Linear Regression

DCS310

Sun Yat-sen University

Outline

- Introduction
- Single Feature Case
- Multiple Features Case
- Numerical Optimization

Introduction

- What is regression?

  Based on the given features, predict the values of interested variables

- Example: House price prediction

Features

Interested variable

Size (feet) x_1	# bedrooms x_2	# floors x_3	# years (Ages) x_4	Price ($ 1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
….	….	….	….	….

Features: 1) size, 2) # bedrooms, 3) # floors, 4) # years

- Mathematically, regression aims at building a function $f(\cdot)$ to model the relation between input data x and supervised value y

$$\hat{y} = f(x_1, x_2, x_3, x_4)$$

- Linear regression

  Restricting the function $f(\cdot)$ to be of linear form, *i.e.*,

$$f(x_1, x_2 \cdots x_m) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$

  - w_k: model parameters
  - m: number of features

- Objective

Find a set of parameters $\{w_k\}_{k=1}^{m}$ so that the prediction

$$\hat{y} = f(x_1, x_2, \cdots, x_m)$$

is as close as possible to the true y values *for all data samples* in the training dataset

	Size (feet) x_1	# bedrooms x_2	# floors x_3	# years (Ages) x_4	Price (\$ 1000) y
Sample 1 →	2104	5	1	45	460
Sample 2 →	1416	3	2	40	232
Sample 3 →	1534	3	2	30	315
Sample 4 →	852	2	1	36	178
	….	….	….	….	….

Linear model is very understandable, or saying explainable.

Outline

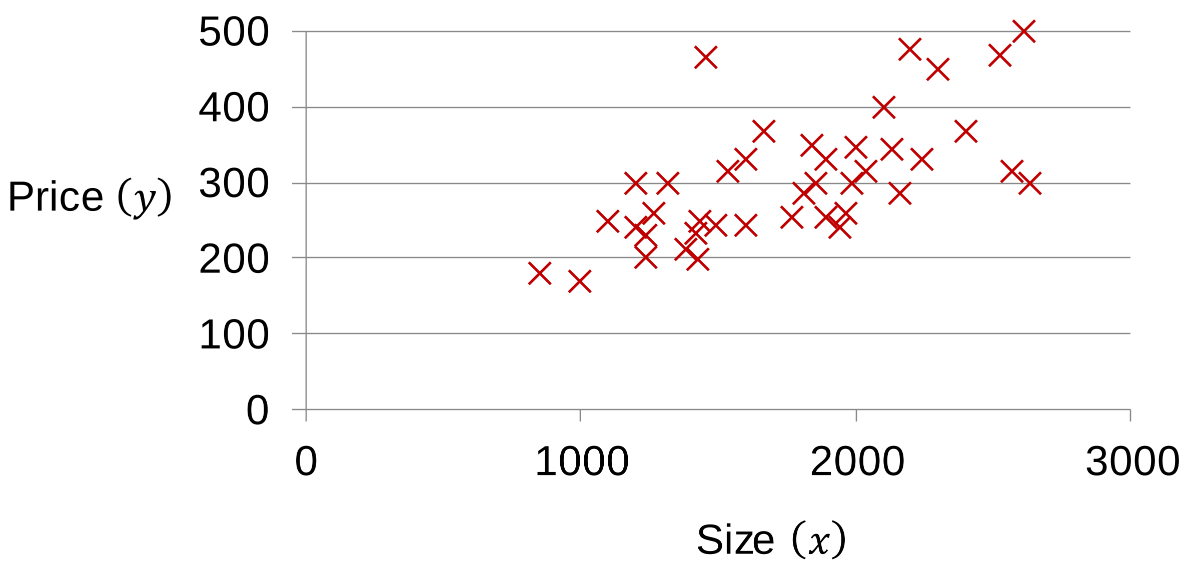
- Introduction
- Single Feature Case
- Multiple Features Case
- Numerical Optimization

Model

- For simplicity, first consider only one feature, *e.g.*, *house size*

Size (feet) x	Price ($ 1000) y
2104	460
1416	232
852	178
….	….

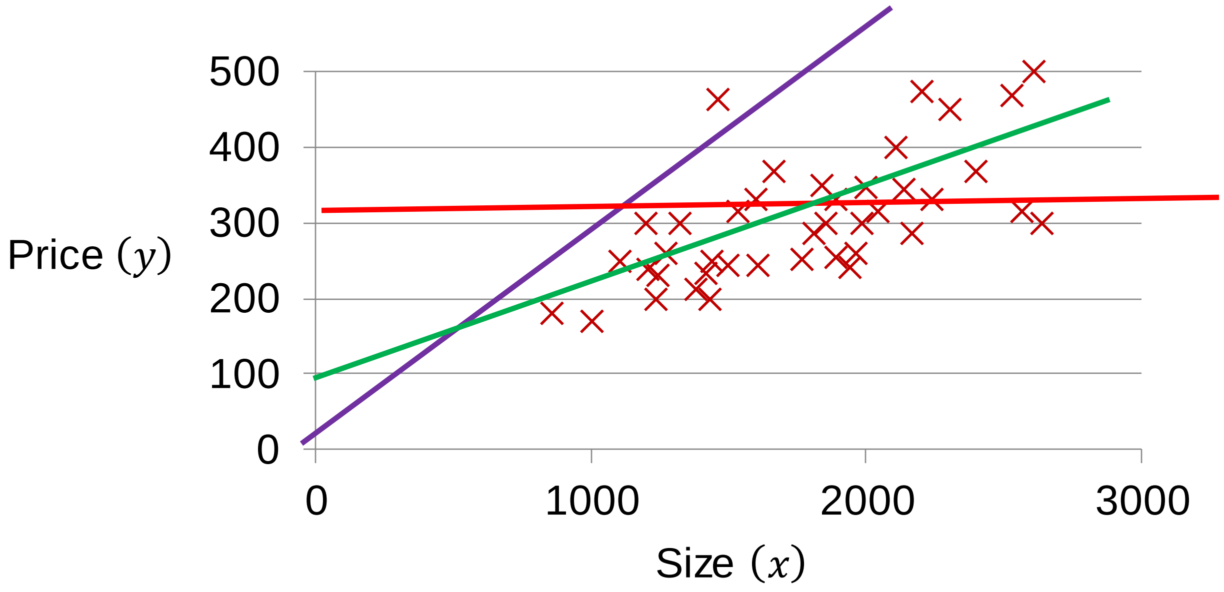
- Plot the (x, y) pairs on a plane



- The prediction function reduces to

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

- For different w_0 and w_1, the function $f(x)$ represents different straight lines



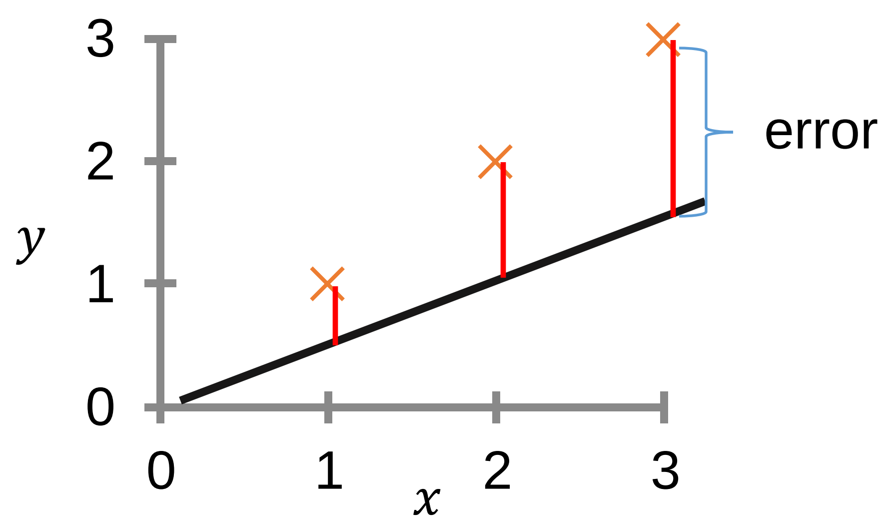
- The objective is to find an appropriate w_0 and w_1 so that the line is as fit as possible with the true y's of all given x

Cost / Loss Function

- Mathematically, the objective can be described as minimizing the cost (loss) function, a.k.a. mean squared error (MSE)

