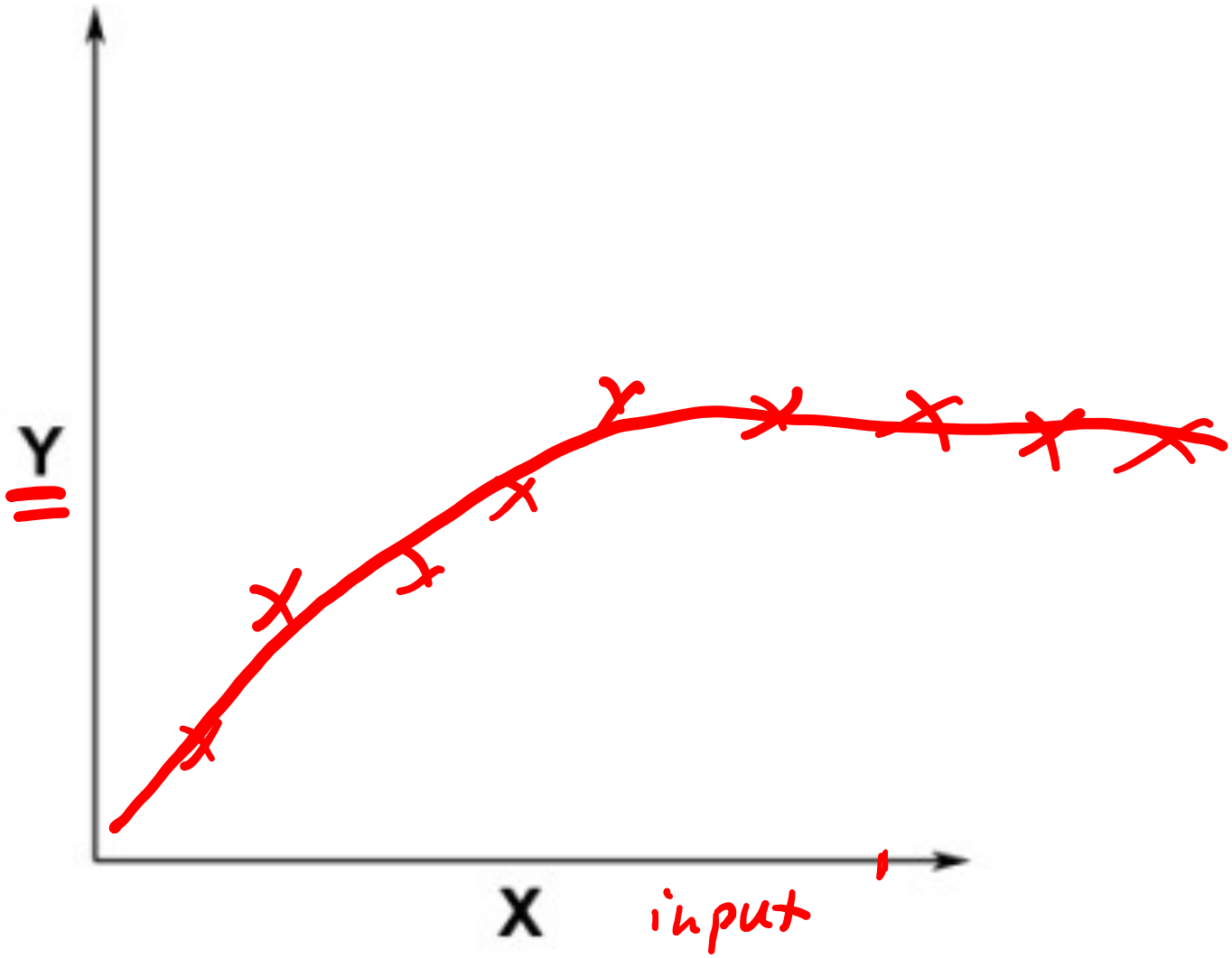
深度学习

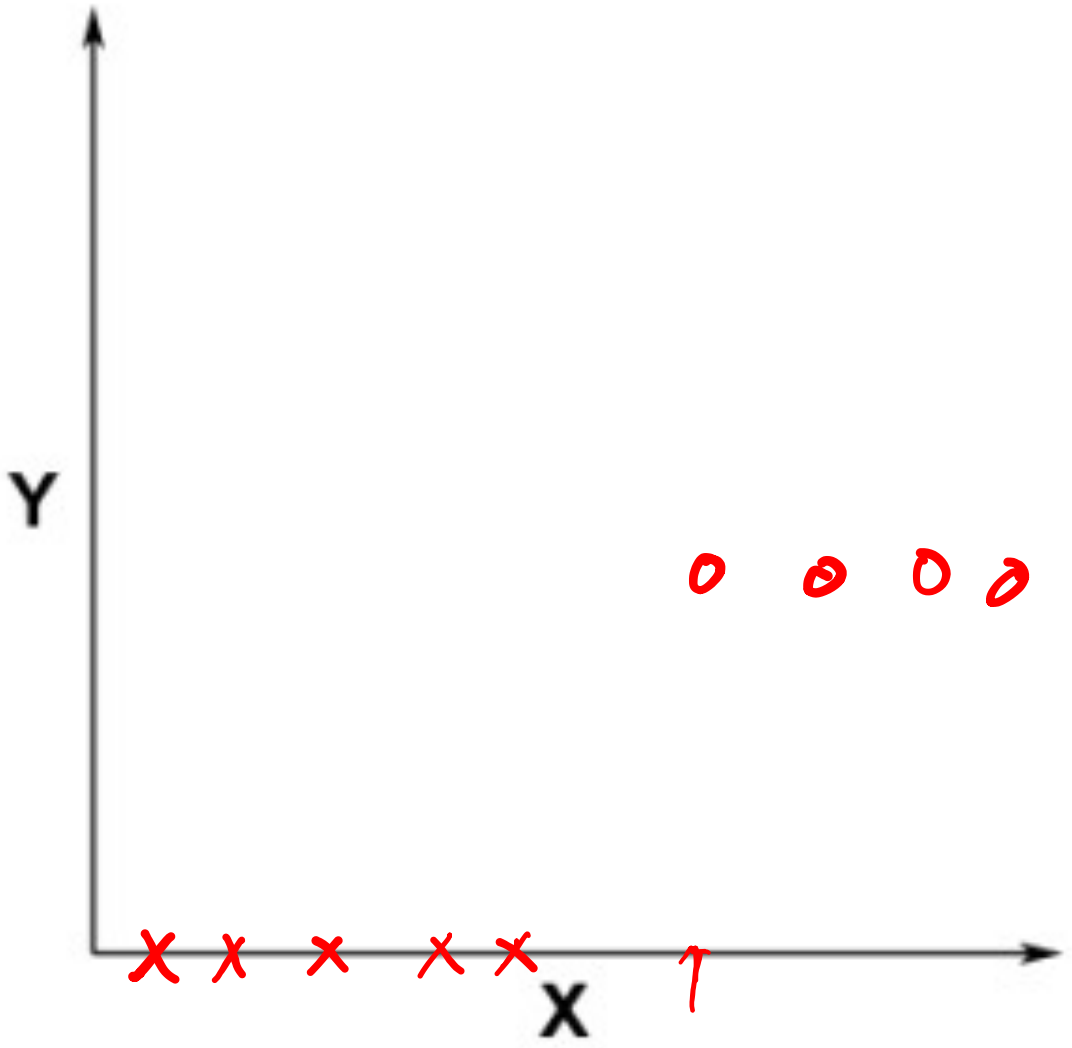
第二讲

王胤

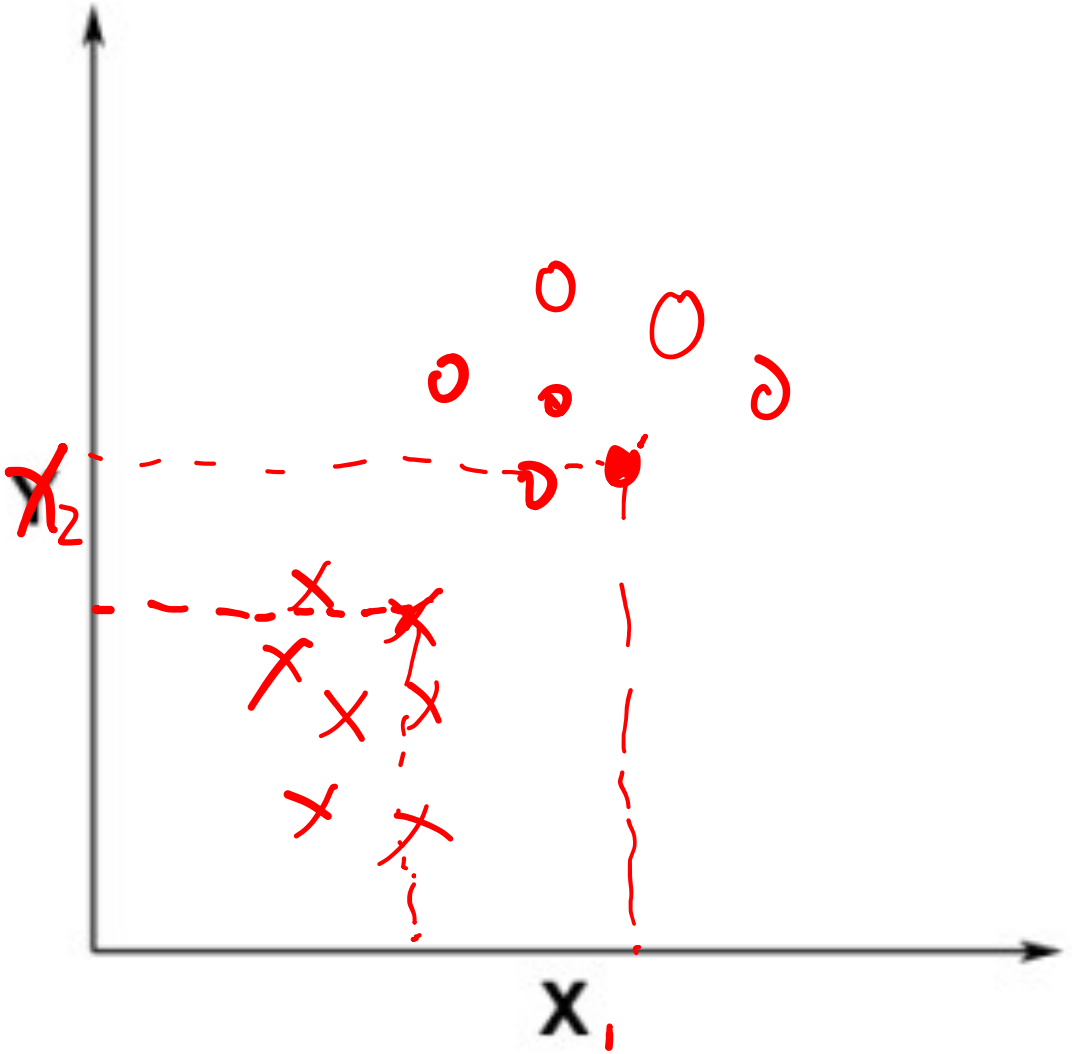
Regression



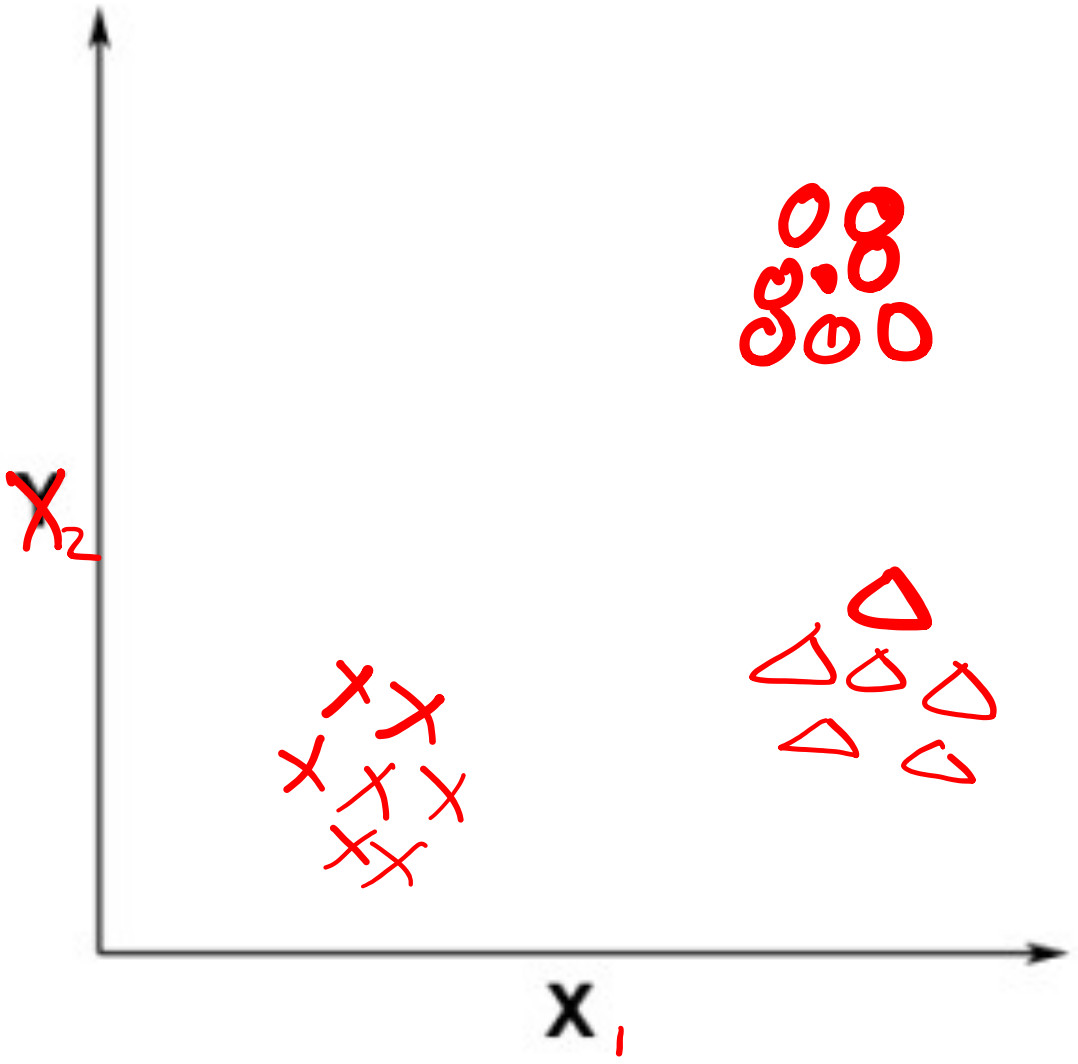
Classification



Classification



Unsupervised



Linear Regression

- Training set (this is your data set)
- Notation (*used throughout the course*)
  - m = number of **training examples**
  - x's = input variables / features
  - t's = output variable "target" variables
    - (x, t) - single training example
    - (x^i, t^i) - specific example (i^{th} training example)

Size in feet² (x)	Price ($) in 1000's (t)
2104	460
1416	232
1534	315
852	178
...	...

