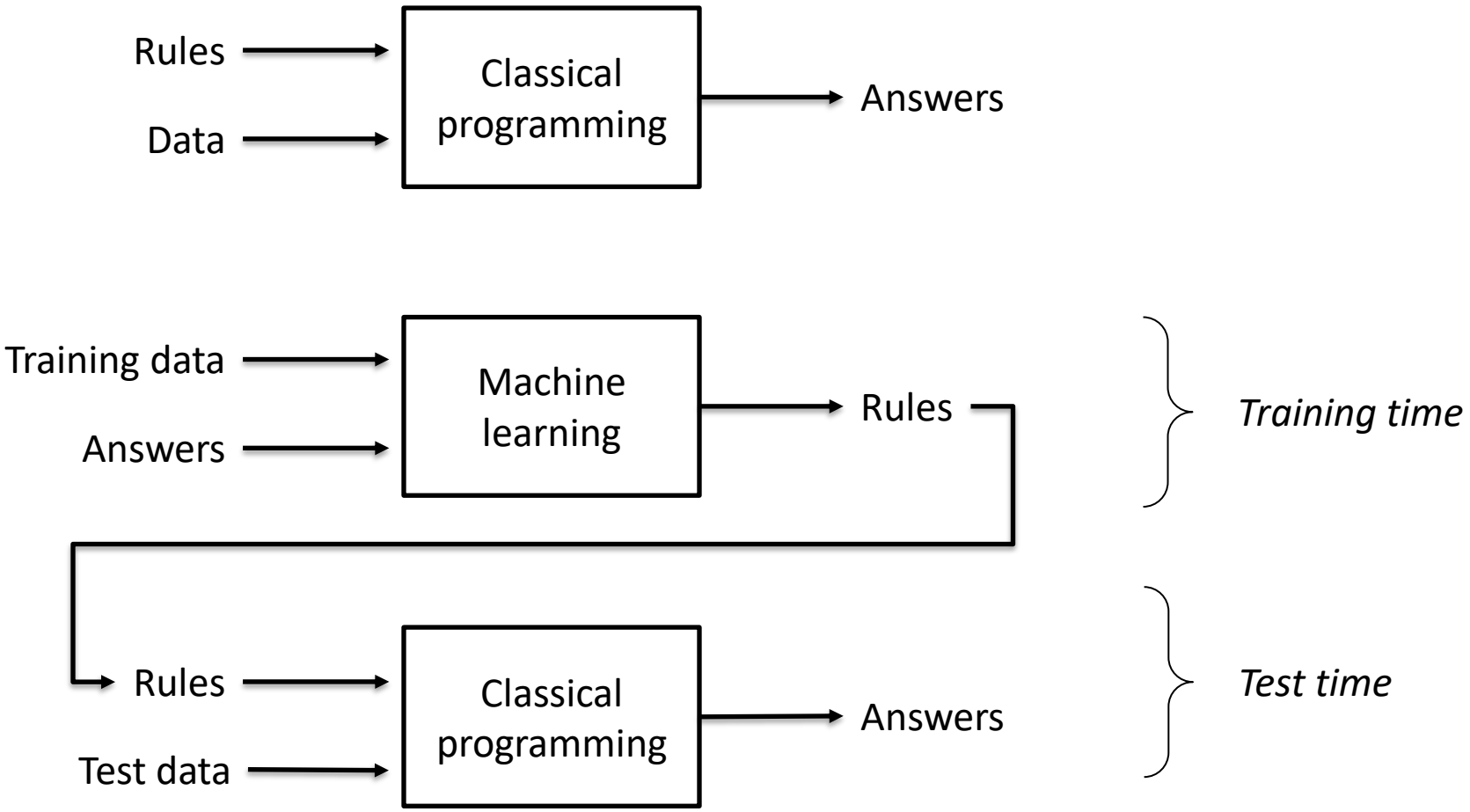
Introduction to Machine Learning I

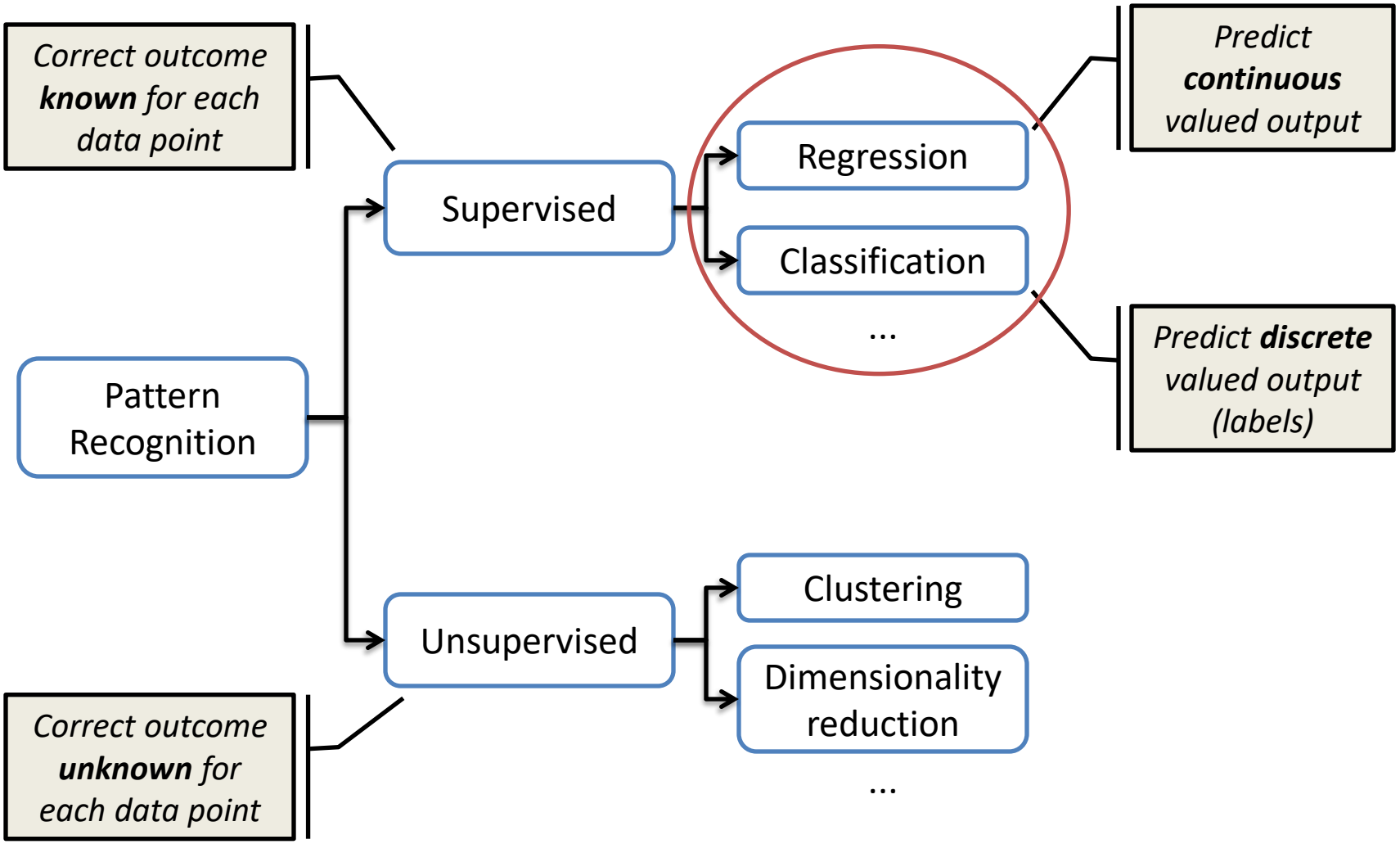
Linear and Polynomial Regression, Gradient Descent, Logistic Regression and Classification

Dimosthenis Karatzas (dimos@cvc.uab.es)

Machine learning



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

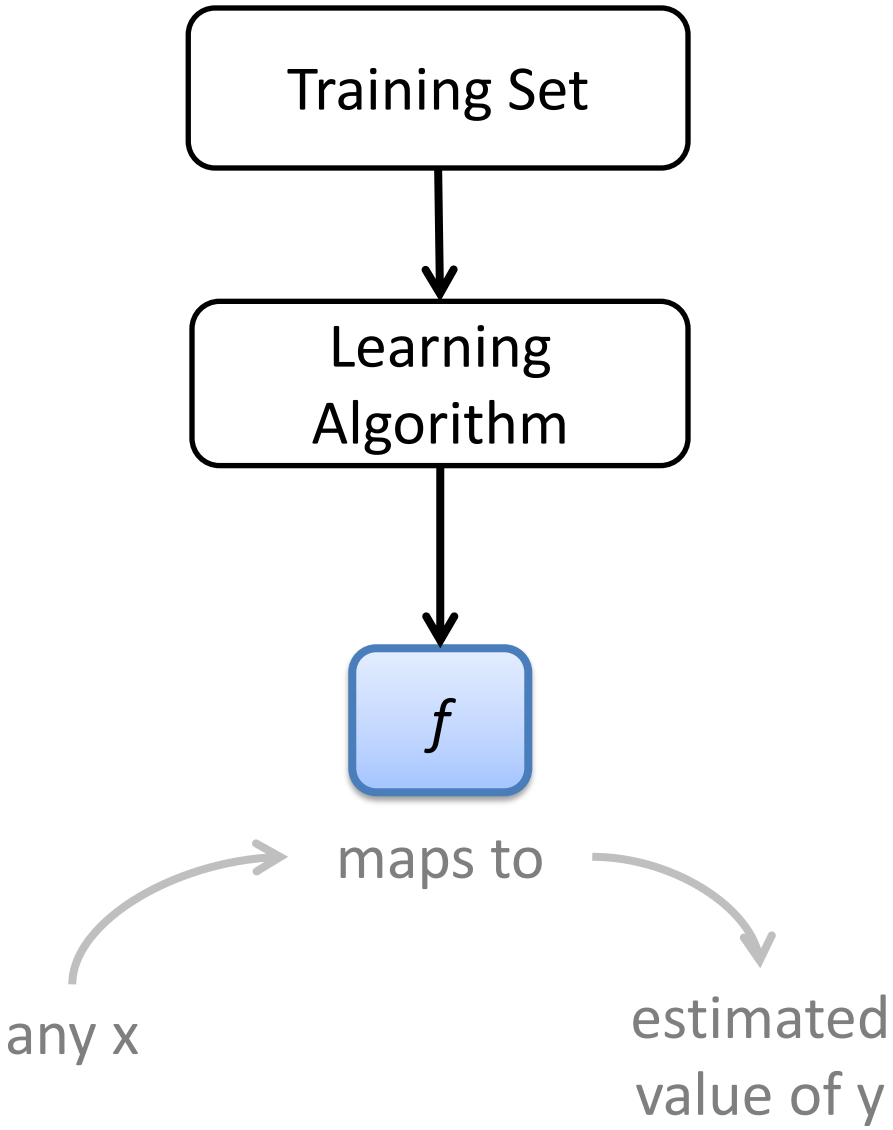
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

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

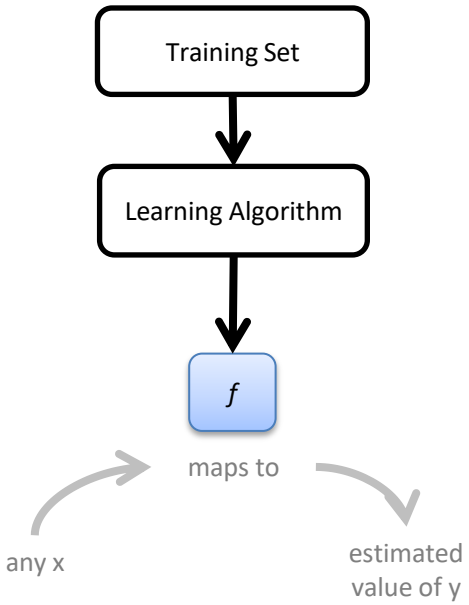
The Process

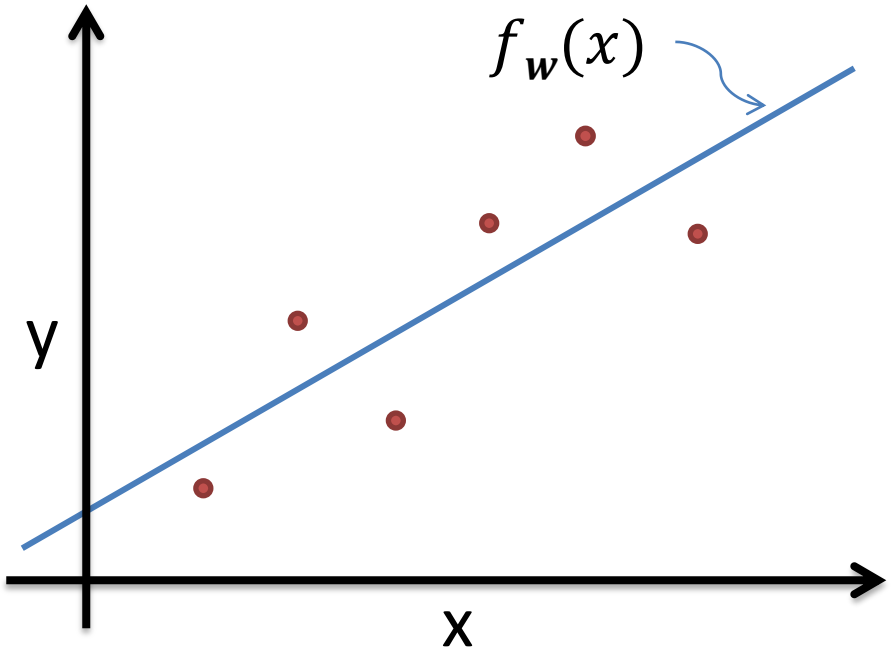

Training Set
Learning Algorithm
f
maps to
any x
estimated value of y

Model Representation

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

$$\hat{y} = f_{\mathbf{w}}(x) = w_0 + w_1 x$$





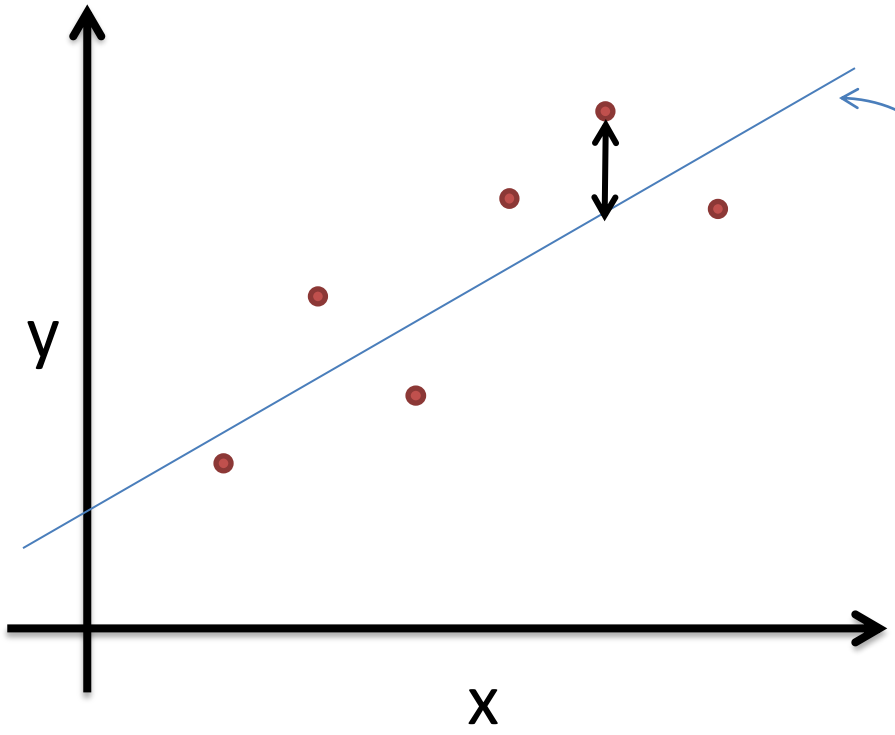
Updated notation

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 (out of m)
$f_{\boldsymbol{w}}(x)$	hypothesis
w_j	j^{th} parameter (weight)

Model representation for univariate linear regression:

$$f_{\boldsymbol{w}}(x) = w_0 + w_1 x$$

The Cost (Loss) Function

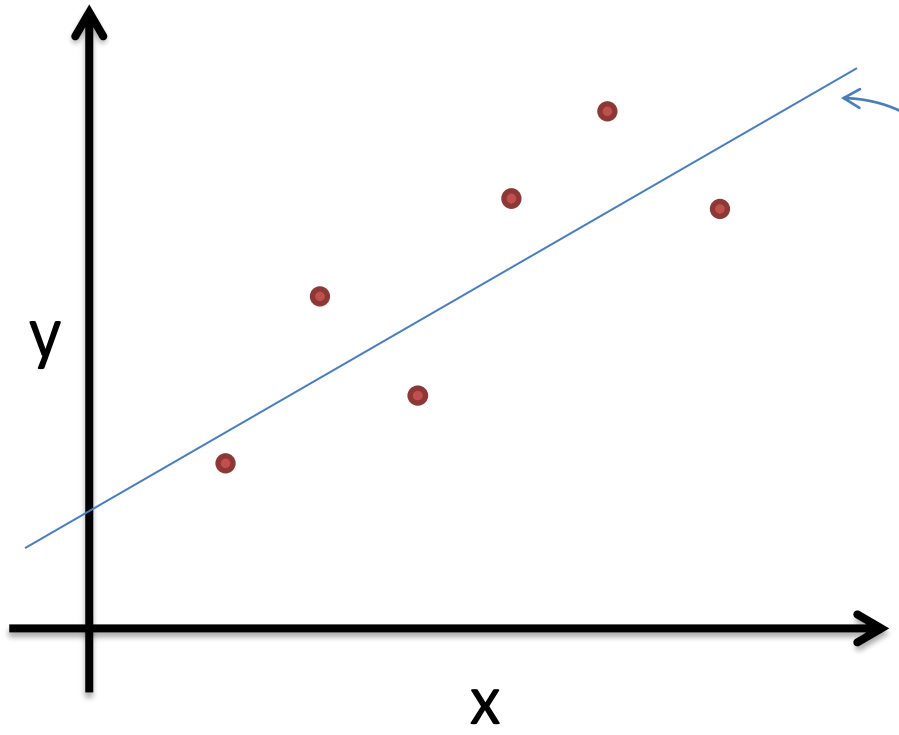


$f_{\boldsymbol{w}}(x)$

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

$J(w_0, w_1)$

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)

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right)^2$$

Insights on the cost function

Hypothesis: $f_w(x) = w_0 + w_1 x$ $\longrightarrow$ $f_w(x) = w_1 x$

0

Parameters: w_0, w_1 $\longrightarrow$ w_1

Cost Function:

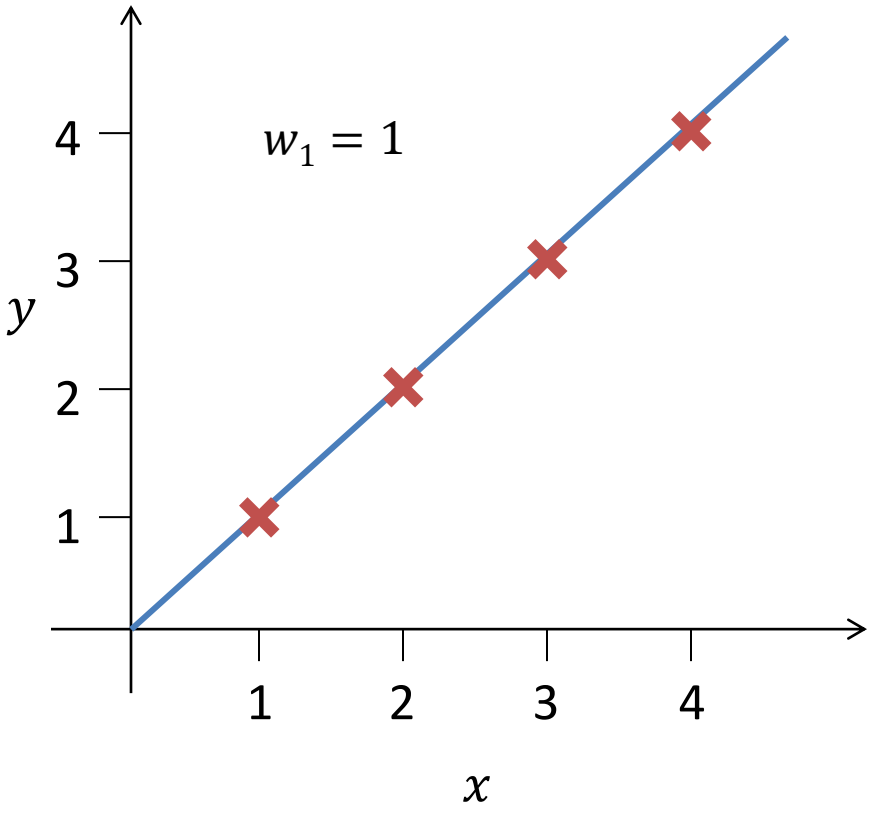
$$J(w_0, w_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right)^2 \longrightarrow J(w_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right)^2$$

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

Insights on the cost function

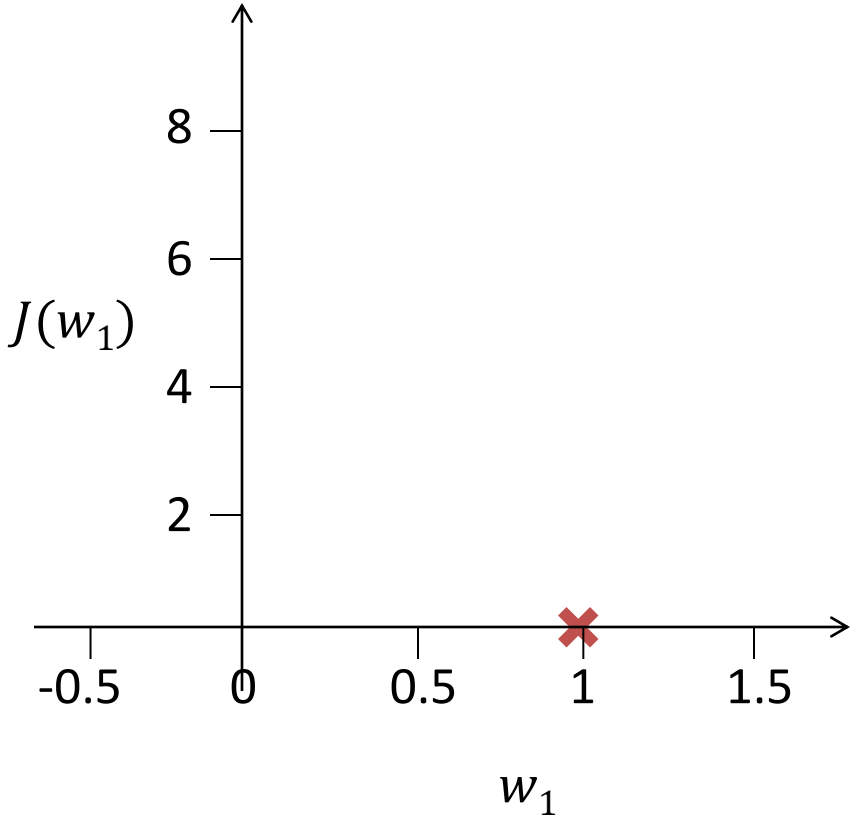
$f_{\boldsymbol{w}}(x)$

(for fixed w_1 this is a function of x)



$J(w_1)$

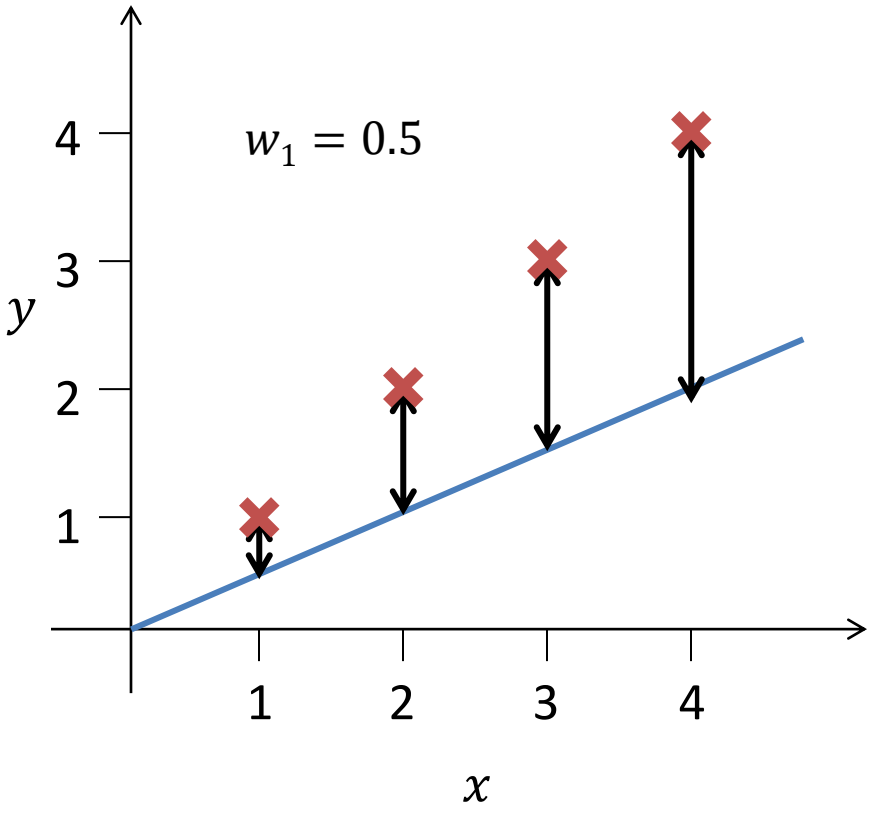
(function of the parameter w_1)



Insights on the cost function

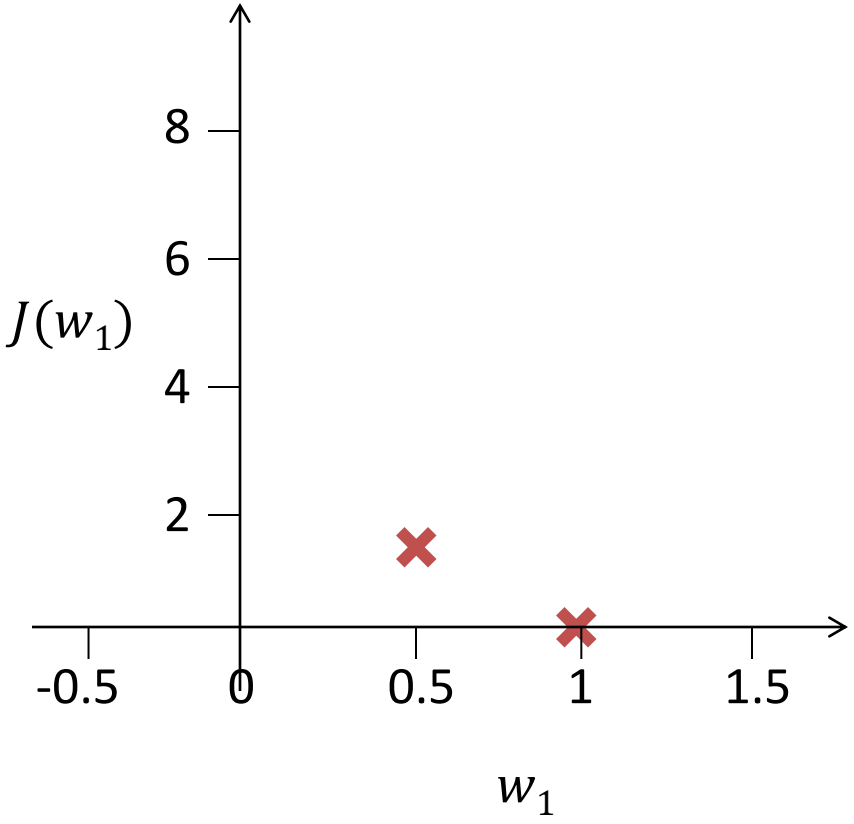
$f_{\boldsymbol{w}}(x)$

(for fixed w_1 this is a function of x)



$J(w_1)$

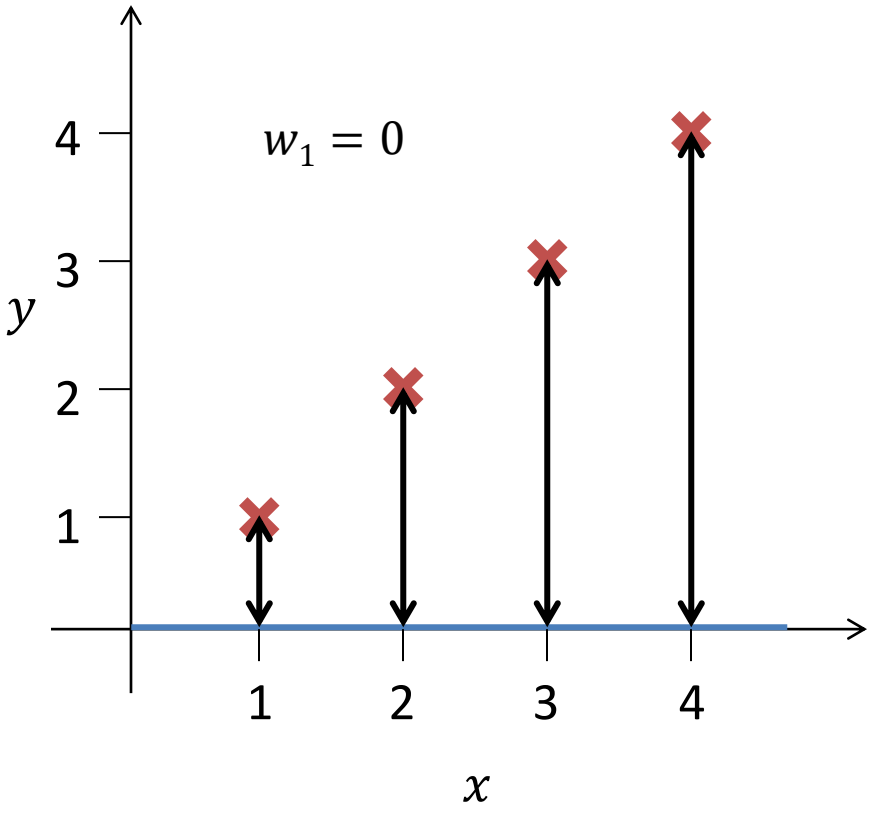
(function of the parameter w_1)



Insights on the cost function

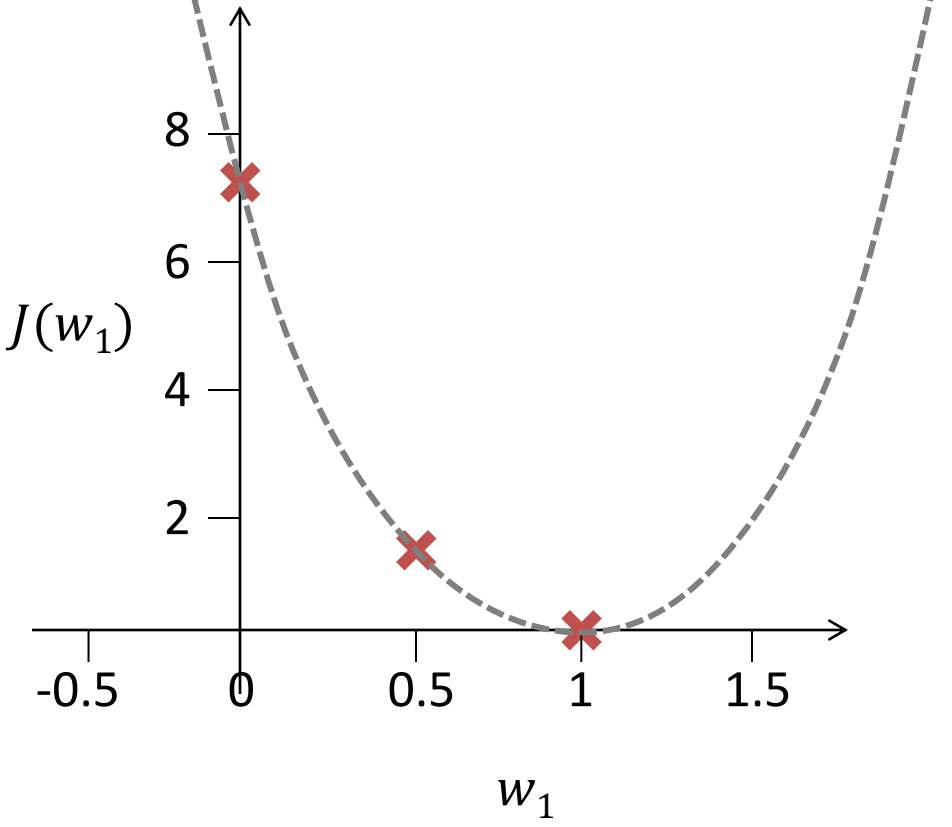
$f_{\boldsymbol{w}}(x)$

(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: $f_w(x) = w_0 + w_1 x$ $\longrightarrow$ $f_w(x) = w_1 x$

0

Parameters: w_0, w_1 $\longrightarrow$ w_1

Cost Function:

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

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

Insights on the cost function

Hypothesis: $f_{\boldsymbol{w}}(x) = w_0 + w_1 x$

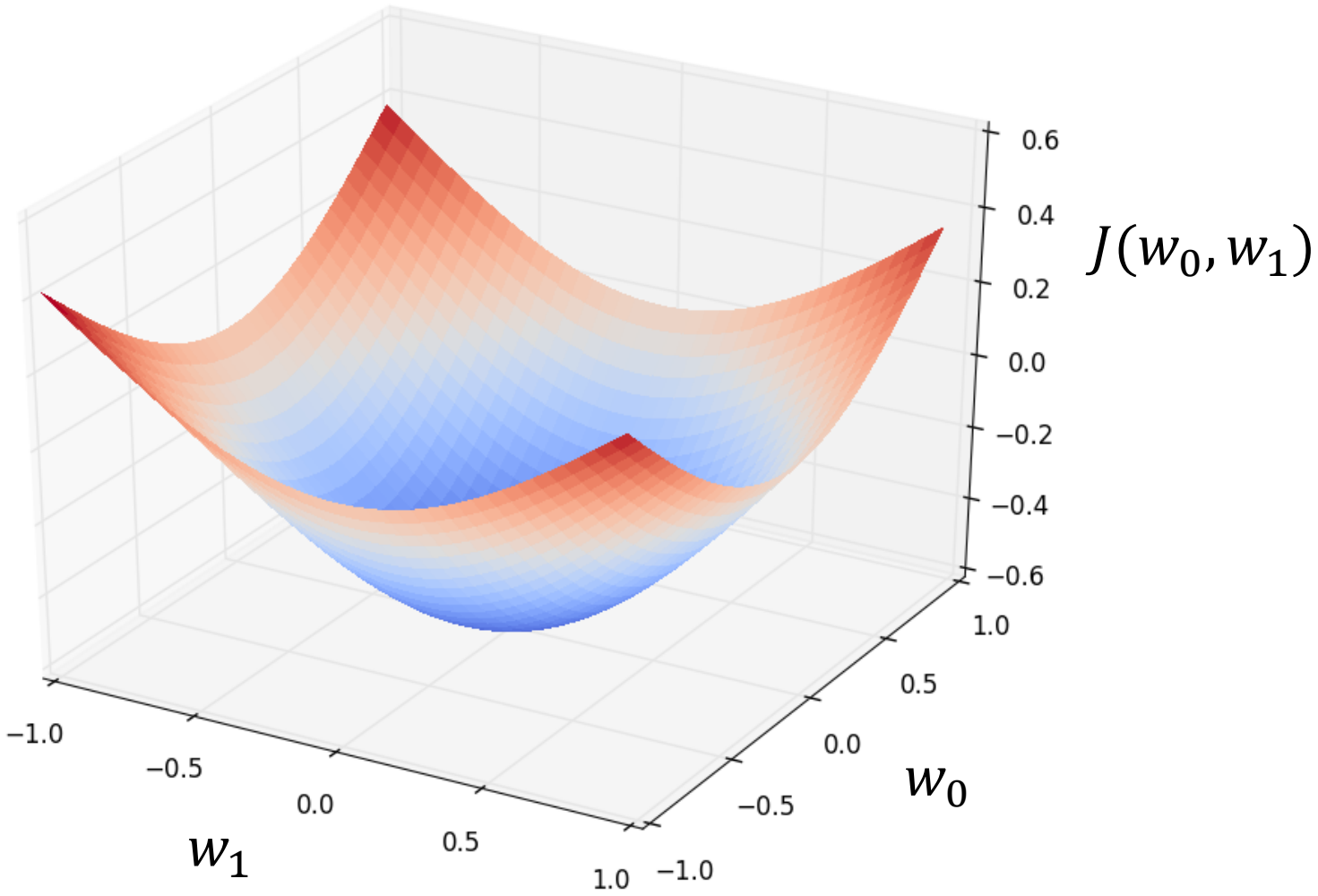
Parameters: w_0, w_1

Cost Function:

