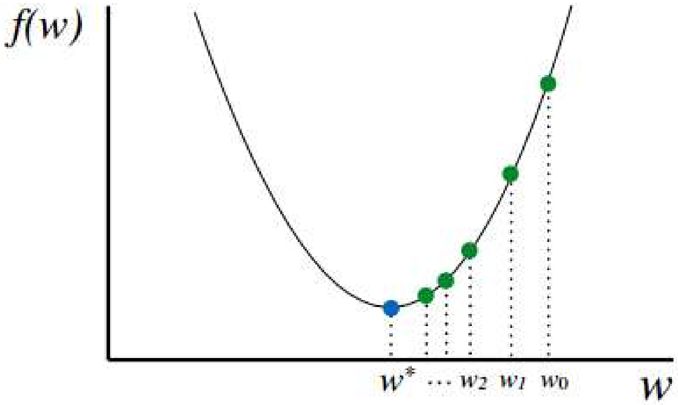
Introduction to AI for postgraduate students

Lecture Note 3
Optimization

Reading Materials

- http://www.deeplearningbook.org/contents/linear_algebra.html
- http://www.stat.cmu.edu/~ryantibs/convexopt/lectures/stochastic-gd.pdf

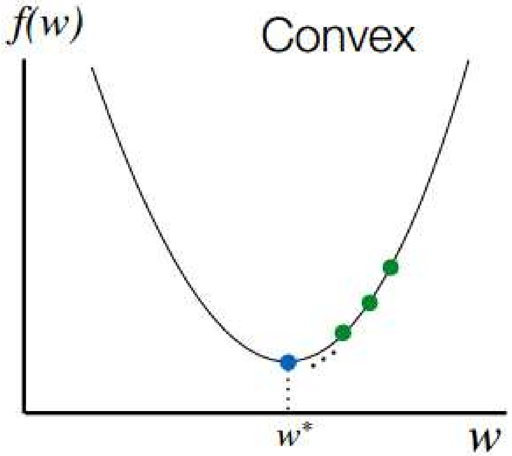
Gradient Descent: Univariate

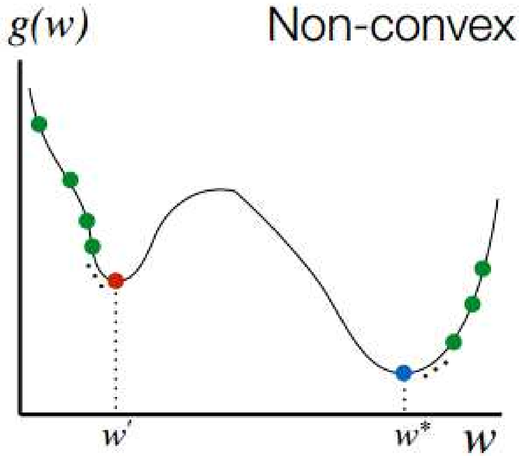


At a random point Start
Repeat

- Determine a descent direction
- Choose a step size Choose
- Update

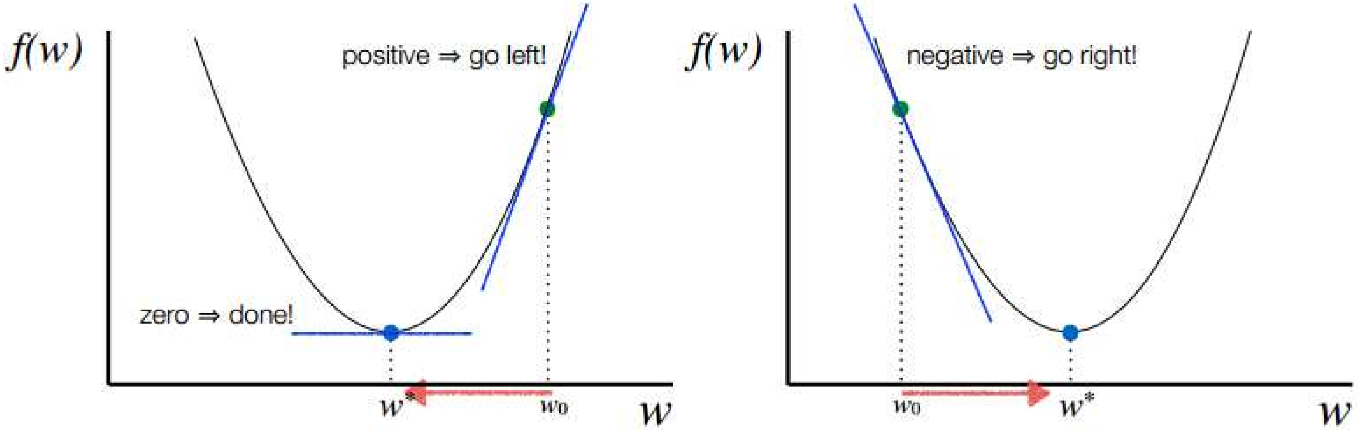
Until some criterion is satisfied



