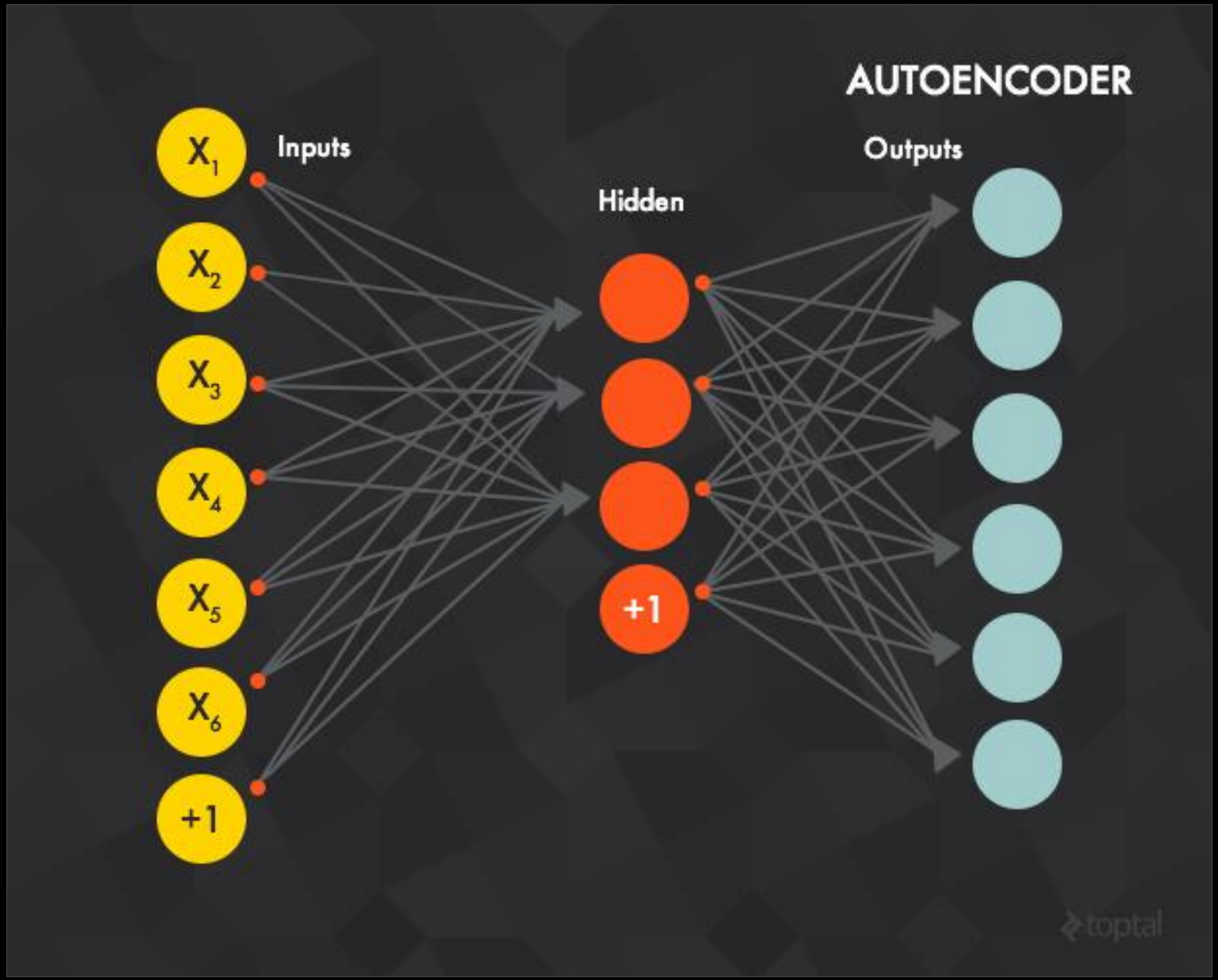
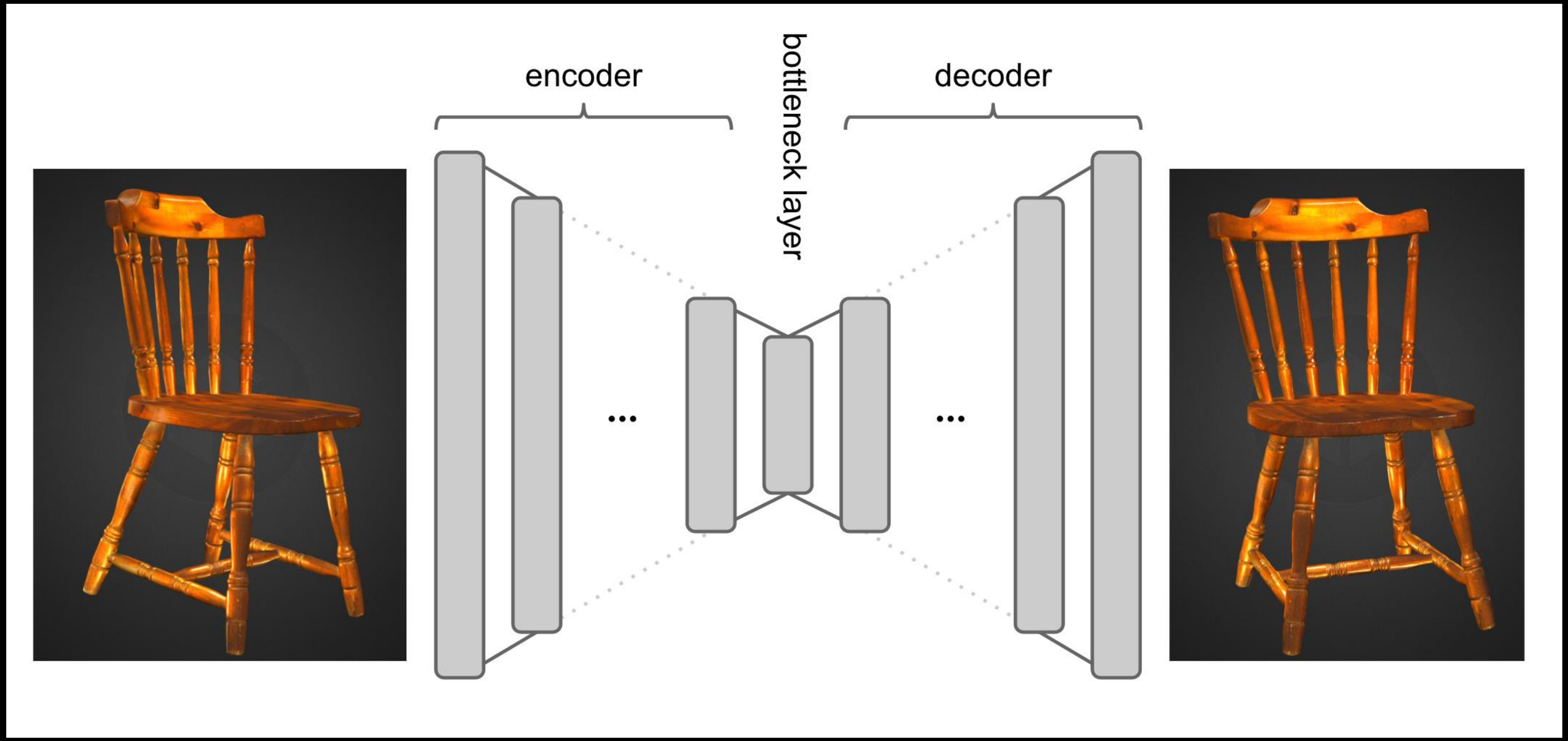
朝向最佳解邁進!

