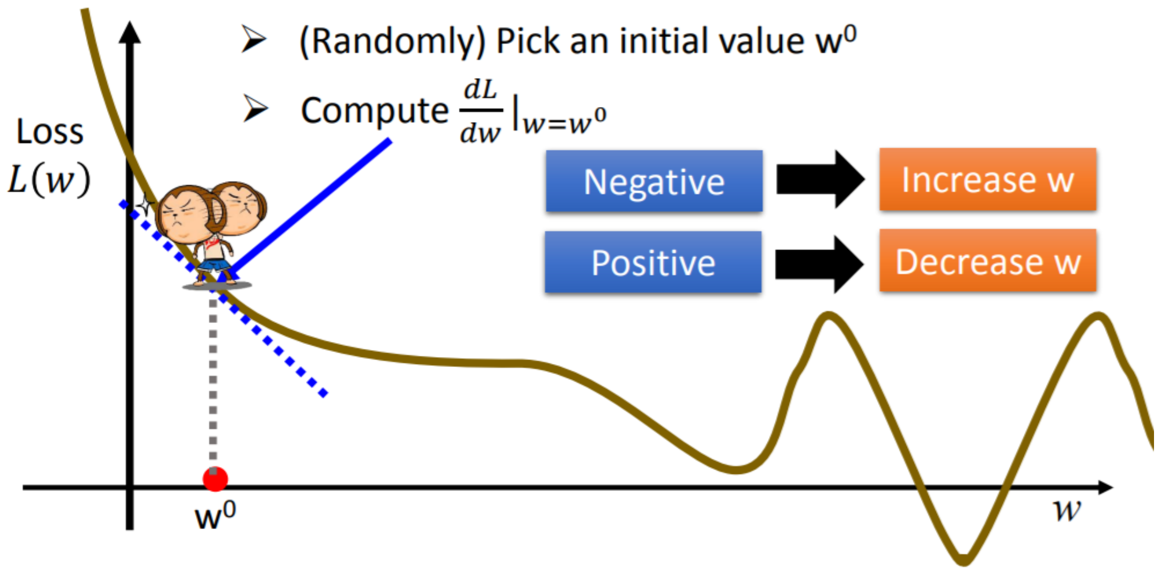
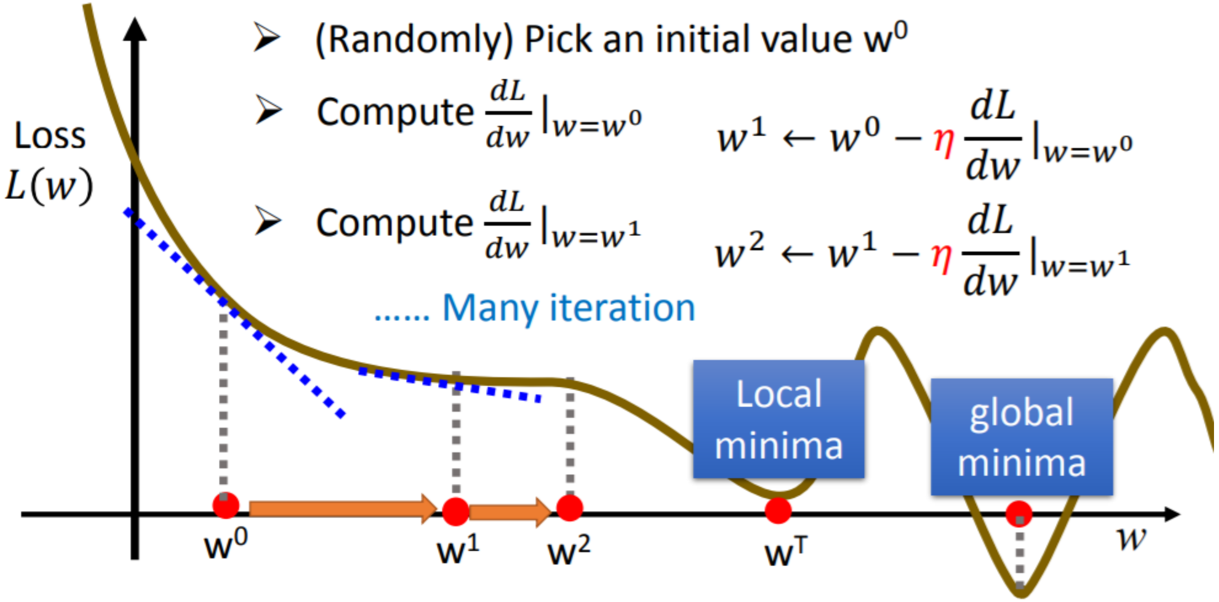
Review – How machine learns from data?

- Define a function to be learned: $y = f(x)$
- Define a loss function $\mathcal{L}$ to describe the error between $\hat{y} = f(x)$ and y
- $\boxed{\text{Find the optimal parameters of } f \text{ that minimize } \mathcal{L}}$

Review – Gradient descent to find optimal parameters that minimize the loss





Reference: 李弘毅 ML Lecture 1 https://youtu.be/CXgbekl66jc

Gradient descent to find optimal parameters that minimize the loss

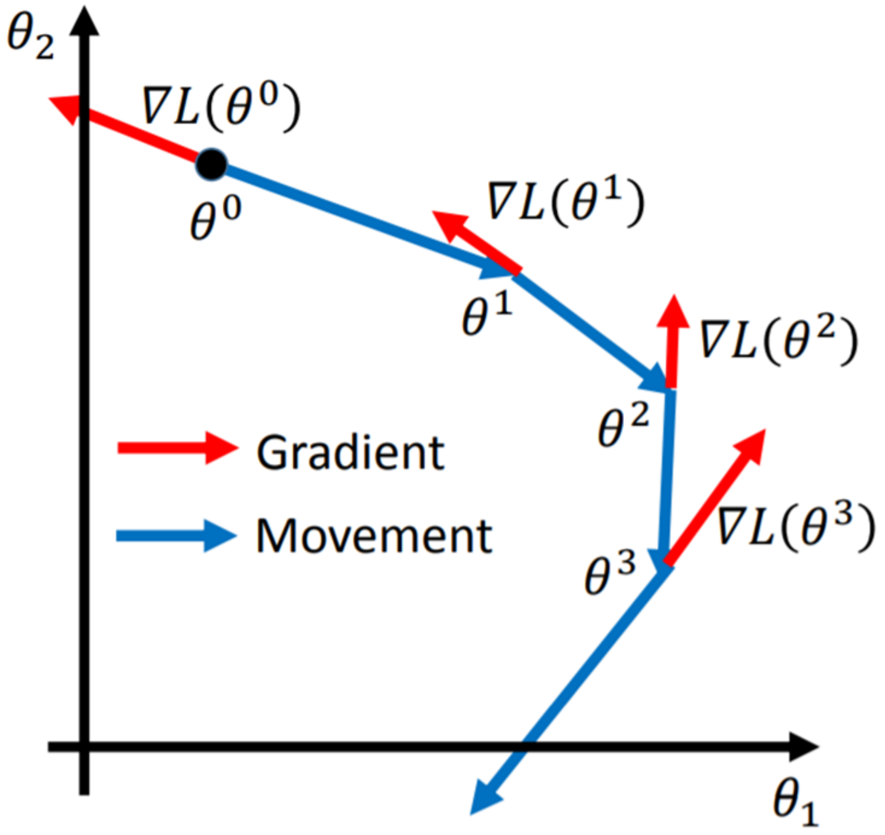
Suppose that θ has two variables $\{\theta_1, \theta_2\}$

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

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

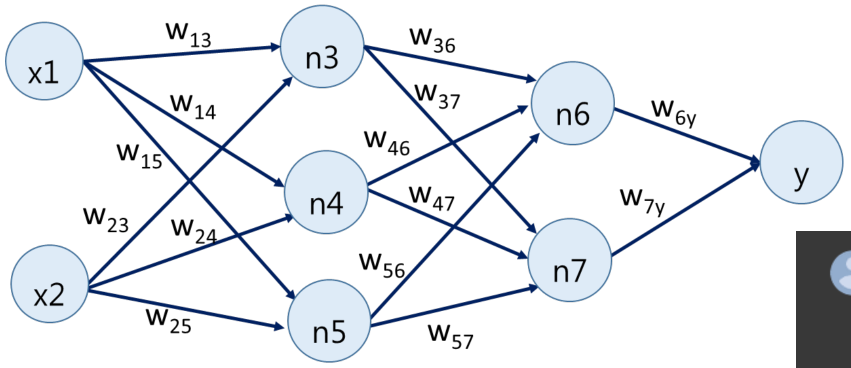
$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^0)/\partial \theta_1 \\ \partial L(\theta_2^0)/\partial \theta_2 \end{bmatrix} \Rightarrow \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

