ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Machine Learning

Lecture 2: Linear regression

Petia Georgieva
(petia@ua.pt)

Linear regression - outline

1. **Univariate linear regression**
   - Cost (loss) function - Mean Square Error (MSE)
   - Cost function convergence
   - Gradient descent algorithm

2. **Multivariate linear regression**
   -Overfitting problem

3. **Regularized linear regression**

CLASSIFICATION vs REGRESSION

Classification- the model output is a label (e.g. integer numbers 0, 1, -1, etc.)

Regression - the model output is a real number

Examples of regression problems:

- Weather forecast
- Predicting wind velocity from temperature, humidity, air pressure
- Time series prediction of stock market indices
- Predicting sales amounts of new product based on advertising expenditure
- Equalization in communication channels (IT-UA)

Supervised Learning – univariate regression

Problem: Learning to predict the housing prices (output) as a function of the living area (input, data feature)

Living area ($feet^2$)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Mean Square Error (MSE)

Linear Model (hypothesis) =>

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Cost (loss) function =>
(Mean Square Error)

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

m – number of training examples

Goal **=>**

$$\min_\theta J(\theta)$$

Gradient descent algorithm =>
iterative algorithm; at each iteration all parameters (theta) are updated simultaneously

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

alpha – learning rate > 0

Linear Regression (computing the gradient)

Cost function =>

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

Cost function gradients =>
vector with parcial derivatives of J with respect to each parameter for one example (m=1)

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y \right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(h_\theta(x) - y \right) \\
&= \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h_\theta(x) - y \right) x_j
\end{aligned}$$

Cost function gradients =>
for m examples

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Linear Regression – iterative gradient descent algorithm (summary)

Linear Model (hypothesis) => $$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

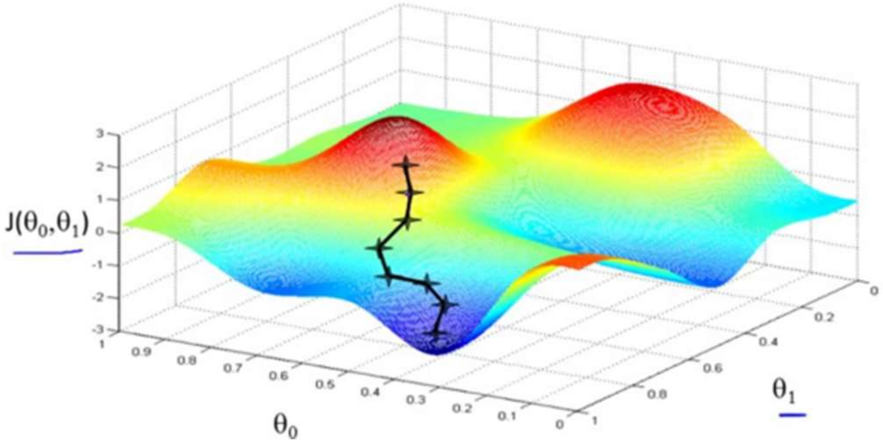
Repeat until J converge {

Compute cost function => $$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Compute cost function gradients => $$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Update parameters => $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}



Batch/mini batch/stochastic gradient descent for parameter update

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

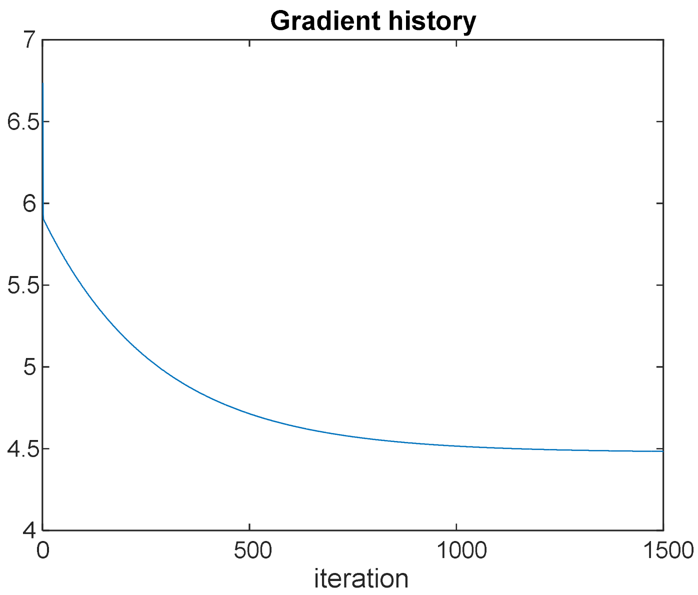
Batch learning (classical approach)
update parameters after all training examples have been processed, repeat several iterations until convergence

Mini batch learning (if big training data):
devide training data into small batches update parameters after each mini batch has been processed, repeat until convergence

Stochastic (incremental) learning (if small training data)
update parameters after every single training example has been processed.

