Gradient Descent

Chun-Hsiang Chan
Undergraduate Program in Intelligent Computing and Big Data, Chung Yuan Christian University
Master Program in Intelligent Computing and Big Data, Chung Yuan Christian University

Outline

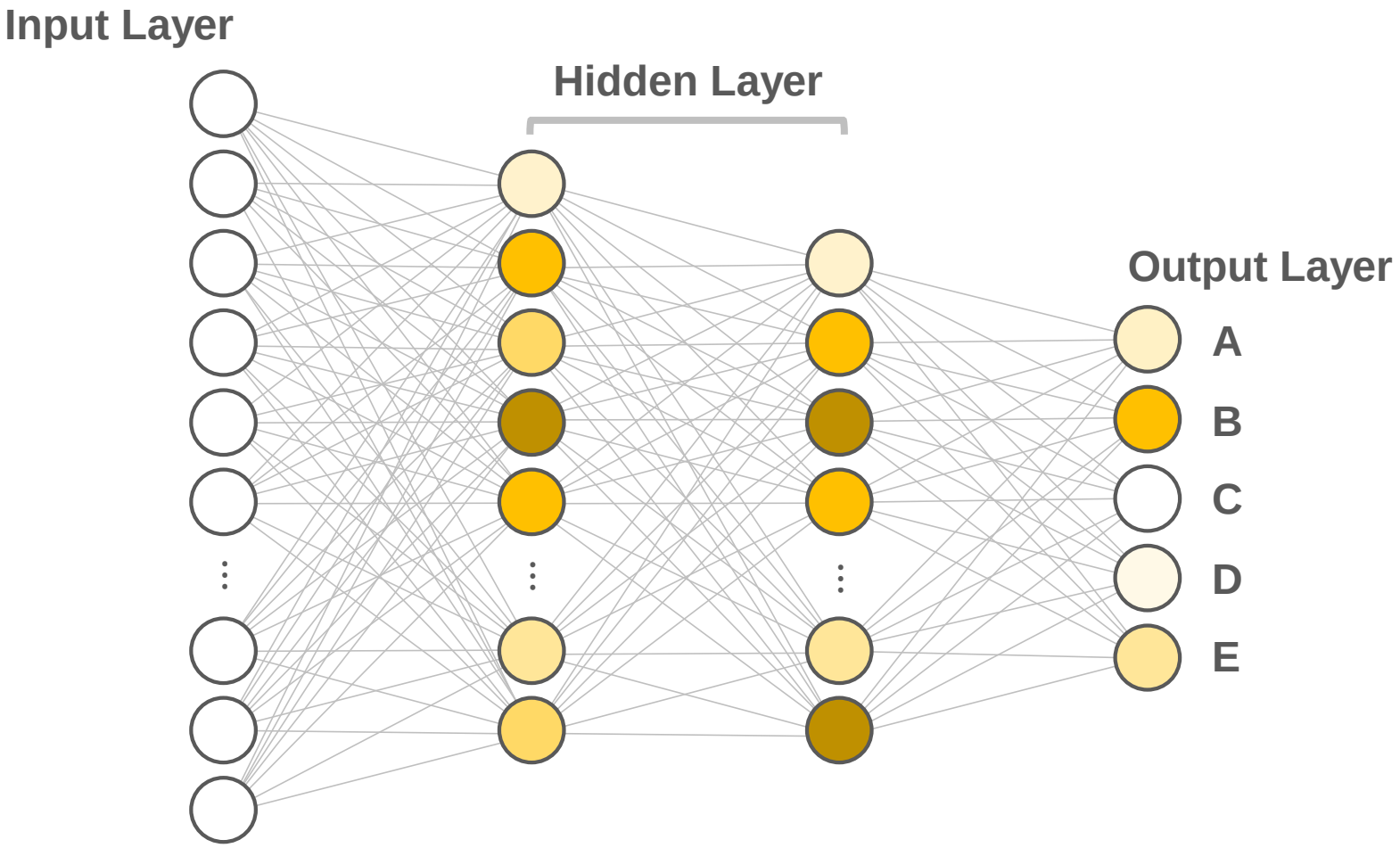
- Finding the Best Parameter Set
- Loss Optimization
- Global Minima and Local Minima
- Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Stochastic Gradient Descent
- Initial Condition
- Learning Rate

Finding the Best Parameter Set

- To optimize the output/ accuracy of the model, we have to find the best parameter set; however, how can we achieve this problem…?
- In practical, we leverage loss/ cost/ error function or reward function to determine the performance of each parameter set.
- Taking the loss function as an example, all you have to do is to find the parameter set, returning a (global or local) minima loss.

