Machine Learning Review

M. Soleymani

Sharif University of Technology

Fall 2017

Some slides have been adapted from Fei Fei Li lectures, cs231n, Stanford 2017

Types of ML problems

- <u>Supervised learning</u> (regression, classification)
  - predicting a target variable for which we get to see examples.
- <u>Unsupervised learning</u>
  - revealing structure in the observed data
- <u>Reinforcement learning</u>
  - partial (indirect) feedback, no explicit guidance
  - Given rewards for a sequence of moves to learn a policy and utility functions

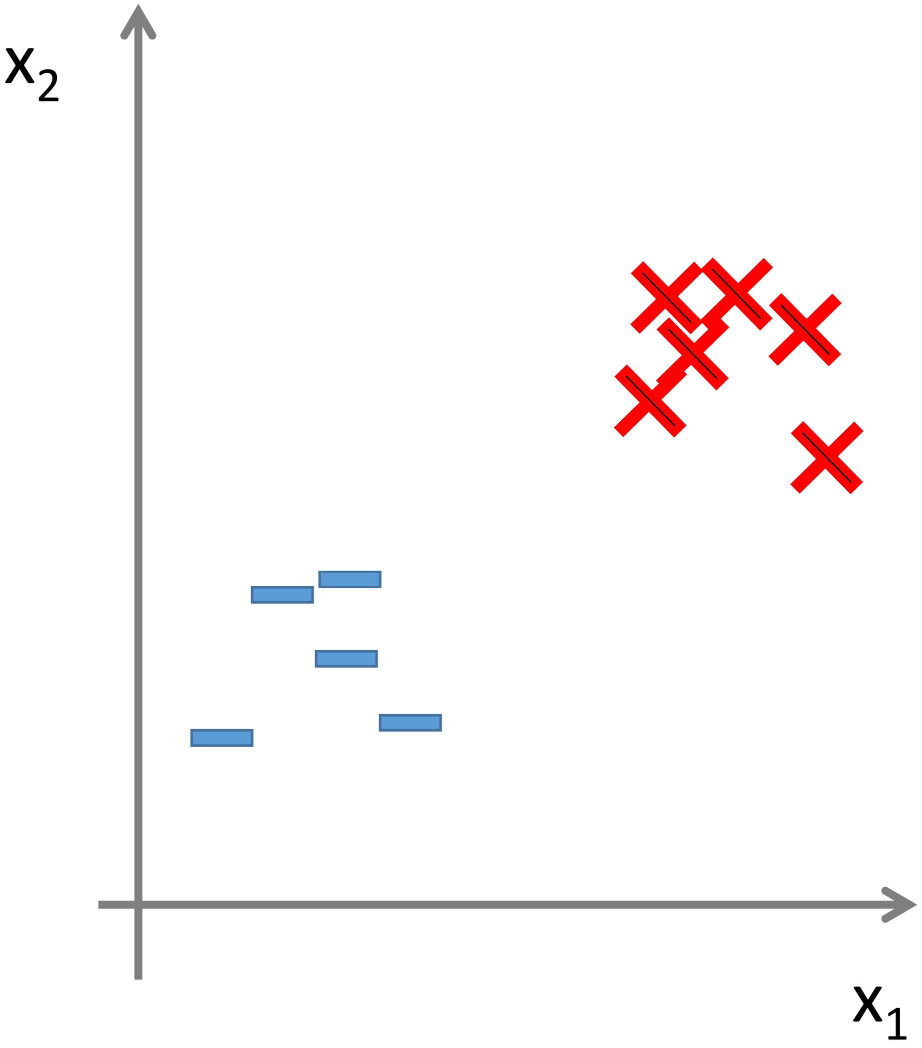
Components of (Supervised) Learning

- Unknown target function: $f: \mathcal{X} \rightarrow \mathcal{Y}$
  - Input space: $\mathcal{X}$
  - Output space: $\mathcal{Y}$

- Training data: $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$

- We use **training set** to find the function that can also predict output on the **test set**

Training data: Example



Training data

x_1	x_2	y
0.9	2.3	1
3.5	2.6	1
2.6	3.3	1
2.7	4.1	1
1.8	3.9	1
6.5	6.8	-1
7.2	7.5	-1
7.9	8.3	-1
6.9	8.3	-1
8.8	7.9	-1
9.1	6.2	-1

Supervised Learning: Regression vs. Classification

- Supervised Learning
  - **Regression**: predict a continuous target variable
    - E.g., $y \in [0,1]$
  - **Classification**: predict a discrete target variable
    - E.g.,$y \in \{1,2, \dots, C\}$

Regression Example

- Housing price prediction

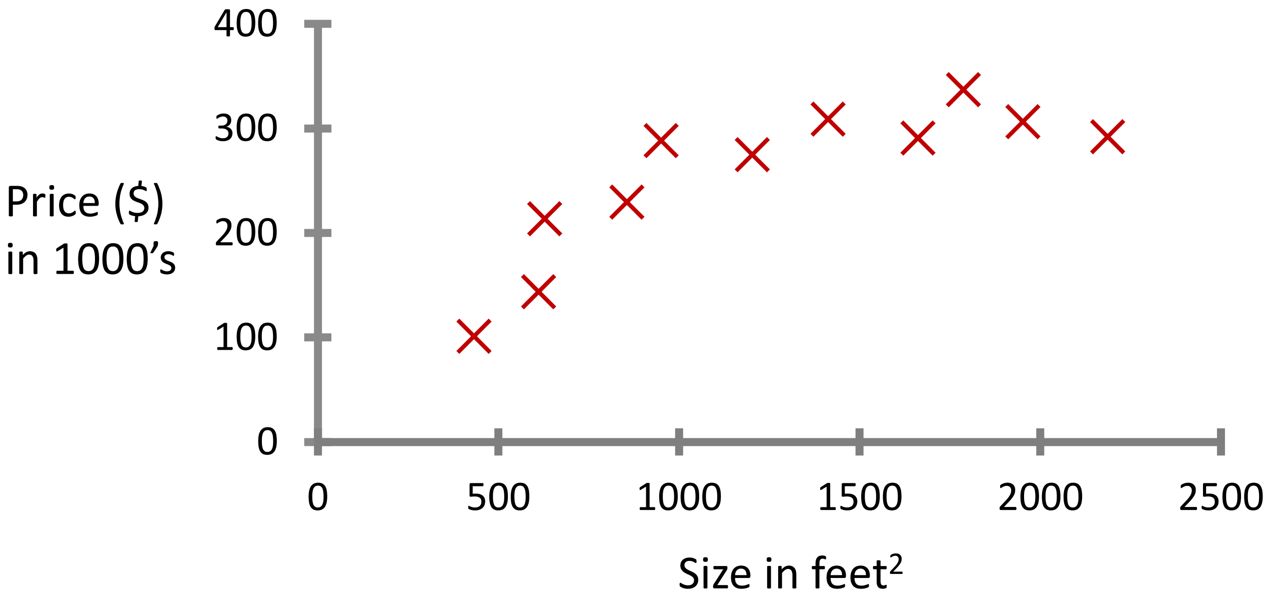


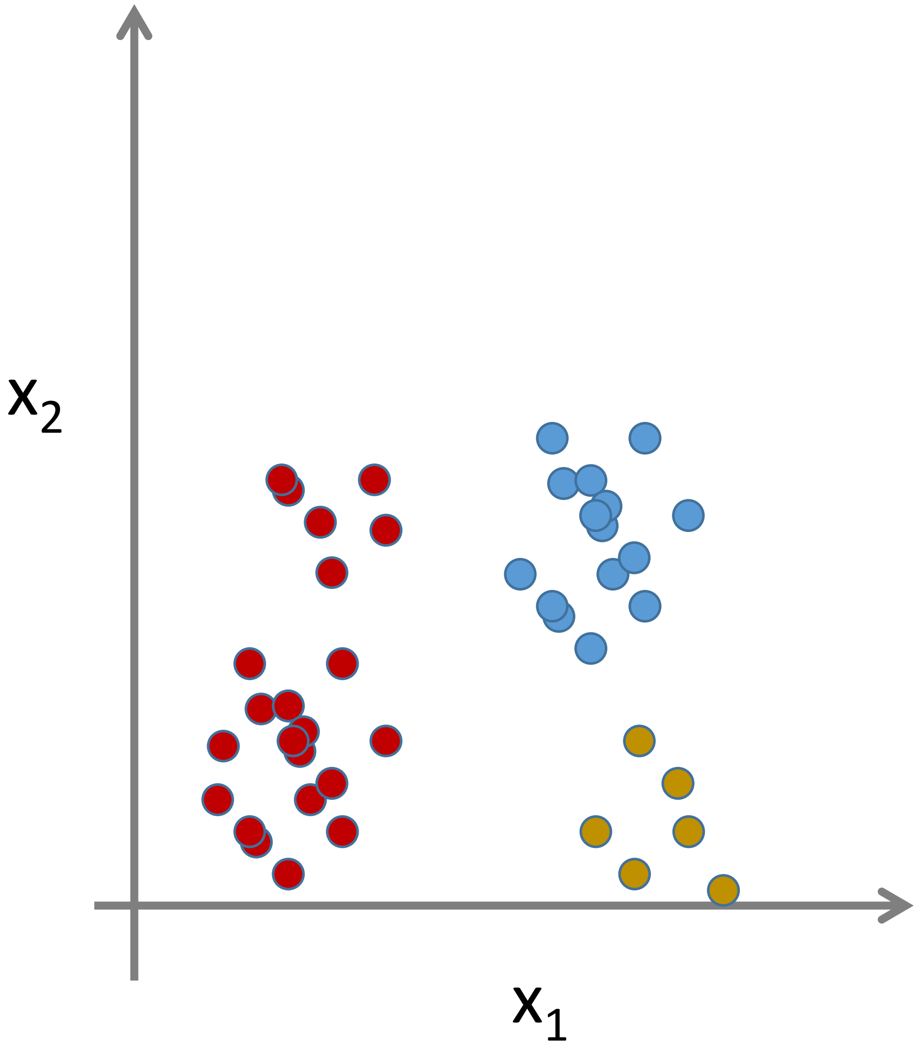
Figure adopted from slides of Andrew Ng,
Machine Learning course, Stanford.

Supervised Learning vs. Unsupervised Learning

- Supervised learning
  - Given: Training set
    - labeled set of N input-output pairs $D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - Goal: learning a mapping from $\boldsymbol{x}$ to y

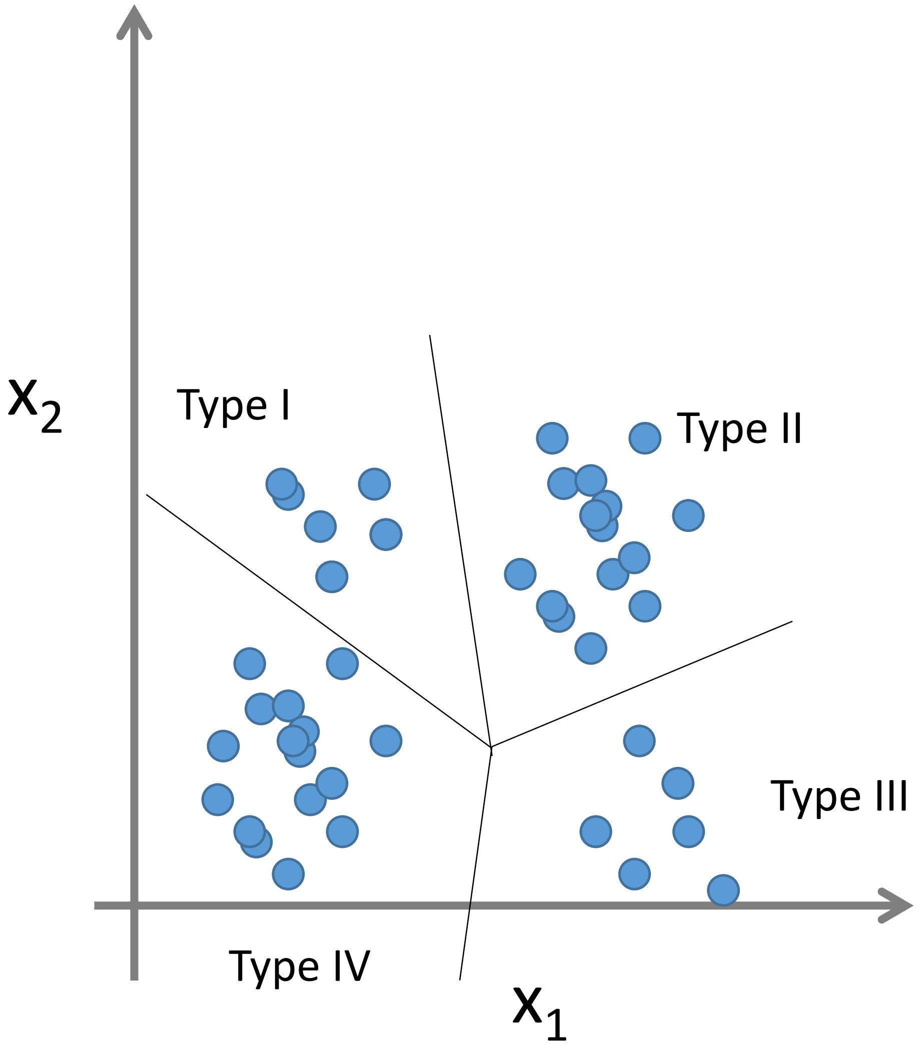
- Unsupervised learning
  - Given: Training set
    - $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$
  - Goal: find groups or structures in the data
    - Discover the intrinsic structure in the data