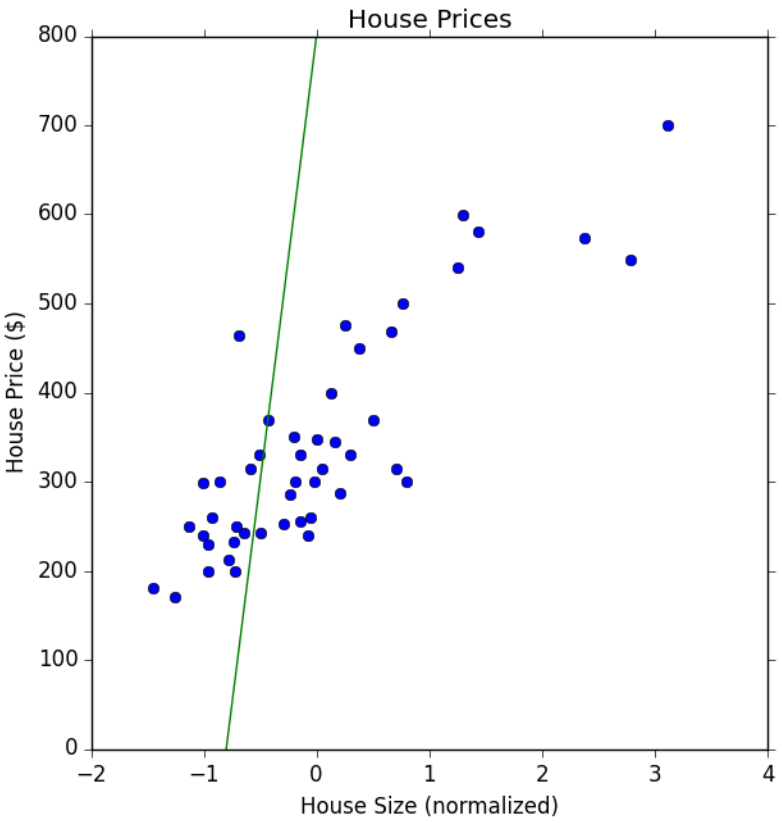
$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w}}\left(x^{(i)}\right) - y^{(i)} \right)^2$$

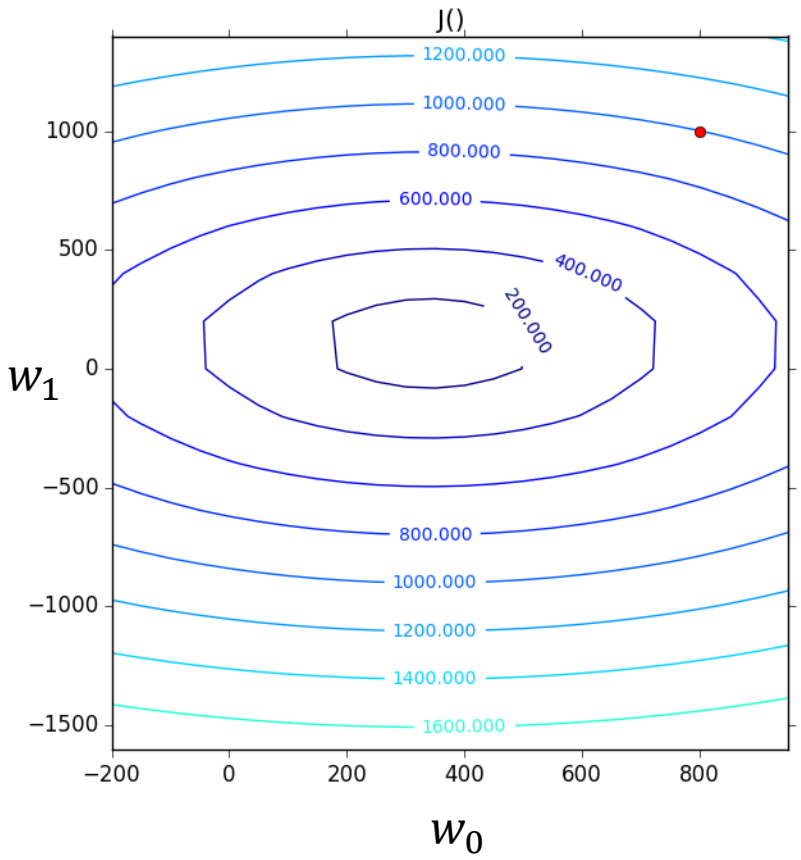
Goal: $\underset{w_0, w_1}{\text{minimise}} \left(J(w_0, w_1) \right)$

Minimizing over two parameters



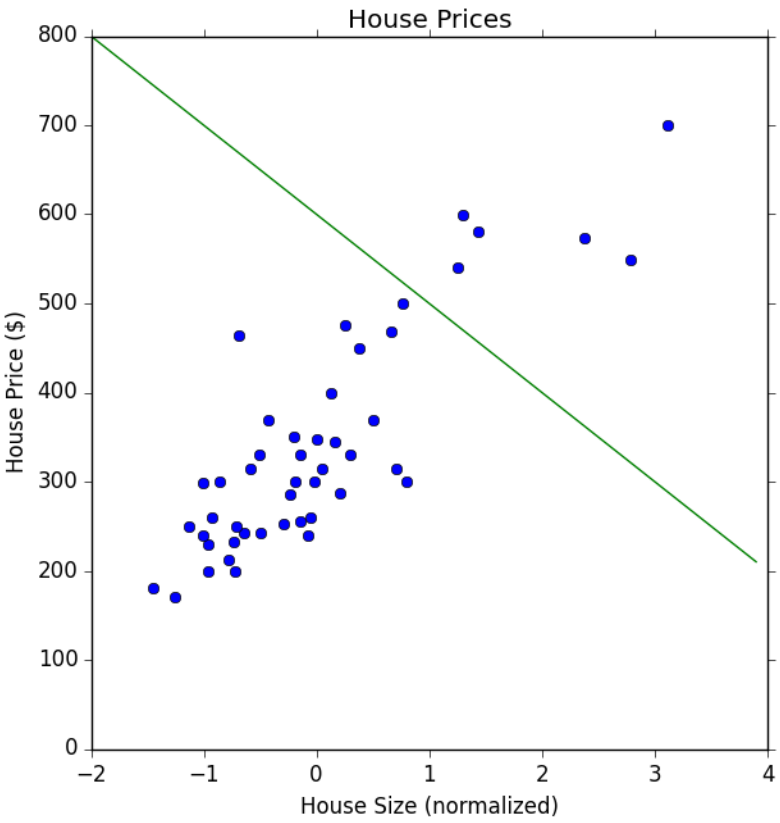
Cost of different hypotheses

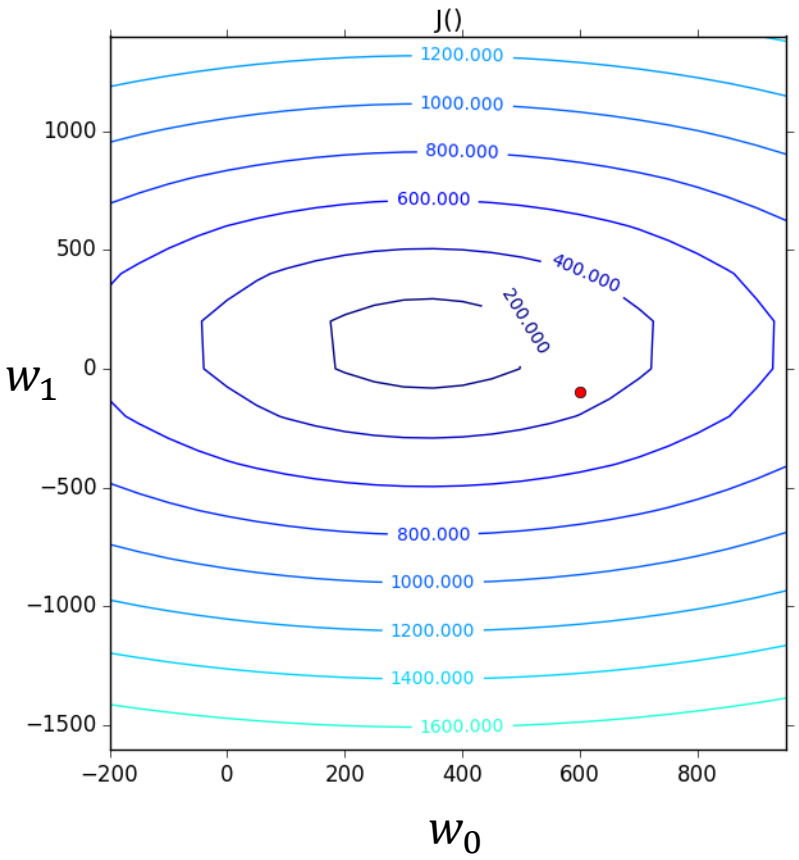




$w_0 = 800, w_1 = 1000$

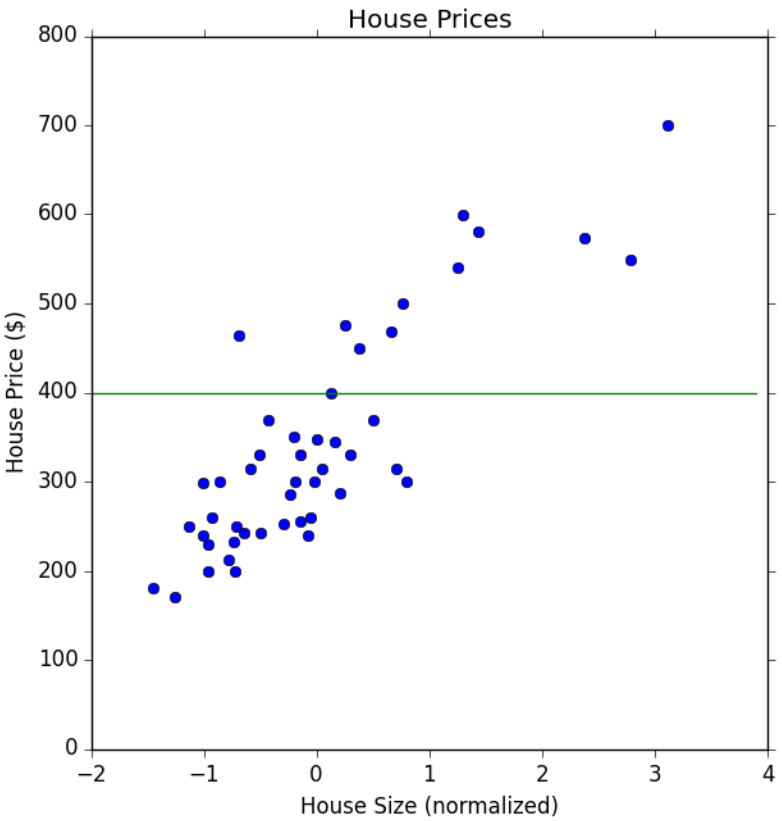
Cost of different hypotheses

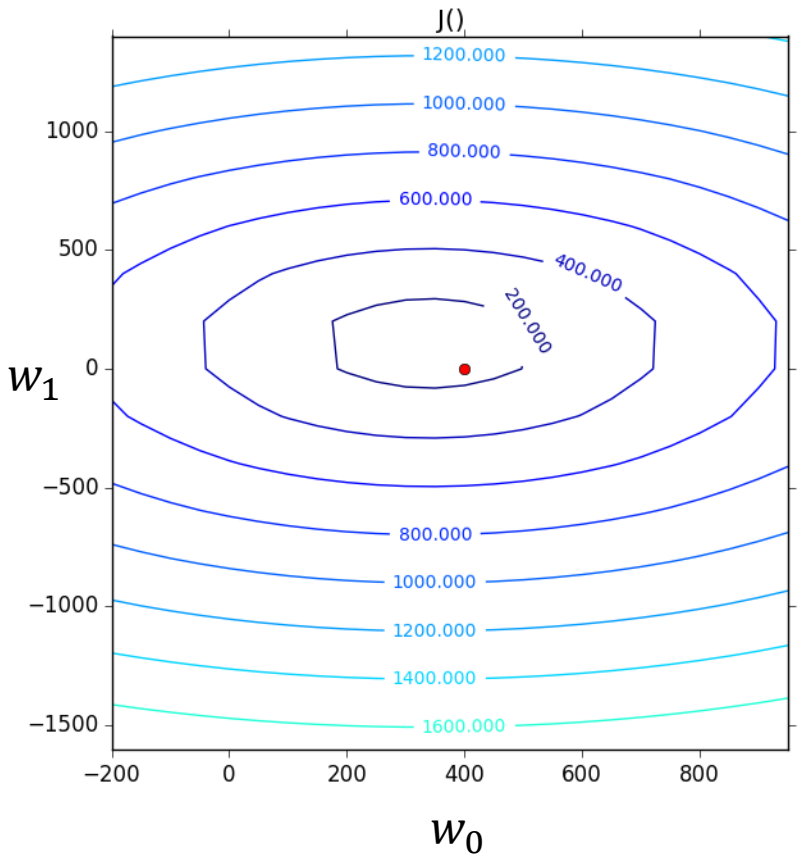




$$w_0 = 600, w_1 = -100$$

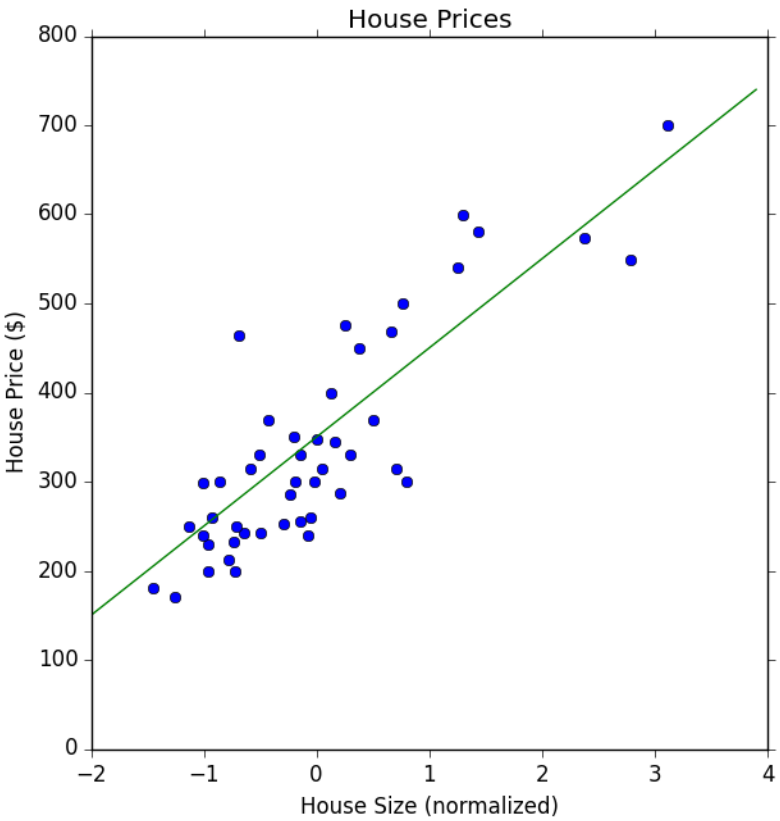
Cost of different hypotheses

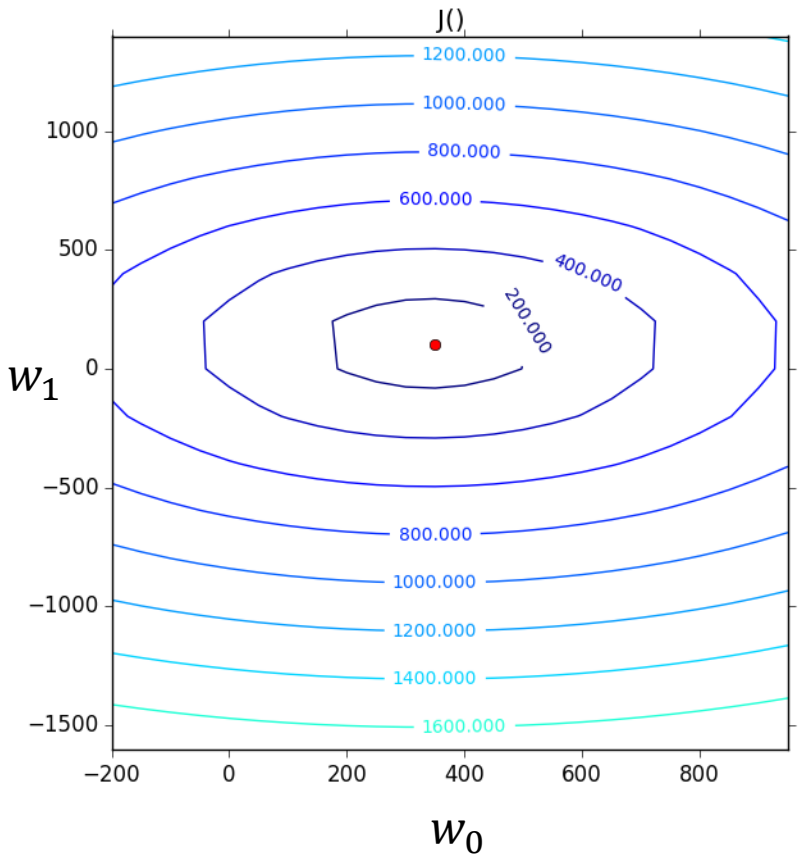




$w_0 = 400, w_1 = 0$

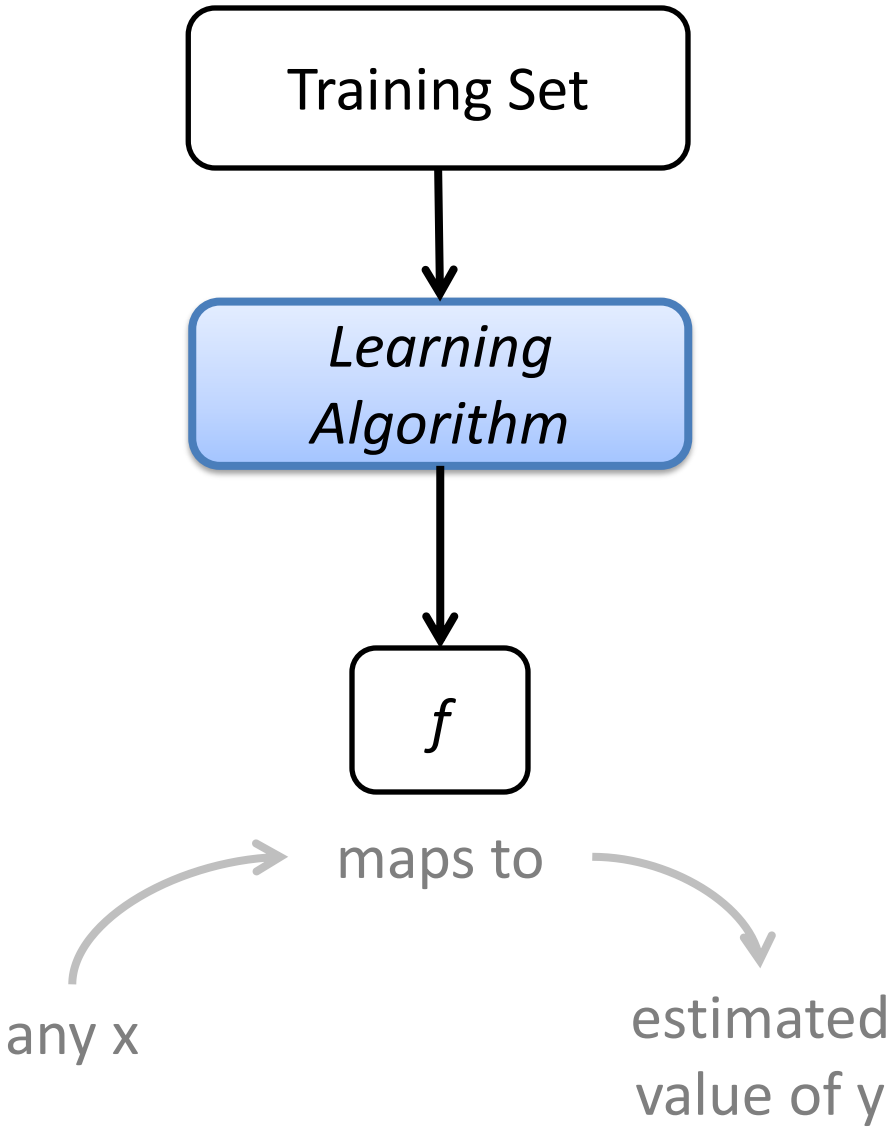
Cost of different hypotheses



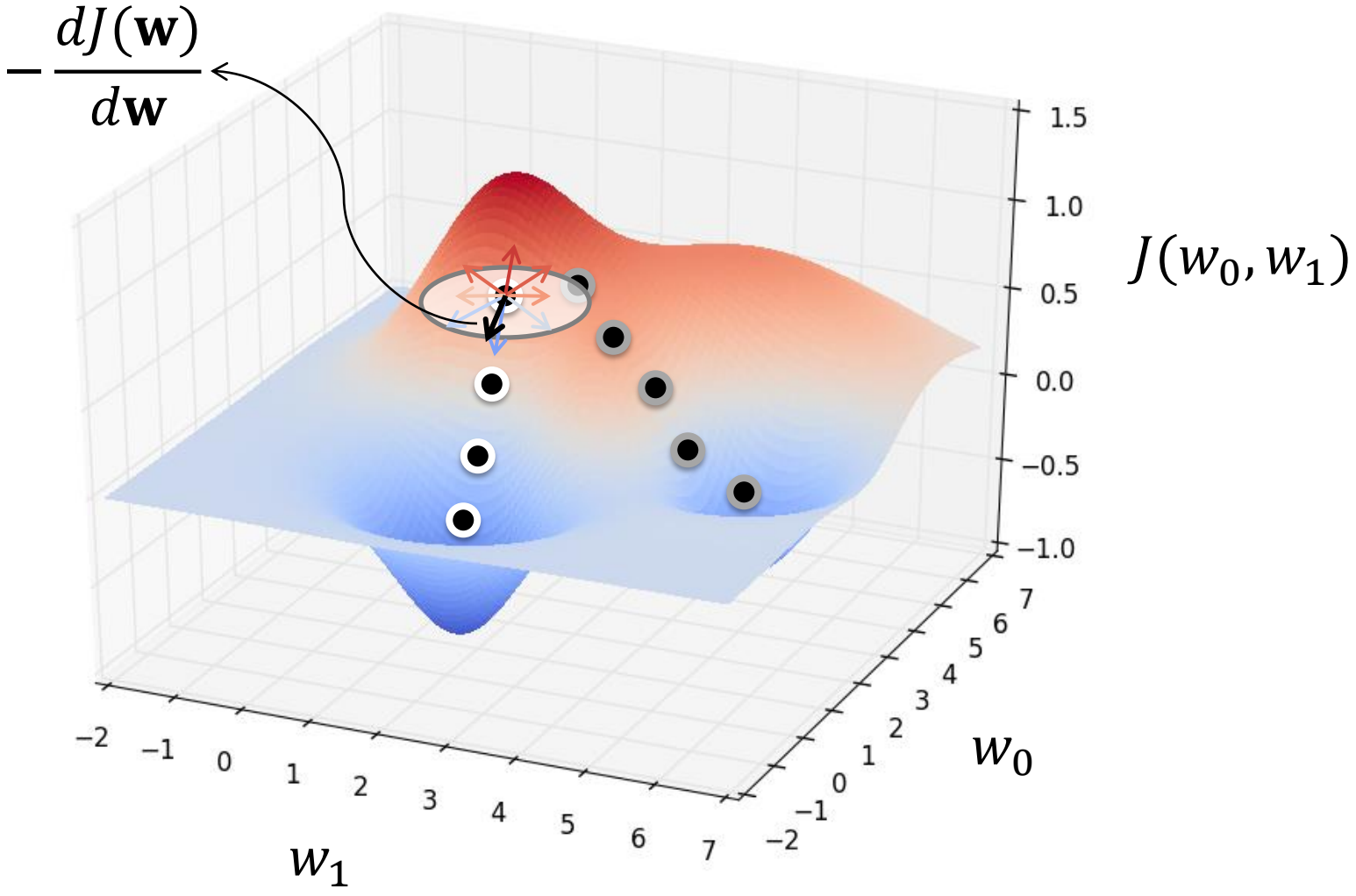


$w_0 = 374, w_1 = 100$

The Process


Training Set
Learning Algorithm
f
maps to
any x
estimated value of y

Gradient Descent Algorithm



Gradient Descent Algorithm

Repeat until convergence

{

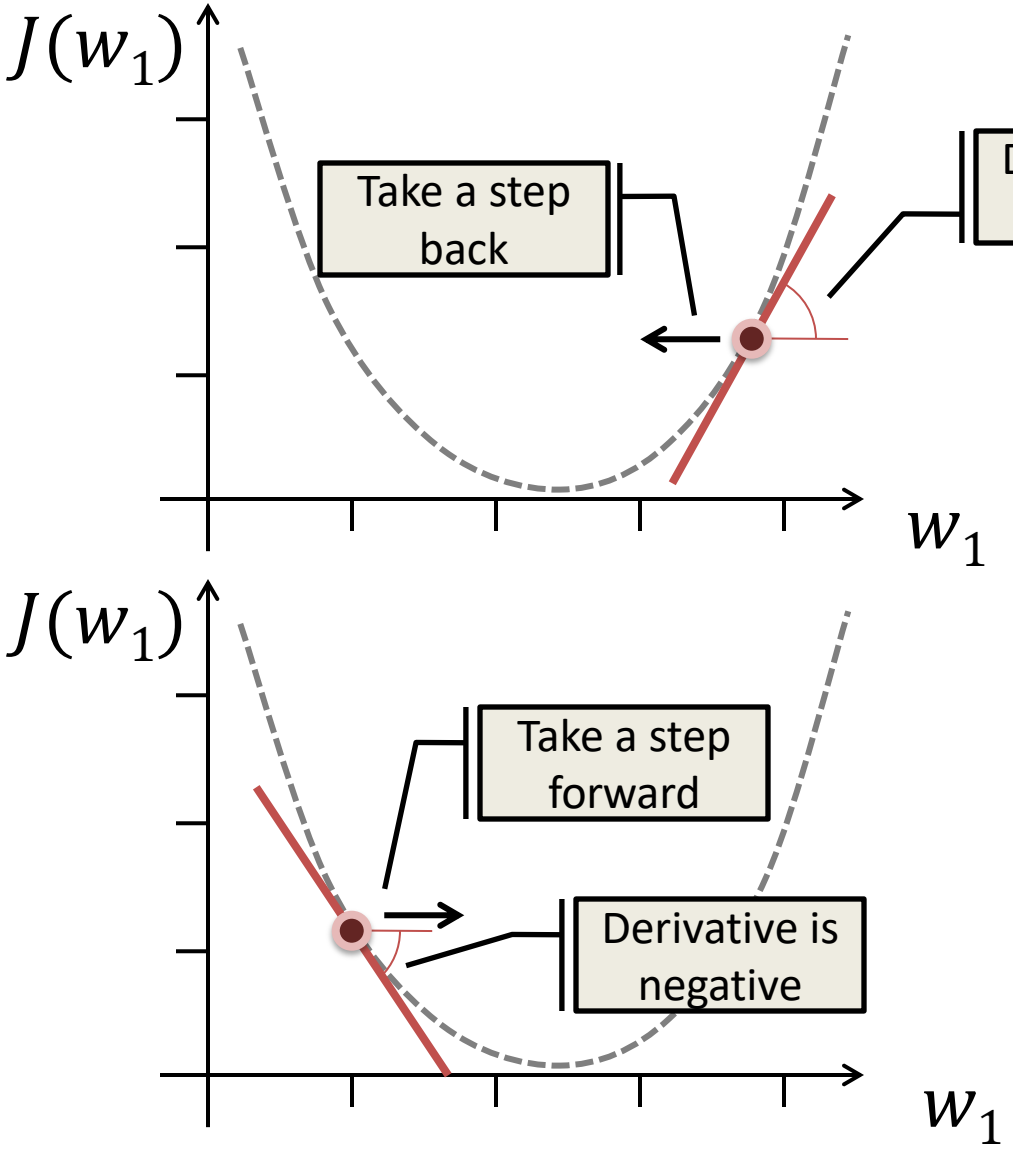
$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1)$$, for (*j=0* and *j=1*)

}

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

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

The derivative – which way

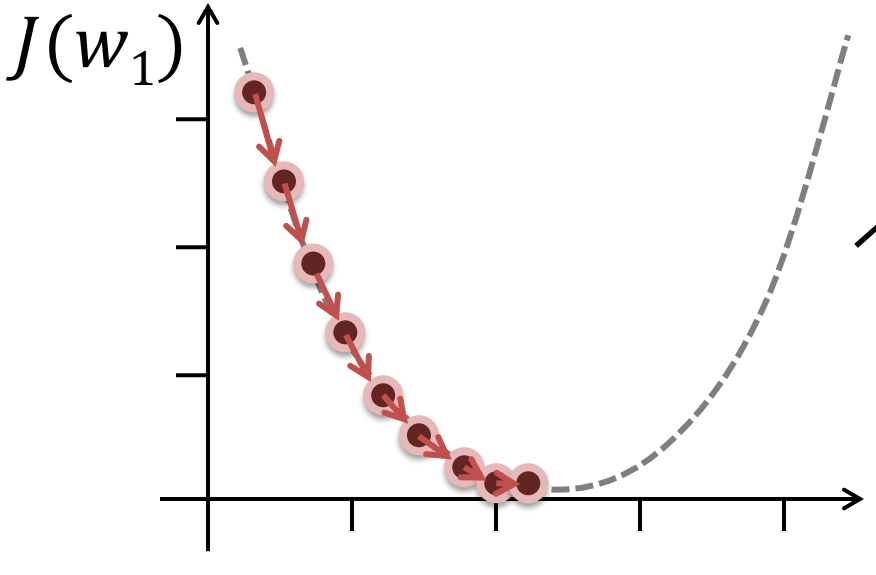


Let's maintain w_0 fixed to simplify this example. Our step change becomes:

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

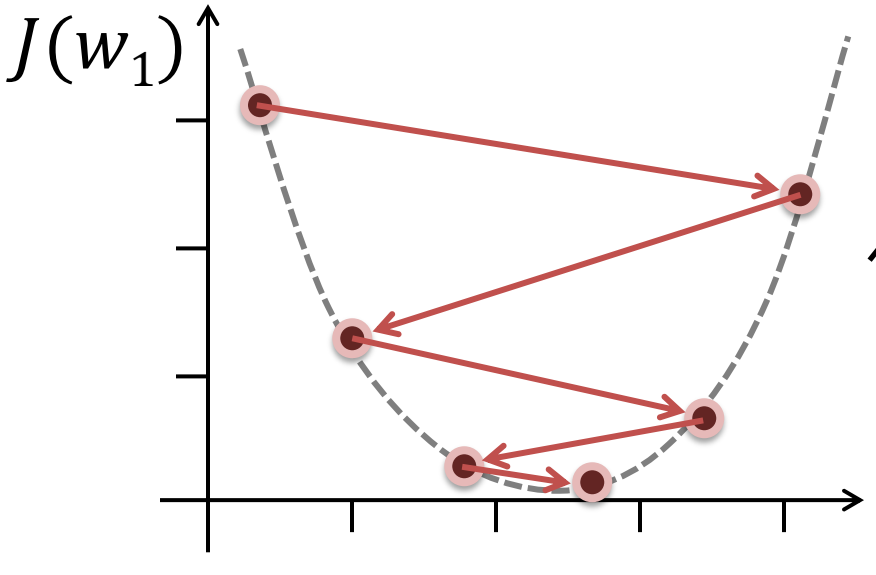
Remember, α is a **positive** number

The learning rate – how fast



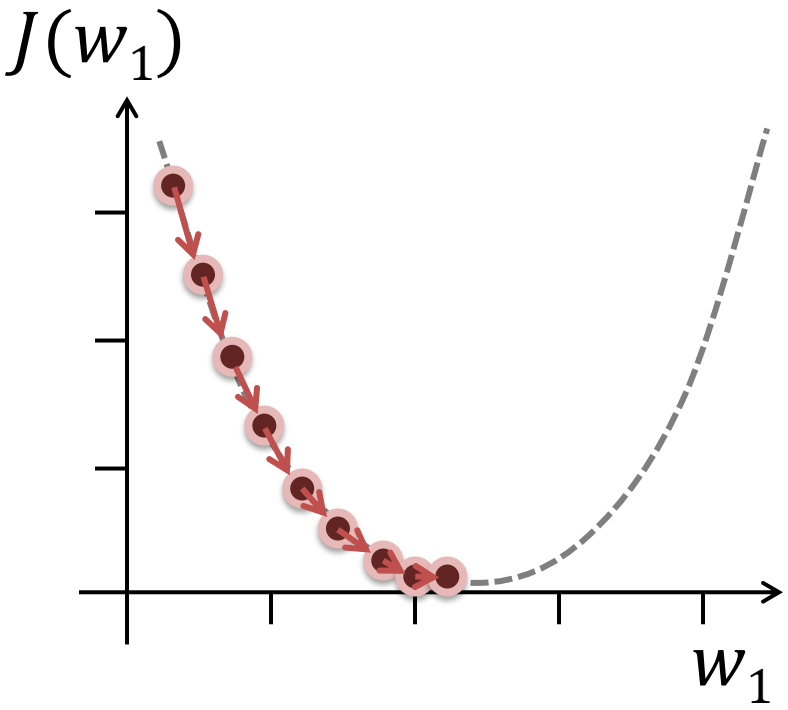
If the learning rate α is too small, it takes many steps to converge

