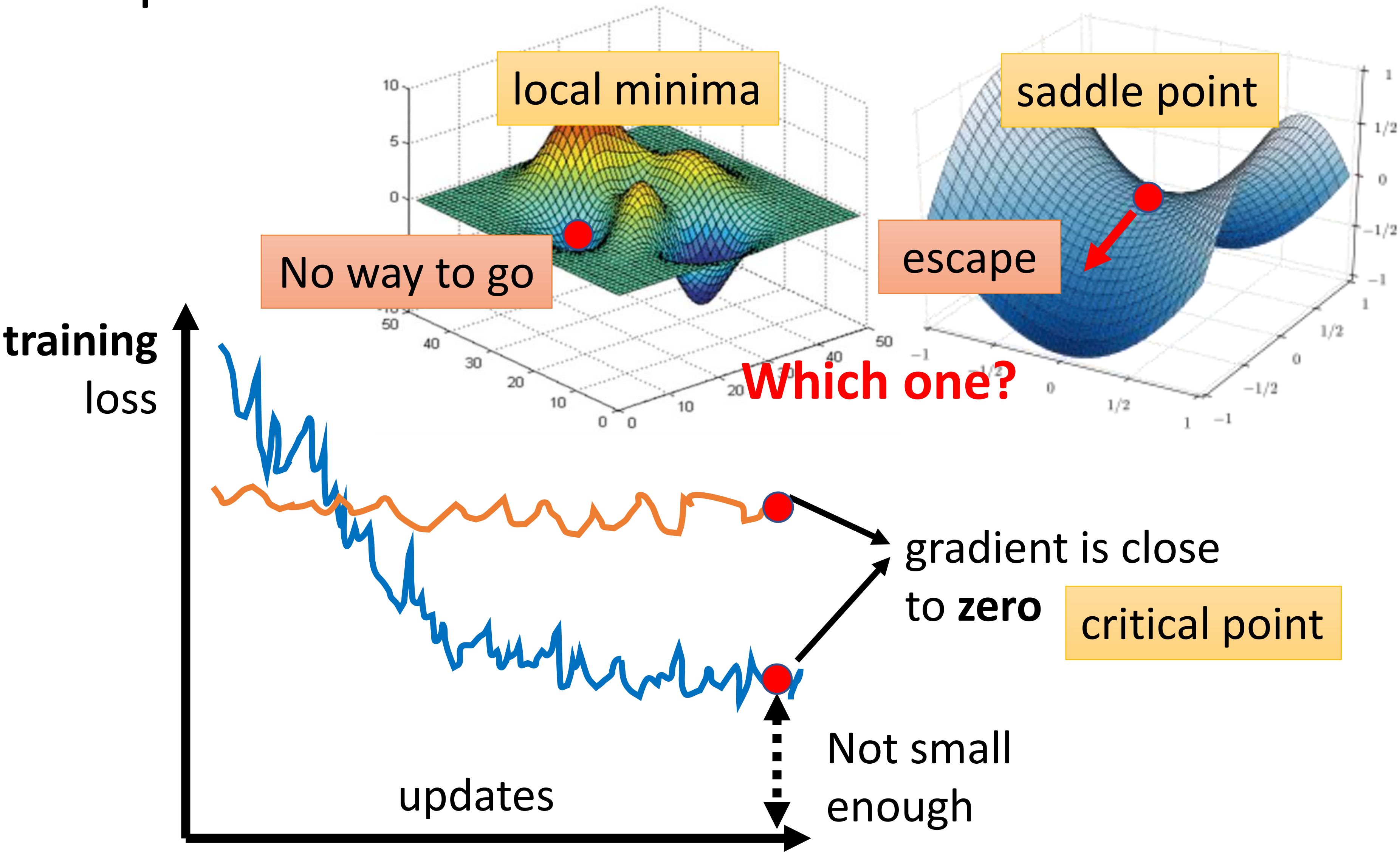
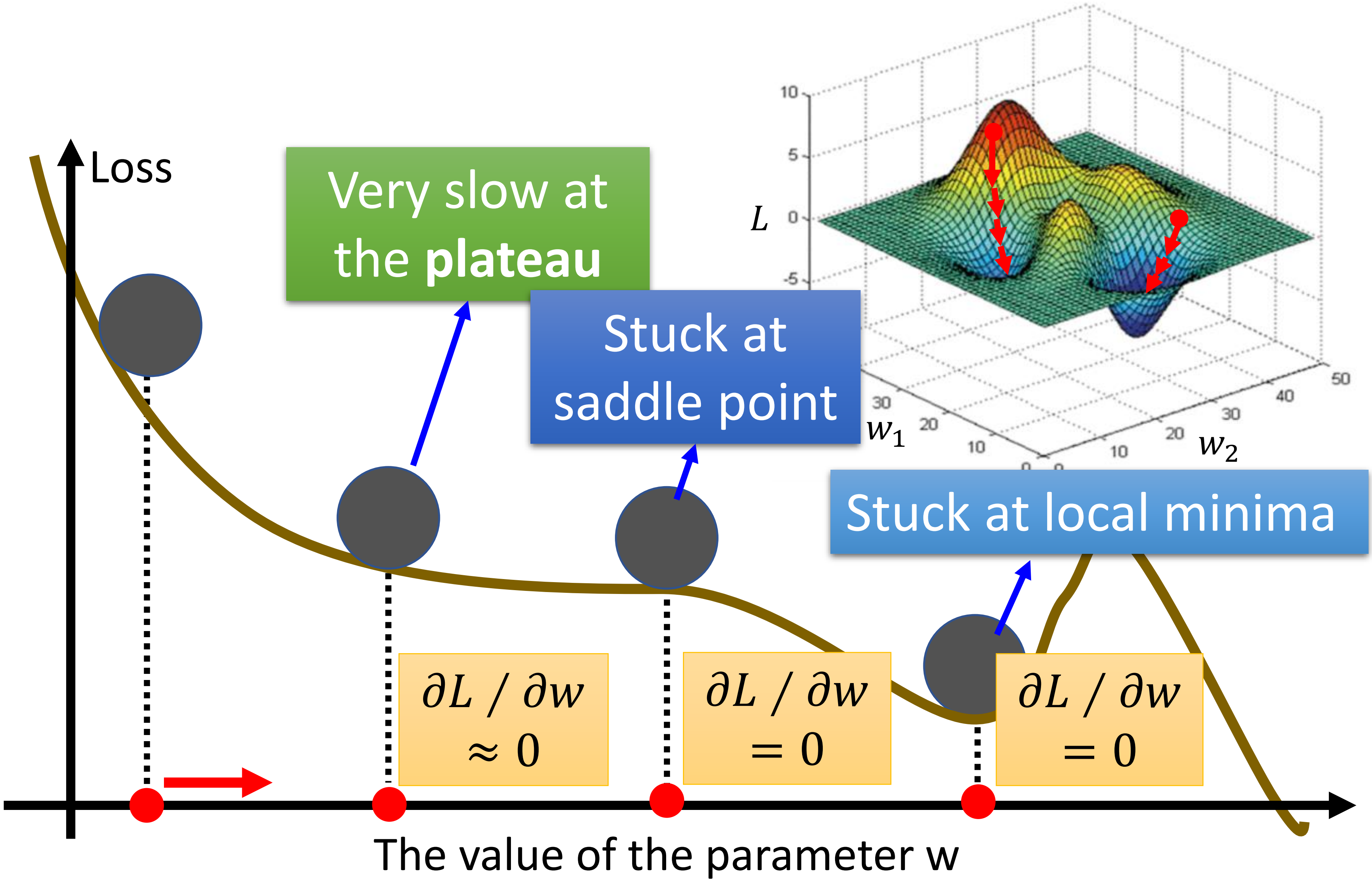
人工智能技术及应用

Artificial Intelligence and Application

Optimization Fails because ……



Small Gradient …

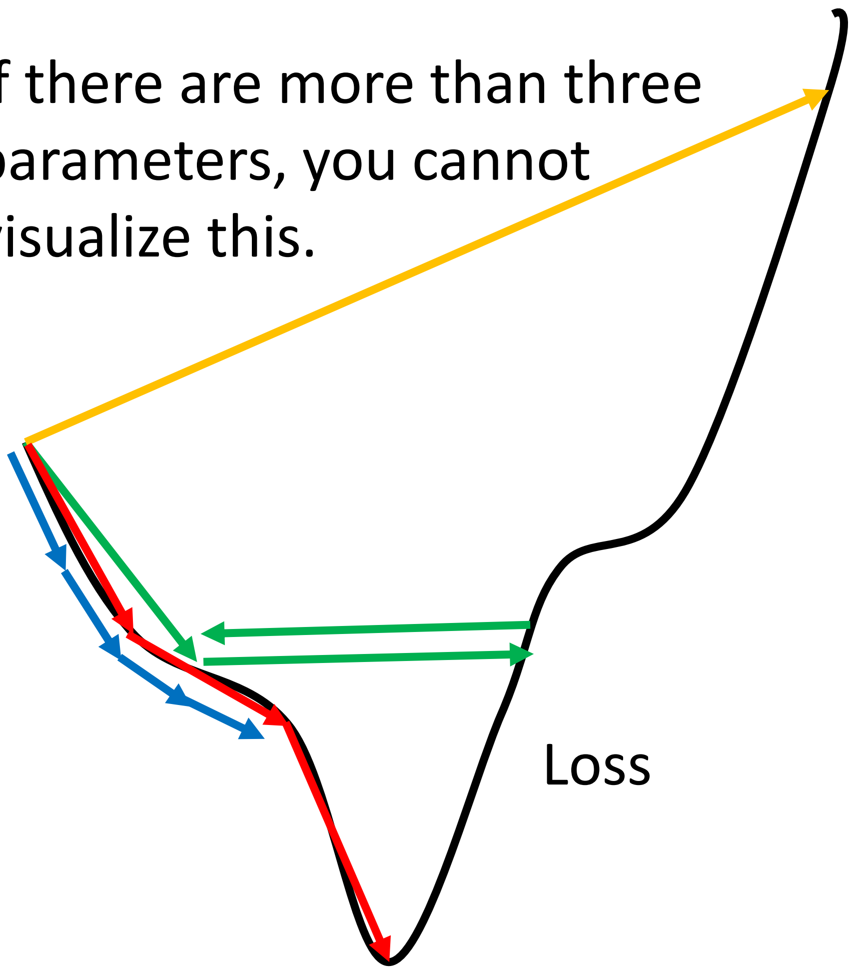


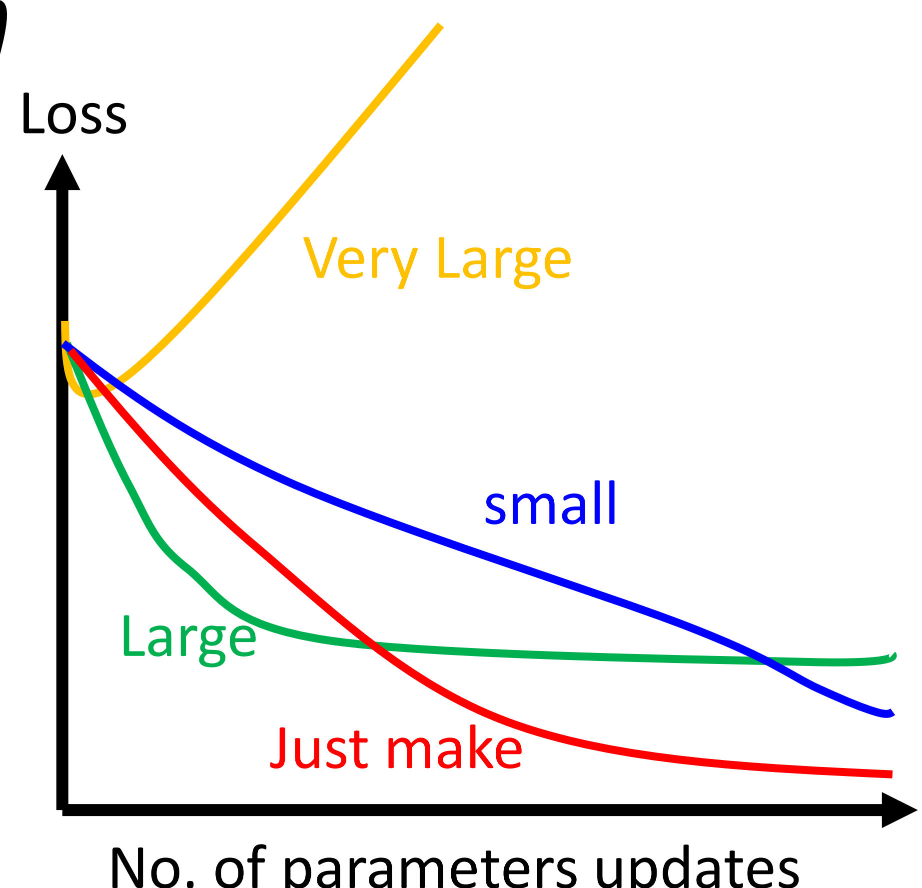
Learning Rate

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

Set the learning rate η carefully

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

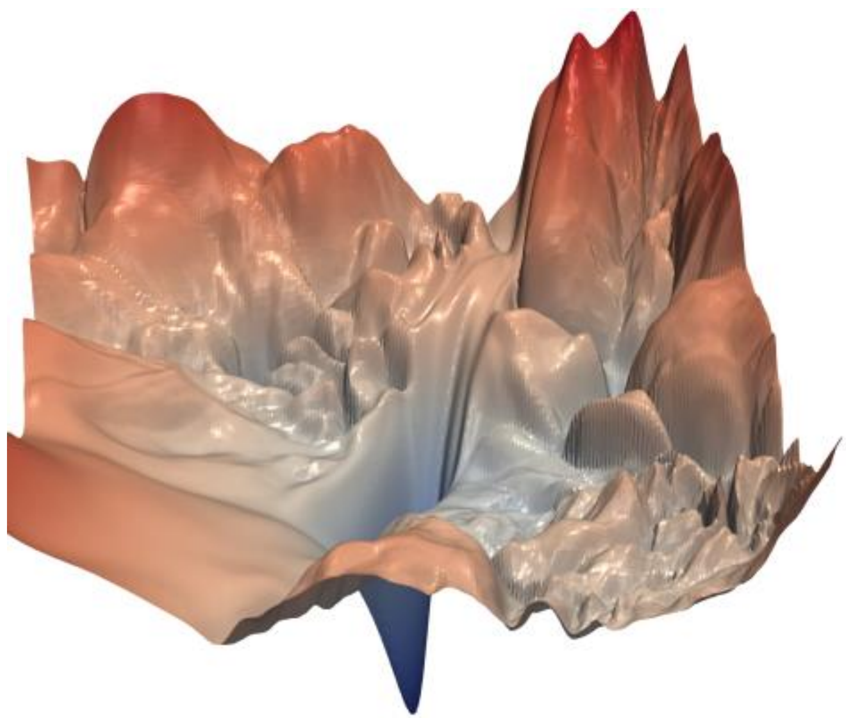




But you can always visualize this.

