Learning: Linear Methods

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Fall 2023

Soleymani

Some slides are based on Klein and Abdeel, CS188, UC Berkeley.

Components of (Supervised) Learning

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

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

- Pick a formula $g: \mathcal{X} \rightarrow \mathcal{Y}$ that approximates the target function f
  - selected from a set of hypotheses $\mathcal{H}$

Supervised Learning: Regression vs. Classification

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

Regression: Example

- Housing price prediction

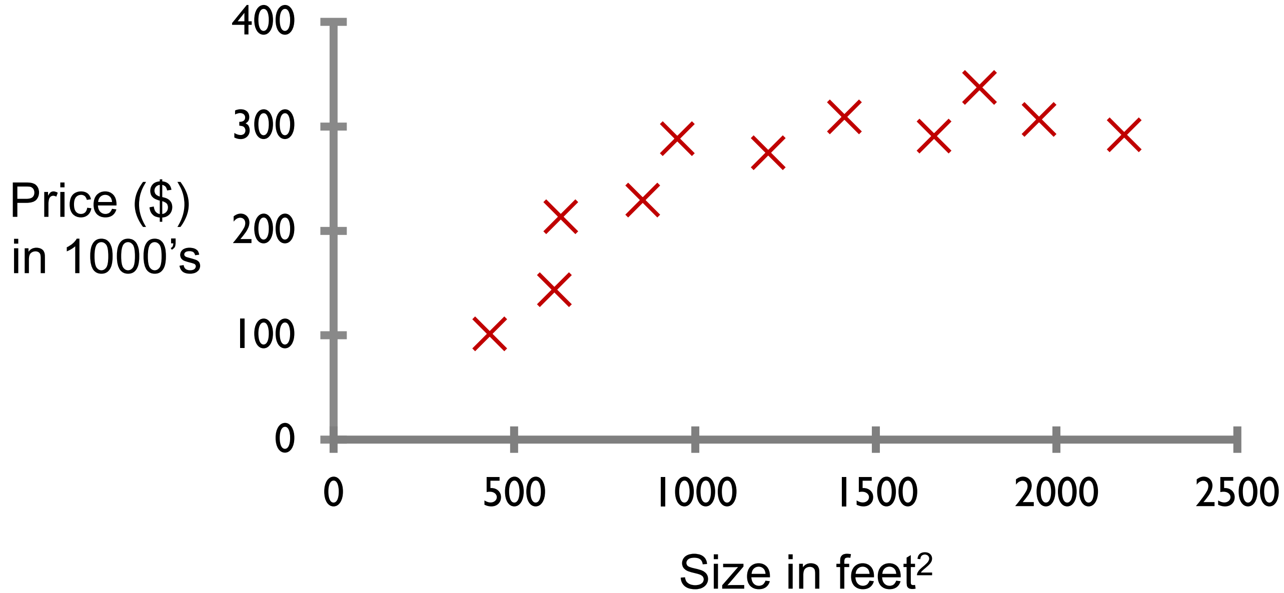
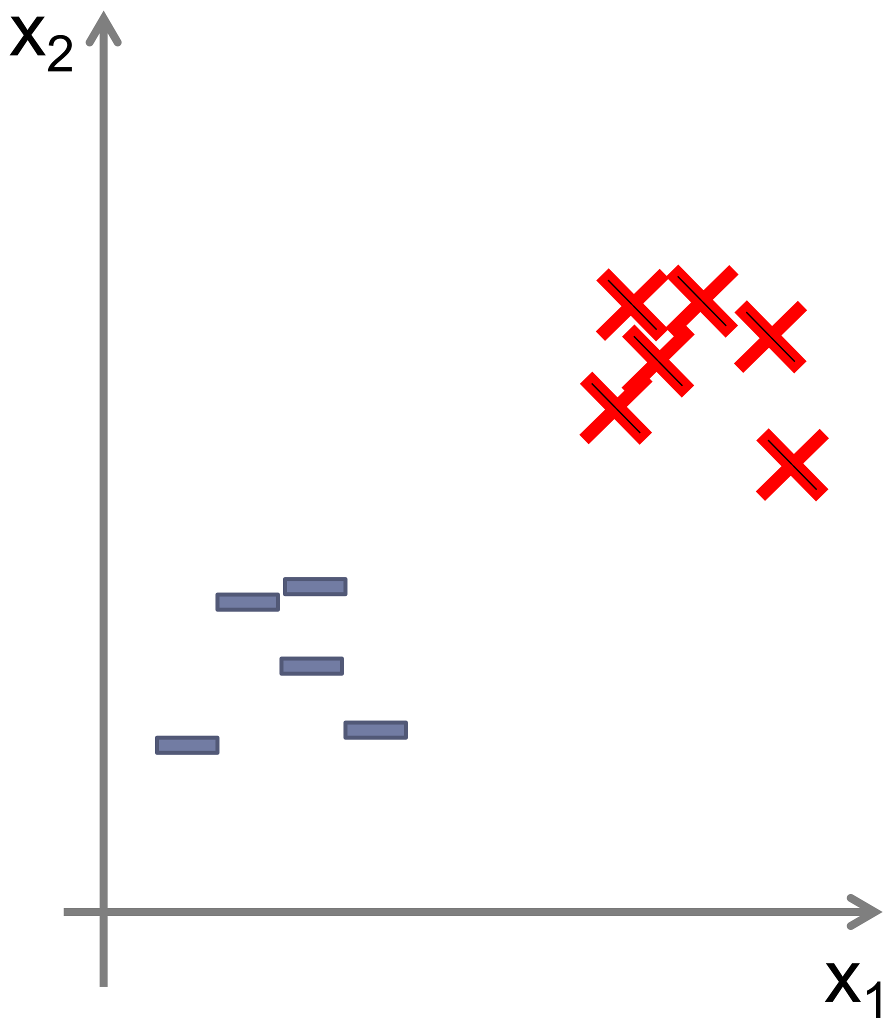


Figure adopted from slides of Andrew Ng

Training data: Example

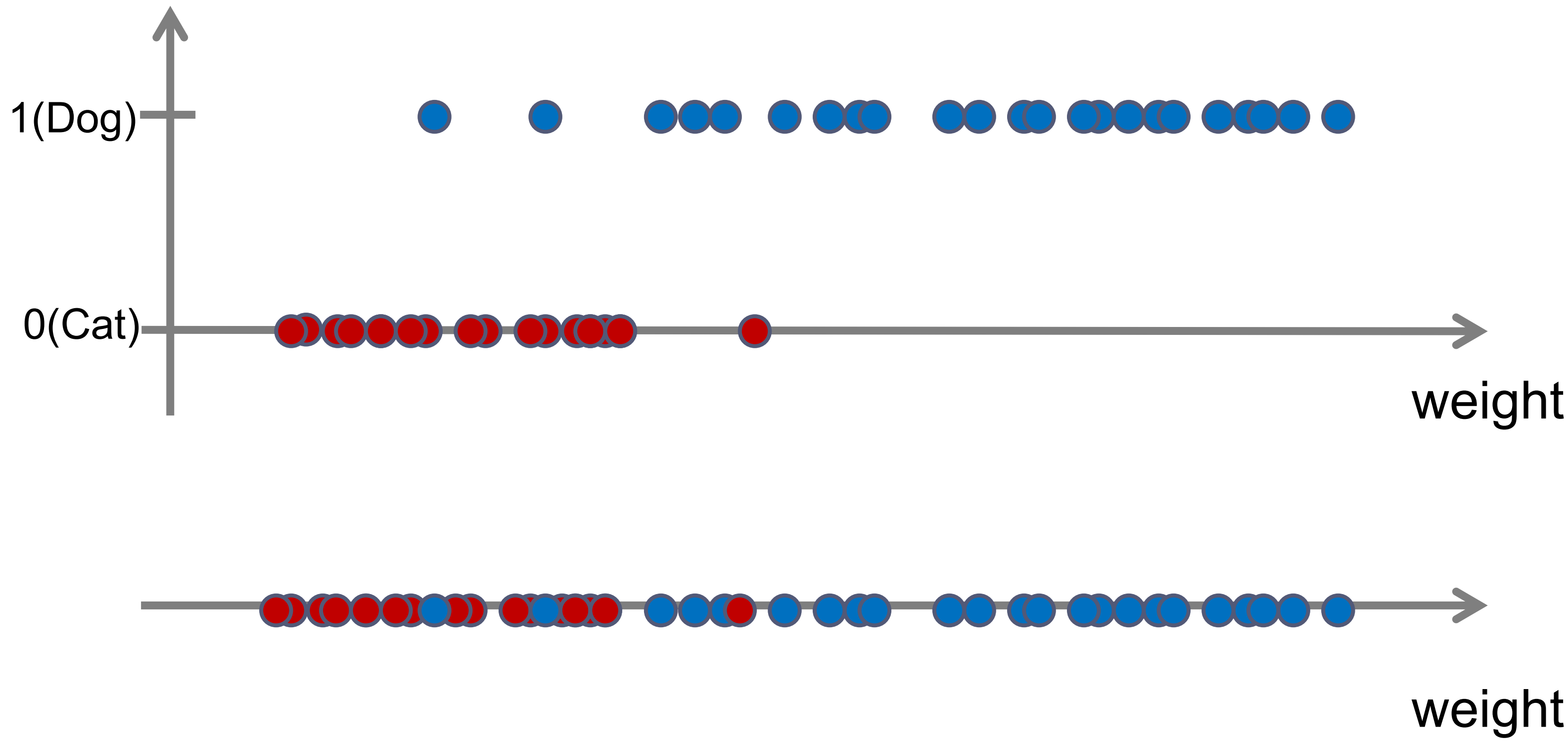


Training data

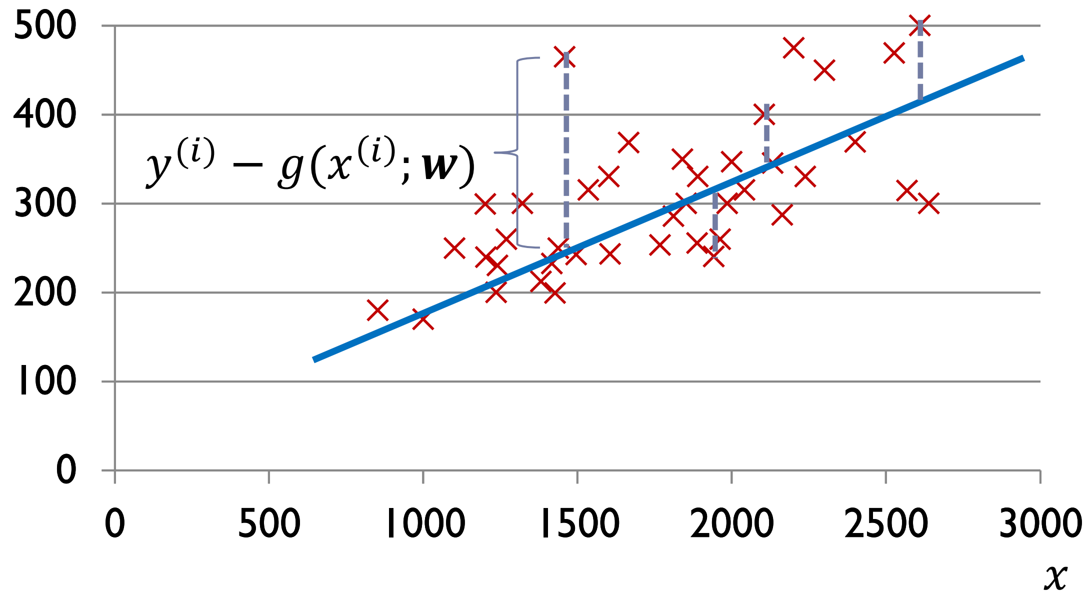
x_1	x_2	y
0.9	2.3	1
3.5	2.6	1
2.6	3.3	1
2.7	4.1	1
1.8	3.9	1
6.5	6.8	-1
7.2	7.5	-1
7.9	8.3	-1
6.9	8.3	-1
8.8	7.9	-1
9.1	6.2	-1

Classification: Example

- Weight (Cat, Dog)