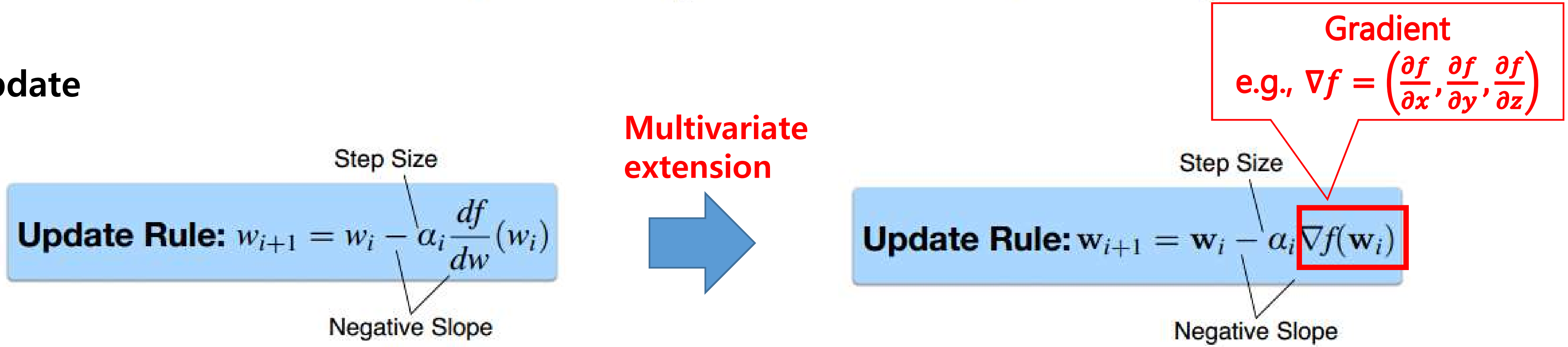


Gradient Descent: Univariate

- **Choosing the descent direction**

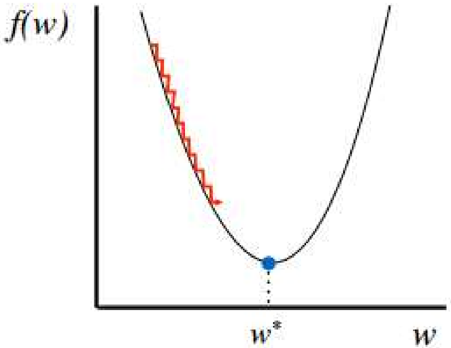


- **Update**

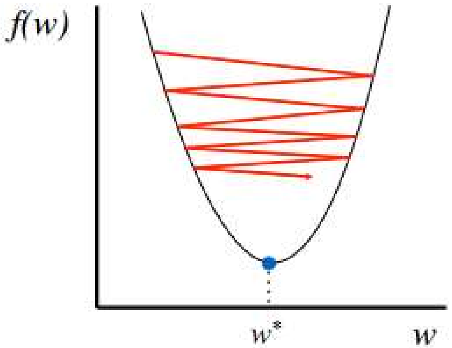


Gradient Descent: Univariate

- **Choosing the step size**

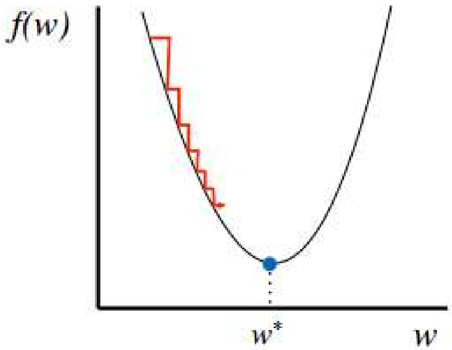


Too small: converge very slowly



Too big: overshoot and even diverge

Better way

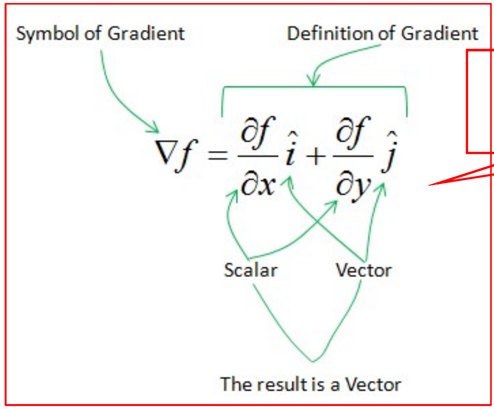


Reduce size over time

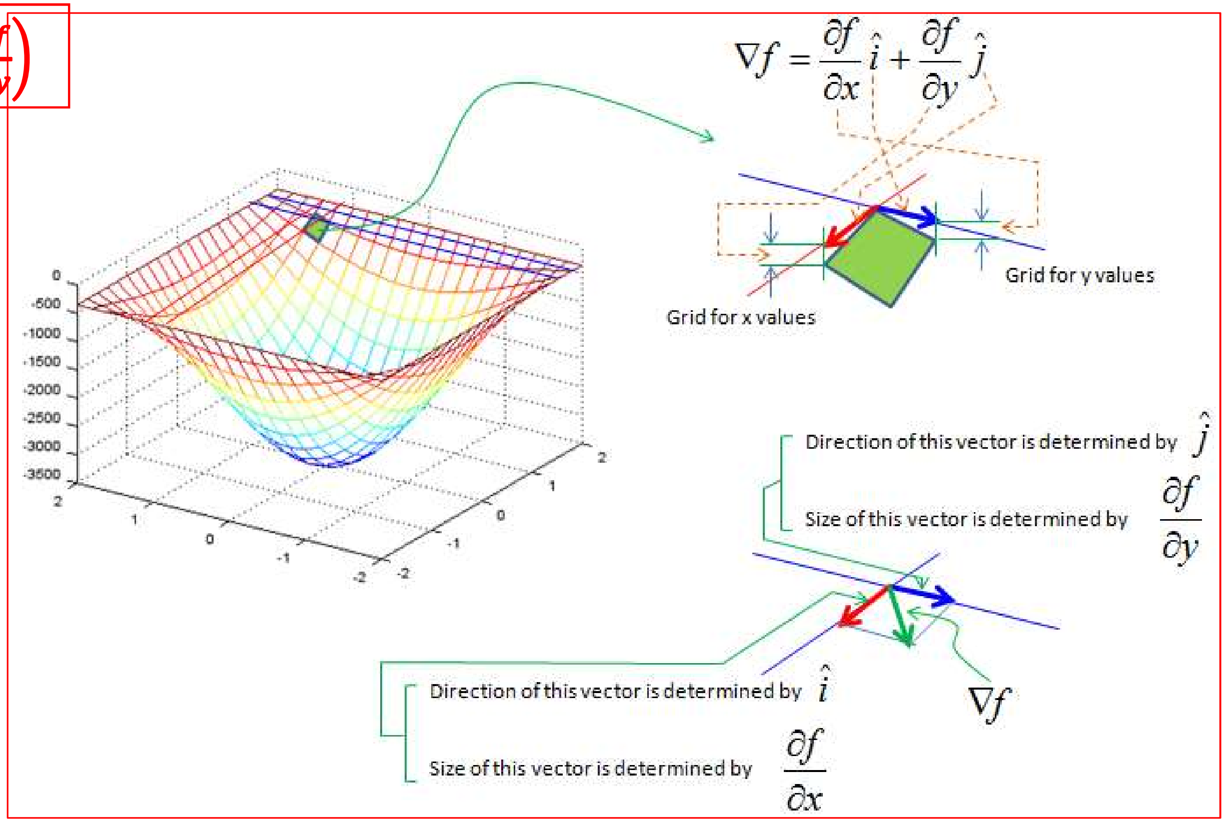
Gradient Descent: Multivariate

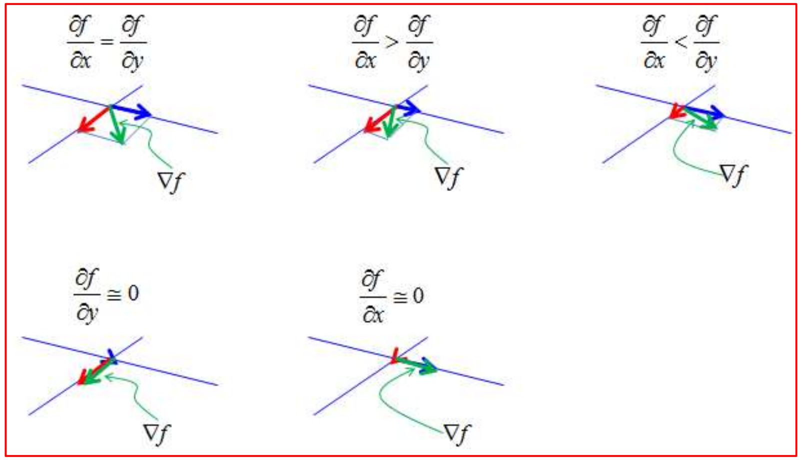
- **Definition of a Gradient vector**

*Fig. from http://www.sharetechnote.com/html/Calculus_Gradient.html



Or $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$





Gradient Descent: Multivariate

- **Consider a 3-D function:** $T(x, y, z)$
- **Gradient vector is defined by:** $\mathbf{grad}\, T \equiv \left(\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y}, \frac{\partial T}{\partial z}\right).$
- **By the chain rule:**

$$\boxed{dT = \frac{\partial T}{\partial x} dx + \frac{\partial T}{\partial y} dy + \frac{\partial T}{\partial z} dz.}$$