Finding the Best Parameter Set

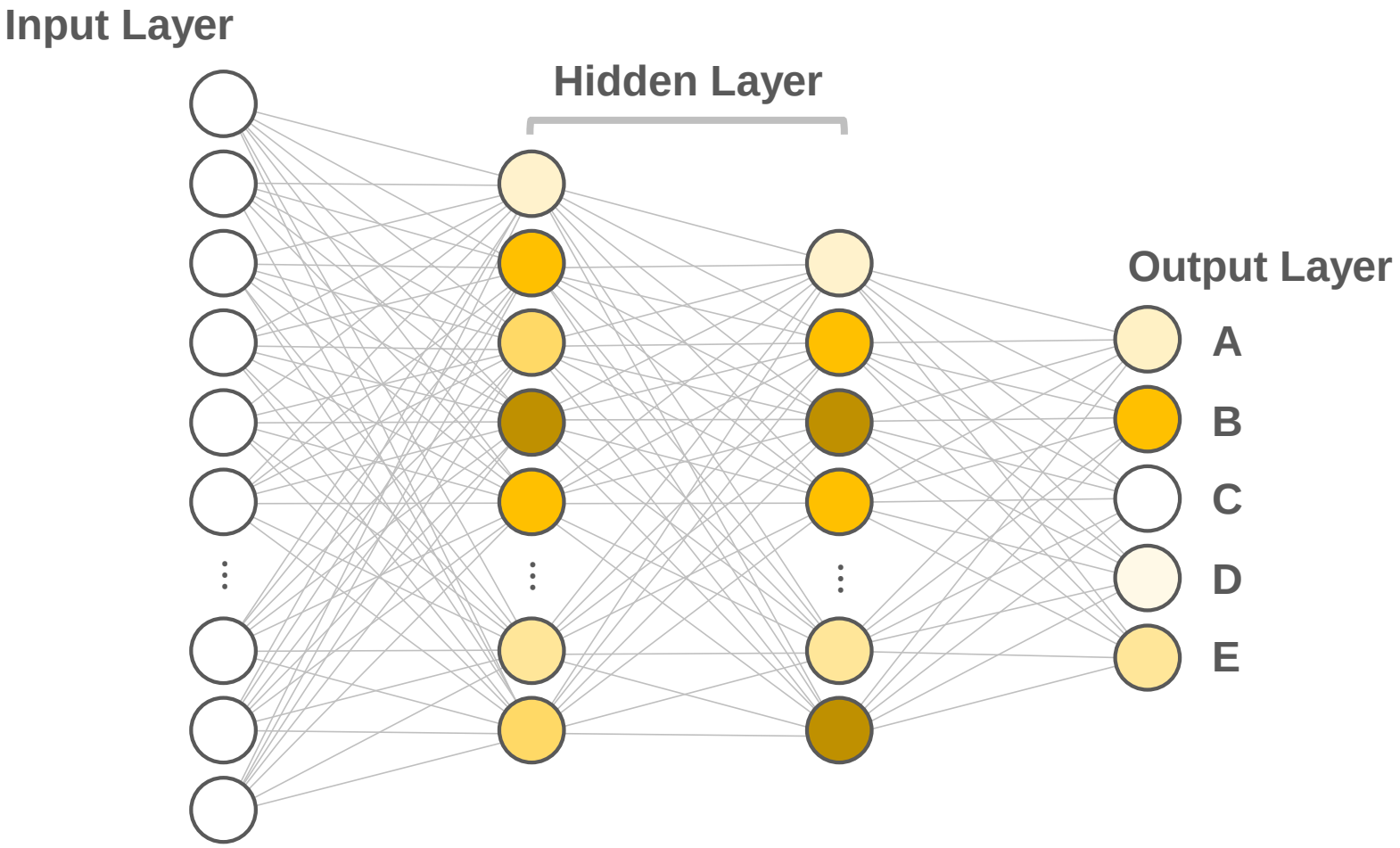
A Neuron Perspective



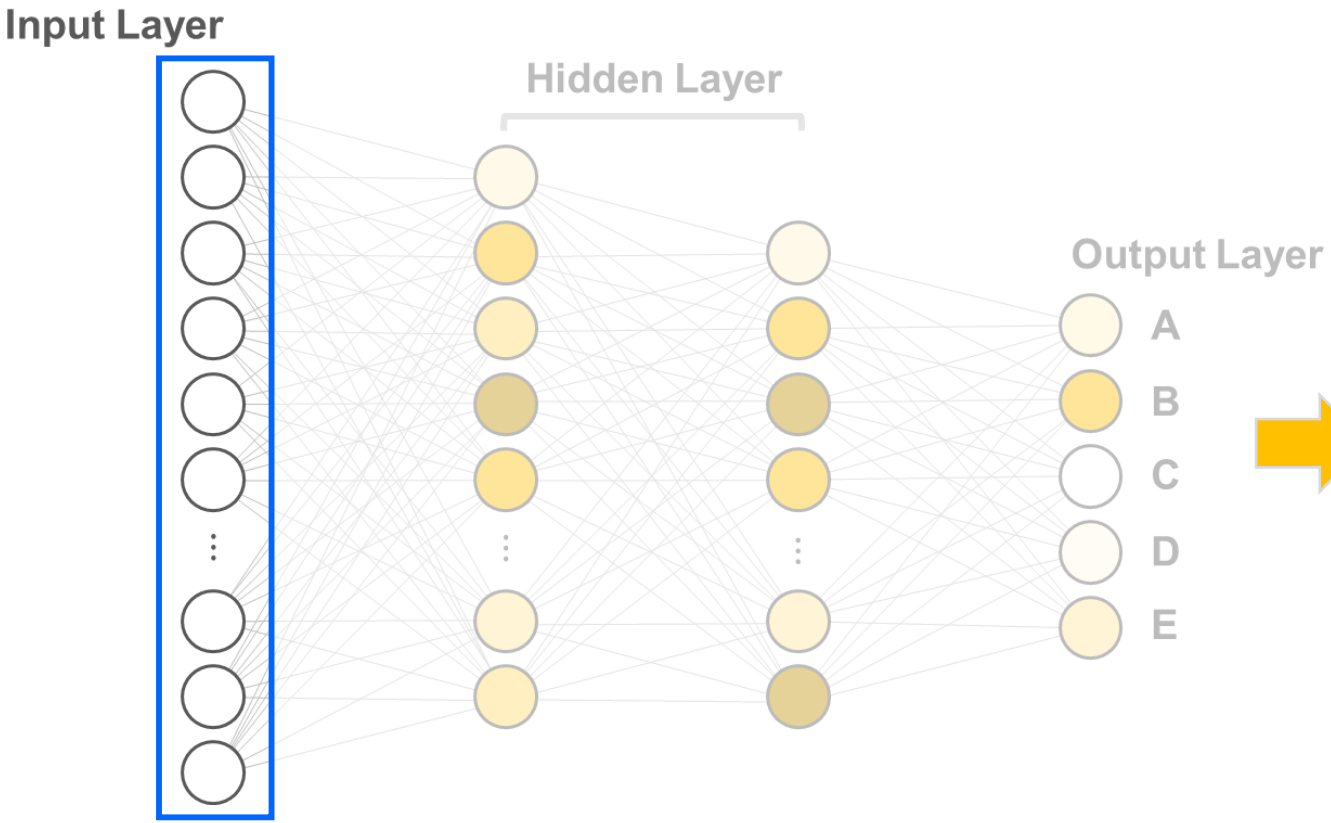
Finding the Best Parameter Set

A Parameter Perspective

Weighting & Bias

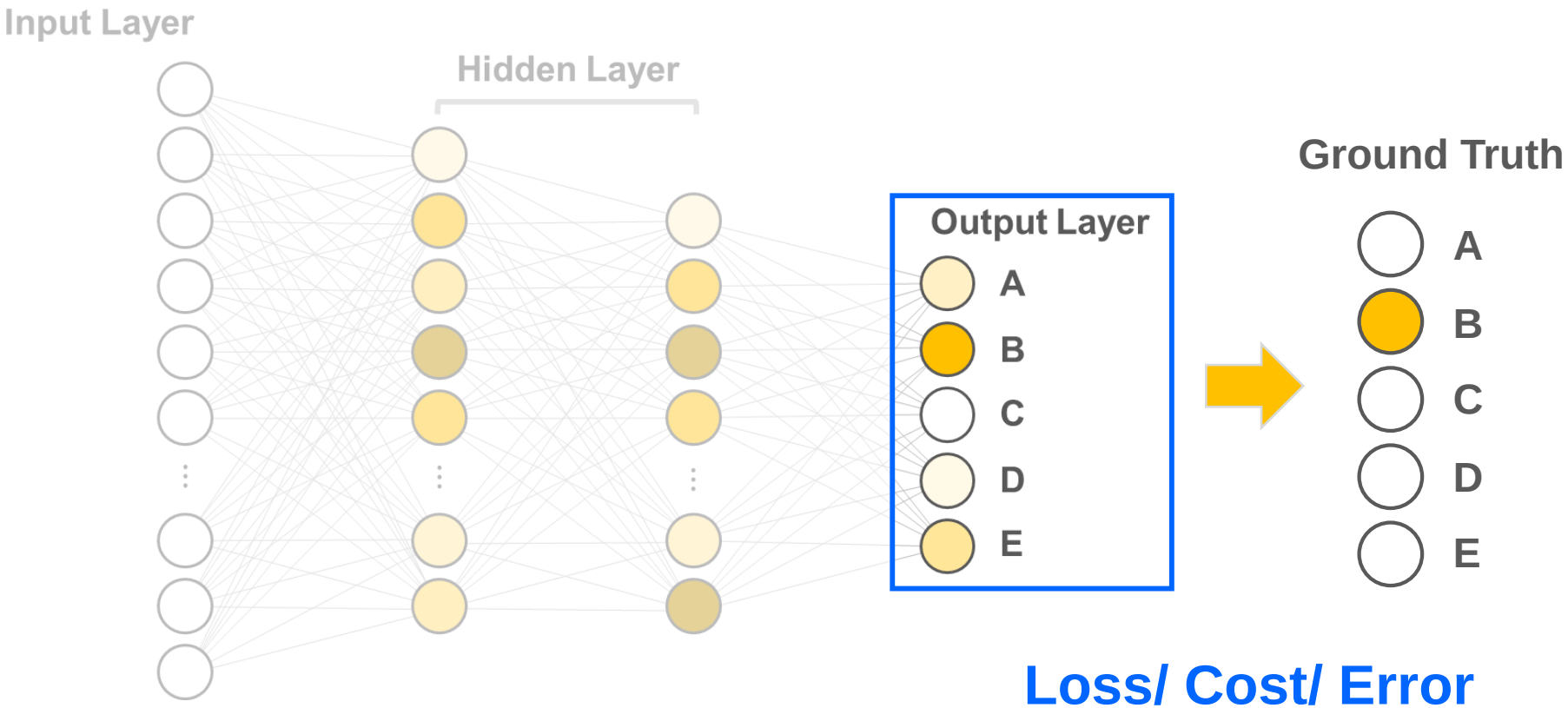


Finding the Best Parameter Set



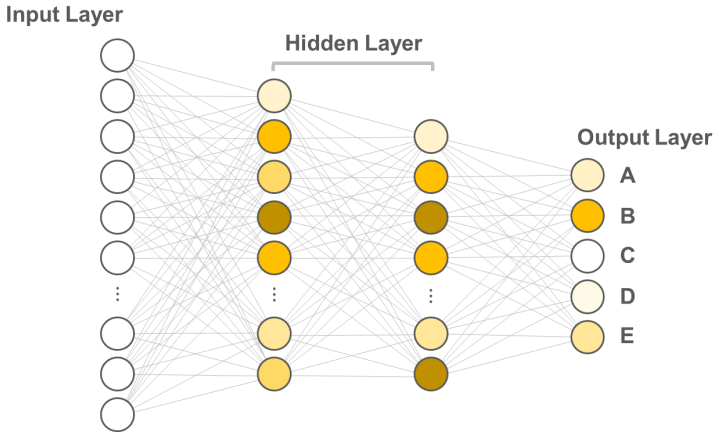
$$\sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n a_n + b)$$

