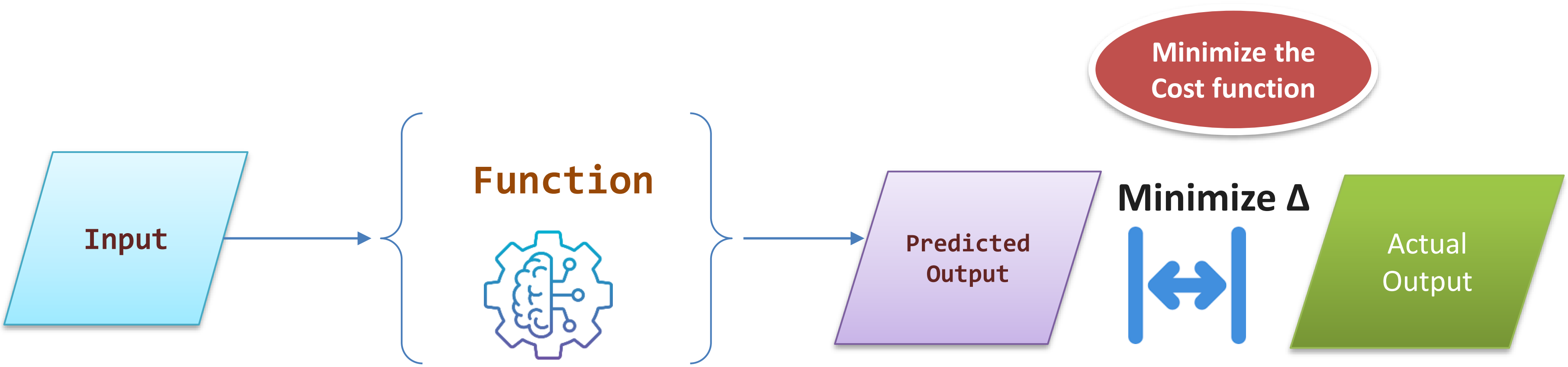
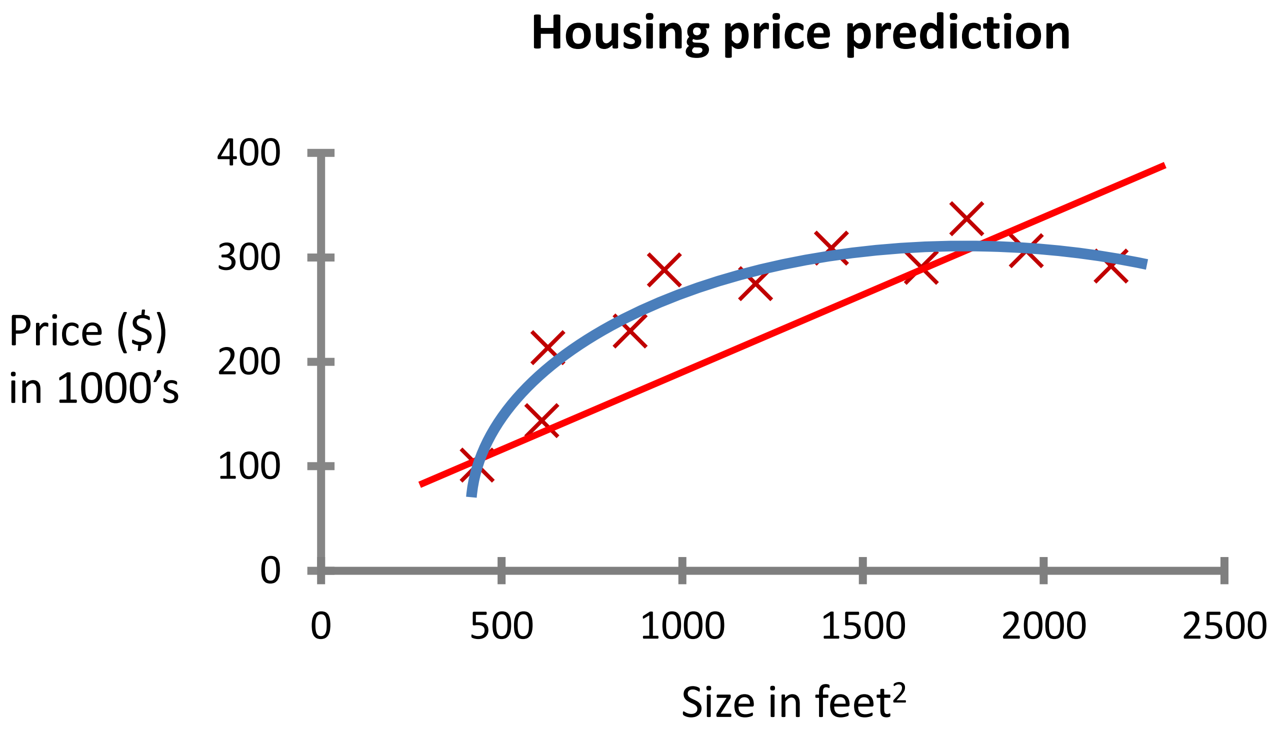
Regression

ML: learn a Function that minimizes the cost

- Start with random function parameters
- Repeat intelligent guessing/approximation of the Function parameters such that the difference between the Predicted Output the Actual Output is reduced
  - i.e., minimize a Cost function a.k.a loss, or error function



Linear Regression with One Variable



<u>Regression</u>: Predict continuous output value (price)

Training set of housing prices

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

Notation:

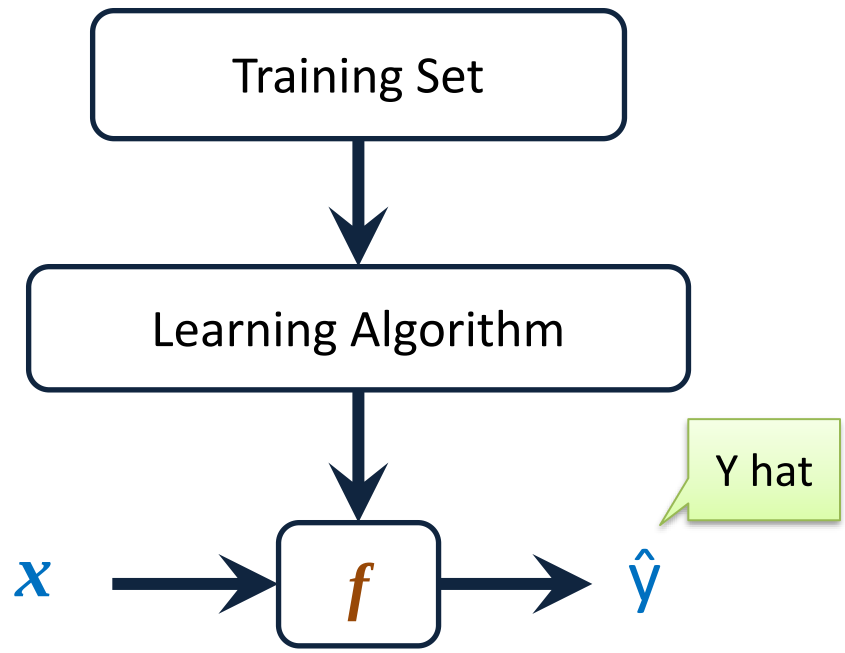
- $\boldsymbol{m}$ = Number of training examples
- $\boldsymbol{x}$ = "input" variable / features
- $\boldsymbol{y}$ = "output" variable / "target" variable

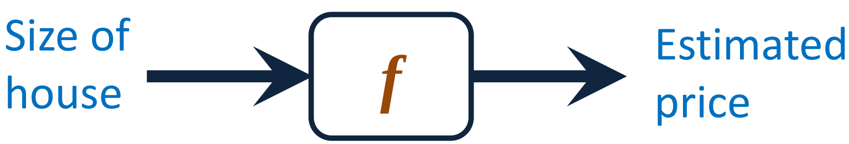
$(x^{(i)}, y^{(i)})$ – the $\boldsymbol{i}^{th}$ training example

$x^{(1)} = 2104$
$x^{(2)} = 1416$
$y^{(1)} = 460$

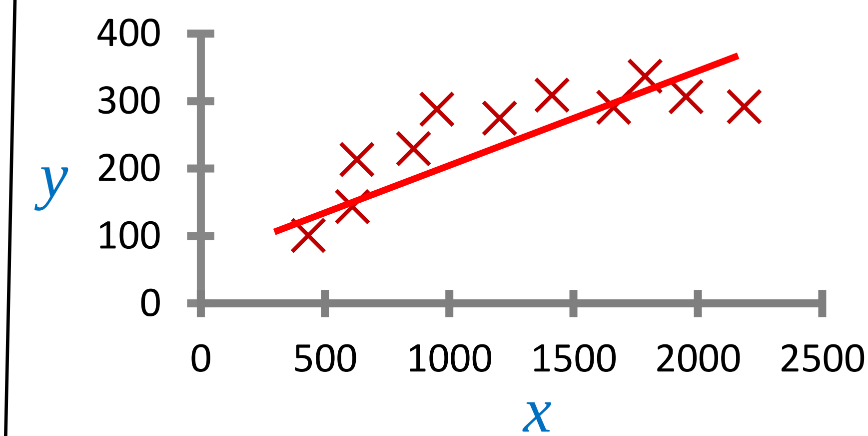
How do we represent f ?

$$f(x) = wx + b$$





w, b are parameters (coefficients) to learn from the training set



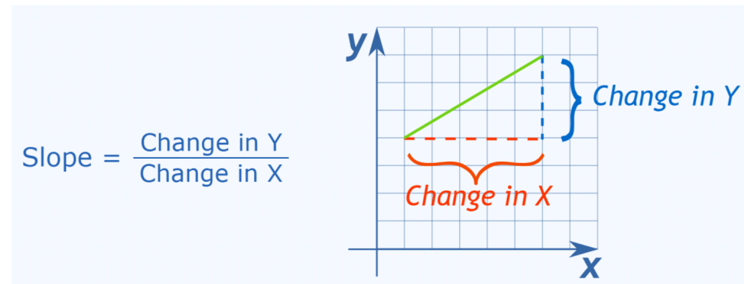
Linear regression with one variable
Univariate linear regression

Given a training set, **learn a function** *f* so that *f(x)* is a "good" predictor for the corresponding value of y (i.e. minimize the error between predicted and actual values)

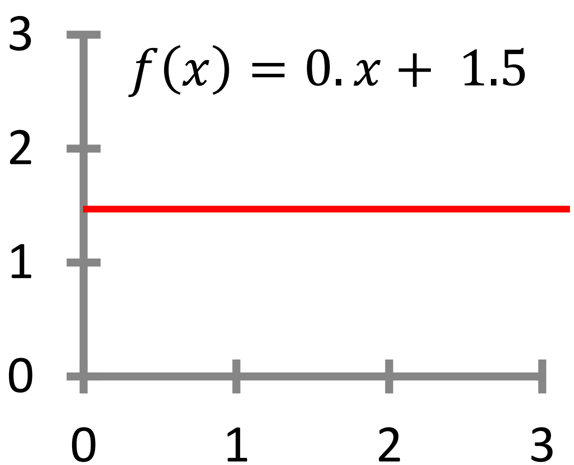
Univariate Linear Regression - Model Representation

$$\boldsymbol{f(x) = wx + b}$$

- ***w*** is the slope of the line
- ***b*** is the y-intercept of the line

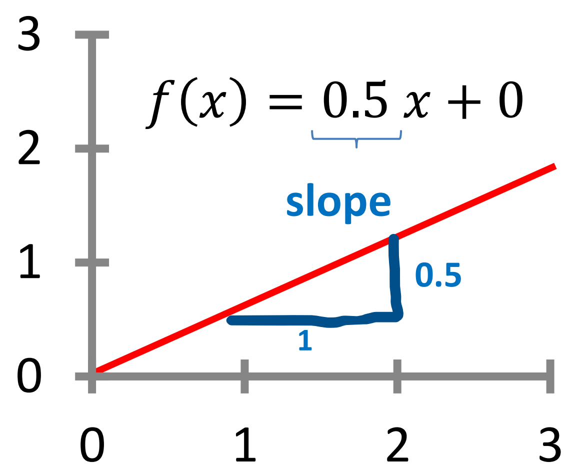


How to choose $\boldsymbol{w}$ and $\boldsymbol{b}$?



