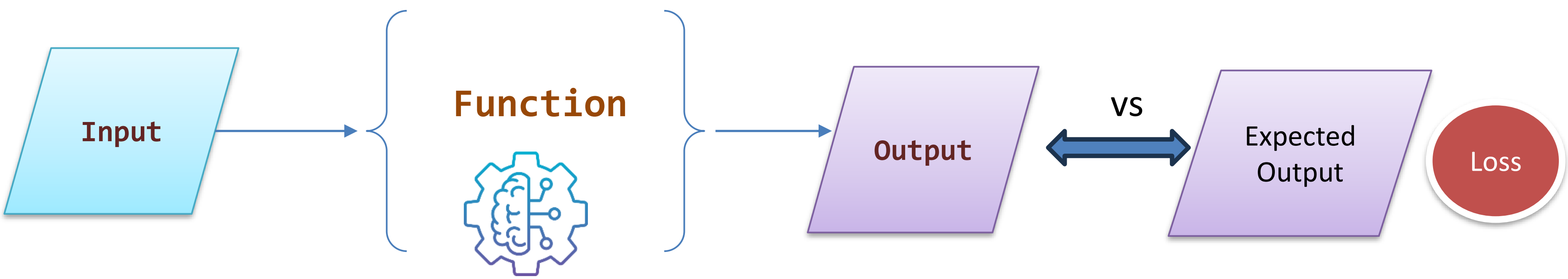
Regression

ML: learn a Function that minimizes the loss

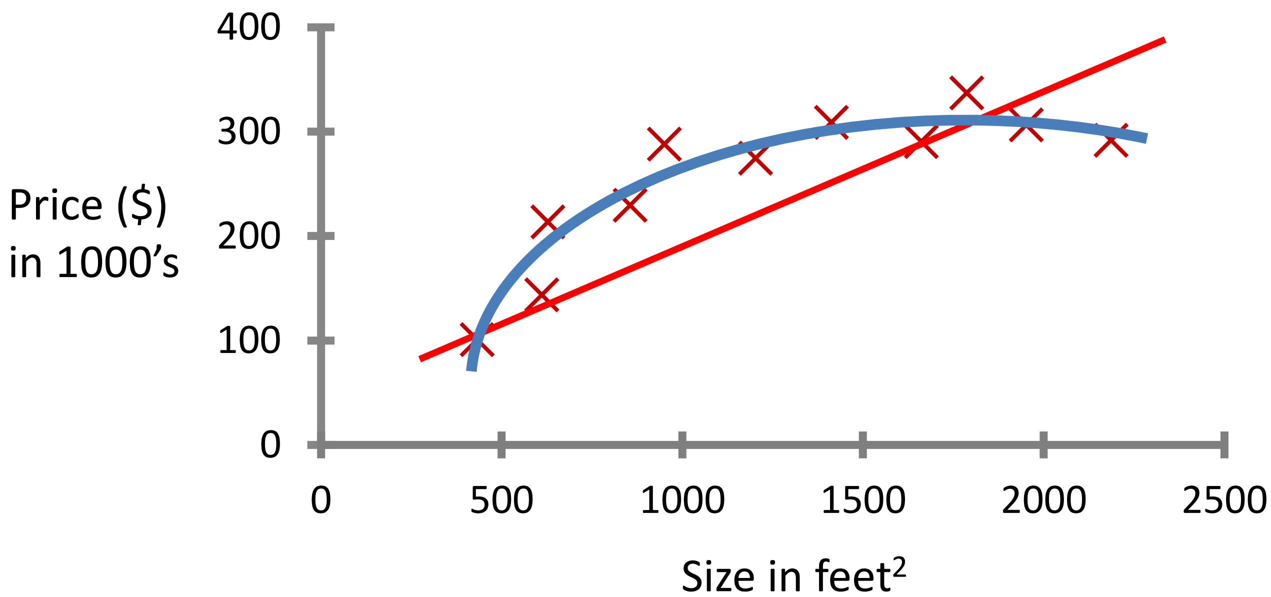
- Start with random function parameters
- Repeat intelligent guessing/approximation of the Function parameters such that the difference between the Predicted Output the Expected Output (i.e., Loss) is reduced



Linear Regression with One Variable

- Model Representation

Housing price prediction.



<u>Supervised Learning</u>

"right answers" given

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

Training set of housing prices

Size in feet^2 (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

$x^{(1)} = 2104$

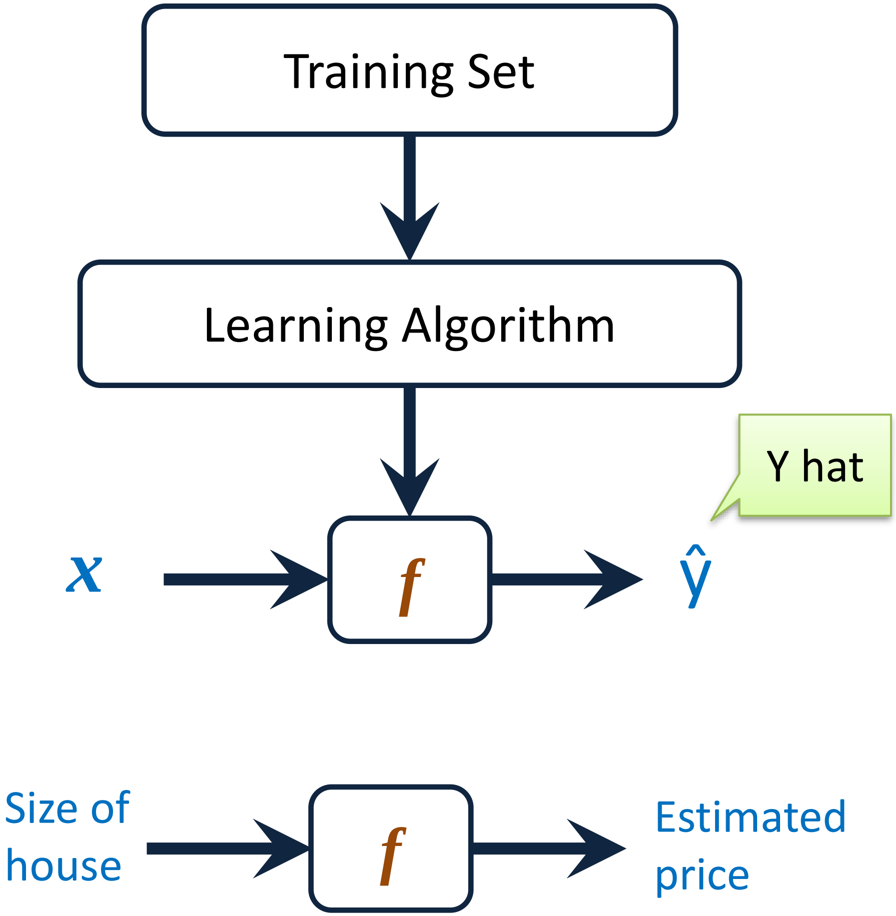
$x^{(2)} = 1416$

$y^{(1)} = 460$

(x, y) – single training example

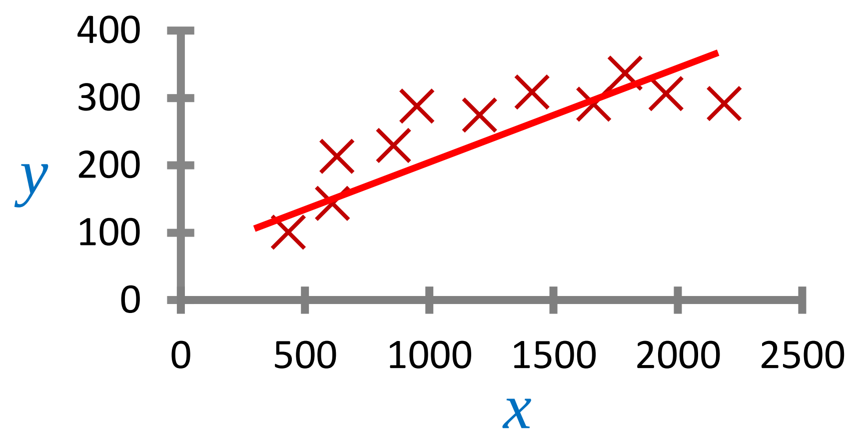
$(x^{(i)}, y^{(i)})$ – the i^{th} training example

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

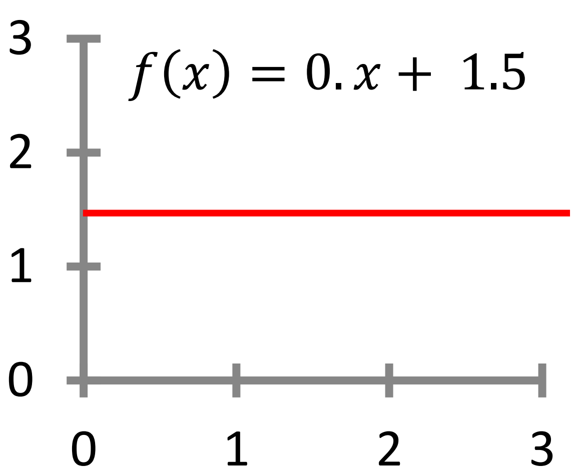
Univariate linear regression

Given a training set, learn a function f so that $f(x)$ is a “good” predictor for the corresponding value of y

Linear Regression with One Variable

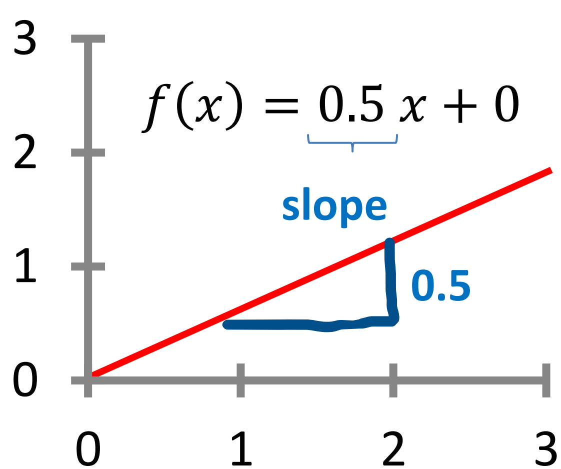
$$f(x) = wx + b$$

How to choose w and b ?



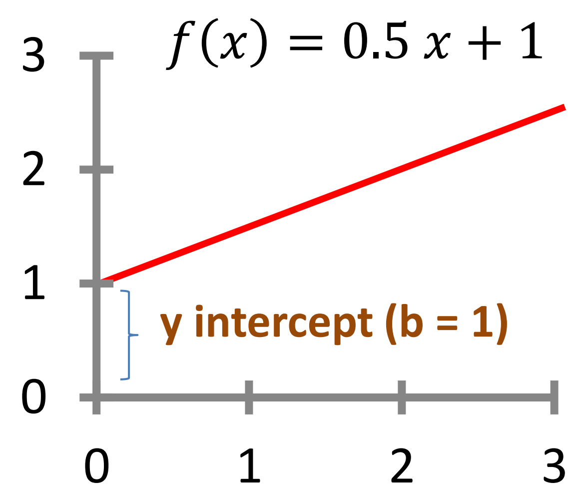
$w = 0$

$b = 1.5$



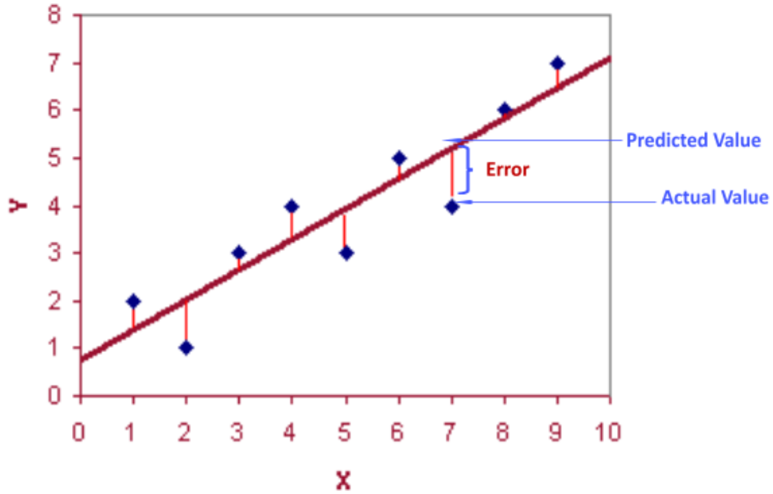
$w = 0.5$

$b = 0$



$w = 0.5$

$b = 1$



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:

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

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

With **m** = number of training examples

Simplified

Function:

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

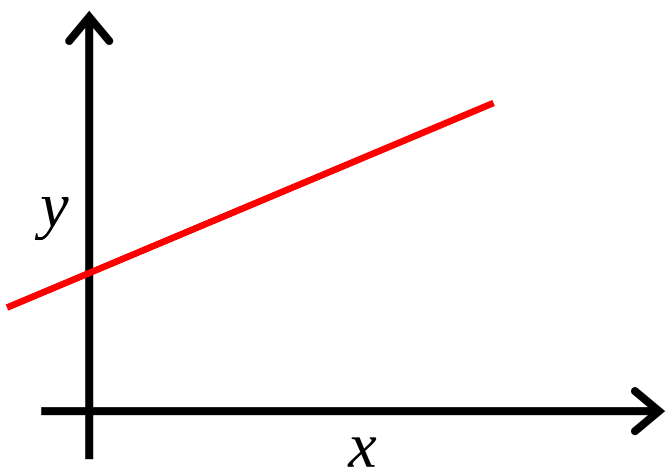
Parameters:

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

Cost Function:

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

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

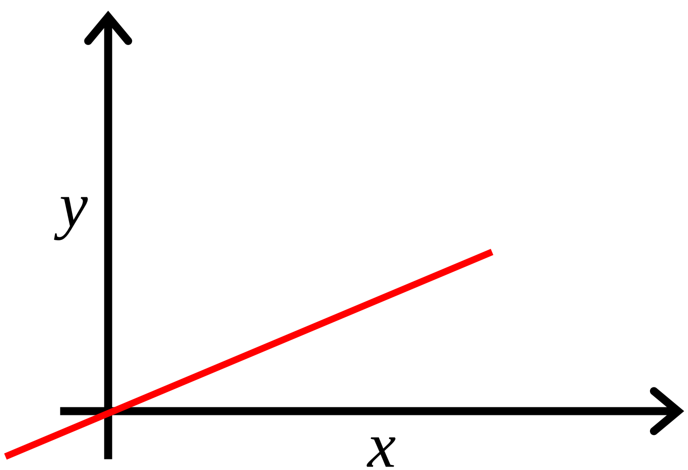


$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x}$$

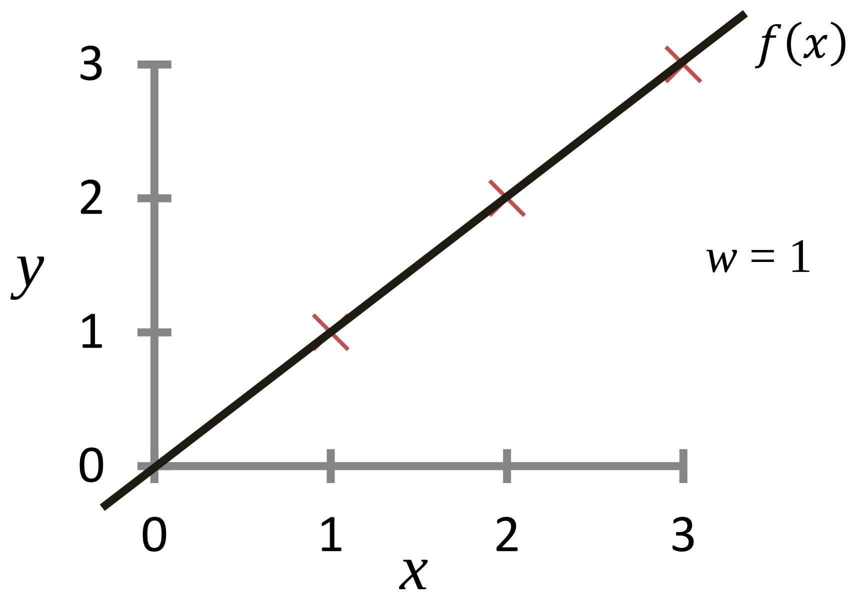
$\boldsymbol{w}$

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

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



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

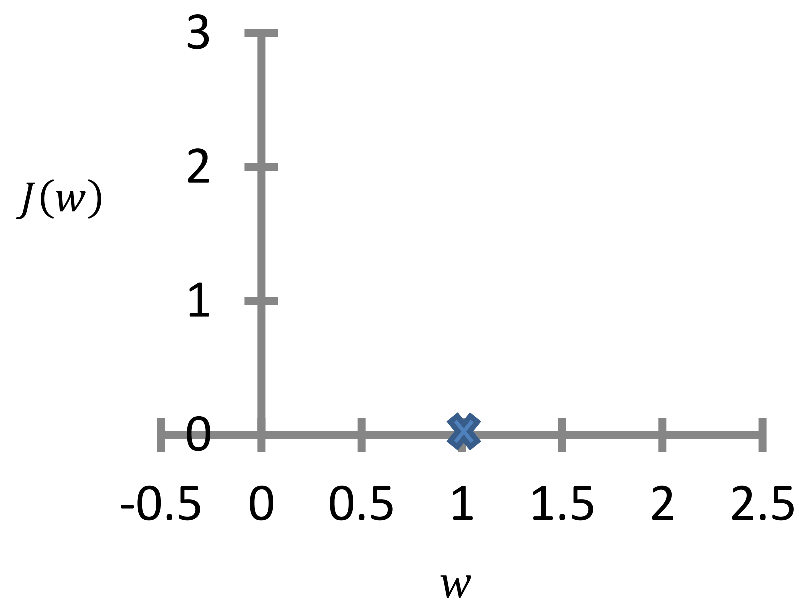


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2$$

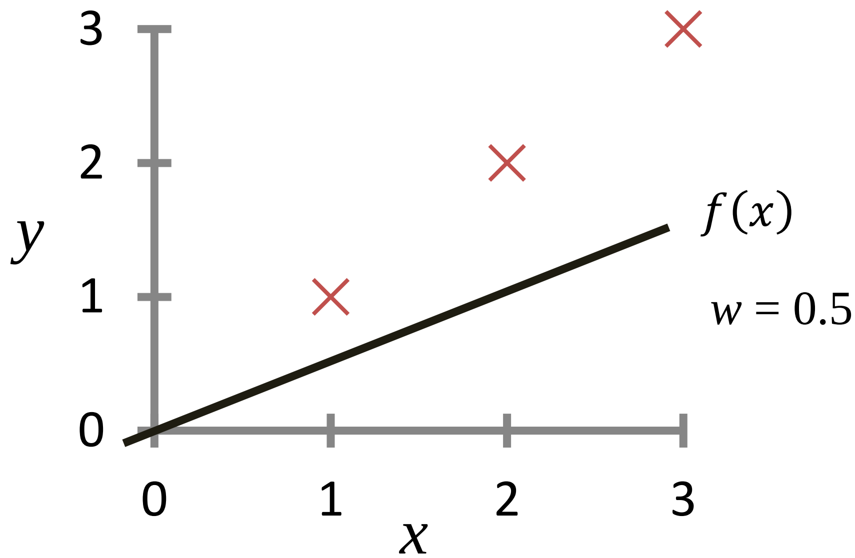
$$= \frac{1}{2m}(0^2 + 0^2 + 0^2) = 0$$

$\boldsymbol{J(w)}$

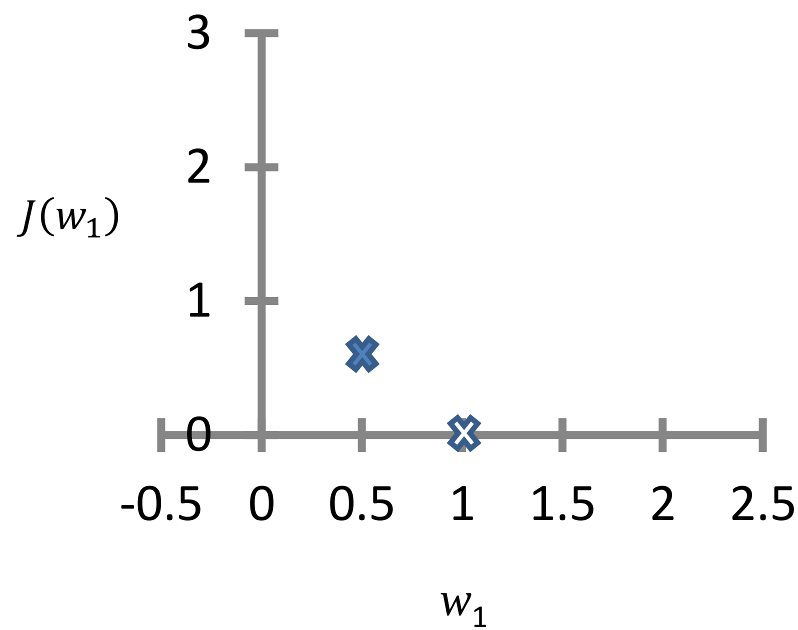
(function of the parameter w)



$f(x) = wx$



$J(w)$

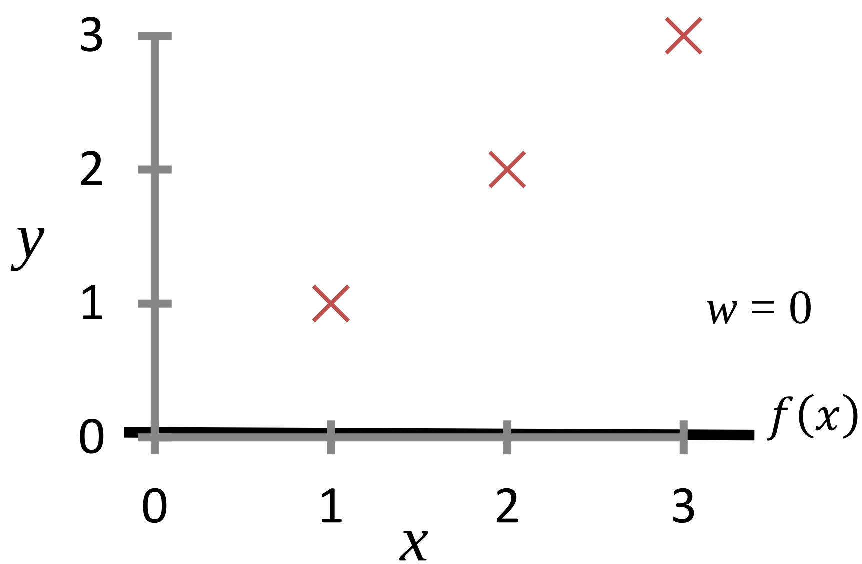


$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2$$

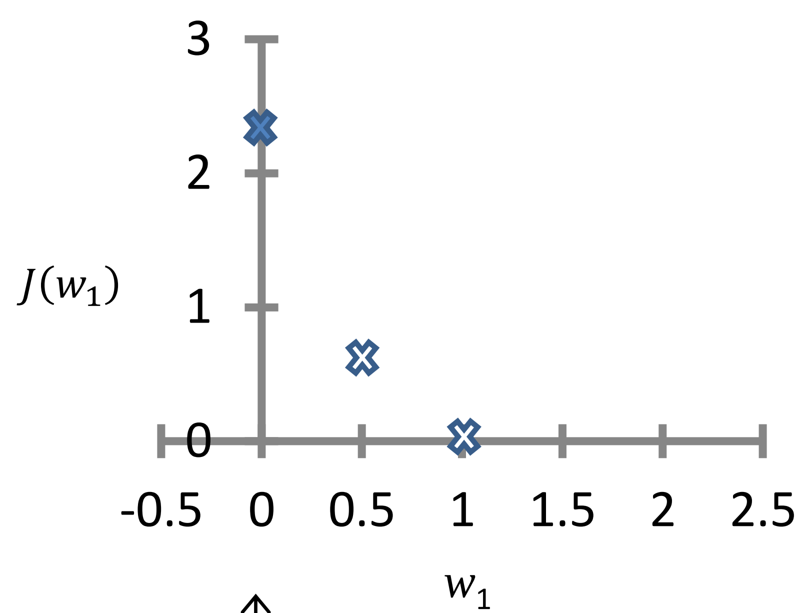
$$= \frac{1}{2m}\left((0.5 - 1)^2 + (1\text{-}2)^2 + (1.5\text{-}3)^2\right)$$