m = 47

Hypothesis, Cost Function

$$y_\omega(x) = \omega_0 + \omega_1 x$$

$$C(\omega) = \frac{1}{2m} \sum \left(y_\omega(x^{(i)}) - t^{(i)} \right)^2$$

$$\min_\omega C(\omega)$$

Assume $\omega_0 = 0$

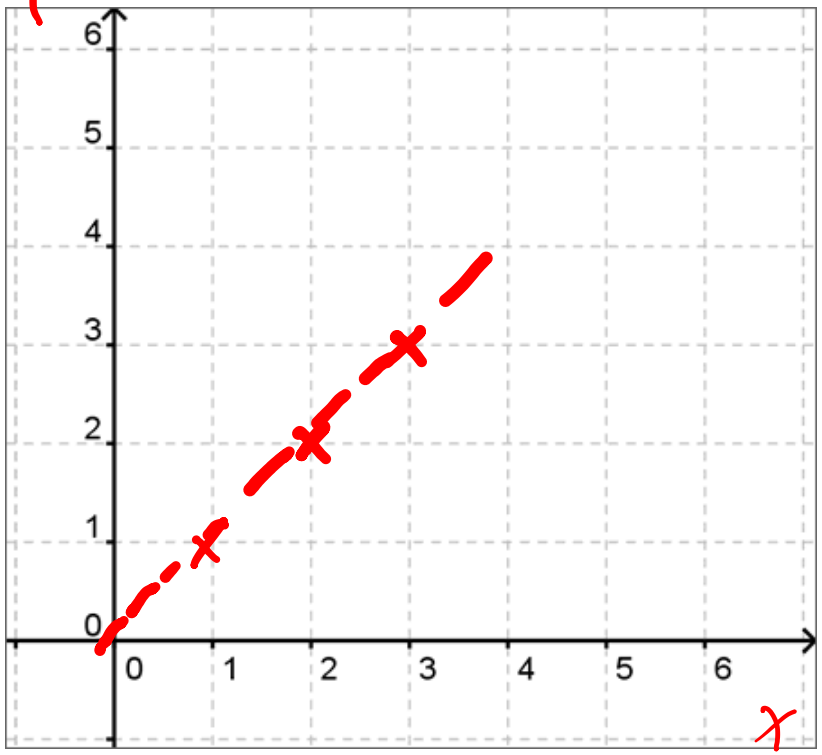
$$y_\omega(x) = \omega_1 x$$

$$C(\omega_1) = \frac{1}{2m}\sum_{i=1}^{m}(y_\omega(x^{(i)}) - t^{(i)})^2$$

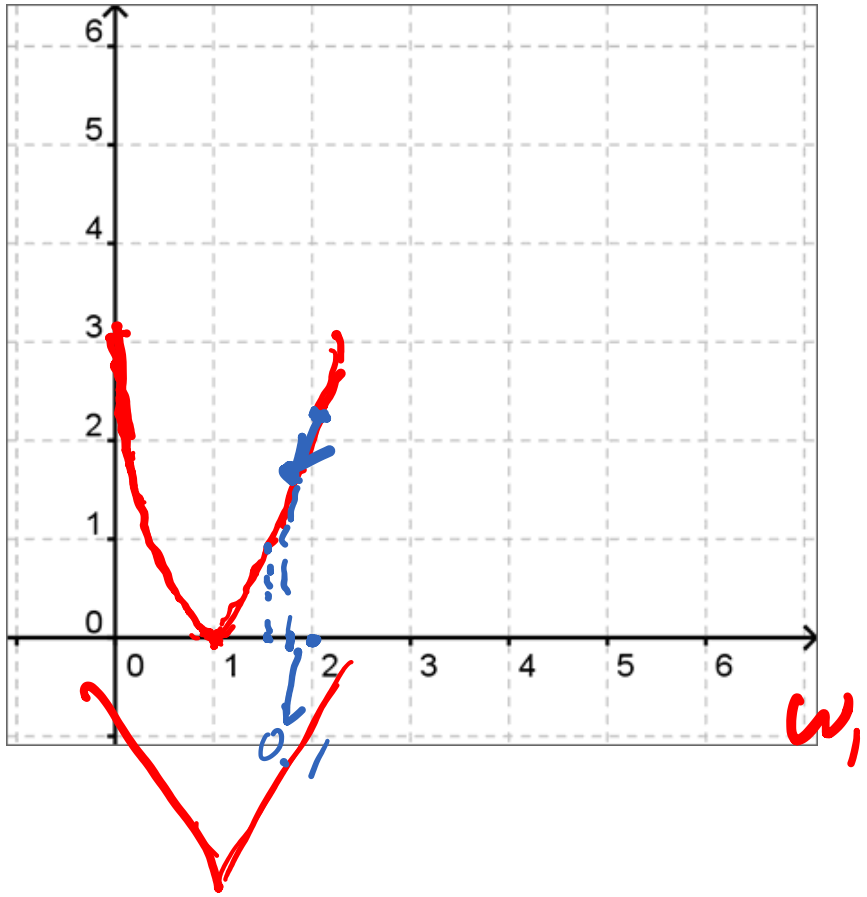
minimize $C(\omega_1)$

$y_\omega(x)$ vs. $C(\omega_1)$

$y_\omega(x) = \omega_1 x$

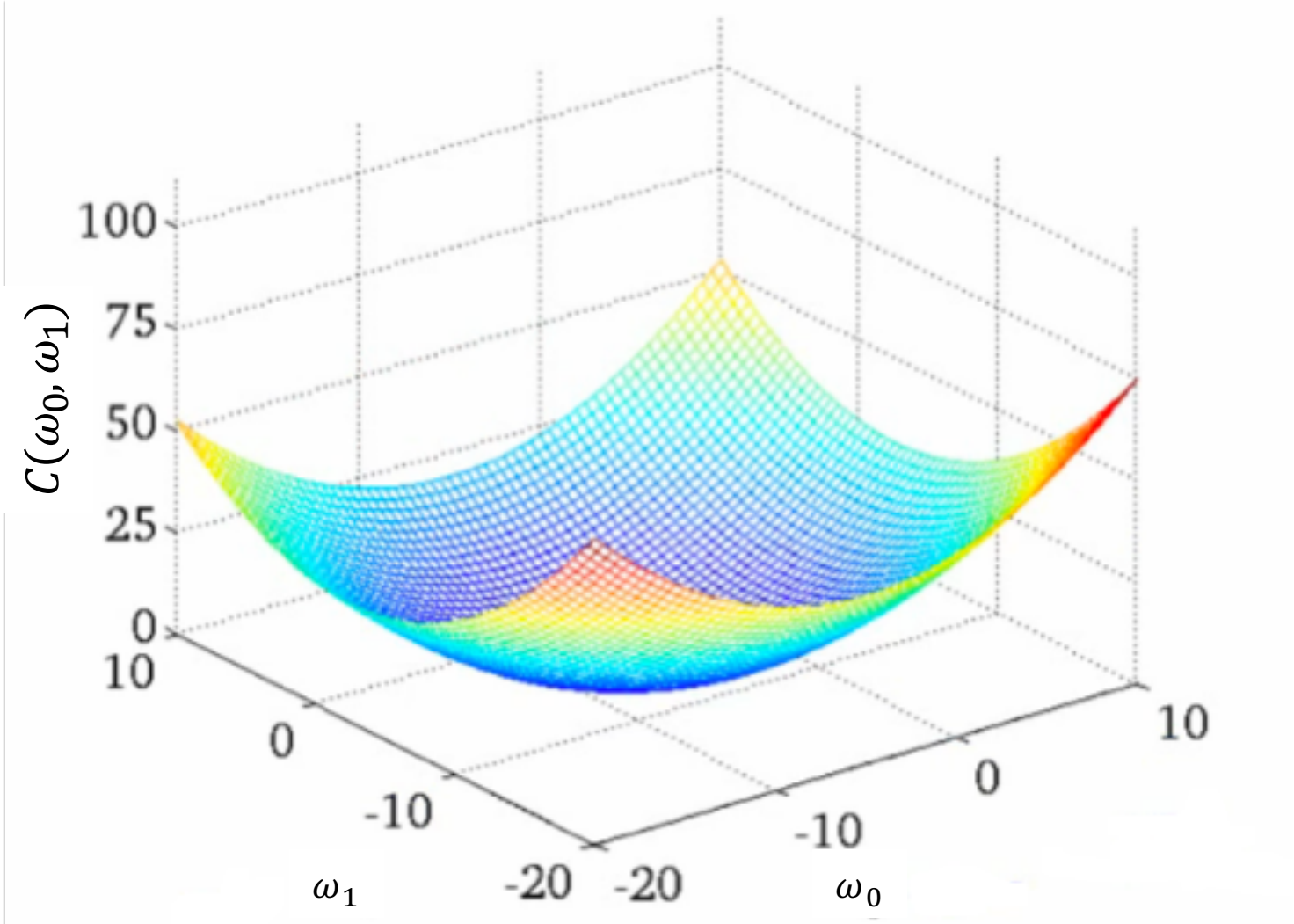


$C(\omega_1)$



$C(\omega_0, \omega_1)$

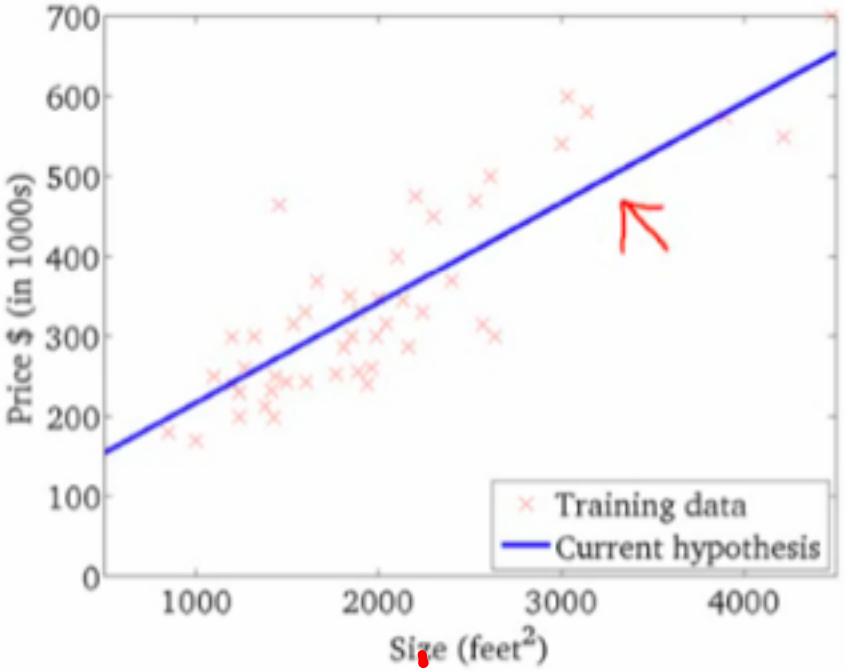
$y = w_0 + w_1 x$



$y_\omega(x)$ vs. $C(\omega_0, \omega_1)$

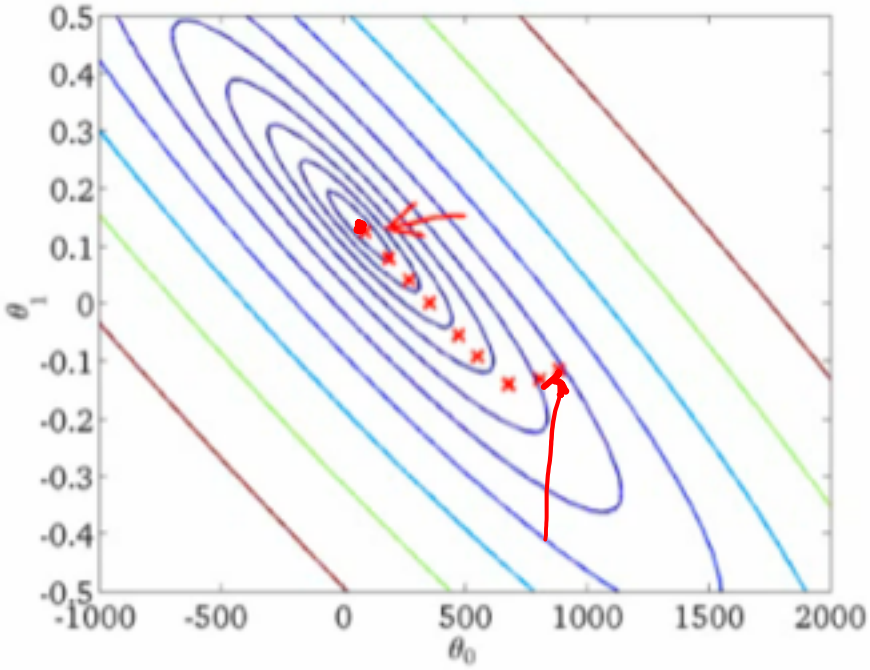
$y_\omega(x)$

(for fixed ω_0, ω_1, this is a function of x)



