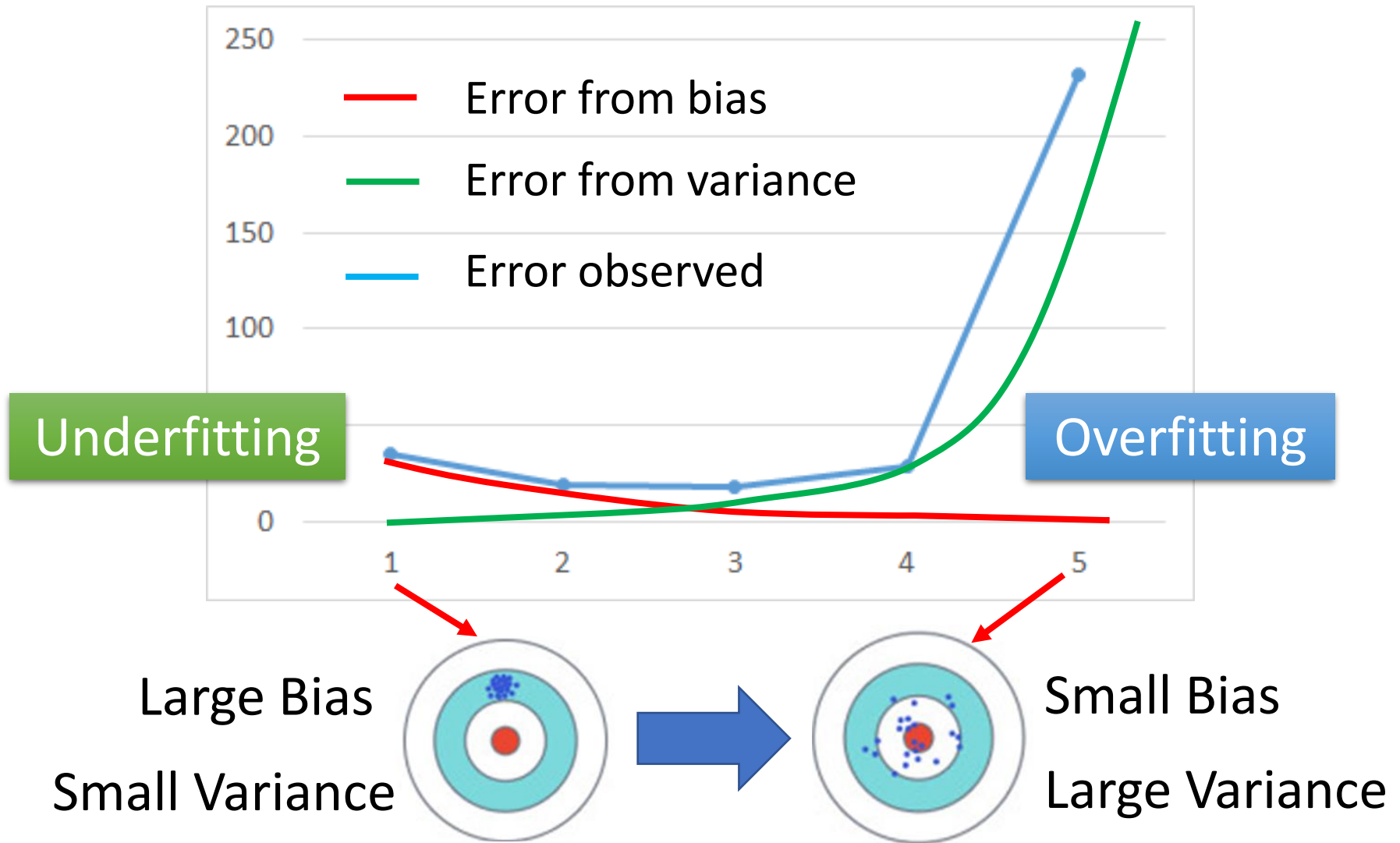




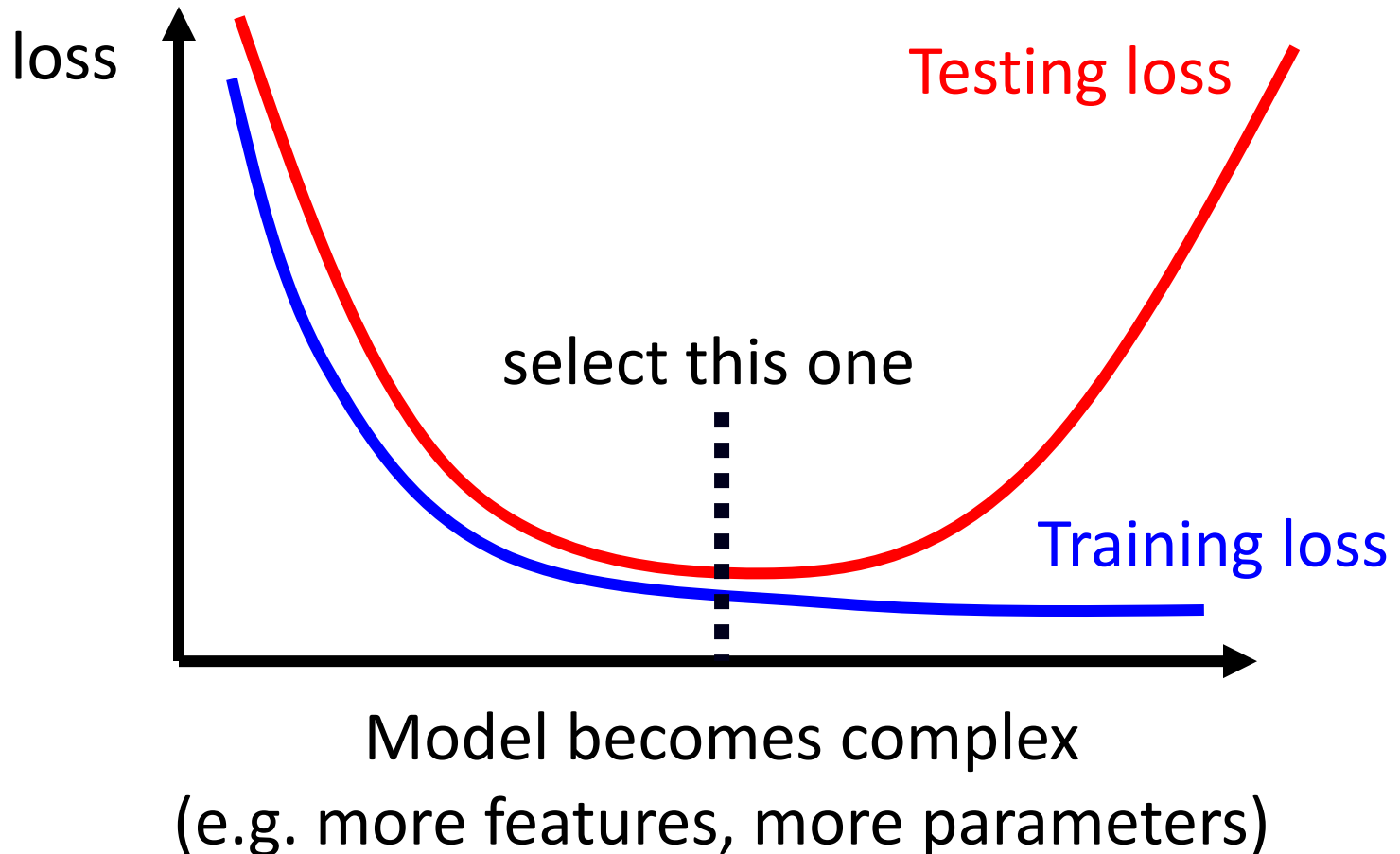
人工智能技术及应用

Artificial Intelligence and Application

Bias v.s. Variance



Bias-Complexity Trade-off



When
optimization
fails...



Review: Gradient Descent

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

$$\theta^* = \arg \min_{\theta} L(\theta) \quad L: \text{loss function} \quad \theta: \text{parameters}$$

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

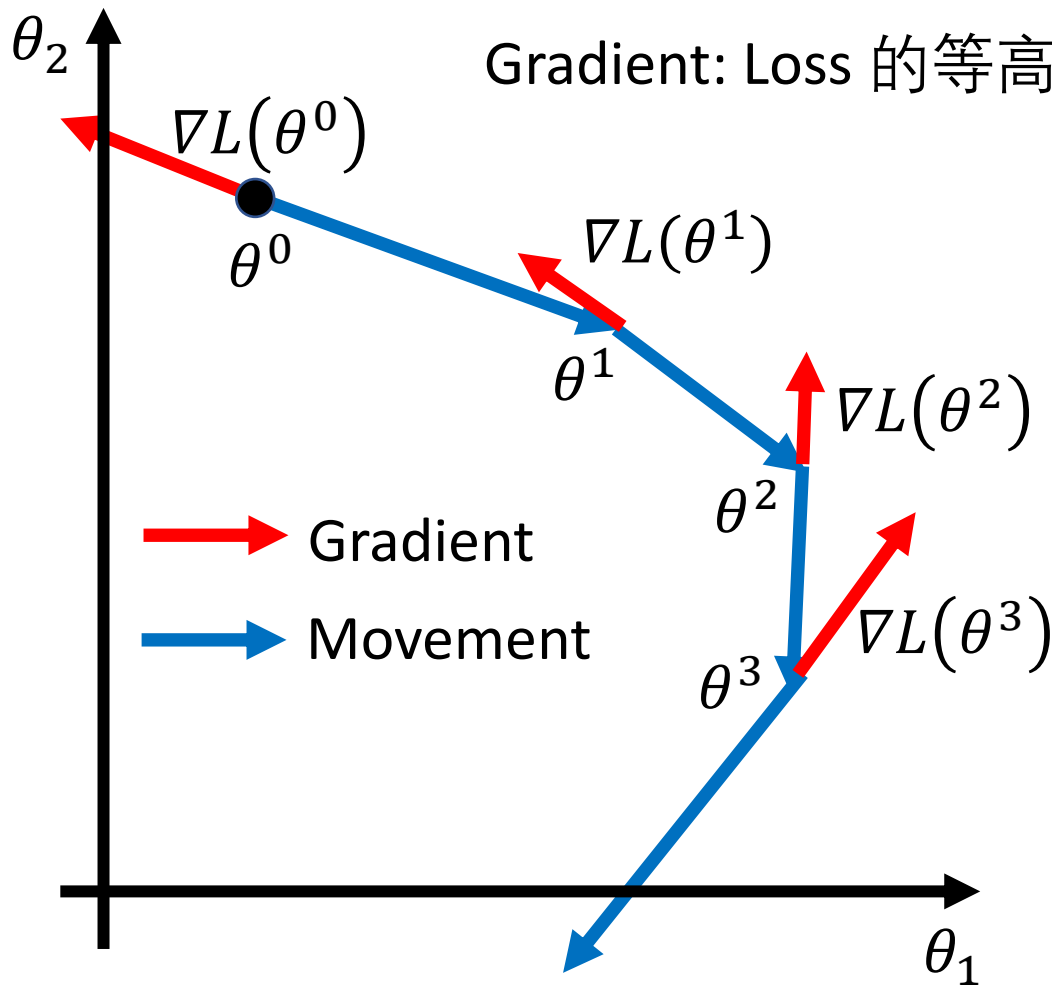
Randomly start at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$

