CENG 463
Machine Learning

Lecture 05 - Multivariate Linear Regression

Slides were prepared using the material provided in Stanford's Machine Learning Course by Andrew Ng.

Multiple Variables

- By ‘multivariate’, we mean there are multiple variables/features:

x_1	x_2	x_3	x_4	y
Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	46	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

- Notation:
  - **n**: number of features (4 in this example)
  - $\mathbf{x}^{(i)}$: input features of i^{th} training example
  - $\mathbf{x}_j^{(i)}$: value of the feature j in the i^{th} training example

Multiple Variables

- Hypothesis for house pricing example:
  - With new features: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$
  - In general: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots\dots + \theta_n x_n$
- To vectorize our computations, we can define $x_0 = 1$.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \qquad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad h_\theta(x) = \theta^T \cdot x$$

Gradient Descent with Multiple Variables

- Hypothesis:
  - $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots.. + \theta_n x_n$
- Parameters:
  - $\theta_0, \theta_1, \theta_2, \ldots \theta_n \longrightarrow \theta$
- Cost function:
  - $J(\theta_0, \theta_1, \theta_2, \ldots \theta_n) \longrightarrow J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left| h_\theta\left(x^{(i)}\right) - y^{(i)} \right|^2$
- Gradient Descent:
  - $\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$

Gradient Descent with Multiple Variables

- Previously (n=1):

repeat until convergence {

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

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

}

- Simultaneous update

- Now (n > 1):

$x_0^{(i)} = 1$

repeat until convergence {

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

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

...

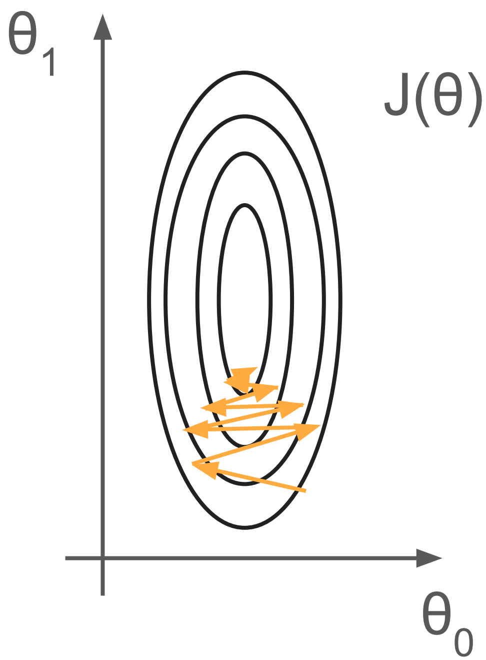
$$\theta_n := \theta_n - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_n^{(i)}$$

}

- Simultaneous update

Feature Scaling

- To make the gradient descent converge more quickly, we have to make sure the features are on a similar scale.
- Otherwise the cost function will have a <u>skewed</u> shape.
- Example:
  - x_1 = size (0-2000 feet^2)
  - x_2 = number of bedrooms (1-5)

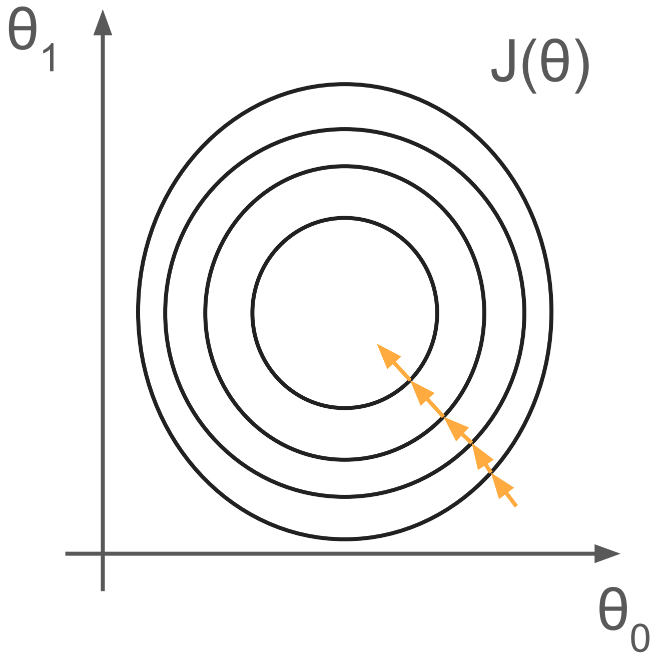


Feature Scaling

- When the features are scaled, cost function has a more 'balanced' shape, making the gradient descent converge more quickly.
- Example:

