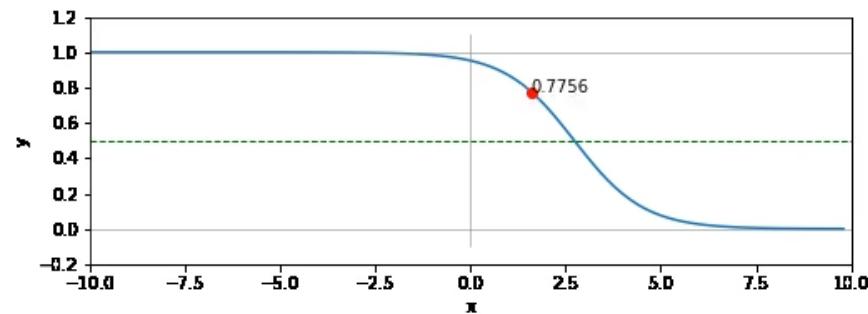


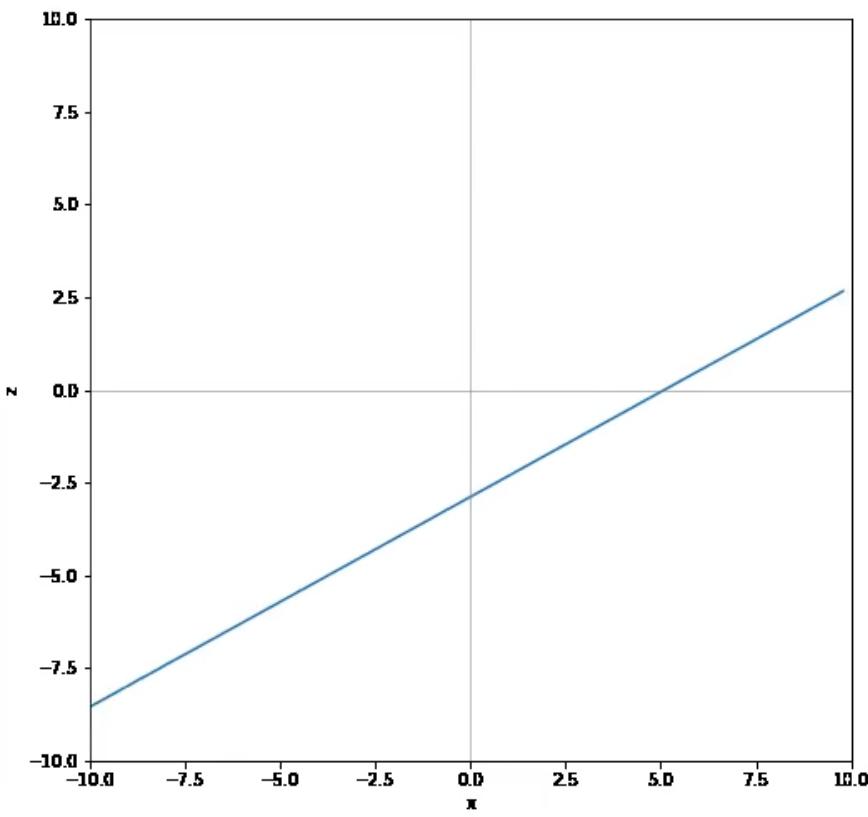
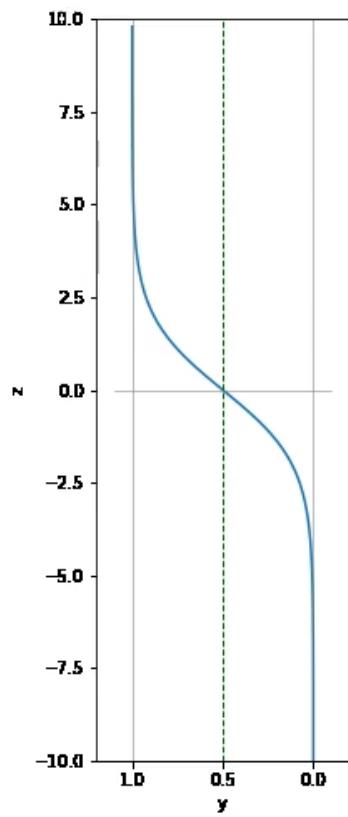
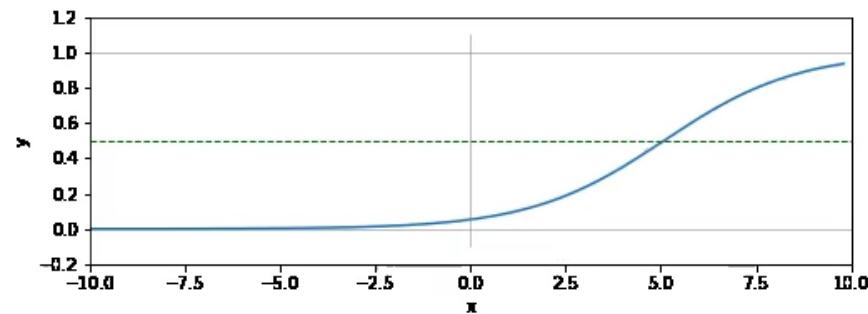
$$y = g(z) = g(f(x))$$

χ



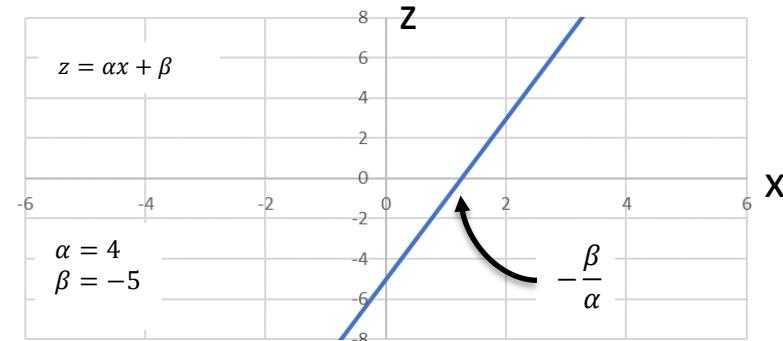
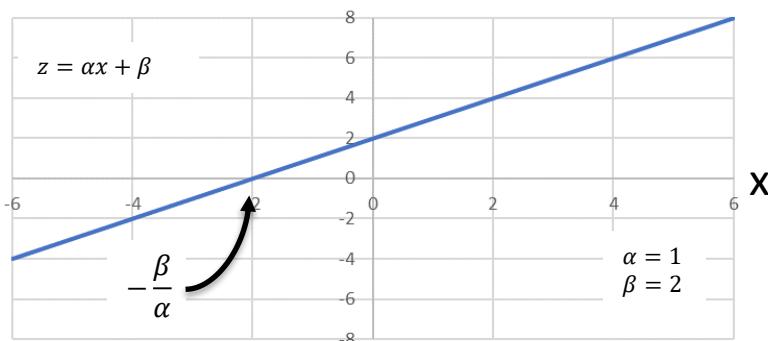
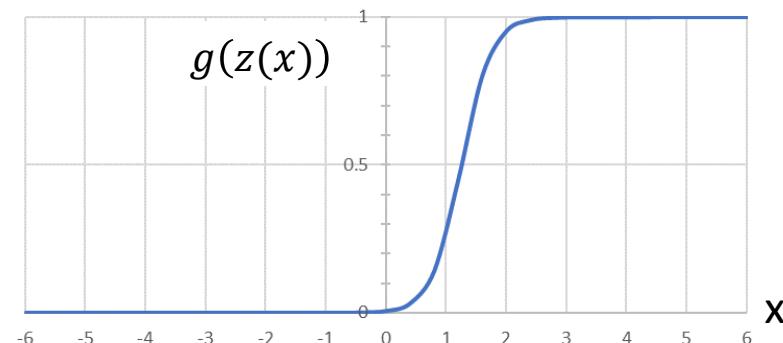
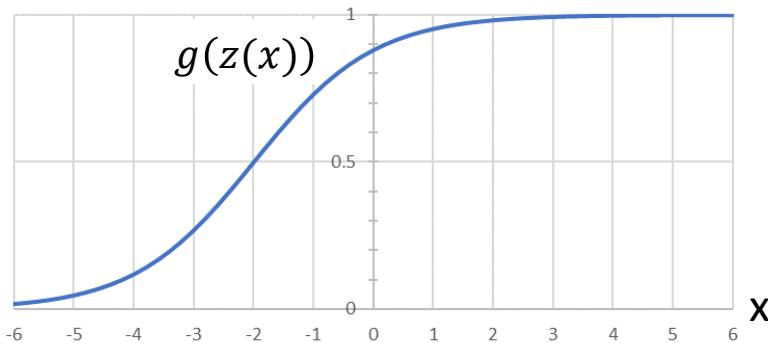
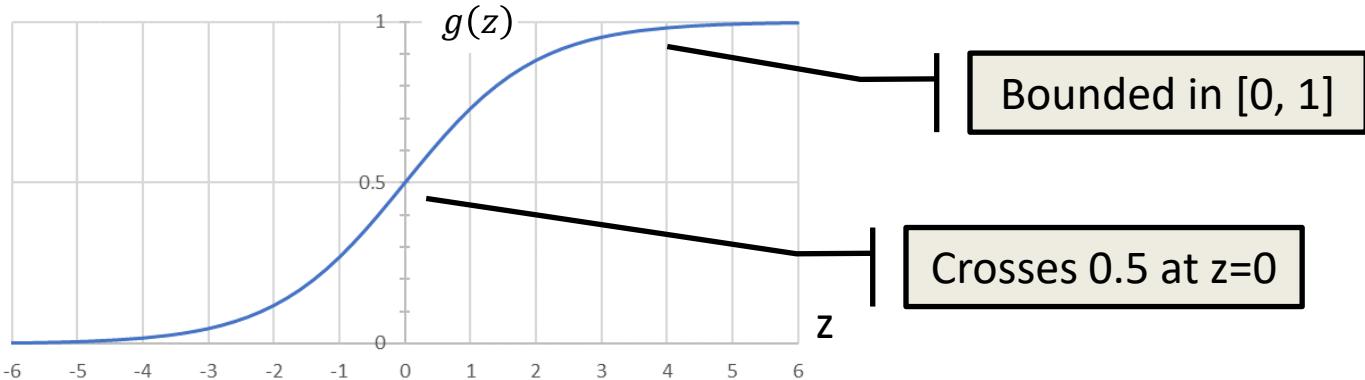