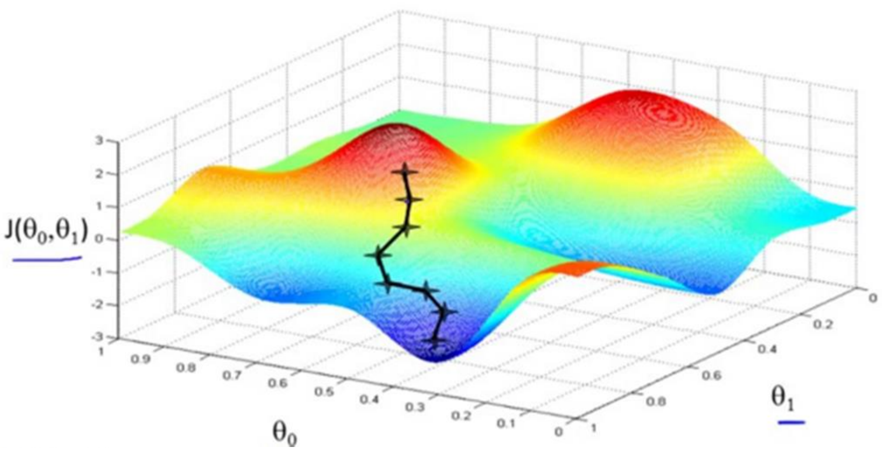
Cost function convergence

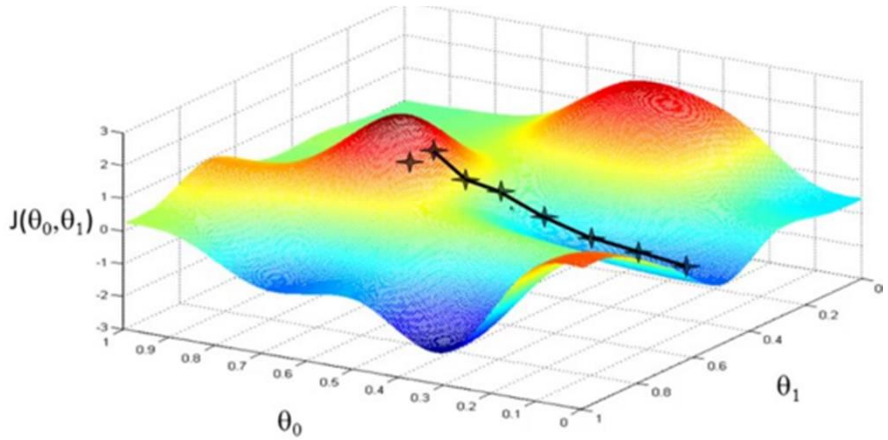
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$



Linear Regression (LR):

starting from different initial values of the parameters the cost function J should always converge (**<u>maybe to a local minimum !!!</u>**) if LR works properly.



