Math 4570 Matrix Methods for DA and ML

Section 9. Gradient Descent

1. Gradient Decent
2. Stochastic Gradient Decent
3. Newton's Method
4. More descent methods

Instructor: He Wang
Department of Mathematics
Northeastern University

Taylor Expansion

- Taylor Expansion of $f: \mathbb{R} \to \mathbb{R}$

$$f(a+s) = f(a) + sf'(a) + \frac{1}{2!}s^2 f''(a) + \frac{1}{3!}s^3 f'''(a) + \cdots$$

- Taylor Expansion of $f: \mathbb{R}^d \to \mathbb{R}$

$$f(\vec{a}+\vec{s}) = f(\vec{a}) + \vec{s}^T \nabla f(\vec{a}) + \frac{1}{2!}\vec{s}^T H\big(f(\vec{a})\big)\vec{s} + \cdots$$

$$= f(\vec{a}) + \sum s_i \frac{\partial f}{\partial x_i} + \sum \frac{\partial^2 f}{\partial x_i x_j} s_i s_j + \cdots$$

- Taylor Expansion of $F: \mathbb{R}^d \to \mathbb{R}^m$

$$F(\vec{a}+\vec{s}) = F(\vec{a}) + \left(\frac{\partial F(\vec{a})}{\partial \vec{x}}\right)^T \vec{s}^T + \frac{1}{2!}\begin{bmatrix} \vec{s}^T H\big(F_1(\vec{a})\big)\vec{s} \\ \vdots \\ \vec{s}^T H\big(F_m(\vec{a})\big)\vec{s} \end{bmatrix} + \cdots$$

Gradient Descent

Goal: find the local/global minimum of the cost function $J(\vec{\theta}\,)$.

Examples:

$$J(\vec{\theta}\,) = RSS(\vec{\theta}) = \| X\vec{\theta} - \vec{y}\|^2$$

$$J^{Ridge}(\vec{\theta}) = RSS(\vec{\theta}) + \lambda \|\vec{\theta}\|^2$$

$$J^{Lasso}(\vec{\theta}) = RSS(\vec{\theta}) + \lambda \|\vec{\theta}\|_1^{\;2}$$

$\lambda = 0$

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

lab1

Scikit-

Statmodel

$$\vec{\theta} = (X^T X + \lambda I)^{-1} X^T \vec{y}$$

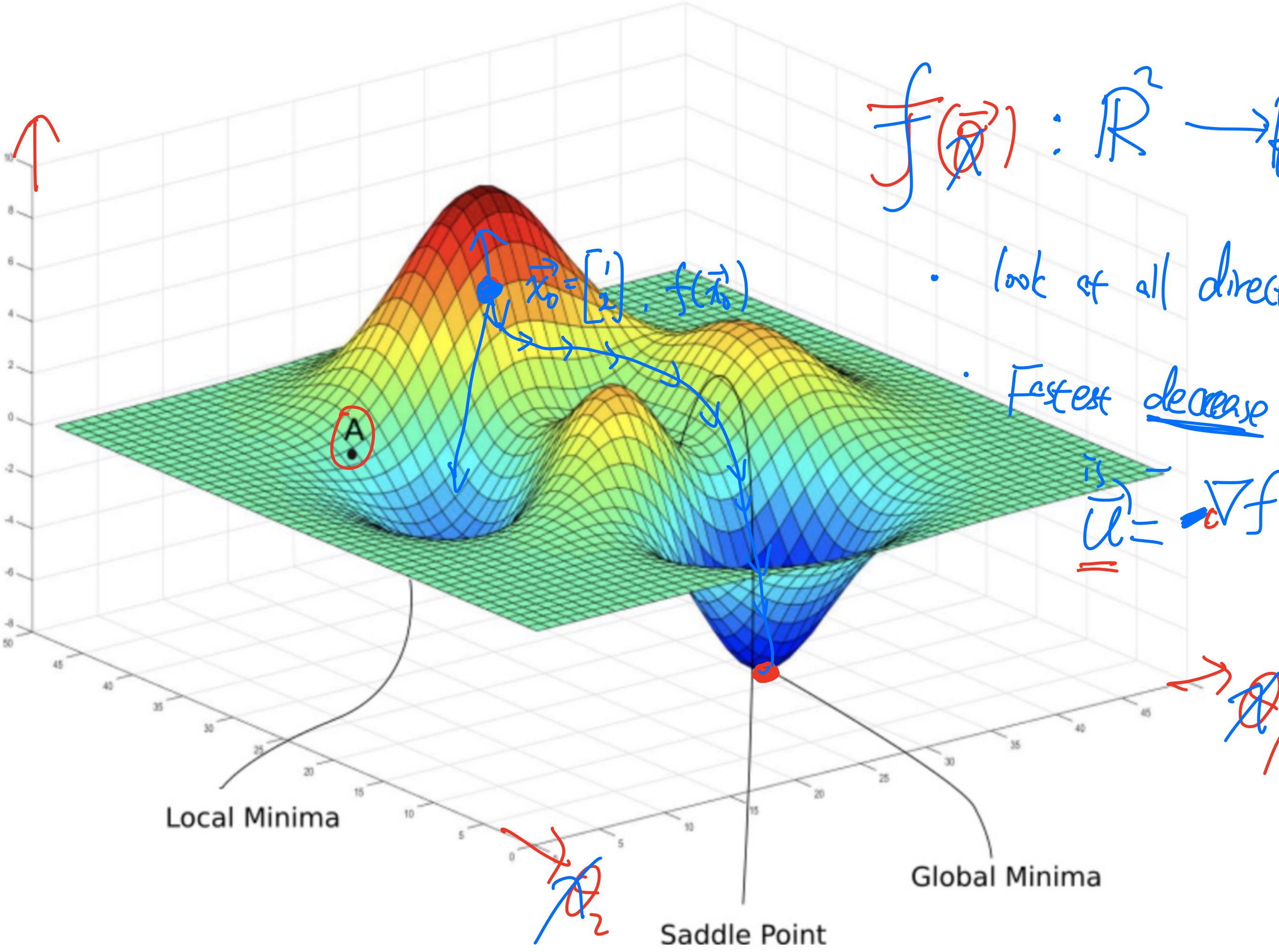
Method: find critical points by solving $\nabla J(\vec{\theta}\,) = 0$

Difficulty:

1. No closed formula or too complicated to find a closed formula for the minimum.
2. Too complicated to compute even we have a formula, as the inverse.