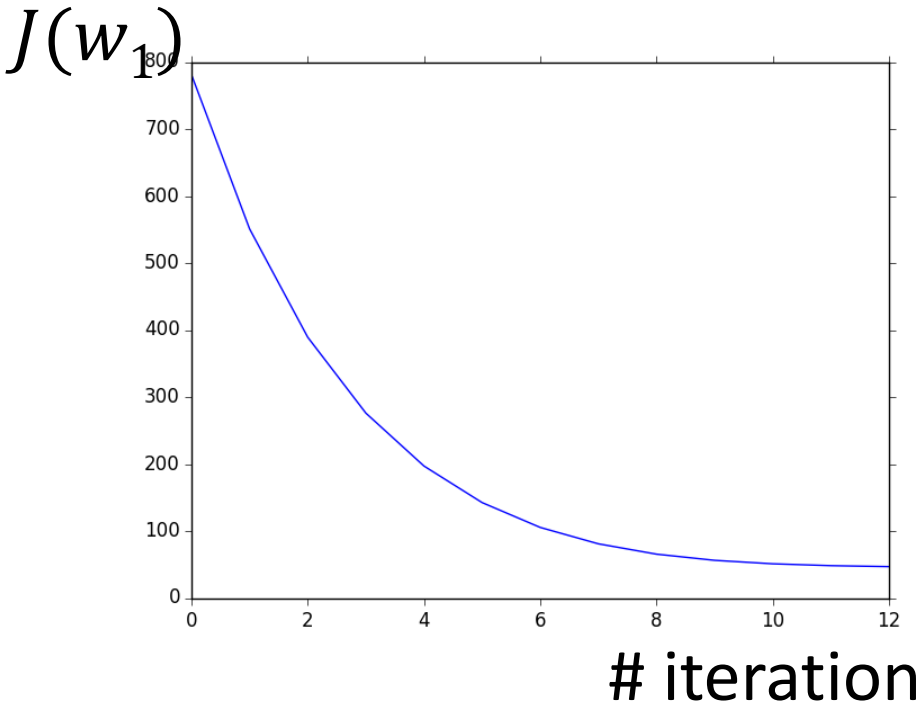
$$w_1 := w_1 - \alpha \frac{d}{dw_1} J(w_1)$$



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

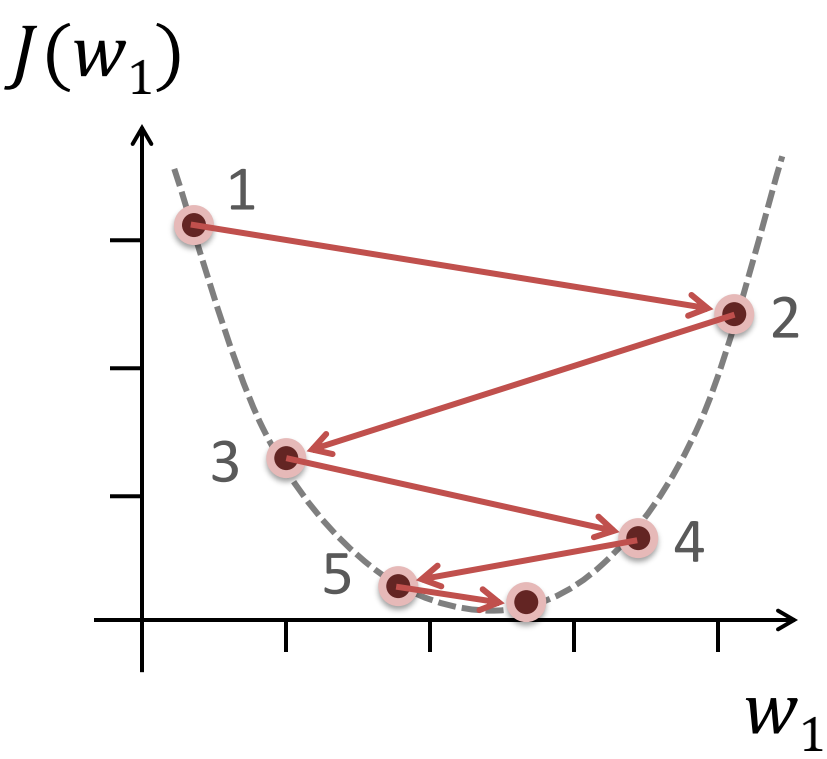


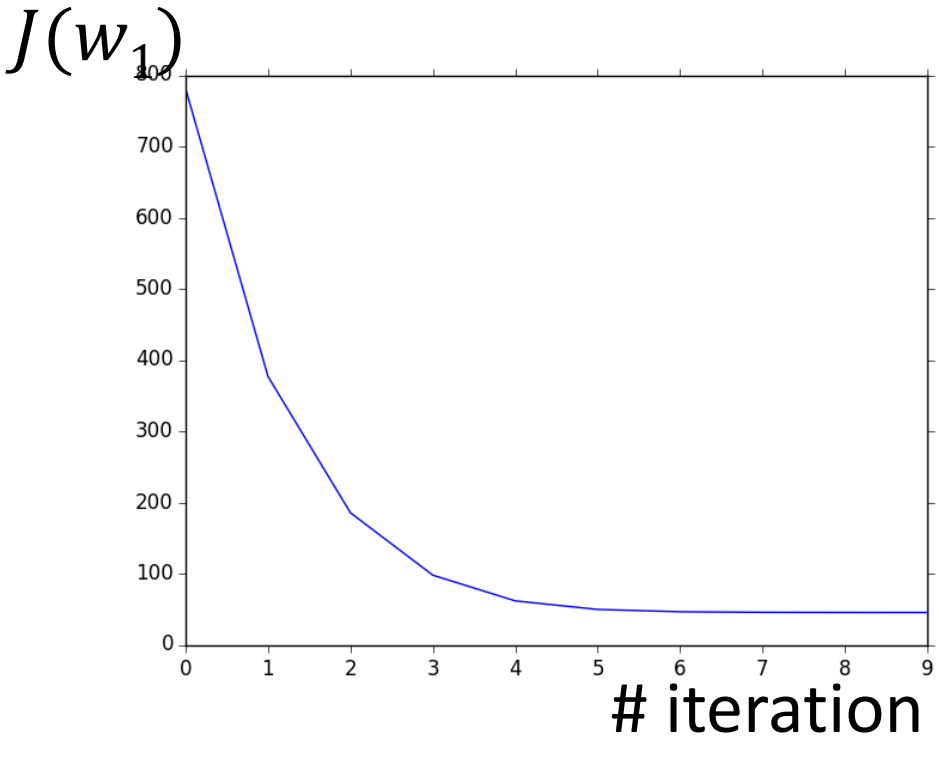


Plotting the evolution of the cost function is a good sanity check for verify that the learning rate is correctly set.

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

Choosing the learning rate

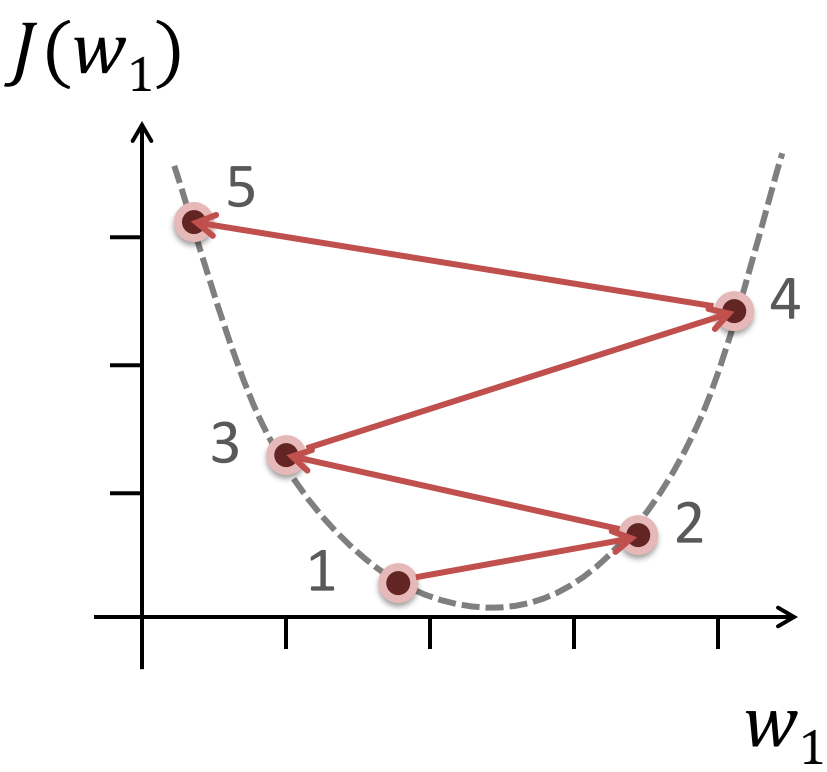


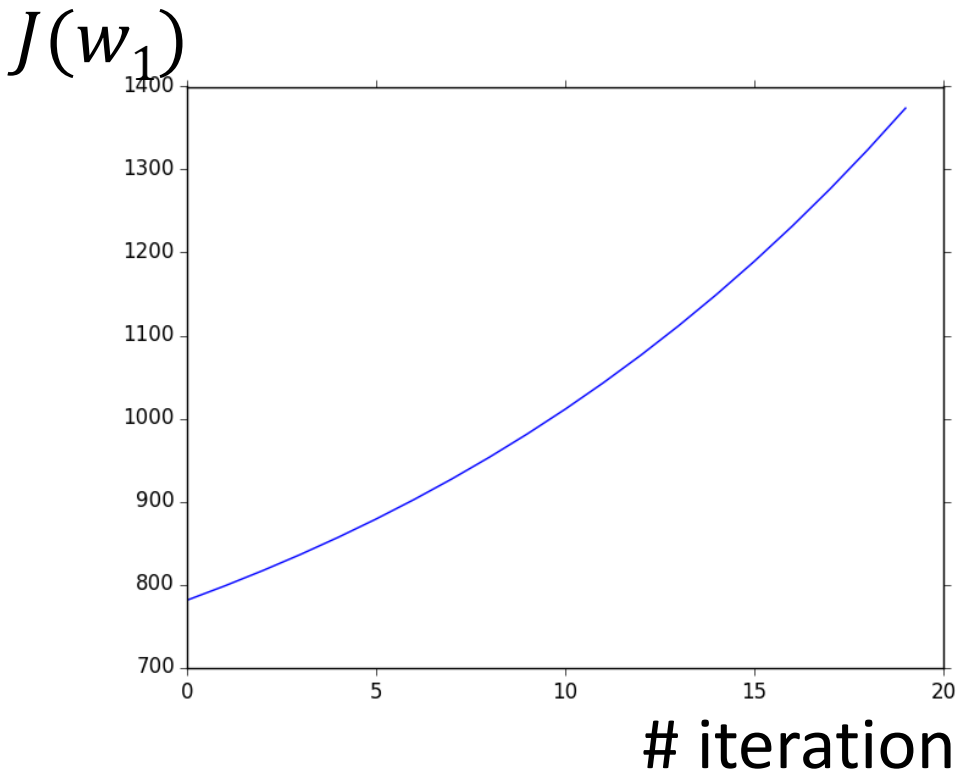


But, overshooting the minimum does not necessarily mean that the algorithm will diverge

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

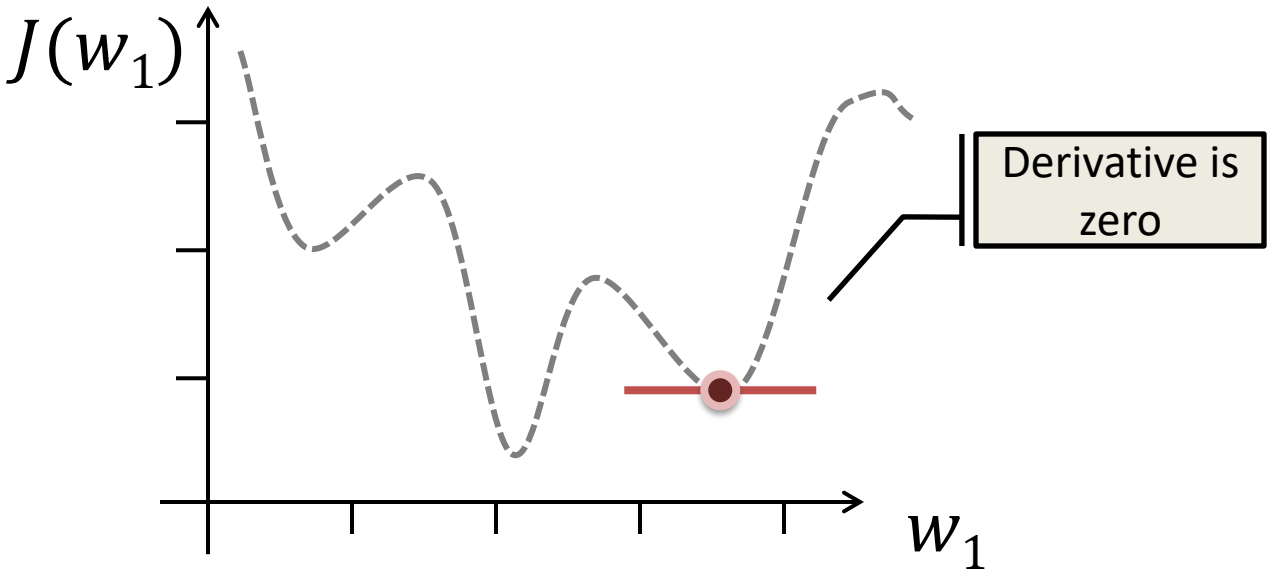
Choosing the learning rate





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

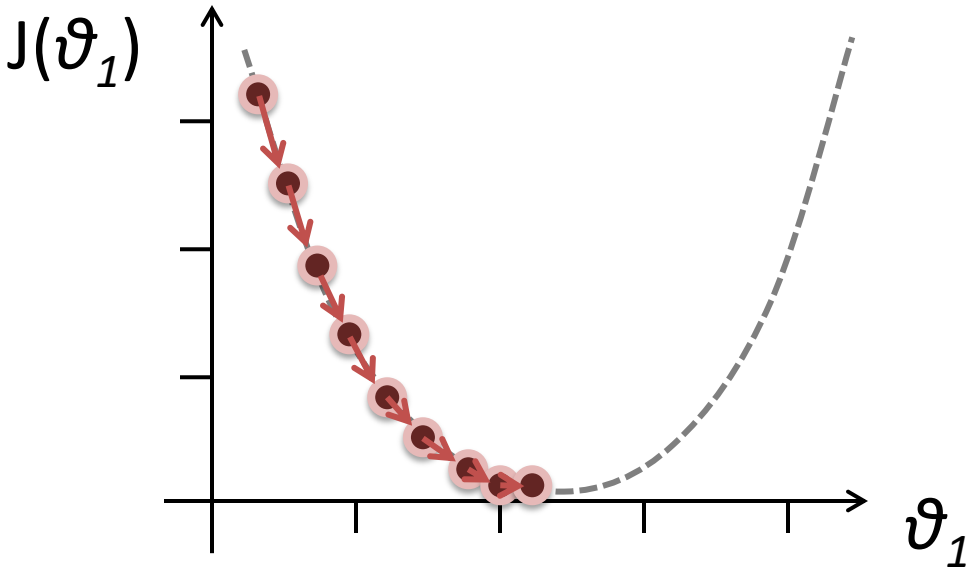
Converging



What happens when we reach a local minimum?

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

Converging

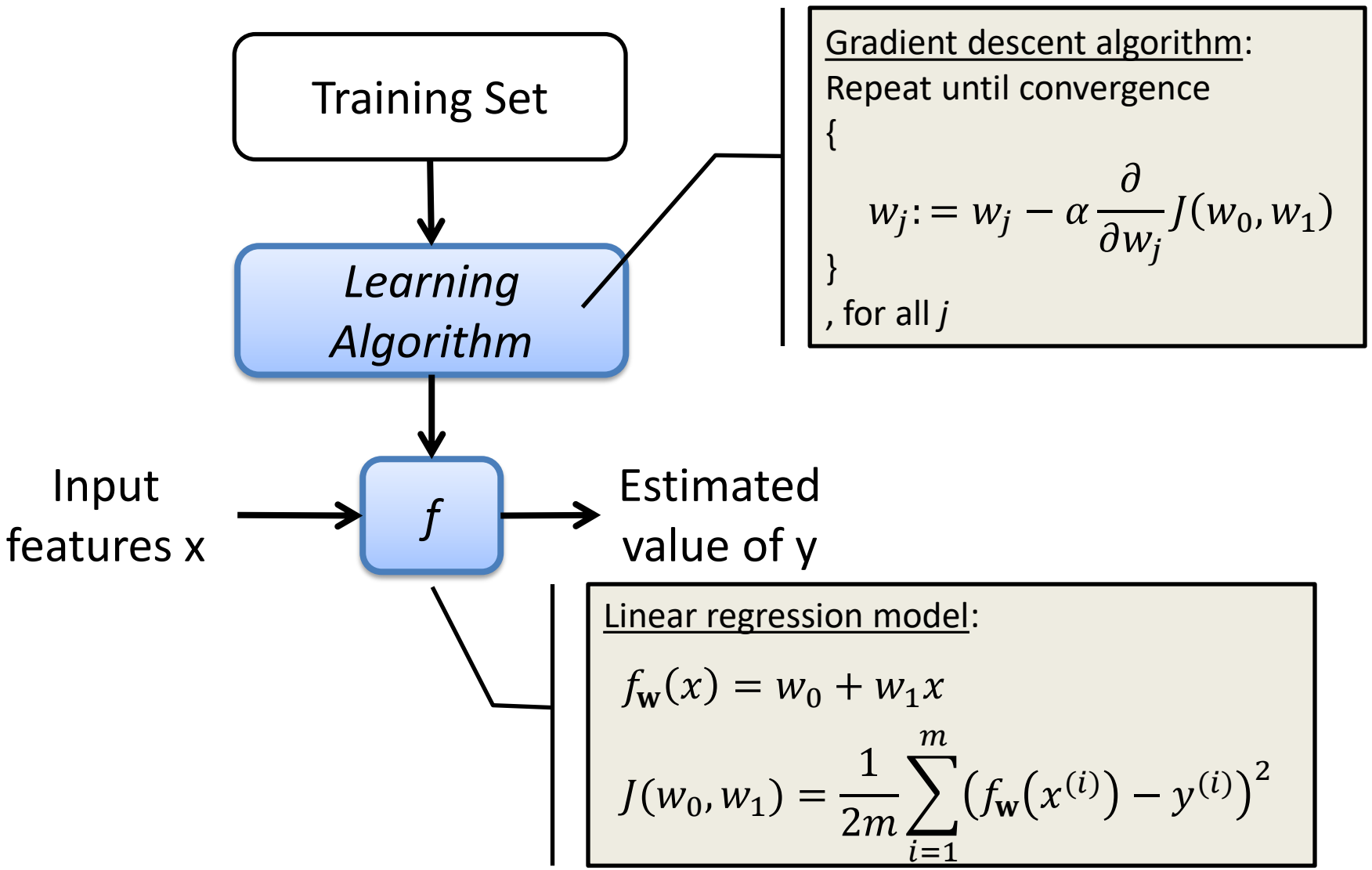


$$w_1 := w_1 - \alpha \frac{d}{dw_1} J(w_1)$$

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

$$\frac{\partial}{\partial \mathrm{w}_j} J(\mathrm{w}_0, \mathrm{w}_1) = \frac{\partial}{\partial \mathrm{w}_j} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right)^2$$

$$= \frac{\partial}{\partial \mathrm{w}_j} \frac{1}{2m} \sum_{i=1}^{m} \left(\mathrm{w}_0 + \mathrm{w}_1 x^{(i)} - y^{(i)} \right)^2$$

$$j = 0: \frac{\partial}{\partial \mathrm{w}_0} J(\mathrm{w}_0, \mathrm{w}_1) = \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right)$$

$$j = 1: \frac{\partial}{\partial \mathrm{w}_1} J(\mathrm{w}_0, \mathrm{w}_1) = \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right) \cdot x^{(i)}$$

Gradient Descent Algorithm

Repeat until convergence

{

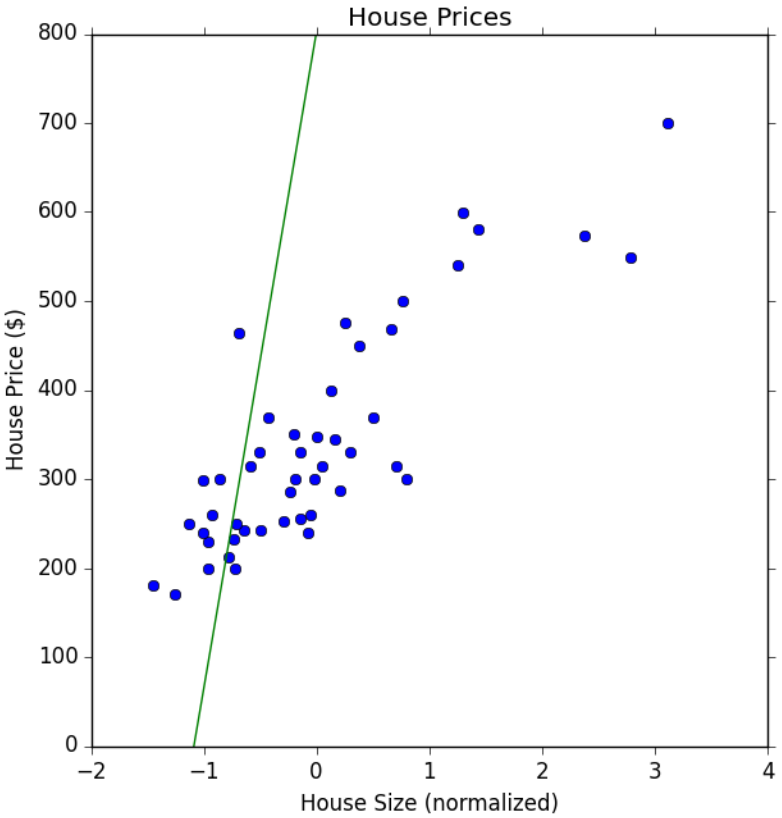
$$\mathrm{w}_0 := \mathrm{w}_0 - \alpha \frac{\partial}{\partial \mathrm{w}_0} J(\mathrm{w}_0, \mathrm{w}_1) = \mathrm{w}_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right)$$

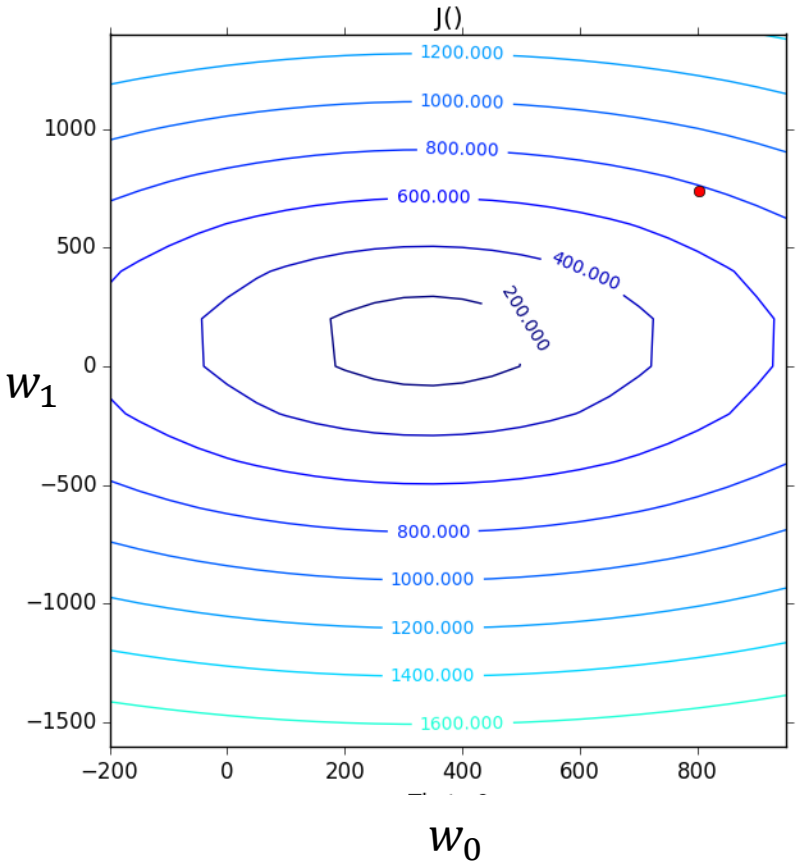
$$\mathrm{w}_1 := \mathrm{w}_1 - \alpha \frac{\partial}{\partial \mathrm{w}_1} J(\mathrm{w}_0, \mathrm{w}_1) = \mathrm{w}_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right) \cdot x^{(i)}$$

}

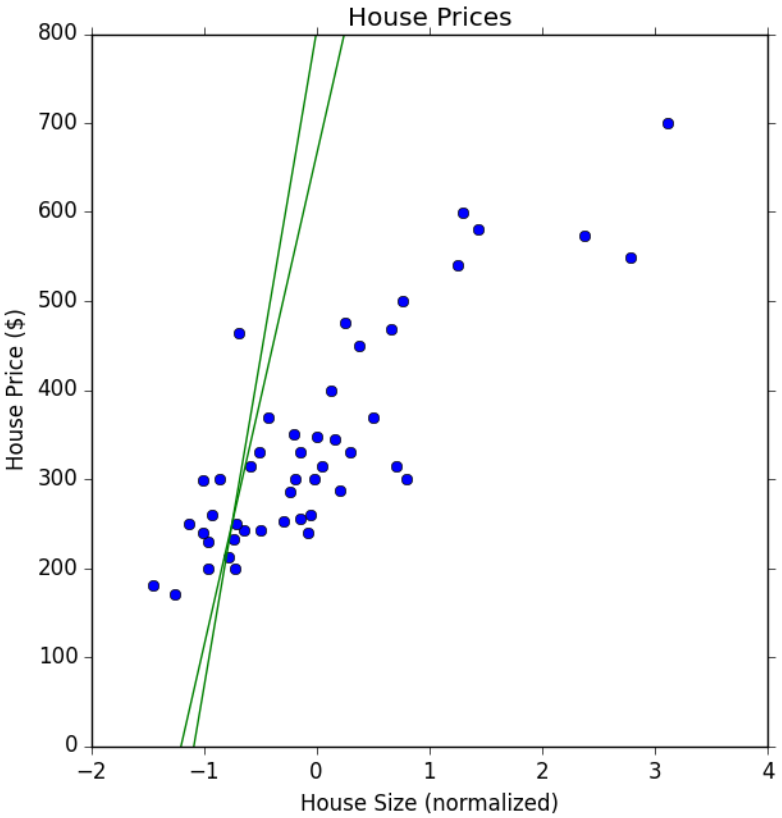
Where w_0 and w_1 should be updated simultaneously

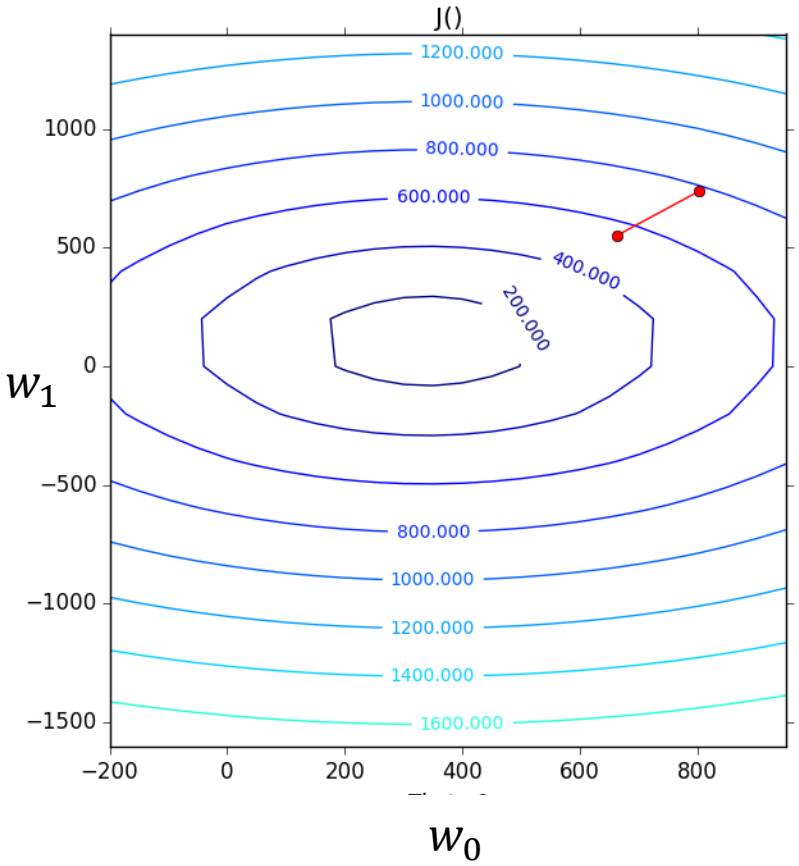
Step by Step


House Prices
House Price ($)
House Size (normalized)
800
700
600
500
400
300
200
100
0
−2
−1
0
1
2
3
4

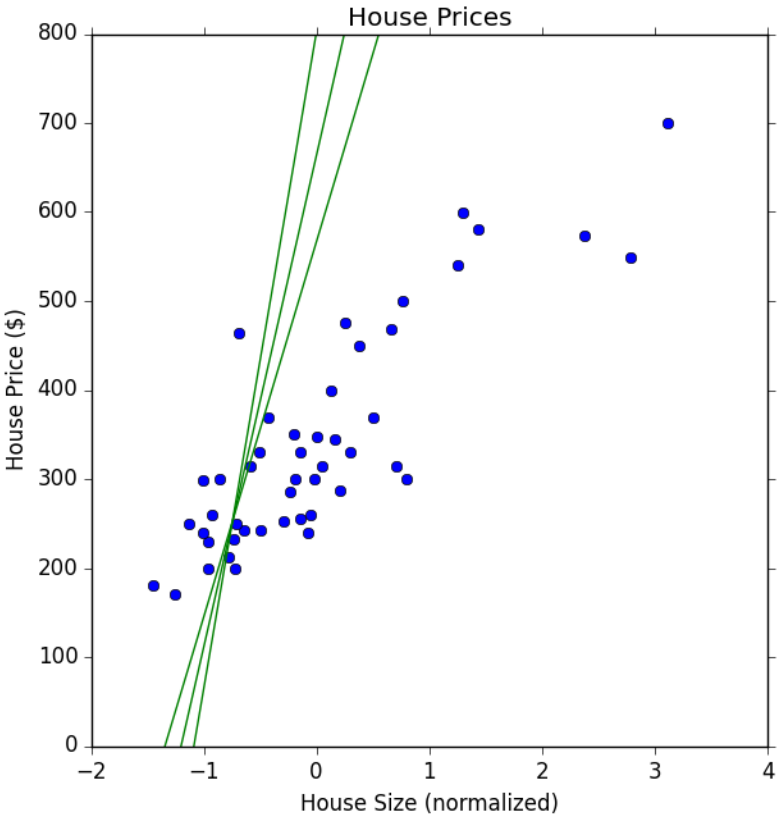

J()
1200.000
1000.000
800.000
600.000
400.000
200.000
800.000
1000.000
1200.000
1400.000
1600.000
1000
500
0
−500
−1000
−1500
−200
0
200
400
600
800
w_1
w_0

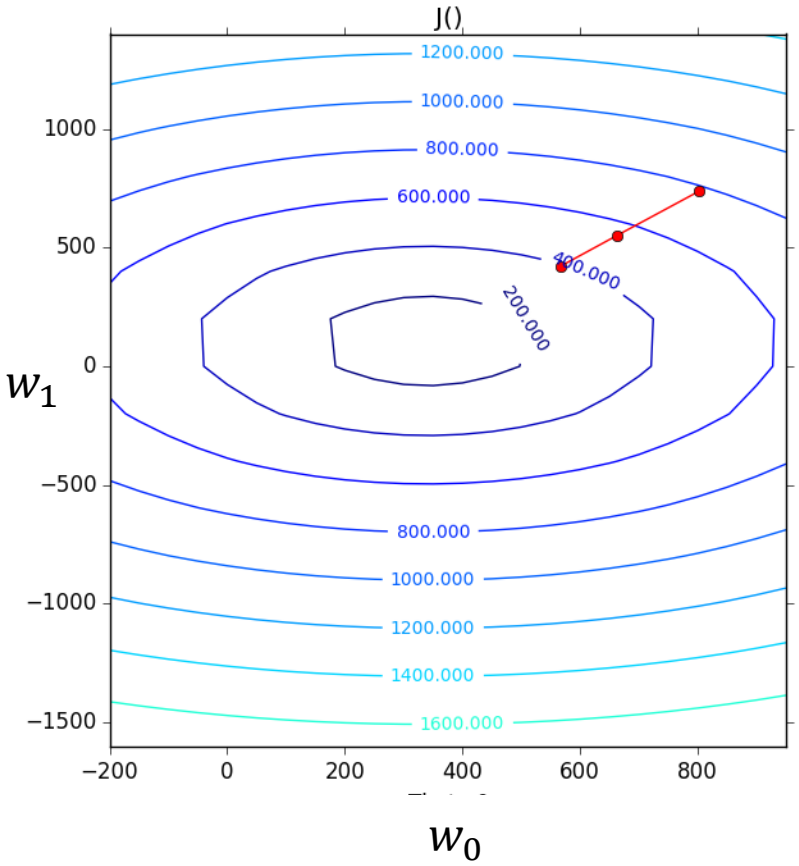
Step by Step


House Prices
House Price ($)
House Size (normalized)
800
700
600
500
400
300
200
100
0
−2
−1
0
1
2
3
4

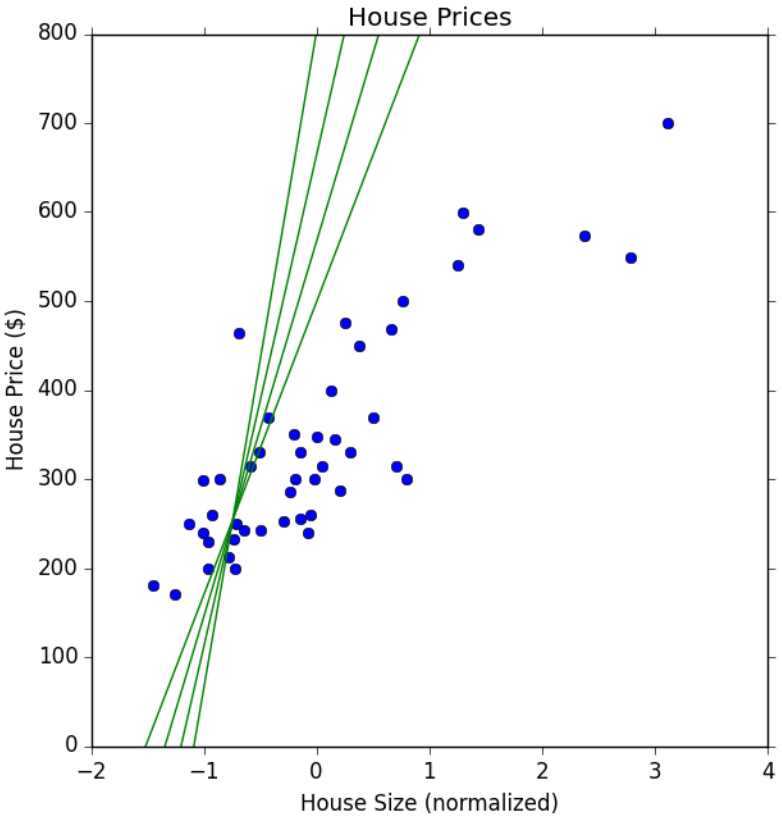

J()
1200.000
1000.000
800.000
600.000
400.000
200.000
800.000
1000.000
1200.000
1400.000
1600.000
1000
500
0
−500
−1000
−1500
−200
0
200
400
600
800
w_1
w_0