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



Reference: 李弘毅 ML Lecture 3-1 https://youtu.be/yKKNr-QKz2Q

Review – Practice gradient calculation



$$\nabla L = [\frac{\partial L}{w_{13}}, \frac{\partial L}{w_{14}}, \frac{\partial L}{b_3}, .., \frac{\partial L}{w_{6y}}, ..]$$

```python
for name, param in MyNet.named_parameters():
    if param.requires_grad:
        print(name, '\n', param.data, '\n', param.grad)
```

```
0.weight
 tensor([[ 0.4343,  0.6134],
        [-0.2977,  0.1472],
        [-0.3772, -0.2806]])
 None
0.bias
 tensor([ 0.2081, -0.5165,  0.3692])
 None
1.weight
 tensor([[ 0.5137,  0.5441,  0.4459],
        [-0.0712,  0.0366, -0.2097]])
 None
1.bias
 tensor([0.2730, 0.5439])
 None
2.weight
 tensor([[0.5989, 0.3610]])
 None
2.bias
 tensor([-0.3608])
 None
```

```
0.weight
 tensor([[ 0.4343,  0.6134],
        [-0.2977,  0.1472],
        [-0.3772, -0.2806]])
 tensor([[-80.2851, -46.4245],
        [-96.5485, -55.8287],
        [-54.4879, -31.5074]])
0.bias
 tensor([ 0.2081, -0.5165,  0.3692])
 tensor([-10.9224, -13.1350,  -7.4128])
1.weight
 tensor([[ 0.5137,  0.5441,  0.4459],
        [-0.0712,  0.0366, -0.2097]])
 tensor([[-139.3767,   48.2291,   83.4421],
        [ -84.0035,   29.0681,   50.2912]])
1.bias
 tensor([0.2730, 0.5439])
 tensor([-23.2005, -13.9832])
2.weight
 tensor([[0.5989, 0.3610]])
 tensor([[-24.1840, -30.7693]])
2.bias
 tensor([-0.3608])
 tensor([-38.7382])
```

Reference: 許洺嘉(1071414)

Review – Practice gradient calculation

$$Loss\ function\ (MSE):\quad \mathcal{L} = (y - \boxed{\hat{y}})^2$$

$$\hat{y} = w_{6y}n_6 + w_{7y}n_7$$

$$\therefore\ \mathcal{L} = (y - w_{6y}n_6 + w_{7y}n_7)^2$$

$$\therefore\ \nabla\mathcal{L}(w_6) = \frac{\partial\mathcal{L}}{\partial w_{6y}} = \frac{\partial\mathcal{L}}{\partial\hat{y}}\frac{\partial\hat{y}}{\partial w_{6y}}\ (by\ chain\ rule)$$

$$\boxed{\frac{\partial\mathcal{L}}{\partial\hat{y}} = 2(y - \hat{y})\cdot(-1)}\qquad\boxed{\frac{\partial\hat{y}}{\partial w_{6y}} = n_6}$$

$$\therefore\ \nabla\mathcal{L}(w_6) = 2(y - \hat{y})\cdot(-1)\cdot n_6$$

n6	n7
0.69657858	0.51114952
0.859582	0.56898564
0.543156	0.90834304

W6Y	0.5989
W7Y	0.361

X*W	+	BY	=	Y_hat	Y
0.601705888		-0.3608		0.24090589	7
0.720207476		-0.3608		0.35940748	12
0.653207966		-0.3608		0.29240797	40

GradientsPerData
-9.416480357
-20.01208761
-43.13483372

Gradient_n6
-24.1878006

Reference: 許洺嘉(1071414)

