Gradient Descent

顏安孜

azyen@nycu.edu.tw

The following slides are selected from the course material of Machine Learning by Prof. Hung-Yi Lee

Review: Gradient Descent

- In step 3, we have to solve the following optimization problem:

$$\theta^* = \arg\min_{\theta} L(\theta)$$

L: loss function θ: parameters

Suppose that θ has two variables $\{\theta_1, \theta_2\}$

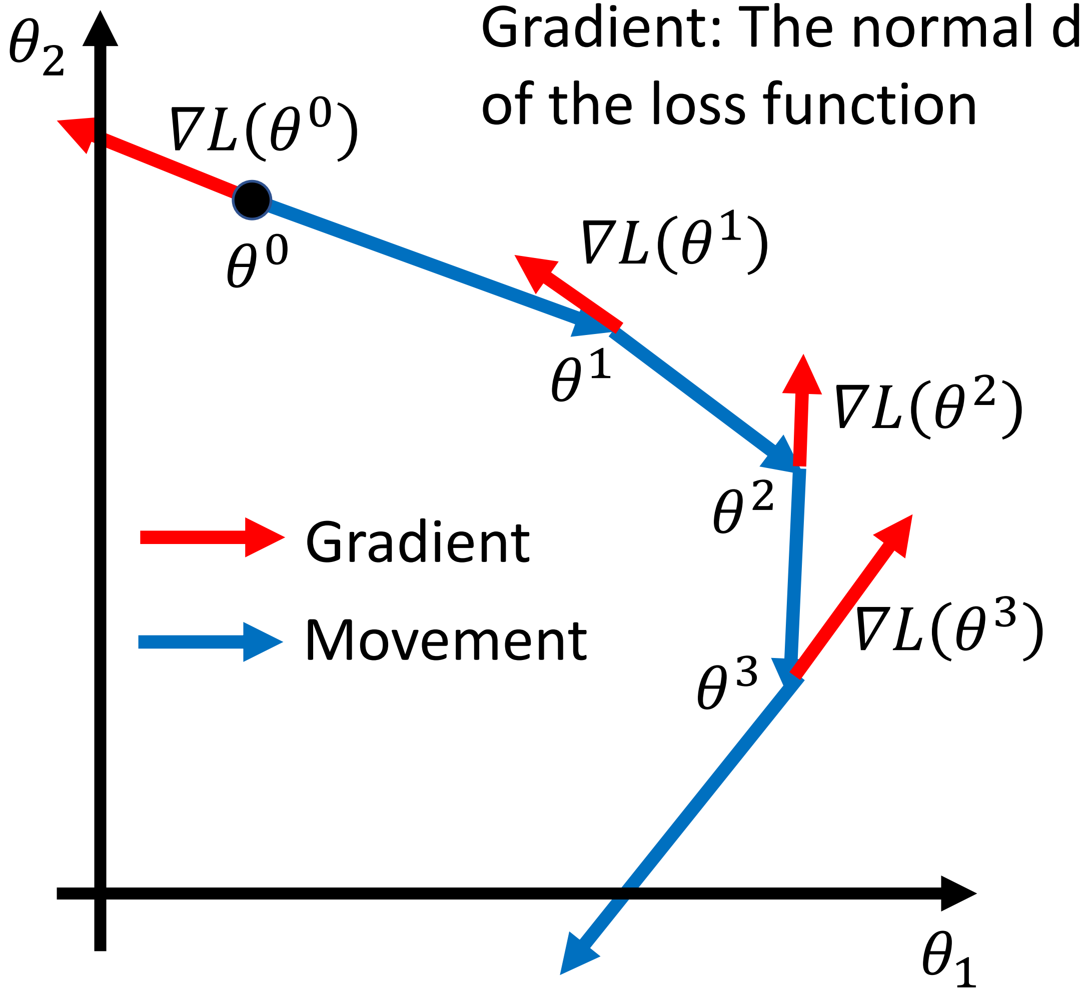
Randomly start at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta_1)/\partial \theta_1 \\ \partial L(\theta_2)/\partial \theta_2 \end{bmatrix}$$

$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^0)/\partial \theta_1 \\ \partial L(\theta_2^0)/\partial \theta_2 \end{bmatrix} \Rightarrow \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

$$\begin{bmatrix} \theta_1^2 \\ \theta_2^2 \end{bmatrix} = \begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^1)/\partial \theta_1 \\ \partial L(\theta_2^1)/\partial \theta_2 \end{bmatrix} \Rightarrow \theta^2 = \theta^1 - \eta \nabla L(\theta^1)$$