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

$f(x) = wx$



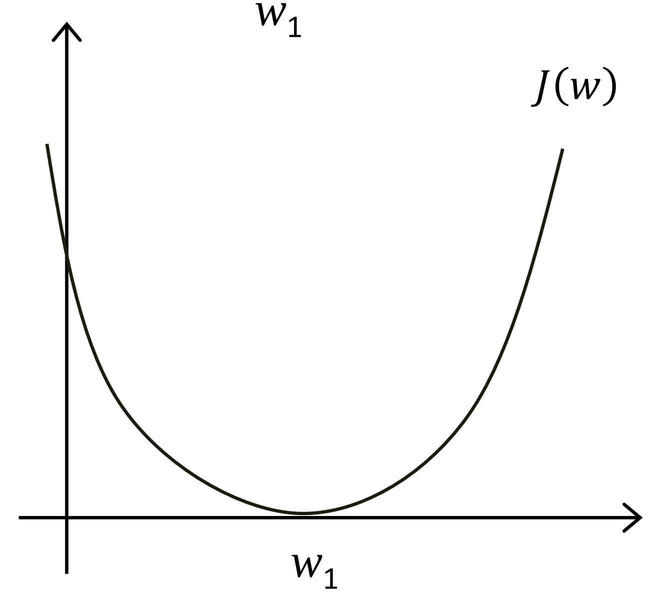
$J(w)$



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

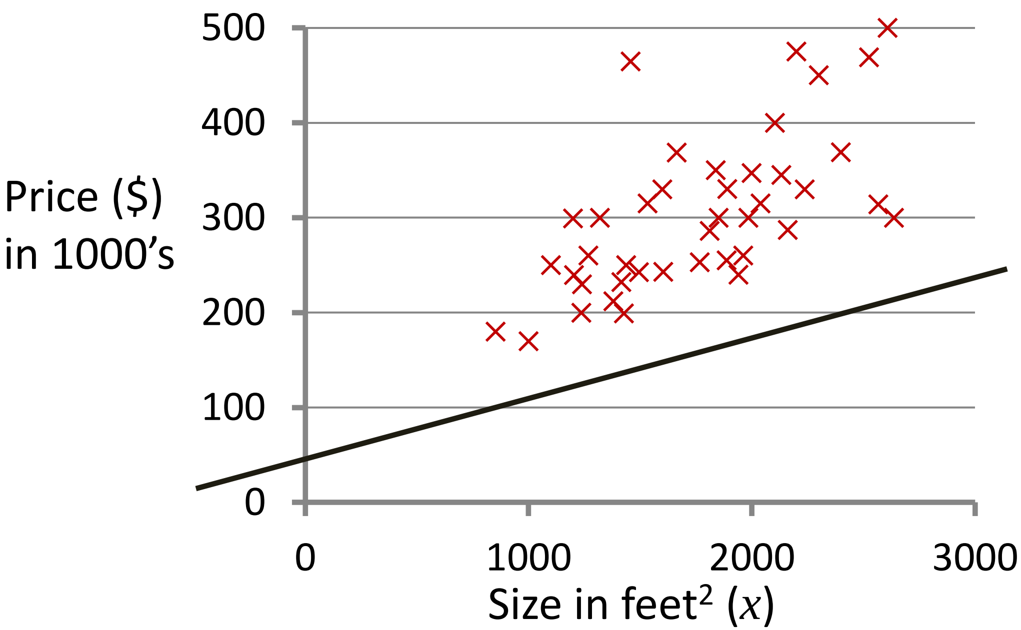
$$= \frac{1}{2m}(1^2 + 2^2 + 3^2)$$

$$= \frac{1}{2\times 3}(14) = \frac{14}{6} = 2.3$$



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

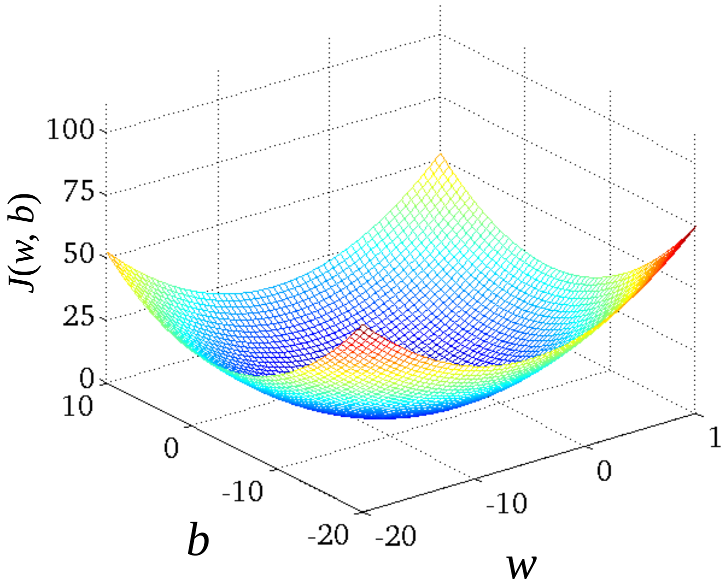
(for fixed w, b this is a function of x)



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

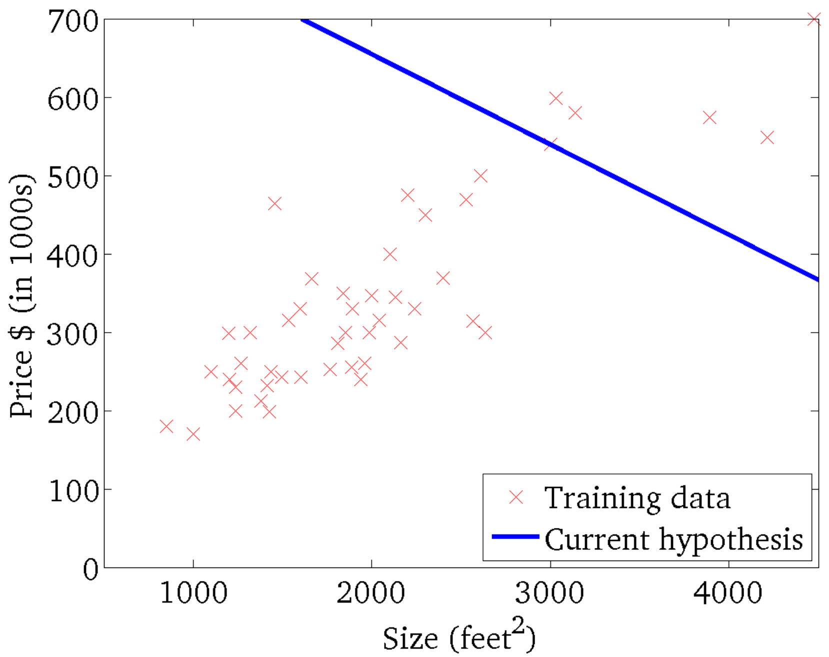
$$J(w, b)$$

(function of the parameters w, b)



$f(x) = wx + b$

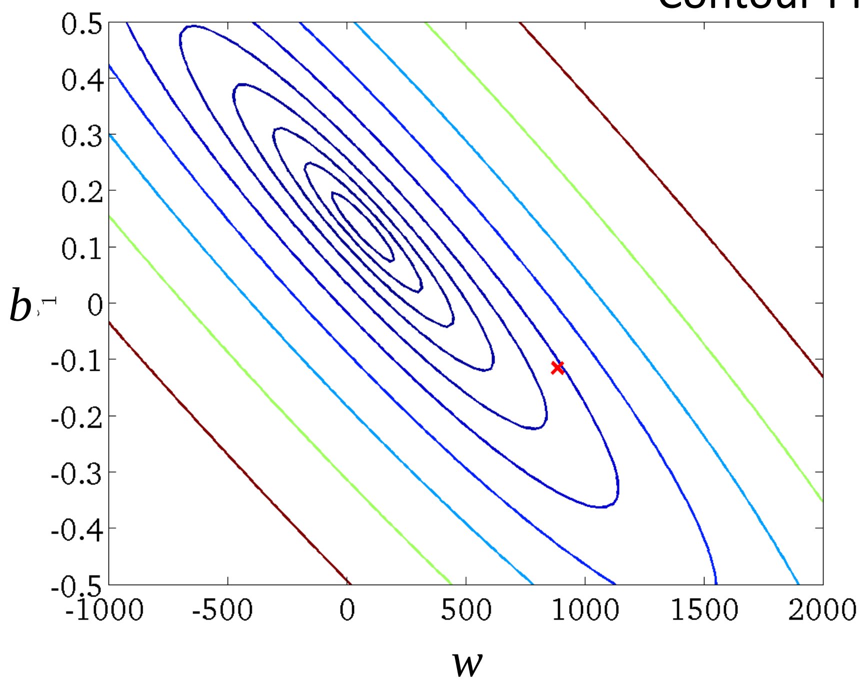
(for fixed w, b this is a function of x)

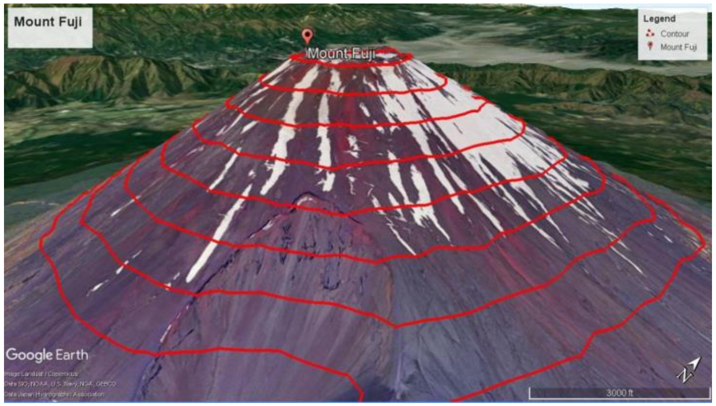


$J(w, b)$

(function of the parameters w, b)

Contour Plot



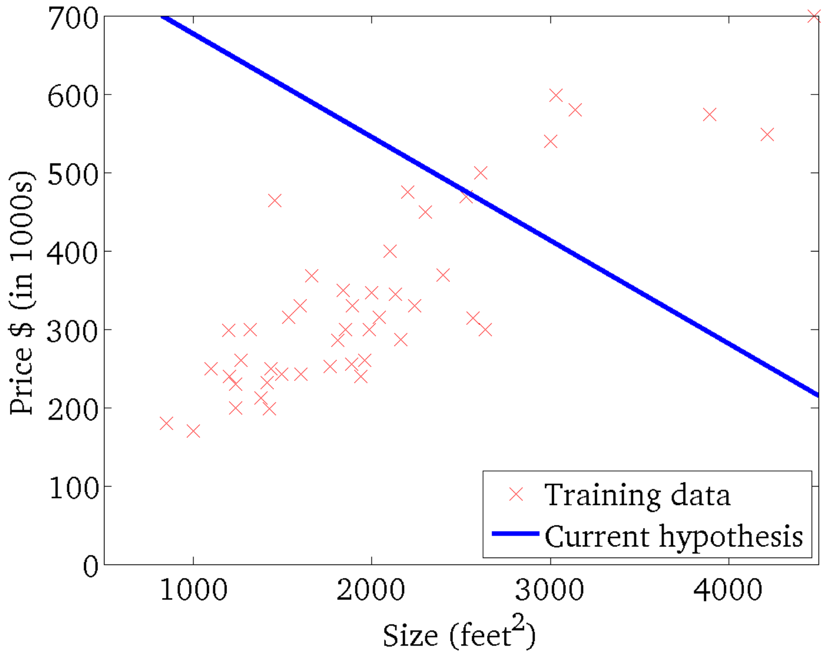


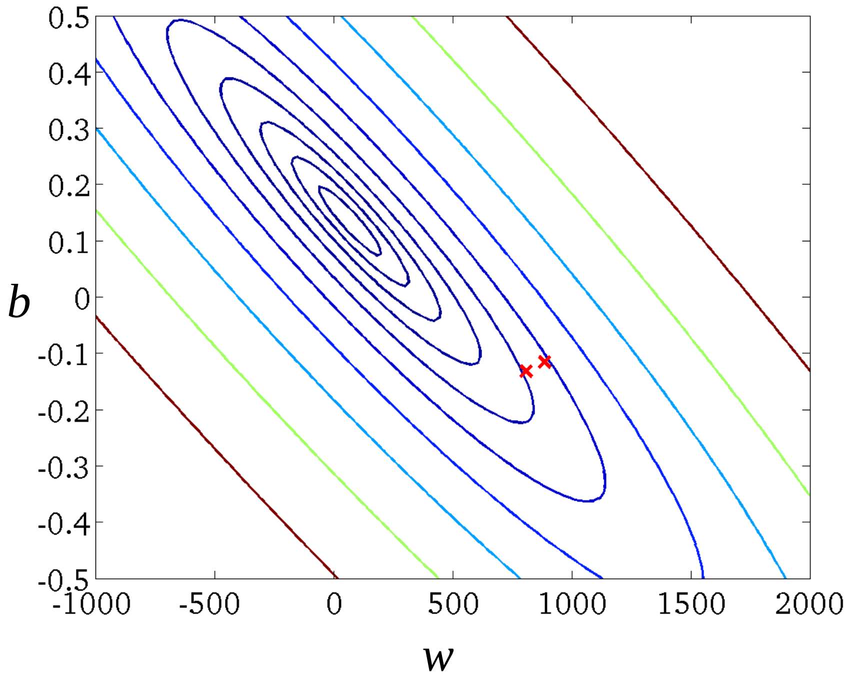
$$f(x) = wx + b$$

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

$$J(w, b)$$

(function of the parameters w, b)