$C(\omega_0, \omega_1)$

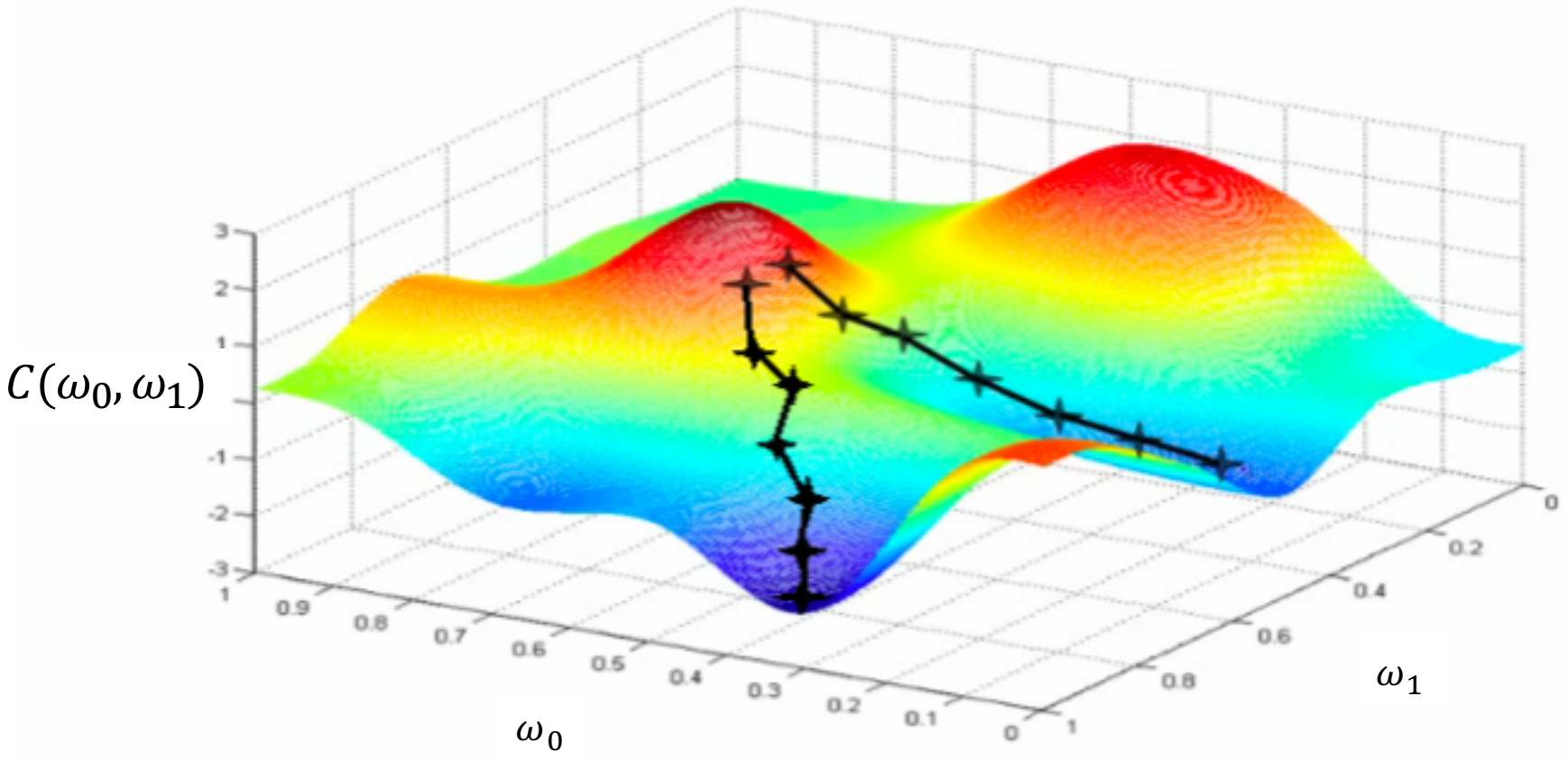
(function of the parameters ω_0, ω_1)



Gradient Descent Algorithm

- Problem
  - We have $C(\omega_0, \omega_1)$
  - We want to get **min** $\boldsymbol{C(\omega_0, \omega_1)}$
- Gradient descent applies to more general functions
  - $C(\omega_0, \omega_1, \ldots \omega_n)$
  - min $C(\omega_0, \omega_1, \ldots \omega_n)$
- Start with initial guesses
  - Start at 0,0 (or any other value)
  - Keeping changing ω_0 and ω_1 a little bit to try and reduce $C(\omega_0, \omega_1)$
- Each time you change the parameters, you select the gradient which reduces $C(\omega_0, \omega_1)$ the most possible
- Repeat until you converge to a local minimum

Local Minimum



A More Formal Definition

- Do the following until convergence

$$\omega_j := \omega_j - \alpha \frac{\partial}{\partial \omega_j} C(\omega_0, \omega_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

- There is a subtly about how this gradient descent algorithm is implemented

$$\text{temp0} := \omega_0 - \alpha \frac{\partial}{\partial \omega_0} C(\omega_0, \omega_1)$$

$$\text{temp1} := \omega_1 - \alpha \frac{\partial}{\partial \omega_1} C(\omega_0, \omega_1)$$

$$\omega_0 := \text{temp0}$$

$$\omega_1 := \text{temp1}$$

Linear Regression Case

$$\frac{\partial}{\partial \omega_j} C(\omega_0, \omega_1) = \frac{\partial}{\partial w_j} \frac{1}{2m} \sum \left(w_0 + w_1 x^i - t^i \right)^2$$

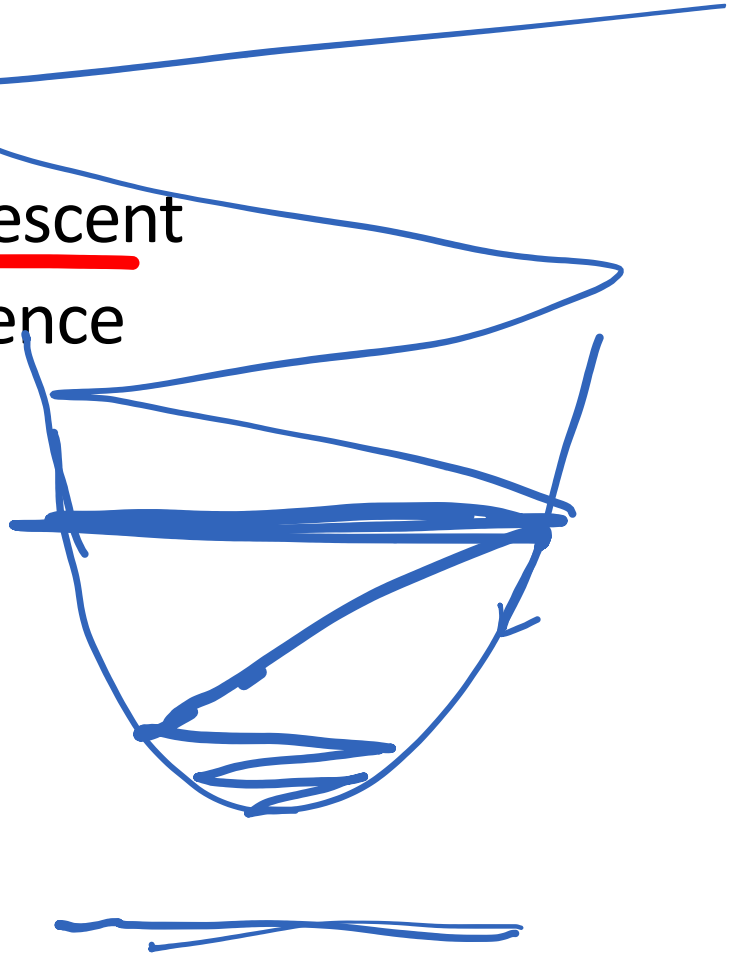
$$= \frac{1}{m} \sum \frac{\partial}{\partial w_j} \left(w_0 + w_1 x^i \right) \cdot \left(y(x) - t \right)$$

~~$w_0 \rightarrow w_0 - \alpha \sum \left(y(x^i) - t^i \right)$~~

$$w_i \rightarrow w_i - \alpha \left(\sum_i \left(y(x^i) - t^i \right) \cdot x_i^i \right)$$

Practical Issues

- Global vs. local minimum
- Single vs. batch gradient descent
- Learning rate and convergence



Multiple Variables

- Multiple variables = multiple features
- In original version we had
  - X = house size, use this to predict
  - t = house price
- If in a new scheme we have more variables (such as number of bedrooms, number floors, age of the home)
  - x_1, x_2, x_3, x_4 are the four features
    - x_1 - size (feet squared)
    - x_2 - Number of bedrooms
    - x_3 - Number of floors
    - x_4 - Age of home (years)
  - t is the output variable (price)

More Notation

- **n**
  - number of features (n = 4)
- **m**
  - number of examples (i.e. number of rows in a table)
- **x^i**
  - vector of the input for an example (so a vector of the four parameters for the i^{th} input example)
  - i is an index into the training set
  - So
    - x is an n-dimensional feature vector
    - x^3 is, for example, the 3rd house, and contains the four features associated with that house
- **x_j^i**
  - The value of feature j in the ith training example
  - So
    - x_2^3 is, for example, the number of bedrooms in the third house

Hypothesis

- Previously our hypothesis took the form;
  - $y_\omega(x) = \omega_0 + \omega_1 x$
    - Here we have two parameters (theta 1 and theta 2) determined by our cost function
    - One variable x
- Now we have multiple features
  - $y_\omega(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4$
- For example
  - $y_\omega(x) = 80 + 0.1x_1 + 0.01x_2 + 3x_3 - 2x_4$
    - An example of a hypothesis which is trying to predict the price of a house
    - Parameters are still determined through a cost function