Review: Gradient Descent



Gradient: The normal direction of the contour lines of the loss function

Start at position θ^0

Compute gradient at θ^0

Move to $\theta^1 = \theta^0 - \eta\nabla L(\theta^0)$

Compute gradient at θ^1

Move to $\theta^2 = \theta^1 - \eta\nabla L(\theta^1)$

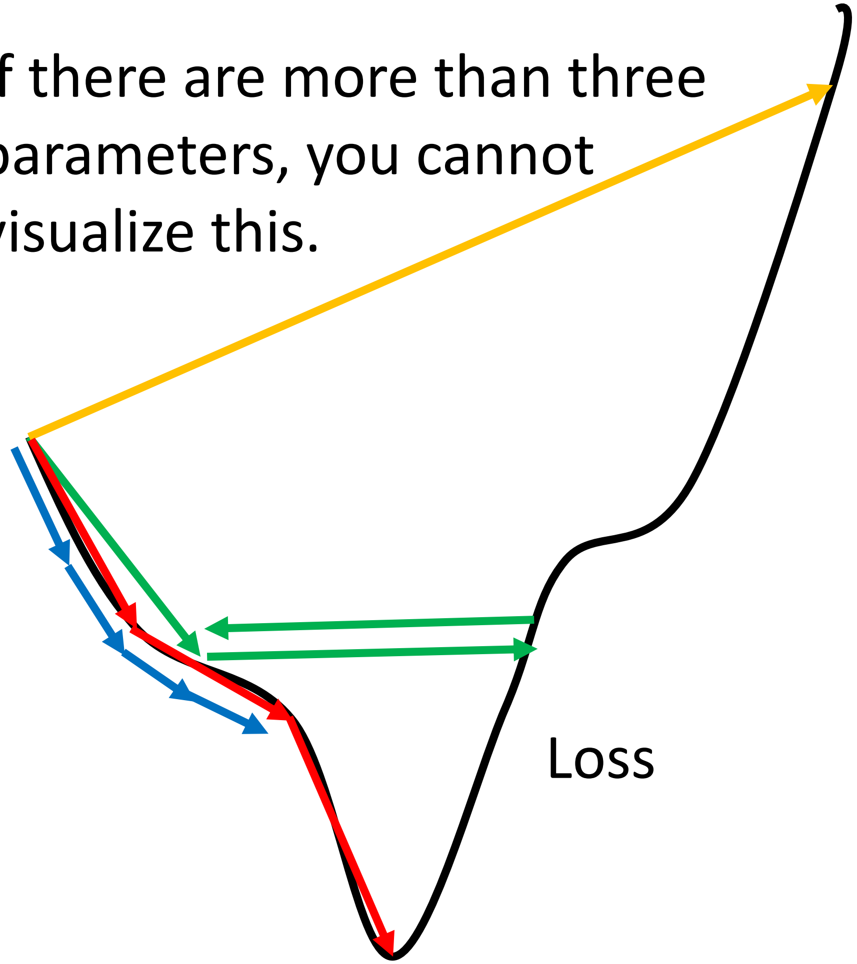
⋮

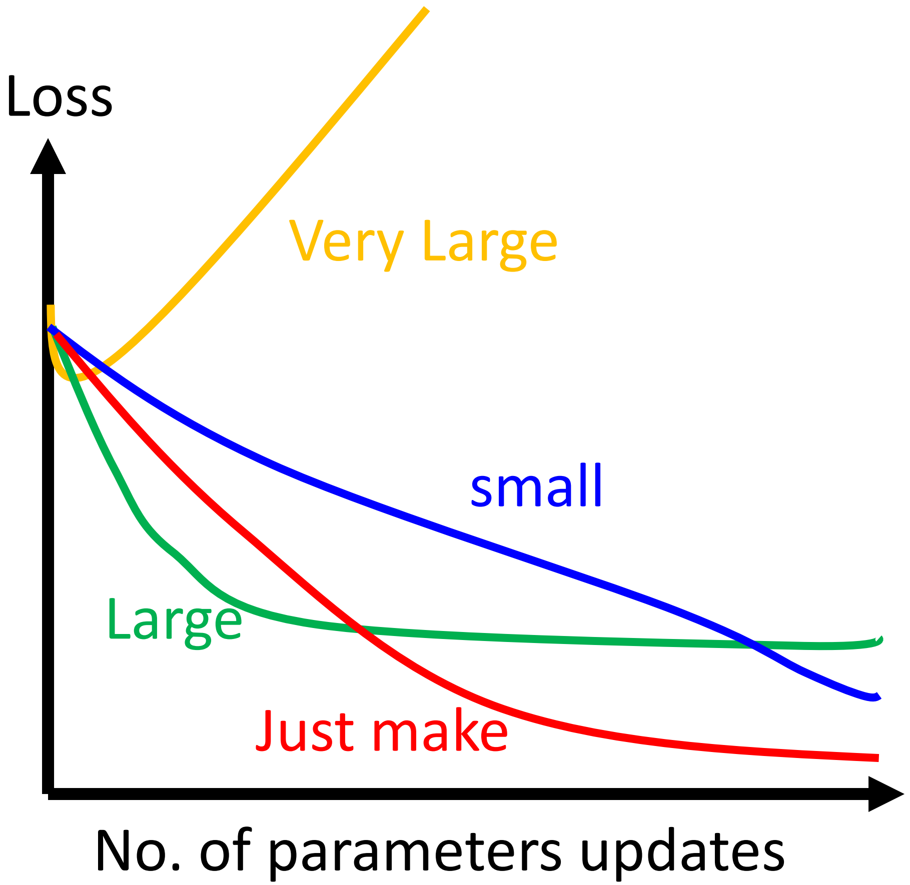
Learning Rate

$$\theta^{i} = \theta^{i-1} - \eta \nabla L\left(\theta^{i-1}\right)$$

Set the learning rate η carefully

If there are more than three parameters, you cannot visualize this.





But you can always visualize this.

Adaptive Learning Rates

- Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g. 1/t decay: $\eta^t = \eta / \sqrt{t+1}$
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates

Gradient descent optimization algorithms: https://www.ruder.io/optimizing-gradient-descent/#gradientdescentoptimizationalgorithms

Adagrad

$$\eta^t = \frac{\eta}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

w is one parameters

Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

σ^t: ***root mean square*** of the previous derivatives of parameter w

Parameter dependent

Adagrad

σ^t: ***root mean square*** of the previous derivatives of parameter w

$$w^1 \leftarrow w^0 - \frac{\eta^0}{\sigma^0} g^0 \qquad \sigma^0 = \sqrt{(g^0)^2}$$

$$w^2 \leftarrow w^1 - \frac{\eta^1}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\frac{1}{2}\left[(g^0)^2 + (g^1)^2\right]}$$

$$w^3 \leftarrow w^2 - \frac{\eta^2}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\frac{1}{3}\left[(g^0)^2 + (g^1)^2 + (g^2)^2\right]}$$

$$\vdots$$

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$

Adagrad

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

$$w^{t+1} \leftarrow w^t - \frac{\boxed{\eta^t}}{\boxed{\sigma^t}} g^t$$

$$\eta^t = \frac{\eta}{\cancel{\sqrt{t+1}}}$$

1/t decay

$$\sigma^t = \sqrt{\frac{1}{\cancel{t+1}} \sum_{i=0}^{t} (g^i)^2}$$

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t} (g^i)^2}} g^t$$

Contradiction?

$$\eta^t = \frac{\eta}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

Vanilla Gradient descent

$$w^{t+1} \leftarrow w^t - \eta^t \underline{g^t}$$

Larger gradient, larger step

Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\underline{\sum_{i=0}^{t}(g^i)^2}}} \underline{g^t}$$

Larger gradient, larger step

Larger gradient, smaller step

Intuitive Reason

$$\eta^t = \frac{\eta}{\sqrt{t+1}} \quad g^t = \frac{\partial L(\theta^t)}{\partial w}$$