Linear regression

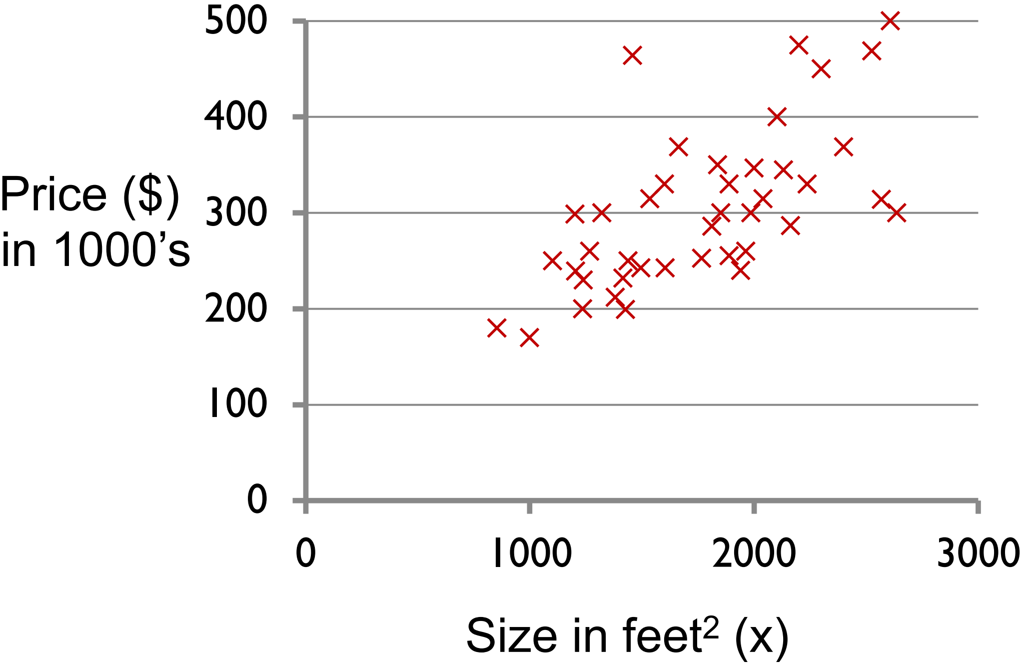


Cost function:

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

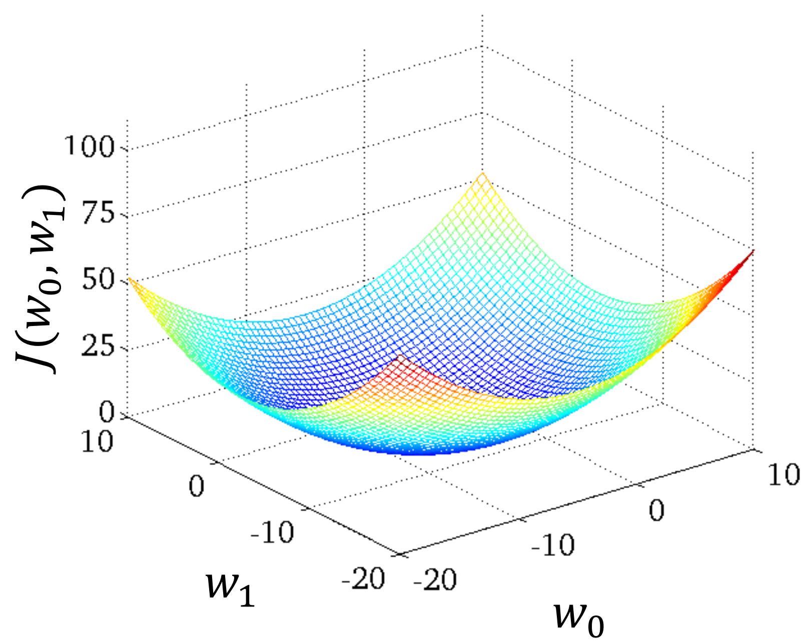
$$= \sum_{i=1}^{n} \left(y^{(i)} - w_0 - w_1 x^{(i)}\right)^2$$

Cost function



$J(\mathbf{w})$

(function of the parameters w_0, w_1)



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

Review: Gradient Descent

- First-order optimization algorithm to find $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} J(\boldsymbol{w})$
  - Also known as "**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 $\boldsymbol{w}^t$

Gradient ascent takes steps proportional to (the positive of) the gradient to find a local maximum of the function

Review: Gradient descent

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

Step size
(Learning rate parameter)

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

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \left[\frac{\partial J(\boldsymbol{w})}{\partial w_1}, \frac{\partial J(\boldsymbol{w})}{\partial w_2}, \dots, \frac{\partial J(\boldsymbol{w})}{\partial w_d}\right]^T$$

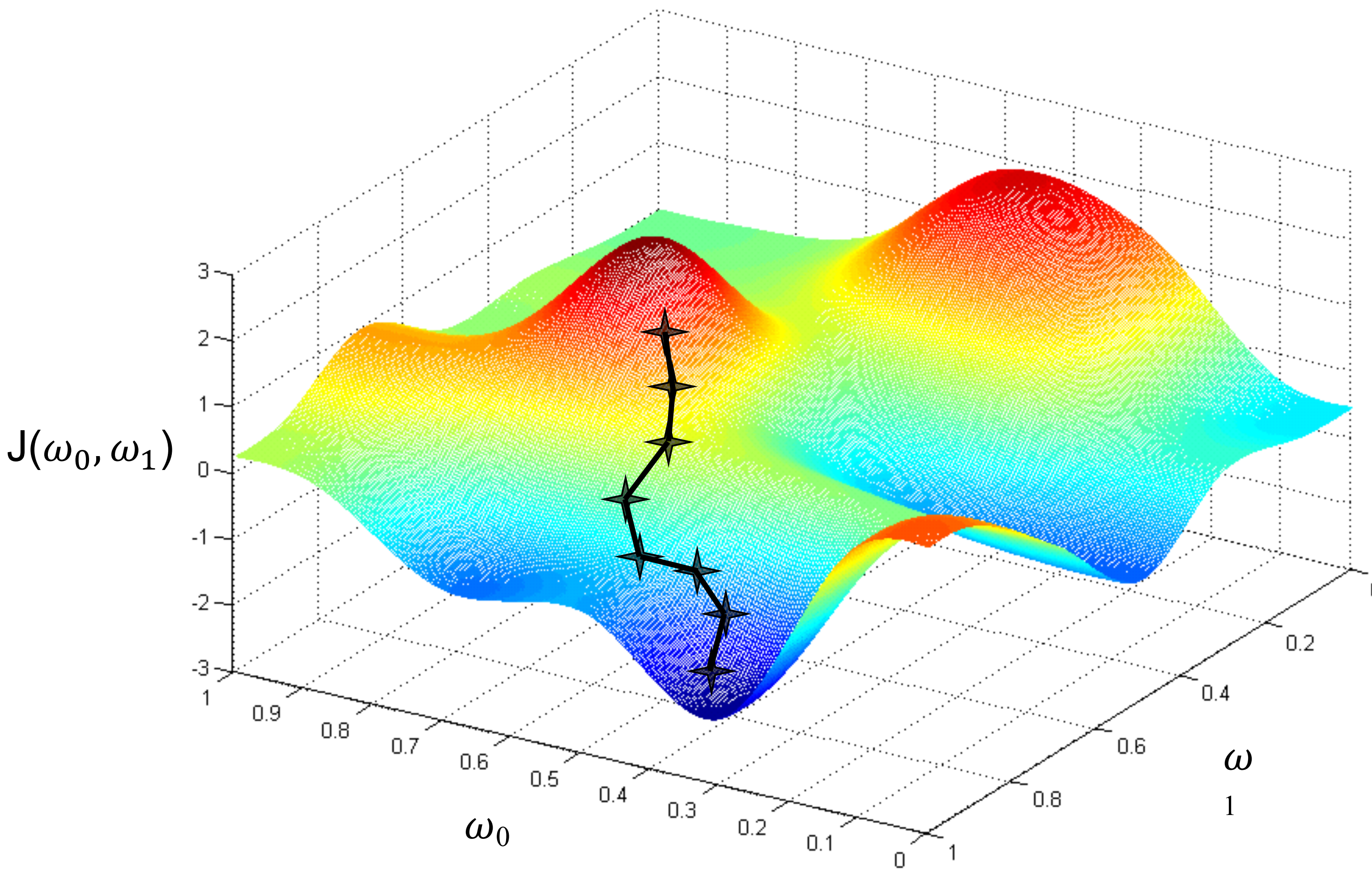
- If η is small enough, then $J(\boldsymbol{w}^{t+1}) \leq J(\boldsymbol{w}^{t})$.
- η can be allowed to change at every iteration as η_t.

Review: Gradient Descent Disadvantages

- Local minima problem

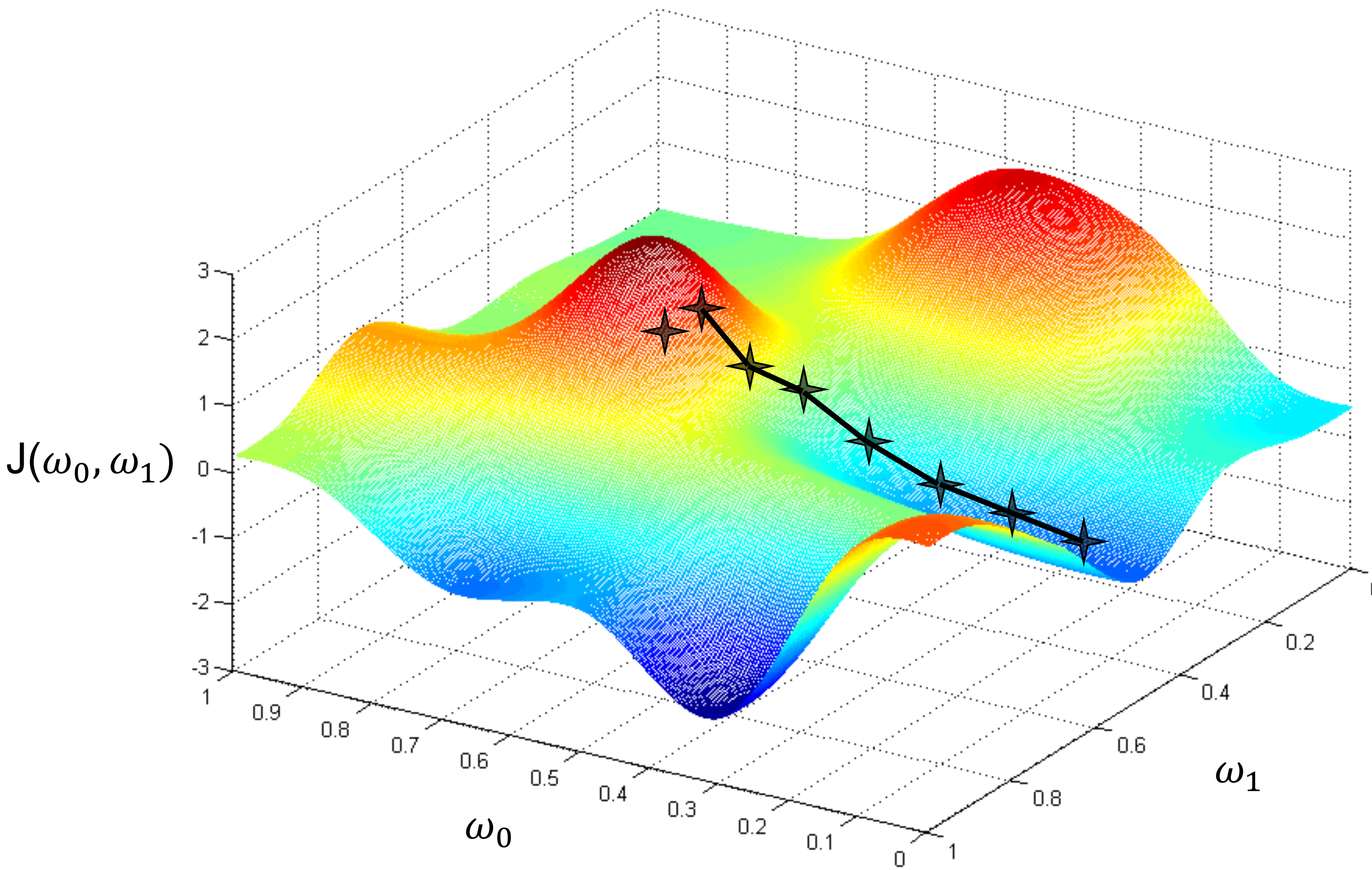
- However, when J is convex, all local minima are also global minima $\Rightarrow$ gradient descent can converge to the global solution.