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

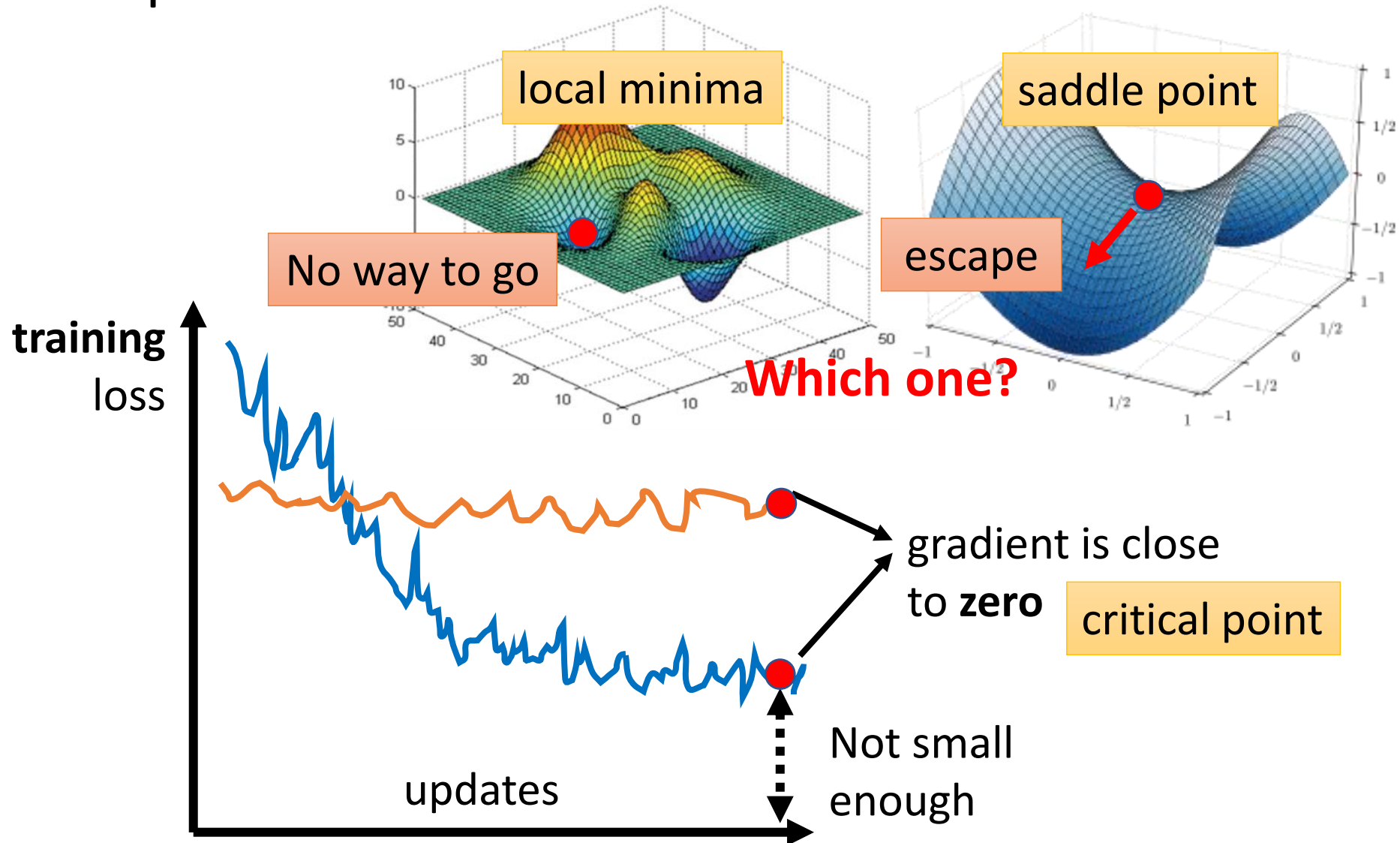
$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \partial L(\theta_1^0)/\partial \theta_1 \\ \partial L(\theta_2^0)/\partial \theta_2 \end{bmatrix} \quad \Rightarrow \quad \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

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

Review: Gradient Descent



Optimization Fails because



Taylor Series Approximation

$L(\boldsymbol{\theta})$ around $\boldsymbol{\theta} = \boldsymbol{\theta}'$ can be approximated below

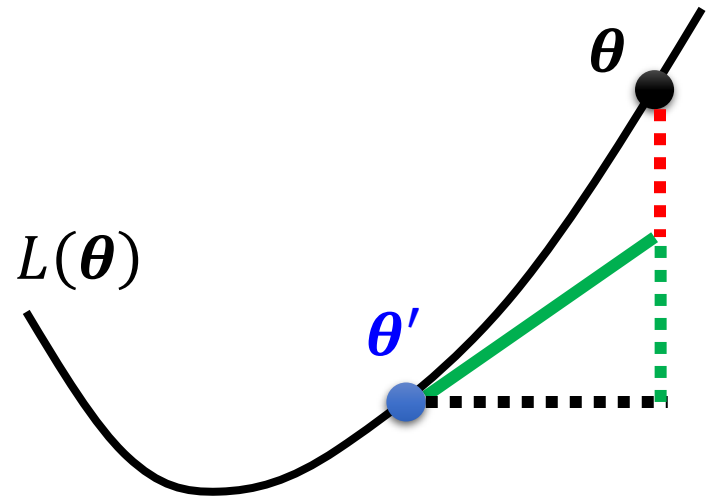
$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \boldsymbol{g} + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \boldsymbol{H} (\boldsymbol{\theta} - \boldsymbol{\theta}')$$

Gradient \boldsymbol{g} is a vector

$$\boldsymbol{g} = \nabla L(\boldsymbol{\theta}') \quad g_i = \frac{\partial L(\boldsymbol{\theta}')}{\partial \theta_i}$$

Hessian \boldsymbol{H} is a matrix

$$H_{ij} = \frac{\partial^2}{\partial \theta_i \partial \theta_j} L(\boldsymbol{\theta}')$$



Hessian

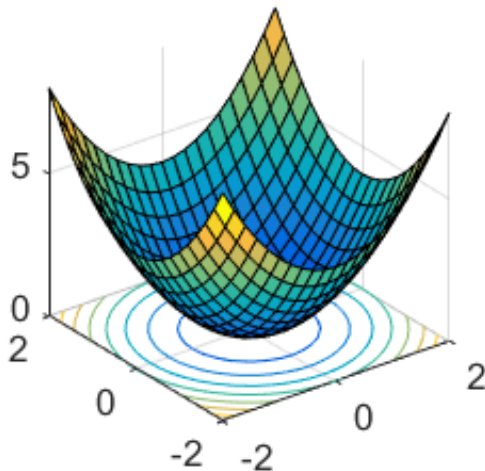
$L(\boldsymbol{\theta})$ around $\boldsymbol{\theta} = \boldsymbol{\theta}'$ can be approximated below

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \cancel{(\boldsymbol{\theta} - \boldsymbol{\theta}')^T \mathbf{g}} + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}')$$

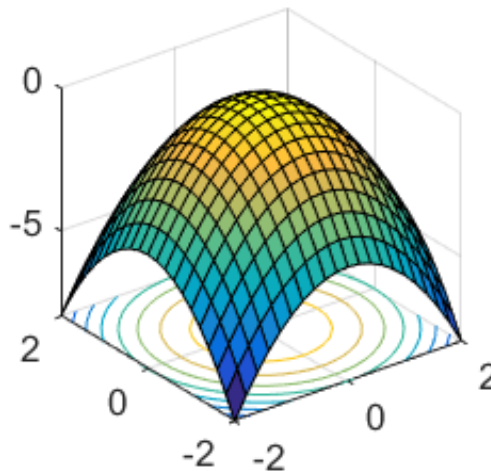
At critical point

telling the properties of critical points

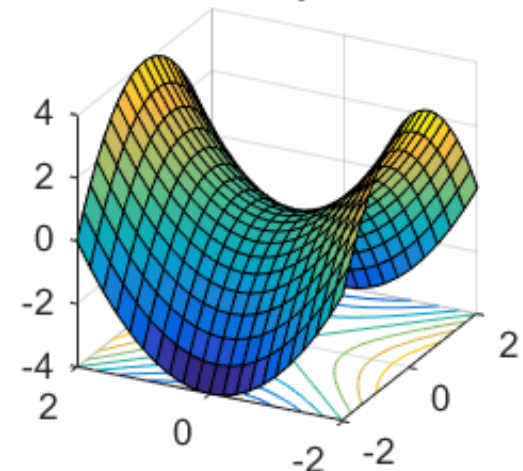
local min



local max



saddle point



At critical point:

$$\mathbf{v}^T \mathbf{H} \mathbf{v}$$

Hessian

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}')$$

For all \mathbf{v}

$$\mathbf{v}^T \mathbf{H} \mathbf{v} > 0 \quad \Rightarrow \quad \text{Around } \boldsymbol{\theta}': L(\boldsymbol{\theta}) > L(\boldsymbol{\theta}') \quad \Rightarrow \quad \text{Local minima}$$

= \mathbf{H} is positive definite = All eigen values are positive. \uparrow

For all \mathbf{v}

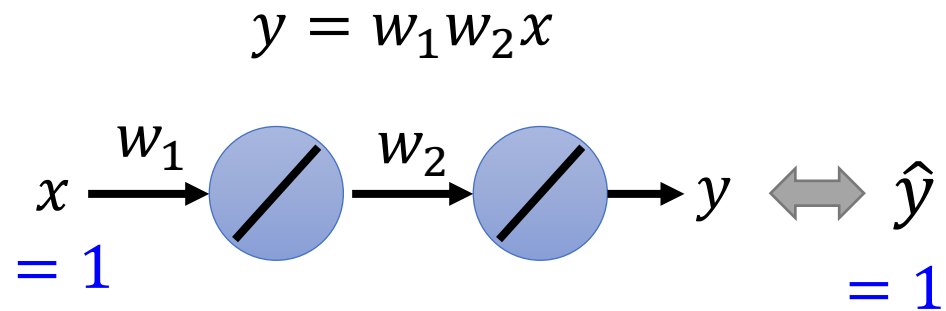
$$\mathbf{v}^T \mathbf{H} \mathbf{v} < 0 \quad \Rightarrow \quad \text{Around } \boldsymbol{\theta}': L(\boldsymbol{\theta}) < L(\boldsymbol{\theta}') \quad \Rightarrow \quad \text{Local maxima}$$

= \mathbf{H} is negative definite = All eigen values are negative. \uparrow

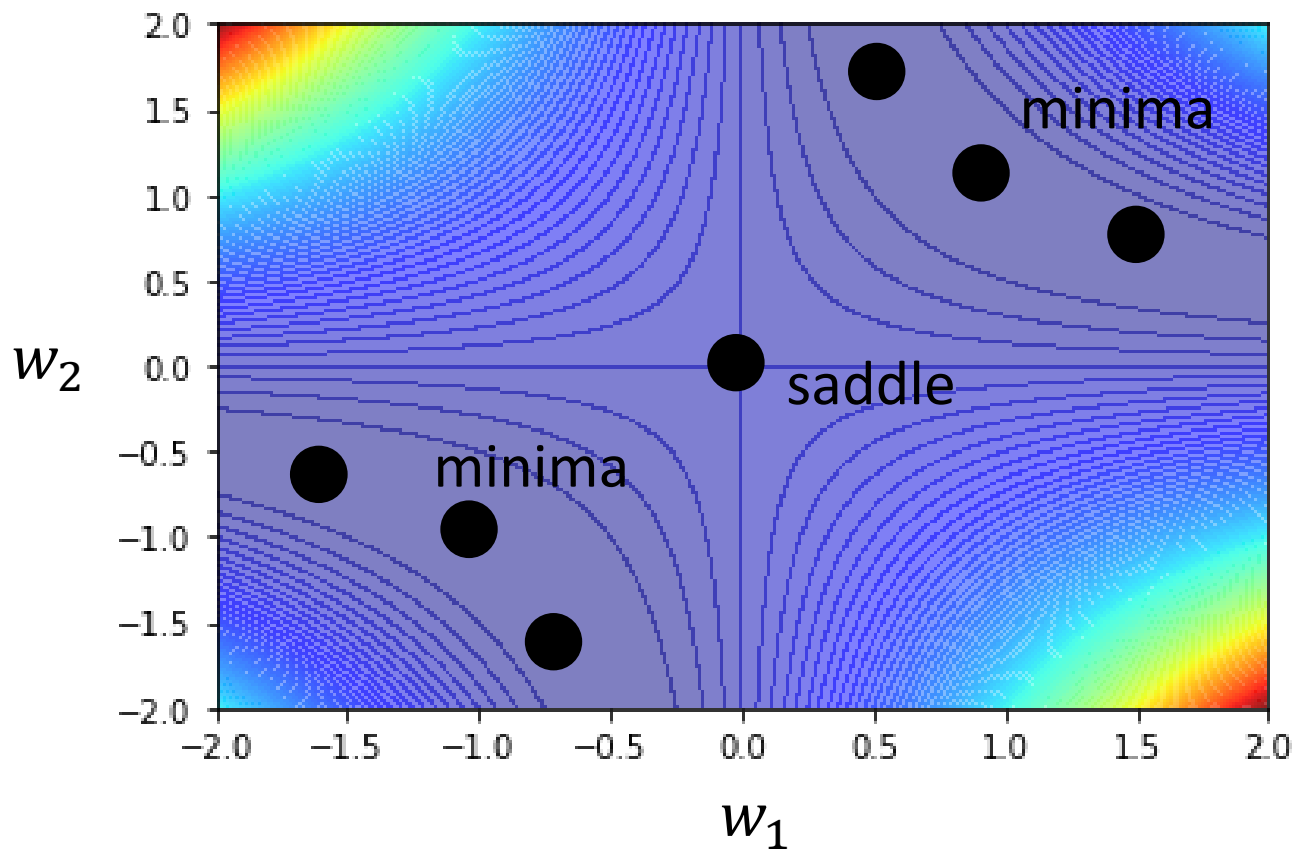
$$\text{Sometimes } \mathbf{v}^T \mathbf{H} \mathbf{v} > 0, \text{ sometimes } \mathbf{v}^T \mathbf{H} \mathbf{v} < 0 \quad \Rightarrow \quad \text{Saddle point}$$