$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000} \qquad x_2 = \frac{\text{number of bedrooms}}{5}$$

Approximately $0 \le x_1 \le 1$ and $0 \le x_2 \le 1$



Mean Normalization

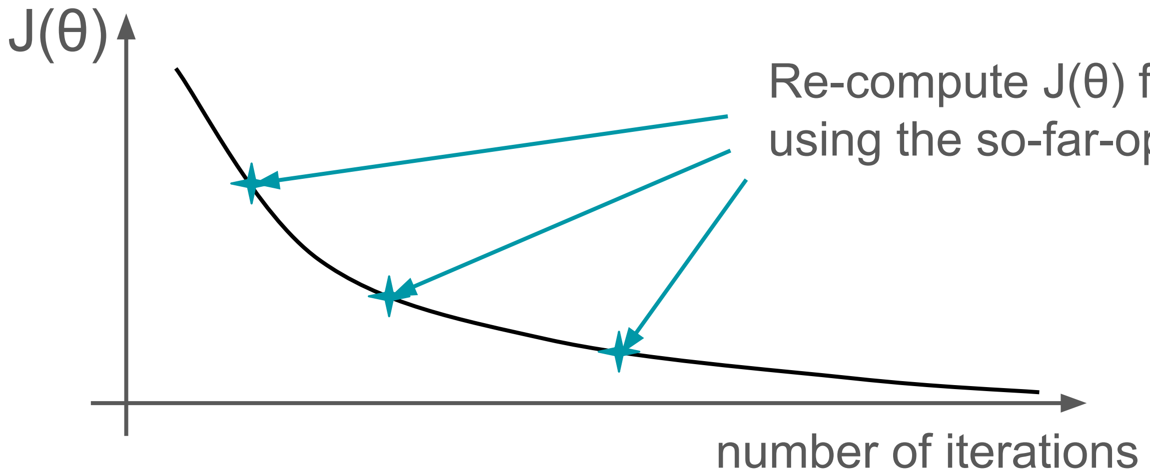
- In addition, mean normalization can be applied to the features.
- All we have to do is to replace x_i with $x_i - \mu_i$ to make features have approximately zero mean. (except for $x_0 = 1$)
- In ideal case, you would have:
  - $-0.5 < x_1, x_2 < 0.5$ or $-1 < x_1, x_2 < 1$
- E.g.

$$x1 = \frac{\text{size – mean(size)}}{\text{max. size after subtracting the mean}}$$

$$x2 = \frac{\text{\# of bedrooms – mean(\# of bedrooms)}}{\text{max. \# of bedrooms after subtracting the mean}}$$

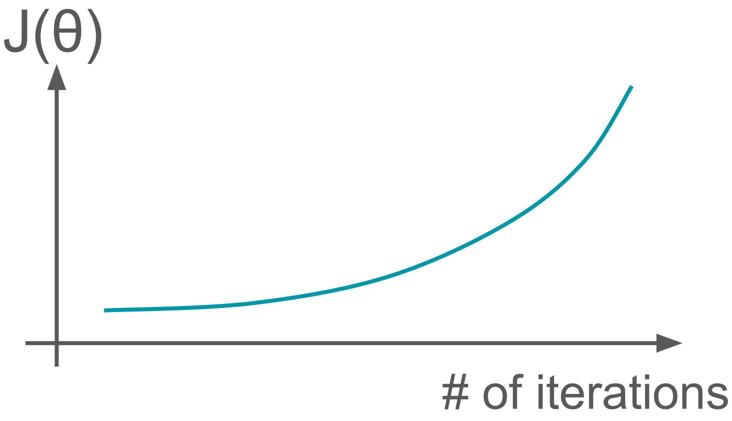
Learning Rate

- How do you know that your gradient descent works?
  - $J(\theta)$ should decrease after every iteration.