Supervised Learning: Samples



Classification

Unsupervised Learning: Samples



Clustering

Reinforcement Learning

- Provides only an indication as to whether an action is correct or not

Data in supervised learning:

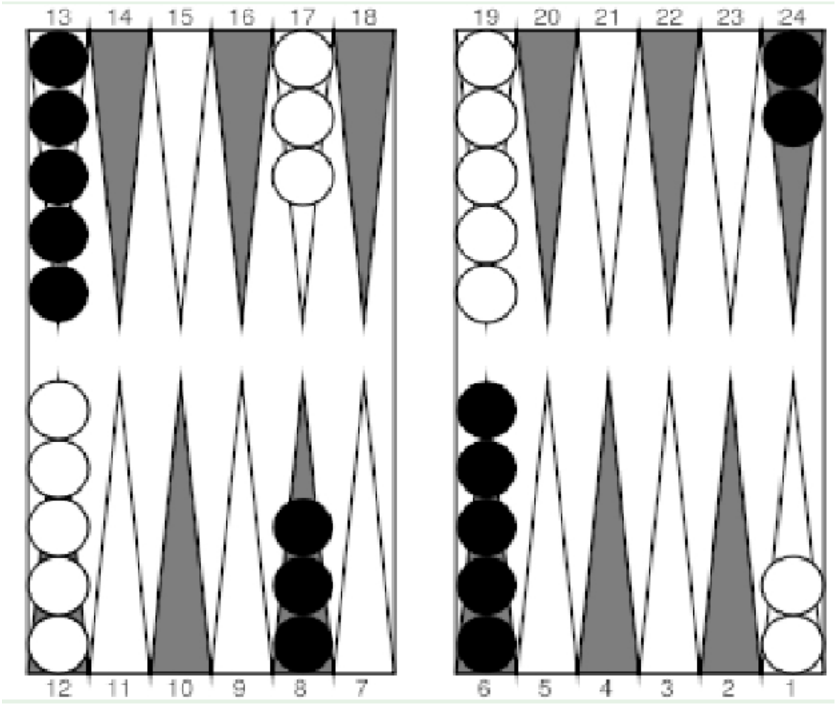
(input, correct output)

Data in Reinforcement Learning:

(input, some output, a grade of reward for this output)

Reinforcement Learning

- Typically, we need to get a sequence of decisions
  - it is usually assumed that reward signals refer to the entire sequence



Components of (Supervised) Learning

- Unknown target function: $f: \mathcal{X} \to \mathcal{Y}$
  - Input space: $\mathcal{X}$
  - Output space: $\mathcal{Y}$

- Training data: $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)$

- We use **training set** to find the function that can also predict output on the **test set**

Generalization

- We don’t intend to memorize data but need to figure out the pattern.
- A core objective of learning is to generalize from the experience.
  - Generalization: ability of a learning algorithm to perform accurately on new, unseen examples after having experienced.

(Typical) Steps of solving supervised learning problem

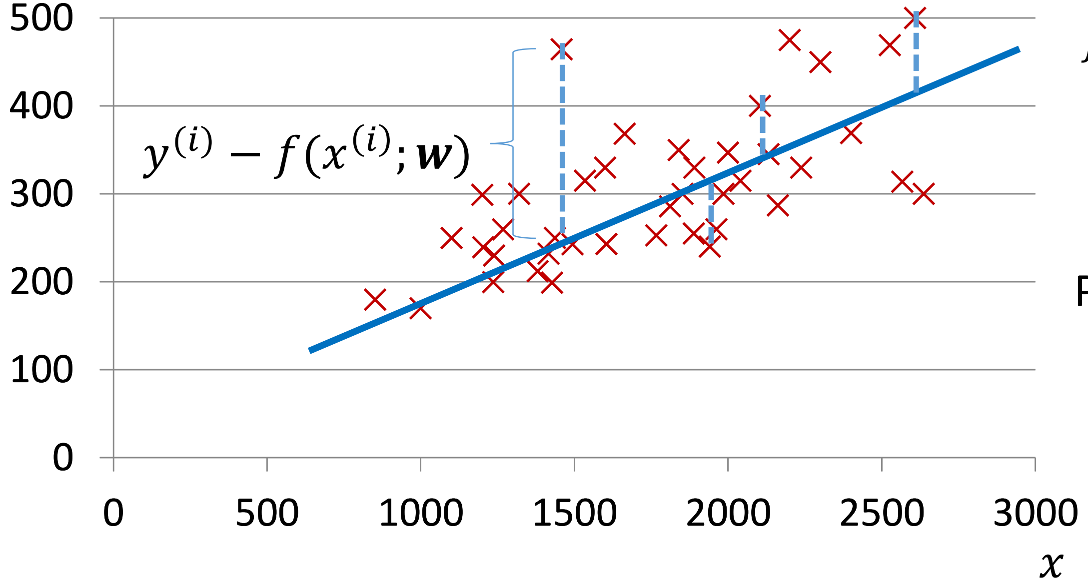
- Select the hypothesis space
  - A class of parametric models that map each input vector, **x**, into a predicted output y.

- Define a **loss function** that quantifies how much undesirable is each parameter vector across the training data.

- Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

- Evaluate the obtained model

Linear regression: square error loss function



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

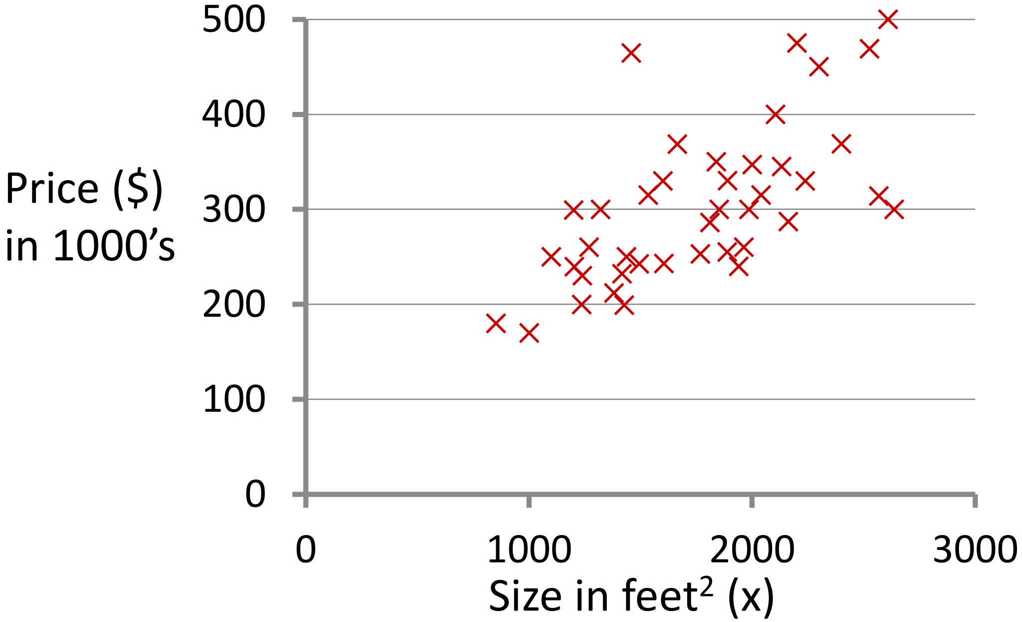
$$\boldsymbol{w} = [w_0, w_1]$$

Parameters that be found

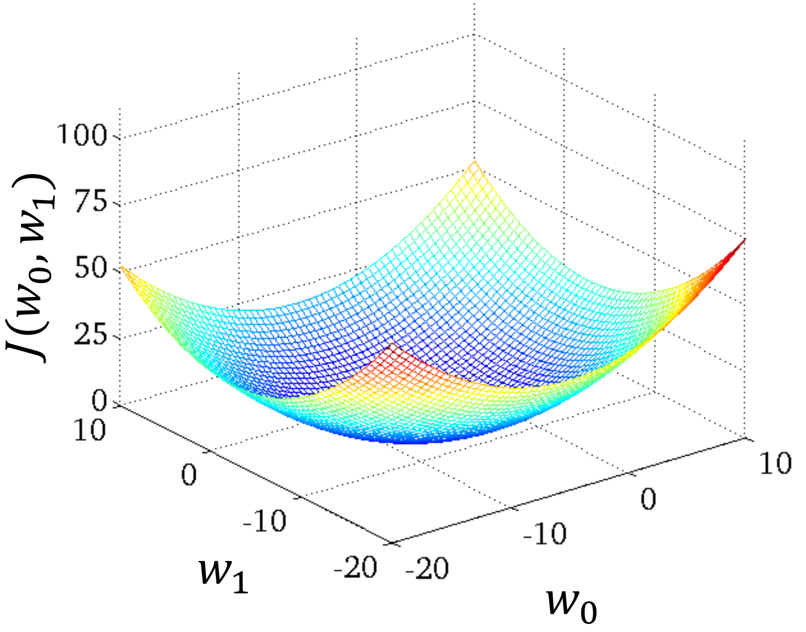
Cost function:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(y^{(i)} - (w_0 + w_1 x^{(i)})\right)^2$$

Cost function: example



$J(\boldsymbol{w})$
(function of the parameters w_0, w_1)

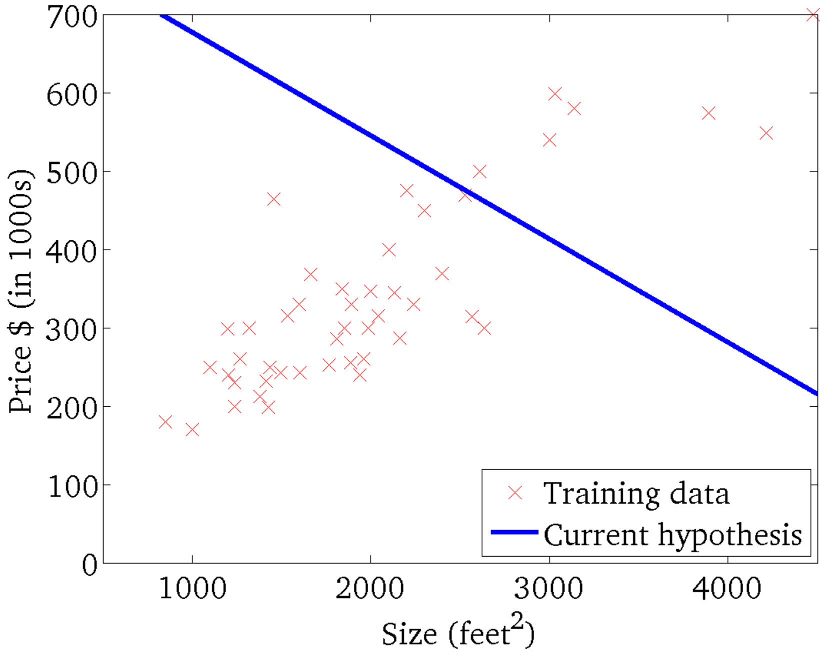


This example has been adapted from: Prof. Andrew Ng's slides

Cost function: example

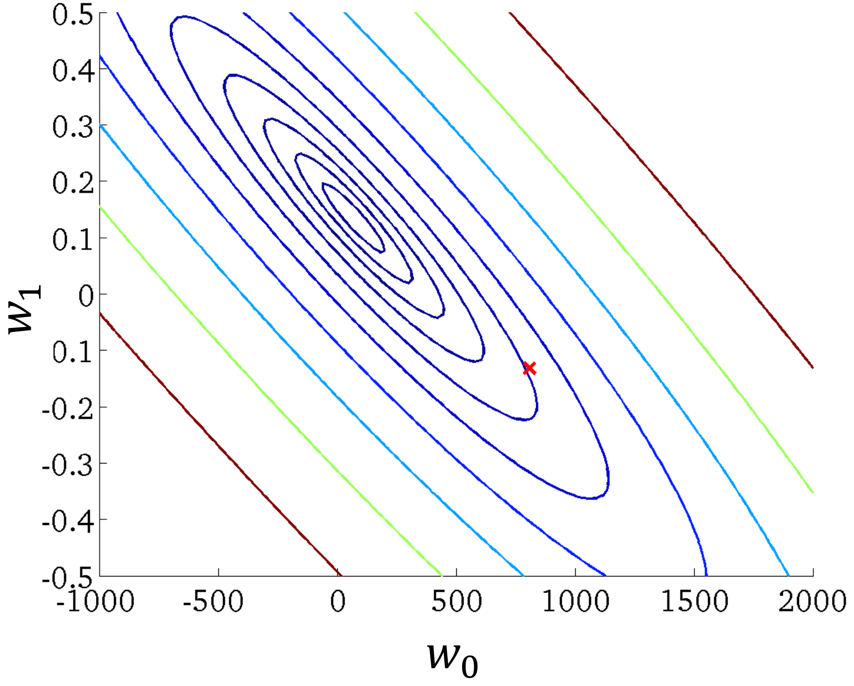
$$f(x; w_0, w_1) = w_0 + w_1 x$$

(for fixed w_0, w_1, this is a function of x)



$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)