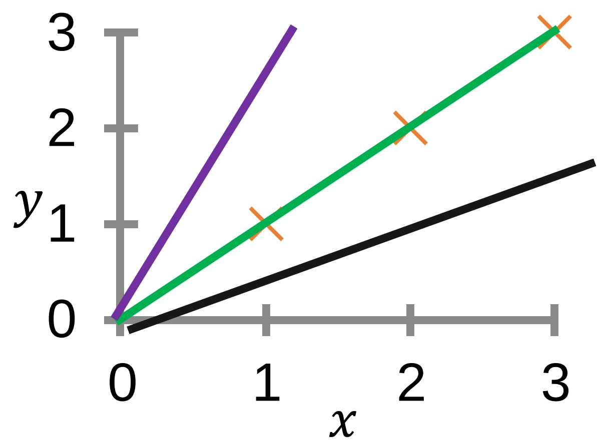
$$L(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2$$

where $x^{(i)}$ and $y^{(i)}$ denote the feature and target value of the i-th training sample; n is the number of training samples



- Substituting $f\left(x^{(i)}\right) = w_0 + w_1 x^{(i)}$ into $L(w_0, w_1)$ gives

$$L(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}\left(w_0 + w_1 x^{(i)} - y^{(i)}\right)^2$$



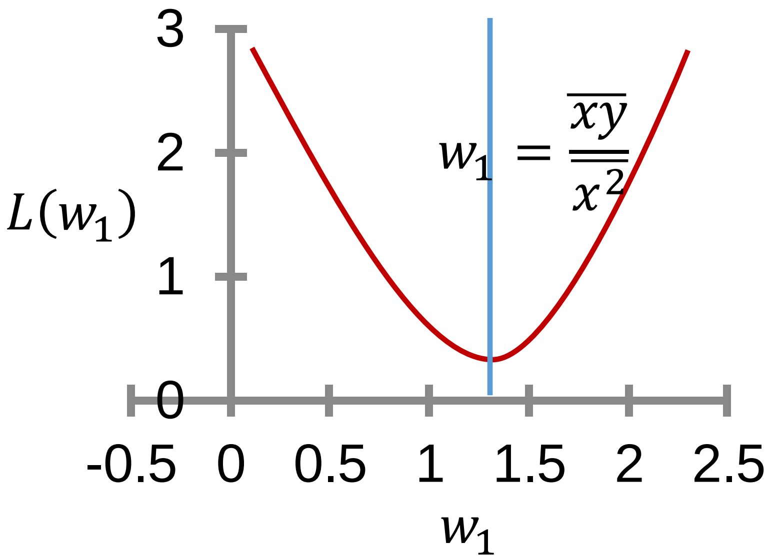
Remark: To better understand this cost function, we simplify it by setting $w_0 = 0$

- Then, the cost function becomes

$$\boxed{L(w_1) = \overline{x^2}w_1^2 - 2\overline{xy}w_1 + \overline{y^2}}$$

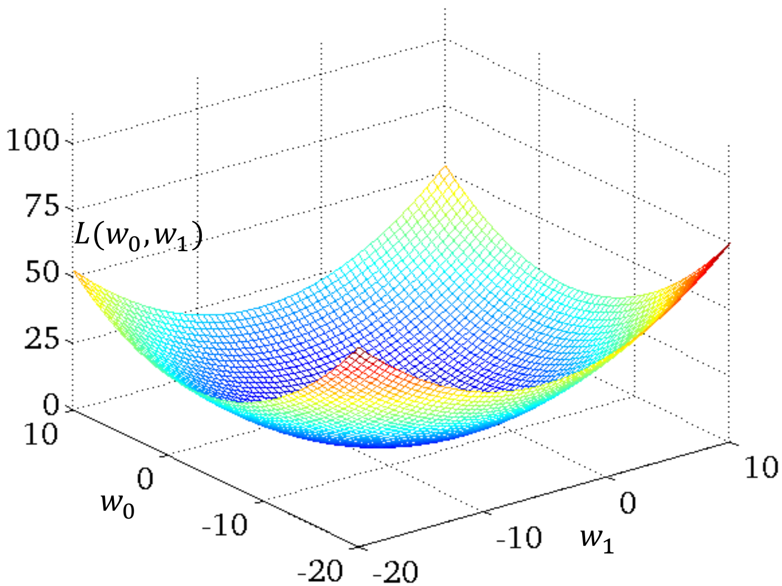
where $\overline{x^2} = \frac{\sum_{i=1}^{n}\left(x^{(i)}\right)^2}{n}$, $\overline{xy} = \frac{\sum_{i=1}^{n} x^{(i)} y^{(i)}}{n}$ and $\overline{y^2} = \frac{\sum_{i=1}^{n}\left(y^{(i)}\right)^2}{n}$

- The cost function is a quadratic function *w.r.t.* w_1



$$L(w_1) = \overline{x^2}w_1^2 - 2\overline{xy}w_1 + \overline{y^2}$$

- If w_0 is taken into account, the cost function $L(w_0, w_1)$ is still a quadratic function, but is two-dimensional



$$L(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}\left(w_0 + w_1 x^{(i)} - y^{(i)}\right)^2$$

- The best w_0 and w_1 can be found by setting the derivatives to zero

$$\frac{\partial L}{\partial w_0} = \frac{2}{n}\sum_{i=1}^{n}\left(w_0 + w_1 x^{(i)} - y^{(i)}\right) = 0$$

$$\frac{\partial L}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{n}\left(w_0 + w_1 x^{(i)} - y^{(i)}\right) x^{(i)} = 0$$

- It can be derived that the best w_0 and w_1 are

$$w_0 = \frac{\overline{xy}\bar{x} - \overline{x^2}\bar{y}}{\bar{x}^2 - \overline{x^2}}$$

$$w_1 = \frac{\bar{x}\bar{y} - \overline{xy}}{\bar{x}^2 - \overline{x^2}}$$

Outline

- Introduction
- Single Feature Case
- Multiple Features Case
- Numerical Optimization

- Training data from single feature to multiple feature case, a.k.a. multivariate linear regression

Size (feet) x	Price ($ 1000) y
2104	460
$\boxed{1416}$ →	232
1534	315
852	178
….	….

Size (feet) x_1	# bedrooms x_2	# floors x_3	# years (Ages) x_4	Price ($ 1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
….	….	….	….	….

- The function of a general linear regression is

$$f(x_1, x_2 \cdots x_m) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$

  - x_i is the i-th feature

- Working with the scalar form is cumbersome. Reformulating it into a matrix-form gives

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

  - $\boldsymbol{x} = [1, x_1, x_2, \cdots, x_m] \in \mathbb{R}^{1\times(m+1)}$ is the feature row vector
  - $\boldsymbol{w} = [w_0, w_1, w_2, \cdots, w_m]^T \in \mathbb{R}^{(m+1)\times 1}$ is the parameter column vector

By setting the first element in $\boldsymbol{x}$ to be 1, w_0 can be treated the same as the other parameters w_k

Cost Function

- The objective is still to find a $\boldsymbol{w}$ such that the prediction

$$f\left(\boldsymbol{x}^{(i)}\right) = \boldsymbol{x}^{(i)}\boldsymbol{w}$$

is close to the true value $y^{(i)}$, where $\boldsymbol{x}^{(i)}$ and $y^{(i)}$ is the feature vector and target value of the i-th training sample

Size (feet) x_1	# bedrooms x_2	# floors x_3	# years (Ages) x_4	Price ($ 1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
….	….	….	….	….

- Thus, the cost function can be represented as

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

- The cost function can be further written as

$$L(\boldsymbol{w}) = \frac{1}{n}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2$$

where $\boldsymbol{X}$ and $\boldsymbol{y}$ are the feature matrix the target vector, defined as

$$\boldsymbol{X} \triangleq \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \vdots \\ \boldsymbol{x}^{(n)} \end{bmatrix} \in \mathbb{R}^{n\times(m+1)}, \qquad \boldsymbol{y} \triangleq \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \in \mathbb{R}^{n\times 1}$$

n行m+1列

w_0

	Size (feet)	# bedrooms	# floors	# years (Ages)	Price ($ 1000)
x_0	x_1	x_2	x_3	x_4	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
1	….	….	….	….	….