$\vec{f}(\vec{x}) : \mathbb{R}^2 \to \mathbb{R}^1$

- look at all directions $\vec{u} \in \mathbb{R}^2$
- Fastest decrease direction is $\vec{u} = -c\nabla f(\vec{x}_0)$



$J(\vec{\theta})$

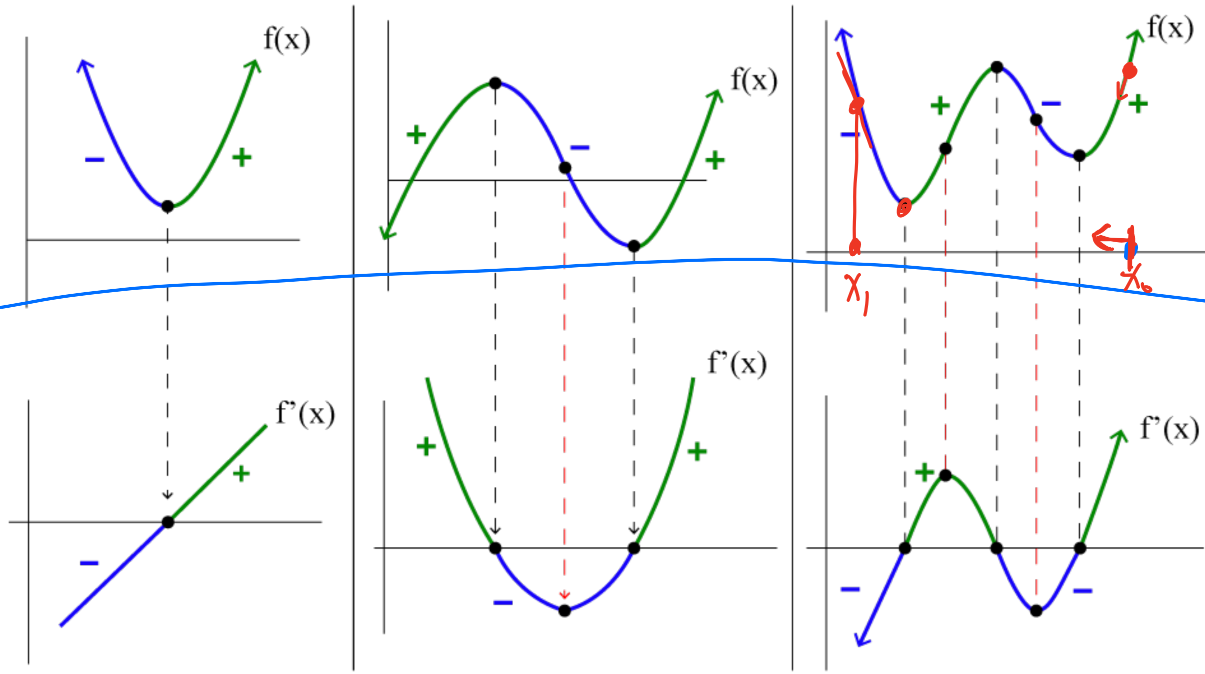
Suppose $f(\vec{x})$ is a differentiable function $\mathbb{R}^d \to \mathbb{R}$.

Question: Which **direction** has the largest rate of change?

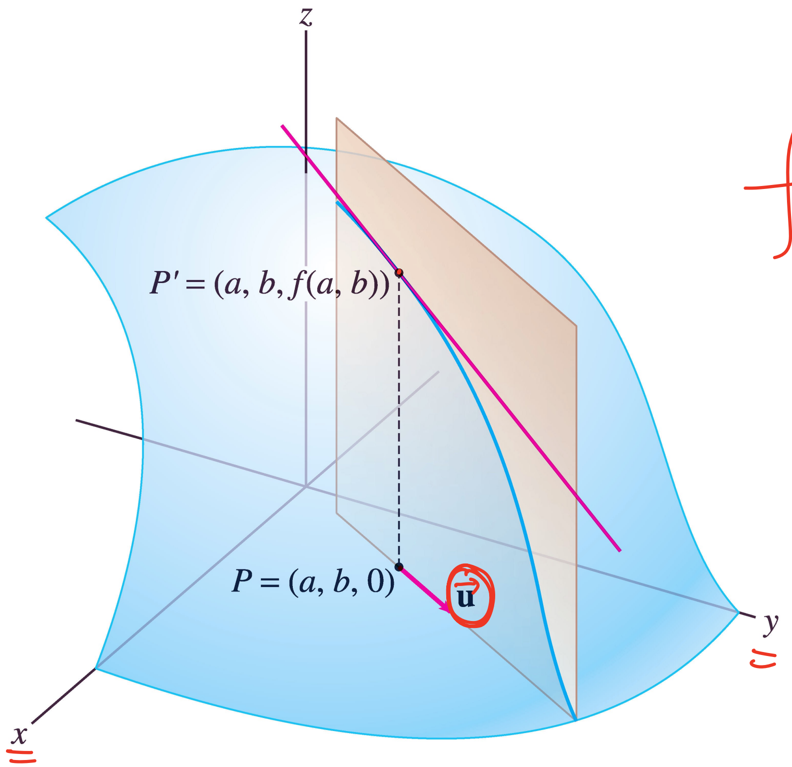
$f'(x_0) > 0$ left $-$

$f'(x_1) < 0$ right $+$

$d = 1$



Directional derivative:



$f(\vec{x})$

$f(x, y)$

Definition: Let $\vec{u}$ be a unit vector in $\mathbb{R}^d$. The directional derivative of $f(\vec{x})$ at point $\vec{a} \in \mathbb{R}^d$ in direction $\vec{u}$ is

$$D_{\vec{u}} f(\vec{x}) = \lim_{t \to 0} \frac{f(\vec{a} + t\vec{u}) - f(\vec{a})}{t}$$

This is just using the Chain Rule on the composition of $f(\vec{x})$ and the path

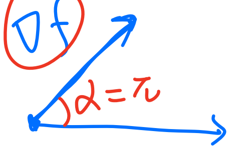
$$\frac{d\, f(\vec{a} + t\vec{u})}{dt} \qquad \vec{x}(t) = \vec{a} + t\,\vec{u}$$

Theorem: The directional derivative of $f(\vec{x})$ in direction $\vec{u}$ is computed by

$$D_{\vec{u}} f(\vec{x}) = \nabla f \cdot \vec{u}$$

$$\vec{u} = -\nabla f$$

$$\begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \partial f \end{bmatrix} = \nabla f$$

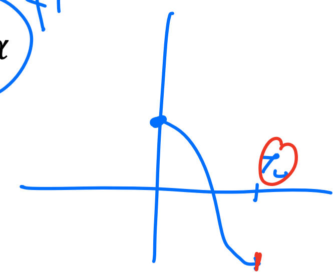


Theorem: The **maximum** value of the directional derivative $D_{\vec{u}} f(\vec{x})$ is $\|\nabla f(\vec{x})\|$ and it occurs when $\vec{u}$ has the same direction as the gradient vector $\nabla f(\vec{x})$.