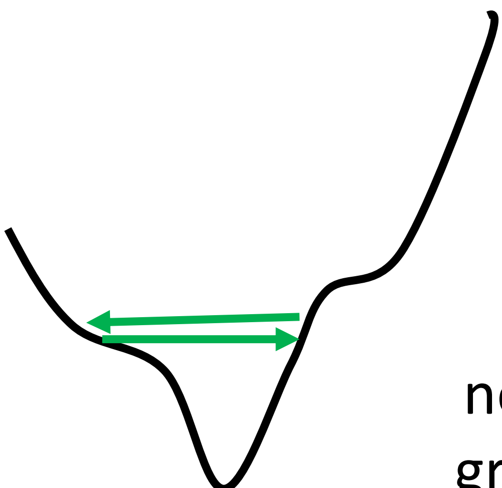
Error surface is rugged …

Tips for training: **Adaptive Learning Rate**

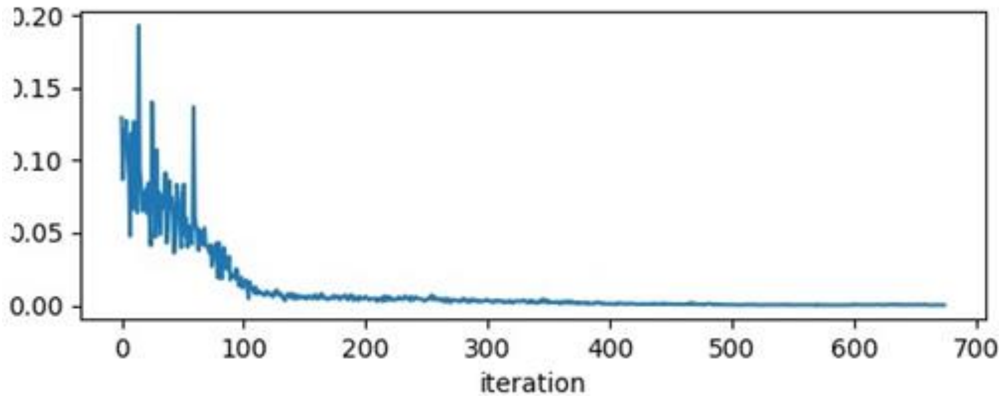


Training stuck ≠ Small Gradient

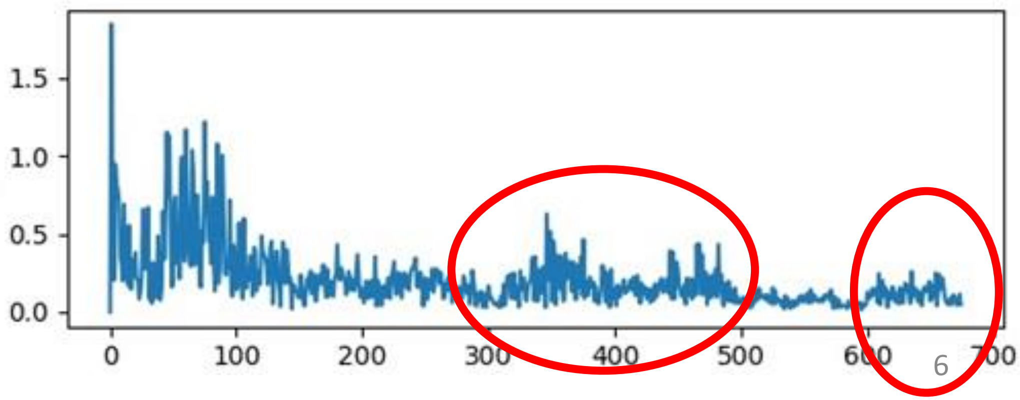
- People believe training stuck because the parameters are around a critical point …



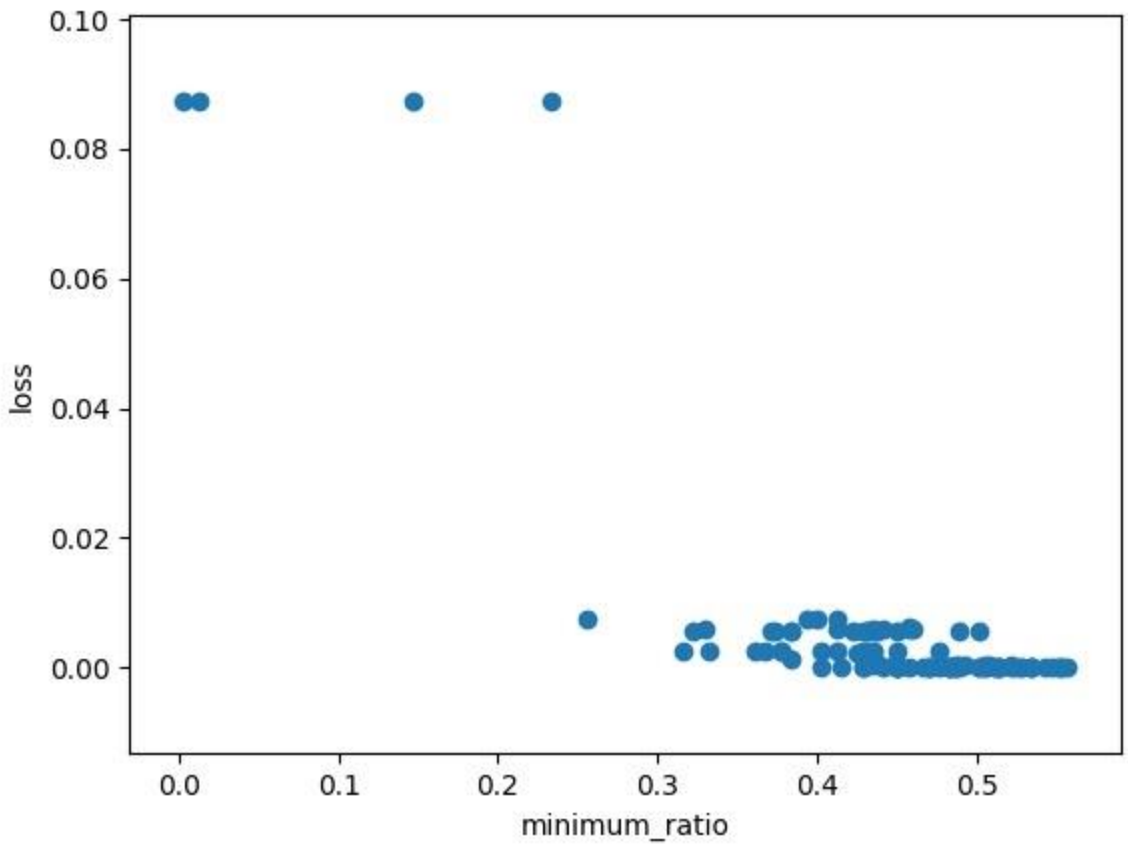
loss



norm of gradient

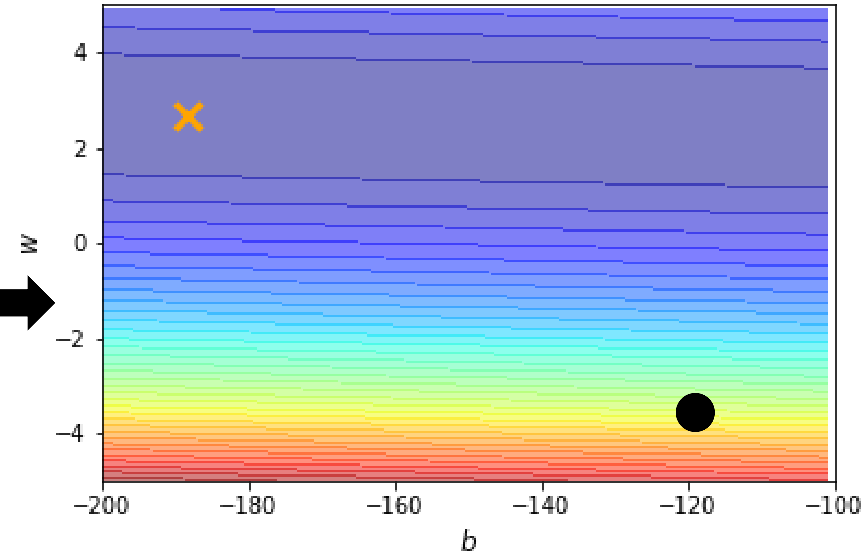


Wait a minute …

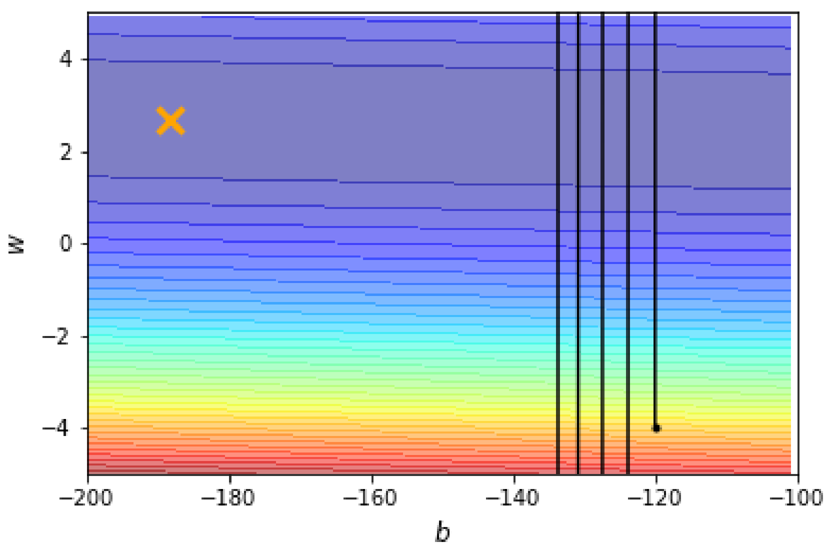


Training can be difficult even without critical points.

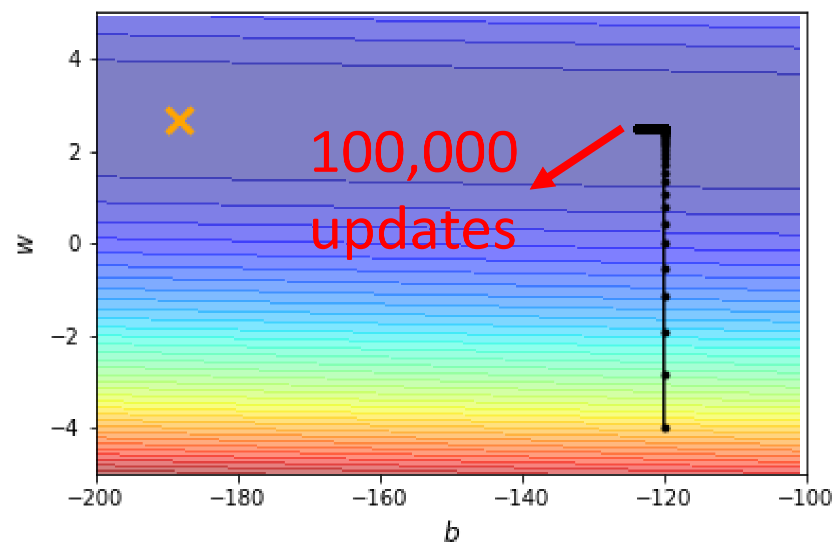
This error surface is convex.