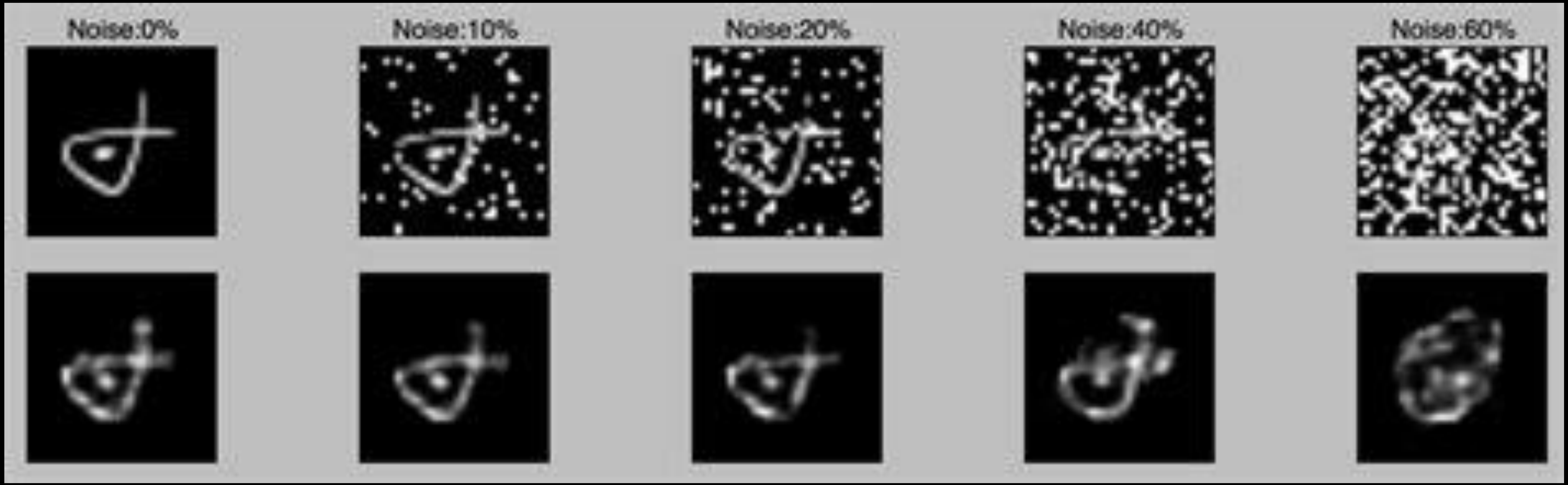
自動編碼器
Auto-encoder

透過壓縮來尋找特徵

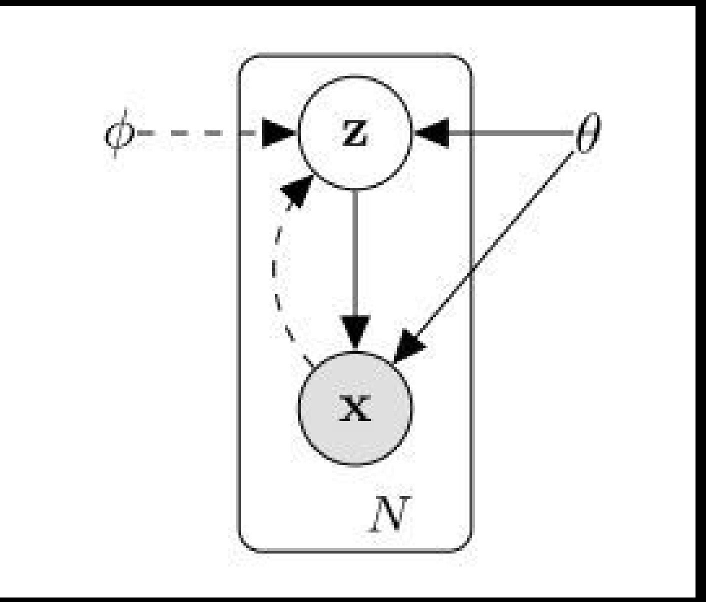


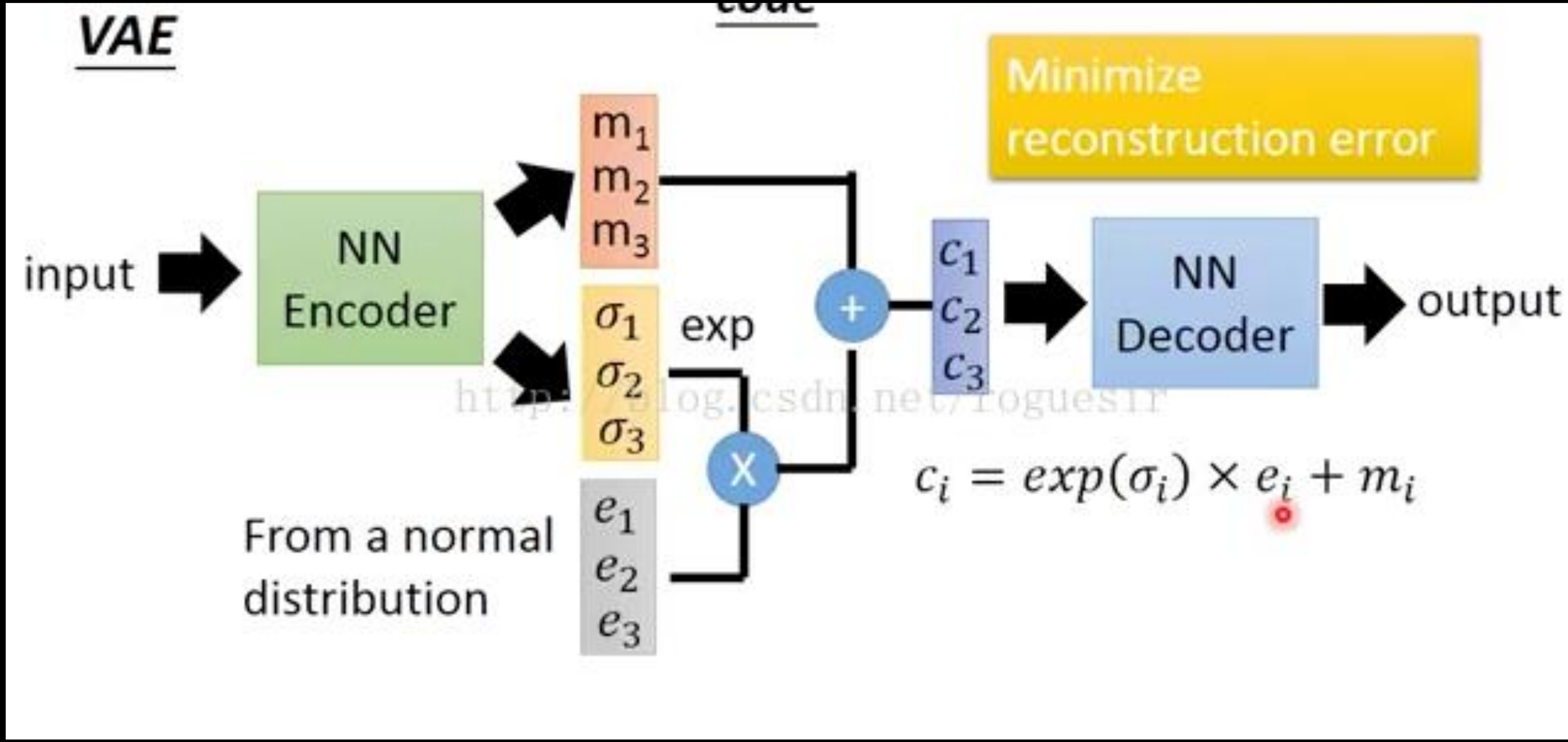


Denoising autoencoder



變分自編碼器（VAE）