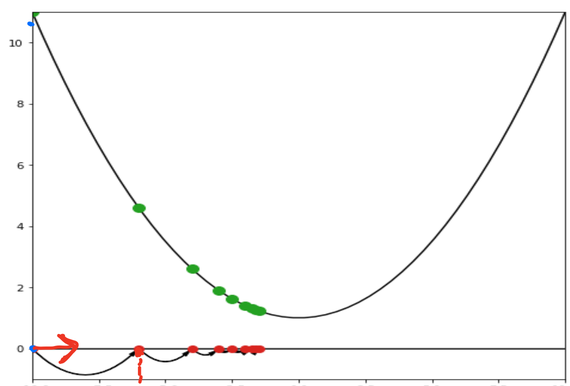
$-1 \le$ f'

$$D_{\vec{u}} f(\vec{x}) = \nabla f \cdot \vec{u} = \|\nabla f(\vec{x})\|\|\vec{u}\| \cos\alpha = \|\nabla f(\vec{x})\| \cos\alpha$$

$$D_{\vec{u}} f(\vec{x}) = \begin{cases} \|\nabla f(\vec{x})\| & when\ \alpha = 0 \\ -\|\nabla f(\vec{x})\| & when\ \alpha = \pi \end{cases}$$



The **absolute minimum** value of the directional derivative $D_{\vec{u}} f(\vec{x})$ occurs when $\vec{u}$ has the same direction $-\nabla f(\vec{x})$.



$f(x) = 3x^2 + 1$

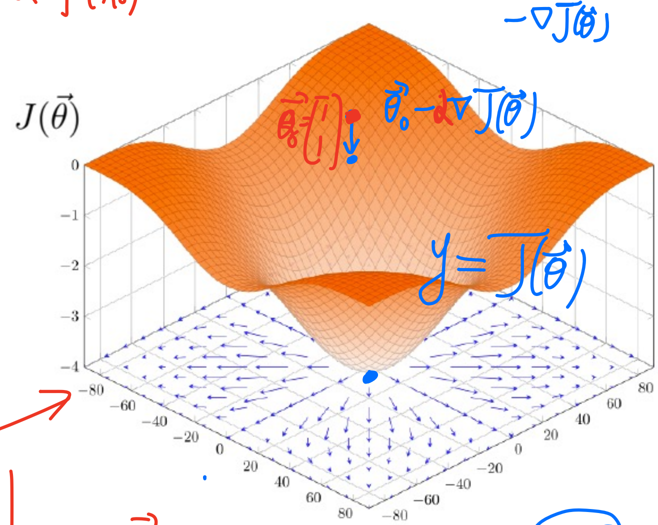
$f'(x) = 6x \qquad f'(-10) = -60$

Fastest decrease is $-(-60)$

$x_0 = -10 \quad x_1 = x_0 + \alpha(-f'(x_0)) = x_0 - \alpha f'(x_0)$

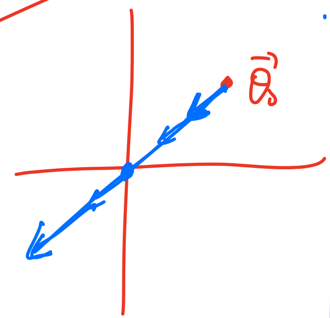
learning rate.

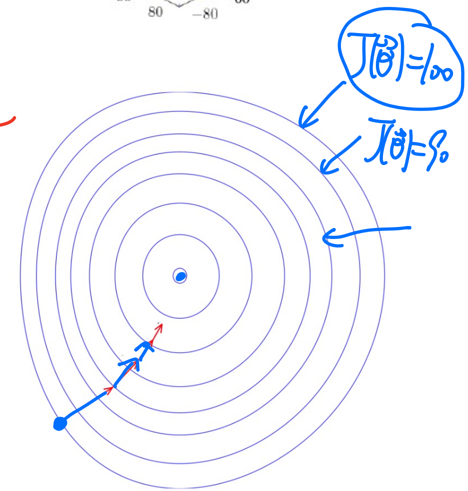
Example: $f(\theta) = \theta^2$



Example: $f(\vec{\theta}) = \theta_1^2 + \theta_2^2$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ 2\theta_2 \end{bmatrix}$$





$$\vec{\theta}_1 = \vec{\theta}_0 - \alpha \nabla f(\vec{\theta}_0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \alpha \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

➢ Gradient Descent:

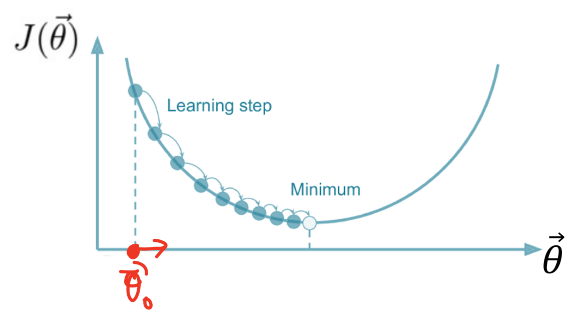
Goal: find the local/global minimum of the cost function $J(\vec{\theta}\,)$.

Gradient Descent Algorithm:

- Start with $\vec{\theta}^{\,ini} =$ some initial value. $= \vec{\theta}_0$
- Repeat $\vec{\theta}^{\,\text{next}} = \vec{\theta}^{\,current} - \alpha\, \nabla J(\vec{\theta}^{\,current})$ until converge.