Learning rate **cannot** be **one-size-fits-all**



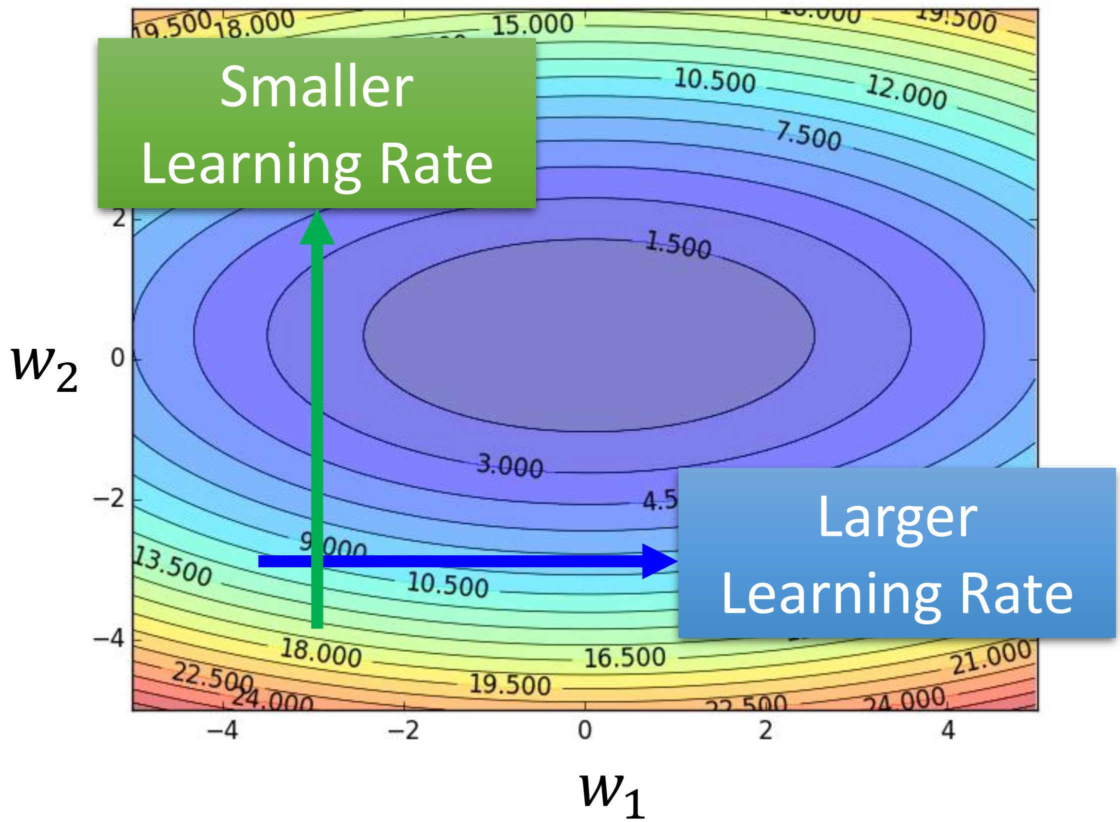
$\eta = 10^{-2}$



$\eta = 10^{-7}$

Different parameters needs different learning rate

Formulation for **one** parameter:



$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^t - \boxed{\eta}\, \boldsymbol{g}_i^t$$

$$\boldsymbol{g}_i^t = \frac{\partial L}{\partial \boldsymbol{\theta}_i} |_{\boldsymbol{\theta}=\boldsymbol{\theta}^t}$$

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^t - \boxed{\frac{\eta}{\sigma_i^t}}\, \boldsymbol{g}_i^t$$

Parameter dependent

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

Root Mean Square

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \boxed{\frac{\eta}{\sigma_i^t}} \boldsymbol{g}_i^{t}$$

$$\boldsymbol{\theta}_i^{1} \leftarrow \boldsymbol{\theta}_i^{0} - \frac{\eta}{\sigma_i^0} \boldsymbol{g}_i^{0} \qquad \sigma_i^0 = \sqrt{\left(\boldsymbol{g}_i^{0}\right)^2} = \left|\boldsymbol{g}_i^{0}\right|$$

$$\boldsymbol{\theta}_i^{2} \leftarrow \boldsymbol{\theta}_i^{1} - \frac{\eta}{\sigma_i^1} \boldsymbol{g}_i^{1} \qquad \sigma_i^1 = \sqrt{\frac{1}{2}\left[\left(\boldsymbol{g}_i^{0}\right)^2 + \left(\boldsymbol{g}_i^{1}\right)^2\right]}$$

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

$$\vdots$$

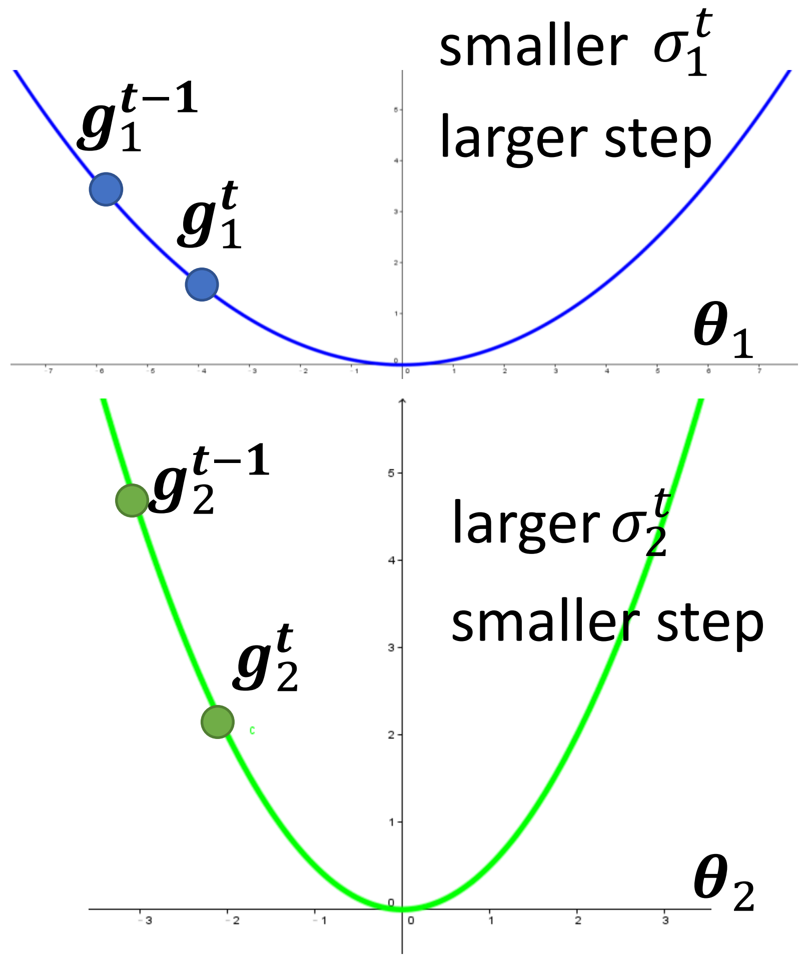
$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \frac{\eta}{\sigma_i^t} \boldsymbol{g}_i^{t} \qquad \sigma_i^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}\left(\boldsymbol{g}_i^{t}\right)^2}$$

Root Mean Square

$$\boldsymbol{\theta}_i^{\boldsymbol{t+1}} \leftarrow \boldsymbol{\theta}_i^{\boldsymbol{t}} - \boxed{\frac{\eta}{\sigma_i^t}} \boldsymbol{g}_i^{\boldsymbol{t}}$$

$$\sigma_i^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}\left(\boldsymbol{g}_i^{\boldsymbol{t}}\right)^2}$$

Used in **Adagrad**



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

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

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

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

$$\vdots$$

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

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

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

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

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

Vanilla Gradient descent

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

Larger gradient, larger step

Adagrad

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

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 反差

特別大

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

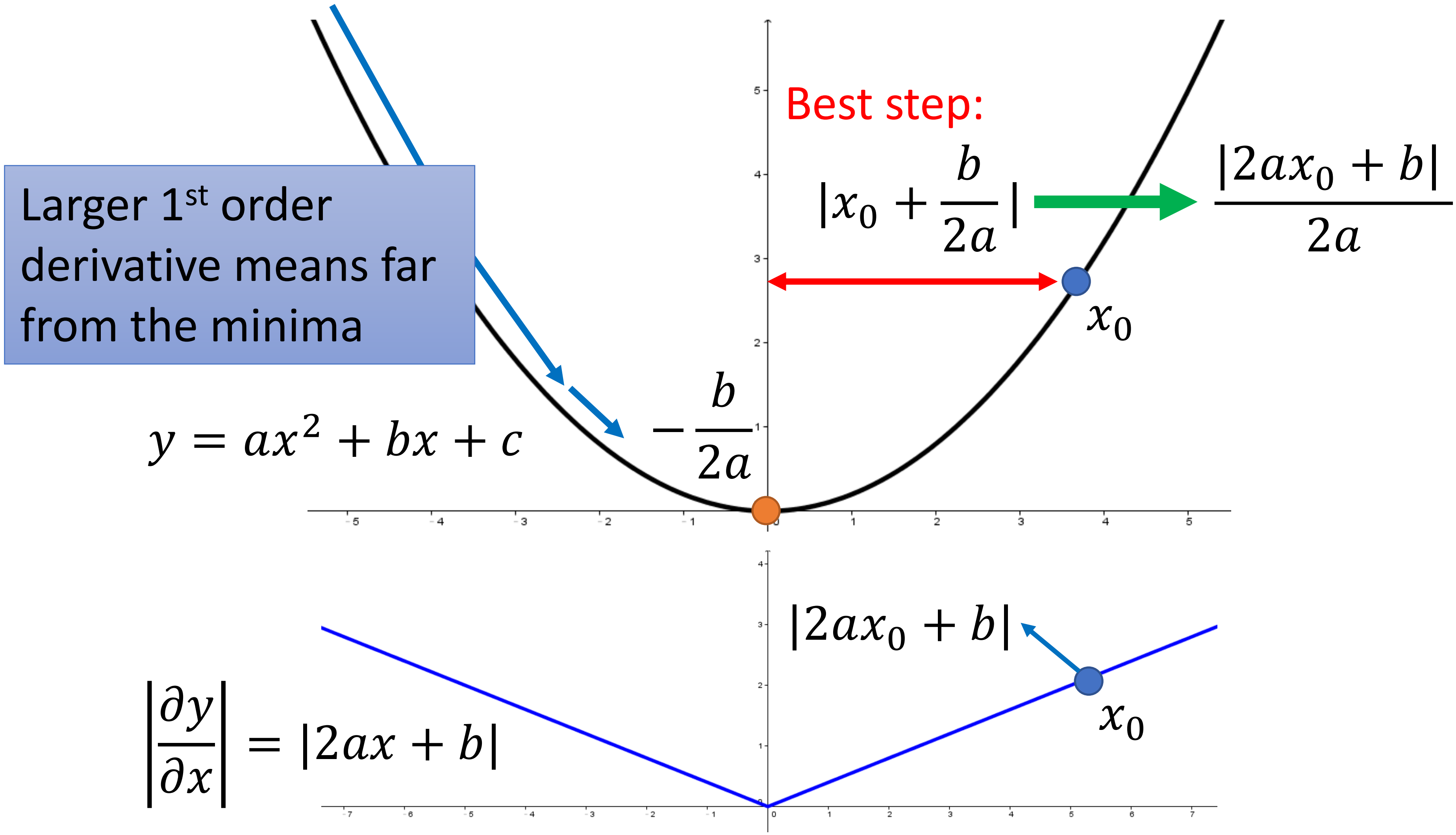
g^0	g^1	g^2	g^3	g^4	……
10.8	20.9	31.7	12.1	0.1	……

特別小

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

造成反差的效果

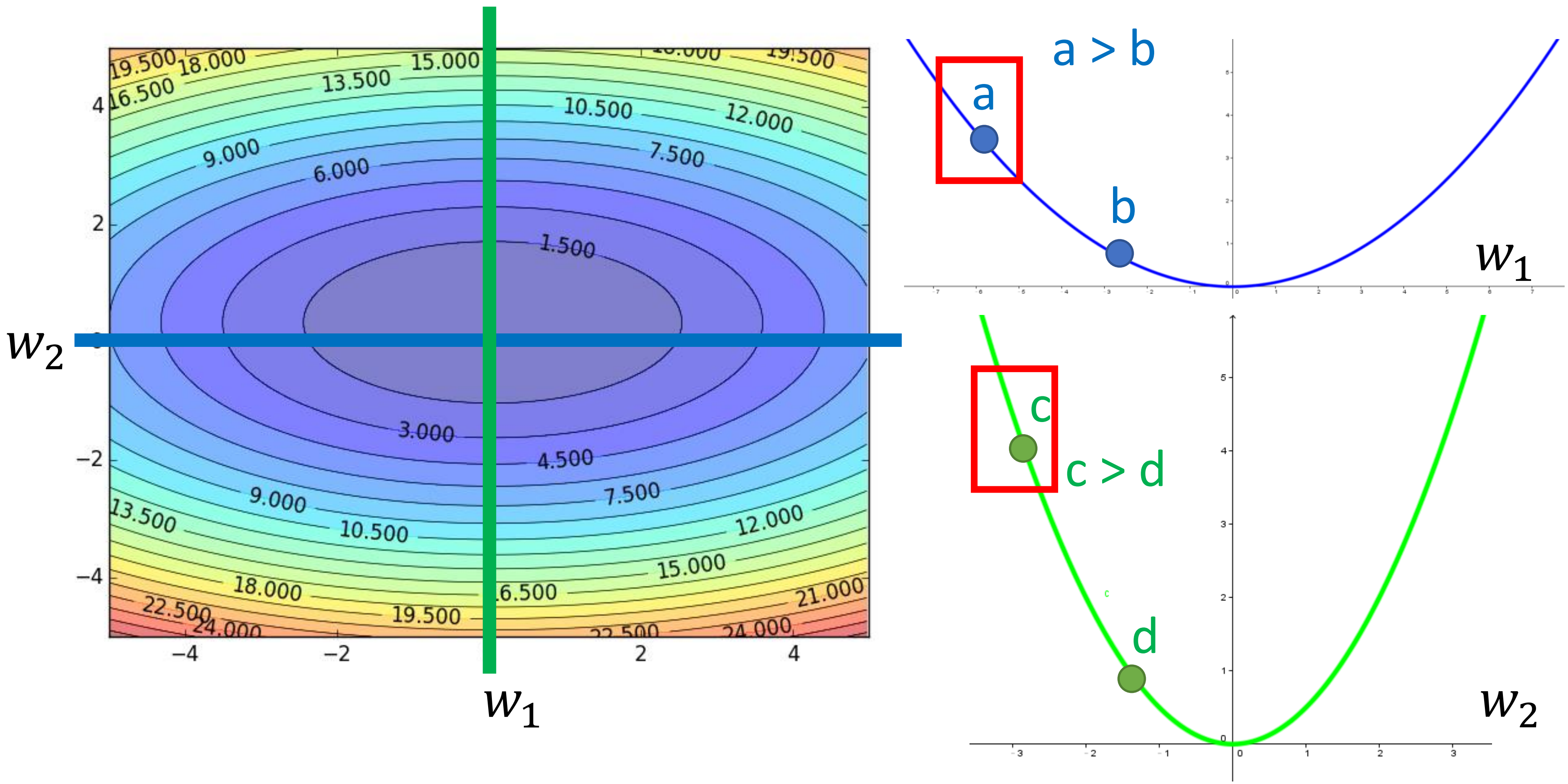
Larger gradient, larger steps?