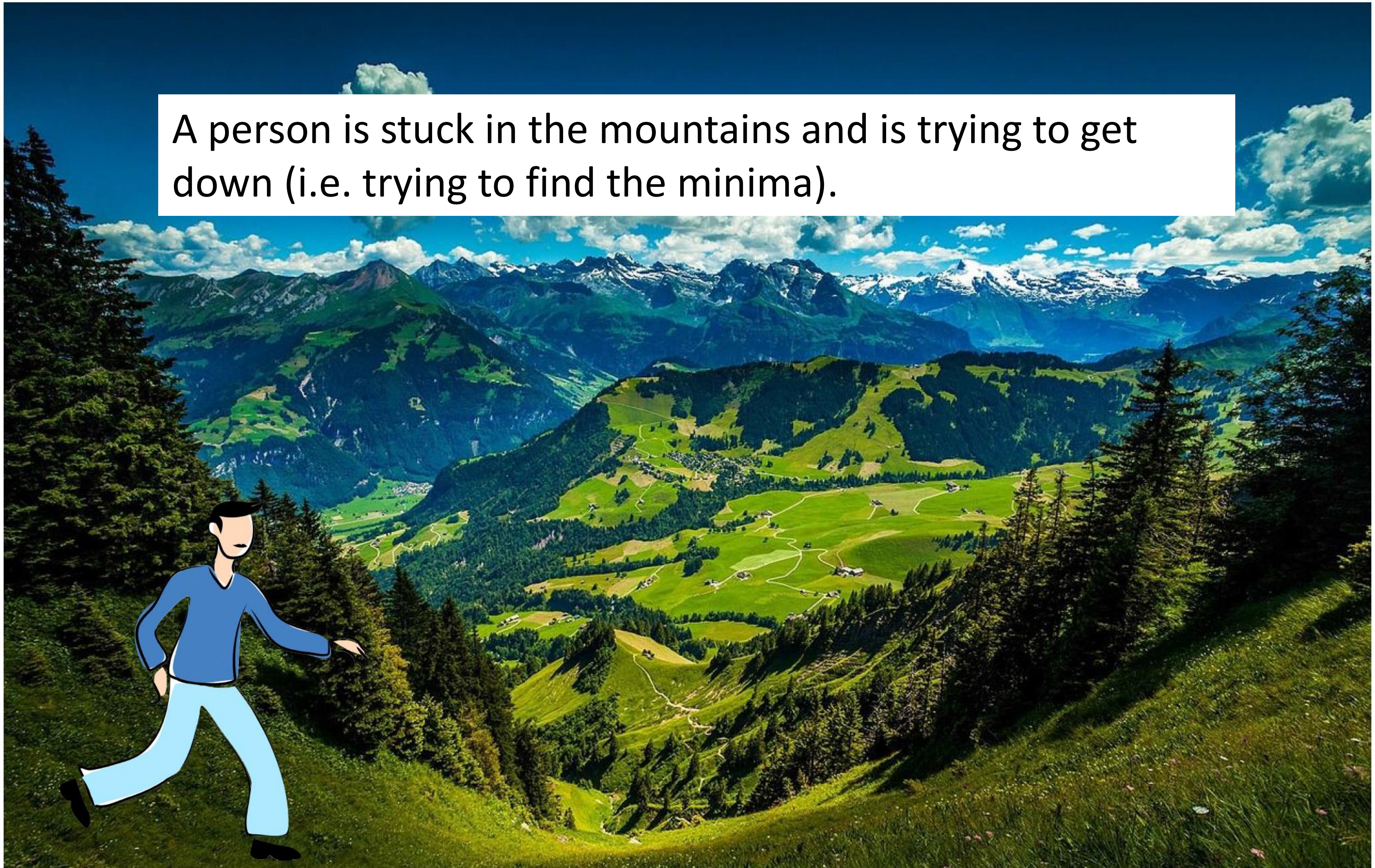


This example has been adapted from: Prof. Andrew Ng's slides

Review: Iterative optimization of cost function

- Cost function: $J(\boldsymbol{w})$
- Optimization problem: $\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$
- Steps:
  - Start from $\boldsymbol{w}^0$
  - Repeat
    - Update $\boldsymbol{w}^t$ to $\boldsymbol{w}^{t+1}$ in order to reduce J
    - $t \leftarrow t + 1$
  - until we hopefully end up at a minimum

How to optimize parameters?



Follow up the slope

How to compute the slope?

- In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

  - the slope of the error surface can be calculated by taking the derivative of the error function at that point

- In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

- The direction of steepest descent is the **negative gradient**

Gradient descent (or steepest descent)

- In each step, takes steps proportional to the negative of the gradient vector of the function at the current point $\boldsymbol{w}^t$:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \gamma_t \nabla J(\boldsymbol{w}^t)$$

  - $J(\boldsymbol{w})$ <u>decreases fastest</u> if one goes from $\boldsymbol{w}^t$ in the direction of $-\nabla J(\boldsymbol{w}^t)$
  - Assumption: $J(\boldsymbol{w})$ is defined and differentiable in a neighborhood of a point $\boldsymbol{w}^t$

Learning rate: The amount of time he travels before taking another measurement is the learning rate of the algorithm.

Gradient descent

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

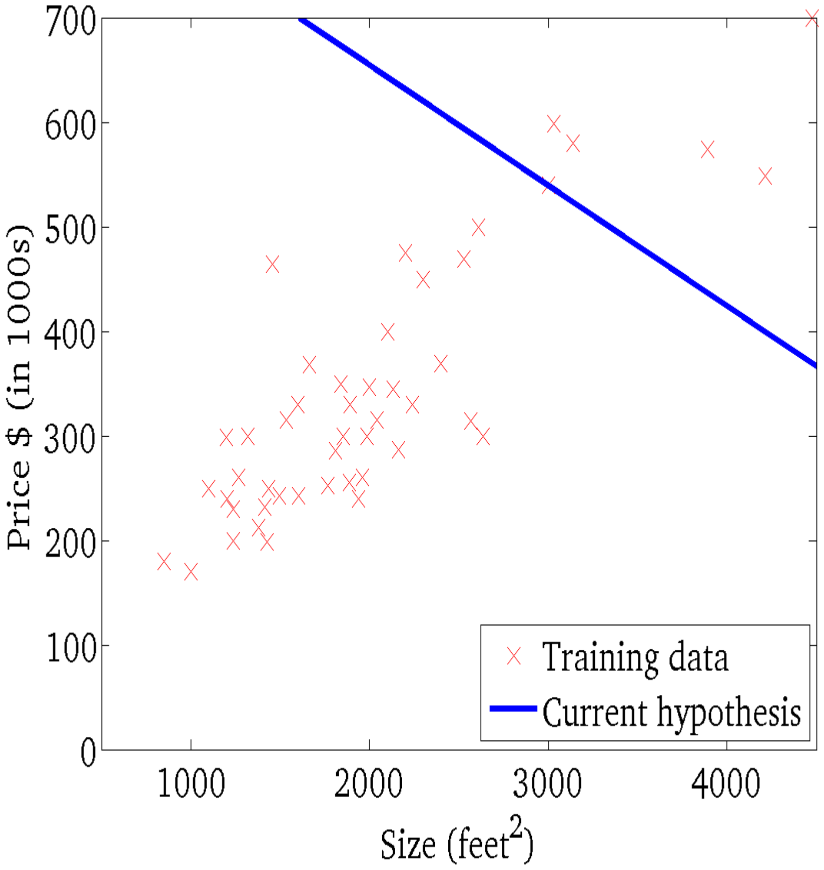
Step size
(Learning rate parameter)

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \eta \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{t})$$

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = [\frac{\partial J(\boldsymbol{w})}{\partial w_0}, \frac{\partial J(\boldsymbol{w})}{\partial w_2}, \ldots, \frac{\partial J(\boldsymbol{w})}{\partial w_d}]$$

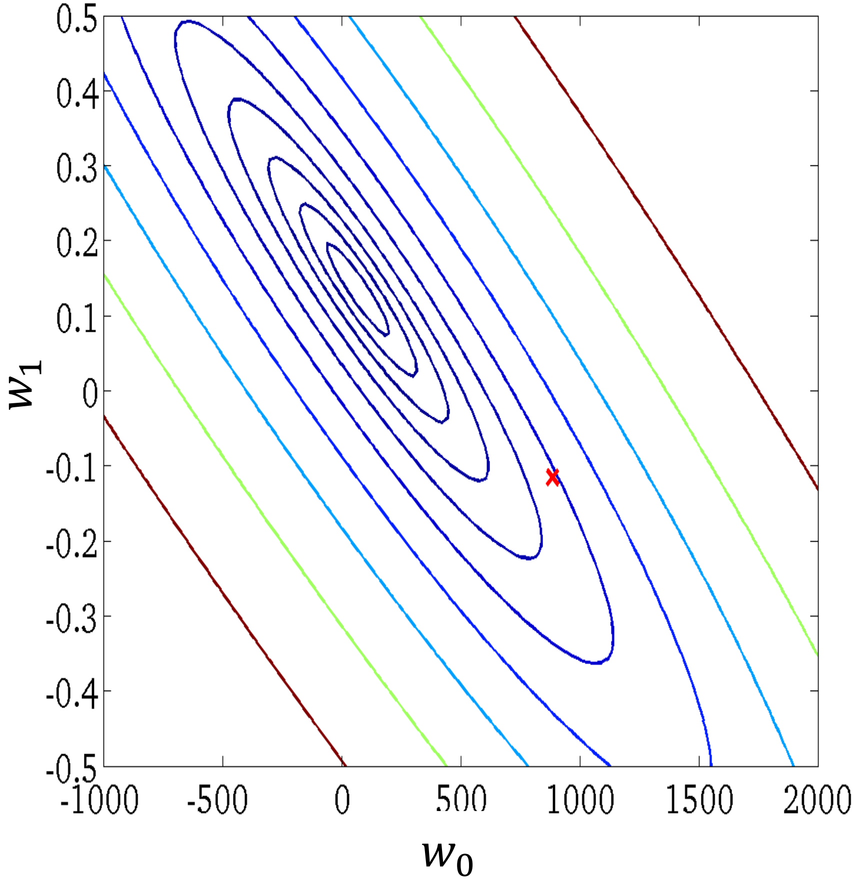
- If η is small enough, then $J(\boldsymbol{w}^{t+1}) \leq J(\boldsymbol{w}^{t})$.
- η can be allowed to change at every iteration as η_t.

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



$$J(w_0, w_1)$$

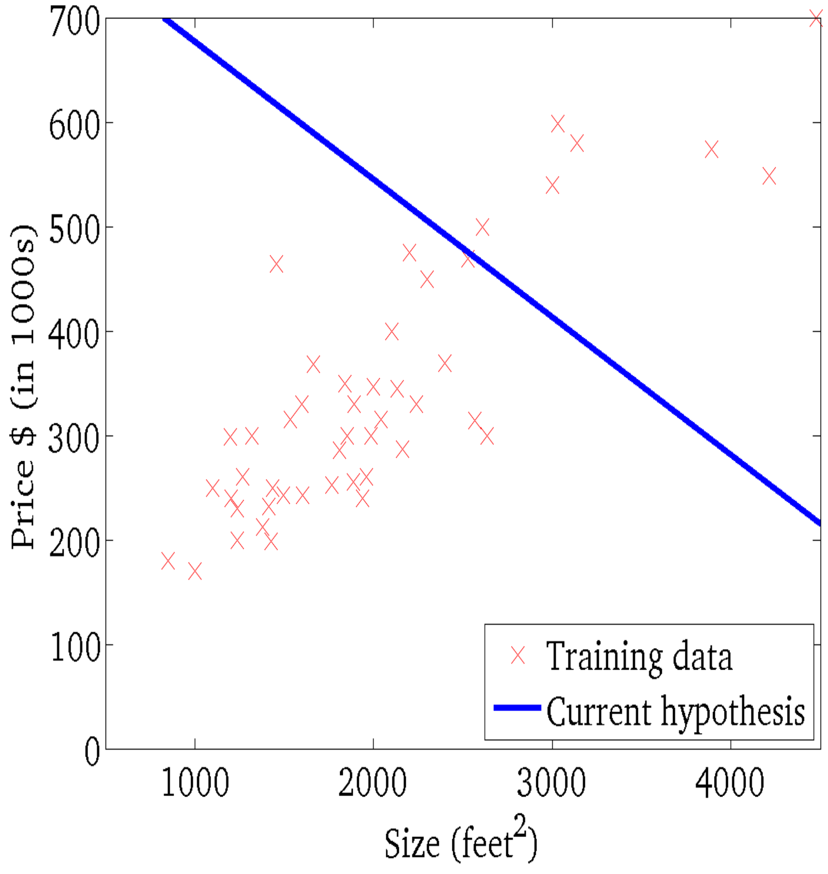
(function of the parameters w_0, w_1)



$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \eta \sum_{i=1}^{N} \left({\boldsymbol{w}^{t}}^{T} \boldsymbol{x}^{(i)} - y^{(i)} \right) \boldsymbol{x}^{(i)}$$