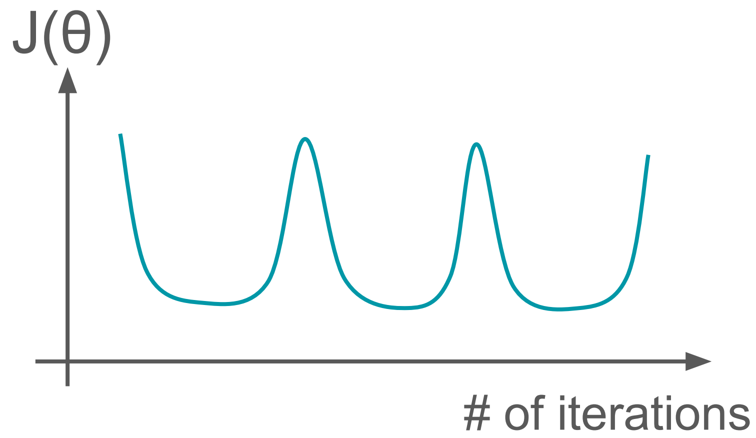


- Where to stop?
  - Declare convergence when $J(\theta)$ decreases less than ε for one step. Typical value for ε might be 10^{-3}.

Learning Rate



Gradient descent is not working!



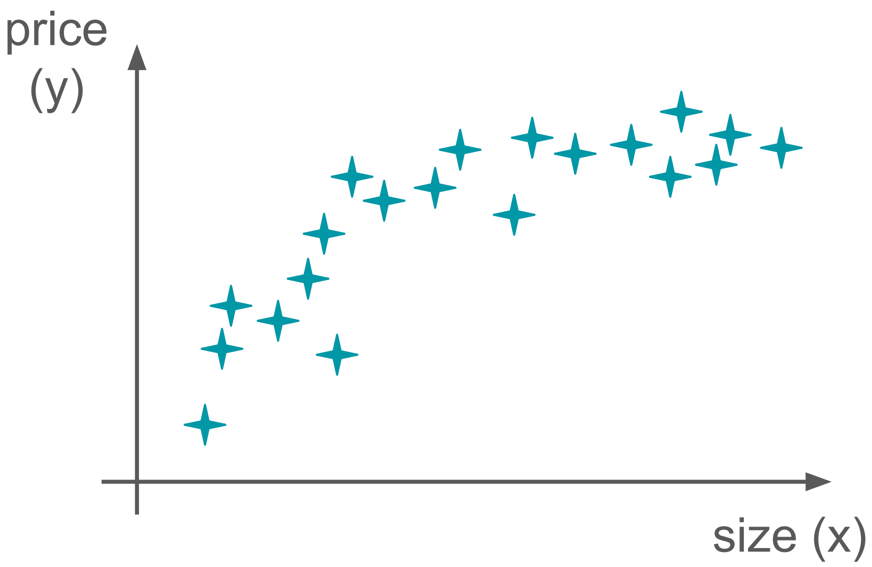
Gradient descent is not converging!
Try smaller α.

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

- But if α too small, it can be very slow to converge.
  - Best practice is to try different α, changing it by an order of 3, e.g. 1, 0.3, 0.1, 0.03, 0.01...

Polynomial Regression

- We can model polynomial functions using linear regression.



- We may want to fit a quadratic model:
  $\theta_0 + \theta_1 x + \theta_2 x^2$
  or a cubic model:
  $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