More details about gradient descent

```
# initialize NN weights
for name, param in MyNet.named_parameters():
  if(param.requires_grad):
    torch.nn.init.normal_(param, mean=0.0, std=0.02)
loss_func = torch.nn.MSELoss()
optimizer = torch.optim.Adam(MyNet.parameters(), lr=0.0003)

# train NN
print("epoch", end=": ")
epoch_lossLst=[]
for epoch in range(1, 700):
  if(epoch%100 == 0):
    print(epoch, end=",")
  for (batchX, batchY) in loader:
    batchY_hat = MyNet(batchX)
    loss = loss_func(batchY_hat, batchY)
    epoch_lossLst.append(float(loss))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Fixed or adaptive lr ?

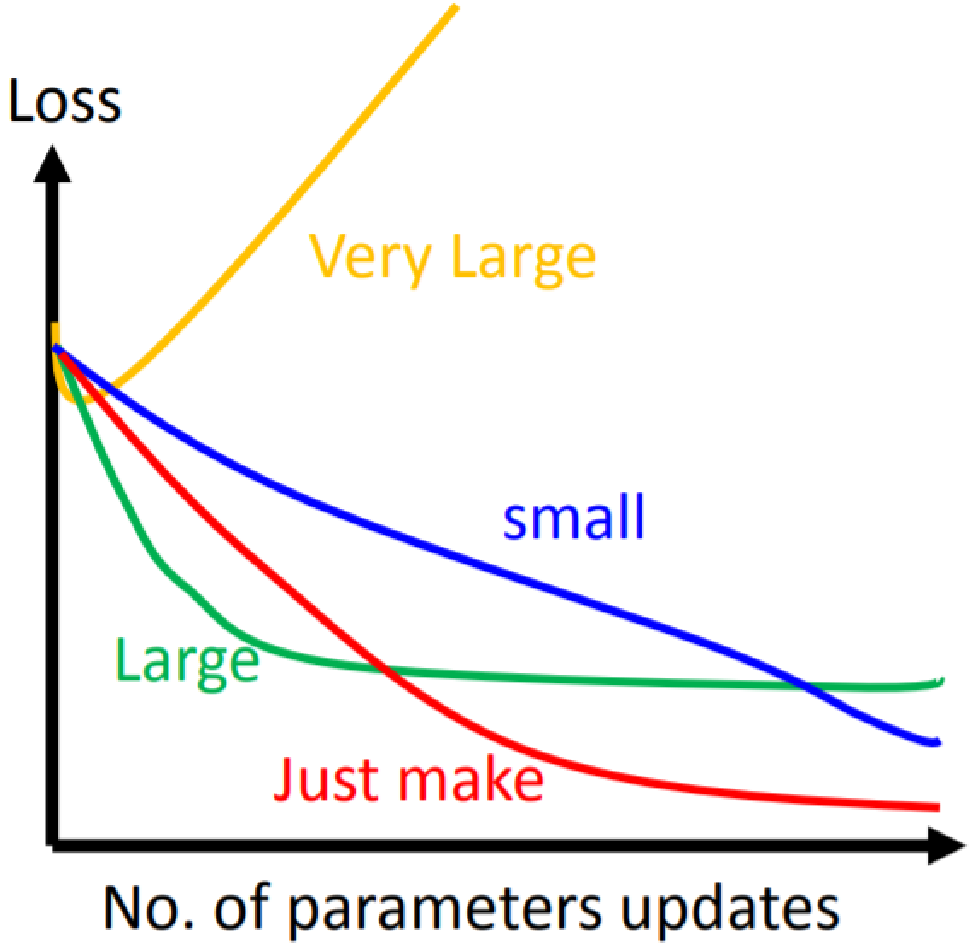
Give different parameters different lr?

Why mini-batch?

Learning rates

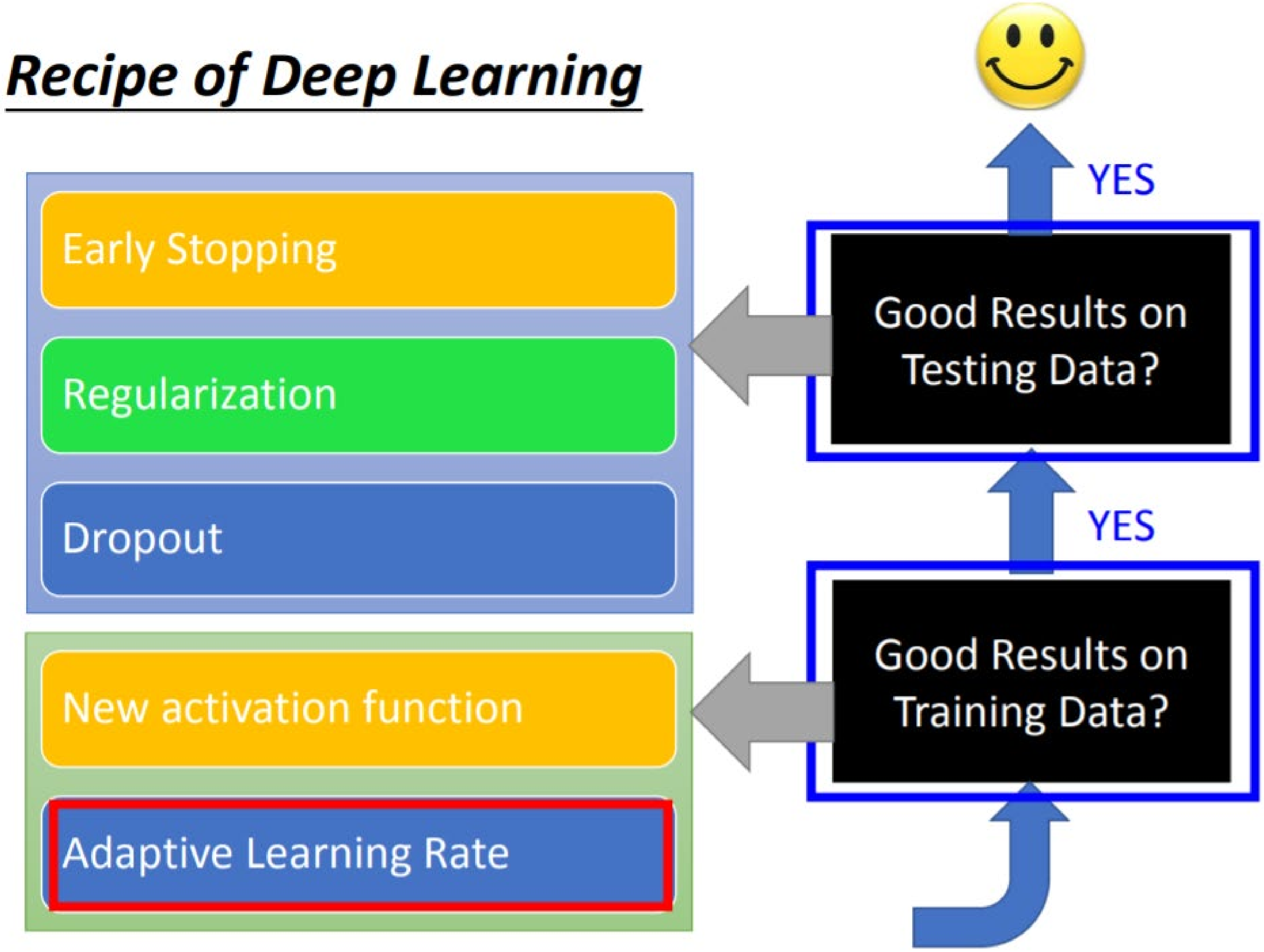
3. Gradient descent.ipynb

- 0.5
- 0.05
- 0.0003



Reference: 李弘毅 ML Lecture 3-1
https://youtu.be/yKKNr-QKz2Q

Adaptive learning rates



Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

Learning rate schedule

- 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

3. Gradient descent.ipynb

Adagrad

- Give different parameters different learning rates.
- Divide the learning rate of each parameter by the root mean square of its previous derivatives