Review: Problem of Gradient Descent with Non-convex Cost Functions



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

Review: Problem of Gradient Descent with Non-convex Cost Functions



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

Gradient Descent for SSE Cost Function

- $J(\boldsymbol{w})$: Sum of squares error

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

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

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

- Weight update rule for $g(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$:

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

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

Gradient Descent for SSE Cost Function

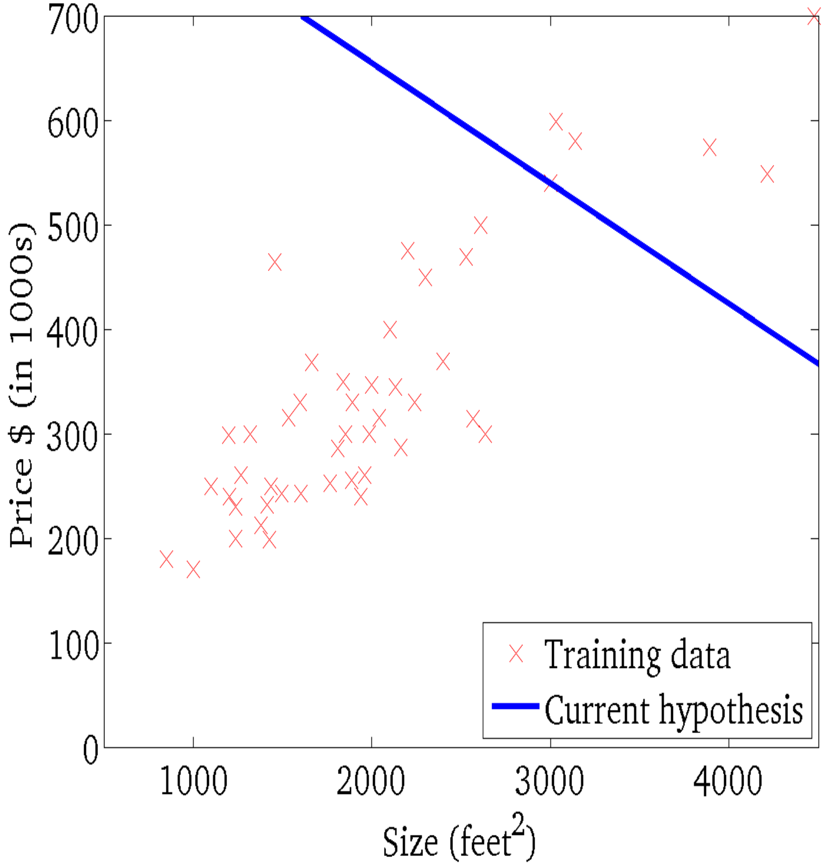
- Weight update rule: $g(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x}$

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

Batch mode: each step considers all training data

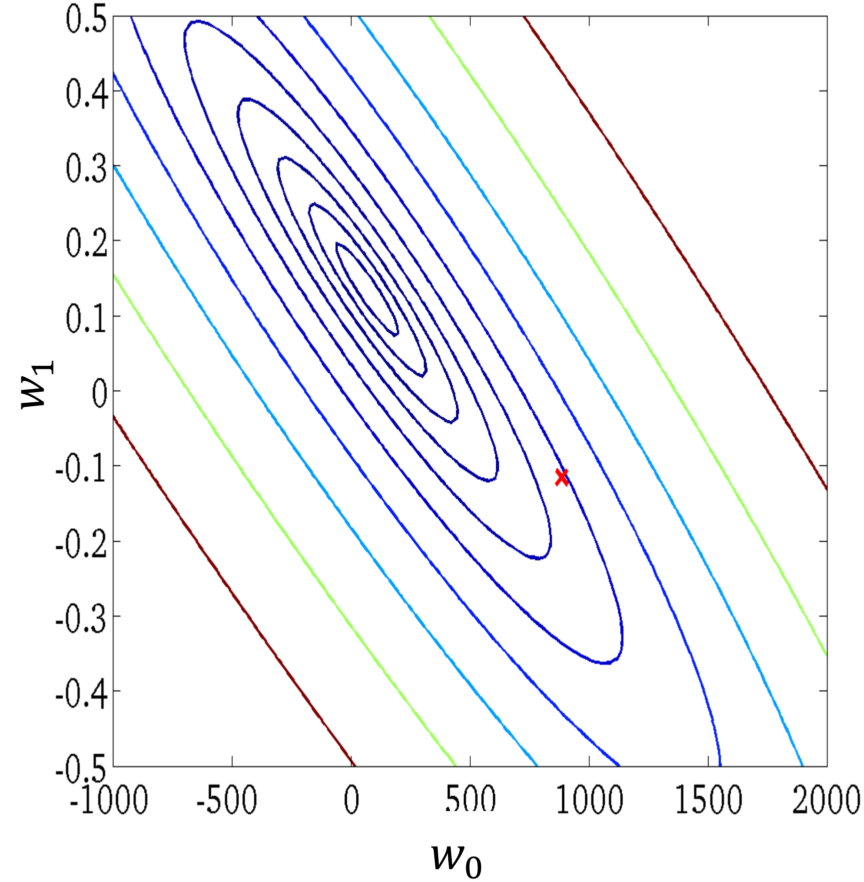
- η: too small → gradient descent can be slow.
- η: too large → gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

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



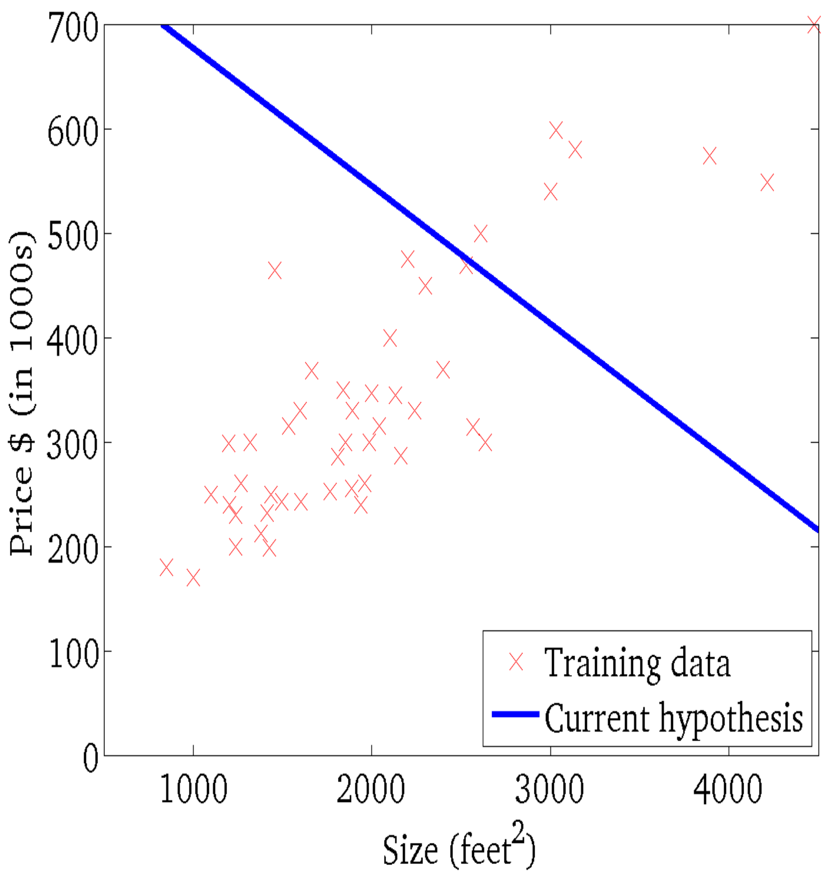
$J(w_0, w_1)$

(function of the parameters w_0, w_1)



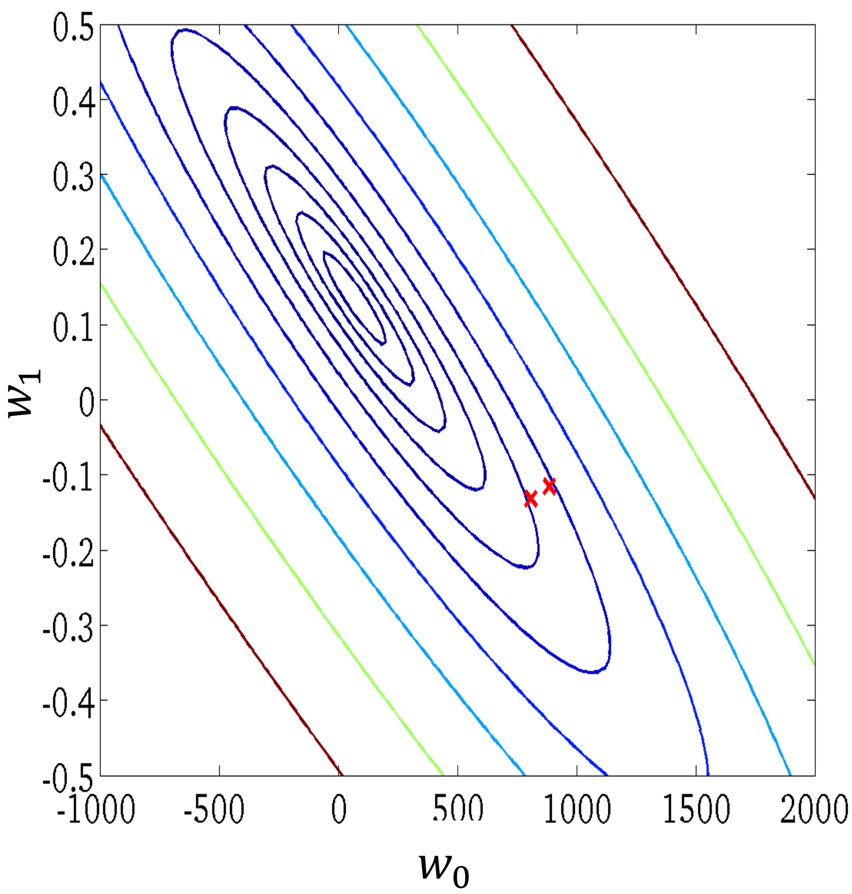
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



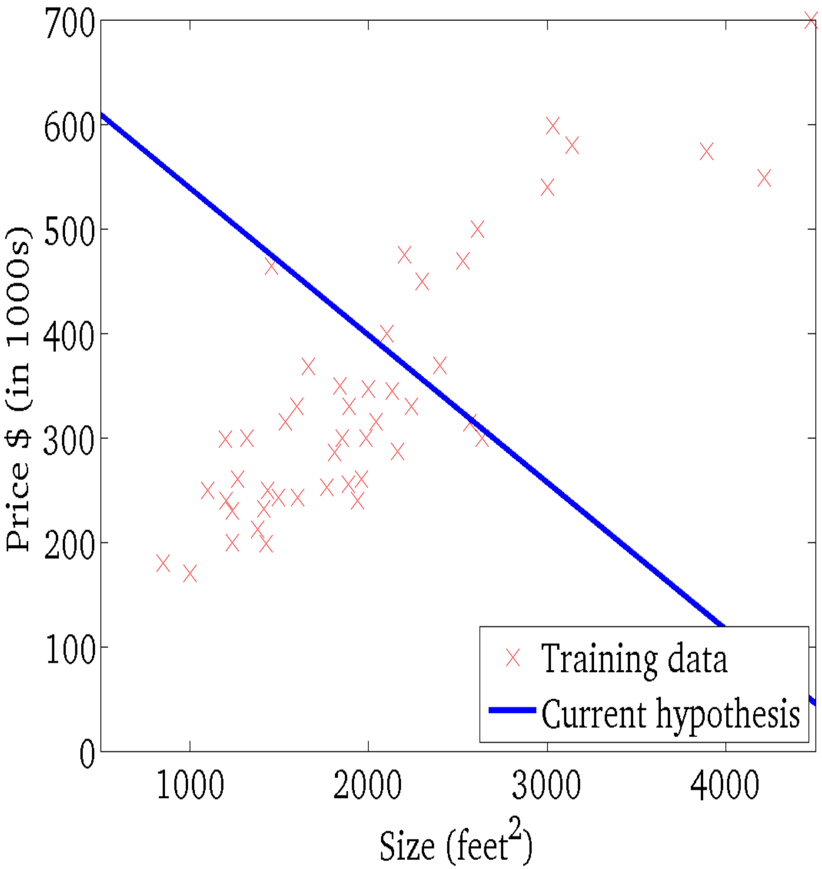
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