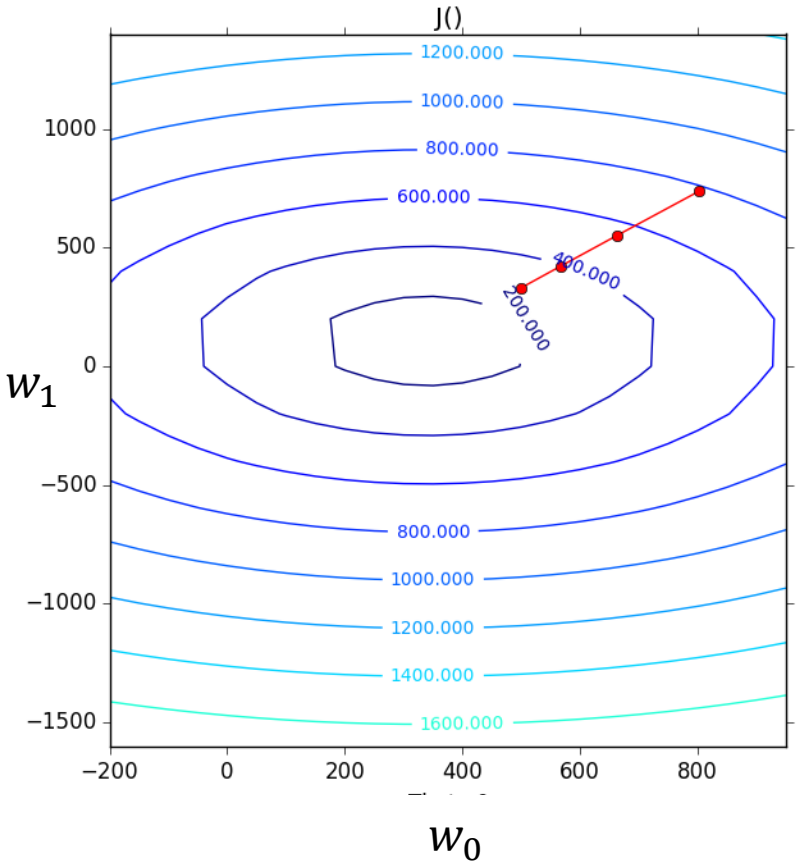
Step by Step


House Prices
House Price ($)
House Size (normalized)
800
700
600
500
400
300
200
100
0
−2
−1
0
1
2
3
4

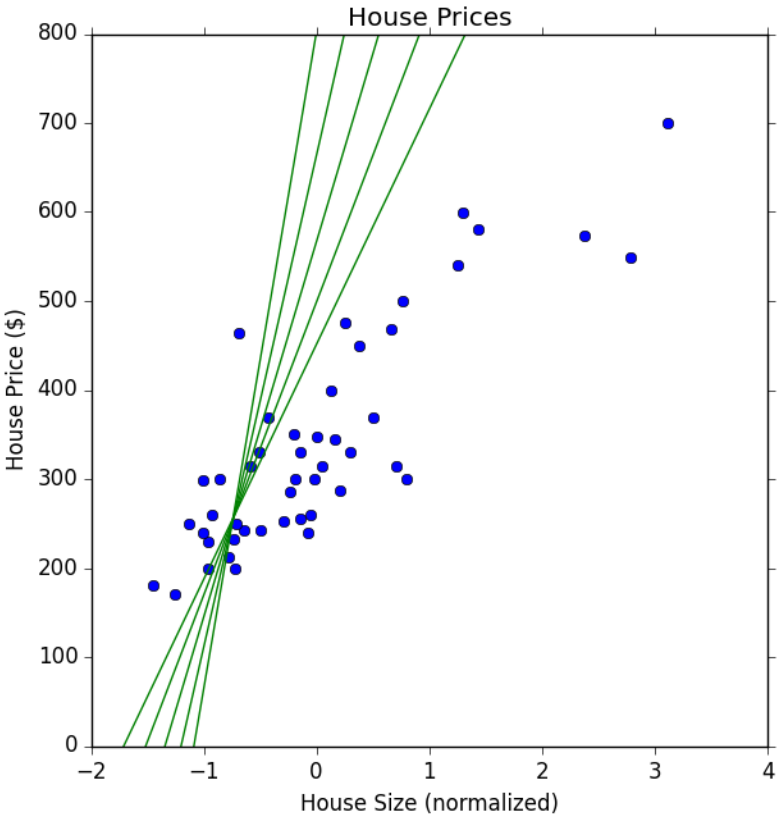

J()
1200.000
1000.000
800.000
600.000
400.000
200.000
800.000
1000.000
1200.000
1400.000
1600.000
1000
500
0
−500
−1000
−1500
−200
0
200
400
600
800
w_1
w_0

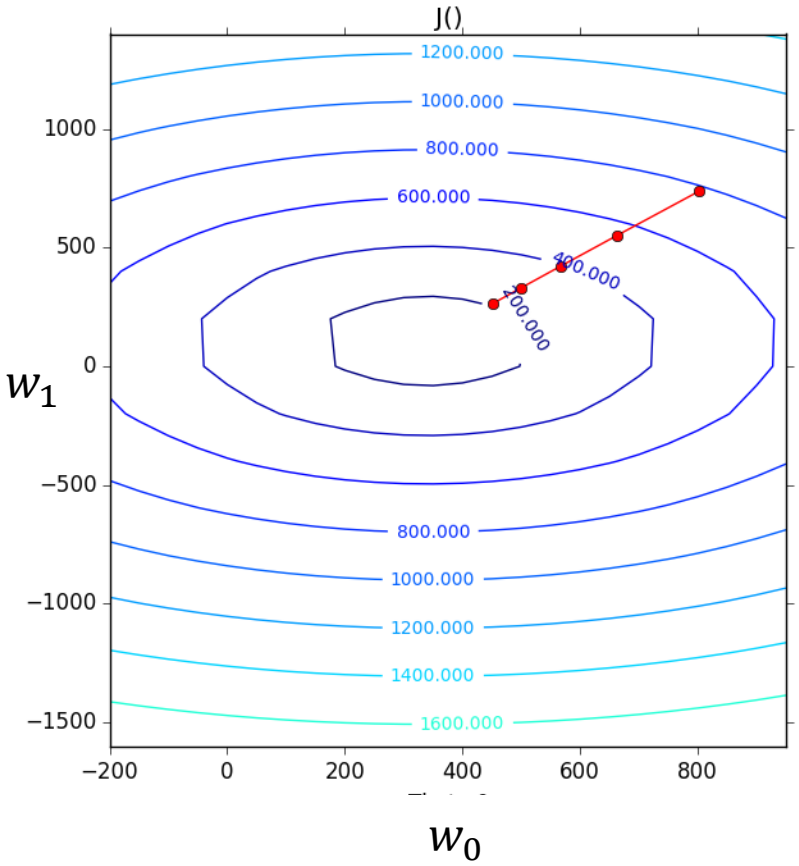
Step by Step


House Prices
House Price ($)
House Size (normalized)
800
700
600
500
400
300
200
100
0
−2
−1
0
1
2
3
4

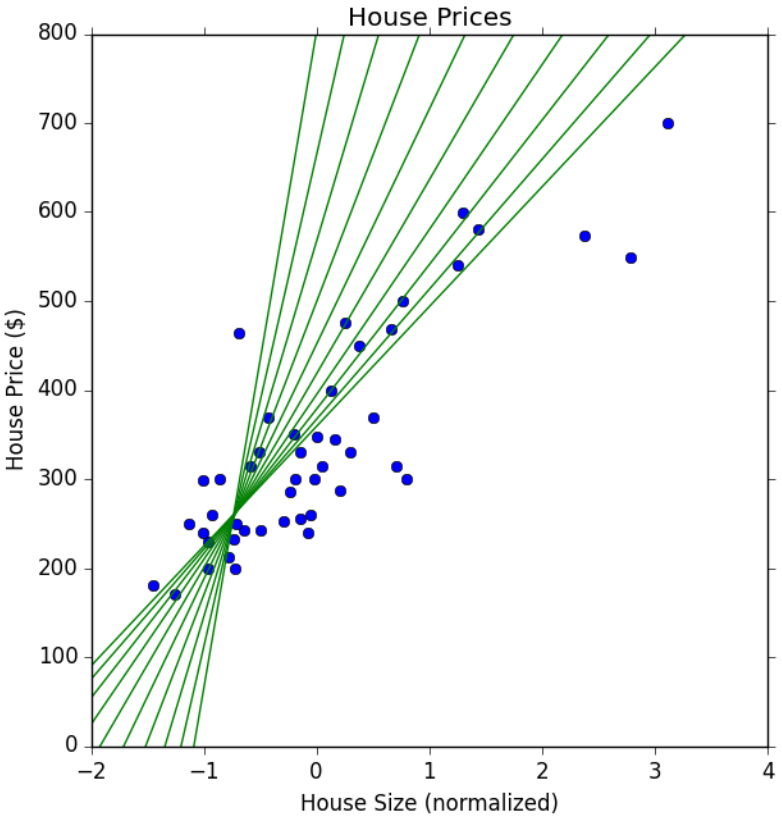

J()
1200.000
1000.000
800.000
600.000
400.000
200.000
800.000
1000.000
1200.000
1400.000
1600.000
1000
500
0
−500
−1000
−1500
−200
0
200
400
600
800
w_1
w_0

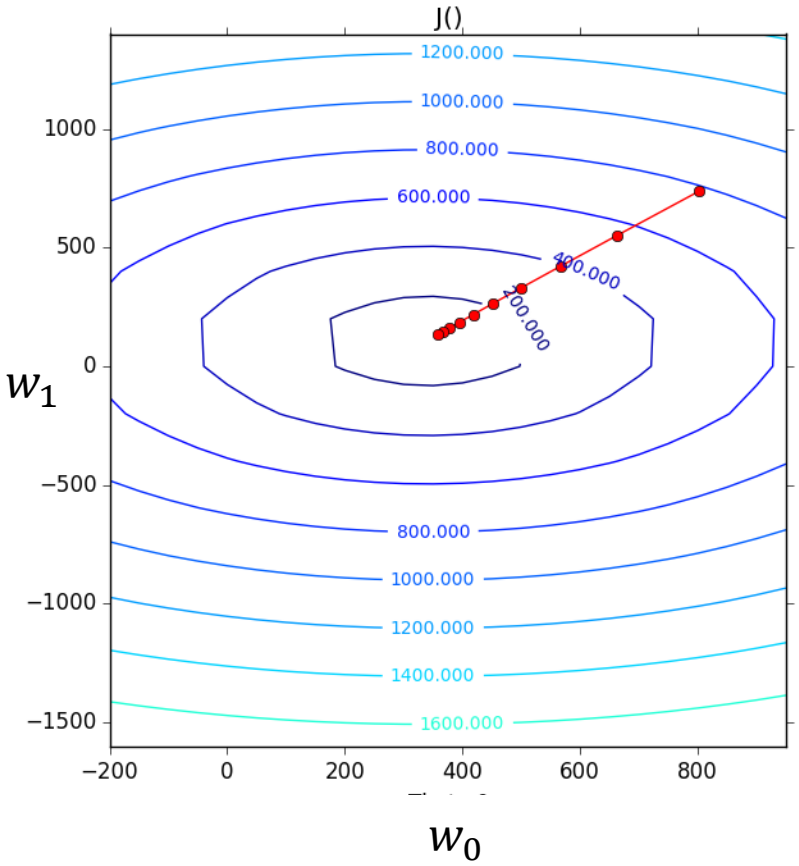
Step by Step


House Prices
House Price ($)
House Size (normalized)
800
700
600
500
400
300
200
100
0
−2
−1
0
1
2
3
4

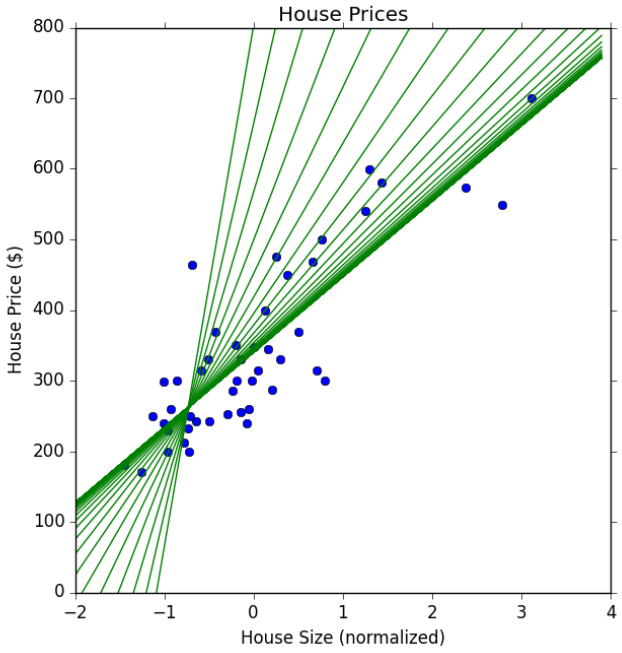

J()
1200.000
1000.000
800.000
600.000
400.000
200.000
800.000
1000.000
1200.000
1400.000
1600.000
1000
500
0
−500
−1000
−1500
−200
0
200
400
600
800
w_1
w_0

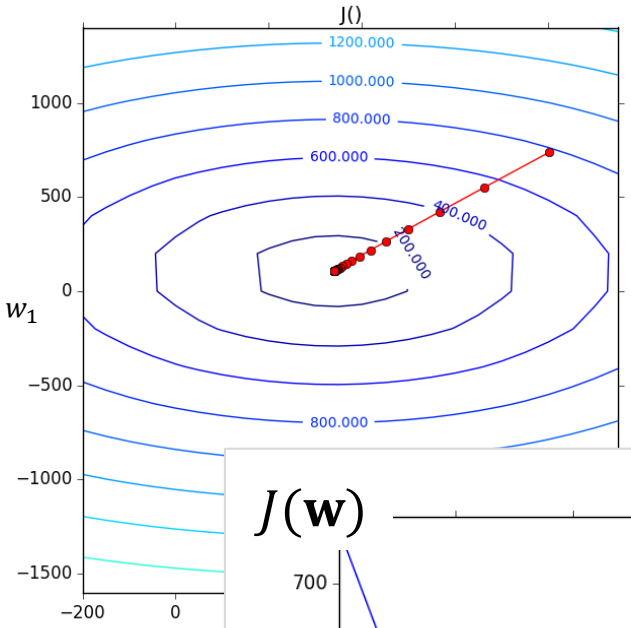
Step by Step

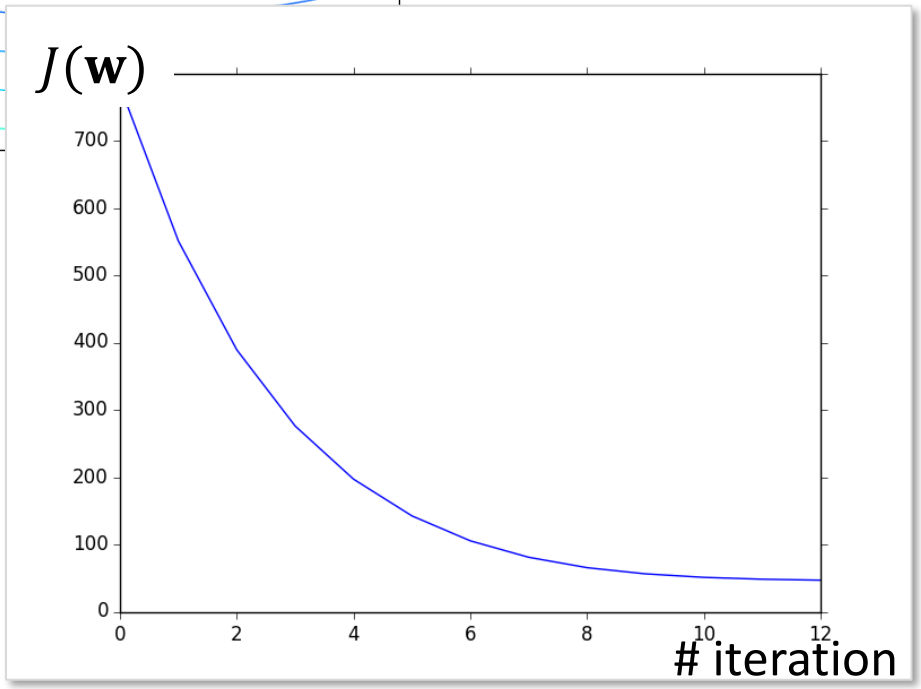

House Prices
House Price ($)
House Size (normalized)
800
700
600
500
400
300
200
100
0
−2
−1
0
1
2
3
4


J()
1200.000
1000.000
800.000
600.000
400.000
200.000
800.000
1000.000
1200.000
1400.000
1600.000
1000
500
0
−500
−1000
−1500
−200
0
200
400
600
800
w_1
w_0

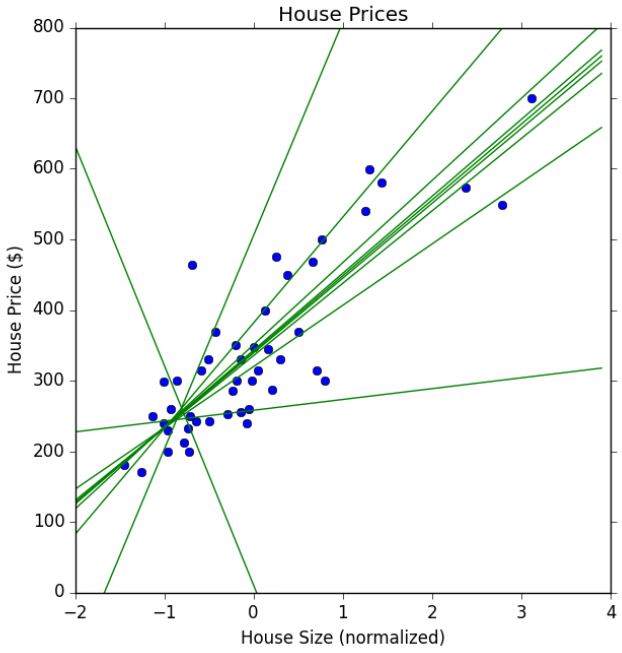
A look at the cost function

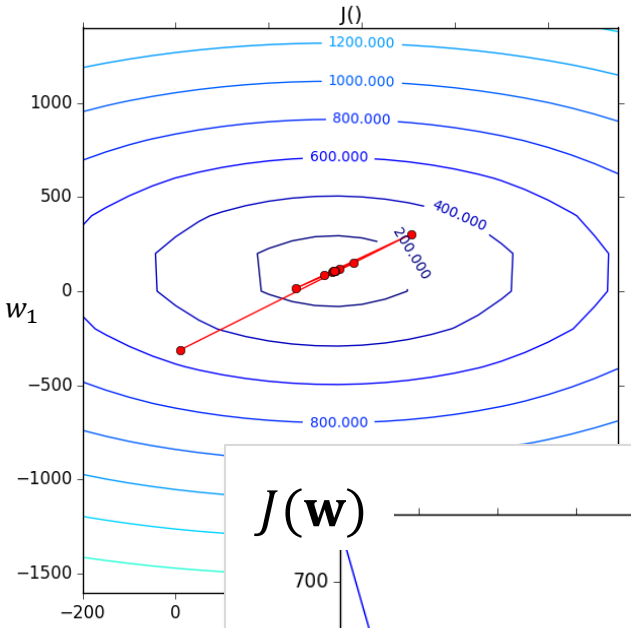


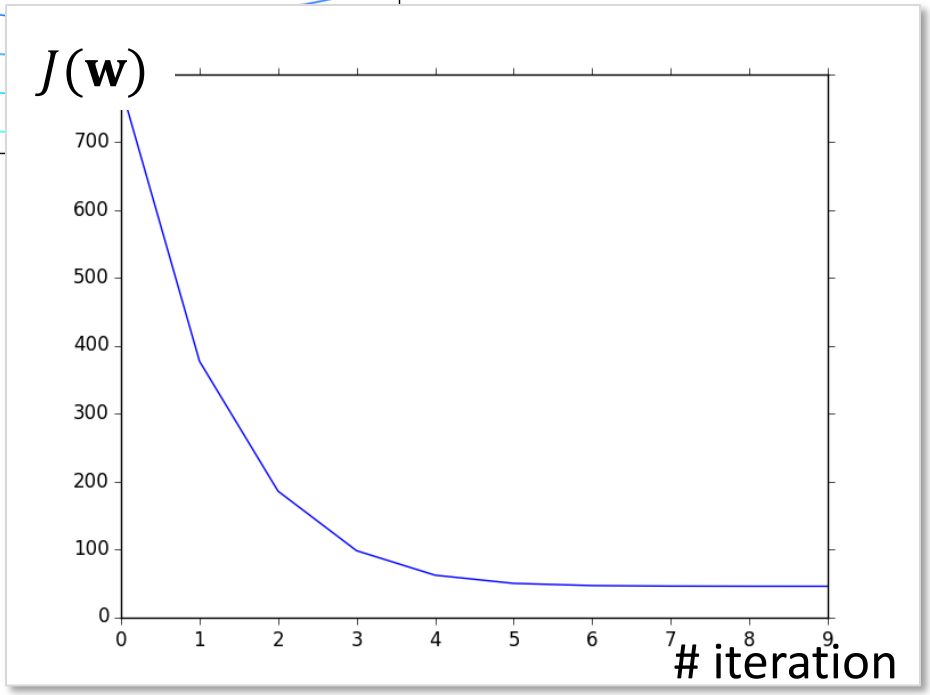




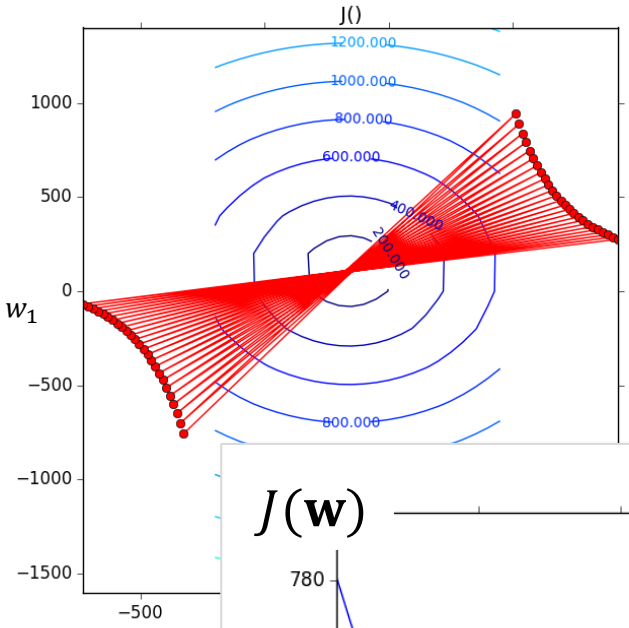
A look at the cost function

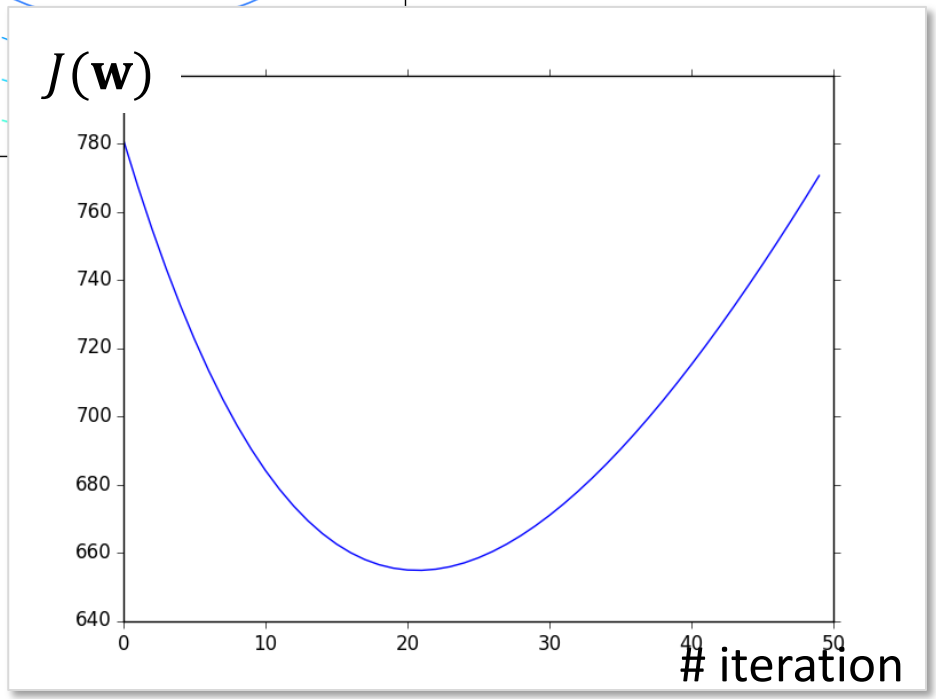






A look at the cost function





Alternative optimisation algorithms

- Batch Gradient Descent: each step of gradient descent uses all the training examples
- Stochastic Gradient Descent: each step uses one training example, examples might get shuffled in each loop to avoid cycles
- Newton's method
- Conjugate gradient method
- Normal equation

MULTIPLE REGRESSION

One Feature Scenario

Size in feet² (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

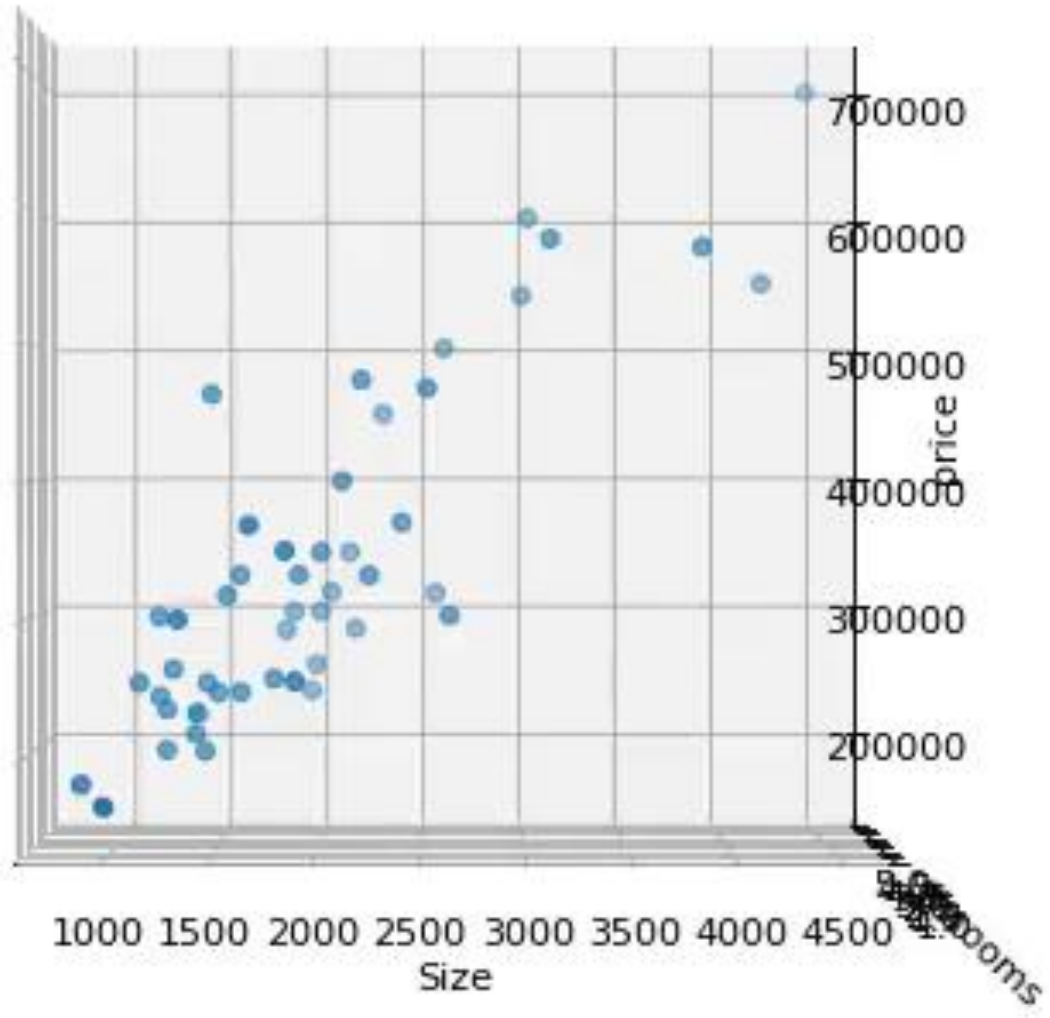
Multiple Feature Scenario

Size (feet²) (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 $\boldsymbol{x_0 = 1}$:

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

Multiple Variables Hypothesis

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

where:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \cdots \\ w_n \end{bmatrix} \quad n+1 \qquad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \quad n+1$$

Linear Regression Revisited

n=1	***n≥1***
Hypothesis:	
$f_{\boldsymbol{w}}(x) = w_0 + w_1 x$	$f_{\boldsymbol{w}}(\mathbf{x}) = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$
Parameters:	
w_0, w_1	$w_0, w_1, \dots, w_n$
Cost Function:	
$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w}}\left(x^{(i)}\right) - y^{(i)} \right)^2$	$J(w_0, w_1, \dots w_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)} \right)^2$
Goal:	
$\underset{w_0, w_1}{\text{minimise}}\left(J(w_0, w_1)\right)$	$\underset{w_0, w_1, \dots w_n}{\text{minimise}}\left(J(w_0, w_1, \dots w_n)\right)$

Linear Regression Revisited

n=1	***n≥1***
Hypothesis:	
$f_{\boldsymbol{w}}(x) = w_0 + w_1 x$	$f_{\boldsymbol{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$
Parameters:	
w_0, w_1	$\mathbf{w}$
Cost Function:	
$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w}}\left(x^{(i)}\right) - y^{(i)} \right)^2$	$J(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^{m} \left(f_{\boldsymbol{w}}\left(\mathbf{x}^{(i)}\right) - y^{(i)} \right)^2$
Goal:	
$\underset{w_0, w_1}{\text{minimise}}\left(J(w_0, w_1)\right)$	$\underset{\mathbf{w}}{\text{minimise}}\left(J(\mathbf{w})\right)$

Gradient Descent Revisited

n=1

Repeat until convergence

{

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right)$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w}}\left(x^{(i)}\right) - y^{(i)} \right) x^{(i)}$$

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

(simultaneously for j=0 and j=1)

}

n≥1

Repeat until convergence

{

$x_0^{(i)} = 1$

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

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

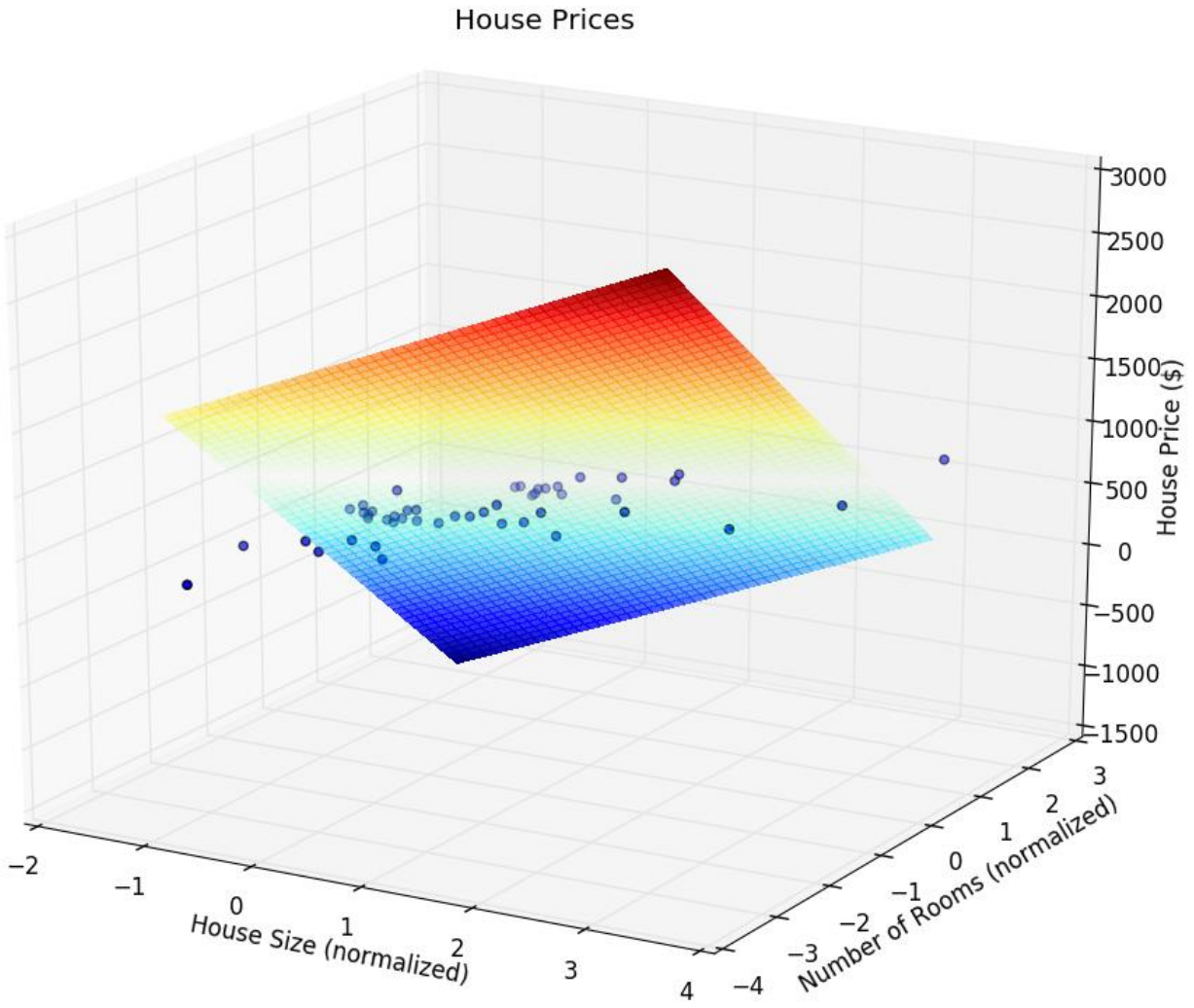
$\vdots$

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

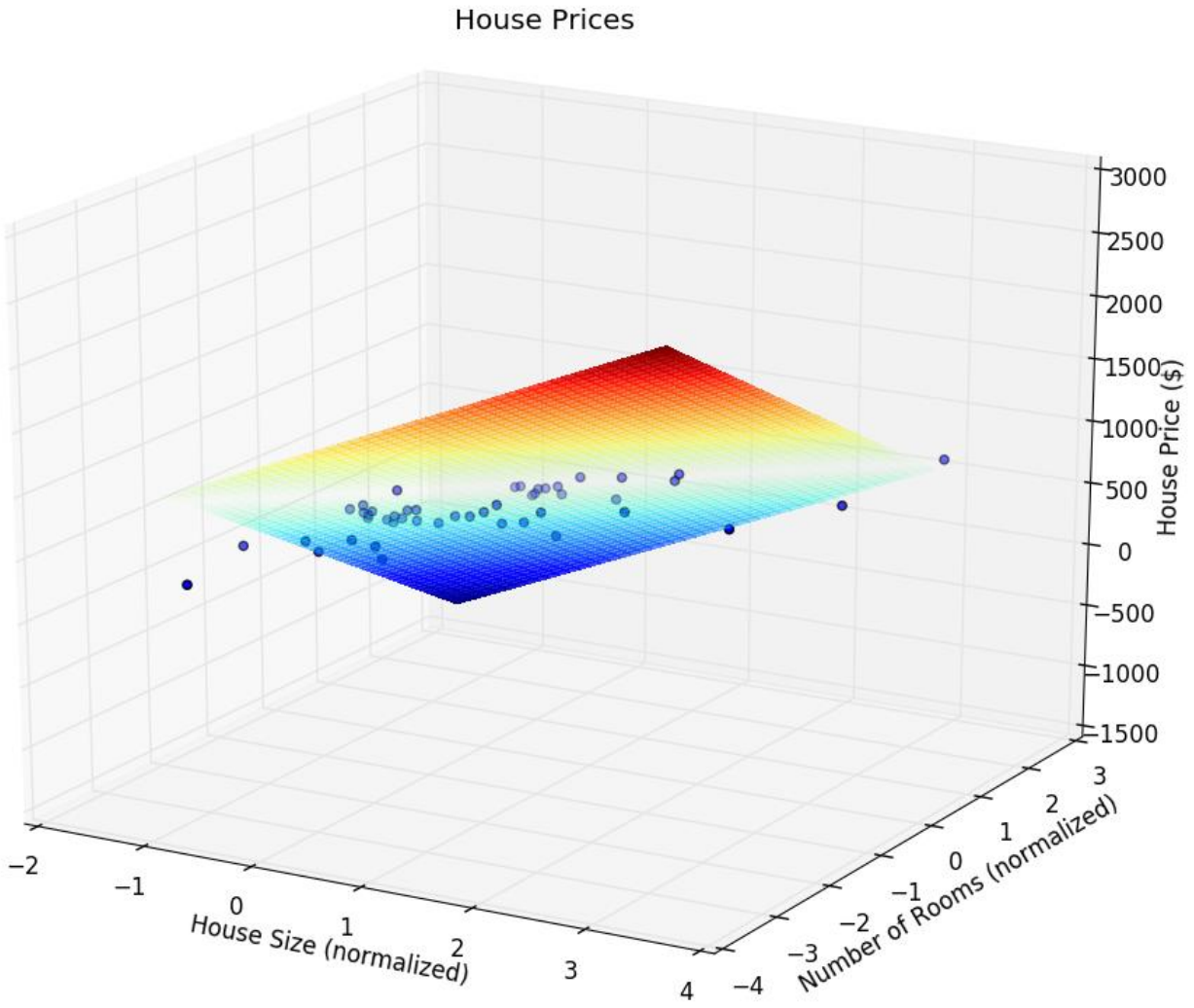
(simultaneously for all j)