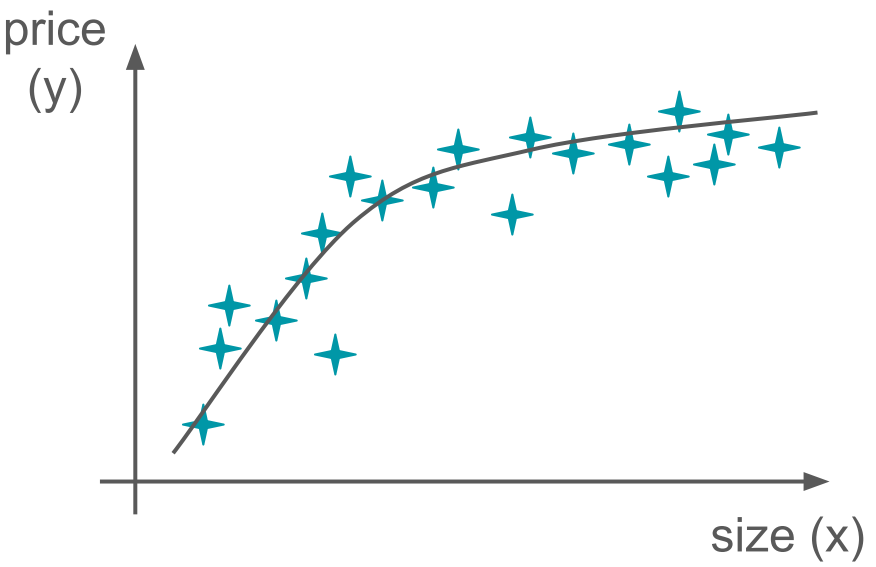
Polynomial Regression

- We modify our single variable (house size) hypothesis to cover polynomial models:
  $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \ldots$
  $h_\theta(x) = \theta_0 + \theta_1 \cdot size + \theta_2 \cdot size^2 + \theta_3 \cdot size^3 + \ldots$
  $h_\theta(x) = \theta_0 + \theta_1 \quad x_1 \quad + \theta_2 \quad x^2 \quad + \theta_3 \quad x^3 \quad + \ldots$

- Algorithmically, it is still a multivariate linear regression.
- The new features we have are:
    - $x_1 = size$
    - $x_2 = size^2$
    - $x_3 = size^3$
    - ...

Polynomial Regression

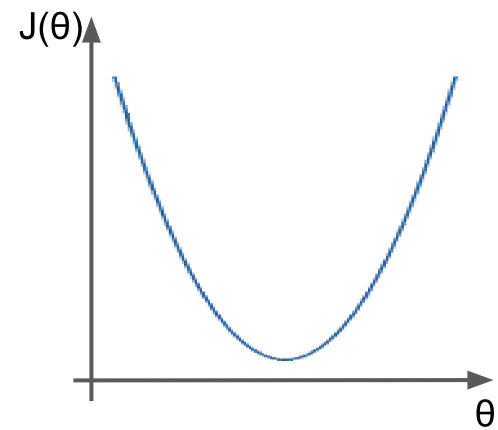
- Don’t forget: Features do not have to be the terms of a regular polynomial.
  - E.g. $\theta_0 + \theta_1 x + \theta_2 \sqrt{x}$



Normal Equation

- There is another way to solve for θ **analytically**; i.e. we can estimate optimum parameters (θ) in one step using linear algebra.
- If there was one variable: $J(\theta) = a + b\theta + c\theta^2$, the solution would be easy using calculus:

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$



- In multivariate case, it is not easy.
    - For all j:

$$\frac{\partial J(\theta)}{\partial \theta_j} = 0$$

Normal Equation

- Instead, we construct an equation and use linear algebra to solve it. For this example, assume m = 4.

x_1	x_2	x_3	x_4	y
Size (feet^2)	**Number of bedrooms**	**Number of floors**	**Age of home (years)**	**Price ($1000)**
2104	5	1	46	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 & 460 \\ 1 & 1416 & 3 & 2 & 40 & 232 \\ 1 & 1534 & 3 & 2 & 30 & 315 \\ 1 & 852 & 2 & 1 & 36 & 178 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$X \cdot \theta = y$$

Normal Equation

- Since our solution is an approximation, i.e. values contain noise, the following is not exactly true:

$$X \cdot \theta = y \quad \text{or} \quad X \cdot \theta - y = 0$$

- We need the best θ to minimize $X \cdot \theta - y$.
- It is also called least-squares problem, solved by:

$$\theta = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

- In PYTHON, this can be calculated using numpy:
  `θ = numpy.linalg.pinv(X)*y`
  where `pinv` stands for pseudo-inverse.

Normal Equation: General Case

- For **m** samples, **n** variables/features:

  $(x^{(1)},y^{(1)}), \ldots ,(x^{(m)},y^{(m)})$

- We design a matrix and a vector:

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ . \\ . \\ . \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$X = \begin{bmatrix} .. & x^{(1)T} & .. \\ .. & x^{(2)T} & .. \\ .. & .. & .. \\ .. & x^{(m)T} & .. \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ .. \\ y^{(m)} \end{bmatrix}$$

to be used in `θ = numpy.linalg.pinv(X)*y`

Normal Equation vs. Gradient Descent

Gradient Descent

- Need to choose α .
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α .
- Don’t need to iterate.
- No need feature scaling.
- Need to compute pinv(X) where X^T is nxm. It is slow if n is very large.

Summary

- We have learned about:
  - How to handle multiple variables for linear regression
  - Gradient descent with multiple variables
  - How to choose learning rate (α)?
  - Polynomial Regression
  - Normal Equation
  - Normal Equation vs. Gradient Descent