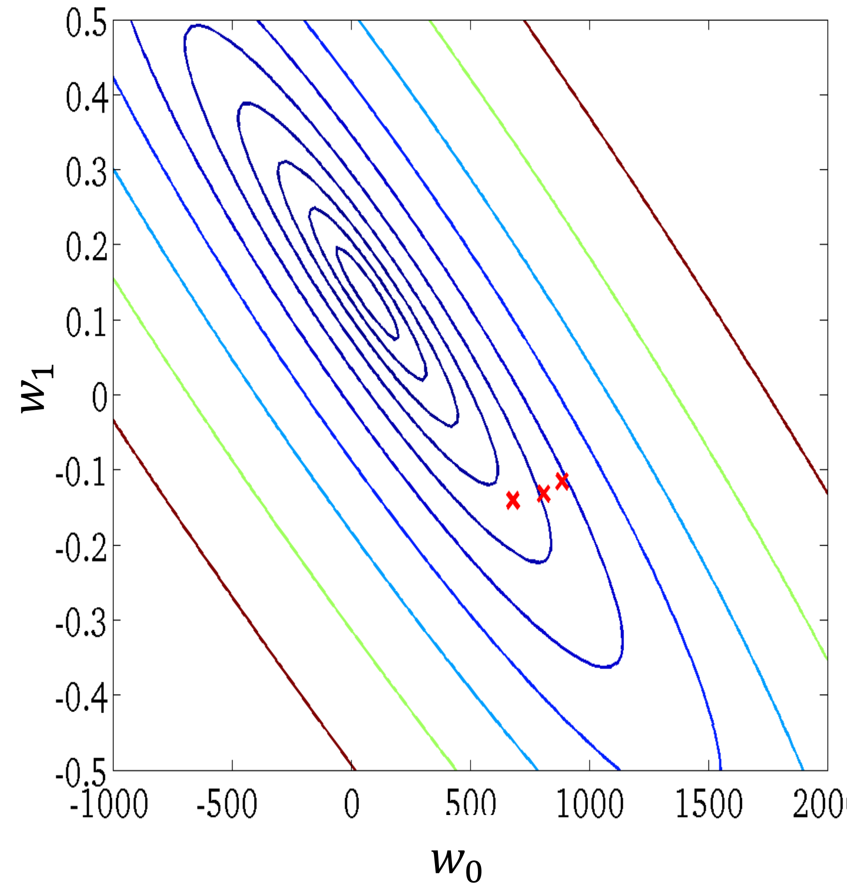
 This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



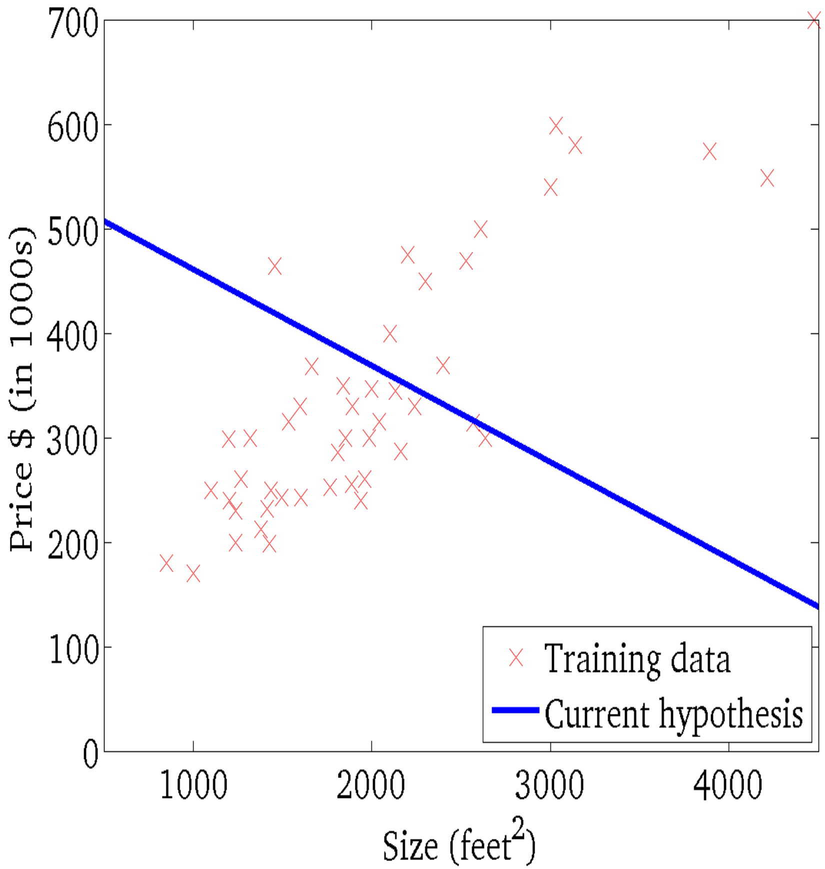
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



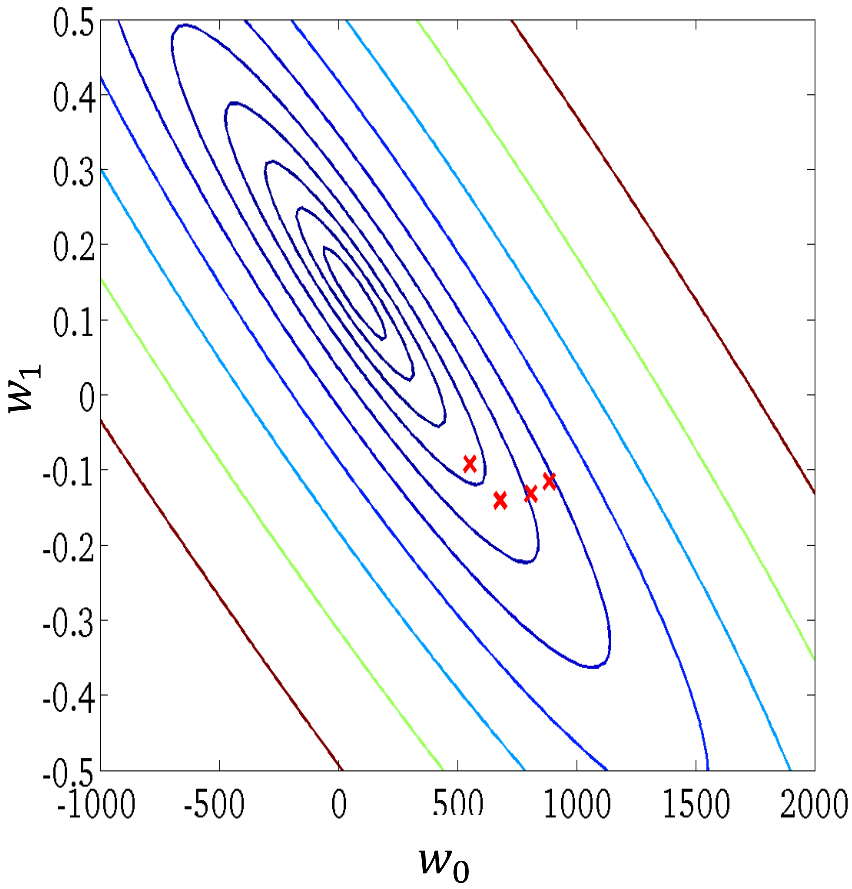
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



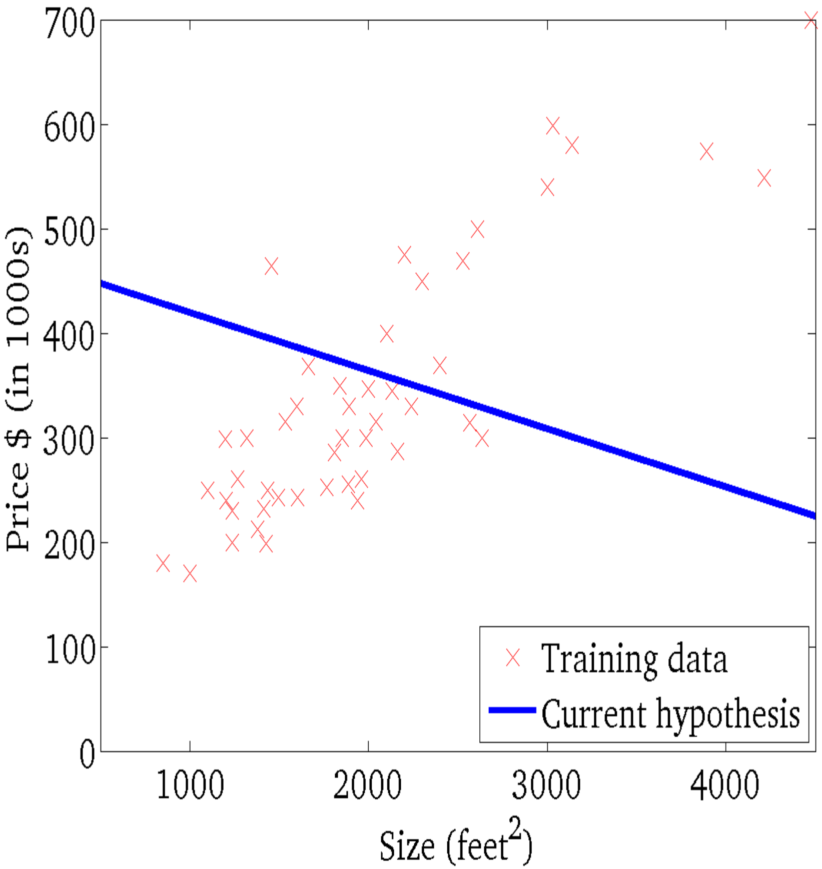
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



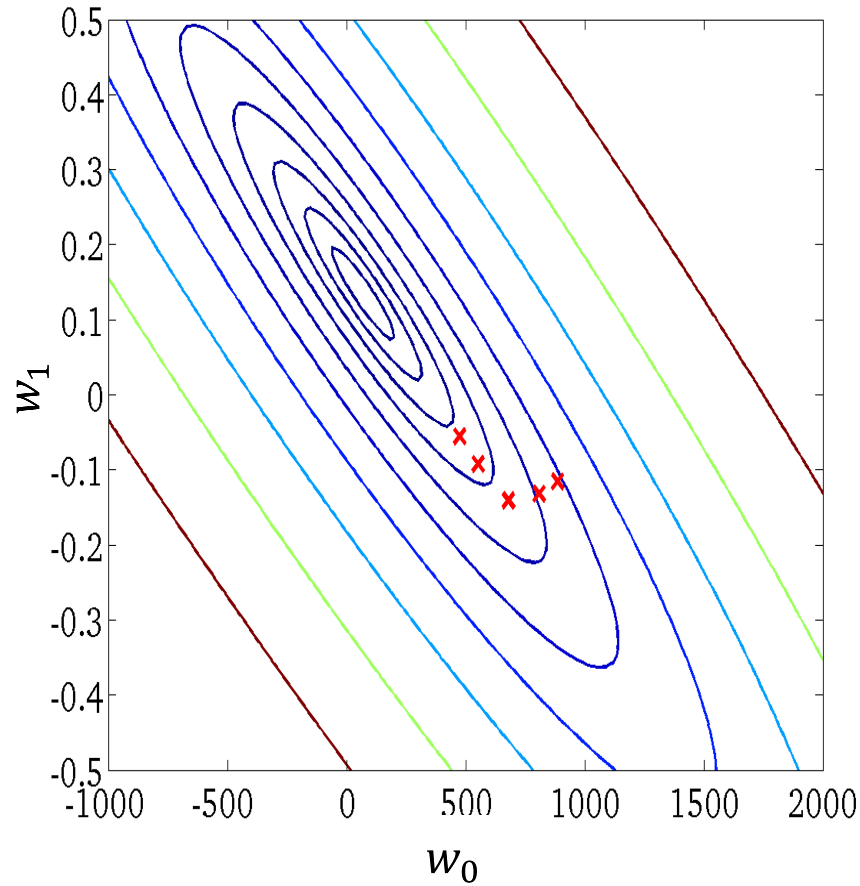
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



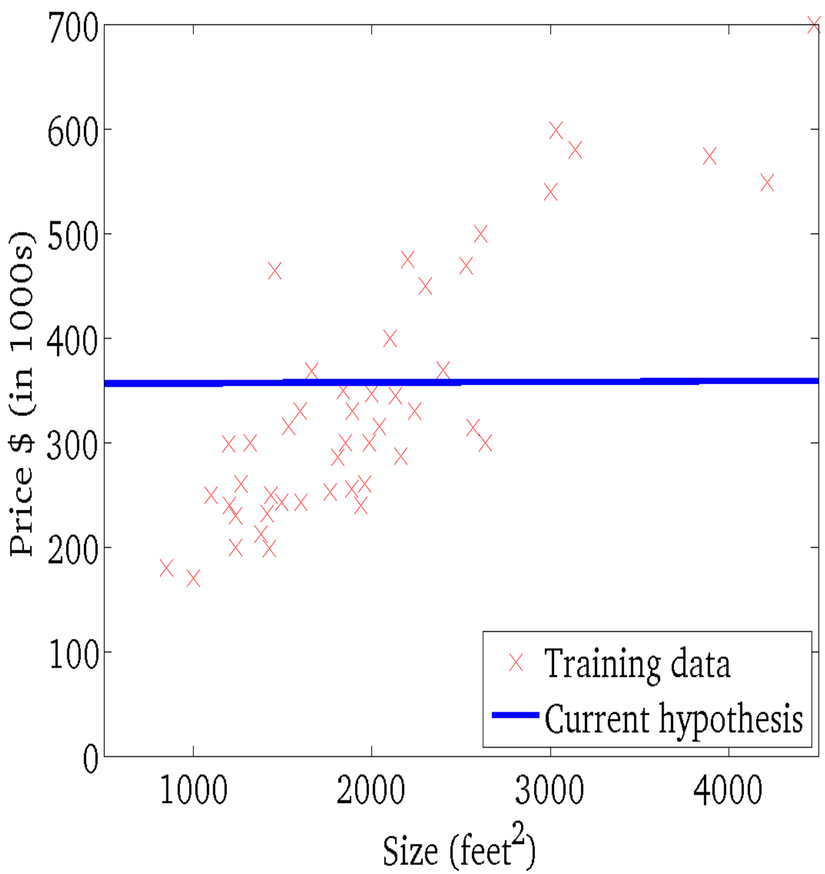
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



