Departamento de Eletrónica, Telecomunicações e Informática

LINEAR REGRESSION

Author: Petia Georgieva
Edited by: Susana Brás (Susana.bras@ua.pt)

universidade de aveiro

Supervised Learning

Examples (regression or classification) ?

- 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

LINEAR REGRESSION - outline

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

2. **Multivariate linear regression**

   -Overfitting problem

3. **Regularization =>** way to deal with overfitting

Supervised Learning - CLASSIFICATION vs REGRESSION

Classification- the label is an integer number.
(e.g. 0, 1 for binary classification)

Regression - the label 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

Standard Notations in this course

x – input vector of features, attributes

y – output vector of labels, ground truth, target

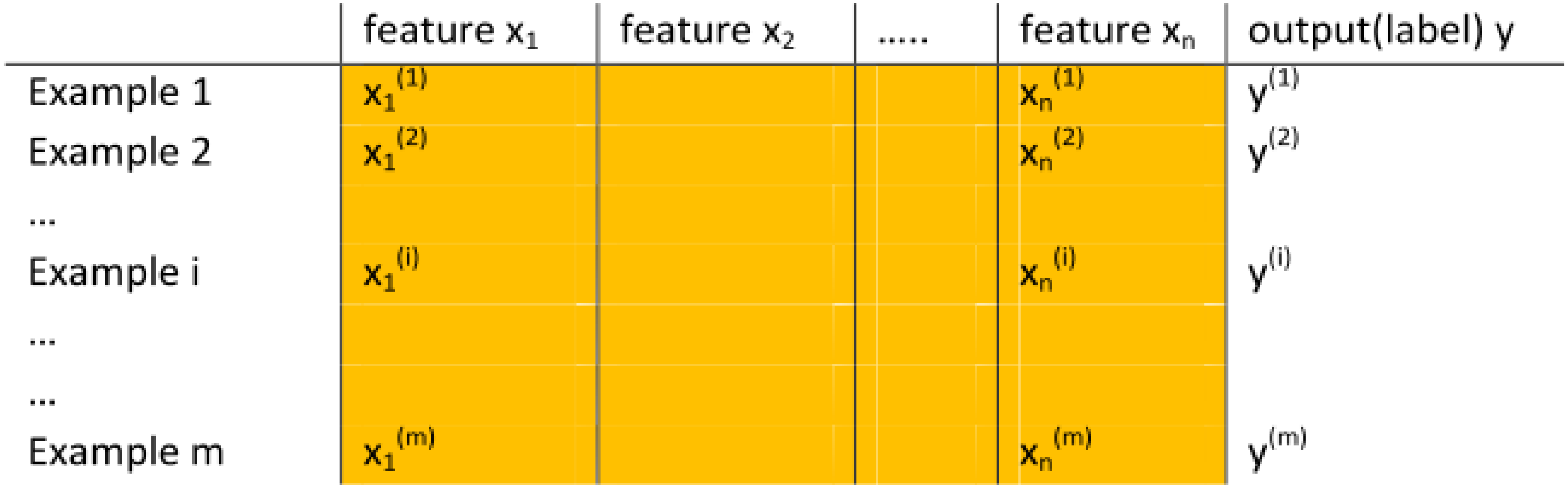
m - number of training examples

n – number of features

$h_\theta(x)$ - model (hypothesis)

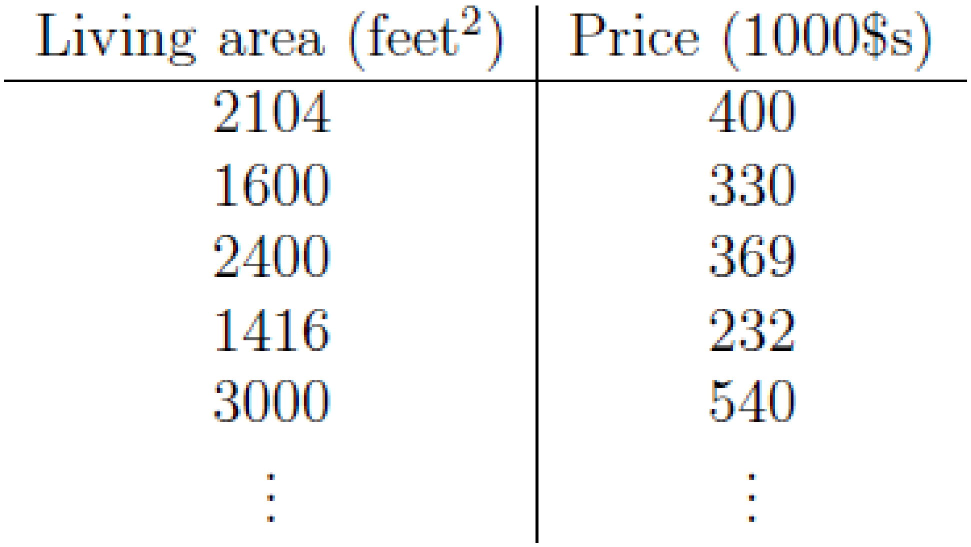
θ - vector of model parameters

Training set: data matrix X (m rows, n columns)