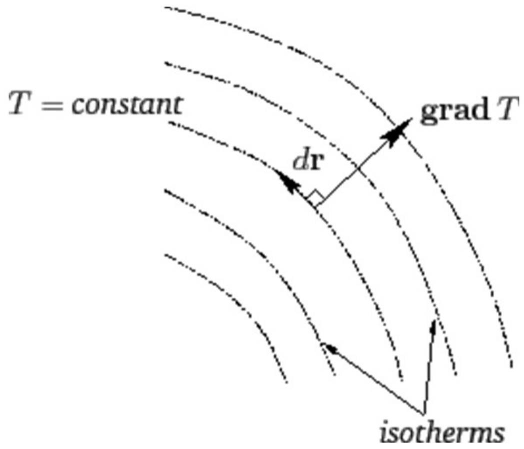
$$\mathbf{dr} \equiv (dx, dy, dz)$$

$$\Downarrow$$

$$dT = \mathbf{grad}\, T \cdot \mathbf{dr}.$$

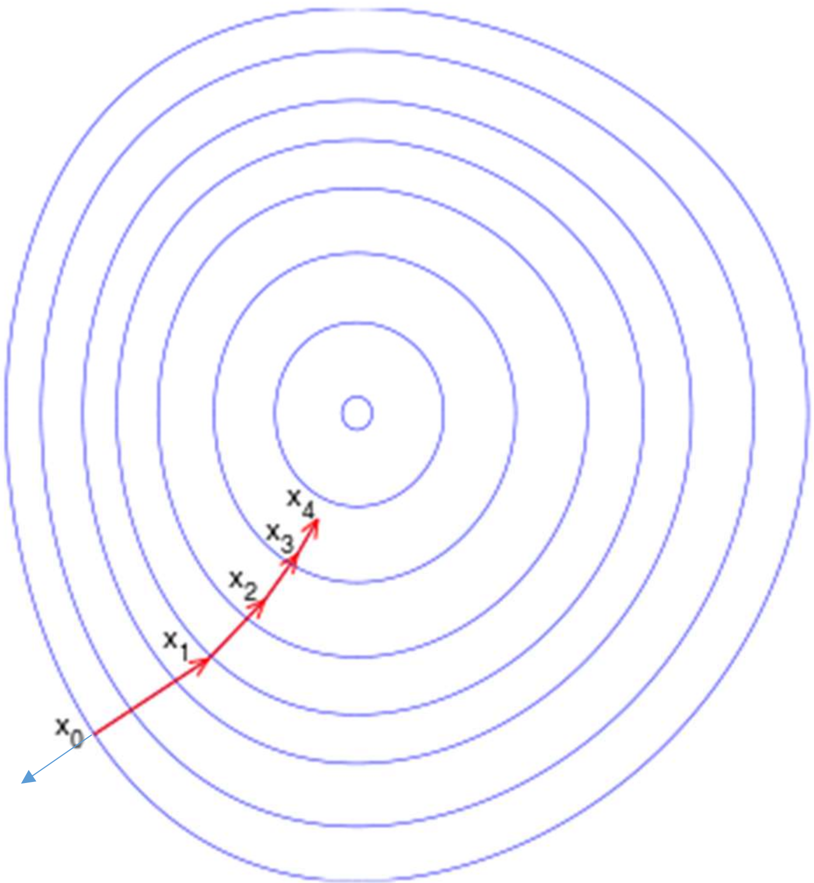
- **Suppose that** d**T**=0 **for some** dr**:**

