



Machine Learning

Logistic Regression

Advanced optimization

Optimization algorithm

Cost function $J(\theta)$. Want \min_{θ} $J(\theta)$.

Given θ , we have code that can compute

$\rightarrow - J(\theta)$
 $\rightarrow - \frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \dots, n$)

Gradient descent:

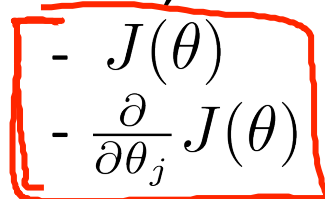
Repeat {

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

}



Optimization algorithm

Given θ , we have code that can compute





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


(for $j = 0, 1, \dots, n$)



Optimization algorithms:

- 
- Gradient descent
 - Conjugate gradient
 - BFGS
 - L-BFGS
- 

Advantages:

- No need to manually pick α
 - Often faster than gradient descent.
- 

Disadvantages:

- More complex
- 

Example:

$\min_{\theta} J(\theta)$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$\theta_1 = 5, \theta_2 = 5.$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

```
function [jVal, gradient]
    = costFunction(theta)
    jVal = (theta(1)-5)^2 + ...
           (theta(2)-5)^2;
    gradient = zeros(2,1);
    gradient(1) = 2*(theta(1)-5);
    gradient(2) = 2*(theta(2)-5);
```

```
options = optimset('GradObj', 'on', 'MaxIter', '100');
```

```
initialTheta = zeros(2,1);
```

```
[optTheta, functionVal, exitFlag] ...
    = fminunc(@costFunction, initialTheta, options);
```

↑

↑

$\theta \in \mathbb{R}^d$ $d \geq 2$

$$\underline{\text{theta}} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Handwritten annotations in blue:

- θ_0 is labeled $\text{theta}(1)$ with an arrow pointing to it.
- θ_1 is labeled $\text{theta}(2)$ with an arrow pointing to it.
- θ_n is labeled $\text{theta}(n+1)$ with an arrow pointing to it.

```
function [jVal, gradient] = costFunction(theta)
```

```
    jVal = [code to compute  $J(\theta)$ ];
```

```
    gradient(1) = [code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$ ];
```

```
    gradient(2) = [code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$ ];
```

```
    :
```

```
    gradient(n+1) = [code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$ ] ;
```