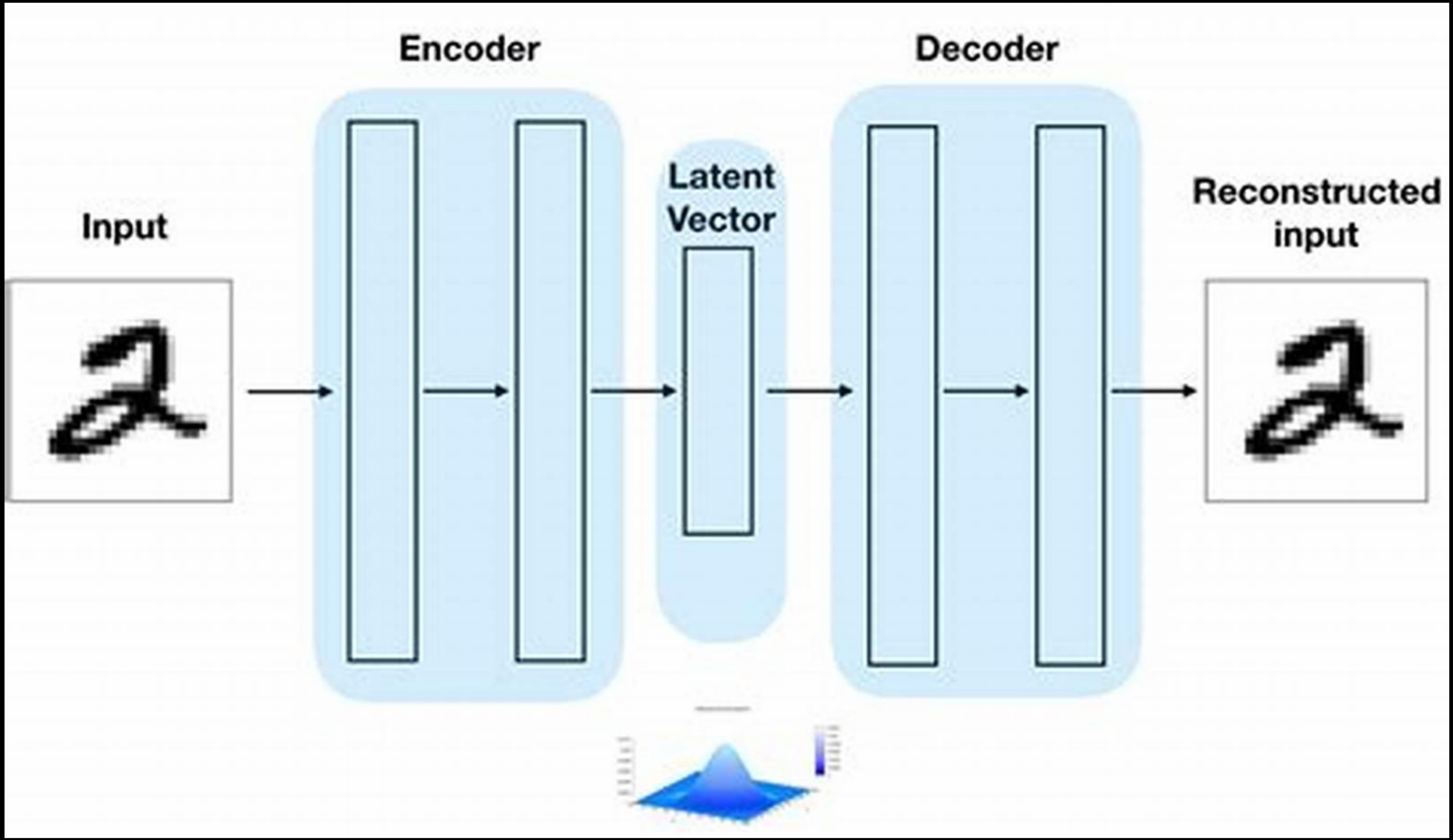




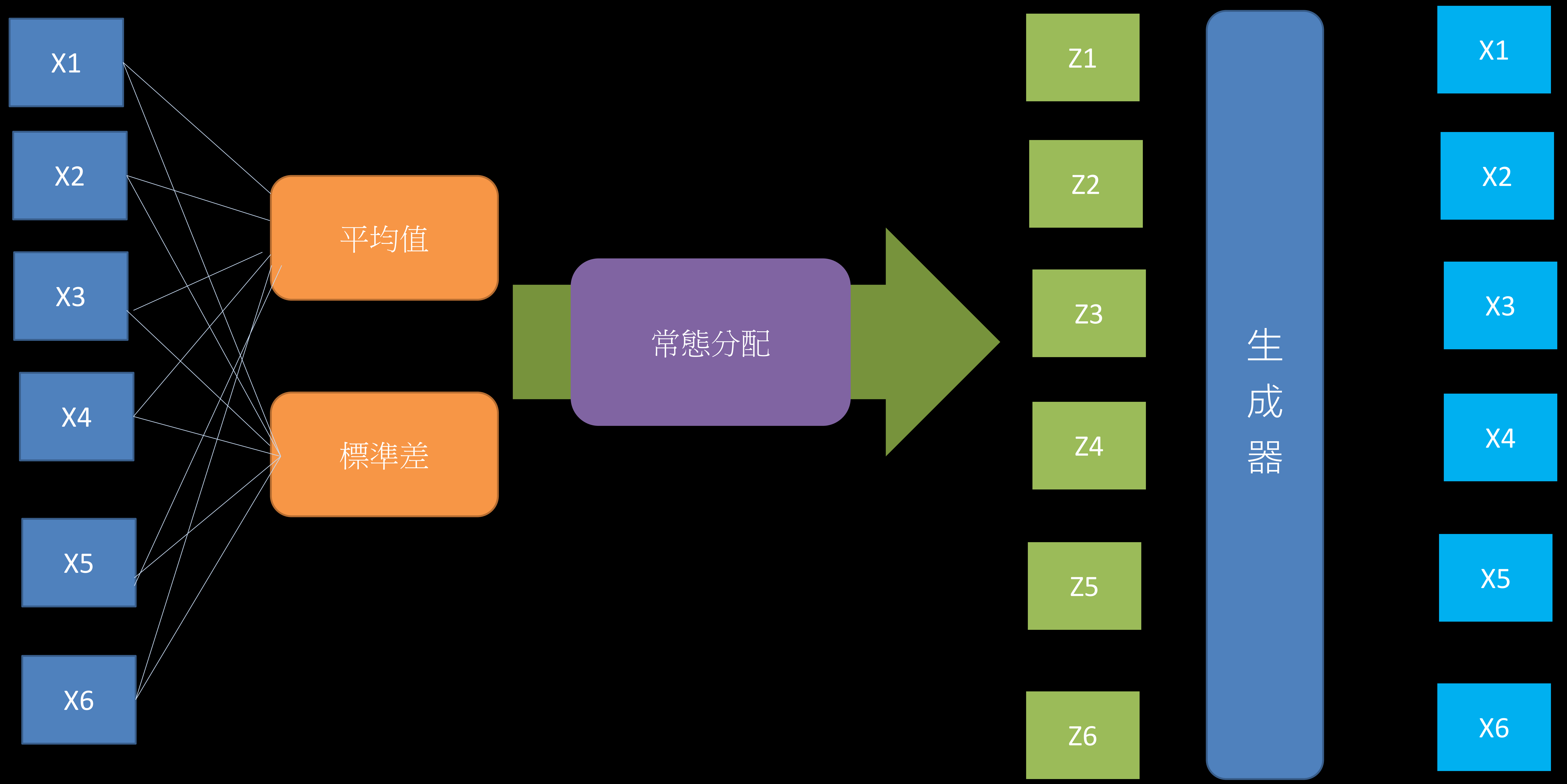
觀測到的資料是X，而由隱變數Z產生，由Z->X是生成模型，從自編碼器（auto-encoder）的角度來看，就是解碼器；而X->Z由是識別模型（recognition model），類似於自編碼器的編碼器。