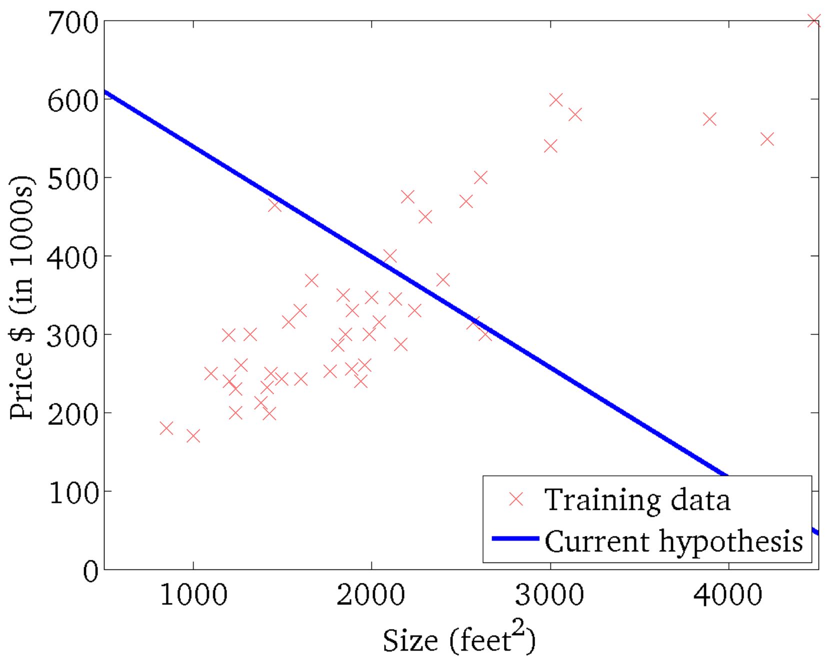


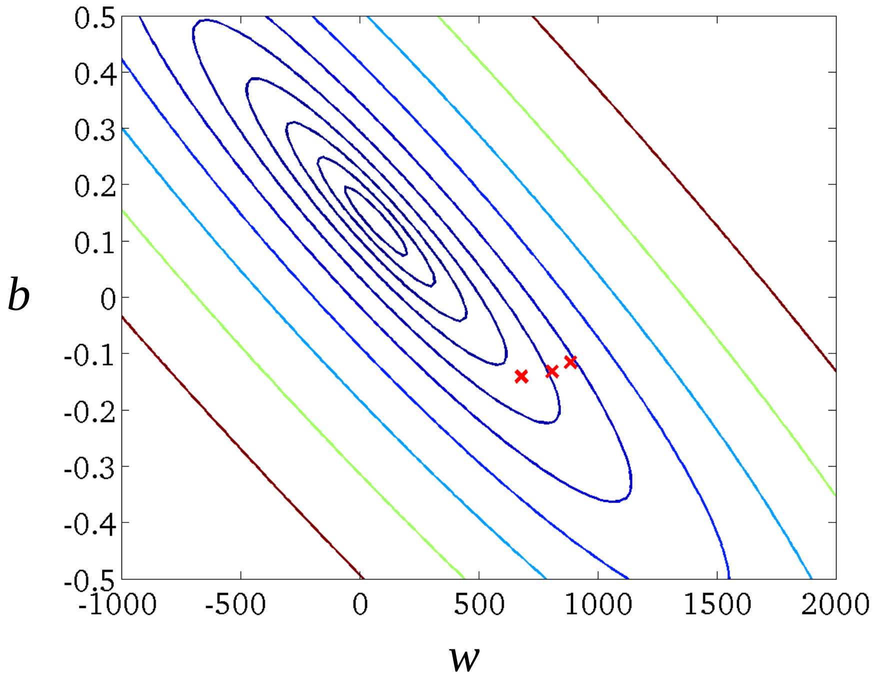
$$f(x) = wx + b$$

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

$$J(w, b)$$

(function of the parameters w, b)



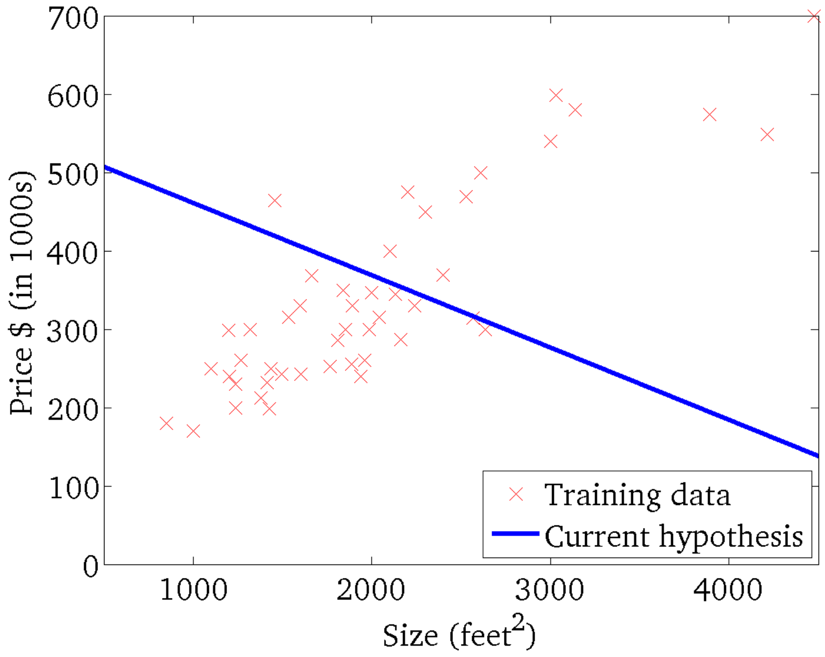


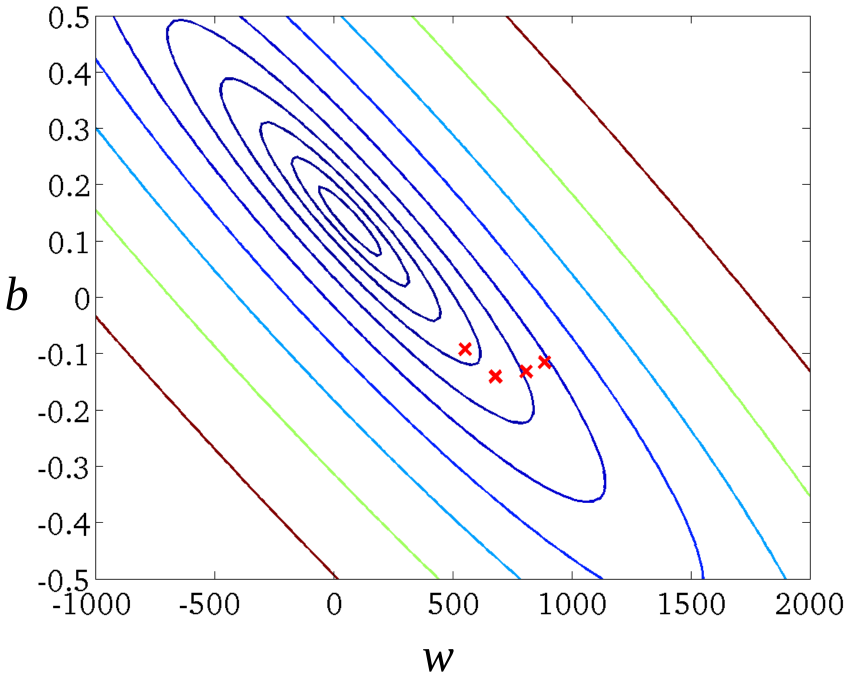
$$f(x) = wx + b$$

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

$$J(w, b)$$

(function of the parameters w, b)



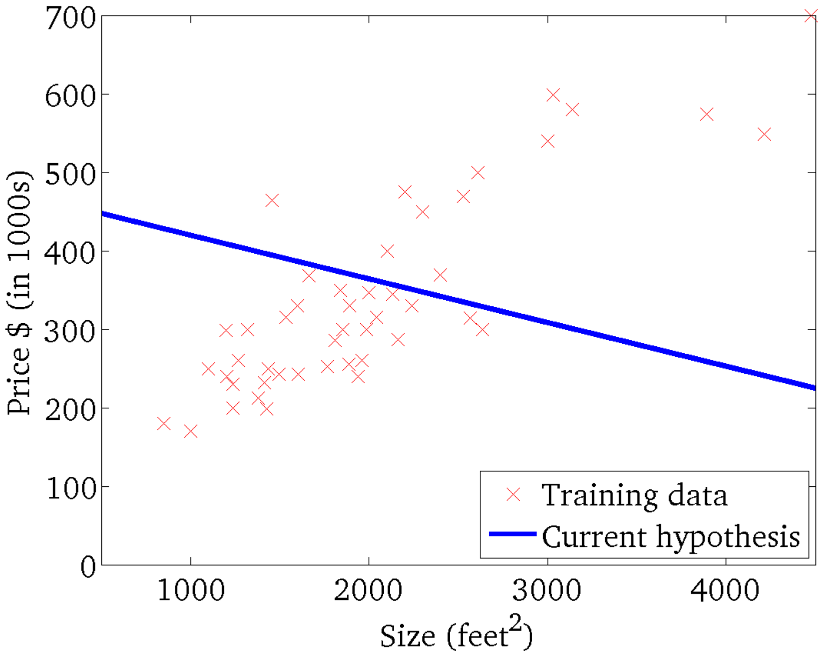


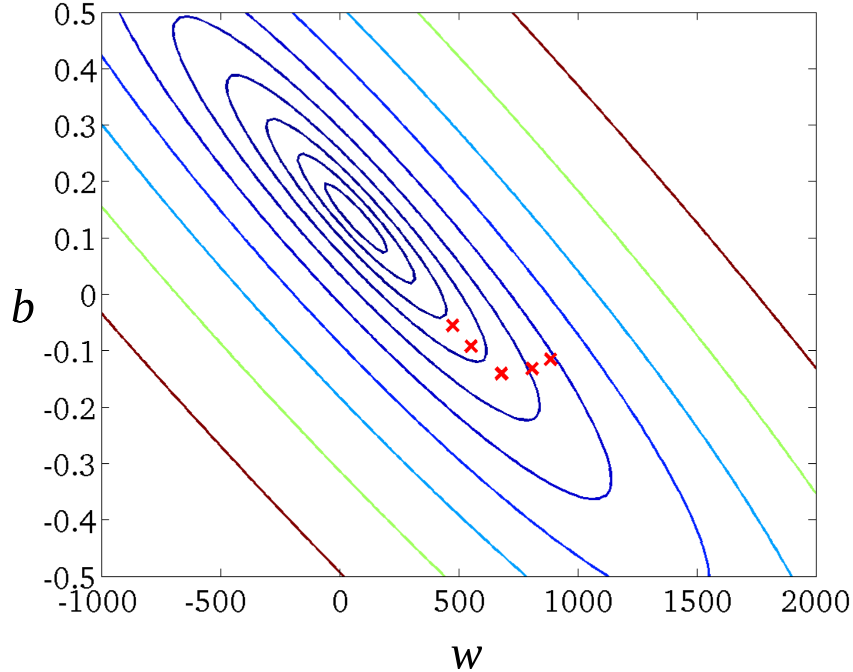
$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + \boldsymbol{b}$$

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

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

(function of the parameters w, b)



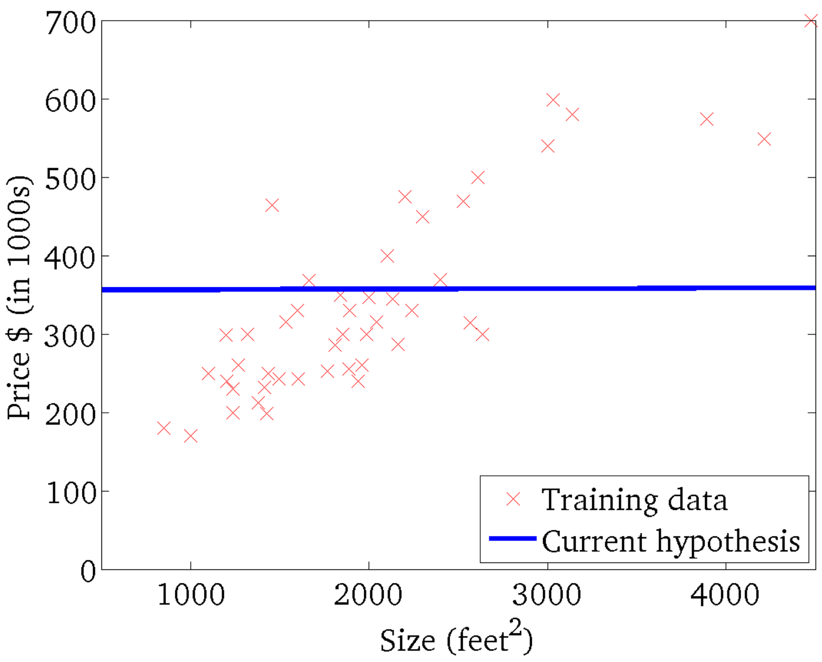


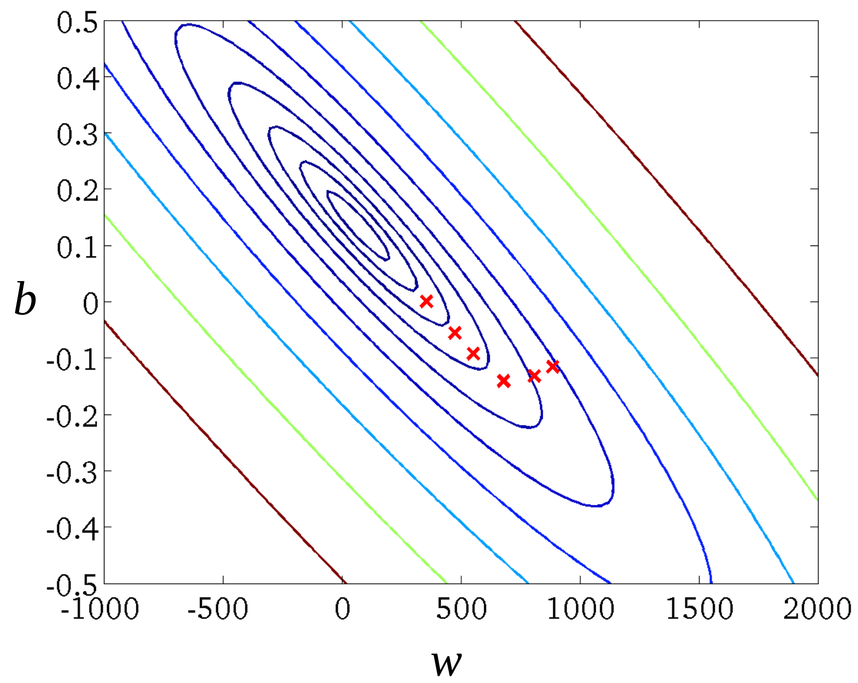
$$f(x) = wx + b$$

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

$$J(w, b)$$

(function of the parameters w, b)