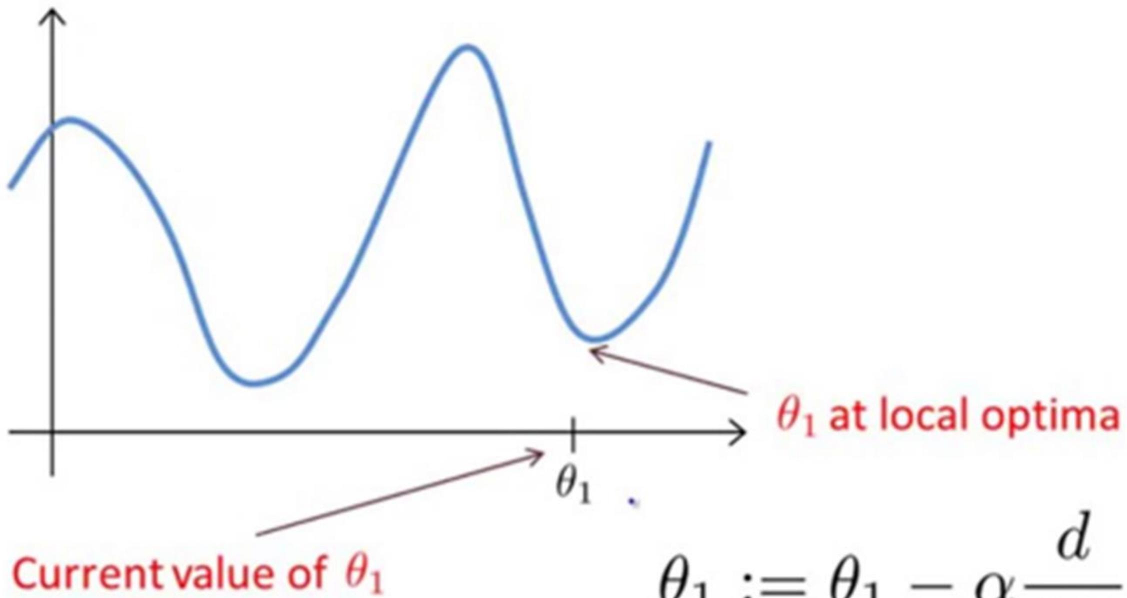


Cost function – local minimum

Suppose θ_1 is at a local optima as shown in the figure.

What will one step of Gradient Descent do ?

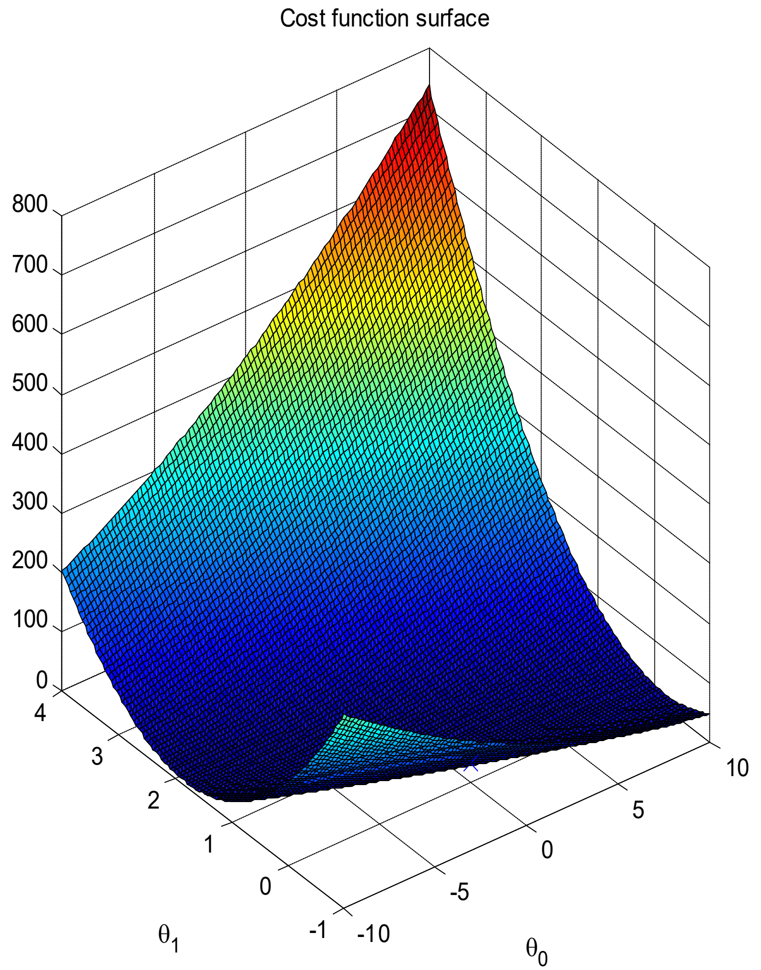
1) Leave θ_1 unchanged
2) Change θ_1 in a random direction
3) Decrease θ_1
4) Move θ_1 in direction to the global minimum of J