- How surprise it is

Very big

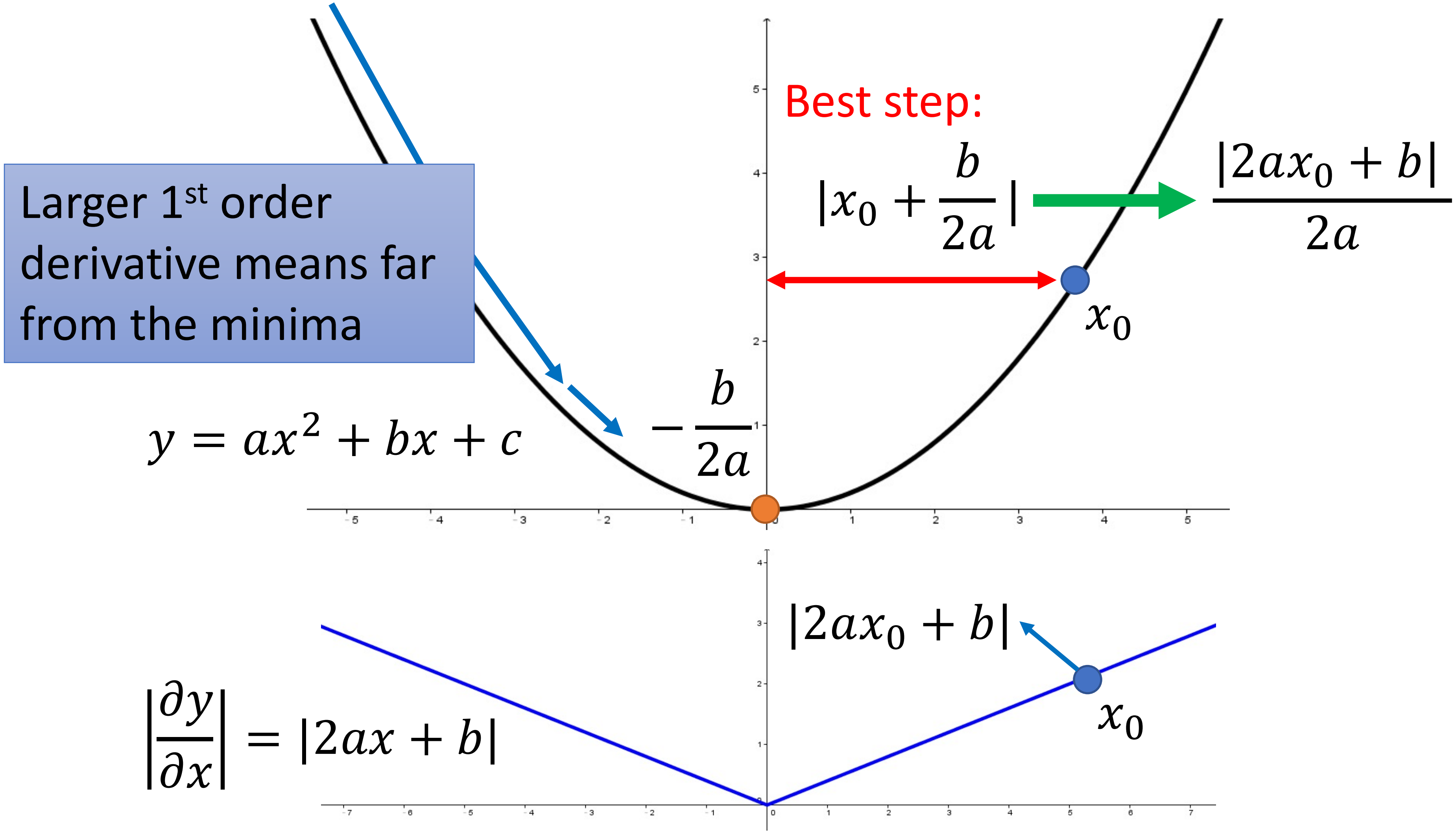
g^0	g^1	g^2	g^3	g^4	……
0.001	0.001	0.003	0.002	0.1	……

g^0	g^1	g^2	g^3	g^4	……
10.8	20.9	31.7	12.1	0.1	……

Very small

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

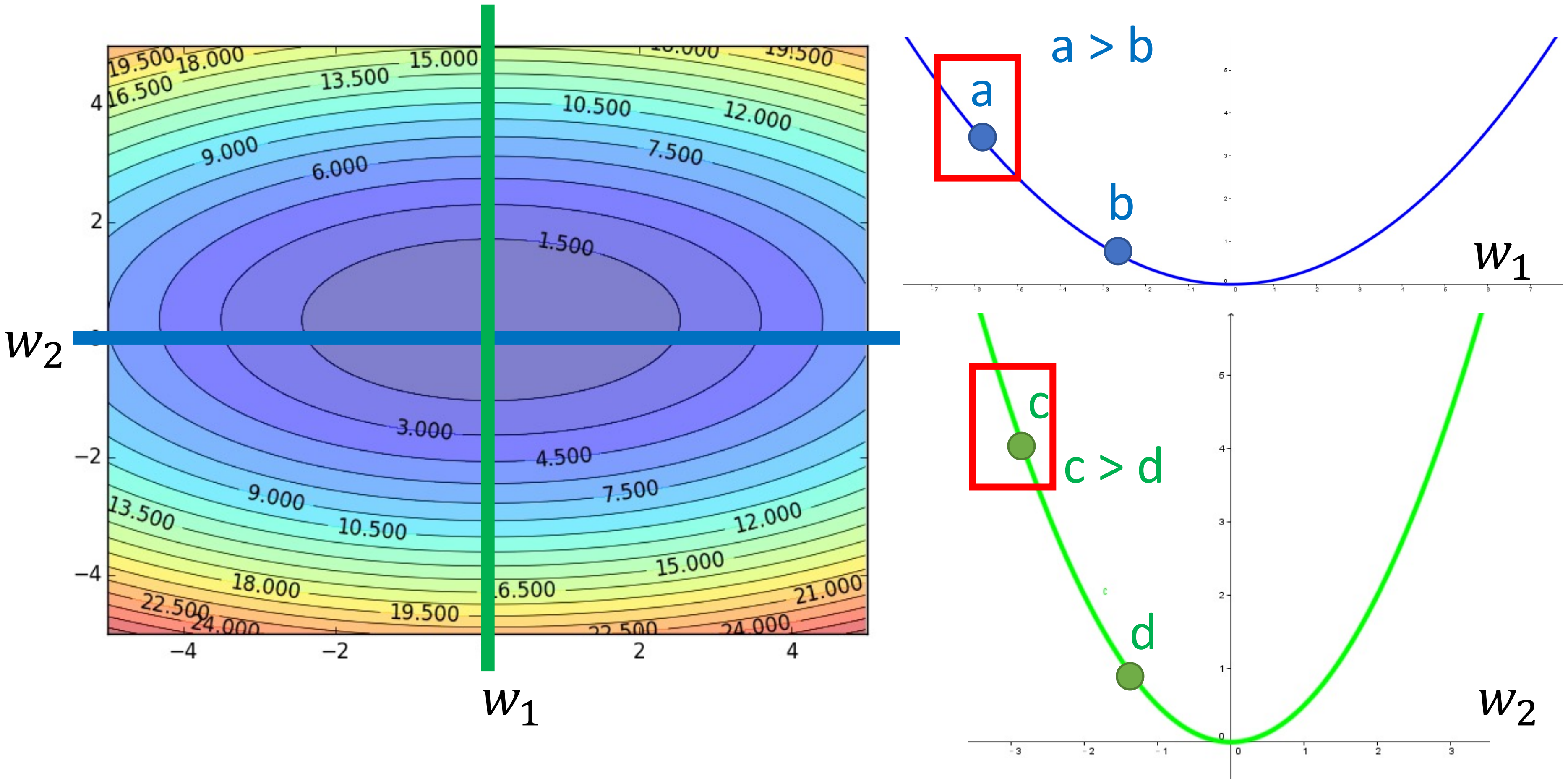
Larger gradient, larger steps?