Comparison between different parameters

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

Do not cross parameters

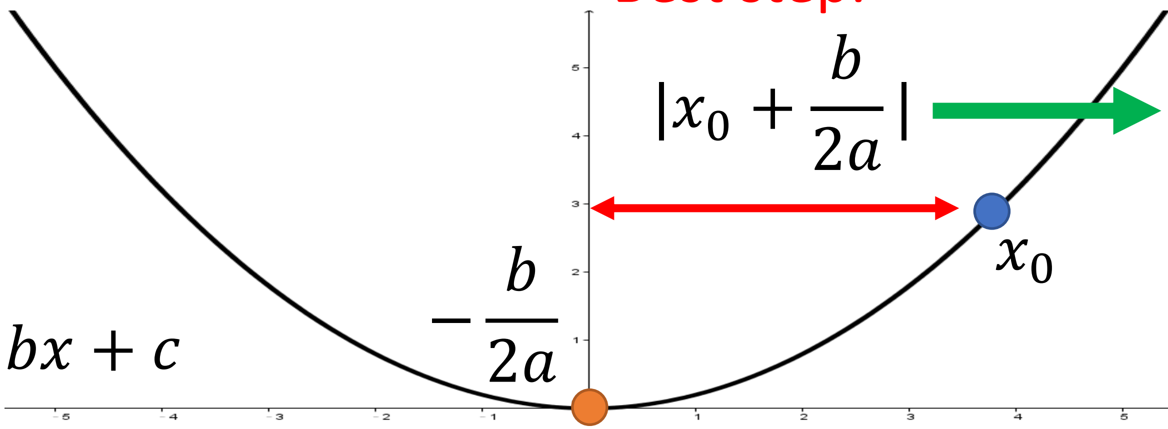


Second Derivative

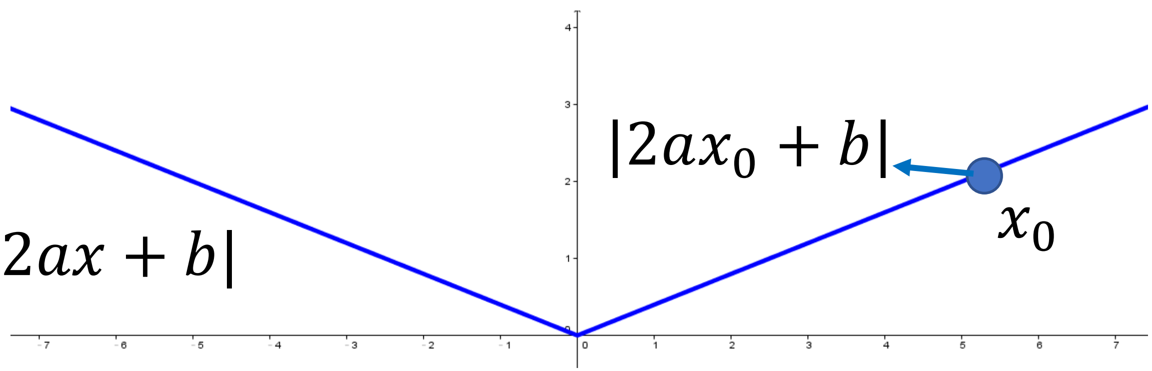
Best step:

$$\frac{|2ax_0 + b|}{\boxed{2a}}$$

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



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



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

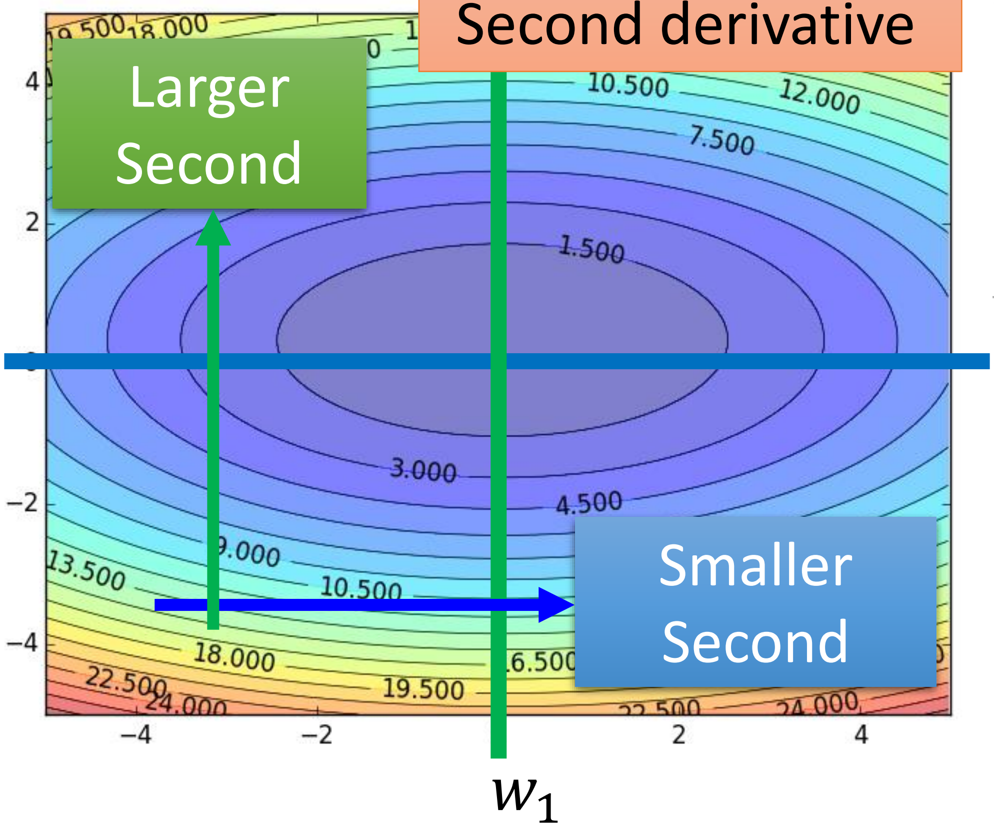
The best step is $\dfrac{|\text{First derivative}|}{\text{Second derivative}}$

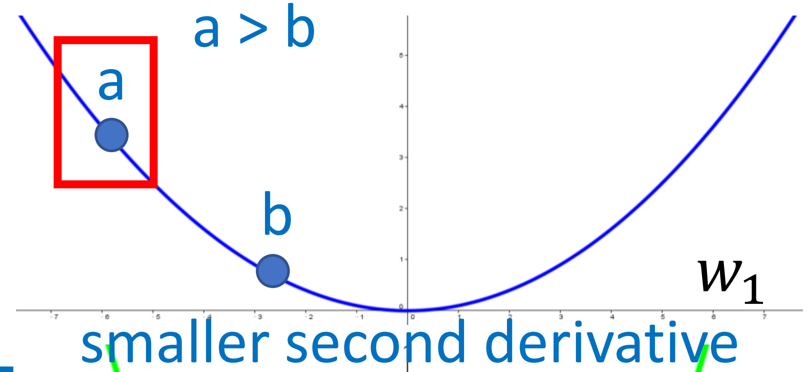
Comparison between different parameters

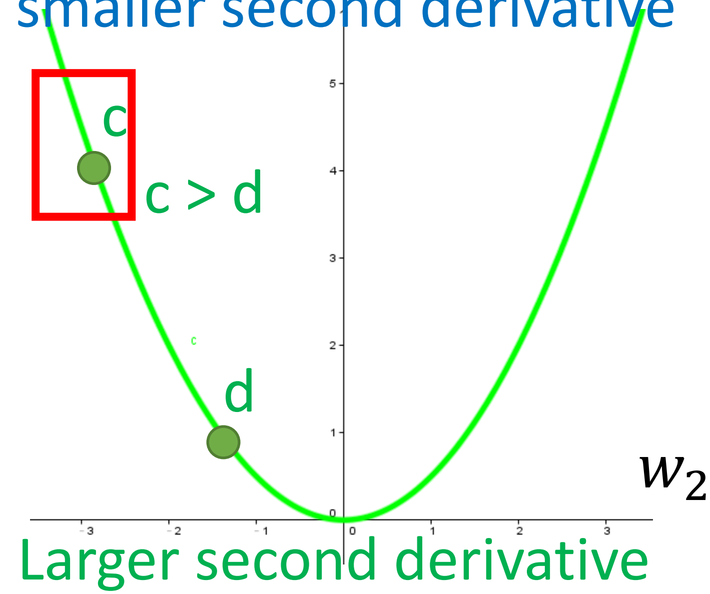
~~Larger 1st order derivative means far from the minima~~

Do not cross parameters

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







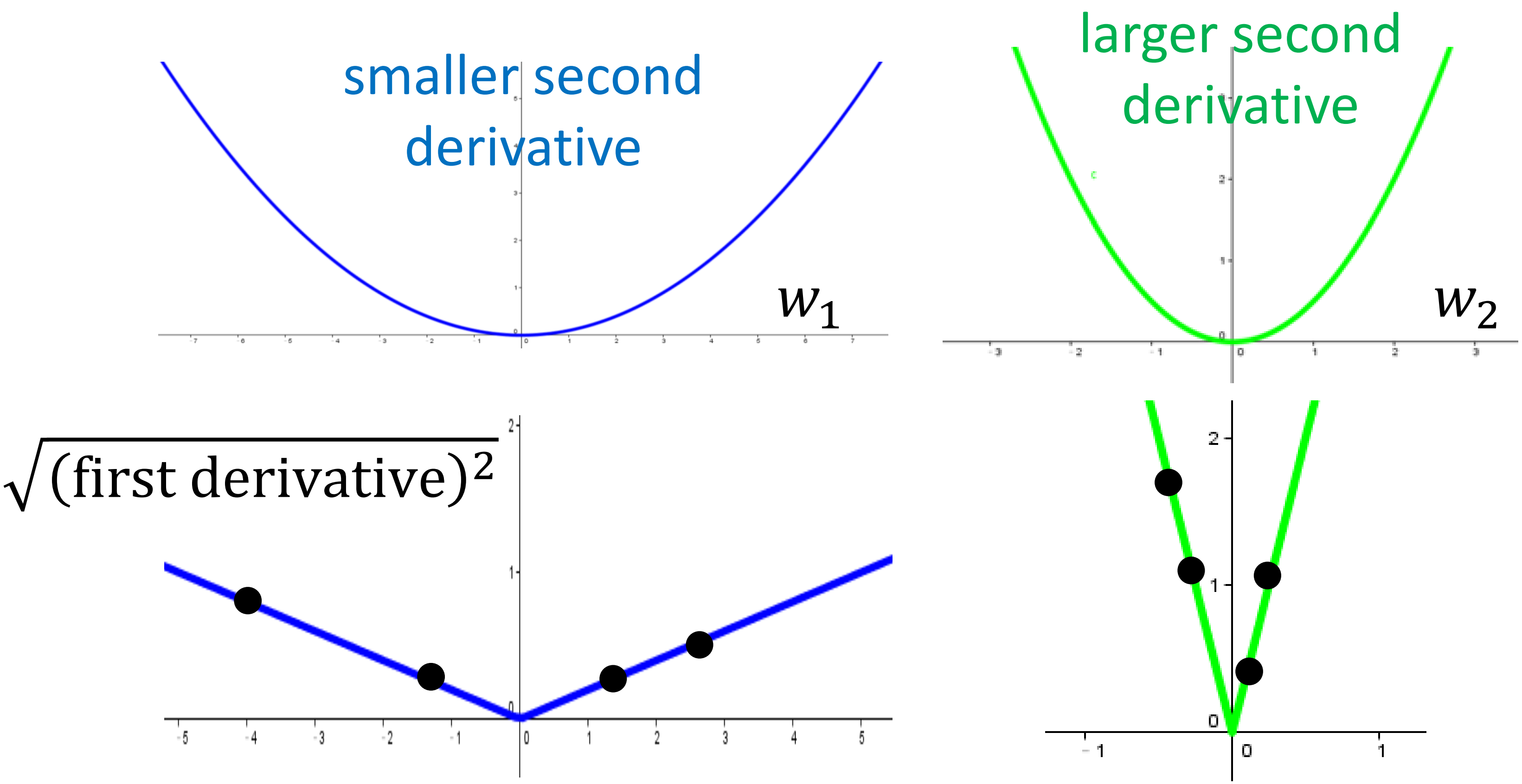
$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

?