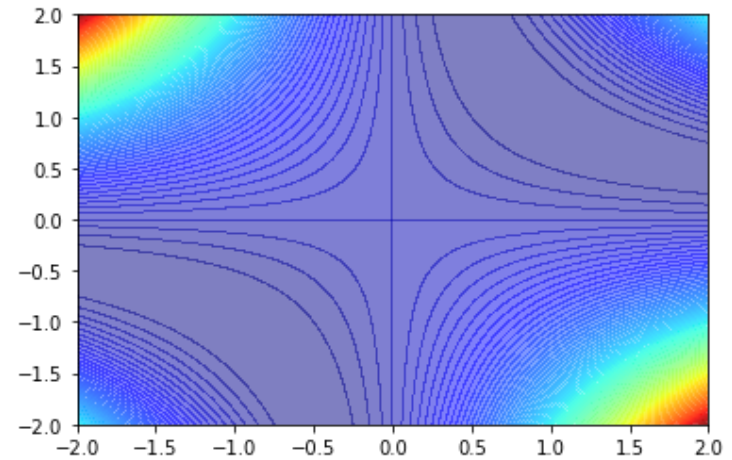
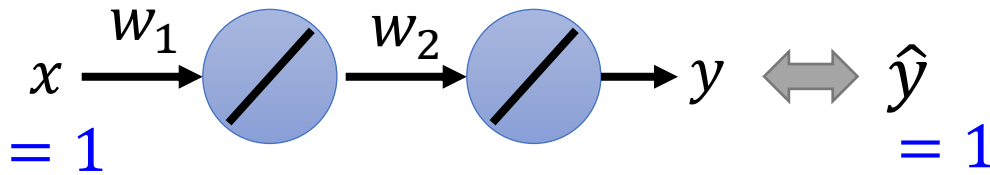
Some eigen values are positive, and some are negative. \uparrow

Example



Error Surface





$$L = (\hat{y} - w_1 w_2 x)^2 = (1 - w_1 w_2)^2$$

$$\frac{\partial L}{\partial w_1} = 2(1 - w_1 w_2)(-w_2) = 0$$

$$\frac{\partial L}{\partial w_2} = 2(1 - w_1 w_2)(-w_1) = 0$$

Critical point: $w_1 = 0, w_2 = 0$

$$H = \begin{bmatrix} 0 & -2 \\ -2 & 0 \end{bmatrix} \quad \lambda_1 = 2, \lambda_2 = -2$$

Saddle point

g

H

$$\frac{\partial^2 L}{\partial w_1^2} = 2(-w_2)(-w_2) = 0$$

$$\frac{\partial^2 L}{\partial w_2 \partial w_1} = -2 + 4w_1 w_2 = -2$$

$$\frac{\partial^2 L}{\partial w_1 \partial w_2} = -2 + 4w_1 w_2 = -2$$

$$\frac{\partial^2 L}{\partial w_2^2} = 2(-w_1)(-w_1) = 0$$

Don't afraid of saddle point?

$$\mathbf{v}^T \mathbf{H} \mathbf{v}$$

At critical point: $L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}')$

Sometimes $\mathbf{v}^T \mathbf{H} \mathbf{v} > 0$, sometimes $\mathbf{v}^T \mathbf{H} \mathbf{v} < 0 \Rightarrow$ Saddle point

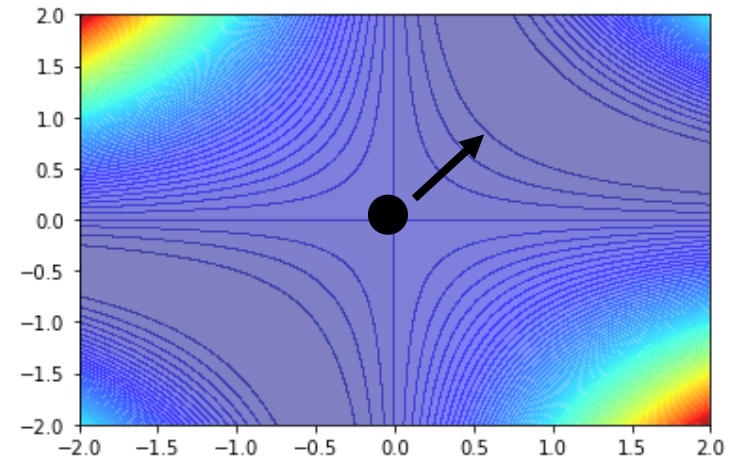
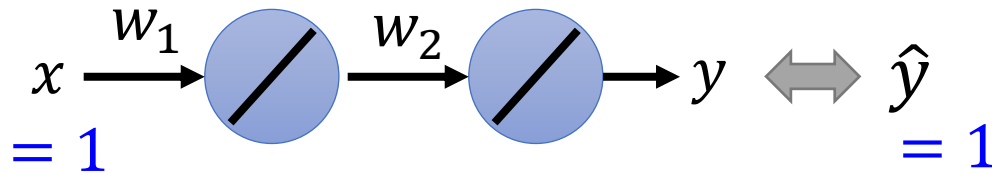
\mathbf{H} may tell us parameter update direction!

\mathbf{u} is an eigen vector of \mathbf{H}
 λ is the eigen value of \mathbf{u}
 $\lambda < 0$

$\Rightarrow \mathbf{u}^T \mathbf{H} \mathbf{u} = \mathbf{u}^T (\lambda \mathbf{u}) = \lambda \|\mathbf{u}\|^2 < 0$

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}') + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}')^T \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}') \Rightarrow L(\boldsymbol{\theta}) < L(\boldsymbol{\theta}')$$

$$\boldsymbol{\theta} - \boldsymbol{\theta}' = \mathbf{u} \quad \boldsymbol{\theta} = \boldsymbol{\theta}' + \mathbf{u} \quad \text{Decrease } L$$



$$L = (\hat{y} - w_1 w_2 x)^2 = (1 - w_1 w_2)^2$$

$$\frac{\partial L}{\partial w_1} = 2(1 - w_1 w_2)(-w_2)$$

$$\frac{\partial L}{\partial w_2} = 2(1 - w_1 w_2)(-w_1)$$

Critical point: $w_1 = 0, w_2 = 0$

$$H = \begin{bmatrix} 0 & -2 \\ -2 & 0 \end{bmatrix} \quad \lambda_1 = 2, \lambda_2 = -2$$

Saddle point

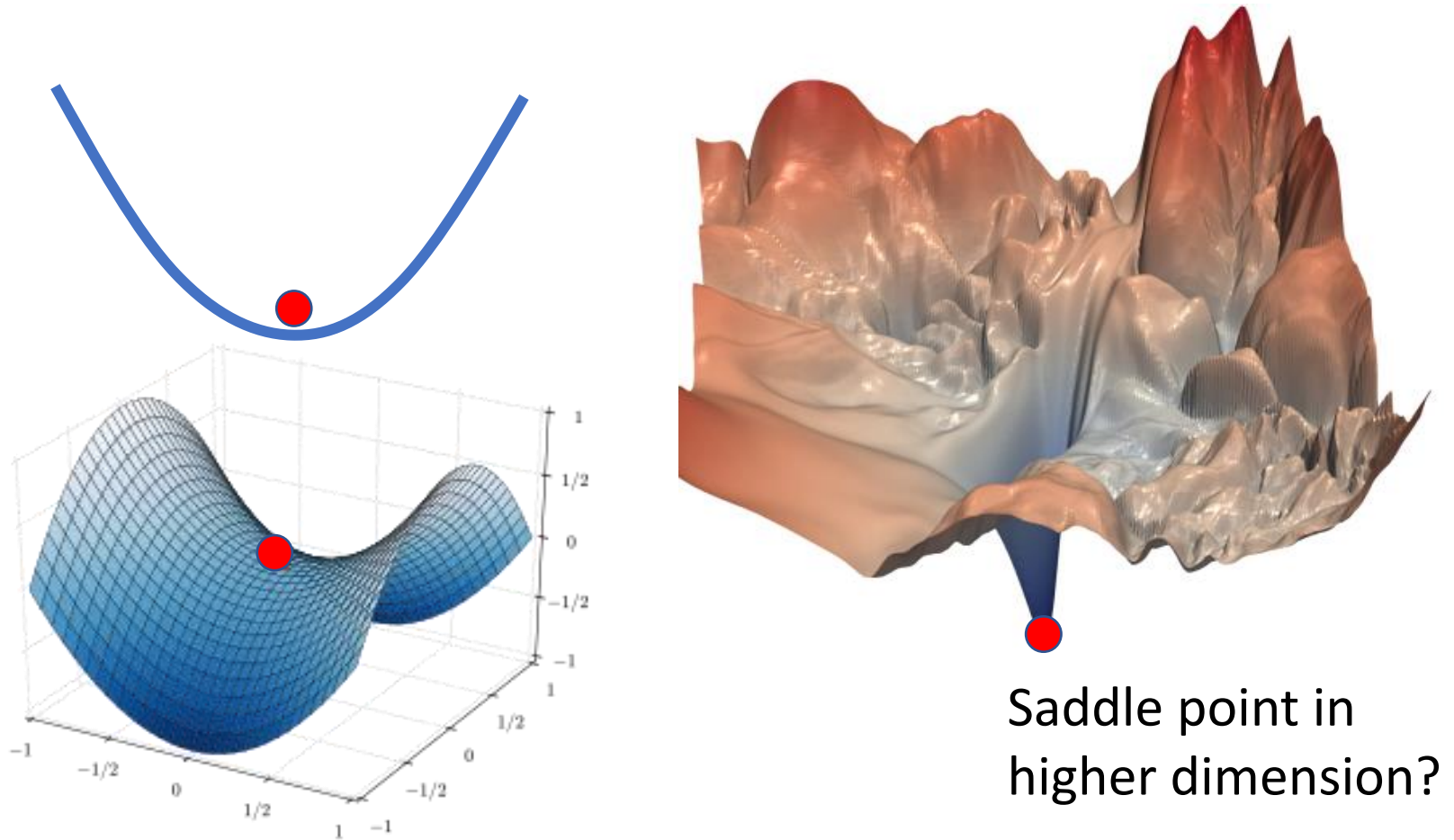
$$\lambda_2 = -2 \quad \text{Has eigenvector } \mathbf{u} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Update the parameter along the direction of \mathbf{u}

You can escape the saddle point and decrease the loss.