η^t : adaptive lr

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

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

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

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

$$\vdots$$

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

Reference: 李弘毅 ML Lecture 3-1 https://youtu.be/yKKNr-QKz2Q

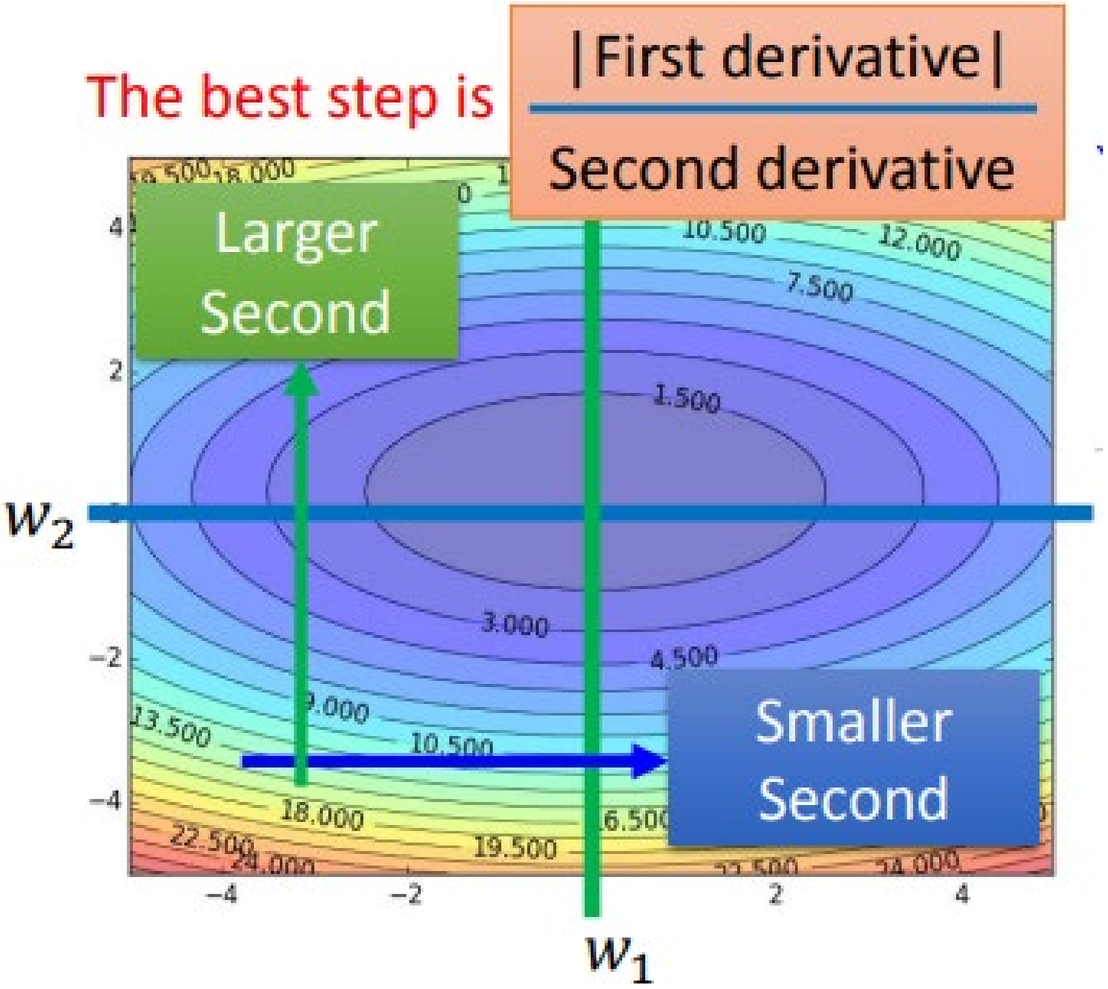
Why divided by root mean square of previous derivatives?

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

Use first derivative to estimate second derivative

3. Gradient descent.ipynb

- Adagrad

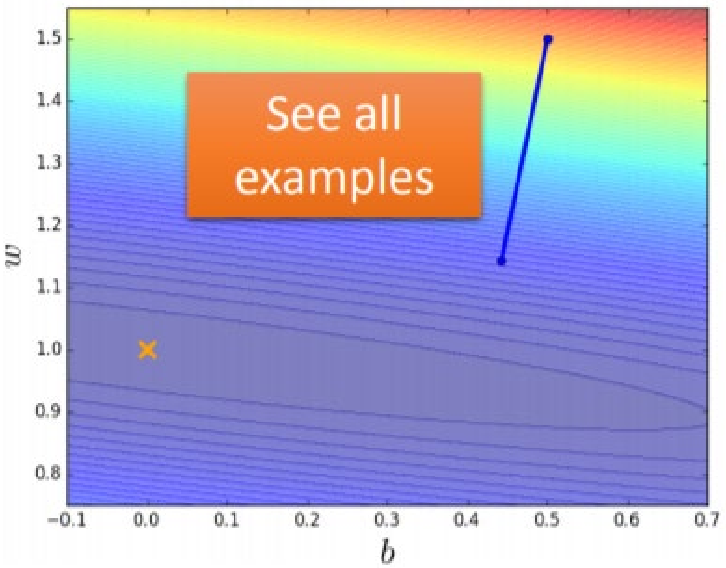


Stochastic Gradient Descent (SGD)

- Make the training faster
- Loss for only one sample

Gradient Descent

Update after seeing all examples $\theta^t = \theta^{t-1} - \eta \nabla L(\theta^{t-1})$



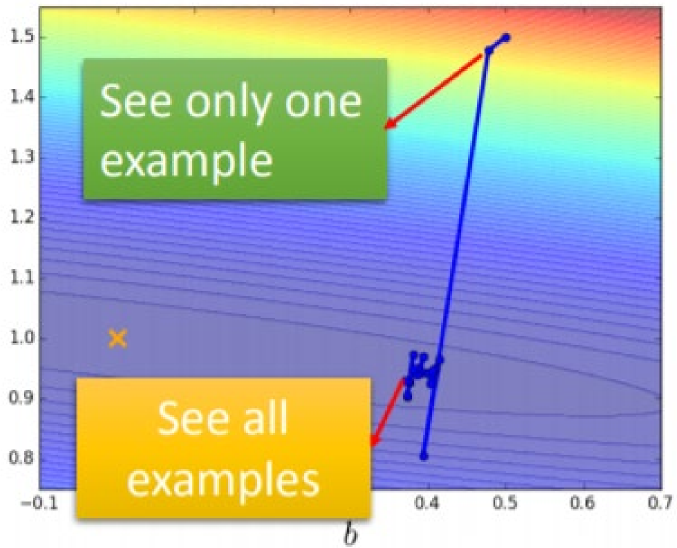
$$\nabla L(\theta) = \begin{bmatrix} \partial L / \partial w_1 \\ \vdots \\ \partial L / \partial w_K \end{bmatrix}$$

Stochastic Gradient Descent

Update for each example

If there are 20 examples, 20 times faster.

$\theta^t = \theta^{t-1} - \eta \nabla L^n (\theta^{t-1})$