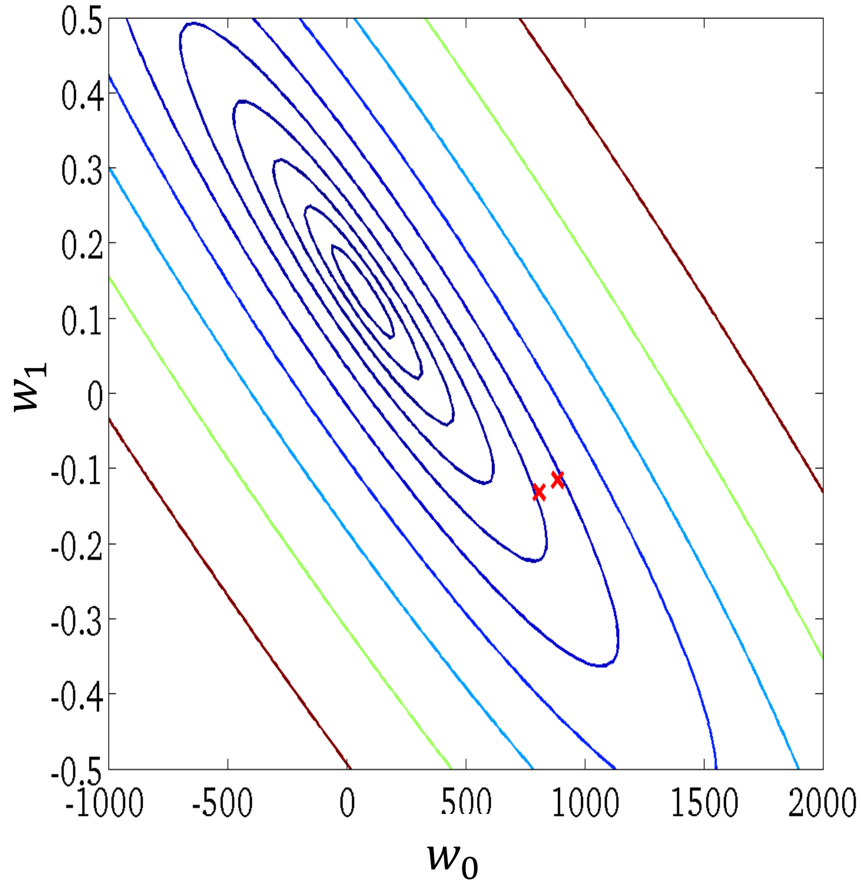
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



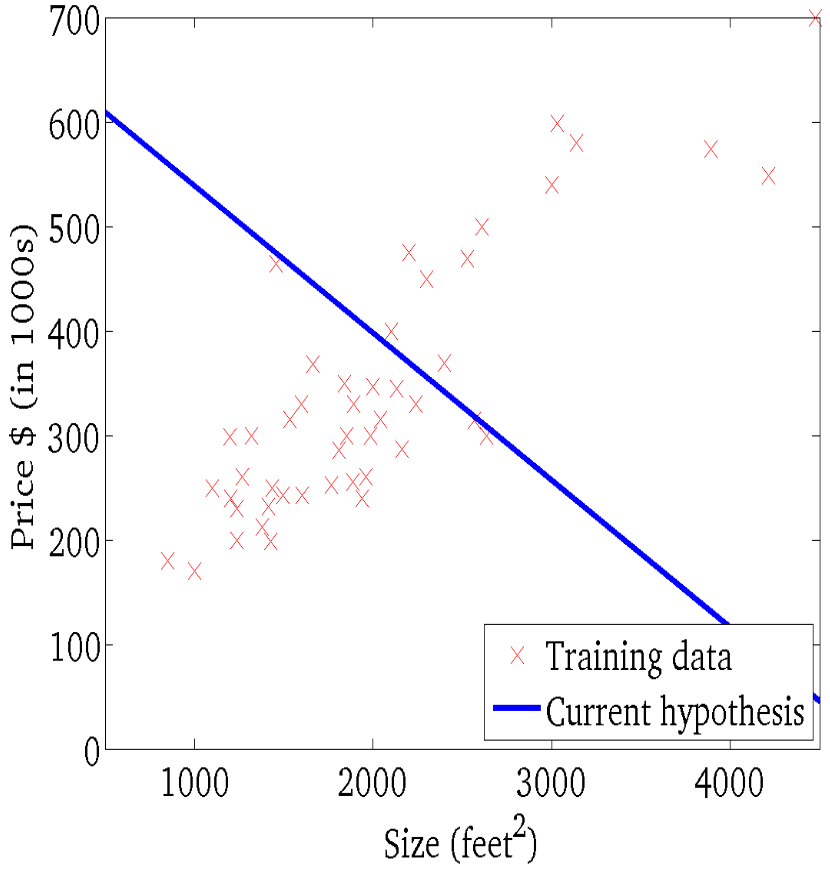
$J(w_0, w_1)$

(function of the parameters w_0, w_1)



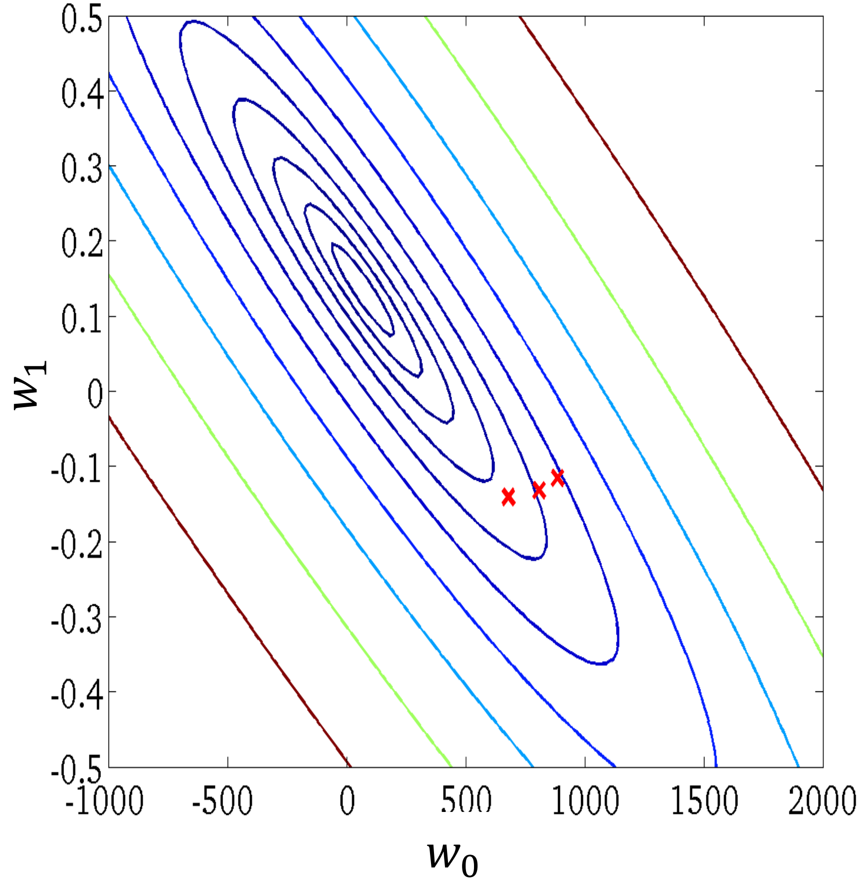
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



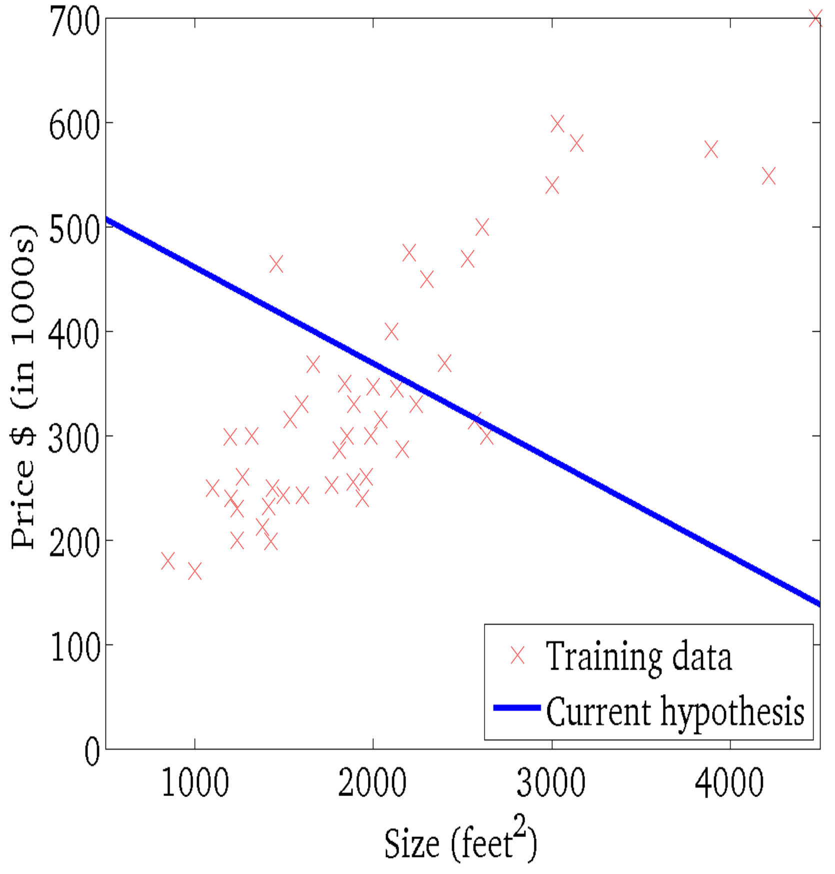
$J(w_0, w_1)$

(function of the parameters w_0, w_1)



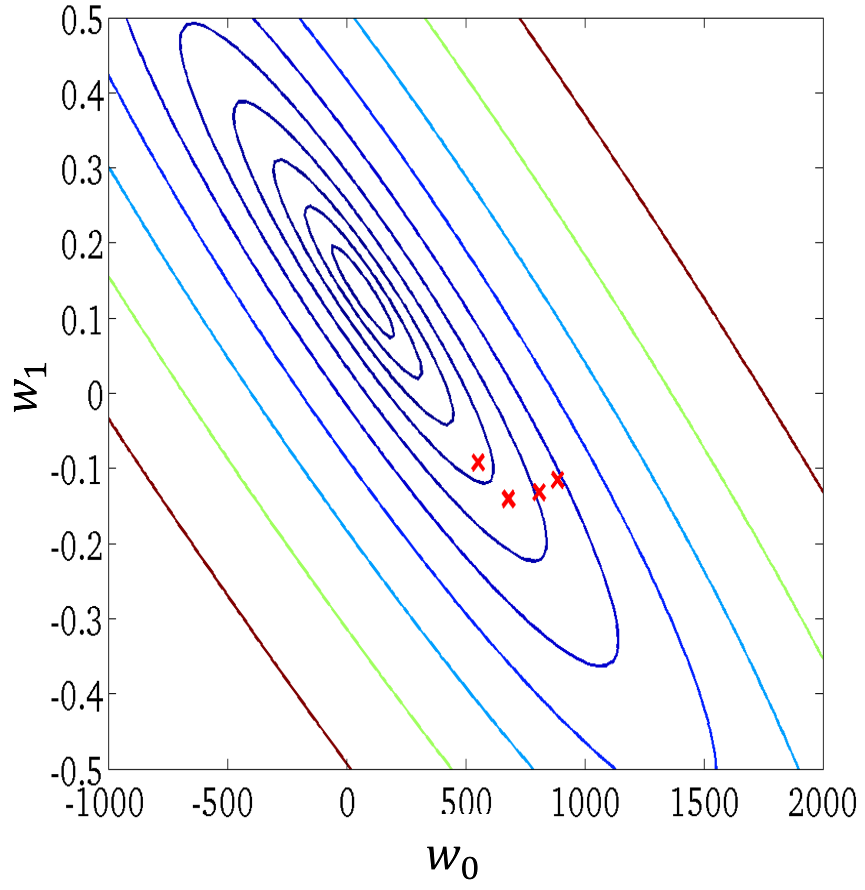
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



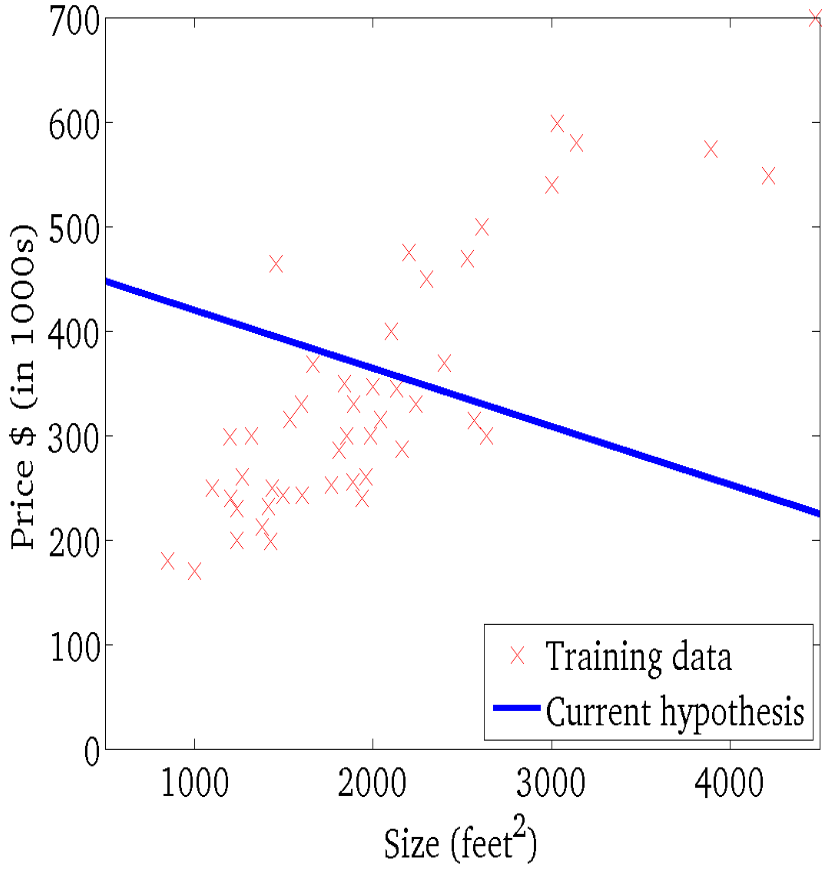
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