(this method is seldom used in practice)

Saddle Point v.s. Local Minima



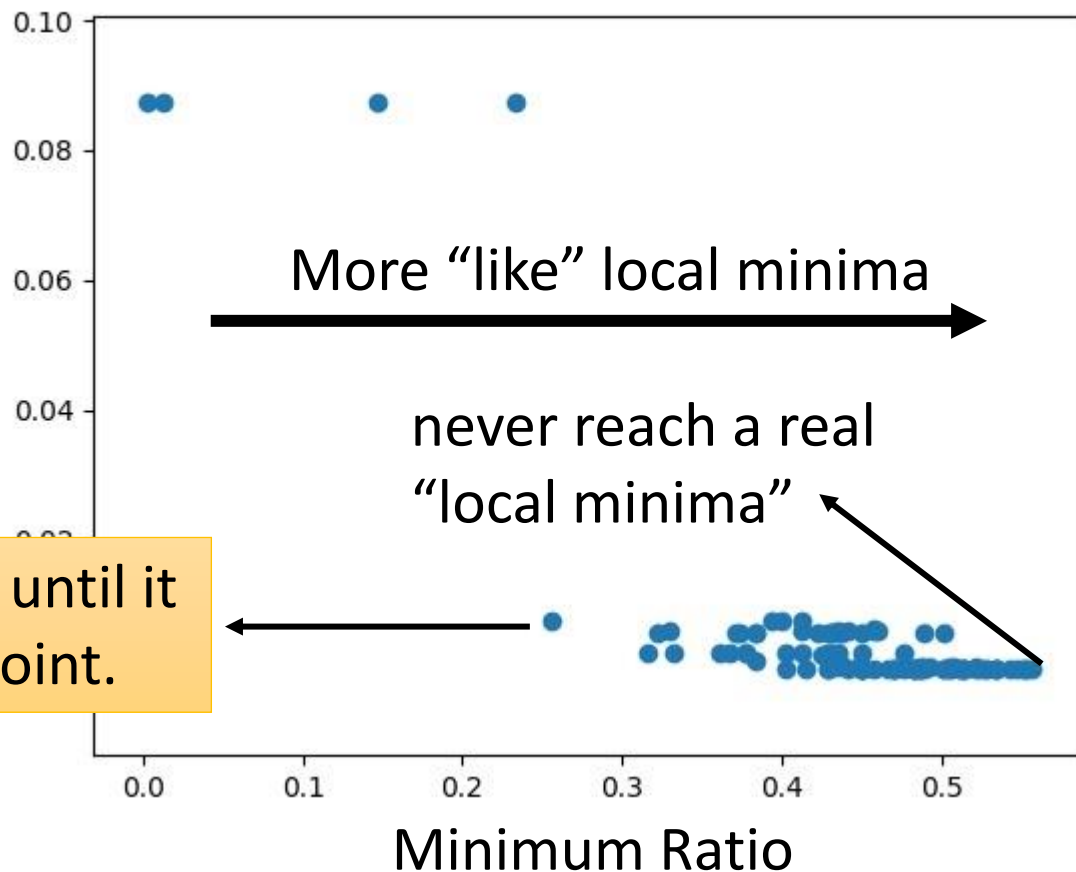
Saddle point in
higher dimension?

When you have lots of parameters, perhaps local minima is rare?

Empirical Study

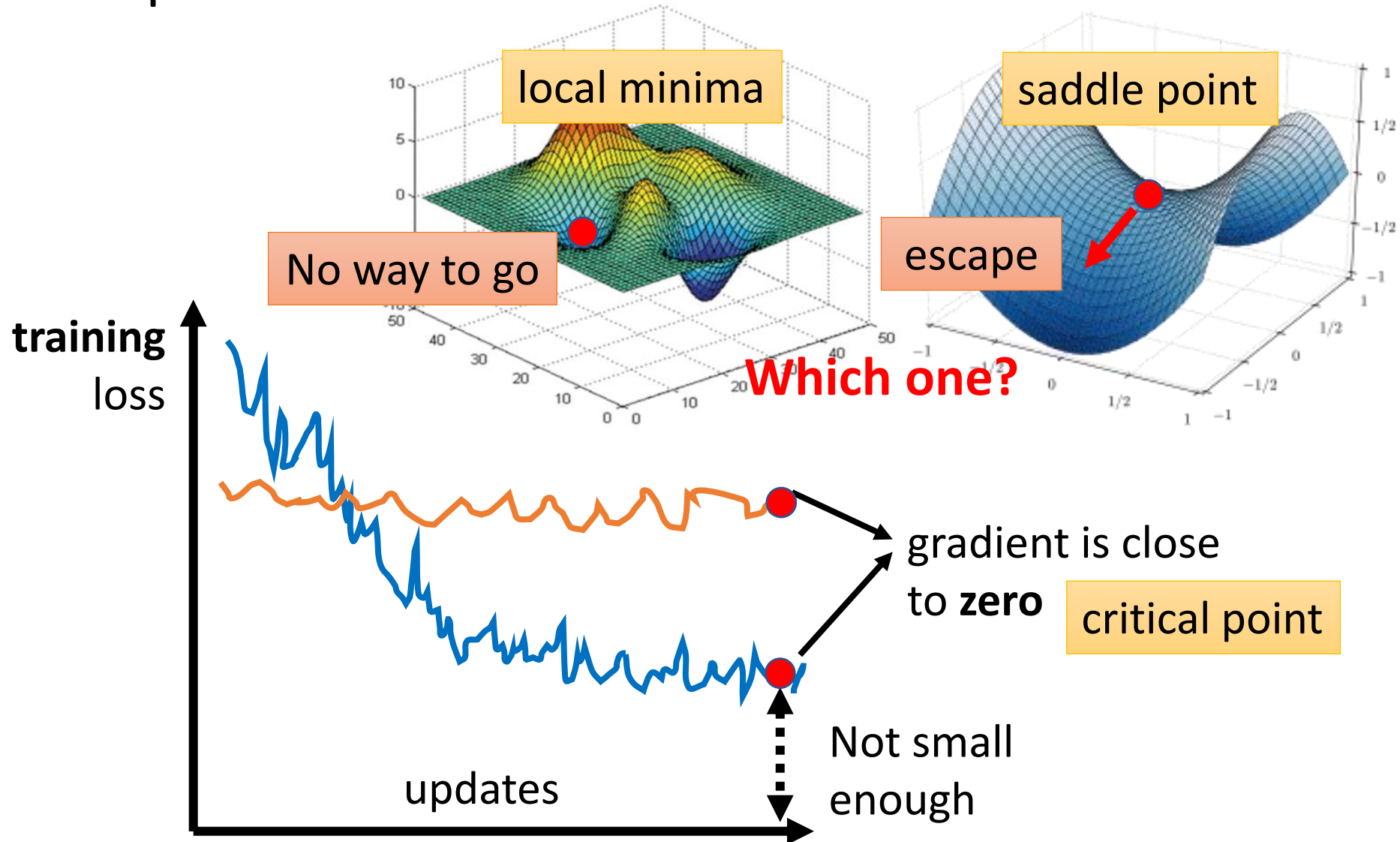
Training
Loss

Train a network once, until it
converges to critical point.

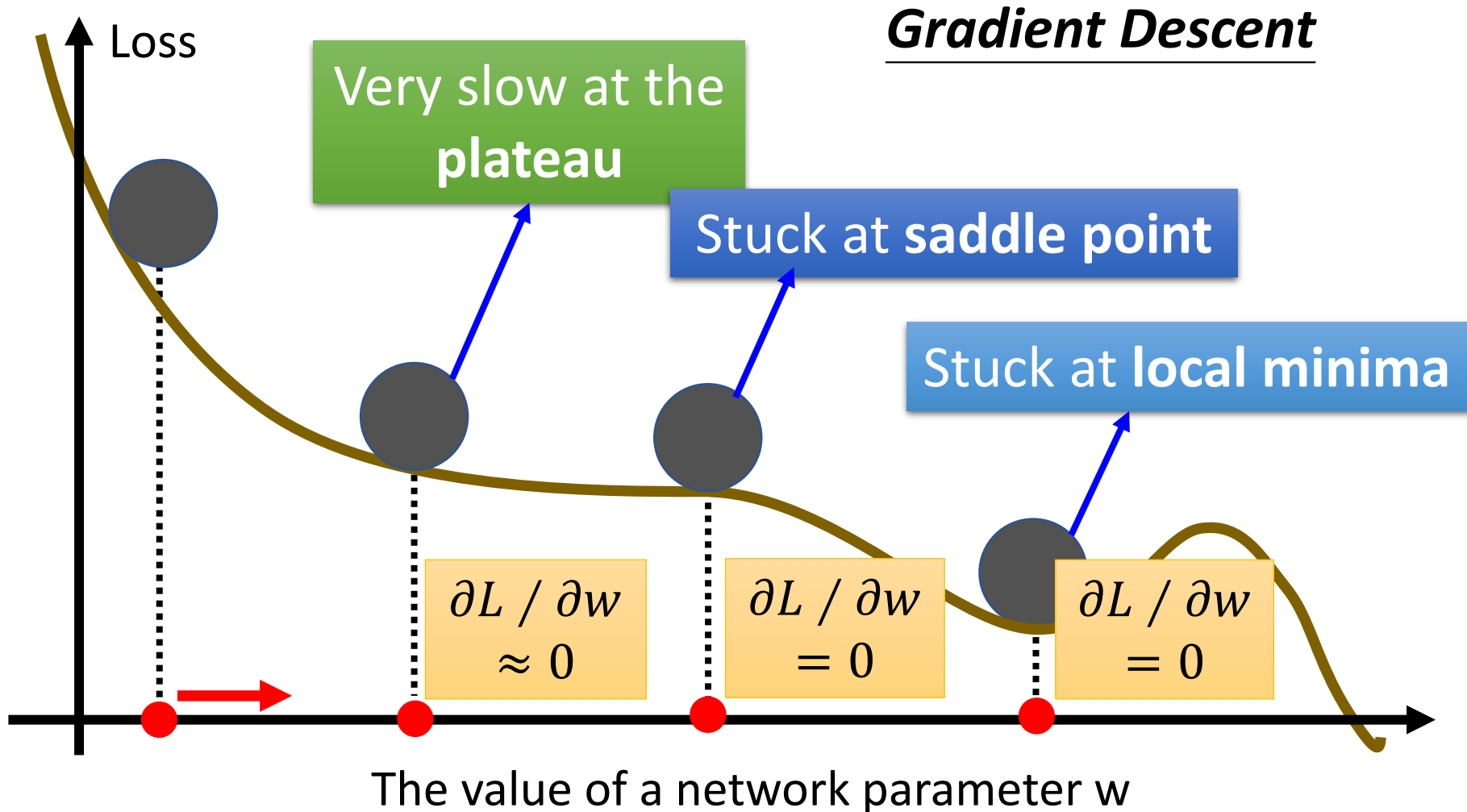


$$\text{Minimum ratio} = \frac{\text{Number of **Positive** Eigen values}}{\text{Number of Eigen values}}$$

Optimization Fails because



Small Gradient ...



How to escape from critical points...