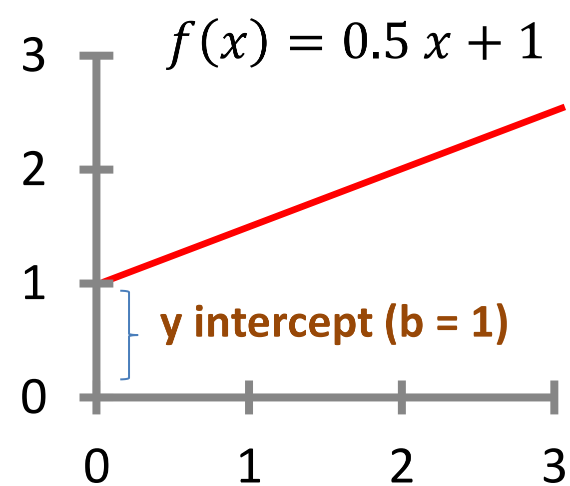
$w = 0$

$b = 1.5$



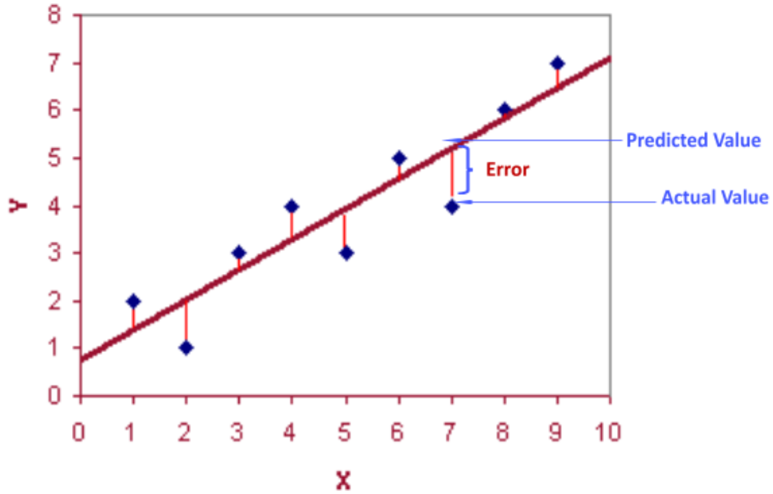
$w = 0.5$

$b = 0$



$w = 0.5$

$b = 1$



Idea: Choose $\boldsymbol{w}$ and $\boldsymbol{b}$ so that $f(x)$ is close to y for our training examples (x, y)

Find w, b:
$\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$

Cost (mean squared error)

Function:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(f\left(x^{(i)}\right) - y^{(i)})^2$$

Goal: $\underset{w,\,b}{\text{minimize}}\ J(w,b)$

With **m** = number of training examples

Function:

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{wx} + \boldsymbol{b}$$

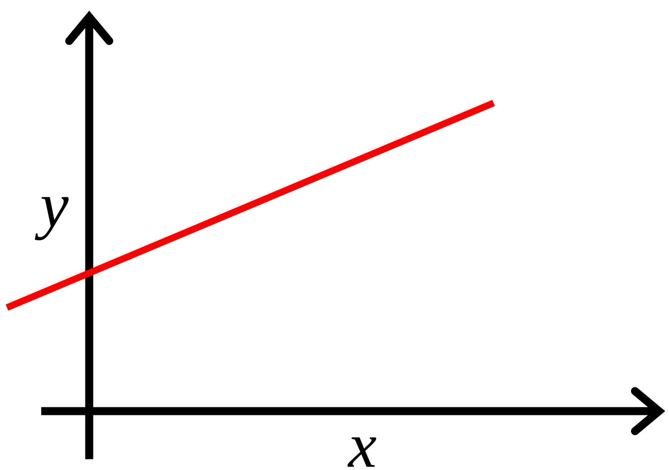
Parameters:

$\boldsymbol{w}, \boldsymbol{b}$

Cost Function:

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

Goal: $\underset{w,b}{\text{minimize}}\ J(w,b)$



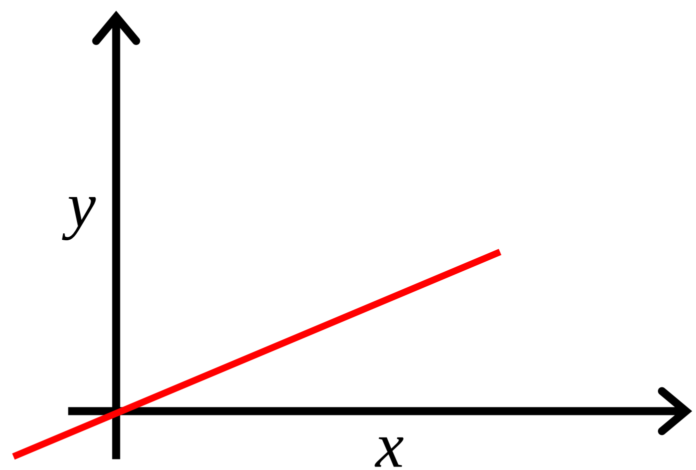
Simplified

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{wx}$$

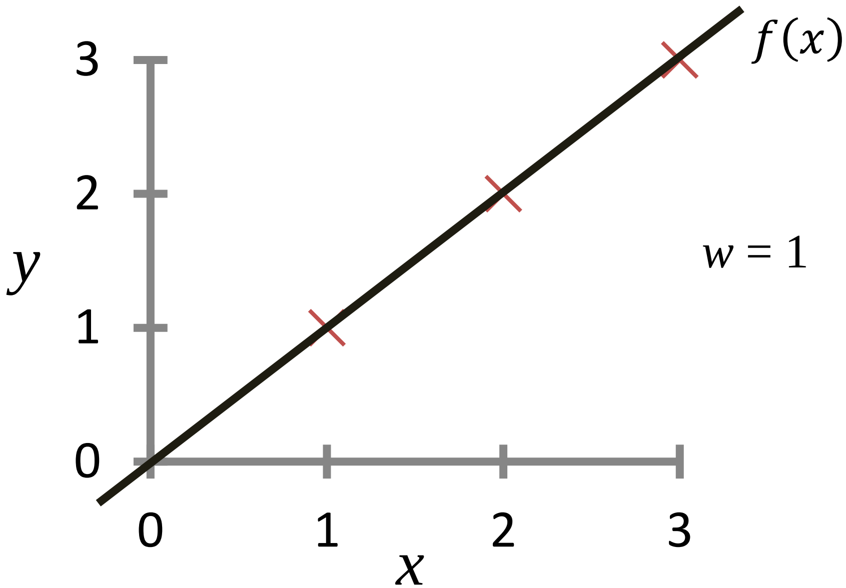
$\boldsymbol{w}$

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

$\underset{w}{\text{minimize}}\ J(w)$



$$\boldsymbol{f(x) = wx}$$

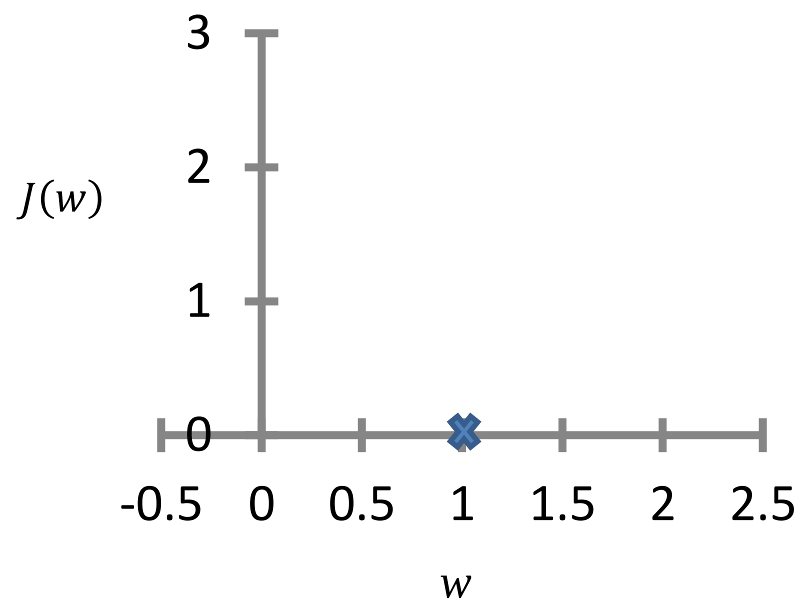


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2$$

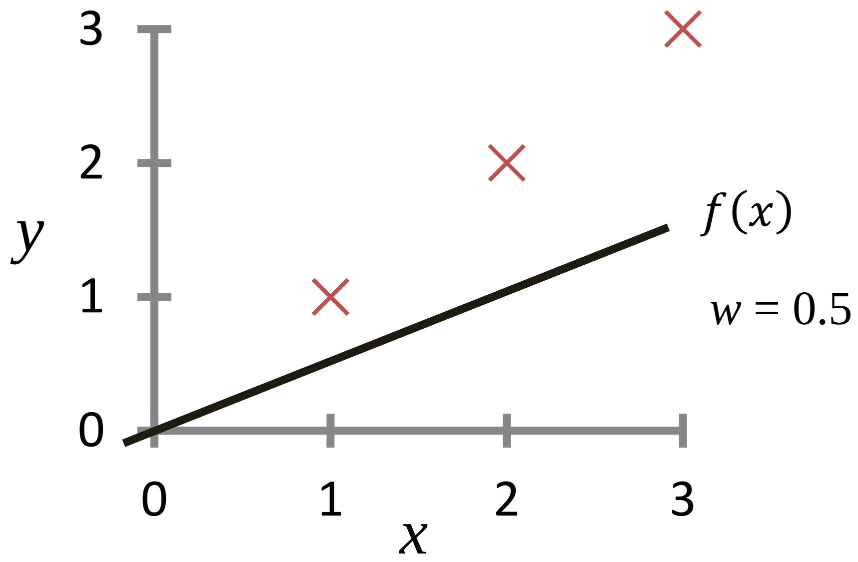
$$= \frac{1}{2m}(0^2 + 0^2 + 0^2) = 0$$

$$\boldsymbol{J(w)}$$

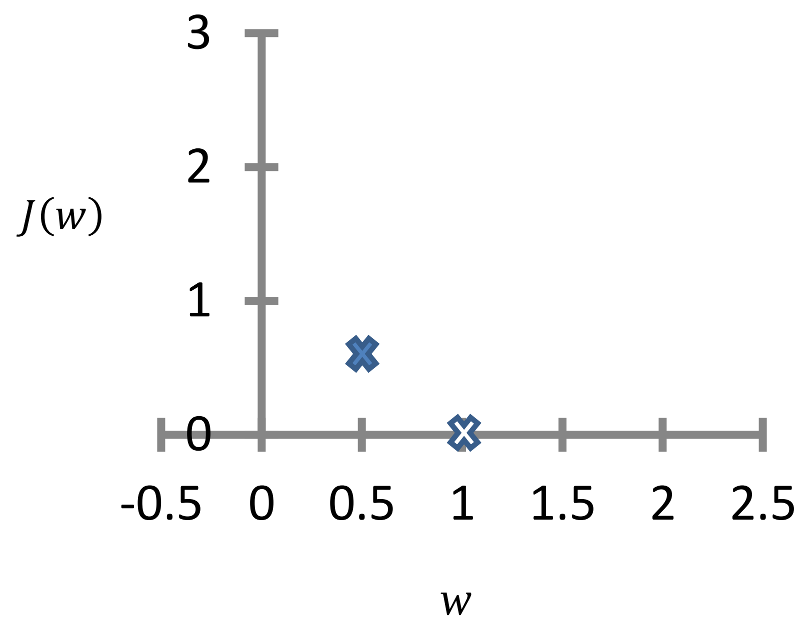
(function of the parameter w)



$\boldsymbol{f(x) = wx}$



$\boldsymbol{J(w)}$

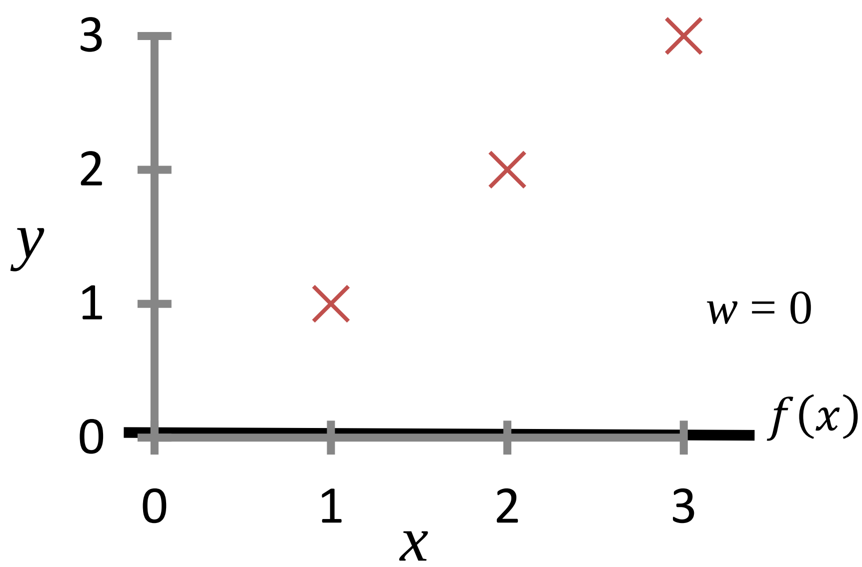


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m}((0.5 - 1)^2 + (1\text{-}2)^2 + (1.5\text{-}3)^2)$$