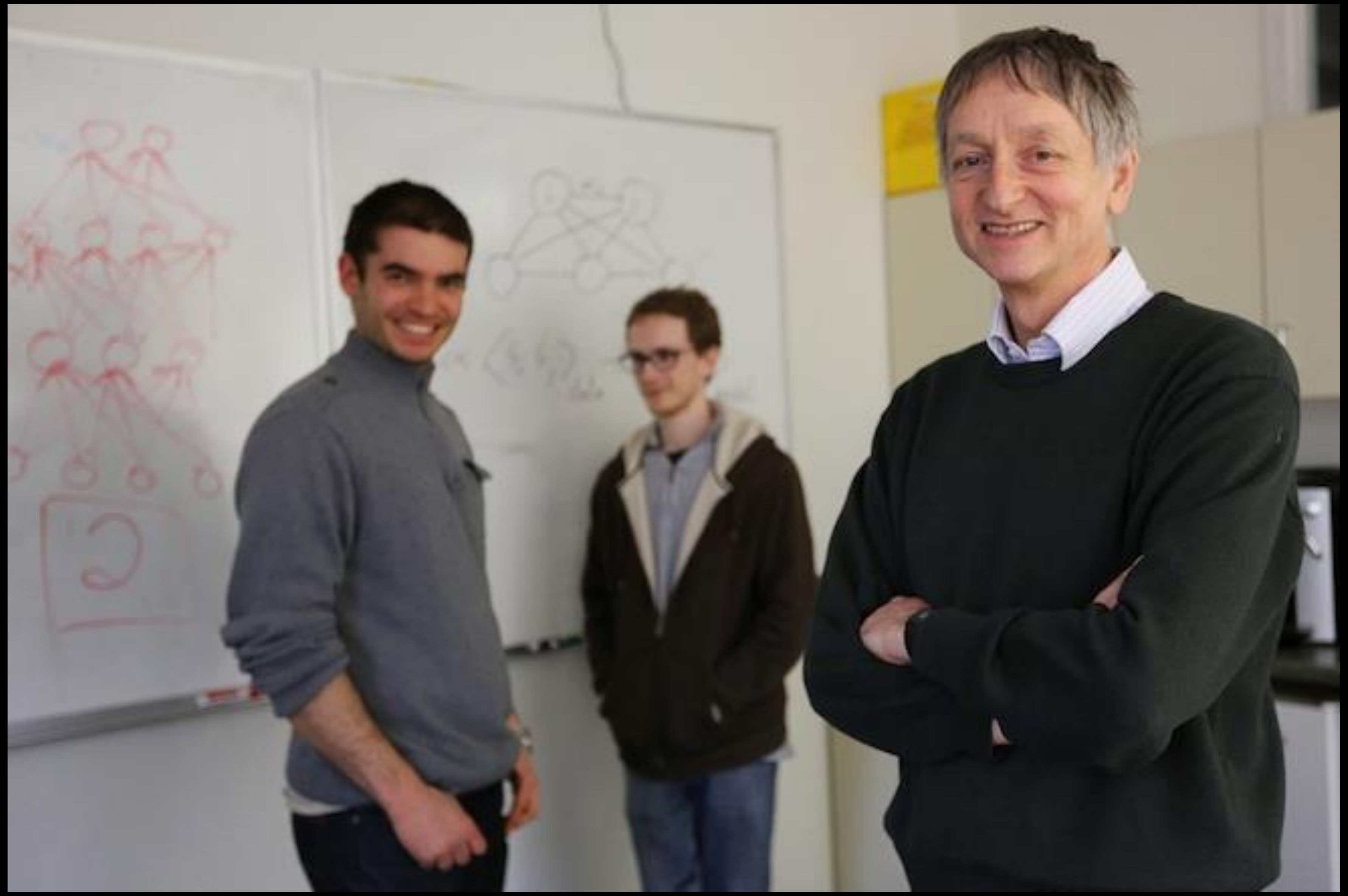
學習圖像的密度函數（PDF）

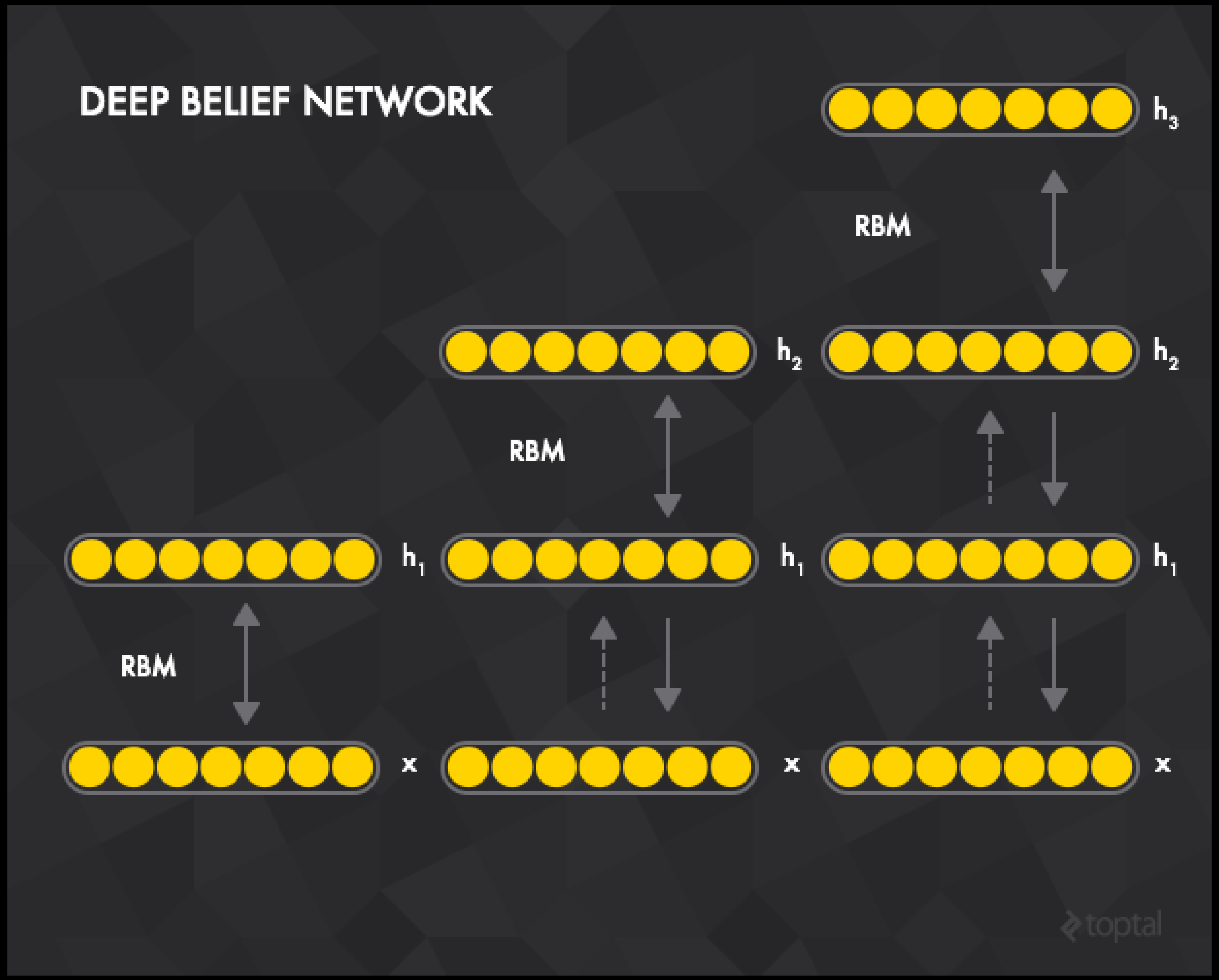
兩個分佈的相似程度，一般採用KL散度


Encoder
Decoder
Latent
Vector
Input
Reconstructed
input

Input image	2-D latent space	5-D latent space	10-D latent space	20-D latent space
				