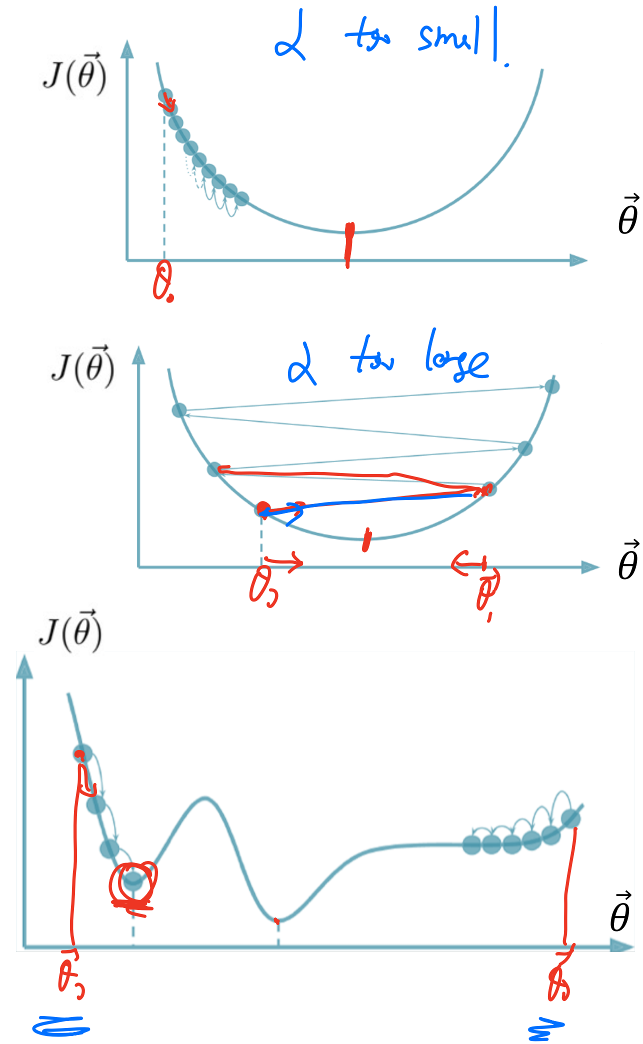
learning rate.

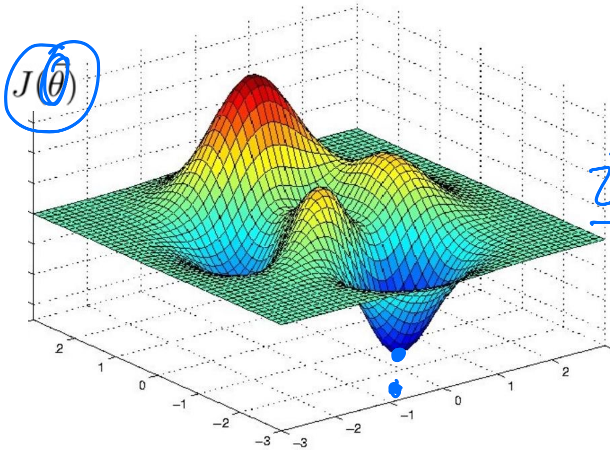
$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}^{next} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} - \alpha \begin{bmatrix} \dfrac{\partial J(\vec{\theta})}{\partial \theta_0} \\ \vdots \\ \dfrac{\partial J(\vec{\theta})}{\partial \theta_d} \end{bmatrix}$$



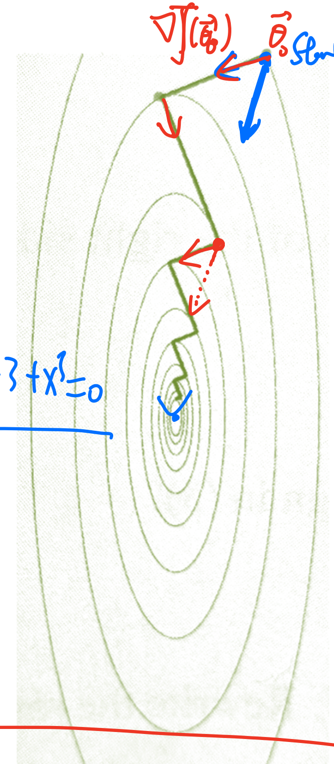
Key points:

- Compute $\nabla J(\vec{\theta})$
- Set initial value $\vec{\theta} = \vec{\theta}_0$
- Set a good learning rate α
    - Set different α and recording the cost
    - Start from large α_0, then smaller α.
    - Set $\alpha_k = \frac{1}{\sqrt{k}}\alpha_0$ or $\alpha_k = \frac{1}{k}\alpha_0$
    - ...





$z^x + e^{x^2} + \sin x + \log x + 3 + x^3 = 0$



- Data $(\vec{x}^{(i)}, y^{(i)}) \rightsquigarrow (X, \vec{y})$
- Model $h_\theta(\vec{x})$
- Cost function $\to \|h_\theta(X) - \vec{y}\|$ / cross-entropy $= J(\vec{\theta})$
- $\boxed{\text{Solve } \nabla J(\vec{\theta}) = \vec{0}}$
- Minimize $J(\vec{\theta})$

Minimize $J(\vec{\theta})$ $\Longleftrightarrow$ Find $\vec{\theta} = \arg\min_{\vec{\theta}} J(\vec{\theta})$ gradient descent.

- Example: (linear regression) $h(\vec{x}) = \vec{\theta}^T\vec{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$

$$J(\vec{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(h(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{n} RSS(\vec{\theta}) = \frac{1}{n}\| h(X) - \vec{y} \|^2$$

$$= \frac{1}{n}(X\vec{\theta} - \vec{y})^T(X\vec{\theta} - \vec{y})$$

For each $j = 0,1,\ldots,d$

$$\frac{\partial}{\partial\theta_j}J(\vec{\theta}) = \frac{\partial}{\partial\theta_j}\left(\frac{1}{n}\sum_{i=1}^{n}\left(h(x^{(i)}) - y^{(i)}\right)^2\right)$$

$$= \frac{1}{n}\left(\sum_{i=1}^{n}\frac{\partial}{\partial\theta_j}\left(h(x^{(i)}) - y^{(i)}\right)^2\right)$$

$$= \frac{1}{n}\cdot\sum_{i=1}^{n}\left(2(h(x^{(i)}) - y^{(i)})\cdot\frac{\partial}{\partial\theta_j}(h(x^{(i)}) - y^{(i)})\right)$$

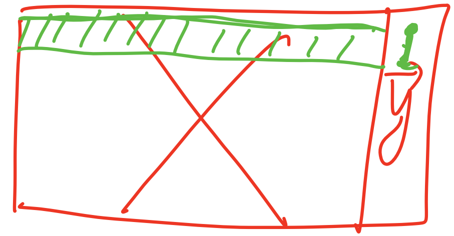
$$= \frac{2}{n}\sum_{i=1}^{n}\left((h(x^{(i)}) - y^{(i)})\cdot\frac{\partial}{\partial\theta_j}\left(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \ldots + \theta_d x_d^{(i)} - y^{(i)}\right)\right)$$

$$= \frac{2}{n}\sum_{i=1}^{n}(h(x^{(i)}) - y^{(i)})\cdot x_j^{(i)}$$

Repeat until converge

$$\theta_j := \theta_j - \alpha\cdot\left(\frac{2}{n}\sum_{i=1}^{n}(h(x^{(i)}) - y^{(i)})\cdot x_j^{(i)}\right)$$

- Example: (linear regression, vector notation)



$$h(\vec{x}) = \vec{\theta}^T \vec{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

$$J(\vec{\theta}) = \frac{1}{n} RSS(\vec{\theta}) := \frac{1}{\mathrm{n}} \left\| \boldsymbol{X}\vec{\theta} - \vec{y} \right\|^2 = \frac{1}{n}\left(\vec{\theta}^T \boldsymbol{X}^{\boldsymbol{T}} \boldsymbol{X}\vec{\theta} - 2\vec{y}^T \boldsymbol{X}\vec{\theta} + \vec{y}^T \vec{y}\right)$$

$$\nabla_{\vec{\theta}} J = \frac{2}{n}\left(X^T X \vec{\theta} - X^T \vec{y}\right)$$

data.