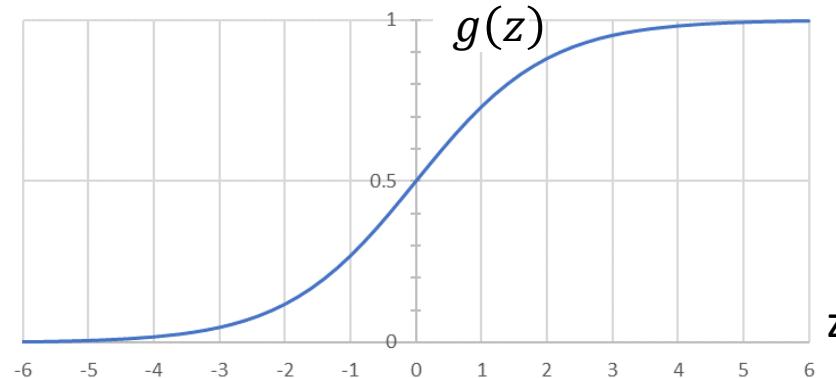
Logistic (Sigmoid) Function

$$g(z) = \frac{1}{1 + e^{-z}}$$

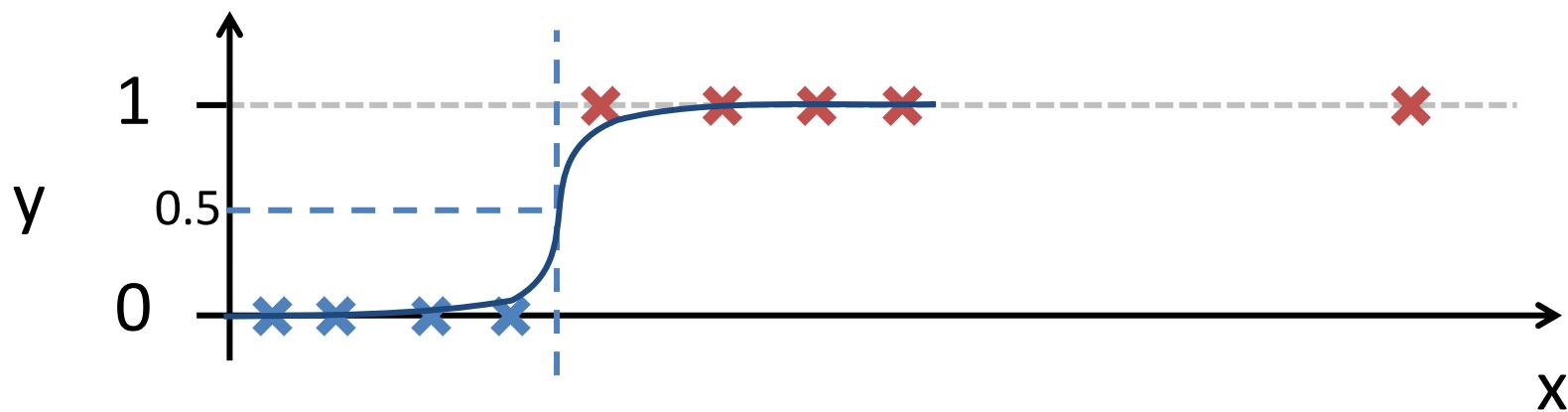


Logistic Regression Model

$$g(z) = \frac{1}{1 + e^{-z}}$$



$$f_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$



Interpretation of hypothesis output

$f_w(\mathbf{x})$ can be interpreted as the **estimated probability** that $y = 1$ given input \mathbf{x}

$$P(y = 0|\mathbf{x}; \mathbf{w}) + P(y = 1|\mathbf{x}; \mathbf{w}) = 1$$

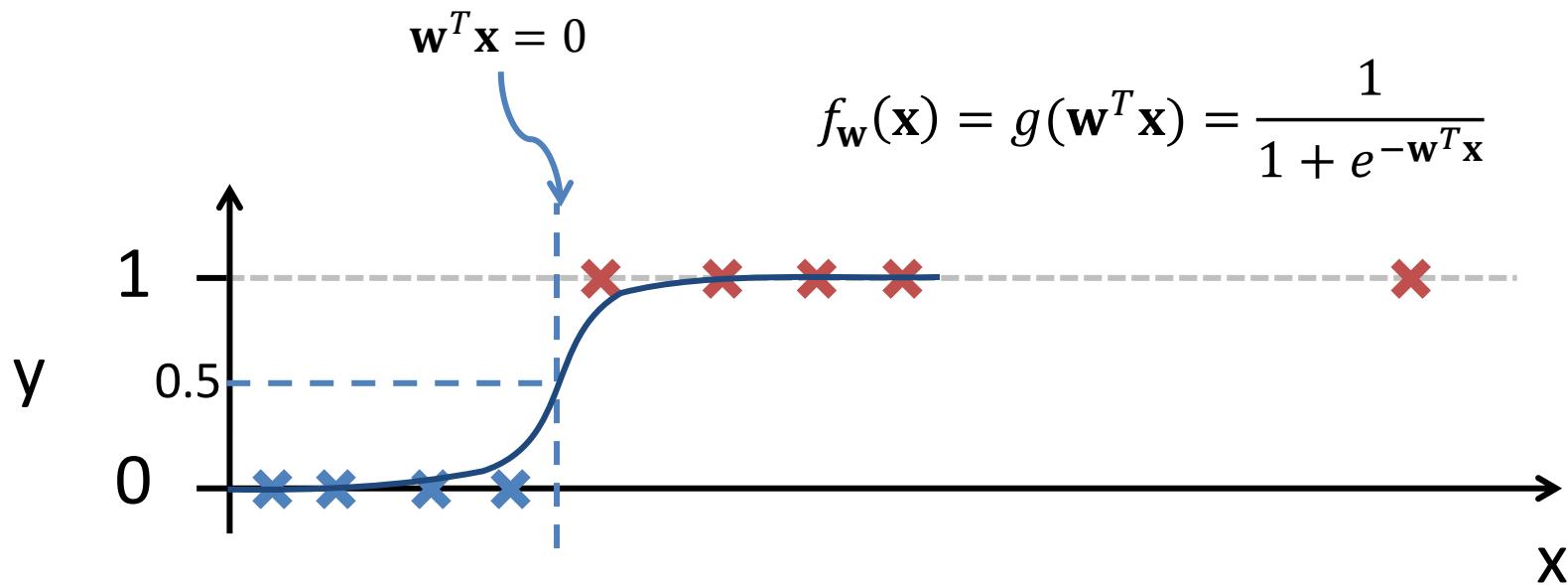
$$P(y = 0|\mathbf{x}; \mathbf{w}) = 1 - P(y = 1|\mathbf{x}; \mathbf{w})$$

So in the example, $f_w(\mathbf{x}) = 0.8$ would mean that we have a 80% chance of the point to be of class ($y = 1$)