X1
X2
X3
X4
X5
X6
平均值
標準差
常態分配
Z1
Z2
Z3
Z4
Z5
Z6
生成器
X1
X2
X3
X4
X5
X6

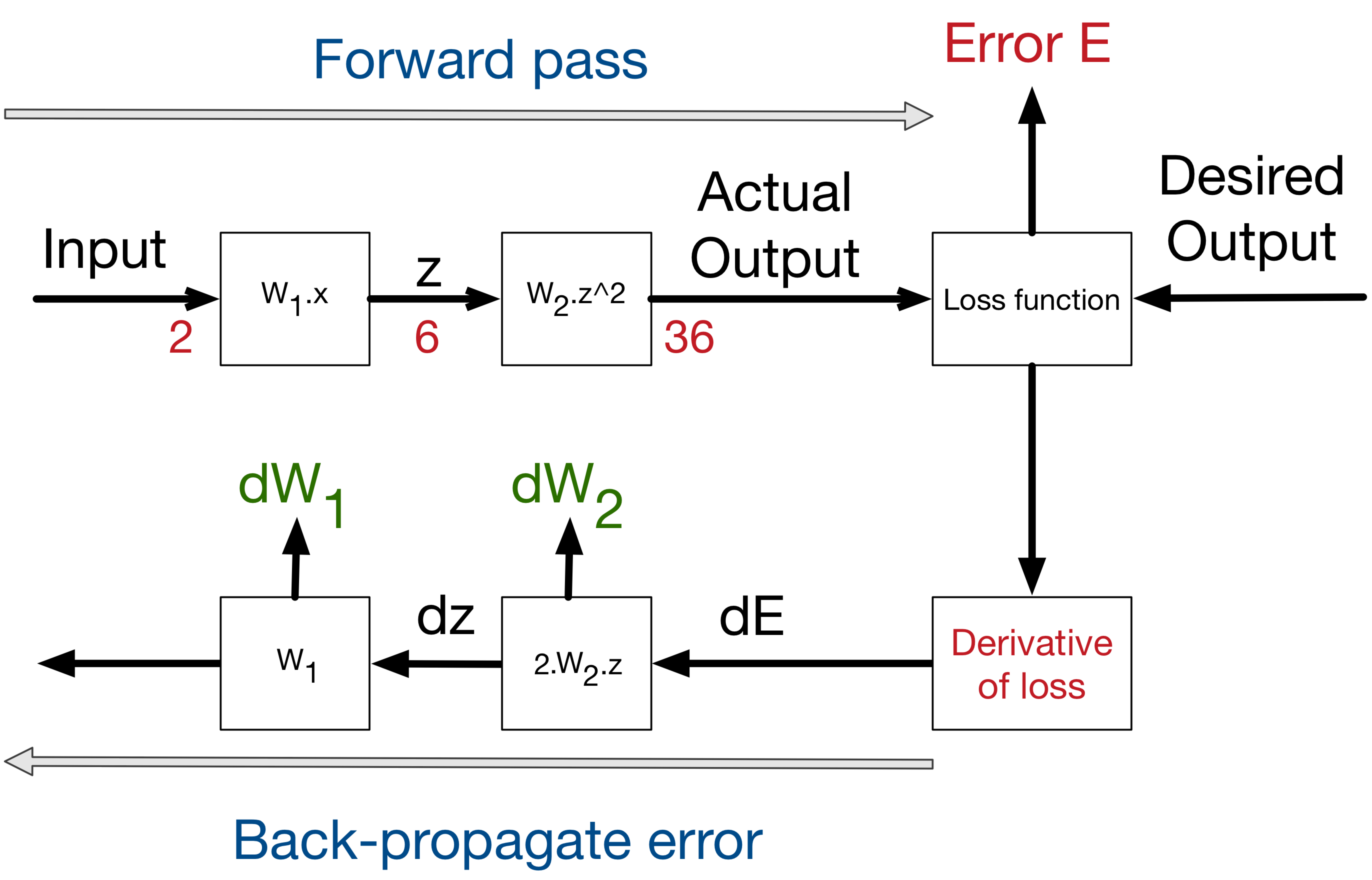




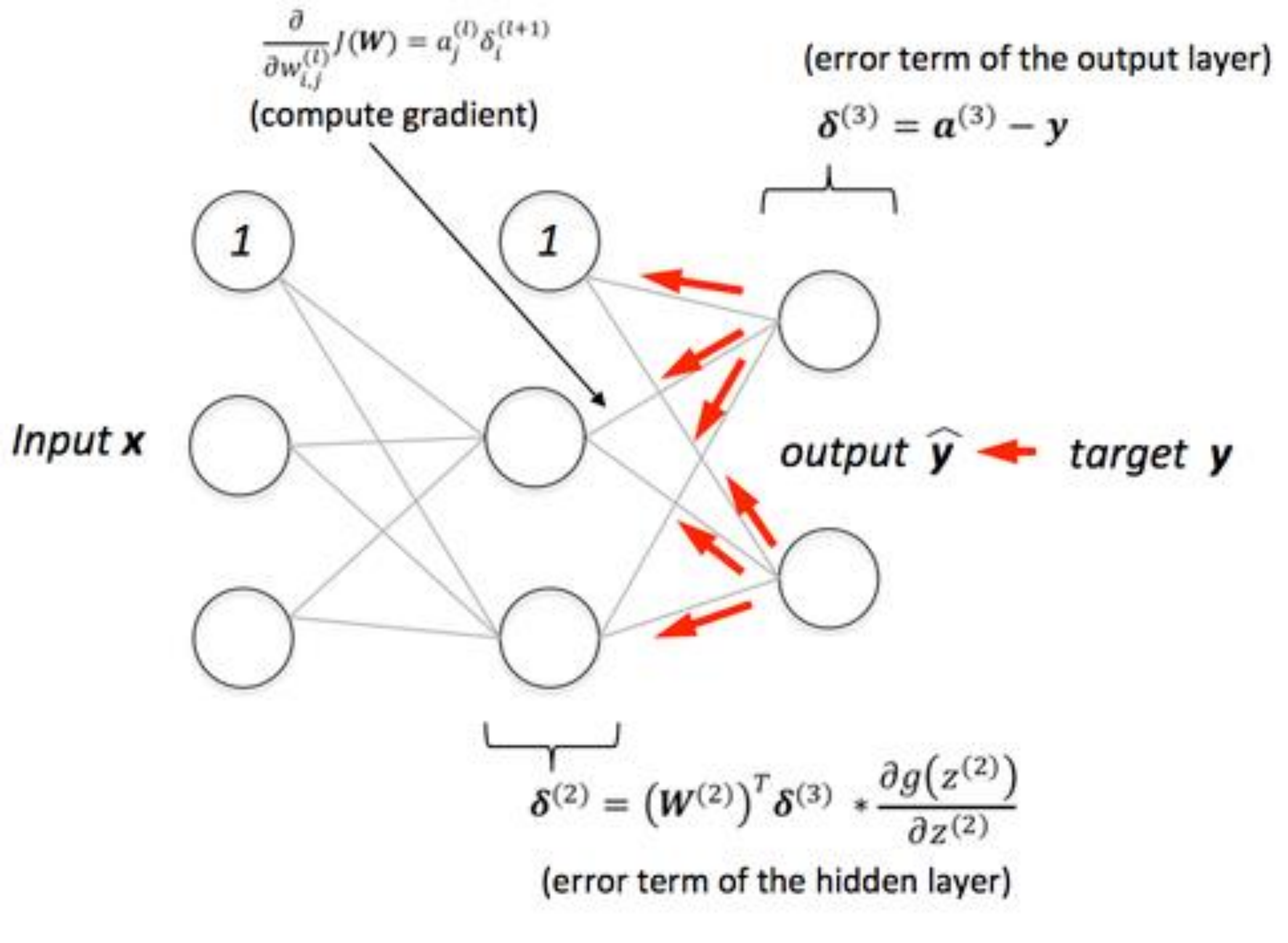
A fast learning algorithm for deep belief nets: 2006