$\boldsymbol{X} =$ (columns x_0–x_4), $\boldsymbol{y} =$ (column y)

- The gradient of the cost function w.r.t. $\boldsymbol{w}$ is

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{2}{n}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

- Since $L(\boldsymbol{w})$ is a convex function, its optima can be found by setting

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{2}{n}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) = 0$$

特征冗余

- Solving the equation gives

$$\underline{\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}}$$

特征数量 m 大于样本数量 n，X^TX 不满秩，则不可逆

- It can be verified that when the number of feature is 1, the result reduces to

$$w_0 = \frac{\overline{xy}\bar{x} - \overline{x^2}\bar{y}}{\bar{x}^2 - \overline{x^2}}, \qquad w_1 = \frac{\bar{x}\bar{y} - \overline{xy}}{\bar{x}^2 - \overline{x^2}}$$

- What about $\boldsymbol{X}^T\boldsymbol{X}$ is NOT invertible, i.e. not full-rank? Resulting multiple solutions, which one to select? Regularization

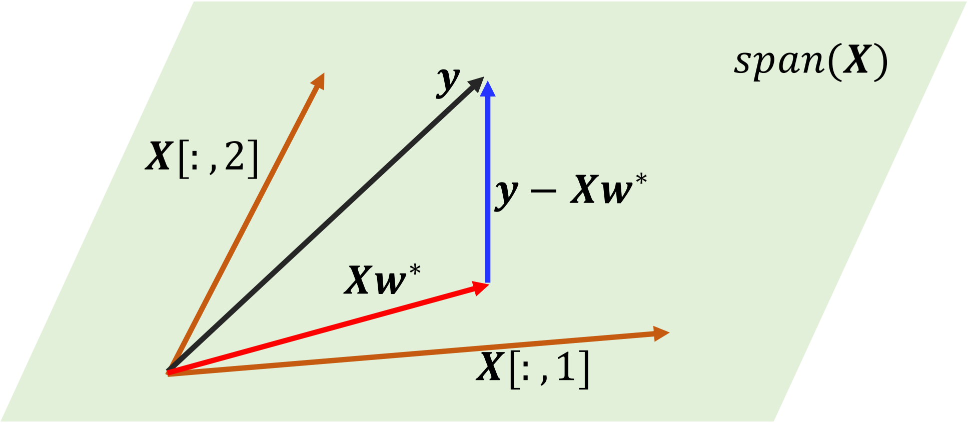
Geometric Interpretation

- From the requirement of $\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w}^* - \boldsymbol{y}) = \boldsymbol{0}$, we can see that

$$\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}^* \perp span(\boldsymbol{X})$$ 正交

Xw* 为 y 在 span(X)中的投影

- The result suggests that $\boldsymbol{X}\boldsymbol{w}^*$ can be understood as *the projection of $\boldsymbol{y}$ onto the space spanned by $\boldsymbol{X}$*



Outline

- Introduction
- Single Feature Case
- Multiple Features Case
- Numerical Optimization

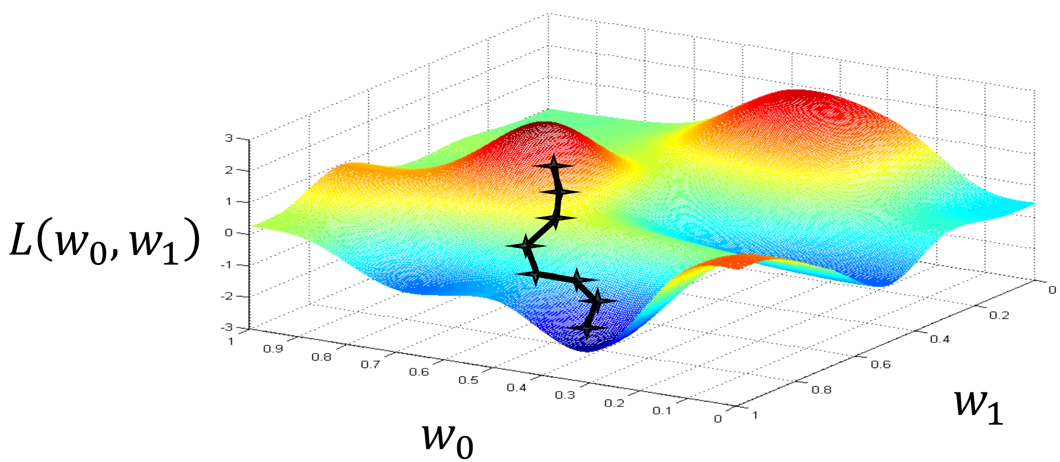
Gradient Descent

- Analytical solutions do *not always exist*, or evaluating the analytical expression is *computational expensive*
- Then, we should resort to numerical methods, *e.g.*, the gradient descent

梯度下降法

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - r \cdot \left.\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}^{(t)}}$$

  - r: the learning rate

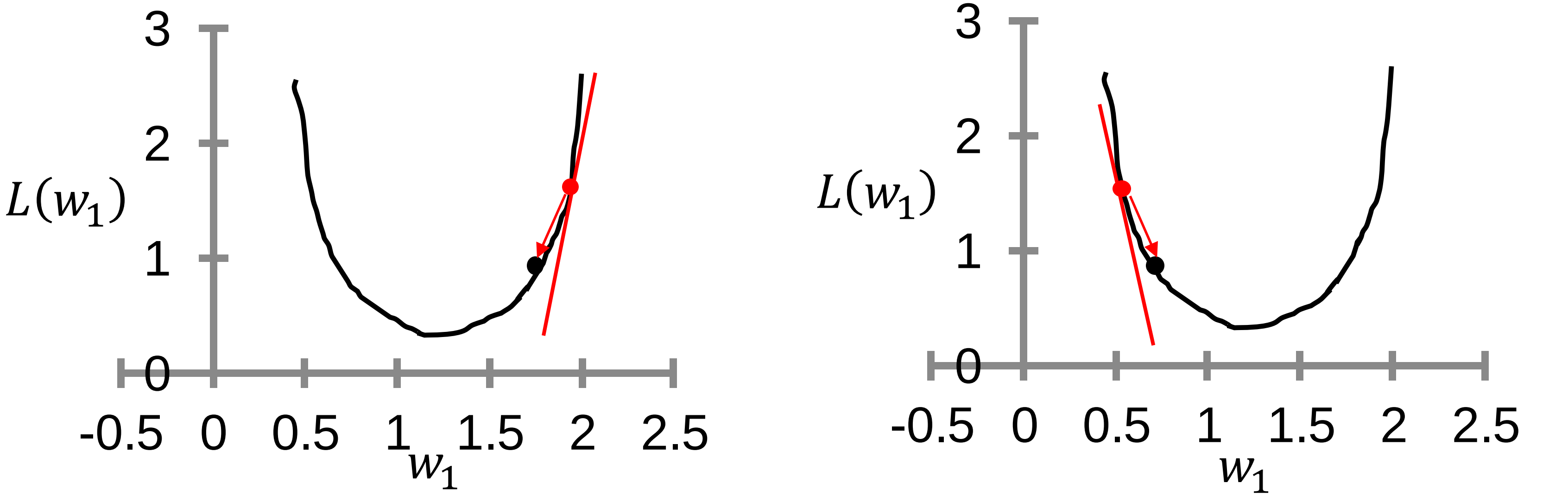


- Let us take the single-feature case and set $w_0 = 0$ as an example, in which the loss function becomes

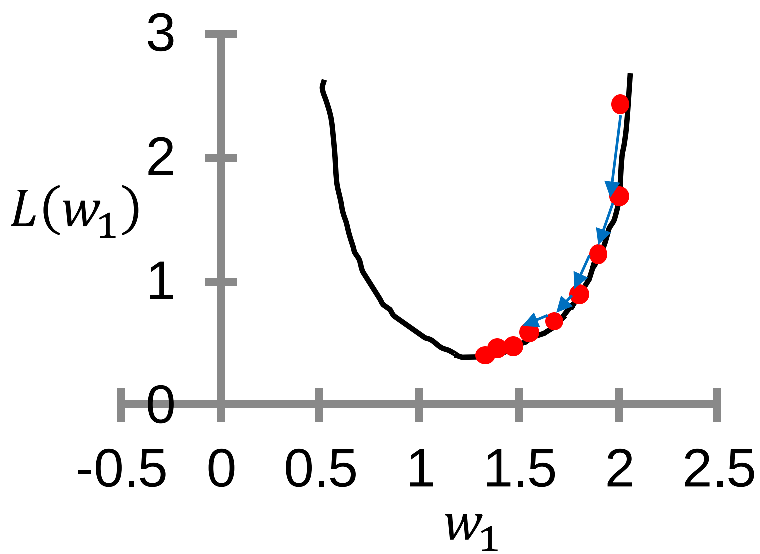
$$L(w_1) = \frac{1}{n}\sum_{i=1}^{n}\left(w_1 x^{(i)} - y^{(i)}\right)^2$$