The best step is

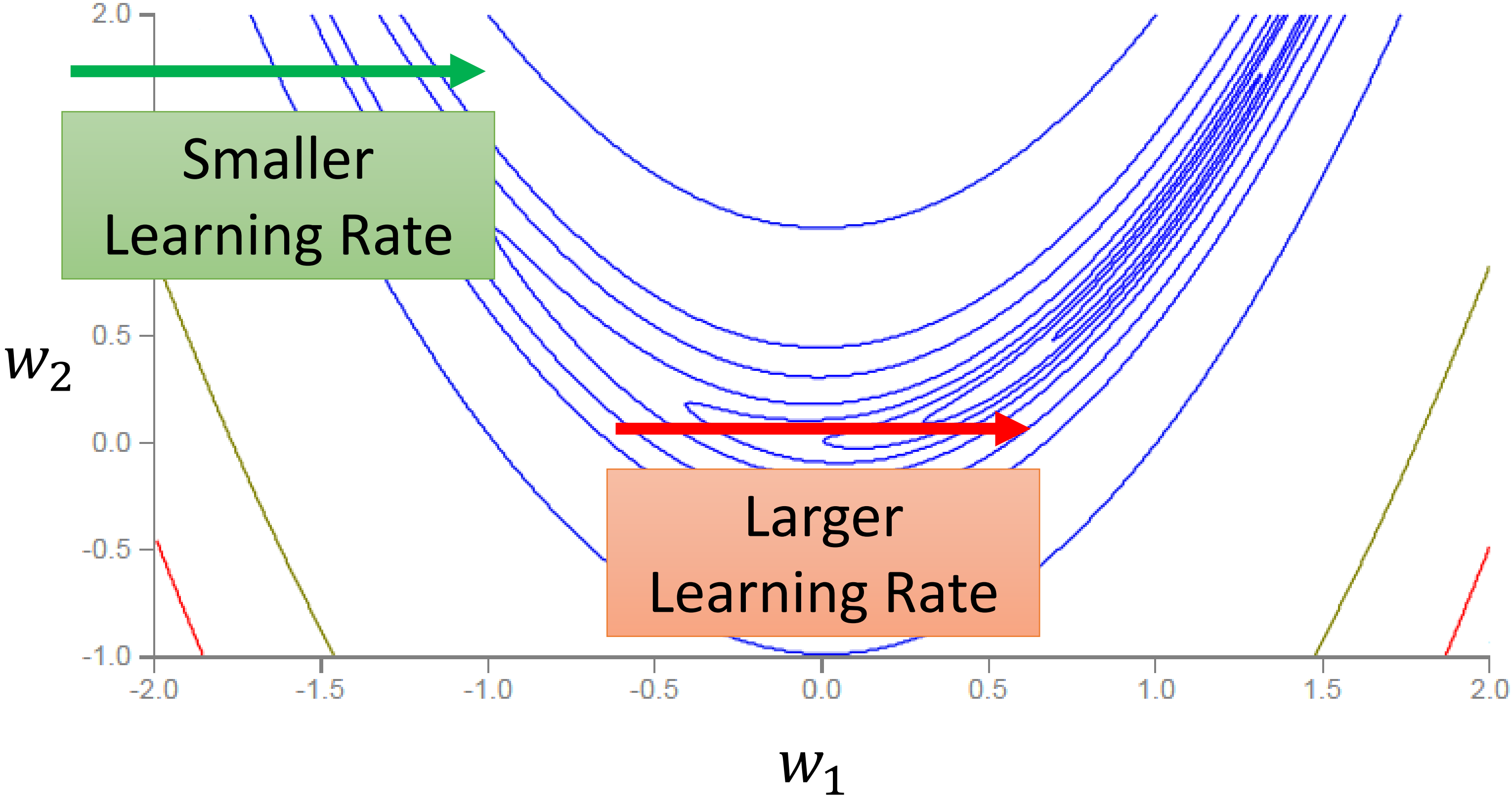
$$\frac{|\text{First derivative}|}{\text{Second derivative}}$$

Use *first derivative* to estimate *second derivative*



Learning rate adapts dynamically

Error Surface can be very complex.



RMSProp

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \boxed{\frac{\eta}{\sigma_i^t}} \boldsymbol{g}_i^{t}$$

$$\boldsymbol{\theta}_i^{1} \leftarrow \boldsymbol{\theta}_i^{0} - \frac{\eta}{\sigma_i^0} \boldsymbol{g}_i^{0} \qquad \sigma_i^0 = \sqrt{\left(\boldsymbol{g}_i^{0}\right)^2}$$

$$0 < \alpha < 1$$

$$\boldsymbol{\theta}_i^{2} \leftarrow \boldsymbol{\theta}_i^{1} - \frac{\eta}{\sigma_i^1} \boldsymbol{g}_i^{1} \qquad \sigma_i^1 = \sqrt{\alpha\left(\sigma_i^0\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{1}\right)^2}$$

$$\boldsymbol{\theta}_i^{3} \leftarrow \boldsymbol{\theta}_i^{2} - \frac{\eta}{\sigma_i^2} \boldsymbol{g}_i^{2} \qquad \sigma_i^2 = \sqrt{\alpha\left(\sigma_i^1\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{2}\right)^2}$$

$$\vdots$$

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \frac{\eta}{\sigma_i^t} \boldsymbol{g}_i^{t} \qquad \sigma_i^t = \sqrt{\alpha\left(\sigma_i^{t-1}\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{t}\right)^2}$$

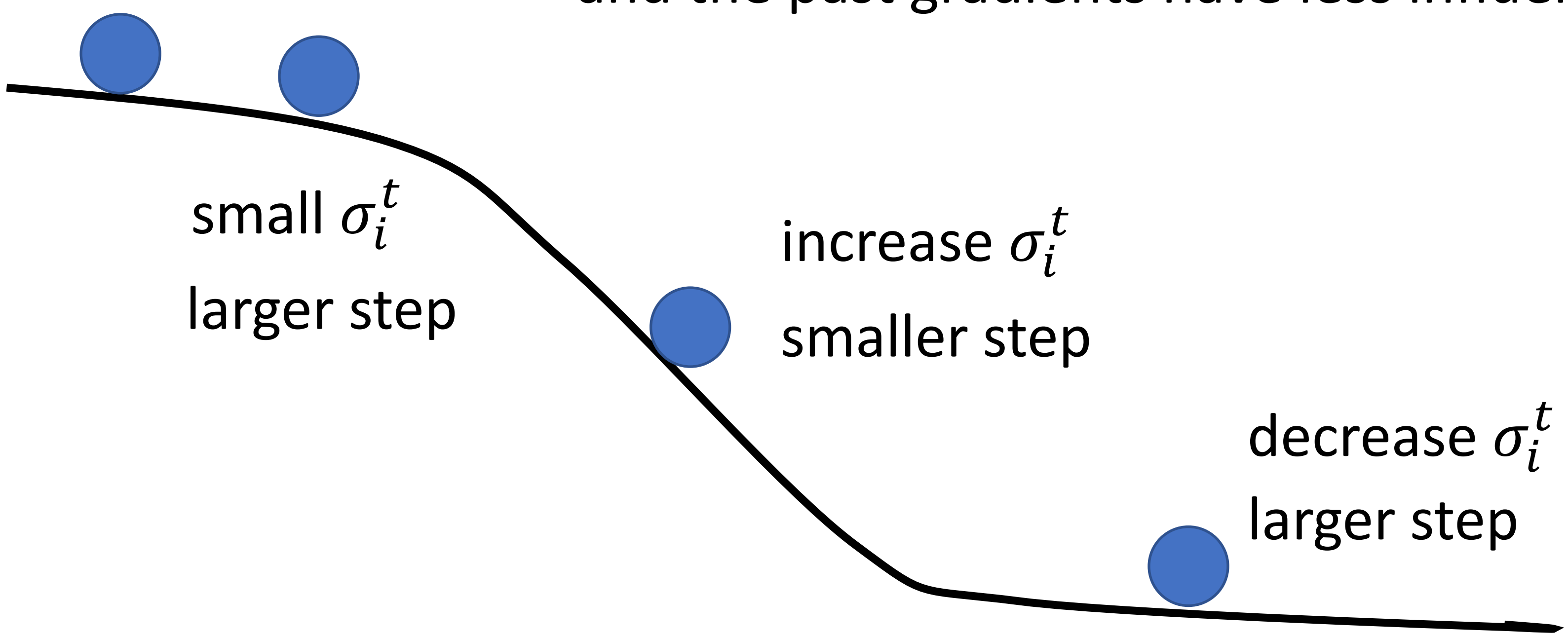
RMSProp

$$\boldsymbol{g}_i^{\boldsymbol{1}} \ \boldsymbol{g}_i^{\boldsymbol{2}} \cdots\cdots \ \boldsymbol{g}_i^{\boldsymbol{t-1}}$$

$$0 < \alpha < 1$$

$$\boldsymbol{\theta}_i^{\boldsymbol{t+1}} \leftarrow \boldsymbol{\theta}_i^{\boldsymbol{t}} - \boxed{\frac{\eta}{\sigma_i^t}} \boldsymbol{g}_i^{\boldsymbol{t}} \qquad \sigma_i^t = \sqrt{\alpha\left(\sigma_i^{t-1}\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{\boldsymbol{t}}\right)^2}$$

The recent gradient has larger influence, and the past gradients have less influence.



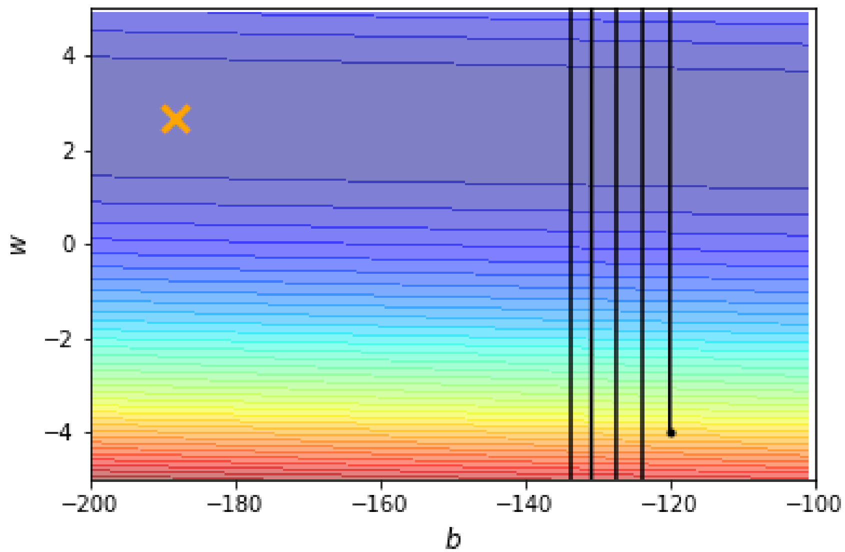
Original paper: https://arxiv.org/pdf/1412.6980.pdf

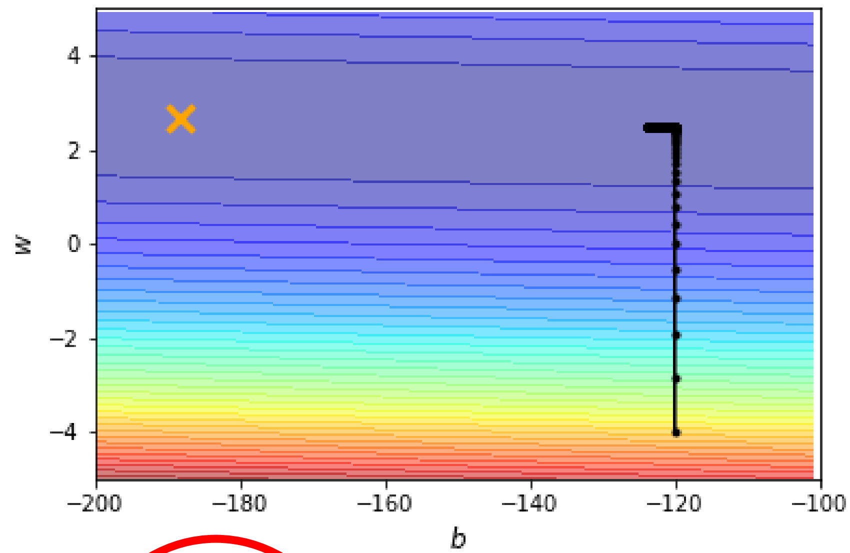
Adam: RMSProp + Momentum

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector) → for RMSprop
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

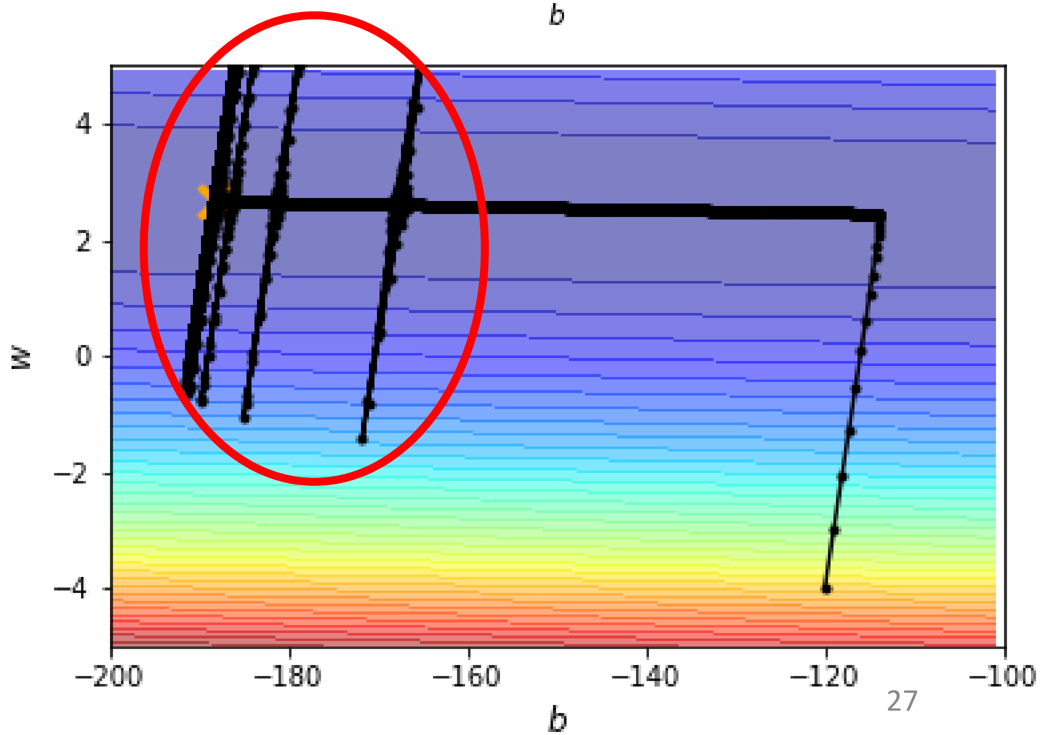
Without Adaptive Learning Rate





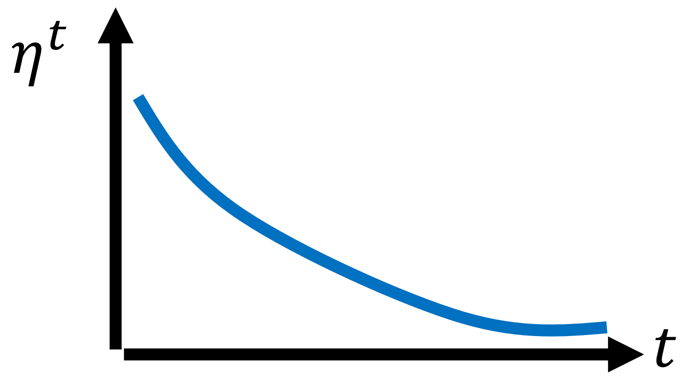
$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^t - \boxed{\frac{\eta}{\sigma_i^t}} \boldsymbol{g}_i^t$$

$$\sigma_i^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}\left(\boldsymbol{g}_i^t\right)^2}$$