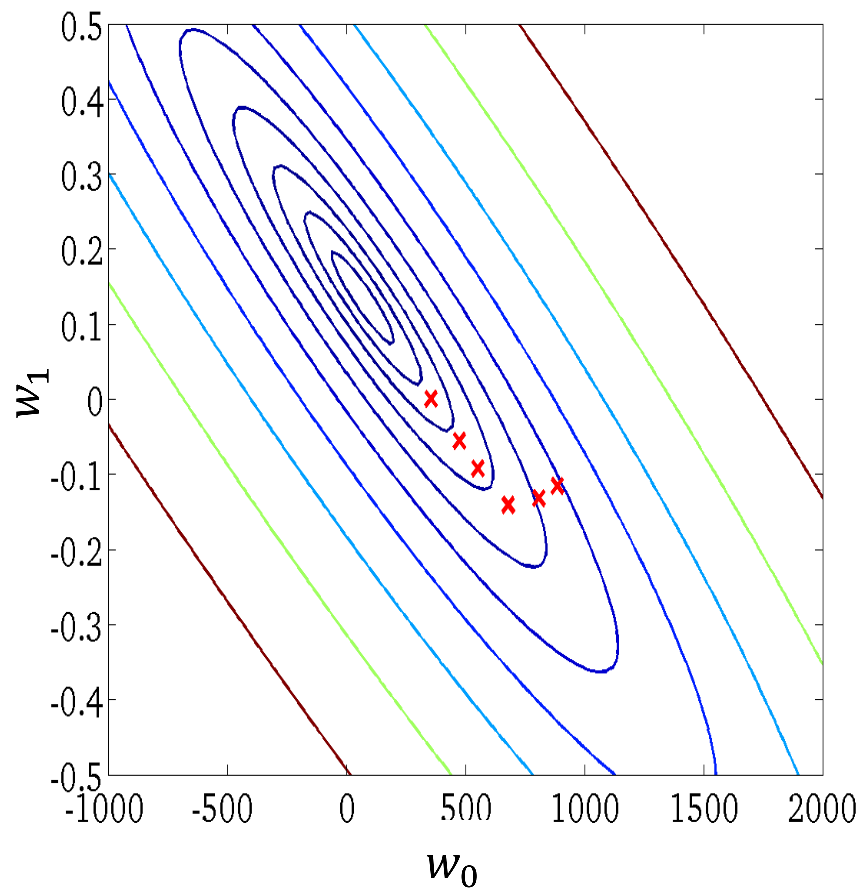
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



$J(w_0, w_1)$

(function of the parameters w_0, w_1)

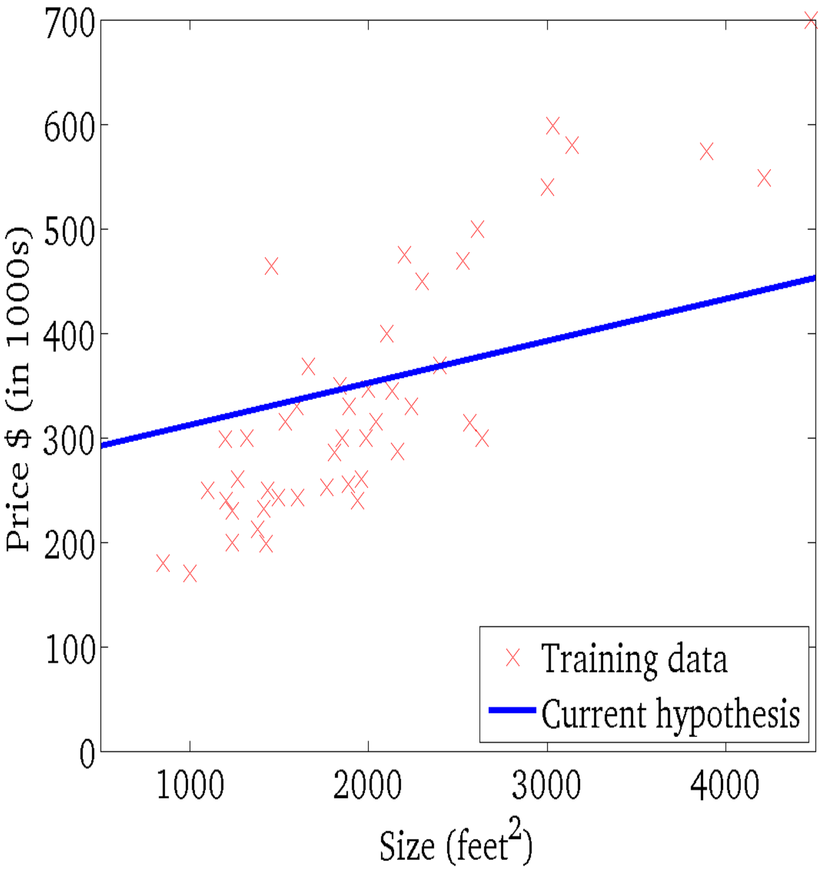


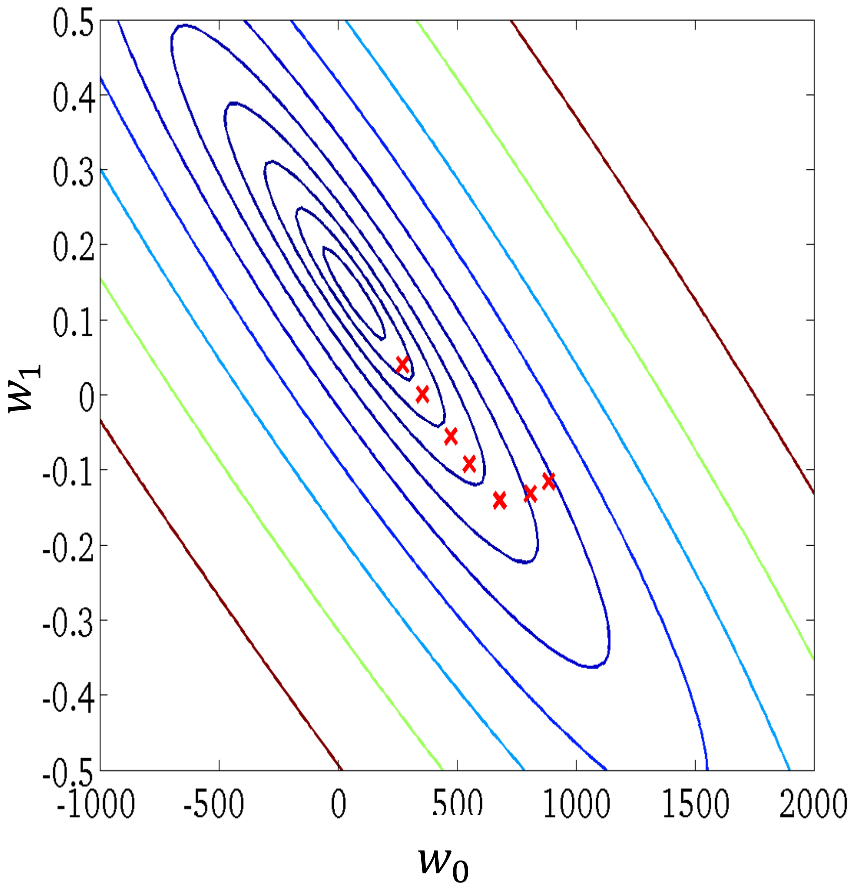
This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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

$J(w_0, w_1)$

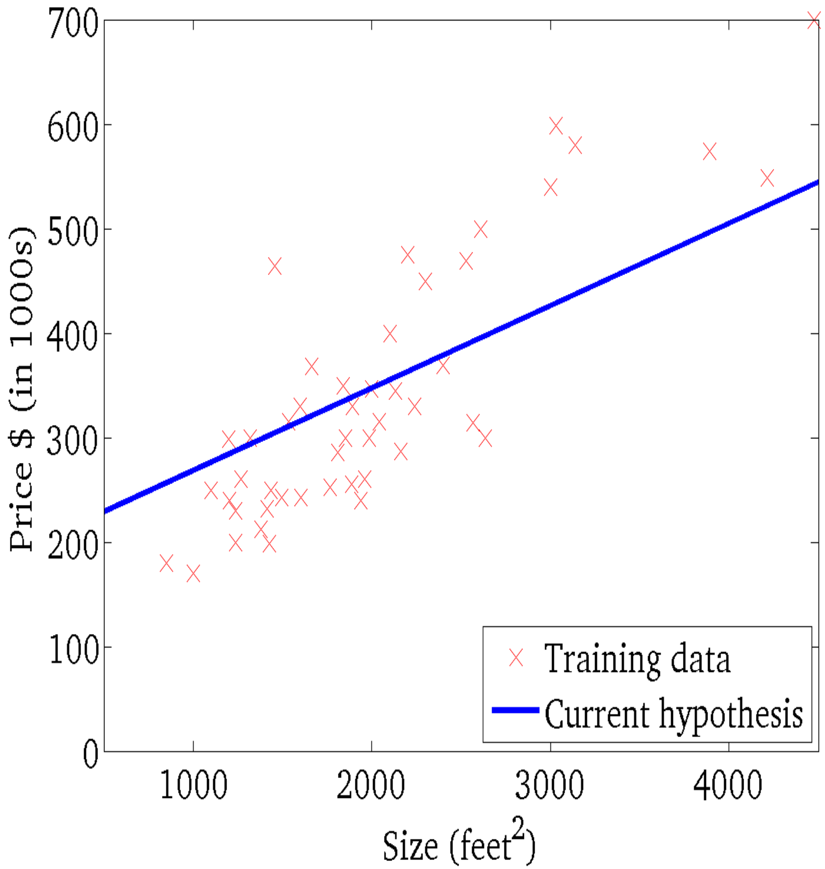
(function of the parameters w_0, w_1)





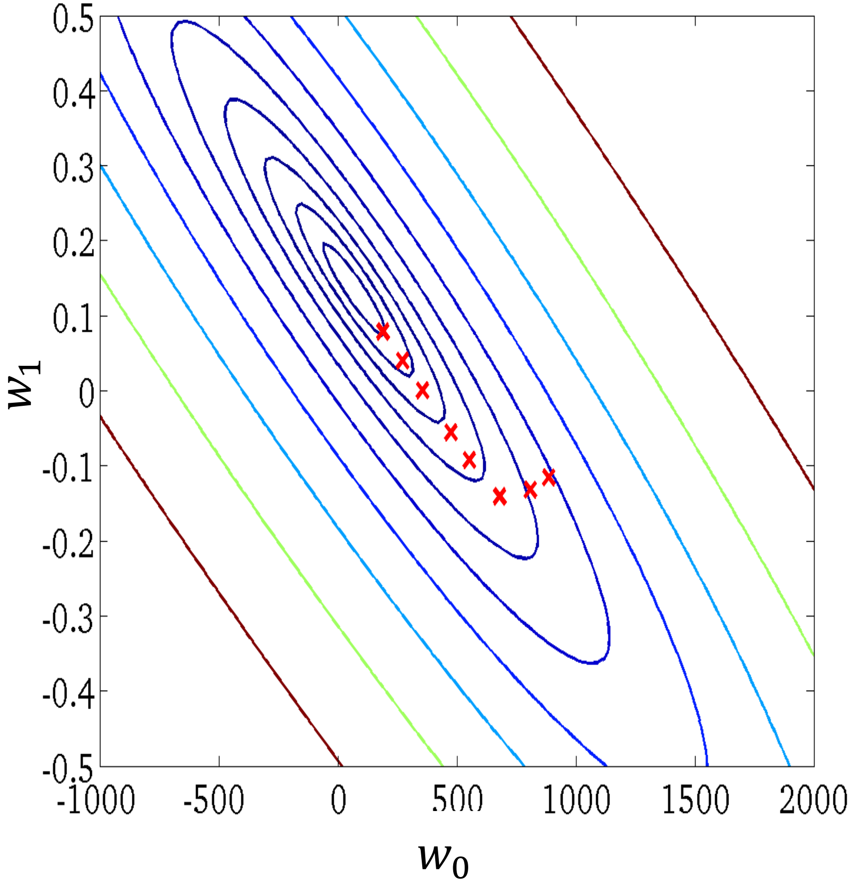
 This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