$$= \frac{1}{2\times 3}(3.5) = \frac{3.5}{6} = 0.58$$

$$f(x) = wx$$

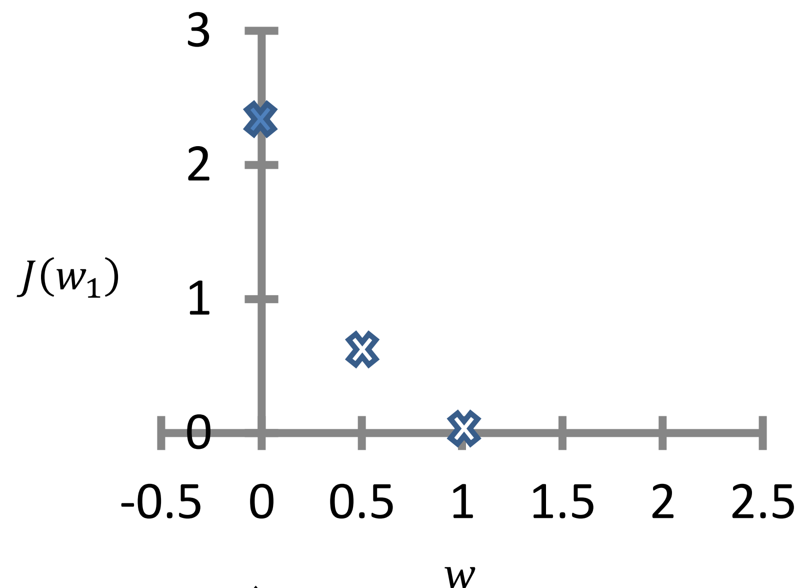


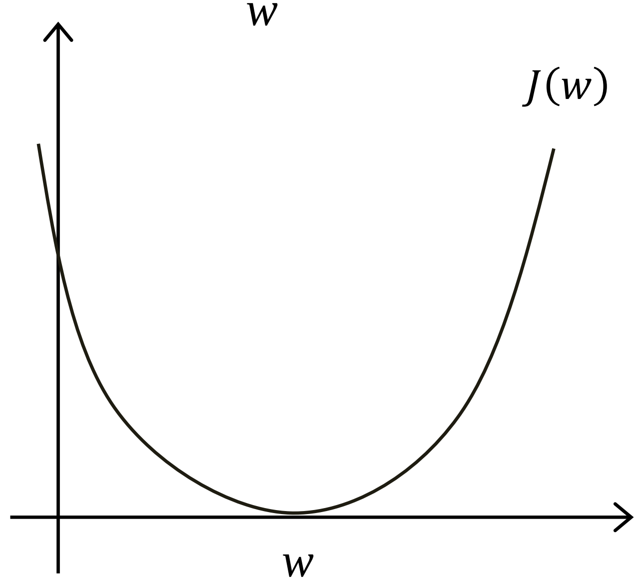
$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m}(1^2 + 2^2 + 3^2)$$

$$= \frac{1}{2\times3}(14) = \frac{14}{6} = 2.3$$

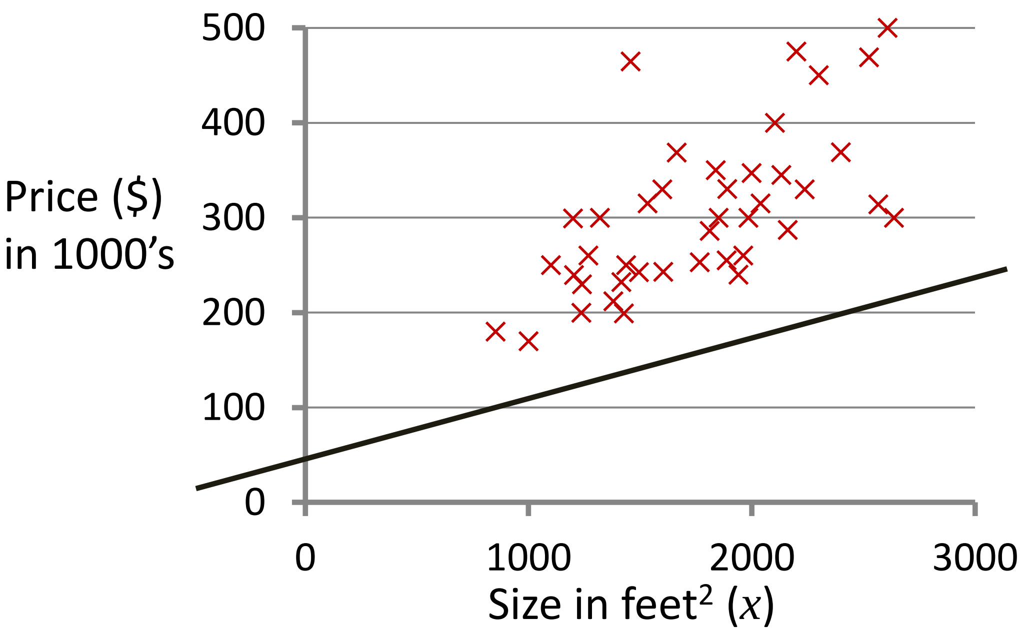
$$J(w)$$





$$\boldsymbol{f(x) = wx + b}$$

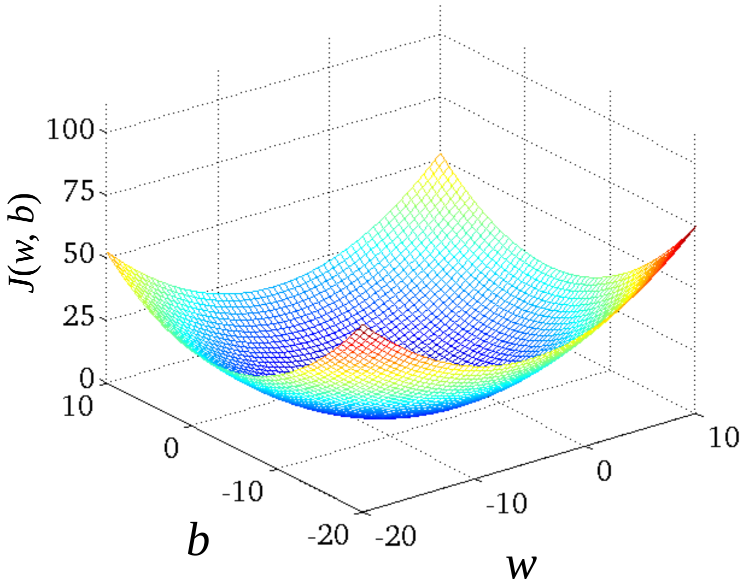
(for fixed w, b this is a function of x)



$$f(x) = 50 + 0.06x$$

$$\boldsymbol{J(w, b)}$$

(function of the parameters w, b)

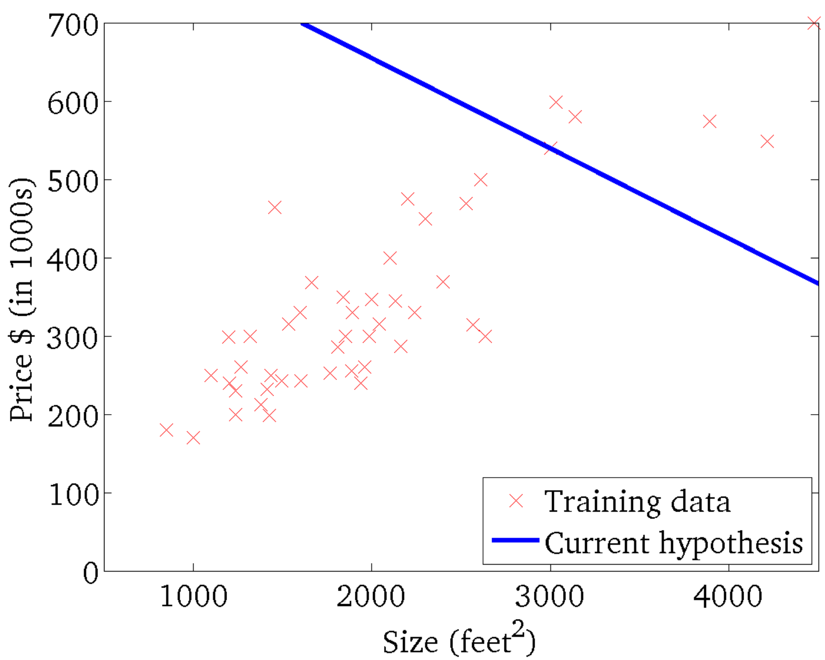


$$f(x) = wx + b$$

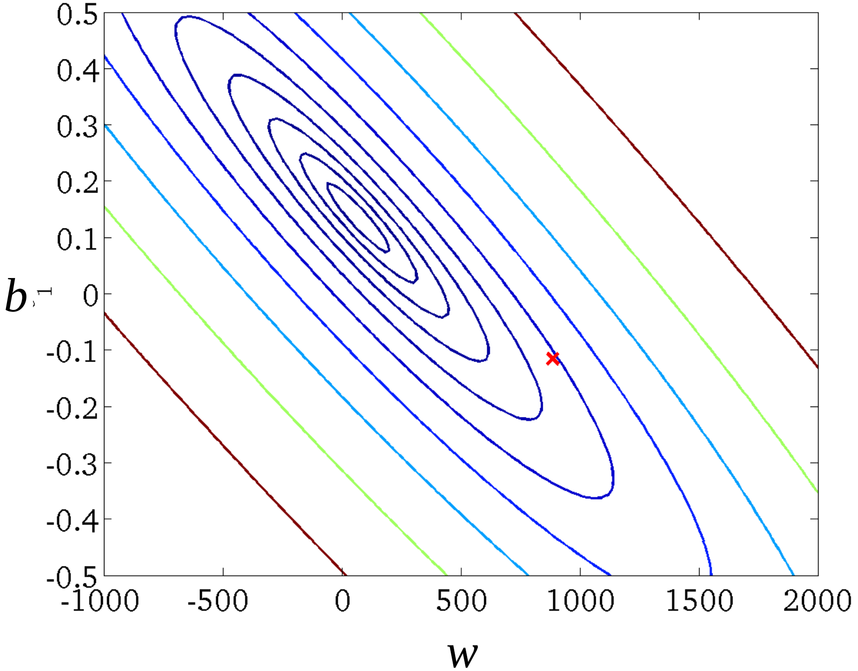
(for fixed w, b this is a function of x)

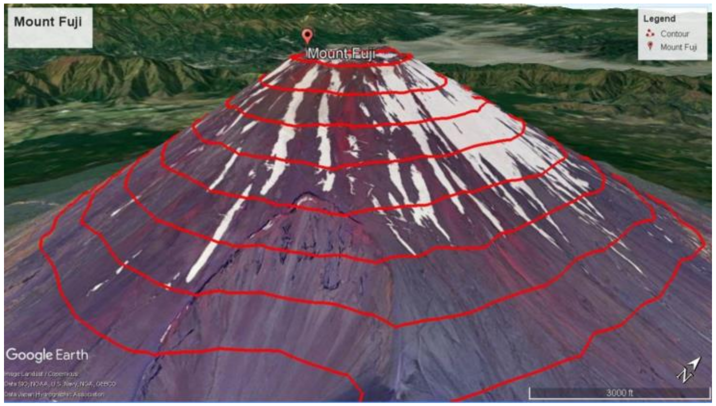
$$J(w, b)$$

(function of the parameters w, b)



Contour Plot



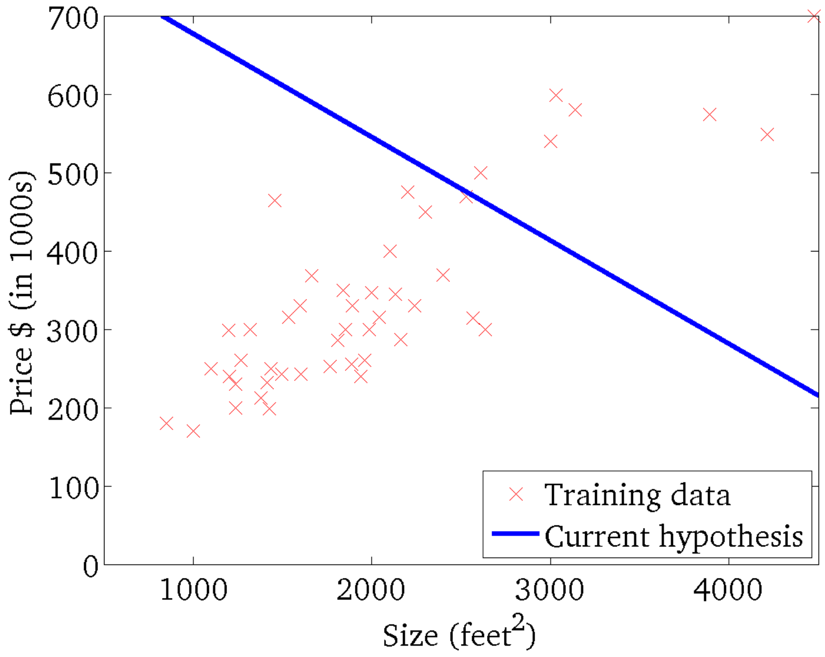


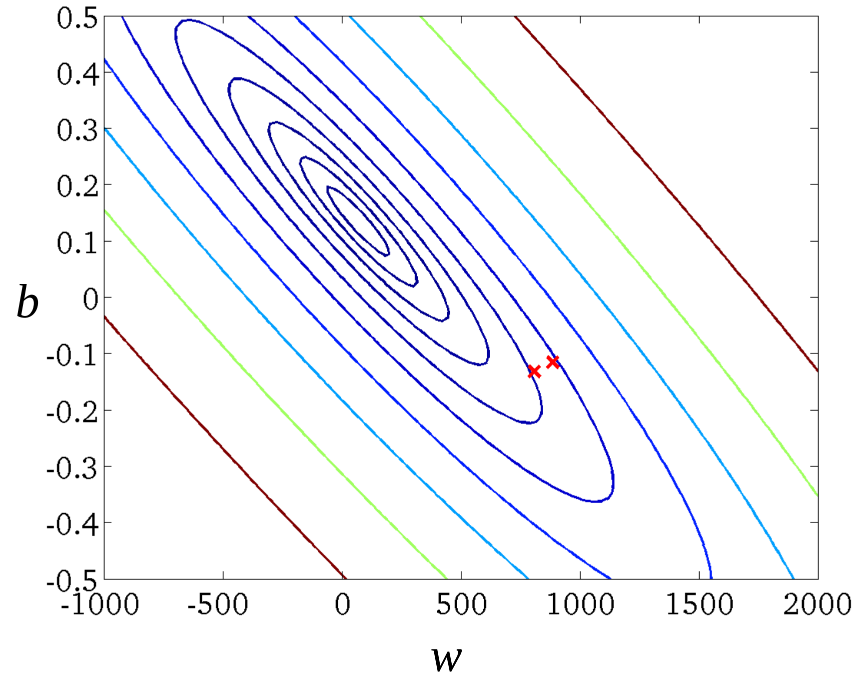
$$f(x) = wx + b$$

(for fixed w, b this is a function of x)

$$J(w, b)$$

(function of the parameters w, b)