Tips for training: **Batch and Momentum**

Batch



Review: Optimization

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

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \end{bmatrix}$$

➤ (Randomly) Pick initial values $\boldsymbol{\theta}^0$

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

$$\mathbf{g} = \nabla L(\boldsymbol{\theta}^0)$$

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

Review: Optimization

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

- (Randomly) Pick initial values $\boldsymbol{\theta}^0$
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^0)$
 $\boldsymbol{\theta}^1 \leftarrow \boldsymbol{\theta}^0 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^1)$
 $\boldsymbol{\theta}^2 \leftarrow \boldsymbol{\theta}^1 - \eta \boldsymbol{g}$
- Compute gradient $\boldsymbol{g} = \nabla L(\boldsymbol{\theta}^2)$
 $\boldsymbol{\theta}^3 \leftarrow \boldsymbol{\theta}^2 - \eta \boldsymbol{g}$

Optimization with Batch

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

➤ (Randomly) Pick initial values θ^0

➤ Compute gradient $g^0 = \nabla L^1(\theta^0)$ L^1

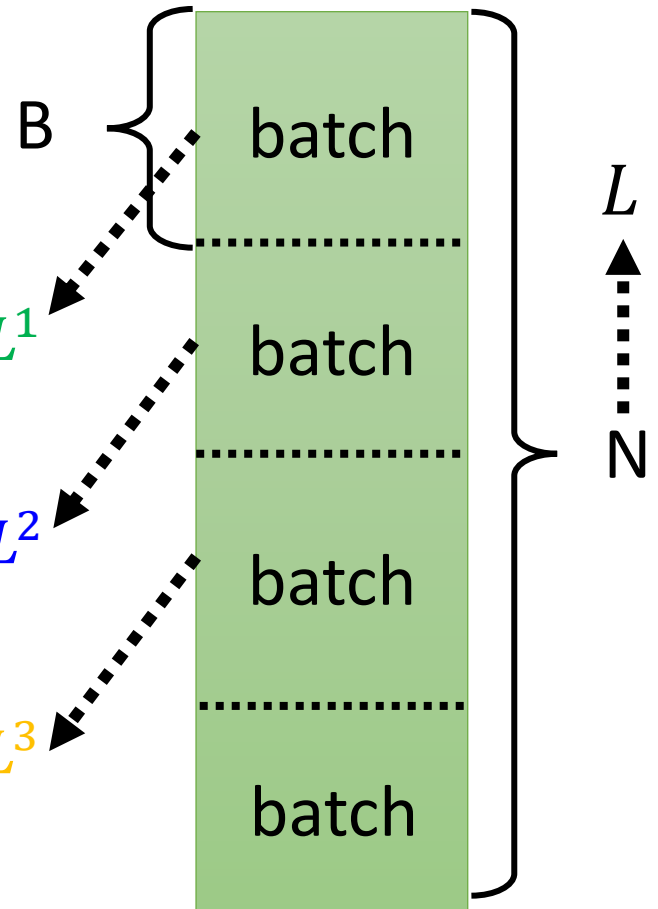
$$\text{update } \theta^1 \leftarrow \theta^0 - \eta g^0$$

➤ Compute gradient $g^1 = \nabla L^2(\theta^1)$ L^2

$$\text{update } \theta^2 \leftarrow \theta^1 - \eta g^1$$

➤ Compute gradient $g^3 = \nabla L^3(\theta^2)$ L^3

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



1 **epoch** = see all the batches once → **Shuffle** after each epoch

Optimization of New Model

Example 1

- 10,000 examples ($N = 10,000$)
- Batch size is 10 ($B = 10$)

How many update in **1 epoch**?

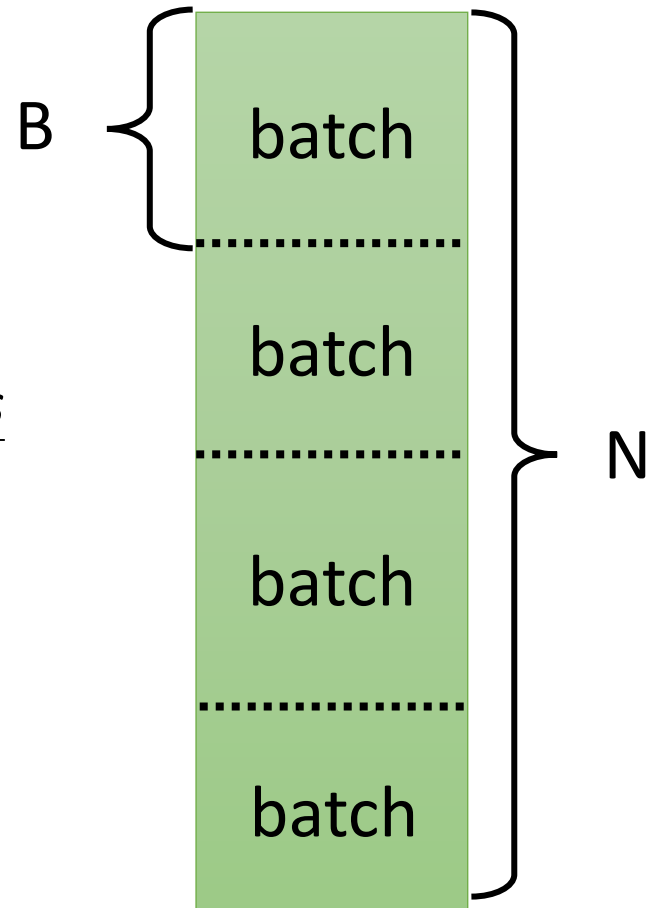
1,000 updates

Example 2

- 1,000 examples ($N = 1,000$)
- Batch size is 100 ($B = 100$)

How many update in **1 epoch**?

10 updates

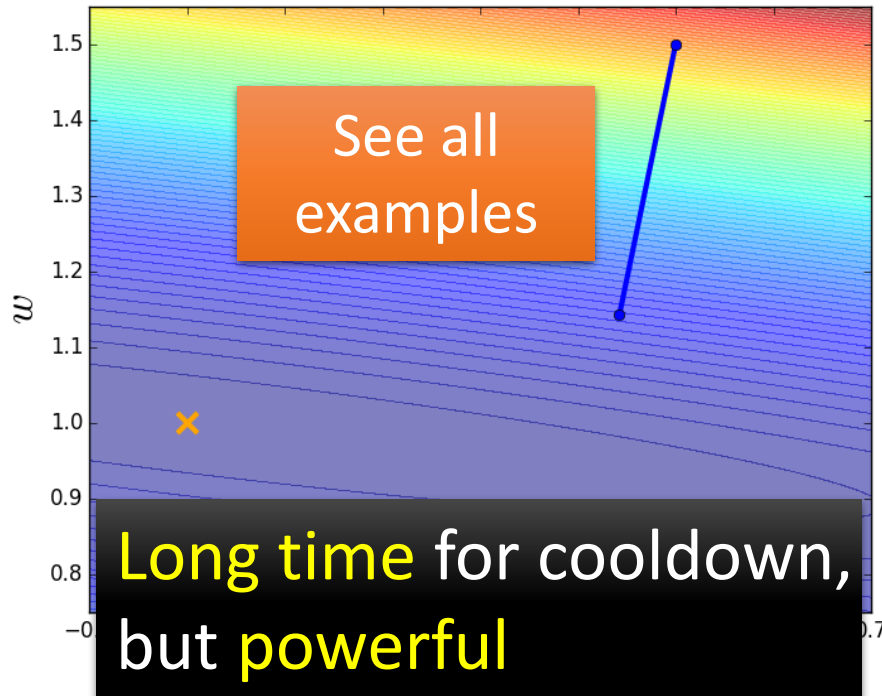


Small Batch v.s. Large Batch

Consider 20 examples ($N=20$)

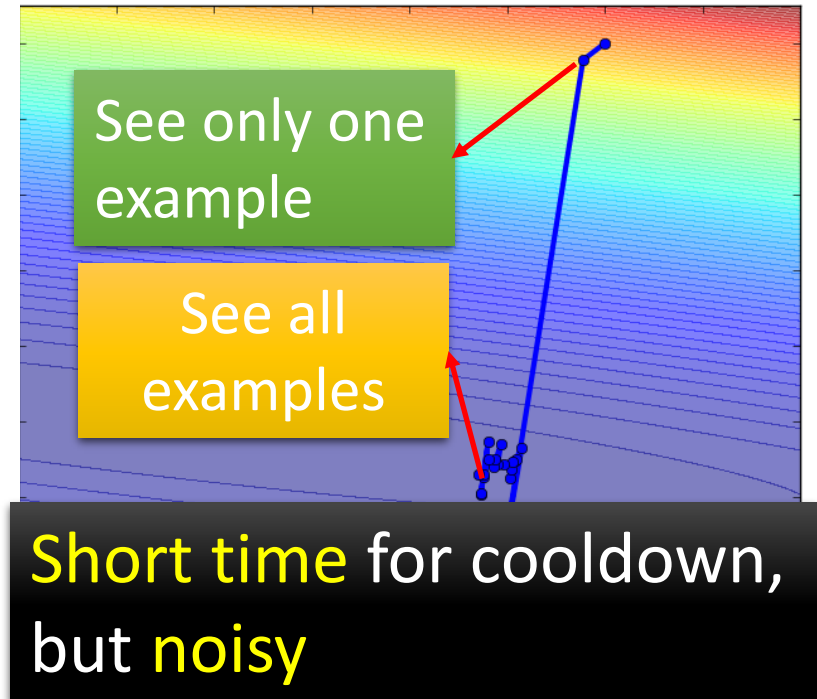
Batch size = N (Full batch)

Update after seeing all
the 20 examples



Batch size = 1

Update for each example
Update 20 times in an epoch



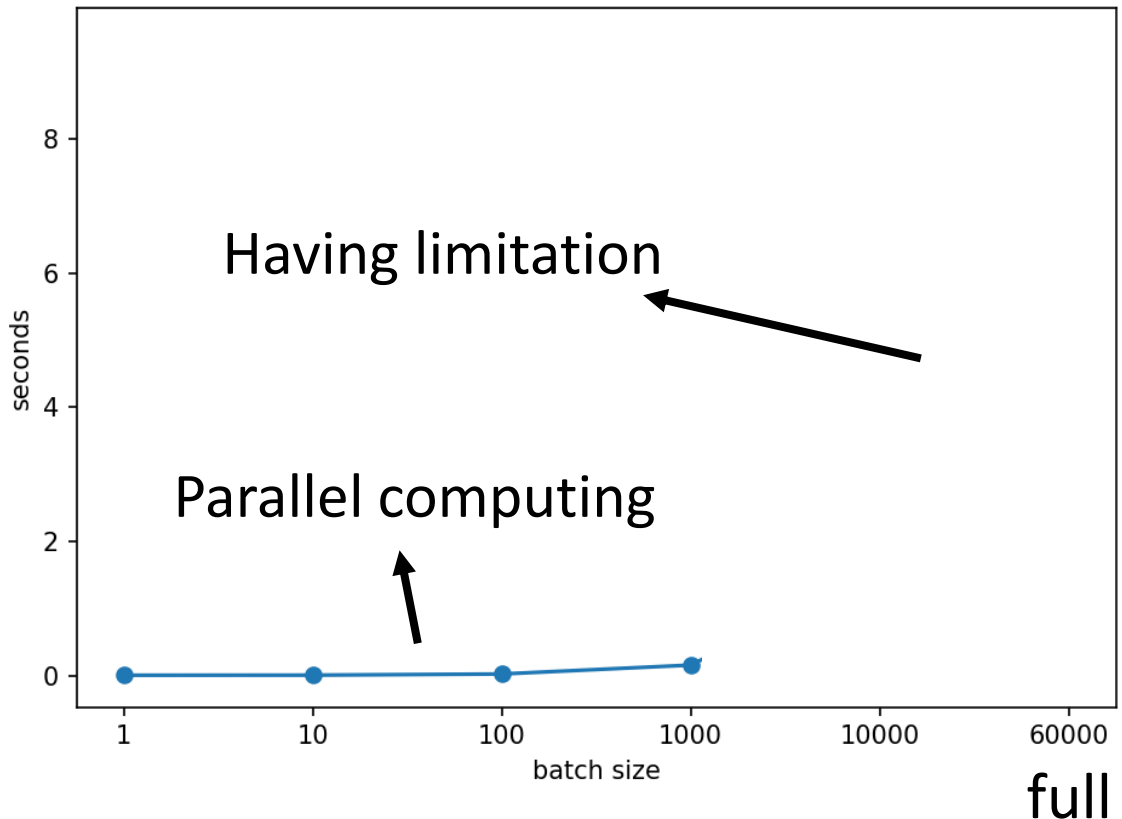
Small Batch v.s. Large Batch

- Larger batch size does **not** require longer time to compute gradient (unless batch size is too large)