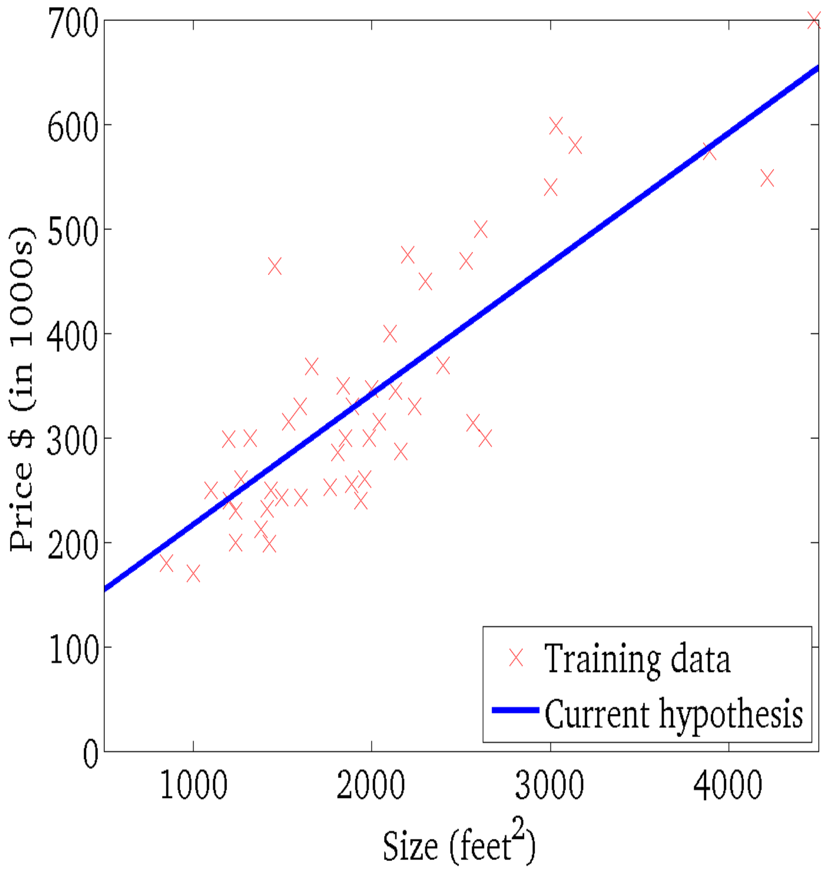
$J(w_0, w_1)$

(function of the parameters w_0, w_1)



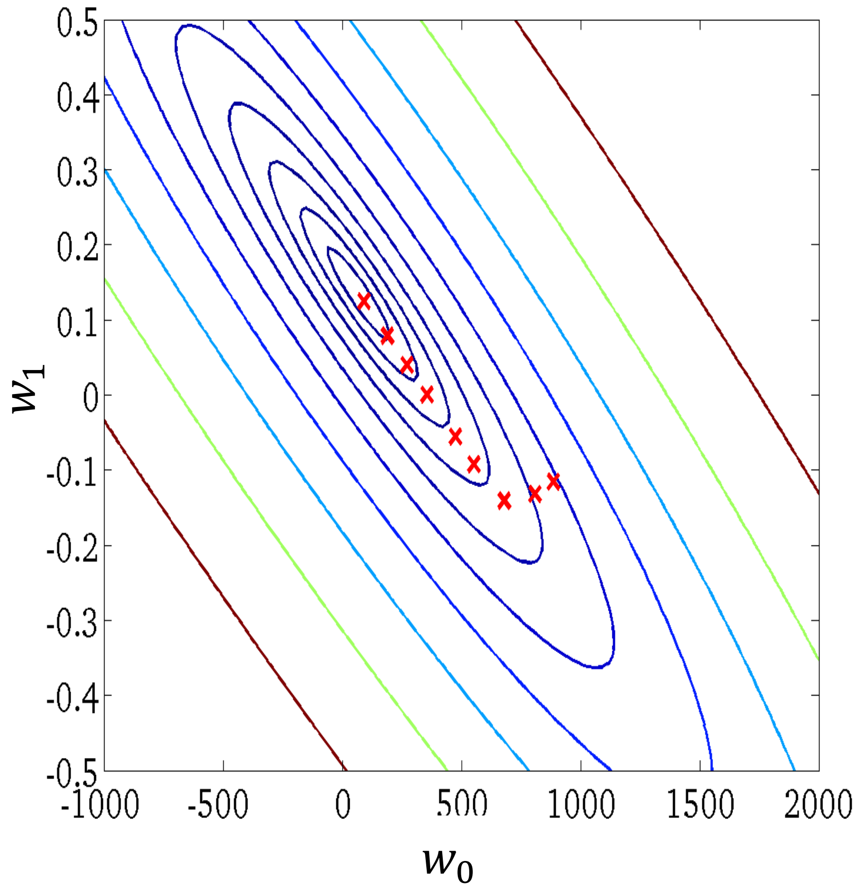
 This example has been adopted from: Prof. Ng's slides (ML Online Course, Stanford)

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



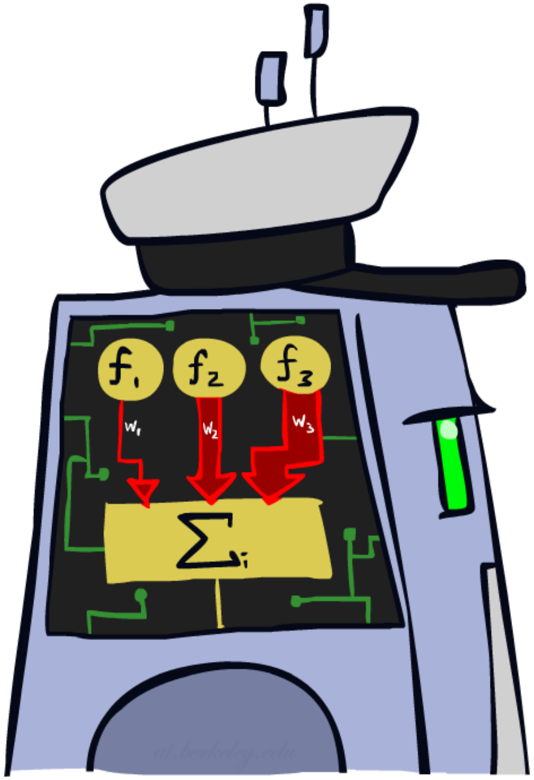
$$J(w_0, w_1)$$

(function of the parameters w_0, w_1)



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

Linear Classifiers



Error-Driven Classification

Feature Vectors

x y

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

→

```
# free         : 2
YOUR_NAME      : 0
MISSPELLED     : 2
FROM_FRIEND    : 0
...
```

→ SPAM or +

→

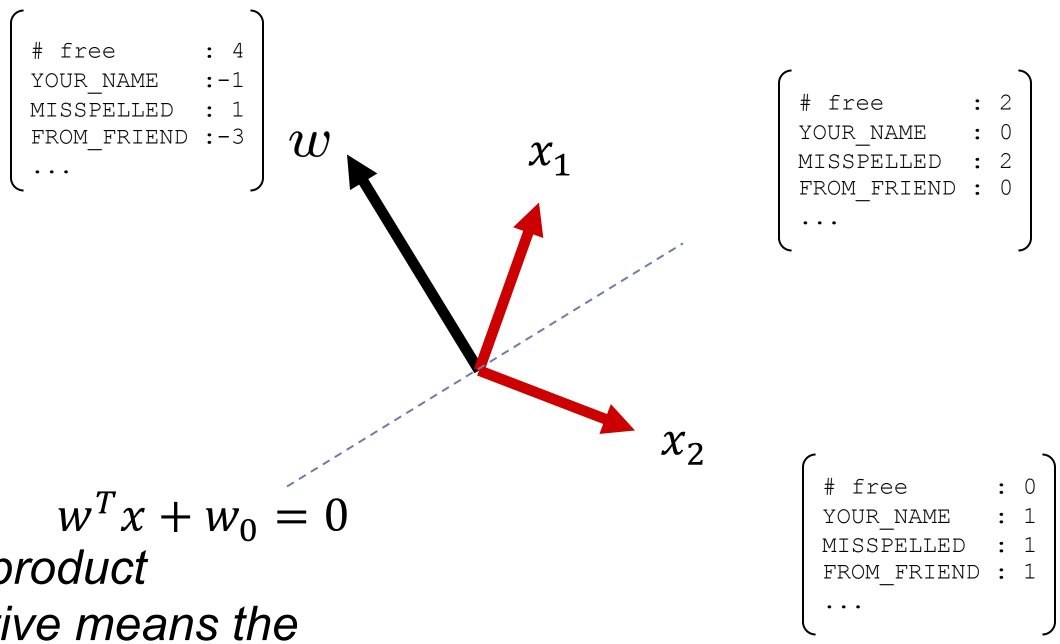
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

→ "2"

We show input by $\boldsymbol{x}$ or $f(\boldsymbol{x})$

Weights

- Binary case: compare features to a weight vector to identify the class
- Learning: figure out the weight vector from examples



$$w^T x + w_0 = 0$$

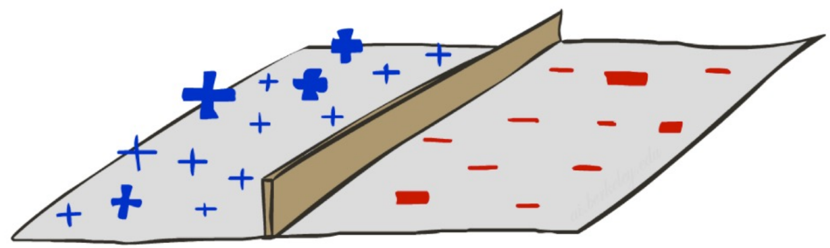
Dot product positive means the positive class

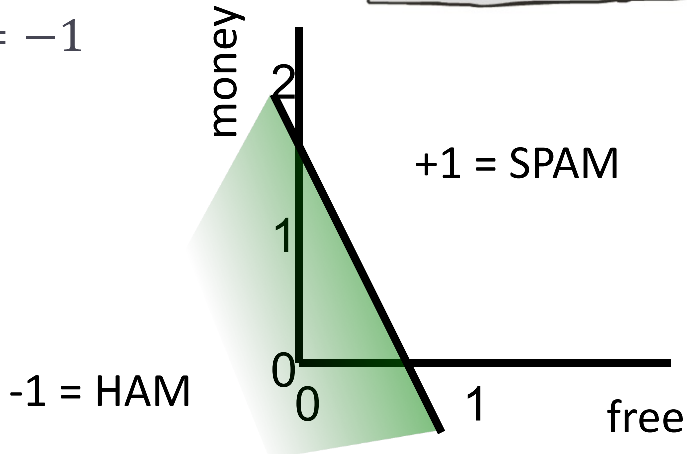
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to $\hat{y} = +1$
  - Other corresponds to $\hat{y} = -1$

w

```
BIAS  : -3
free  :  4
money :  2
...
```



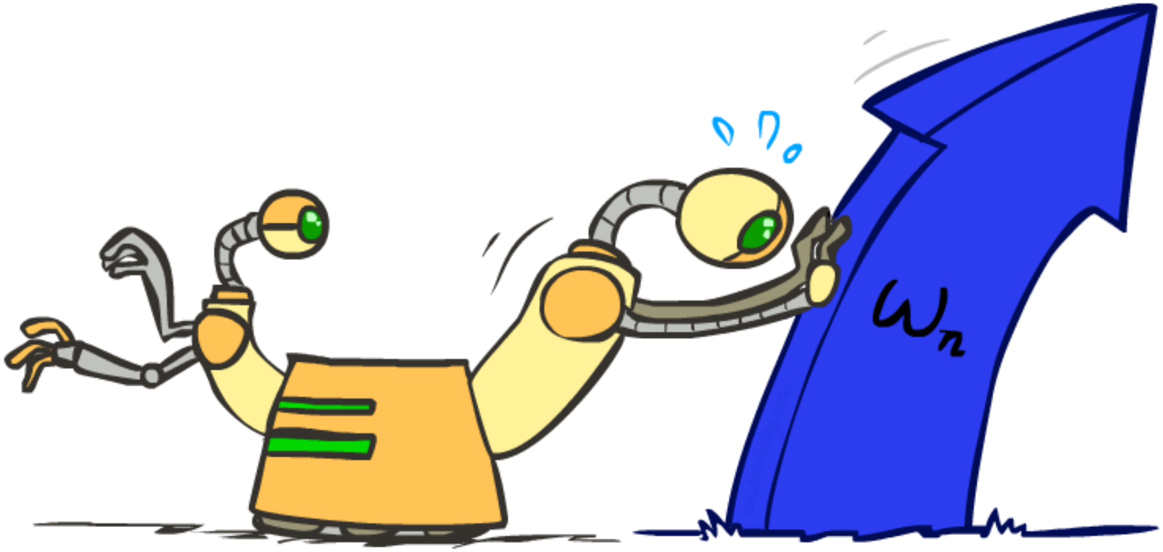


$\boldsymbol{x}^T\boldsymbol{w} = 0$

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

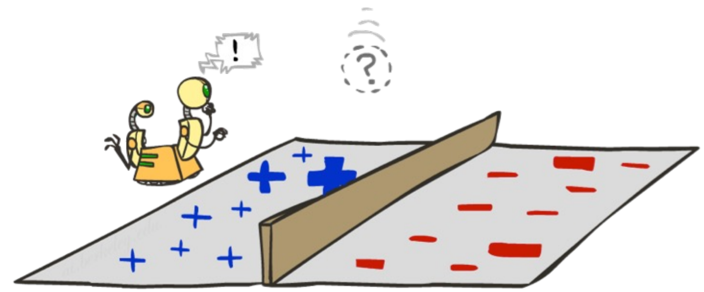
$$\boldsymbol{x}^T\boldsymbol{w} = w_0 + w_1x_1 + \ldots w_dx_d$$