- The parameter w_1 can be updated as

$$w_1^{(t+1)} = w_1^{(t)} - r \cdot \left.\frac{\partial L(w_1)}{\partial w_1}\right|_{w_1 = w_1^{(t)}}$$

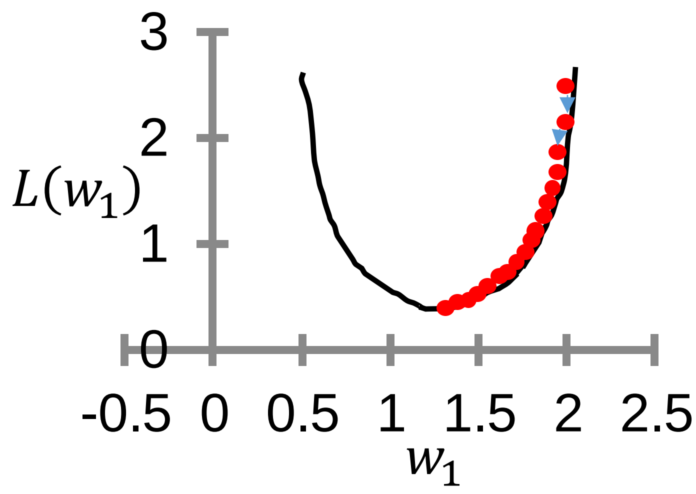


- With appropriate learning rate, the model parameter is iteratively updated, and will eventually converge to the optima

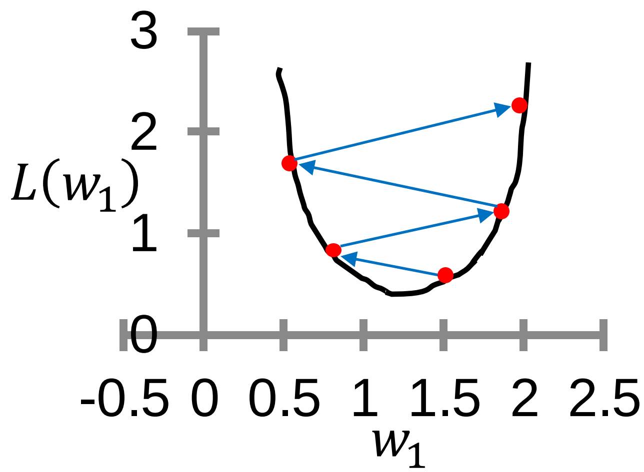


- As approaching the optimal value, the gradient becomes smaller and smaller. Thus, *even if the learning rate is fixed*, the updating intervals also approach 0 as the iteration proceeds, as long as the rate is set appropriately

- If the learning rate is too small, the convergence speed will be very slow



- If the learning rate is too large, the iteration may diverge

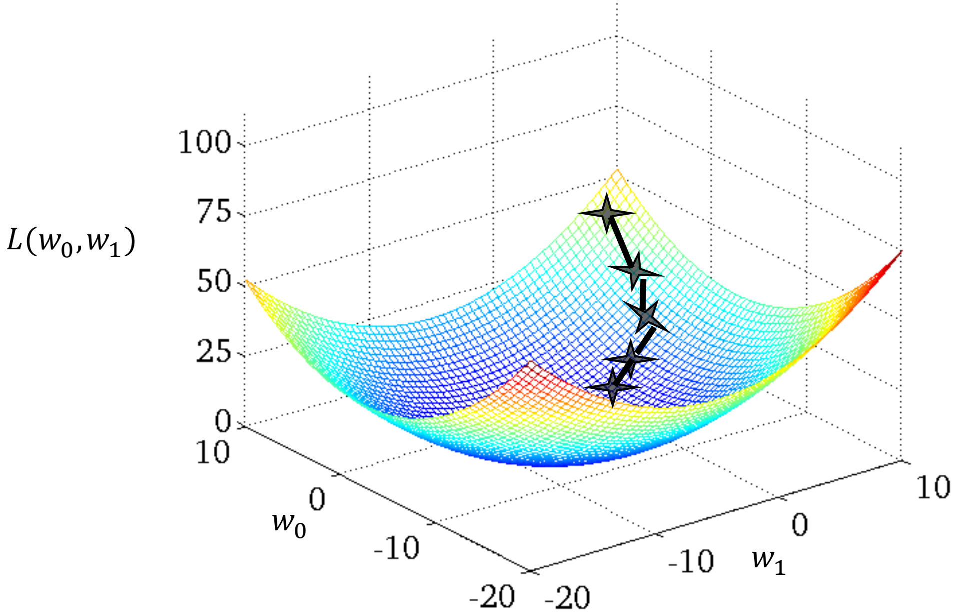


- So, setting appropriate learning rate is important

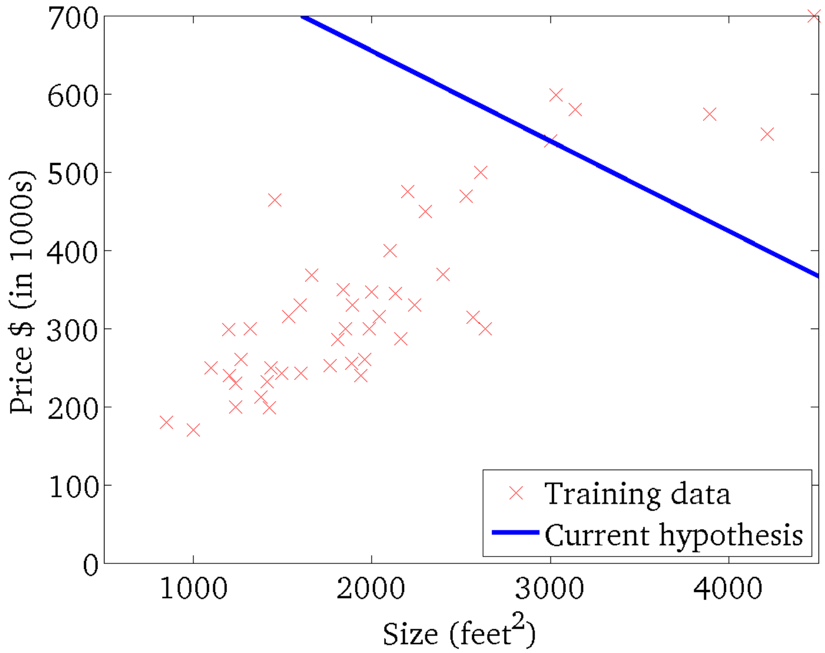
- Now consider the case with both w_0 and w_1

$$w_0^{(t+1)} = w_0^{(t)} - r \cdot \left.\frac{\partial L(w_0, w_1)}{\partial w_0}\right|_{w_0=w_0^{(t)},\, w_1=w_1^{(t)}}$$

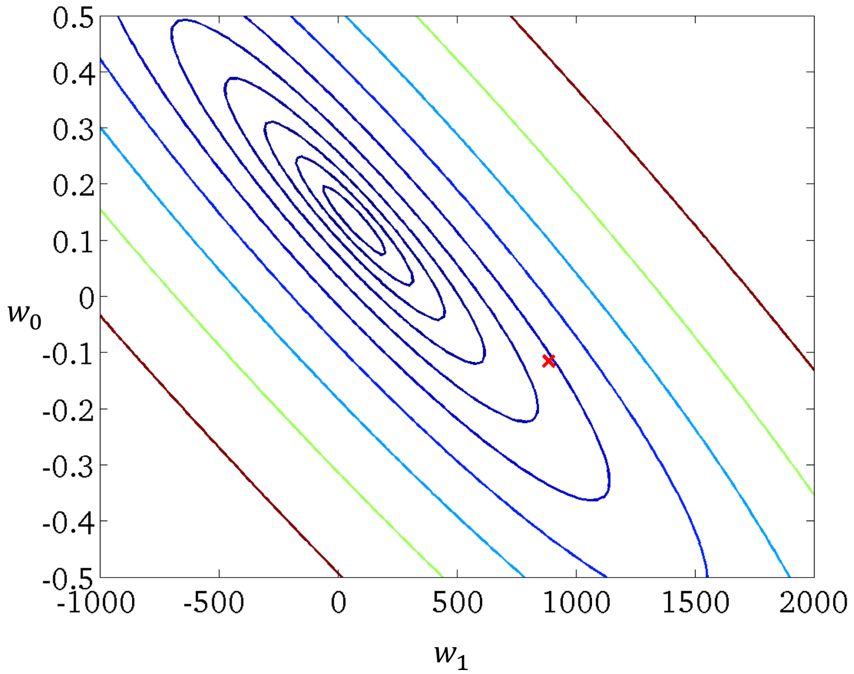
$$w_1^{(t+1)} = w_1^{(t)} - r \cdot \left.\frac{\partial L(w_0, w_1)}{\partial w_1}\right|_{w_0=w_0^{(t)},\, w_1=w_1^{(t)}}$$