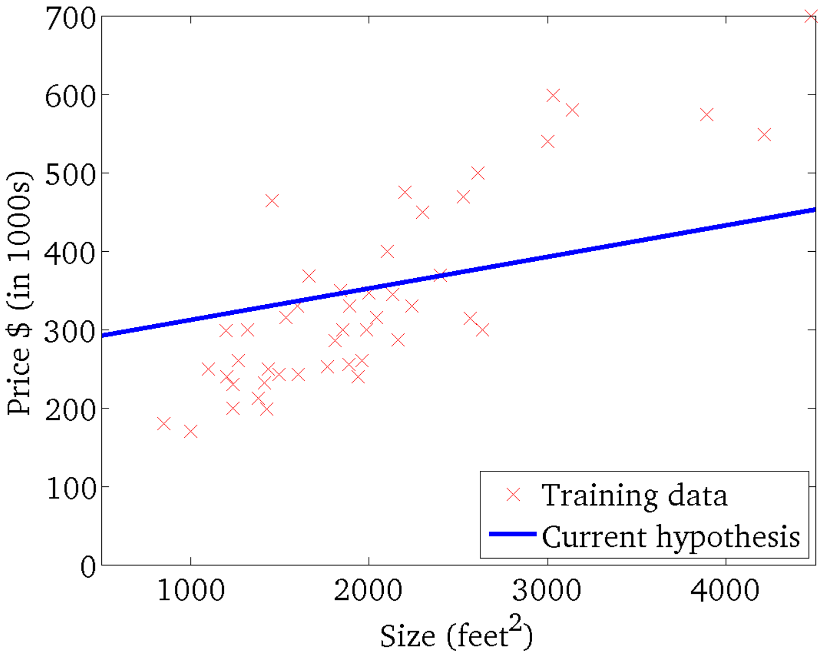


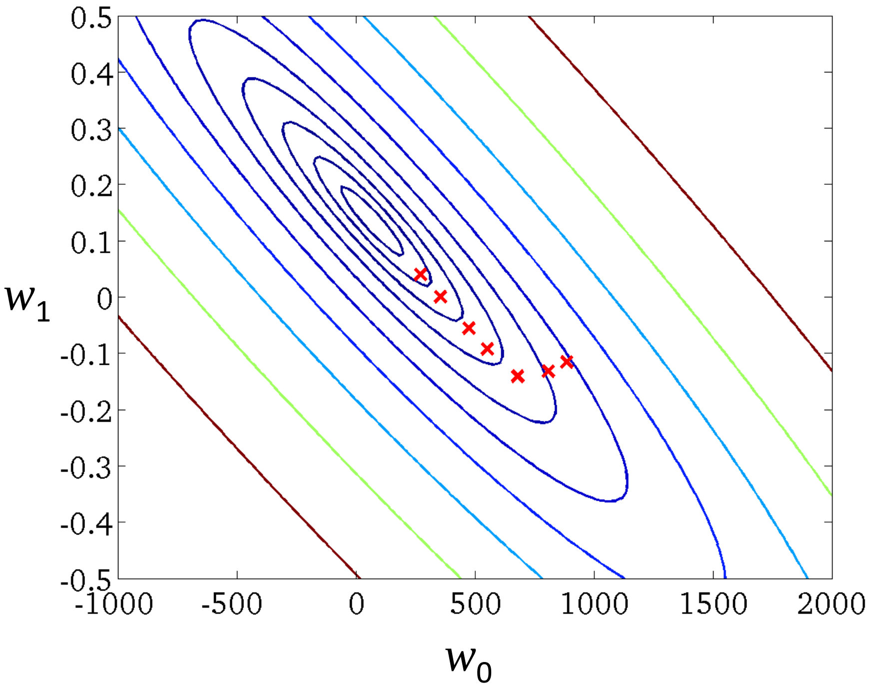
$$f(x) = wx + b$$

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

$$J(w, b)$$

(function of the parameters w, b)



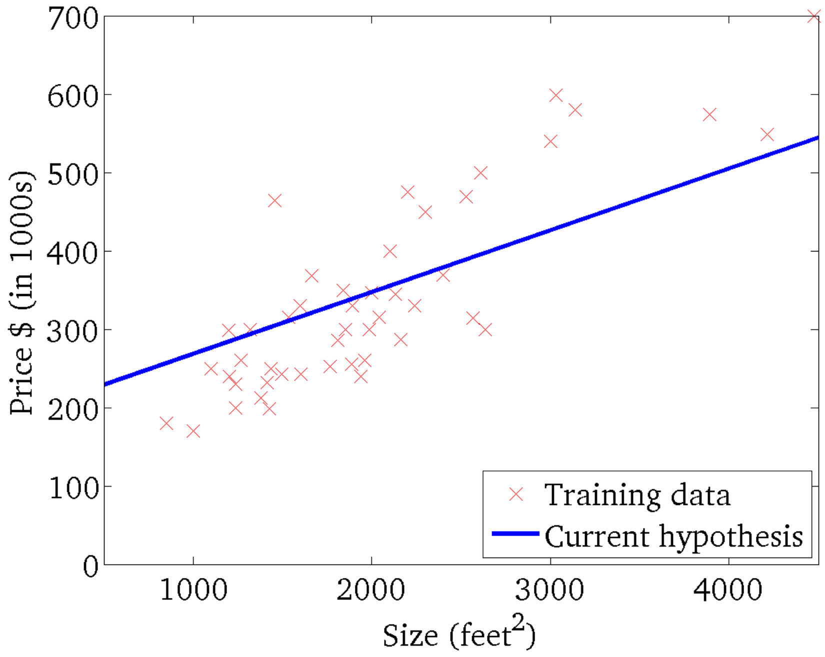


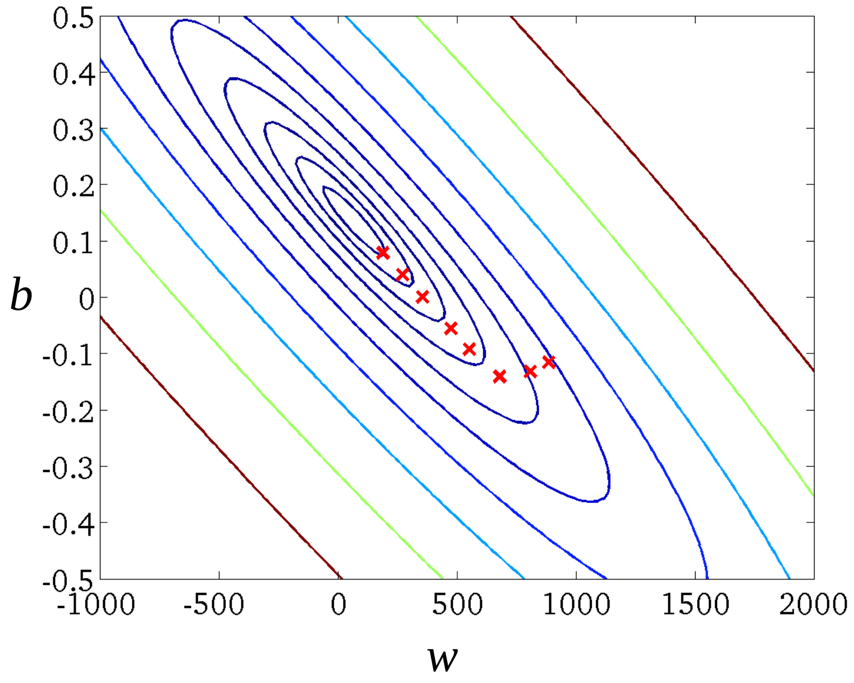
$$f(x) = wx + b$$

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

$$J(w, b)$$

(function of the parameters w, b)



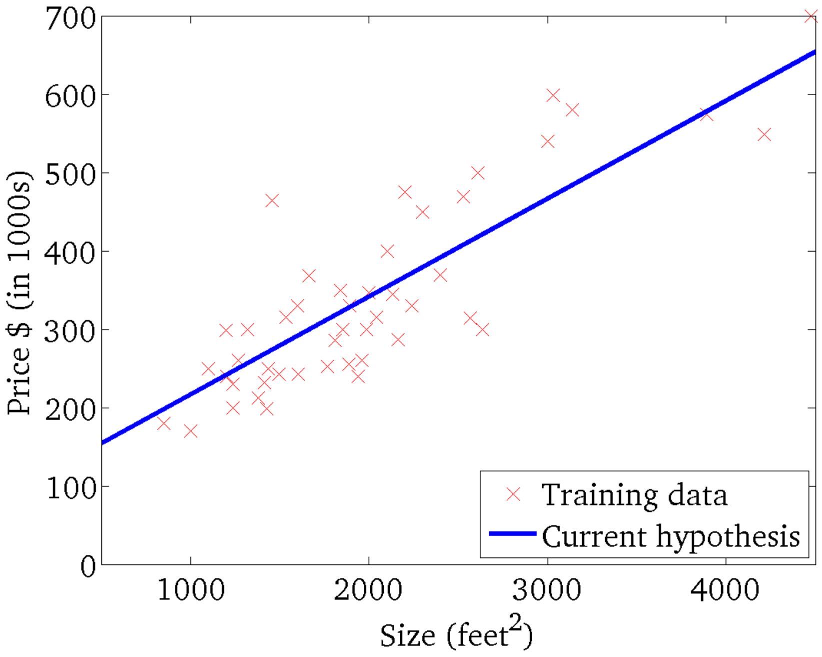


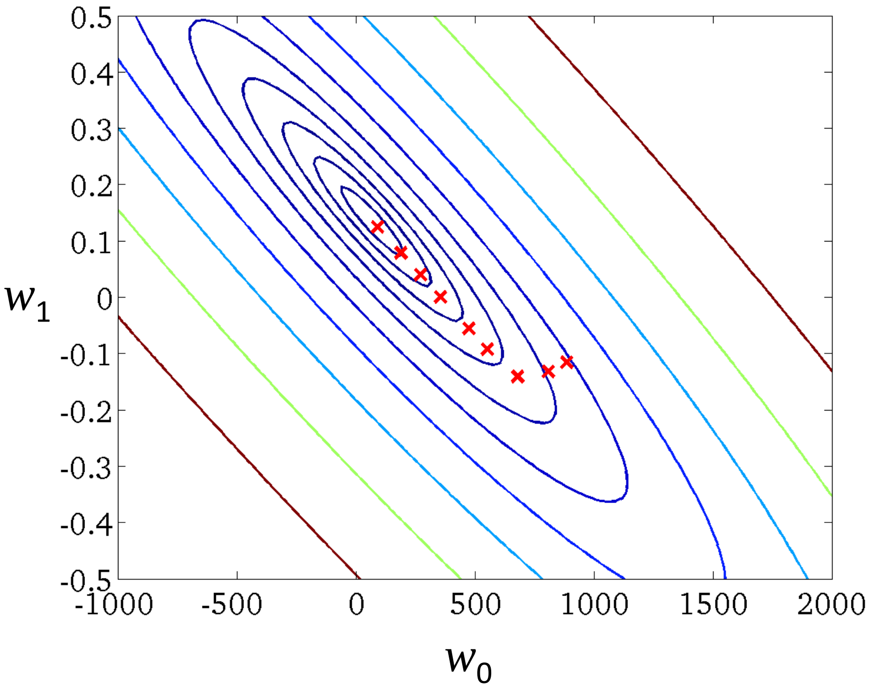
$$f(x) = wx + b$$

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

$$J(w, b)$$

(function of the parameters w, b)