Weight Updates

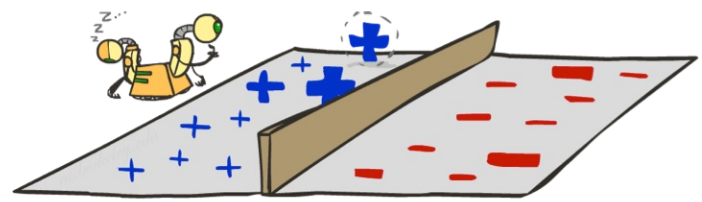


Learning: Binary Perceptron

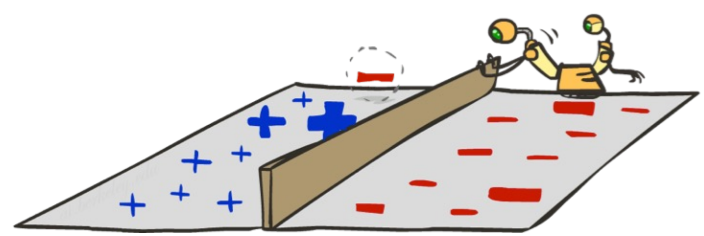
- Start with weights = 0
- For each training instance:
  - Classify with current weights



  - If correct (i.e., $\hat{y} = y$), no change!

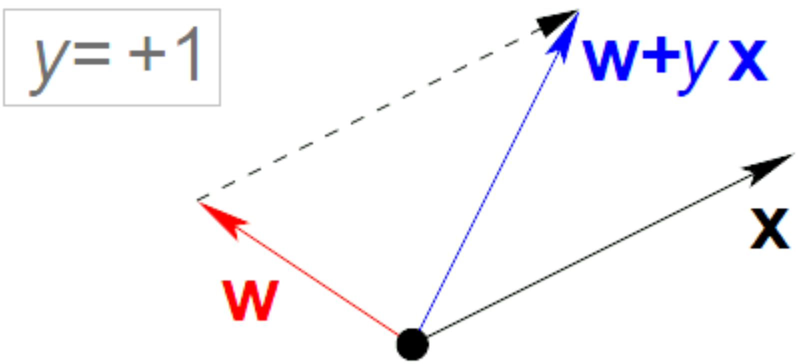


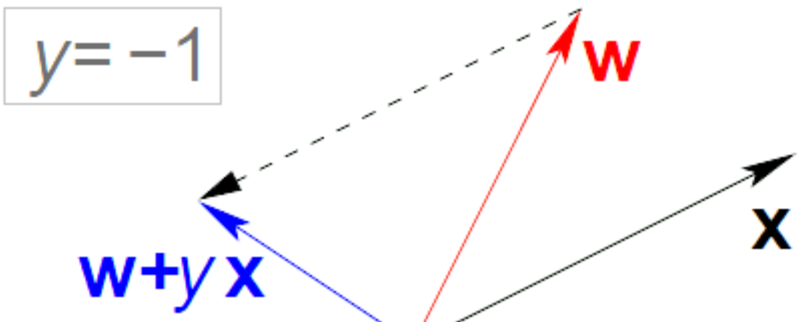
  - If wrong: adjust the weight vector



$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \boldsymbol{x}^{(i)} y^{(i)}$$

Perceptron: Example

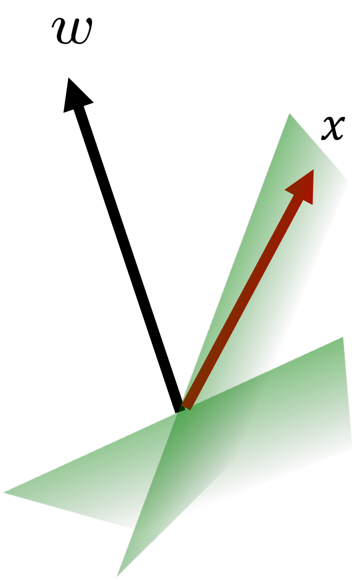




Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$\hat{y} = \begin{cases} +1 & \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1 & \boldsymbol{w}^T\boldsymbol{x} < 0 \end{cases}$$



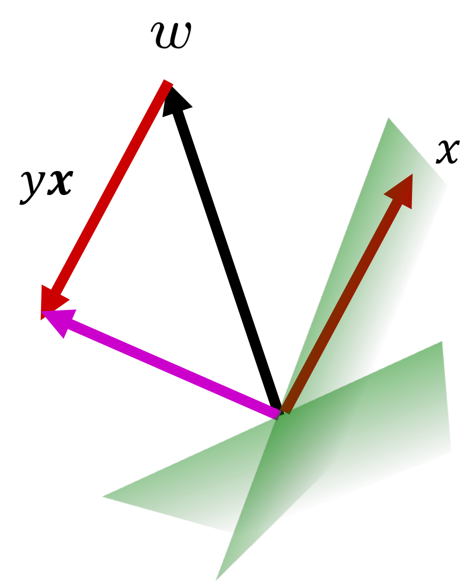
Learning: Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$\hat{y} = \begin{cases} +1 & \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ -1 & \boldsymbol{w}^T\boldsymbol{x} < 0 \end{cases}$$

  - If correct (i.e., $\hat{y} = y$), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector (subtract if y is -1):

$$\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}y$$



Perceptron criterion

- Two-class: $y \in \{-1, 1\}$
  - $y = -1$ for C_2, $\quad y = 1$ for C_1

- Goal: $\forall i, \boldsymbol{x}^{(i)} \in C_1 \;\Rightarrow \boldsymbol{w}^T \boldsymbol{x}^{(i)} > 0$
  $\forall i, \boldsymbol{x}^{(i)} \in C_2 \;\Rightarrow \boldsymbol{w}^T \boldsymbol{x}^{(i)} < 0$

$$J_P(\boldsymbol{w}) = -\sum_{i \in \mathcal{M}} \boldsymbol{w}^T \boldsymbol{x}^{(i)} y^{(i)}$$

$\mathcal{M}$: subset of training data that are misclassified

Many solutions? Which solution among them?

Batch Perceptron

"Gradient Descent" to solve the optimization problem:

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

$$\nabla_{\boldsymbol{w}} J_P(\boldsymbol{w}) = -\sum_{i \in \mathcal{M}} \boldsymbol{x}^{(i)} y^{(i)}$$

Batch Perceptron converges in finite number of steps for linearly separable data:

Initialize $\boldsymbol{w}$
Repeat
    $\boldsymbol{w} = \boldsymbol{w} + \eta \sum_{i \in \mathcal{M}} \boldsymbol{x}^{(i)} y^{(i)}$
Until convergence

Stochastic Gradient Descent for Perceptron

- Single-sample perceptron:
  - If $\boldsymbol{x}^{(i)}$ is misclassified:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^{t} + \eta \boldsymbol{x}^{(i)} y^{(i)}$$

- Perceptron convergence theorem: for linearly separable data
  - If training data are linearly separable, the single-sample perceptron is also guaranteed to find a solution in a finite number of steps

Fixed-Increment single sample Perceptron

η can be set to 1 and proof still works →

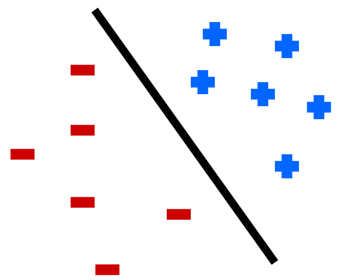
Initialize $\boldsymbol{w}, t \leftarrow 0$
repeat
  $t \leftarrow t + 1$
  $i \leftarrow t \bmod N$
  if $\boldsymbol{x}^{(i)}$ is misclassified then
    $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}^{(i)} y^{(i)}$
Until all patterns properly classified

Properties of Perceptrons

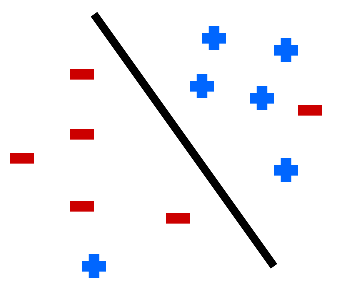
- Separability: true if some parameters get the training set perfectly classified
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable



Non-Separable



Multiclass Decision Rule

- If we have multiple classes:
  - A weight vector for each class:

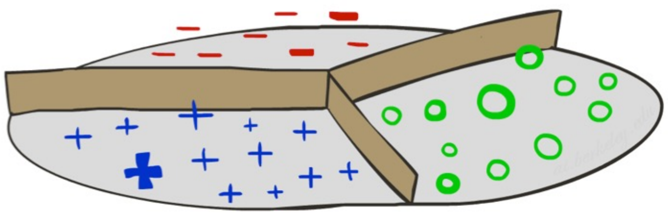
$$w_y$$

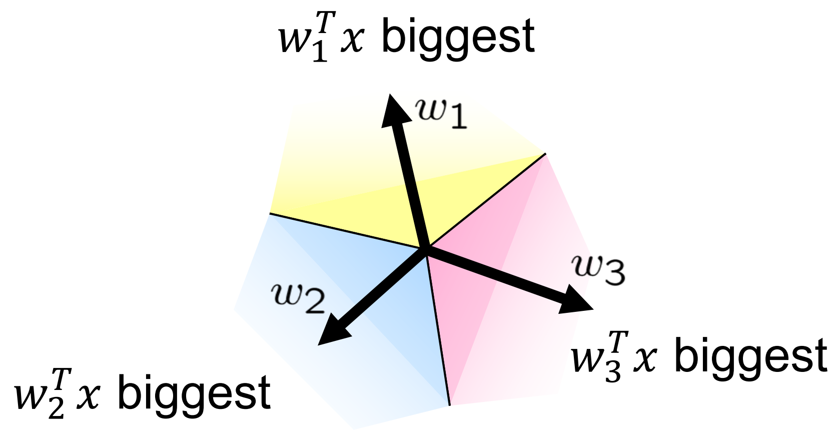
  - Score (activation) of a class y:

$$w_y^T x$$

  - Prediction highest score wins

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, w_y^T x$$

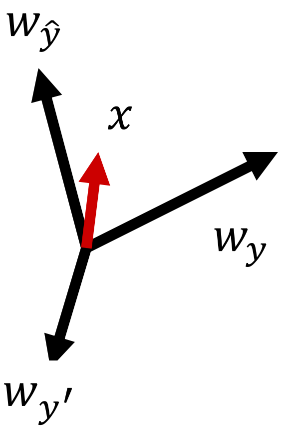




Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights



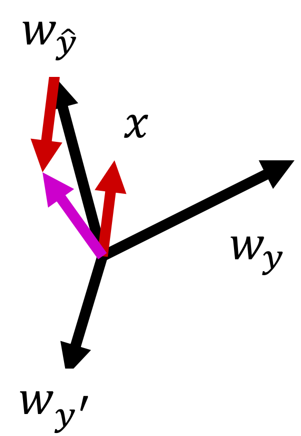
Learning: Multiclass Perceptron

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$\hat{y} = \operatorname*{argmax}_y w_y^T x$$

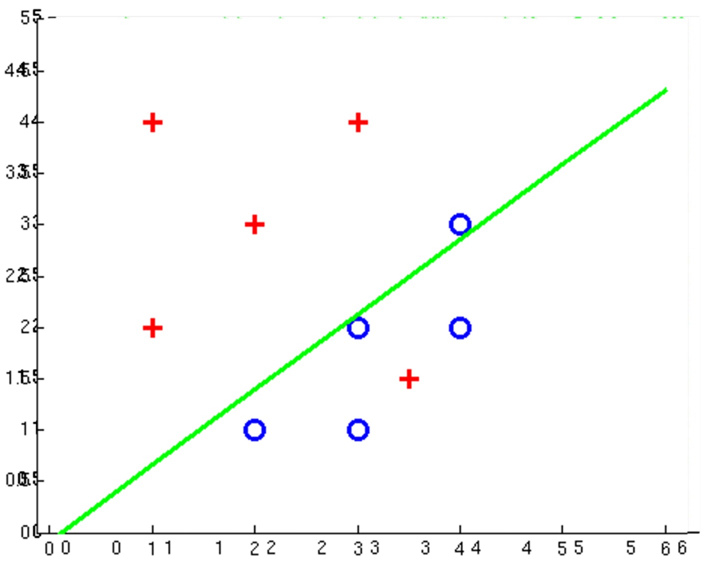
- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_{\hat{y}} = w_{\hat{y}} - x$$
$$w_y = w_y + x$$