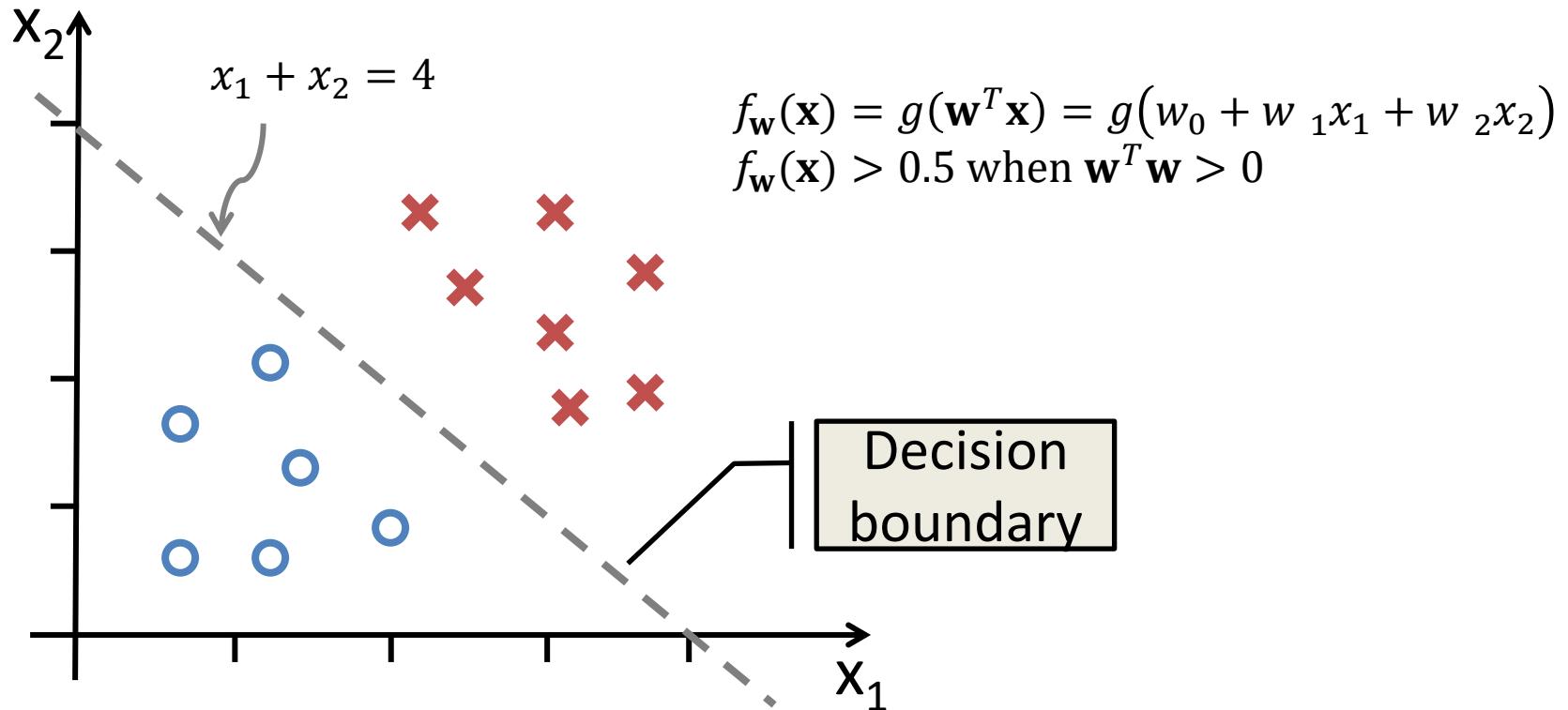
The Decision Boundary

Suppose we predict “ $y = 1$ ” if $f_w(\mathbf{x}) \geq 0.5$
predict “ $y = 0$ ” if $f_w(\mathbf{x}) < 0.5$

The decision boundary is defined by all the points x for which $f_w(\mathbf{x}) = 0.5$ which is equivalent to $\mathbf{w}^T \mathbf{x} = 0$



Decision Boundary

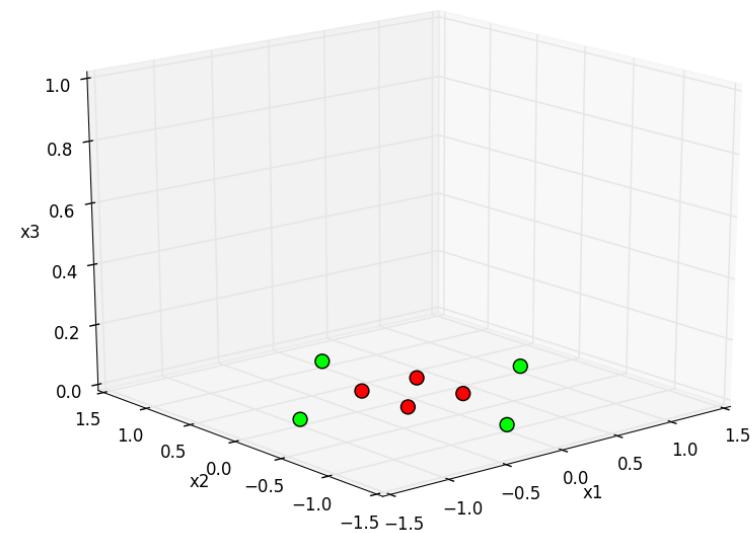
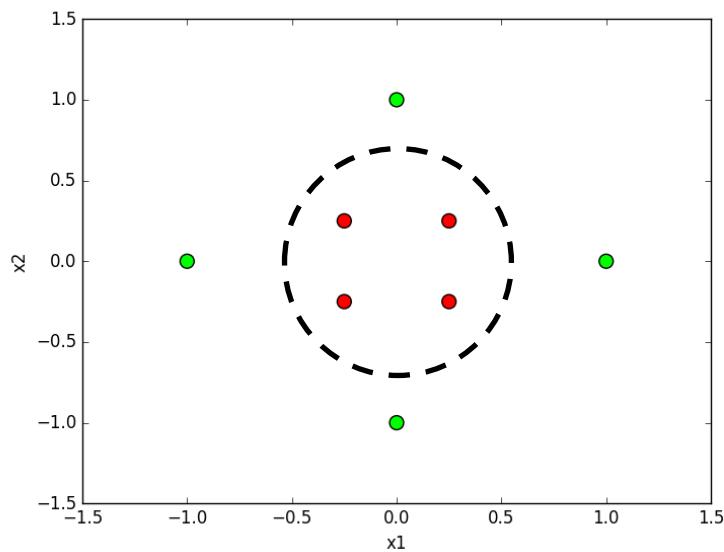


$$y = 1, \text{ when } f_{\mathbf{w}}(\mathbf{x}) \geq 0.5 \Leftrightarrow \mathbf{w}^T \mathbf{x} = -4 + x_1 + x_2 \geq 0$$

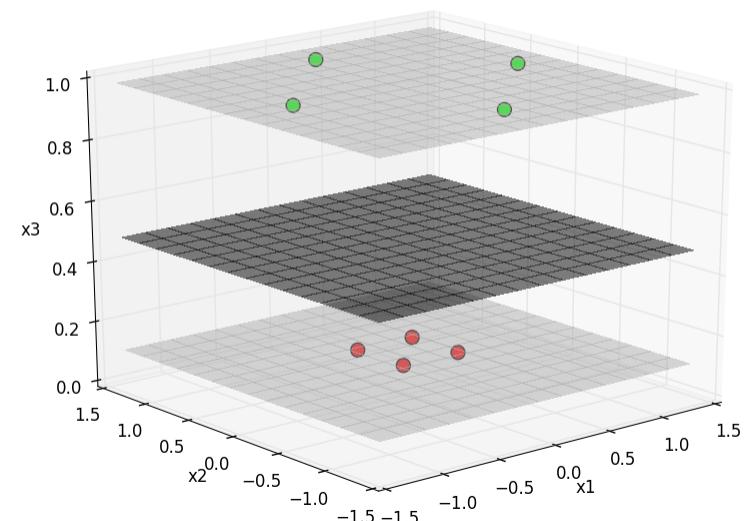
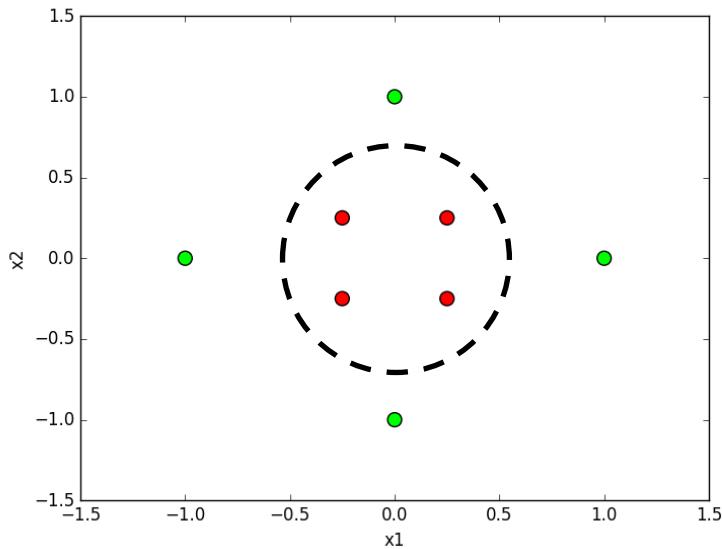
e.g. $\mathbf{w} = \begin{bmatrix} -4 \\ 1 \\ 1 \end{bmatrix}$

$$y = 1, \text{ when } x_1 + x_2 \geq 4$$
$$y = 0, \text{ when } f_{\mathbf{w}}(\mathbf{x}) < 0.5 \Leftrightarrow \mathbf{w}^T \mathbf{x} = -4 + x_1 + x_2 < 0$$
$$y = 0, \text{ when } x_1 + x_2 < 4$$

Non linear decision boundaries



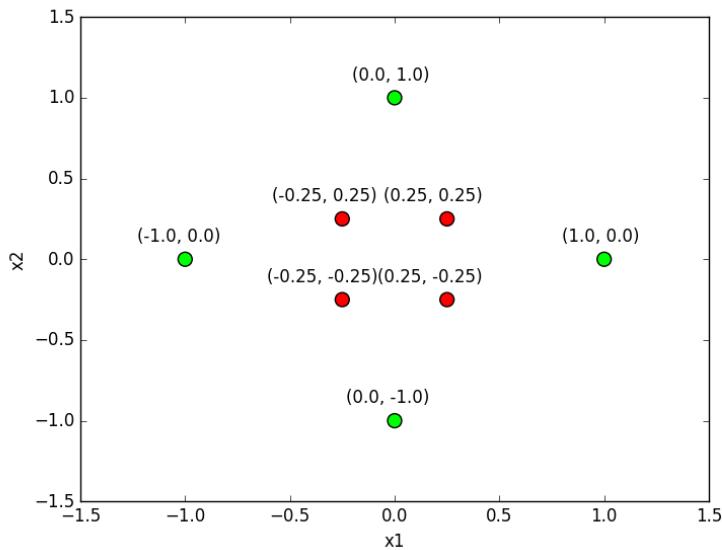
Non linear decision boundaries



Which method is better?