Linear Regression with One Variable

- Gradient decent

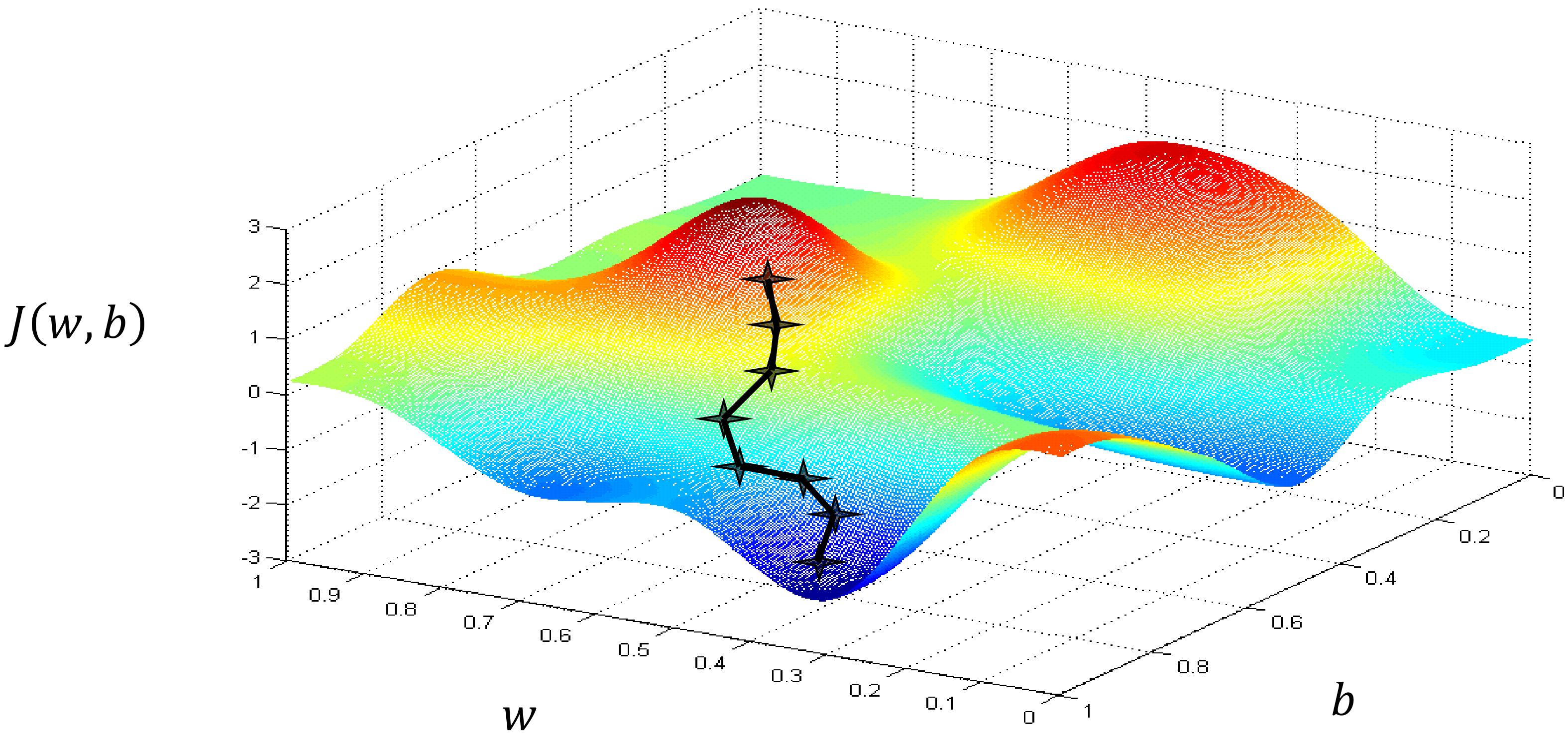
Have a cost function $\boldsymbol{J(w, b)}$

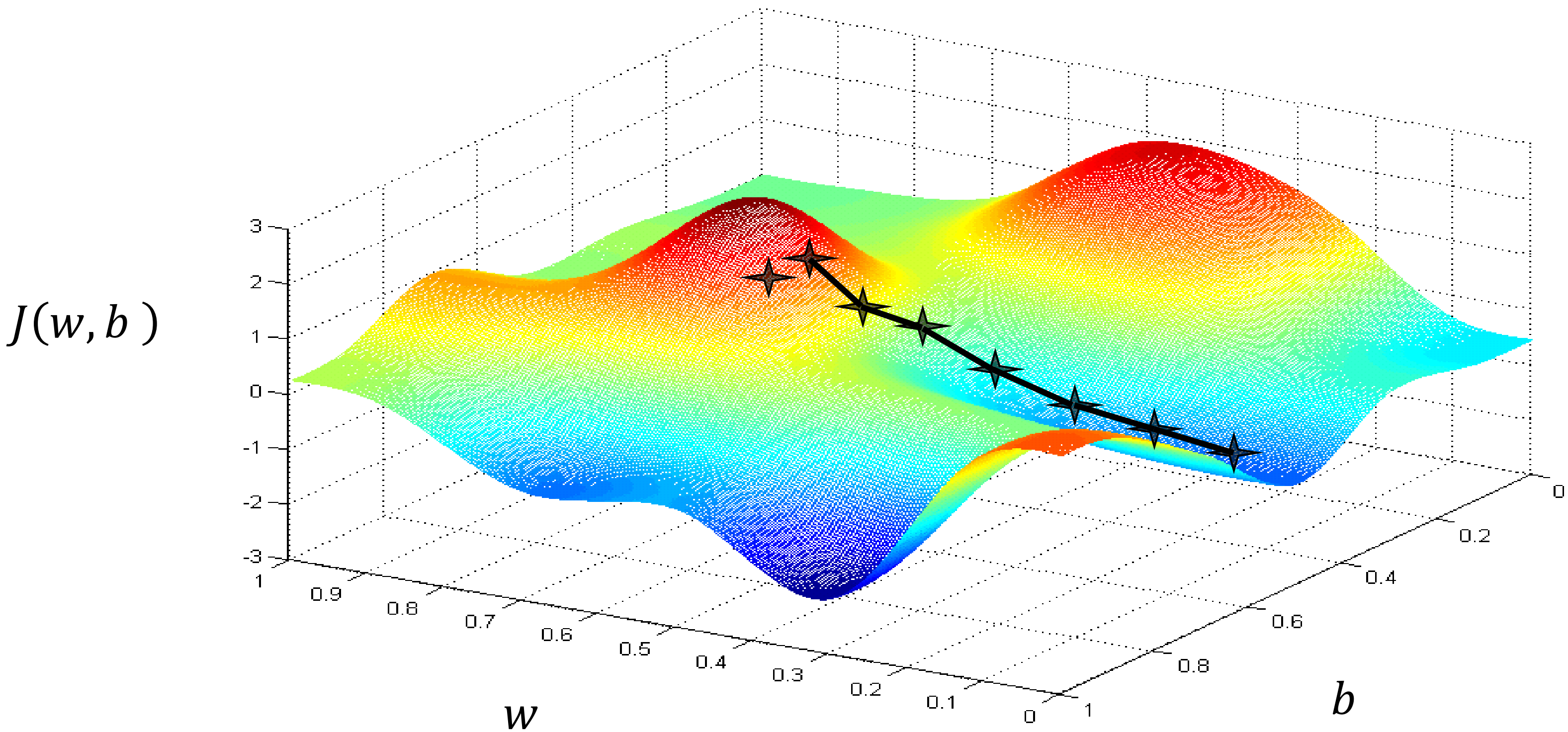
Want $\min_{w,b} \boldsymbol{J(w, b)}$

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

Repeat until convergence

$$\left\{\begin{array}{l} w = w - \alpha \boxed{\frac{\partial}{\partial w} J(w,b)} \\ b = b - \alpha \frac{\partial}{\partial b} J(w,b) \end{array}\right.$$

Learning rate

Derivative

Simultaneously update w and b

Correct: Simultaneous update

$$\left.\begin{array}{l} tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b) \\ tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w,b) \end{array}\right\}$$