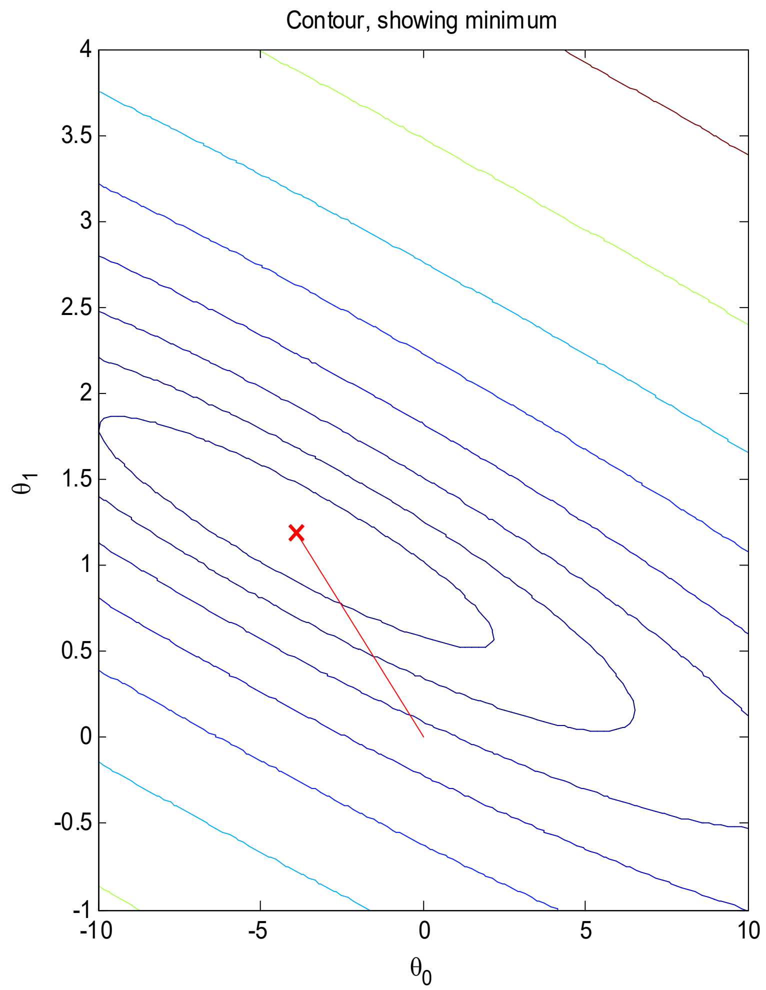


$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Cost function

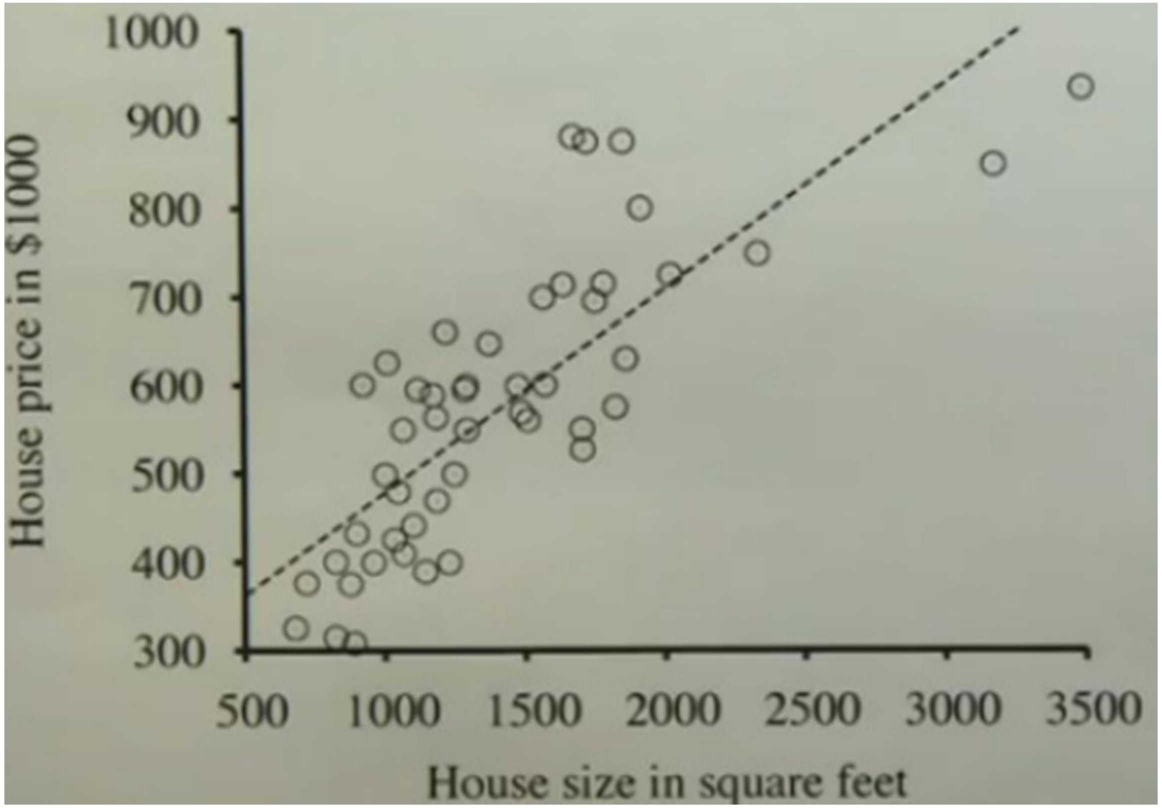
Cost function for range of values of model parameters (thetas)





Linear regression model

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$



Given the house area, what is the most likely house price?
If univariate linear regression model is not sufficiently good model, add more data, ex. # bedrooms.