Vector Form

- For convenience of notation, $x_0 = 1$
- Considering this, hypothesis can be written

$$y_\omega(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \omega_4 x_4$$

- More conveniently

$$y_\omega(x) = \omega^T x$$

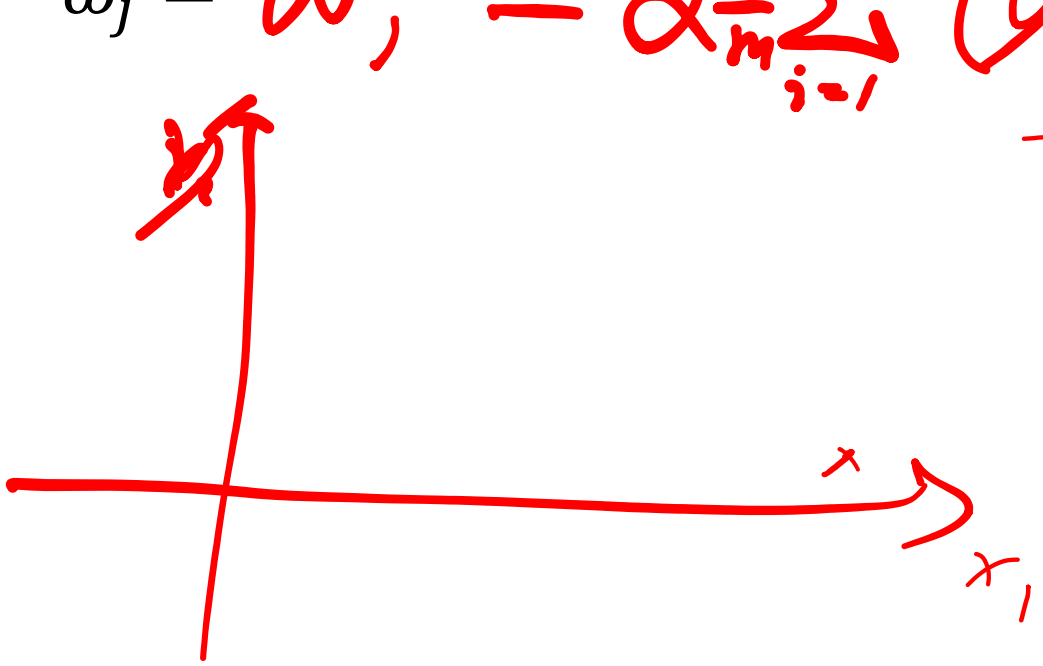
$$y = \omega^T x$$

Gradient Descent For Multiple Variables

- Our cost function is

$$C(\omega_0, \omega_1, \ldots, \omega_n) = \frac{1}{2m}\sum_{i=1}^{m}\left(y_\omega\left(x^{(i)}\right) - t^{(i)}\right)^2$$

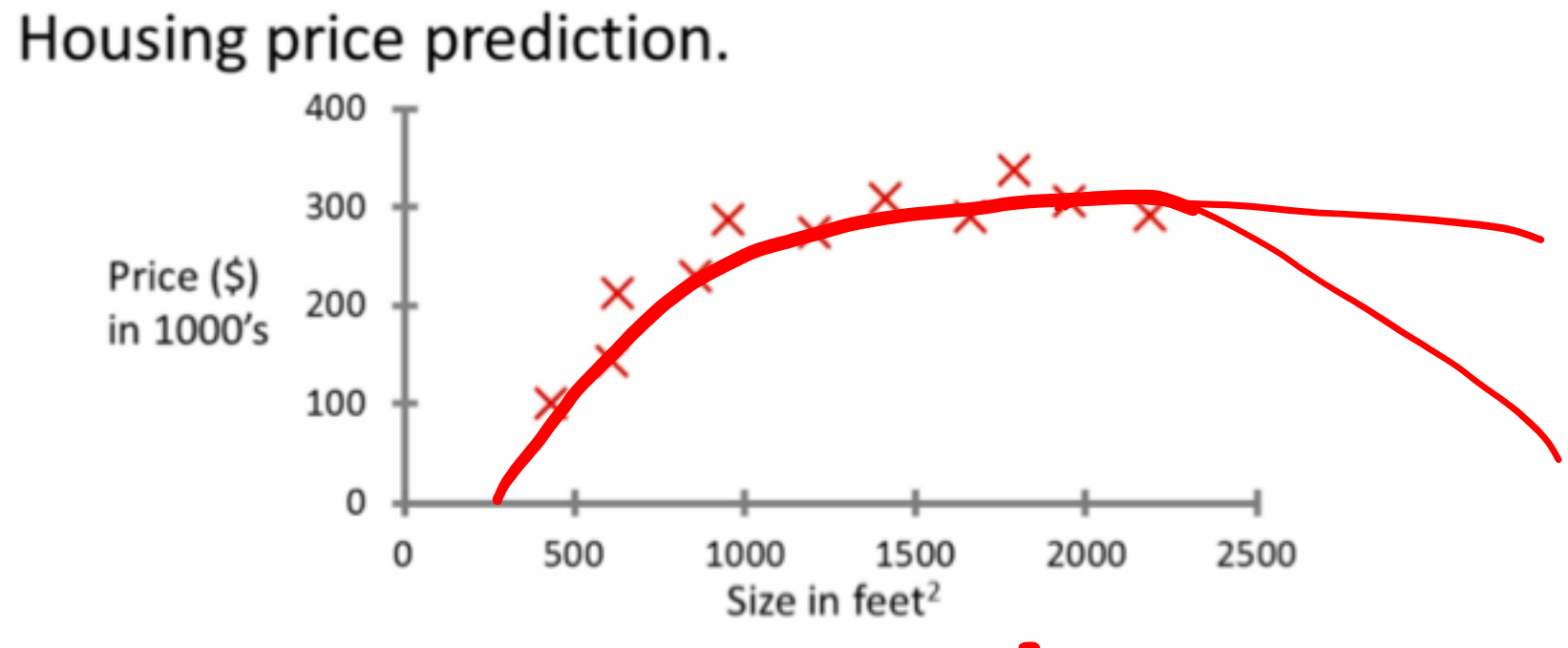
- $\omega_j = \omega_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(y_\omega\left(x^{(i)}\right) - t^{(i)}\right)\cdot x_j^{(i)}$



Practical Issues

- Feature scaling
- Mean normalization
- Learning rate

Polynomial Regression



Normal Equation (Least Square)