To RecogniZe Shapes, First Learn to Generate Images: 2006

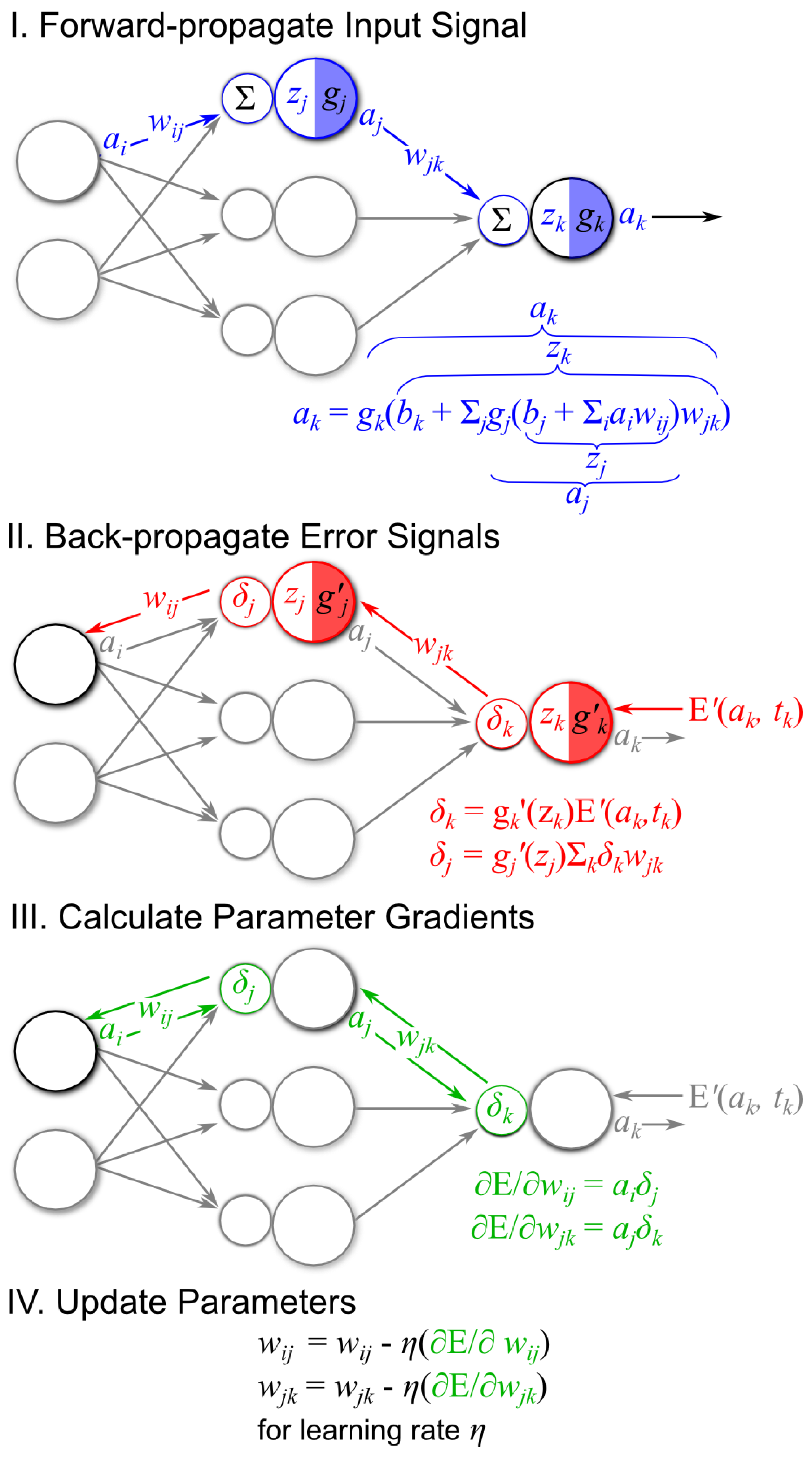
Reducing the dimensionality of data with neural networks : 2006



Forward pass
Error E
Input
2
$W_1.x$
z
6
$W_2.z$^2
Actual Output
36
Loss function
Desired Output
dW_1
dW_2
dz
dE
W_1
$2.W_2.z$
Derivative of loss
Back-propagate error



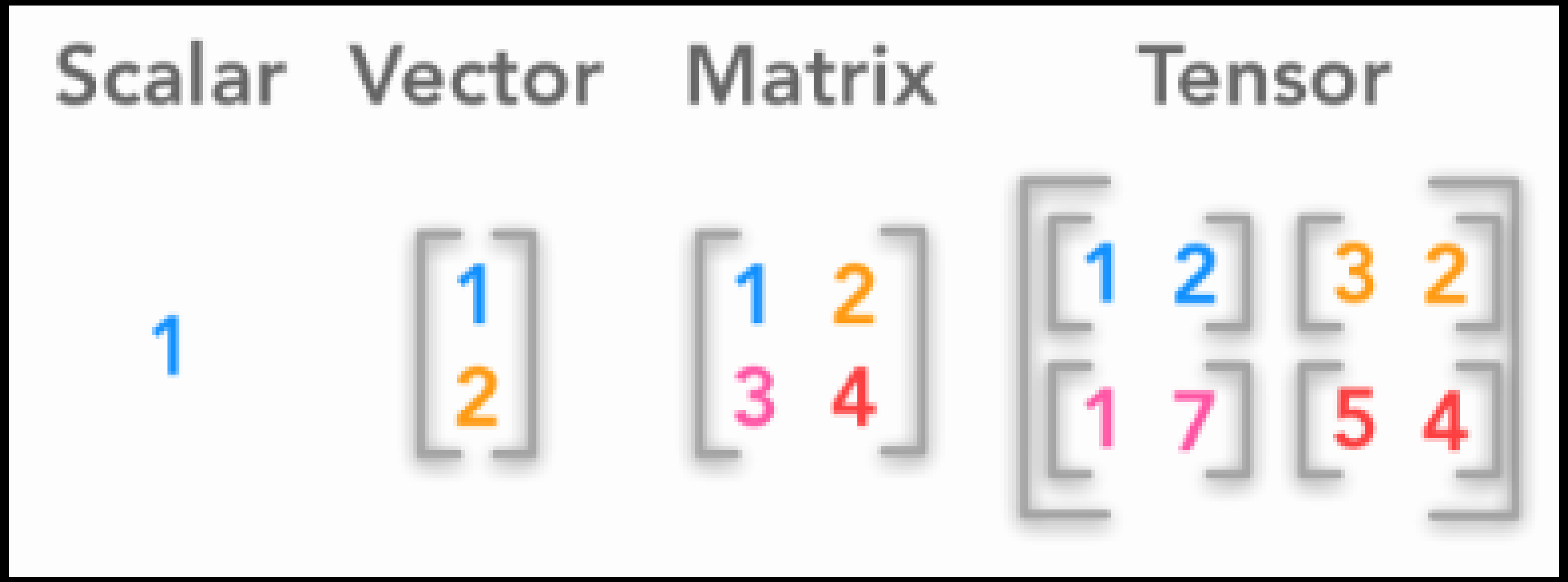
$\frac{\partial}{\partial w_{i,j}^{(l)}} J(\boldsymbol{W}) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\boldsymbol{\delta}^{(3)} = \boldsymbol{a}^{(3)} - \boldsymbol{y}$
1
1
Input **x**
output $\widehat{\mathbf{y}}$
target **y**
$\boldsymbol{\delta}^{(2)} = \left(\boldsymbol{W}^{(2)}\right)^T \boldsymbol{\delta}^{(3)} * \frac{\partial g\left(z^{(2)}\right)}{\partial z^{(2)}}$
(error term of the hidden layer)