$$\boxed{dT = \mathbf{grad}\, T \cdot \mathbf{dr} = 0.}$$

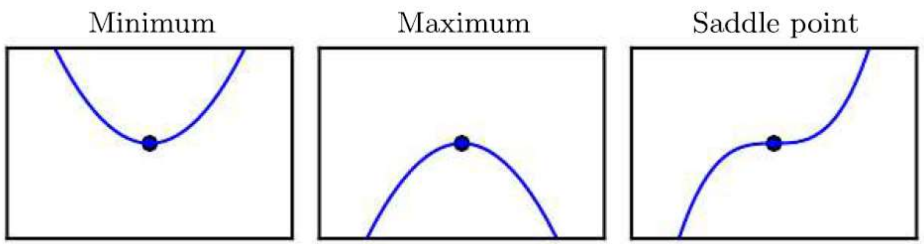


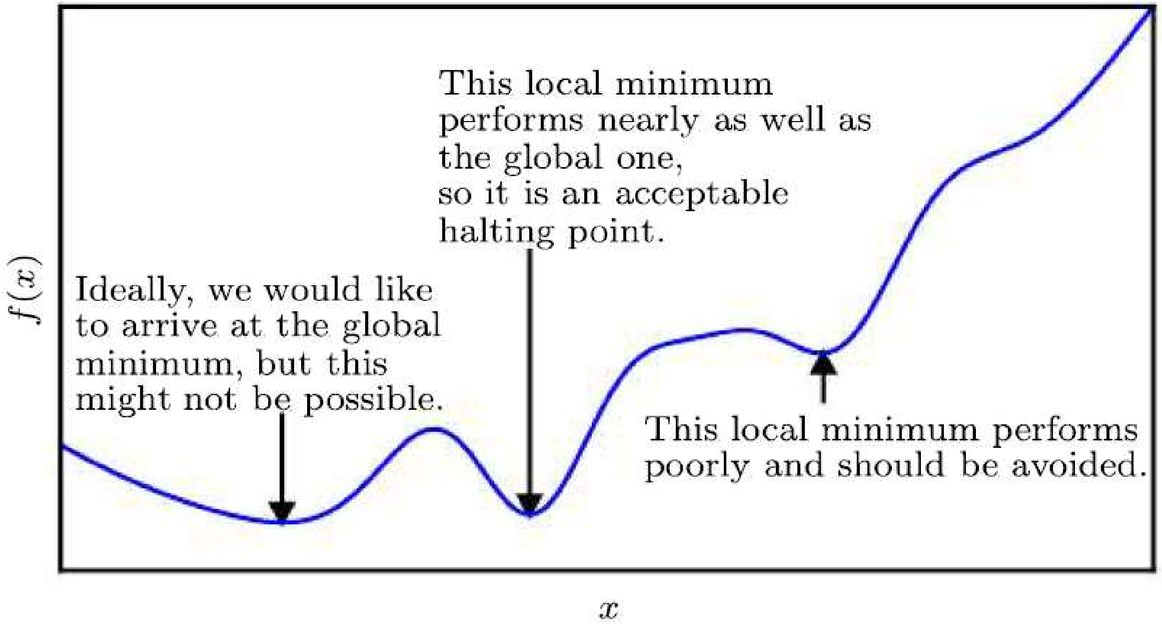
Gradient Descent: Multivariate

- **Illustration of gradient vectors and contours**



Limitations of Gradient Descent





Second Derivation Method: Gradient Descent

- **Directional derivative in direction "u" (unit vector)**
  - Slope of the function *f* in direction **u**

$$\boxed{\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u})}$$

  - By the chain rule:

$$\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u}) \Rightarrow \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \text{ when } \alpha = 0$$

- **We find the unit vector "u" such that the directional derivative is minimized → for steepest descent**

$$\min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u} = 1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

$$= \min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u} = 1} ||\boldsymbol{u}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\theta$$

where θ is the angle between $\boldsymbol{u}$ and the gradient

$$\Rightarrow \min_{\boldsymbol{u}} \cos\theta \Rightarrow$$

Solution: vector u should be in the **opposite** direction of the Gradient

Second Derivation Method: Gradient Descent

- **Steepest descent update**

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

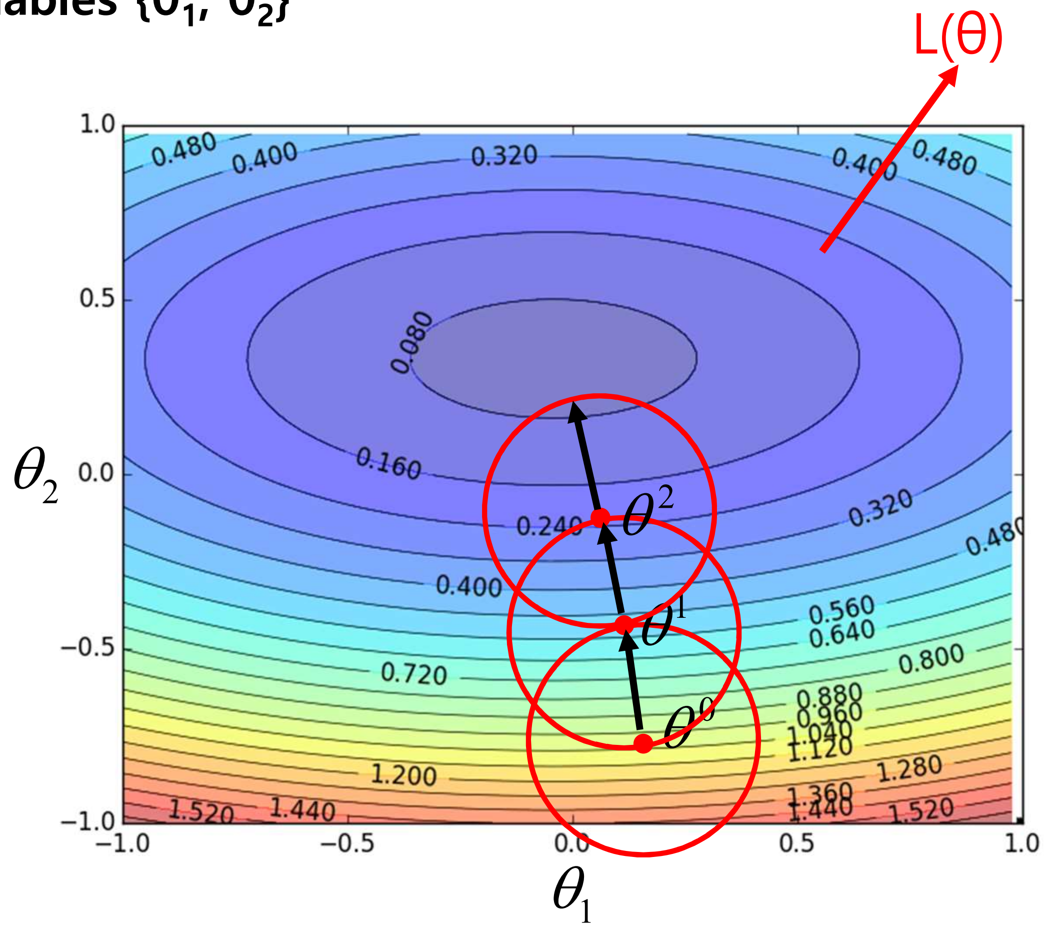
Learning rate or
Step size

- **Analytical discussion on choosing the step size**
  - Read "4.3.1 Beyond the Gradient: Jacobian and Hessian Matrices"