}

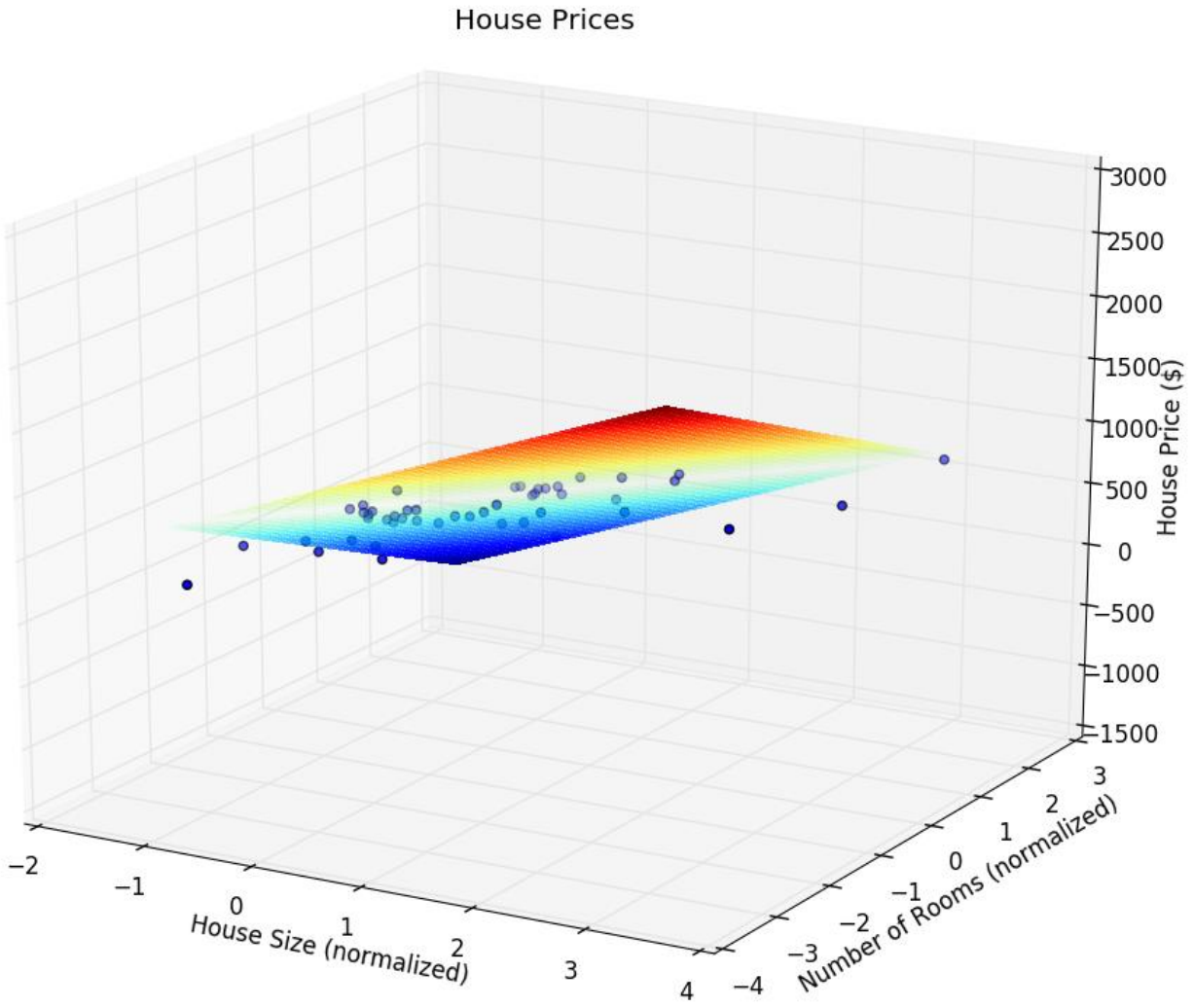
Multiple Regression Example



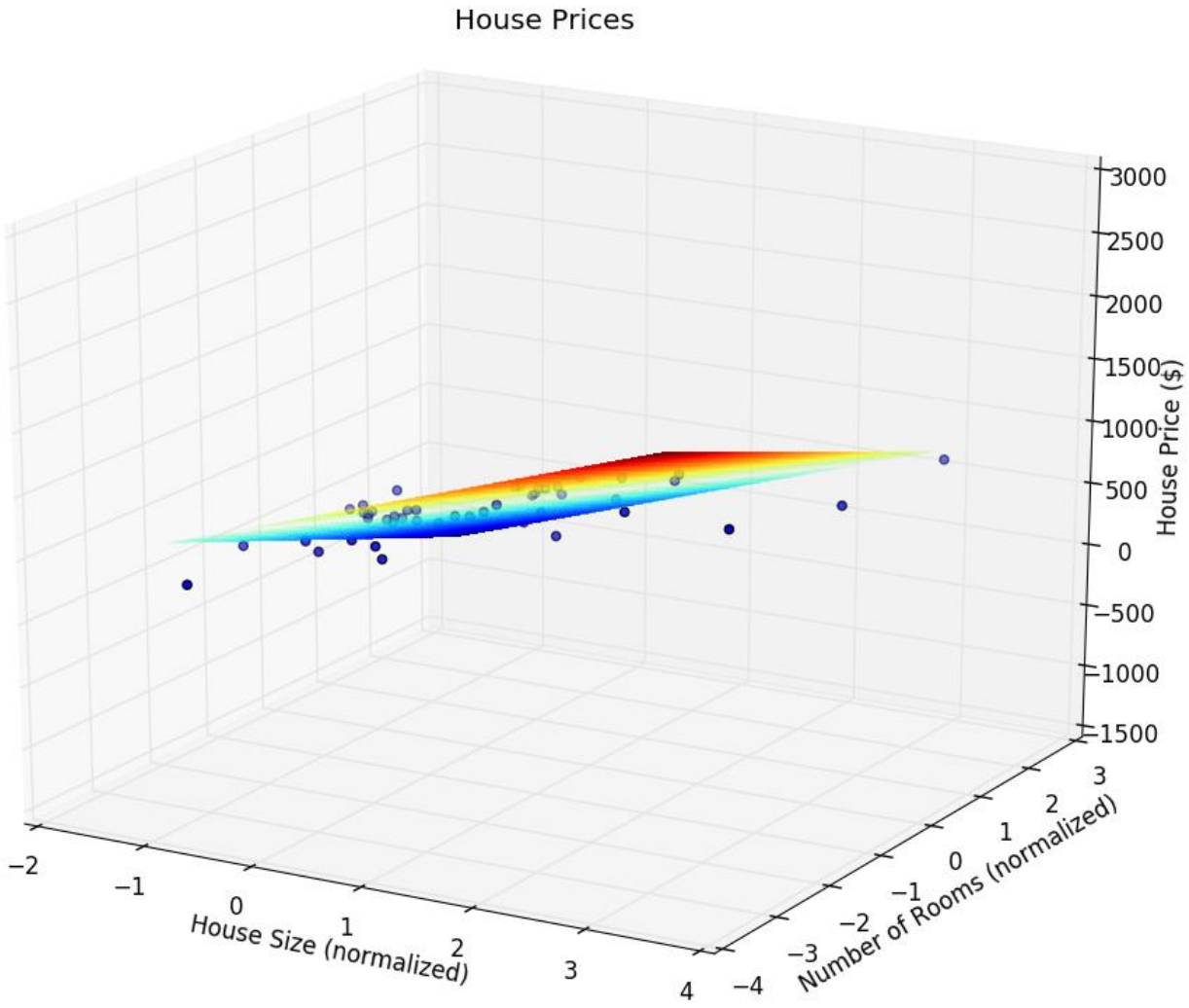
Multiple Regression Example



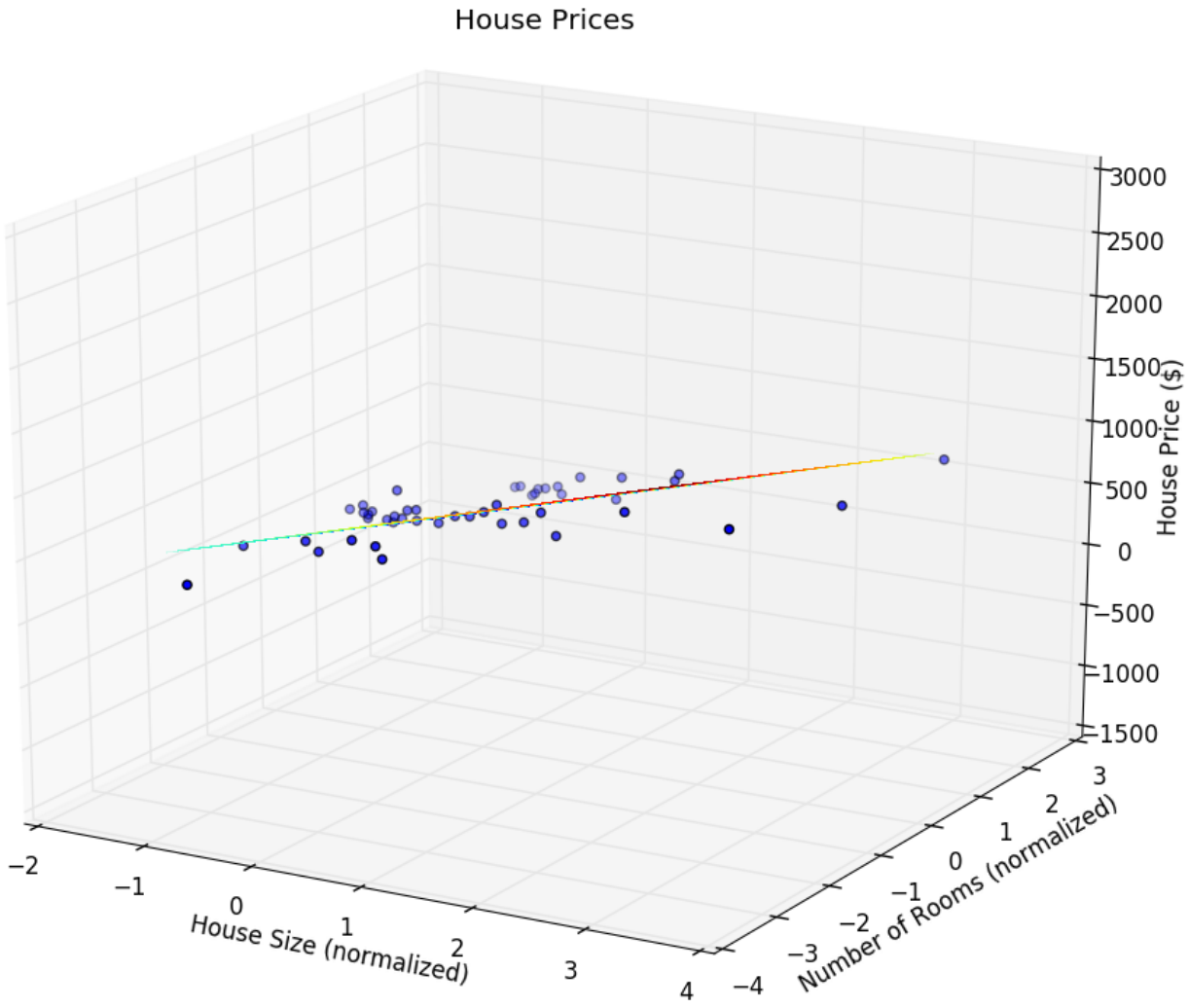
Multiple Regression Example



Multiple Regression Example



Multiple Regression Example



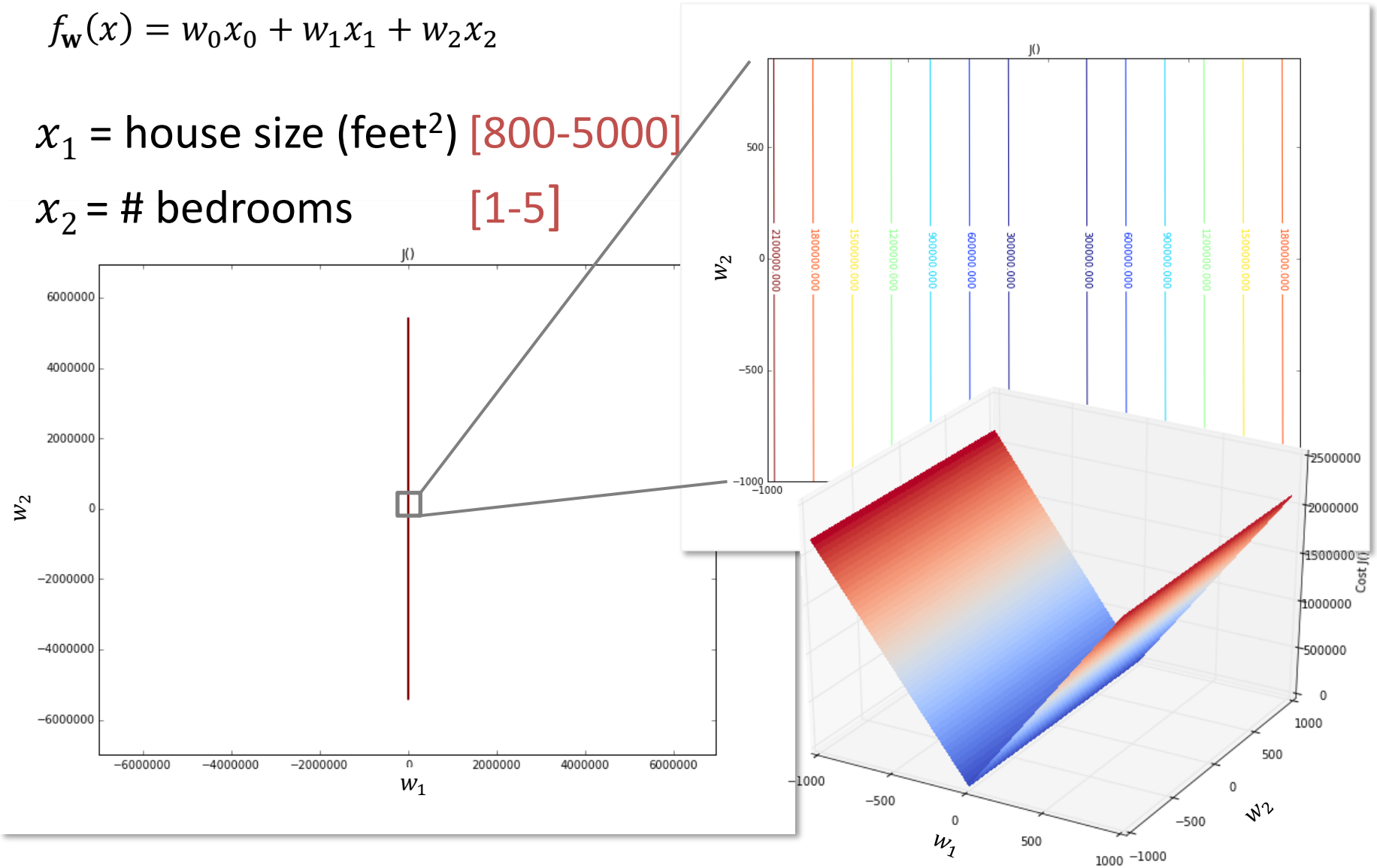
FEATURE SCALING

Feature ranges – the problem

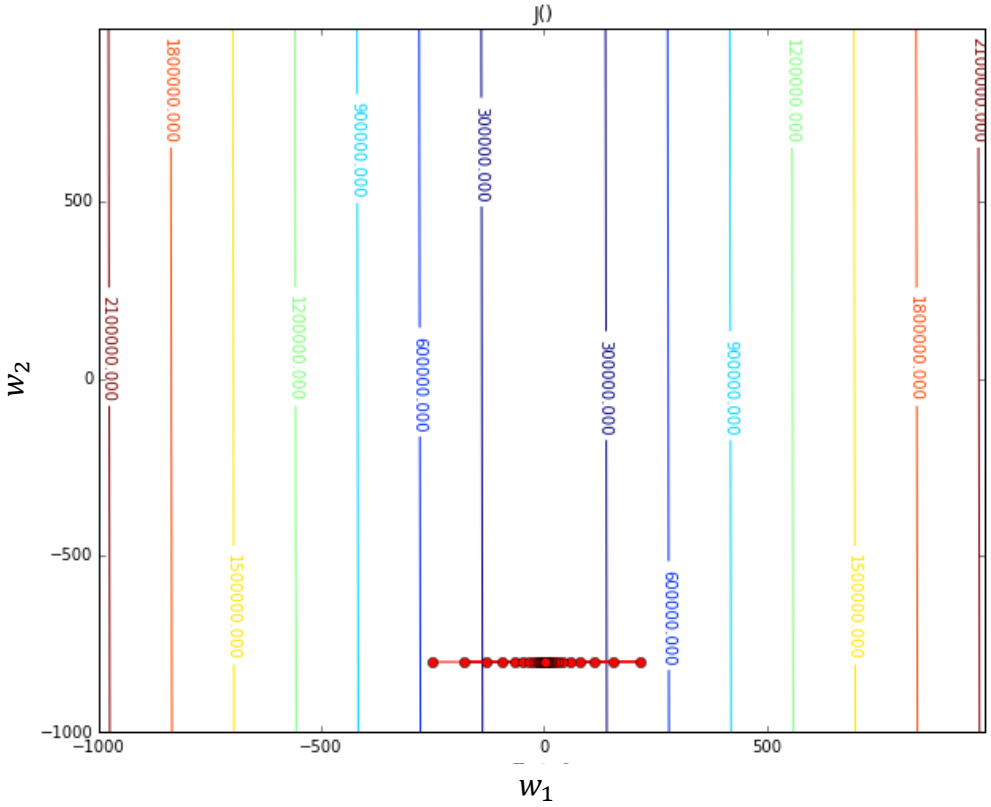
$f_{\mathbf{w}}(x) = w_0 x_0 + w_1 x_1 + w_2 x_2$

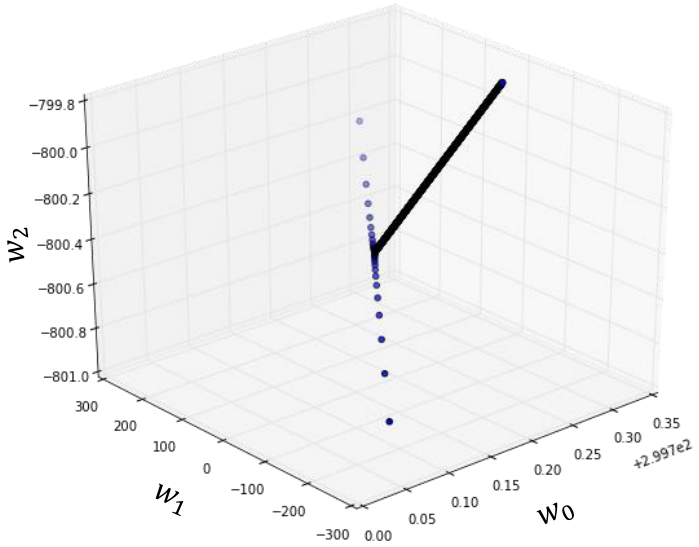
x_1 = house size (feet^2) [800-5000]

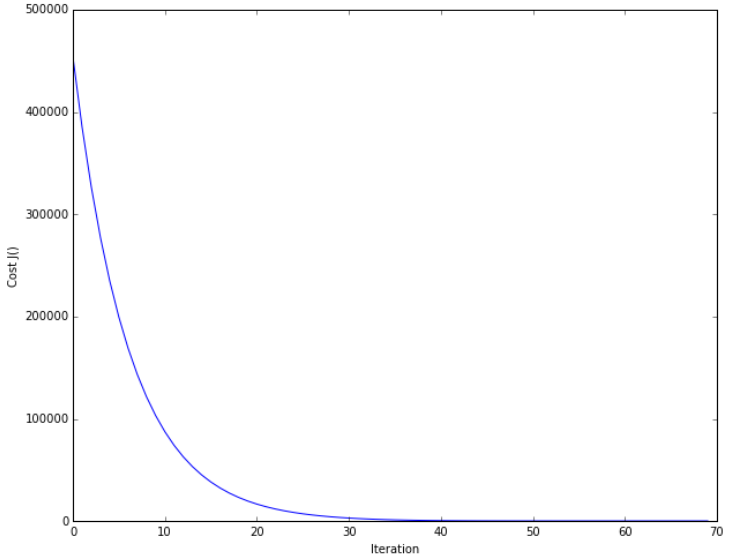
x_2 = # bedrooms [1-5]



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 !!!

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

$$\mu_j = \overline{x_j} = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

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

Feature scaling – standardisation

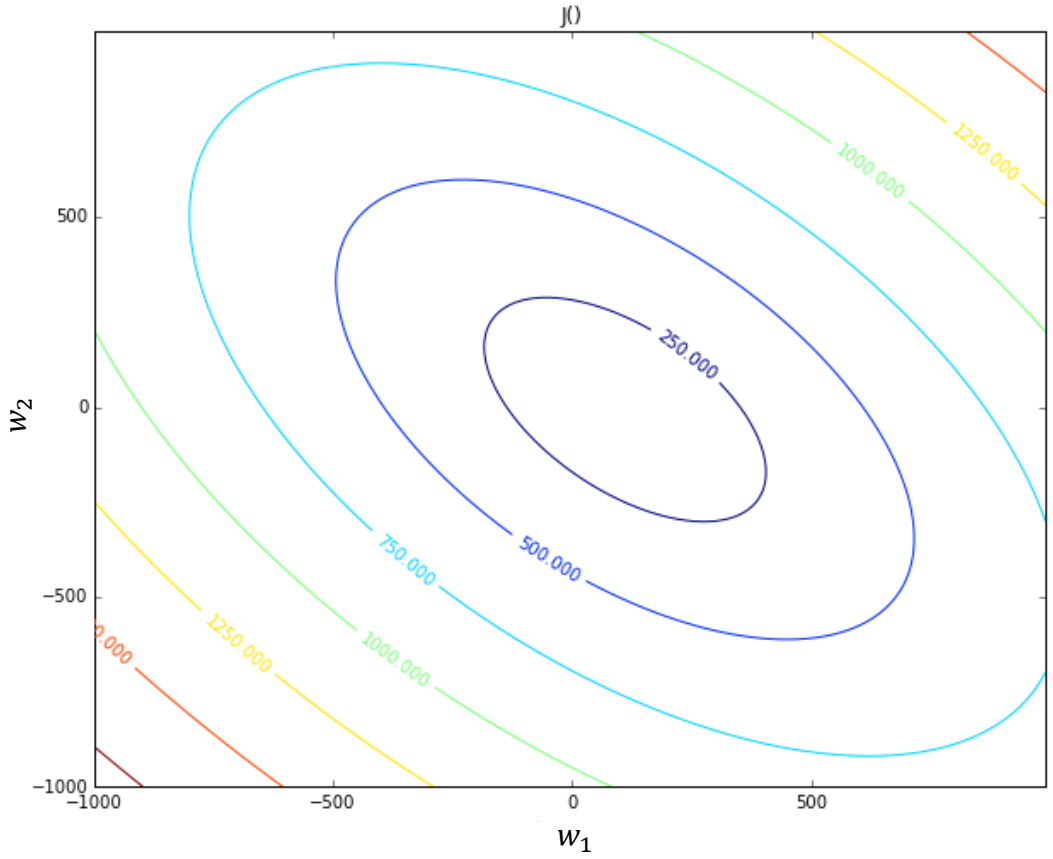
house size (feet^2) [800, 5000] → [-1.44, 3.12]

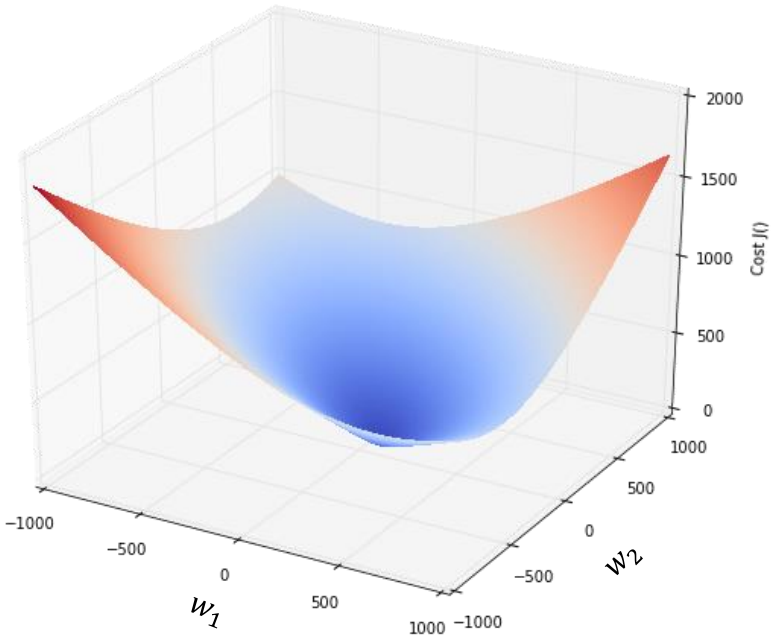
$$x_1^{(i)} = \frac{x_1^{(i)} - \mu_1}{s_1} = \frac{x_1^{(i)} - 2000.68}{794.7}$$

# of bedrooms [1-5] → [-2.85, 2.40]

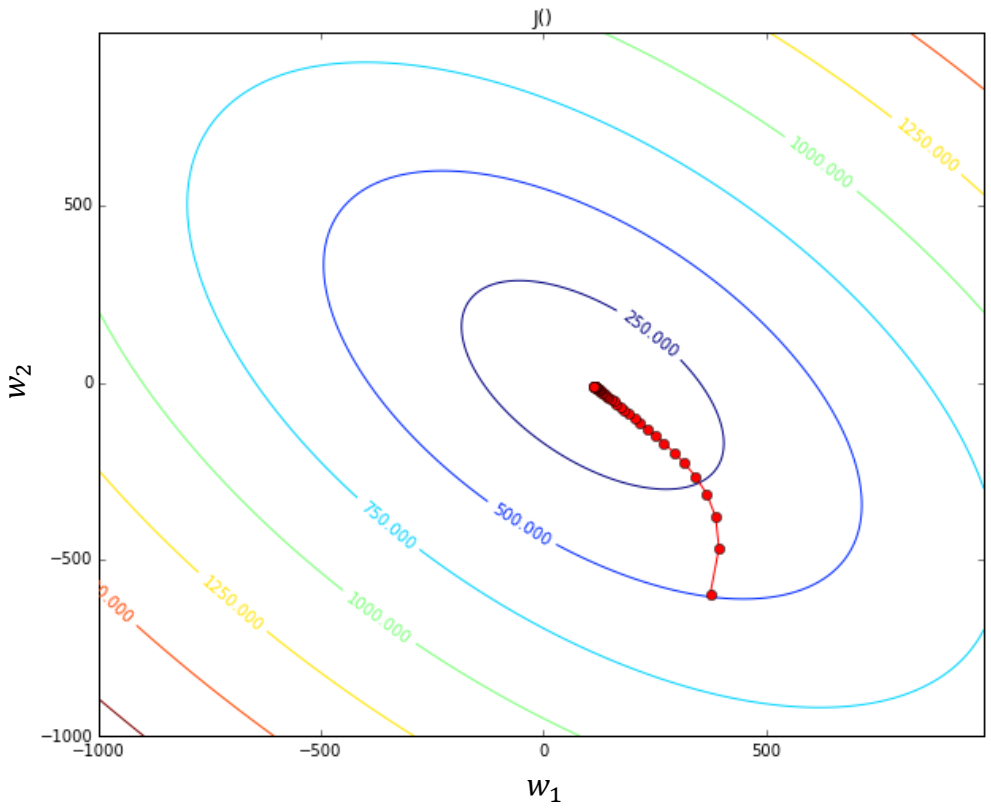
$$x_2^{(i)} = \frac{x_2^{(i)} - \mu_2}{s_2} = \frac{x_2^{(i)} - 3.17}{0.76}$$