Finding the Best Parameter Set



Loss/ Cost/ Error

Finding the Best Parameter Set

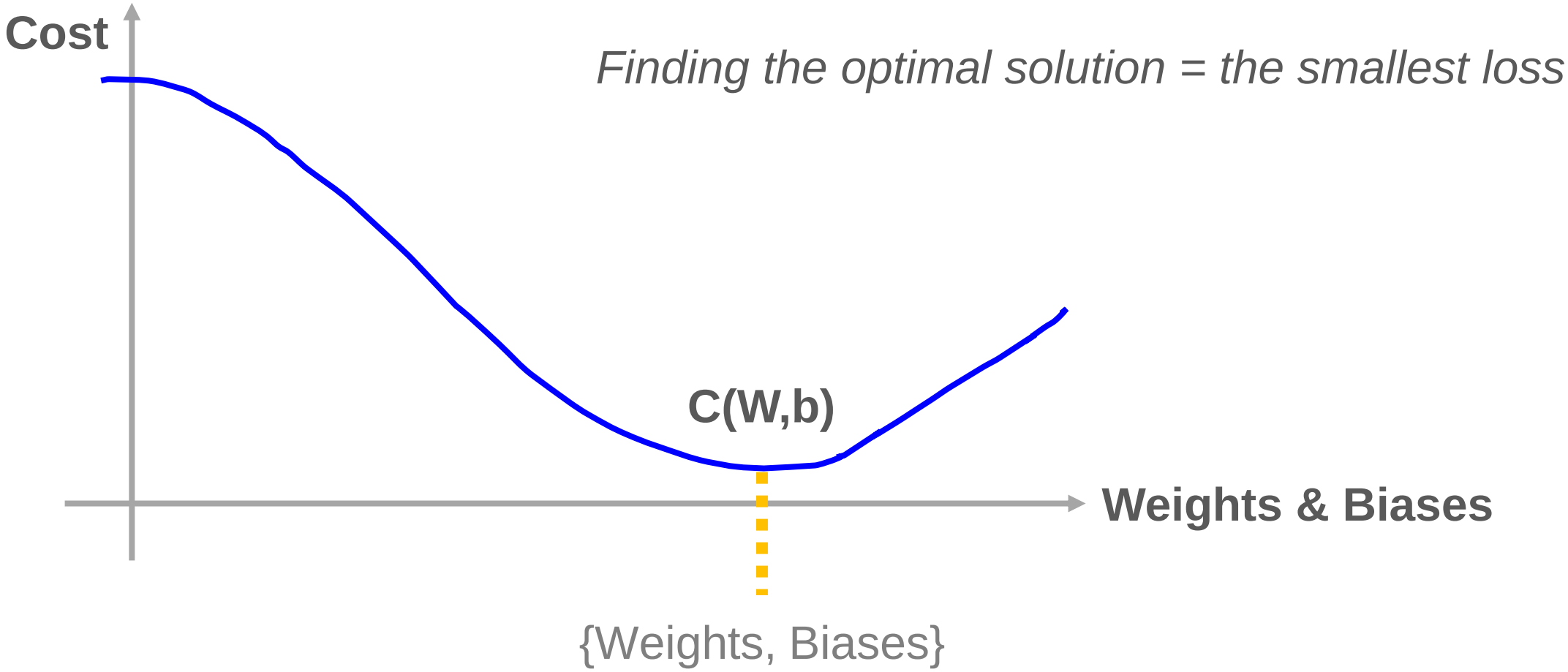


Loss/ Cost/ Error

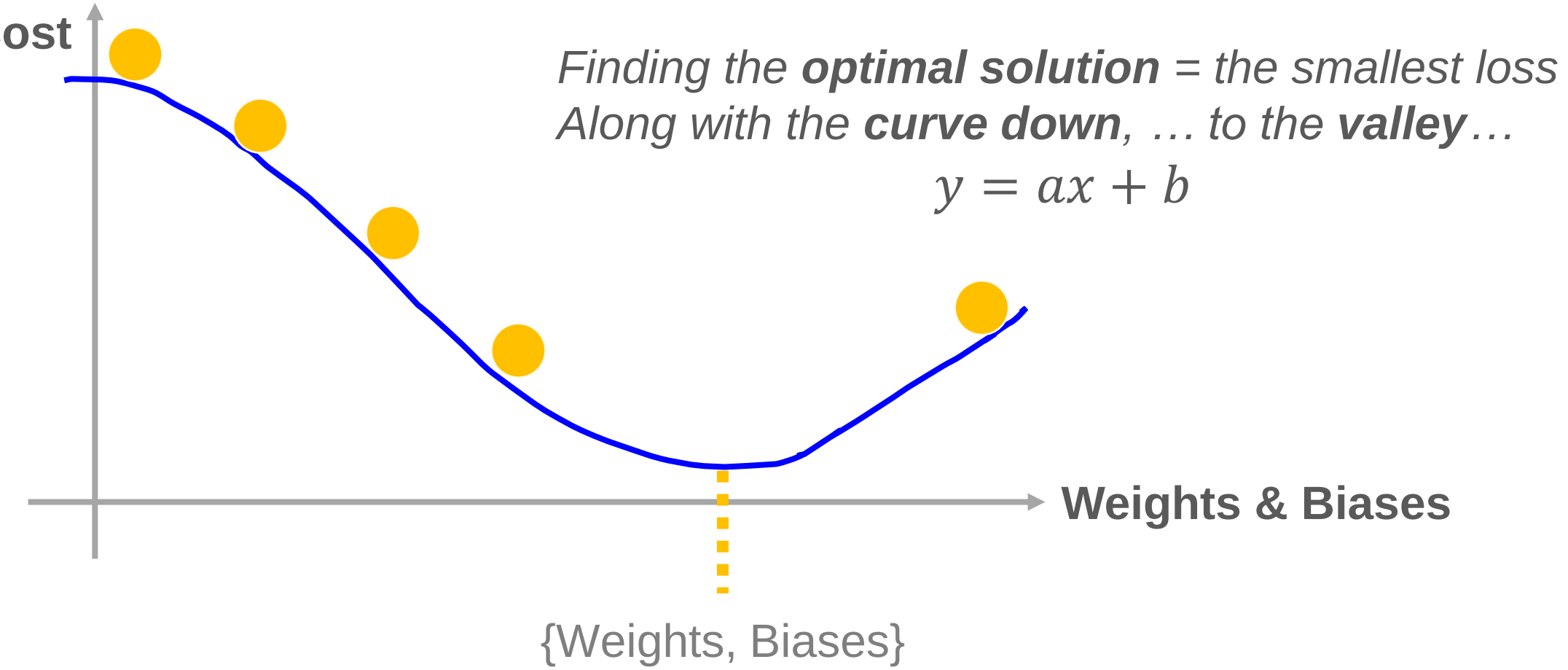
$$C(\underbrace{w_1^1, w_2^1, w_3^1, \dots, w_1^2, w_2^2, \dots, w_n^h}_{\text{Weights}}, \dots, \underbrace{b^1, b^2, \dots, b^h}_{\text{Biases}})$$

Target: minimize the loss/ cost/ error of outputs

Loss Function and Optimization



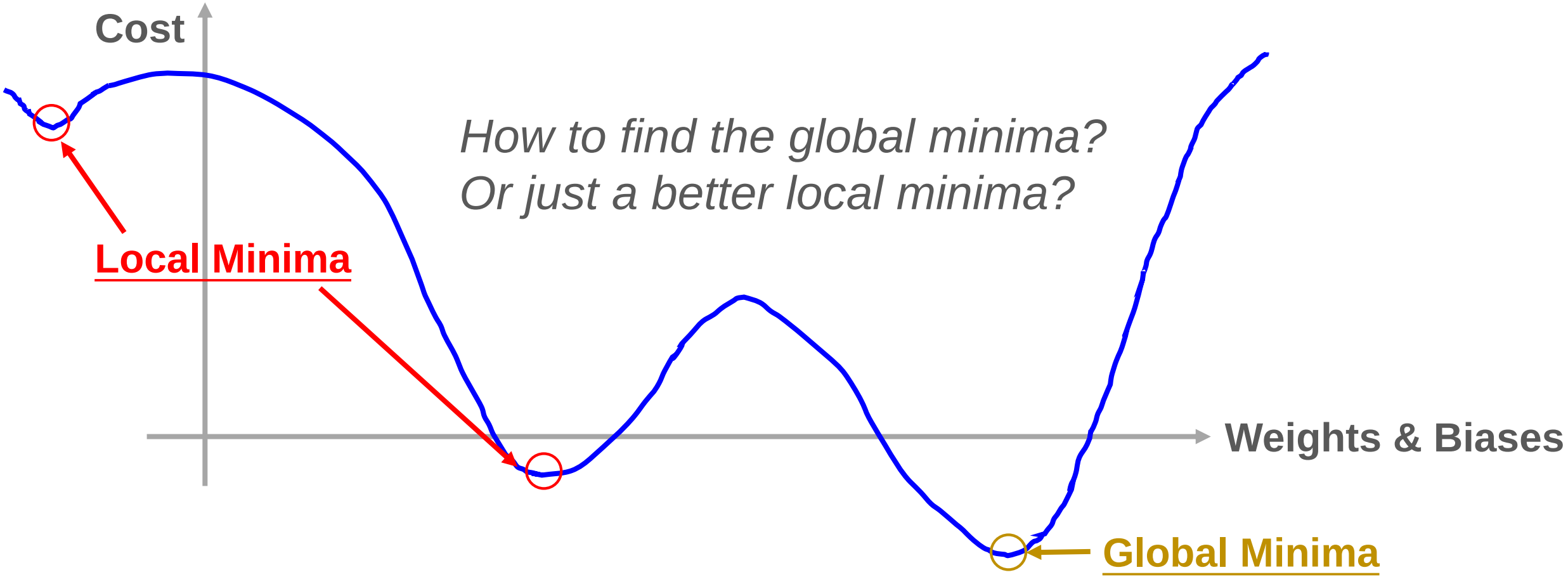
Loss Function and Optimization



*Finding the **optimal solution** = the smallest loss*
*Along with the **curve down**, … to the **valley**…*

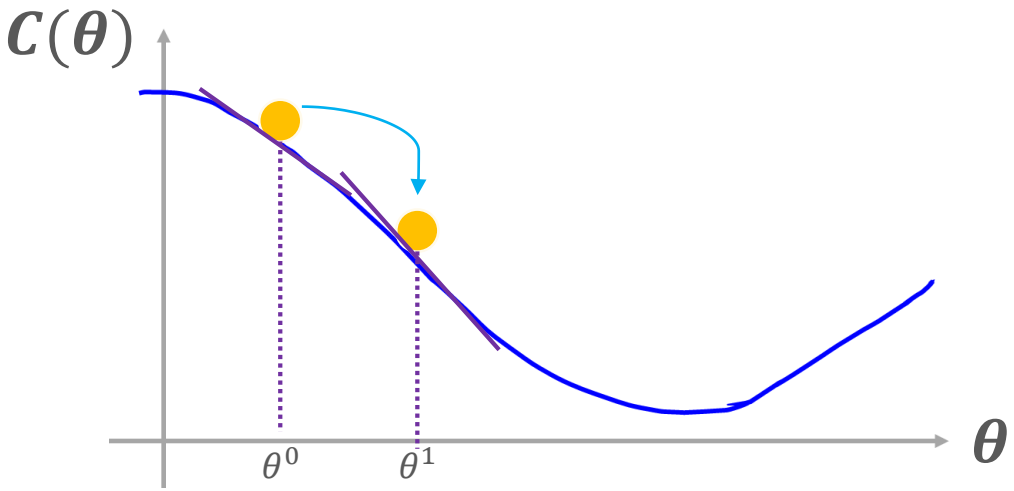
$$y = ax + b$$

Global Minima and Local Minima



Gradient Descent

Which way?
How does it work?