They are equivalent! But one is much simpler than the other

Non linear decision boundaries

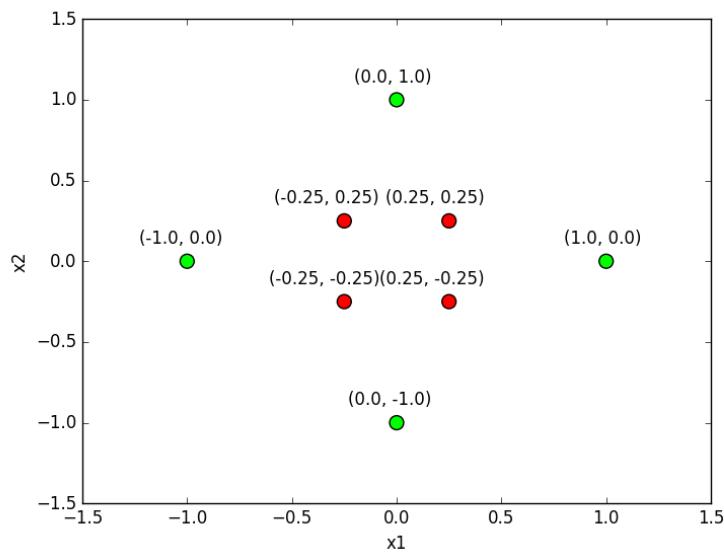


We need a way to be able to select a class, by applying a threshold, i.e. we need to find a function that gives the **green** points a higher value than the **red** ones or vice versa

Which of the following would work?

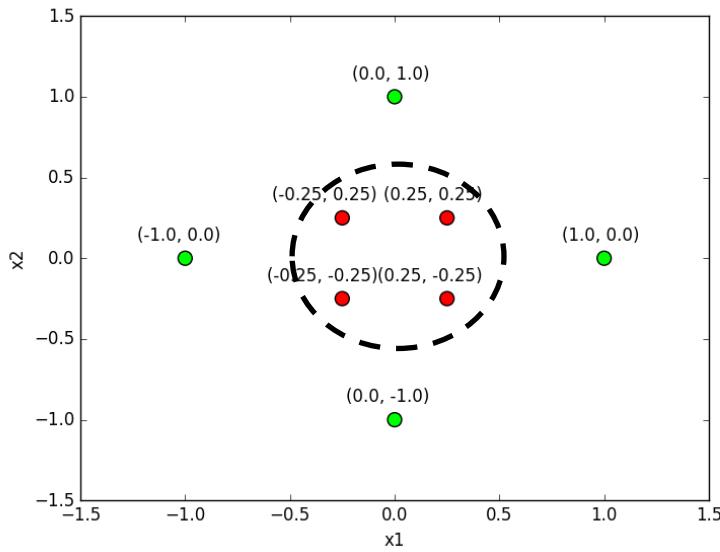
1. $x_1 + x_2$ (linear)
2. $x_1^2 + x_2^2$
3. $x_1 x_2$

Non linear decision boundaries



| x_1 | x_2 | $x_1 + x_2$ | $x_1^2 + x_2^2$ | xy |
|-------|-------|-------------|-----------------|---------|
| -1 | 0 | -1 | 1 | 0 |
| -0.25 | 0.25 | 0 | 0.125 | -0.0625 |
| -0.25 | -0.25 | -0.5 | 0.125 | 0.0625 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | -1 | -1 | 1 | 0 |
| 0.25 | -0.25 | 0 | 0.125 | -0.0625 |
| 0.25 | 0.25 | 0.5 | 0.125 | 0.0625 |
| 1 | 0 | 1 | 1 | 0 |

Non linear decision boundaries



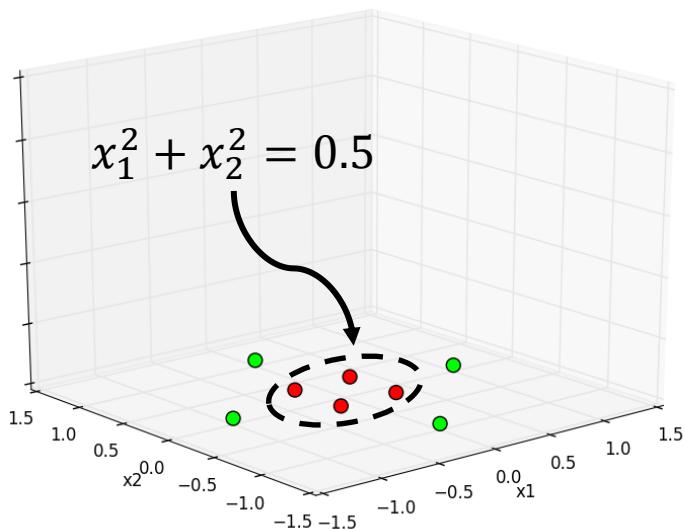
| x_1 | x_2 | $x_1 + x_2$ | $x_1^2 + x_2^2$ | xy |
|-------|-------|-------------|-----------------|---------|
| -1 | 0 | -1 | 1 | 0 |
| -0.25 | 0.25 | 0 | 0.125 | -0.0625 |
| -0.25 | -0.25 | -0.5 | 0.125 | 0.0625 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | -1 | -1 | 1 | 0 |
| 0.25 | -0.25 | 0 | 0.125 | -0.0625 |
| 0.25 | 0.25 | 0.5 | 0.125 | 0.0625 |
| 1 | 0 | 1 | 1 | 0 |