**Time for
each update**

MNIST: digit
classification

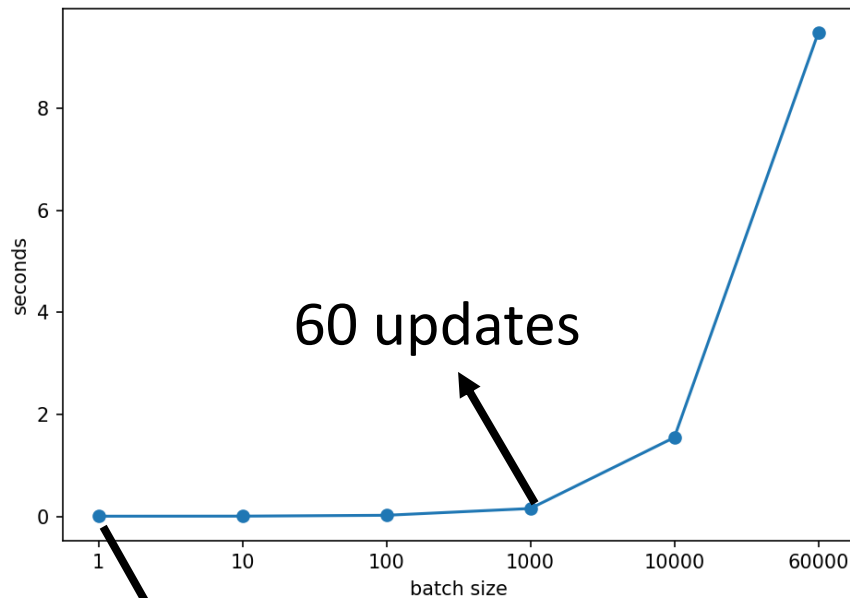
Tesla V100 GPU



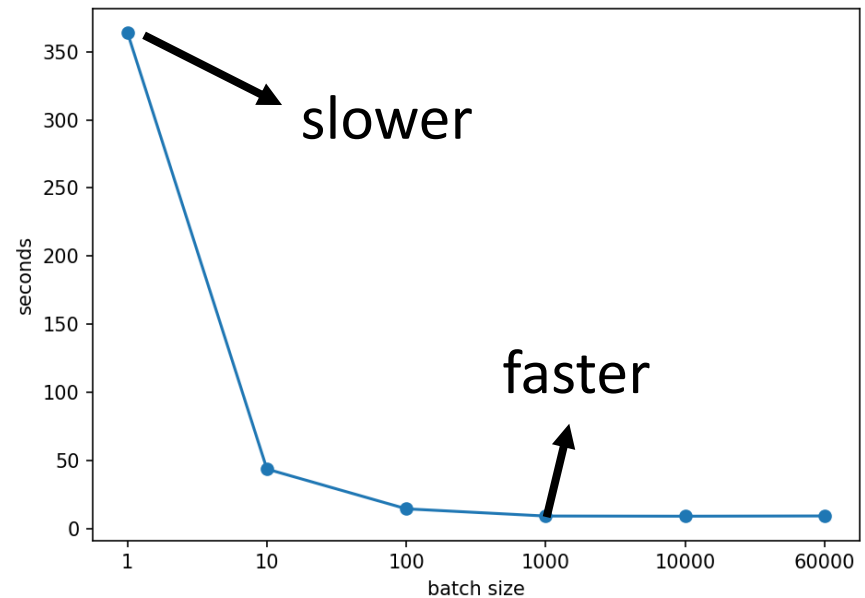
Small Batch v.s. Large Batch

- Smaller batch requires longer time for one epoch (longer time for seeing all data once)

Time for one **update**



Time for one **epoch**

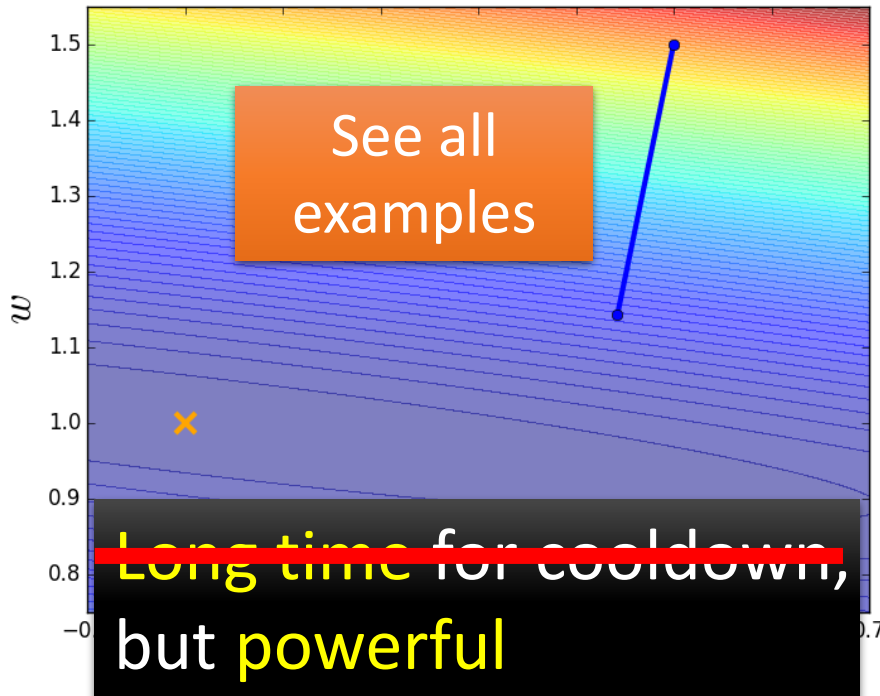


Small Batch v.s. Large Batch

Consider 20 examples ($N=20$)

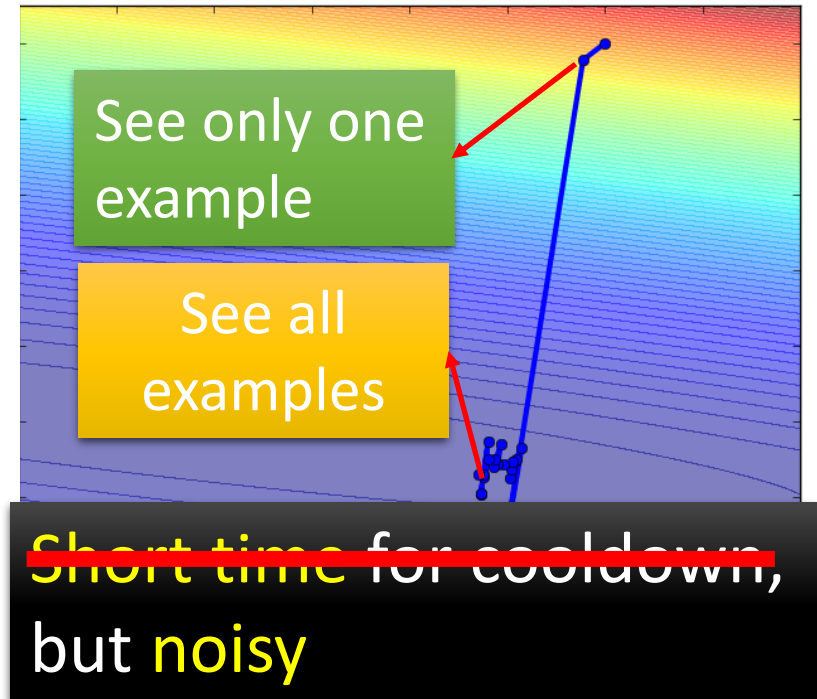
Batch size = N (Full Batch)

Update after seeing all
the 20 examples



Batch size = 1

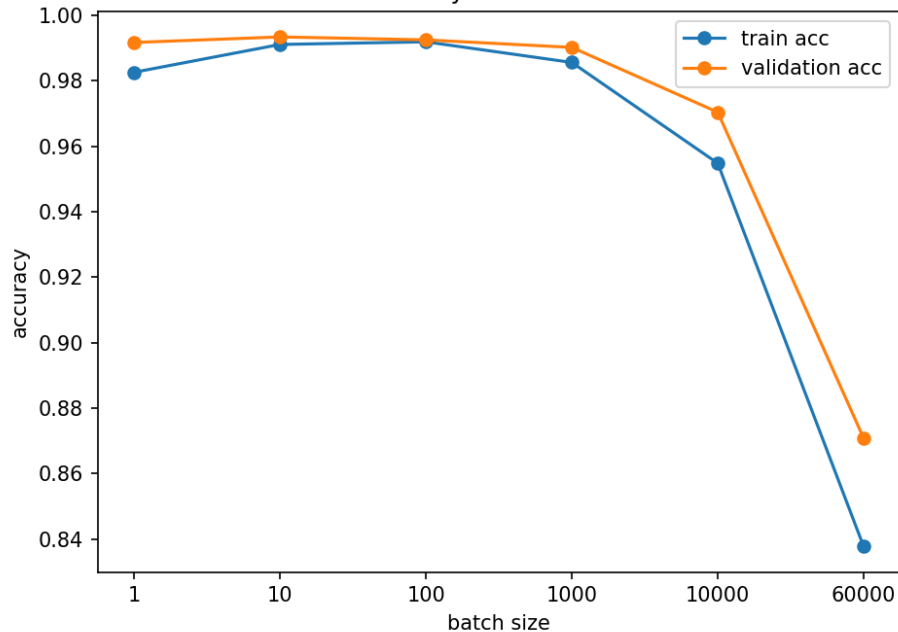
Update for each example
Update 20 times in an epoch