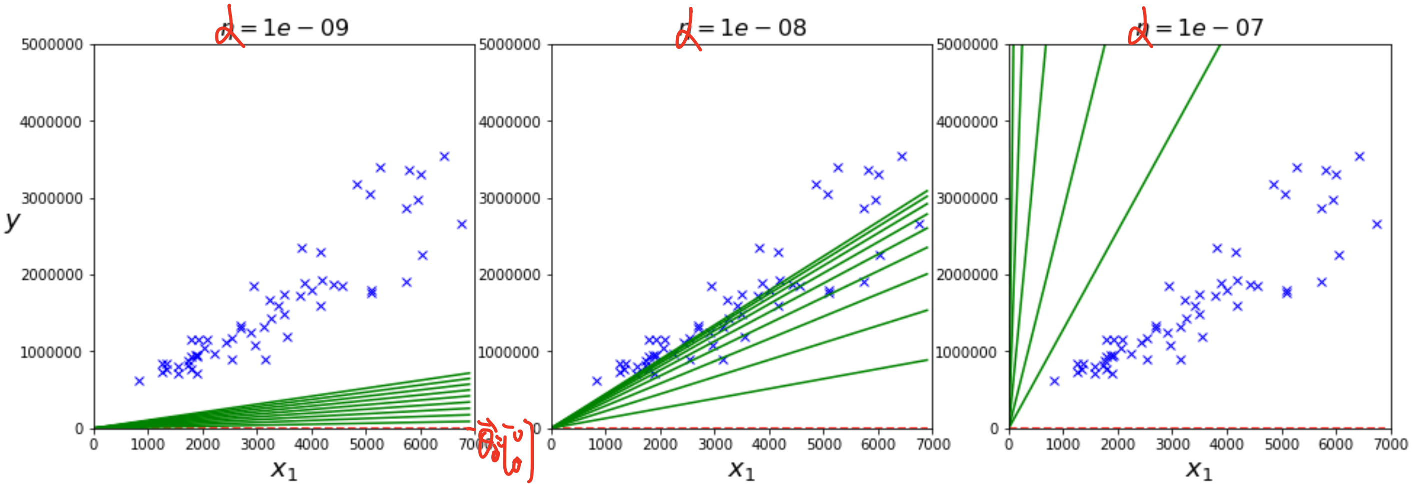
Gradient descent formula: $\vec{\theta}^{\text{next}} = \vec{\theta} - \alpha \boxed{\frac{2}{n} X^T (X\vec{\theta} - \vec{y})}$

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

Python (broadcast): $\vec{\theta}^{\text{next}} = \vec{\theta} - \alpha \frac{2}{n} \operatorname{sum}\left[(X\vec{\theta} - \vec{y}) * X\right]$

Golden Rule: If you can use vector, never use a for loop.

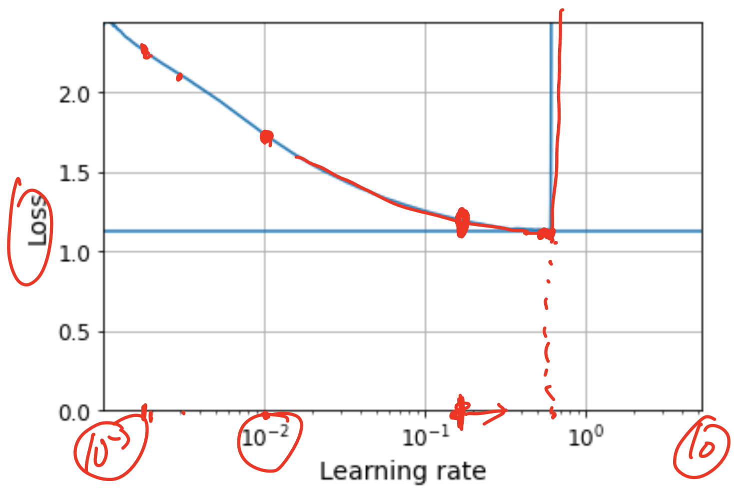
We ran the update rule for all the training examples $(X, \vec{y})$ at once, which is called **(batch) gradient descent.**



$f(x_1) = \theta_0 + \theta_1 x_1$

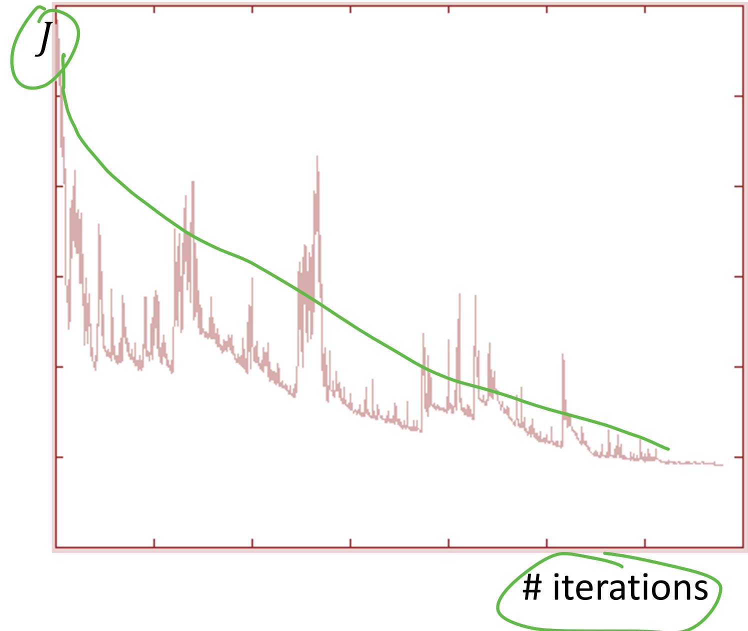
Find a good learning rate:

For different learning rate
Use a small data set
Repeat 100 times



$\cdot\ \vec{\theta}^{next} = \vec{\theta} - \alpha \nabla J$ ∇J $\nabla J = \frac{1}{n}\left(X^T X - X^T \vec{y}\right)$

➢ Stochastic Gradient Descent (SGD):



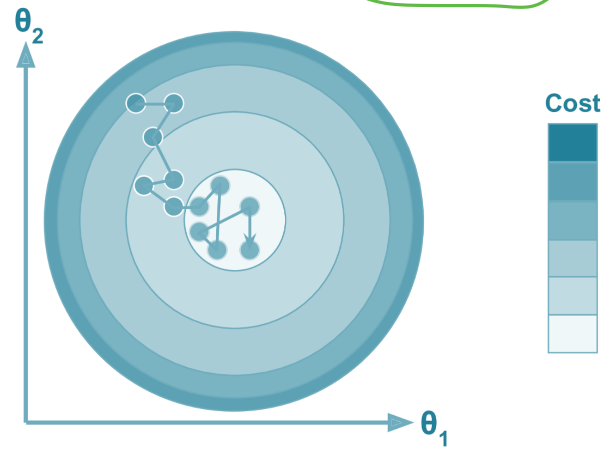
For each step, we use only one data point $(\vec{x}^{(i)}, y^{(i)})$ to find descent direction.

- $\vec{\theta}^{\text{next}} = \vec{\theta} - \alpha \nabla J(\vec{\theta}; \vec{x}^{(i)}, y^{(i)})$

∇J

For example, in linear regression,

$$\vec{\theta}^{\text{next}} = \vec{\theta} - \alpha \vec{x}^{(i)}(\vec{x}^{(i)^{T}}\vec{\theta} - y^{(i)})$$

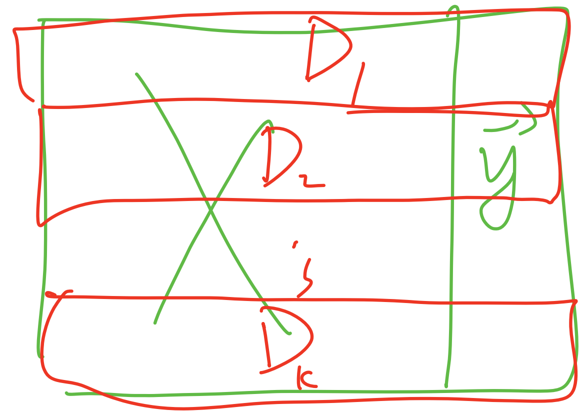


Remark:
1. Randomly with replacement, or use a random order on the data.
2. It is fast.
3. It may achieve global minimum.
4. We call an epoch for repeating a data set

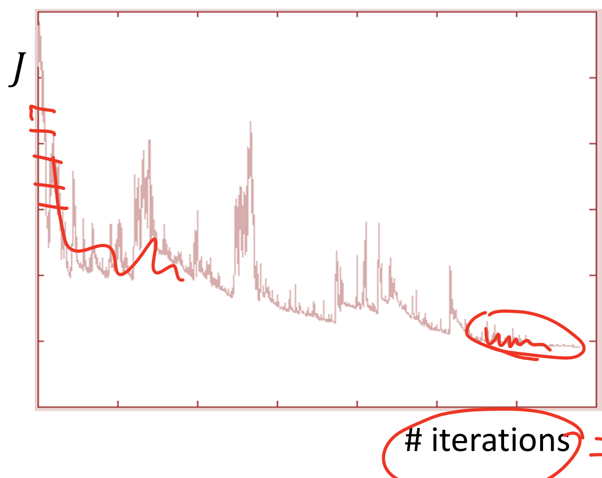
- Mini-batch Gradient Descent:

For each step, we use only a subset of data points $D_j \subset D$ to find descent direction $\nabla J(\vec{\theta}; D_j)$.

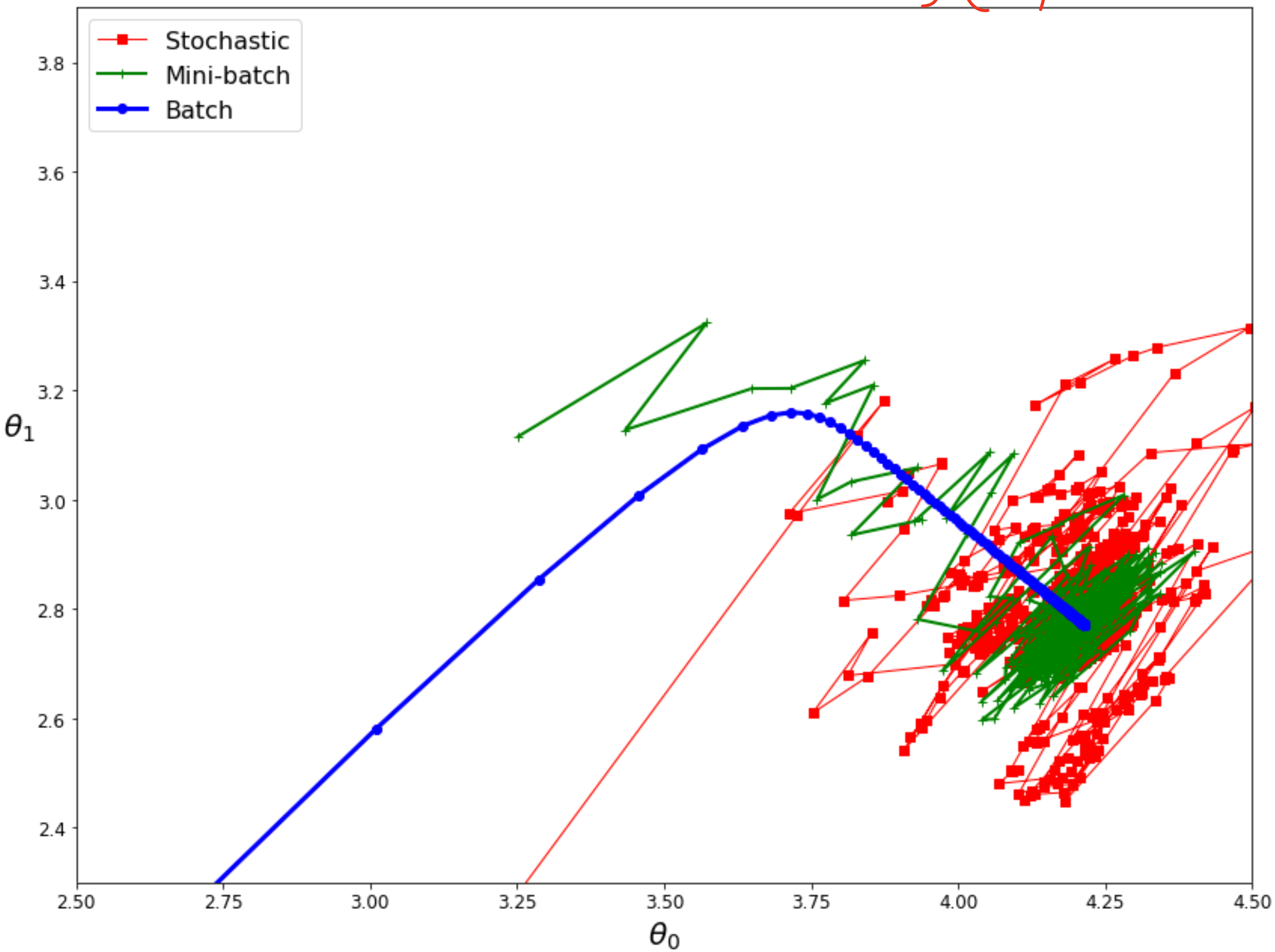
- $\vec{\theta}^{\text{next}} = \vec{\theta} - \alpha \, \nabla J(\vec{\theta}; D_j)$