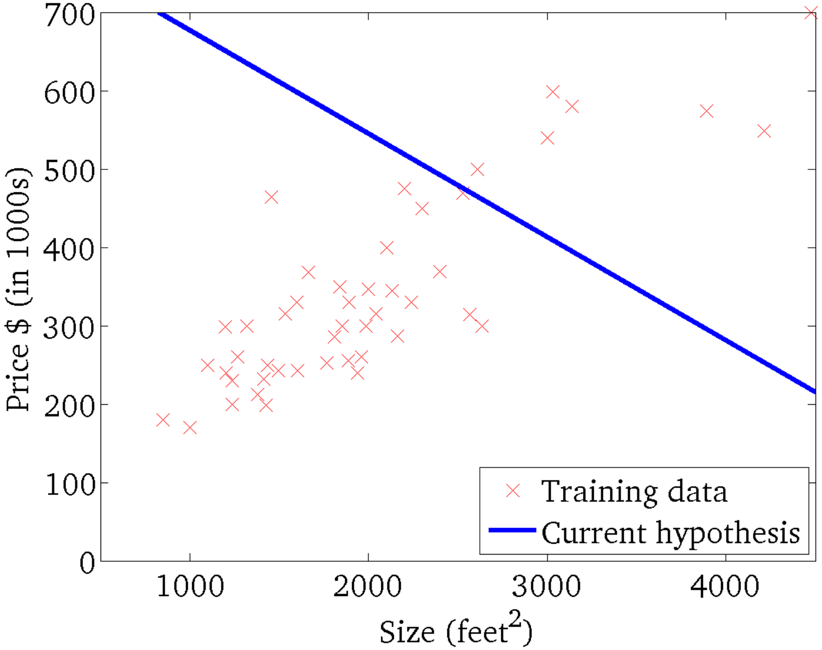
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



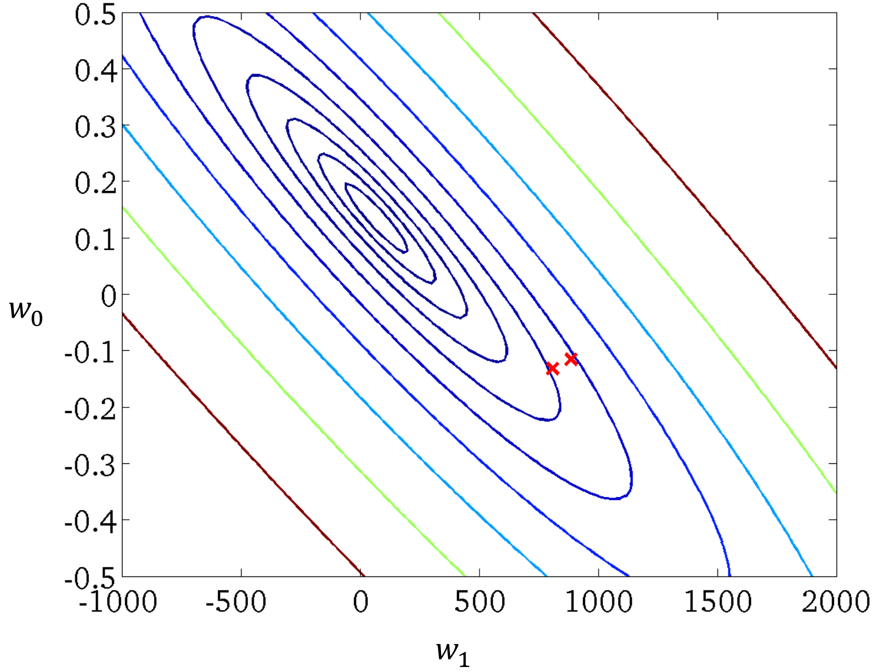
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)}\right)$



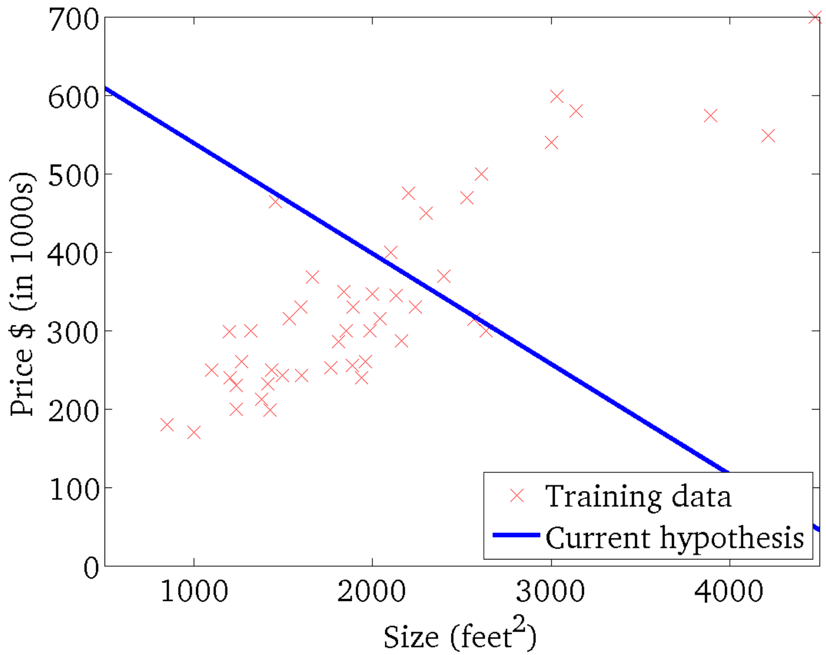
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



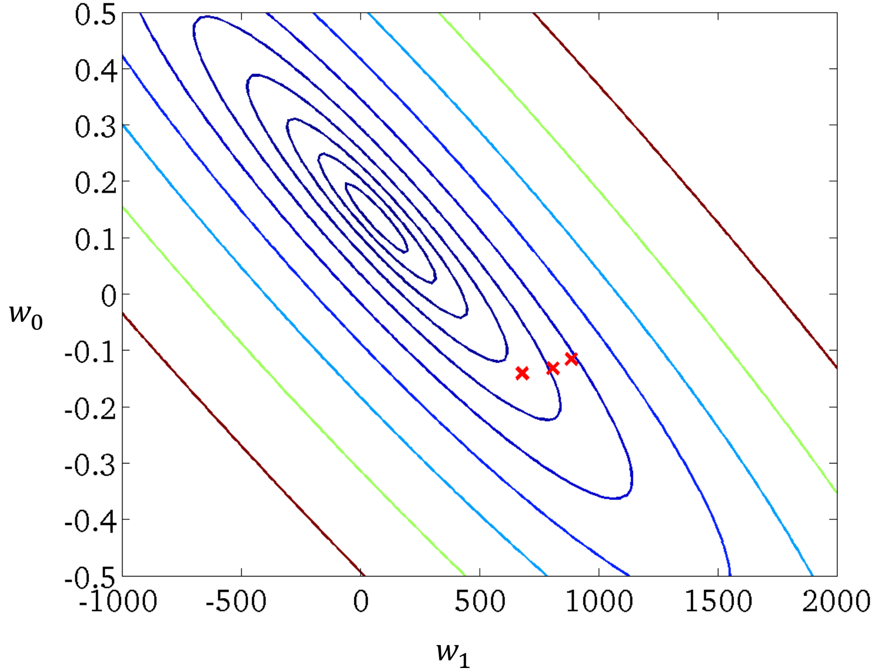
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)}\right)$