Comparison between different parameters

~~Larger 1st order derivative means far from the minima~~

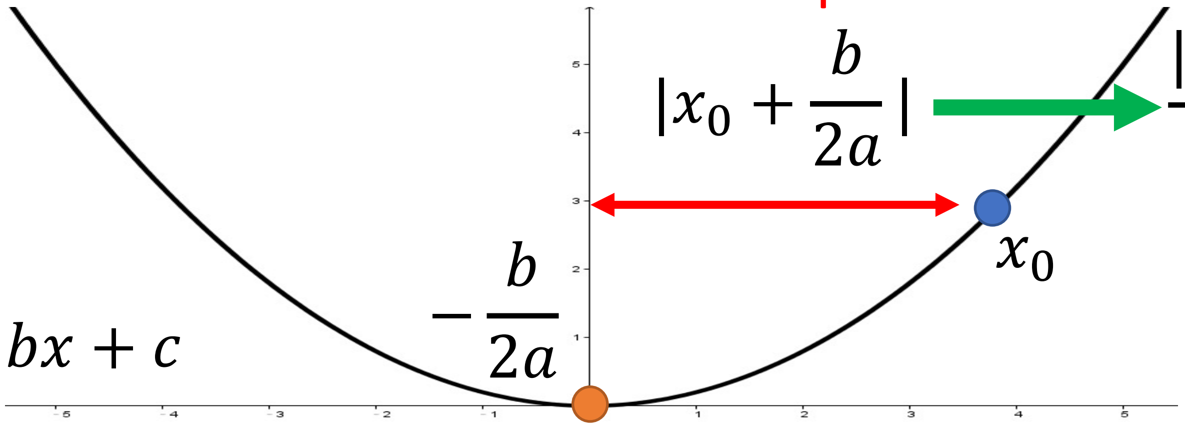
Do not cross parameters



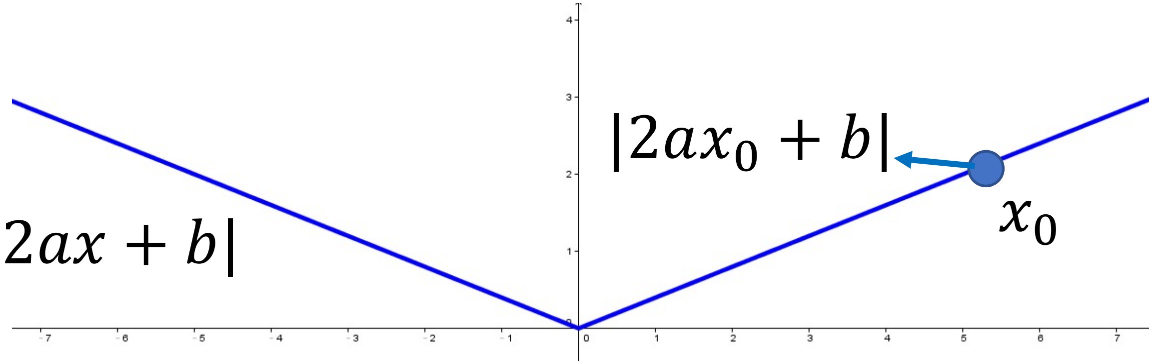
Second Derivative

Best step:

$$\left|x_0 + \frac{b}{2a}\right| \rightarrow \frac{|2ax_0 + b|}{\boxed{2a}}$$



$$y = ax^2 + bx + c$$



$$\left|\frac{\partial y}{\partial x}\right| = |2ax + b|$$

$$\frac{\partial^2 y}{\partial x^2} = 2a$$

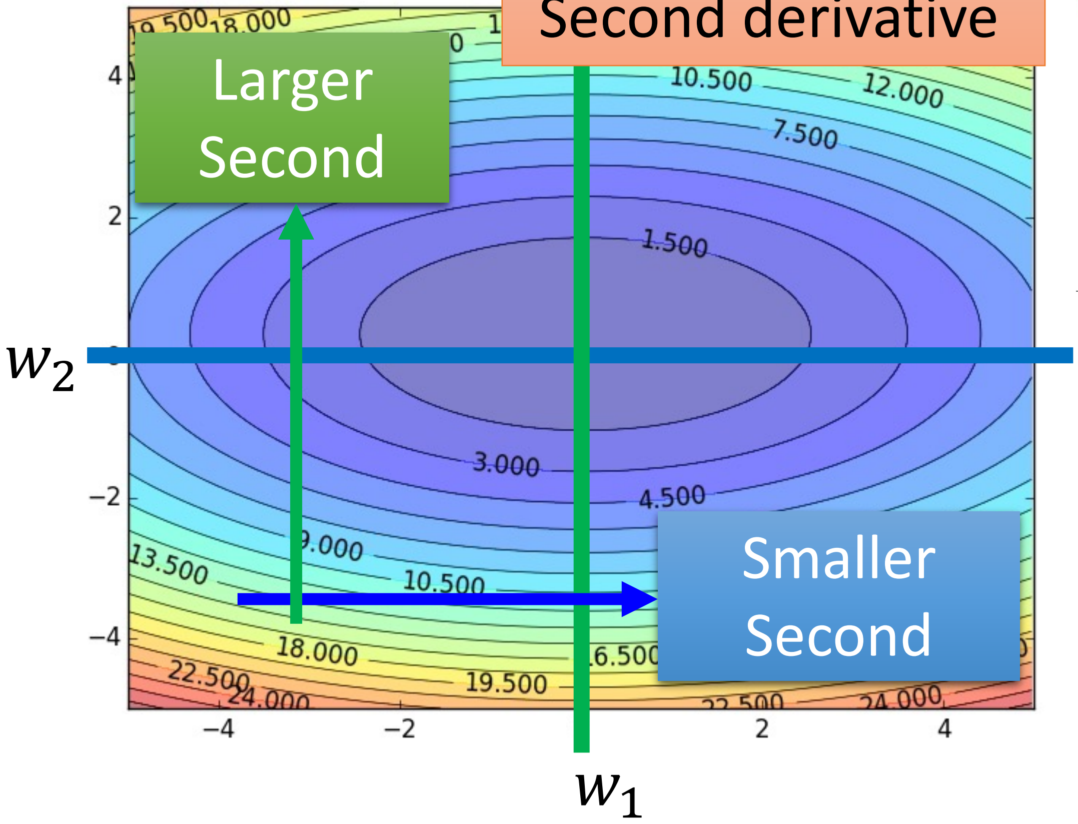
The best step is $\dfrac{|\text{First derivative}|}{\text{Second derivative}}$