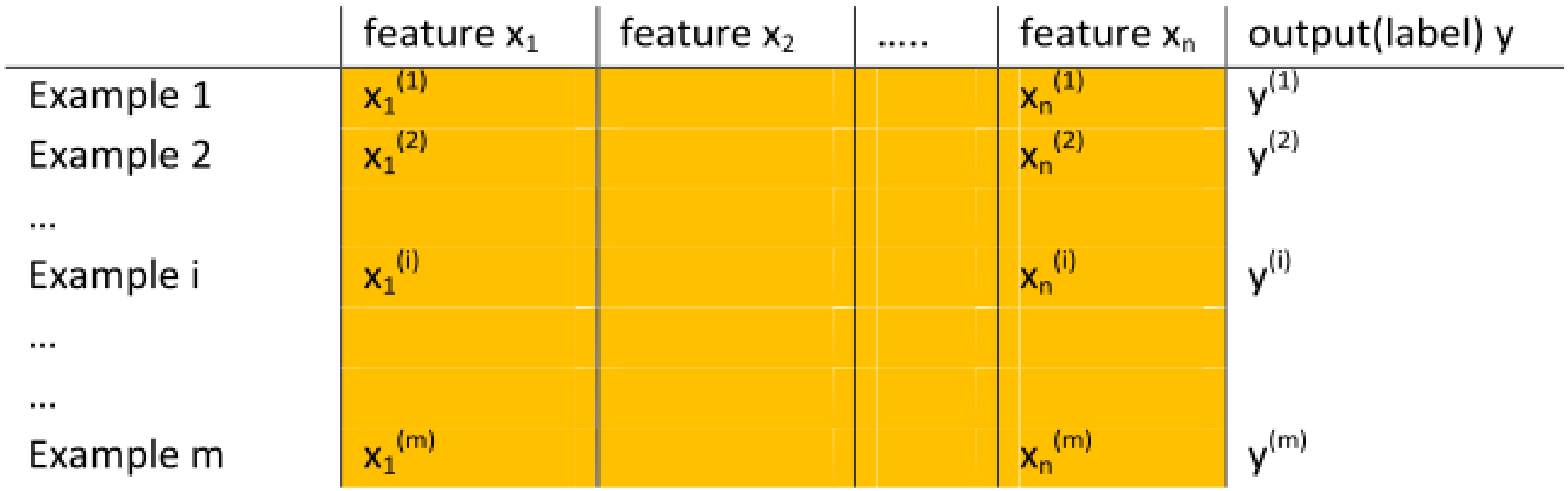


	feature x_1	feature x_2	.....	feature x_n	output(label) y
Example 1	$x_1^{(1)}$			$x_n^{(1)}$	$y^{(1)}$
Example 2	$x_1^{(2)}$			$x_n^{(2)}$	$y^{(2)}$
...					
Example i	$x_1^{(i)}$			$x_n^{(i)}$	$y^{(i)}$
...					
...					
Example m	$x_1^{(m)}$			$x_n^{(m)}$	$y^{(m)}$

Example



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



	feature x_1	feature x_2	.....	feature x_n	output(label) y
Example 1	$x_1^{(1)}$			$x_n^{(1)}$	$y^{(1)}$
Example 2	$x_1^{(2)}$			$x_n^{(2)}$	$y^{(2)}$
...					
Example i	$x_1^{(i)}$			$x_n^{(i)}$	$y^{(i)}$
...					
...					
Example m	$x_1^{(m)}$			$x_n^{(m)}$	$y^{(m)}$

Supervised Learning – regression

Regression is a type of statistical technique used in data science to analyze the relationship between variables.

Regression is an important tool for data science because:

1. Lets you identify relationships between variables

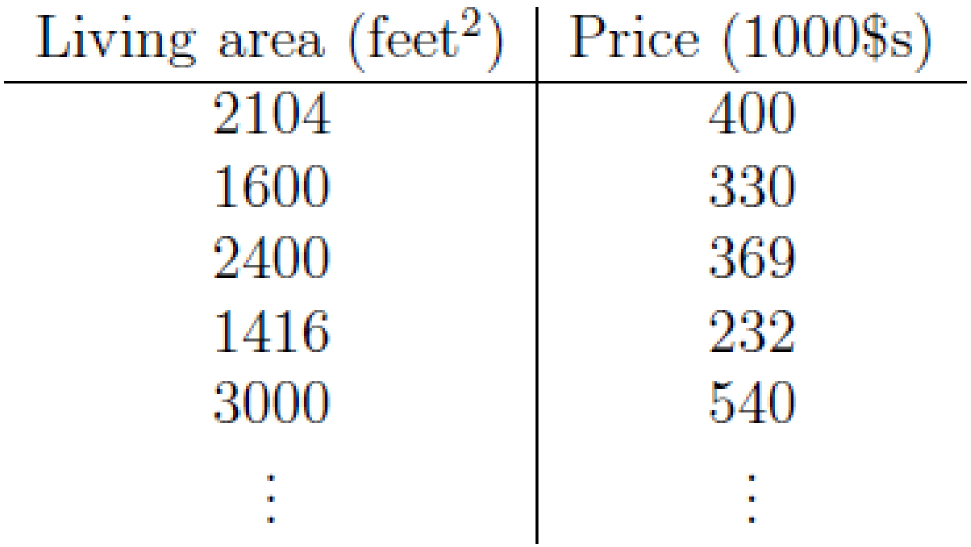
2. Make predictions based on those relationships

3. Assess how well a given model fits the data

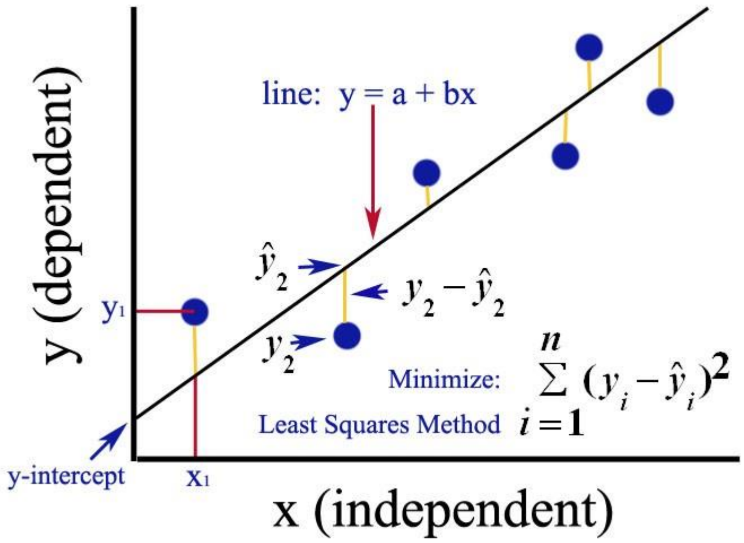
Supervised Learning – univariate regression

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

observation – prediction = Error (“residual”)