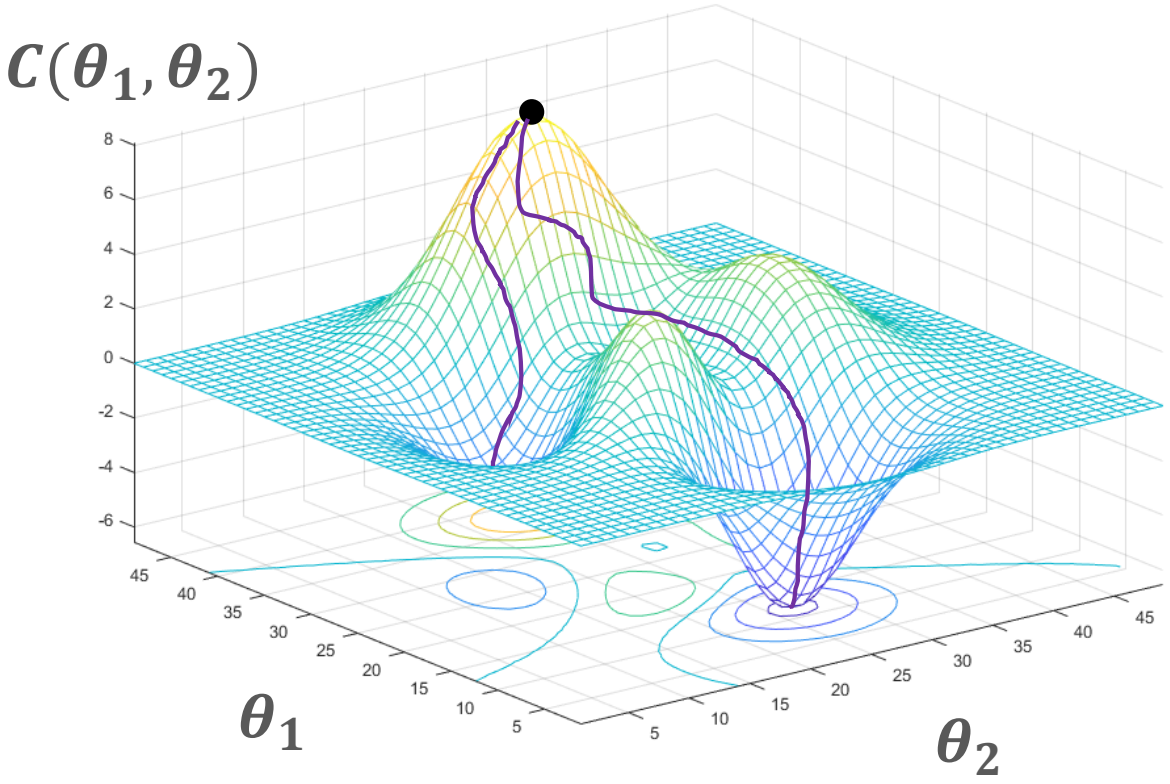
Assume that θ has only one variable…

- Randomly initiate at θ^0
- Compute $\frac{dC(\theta^0)}{d\theta}$: $\theta^1 \leftarrow \theta^0 - \eta\frac{\partial(\theta^0)}{\partial\theta}$
- Compute $\frac{dC(\theta^1)}{d\theta}$: $\theta^2 \leftarrow \theta^1 - \eta\frac{\partial(\theta^1)}{\partial\theta}$
- …

➔ $\theta^{i+1} \leftarrow \theta^i - \eta\nabla_\theta C(\theta^i)$

where η is the learning rate

Gradient Descent



Assume that θ has only one variable…

- Randomly initiate at $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$
- Compute the gradients of $C(\theta)$ at θ^0:

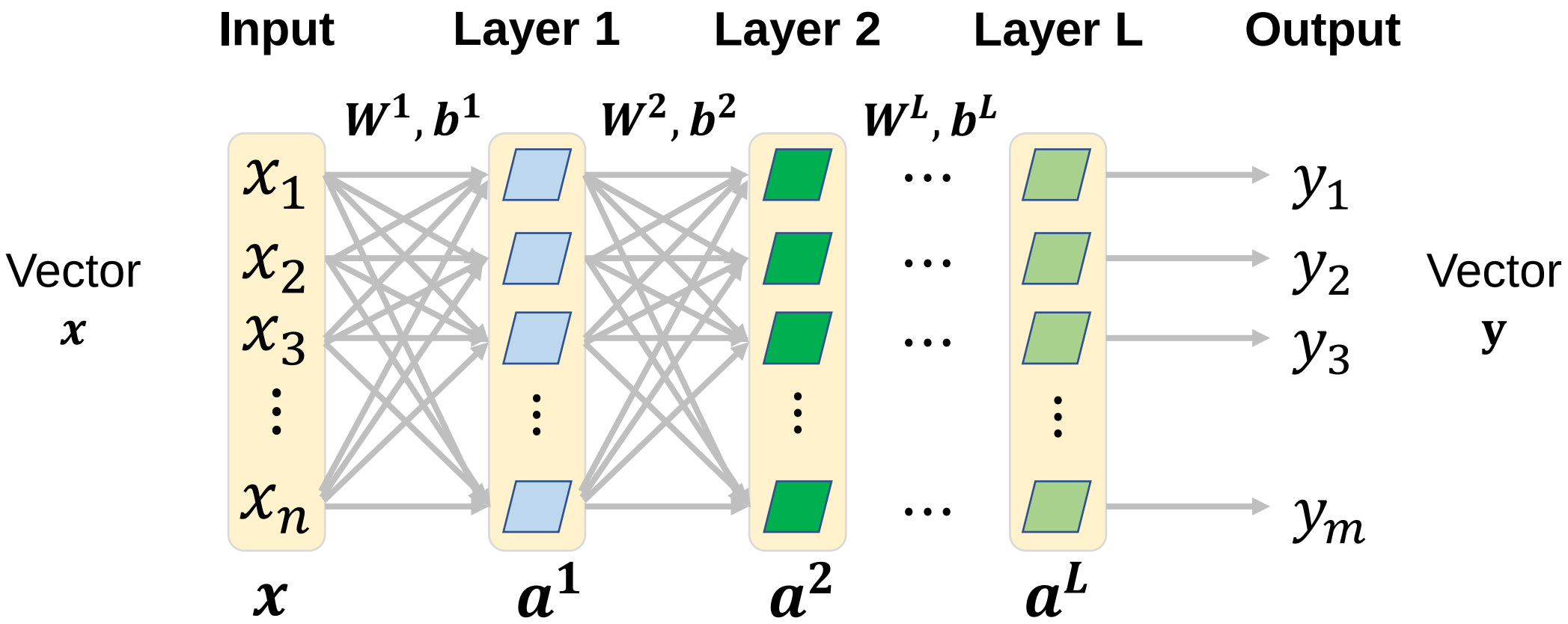
$$\nabla_\theta C(\theta^0): \begin{bmatrix} \frac{\partial C(\theta_1^0)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^0)}{\partial \theta_2} \end{bmatrix}$$

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$$

- Update: $\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial C(\theta_1^0)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^0)}{\partial \theta_2} \end{bmatrix}$
- Compute the gradients of $C(\theta)$ at θ^1:

$$\nabla_\theta C(\theta^1): \begin{bmatrix} \frac{\partial C(\theta_1^1)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^1)}{\partial \theta_2} \end{bmatrix}$$

FNN (fully connected feedforward network)



$$y = f(x) = \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

Gradient Descent

$$y = f(x) = \sigma(W^L \dots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \dots + b^L)$$
$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$
$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$
$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while $(\theta^{(i+1)} \neq \theta^i)$

{

compute the gradients at θ^i

update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$

}

Gradient Descent – Simple Case | Square Loss

- Update three parameters for $t-th$ iteration

$$w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial C\left(\theta^{(t)}\right)}{\partial w_1}$$

$$w_2^{(t+1)} = w_2^{(t)} - \eta \frac{\partial C\left(\theta^{(t)}\right)}{\partial w_2}$$

$$b_1^{(t+1)} = b_1^{(t)} - \eta \frac{\partial C\left(\theta^{(t)}\right)}{\partial b_1}$$

Square loss

$$C(\theta) = \sum_{\forall x} \|\hat{y} - f(x;\theta)\| = \left(\hat{y} - f(x;\theta)\right)^2$$

$$\begin{bmatrix} w_1^{i+1} \\ w_2^{i+1} \\ b^{i+1} \end{bmatrix} \leftarrow \begin{bmatrix} w_1^{i} \\ w_2^{i} \\ b^{i} \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial C(\theta)}{\partial w_1} \\ \frac{\partial C(\theta)}{\partial w_2} \\ \frac{\partial C(\theta)}{\partial b} \end{bmatrix}$$

Gradient Descent – Simple Case | Square Loss

- **Square Loss**

$$\frac{\partial C(\theta)}{\partial w_1} = \frac{\partial}{\partial w_1}(f(x;\theta) - \hat{y})^2$$

$$\frac{\partial C(\theta)}{\partial w_1} = 2(f(x;\theta) - \hat{y})\frac{\partial}{\partial w_1} f(x;\theta)$$

$$\frac{\partial C(\theta)}{\partial w_1} = 2(\sigma(Wx + b) - \hat{y})\frac{\partial}{\partial w_1}\sigma(Wx + b)$$

$f(x;\theta) = \sigma(Wx + b)$

Gradient Descent – Simple Case | Square Loss

$$\frac{\partial \sigma(Wx+b)}{\partial w_1} = \frac{\partial \sigma(Wx+b)}{\partial (Wx+b)} \frac{\partial (Wx+b)}{\partial w_1}$$