$$x_1^2 + x_2^2 = 0.125$$

$$x_1^2 + x_2^2 = 0.5$$

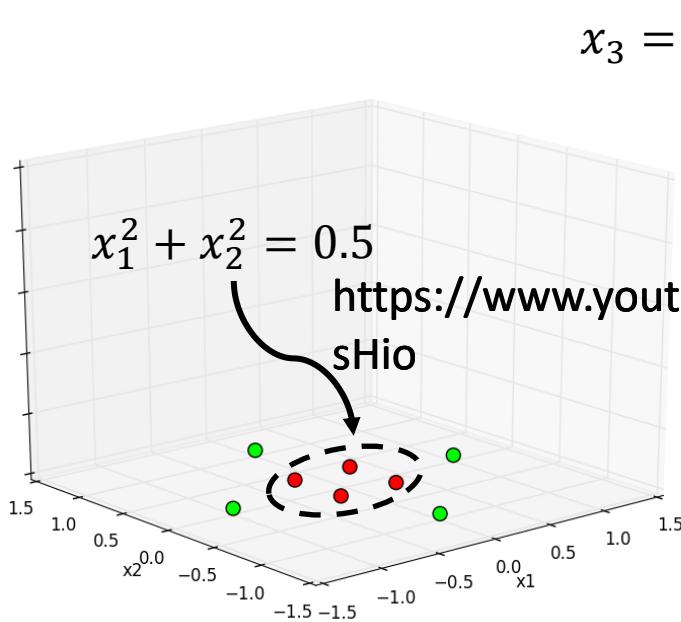
$$x_1^2 + x_2^2 = 1$$

Non Linear Decision Boundaries

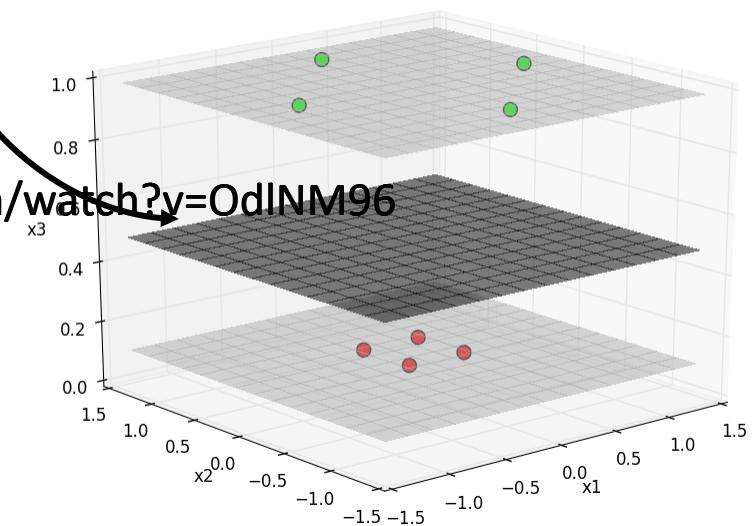


$$(x_1, x_2) \rightarrow (x_1, x_2, x_3 = x_1^2 + x_2^2)$$

Non Linear Decision Boundaries

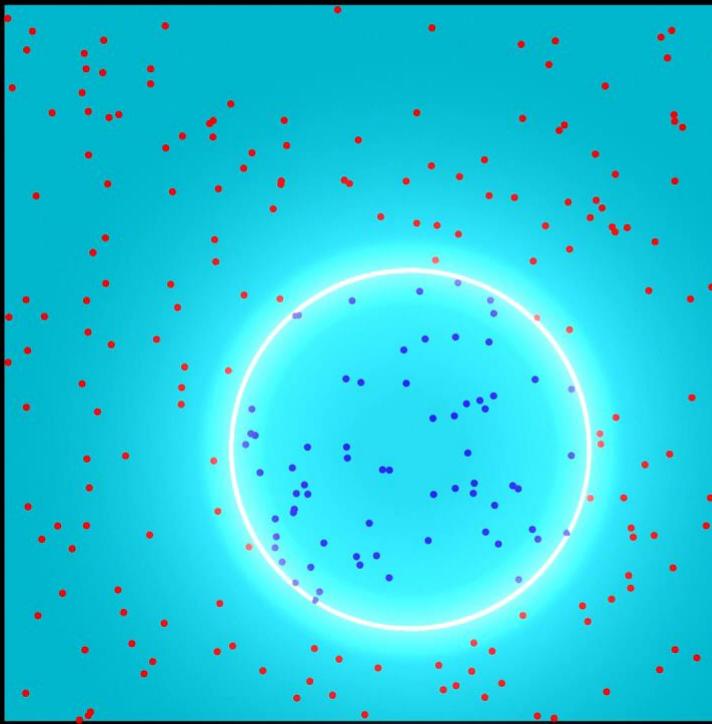


$$x_3 = x_1^2 + x_2^2 = 0.5$$



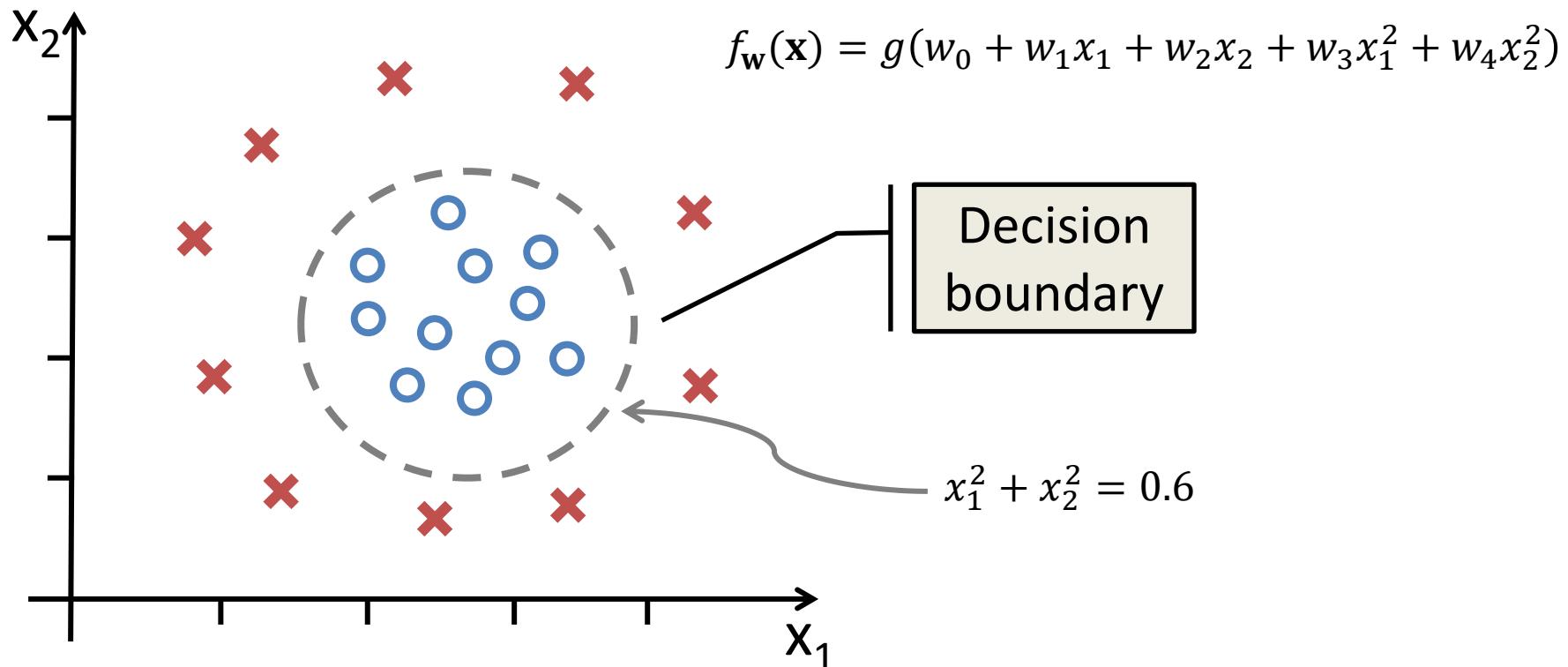
$$(x_1, x_2) \rightarrow (x_1, x_2, x_3 = x_1^2 + x_2^2)$$

Non-Linearly Separable Classes



Using a polynomial kernel they can be linearly separated in 3D space

Non linear decision boundaries



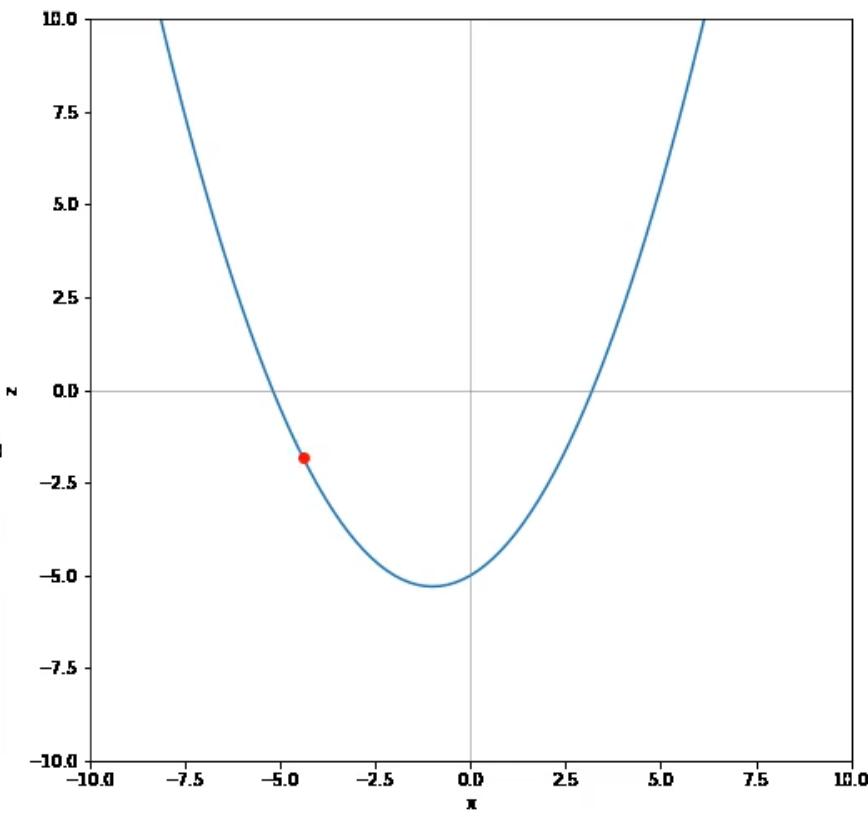
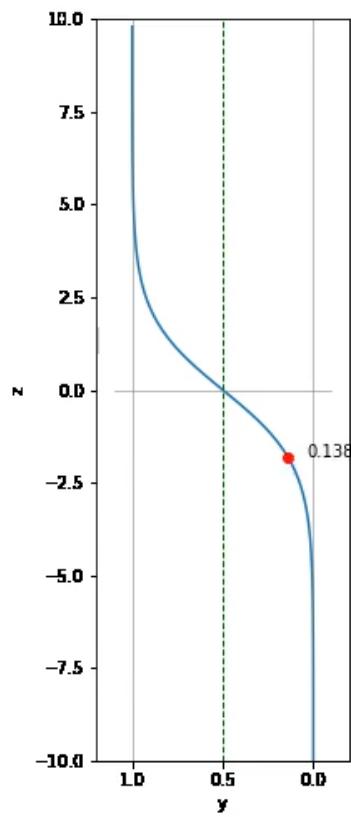
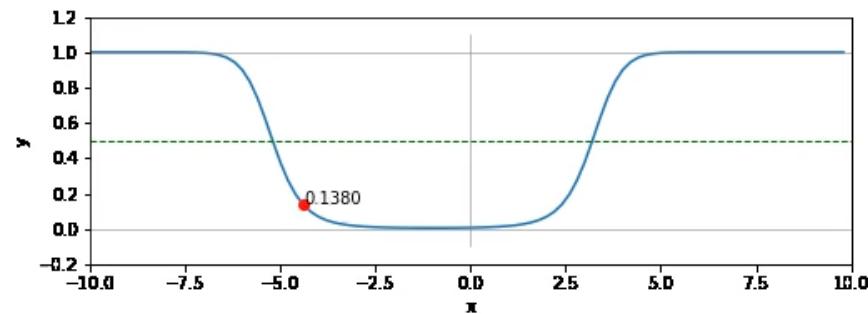
e.g. $\mathbf{w} = \begin{bmatrix} -0.6 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