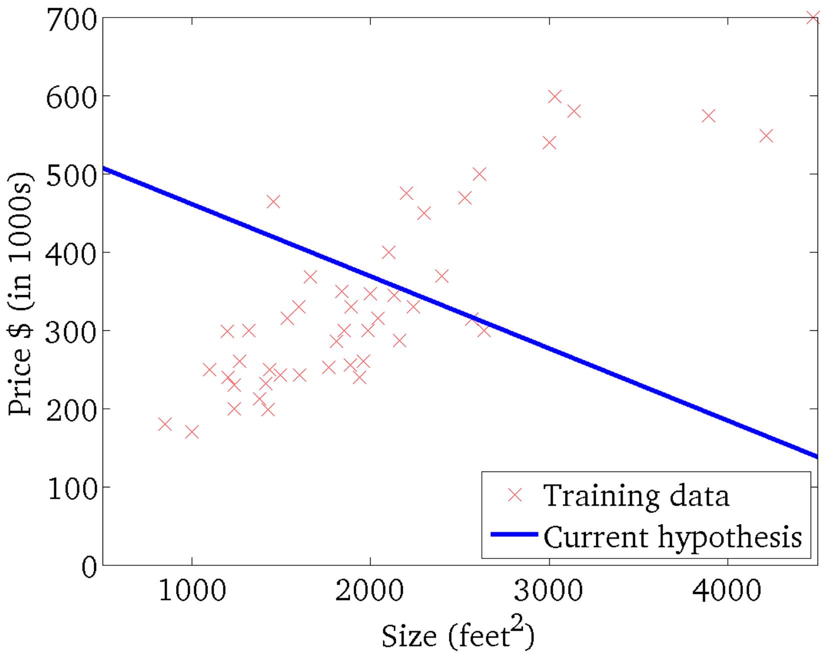
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



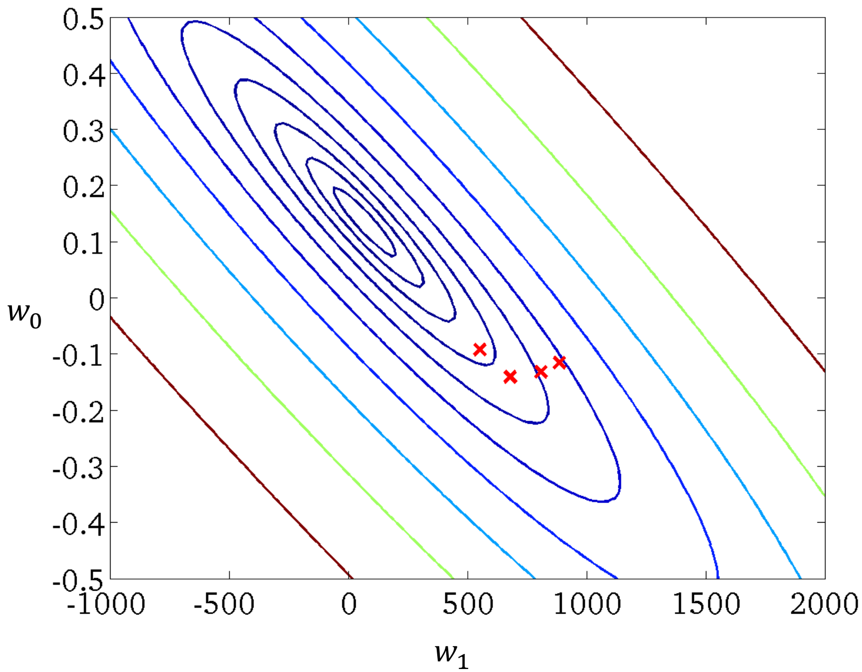
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)})\right)$



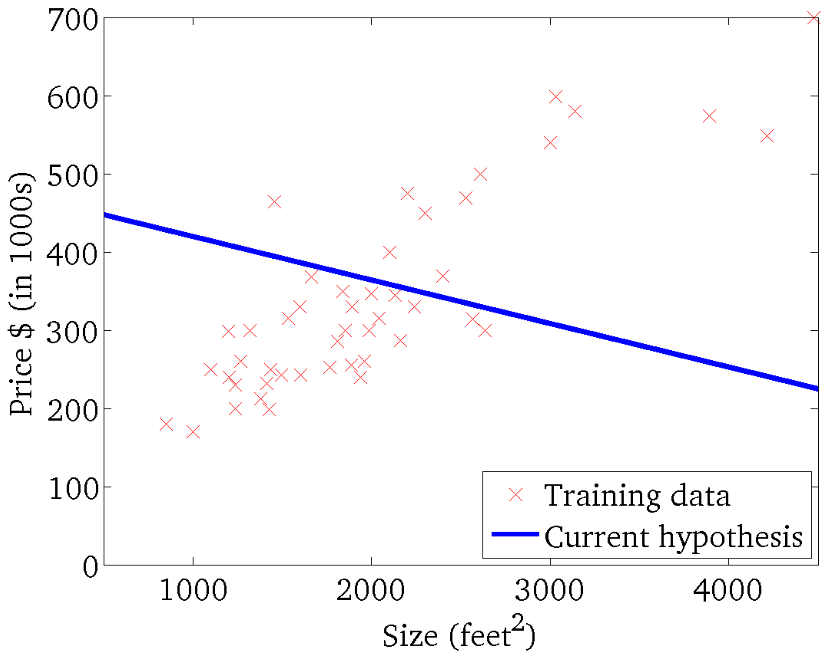
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



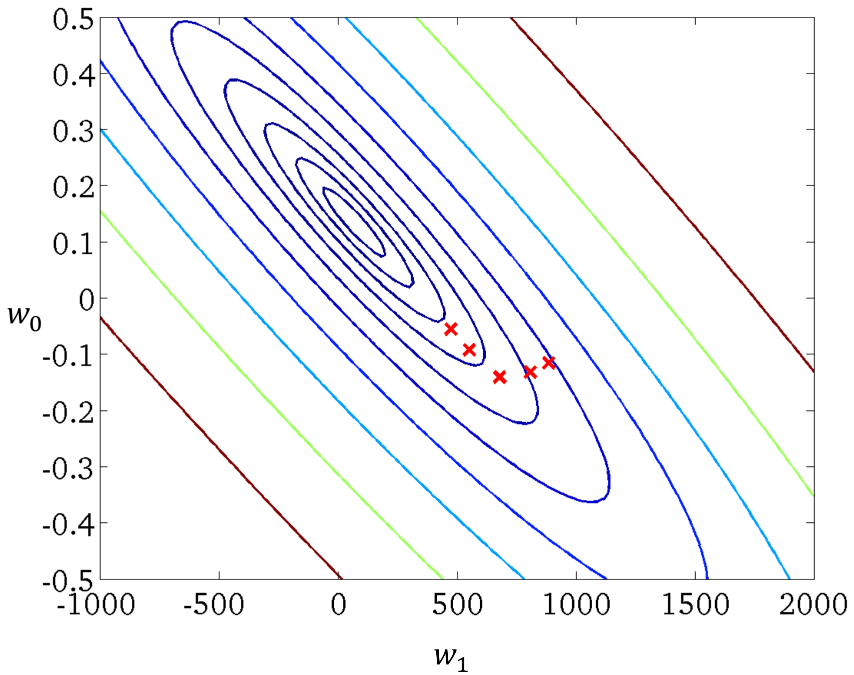
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)}\right)$



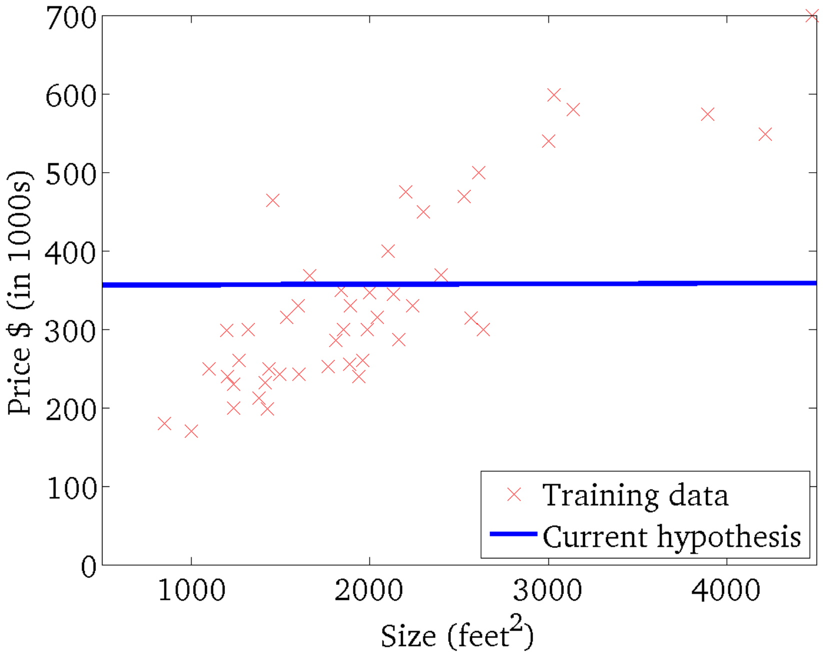
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