Small Batch v.s. Large Batch

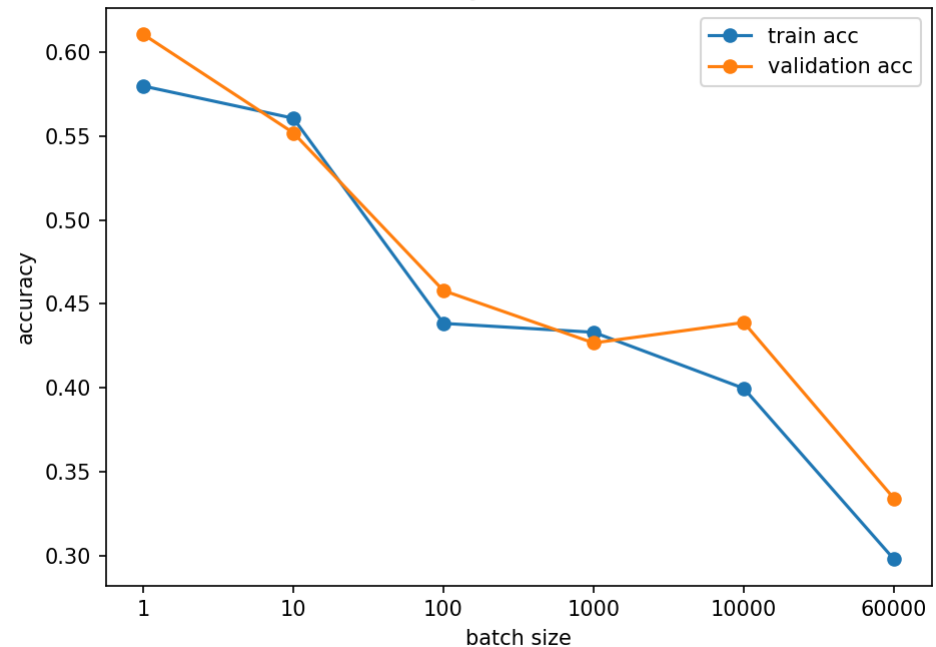
MNIST

Accuracy vs. Batch Size



CIFAR-10

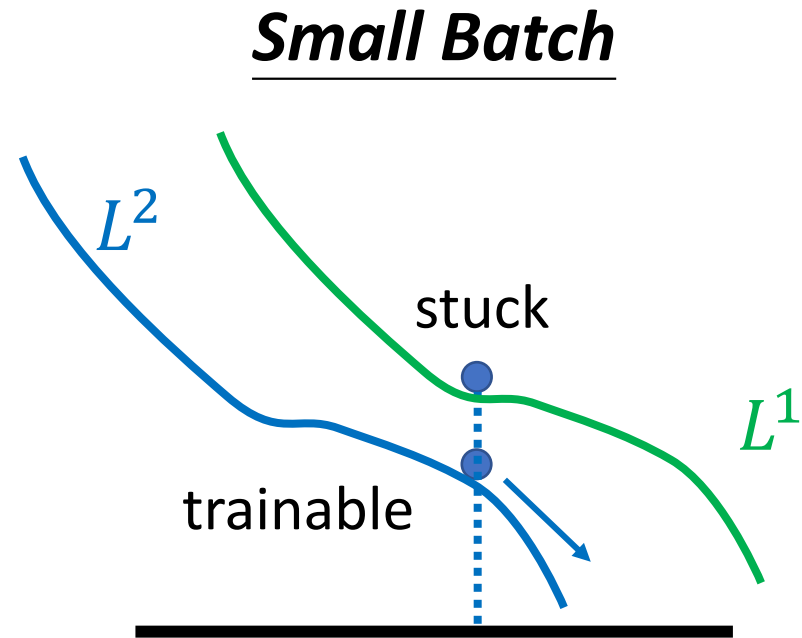
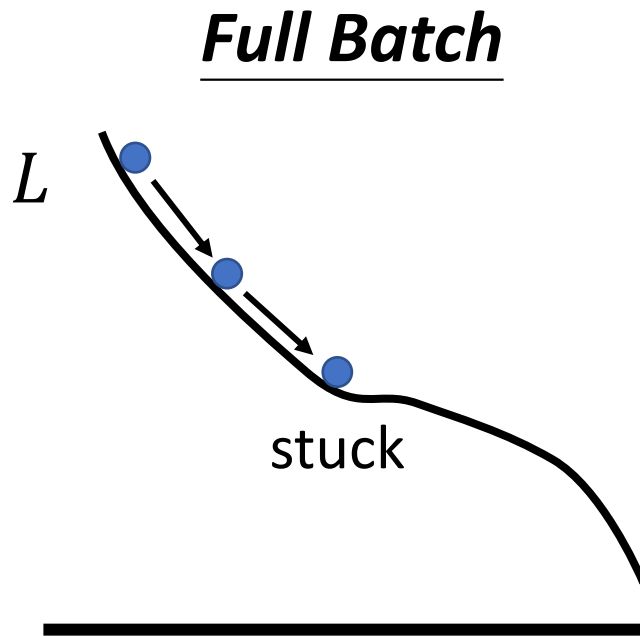
Accuracy vs. Batch Size



- Smaller batch size has better performance
- What's wrong with large batch size? Optimization Fails

Small Batch v.s. Large Batch

- Smaller batch size has better performance
- “Noisy” update is better for training



Small Batch v.s. Large Batch

- Small batch is better on testing data?

SB = 256

LB =

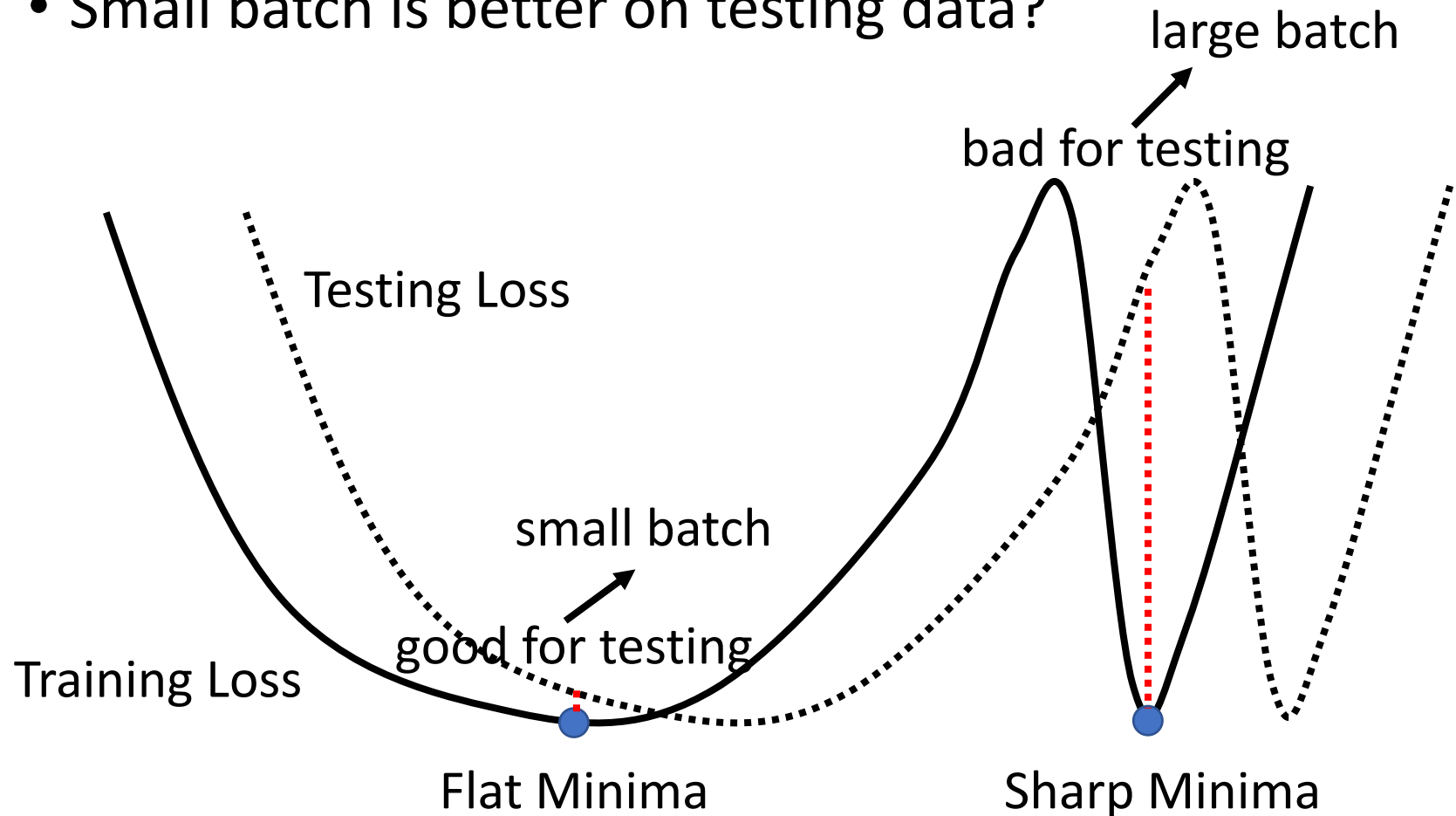
0.1 x data set

Name	Network Type	Data set
F_1	Fully Connected	MNIST (LeCun et al., 1998a)
F_2	Fully Connected	TIMIT (Garofolo et al., 1993)
C_1	(Shallow) Convolutional	CIFAR-10 (Krizhevsky & Hinton, 2009)
C_2	(Deep) Convolutional	CIFAR-10
C_3	(Shallow) Convolutional	CIFAR-100 (Krizhevsky & Hinton, 2009)
C_4	(Deep) Convolutional	CIFAR-100




Name	Training Accuracy		Testing Accuracy	
	SB	LB	SB	LB
F_1	99.66% \pm 0.05%	99.92% \pm 0.01%	98.03% \pm 0.07%	97.81% \pm 0.07%
F_2	99.99% \pm 0.03%	98.35% \pm 2.08%	64.02% \pm 0.2%	59.45% \pm 1.05%
C_1	99.89% \pm 0.02%	99.66% \pm 0.2%	80.04% \pm 0.12%	77.26% \pm 0.42%
C_2	99.99% \pm 0.04%	99.99% \pm 0.01%	89.24% \pm 0.12%	87.26% \pm 0.07%
C_3	99.56% \pm 0.44%	99.88% \pm 0.30%	49.58% \pm 0.39%	46.45% \pm 0.43%
C_4	99.10% \pm 1.23%	99.57% \pm 1.84%	63.08% \pm 0.5%	57.81% \pm 0.17%

Small Batch v.s. Large Batch

- Small batch is better on testing data?



Small Batch v.s. Large Batch

	Small	Large
Speed for one update (no parallel)	Faster	Slower
Speed for one update (with parallel)	Same	Same (not too large)
Time for one epoch	Slower	Faster 
Gradient	Noisy	Stable
Optimization	Better 	Worse
Generalization	Better 	Worse

Batch size is a hyperparameter you have to decide.

鱼和熊掌能否兼得？

- Large Batch Optimization for Deep Learning: Training BERT in 76 minutes (<https://arxiv.org/abs/1904.00962>)
- Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes (<https://arxiv.org/abs/1711.04325>)
- Stochastic Weight Averaging in Parallel: Large-Batch Training That Generalizes Well (<https://arxiv.org/abs/2001.02312>)
- Large Batch Training of Convolutional Networks (<https://arxiv.org/abs/1708.03888>)
- Accurate, large minibatch sgd: Training imagenet in 1 hour (<https://arxiv.org/abs/1706.02677>)