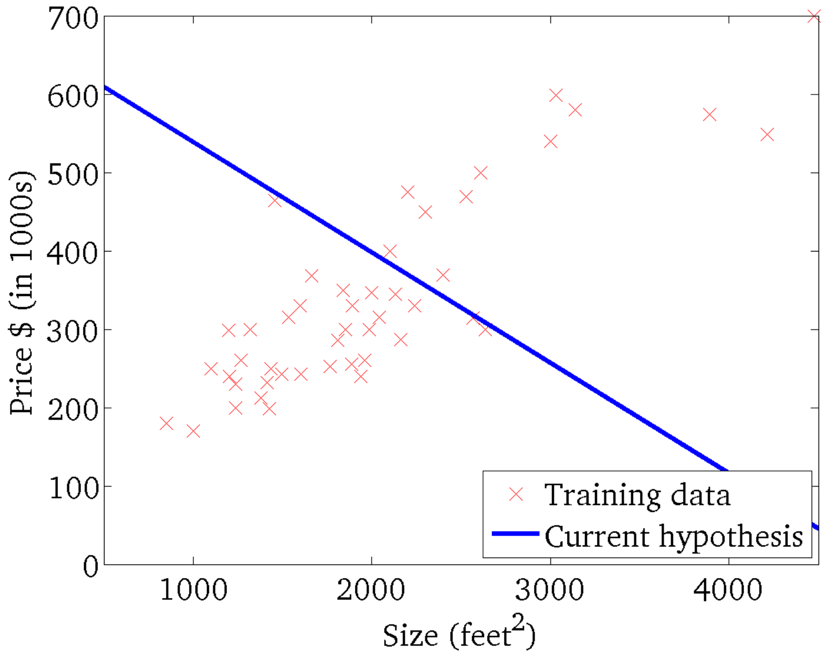


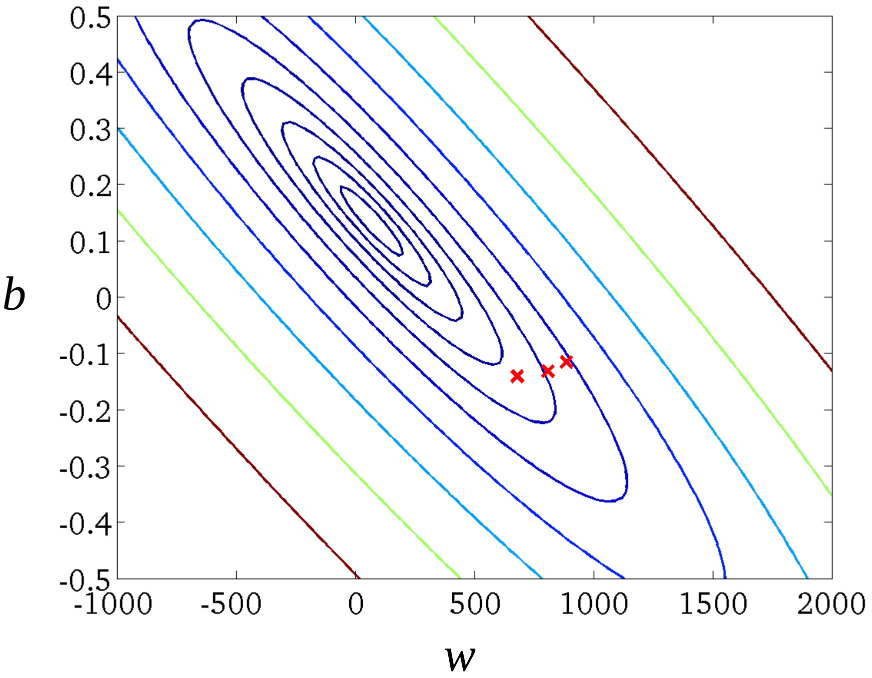
$$f(x) = wx + b$$

(for fixed w, b this is a function of x)

$$J(w, b)$$

(function of the parameters w, b)



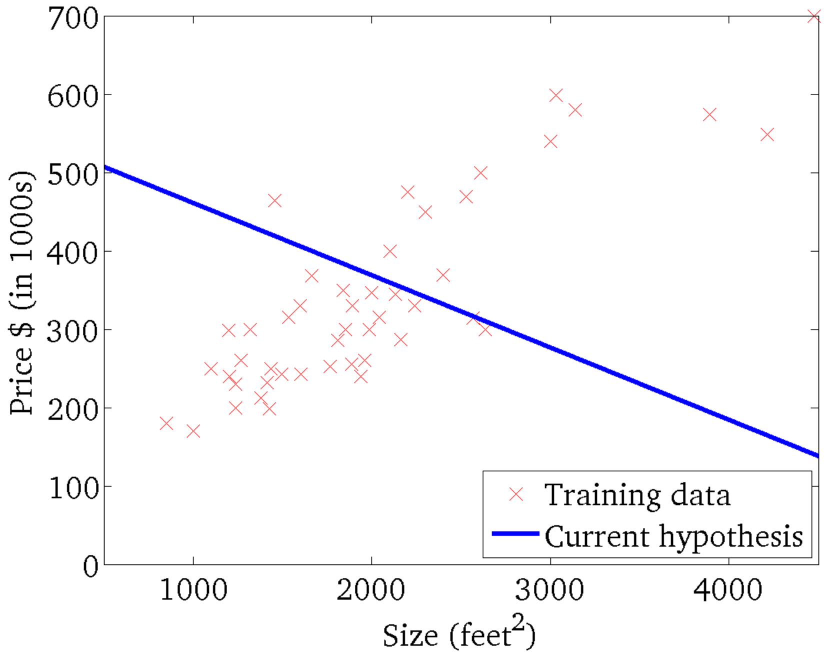


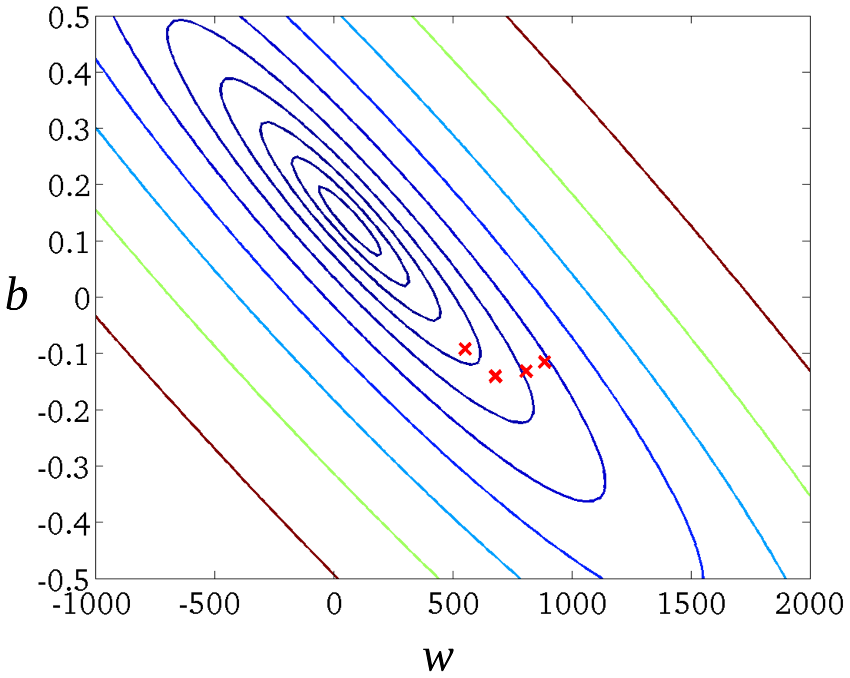
$$f(x) = wx + b$$

(for fixed w, b this is a function of x)

$$J(w, b)$$

(function of the parameters w, b)



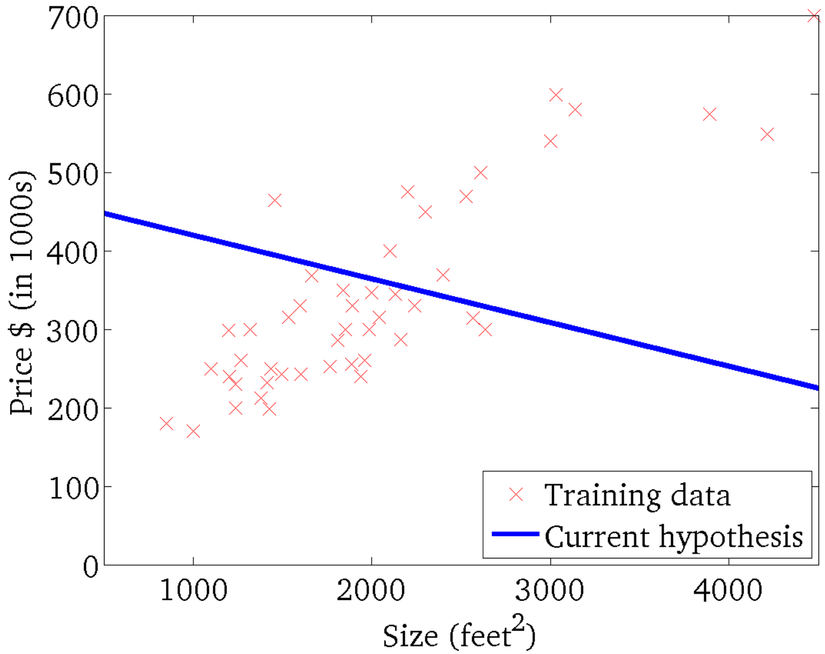


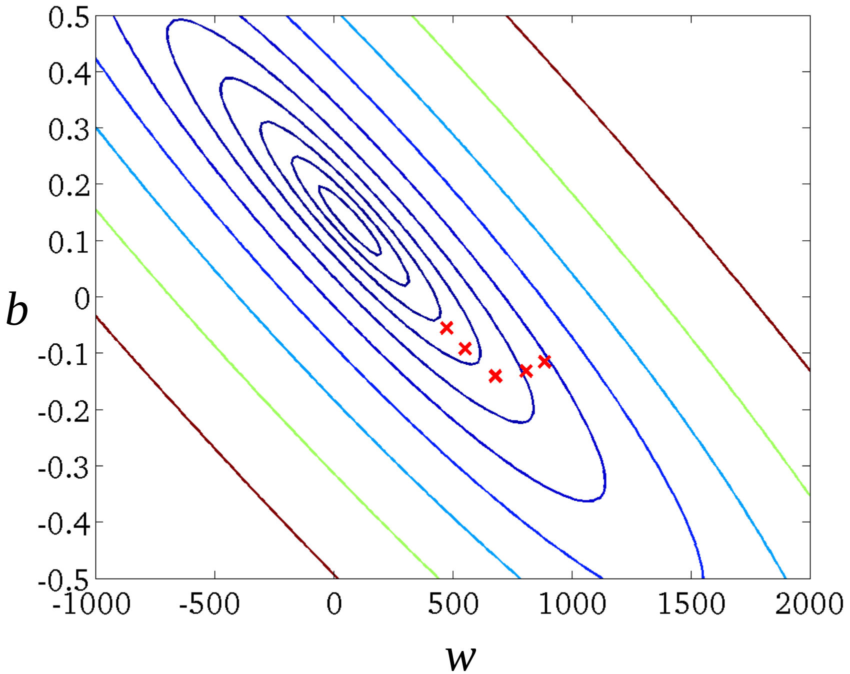
$$f(x) = wx + b$$

(for fixed w, b this is a function of x)

$$J(w, b)$$

(function of the parameters w, b)