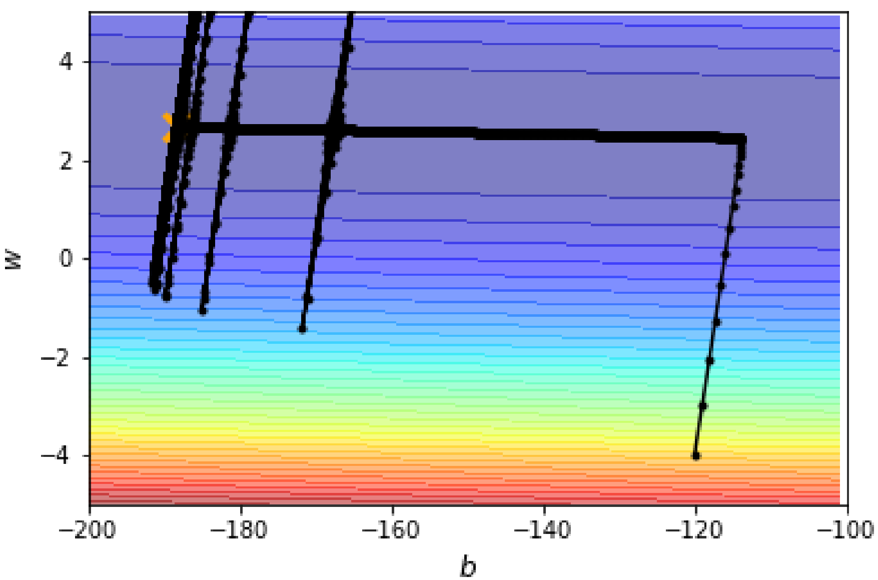
Learning Rate Scheduling

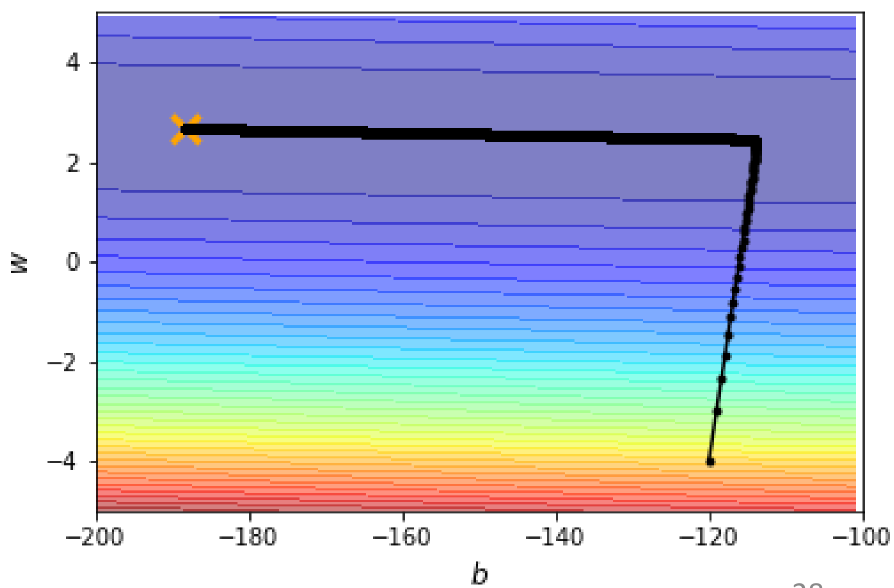
$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\boxed{\eta^t}}{\sigma_i^t} g_i^t$$



Learning Rate Decay

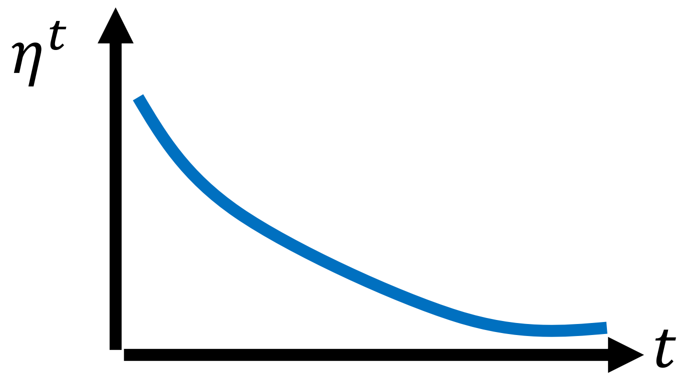
As the training goes, we are closer to the destination, so we reduce the learning rate.





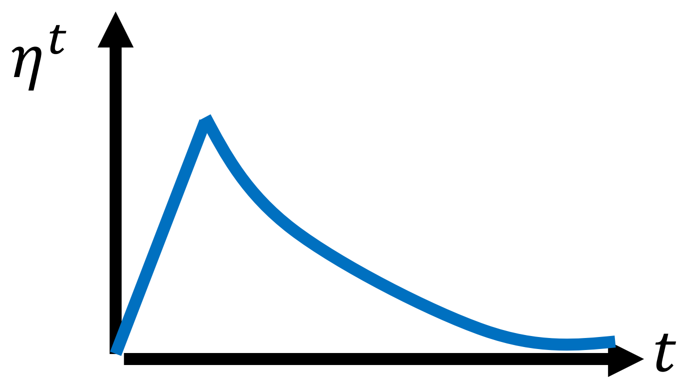
Learning Rate Scheduling

$$\theta_i^{t+1} \leftarrow \theta_i^t - \frac{\boxed{\eta^t}}{\sigma_i^t} g_i^t$$



Learning Rate Decay

As the training goes, we are closer to the destination, so we reduce the learning rate.



Warm Up

Increase and then decrease?

We further explore $n = 18$ that leads to a 110-layer ResNet. In this case, we find that the initial learning rate of 0.1 is slightly too large to start converging[5]. So we use 0.01 to warm up the training until the training error is below 80% (about 400 iterations), and then go back to 0.1 and continue training. The rest of the learning schedule is as done previously. This 110-layer network converges well (Fig. 6, middle). It has *fewer* parameters than other deep and thin

[5] With an initial learning rate of 0.1, it starts converging (<90% error) after several epochs, but still reaches similar accuracy.

Residual Network

https://arxiv.org/abs/1512.03385

5.3 Optimizer

We used the Adam optimizer [17] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \qquad (3)$$

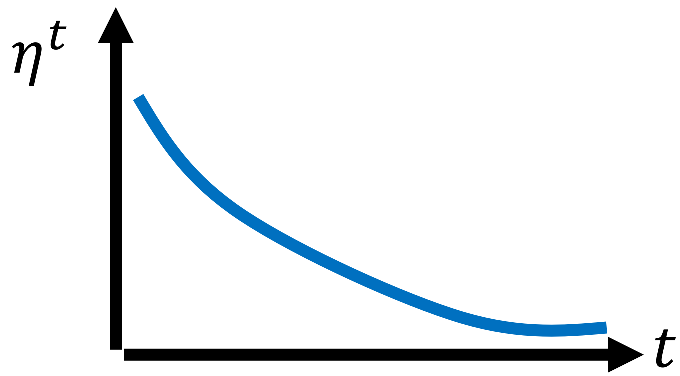
This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

Transformer

https://arxiv.org/abs/1706.03762

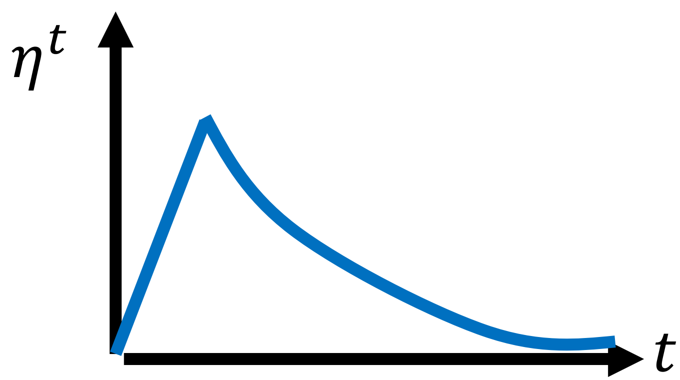
Learning Rate Scheduling

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \frac{\boxed{\eta^t}}{\sigma_i^t} \boldsymbol{g}_i^t$$



Learning Rate Decay

After the training goes, we are close to the destination, so we reduce the learning rate.



Warm Up

Increase and then decrease?

At the beginning, the estimate of σ_i^t has large variance.

Please refer to **RAdam** https://arxiv.org/abs/1908.03265

Summary of Optimization

(Vanilla) Gradient Descent

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \eta \boldsymbol{g}_i^{t}$$

Various Improvements

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \frac{\eta^t}{\sigma_i^t} \boldsymbol{m}_i^{t}$$

Learning rate scheduling

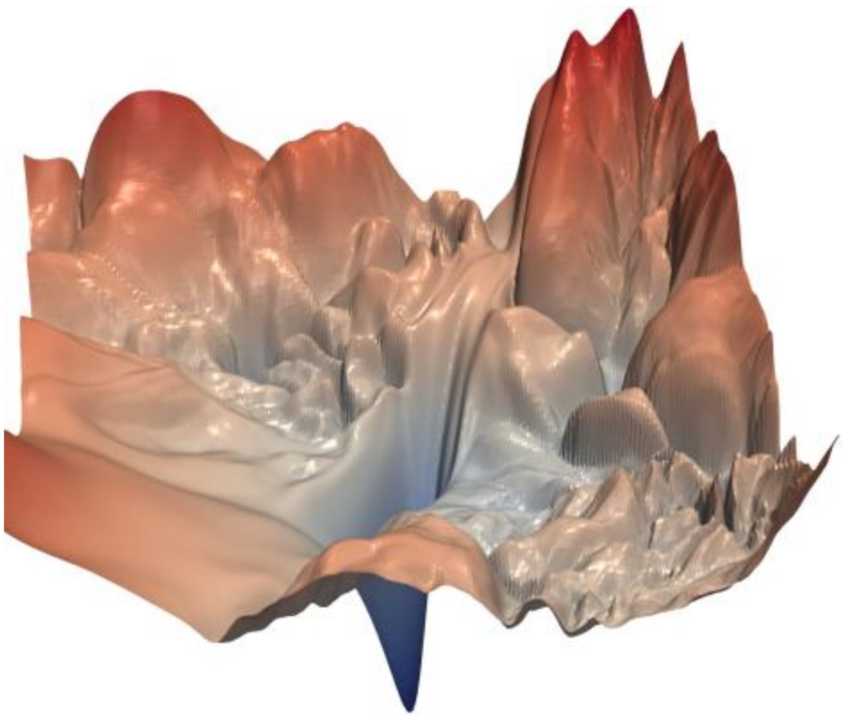
Momentum: weighted sum of the previous gradients

Consider direction

root mean square of the gradients

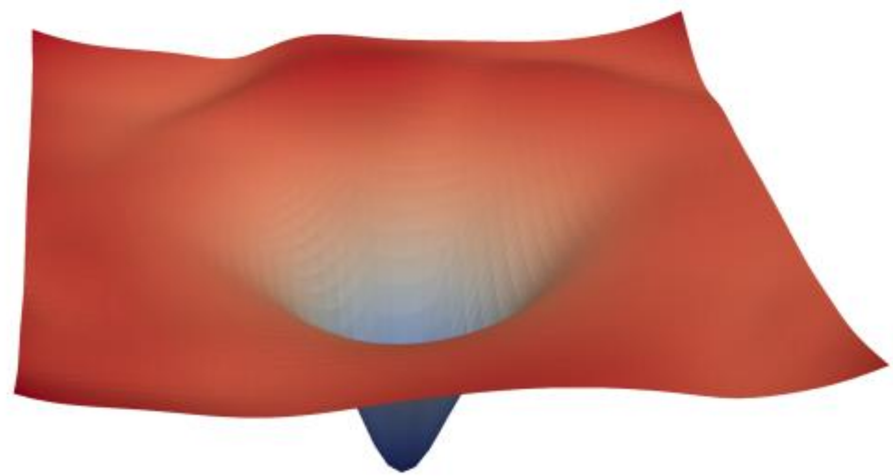
only magnitude

Next



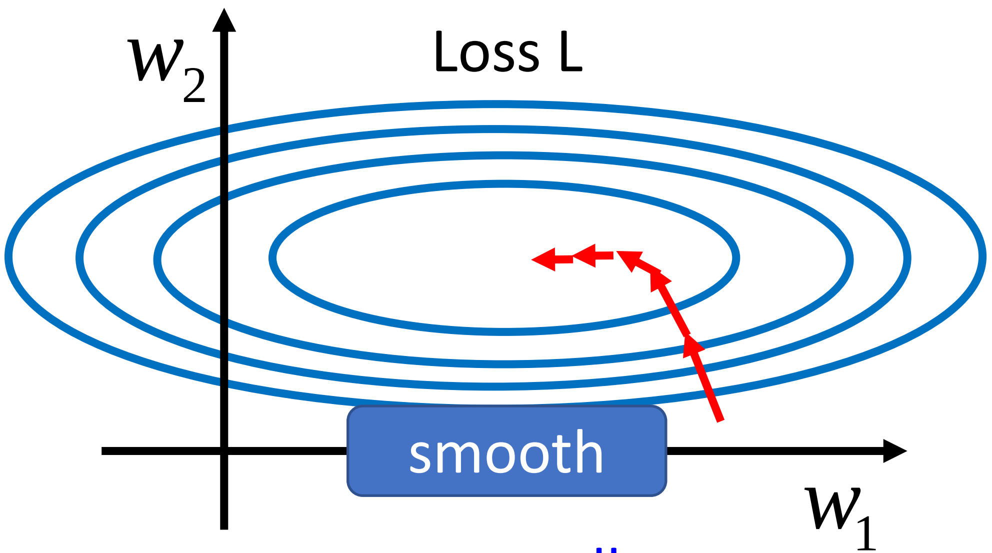
Better optimization strategies: If the mountain won't move, build a road around it.