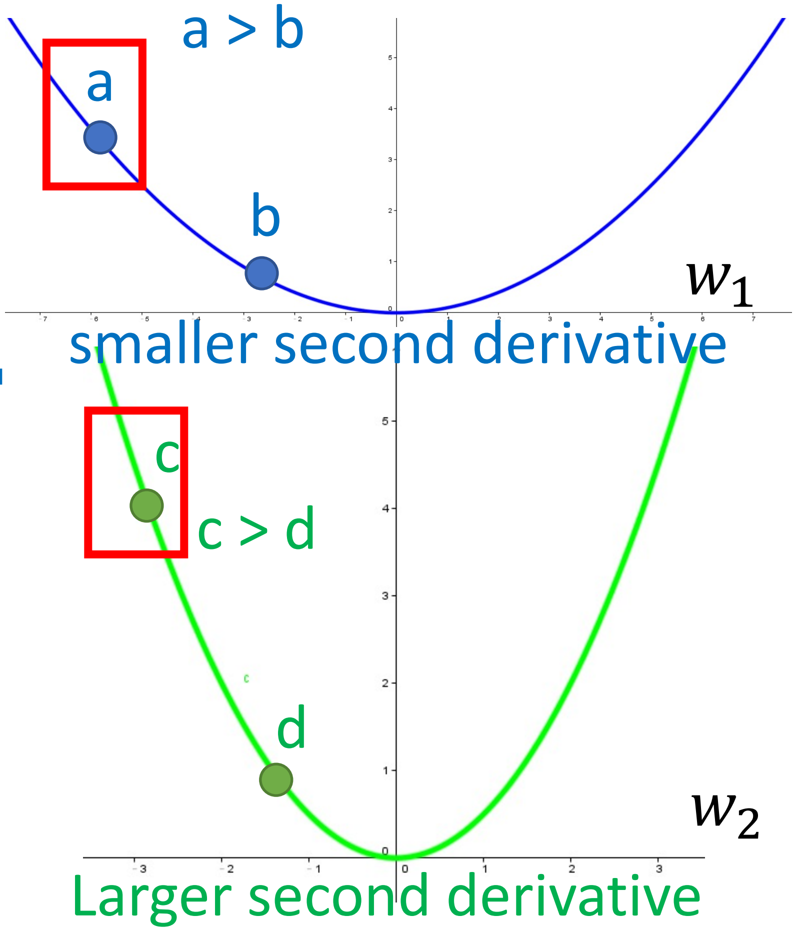
Comparison between different parameters

~~Larger 1^{st} order derivative means far from the minima~~

Do not cross parameters

The best step is $\dfrac{|\text{First derivative}|}{\text{Second derivative}}$





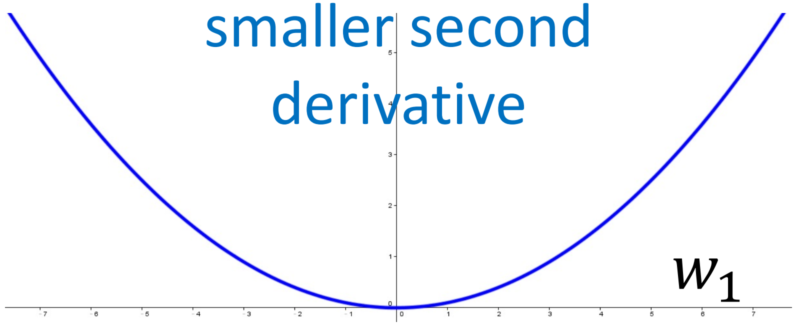
$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}}\, g^t$$

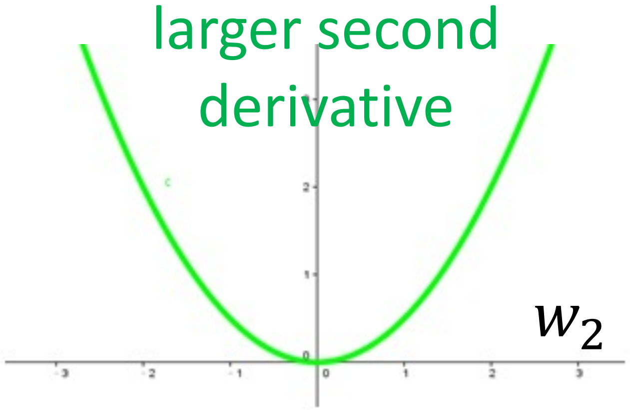
The best step is

$$\frac{|\text{First derivative}|}{\text{Second derivative}}$$

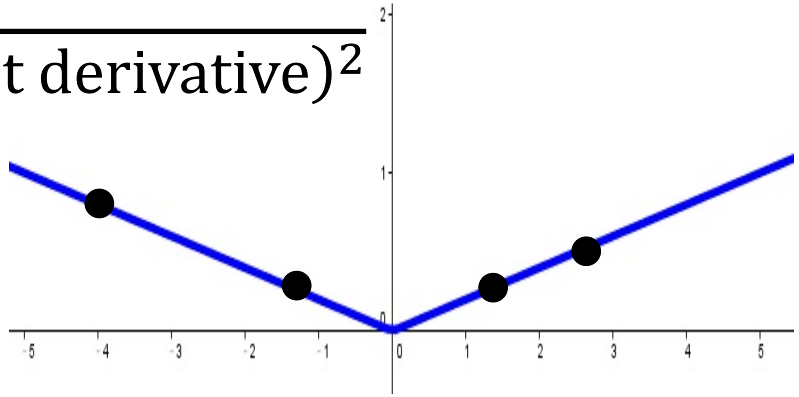
?

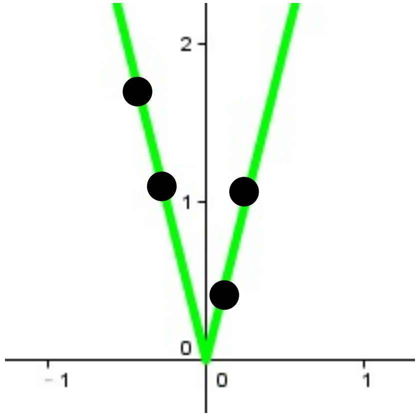
Use *first derivative* to estimate *second derivative*





$\sqrt{(\text{first derivative})^2}$





Implement Adagrad by NumPy

```
import numpy as np
Load features and labels as numpy arrays
    e.g., X = np.array(X)
Initial bias (b), weight (w), learning rate (lr), and iteration
lr_b, lr_w = 0.0, 0.0 // learning rate of bias and weight
for i in range(iteration):
    b_grad, w_grad = 0.0, 0.0
    for j in range(length of X):
        b_grad = b_grad – 2.0*(y[j] – b – w * X[j]) * 1
        w_grad = w_grad – 2.0*(y[j] – b – w * X[j]) * X[j]

    lr_b = lr_b + b_grad ** 2      // customized learning rate
    lr_w = lr_w + w_grad ** 2

    b = b – lr / np.sqrt(lr_b) * b_grad
    w = w – lr / np.sqrt(lr_w) * w_grad
```

You can also use dot product:
y = np.dot(x,w) + b

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

Gradient Descent

Tip 2: Stochastic Gradient Descent

Make the training faster

Stochastic Gradient Descent

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i^n \right) \right)^2$$

Loss is the summation over all training examples

- ***Gradient Descent*** $\theta^i = \theta^{i-1} - \eta \nabla L\left(\theta^{i-1}\right)$
- ***Stochastic Gradient Descent*** Faster!