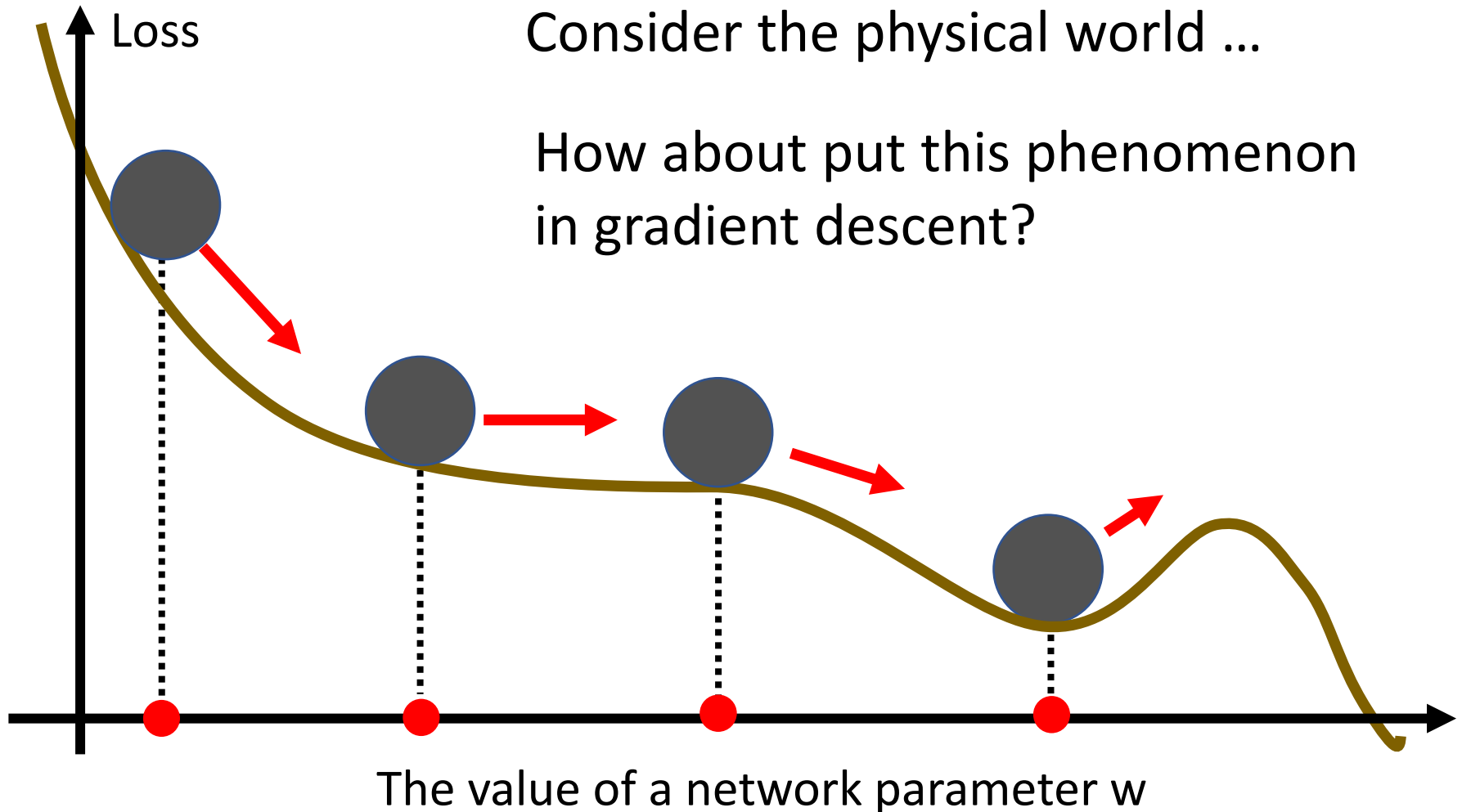
Momentum



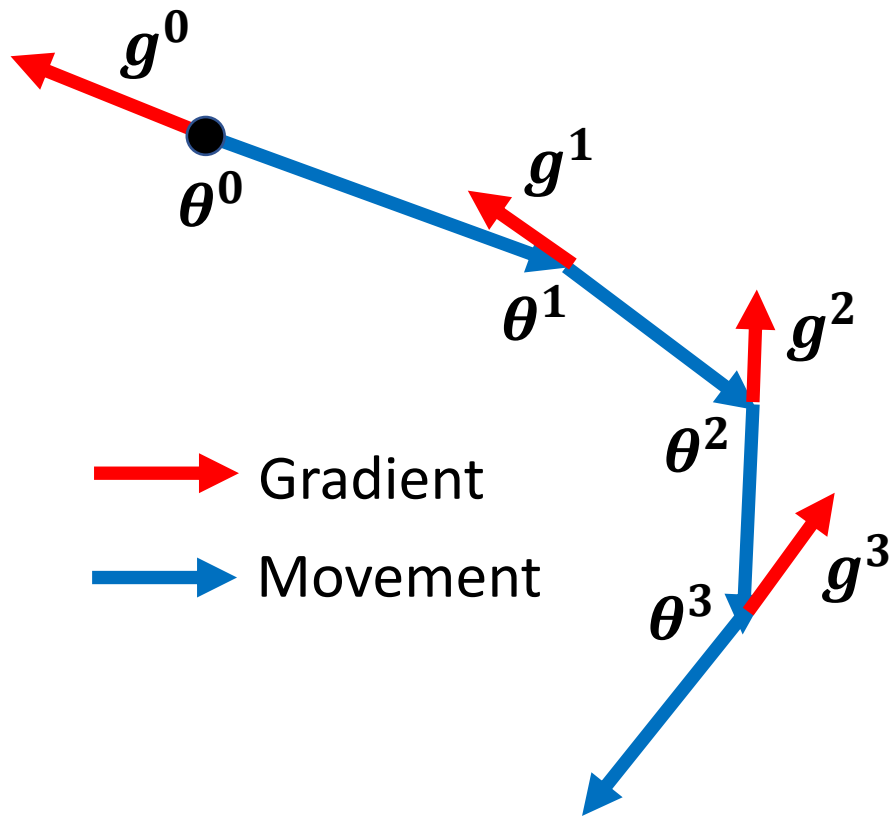
Small Gradient ...

Consider the physical world ...

How about put this phenomenon
in gradient descent?



(Vanilla) Gradient Descent



Starting at θ^0

Compute gradient g^0

Move to $\theta^1 = \theta^0 - \eta g^0$

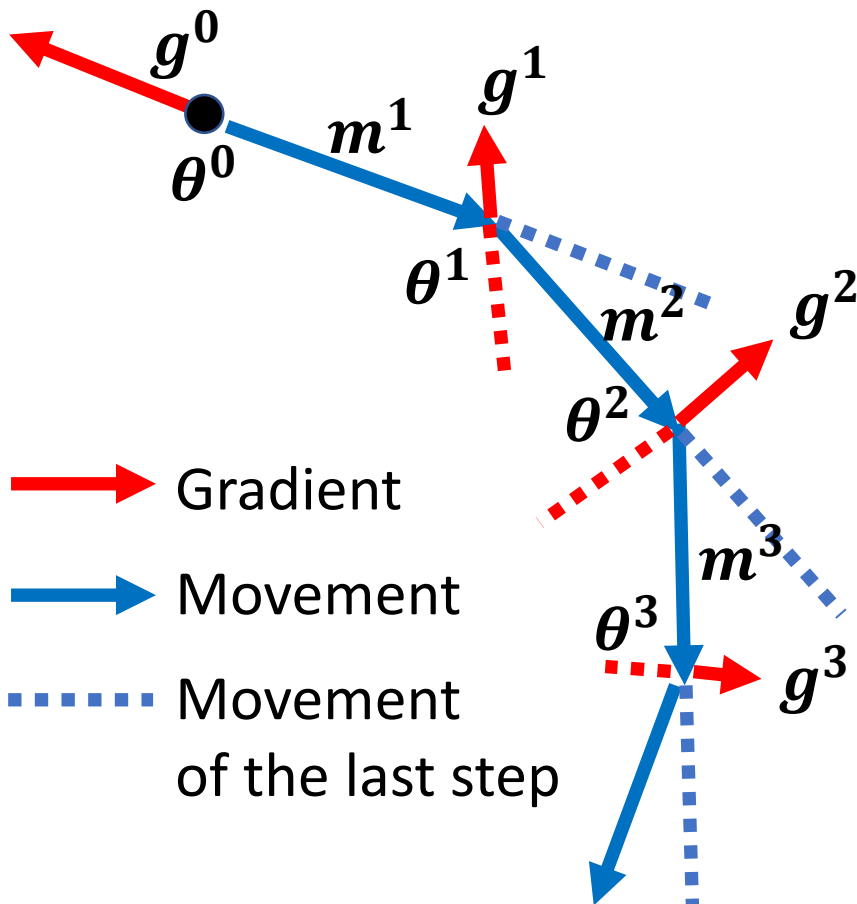
Compute gradient g^1

Move to $\theta^2 = \theta^1 - \eta g^1$

⋮

Gradient Descent + Momentum

Movement: **movement of last step** minus **gradient** at present



Starting at θ^0

Movement $\mathbf{m}^0 = \mathbf{0}$

Compute gradient \mathbf{g}^0

Movement $\mathbf{m}^1 = \lambda \mathbf{m}^0 - \eta \mathbf{g}^0$

Move to $\theta^1 = \theta^0 + \mathbf{m}^1$

Compute gradient \mathbf{g}^1

Movement $\mathbf{m}^2 = \lambda \mathbf{m}^1 - \eta \mathbf{g}^1$

Move to $\theta^2 = \theta^1 + \mathbf{m}^2$

Movement not just based on gradient, but previous movement.

Gradient Descent + Momentum

Movement: **movement of last step** minus **gradient** at present

\mathbf{m}^i is the weighted sum of all the previous gradient: $\mathbf{g}^0, \mathbf{g}^1, \dots, \mathbf{g}^{i-1}$

$$\mathbf{m}^0 = \mathbf{0}$$

$$\mathbf{m}^1 = -\eta \mathbf{g}^0$$

$$\mathbf{m}^2 = -\lambda \eta \mathbf{g}^0 - \eta \mathbf{g}^1$$

\vdots

Starting at $\boldsymbol{\theta}^0$

Movement $\mathbf{m}^0 = \mathbf{0}$

Compute gradient \mathbf{g}^0

Movement $\mathbf{m}^1 = \lambda \mathbf{m}^0 - \eta \mathbf{g}^0$

Move to $\boldsymbol{\theta}^1 = \boldsymbol{\theta}^0 + \mathbf{m}^1$

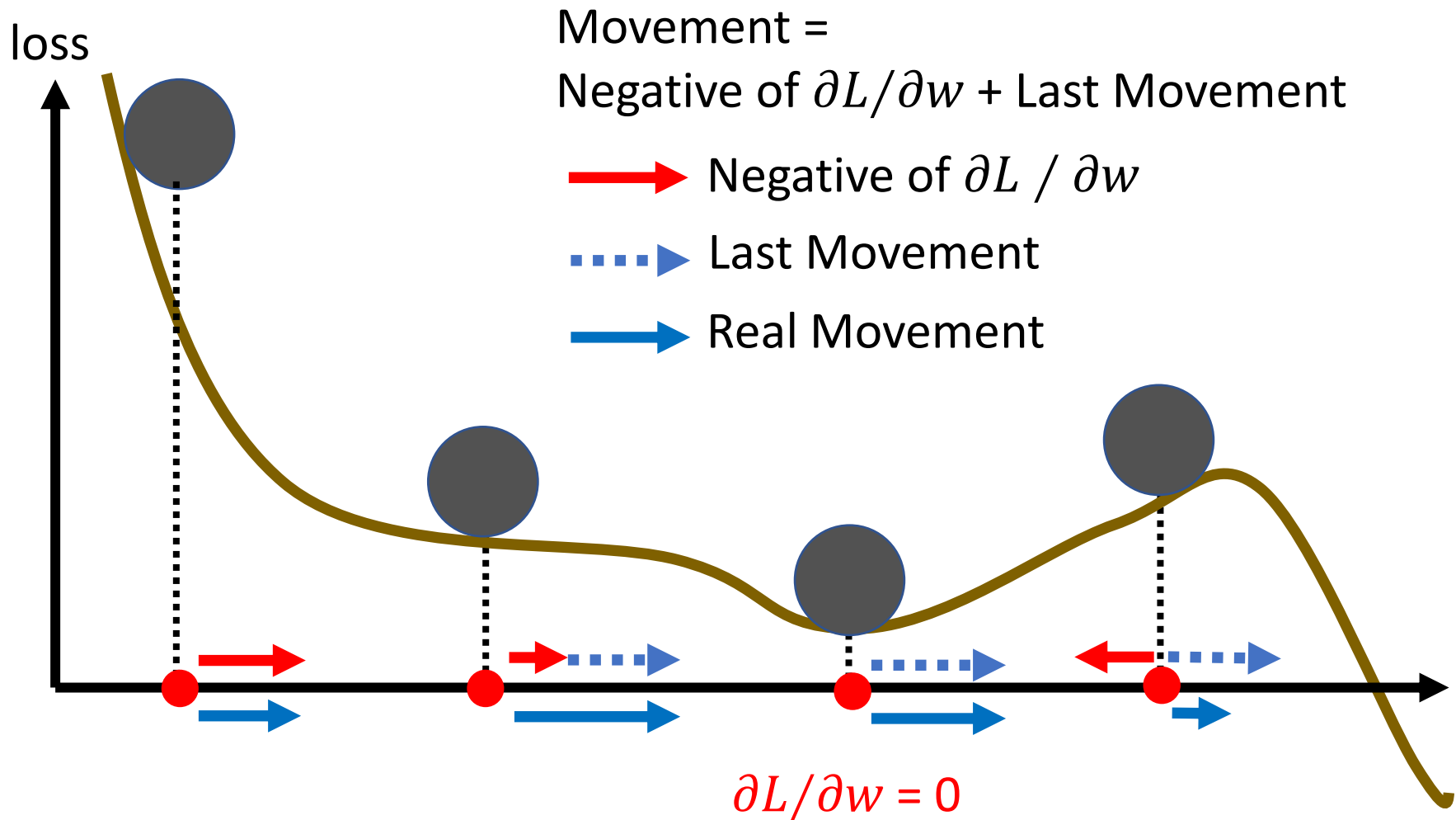
Compute gradient \mathbf{g}^1

Movement $\mathbf{m}^2 = \lambda \mathbf{m}^1 - \eta \mathbf{g}^1$

Move to $\boldsymbol{\theta}^2 = \boldsymbol{\theta}^1 + \mathbf{m}^2$

Movement not just based on gradient, but previous movement.

Gradient Descent + Momentum



Concluding Remarks

- Critical points have zero gradients.
- Critical points can be either saddle points or local minima.
 - Can be determined by the Hessian matrix.
 - It is possible to escape saddle points along the direction of eigenvectors of the Hessian matrix.
 - Local minima may be rare.
- Smaller batch size and momentum help escape critical points.