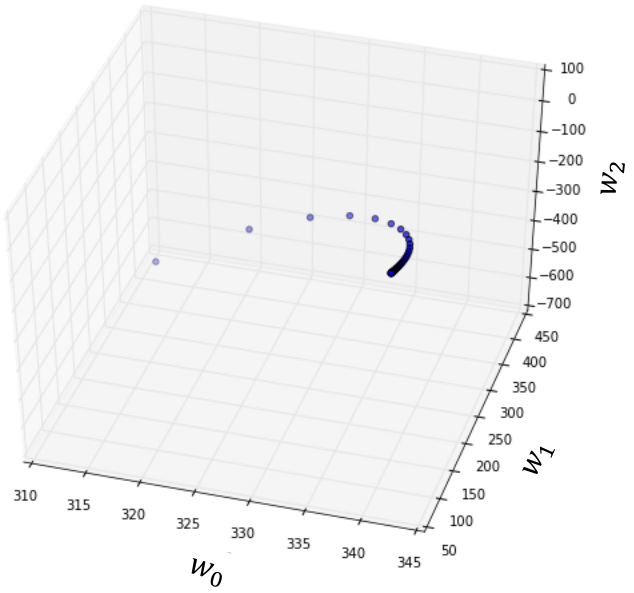
Feature scaling – standardisation

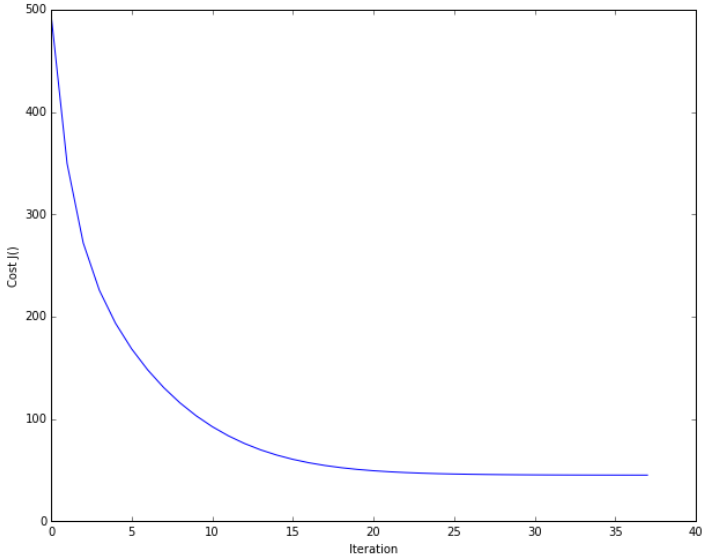




Feature scaling – standardisation

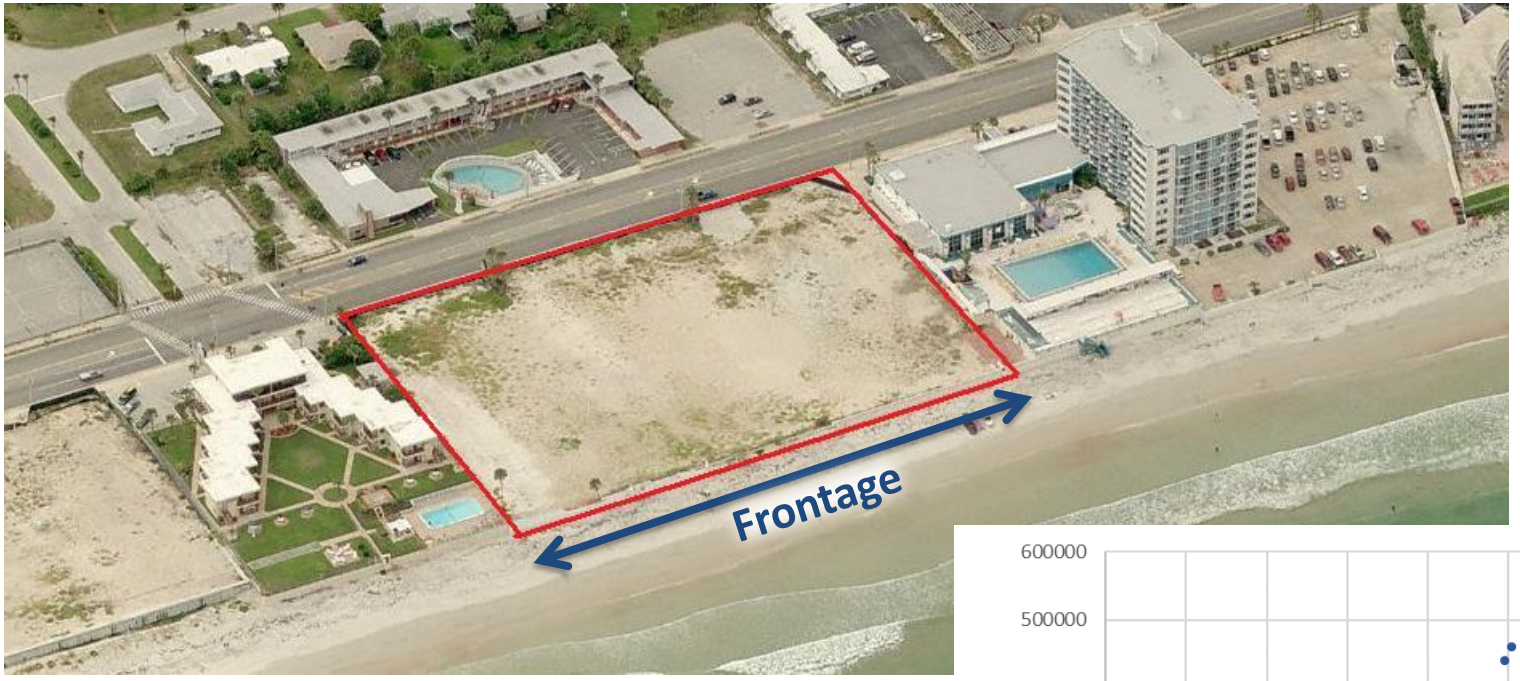


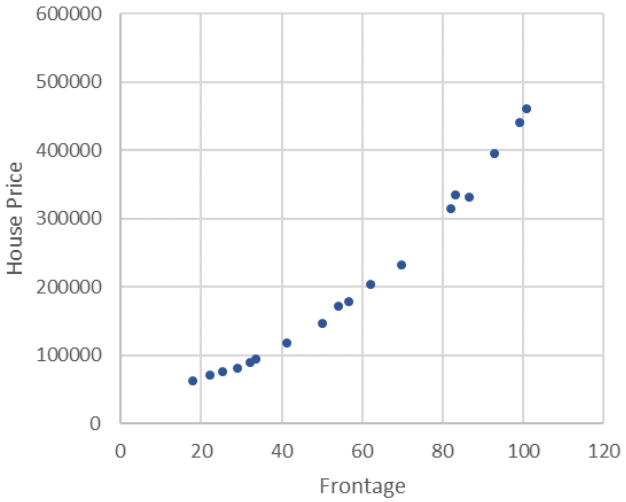




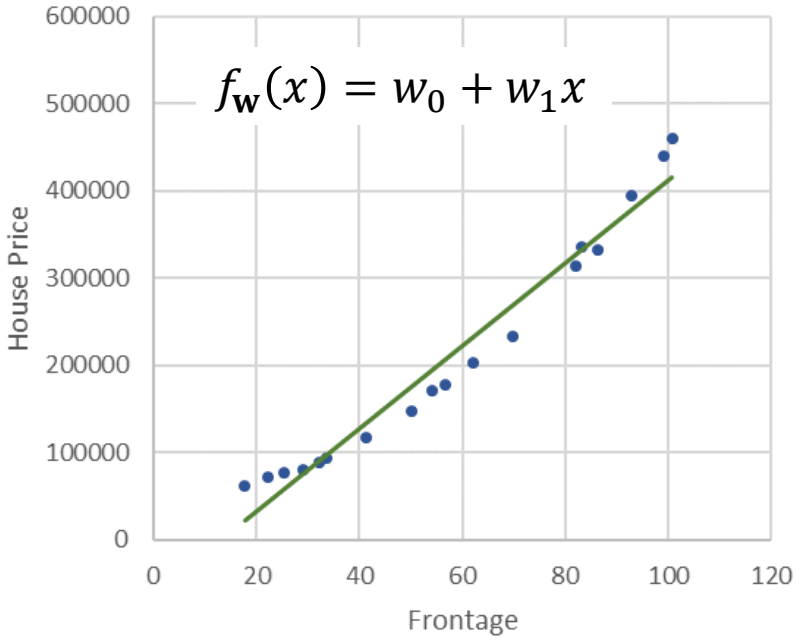
POLYNOMIAL REGRESSION

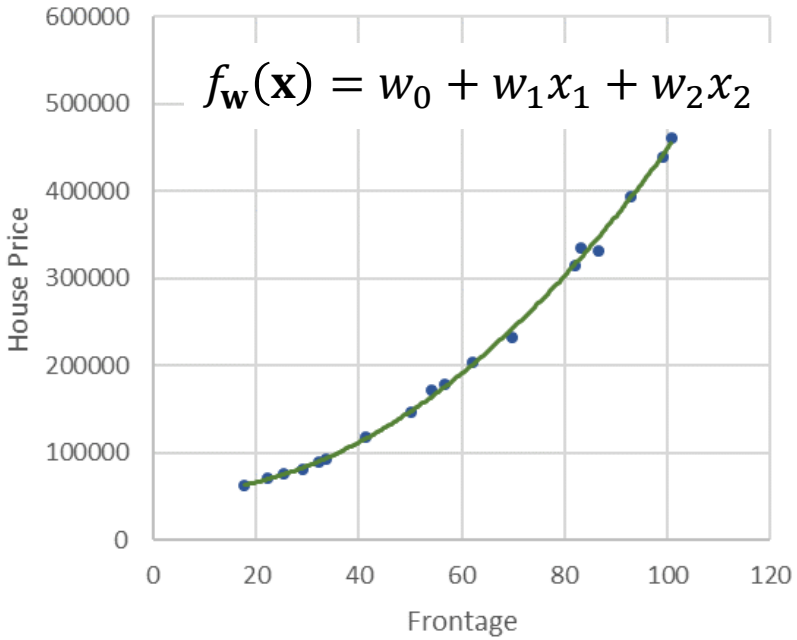
Selling parcels





Polynomial models for house prices



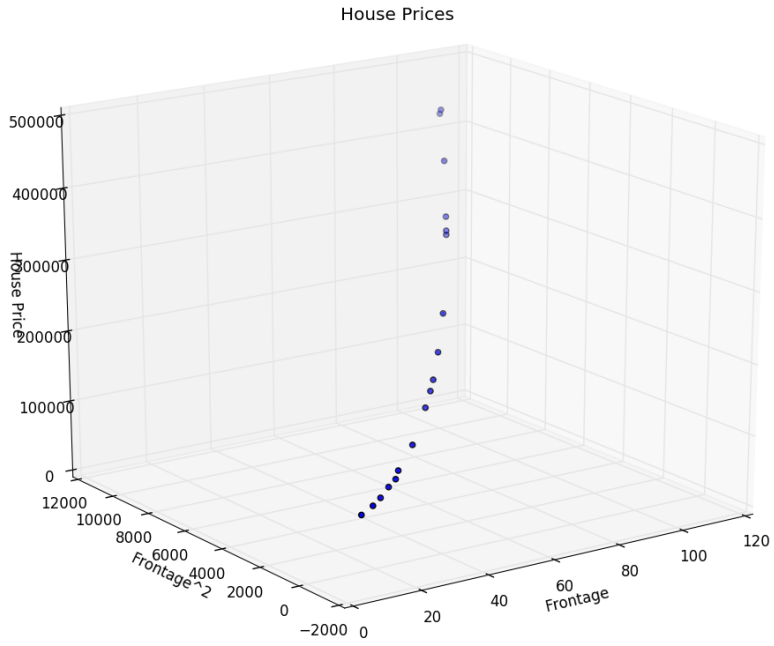


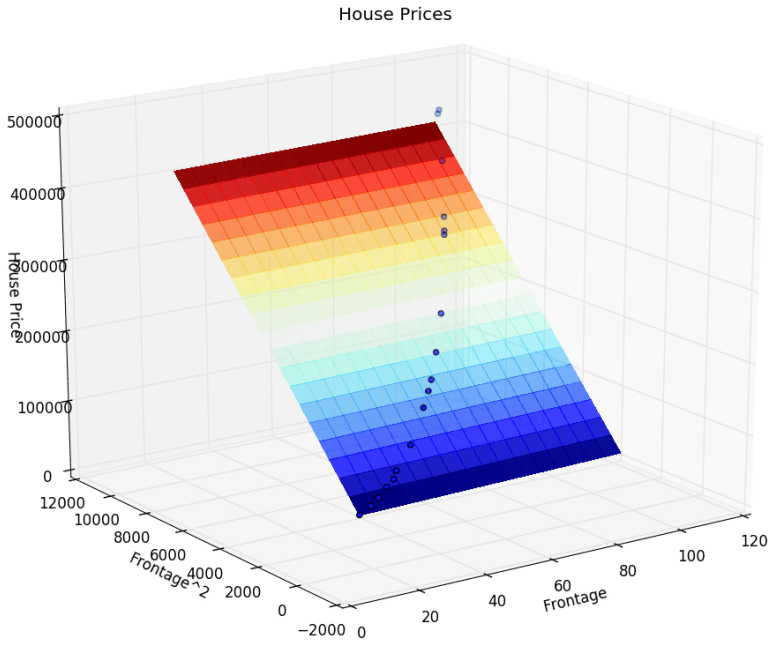
$$f_{\mathbf{w}}(frontage) = w_0 + w_1(frontage) + w_2(frontage^2)$$

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

$$x_1 = frontage$$
$$x_2 = frontage^2$$

Polynomial models for house prices



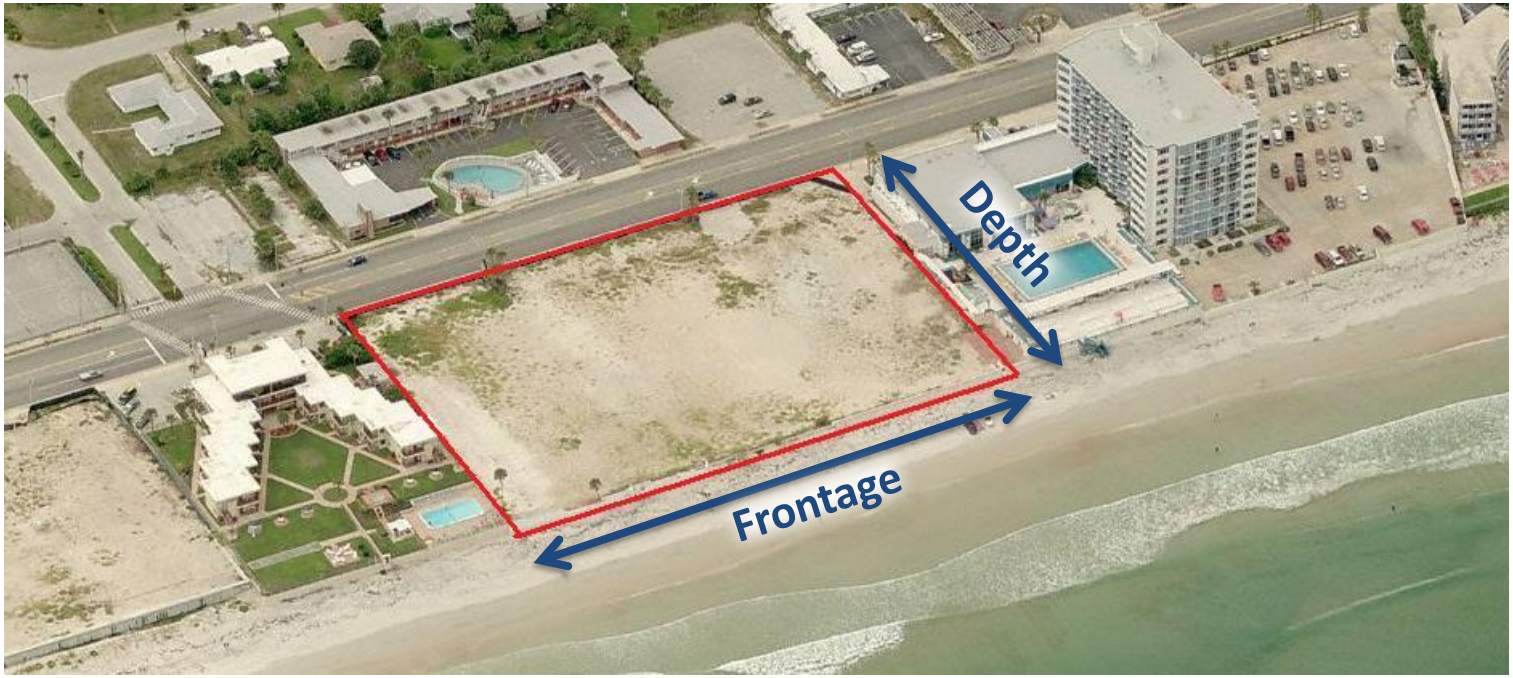


$$f_{\mathbf{w}}(frontage) = w_0 + w_1(frontage) + w_2(frontage^2)$$

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

$$x_1 = frontage$$
$$x_2 = frontage^2$$

Creating new features



x_1 = *Frontage*

x_2 = *Depth*

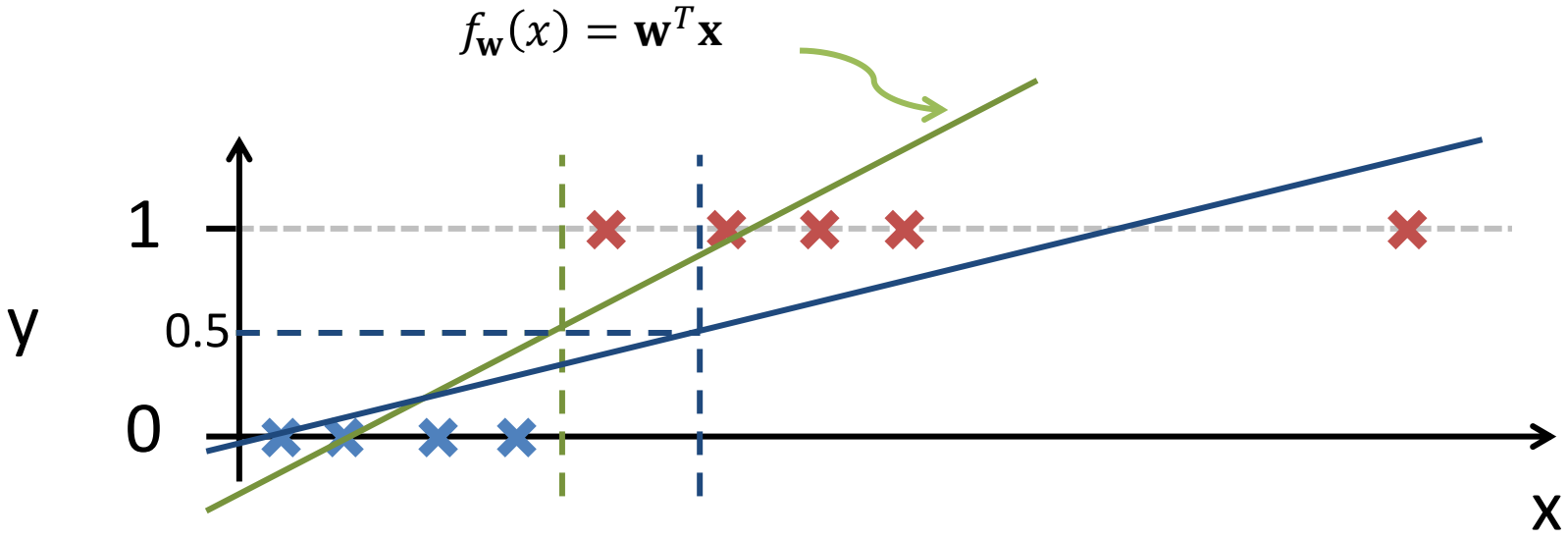
x_3 = *Area* = *Frontage* x *Depth* = $x_1 x_2$

LOGISTIC REGRESSION

And how to use it to do classification

Classification

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

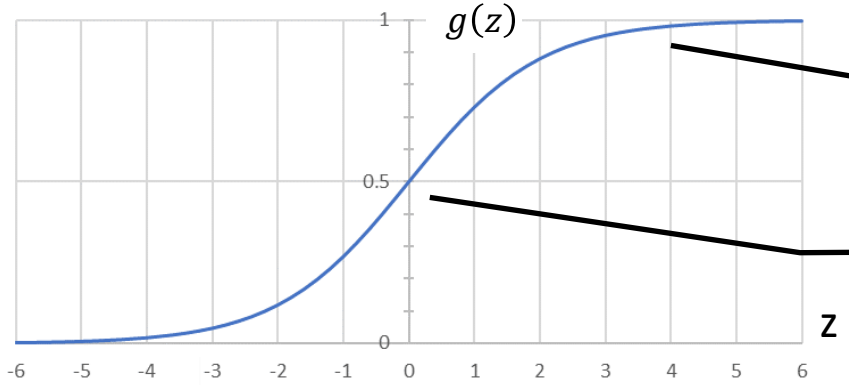


Idea: do linear regression and then **threshold** the prediction at 0.5 so that:

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

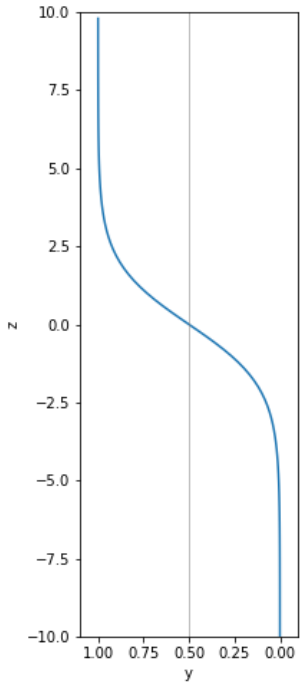
Logistic (Sigmoid) Function

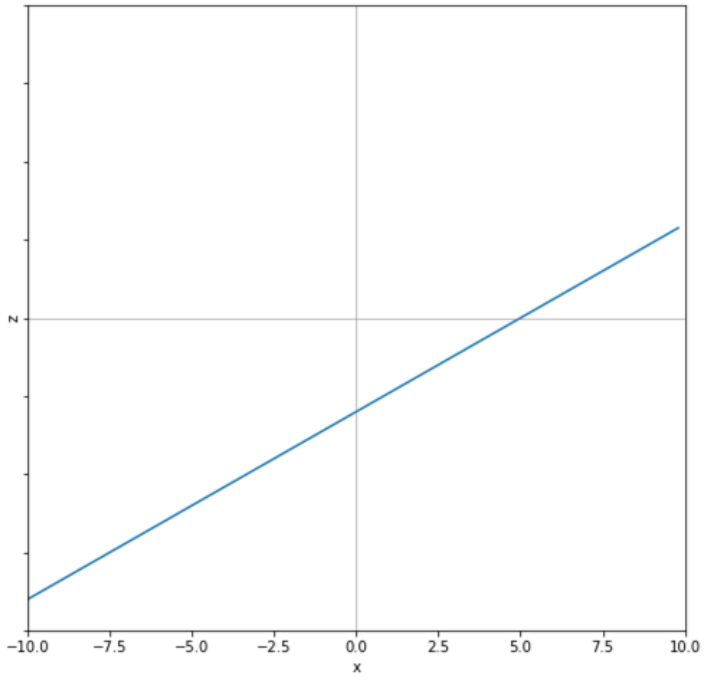
$$g(z) = \frac{1}{1 + e^{-z}}$$



Bounded in [0, 1]

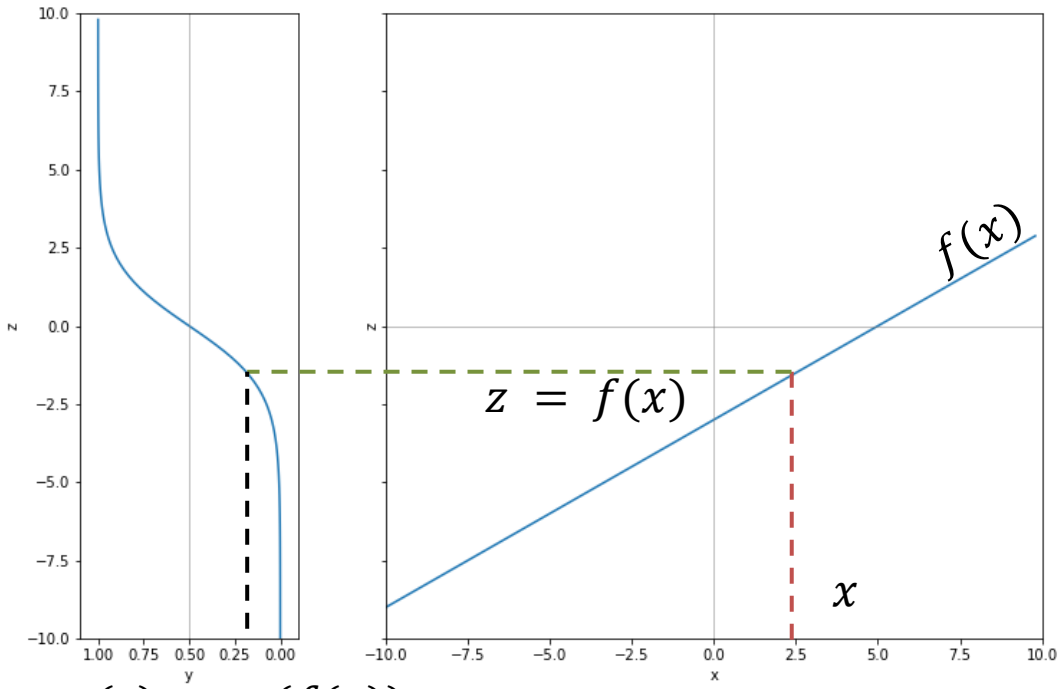
Crosses 0.5 at z=0



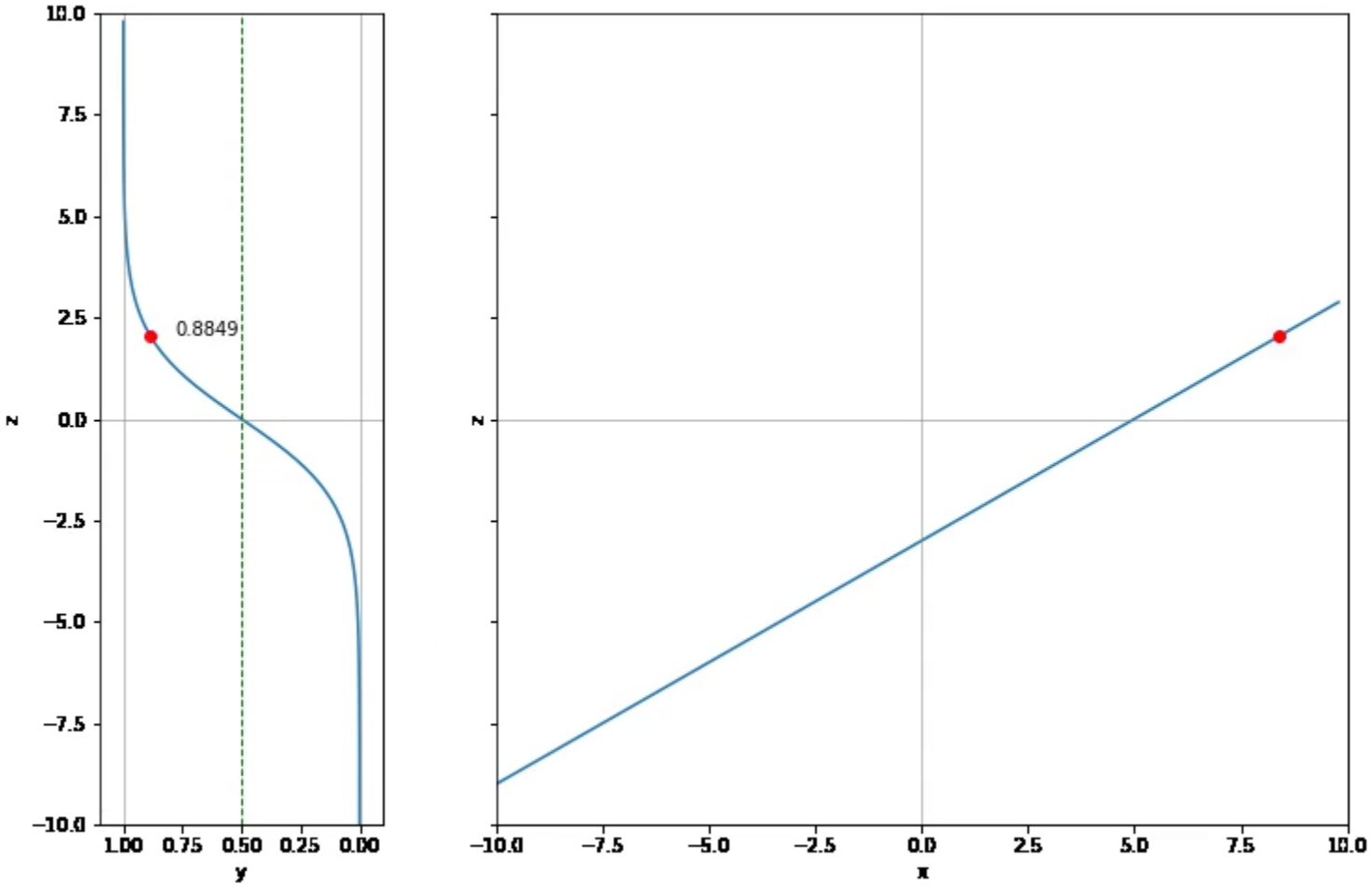


Playing with the Logistic

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

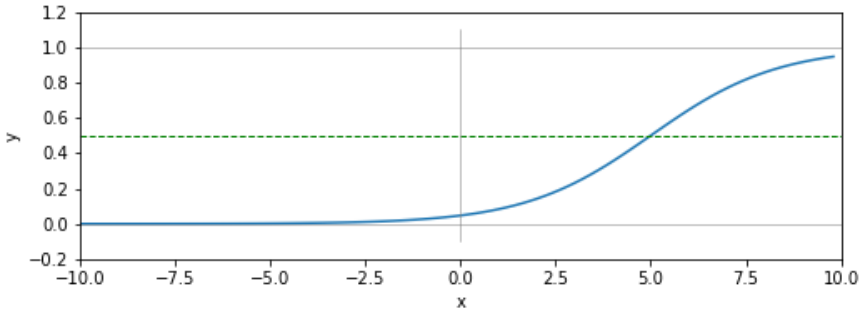


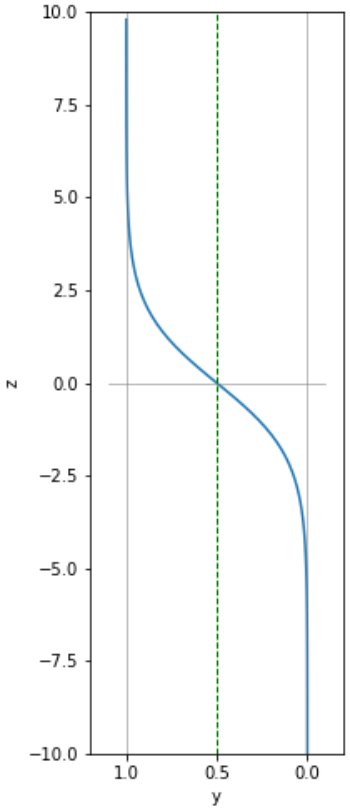
$$y = g(z) = g(f(x))$$

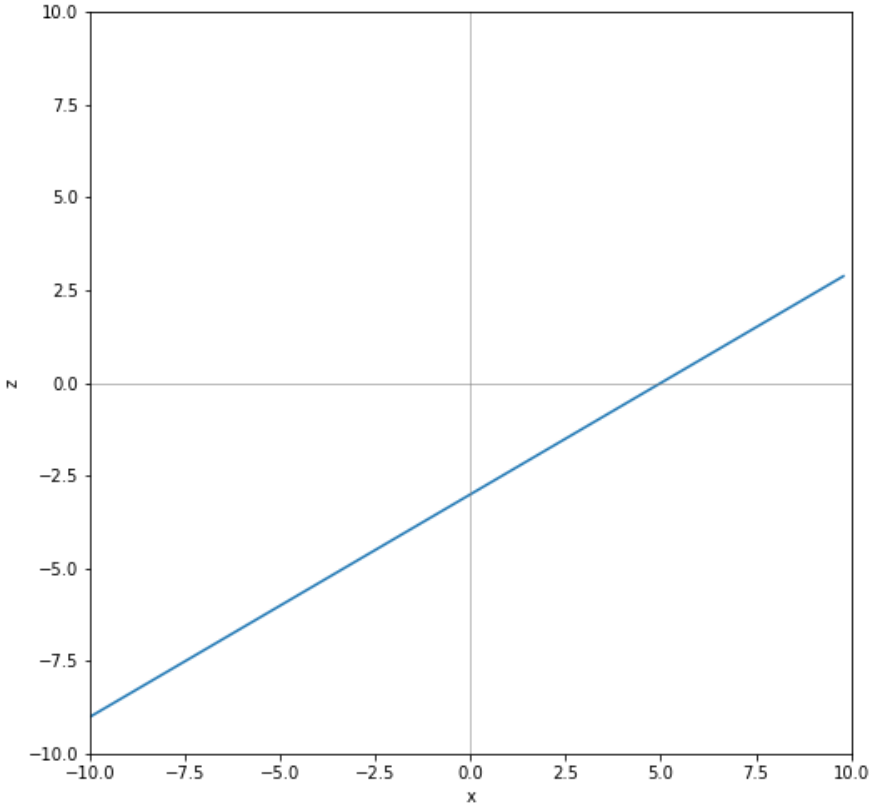

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
0.8849
1.00
0.75
0.50
0.25
0.00
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

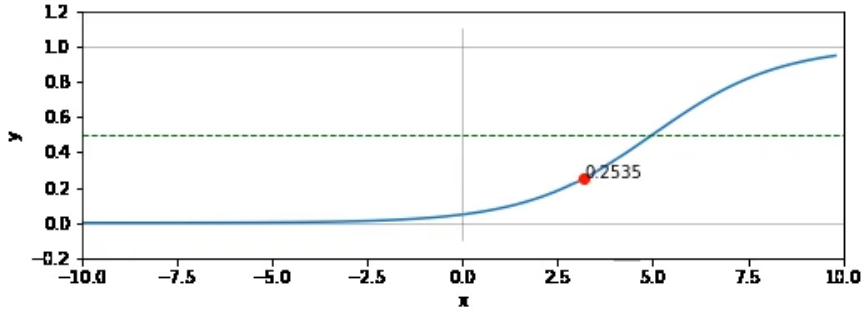
x

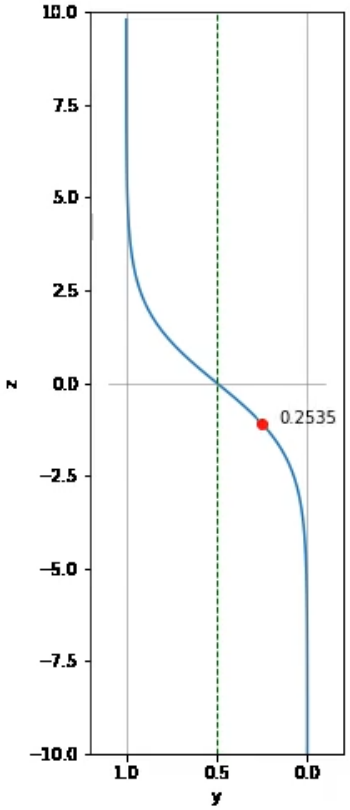
$y = g(z) = g(f(x))$

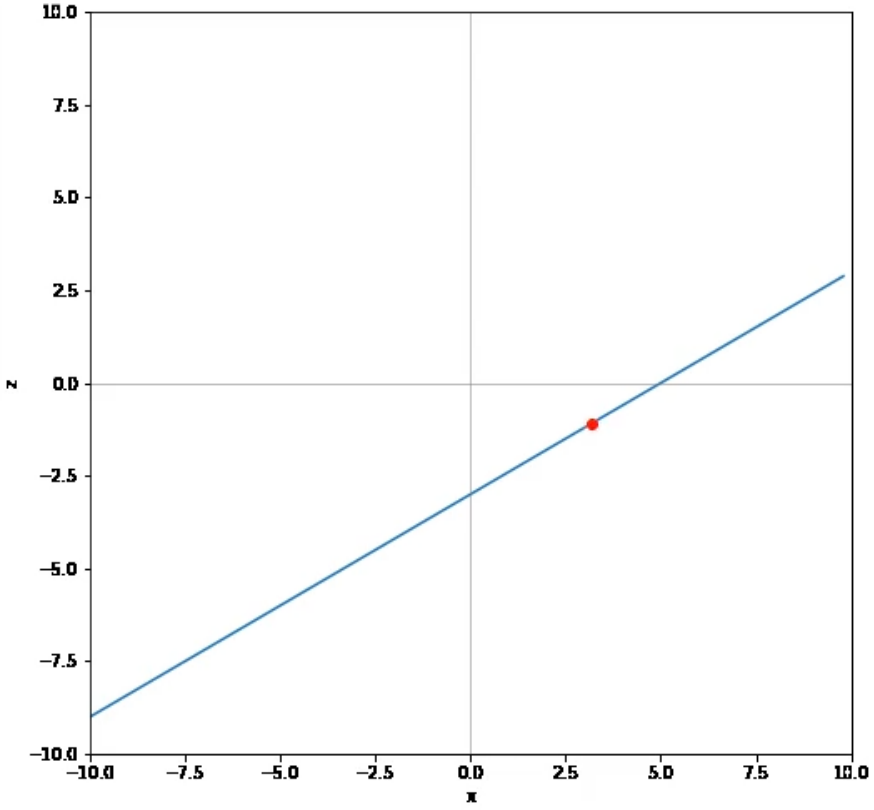

1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

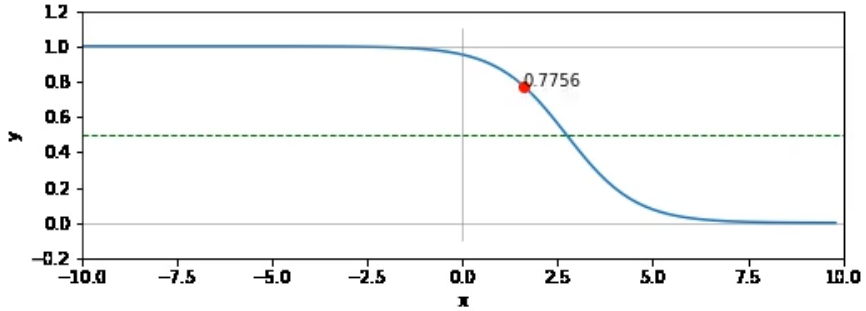

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
1.0
0.5
0.0
y

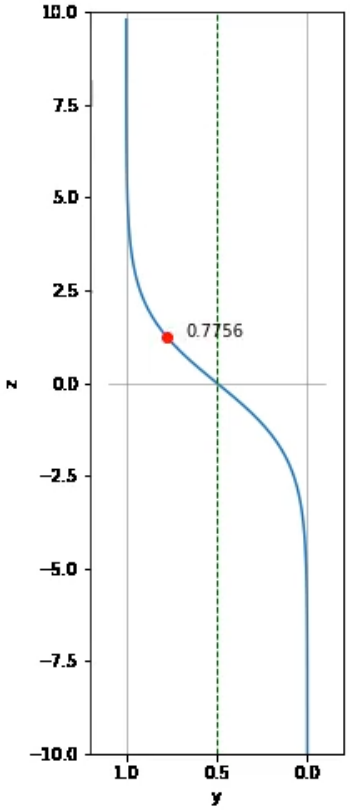

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

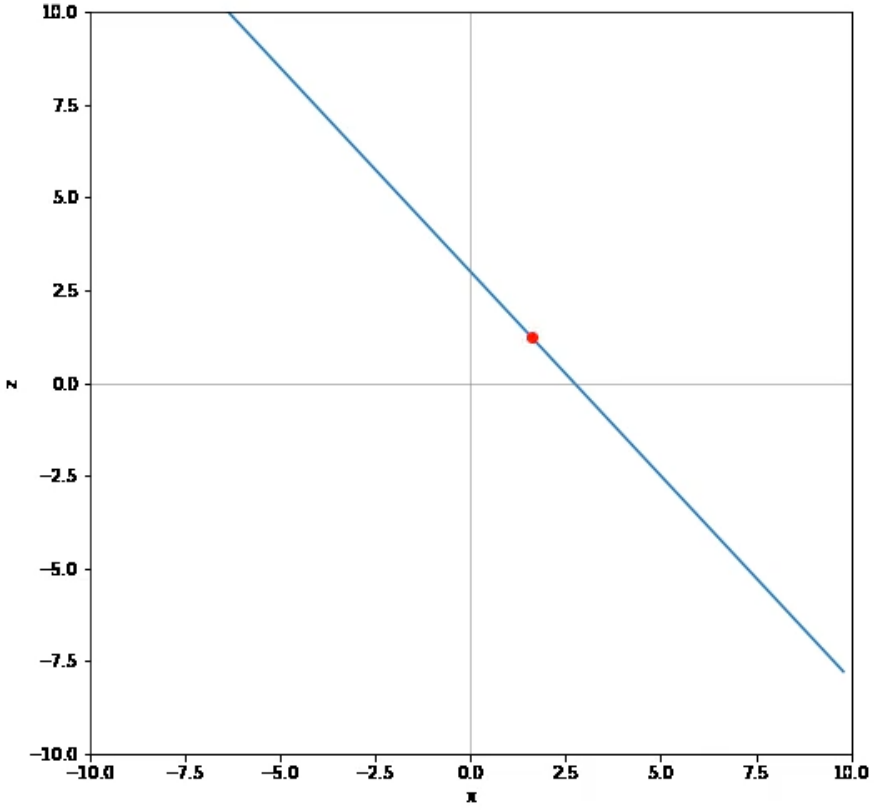

1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
y
0.2535
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