Pick an example x^n

$$L^n = \left(\hat{y}^n - \left(b + \sum w_i x_i^n \right) \right)^2 \qquad \theta^i = \theta^{i-1} - \eta \nabla L^n\left(\theta^{i-1}\right)$$

Loss for only one example

Stochastic Gradient Descent

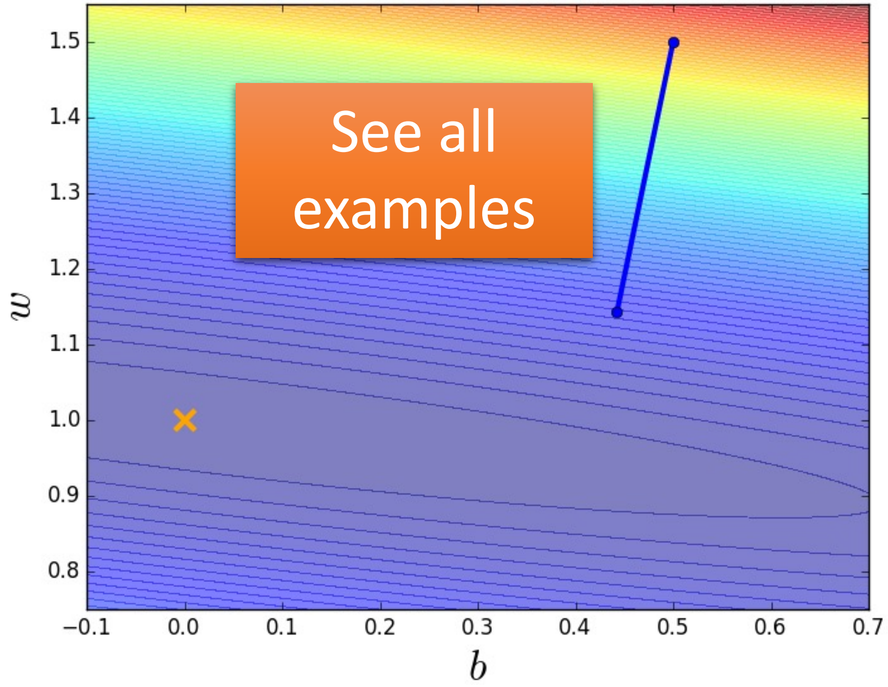
Gradient Descent

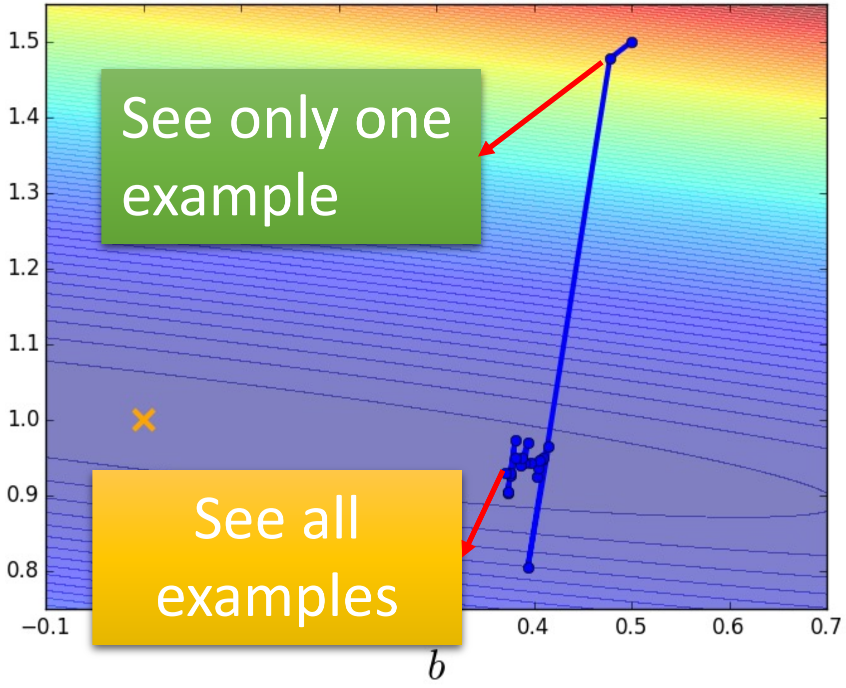
Update after seeing all examples

Stochastic Gradient Descent

Update for each example

If there are 20 examples, 20 times faster.





Batch

$$\boldsymbol{\theta}^* = arg\min_{\boldsymbol{\theta}} L$$

$$\boldsymbol{g} = \begin{bmatrix} \frac{\partial L}{\partial \theta_1}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \frac{\partial L}{\partial \theta_2}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \vdots \end{bmatrix}$$

gradient

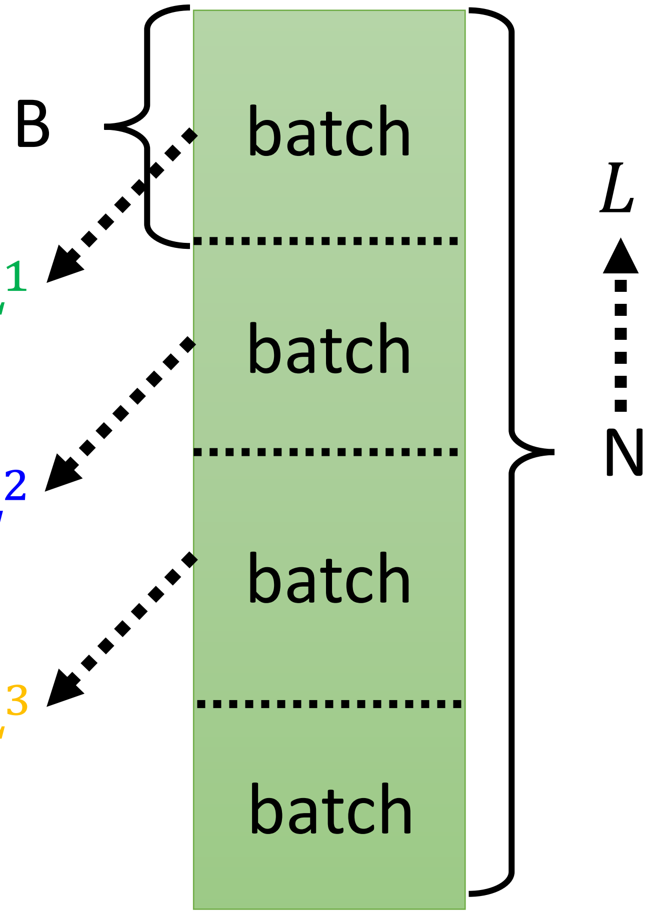
1. (Randomly) Pick initial values $\boldsymbol{\theta}^0$
2. Compute gradient $\boldsymbol{g} = \nabla L^1(\boldsymbol{\theta}^0)$ L^1

   update $\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$
3. Compute gradient $\boldsymbol{g} = \nabla L^2(\boldsymbol{\theta}^1)$ L^2

   update $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g}$
4. Compute gradient $\boldsymbol{g} = \nabla L^3(\boldsymbol{\theta}^2)$ L^3

   update $\boldsymbol{\theta}^3 \leftarrow \boldsymbol{\theta}^2 - \eta \boldsymbol{g}$