Examples: Perceptron

- Non-Separable Case



Logistic Regression

$K = 2$

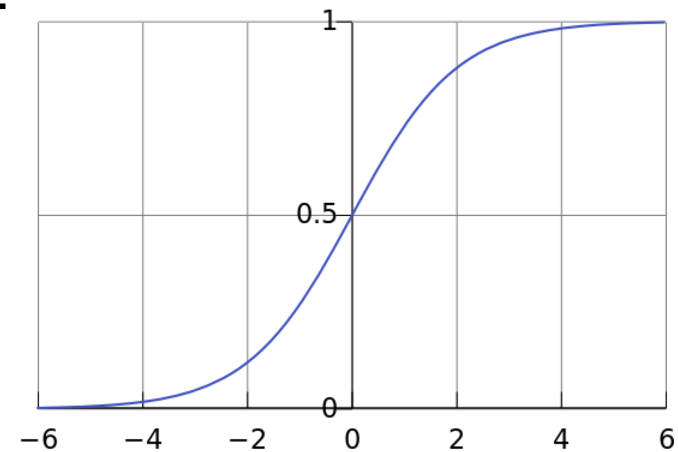
$$g(\boldsymbol{x}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

$\boldsymbol{x} = [1, x_1, \dots, x_d]$
$\boldsymbol{w} = [w_0, w_1, \dots, w_d]$

$\sigma(.)$ is an activation function

- Sigmoid (logistic) function
  - Activation function

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



Logistic Regression: Cost Function

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, J(\boldsymbol{w})$$

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} -y^{(i)} \log\left(\sigma\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)\right) - (1 - y^{(i)}) \log\left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)\right)$$

- $J(\boldsymbol{w})$ is convex w.r.t. parameters.

Logistic Regression: Loss Function

$$\text{Loss}\big(y, f(\boldsymbol{x}; \boldsymbol{w})\big) = -y \times \log\big(\sigma(\boldsymbol{x}; \boldsymbol{w})\big) - (1-y) \times \log(1 - \sigma(\boldsymbol{x}; \boldsymbol{w}))$$

Since $y = 1$ or $y = 0$
$\Rightarrow$

$$\text{Loss}\big(y, \sigma(\boldsymbol{x}; \boldsymbol{w})\big) = \begin{cases} -\log(\sigma(\boldsymbol{x}; \boldsymbol{w})) & \text{if } y = 1 \\ -\log(1 - \sigma(\boldsymbol{x}; \boldsymbol{w})) & \text{if } y = 0 \end{cases}$$

How is it related to zero-one loss?

$$\text{Loss}(y, \hat{y}) = \begin{cases} 1 & y \neq \hat{y} \\ 0 & y = \hat{y} \end{cases}$$

$$\sigma(\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{1 + exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

Logistic Regression: Gradient Descent

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

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\sigma\left(\boldsymbol{w}^{T} \boldsymbol{x}^{(i)}\right) - y^{(i)}\right) \boldsymbol{x}^{(i)}$$

- Is it similar to gradient of SSE for linear regression?

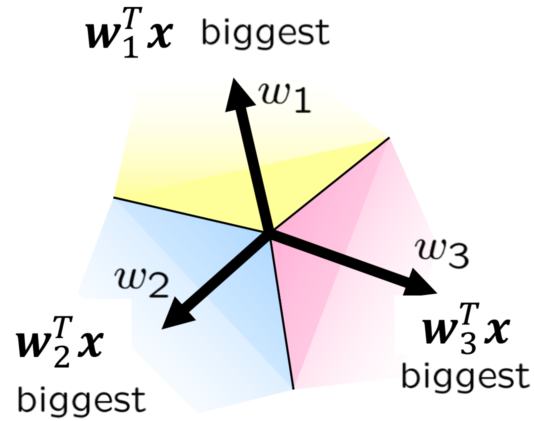
$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\boldsymbol{w}^{T} \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$$

Multi-class Classifier

$\boldsymbol{w}_k$

$s_k = \boldsymbol{w}_k^T \boldsymbol{x}$

$\hat{y} = \underset{k}{\operatorname{argmax}}\, \boldsymbol{s_k}$



Multi-class Classifier

- $\boldsymbol{W} = [\boldsymbol{w}_1 \quad \cdots \quad \boldsymbol{w}_K]$ 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

- $g(\boldsymbol{x}; \boldsymbol{W})$ contains K numbers giving class scores for the input $\boldsymbol{x}$
  - $g(\boldsymbol{x}; \boldsymbol{W}) = [g_1(\boldsymbol{x}, \boldsymbol{W}), \dots, g_K(\boldsymbol{x}, \boldsymbol{W})]^T$

Multi-class Logistic Regression

- $g(\boldsymbol{x};\boldsymbol{W}) = [g_1(\boldsymbol{x},\boldsymbol{W}),\dots,g_K(\boldsymbol{x},\boldsymbol{W})]^T$

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

$$g_k(\boldsymbol{x};\boldsymbol{W}) = \frac{\exp(\boldsymbol{w}_k^T\boldsymbol{x}\ )}{\sum_{j=1}^{K}\exp(\boldsymbol{w}_j^T\boldsymbol{x}\ )}$$

- This is the softmax on $\boldsymbol{s} = [s_1,\dots,s_K]^T = [\boldsymbol{w}_1^T\boldsymbol{x},\dots,\boldsymbol{w}_K^T\boldsymbol{x}] = \boldsymbol{W}^T\ \boldsymbol{x}$

Logistic Regression: Multi-class

$$\widehat{\boldsymbol{W}} = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

$$J(\boldsymbol{W}) = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_k^{(i)} \log\left(g_k\left(\boldsymbol{x}^{(i)};\boldsymbol{W}\right)\right)$$

$\boldsymbol{y}$ is a vector of length K (1-of-K coding)
e.g., $\boldsymbol{y} = [0,0,1,0]^T$ when the target class is C_3

$$\boldsymbol{W} = [\boldsymbol{w}_1 \quad \cdots \quad \boldsymbol{w}_K]$$

Logistic Regression: Multi-class

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

$$\nabla_{\boldsymbol{w}_j} J(\boldsymbol{W}) = \sum_{i=1}^{n} \left(g_j\left(\boldsymbol{x}^{(i)}; \boldsymbol{W}\right) - y_j^{(i)} \right) \boldsymbol{x}^{(i)}$$

Logistic Regression: Probabilistic Perspective

Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:

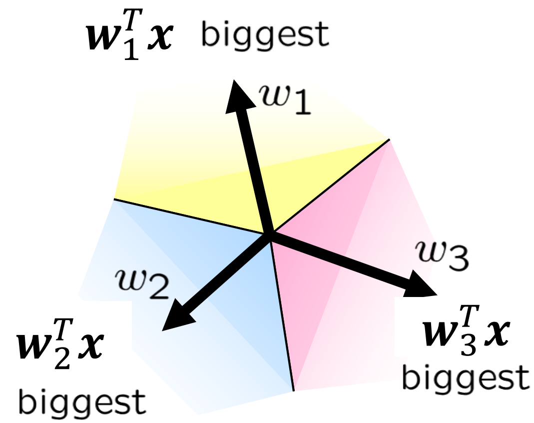
$$P\left(y^{(i)} = +1 \middle| x^{(i)}; w\right) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}}$$

$$P\left(y^{(i)} = 0 \middle| x^{(i)}; w\right) = 1 - \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}}$$

= Two-class Logistic Regression

Multiclass Logistic Regression

- Multi-class linear classification
  - A weight vector for each class: $\boldsymbol{w}_k$
  - Score (activation) of a class y: $z_k = \boldsymbol{w}_k^T \boldsymbol{x}$
  - Prediction w/highest score wins $\hat{y} = \underset{k}{\operatorname{argmax}} \, \boldsymbol{w}_k^T \boldsymbol{x}$



- How to make the scores into probabilities?

$$\underbrace{z_1, z_2, z_3}_{\text{original activations}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{softmax activations}}$$

Logistic Regression: Probabilistic Perspective

Maximum likelihood estimation:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)};w)$$

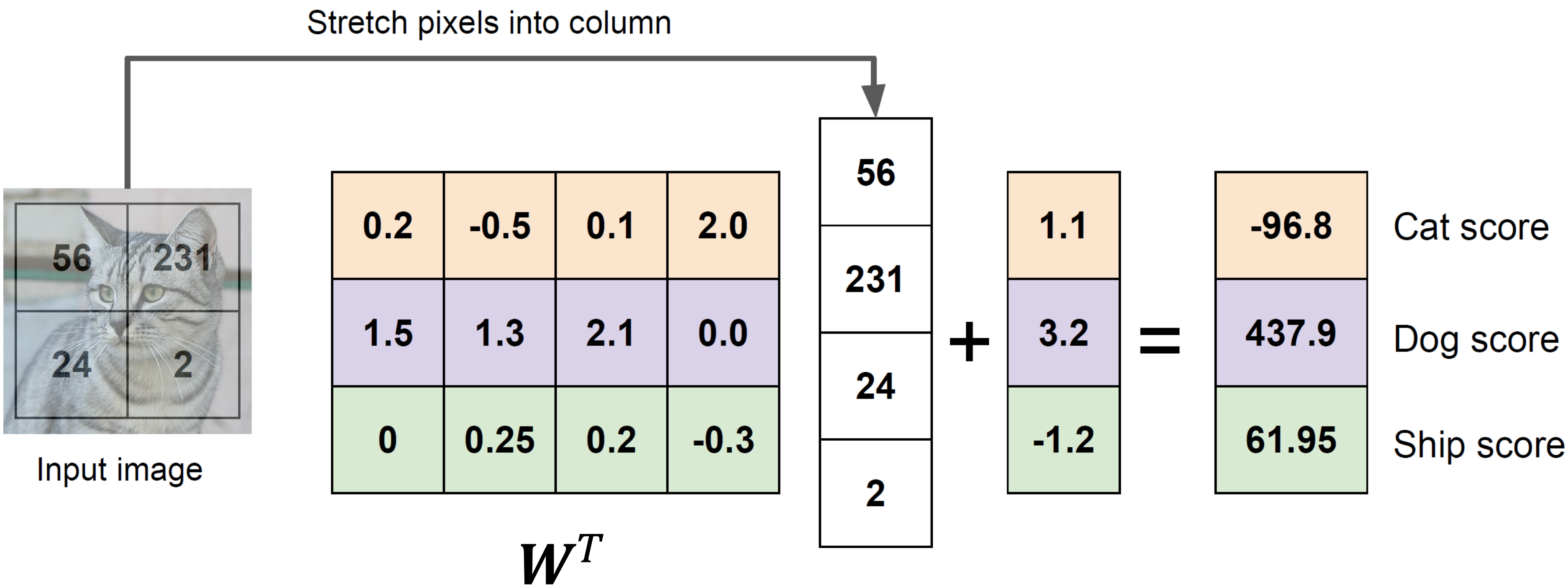
with:

$$P\left(y^{(i)}\middle|x^{(i)};w\right) = \frac{e^{\boldsymbol{w}_{y^{(i)}}^T \boldsymbol{x}^{(i)}}}{\sum_{k=1}^{K} e^{\boldsymbol{w}_k^T \boldsymbol{x}^{(i)}}}$$

= Multi-Class Logistic Regression

Example

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_4 \end{bmatrix} \qquad \boldsymbol{W}^T = \begin{bmatrix} \boldsymbol{w}_1 \\ \boldsymbol{w}_2 \\ \boldsymbol{w}_3 \end{bmatrix}_{3\times4} \qquad \boldsymbol{w_0} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$



How can we tell whether this $\boldsymbol{W}$ and $\boldsymbol{w}_0$ is good or bad?

This slide has been adopted from Fei Fei Li and colleagues lectures, cs231n, Stanford 2017

Softmax Classifier Loss: Example

$$s_k = \boldsymbol{w}_k^T \boldsymbol{x}$$

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

unnormalized probabilities

		exp		normalize	
cat	**3.2**	→	**24.5**	→	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

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

This slide has been adopted from Fei Fei Li and colleagues lectures, cs231n, Stanford 2017

Summary

- Linear regression
  - Sum of Squares Error (SSE)
  - Gradient descent

- Linear classification
  - Perceptron
  - Logistic regression