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

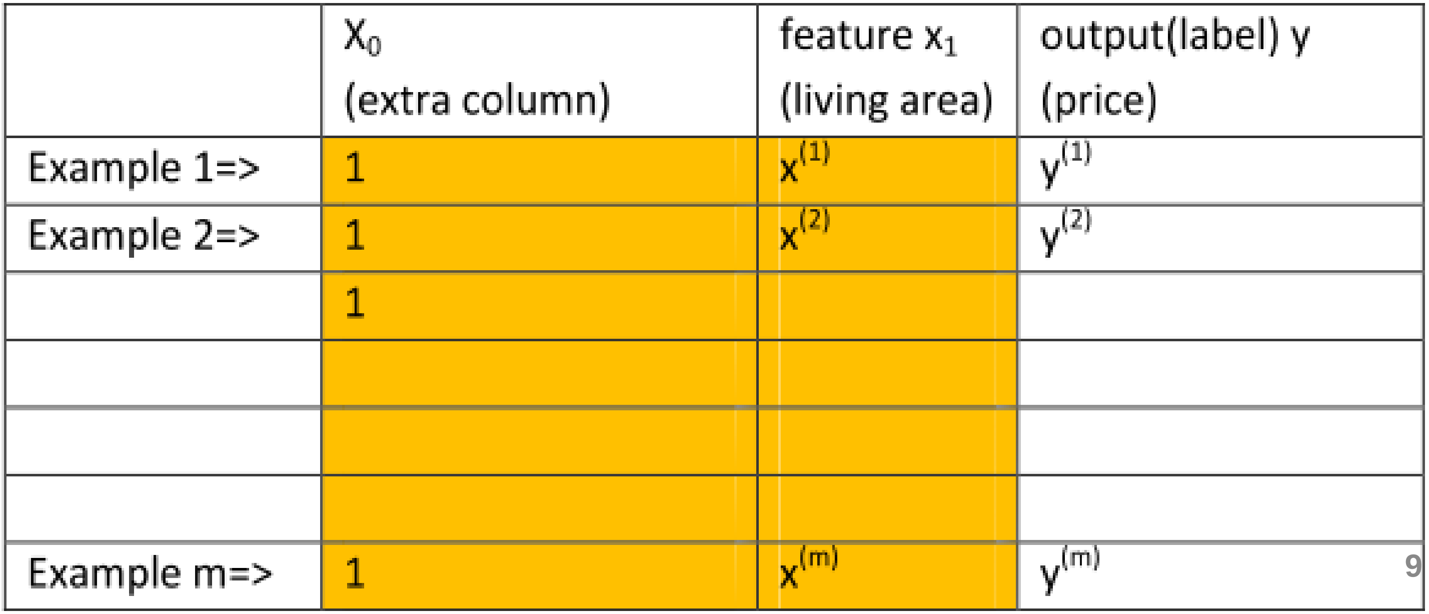


Supervised Learning – univariate regression

$$h_\theta(x) = \theta_0 + \theta_1 x_1 = \begin{bmatrix} 1 & x_1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \vec{x}^T \vec{\theta}$$

=> in Python => np.dot(X, Theta)

$$\vec{x} = \begin{bmatrix} x_0 = 1 \\ x_1 \end{bmatrix}$$



	X_0 (extra column)	feature x_1 (living area)	output(label) y (price)
Example 1=>	1	$x^{(1)}$	$y^{(1)}$
Example 2=>	1	$x^{(2)}$	$y^{(2)}$
	1		
Example m=>	1	$x^{(m)}$	$y^{(m)}$

Gradient descent algorithm

Linear Model (hypothesis) =>

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

Cost (loss) function **=>**
(Mean Squared 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 **=>**

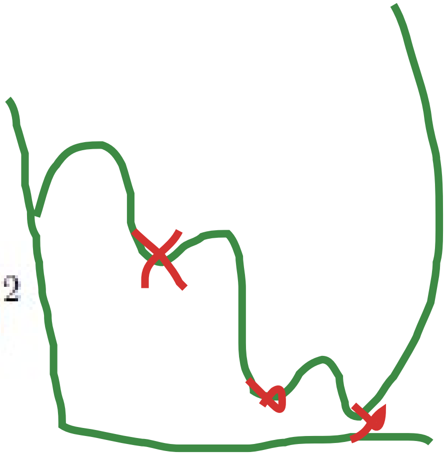
$$\min_\theta J(\theta)$$

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

Cost function gradients



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)

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

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

Cost function gradients **=>** for *m* examples

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

Linear Regression – iterative gradient descent algorithm (summary)

Inicialize model parameters *(e.g. θ =0)*

{ **Repeat until J converge (# of iterations)**

Compute model predictions for *m* examples (vector) =>

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

Compute cost function (scalar) =>

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

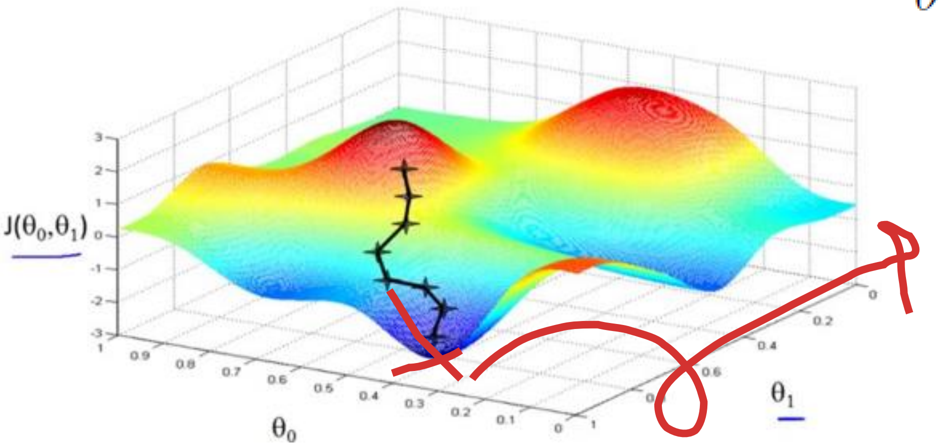
Compute cost function gradients (vector) =>

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

Update model parameters /weights (vector) =>

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

}



local minimum

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

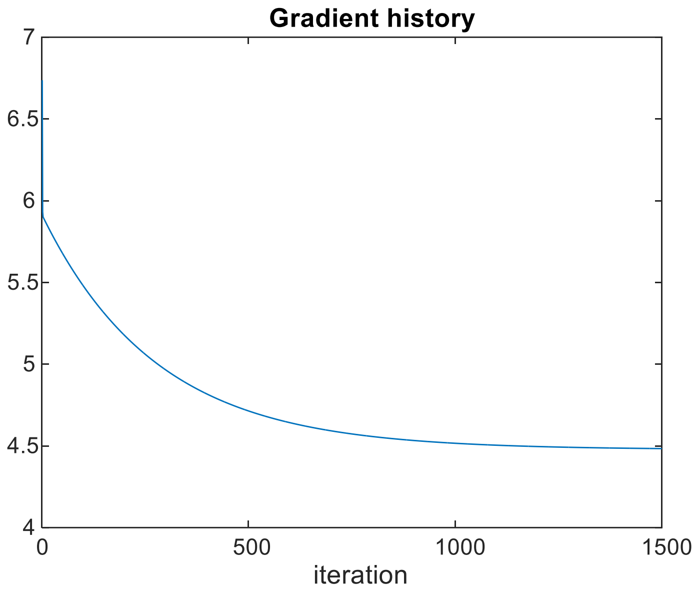
Mini batch learning (if big training data):
devide training data into small batches, update parameters after each mini batch has been processed, repeat until convergence

Stochastic (incremental) learning (large-scale ML problems**):**
update parameters after every single training example has been processed.