1 **epoch** = see all the batches once



$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \\ \vdots \end{bmatrix} \leftarrow \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \\ \vdots \end{bmatrix} - \begin{bmatrix} \eta \frac{\partial L}{\partial \theta_1}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \eta \frac{\partial L}{\partial \theta_2}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^0} \\ \vdots \end{bmatrix}$$

$$\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$$

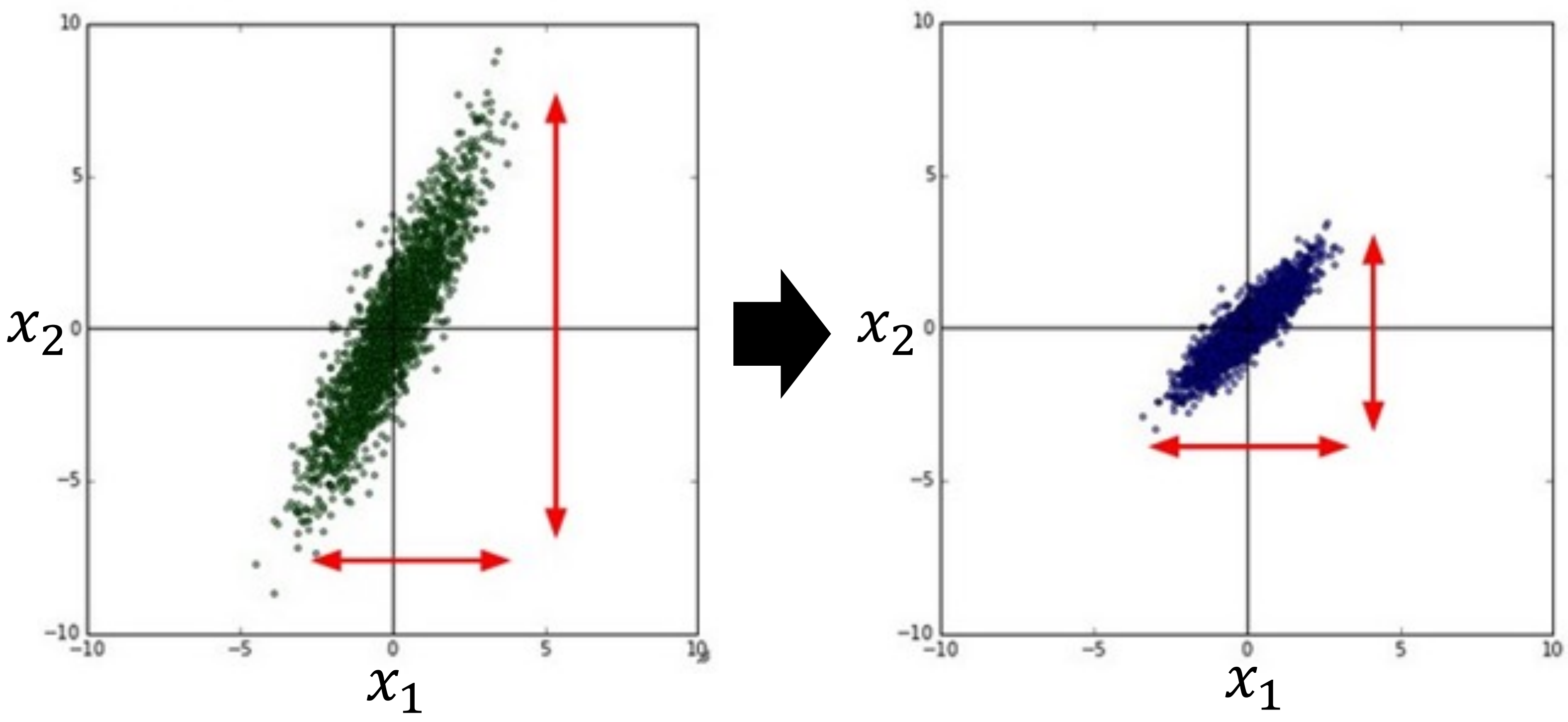
Gradient Descent

Tip 3: Feature Scaling

Feature Scaling

Source of figure: http://cs231n.github.io/neural-networks-2/

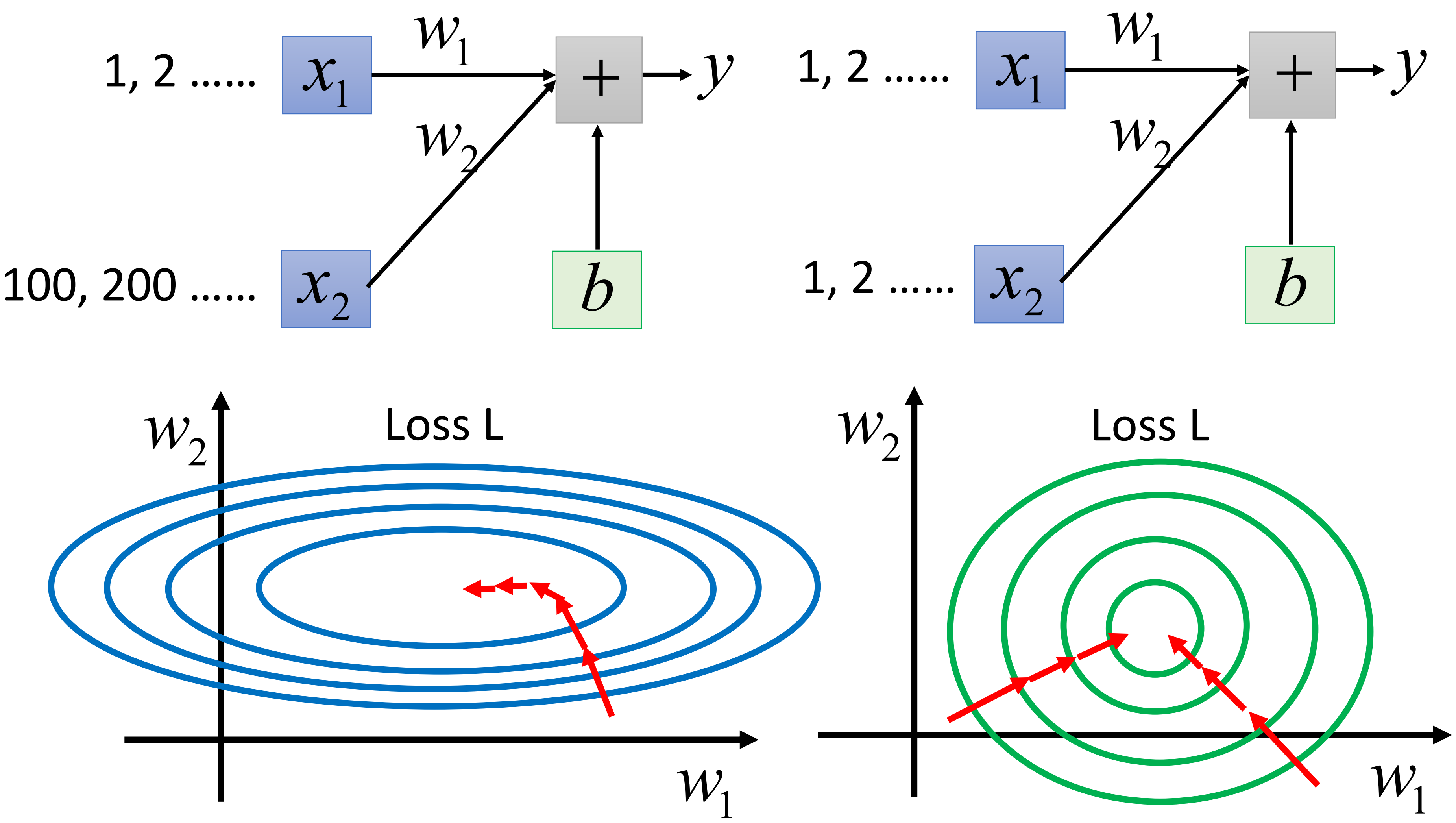
$$y = b + w_1 x_1 + w_2 x_2$$



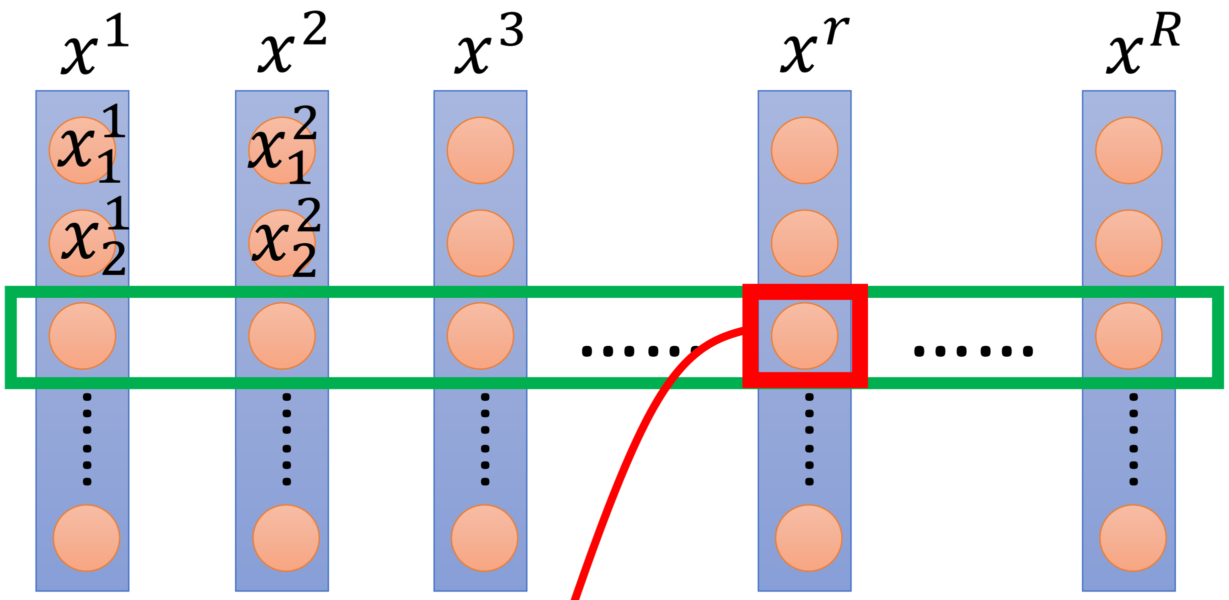
Make different features have the same scaling

Feature Scaling