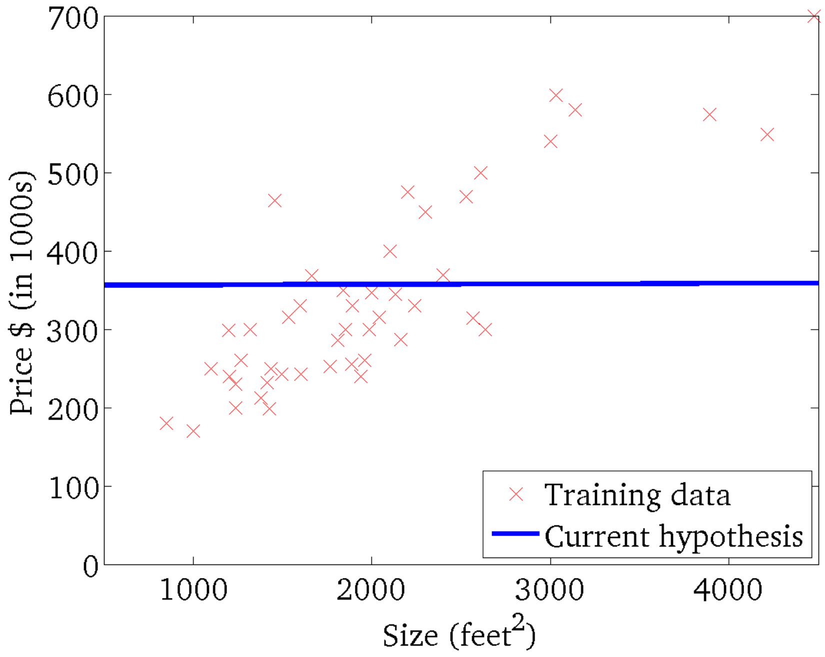


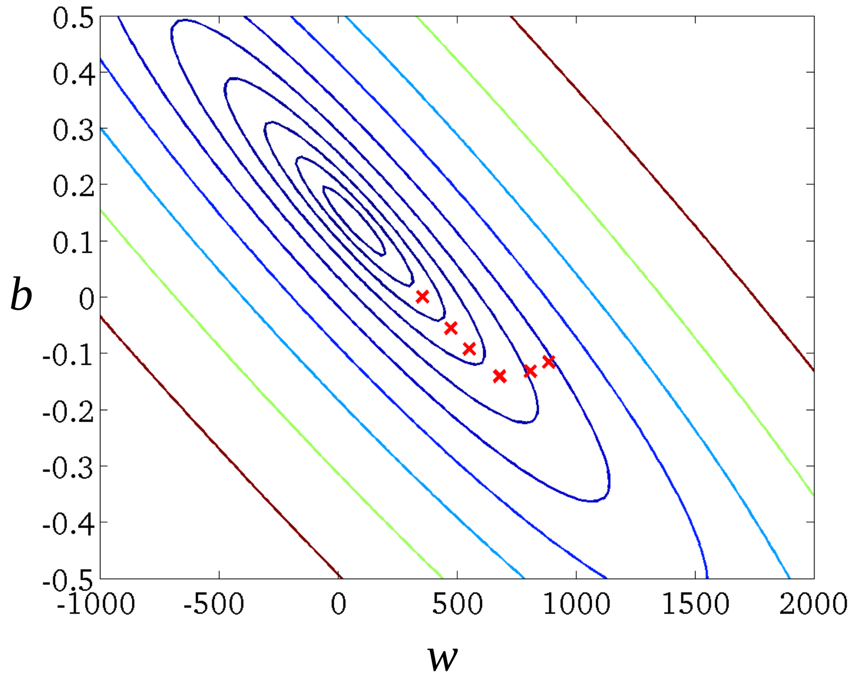
$$f(x) = wx + b$$

(for fixed w, b this is a function of x)

$$J(w, b)$$

(function of the parameters w, b)



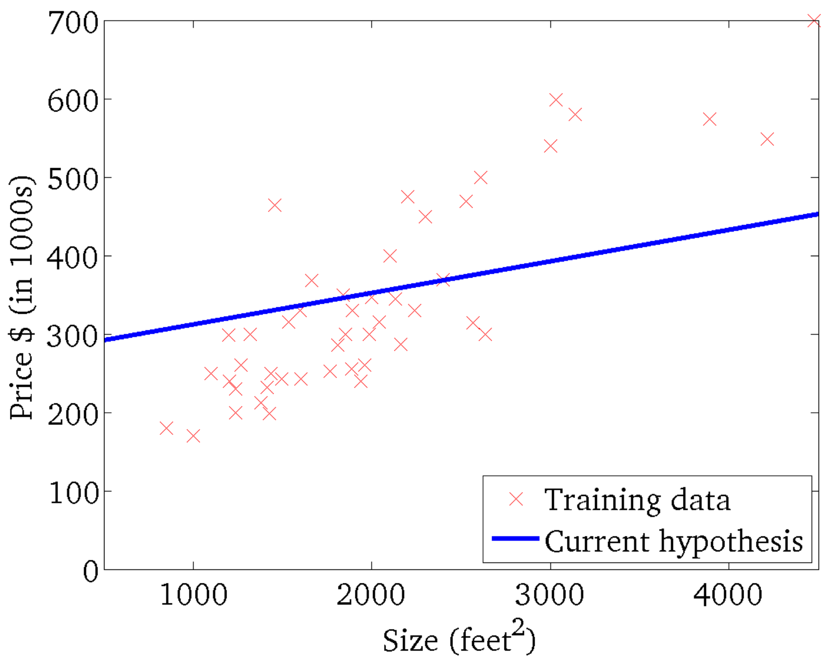


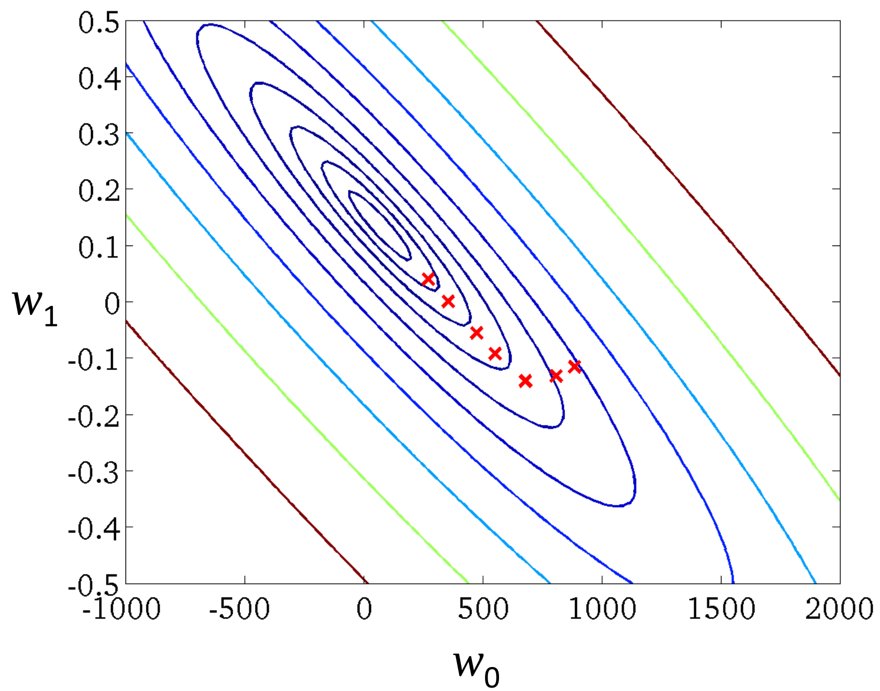
$\boldsymbol{f(x) = wx + b}$

(for fixed w, b this is a function of x)

$\boldsymbol{J(w, b)}$

(function of the parameters w, b)



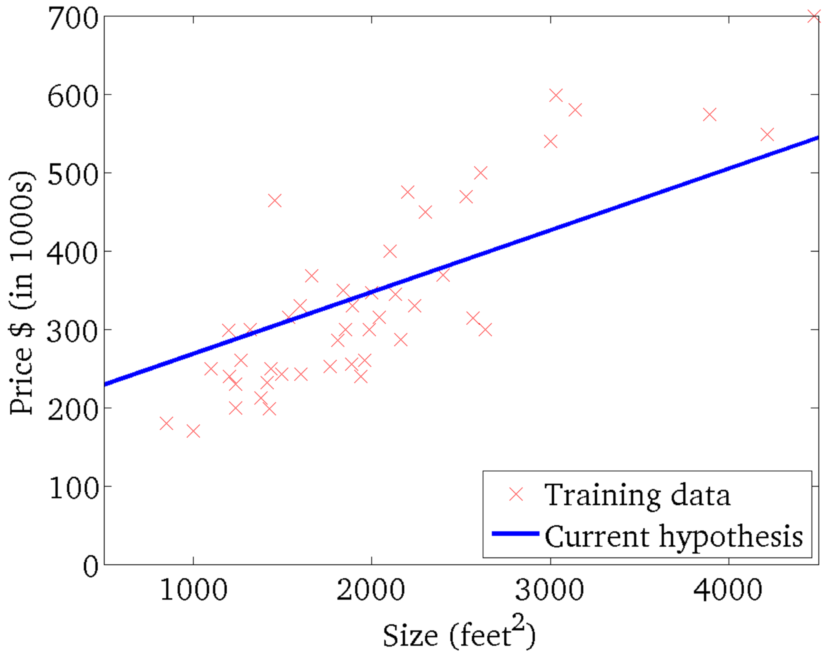


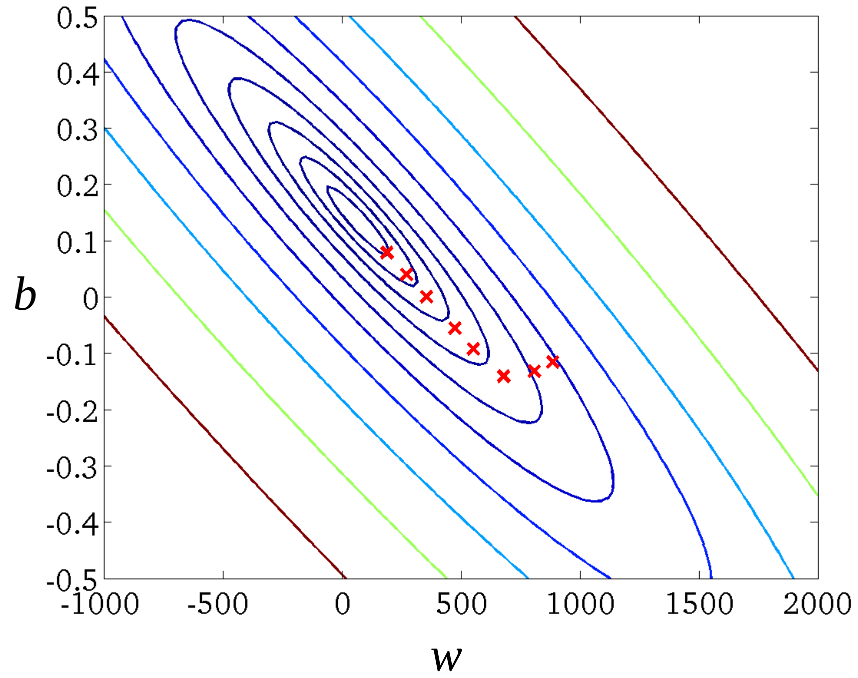
$$f(x) = wx + b$$

(for fixed w, b this is a function of x)

$$J(w, b)$$

(function of the parameters w, b)