Stochastic Gradient Descent (SGD): the true gradient is approximated by a gradient at a single example. Adaptive learning rate.

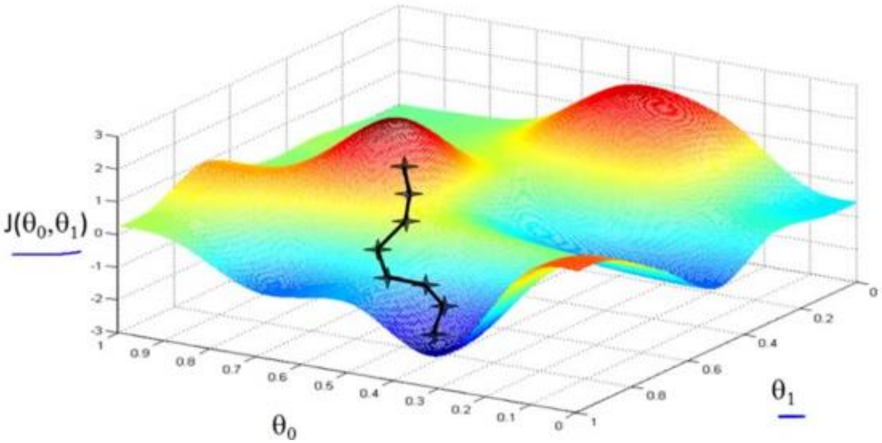
Cost function convergence

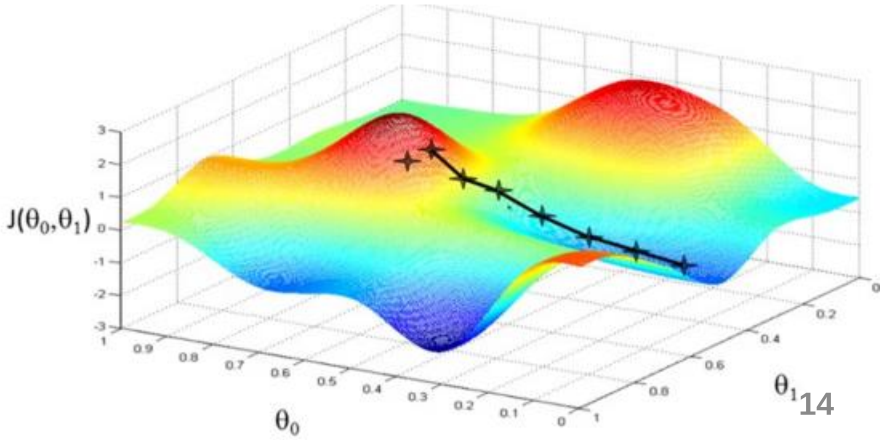
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$



Linear Regression (LinReg):

starting from different initial values of θ the cost function J should always converge (**maybe to a local minimum !!!**) if LinReg works properly.