If each minibatch D_j contains one point, it is Stochastic Gradient Descent.
If each minibatch D_j contains all points, it is batch Gradient Descent.



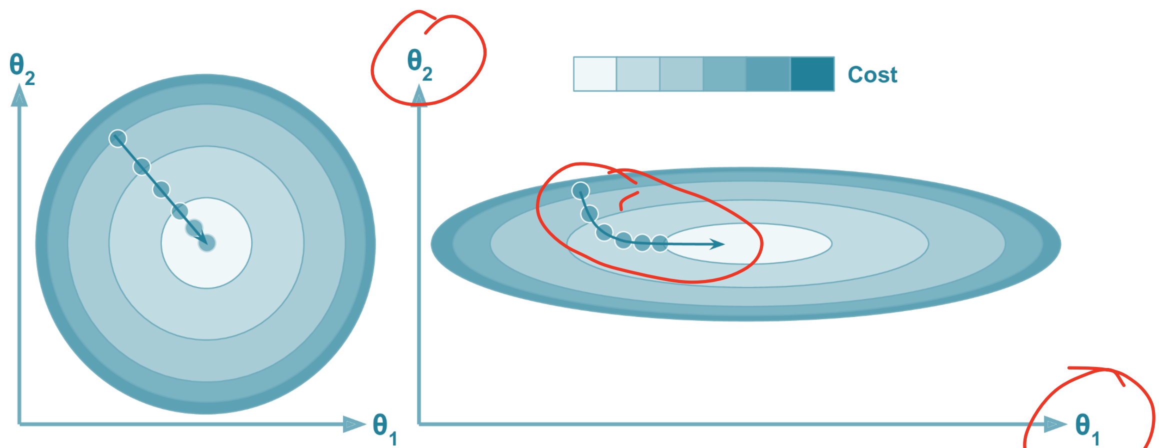
$J(\vec{\theta}) = k$


Stochastic
Mini-batch
Batch
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
θ_1
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.25
4.50
θ_0

Remarks:

1. Normal equation
2. Stochastic gradient descent
3. Batch gradient descent
4. Mini batch gradient descent

Scale the features first: normalization or standardization



· Our problem: Solve $\nabla J(\vec{\theta}) = 0.$

Newton' method

Find **root** of a function $f: \mathbb{R} \to \mathbb{R}$.

Solve $f(x) = 0$

Newton' method Algorithm

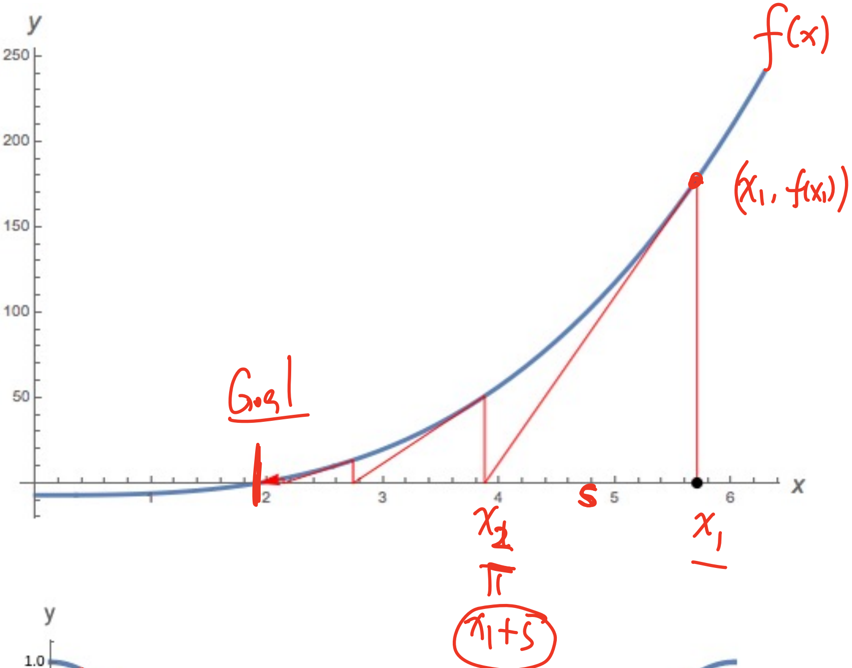
1. Make a guess x_0
2. Repeat

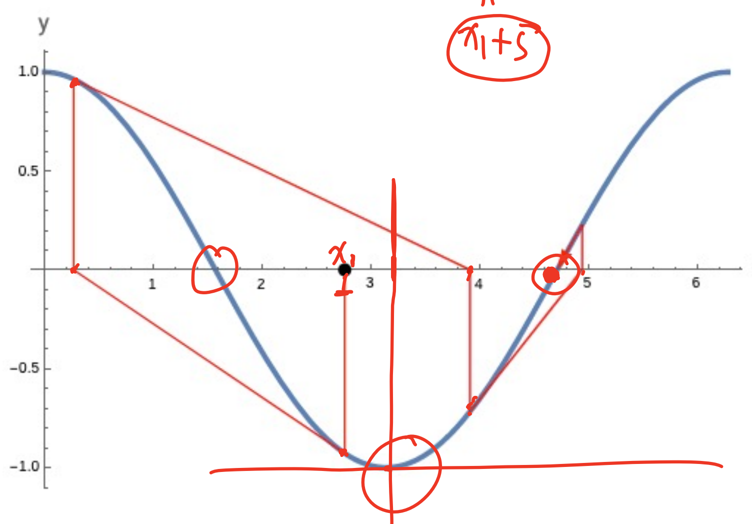
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Reason: $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$

$$f(x_1 + s) \approx f(x_1) + sf'(x_1) = 0$$

$$s = -\frac{f(x_k)}{f'(x_k)}$$





High dimension Newton's method for $F: \mathbb{R}^m \to \mathbb{R}^m$

$\nabla J = 0$

$F = \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix}$

Repeat $\vec{x}_{k+1} = \vec{x}_k - B^{-1} F(\vec{x}_k)$

$$\text{where, } B = \left(\frac{\partial F(\vec{x}_k)}{\partial \vec{x}}\right)^T = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_m} \end{bmatrix}$$