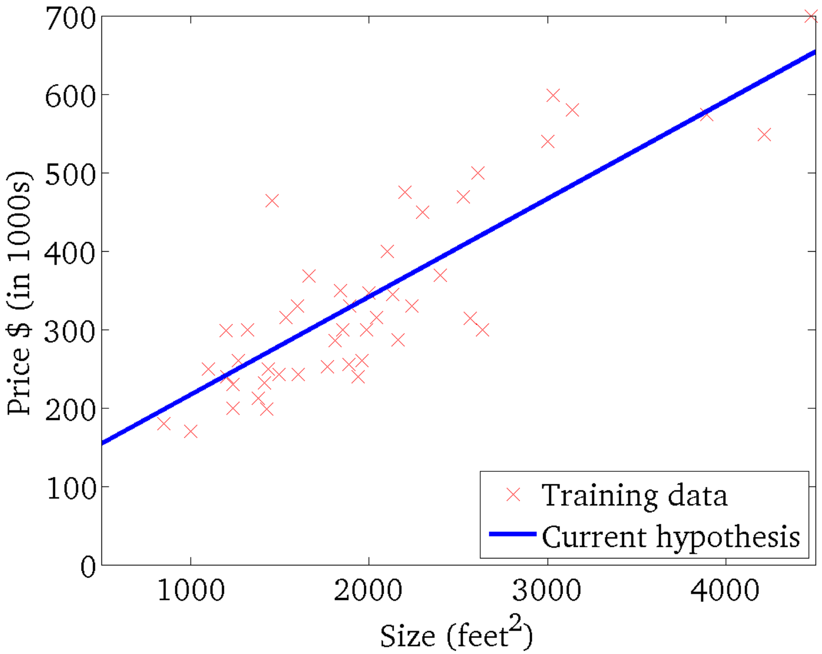


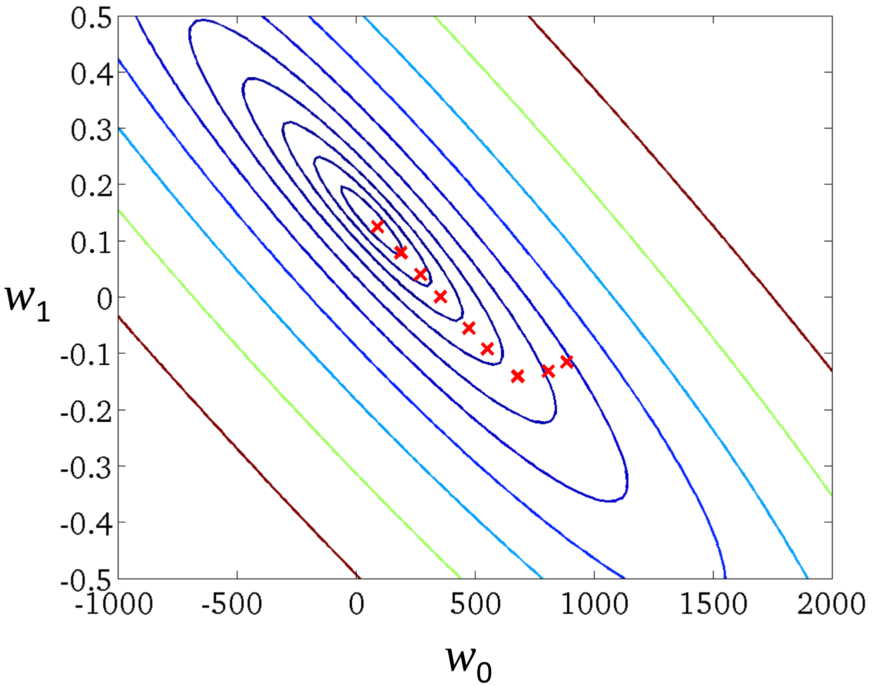
$$f(x) = wx + b$$

(for fixed w, b this is a function of x)

$$J(w, b)$$

(function of the parameters w, b)





Linear Regression - Gradient decent

- Gradient decent

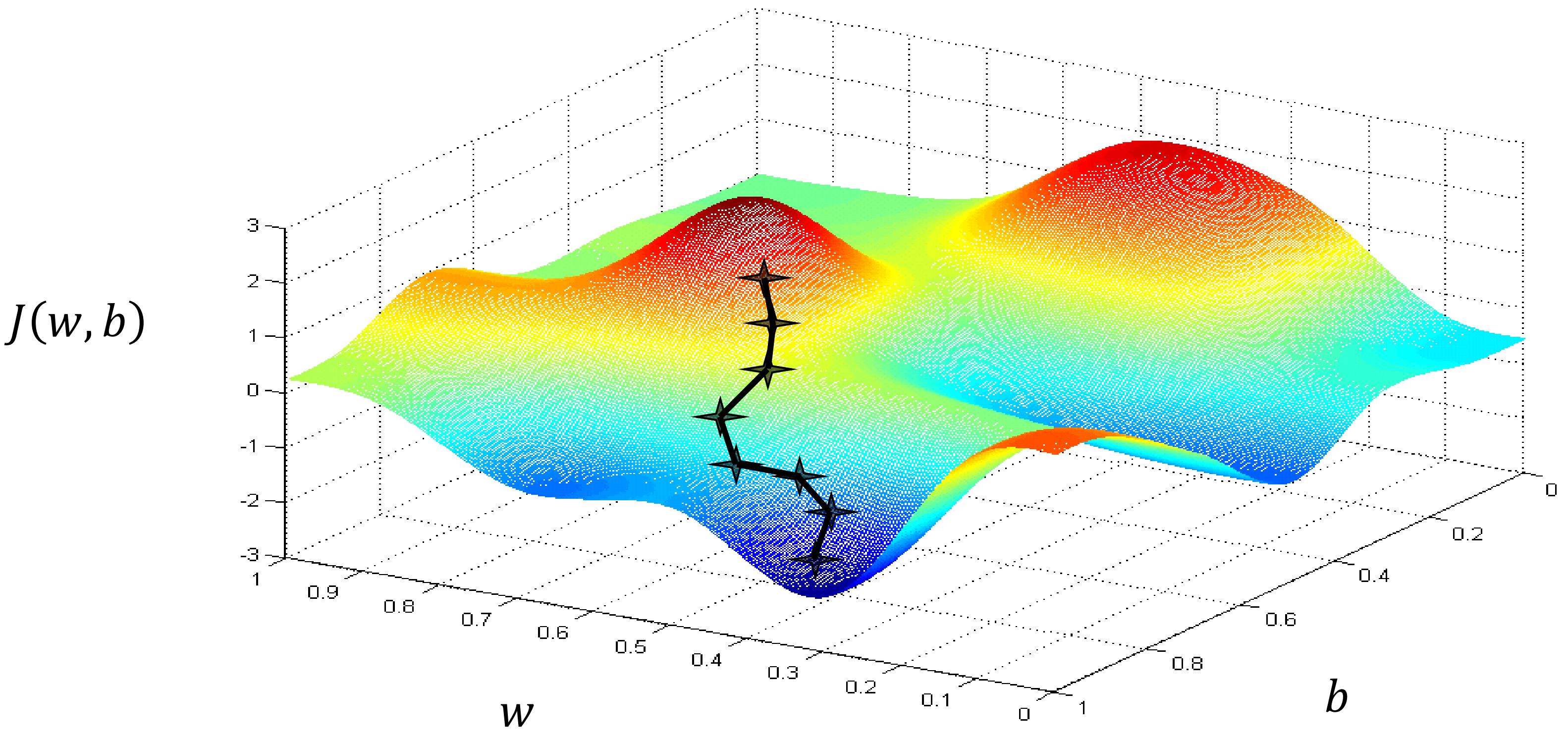
Have a cost function $\boldsymbol{J(w, b)}$

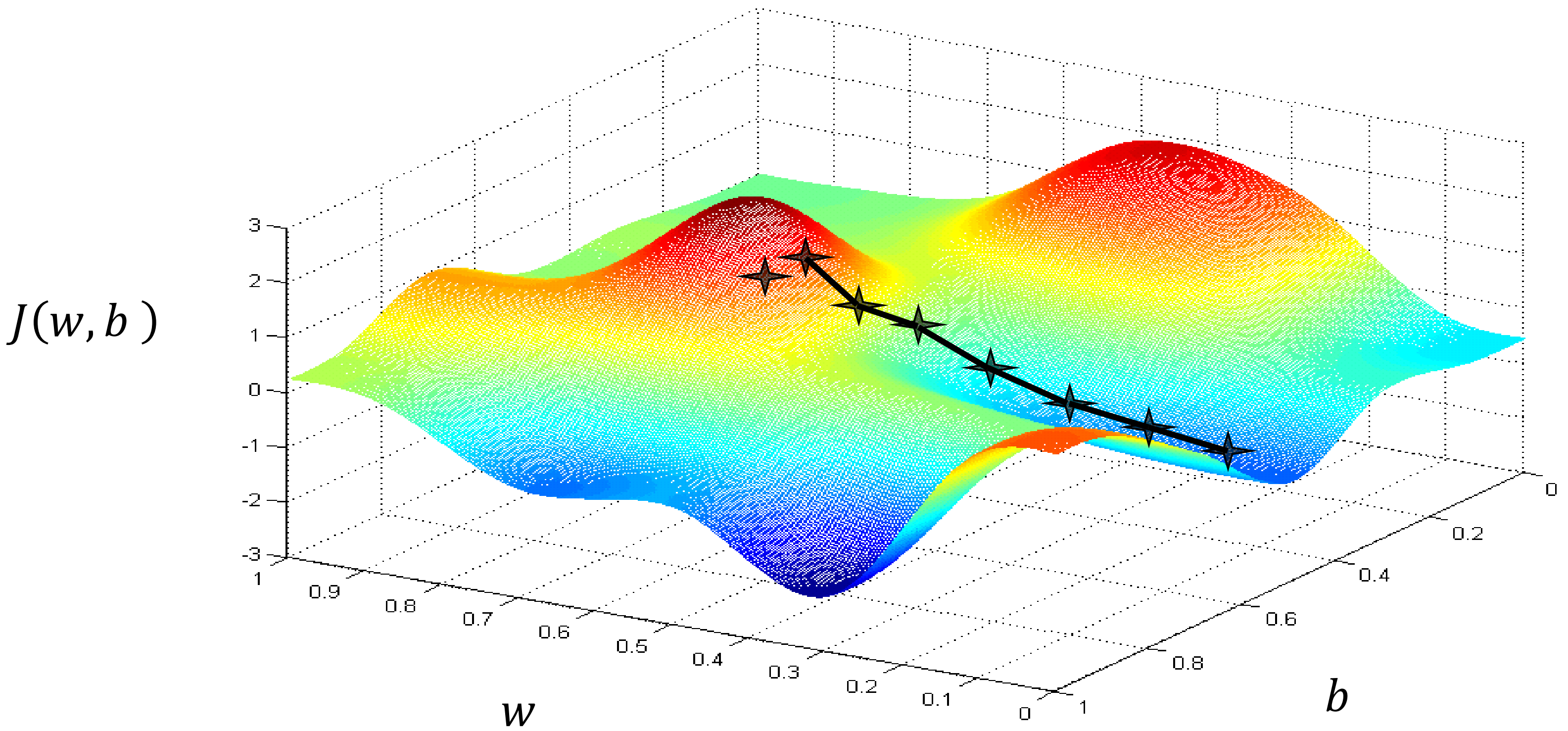
Want to find w and be $\min_{w,b} \boldsymbol{J(w, b)}$

Outline:

- Start with some $\boldsymbol{w, b}$ (say 0, 0)
- Keep changing $\boldsymbol{w, b}$ to reduce $\boldsymbol{J(w, b)}$

  until we hopefully end up at a minimum


$J(w,b)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
w
0
0.2
0.4
0.6
0.8
1
b



$J(w, b)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
w
0
0.2
0.4
0.6
0.8
1
b

Gradient descent algorithm

1. Initialize the values of ***w*** and ***b*** to some arbitrary values
2. Calculate the predicted values of *y* using the current values of ***w*** and ***b***
3. Calculate the gradients of the cost function with respect to ***w*** and ***b***
4. Update the values of ***w*** and ***b*** using the gradients and a learning rate
5. Repeat steps 2-4 until convergence (i.e., until the cost function converges to a minimum)

Gradient descent algorithm

Repeat until convergence

$$w = w - \alpha \boxed{\frac{\partial}{\partial w} J(w,b)}$$

Learning rate

Derivative

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

Simultaneously update w and b

Correct: Simultaneous update

$$tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

$$w = tmp\_w$$

$$b = tmp\_b$$

Incorrect

$$tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

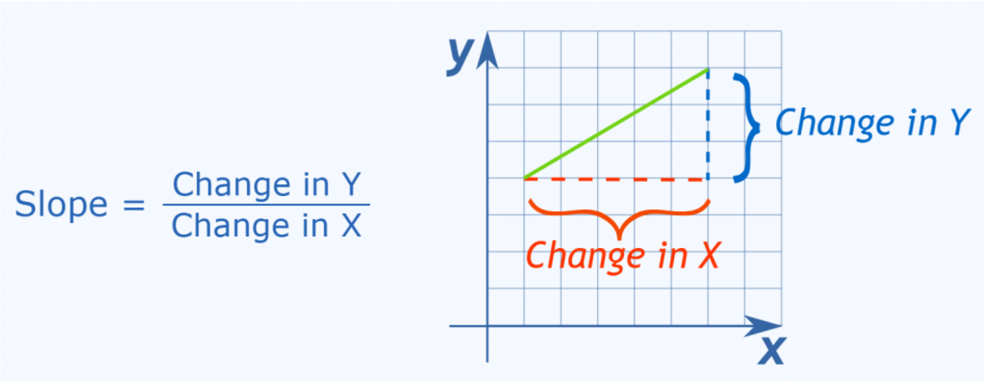
$$w = tmp\_w$$

$$tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

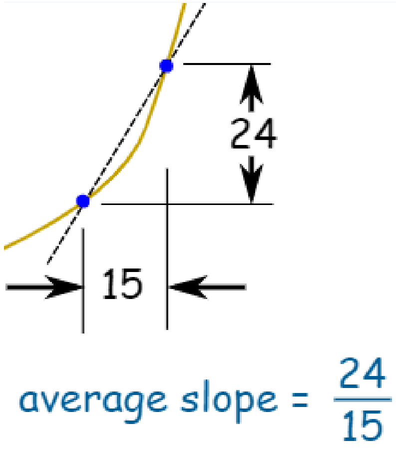
$$b = tmp\_b$$

Derivative 101

- Source https://www.mathsisfun.com/calculus/derivatives-introduction.html
- **Derivatives**: it is all about slope!