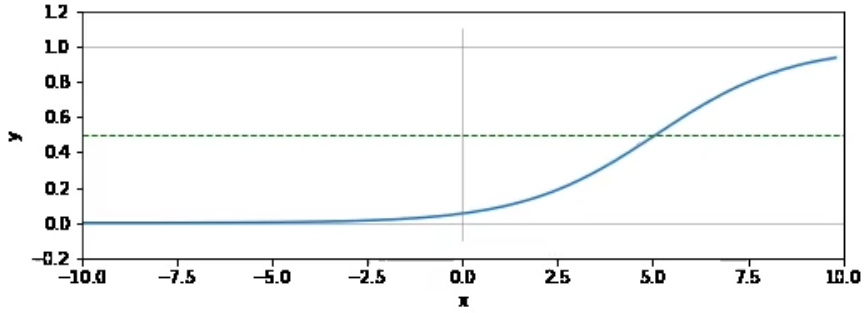

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
0.2535
1.0
0.5
0.0
y

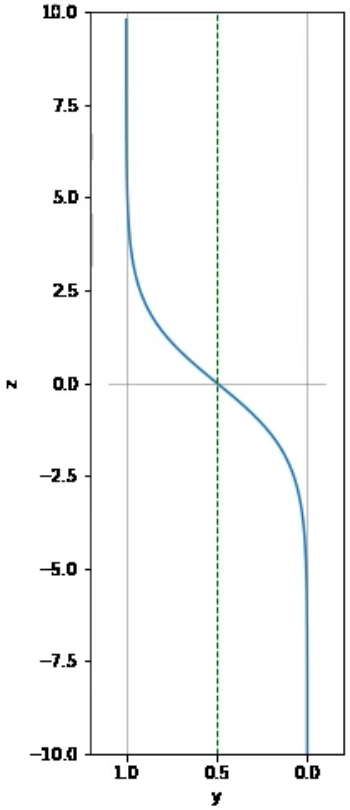

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

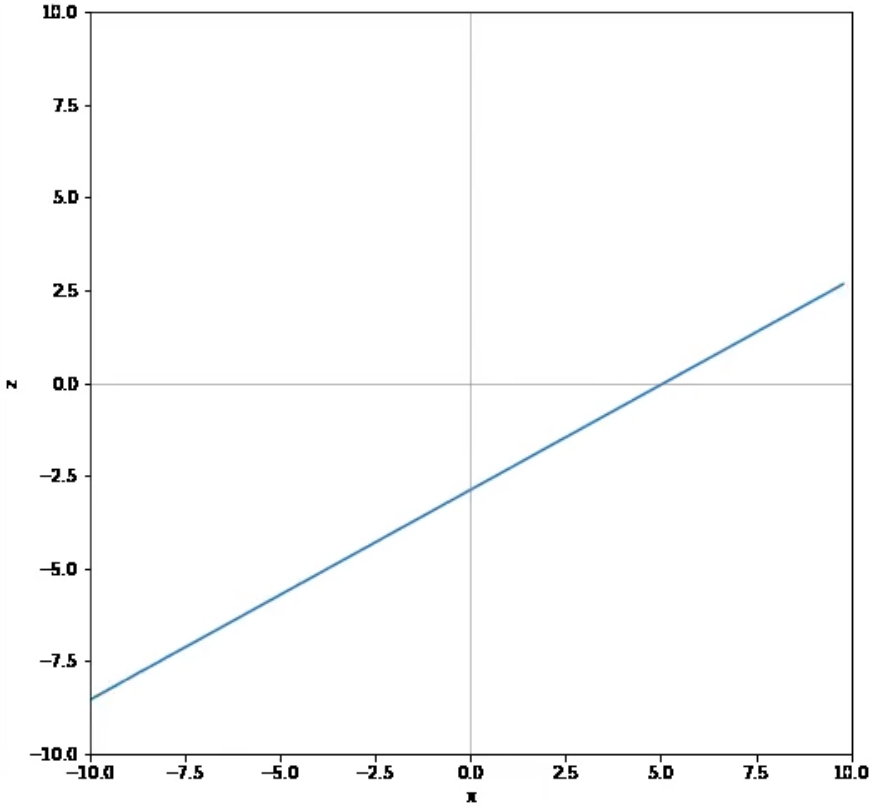

0.7756
y
x


0.7756
z
y


z
x

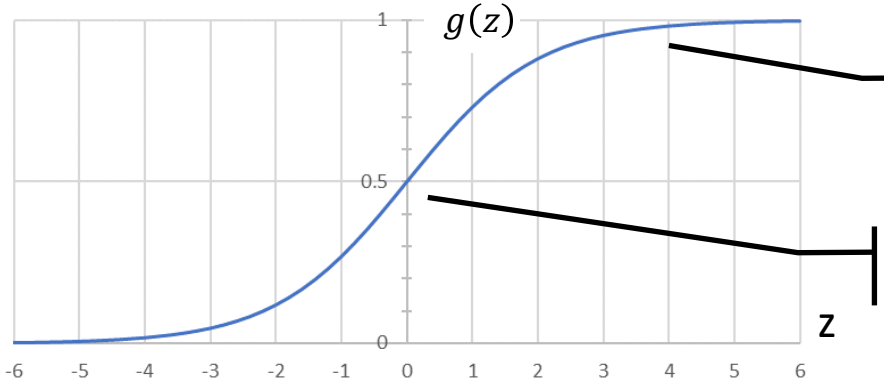

1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
1.0
0.5
0.0
y


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

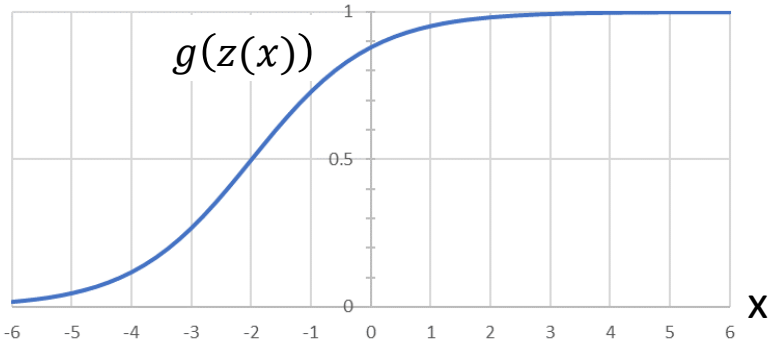
Logistic (Sigmoid) Function

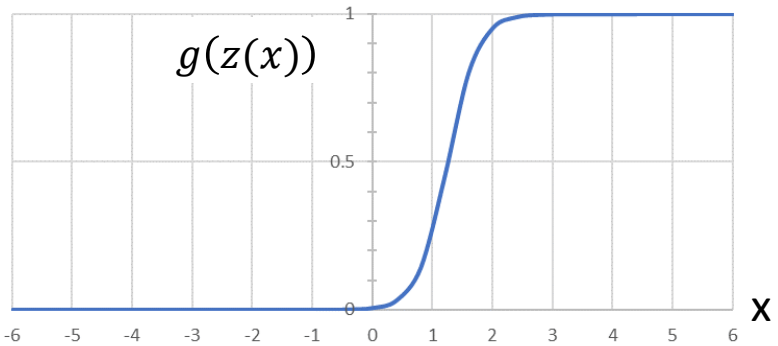
$$g(z) = \frac{1}{1 + e^{-z}}$$

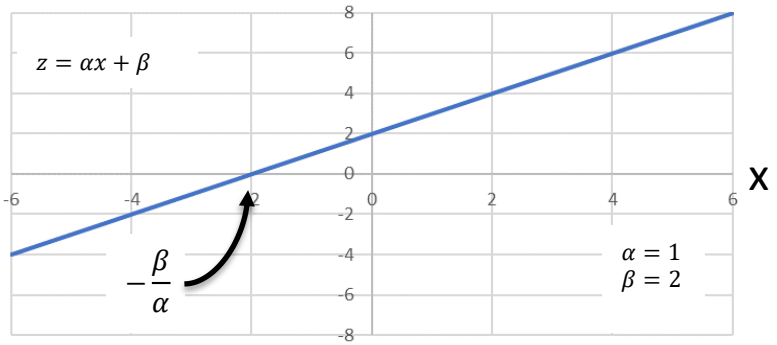


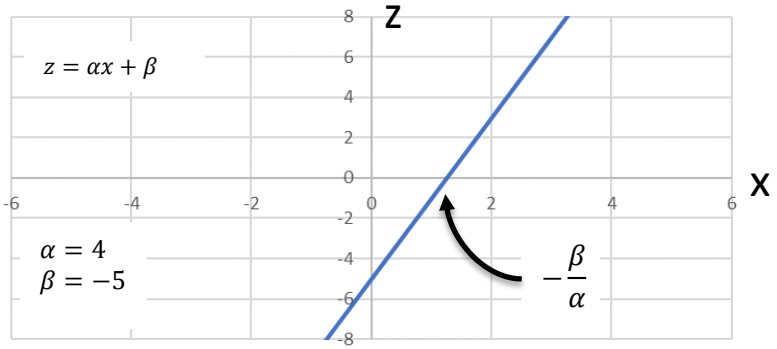
Bounded in [0, 1]

Crosses 0.5 at z=0



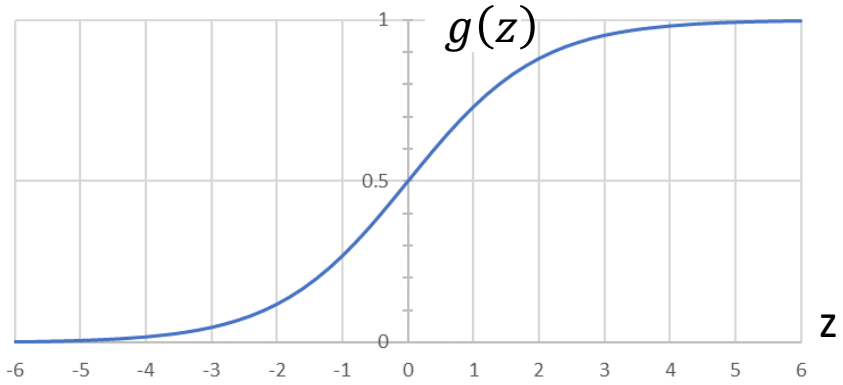




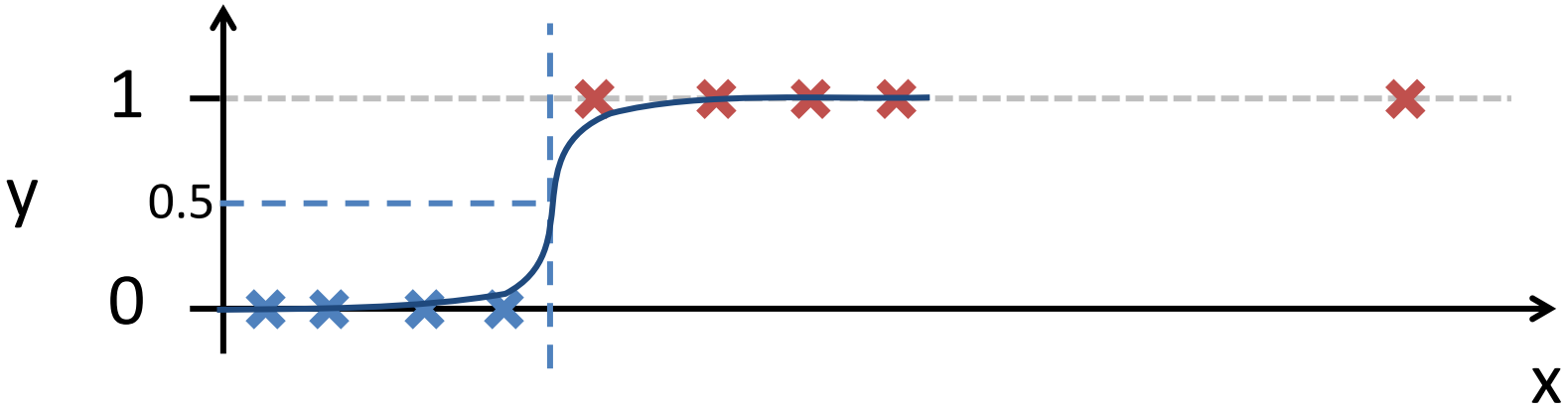


Logistic Regression Model

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



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



Interpretation of hypothesis output

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

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

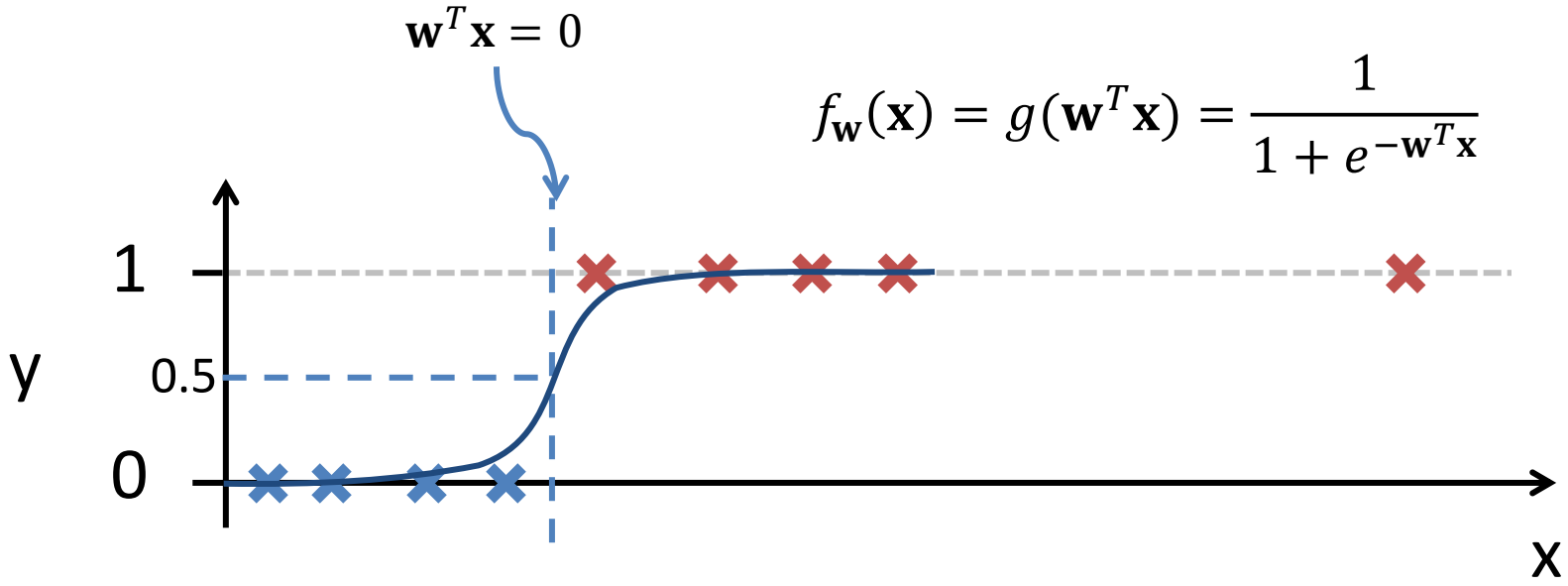
$$P(y = 0|\mathbf{x}; \mathbf{w}) = 1 - P(y = 1|\mathbf{x}; \mathbf{w})$$

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

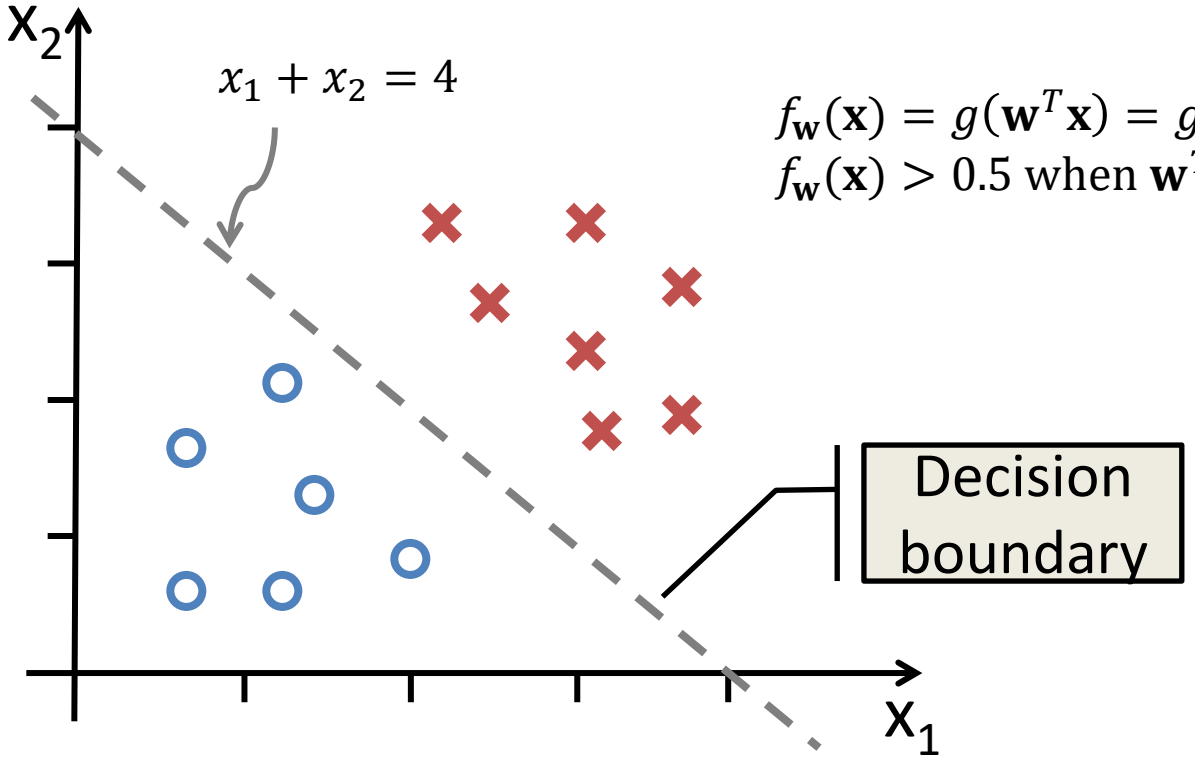
The Decision Boundary

Suppose we predict "$y = 1$" if $f_{\mathbf{w}}(\mathbf{x}) \geq 0.5$
predict "$y = 0$" if $f_{\mathbf{w}}(\mathbf{x}) < 0.5$

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



Decision Boundary



$$f_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x}) = g(w_0 + w_1 x_1 + w_2 x_2)$$
$$f_{\mathbf{w}}(\mathbf{x}) > 0.5 \text{ when } \mathbf{w}^T\mathbf{w} > 0$$

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

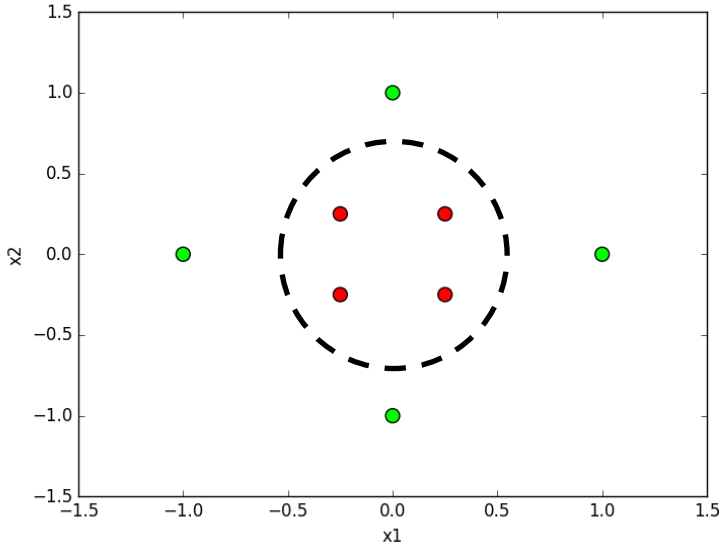
$$y = 1, \text{ when} \quad x_1 + x_2 \geq 4$$

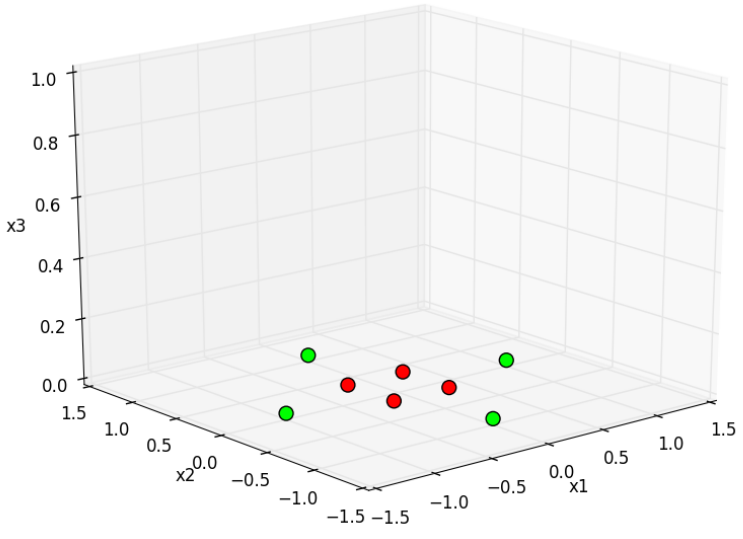
$$e.g. \quad \mathbf{w} = \begin{bmatrix} -4 \\ 1 \\ 1 \end{bmatrix}$$

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

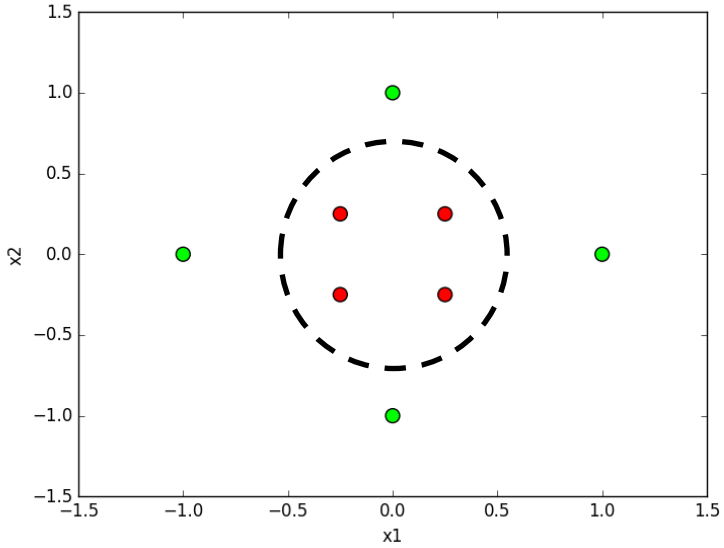
$$y = 0, \text{ when} \quad x_1 + x_2 < 4$$

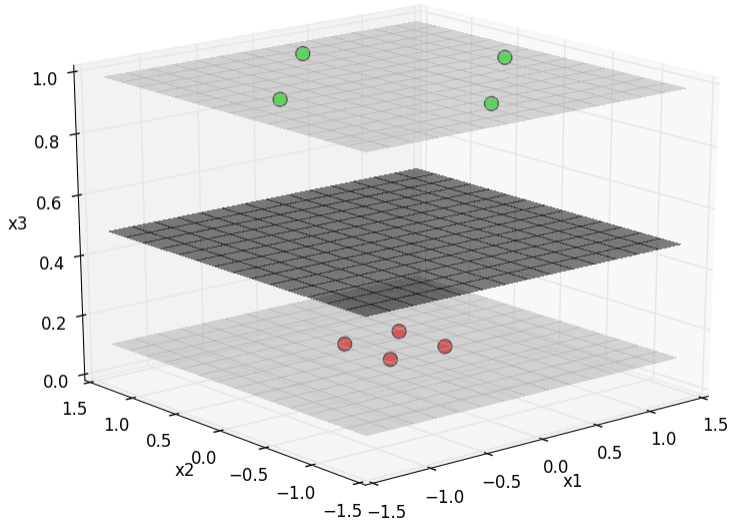
Non linear decision boundaries





Non linear decision boundaries

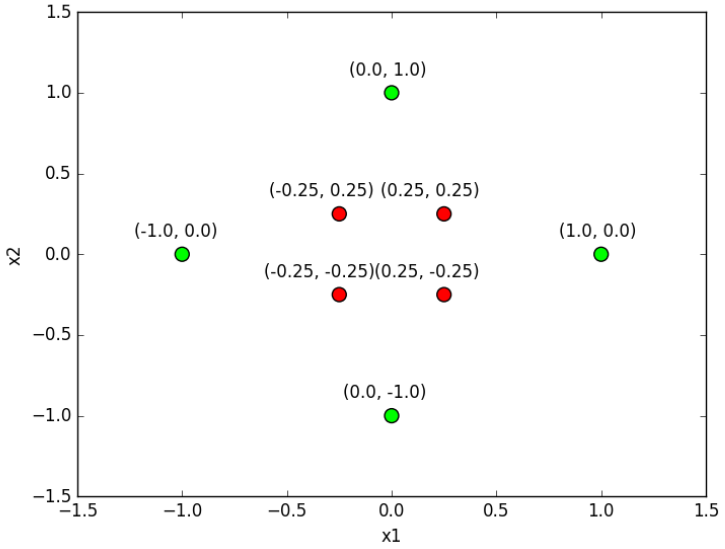




Which method is better?

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

Non linear decision boundaries

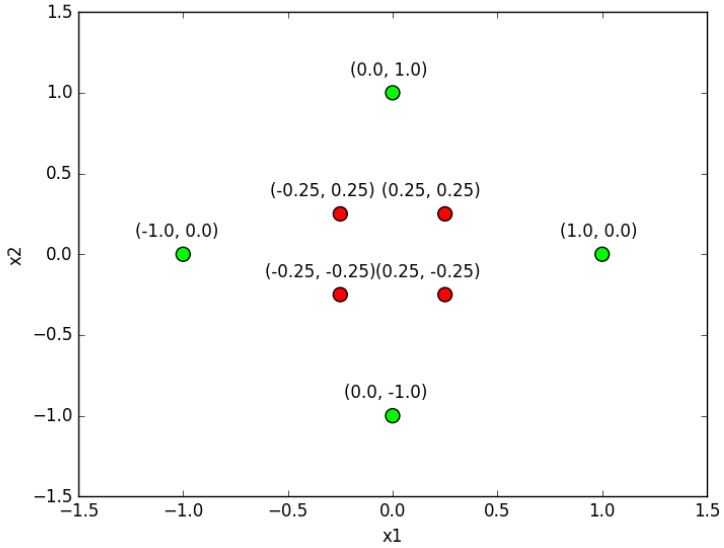


We need a way to be able to select a class, by applying a threshold, i.e. we need to find a function that gives the green points a higher value than the red ones or vice versa

Which of the following would work?

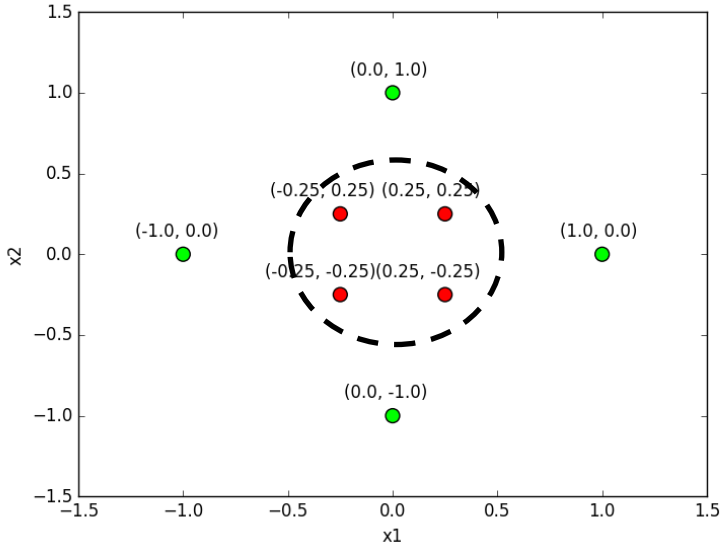
1. $x_1 + x_2$ (linear regression)
2. $x_1^2 + x_2^2$
3. $x_1 x_2$