$$\nabla L(\theta) = \begin{bmatrix} \partial L^n / \partial w_1 \\ \vdots \\ \partial L^n / \partial w_K \end{bmatrix}$$

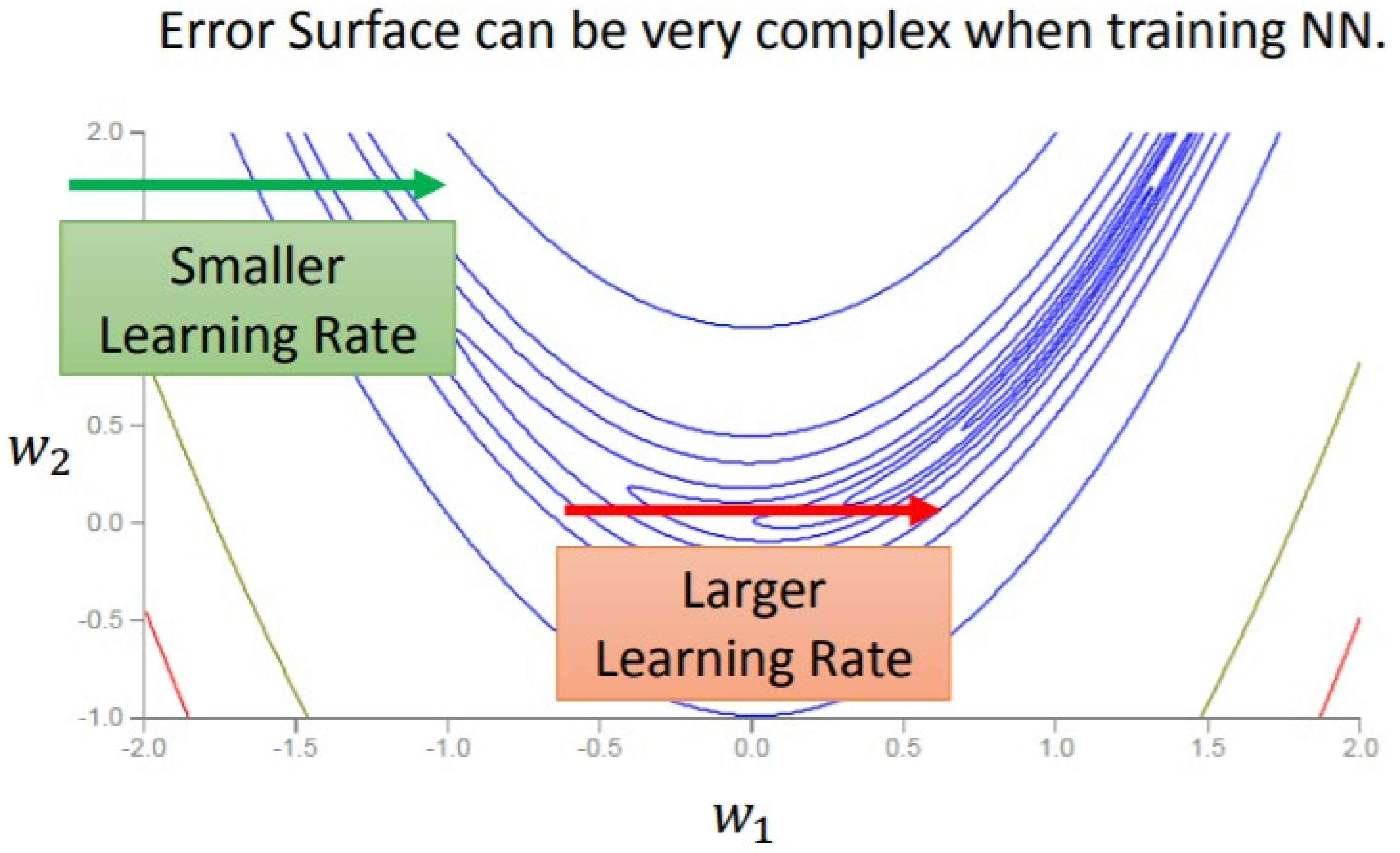
Reference: 李弘毅 ML Lecture 3-1 https://youtu.be/yKKNr-QKz2Q

SGD

3. Gradient descent.ipynb

- SGD

RMS Prop



Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

RMS Prop

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

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

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

$$\vdots$$

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

Root Mean Square of the gradients
with previous gradients being decayed

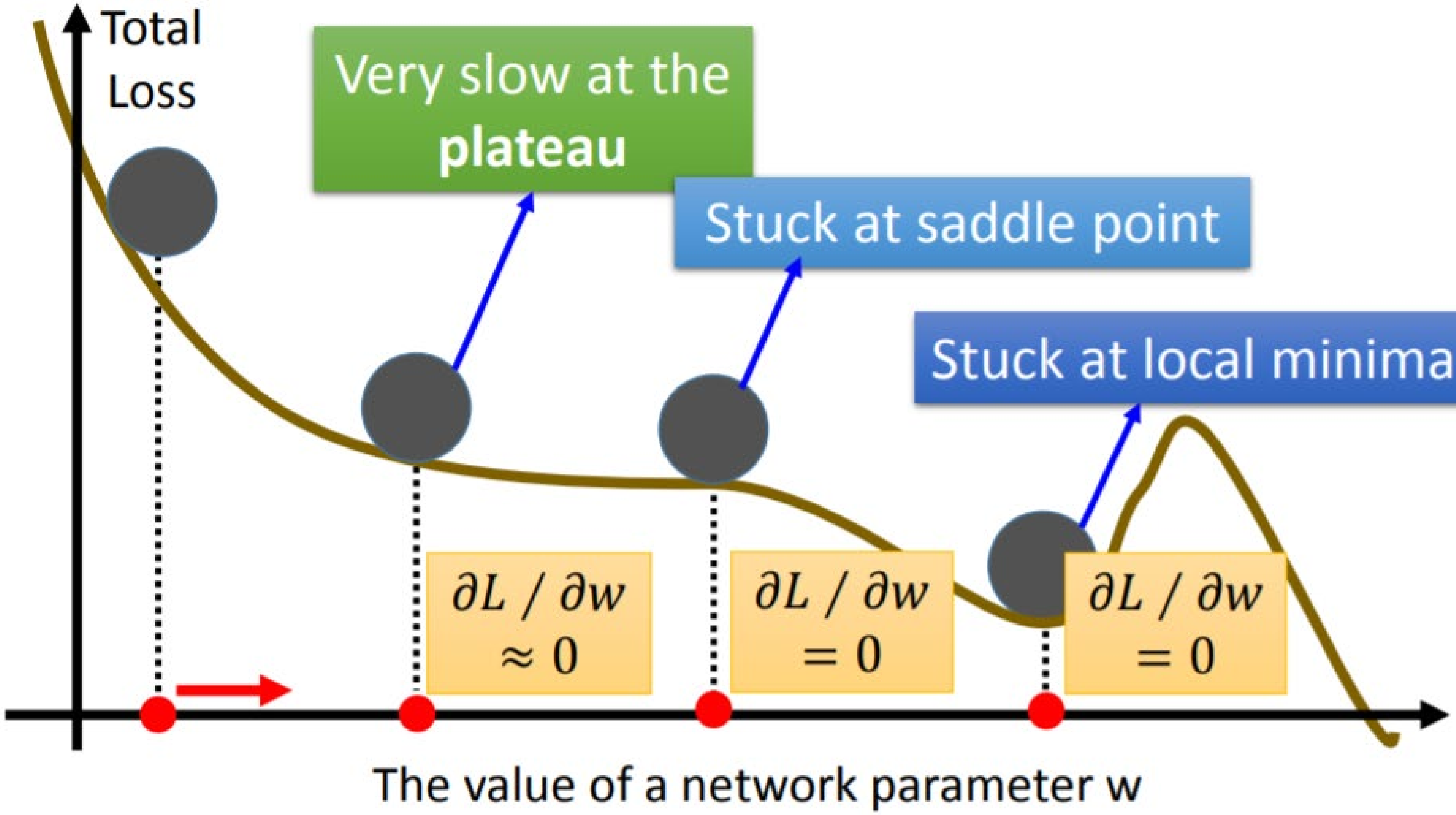
Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

RMS Prop

3. Gradient descent.ipynb

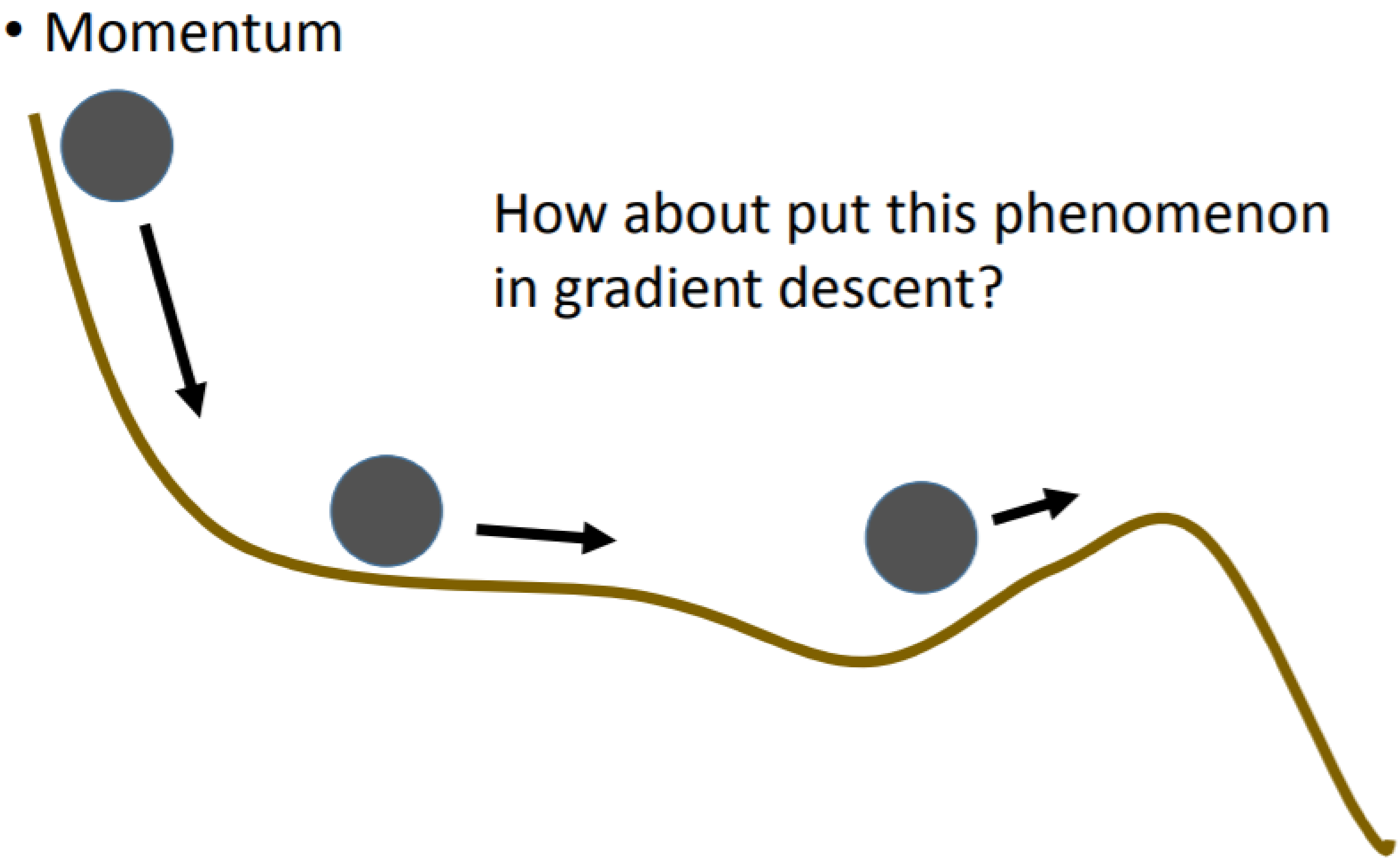
- RMS Prop

Hard to find optimal parameters



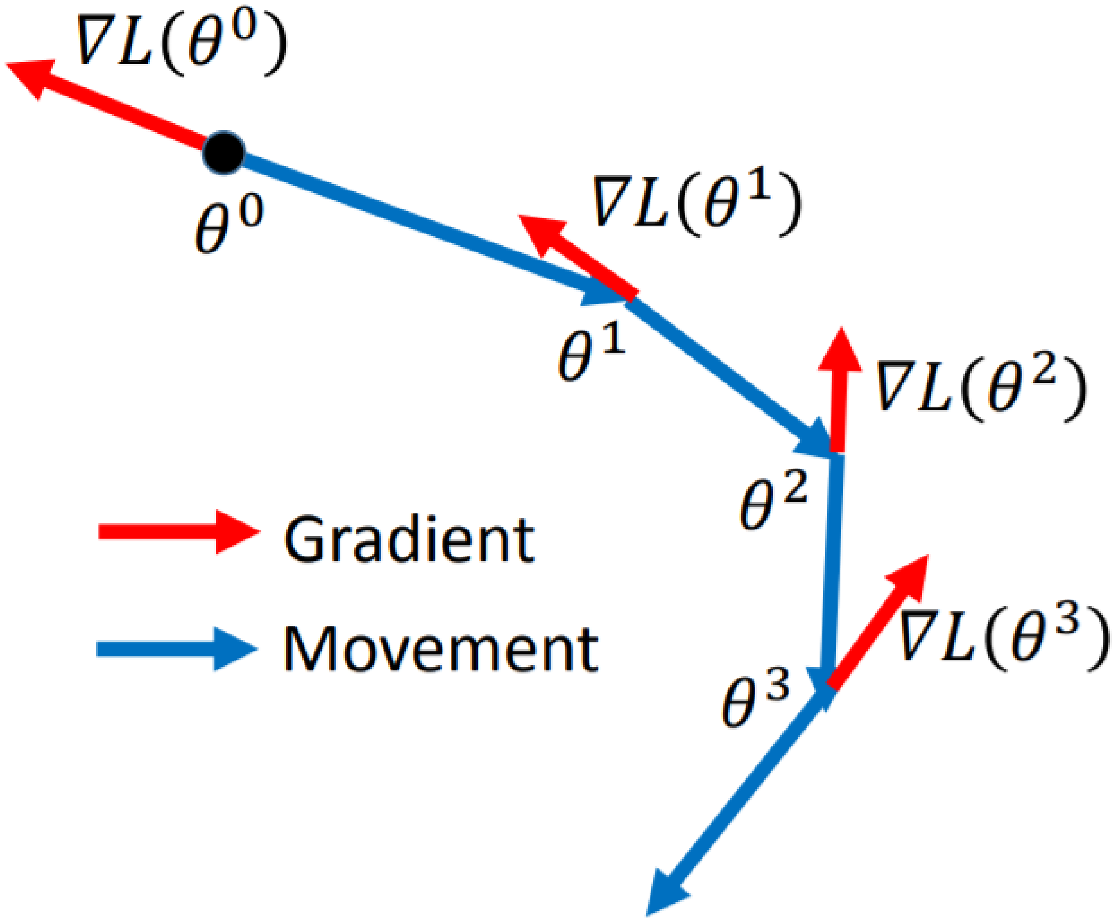
Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

Momentum



Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

Vanilla gradient decent



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

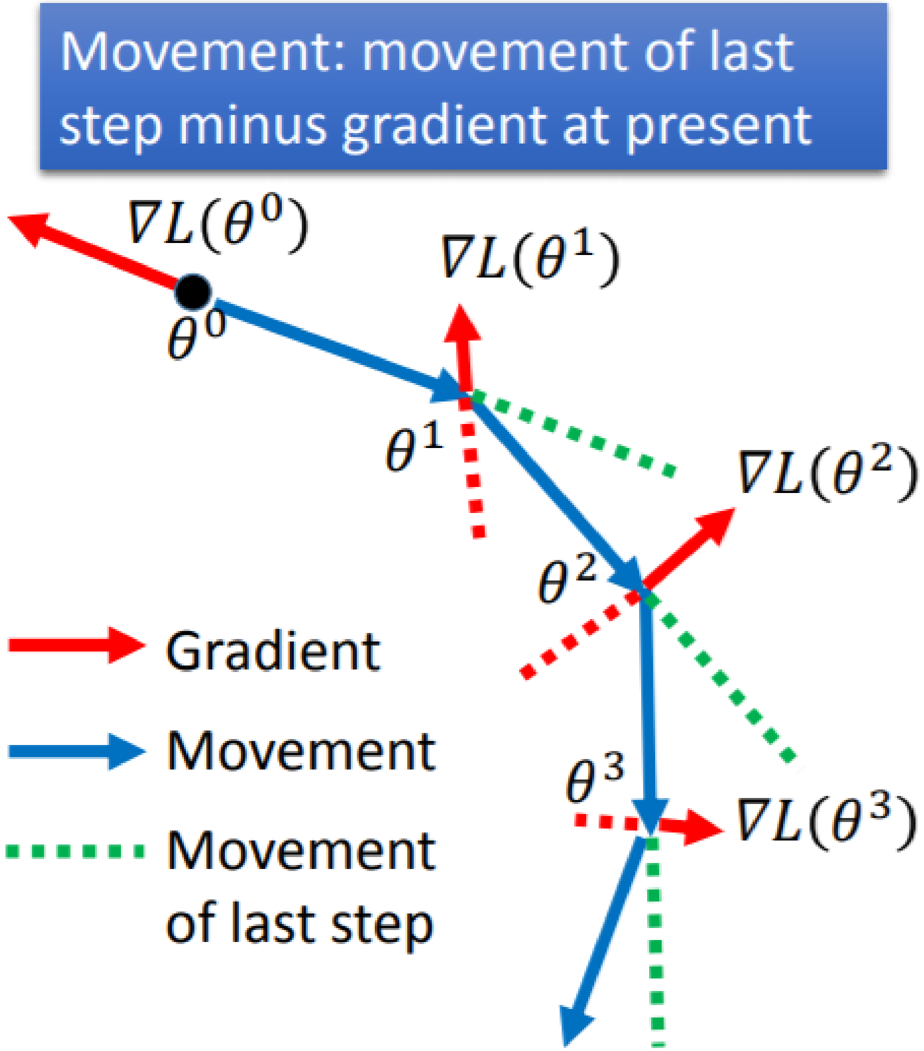
Move to $\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

⋮

Stop until $\nabla L(\theta^t) \approx 0$