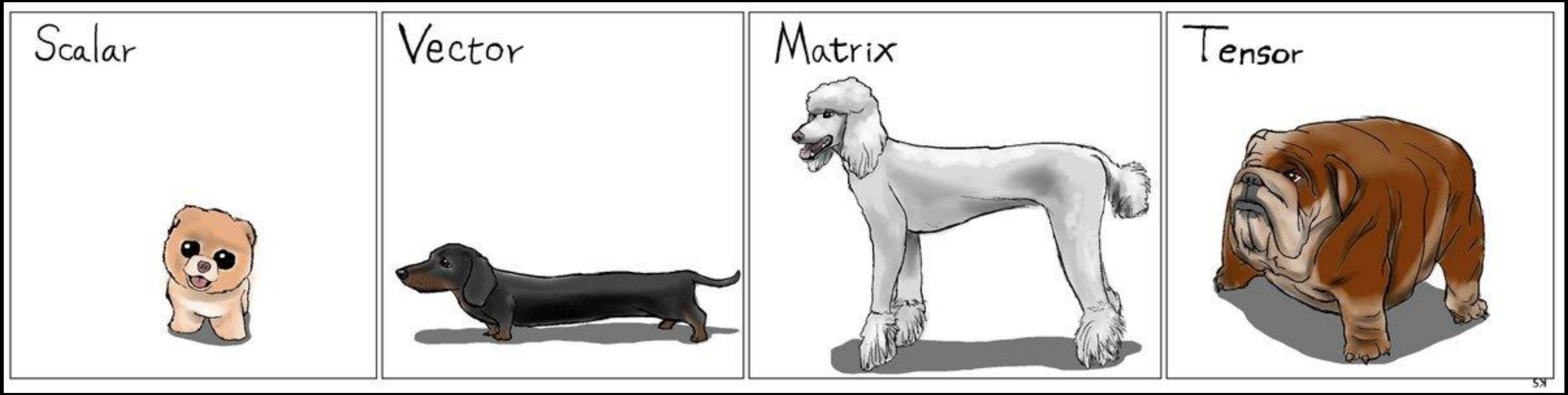


順向傳導

逆向傳導

計算圖


Scalar
Vector
Matrix
Tensor
1
1 2
1 2 3 4
1 2 3 2 1 7 5 4


Scalar
Vector
Matrix
Tensor

給定待最佳化的模型參數 $\theta \in \mathbb{R}^d$

損失函數 $J(\theta)$

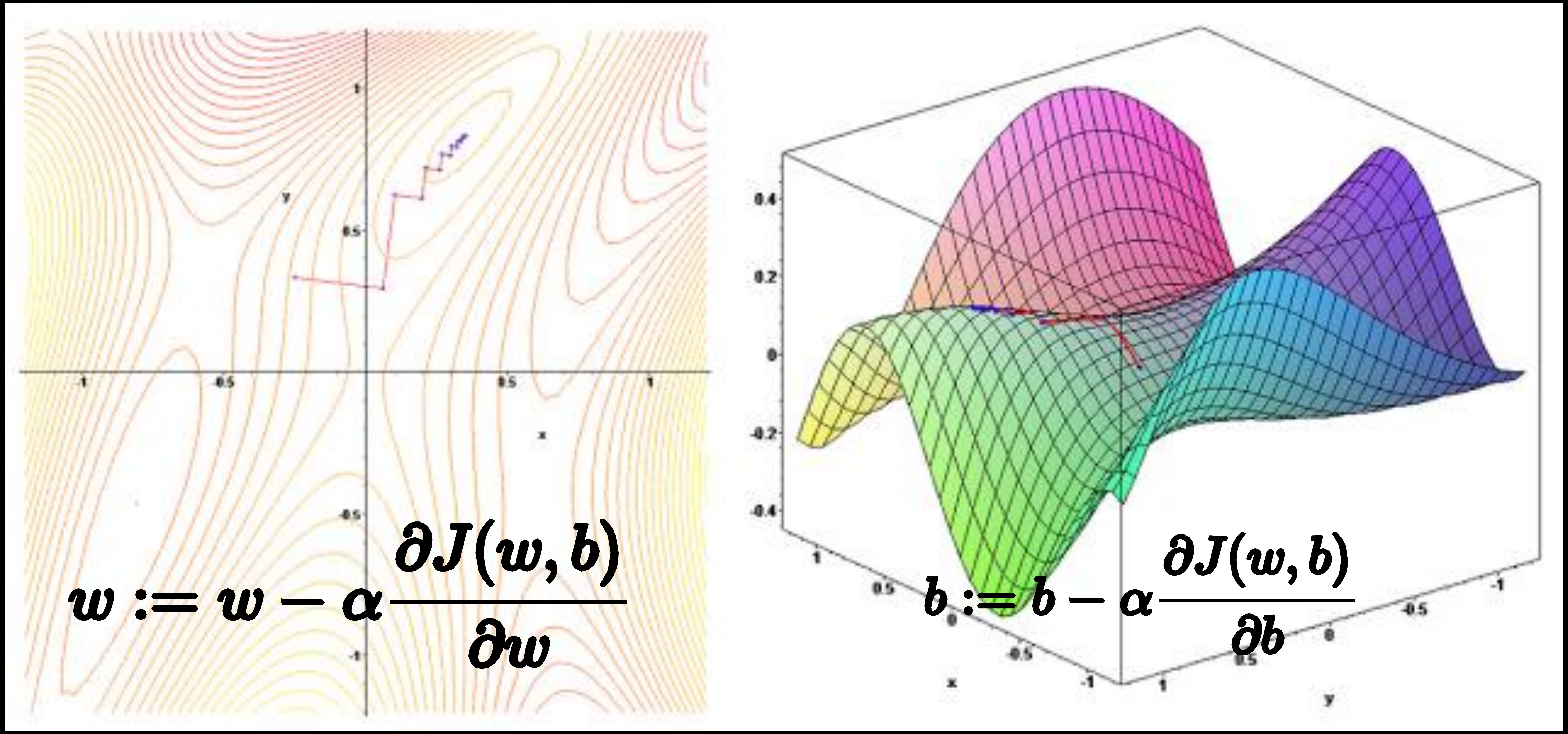
沿著 $\nabla_\theta J(\theta)$ 梯度向下的方向來更新 θ