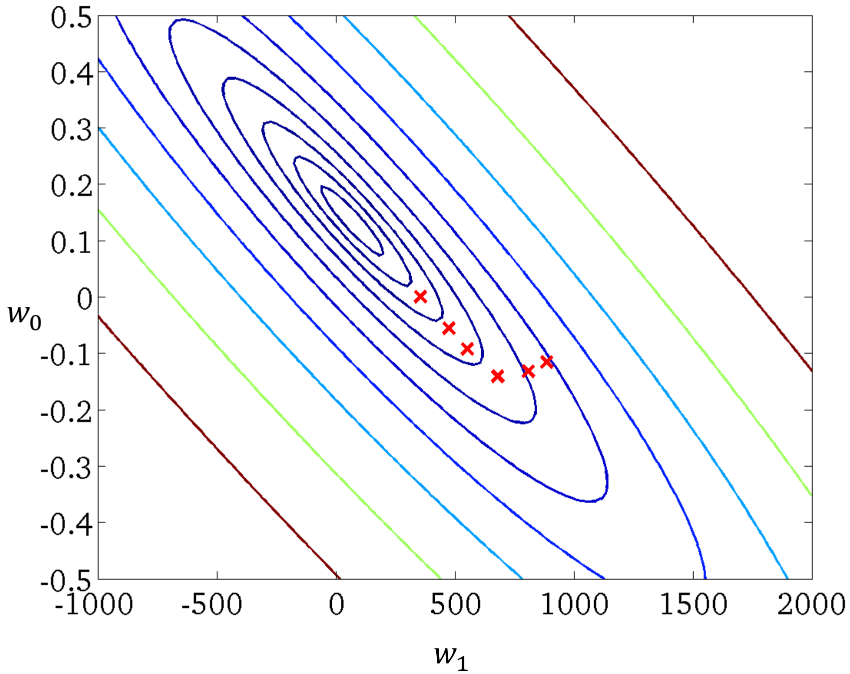
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)}\right)$



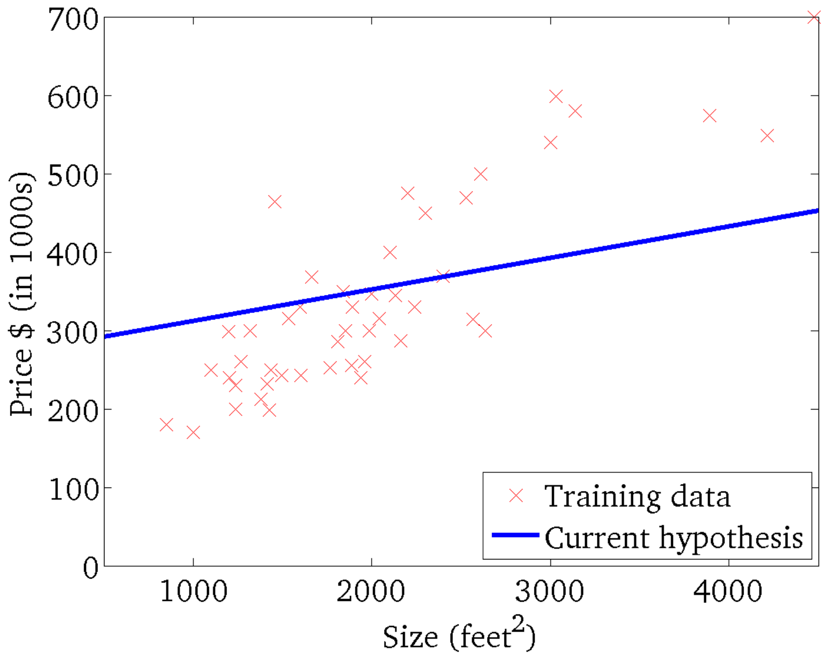
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



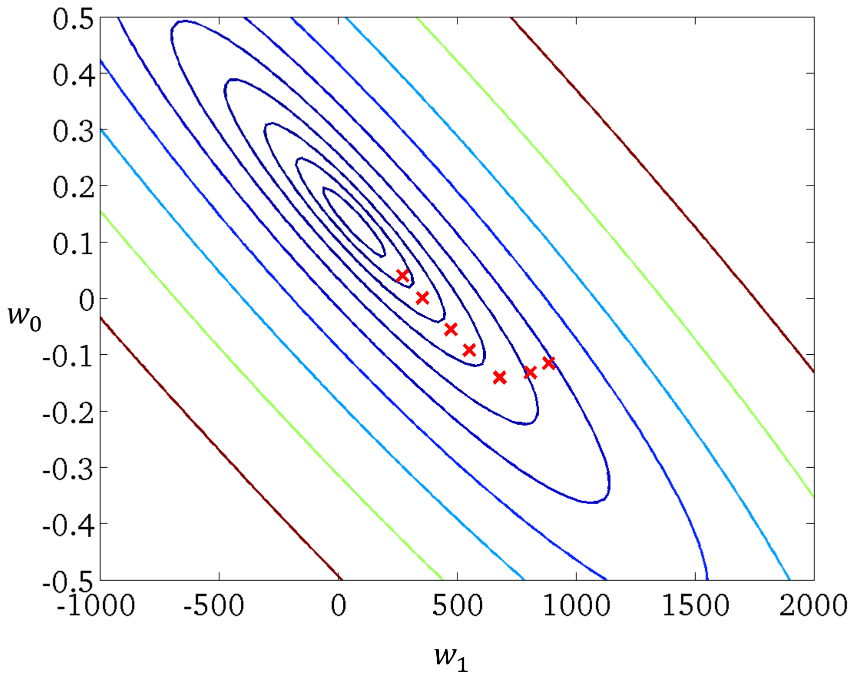
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)})\right)$



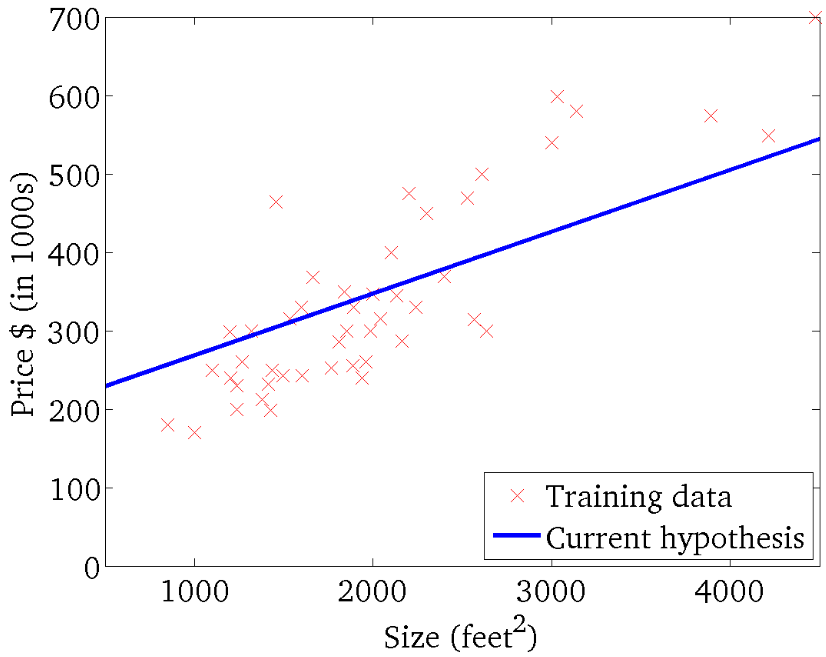
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



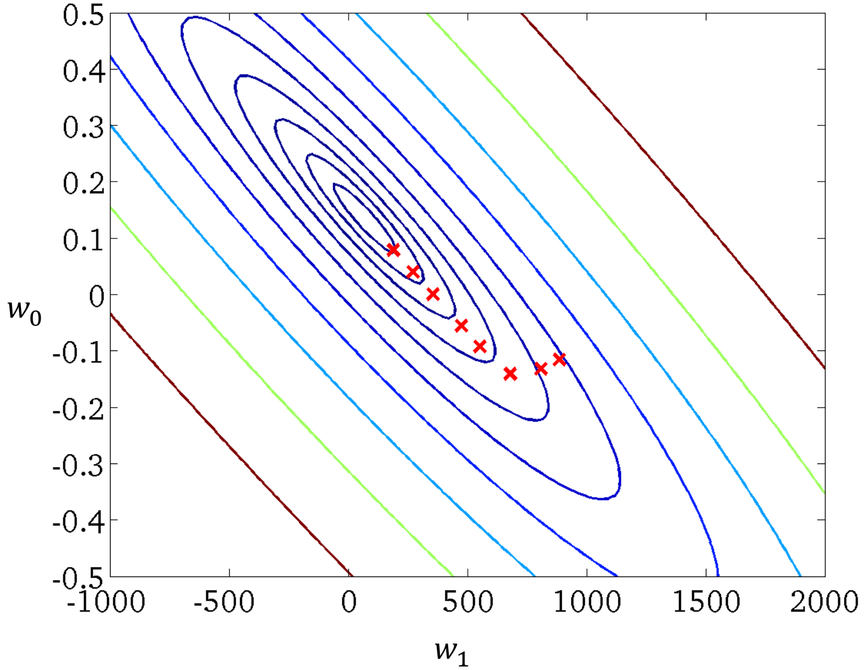
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)}\right)$