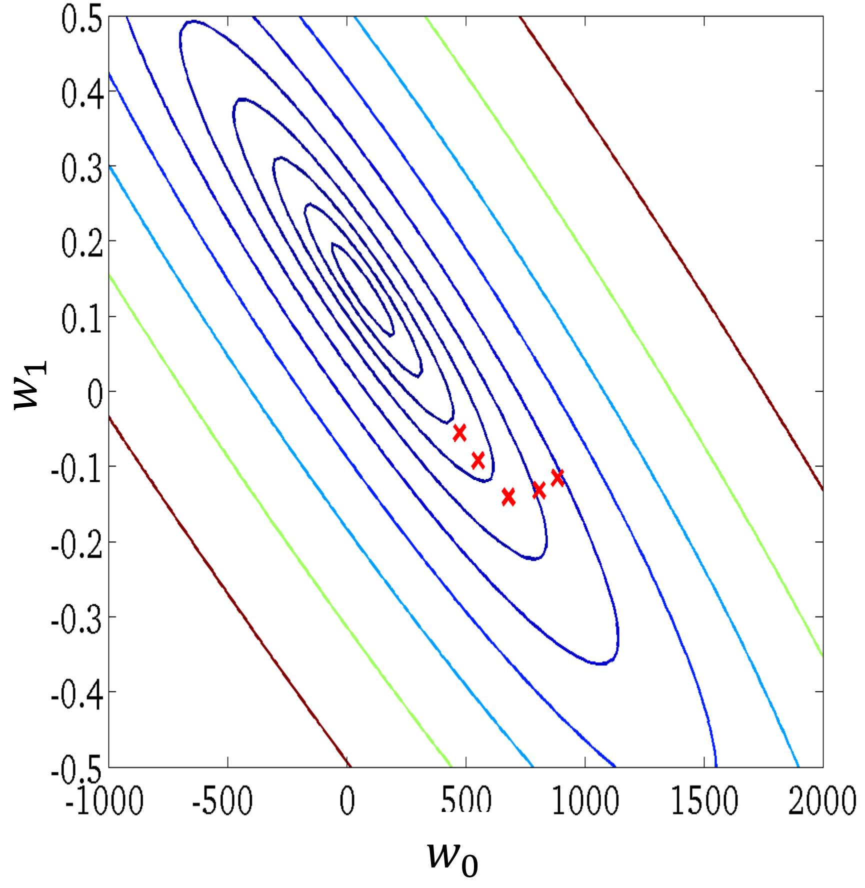
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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

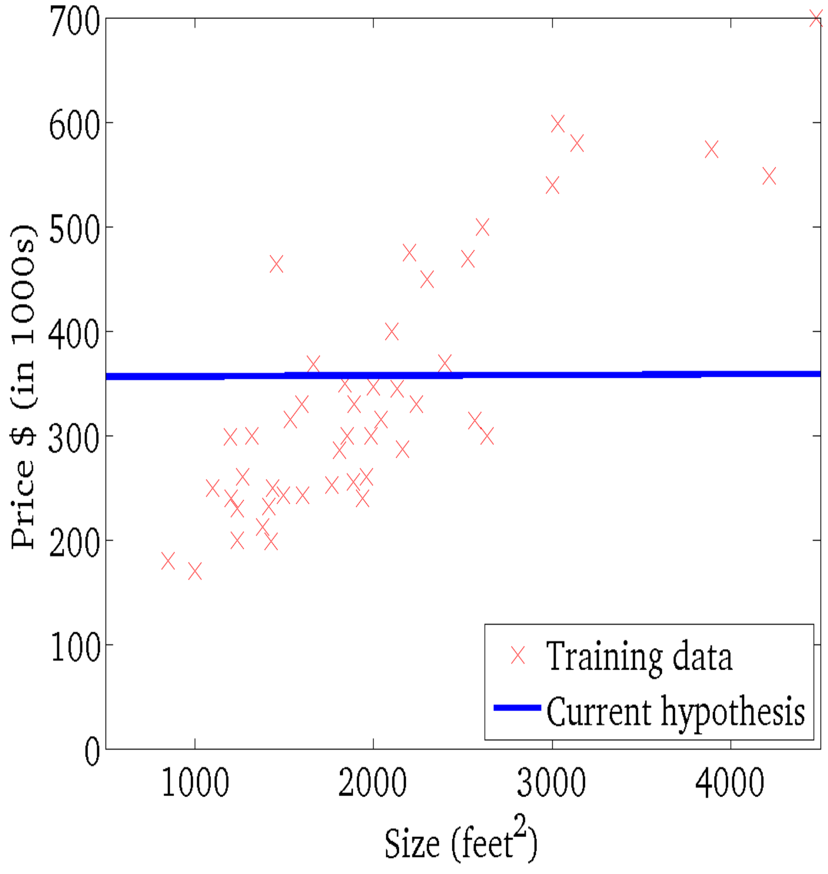


$J(w_0, w_1)$

(function of the parameters w_0, w_1)

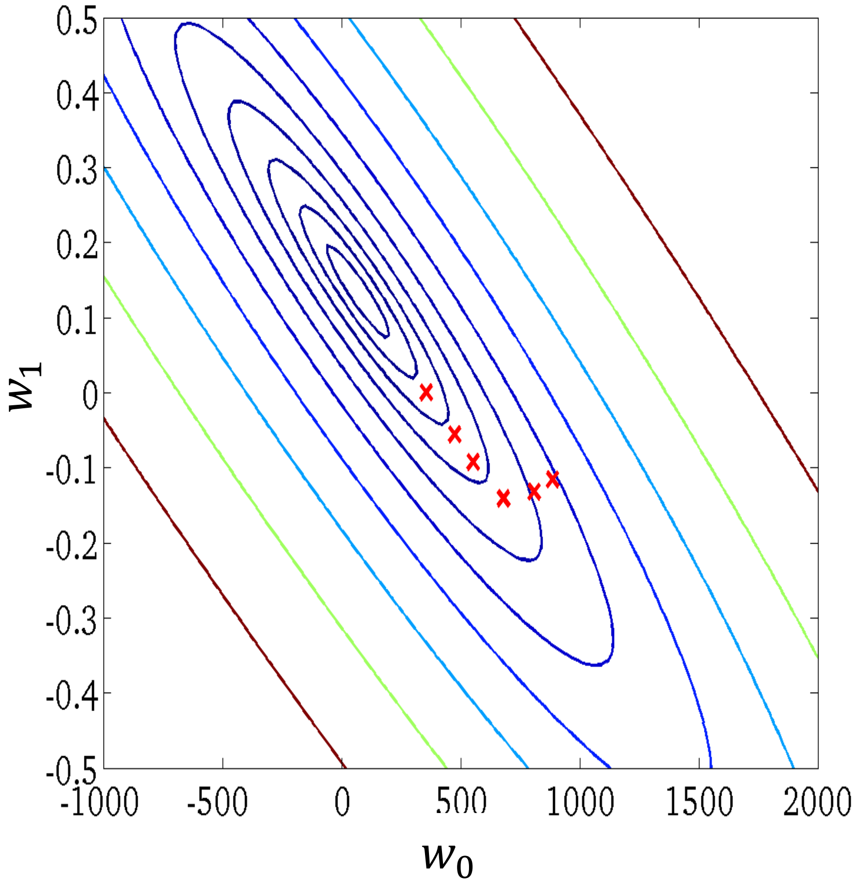


$$f(x; w_0, w_1) = w_0 + w_1 x$$

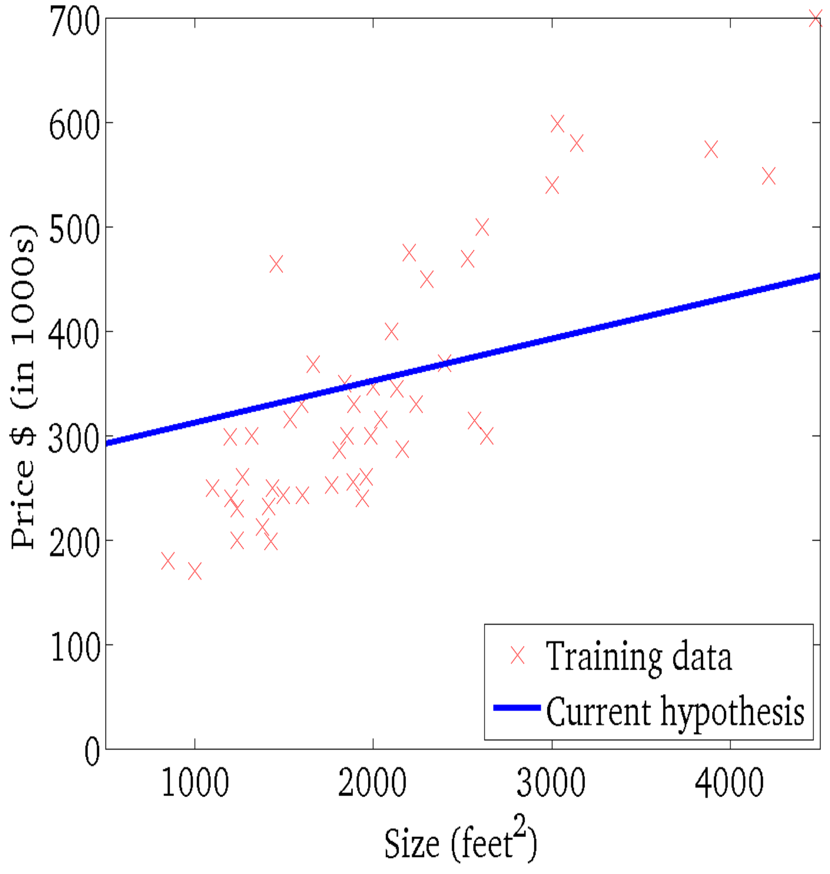


$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)

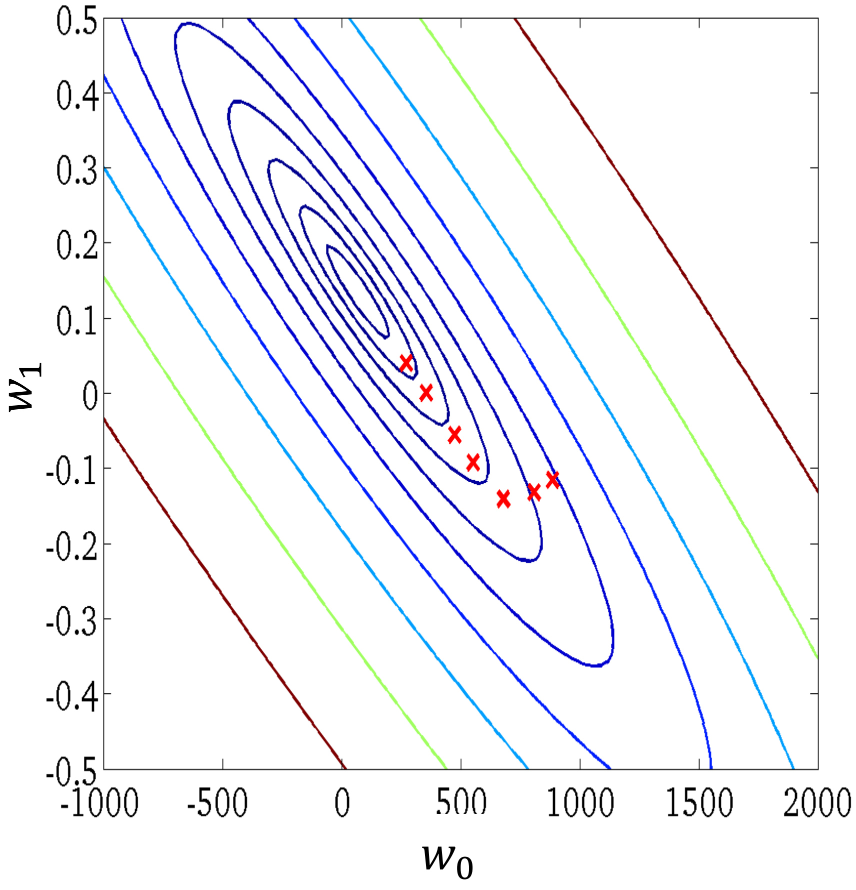


$$f(x; w_0, w_1) = w_0 + w_1 x$$



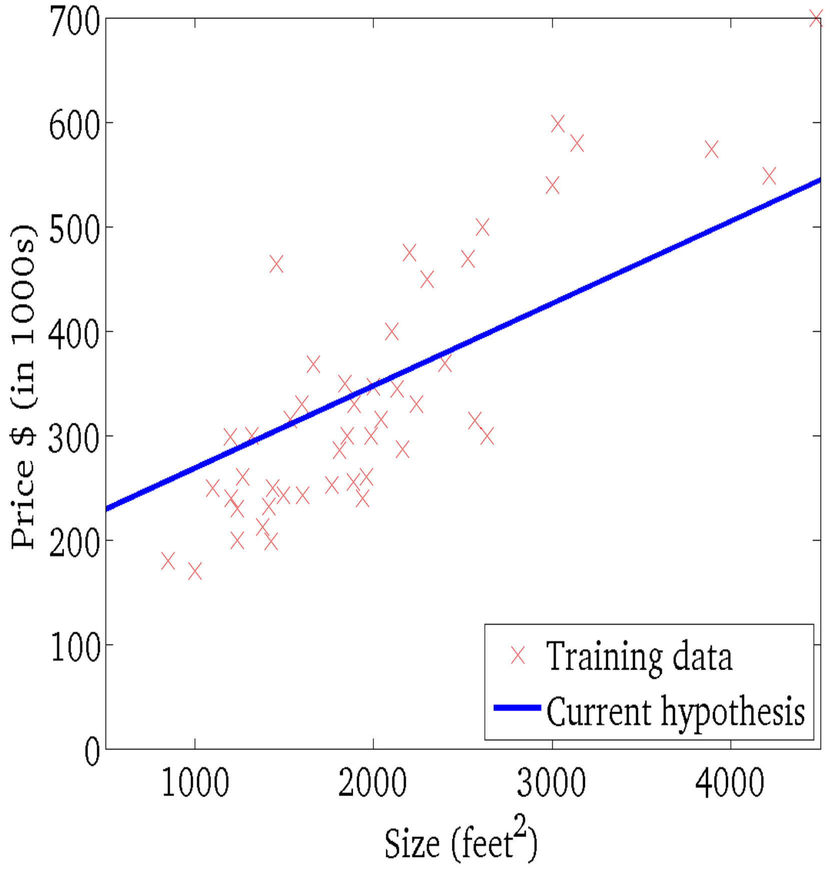
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