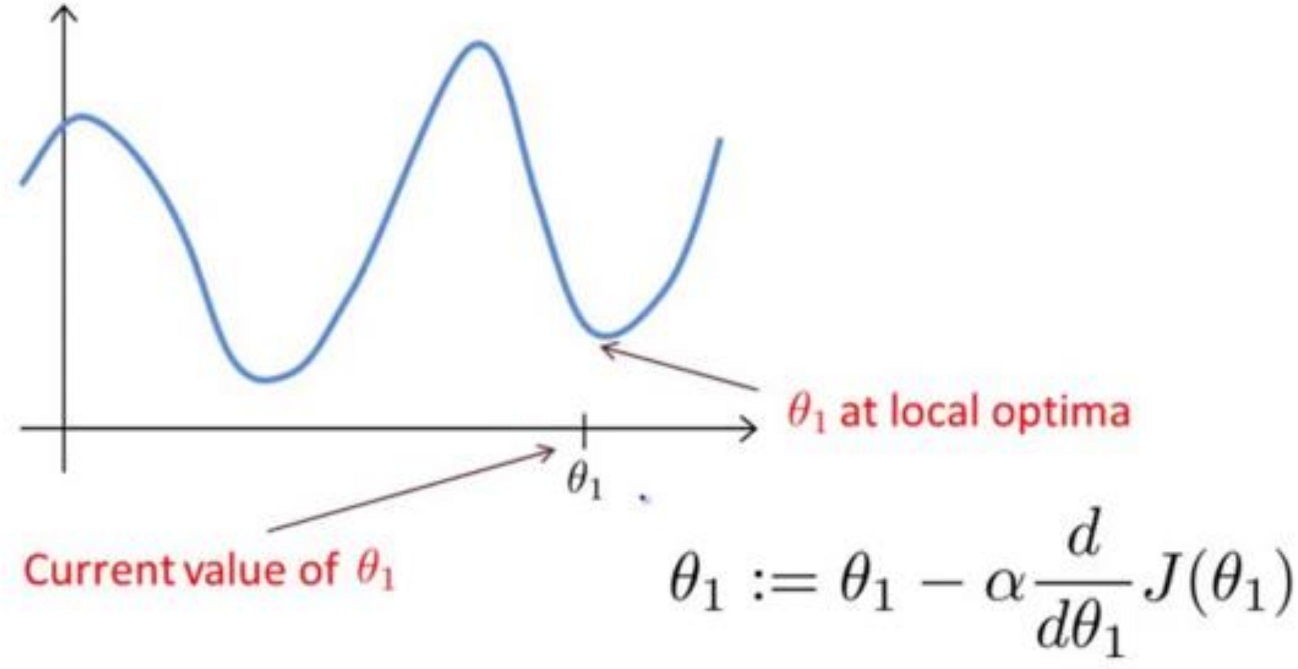


Lin Reg Cost function – local minimum

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

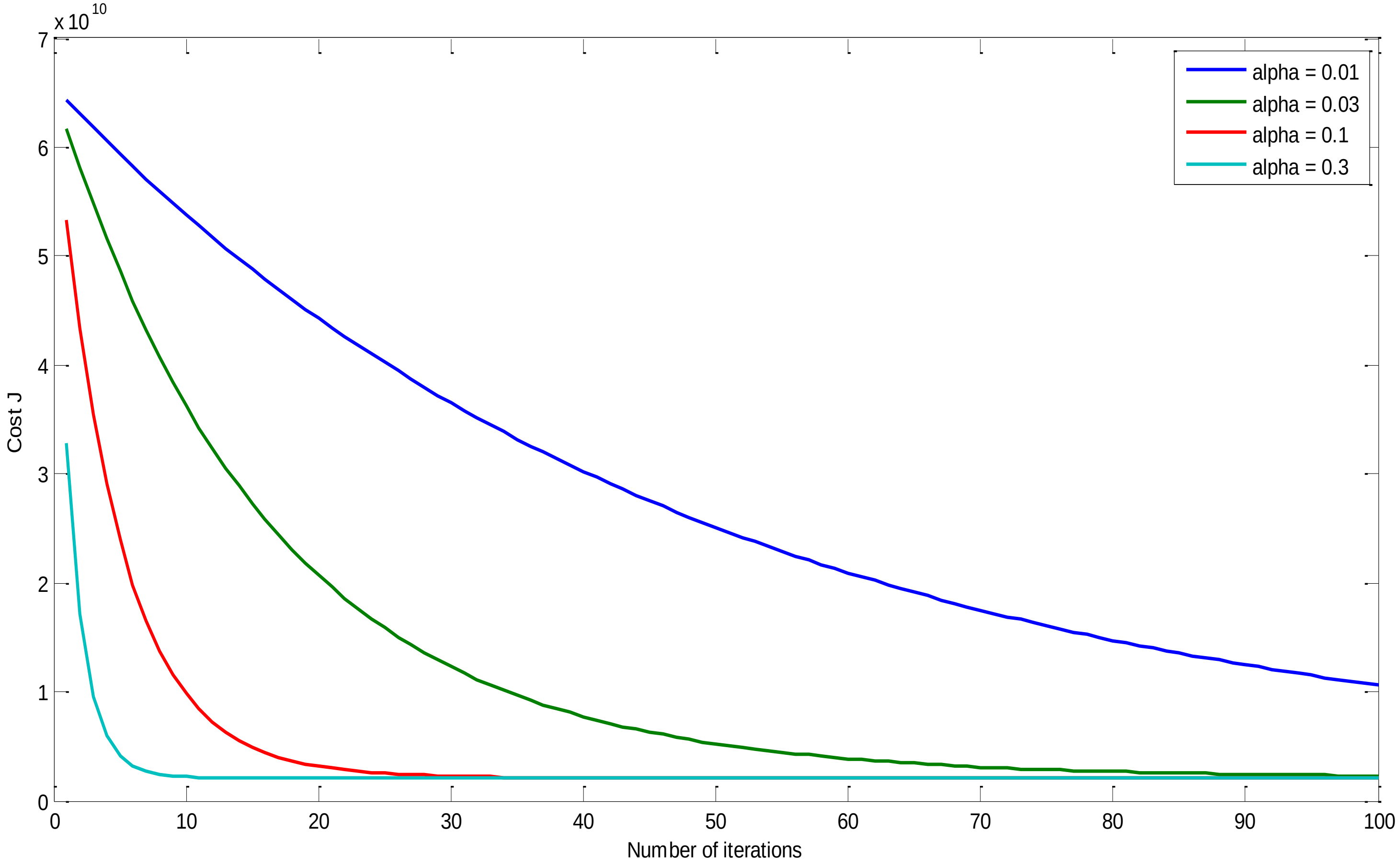
What will do Gradient Descent algorithm at the next step ?

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



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

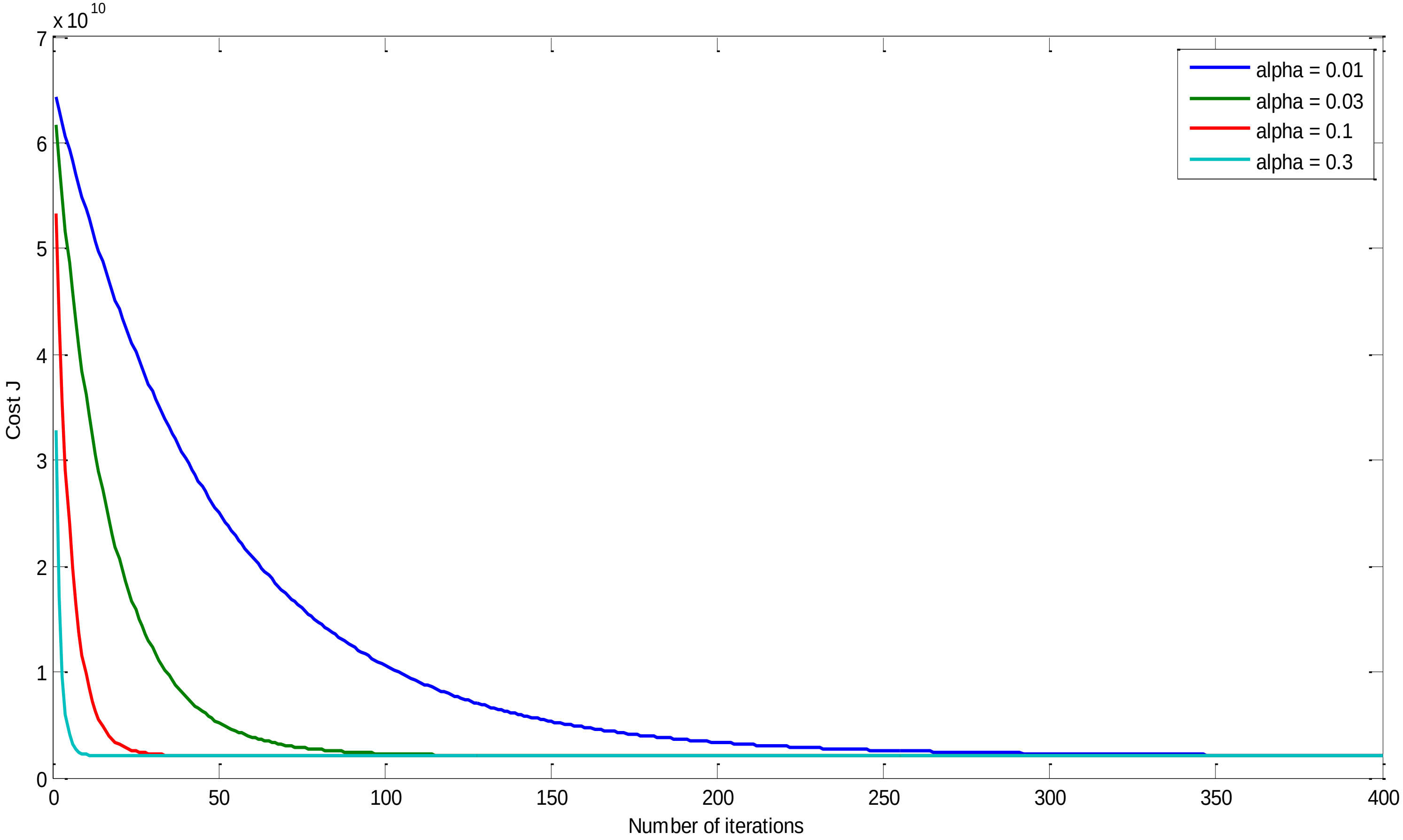
Cost function convergence changing the learning rate (α) -100 iter.