Supervised Learning – multivariate regression

Problem: Learning to predict the housing price as a function of living area & number of bedrooms.

Living area (feet^2)	#bedrooms	Price (1000\$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Multivariate regression

n –features; m – training examples

X – feature mx(n+1) matrix **y – output (mx1) vector** **parameter (n+1)x1 vector**

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Error (mx1) vector

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}. \end{aligned}$$

Multivariate gradient

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

gradient column (n+1)x1 vector

$$\begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} \sum_{i=1}^{m}\left((h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}\right) \\ \sum_{i=1}^{m}\left((h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}\right) \\ \sum_{i=1}^{m}\left((h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}\right) \\ \vdots \\ \sum_{i=1}^{m}\left((h_\theta(x^{(i)}) - y^{(i)})x_n^{(i)}\right) \end{bmatrix}$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left((h_\theta(x^{(i)}) - y^{(i)})x^{(i)}\right)$$

Implementation of multivariate regression

If Z is a column vector => $z^T z = \sum_i z_i^2$

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

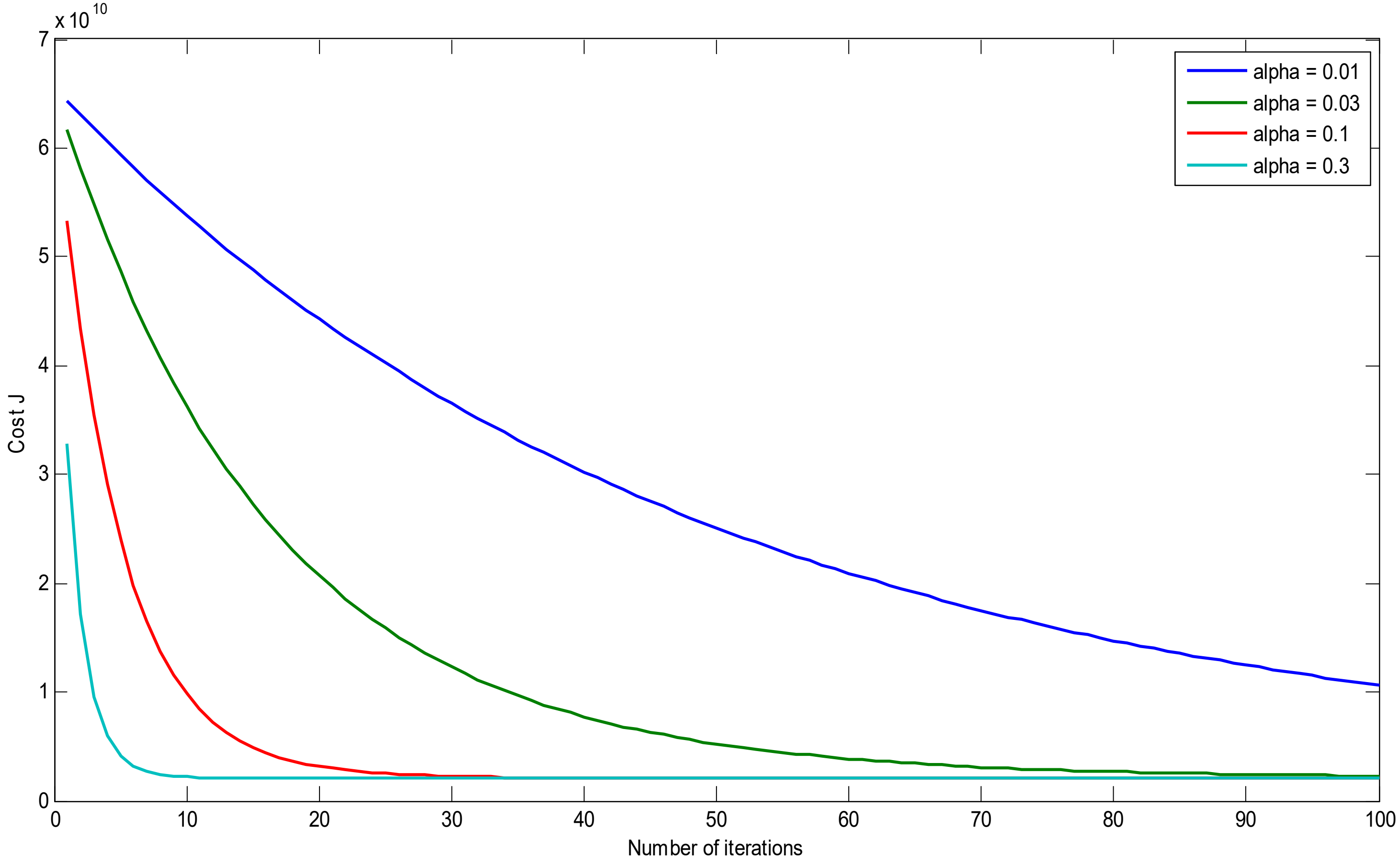
$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