Reference: 李弘毅 ML 9-1 https://youtu.be/xki61j7z-30

Momentum



Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta \nabla L(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

Movement $v^2 = \lambda v^1 - \eta \nabla L(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

Movement not just based on gradient, but previous movement.

Reference: 李弘毅 ML 9-1 https://youtu.be/xki61j7z-30

Momentum

Movement: movement of last step minus gradient at present

v^i is actually the weighted sum of all the previous gradient: $\nabla L(\theta^0)$,$\nabla L(\theta^1)$, … $\nabla L(\theta^{i-1})$

$v^0 = 0$

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

$v^2 = -\lambda\,\eta\nabla L(\theta^0) - \eta\nabla L(\theta^1)$

⋮

Start at point θ^0

Movement $v^0=0$

Compute gradient at θ^0

Movement $v^1 = \lambda v^0 - \eta\nabla L(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

Movement $v^2 = \lambda v^1 - \eta\nabla L(\theta^1)$

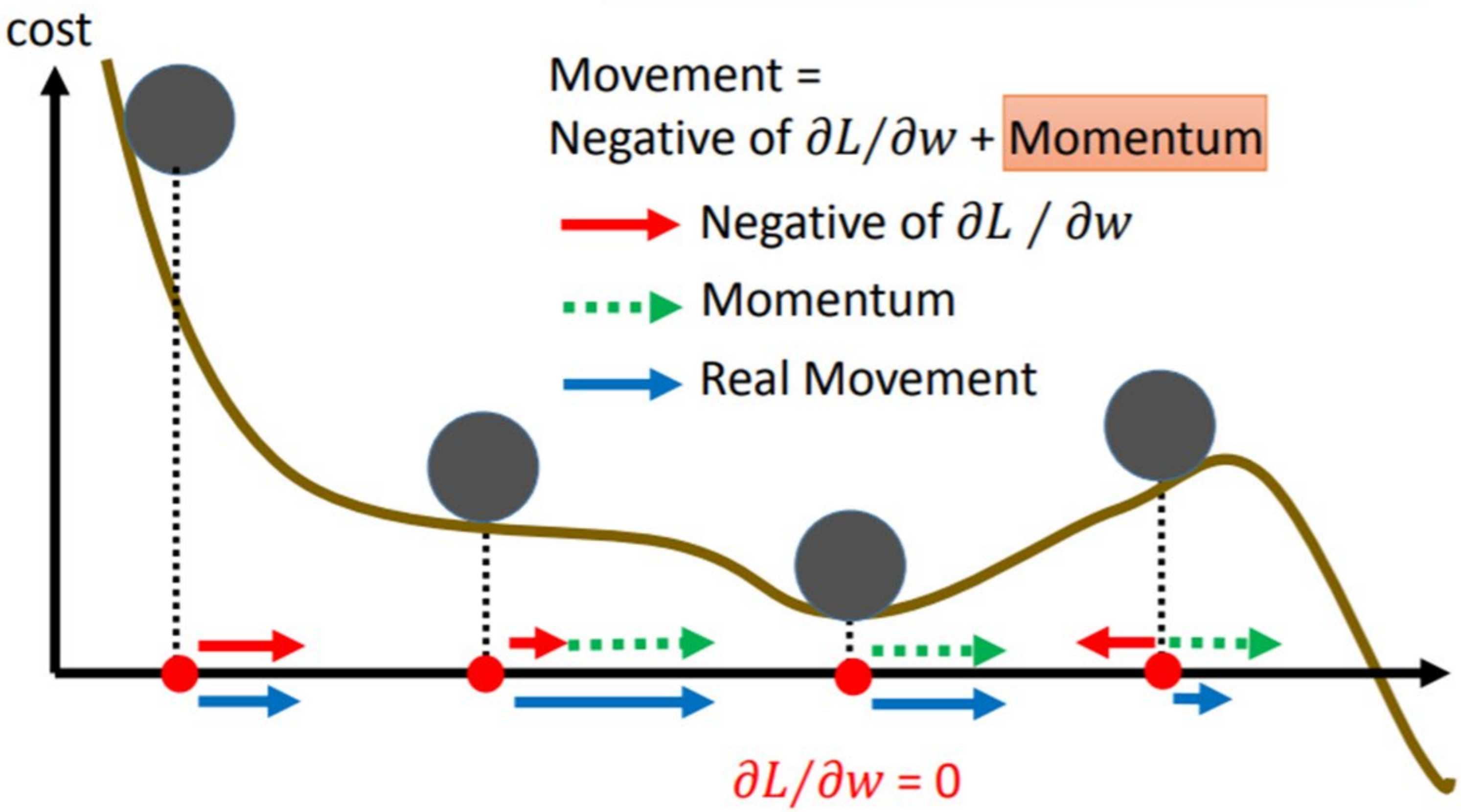
Move to $\theta^2 = \theta^1 + v^2$

Movement not just based on gradient, but previous movement

Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

Momentum

Still not guarantee reaching global minima, but give some hope ……



Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

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)

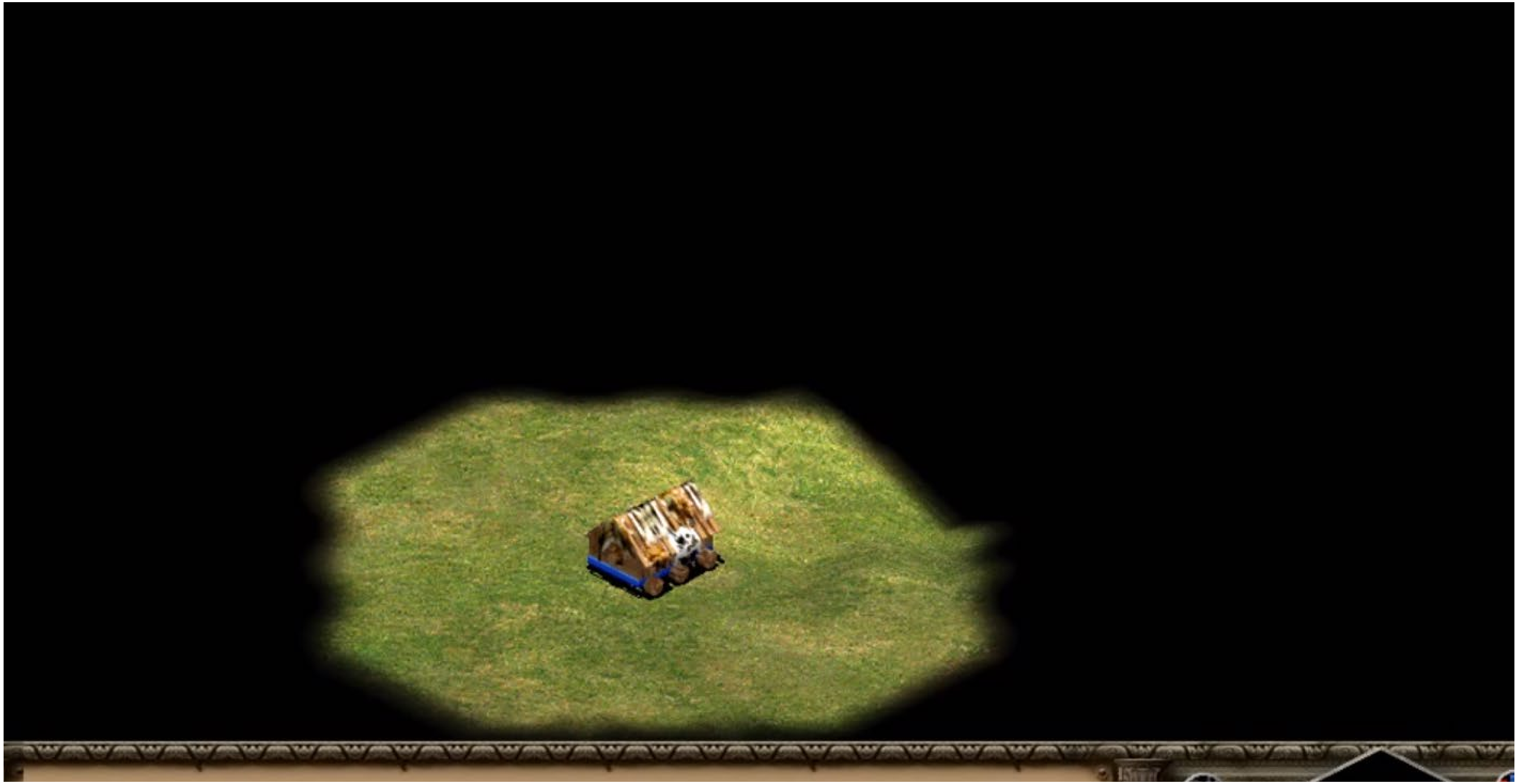
Reference: 李弘毅 ML Lecture 9-1 https://youtu.be/xki61j7z-30

Adam

3. Gradient descent.ipynb

- Adam

Why gradient decent can find local minimum?



Reference: 李弘毅 ML Lecture 3-2 https://youtu.be/1_HBTJyWgNA

Why gradient decent can find local minimum?

Based on Taylor Series:

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

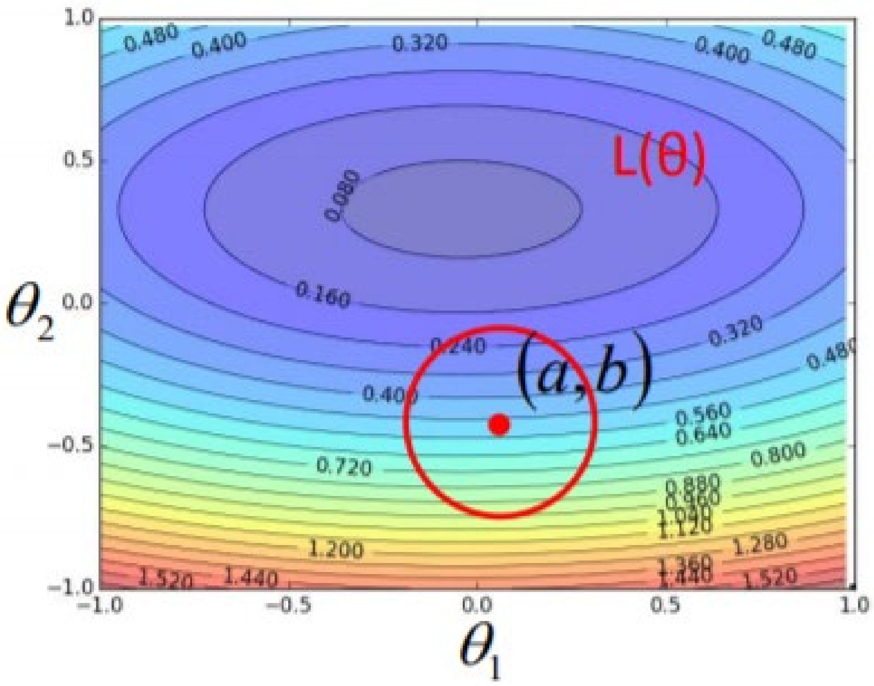
$$L(\theta) \approx L(a,b) + \frac{\partial L(a,b)}{\partial \theta_1}(\theta_1 - a) + \frac{\partial L(a,b)}{\partial \theta_2}(\theta_2 - b)$$

$$s = L(a,b)$$

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

$$L(\theta)$$

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

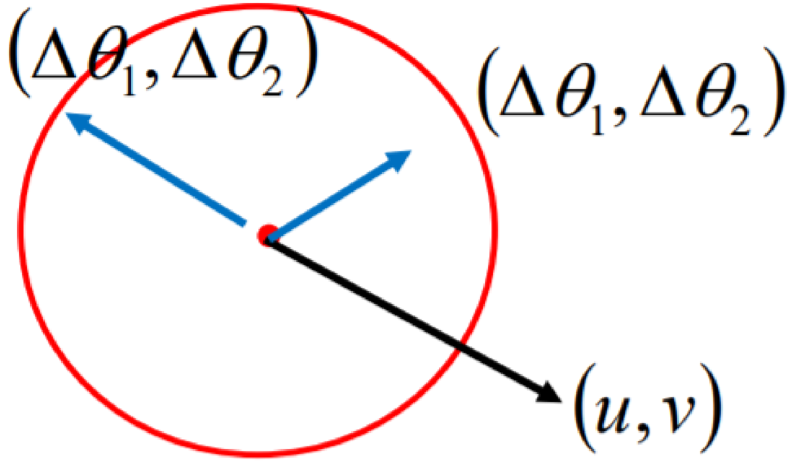


Reference: 李弘毅 ML Lecture 3-1 https://youtu.be/yKKNr-QKz2Q

Why gradient decent can find local minimum?

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 $L(\theta)$

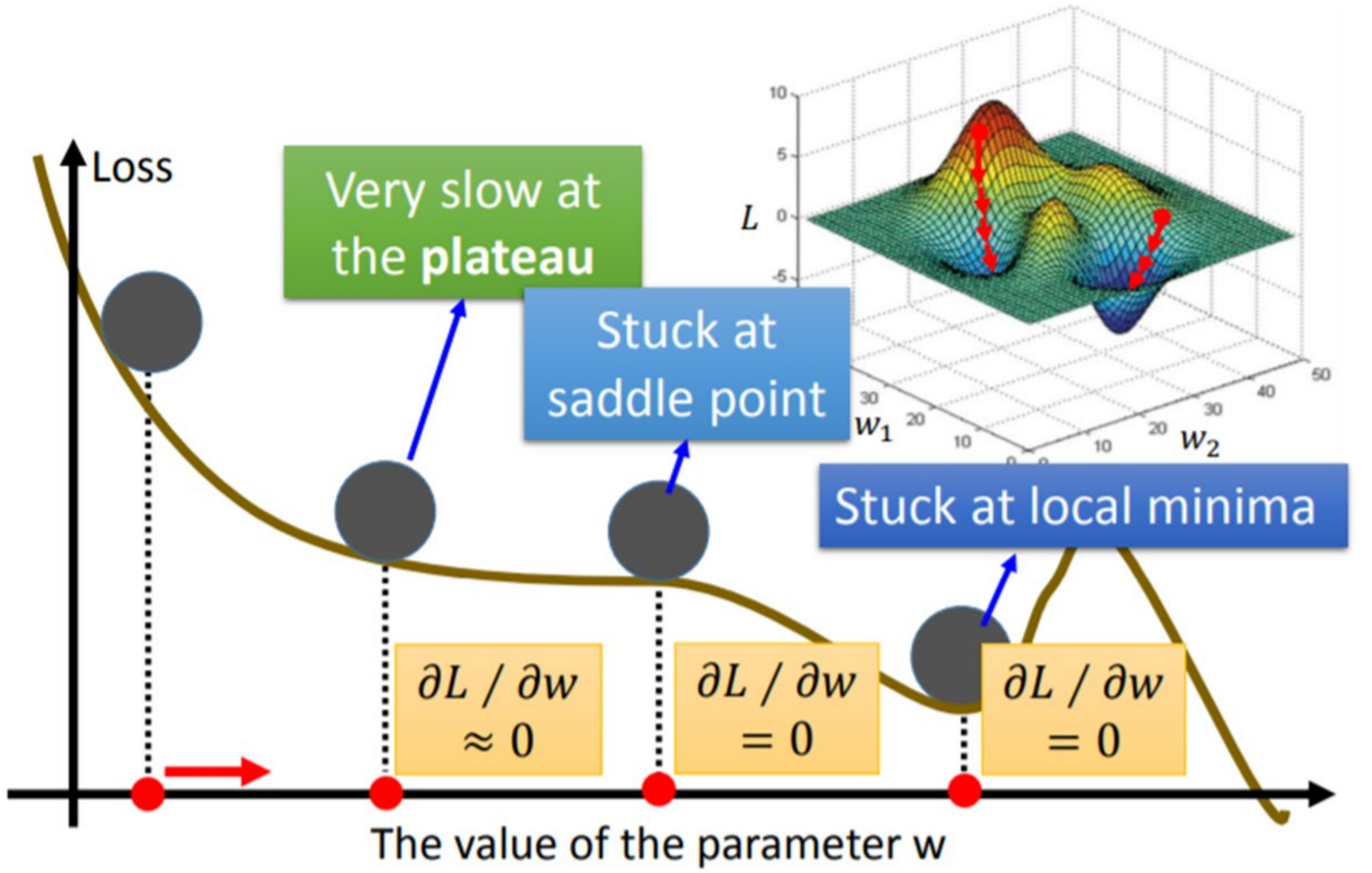
$$\underbrace{(\theta_1 - a)}_{\Delta\theta_1}^2 + \underbrace{(\theta_2 - b)}_{\Delta\theta_2}^2 \le d^2$$

To minimize $L(\theta)$