We can find
an **average** slope between
two points



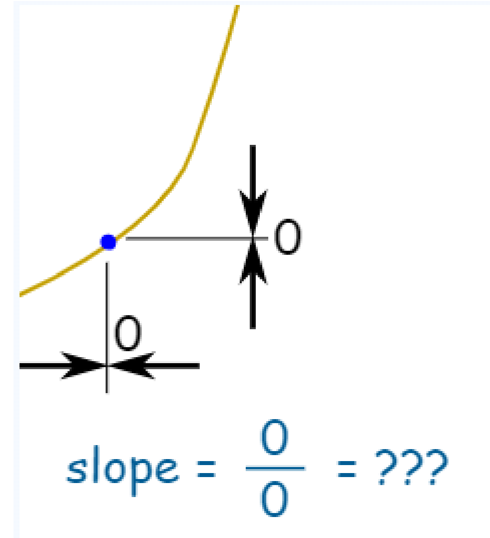
But how do we find the slope at a point?

- There is nothing to measure!

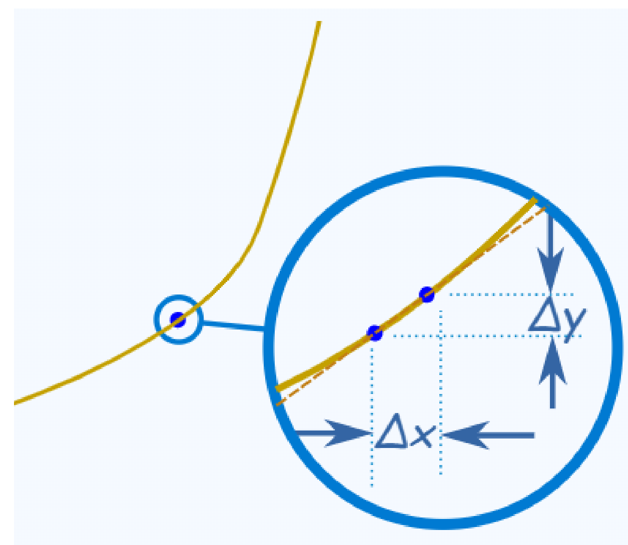
  slope 0/0 = ????

- But with derivatives we use a small difference ...

... then have it shrink towards zero



- Fill in this slope formula: $\dfrac{\Delta y}{\Delta x} = \dfrac{f(x+\Delta x) - f(x)}{\Delta x}$
- Simplify it as best we can
- Then make $\boldsymbol{\Delta x}$ shrink towards zero.



Derivative Example - $f(x) = x^2$

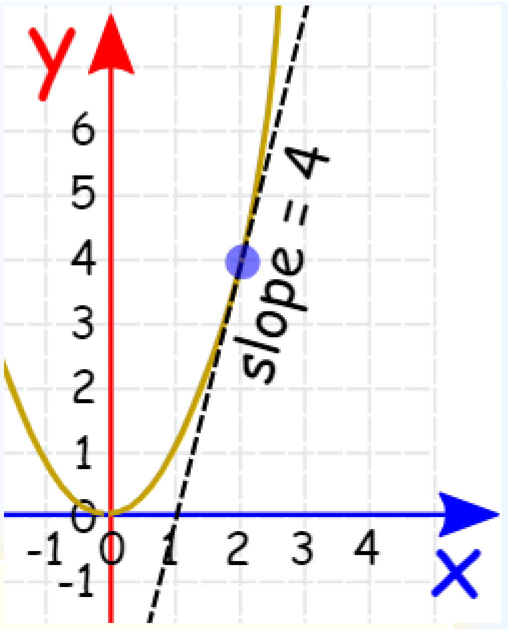
The slope formula is:	$\dfrac{f(x+\Delta x) - f(x)}{\Delta x}$
Use $f(x) = x^2$:	$\dfrac{(x+\Delta x)^2 - x^2}{\Delta x}$
Expand $(x+\Delta x)^2$ to $x^2+2x\,\Delta x+(\Delta x)^2$:	$\dfrac{x^2 + 2x\,\Delta x + (\Delta x)^2 - x^2}{\Delta x}$
Simplify (x^2 and $-x^2$ cancel):	$\dfrac{2x\,\Delta x + (\Delta x)^2}{\Delta x}$
Simplify more (divide through by Δx):	$2x + \Delta x$
Then, **as Δx heads towards 0** we get:	$2x$

Result: the derivative of $\mathbf{x^2}$ is **2x** $\quad \dfrac{d}{dx}x^2 = 2x$

- In other words, the slope at x is 2x

Interpretation of Derivative

- So what does $\frac{d}{dx}x^2 = 2x$ mean?
- It means that, for the function x^2, the slope or "rate of change" at any point is **2x**
- So when **x=2** the slope is **2x = 4**
- Or when **x=5** the slope is **2x = 10**, and so on



Gradient descent algorithm

repeat until convergence {

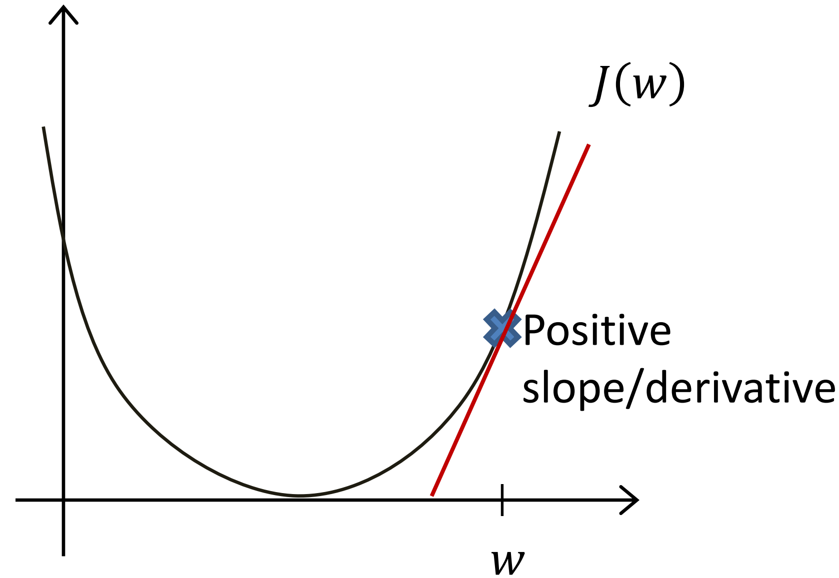
$$w = w - \alpha \frac{d}{dw} J(w)$$

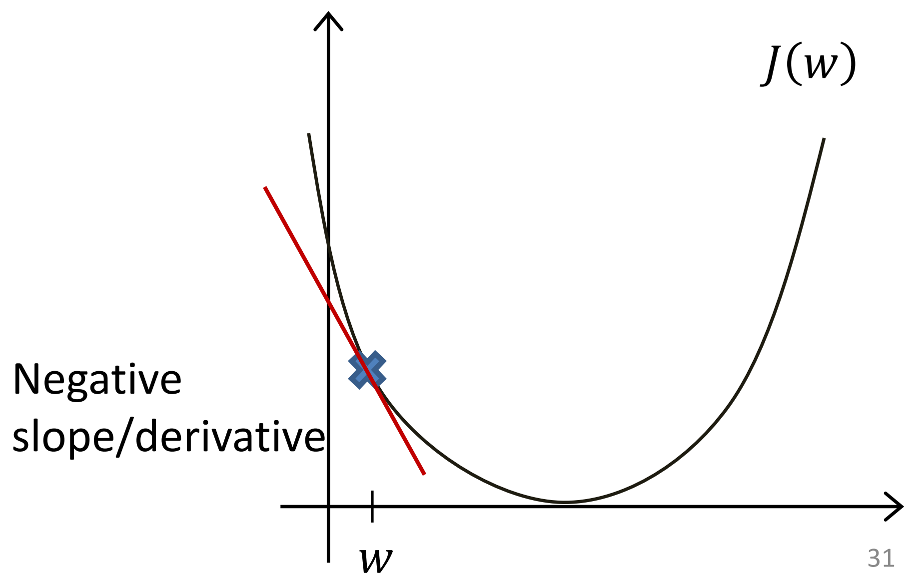
}

unsplash.com/photos/3m6vbzY69s4

Simplified

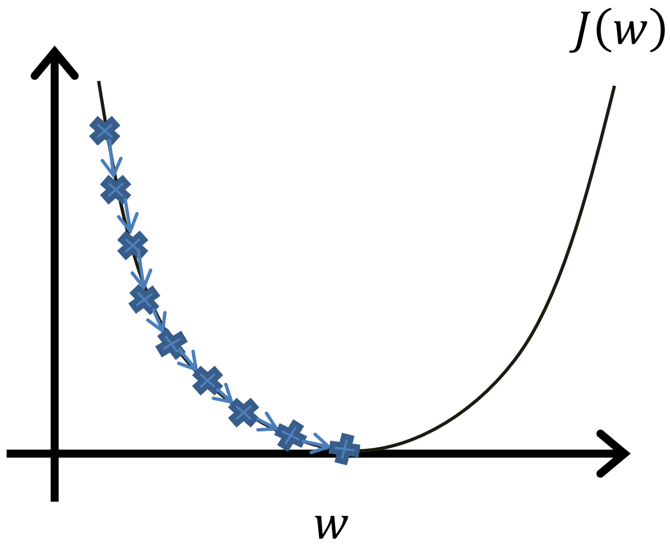
$$w = w - \alpha \frac{d}{dw} J(w)$$



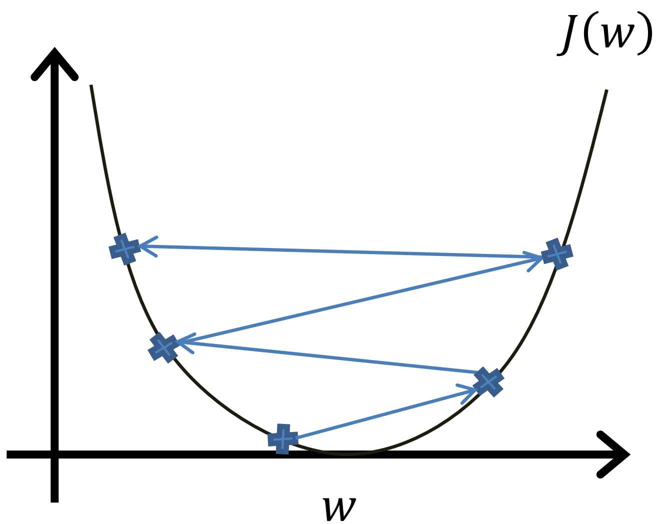


$$\mathrm{w} = w - \alpha \frac{d}{dw} J(w)$$

If α is too small, gradient descent can be slow



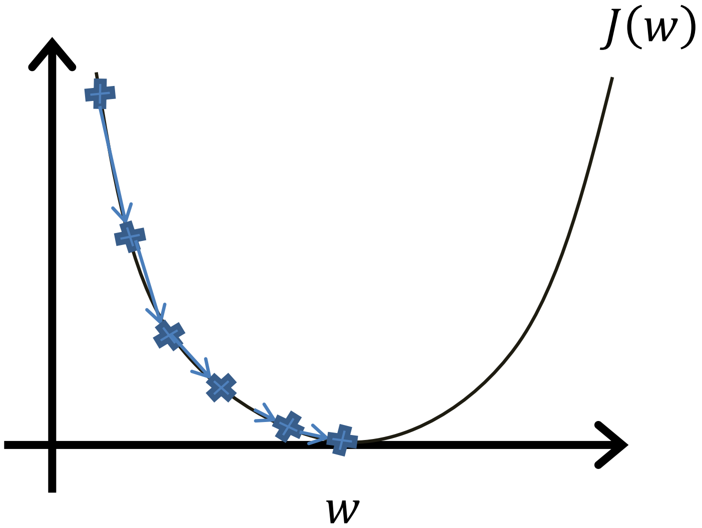
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge



Gradient descent can converge to a local minimum, even with the learning rate α fixed

$$w = w - \alpha \frac{d}{dw} J(w)$$

As we approach a local minimum, gradient descent will automatically take smaller steps (the slope gets smaller). So, no need to decrease α over time



Linear Regression with One Variable

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

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

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha\frac{\partial}{\partial w}J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)}$$

$$b = b - \alpha\frac{\partial}{\partial b}J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)$$

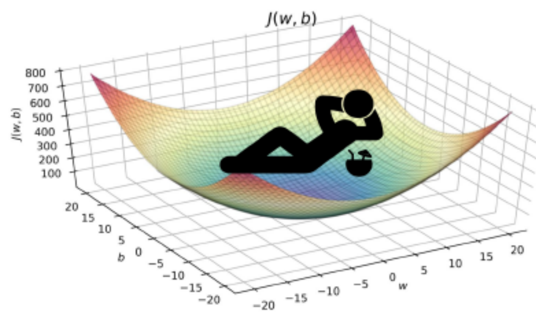
}

“Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples, m

	x size in feet²	y price in \$1000’s
(1)	2104	400
(2)	1416	232
(3)	1534	315
(4)	852	178
…	…	…
(47)	3210	870

$$\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$



Linear Regression with multiple variables

Multiple features (variables)

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features

x_j = j^{th} feature

$\vec{x}^{(i)}$ = features of i^{th} training example

$x_j^{(i)}$ = value of feature j in i^{th} training example

Hypothesis:

Previously: $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + \boldsymbol{b}$

Now: Multivariate linear regression.