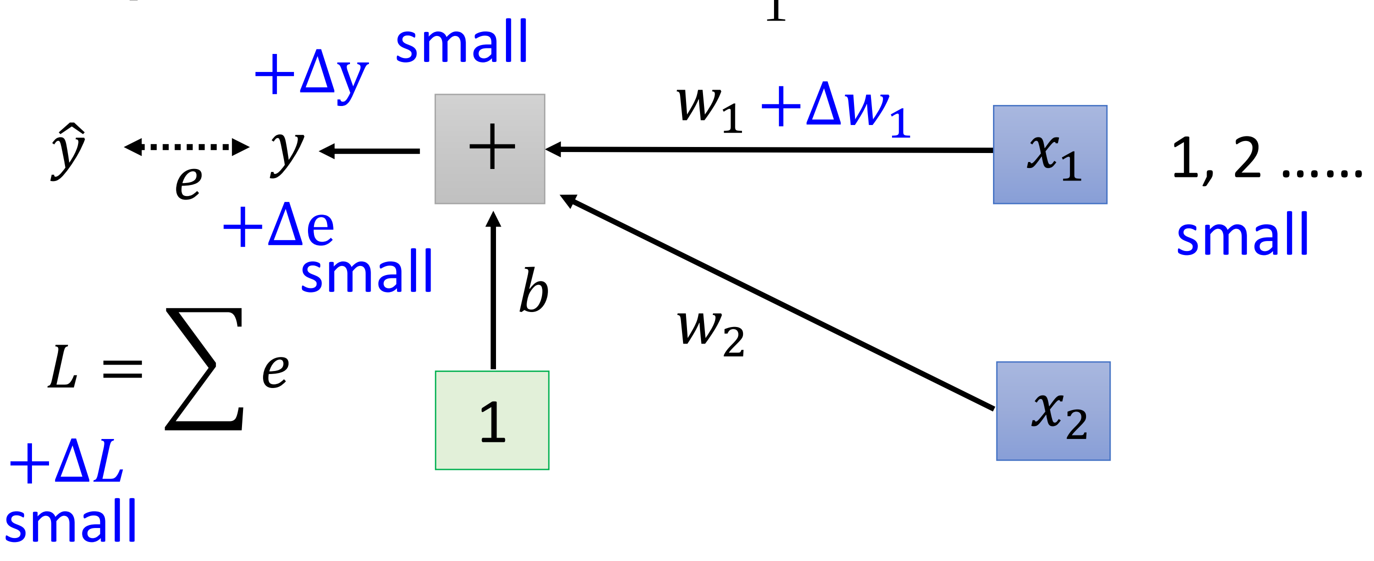
Next



Can we change the error surface? Directly move the mountain!

Changing Landscape

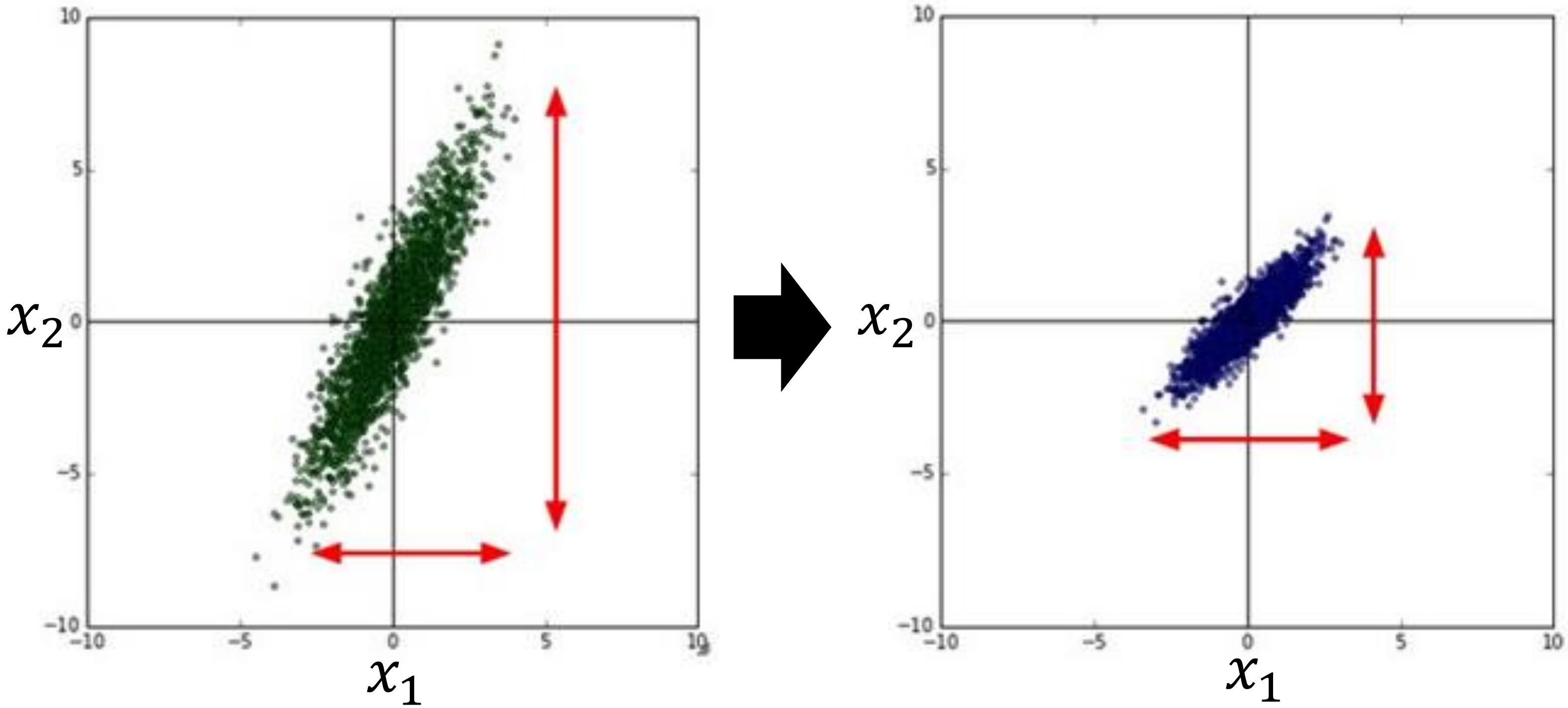




Feature Scaling

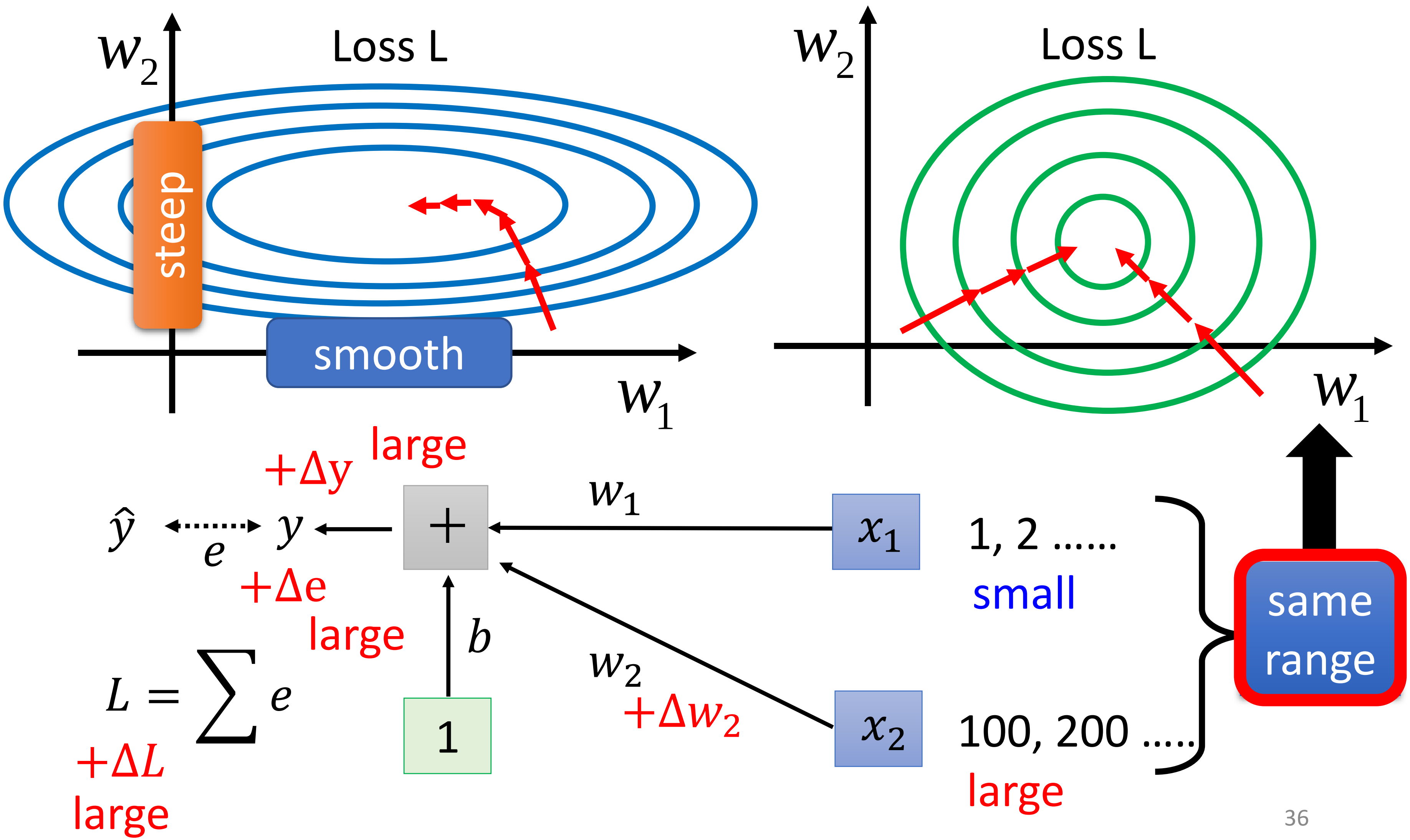
Source of figure: http://cs231n.github.io/neural-networks-2/

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

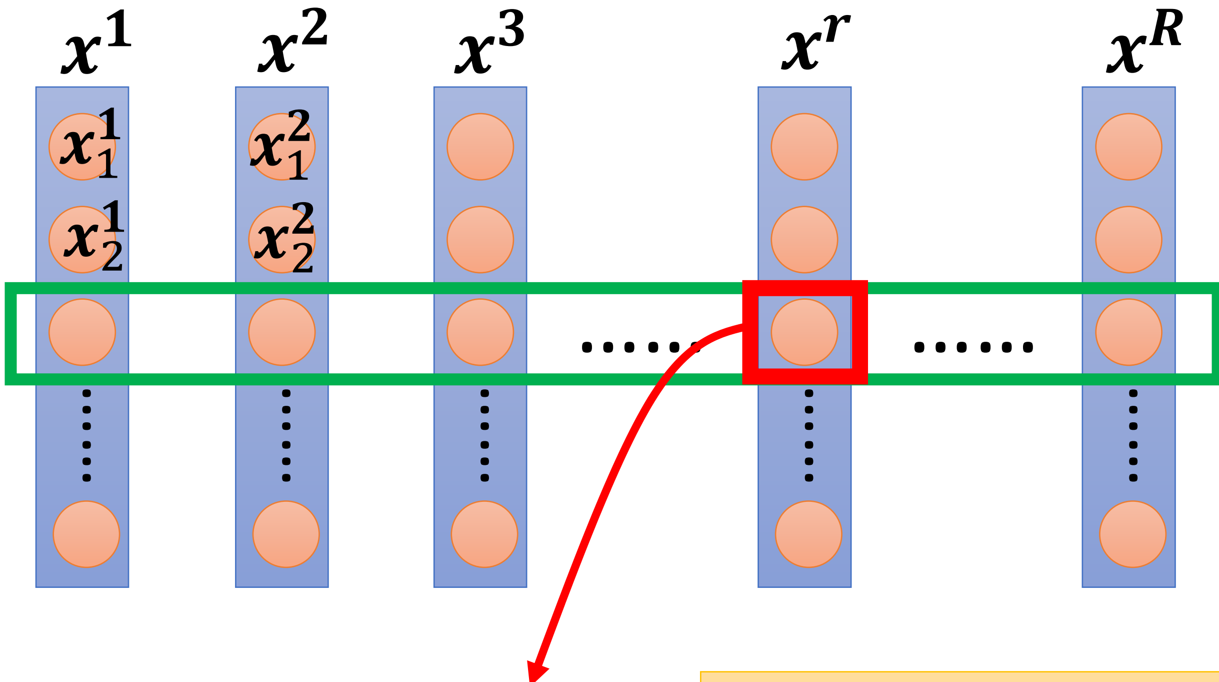


Make different features have the same scaling

Changing Landscape



Feature Normalization



For each dimension i:

mean: m_i

standard deviation: σ_i

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

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

In general, feature normalization makes gradient descent converge faster.