Chain rule: $\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$

$$\frac{\partial g(z)}{\partial z} = [1-g(z)]g(z), where\ g(z) = \frac{1}{1+e^{-x}}$$

$$= [1-\sigma(Wx+b)]\sigma(Wx+b)\frac{\boldsymbol{\partial (Wx+b)}}{\boldsymbol{\partial w_1}}$$

$$\frac{\boldsymbol{\partial (Wx+b)}}{\boldsymbol{\partial w_1}} = \frac{\partial (w_1x_1 + w_2x_2 + b)}{\partial w_1} = x_1 \Longrightarrow [1-\sigma(Wx+b)]\sigma(Wx+b)x_1$$

$$\frac{\partial C(\theta)}{\partial w_1} = 2(\sigma(Wx+b)-\hat{y})[1-\sigma(Wx+b)]\sigma(Wx+b)x_1$$

Gradient Descent – Simple Case | Square Loss

Update three parameters for $t-th$ iteration

$$w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial w_1}$$

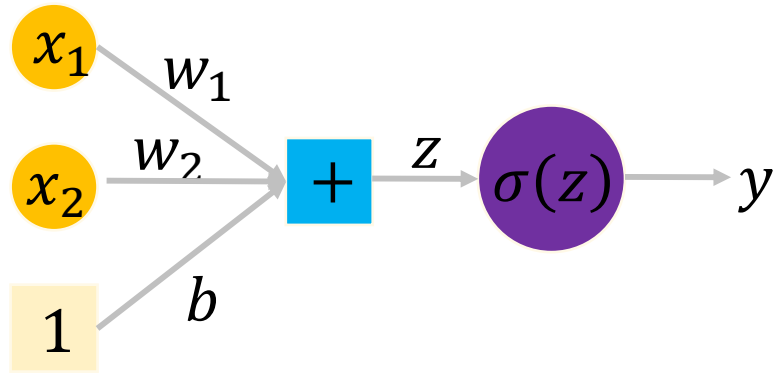
$$\frac{\partial C(\theta)}{\partial w_1} = 2(\sigma(Wx+b) - \hat{y})[1-\sigma(Wx+b)]\sigma(Wx+b)x_1$$

$$w_2^{(t+1)} = w_2^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial w_2}$$

$$\frac{\partial C(\theta)}{\partial w_2} = 2(\sigma(Wx+b) - \hat{y})[1-\sigma(Wx+b)]\sigma(Wx+b)x_2$$

$$b^{(t+1)} = b^{(t)} - \eta \frac{\partial C(\theta^{(t)})}{\partial b}$$

$$\frac{\partial C(\theta)}{\partial b} = 2(\sigma(Wx+b) - \hat{y})[1-\sigma(Wx+b)]\sigma(Wx+b)$$



Gradient Descent – Simple Case | Square Loss

Algorithm

Initialization: set parameter θ, b at random

while (does not meet the terminating criteria)

{

for training sample $\{x, \hat{y}\}$, compute gradient and update parameters

$$\theta^{i+1} \leftarrow \theta^{i} - \eta \nabla_{\theta} C\left(\theta^{i}\right)$$

}

$$w_1^{(t+1)} = w_1^{(t)} - \eta \frac{\partial C\left(\theta^{(t)}\right)}{\partial w_1} \mid \frac{\partial C(\theta)}{\partial w_1} = 2(\sigma(Wx+b) - \hat{y})[1 - \sigma(Wx+b)]\sigma(Wx+b)x_1$$

$$w_2^{(t+1)} = w_2^{(t)} - \eta \frac{\partial C\left(\theta^{(t)}\right)}{\partial w_2} \mid \frac{\partial C(\theta)}{\partial w_2} = 2(\sigma(Wx+b) - \hat{y})[1 - \sigma(Wx+b)]\sigma(Wx+b)x_2$$

$$b^{(t+1)} = b^{(t)} - \eta \frac{\partial C\left(\theta^{(t)}\right)}{\partial b} \mid \frac{\partial C(\theta)}{\partial b} = 2(\sigma(Wx+b) - \hat{y})[1 - \sigma(Wx+b)]\sigma(Wx+b)$$

Gradient Descent

$$y = f(x) = \sigma(W^L \dots \sigma(W^2\sigma(W^1 x + b^1) + b^2) \dots + b^L)$$

$$\theta = \{W^1, b^1, W^2, b^2, \dots, W^L, b^L\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm

Initialization: start at θ^0

while $(\theta^{(i+1)} \neq \theta^i)$

{

compute the gradients at θ^i

update parameters

$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$

}

Gradient descent in neural network requires to update millions of parameters …
However, it is time-consuming and an inefficient process…
Therefore, we will introduce another solution – **backpropagation**.

Gradient Descent Problem

$$\theta^{i+1} = \theta^{i} - \eta \nabla C(\theta^{i})$$

$$C(\theta) = \frac{1}{K}\sum_{k} \| f(x_k;\theta) - \widehat{y_k} \| = \frac{1}{K}\sum_{k} C_k(\theta)$$

$$\nabla C(\theta^{i}) = \frac{1}{K}\sum_{k} \nabla C_k(\theta^{i})$$

According to this formula, the parameters will be updated after processing all training samples ➔ too slow & inefficient process

Stochastic Gradient Descent (SGD)

Gradient Descent

$$\theta^{i+1} = \theta^{i} - \eta \nabla C(\theta^{i}), \nabla C(\theta^{i}) = \frac{1}{K}\sum_{k} \nabla C_k(\theta^{i})$$

Stochastic Gradient Descent (SGD)

➔Select a training sample x_k

$$\theta^{i+1} = \theta^{i} - \eta \nabla C_k(\theta^{i})$$

➔If all training samples have same probability (uniform probability) to be selected