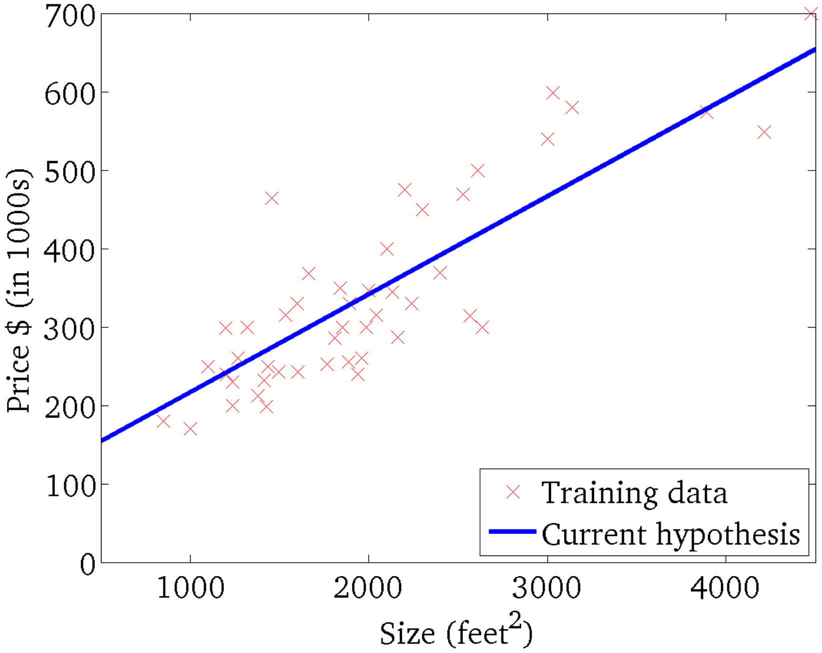
The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



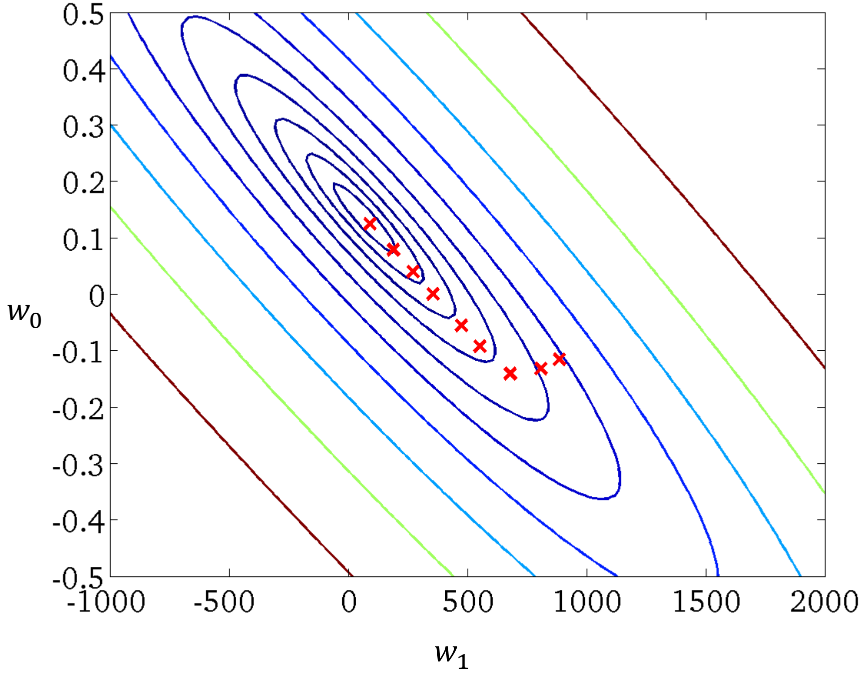
The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)})\right)$



The function $f(x) = w_0^{(t)} + w_1^{(t)} x$



The contours of $L(w_0, w_1)$ and the track of $\left((w_0^{(t)}, w_1^{(t)}\right)$



Stochastic Gradient Descent

- The GD algorithm need to evaluate the gradient of loss w.r.t. model parameters $\boldsymbol{w}$ *at every iteration*

- Generally, the gradient takes the form

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial \ell\left(\boldsymbol{w}, \boldsymbol{x}^{(i)}, y^{(i)}\right)}{\partial \boldsymbol{w}}$$

- Every iteration requires computing the gradient for all *data samples in the training dataset*

The complexity would be extremely high for large datasets

- To reduce the complexity, we can estimate the gradient $\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}}$ using a small portion of the dataset, *i.e. mini-batch*

- How to obtain the mini-batches?
  - Reshuffling
  - Segmenting

波动不可太大，会导致 diverge

batchSize = 4

	first batch				second batch				leftovers		
sequence[]	2	4	8	7	0	9	5	1	6	3	Length = 10
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	

- Update:

A noisy estimate to the true gradient

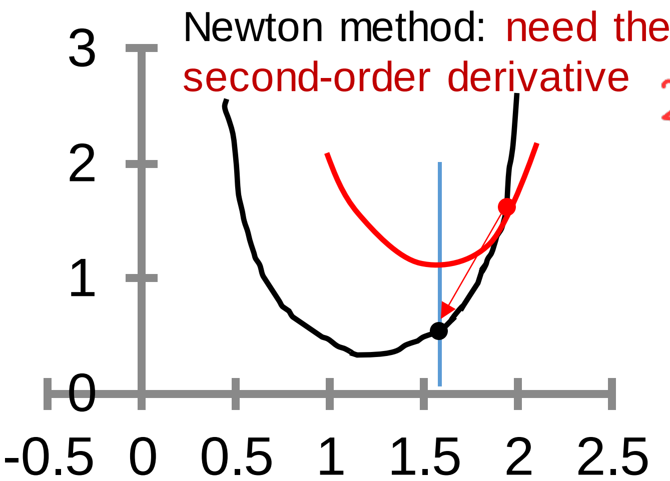
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + r \cdot \boxed{\frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \frac{\partial \ell\left(\boldsymbol{w}, \boldsymbol{x}^{(i)}, y^{(i)}\right)}{\partial \boldsymbol{w}}}$$

where $\mathcal{B}_t$ is a mini-batch of the dataset at the t-th iteration

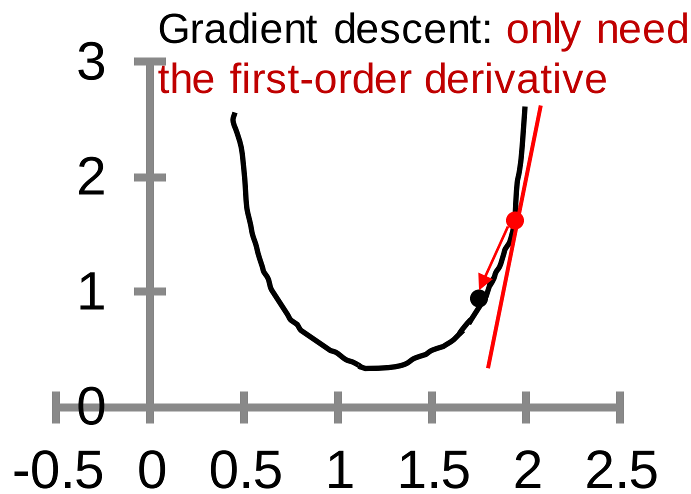
Other Optimization Methods

- There also exist many other optimization methods

  1) Newton method 牛顿法



2阶导数



Advantages

- No need to manually choose the learning rate
- Faster convergence rate

Disadvantages

- More expensive

2) Quasi-Newton methods

3) Conjugate gradient method

4) Coordinated descent method

⋮

These methods generally converge faster than the gradient method, but are more computationally expensive