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

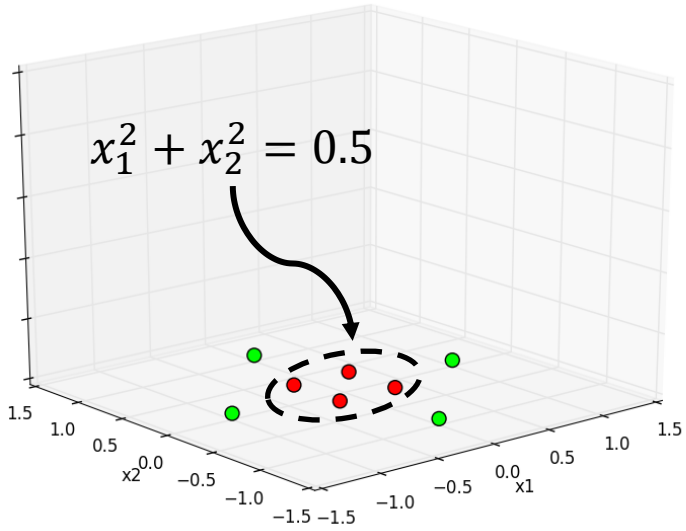
Non linear decision boundaries



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

$x_1^2 + x_2^2 = 0.125$ $\quad x_1^2 + x_2^2 = 0.5$ $\quad x_1^2 + x_2^2 = 1$

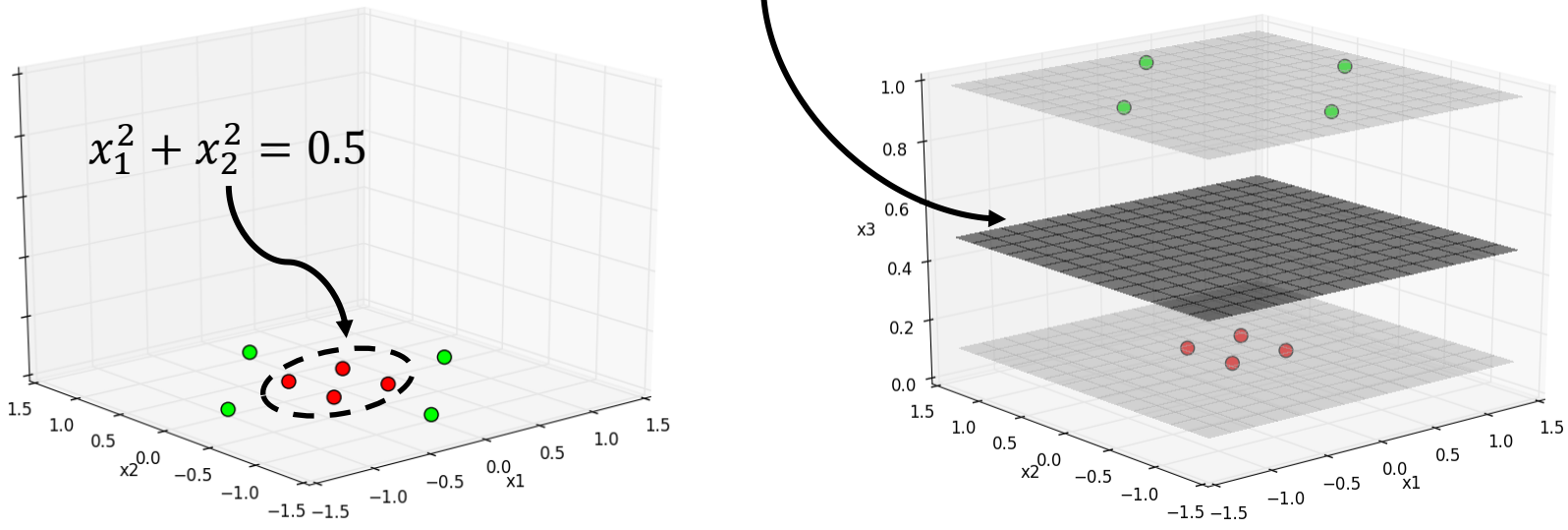
Non Linear Decision Boundaries



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

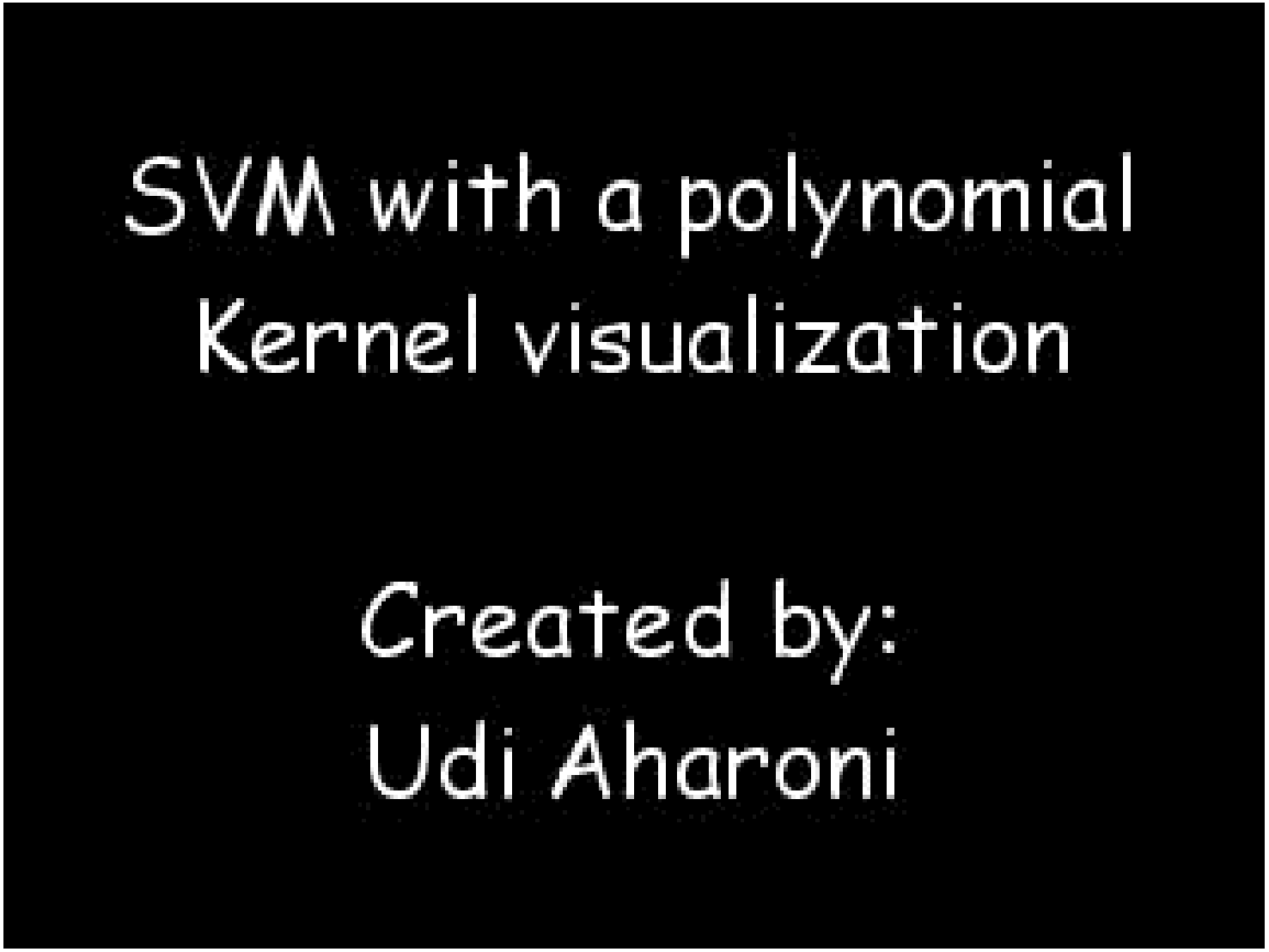
Non Linear Decision Boundaries

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

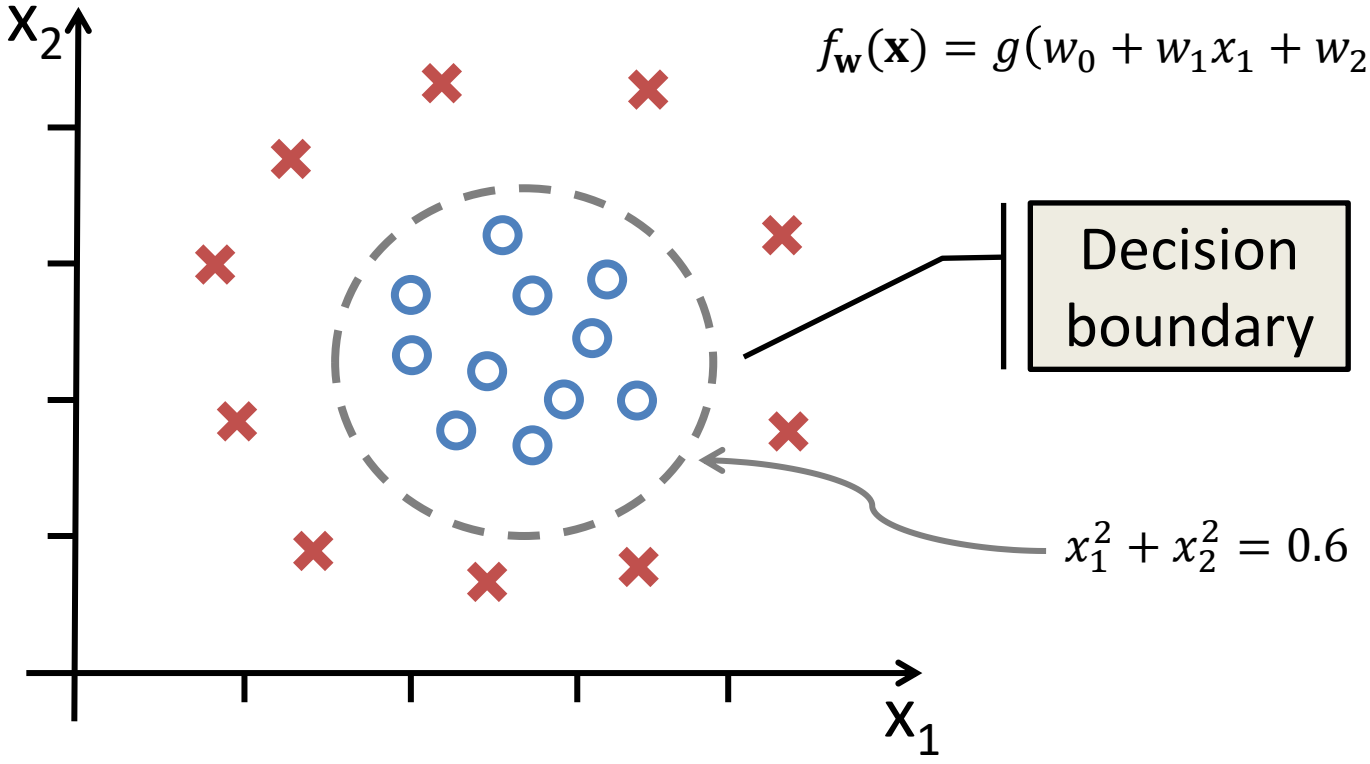


$$(x_1, x_2) \rightarrow (x_1, x_2, x_3 = x_1^2 + x_2^2)$$

Non-Linearly Separable Classes



Non linear decision boundaries



$$f_{\mathbf{w}}(\mathbf{x}) = g(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2)$$

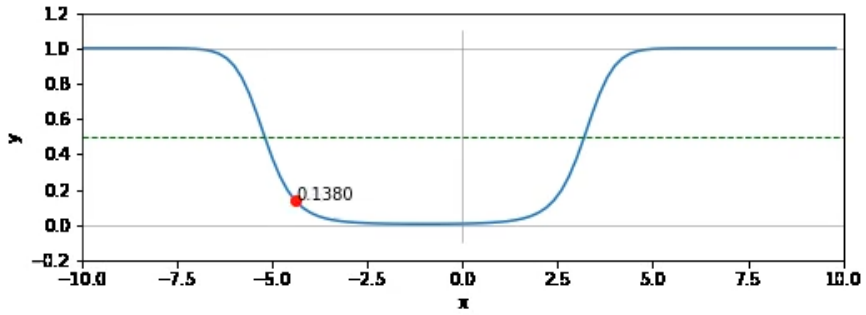
$$e.g. \quad \mathbf{w} = \begin{bmatrix} -0.6 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

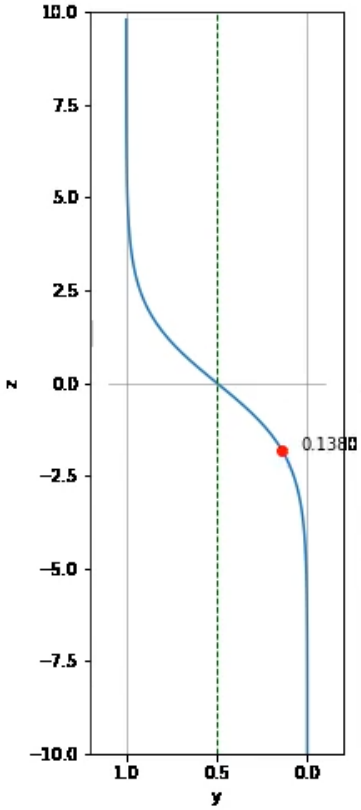
$y = 1$, when $f_{\mathbf{w}}(\mathbf{x}) \geq 0.5 \Leftrightarrow \mathbf{w}^T\mathbf{x} = -0.6 + x_1^2 + x_2^2 \geq 0$

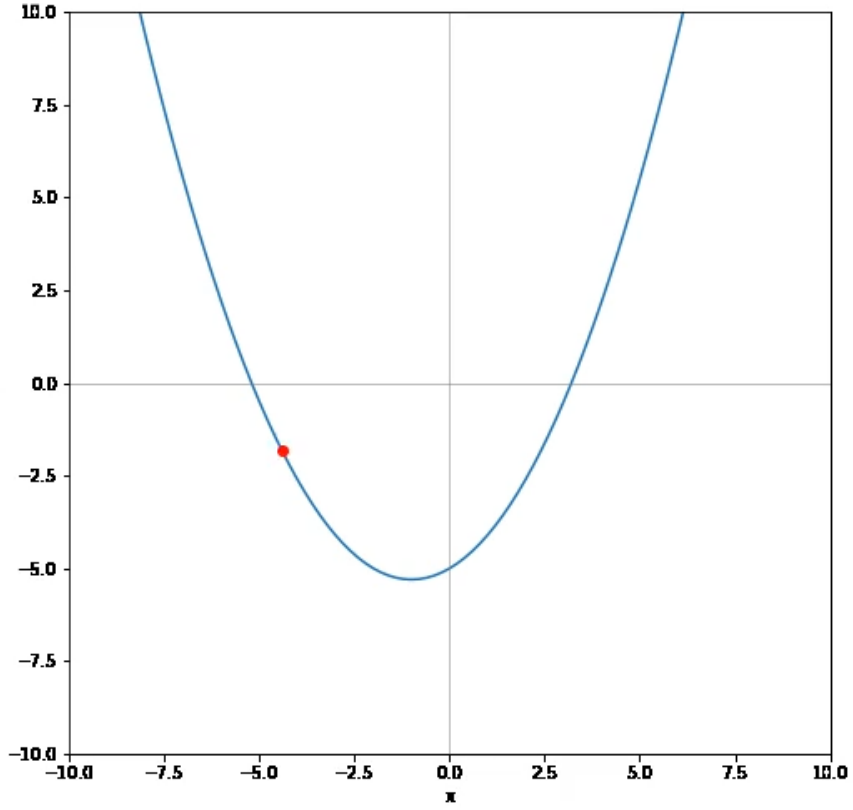
$y = 1$, when $x_1^2 + x_2^2 \geq 0.6$

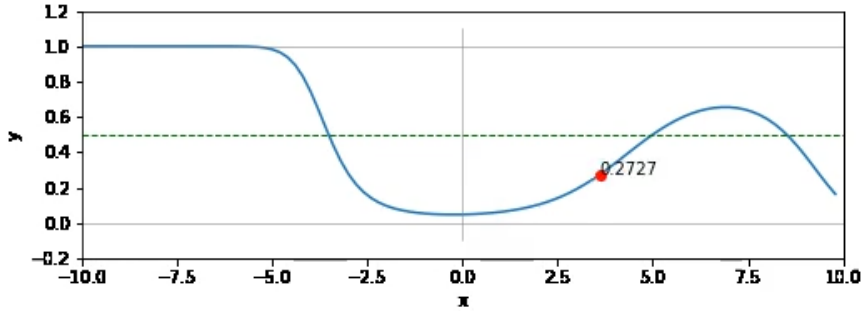
$y = 0$, when $f_{\mathbf{w}}(\mathbf{x}) < 0.5 \Leftrightarrow \mathbf{w}^T\mathbf{x} = -0.6 + x_1^2 + x_2^2 < 0$

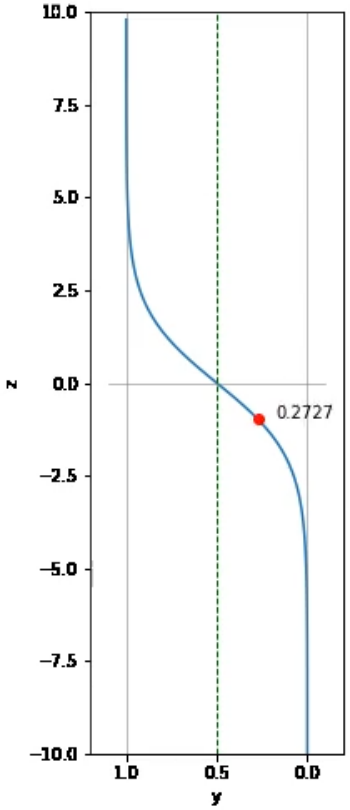
$y = 0$, when $x_1^2 + x_2^2 < 0.6$

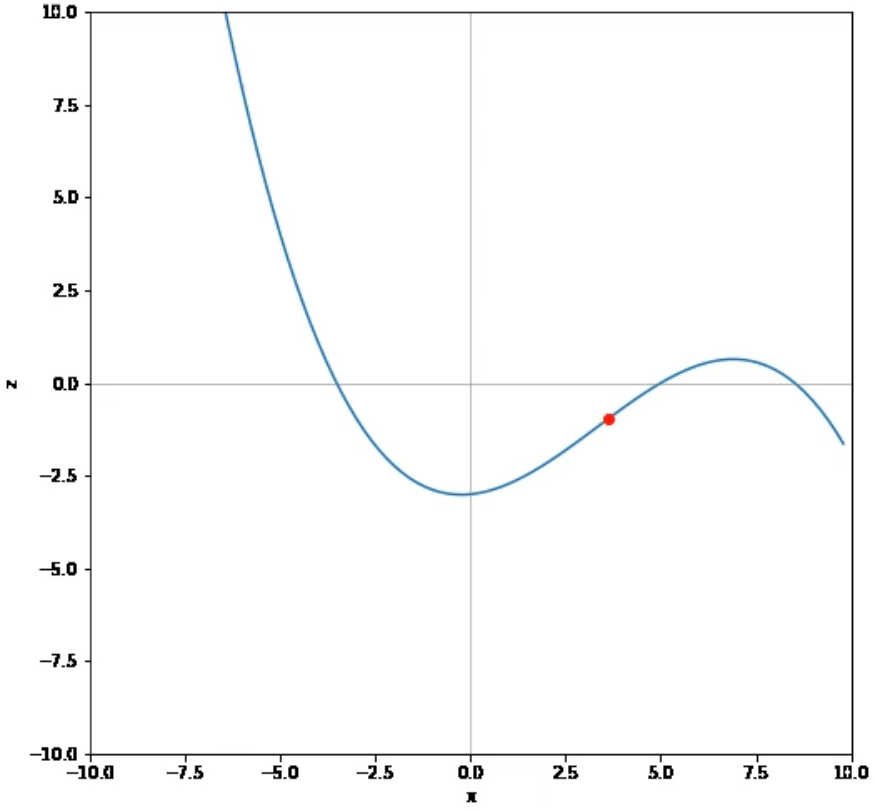

1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
y
0.1380
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
0.1380
1.0
0.5
0.0
y


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

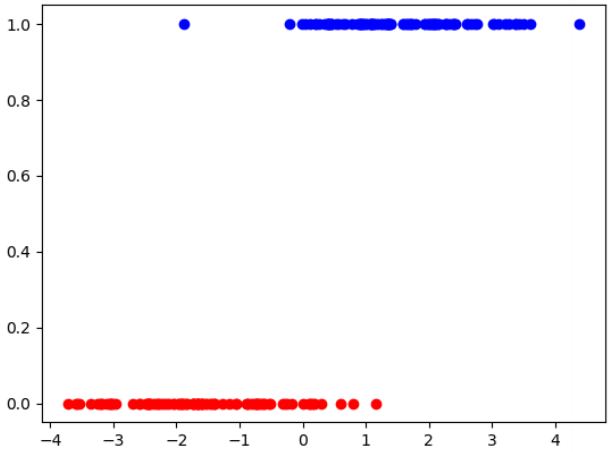

1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
y
0.2727
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

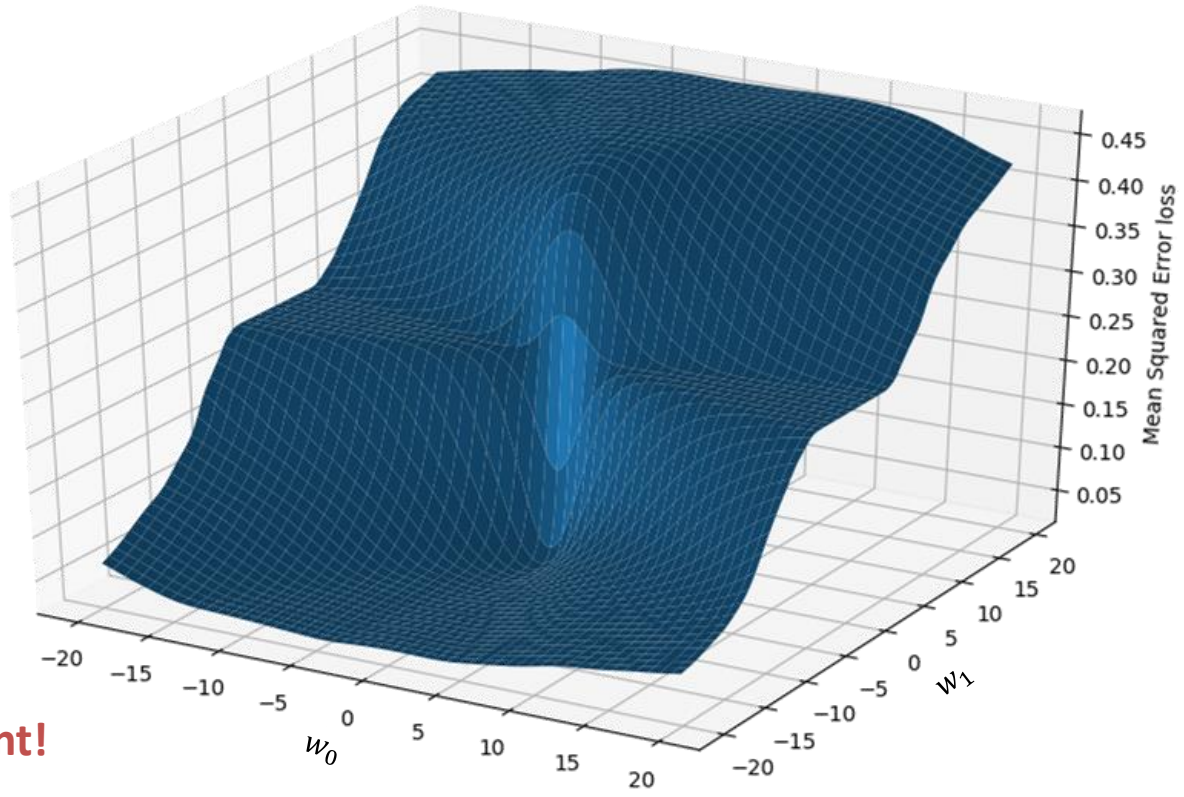

10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
0.2727
1.0
0.5
0.0
y


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
z
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

Mean Squared Error cost

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

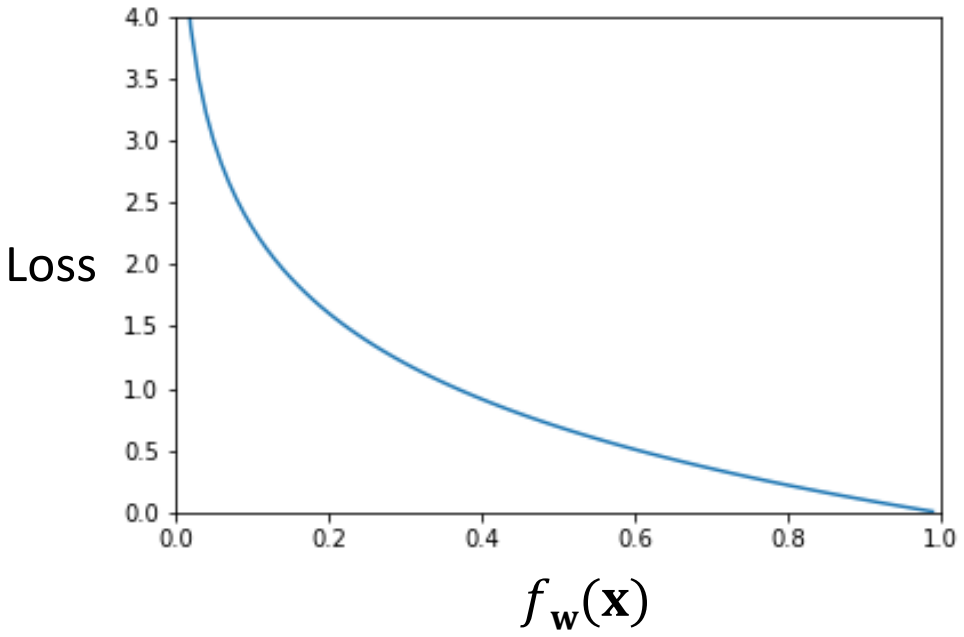




Not an easy one to descent!

Loss function for logistic regression

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



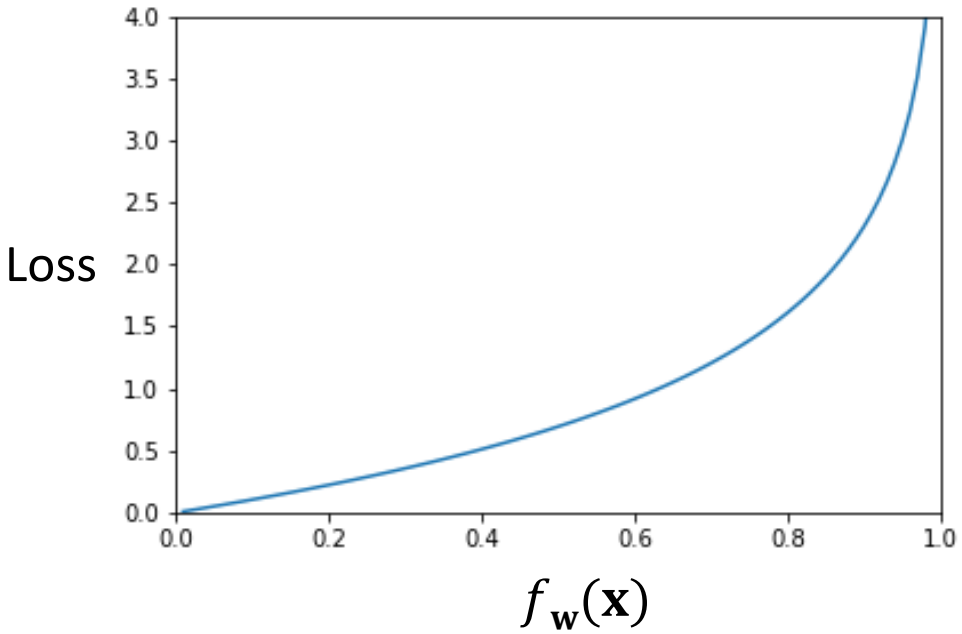
If ***y=1*** AND $f_{\mathbf{w}}(\mathbf{x}) = 1$ (real data and our prediction agree) then the ***Loss = 0***

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

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

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



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

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

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

Simplified loss function

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

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

(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

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

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log f_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - f_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right)\right)\right]$$

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

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

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log f_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - f_{\mathbf{w}}\left(\mathbf{x}^{(i)}\right)\right)\right]$$

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

Repeat
{

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

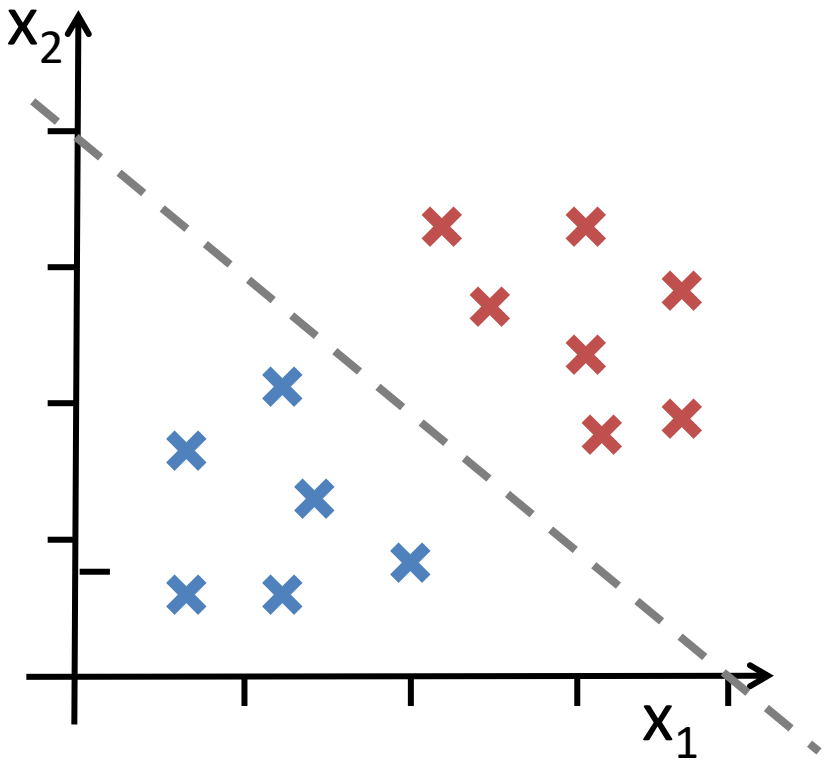
(simultaneously for all w_j)

}

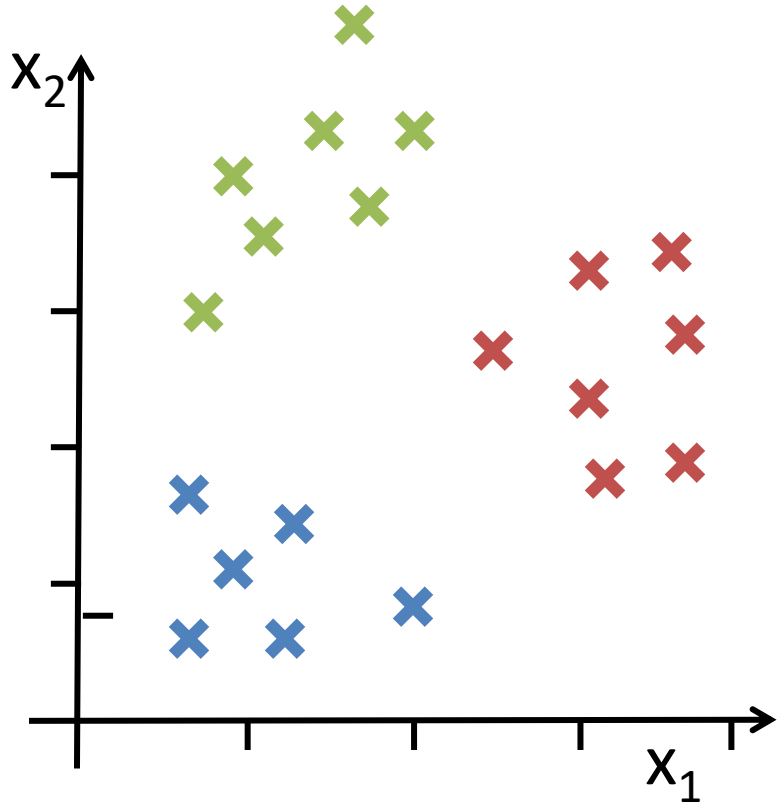
Algorithm looks identical to linear regression, but $f(\mathbf{w})$ is different here!

MULTICLASS CLASSIFICATION

Binary vs Multi-class classification

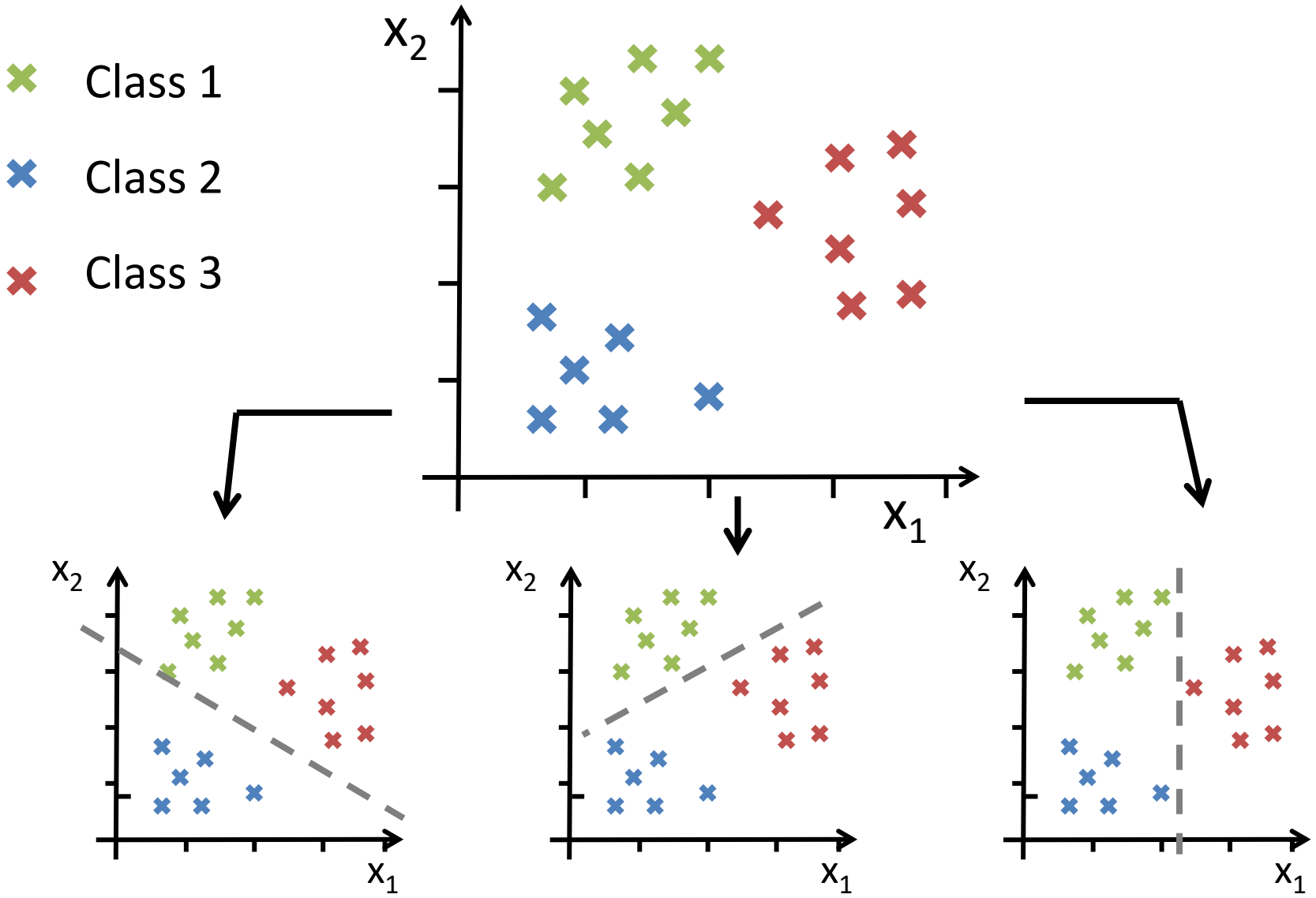


Binary: *y = {0,1}*



Multi-class: *y = {0,1, …, n}*

One vs all (one vs rest)



One vs All Classification

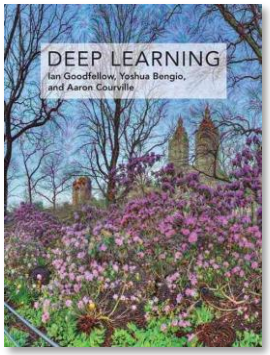
Train a (logistic regression) classifier $f_{\mathbf{w}}^{(c)}(\mathbf{x})$ for each class *c* to predict the probability that *y = c*

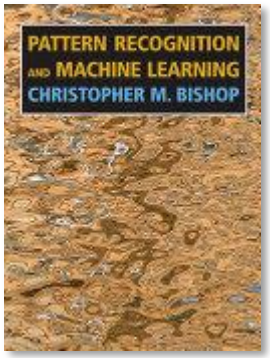
To make a prediction on a new input **x**, pick the class c that maximizes the probability $f_{\mathbf{w}}^{(c)}(\mathbf{x})$

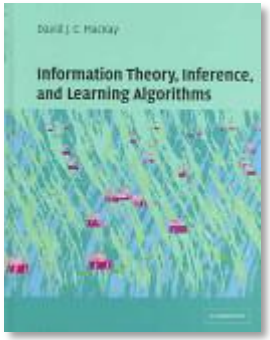
$$\underset{c}{\operatorname{argmax}} f_{\mathbf{w}}^{(c)}(\mathbf{x}) \Leftrightarrow \underset{c}{\operatorname{argmax}} P(y = c|\mathbf{x}, \mathbf{w})$$

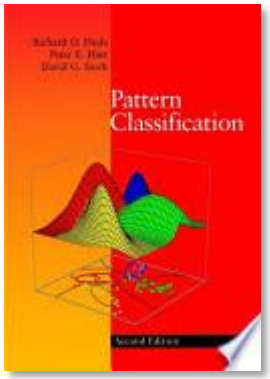
Resources (I)

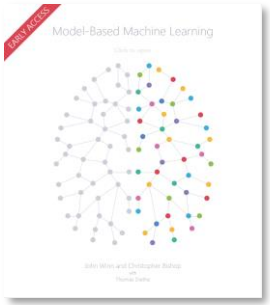
- *I. Goodfellow, Y. Bengio, A. Courville, "Deep Learning", MIT Press, 2016*
  http://www.deeplearningbook.org/
- *C. Bishop, "Pattern Recognition and Machine Learning", Springer, 2006*
  http://research.microsoft.com/en-us/um/people/cmbishop/prml/index.htm
- *D. MacKay, "Information Theory, Inference 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/





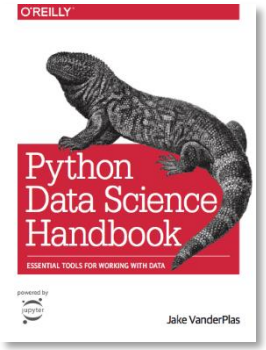




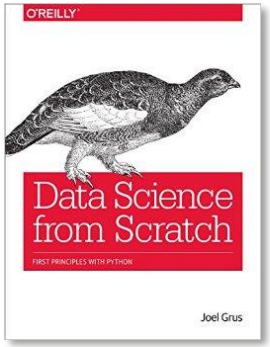


Resources (II)

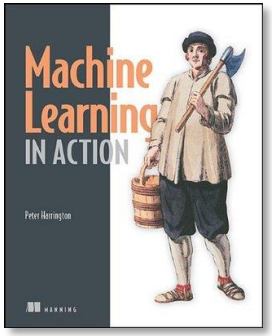
- *"Python Data Science Handbook: Essential tools for working with Data", Jake VanderPlas, O'Reilly Media, 2016, 1st Ed.* https://jakevdp.github.io/PythonDataScienceHandbook/



- *"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/