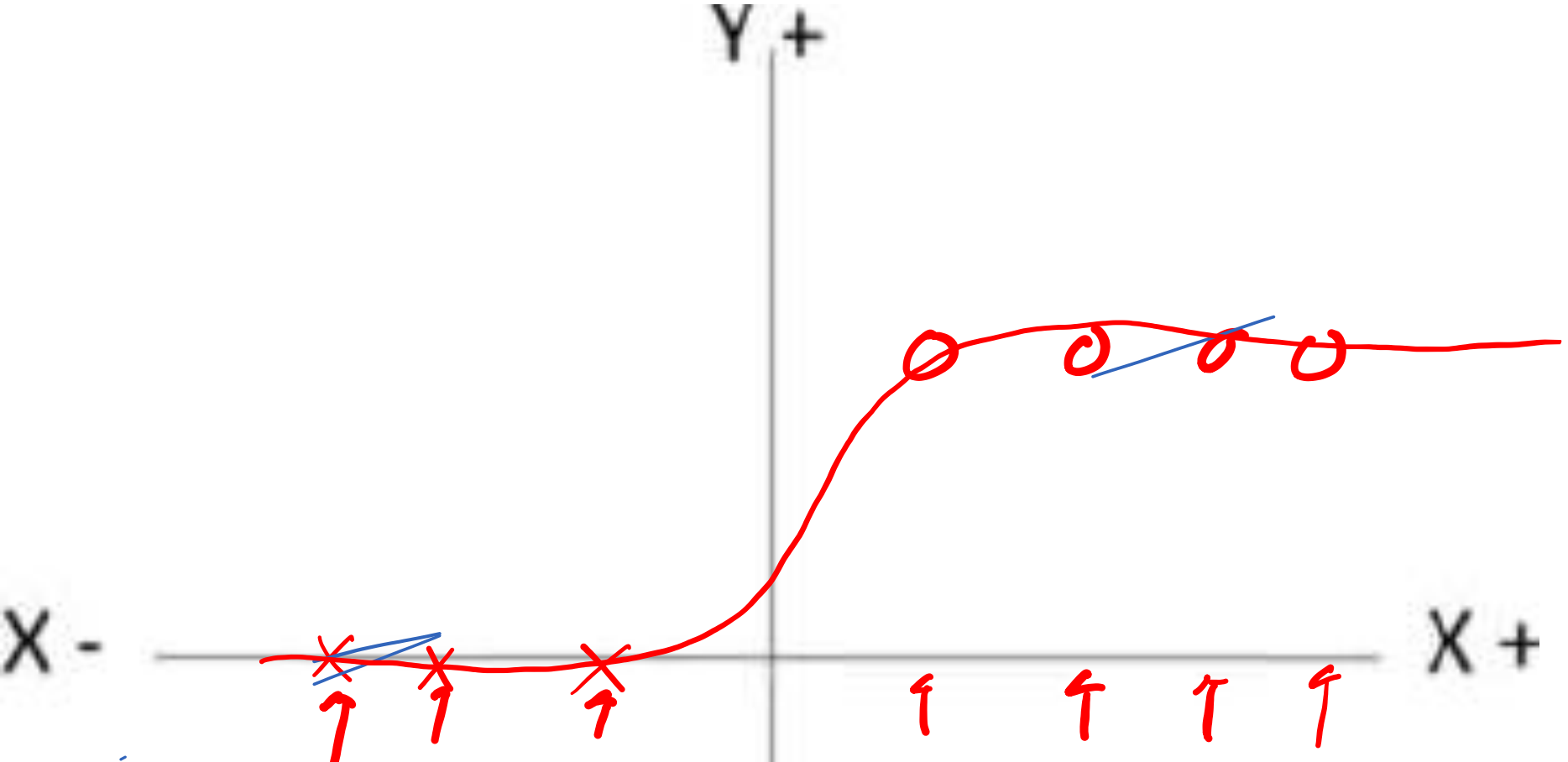
$\min C(w_0, w_1)$

$\sum_{i=0}^{m}$ $O(m)$

$w = (X^T X)^{-1} X^T t$ $O(n^3)$

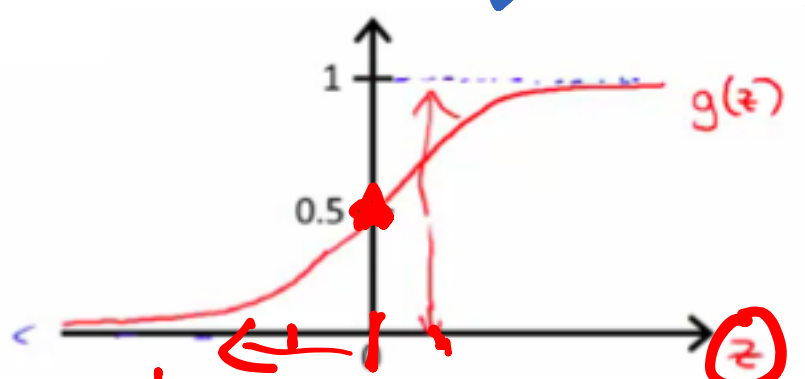
Logistic Regression

sigmoid function

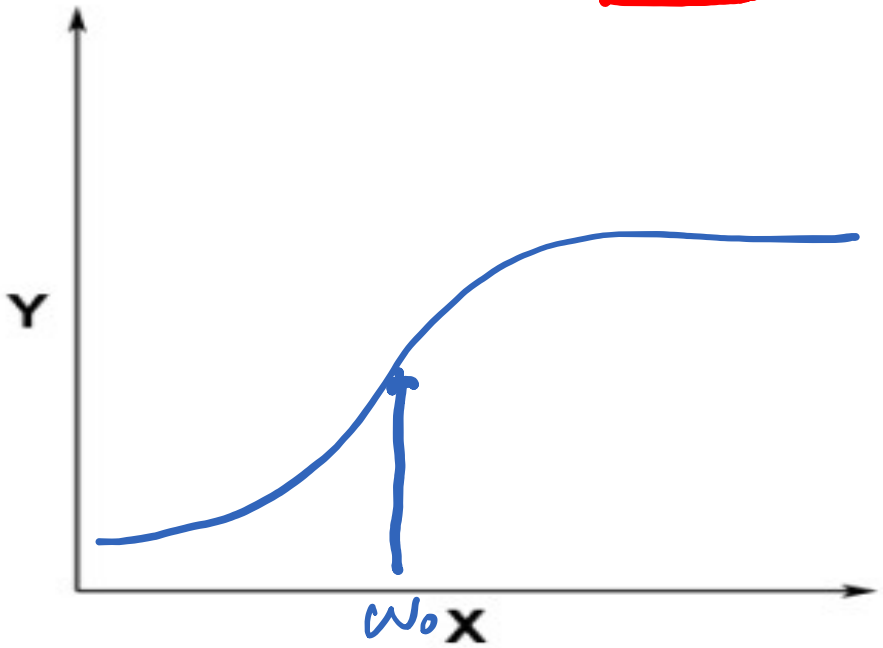


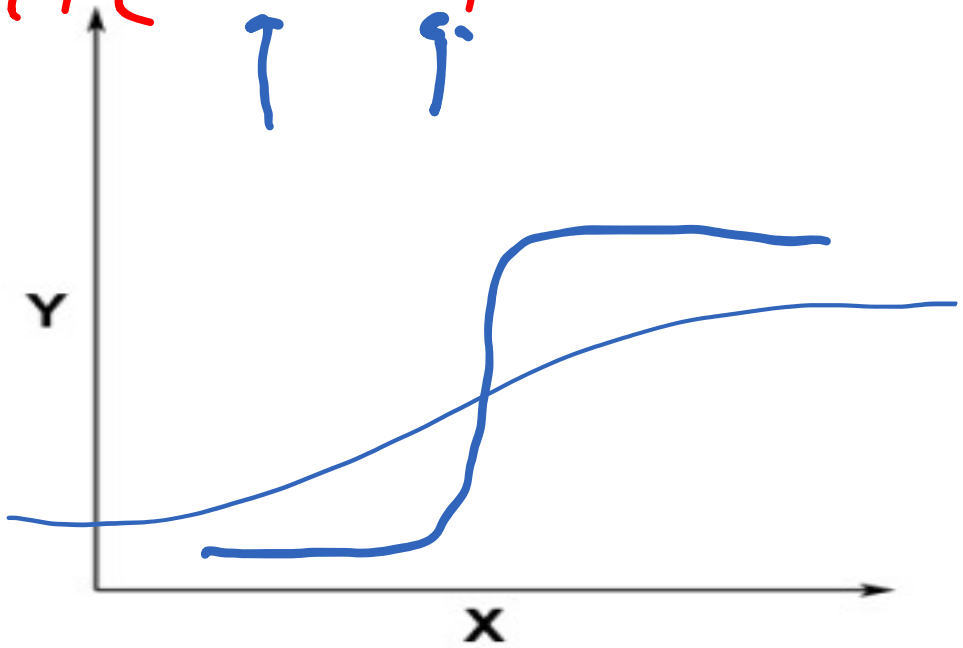
Logistic Function

$$g(z) = 1/(1 + e^{-z})$$



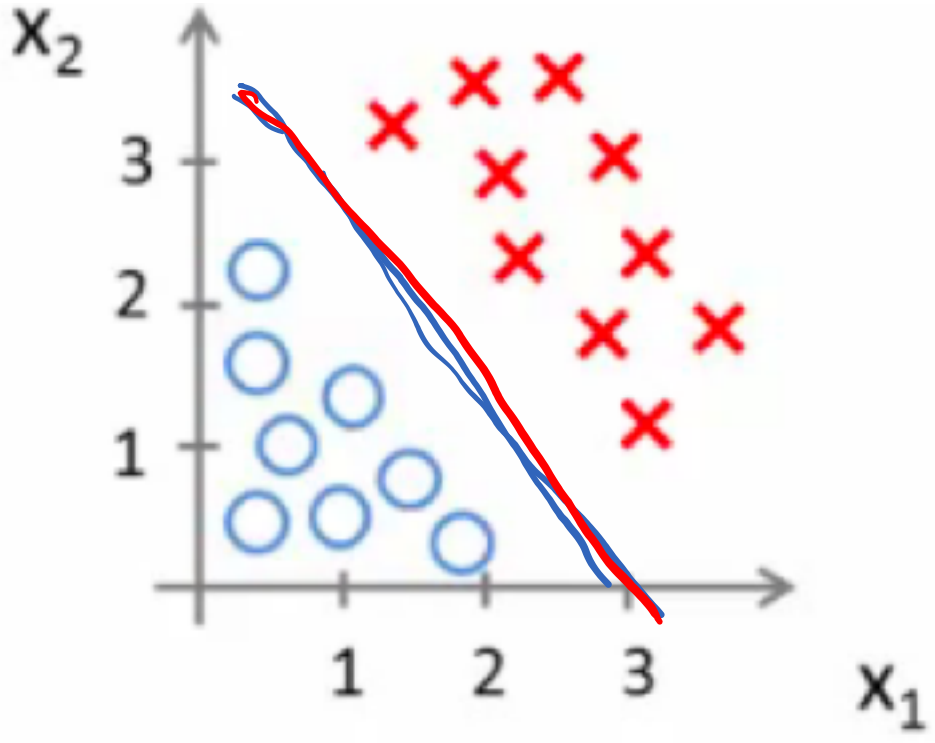
$$y_\omega(x) = \frac{1}{1 + e^{-\omega^T x}} = \frac{1}{1 + e^{-\omega_0 - \omega_1 x}}$$