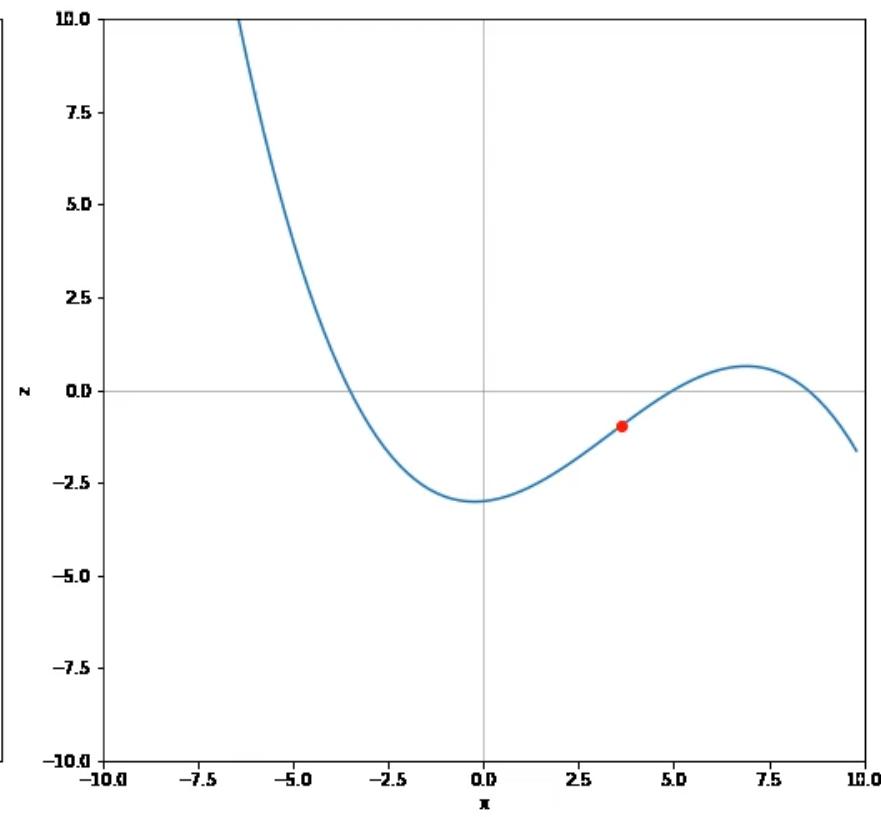
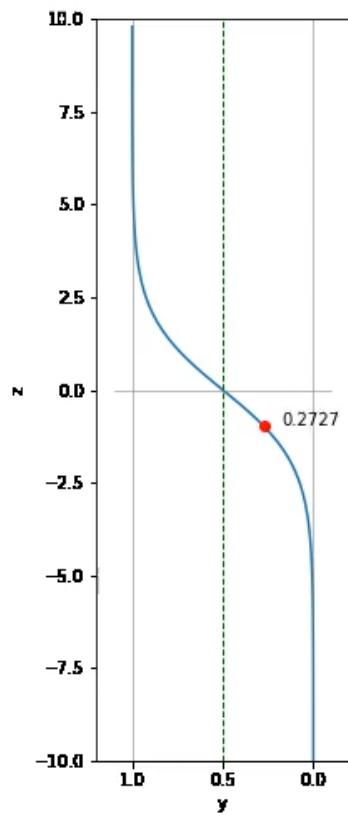
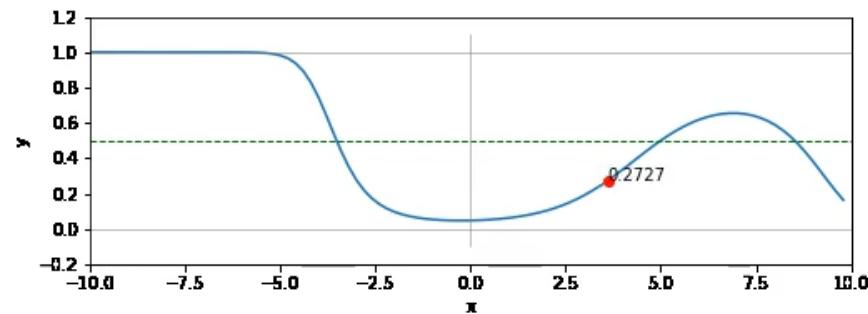
$y = 1, \text{ when } f_{\mathbf{w}}(\mathbf{x}) \geq 0.5 \Leftrightarrow \mathbf{w}^T \mathbf{x} = -0.6 + x_1^2 + x_2^2 \geq 0$

$y = 1, \text{ when } x_1^2 + x_2^2 \geq 0.6$

$y = 0, \text{ when } f_{\mathbf{w}}(\mathbf{x}) < 0.5 \Leftrightarrow \mathbf{w}^T \mathbf{x} = -0.6 + x_1^2 + x_2^2 < 0$

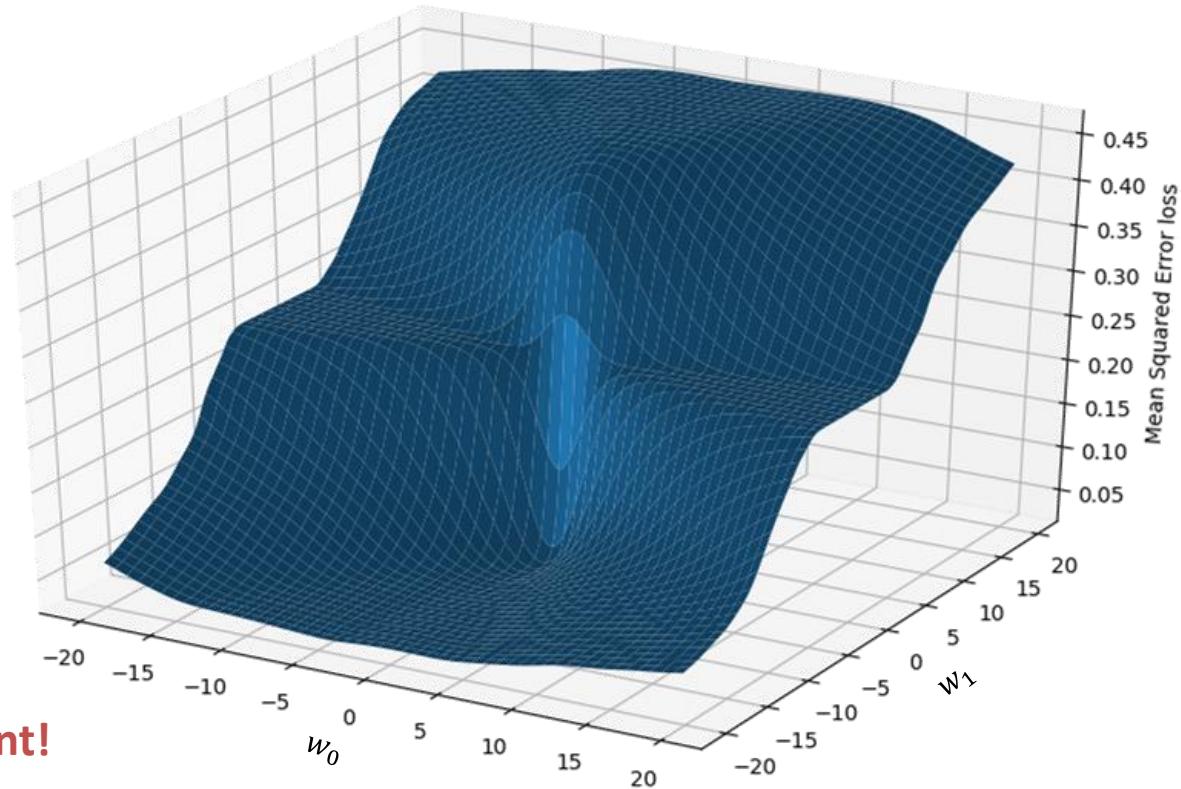
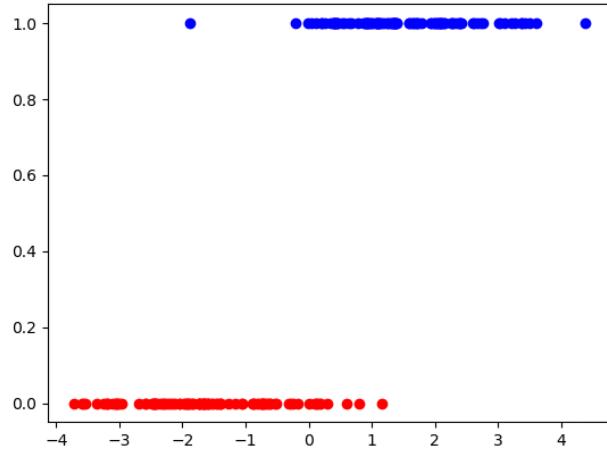
$y = 0, \text{ when } x_1^2 + x_2^2 < 0.6$





Mean Squared Error cost

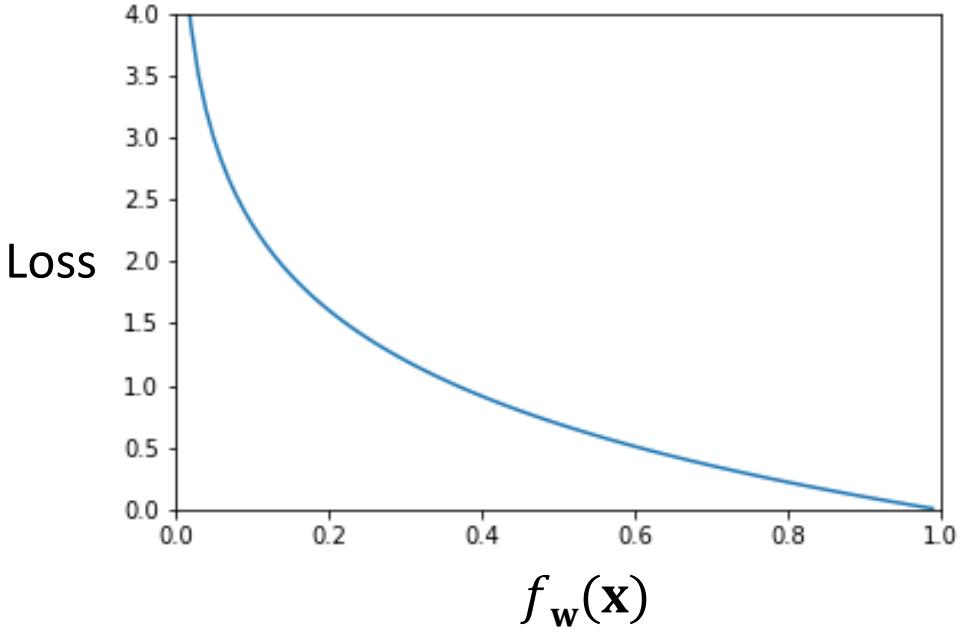
$$J_{MSE}(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^m (f_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)})^2$$



Not an easy one to descent!