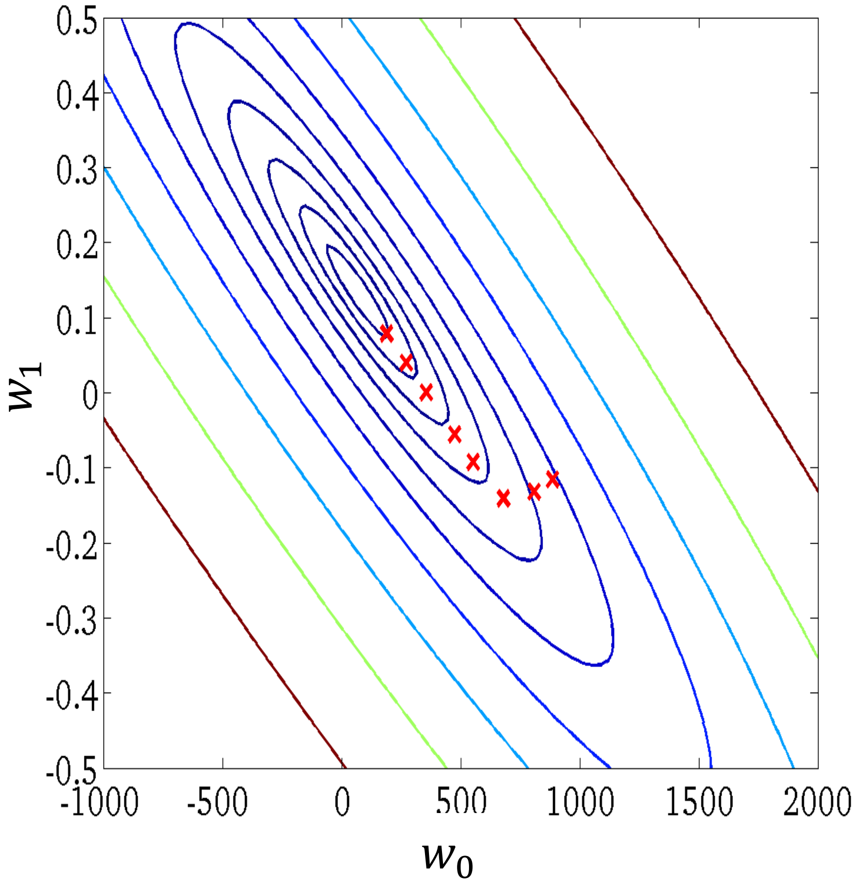
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



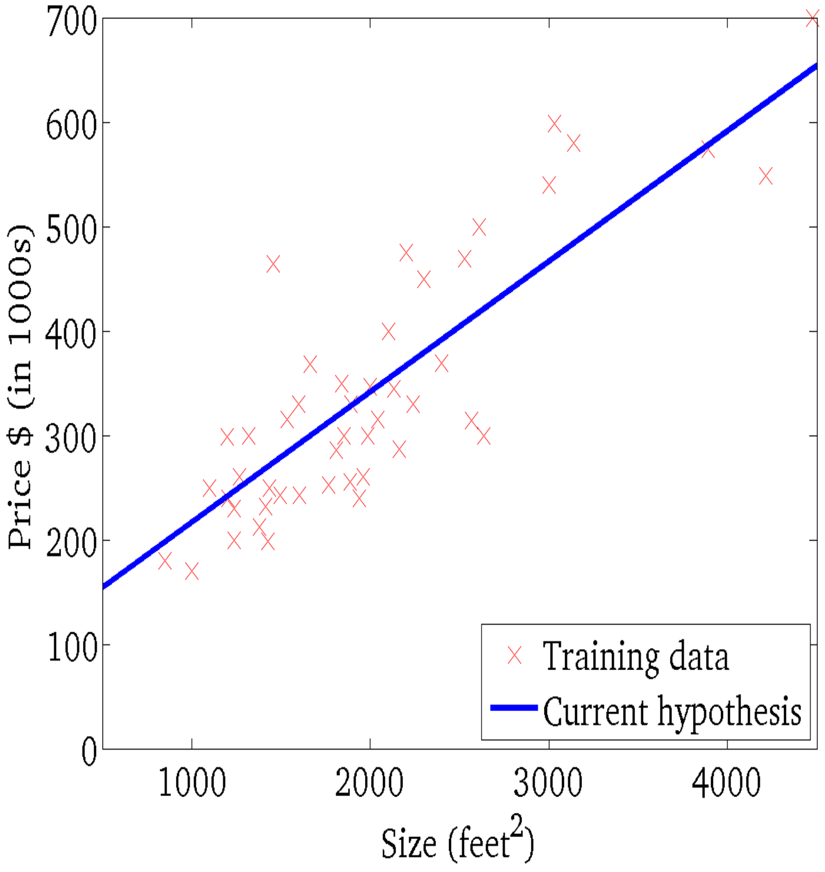
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



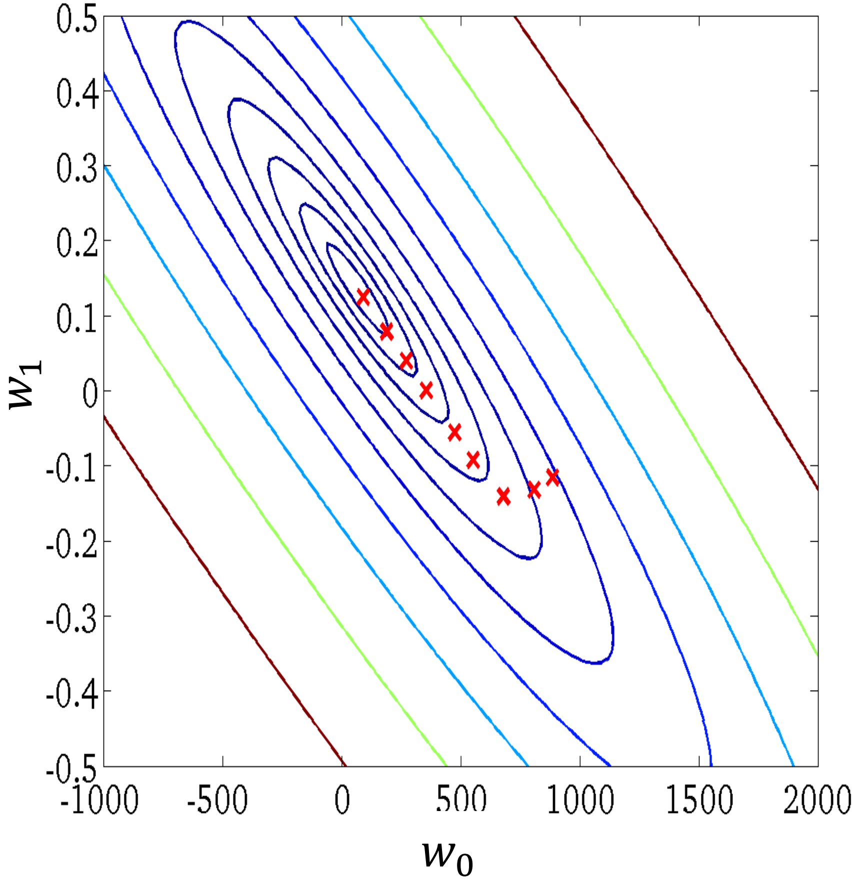
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



$J(w_0, w_1)$

(function of the parameters w_0, w_1)



Gradient descent disadvantages

- Local minima problem

- However, when J is convex, all local minima are also global minima $\Rightarrow$ gradient descent can converge to the global solution.

Stochastic gradient descent

- Batch techniques process the entire training set in one go
  - thus they can be computationally costly for large data sets.

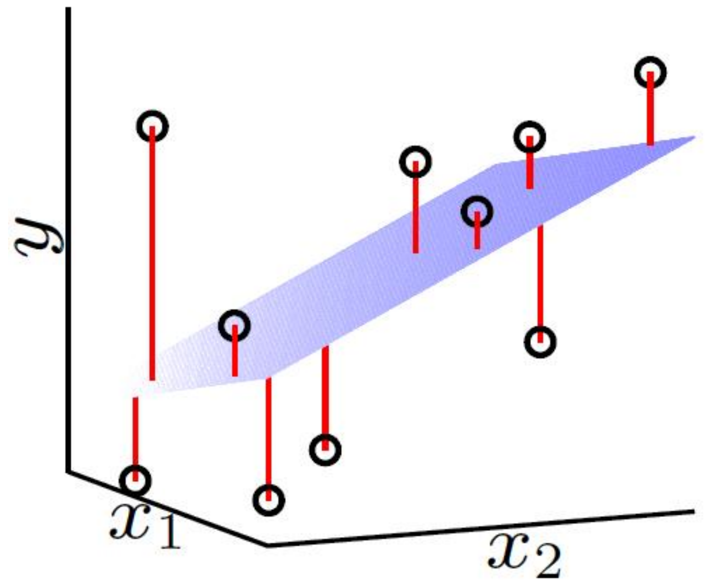
- Stochastic gradient descent: when the cost function can comprise a sum over data points:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} J^{(i)}(\boldsymbol{w})$$

- Update after presentation of a mini-batch S of data:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} - \eta \sum_{j \in S} \nabla_{\boldsymbol{w}} J^{(j)}(\boldsymbol{w})$$

Linear model: multi-dimensional inputs



$$f(\boldsymbol{x};\boldsymbol{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d$$
$$= \boldsymbol{w}^T \boldsymbol{x}$$

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

Generalized linear regression

- Linear combination of fixed non-linear function of the input vector

$$f(\boldsymbol{x};\boldsymbol{w}) \ = w_0 + w_1\phi_1(\boldsymbol{x}) + \ldots w_m\phi_m(\boldsymbol{x})$$

$\{\phi_1(\boldsymbol{x}),\ldots,\phi_m(\boldsymbol{x})\}$: set of basis functions (or features)

$$\phi_i(\boldsymbol{x}): \mathbb{R}^d \rightarrow \mathbb{R}$$

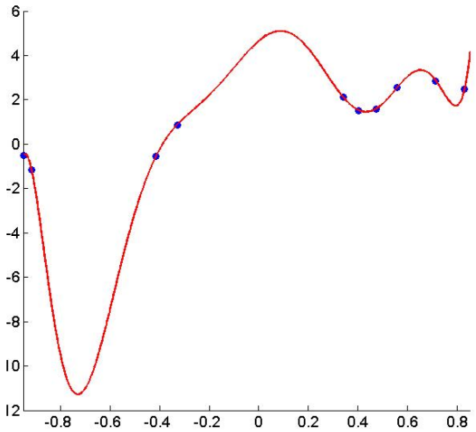
- Polynomial (univariate)

If $\phi_i(x) = x^i$, $i = 1, \ldots, m$, then

$$f(x;\mathbf{w}) = w_0 + w_1 x + \ldots + w_{m-1}x^{m-1} + w_m x^m$$

Model complexity and overfitting

- With limited training data, models may achieve zero training error but a large test error.



- Over-fitting: when the training loss no longer bears any relation to the test (generalization) loss.
  - Fails to generalize to unseen examples.