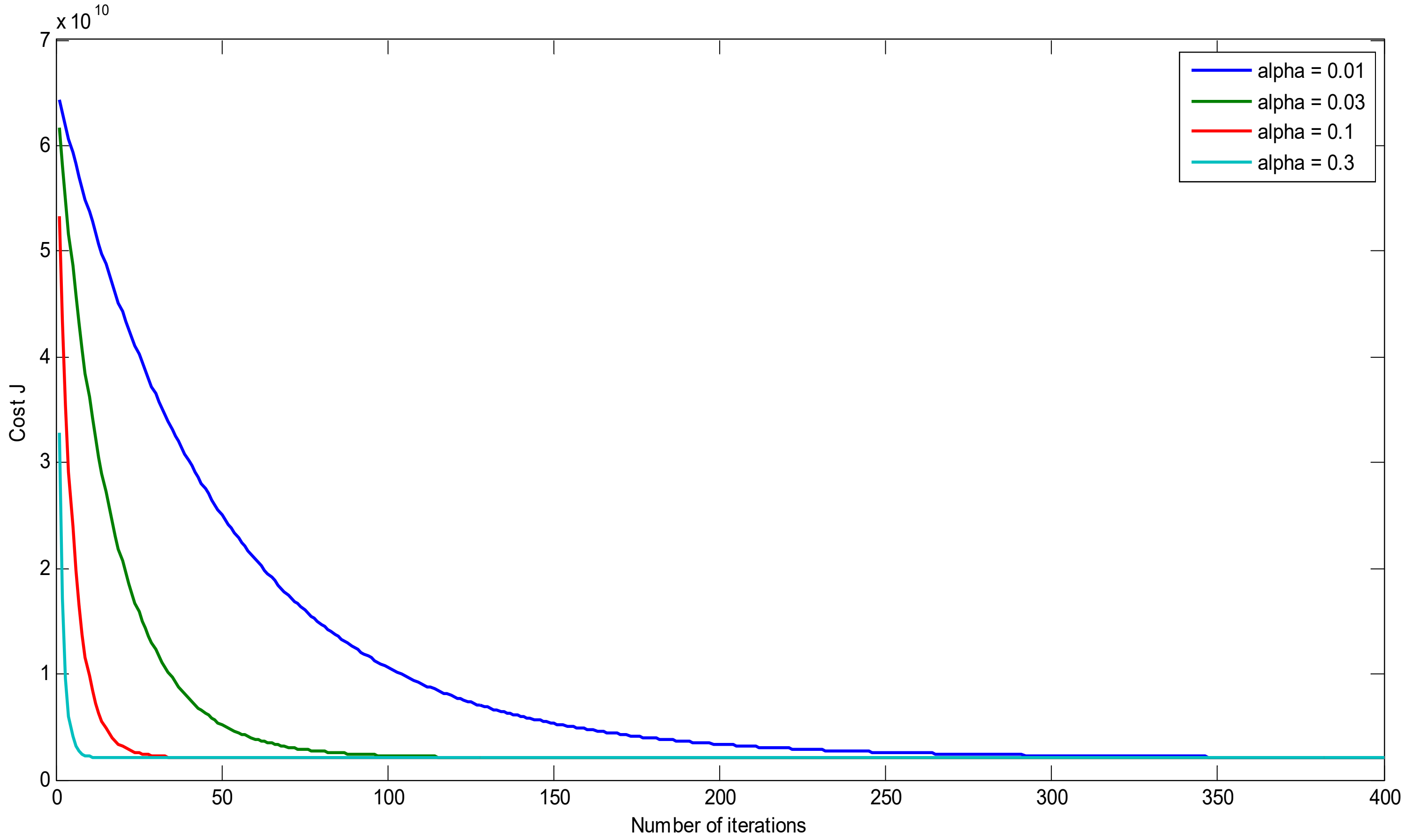
$$\theta_2 := \theta_2 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

Cost function convergence changing the learning rate (alpha) -100 iter.