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{w_1}\boldsymbol{x_1} + \boldsymbol{w_2}\boldsymbol{x_2} + \cdots + \boldsymbol{w_n}\boldsymbol{x_n} + \boldsymbol{b}$$

Example

$$f(x) = \underset{\text{size}}{0.1\,x_1} + \underset{\#\text{bedrooms}}{4\,x_2} + \underset{\#\text{floors}}{10\,x_3} + \underset{\text{years}}{-2\,x_4} + \underset{\text{base price}}{80}$$

Parameters: $w_0, w_1, \ldots, w_n$

Cost function:

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

Gradient descent:

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, \ldots, w_n)$$

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

Gradient Descent

Previously (n=1):

Repeat {

$$w = w - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right)}_{\frac{\partial}{\partial w}J(w)}$$

$$b = b - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) . \; x^{(i)}}_{\frac{\partial}{\partial b}J(w)}$$

}

(simultaneously update w_0 and w_1)

New algorithm $(n \geq 1)$:

Repeat {

$$w_j = w_j - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}}_{\frac{\partial}{\partial w_j}J(w)}$$

(simultaneously update w_j for $j = 0, \dots, n$)

}

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

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

$$w_2 = w_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_2^{(i)}$$

….

Formula to update **b** remains the same

Gradient descent in practice:

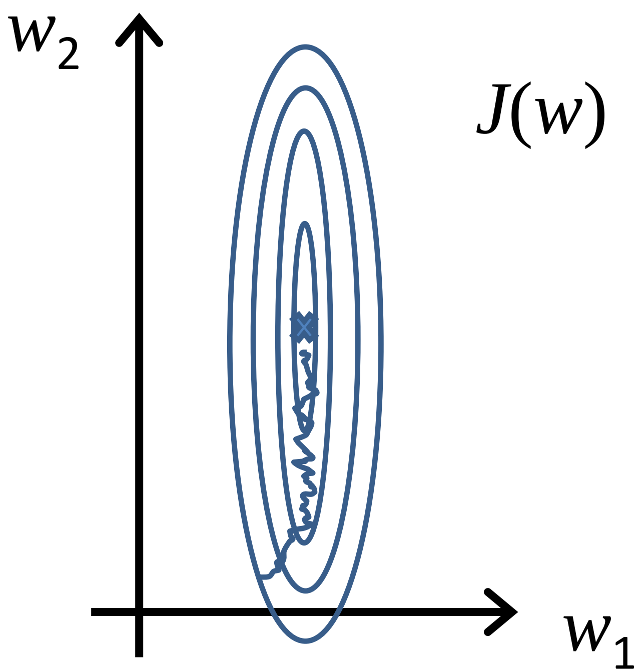
- Feature Scaling
- Regularization
- Regression Evaluation

Feature Scaling: divide the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1.

The idea: Make sure features are on a similar scale. So that the gradient descent converges faster.

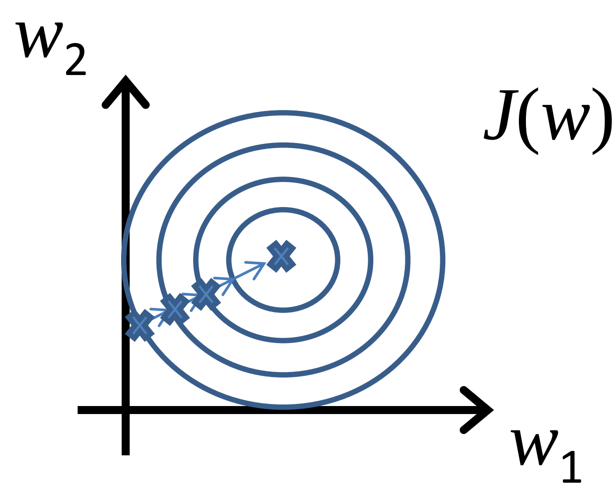
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Rule-of-thumb: Get every feature into approximately a $-1 \leq x_i \leq 1$ range, $-0.5 \leq x_i \leq 0.5$, or other similar small ranges.

Mean normalization

- Replace x_i to make features have approximately zero mean (Do not apply to x_0 = 1):

$$x_i := \frac{x_i - \mu_i}{s_i}$$

Where μ_i is the **average** of all the values for feature (i) (**<u>in the training set</u>**) and s_i is the range of values (max - min), or s_i is the standard deviation.

  – e.g.,

$x_1 = \frac{size - 1000}{2000}$ (average size of the houses is 1000, and ranges from 0 to 2000)

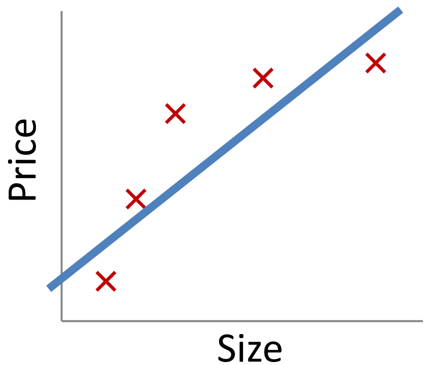
$x_2 = \frac{\#bedrooms - 2}{4}$ (average # of bedrooms is 2, and the range is from 1 to 5)

$$-0.5 \leq x_1 \geq 0.5, -0.5 \leq x_2 \geq 0.5,$$

Regularization

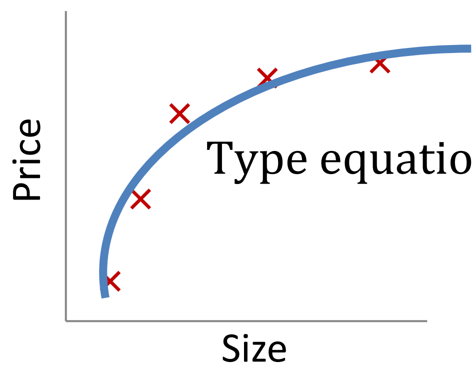
- The problem of overfitting

Example: Linear regression (housing prices)



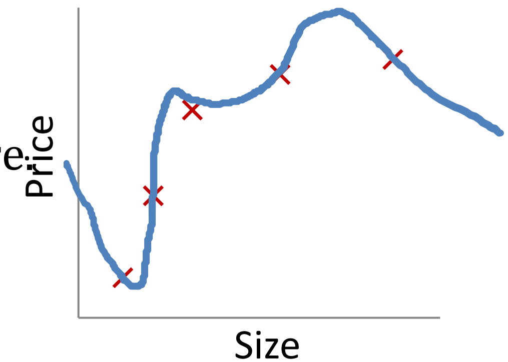
$w_0 + w_1 x$

"underfit/high bias"



$w_0 + w_1 x + w_2 x^2$

"just right"



$w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

"overfit/high variance"

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2 \approx 0$) but fail to generalize to new examples (predict prices on new examples).

Addressing overfitting:

$x_1 =$ size of house

$x_2 =$ no. of bedrooms

$x_3 =$ no. of floors

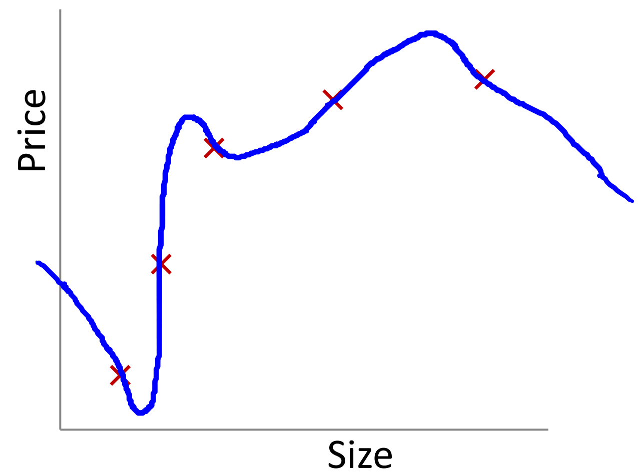
$x_4 =$ age of house

$x_5 =$ average income in neighborhood

$x_6 =$ kitchen size

$\vdots$

x_{100}



Addressing overfitting:

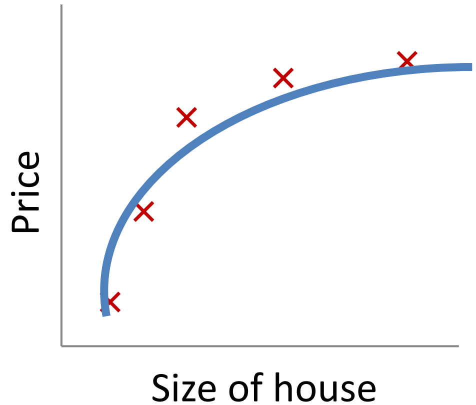
Options:

1. Reduce number of features.
   - Manually select which features to keep.
   - Use feature selection algorithm.
2. Regularization.
   - Keep all the features, but reduce magnitude/values of parameters w_j
   - Works well when we have a lot of features, each of which contributes a bit to predicting y .

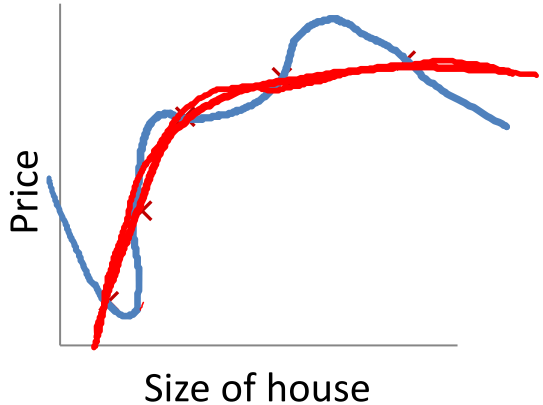
Regularization

- Cost function

Intuition



$$w_0 + w_1 x + w_2 x^2$$



$$w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$$

Suppose we penalize and make w_3, w_4 really small

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (f_w(x^{(i)}) - y^{(i)})^2 + 1000\, w_3^2 + 1000\, w_4^2$$

$$w_3 \approx 0, \quad w_4 \approx 0$$

Regularization

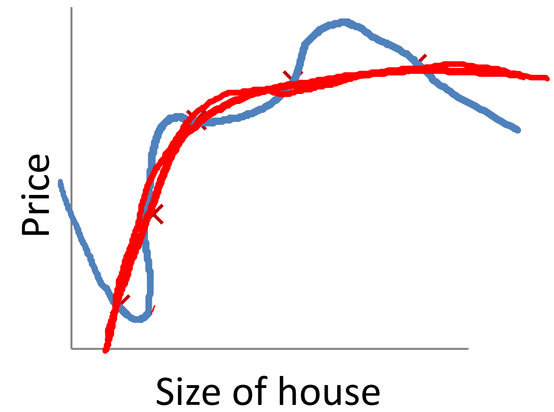
Small values for parameters $w_0, w_1, \dots, w_n$
- "Simpler/smoother" hypothesis
- Less prone to overfitting

Housing:
- Features: $x_1, x_2, \dots, x_{100}$
- Parameters: $w_0, w_1, w_2, \dots, w_{100}$

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

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n} w_j^2\right]$$

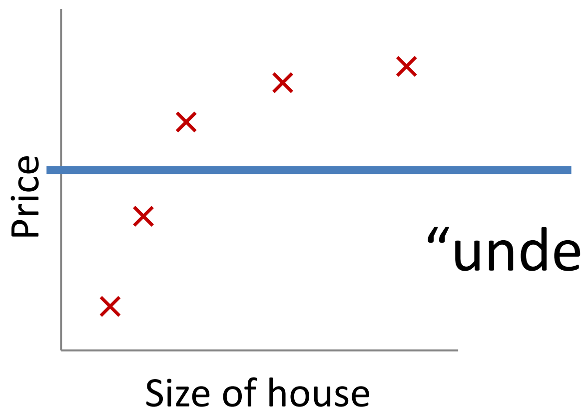
$$\min_w J(w)$$



In regularized linear regression, we choose w to minimize

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m}(f_w(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n} w_j^2\right]$$

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



"underfit"

$w_1 \approx 0,\quad w_2 \approx 0,\quad w_3 \approx 0,\quad w_4 \approx 0$

$w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

Regularized linear regression

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n} w_j^2\right]$$

$$\min_{w} J(w)$$

Gradient descent

Repeat $\{$

$$w_0 = w_0 - \alpha \overbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)}}^{\frac{\partial}{\partial w_0} J(w)}$$

$$w_j = w_j - \alpha \underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} - \frac{\lambda}{m} w_j\right]}_{\frac{\partial}{\partial w_j} J(w) \text{ “Regularized”}}$$

$\}$ $\quad (j = \cancel{0}, 1, 2, 3, \ldots, n)$

$$w_j := w_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(f_w\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

$$1 - \alpha\frac{\lambda}{m} < 1$$

Regression Evaluation

- Performance measured by
  - Mean Squared Error (MSE) $MSE = \frac{1}{n}\sum(y - \hat{y})^2$
  - Root-Mean-Squared-Error (RMSE) $RMSE = \sqrt{\frac{(y-\hat{y})^2}{n}}$
  - Mean-Absolute-Error (MAE) $MAE = \frac{1}{n}\sum|y - \widehat{y}|$
  - ...others