$$E[\nabla C_k(\theta^{i})] = \frac{1}{K}\sum_{k} \nabla C_k(\theta^{i})$$

The model updates after processing each selected training sample ➔ much faster

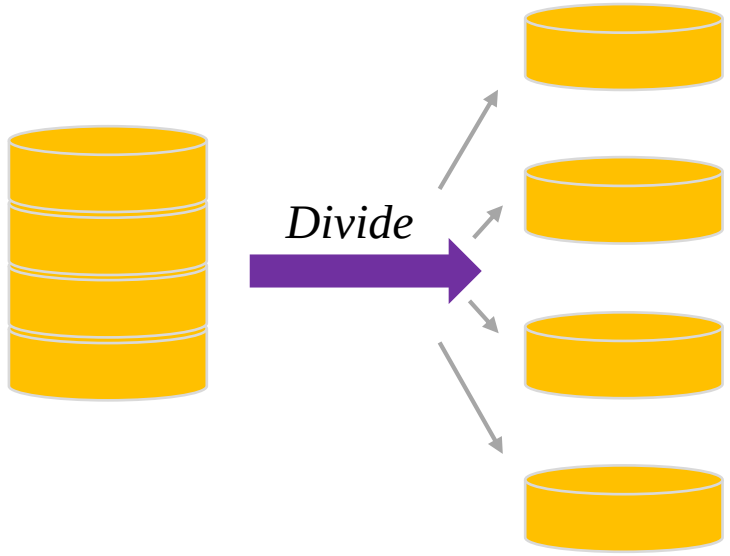
Epoch Definition

Pick x_1 $\theta^1 = \theta^0 - \eta\nabla C_1(\theta^0)$

Pick x_2 $\theta^2 = \theta^1 - \eta\nabla C_2(\theta^1)$

Pick x_k $\theta^k = \theta^{k-1} - \eta\nabla C_k(\theta^{k-1})$

Pick x_K $\theta^K = \theta^{K-1} - \eta\nabla C_K(\theta^{K-1})$

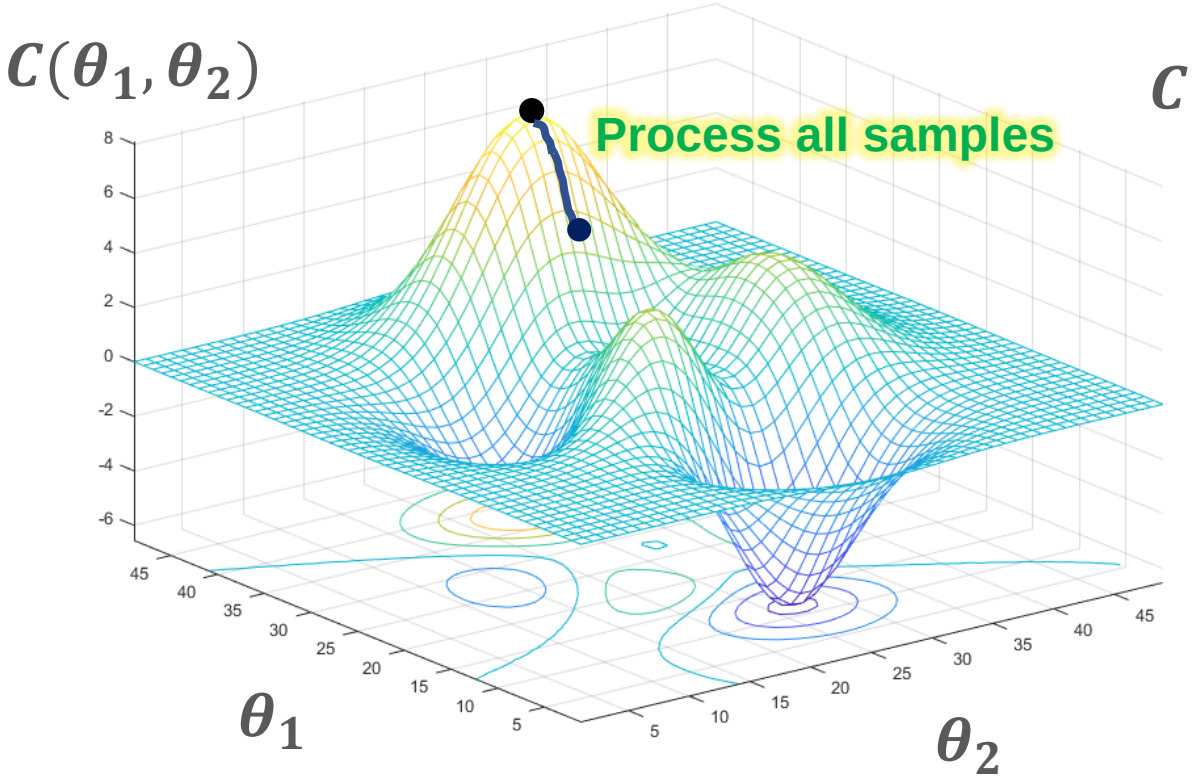


One Epoch

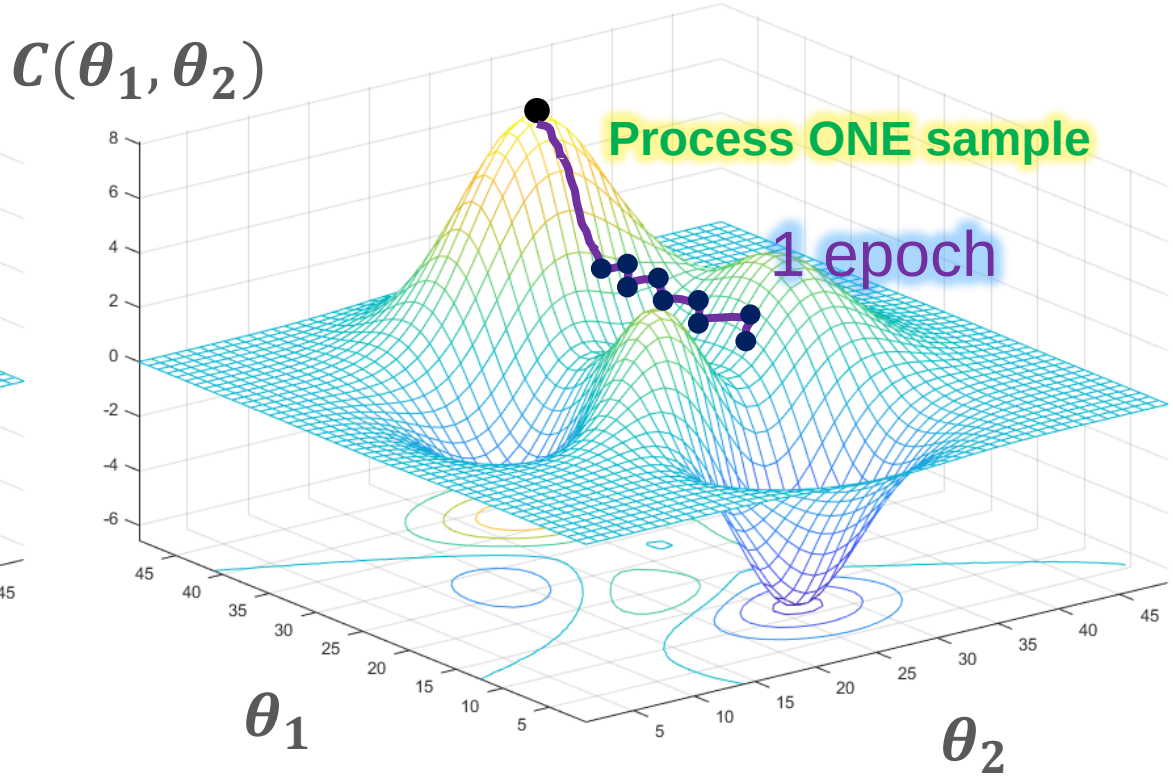
Pick x_1 $\theta^{K+1} = \theta^K - \eta\nabla C_1(\theta^K)$

Gradient Descent & Stochastic Gradient Descent

Gradient Descent

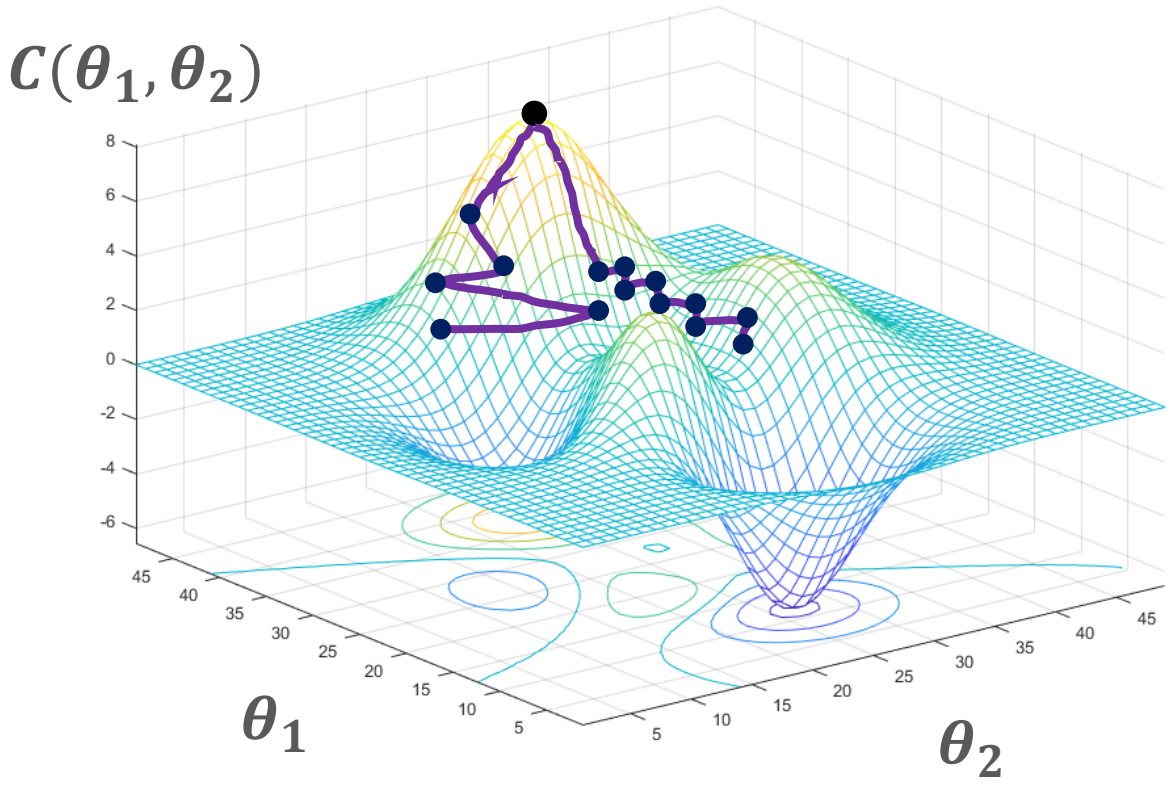


Stochastic Gradient Descent



Gradient Descent & Stochastic Gradient Descent

- Theoretically, the training speed of stochastic gradient descent will be much faster than gradient descent because the update frequency of SGD is higher than gradient descent.
- However, … is it good enough for parameter optimization in neural network?



Mini-Batch Stochastic Gradient Descent

Batch Gradient Descent	$\theta^{i+1} = \theta^i - \eta \frac{1}{K} \sum_k \nabla C_k(\theta^i)$	Use all K samples in each iteration
Stochastic Gradient Descent Select ONE training sample x_k	$\theta^{i+1} = \theta^i - \eta \nabla C_k(\theta^i)$	Use 1 samples in each iteration
Mini-Batch SGD Select a SET of B training samples of each iteration as a batch	$\theta^{i+1} = \theta^i - \eta \frac{1}{B} \sum_{x_k \in b} \nabla C_k(\theta^i)$	Use all B samples in each iteration