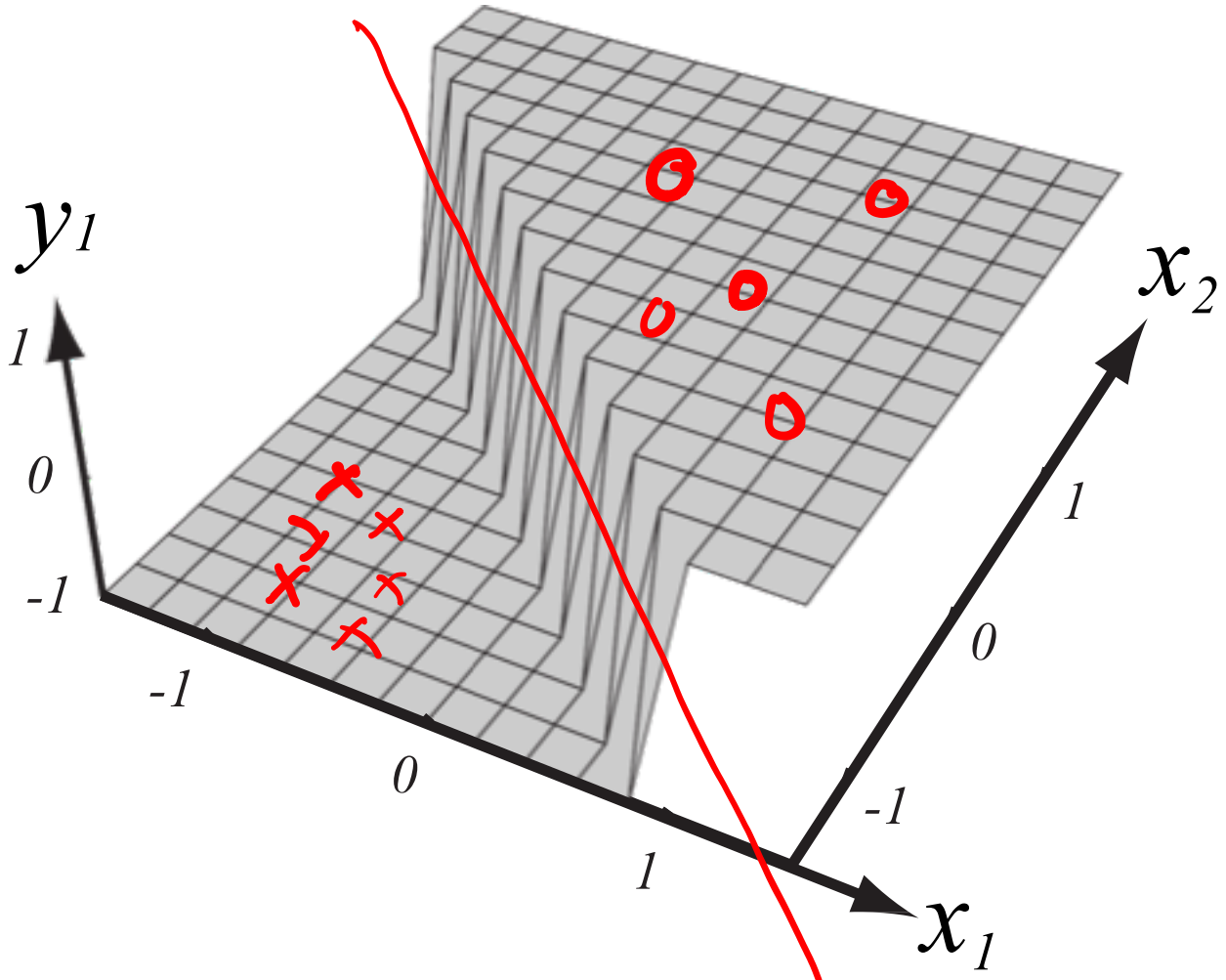
2-Dimension

$$\frac{1}{1 + e^{-w_0 + w_1 x_1 + w_2 x_2}}$$

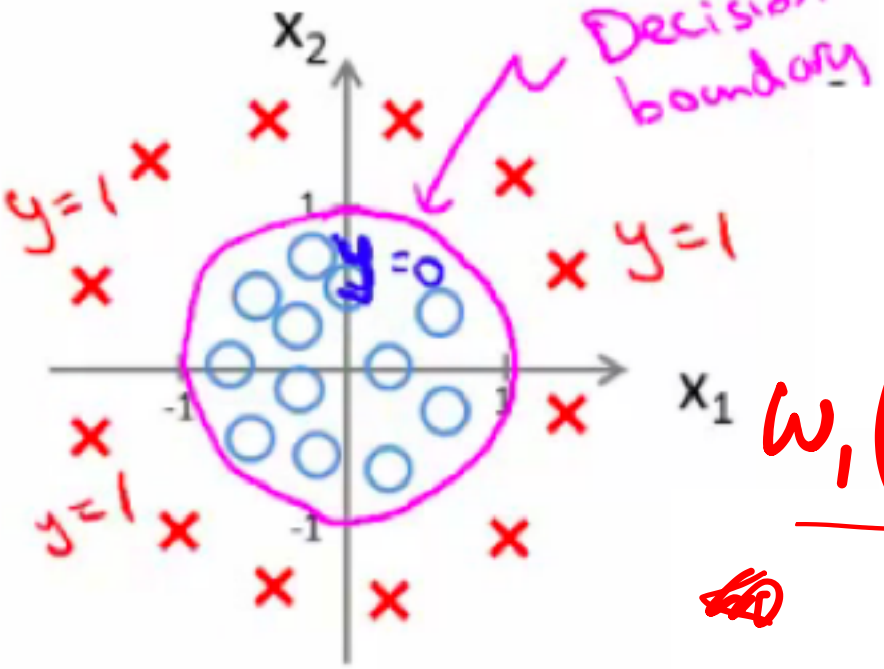


2-Dimension

$w^T x$



Non-Linear



$$y = \frac{1}{1+e^{-z}}$$

$(0, 1)$

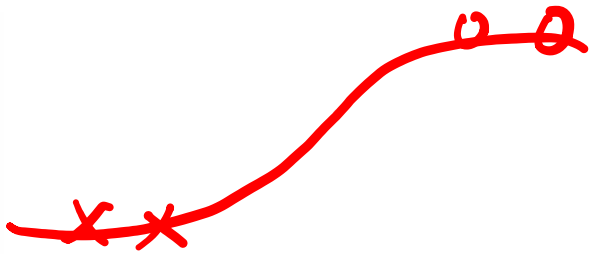
$w^T x$

$w_1(x_1^2 + x_2^2) + w_3$

Cost Function

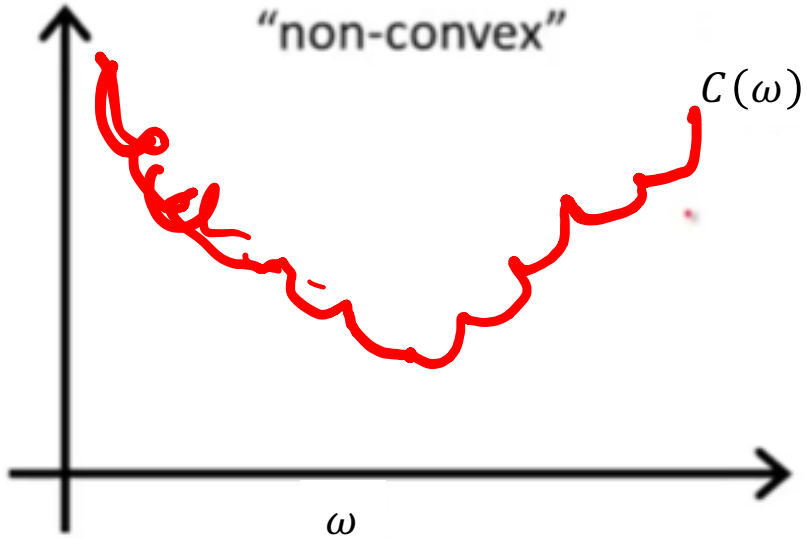
Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$



$$y_\omega(x) = \frac{1}{1 + e^{-\omega^T X}}$$

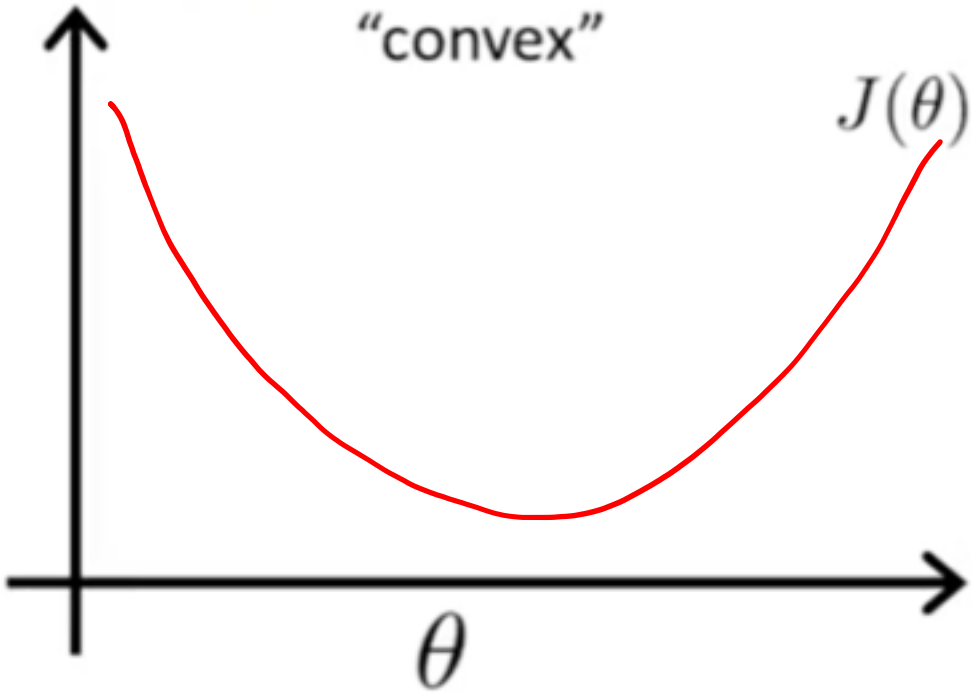
$$C(\omega) = \frac{1}{2m} \sum_{i=1}^{m} (y_\omega(x^{(i)}) - t^{(i)})^2$$

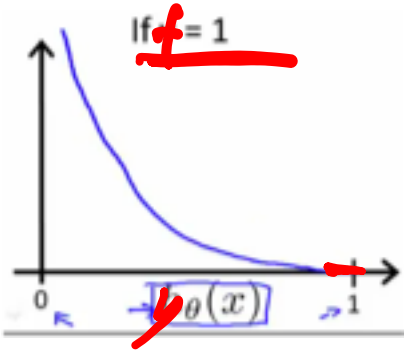


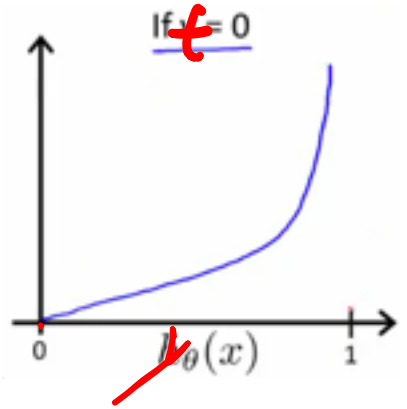
A Convex Cost Function

$$C(\omega) = \frac{1}{m}\sum_{i=1}^{m} Cost(y_\omega(x), t)$$

$$Cost(y_\omega(x), t) = \begin{cases} -\log(1 - y_\omega(x)) & \text{if } t = 0 \\ -\log(y_\omega(x)) & \text{if } t = 1 \end{cases}$$







A Simpler Form

Cross entropy

$$C(\omega) = \frac{1}{m}\sum_{i=1}^{m} Cost(y_\omega(x), t)$$

$$Cost(y_\omega(x), t) = \begin{cases} -\log(1 - y_\omega(x)) & \text{if } t = 0 \\ -\log(y_\omega(x)) & \text{if } t = 1 \end{cases}$$

$t_0 = 1, t_1 = 0$

$t_1 = 1, t_0 = 0$

$$C(\omega) = -\frac{1}{m}\sum_{i=1}^{m} t^{(i)} \log\left(y_\omega\left(x^{(i)}\right)\right) + \left(1 - t^{(i)}\right) \log\left(1 - y_\omega\left(x^{(i)}\right)\right)$$

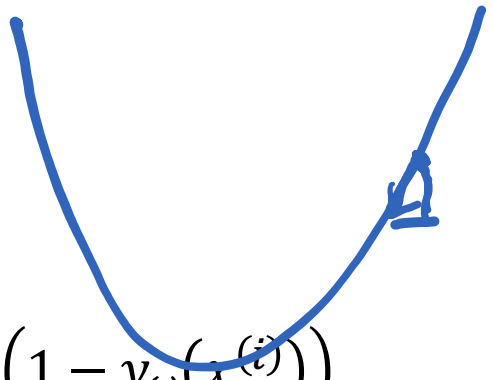
- Why this cost function?