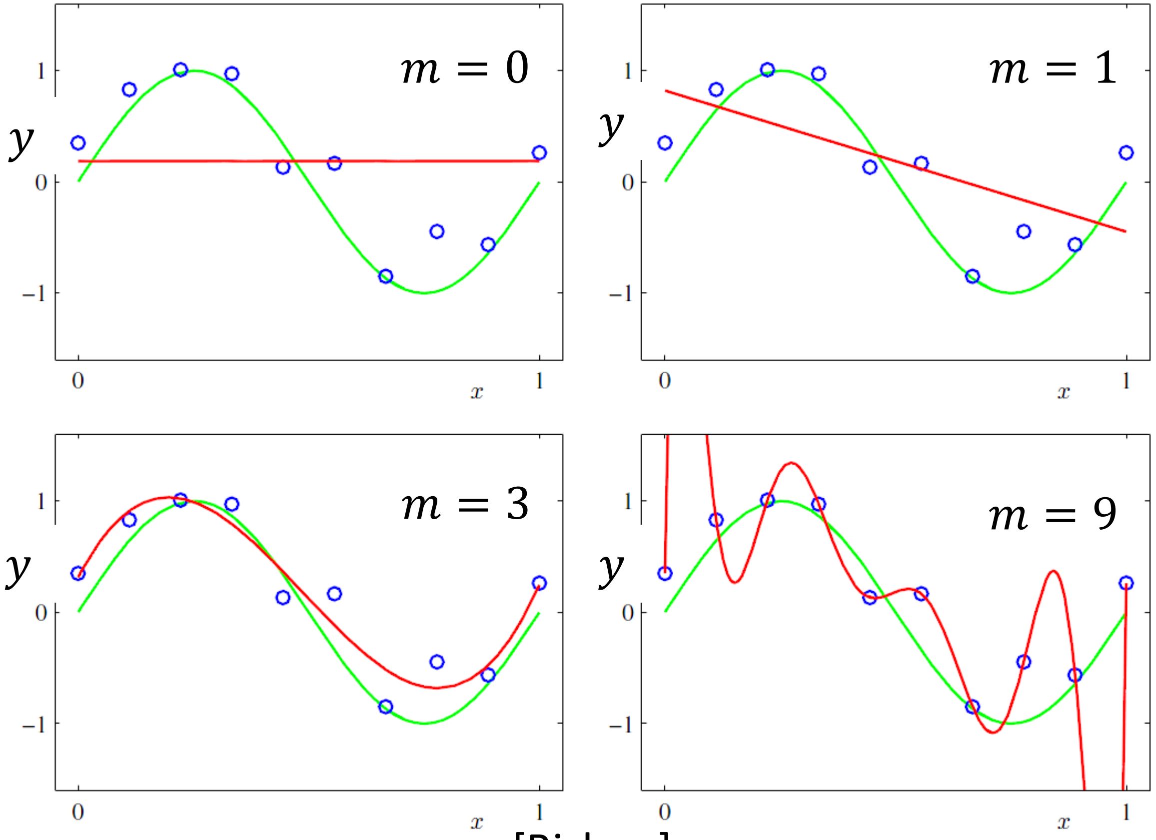
Over-fitting causes

- Model complexity
  - E.g., Model with a large number of parameters (degrees of freedom)

- Low number of training data
  - Small data size compared to the complexity of the model

Model complexity

- Example:
  - Polynomials with larger m are becoming increasingly tuned to the random noise on the target values.

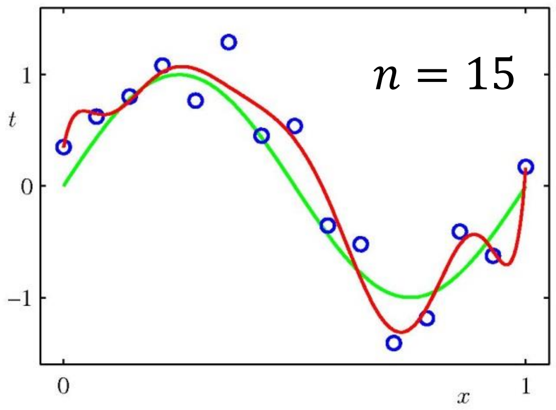


[Bishop]

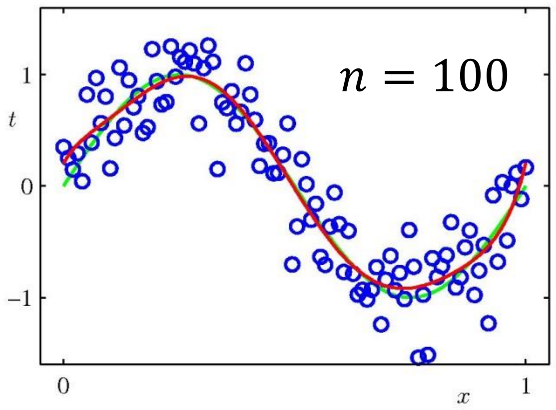
Number of training data & overfitting

- Over-fitting problem becomes less severe as the size of training data increases.

$m = 9$



$m = 9$



[Bishop]

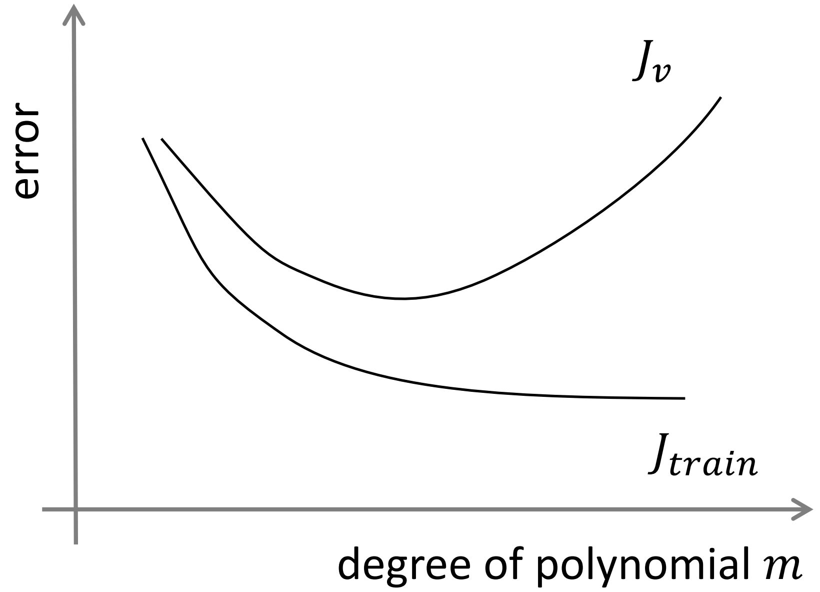
Avoiding over-fitting

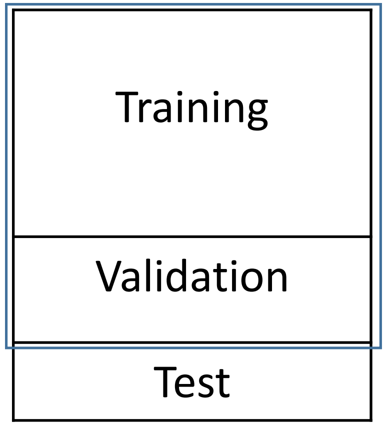
- Determine a suitable value for model complexity
  - **Simple hold-out method**
  - **Cross-validation**

- Regularization (Occam’s Razor)
  - Explicit preference towards simpler models
  - Penalize for the model complexity in the objective function

Simple hold out: training, validation, and test sets

- Simple hold-out chooses the model (hyperparameters) that minimizes error on validation set.

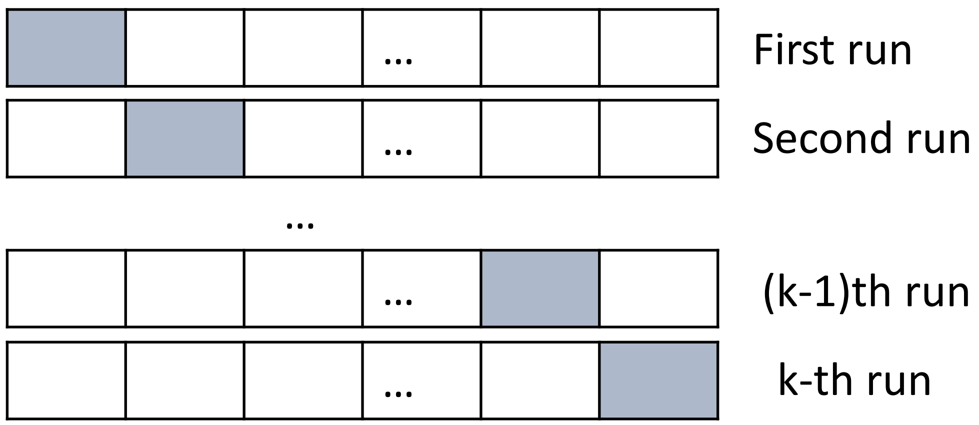




- run on the test set once at the very end!

Cross-Validation (CV): Evaluation

- k-fold cross-validation steps:
  - Shuffle the dataset and randomly partition training data into k groups of approximately equal size
  - for $i = 1$ to k
    - Choose the i-th group as the held-out validation group
    - Train the model on all but the i-th group of data
    - Evaluate the model on the held-out group
  - Performance scores of the model from k runs are **averaged**.



Regularization

- Adding a penalty term in the cost function to discourage the coefficients from reaching large values.

- Ridge regression (weight decay):

$$J(\boldsymbol{w}) = \underbrace{\sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{\phi}\left(\boldsymbol{x}^{(i)}\right)\right)^2}_{\text{Approximation}} + \underbrace{\lambda R(\boldsymbol{w})}_{\text{Generalization}}$$

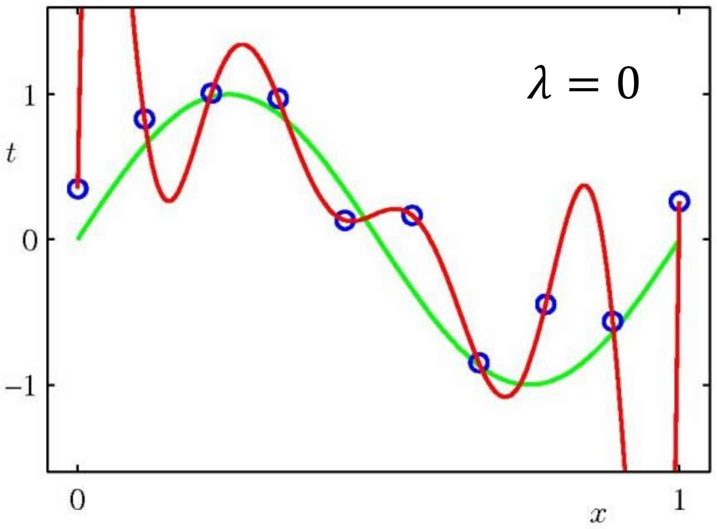
Approximation:
How much model predictions
match training data

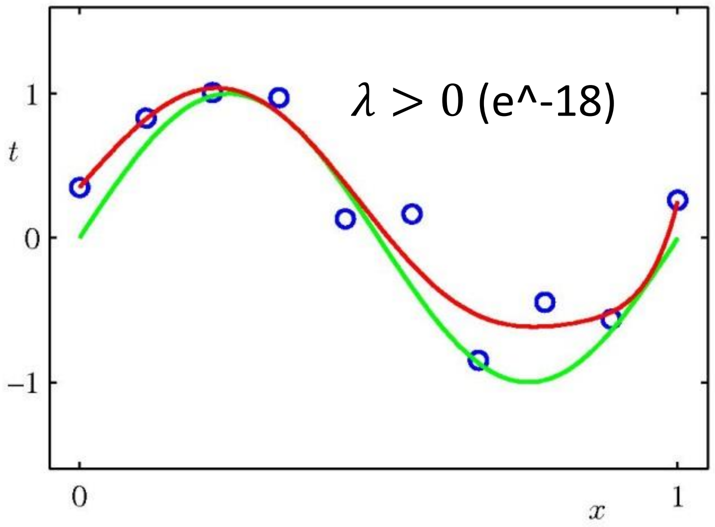
Generalization: prefer simple ones;
Control the variance of the models

e.g. $R(\boldsymbol{w}) = \|\boldsymbol{w}\|^2 = \boldsymbol{w}^T \boldsymbol{w}$

λ: regularization strength
(hyperparameter)

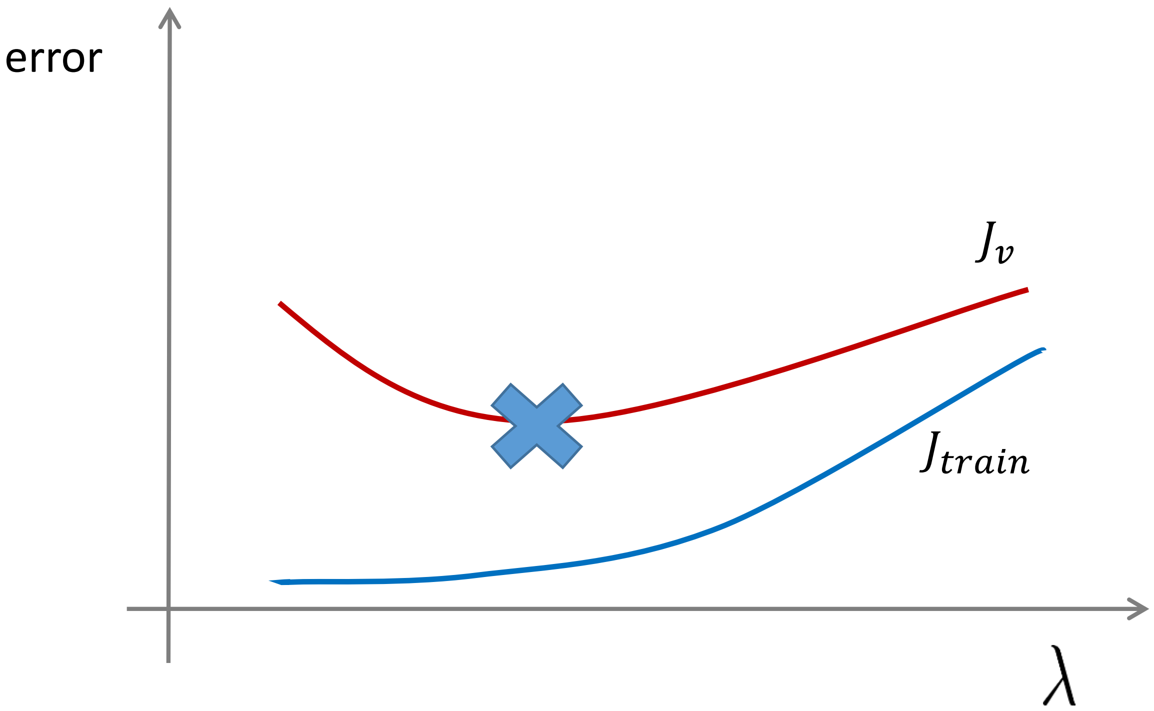
Regularization





[Bishop]