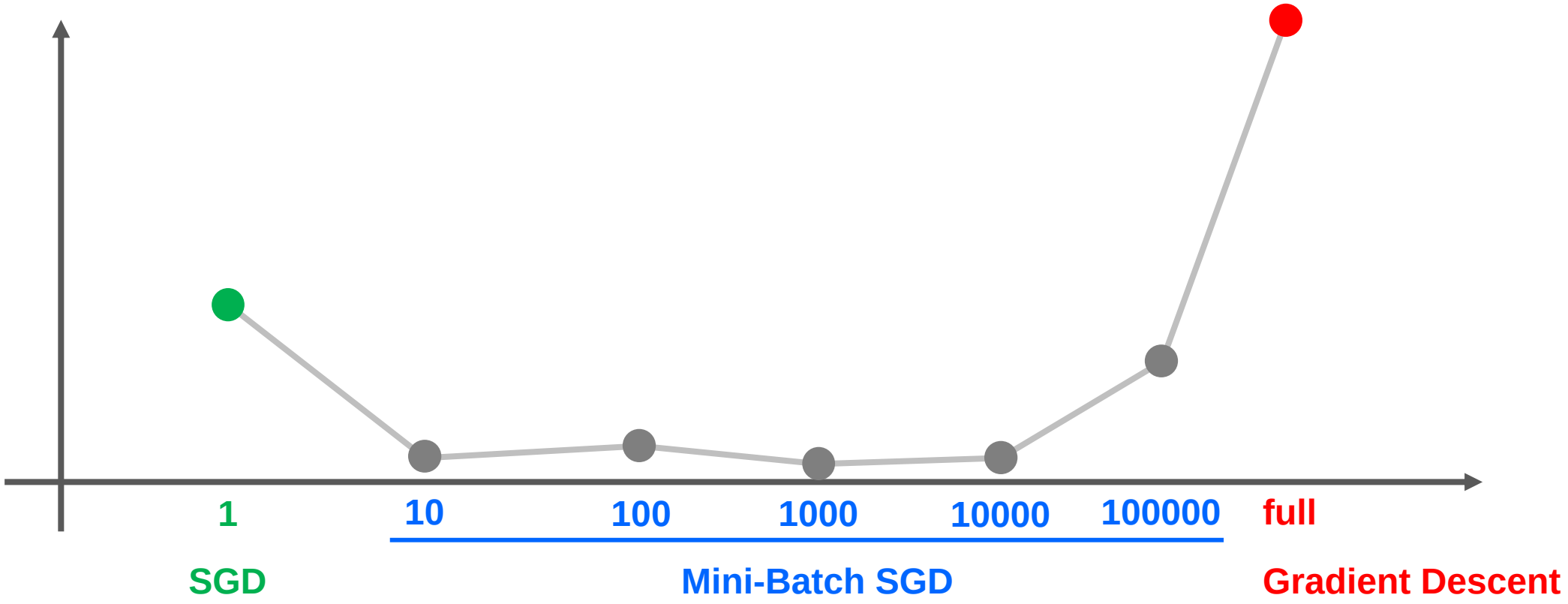
Mini-Batch Stochastic Gradient Descent

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
Sample a minibatch of $\boldsymbol{m}$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L\left(\boldsymbol{f}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{\epsilon}\hat{\boldsymbol{g}}$
end while

Mini-Batch Stochastic Gradient Descent

- **Training speed:** mini-batch > SGD > gradient descent



Mini-Batch Stochastic Gradient Descent

- **Stochastic Gradient Descent**

$$\begin{bmatrix} \vdots \\ z \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \cdots \\ \vdots & W & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ x \\ \vdots \end{bmatrix}, \begin{bmatrix} \vdots \\ z \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \cdots \\ \vdots & W & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ x \\ \vdots \end{bmatrix}, \ldots$$

- **Mini-Batch Stochastic Gradient Descent**

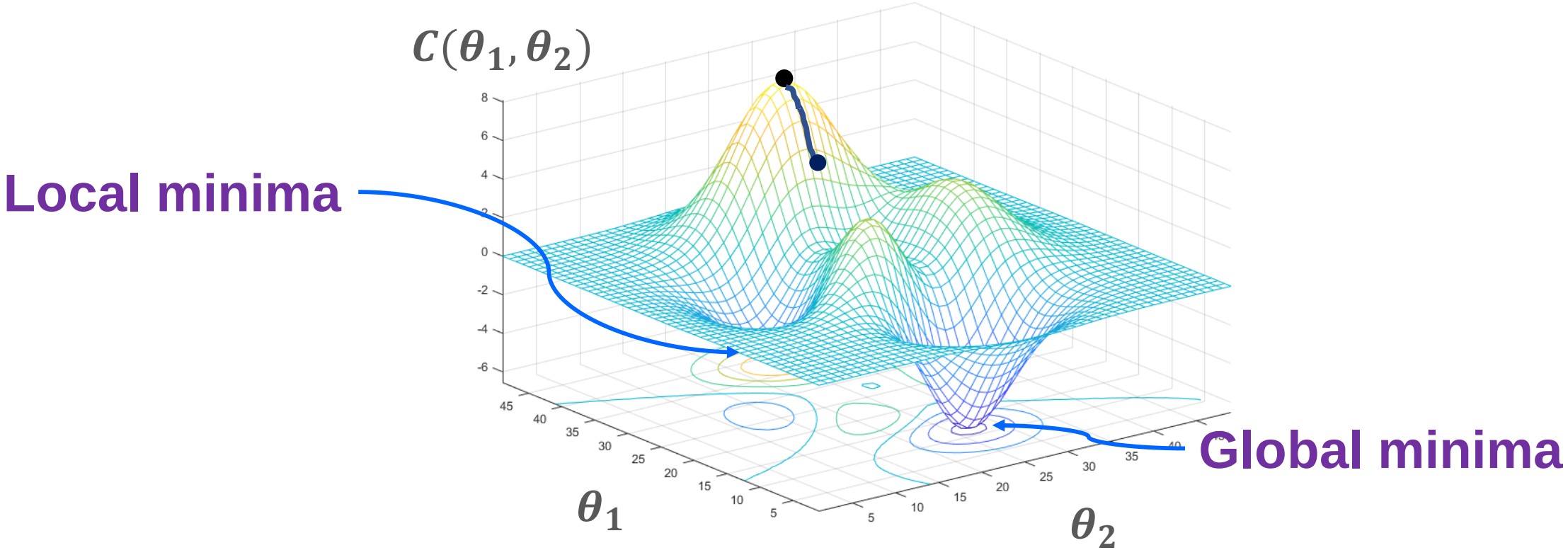
$$\begin{bmatrix} \vdots \\ z \\ \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ z \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \cdots \\ \vdots & W & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ x \\ \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ x \\ \vdots \end{bmatrix}$$

Vectors merges in a Matrix

Vectors merges in a Matrix

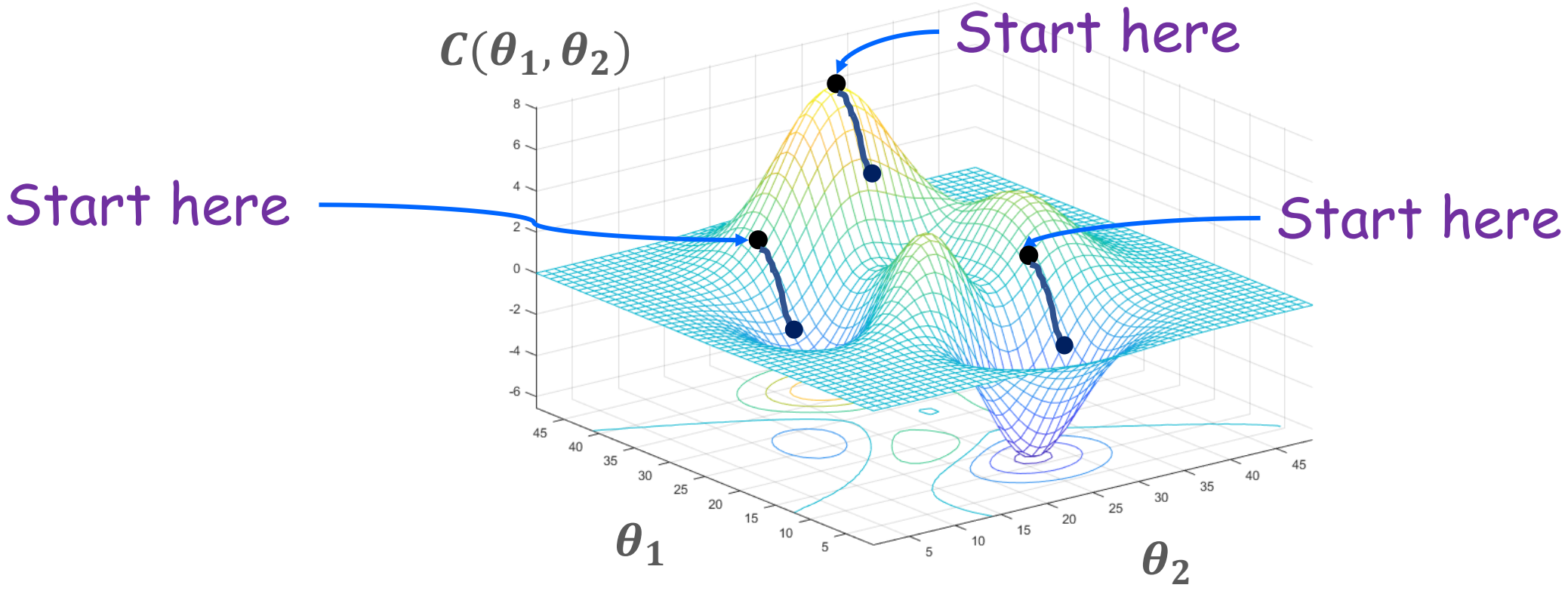
Mini-Batch Solve All Training Problems?

How to avoid obtaining a solution with a local minima?