$$w = tmp\_w$$

$$b = tmp\_b$$

Incorrect

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

$$w = tmp\_w$$

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

$$b = tmp\_b$$

Simplified

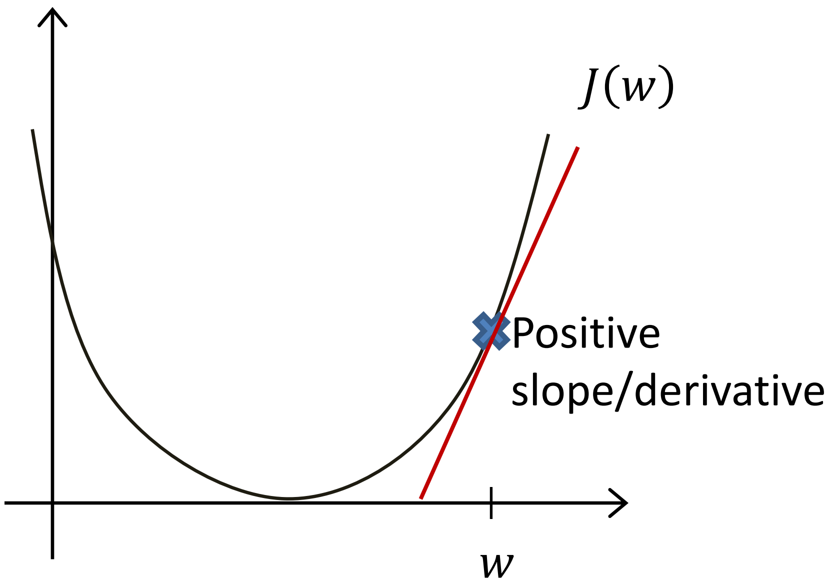
Gradient descent algorithm

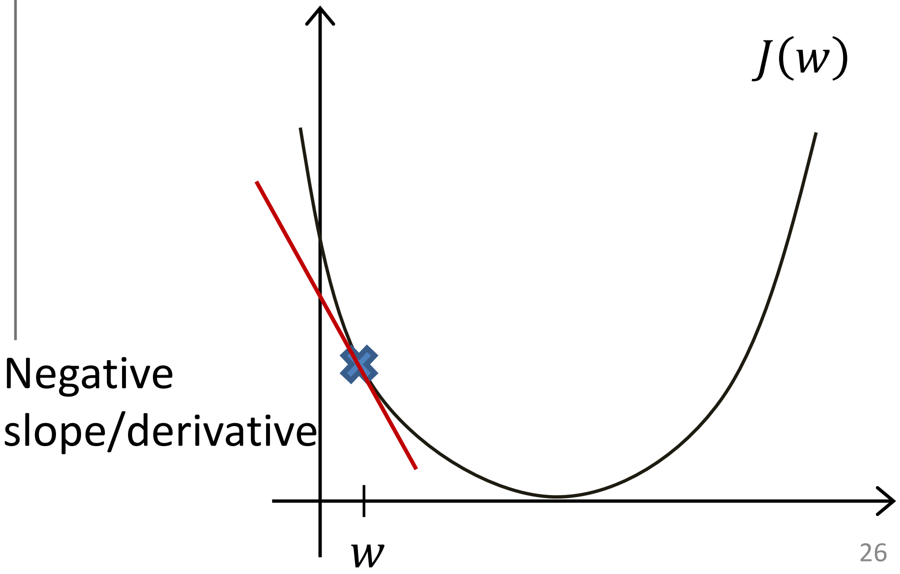
repeat until convergence {

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

}

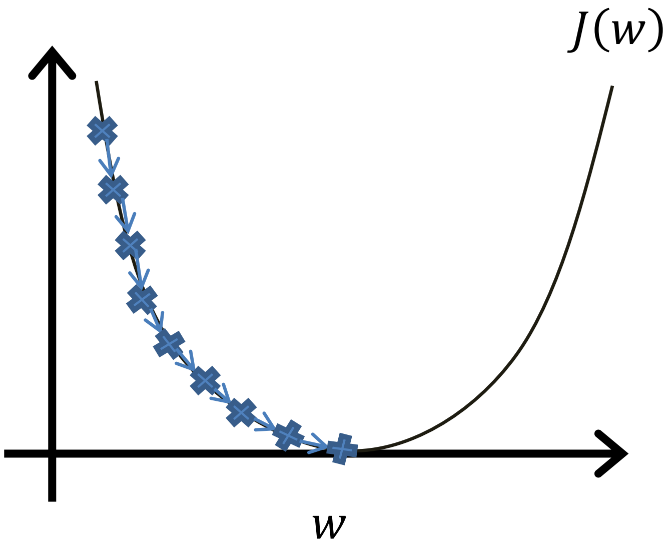
$$w = w - \alpha \frac{d}{dw} J(w)$$



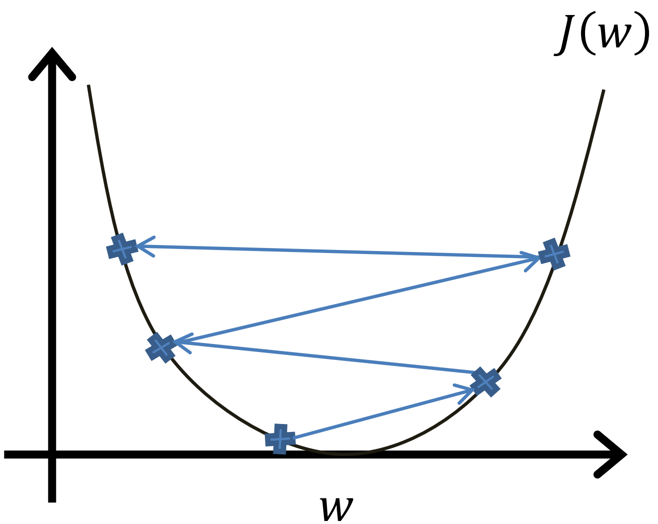


$$\text{w} = w - \alpha \frac{d}{dw} J(w)$$

If α is too small, gradient descent can be slow

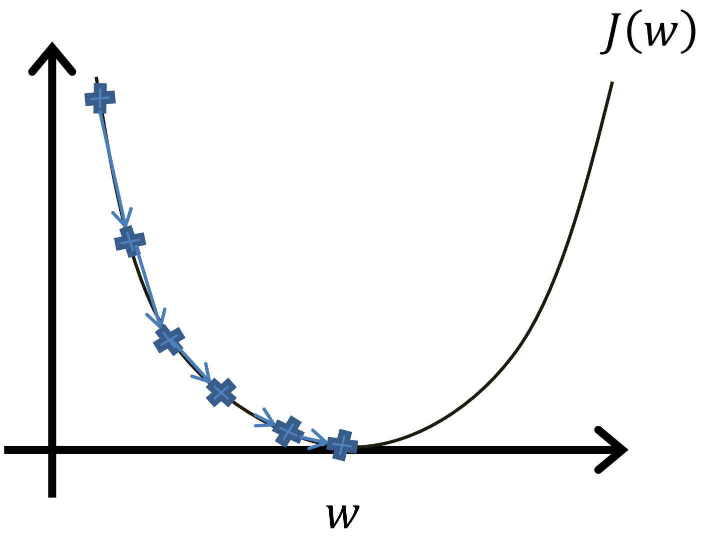


If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge



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

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



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

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

Gradient descent algorithm

repeat until convergence {

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

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

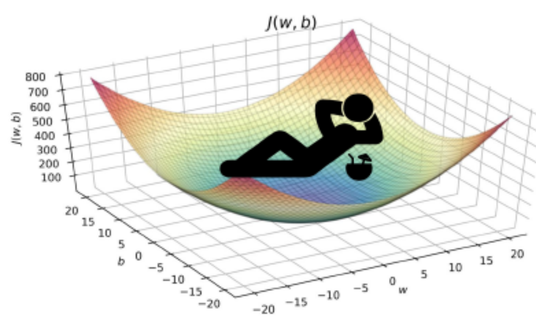
}

"Batch" Gradient Descent

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

	x size in feet^2	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
(multivariate linear regression)

- Gradient descent for multiple features

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

$x^{(i)}$ = input (features) of i^{th} training example.

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

Hypothesis:

Previously: $h_w(x) = w_0 + w_1 x$

Now: Multivariate linear regression.

$$h_w(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

For convenience of notation, define $x_0 = 1$.

$$h_w(x) = W^T x = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

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

Cost function:

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

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 {

$$w_0 := w_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)}_{\frac{\partial}{\partial w_0}J(w)}$$

$$w_1 := w_1 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) .\ x^{(i)}}_{\frac{\partial}{\partial w_1}J(w)}$$

}

(simultaneously update w_0 and w_1)

New algorithm $(n \geq 1)$:

Repeat {

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

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

}

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

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

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

....

Gradient descent in practice:

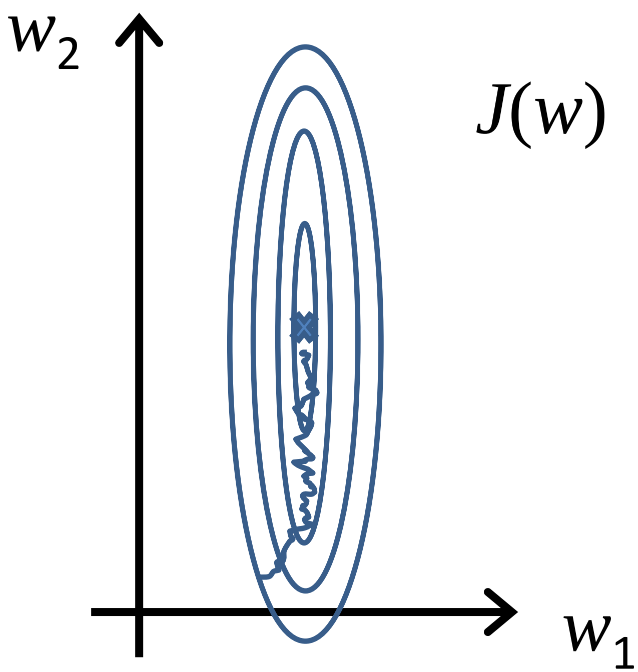
- Feature Scaling
- Debugging
- Learning rate
- Linear Regression with multiple variables
- Normal equation
- Other optimization algorithms
- Regularization
- Regression Evaluation

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.

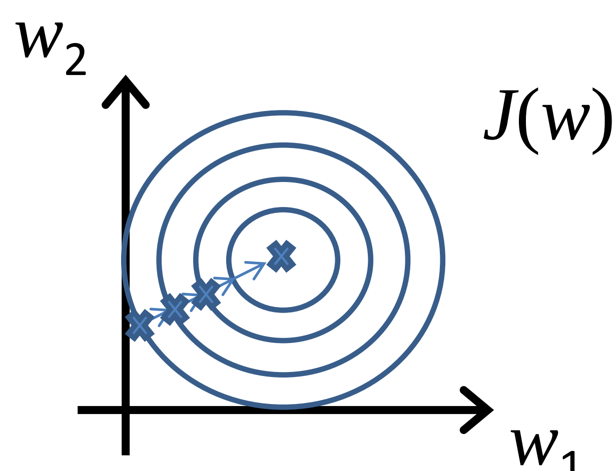
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



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

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



Rule-of-thumb: Get every feature into approximately a $-1 \le x_i \le 1$ range, $-0.5 \le x_i \le 0.5$, 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)

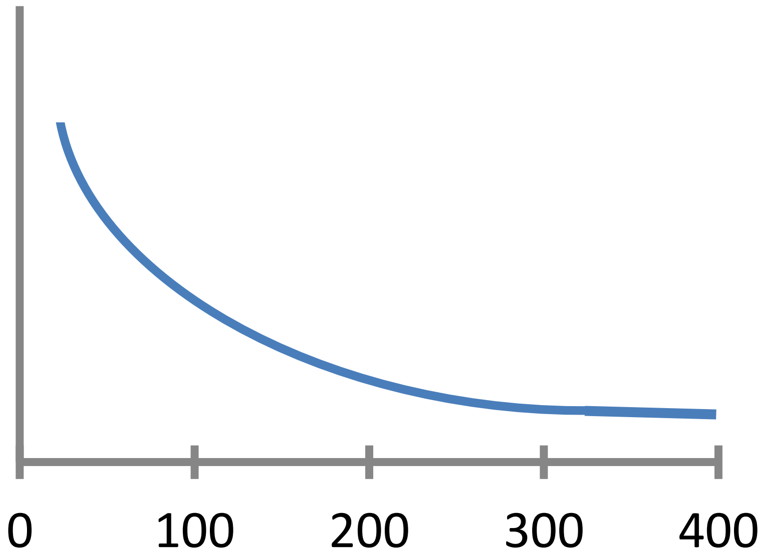
$x_2 = \frac{\#bedrooms - 2}{4}$ (average # of bedrooms is 2, and the range is from 1 to 5)

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

Debugging:

Making sure gradient descent is working correctly.

$\min_{w} j(w)$

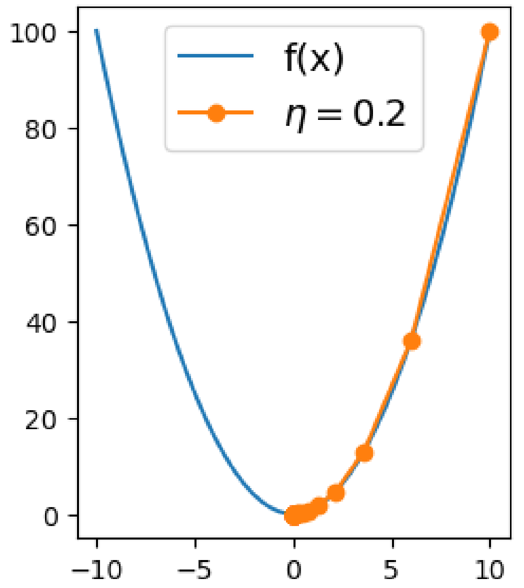


No. of iterations

(data dependent)

automatic convergence test:

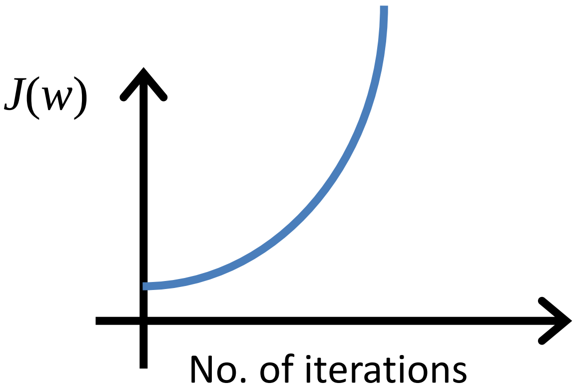
Declare convergence if $J(w)$ decreases by less than 10^{-3} in one iteration.

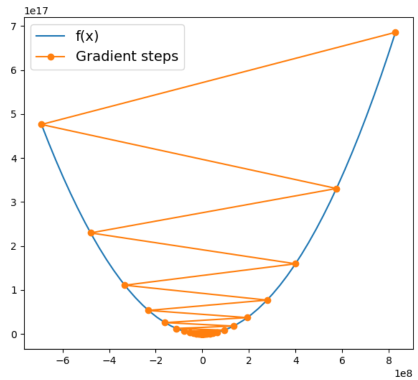


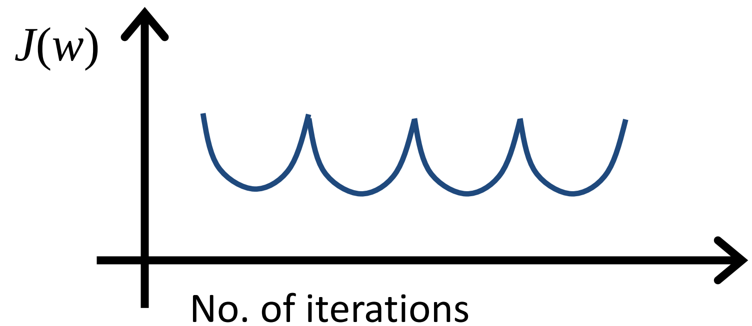
Learning rate:

Gradient descent not working.

Use smaller α.







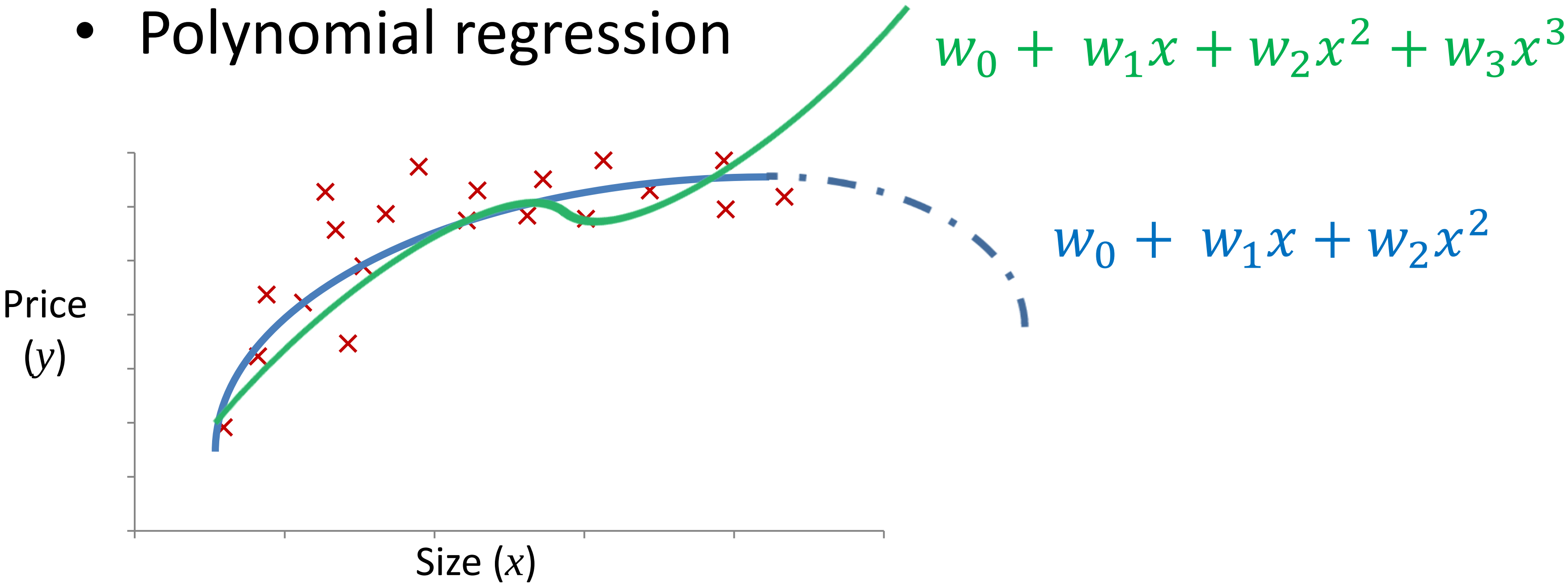
- For sufficiently small α, $J(w)$ should decrease on every iteration.
- If α is too small, gradient descent can be slow to converge.
- If α is too large: $J(w)$ may not decrease on every iteration; may not converge.