Third Derivation Method: Gradient Descent

- **Suppose that θ has two variables {θ_1, θ_2}**

Given a point, we want to find the point with the smallest value nearby.



Third Derivation Method: Gradient Descent

- **Multivariate Taylor series**

$$h(x,y)=h(x_0,y_0)+\frac{\partial h(x_0,y_0)}{\partial x}(x-x_0)+\frac{\partial h(x_0,y_0)}{\partial y}(y-y_0)$$

+ something related to $(x-x_0)^2$ and $(y-y_0)^2$

+ ……

When x and y are close to x_0 and y_0

$$h(x,y)\approx h(x_0,y_0)+\frac{\partial h(x_0,y_0)}{\partial x}(x-x_0)+\frac{\partial h(x_0,y_0)}{\partial y}(y-y_0)$$

Third Derivation Method: Gradient Descent

Based on Taylor Series:

If the red circle is ***small enough***, in the red circle

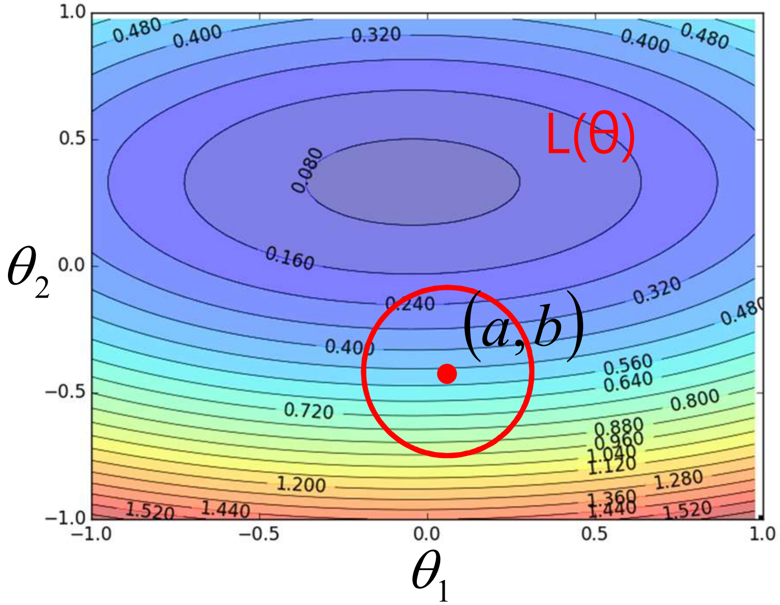
$$\mathrm{L}(\theta) \approx \mathrm{L}(a,b) + \frac{\partial \mathrm{L}(a,b)}{\partial \theta_1}(\theta_1 - a) + \frac{\partial \mathrm{L}(a,b)}{\partial \theta_2}(\theta_2 - b)$$

$$s = \mathrm{L}(a,b)$$

$$u = \frac{\partial \mathrm{L}(a,b)}{\partial \theta_1}, v = \frac{\partial \mathrm{L}(a,b)}{\partial \theta_2}$$

$$\mathrm{L}(\theta)$$

$$\approx s + u(\theta_1 - a) + v(\theta_2 - b)$$



Third Derivation Method: Gradient Descent

Based on Taylor Series:

If the red circle is ***small enough***, in the red circle

$$\mathrm{L}(\theta) \approx s + u(\theta_1 - a) + v(\theta_2 - b)$$

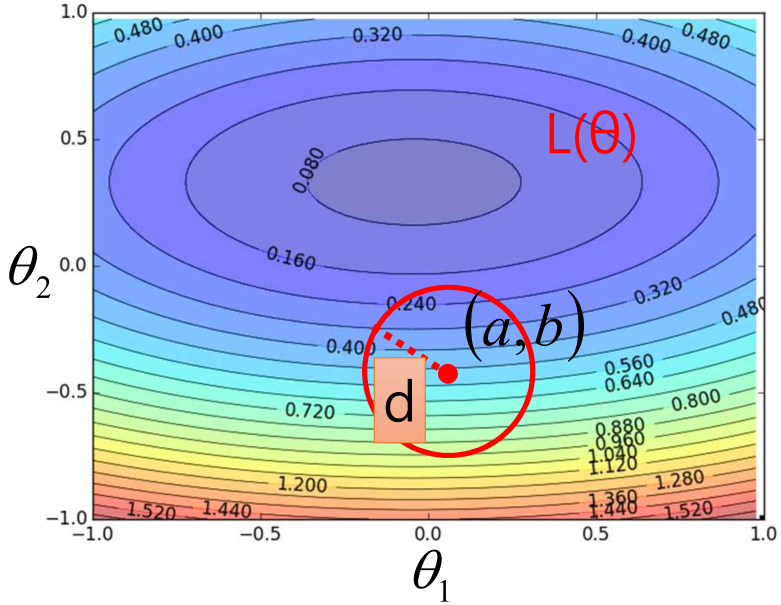
constant

$$s = \mathrm{L}(a,b)$$

$$u = \frac{\partial \mathrm{L}(a,b)}{\partial \theta_1}, v = \frac{\partial \mathrm{L}(a,b)}{\partial \theta_2}$$

Find θ_1 and θ_2 in the red circle ***minimizing*** $\mathrm{L}(\theta)$

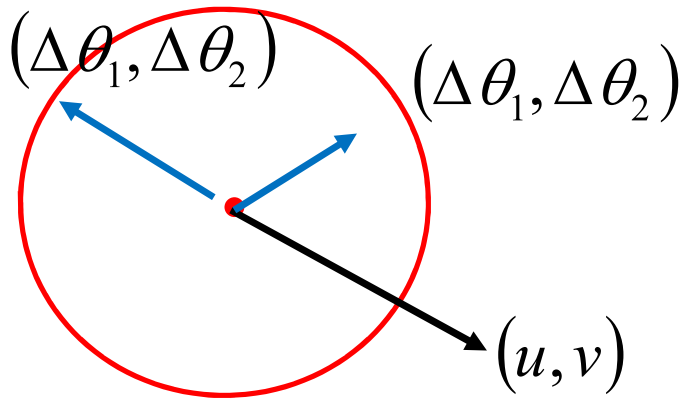
$$(\theta_1 - a)^2 + (\theta_2 - b)^2 \leq d^2$$



Third Derivation Method: Gradient Descent

Red Circle:(If the radius is small)

$$L(\theta) \approx \cancel{s} + u\underbrace{(\theta_1 - a)}_{\Delta\theta_1} + v\underbrace{(\theta_2 - b)}_{\Delta\theta_2}$$



Find θ_1 and θ_2 in the red circle ***minimizing*** $L(\theta)$

$$\underbrace{(\theta_1 - a)}_{\Delta\theta_1}^2 + \underbrace{(\theta_2 - b)}_{\Delta\theta_2}^2 \le d^2$$

To minimize $L(\theta)$

$$\begin{bmatrix} \Delta\theta_1 \\ \Delta\theta_2 \end{bmatrix} = -\eta \begin{bmatrix} u \\ v \end{bmatrix} \Rightarrow \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} - \eta \begin{bmatrix} u \\ v \end{bmatrix}$$

Third Derivation Method: Gradient Descent

Based on Taylor Series:

If the red circle is ***small enough***, in the red circle

constant

$$s = \mathrm{L}(a,b)$$

$$\mathrm{L}(\theta) \approx s + u(\theta_1 - a) + v(\theta_2 - b)$$

$$u = \frac{\partial \mathrm{L}(a,b)}{\partial \theta_1}, v = \frac{\partial \mathrm{L}(a,b)}{\partial \theta_2}$$

Find θ_1 and θ_2 yielding the smallest value of $L(\theta)$ in the circle

$$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} - \eta \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial \mathrm{L}(a,b)}{\partial \theta_1} \\ \frac{\partial \mathrm{L}(a,b)}{\partial \theta_2} \end{bmatrix}$$

This is gradient descent.

Not satisfied if the red circle (learning rate) is not small enough

You can consider the second order term, e.g. Newton's method.

Stochastic Gradient Descent

- **Consider minimizing an average of functions**

$$\min_x \frac{1}{m}\sum_{i=1}^{m} f_i(x)$$

- **Gradient descent would repeat**

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

- **In Stochastic Gradient Descent (SGD) (a.k.a. incremental gradient descent)**

$$\boxed{x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f_{i_k}(x^{(k-1)}), \quad k = 1, 2, 3, \ldots}$$

where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration k

Stochastic Gradient Descent

- **Two rules for choosing i_k at iteration k:**
  - Randomized rule: choose i_k from {1, …, m} uniformly at random
  - Cyclic rule: choose i_k=1, 2, …, m, 1, 2, …, m, …

- **Main appeal of SGD**
  - Iteration cost is independent of m, the number of functions
  - Can also be a big savings in terms of memory usage