Choosing the regularization parameter



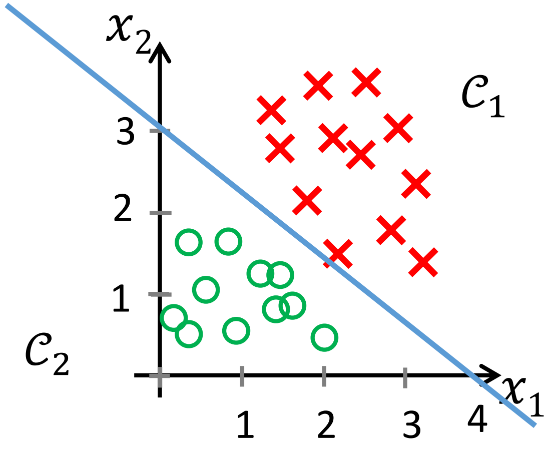
Classification problem

- Given: Training set
  - labeled set of N input-output pairs $D = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - $y \in \{1, \dots, K\}$
- Goal: Given an input $\boldsymbol{x}$, assign it to one of K classes
- Examples:
  - Image classification
  - Speech recogntion
  - …

Linear Classifier example

- Two class example:

$$-\frac{3}{4}x_1 - x_2 + 3 = 0$$



if $\boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 0$ then $\mathcal{C}_1$
else $\mathcal{C}_2$

$$\boldsymbol{w} = \begin{bmatrix} -\frac{3}{4} & -1 \end{bmatrix}$$

$$w_0 = 3$$

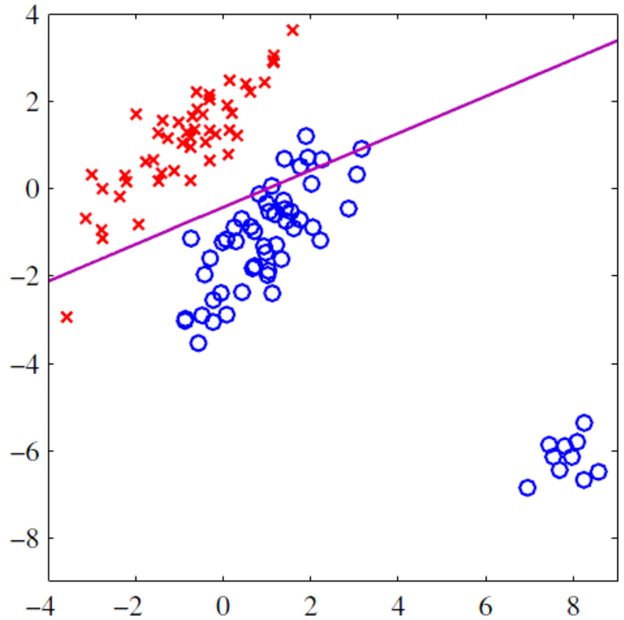
Square error loss function for classification!

$K = 2$

Square error loss is not suitable for classification:

- Least square loss penalizes ‘too correct’ predictions (that they lie a long way on the correct side of the decision)
- Least square loss also lack robustness to noise

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \left(wx^{(i)} + w_0 - y^{(i)}\right)^2$$



Parametric classifier: Multiclass

- $f(\boldsymbol{x}; \boldsymbol{W}) = [f_1(\boldsymbol{x}, \boldsymbol{W}), \ldots, f_K(\boldsymbol{x}, \boldsymbol{W})]^T$

- $\boldsymbol{W} = [\boldsymbol{w}_1 \quad \cdots \quad \boldsymbol{w}_K]$ contains one vector of parameters for each class

Parametric classifier: Linear

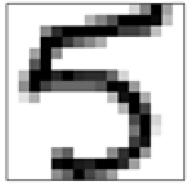
- $f(\boldsymbol{x}; \boldsymbol{W}) = [f_1(\boldsymbol{x}, \boldsymbol{W}), \dots, f_K(\boldsymbol{x}, \boldsymbol{W})]$

- $\boldsymbol{W} = [\boldsymbol{w}_1 \quad \cdots \quad \boldsymbol{w}_K]$ contains one vector of parameters for each class
  - In linear classifiers, $\boldsymbol{W}$ is $d \times K$ where d shows number of features
  - $\boldsymbol{W}^T \boldsymbol{x}$ provides us a vector

- $f(\boldsymbol{x}; \boldsymbol{W})$ contains K numbers giving class scores for the input $\boldsymbol{x}$

Linear classifier

- Output obtained from $\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$



28×28

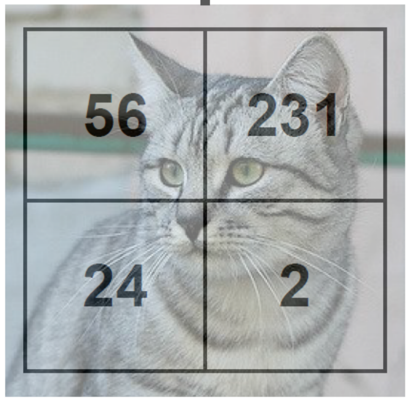
$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_{784} \end{bmatrix}$$

$$\boldsymbol{W}^T = \begin{bmatrix} \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_{10} \end{bmatrix}_{10 \times 784}$$

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{10} \end{bmatrix}$$

Example

Stretch pixels into column

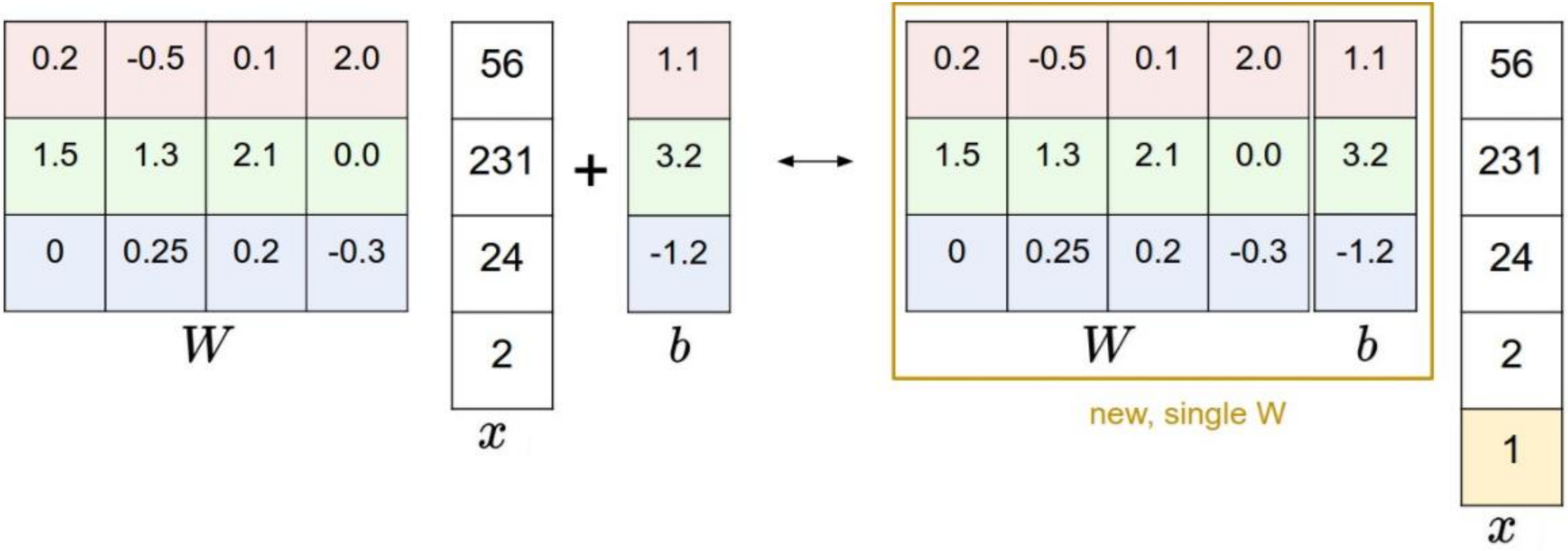


Input image

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W^T} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array}$$

How can we tell whether this W and b is good or bad?

Bias can also be included in the W matrix



Multi-class SVM

$$J(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} L^{(i)} + \lambda R(\boldsymbol{W})$$

Hinge loss:

$$L^{(i)} = \sum_{j \neq y^{(i)}} \max\left(0, 1 + s_j - s_{y^{(i)}}\right)$$

$$s_j \equiv f_j\left(\boldsymbol{x}^{(i)}; \boldsymbol{W}\right) = \boldsymbol{w}_j^T \boldsymbol{x}^{(i)}$$

$$= \sum_{j \neq y^{(i)}} \max\left(0, 1 + \boldsymbol{w}_j^T \boldsymbol{x}^{(i)} - \boldsymbol{w}_{y^{(i)}}^T \boldsymbol{x}^{(i)}\right)$$

L2 regularization:

$$R(\boldsymbol{W}) = \sum_{k=1}^{K}\sum_{l=1}^{d} w_{lk}^2$$

Multi-class SVM loss: Example



3 training examples, 3 classes.
With some W the scores are $W^T x$

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

$$s_j = \boldsymbol{w}_j^T \boldsymbol{x}^{(i)}$$

$$L^{(i)} = \sum_{j \neq y^{(i)}} \max\left(0, 1 + s_j - s_{y^{(i)}}\right)$$

$$\frac{1}{N} \sum_{i=1}^{N} L^{(i)} = \frac{1}{3}(2.9 + 0 + 12.9) = 5.7$$

$$\begin{aligned} L^{(1)} &= \max(0, 1 + 5.1 - 3.2) \\ &+ \max(0, 1 - 1.7 - 3.2) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \end{aligned}$$

$$\begin{aligned} L^{(2)} &= \max(0, 1 + 1.3 - 4.9) \\ &+ \max(0, 1 + 2 - 4.9) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \end{aligned}$$

$$\begin{aligned} L^{(3)} &= \max(0, 2.2 - (-3.1) + 1) \\ &+ \max(0, 2.5 - (-3.1) + 1) \\ &= \max(0, 6.3) + \max(0, 6.6) \\ &= 6.3 + 6.6 = 12.9 \end{aligned}$$

Some questions?

$$L^{(i)} = \sum_{j \neq y^{(i)}} \max\left(0, 1 + s_j - s_{y^{(i)}}\right)$$

- Q1: What if the sum was over all classes? (including $j = y_i$)
- Q2: What if we used mean instead of sum?
- Q3: What if we used $L^{(i)} = \sum_{j \neq y^{(i)}} \max\left(0, 1 + s_j - s_{y^{(i)}}\right)^2$?
- Q4: what is the min/max possible?
- Q5: why do we use regularization term?

Other regularization terms

- **L2 regularization** $\sum_{k=1}^{K} \sum_{l=1}^{d} w_{lk}^2$
- L1 regularization $\sum_{k=1}^{K} \sum_{l=1}^{d} |w_{lk}|$
- Elastic net (L1 + L2) $\beta \sum_{k=1}^{K} \sum_{l=1}^{d} w_{lk}^2 + \sum_{k=1}^{K} \sum_{l=1}^{d} |w_{lk}|$

Softmax Classifier (Multinomial Logistic Regression)

softmax function

$$P\left(Y=k \middle| X=\boldsymbol{x}^{(i)}\right)=\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

$$s_k=f_k\left(\boldsymbol{x}^{(i)}; W\right)=w_k^T \boldsymbol{x}^{(i)}$$

- Maximum log likelihood is equivalent to minimize the negative of log likelihood of the correct class:

$$\begin{aligned} L^{(i)} &= -\log P\left(Y=y^{(i)} \middle| X=x^{(i)}\right) \\ &= -s_{y^{(i)}}+\log \sum_{j=1}^{K} e^{s_j} \end{aligned}$$

Cross-entropy loss

Softmax classifier loss: example

$$L^{(i)} = -\log \frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{K} e^{s_j}}$$

unnormalized probabilities

cat	**3.2**		**24.5**		**0.13**
car	5.1	$\xrightarrow{\text{exp}}$	164.0	$\xrightarrow{\text{normalize}}$	0.87
frog	-1.7		0.18		0.00

$$L^{(1)} = -\log 0.13 = 0.89$$

Cross entropy

$$H(q,p) = -\sum_{x} q(x) \log p(x)$$

- For the loss of the softmax classifier:
  - p: estimated class probabilities $\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$
  - q: the true distribution
    - all probability mass is on the correct class $q\left(Y = y^{(i)}\right) = 1$ $\left(q\left(Y \neq y^{(i)}\right) = 0\right)$.

Relation to KL divergence

$$H(q,p) = H(q) + D_{KL}(q||p)$$

- Since $H(q)$ for the loss of softmax classifier is zero:
  - Minimizing cross entropy is equivalent to minimizing the KL divergence between the two distributions (a measure of distance).
  - cross-entropy loss wants the predicted distribution to have all of its mass on the correct answer.

Recap

Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss

We need $\nabla_W L$ to update weights

Resources

- Deep Learning Book, Chapter 5.
- Please see the following notes:
  - http://cs231n.github.io/linear-classify/
  - http://cs231n.github.io/optimization-1/