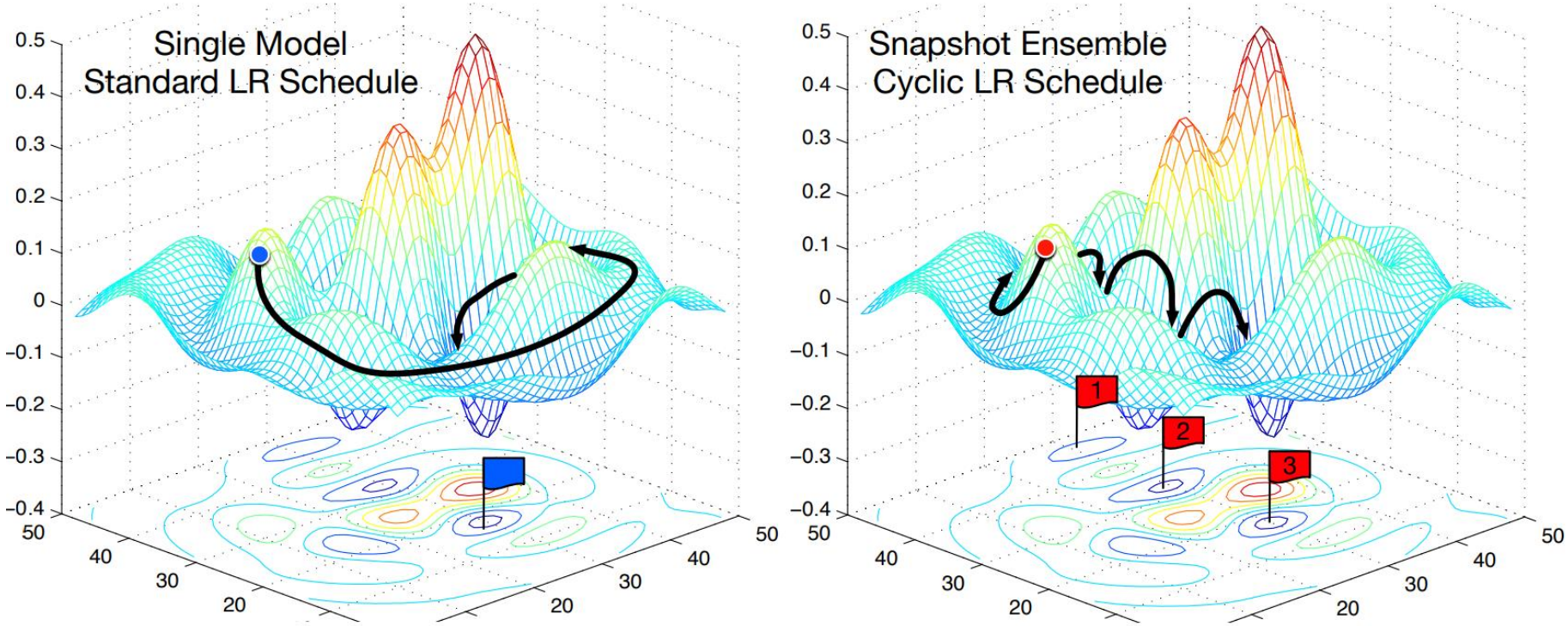
In Addition to Parameter Optimization,…

Two major factors have strong impacts on model training:
initial condition and **learning rate**.



Learning Rate

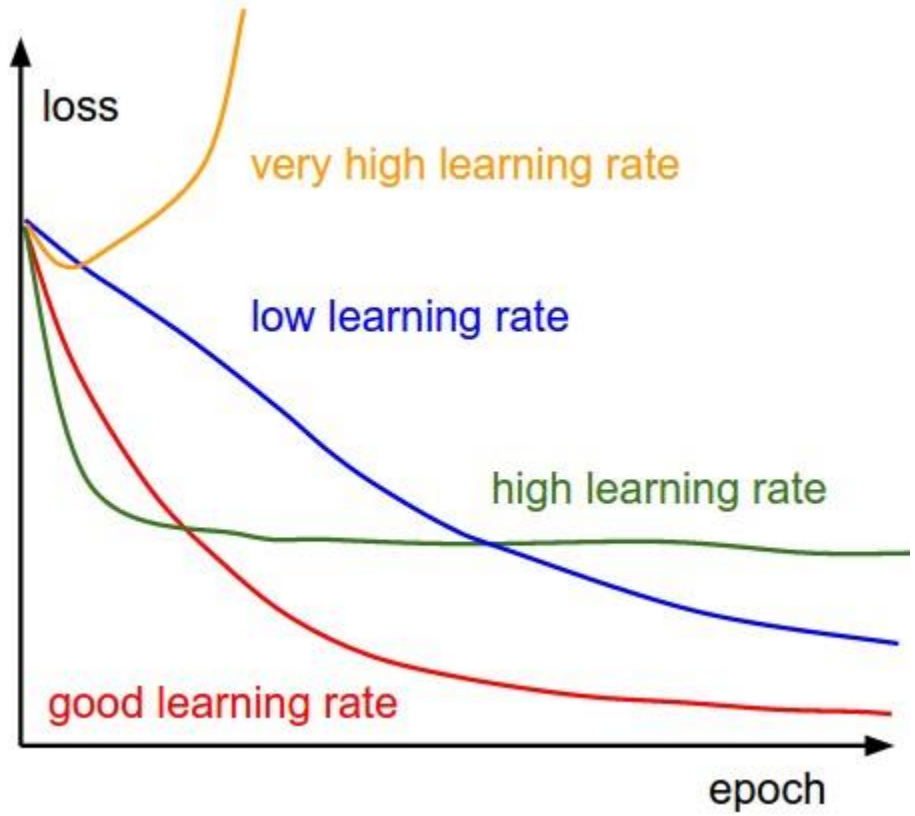
$$\theta^{i+1} = \theta^{i} - \eta \nabla C_k\left(\theta^{i}\right)$$



Left: Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right:** Illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. We take a snapshot at each minimum for test-time ensembling.

Learning Rate

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

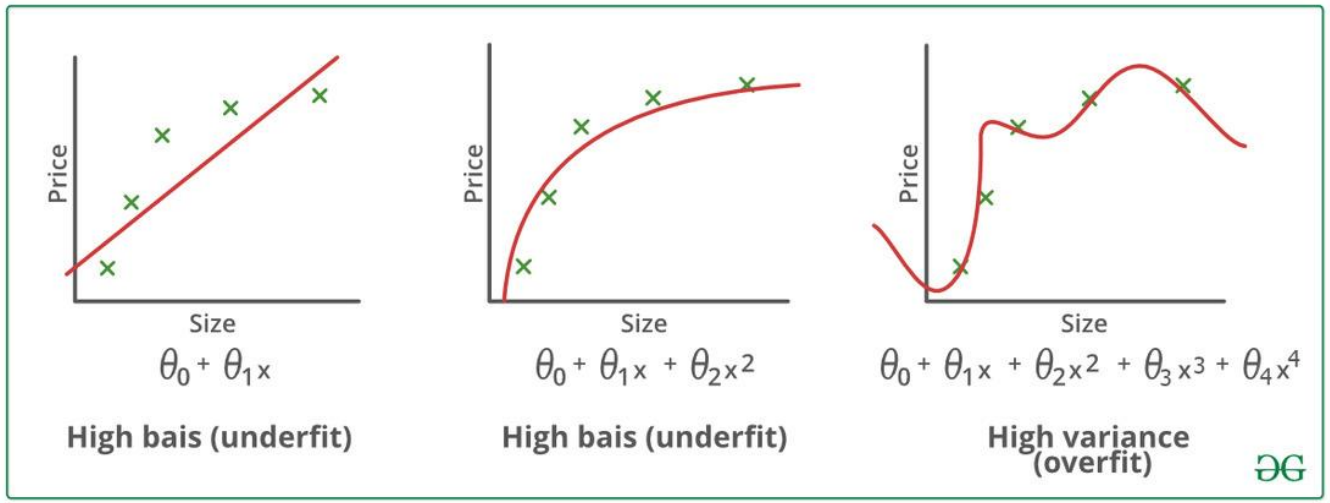


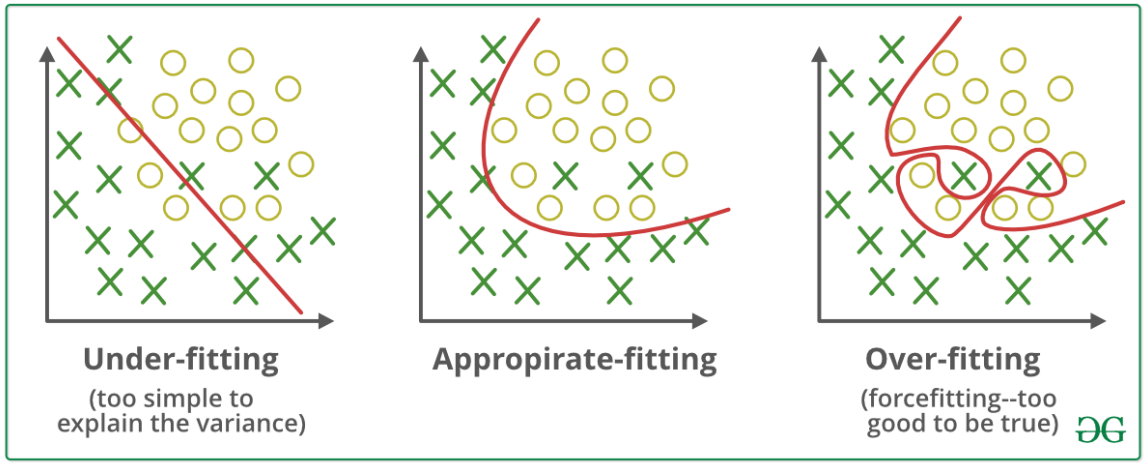
A cartoon depicting the effects of different learning rates. With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape.

Training Optimization

- Shuffle training samples before each epoch
  - Avoid overfitting problem
- Use a fixed batch size for each epoch
  - Accelerate the multiplication speed of matrix computation
- Define an appropriate learning rate for the batch size
  - A larger batch size requires a smaller learning rate
  - A smaller batch size requires a larger learning rate
- Enroll a validation dataset and testing dataset
  - Avoid overfitting problem
  - Validate the model performance

Overfitting Issue





Techniques to reduce overfitting:

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization and Lasso Regularization
5. Use dropout for neural networks to tackle overfitting.

References

- 臺大電機系李宏毅教授講義
- 臺大資工系陳縕儂教授講義
- NVIDIA Deep Learning Tutorial
- https://prinsphield.github.io/posts/2016/02/overview_opt_alg_deep_learning1/
- https://blog.51cto.com/u_13393656/3312623
- https://www.deeplearningbook.org/contents/optimization.html
- https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Thank you for your attention!