Loss function for logistic regression

$$\text{Loss}(f_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(f_{\mathbf{w}}(\mathbf{x})) & \text{,if } y = 1 \\ -\log(1 - f_{\mathbf{w}}(\mathbf{x})) & \text{,if } y = 0 \end{cases}$$



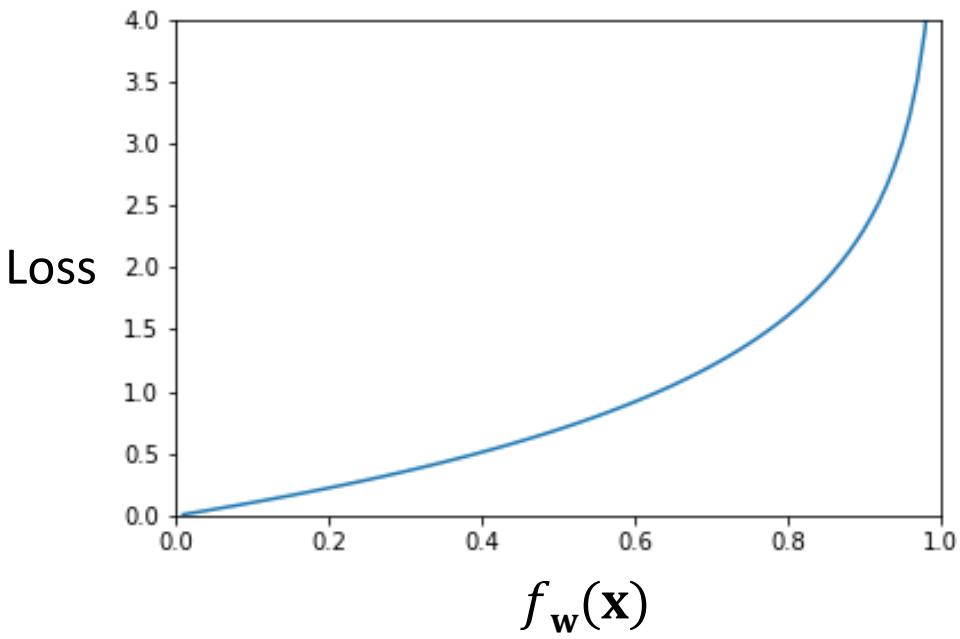
If $y=1$ AND $f_{\mathbf{w}}(\mathbf{x}) = 1$ (real data and our prediction agree) then the **Loss = 0**

If $y=1$ BUT $f_{\mathbf{w}}(\mathbf{x}) \rightarrow 0$ (real data and our prediction disagree) then the **Loss $\rightarrow \infty$**

This captures the intuition that if $f_{\mathbf{w}}(\mathbf{x}) = 0$ (our algorithm predicts that $P(y = 1|\mathbf{x}, \mathbf{w}) = 0$) but $y=1$ we will penalise the learning algorithm by a very large loss

Loss function for logistic regression

$$\text{Loss}(f_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(f_{\mathbf{w}}(\mathbf{x})) & \text{if } y = 1 \\ -\log(1 - f_{\mathbf{w}}(\mathbf{x})) & \text{if } y = 0 \end{cases}$$



If $y=0$ AND $f_{\mathbf{w}}(\mathbf{x}) = 0$ (real data and our prediction agree) then the **Loss = 0**

If $y=0$ BUT $f_{\mathbf{w}}(\mathbf{x}) \rightarrow 1$ (real data and our prediction disagree) then the **Loss $\rightarrow \infty$**

This captures the intuition that if $f_{\mathbf{w}}(\mathbf{x}) = 1$ (our algorithm predicts that $P(y = 1|\mathbf{x}, \mathbf{w}) = 1$) but $y=0$ we will penalise the learning algorithm by a very large loss

Binary Cross-Entropy Loss

$$J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m Loss(f_{\mathbf{w}}(\mathbf{x}^{(i)}), y^{(i)})$$

$$Loss(f_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(f_{\mathbf{w}}(\mathbf{x})) & , \text{if } y = 1 \\ -\log(1 - f_{\mathbf{w}}(\mathbf{x})) & , \text{if } y = 0 \end{cases}$$

(Note that y is always either 0 or 1)

$$Loss(f_{\mathbf{w}}(\mathbf{x}), y) = -y \log(f_{\mathbf{w}}(\mathbf{x})) - (1 - y) \log(1 - f_{\mathbf{w}}(\mathbf{x}))$$

If $y = 1$: $Loss(f_{\mathbf{w}}(\mathbf{x}), y) = -\log(f_{\mathbf{w}}(\mathbf{x}))$

If $y = 0$: $Loss(f_{\mathbf{w}}(\mathbf{x}), y) = -\log(1 - f_{\mathbf{w}}(\mathbf{x}))$

Gradient Descent

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{m} \sum_{i=1}^m Loss(f_{\mathbf{w}}(\mathbf{x}^{(i)}), y^{(i)}) \\ &= -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log f_{\mathbf{w}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - f_{\mathbf{w}}(\mathbf{x}^{(i)})) \right] \end{aligned}$$

To fit parameters \mathbf{w} we want to minimise the cost function $J(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$

Repeat

{

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w})$$

(simultaneously for all w_j)

}

Gradient Descent

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{m} \sum_{i=1}^m Loss(f_{\mathbf{w}}(\mathbf{x}^{(i)}), y^{(i)}) \\ &= -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log f_{\mathbf{w}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - f_{\mathbf{w}}(\mathbf{x}^{(i)})) \right] \end{aligned}$$

To fit parameters \mathbf{w} we want to minimise the cost function $J(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$

Repeat

{

$$w_j := w_j - \alpha \sum_{i=1}^m (f_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

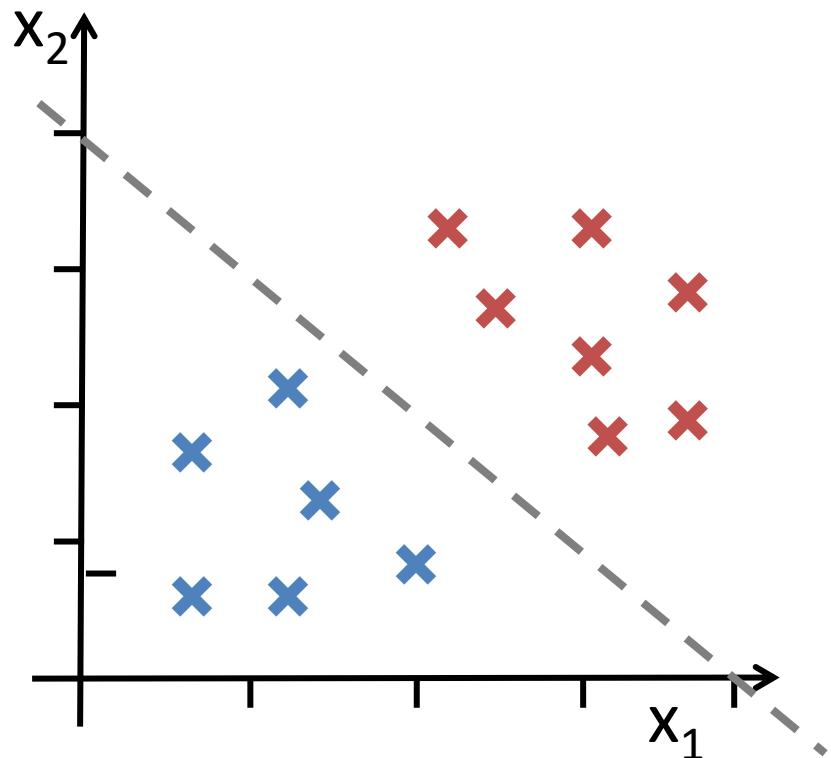
(simultaneously for all w_j)

}

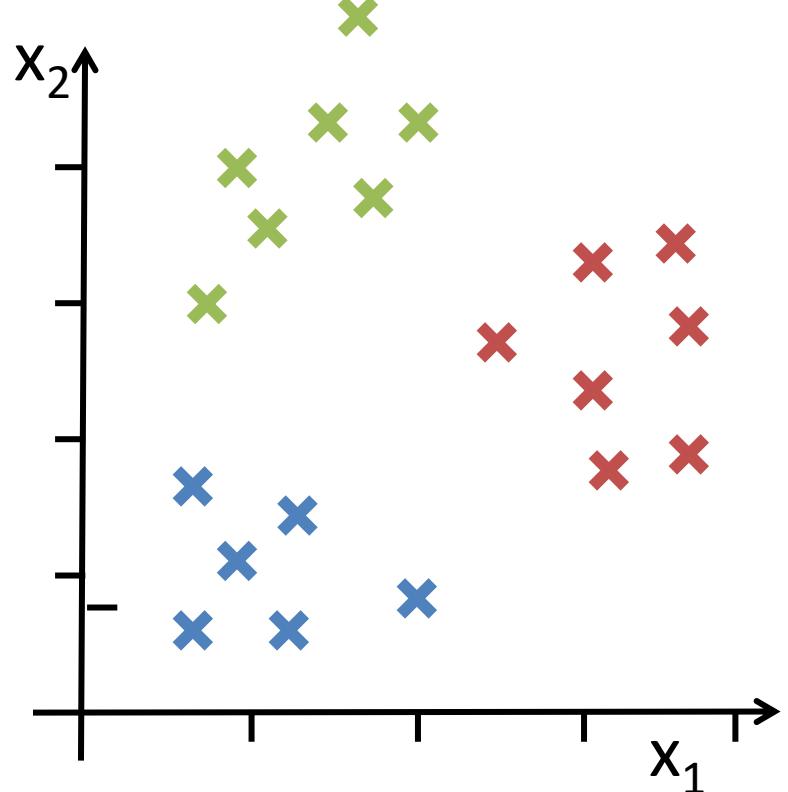
Algorithm looks identical to linear regression, but $f(\mathbf{w})$ is different here!