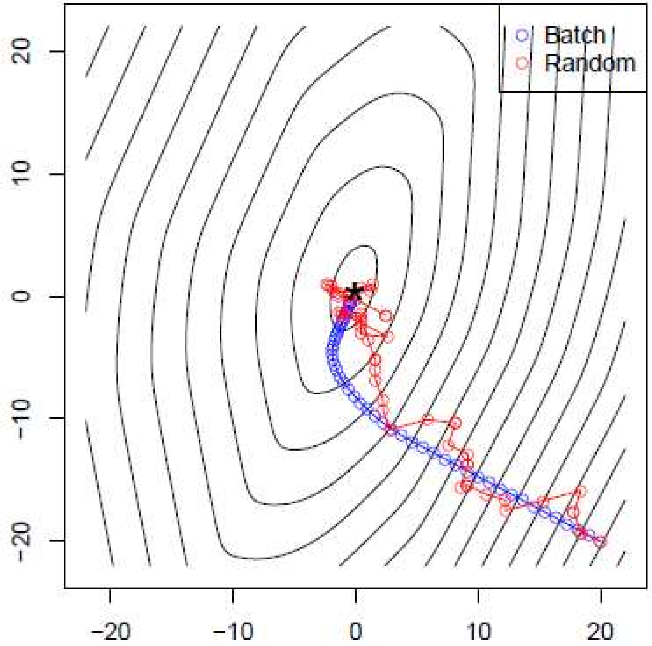
Stochastic Gradient Descent

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0,1\}$, $i = 1, \ldots, n$, recall logistic regression:

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left(-y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta)) \right)}_{f_i(\beta)}$$

Standard in SGD is to use diminishing step sizes, e.g., $t_k = 1/k$

Small example with $n = 10$, $p = 2$ to show the "classic picture" for batch versus stochastic methods:



Blue: batch steps, $O(np)$
Red: stochastic steps, $O(p)$

Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

Mini-Batches Stochastic Gradient Descent

- **We choose a random subset** $I_k \subseteq \{1, \ldots, m\}$, $|I_k| = b \ll m$, **to repeat**

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

- **Example) Consider the problem**

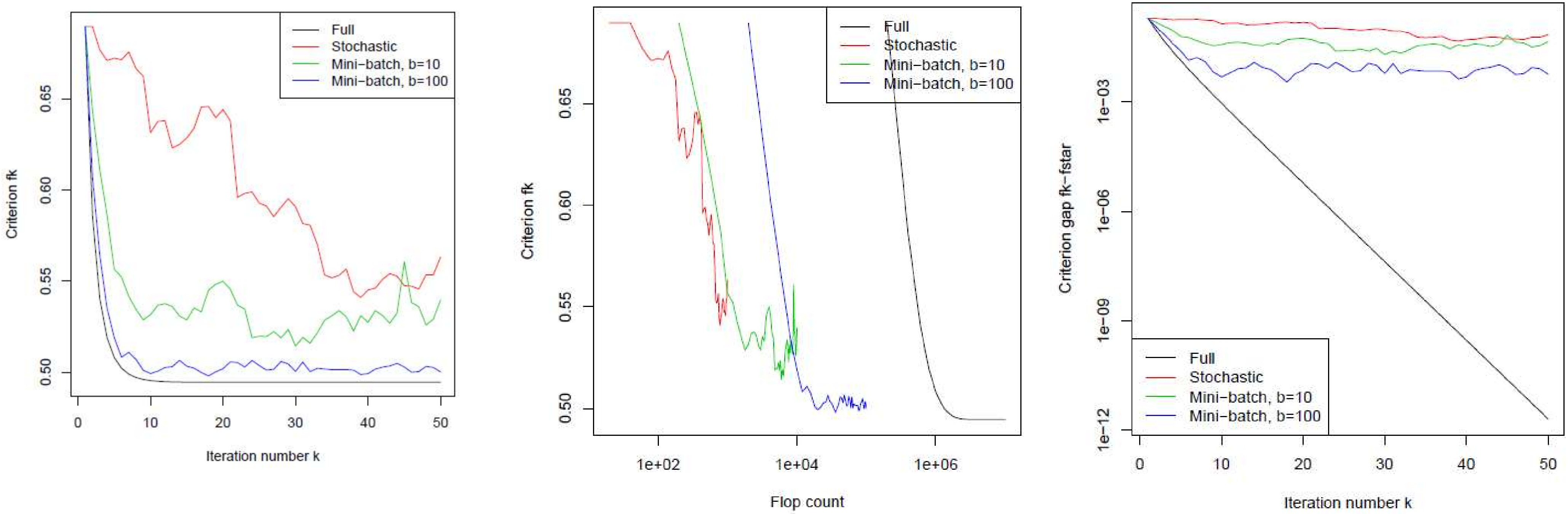
$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \left(-y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \right) + \frac{\lambda}{2} \|\beta\|_2^2$$

Full gradient computation is $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - p_i(\beta)\big) x_i + \lambda \beta$.
Comparison between methods:

- One batch update costs $O(np)$
- One mini-batch update costs $O(bp)$
- One stochastic update costs $O(p)$

Comparison of Gradient Descent Methods

Example with $n = 10,000$, $p = 20$, all methods use fixed step sizes:



Read More About

- **Constraint optimization and KKT conditions**
- **Numerical optimization methods**
  - Genetic algorithm
  - Simulated annealing
  - Newton-Raphson method