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

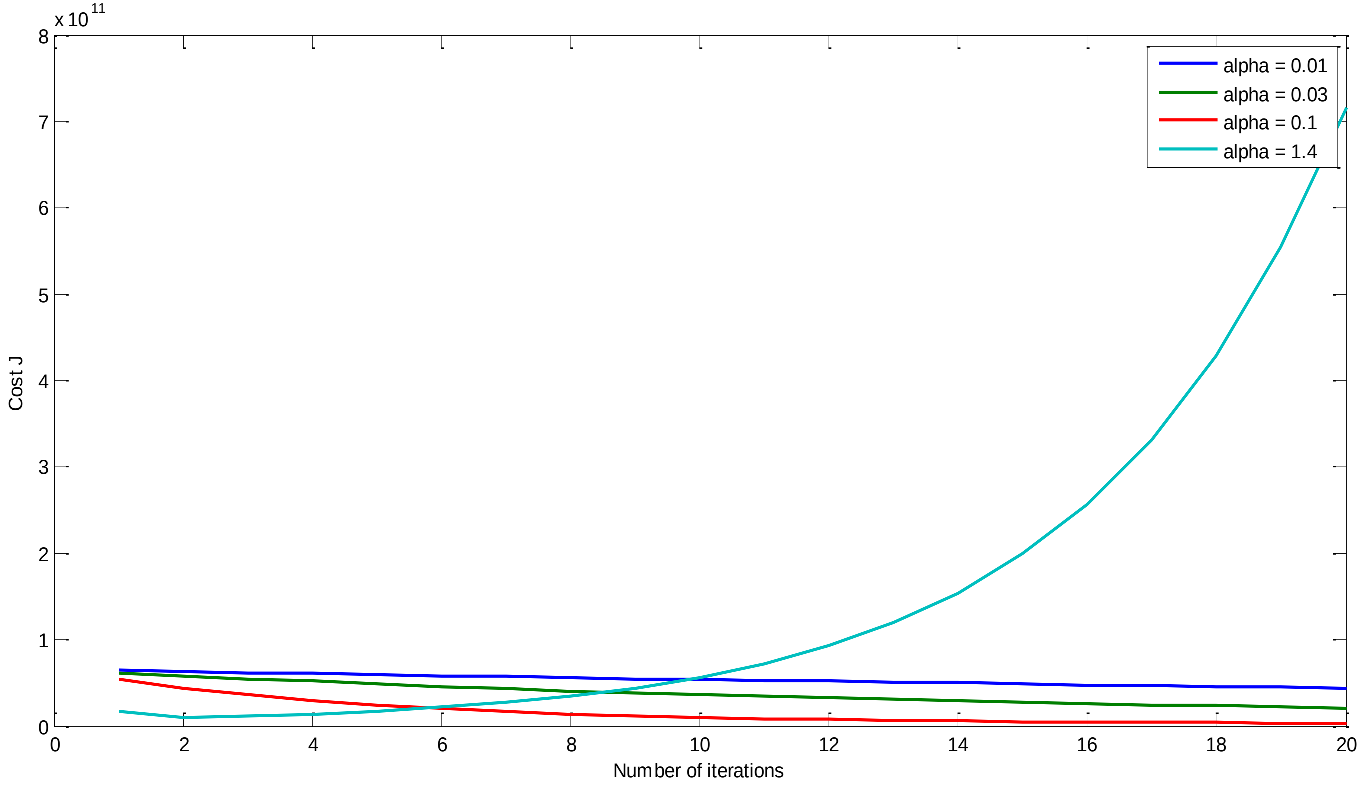
If α too small : slow convergence of the cost function J (Gradient Descent optimization is slow)

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



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

LinReg Cost function convergence - learning rate variation (α)