MULTICLASS CLASSIFICATION

Binary vs Multi-class classification

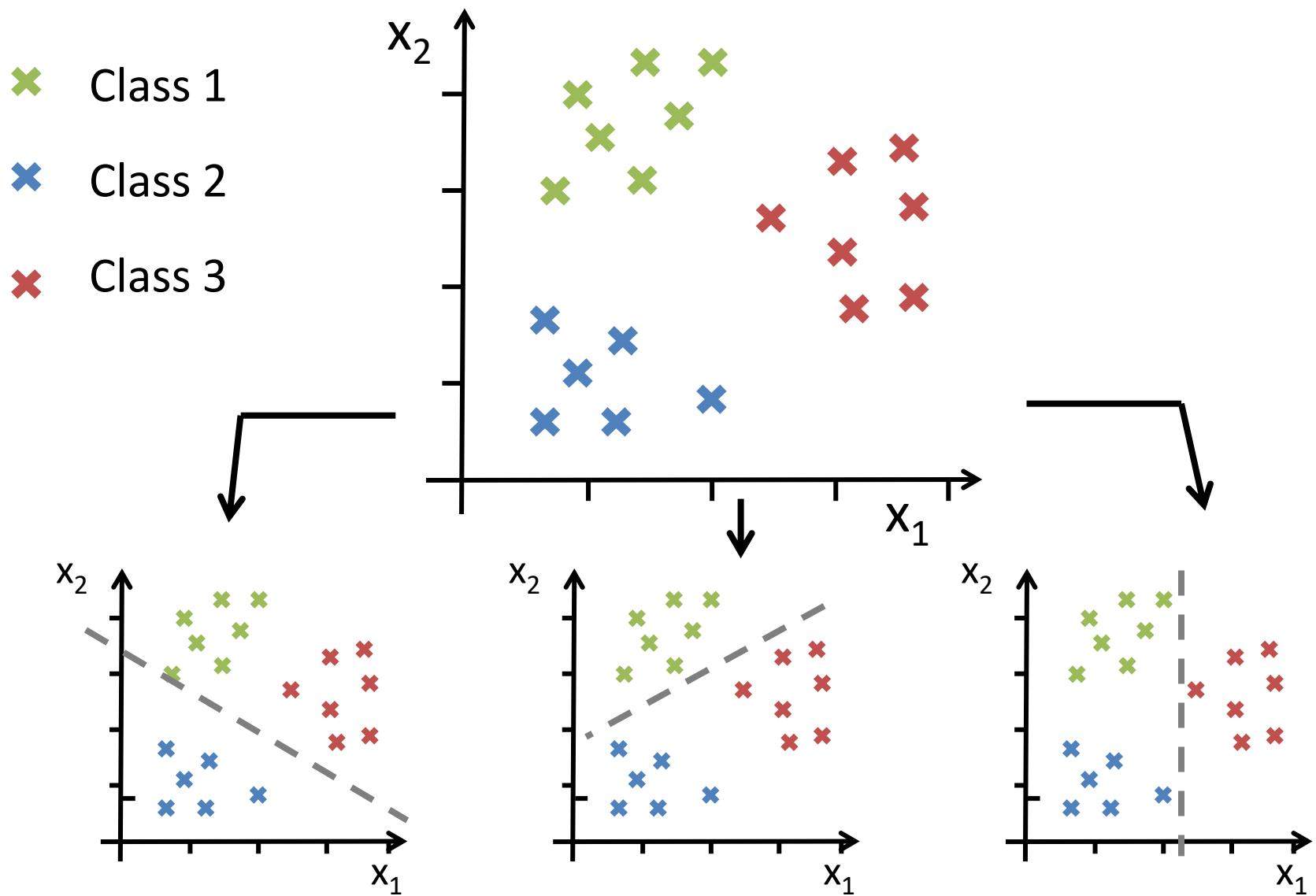


Binary: $y = \{0, 1\}$



Multi-class: $y = \{0, 1, \dots, n\}$

One vs all (one vs rest)



One vs All Classification

Train a (logistic regression) classifier $f_{\mathbf{w}}^{(c)}(\mathbf{x})$ for each class c to predict the probability that $y = c$

To make a prediction on a new input \mathbf{x} , pick the class c that maximizes the probability $f_{\mathbf{w}}^{(c)}(\mathbf{x})$

$$\operatorname{argmax}_c f_{\mathbf{w}}^{(c)}(\mathbf{x}) \Leftrightarrow \operatorname{argmax}_c P(y = c | \mathbf{x}, \mathbf{w})$$

Resources (I)



I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning”, MIT Press, 2016

<http://www.deeplearningbook.org/>



C. Bishop, “Pattern Recognition and Machine Learning”, Springer, 2006

<http://research.microsoft.com/en-us/um/people/cmbishop/prml/index.htm>



D. MacKay, “Information Theory, Inference and Learning Algorithms”, Cambridge University Press, 2003

<http://www.inference.phy.cam.ac.uk/mackay/>



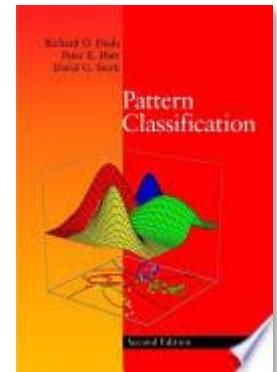
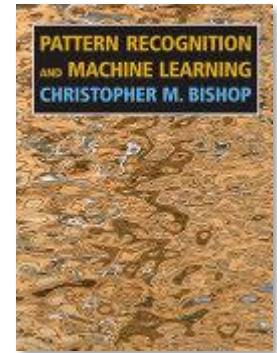
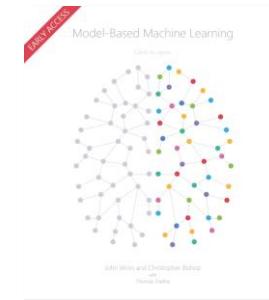
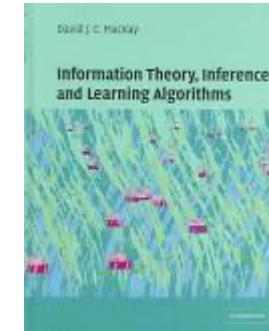
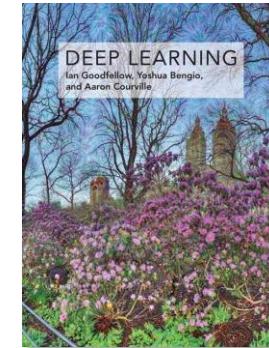
R.O. Duda, P.E. Hart, D.G. Stork, “Pattern Classification”, Wiley & Sons, 2000

http://books.google.com/books/about/Pattern_Classification.html?id=Br33IRC3PkQC



J. Winn, C. Bishop, “Model-Based Machine Learning”, early access

<http://mbmlbook.com/>



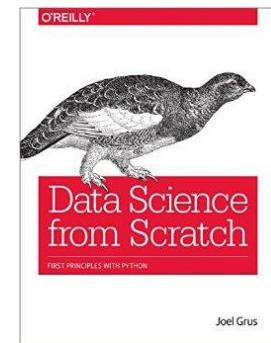
Resources (II)



“Python Data Science Handbook: Essential tools for working with Data”, Jake VanderPlas, O'Reilly Media, 2016, 1st Ed.
<https://jakevdp.github.io/PythonDataScienceHandbook/>



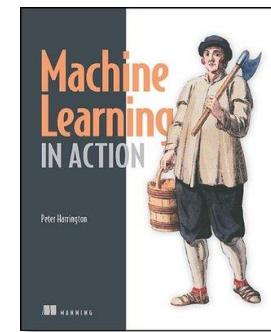
“Data Science from Scratch: First Principles with Python”, Joel Grus, O'Reilly Media, 2015, 1st Ed.
<https://github.com/joelgrus/data-science-from-scratch>



“Machine Learning in Action”, P. Harrington, Manning, 2012



“The Foundations of Data Science”, Ani Adhikari and John DeNero, [Online]
<https://www.inferentialthinking.com/chapters/intro>



Further Info

- Many of the slides of these lectures have been adapted from various highly recommended online lectures and courses:
 - Andrew Ng's *Machine Learning Course*, Coursera
<https://www.coursera.org/course/ml>
 - Andrew Ng's *Deep Learning Specialization*, Coursera
<https://www.coursera.org/specializations/deep-learning>
 - Victor Lavrenko's *Machine Learning Course*
<https://www.youtube.com/channel/UCs7alOMRnxhzfKAJ4JjZ7Wg>
 - Fei Fei Li and Andrej Karpathy's *Convolutional Neural Networks for Visual Recognition*
<http://cs231n.stanford.edu/>
 - Geoff Hinton's *Neural Networks for Machine Learning*, (ex Coursera)
<https://www.youtube.com/playlist?list=PLiPvV5TNogxKKwvKb1RKwkq2hm7ZvpHz0>
 - Luis Serrano's introductory videos
<https://www.youtube.com/channel/UCgBncpylJ1kiVaPyP-PZauQ>
 - Michael Nielsen's *Neural Networks and Deep Learning*
<http://neuralnetworksanddeeplearning.com/>