Gradient Descent

Theory

Question

- When solving:

$$\theta^* = \arg \min_\theta L(\theta) \quad \text{by gradient descent}$$

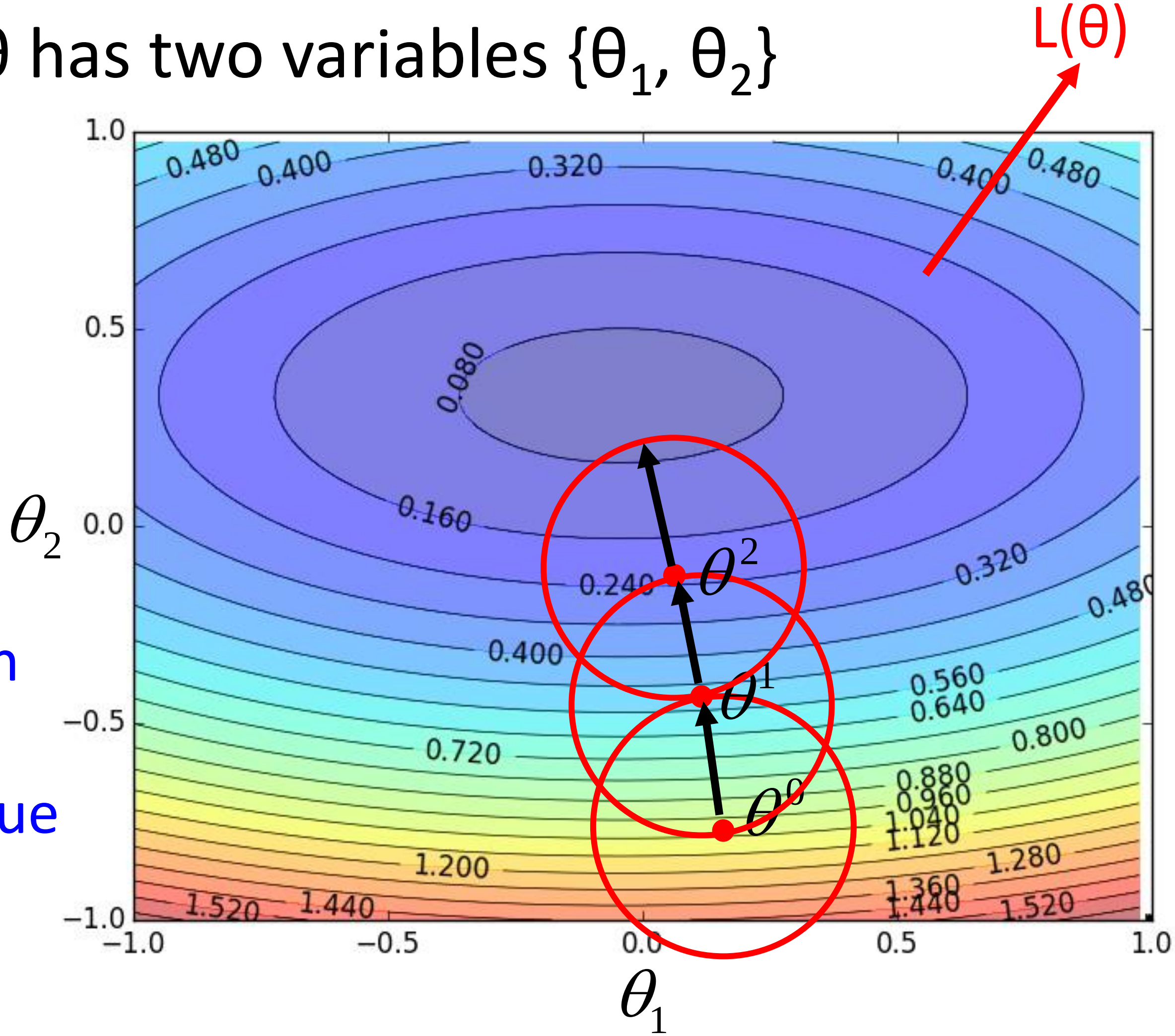
- Each time we update the parameters, we obtain θ that makes $L(\theta)$ smaller.

$$L(\theta^0) > L(\theta^1) > L(\theta^2) > \cdots$$

Is this statement correct?

Formal Derivation

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



Given a point, we can easily find the point with the smallest value nearby. How?

Taylor Series

- **Taylor series**: Let h(x) be any function infinitely differentiable around x = x_0.

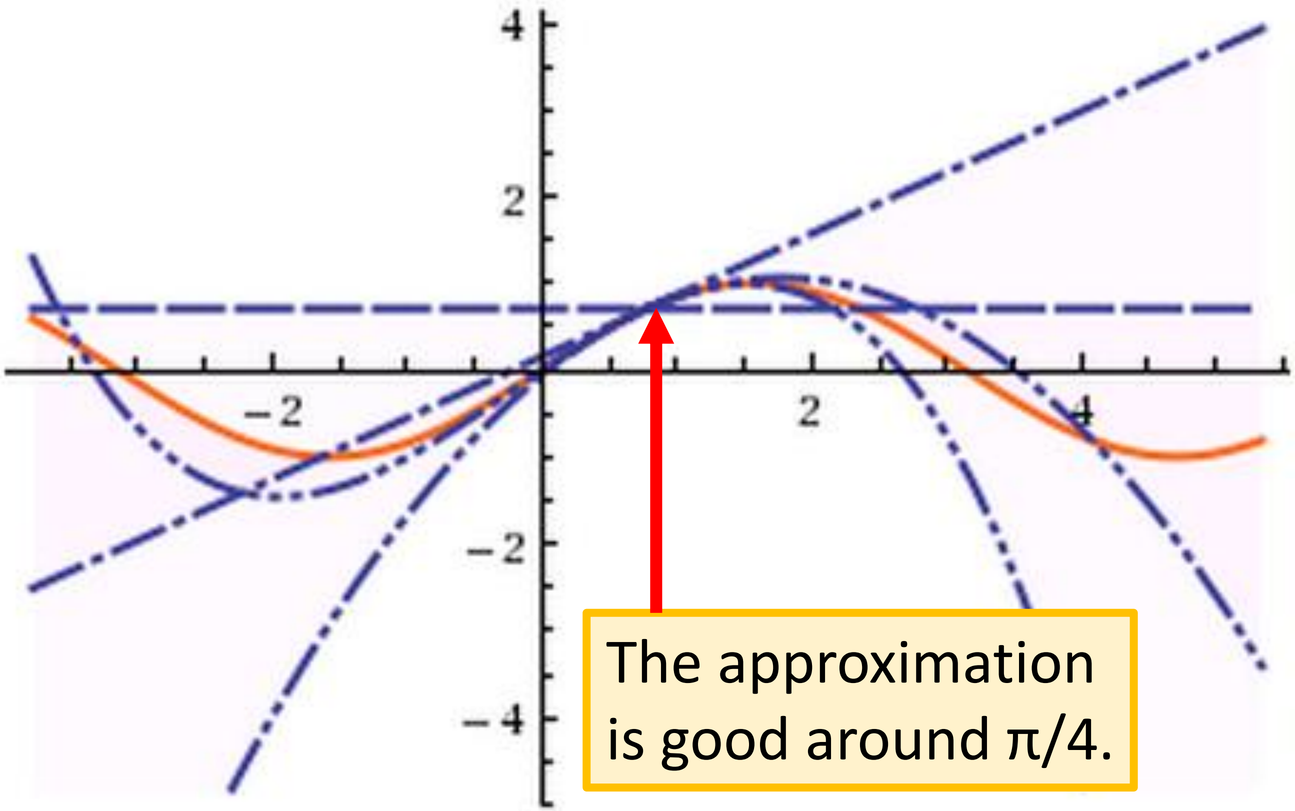
$$\mathrm{h}(\mathrm{x}) = \sum_{k=0}^{\infty} \frac{\mathrm{h}^{(k)}(x_0)}{k!}(x - x_0)^k$$

$$= h(x_0) + h'(x_0)(x - x_0) + \frac{h''(x_0)}{2!}(x - x_0)^2 + \ldots$$

When x is close to x_0 $\Rightarrow$ $h(x) \approx h(x_0) + h'(x_0)(x - x_0)$

E.g. Taylor series for h(x)=sin(x) around $x_0=\pi/4$

$$\sin(x) = \frac{1}{\sqrt{2}} + \frac{x-\frac{\pi}{4}}{\sqrt{2}} - \frac{\left(x-\frac{\pi}{4}\right)^2}{2\sqrt{2}} - \frac{\left(x-\frac{\pi}{4}\right)^3}{6\sqrt{2}} + \frac{\left(x-\frac{\pi}{4}\right)^4}{24\sqrt{2}} + \frac{\left(x-\frac{\pi}{4}\right)^5}{120\sqrt{2}} - \frac{\left(x-\frac{\pi}{4}\right)^6}{720\sqrt{2}} - \frac{\left(x-\frac{\pi}{4}\right)^7}{5040\sqrt{2}} + \frac{\left(x-\frac{\pi}{4}\right)^8}{40\,320\sqrt{2}} + \frac{\left(x-\frac{\pi}{4}\right)^9}{362\,880\sqrt{2}} - \frac{\left(x-\frac{\pi}{4}\right)^{10}}{3\,628\,800\sqrt{2}} + \ldots\ldots$$



Multivariable 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 is close to x_0 and y_0

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

Back to Formal Derivation

Based on Taylor Series:

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

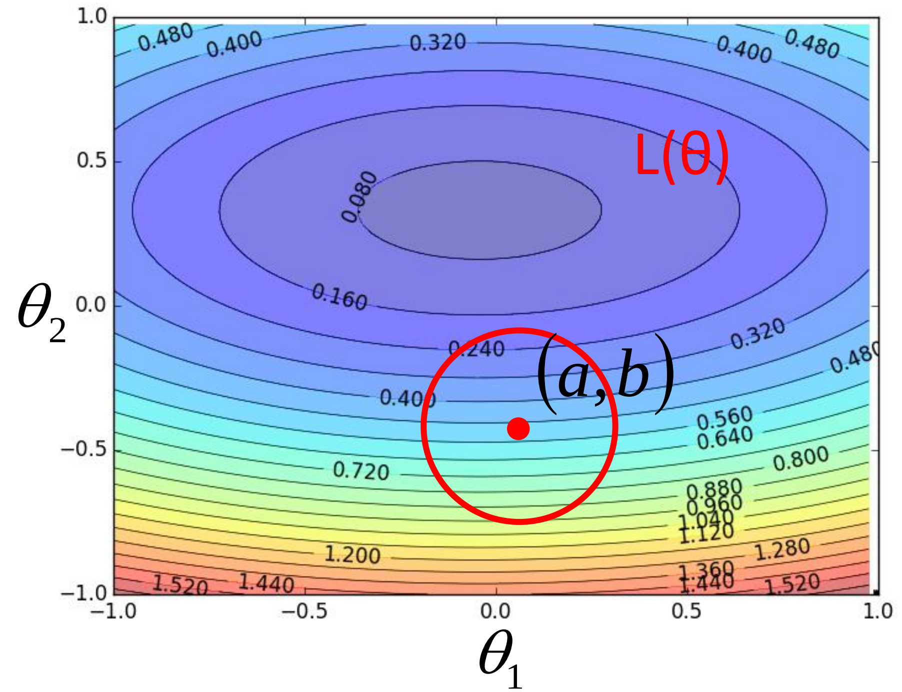
$$\mathrm{L}(\theta) \approx \mathrm{L}(a,b) + \frac{\partial \mathrm{L}(a,b)}{\partial \theta_1}(\theta_1 - a) + \frac{\partial \mathrm{L}(a,b)}{\partial \theta_2}(\theta_2 - b)$$

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

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

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

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



Back to Formal Derivation

Based on Taylor Series:

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

constant

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

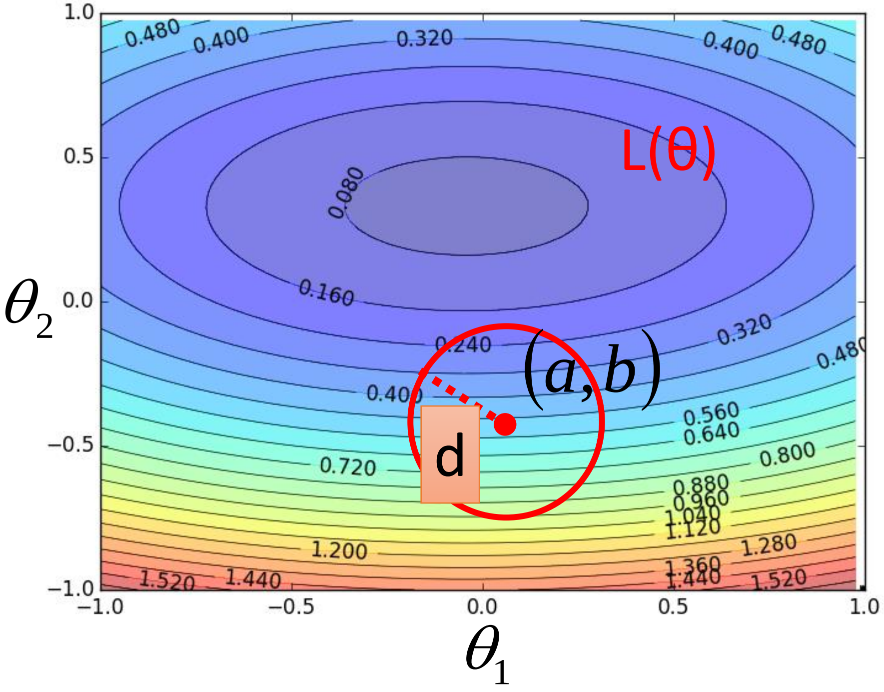
$$\mathrm{L}(\theta) \approx s + u(\theta_1 - a) + v(\theta_2 - b)$$

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

Find θ_1 and θ_2 in the red circle ***minimizing*** L(θ)

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

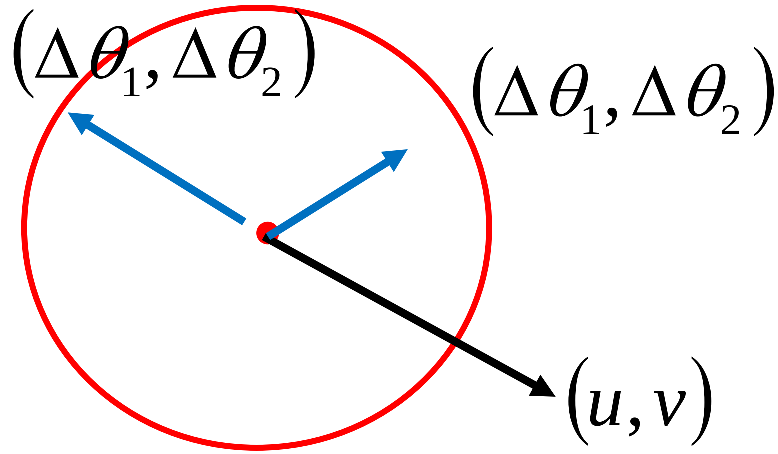
Simple, right?



Gradient descent – two variables

Red Circle: (If the radius is small)

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



Find θ_1 and θ_2 in the red circle ***minimizing*** L(θ)

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

To minimize L(θ)

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

Back to Formal Derivation

constant

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

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

Based on Taylor Series:

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

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

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.