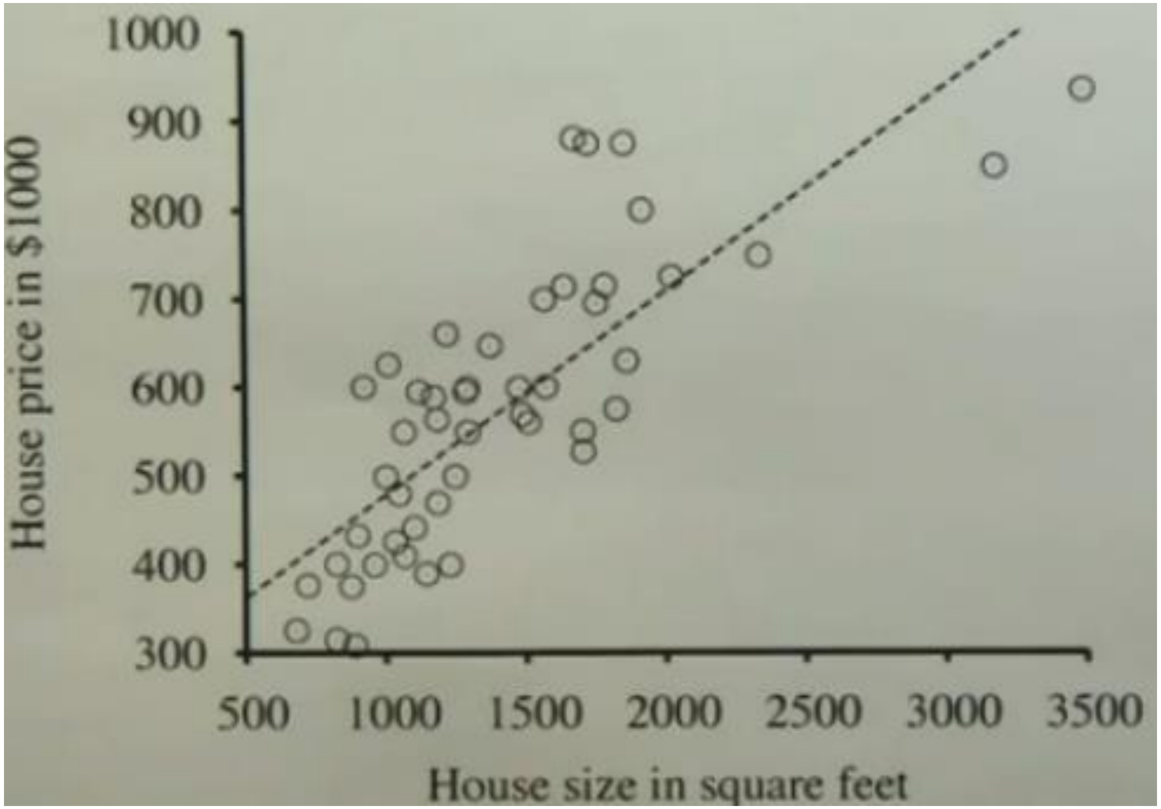


If α too large: the cost function J may not converge. It may diverge **!**

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

Univariate Regression

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



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).

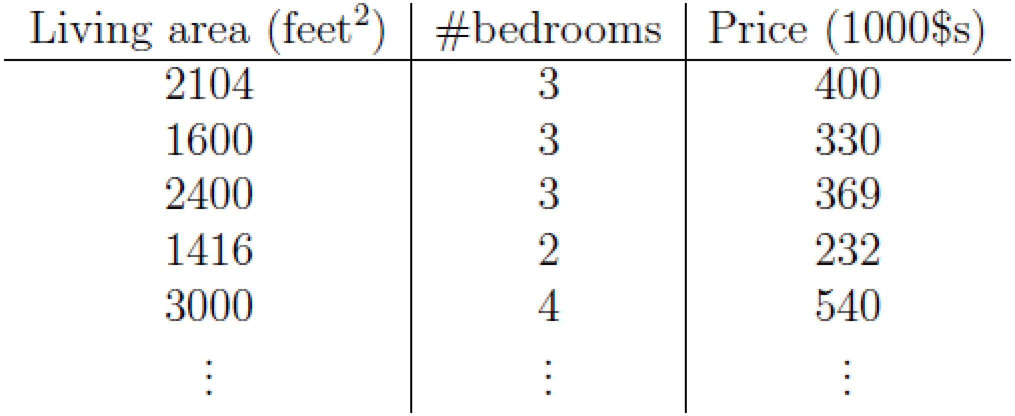
Example

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

	feature x_1	feature x_2	.....	feature x_n	output(label) y
Example 1	$x_1^{(1)}$			$x_n^{(1)}$	$y^{(1)}$
Example 2	$x_1^{(2)}$			$x_n^{(2)}$	$y^{(2)}$
…					
Example i	$x_1^{(i)}$			$x_n^{(i)}$	$y^{(i)}$
…					
…					
Example m	$x_1^{(m)}$			$x_n^{(m)}$	$y^{(m)}$

Multivariate Regression

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

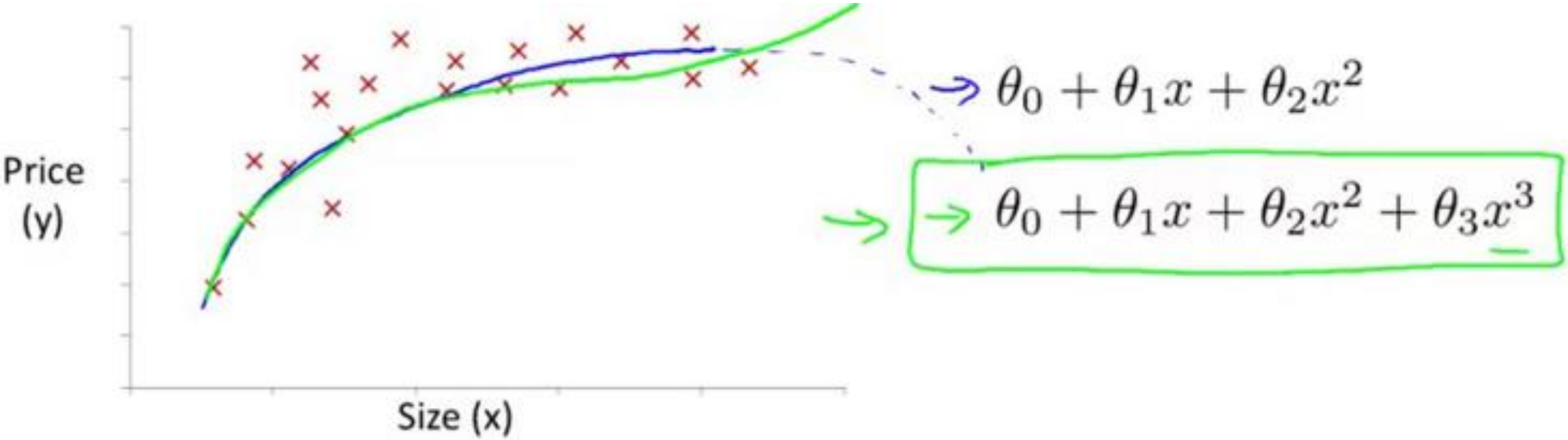


Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = [\theta_0 \quad \theta_1 \quad \theta_2] \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \end{bmatrix} = \vec{\theta}^T \vec{x}$$

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]

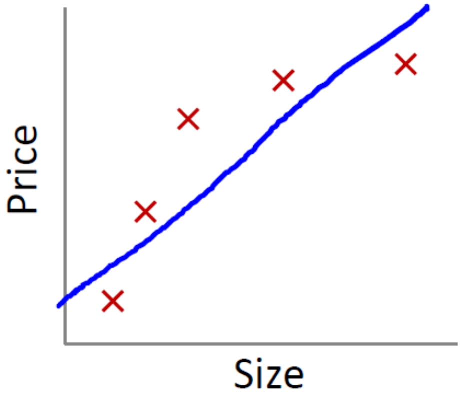


$$
\begin{aligned}
h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\
&= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= (size) \\
x_2 &= (size)^2 \\
x_3 &= (size)^3
\end{aligned}
$$