Application of Newton's method to

Goal: find the local/global minimum of the cost function $J(\vec{\theta}\,)$.

$$Find\ \nabla J(\vec{\theta}\,) = 0$$

$$\vec{\theta}^{\,next} = \vec{\theta} - \underline{B}^{-1} F$$

Let $F(\vec{\theta}) = \nabla J(\vec{\theta}\,) = \begin{bmatrix} \frac{\partial J(\vec{\theta})}{\partial \theta_0} \\ \vdots \\ \frac{\partial J(\vec{\theta})}{\partial \theta_d} \end{bmatrix}$ and apply Newton's method.

$$\vec{\theta}_{k+1} = \vec{\theta}_k - H^{-1} \nabla J(\vec{\theta}_k\,) \rightsquigarrow \arg\min J(\vec{\theta})$$

Here H is the Hessian matrix $H = \begin{bmatrix} \frac{\partial^2 J}{\partial \theta_1{}^2} & \cdots & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \theta_d \partial \theta_1} & \cdots & \frac{\partial^2 J}{\partial \theta_d{}^2} \end{bmatrix}$

G.D. $\vec{\theta}_{k+1} = \vec{\theta}_k - \alpha \nabla J(\vec{\theta}\,)$

Example. Linear Regression. $J(\vec{\theta}) = (X\vec{\theta}-\vec{y})^T(X\vec{\theta}-\vec{y})$

$X_{n\times d}$

$\text{rank}(X)=d$

$$\nabla J = 2X^TX\vec{\theta} - 2X^T\vec{y}$$

$$HJ(\vec{\theta}) = 2X^TX$$

$$\vec{\theta}^{\,next} = \underset{\text{current}}{\vec{\theta}} - H^{-1}\nabla J = \underset{\text{cur}}{\vec{\theta}} - (2X^TX)^{-1}(2X^TX\vec{\theta}_{cur} - 2X^T\vec{y})$$

$$= \cancel{\vec{\theta} - \vec{\theta}} + (X^TX)^{-1}X^T\vec{y}$$

$$= (X^TX)^{-1}X^T\vec{y}$$

Remark: Newton's method is faster, since it depends on the second derivative. However, sometimes it is hard to calculate or it is not invertible.

J

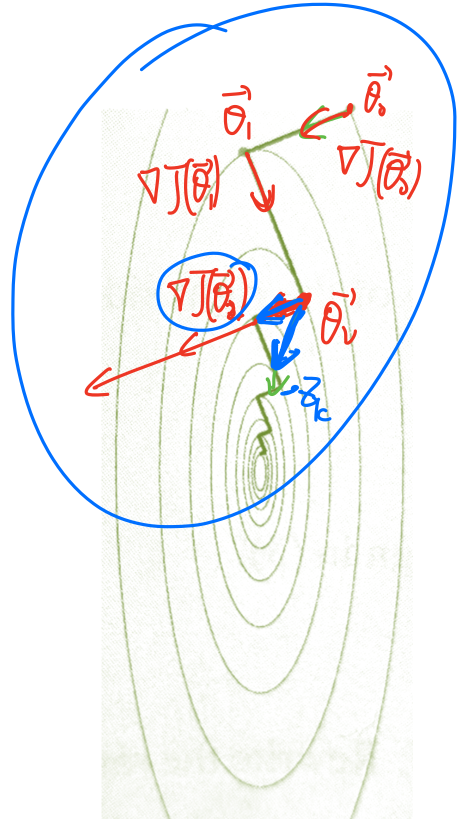
More gradient methods:

Recall GD: $\vec{\theta}^{\text{next}} = \vec{\theta} - \alpha \nabla J(\vec{\theta})$

1. Descent with momentum(memory)

$$\vec{\theta}_{k+1} = \vec{\theta}_k - \alpha \mathrm{Z_k}$$

Here $Z_k = \boxed{\nabla J(\vec{\theta}_k)} + \beta Z_{k-1}$



2. Adaptive Stochastic Gradient Descent

Recall SGD: $\vec{\theta}^{\text{next}} = \vec{\theta} - \alpha\, \nabla J(\vec{\theta}; \vec{x}^{(i)}, y^{(i)})$

Adaptive:

$$\vec{\theta}_{k+1} = \vec{\theta}_k - \alpha_k \mathrm{D_k}$$

$$\text{Here } \alpha_k = \alpha(\nabla J_k, \nabla J_{k-1}, \dots, \nabla J_0)$$

$$D_k = D(\nabla J_k, \nabla J_{k-1}, \dots, \nabla J_0)$$

For example, **ADAGRAD (2011)**

$$\alpha_k = \frac{\alpha}{\sqrt{k}} \left(\frac{1}{k}\, diag \sum_{i=1}^{k} \|\nabla J_i\|^2 \right)^{\frac{1}{2}} \quad \text{and} \quad D_k = \nabla J(\vec{\theta}_k)$$

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12:2121–2159, 2011.

ADAM (2015)

Recursive formula:
$$D_k = \delta D_{k-1} + (1-\delta)\nabla J(\vec{\theta}_k)$$
$$\alpha_k^2 = \beta \alpha_{k-1}^2 + (1-\beta)\left\|\nabla J\ (\overrightarrow{\theta_i})\right\|^2$$

More explicitly,

$$D_k = (1-\delta)\sum_{i=1}^{k} \delta^{k-i} \nabla J(\vec{\theta}_k)$$

$$\alpha_k = \frac{\alpha}{\sqrt{k}}\left((1-\beta)\ diag \sum_{i=1}^{k} \beta^{k-i} \left\|\nabla J\ (\overrightarrow{\theta_i})\right\|^2\right)^{\frac{1}{2}}$$

Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, pages 1–13, 2015.

An overview of gradient descent optimization algorithms

https://arxiv.org/abs/1609.04747