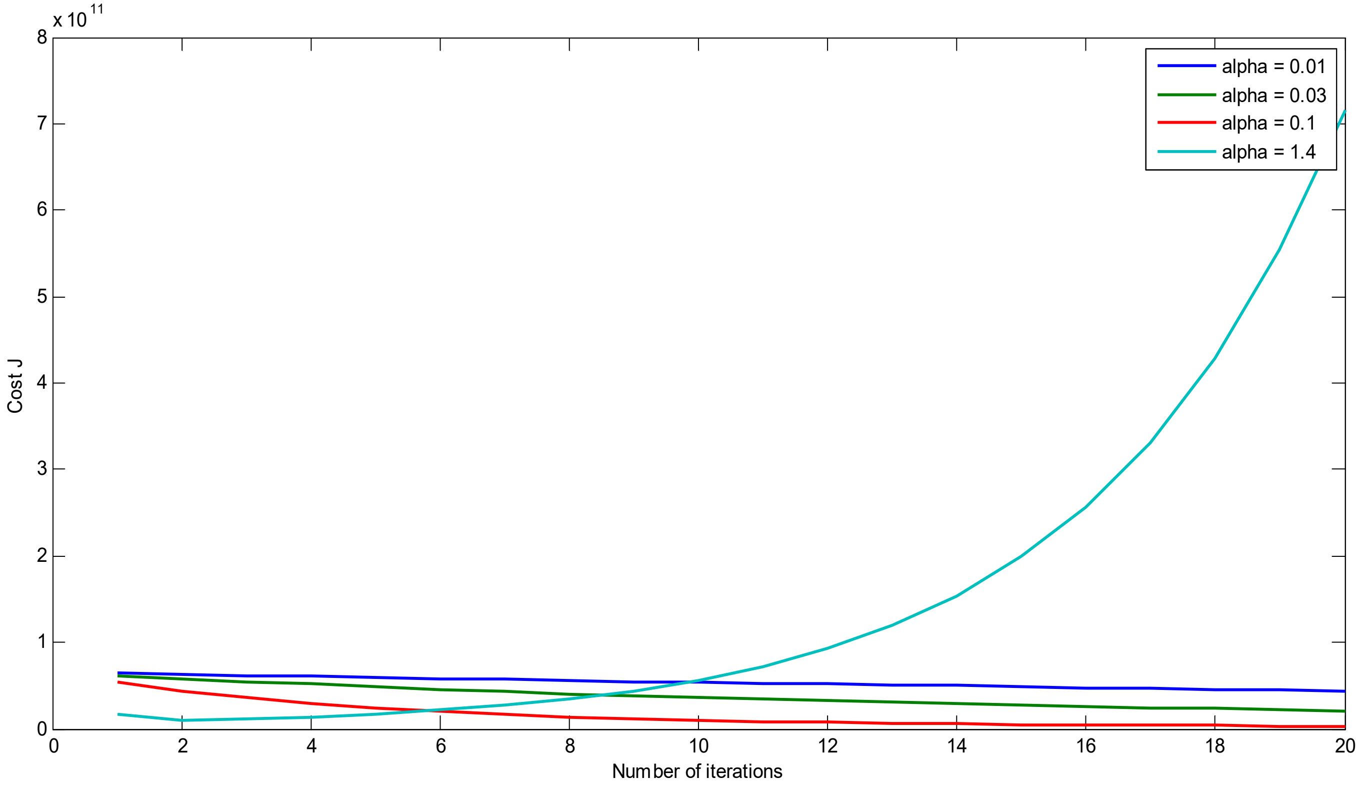


If alpha too small : slow convergence of the cost function J (the Gradient Descent optimization can be slow)

Cost function convergence changing the learning rate (alpha) -400 iter.



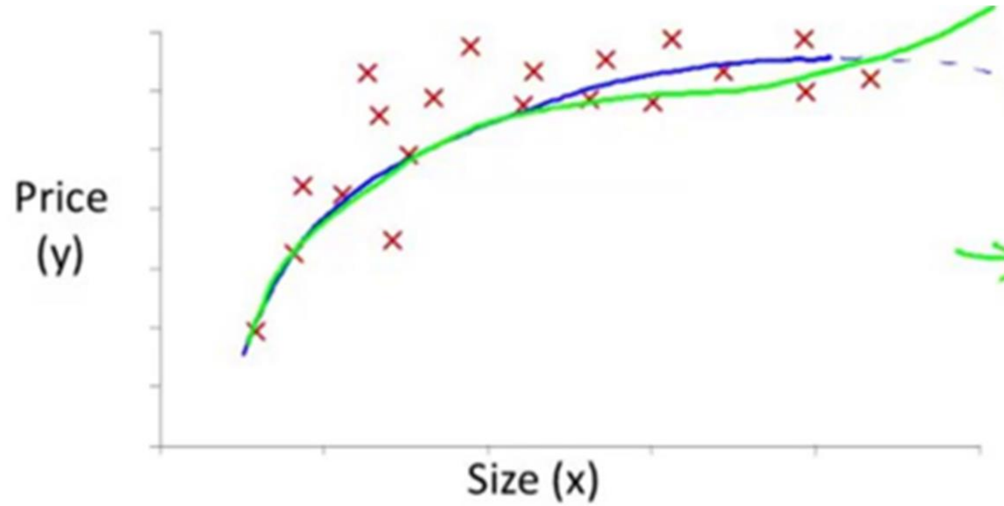
Cost function convergence changing the learning rate (alpha)