To choose α, try

…, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, …

Linear Regression with multiple variables

- Polynomial regression



$$h_w(x) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$
$$= w_0 + w_1(size) + w_2(size)^2 + w_3(size)^3$$

$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

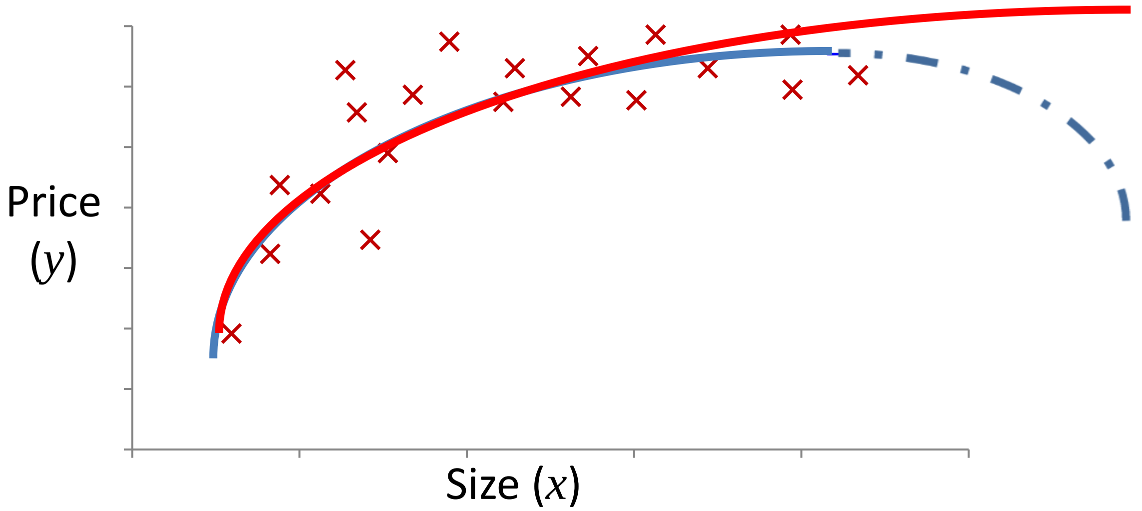
Scales:

Size: 1 – 1000

Size^2: 1 – 1000,000

Size^3: 1 – 10^9

Use feature scaling.

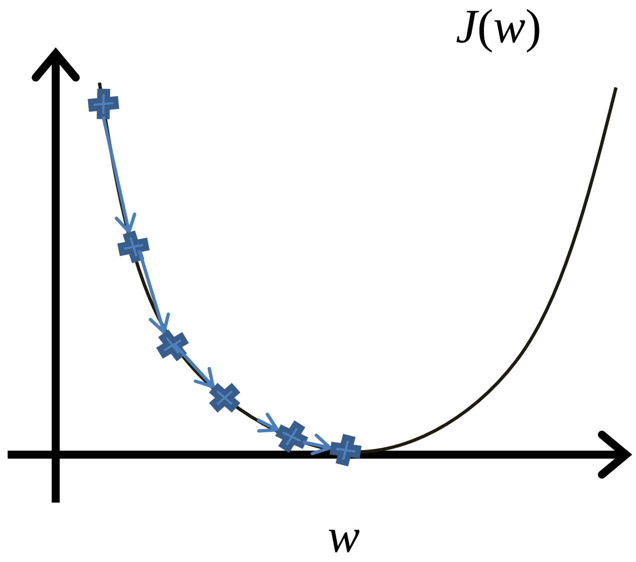


$$h_w(x) = w_0 + \ w_1(size) + w_2(size)^2$$

$$h_w(x) = w_0 + \ w_1(size) + w_2\sqrt{(size)}$$

Normal equation

Gradient Descent



Normal equation: Method to solve for w analytically.

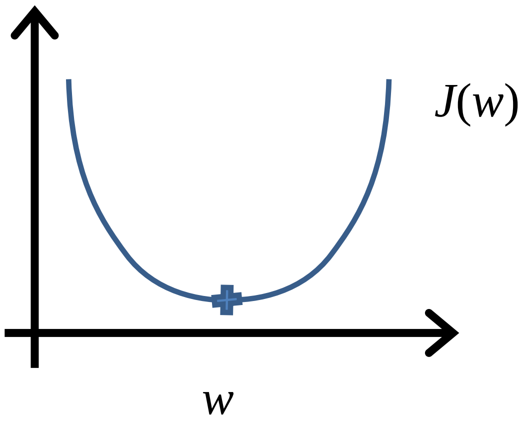
Intuition: If 1D ($w \in \mathbb{R}$)

$$J(w) = aw^2 + bw + c$$

To minimize this function

$$\frac{\partial}{\partial w} J(w) = \cdots = 0$$

Solve for w



$w \in \mathbb{R}^{n+1}$

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

$$\frac{\partial}{\partial w_j} J(w) = \cdots = 0 \quad \text{(for every } j\text{)}$$

Solve for $w_0, w_1, \ldots, w_n$

Examples: $m = 4$.

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$y = Xw$$

$$X^T y = XTX\,w$$

$$\left(X^T X\right)^{-1} X^T y = \left(X^T X\right)^{-1} X^T X\,w$$

$$\left(X^T X\right)^{-1} X^T y = I\,w$$

$$\left(X^T X\right)^{-1} X^T y = w$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \quad X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

y is (m x 1) matrix, X is (m by n) matrix, w is (n by 1) matrix

m training examples, n features.

<u>Gradient Descent</u>

- Need to choose α.
- Needs many iterations.
- Works well even when n is large.

<u>Normal Equation</u>

- No need to choose α.
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$

  O(n^3)
- Slow if n is very large.
- No need for features scaling

Linear Regression with multiple variables

- Normal equation and non-invertibility

Normal equation $w = (X^T X)^{-1} X^T y$

- What if $X^T X$ is non-invertible? (singular/ degenerate)

- Redundant features (linearly dependent).
  E.g. $x_1 =$size in feet^2
  $x_2 =$size in m^2

- Too many features (e.g. $m \leq n$).

Solution:

- Delete some features.

Other Optimization algorithms than "Gradian descent"

- Conjugate gradient
- BFGS
- L-BFGS
- …

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

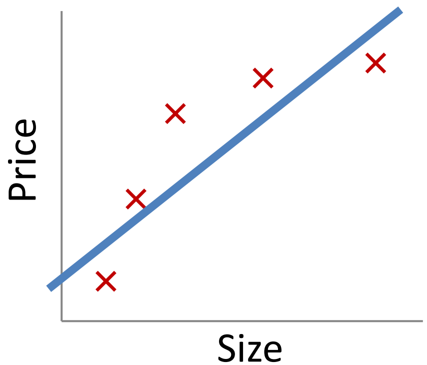
Disadvantages:

- More complex

Regularization

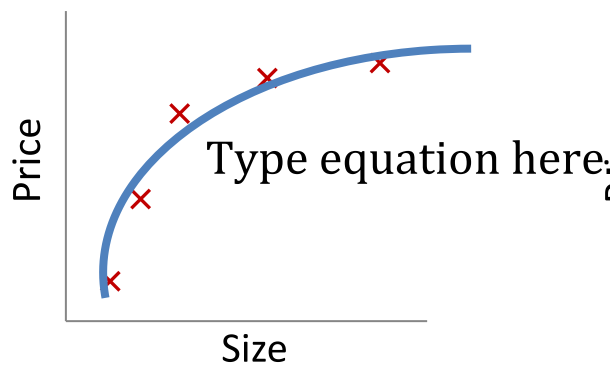
- The problem of overfitting

Example: Linear regression (housing prices)



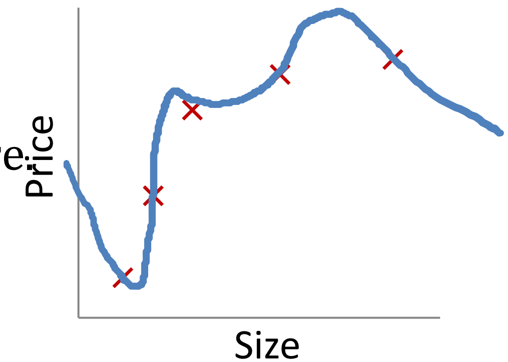
$w_0 + w_1 x$

"underfit/high bias"



$w_0 + w_1 x + w_2 x^2$

"just right"



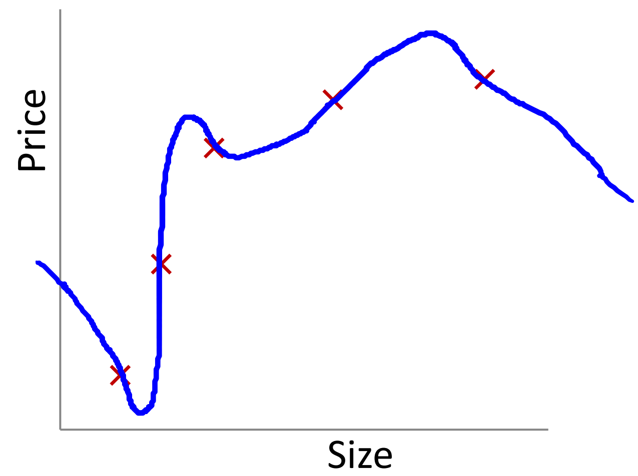
$w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

"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:

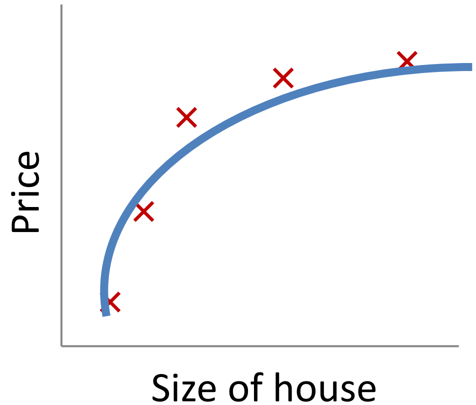
Options:

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 .

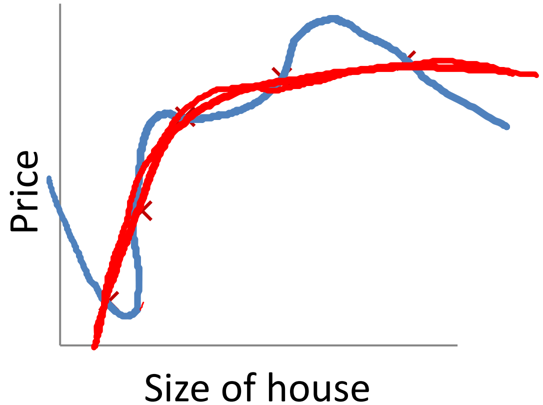
Regularization

- 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.

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

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

Regularization.

Small values for parameters $w_0, w_1, \ldots, w_n$

- "Simpler/smoother" hypothesis
- Less prone to overfitting

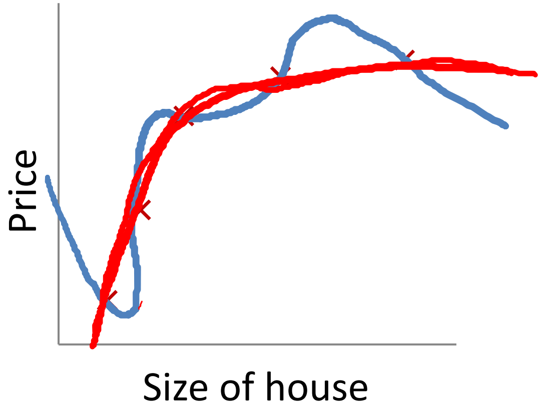
Housing:

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

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

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

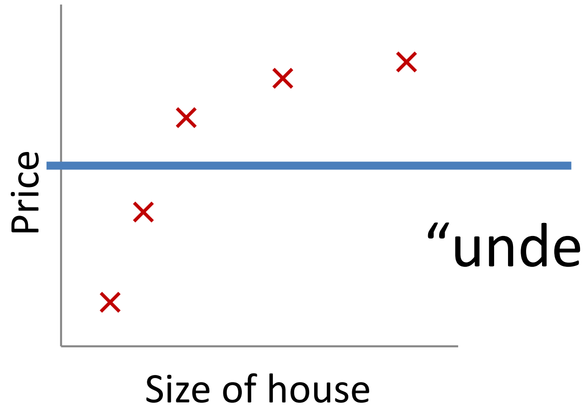
$$\min_w J(w)$$



In regularized linear regression, we choose w to minimize

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

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

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

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

Gradient descent

Repeat $\{$

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

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

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

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

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

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