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

Reference: 李弘毅 ML Lecture 3-1 https://youtu.be/yKKNr-QKz2Q

Limitation of gradient decent

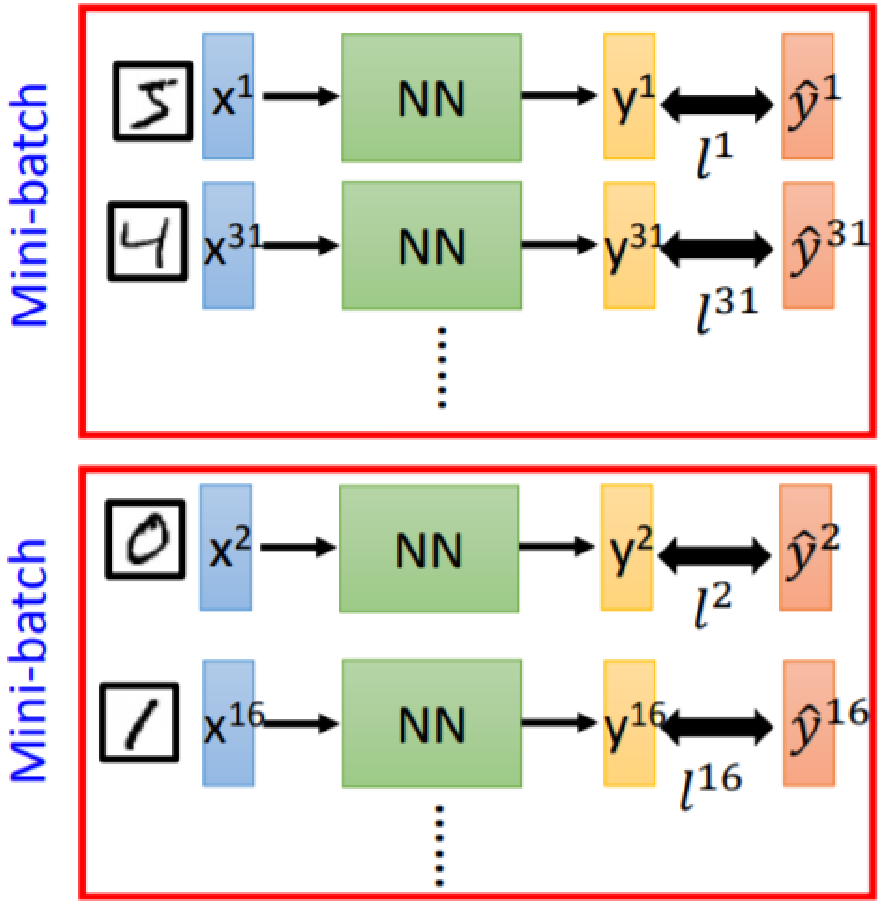


Reference: 李弘毅 ML Lecture 3-1 https://youtu.be/yKKNr-QKz2Q

Mini-batches

We do not really minimize total loss!

Mini-batch



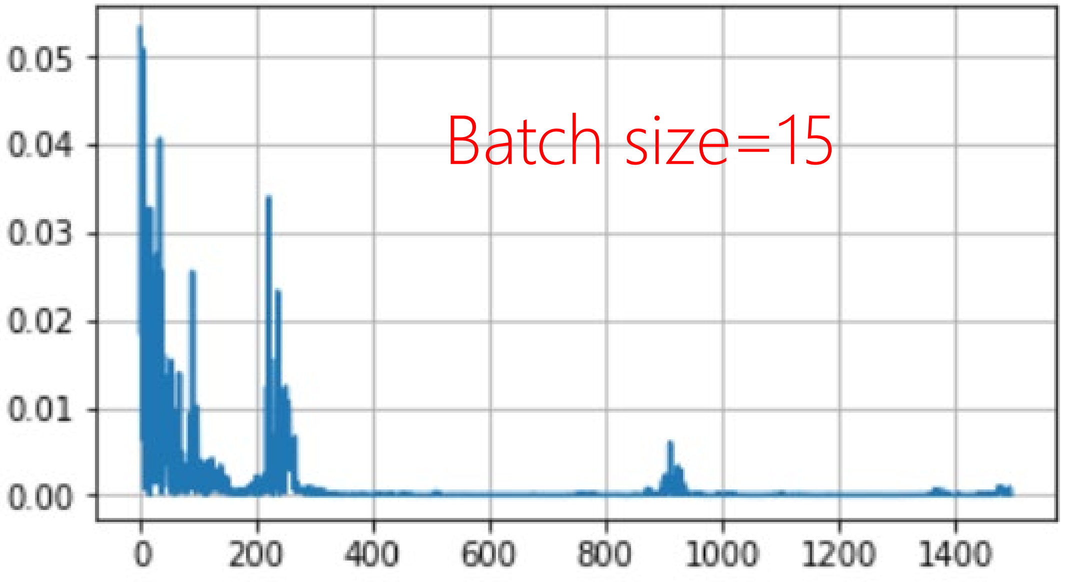
- Randomly initialize network parameters
- Pick the 1st batch
  $L' = l^1 + l^{31} + \cdots$
  Update parameters once
- Pick the 2nd batch
  $L'' = l^2 + l^{16} + \cdots$
  Update parameters once
  ⋮
- Until all mini-batches have been picked

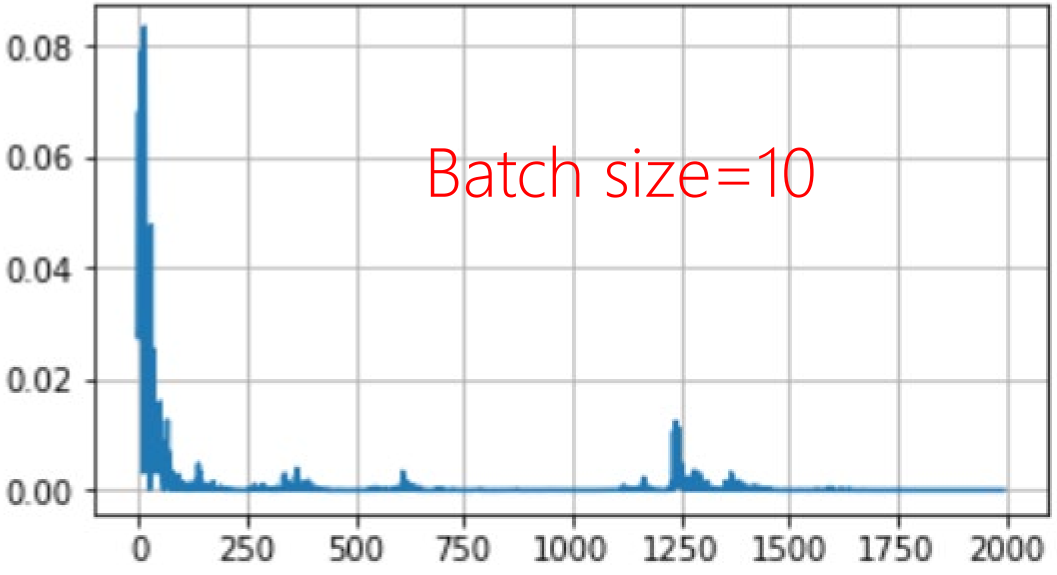
one epoch

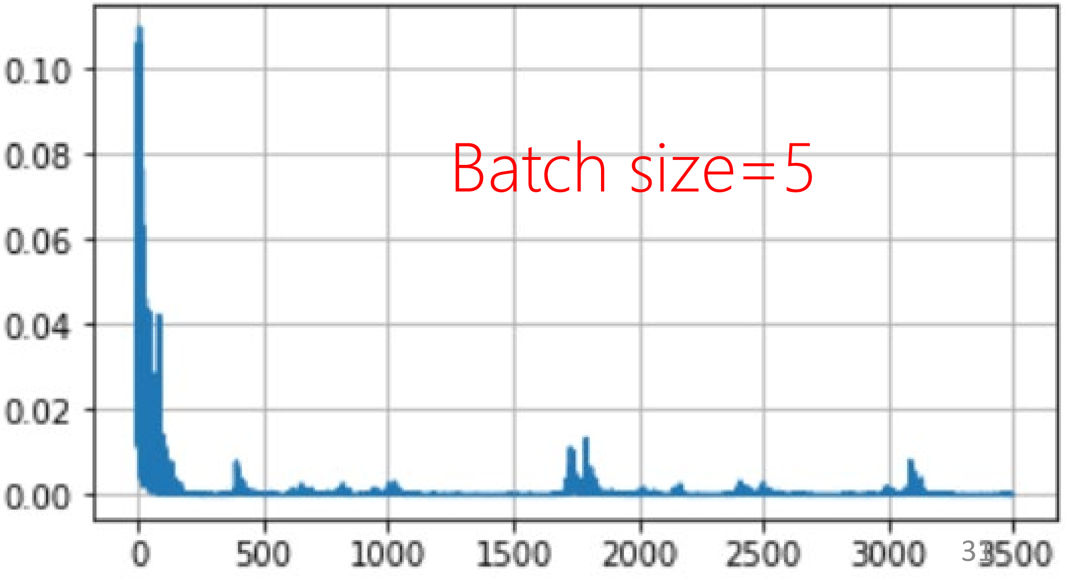
Repeat the above process

Reference: 李弘毅 ML Lecture 8-1 https://youtu.be/Lx3l4lOrquw

Run "3.3. Mini_Batch_Training"





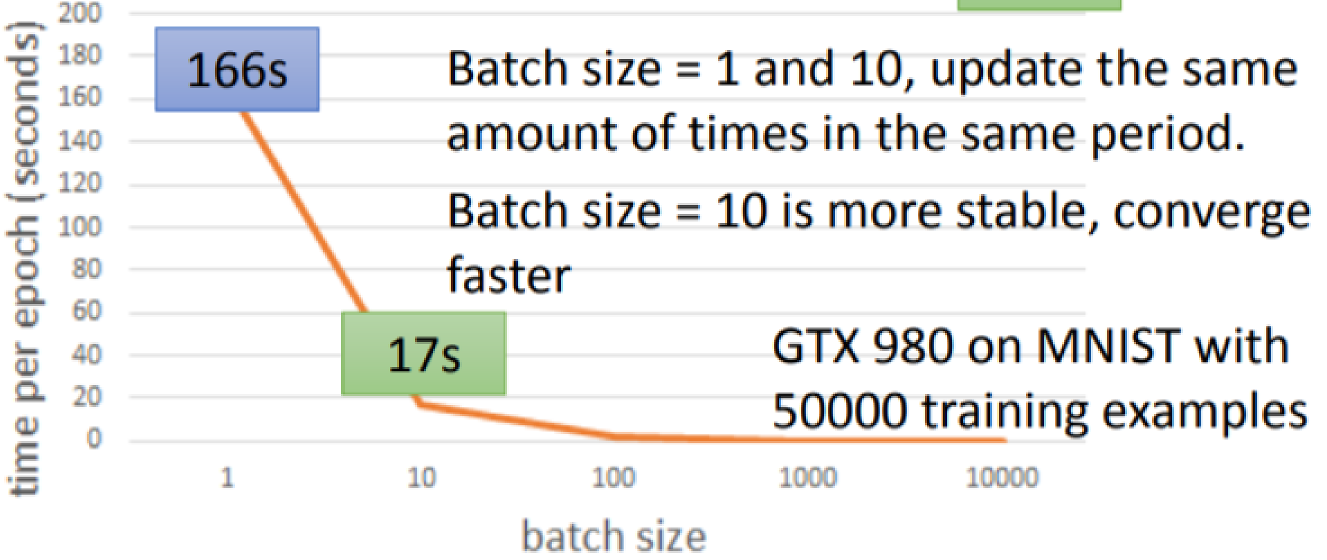


Batch size influences both speed and performance. You have to tune it.

Speed

Very large batch size can yield worse performance

- Smaller batch size means more updates in one epoch
  - E.g. 50000 examples
  - batch size = 1, 50000 updates in one epoch — 166s 1 epoch
  - batch size = 10, 5000 updates in one epoch — 17s 10 epoch



Reference: 李弘毅 ML Lecture 8-1 https://youtu.be/Lx3l4lOrquw