Overfitting problem

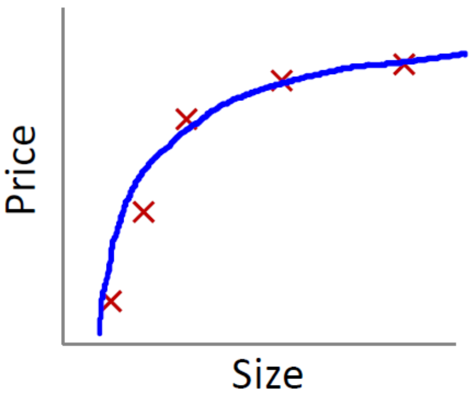
Overfitting: If we have too many features (e.g. high order polynomial model), the model may fit the training data very well but fail to generalize to new examples (e.g. predict prices on new examples).



underfit

(1st order polin. model)

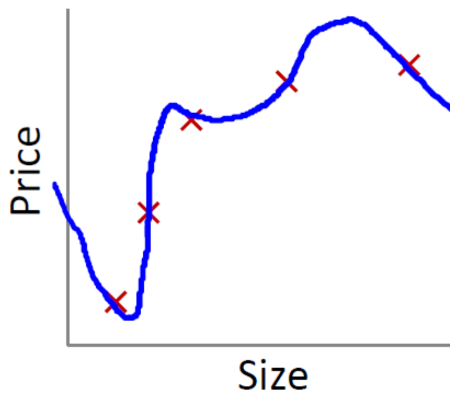
$$h_\theta(x) = \theta_0 + \theta_1 x$$



just right

(3rd order polinom. model)

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



overfit

(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 features (x1,…x100) the model may fit the training data very well but fails to **generalize** to new examples.

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

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \vec{\theta}^T \vec{x}$$

How to deal with overfitting problem ?

1. Reduce number of features.

— Manually select which features to keep.

— Algorithm to select the best model complexity.

2. Regularization (add extra term in the cost function)

Regularization methods shrink model parameters θ towards zero to prevent overfitting by reducing the variance of the model.

2.1 Ridge Regression (L2 norm)

— Reduce magnitude of θ (but never make them =0) => keep all features

— Works well when all features contributes a bit to the output y.

2.2 Lasso Regression (L1 norm)

- May shrink some of the elements of vector θ to become = 0.
- Eliminate some of the features => Serve as feature selection

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
(add extra regularization term
don’t regularize θ_0

Ridge Regression

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

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

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 =>

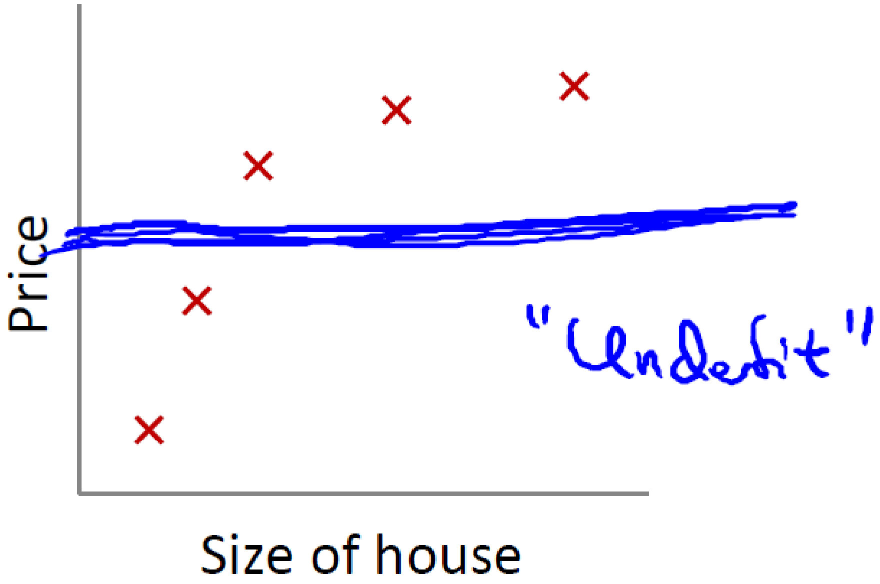
$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{for } j = 0$$

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

Regularized Linear Regression

What if lambda is set to an extremely large value ?

- The overfitting will not be overcome.
- The model will be underfitted (fails to fit even training data).
- Gradient descent will fail to converge.



Regularization: Lasso Regression

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

Ridge Regression shrinks θ towards zero, but never equal to zero => all features are kept in the model.

Lasso Regression may get some θ to become exactly zero => reduces the number of features, serves as feature selection method.

Lasso Regression involves absolute values (not differentiable)=> computing is challenging

Lasso regression is preferred when the goal is **feature selection**, resulting in a simpler and more interpretable model with fewer variables.
Ridge regression tends to favor a model with a higher number of parameters, as it **shrinks less important coefficients** but keeps them in the model.

Regularization: Final notes

Lasso vs. Ridge

Ridge regularization:

- produce more complex models with better prediction power
- may suffer from overfitting due to its reliance on all available features

Lasso regularization:

- reduce the number of features used in a model and eliminate noisy ones
- produce simpler models that are more likely to generalize.

Lasso and Ridge regularization can both be used together.

The combination is known as **elastic net regularization** and can produce **simpler** models while still utilizing most or all of the available features.

Popular in machine learning due to its capability to improve prediction accuracy while minimizing overfitting.

Important Note:
Consider which technique is more suitable for a given problem, some scenarios require specific approach.

Regularization: Final notes

Lasso and Ridge Regression are two of the most popular techniques for regularizing linear models, which often yield more accurate predictions than traditional linear models. These methods reduce the model's complexity by introducing shrinkage or adding a penalty to complex coefficients.

•The Ridge-Lasso approach is limited in requiring the input features to be standardized before fitting the model. It means that any feature with a large range of values can bias results because of its scale relative to other features with smaller ranges.

•Furthermore, if the data points contain outliers or noise, then this could produce inaccurate predictions due to the penalty terms.

•Additionally, Ridge and Lasso Regressions can be slow when applied to large datasets because of the computation time needed to perform regularization.

•Lastly, these methods require careful selection of hyperparameters (i.e., regularization strength), which can induce further computational costs and time.