If alpha too large: the cost function J may no converge **(**decrease at each iteration). It may diverge **!**

Polynomial Regression

If univariate linear regression model is not a good model, try polynomial model.
Univariate (x1=size) housing price problem transformed into multivariate (still linear !!!) regression model x=[x1=size, x2=size^2, x3=size^3]



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

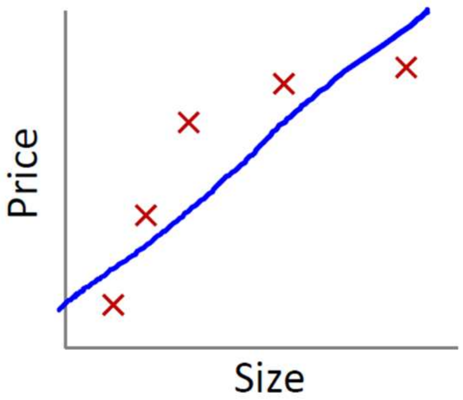
$$\boxed{\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$$

$$
\begin{aligned}
h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\
&= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3
\end{aligned}
$$

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

Overfitting problem

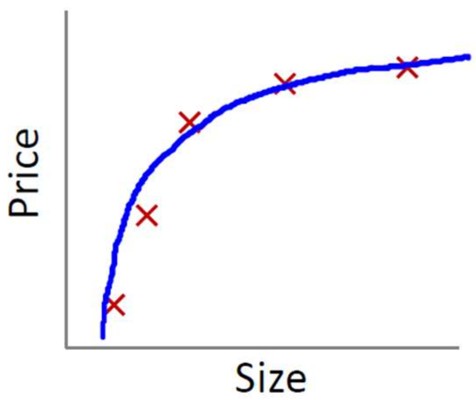
Overfitting: If we have too many features (high order polynomial model), the learned hypothesis may fit the training set very well but fail to generalize to new examples (predict prices on new examples).



underfit- high bias

(1st order polin. model)

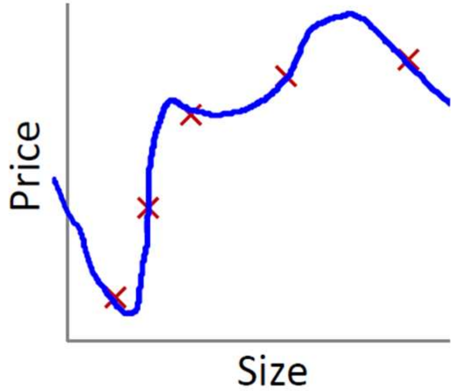
$$h_\theta(x) = \theta_0 + \theta_1 x$$



just right

(3rd order polinom. model)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$



overfit- high variance

(higher ord. polinom. Model)

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + ... + \theta_{16} x^n$$

Overfitting problem

Overfitting: If we have too many different features (x1,...x100) the learned model may fit the training data very well but fail to generalize to new examples.

$$
\begin{aligned}
x_1 &= \text{size of house} \\
x_2 &= \text{no. of bedrooms} \\
x_3 &= \text{no. of floors} \\
x_4 &= \text{age of house} \\
x_5 &= \text{average income in neighborhood} \\
x_6 &= \text{kitchen size} \\
&\vdots \\
x_{100}
\end{aligned}
$$

Overfitting problem

Options:

1. Reduce number of features.
— Manually select which features to keep.
— Algorithm to select the best model complexity (later in course).

2. Regularization.
— Keep all the features, but reduce the magnitude of parameters theta.
— Works well when we have a lot of features, each of which contributes a bit to predict the output y.

Regularized Linear Regression (cost function)

Unregularized cost function =>

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

Regularized cost function (start from j=1 !!!) =>

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

$$\min_\theta J(\theta)$$

Regularized Linear Regression (cost function gradient)

Unregularized cost function gradients =>

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Regularized cost function gradients =>

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1$$

Regularized Linear Regression

What if lambda is set to an extremely large value ?

- Algorithm fails to eliminate overfitting.
- Algorithm results in under-fitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.