$$C(w) = -\frac{1}{m}\sum_{i} t^{(i)} \log y^{(i)}$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{L} t_k^{(i)} \cdot \log y_k^{(i)}$$

$y_1(x) = y_\omega(x)$ $\quad y_0(x) = 1 - y_\omega(x)$

Gradient Descent



$$C(\omega) = -\frac{1}{m}\sum_{i=1}^{m} t^{(i)} \log\left(y_\omega\left(x^{(i)}\right)\right) + \left(1 - t^{(i)}\right) \log\left(1 - y_\omega\left(x^{(i)}\right)\right)$$

$$y_\omega(x) = \frac{1}{1 + e^{-\omega^T X}}$$

$$\omega_j := \omega_j - \alpha \frac{\partial C(\omega)}{\partial \omega_j}$$

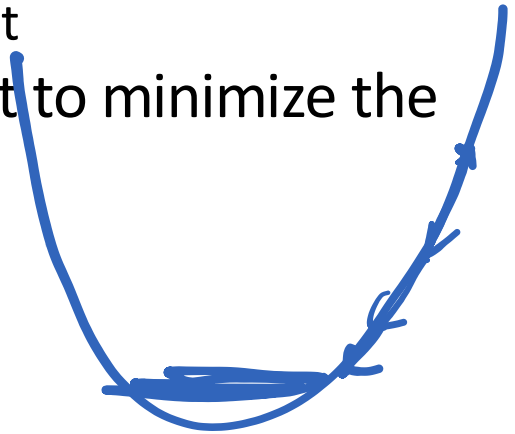
$$\omega_j = \omega_j - \alpha \frac{1}{m}\sum_{i=1}^{m} \left(y_\omega(x^{(i)}) - t^{(i)}\right) \cdot x^{(i)}$$

$\omega^T x$

$$\frac{1}{1 + e^{-\omega^T x}}$$

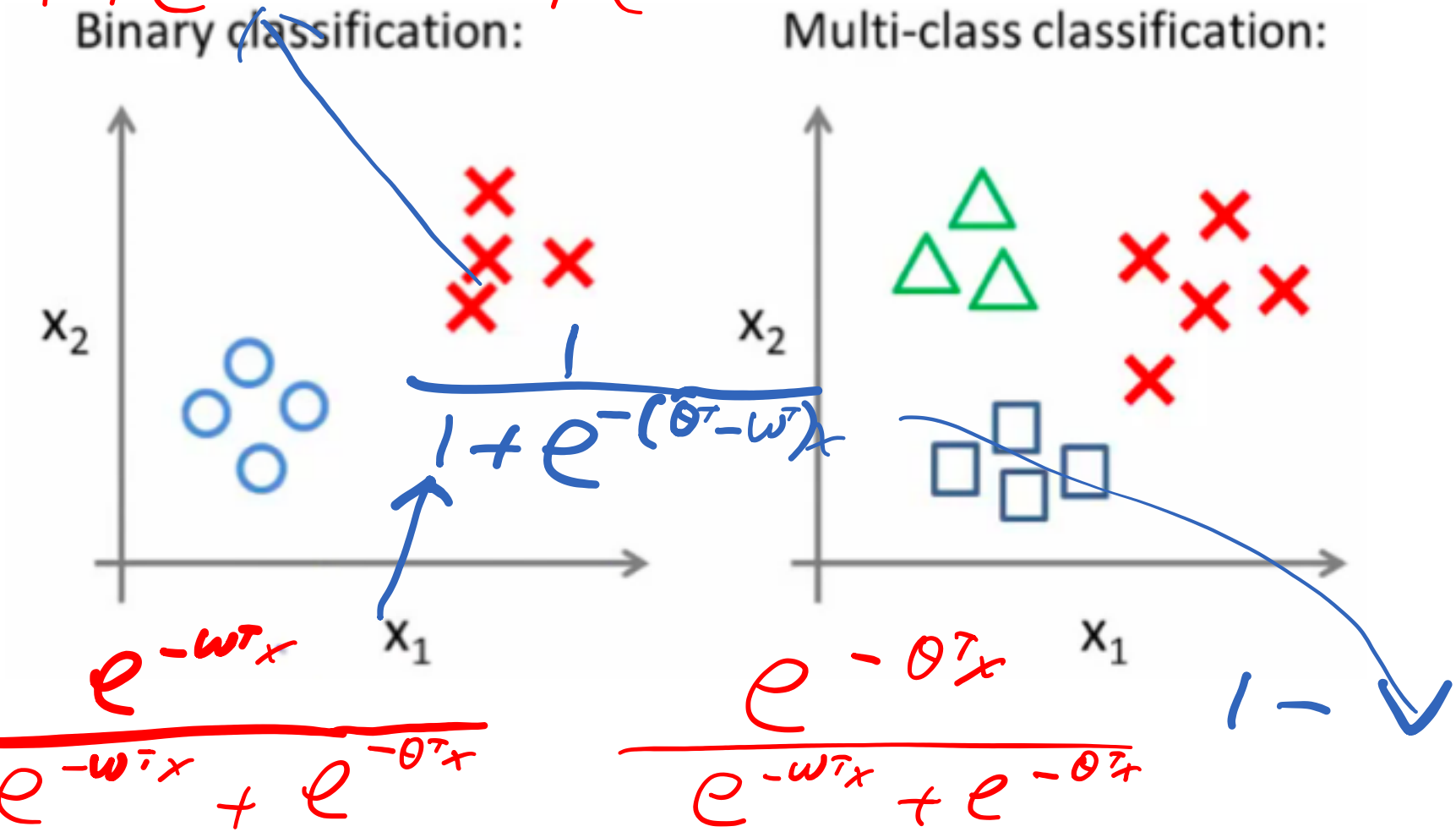
Advanced Optimization

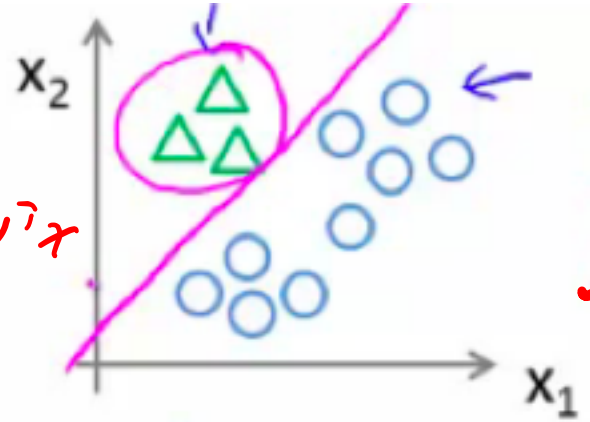
- So, we must;
  - Supply code to compute $C(\omega)$ and the derivatives
  - Then plug these values into gradient descent
- Alternatively, instead of gradient descent to minimize the cost function we could use
  - **Conjugate gradient**
  - **BFGS** (Broyden-Fletcher-Goldfarb-Shanno)
  - **L-BFGS** (Limited memory - BFGS)



- These are *very* complicated algorithms
  - **Advantages**
    - No need to manually pick alpha (learning rate)
      - Have a clever inner loop (line search algorithm) which tries a bunch of alpha values and picks a good one
    - Often faster than gradient descent
      - Do more than just pick a good learning rate
    - Can be used successfully without understanding their complexity
  - **Disadvantages**
    - Could make debugging more difficult
    - Should not be implemented themselves
    - Different libraries may use different implementations - may hit performance

Multiclass Classification



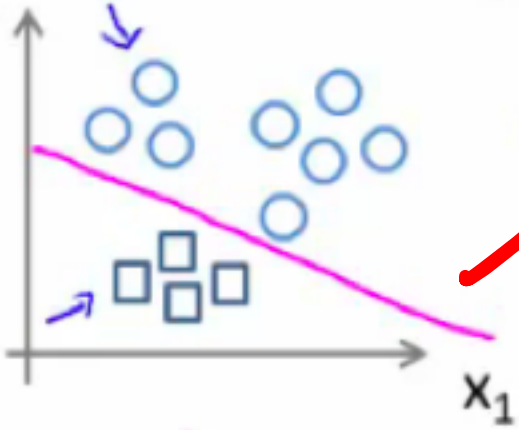


$$\frac{1}{1+e^{-w^T x}}$$

$$h_\theta^{(1)}(x)$$

$$\frac{e^{w^T x}}{e^{w^T x}+e^{\theta^T x}+e^{\gamma^T x}}$$

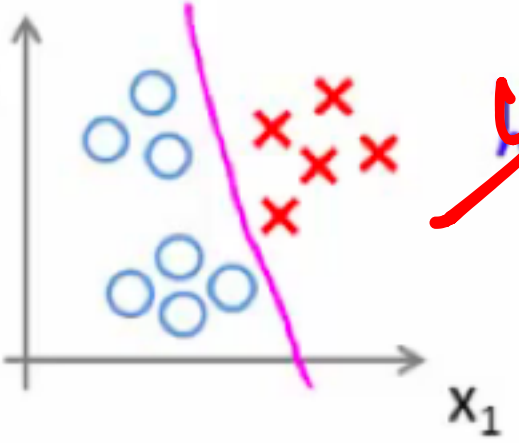
Softmax



$$\frac{1}{1+e^{-\theta^T x}}$$

$$h_\theta^{(2)}(x)$$

$$\frac{e^{\theta^T x}}{e^{w^T x}+e^{\theta^T x}+e^{\gamma^T x}}$$



$$\frac{1}{1+e^{-\gamma^T x}}$$

$$h_\gamma^{(3)}(x)$$

$$\frac{e^{\gamma^T x}}{e^{w^T x}+e^{\theta^T x}+e^{\gamma^T x}}$$


Q&A