$$y = b + w_1 x_1 + w_2 x_2$$



Feature Scaling



For each dimension i:

mean: m_i

standard deviation: σ_i

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

The means of all dimensions are 0, and the variances are all 1

Optimization

$$w^*, b^* = arg \min_{w,b} L(w, b)$$

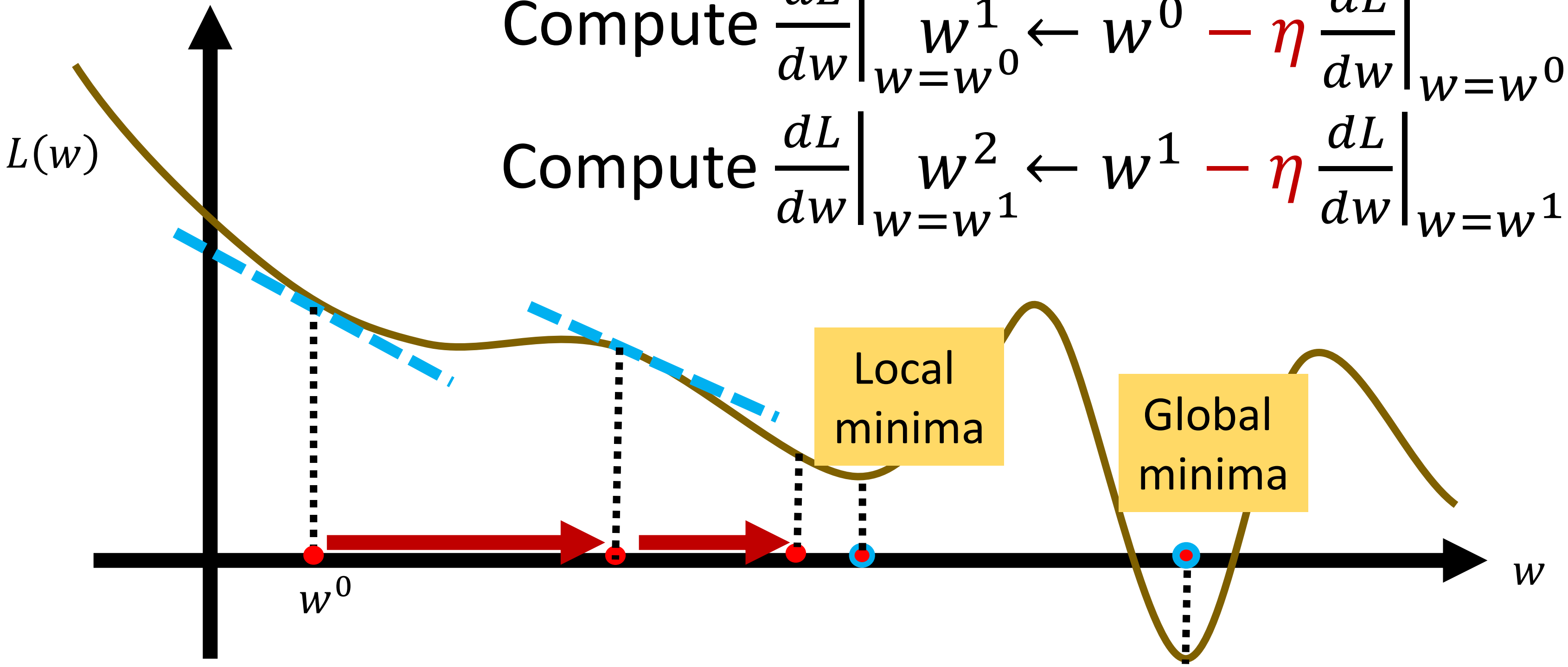
Gradient Descent

Pick an initial value w^0

Compute $\left.\frac{dL}{dw}\right|_{w=w^0}$ $\quad w^1 \leftarrow w^0 - \eta \left.\frac{dL}{dw}\right|_{w=w^0}$

Compute $\left.\frac{dL}{dw}\right|_{w=w^1}$ $\quad w^2 \leftarrow w^1 - \eta \left.\frac{dL}{dw}\right|_{w=w^1}$

Many iterations



Momentum: https://www.ruder.io/optimizing-gradient-descent/#momentum

Reference

- Lecture Slides from Machine Learning by Prof. Hung-Yi Lee
  - https://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/Gradient%20Descent.pdf