學習速率LR決定了每一時刻的更新步長

如何找到神經網路的最優化結果

那就沿著能讓誤差下降最快的陡坡那裏走就對了

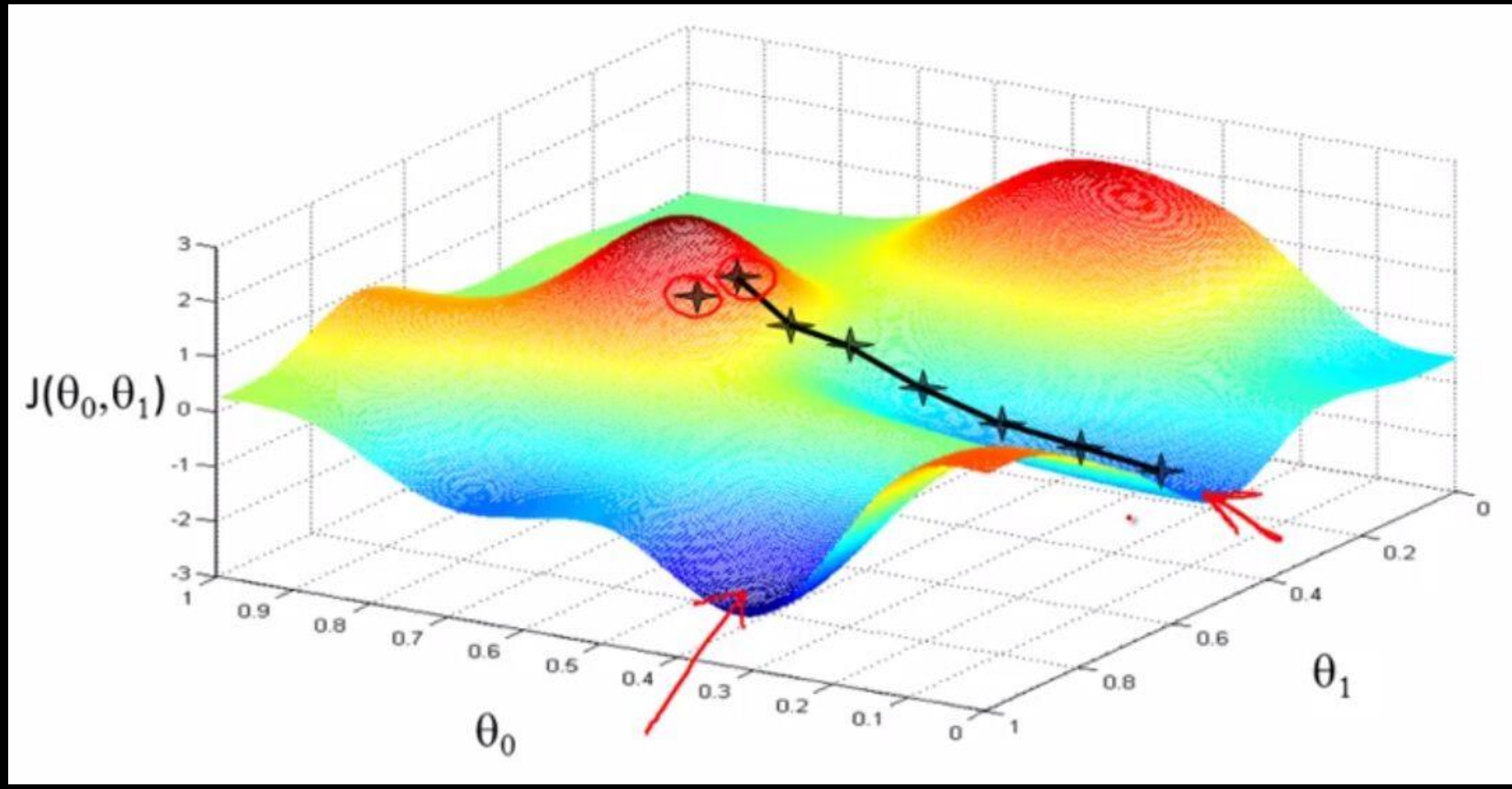
- 需要是凸函數
- 必需連續可微分

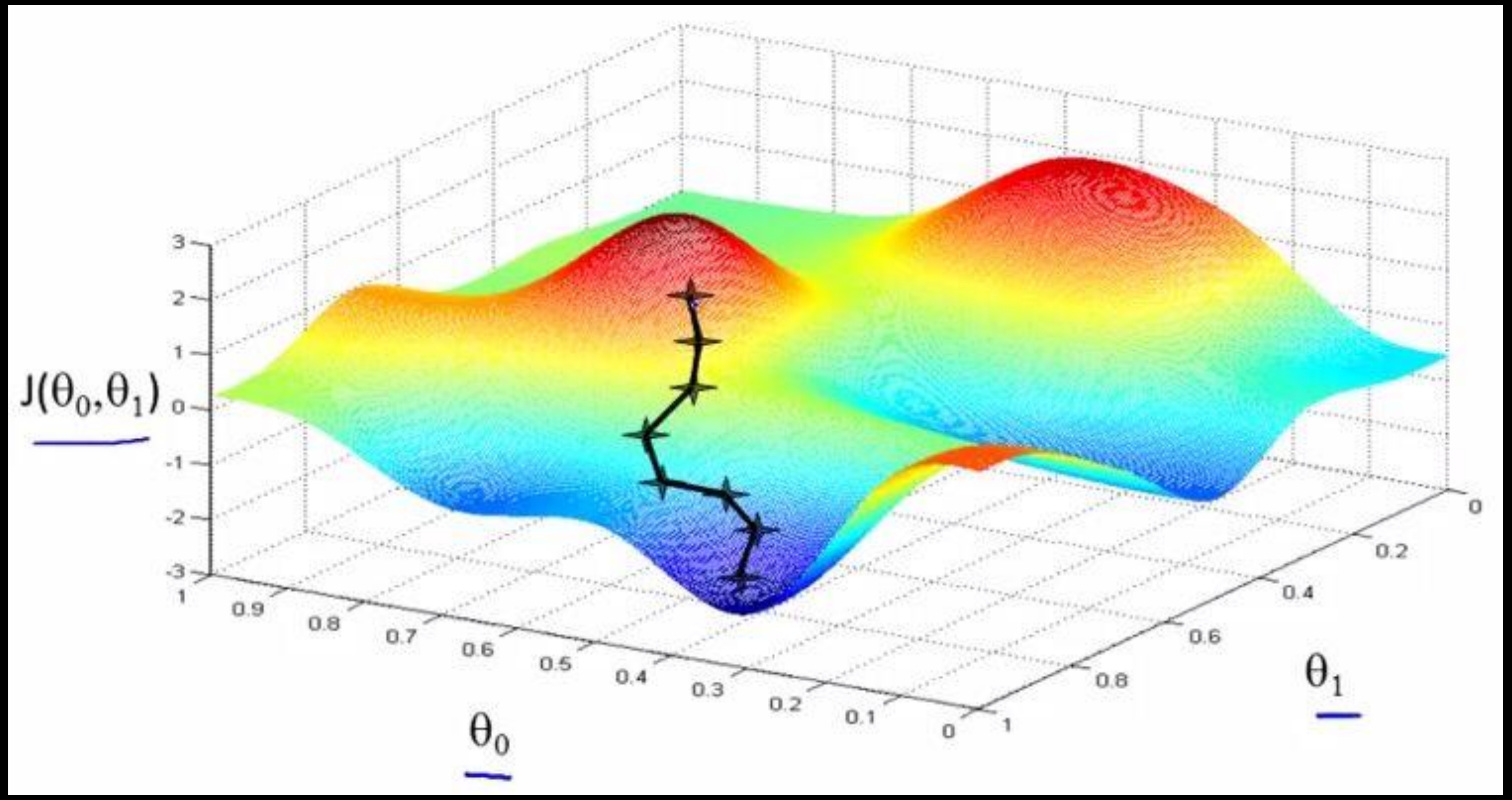


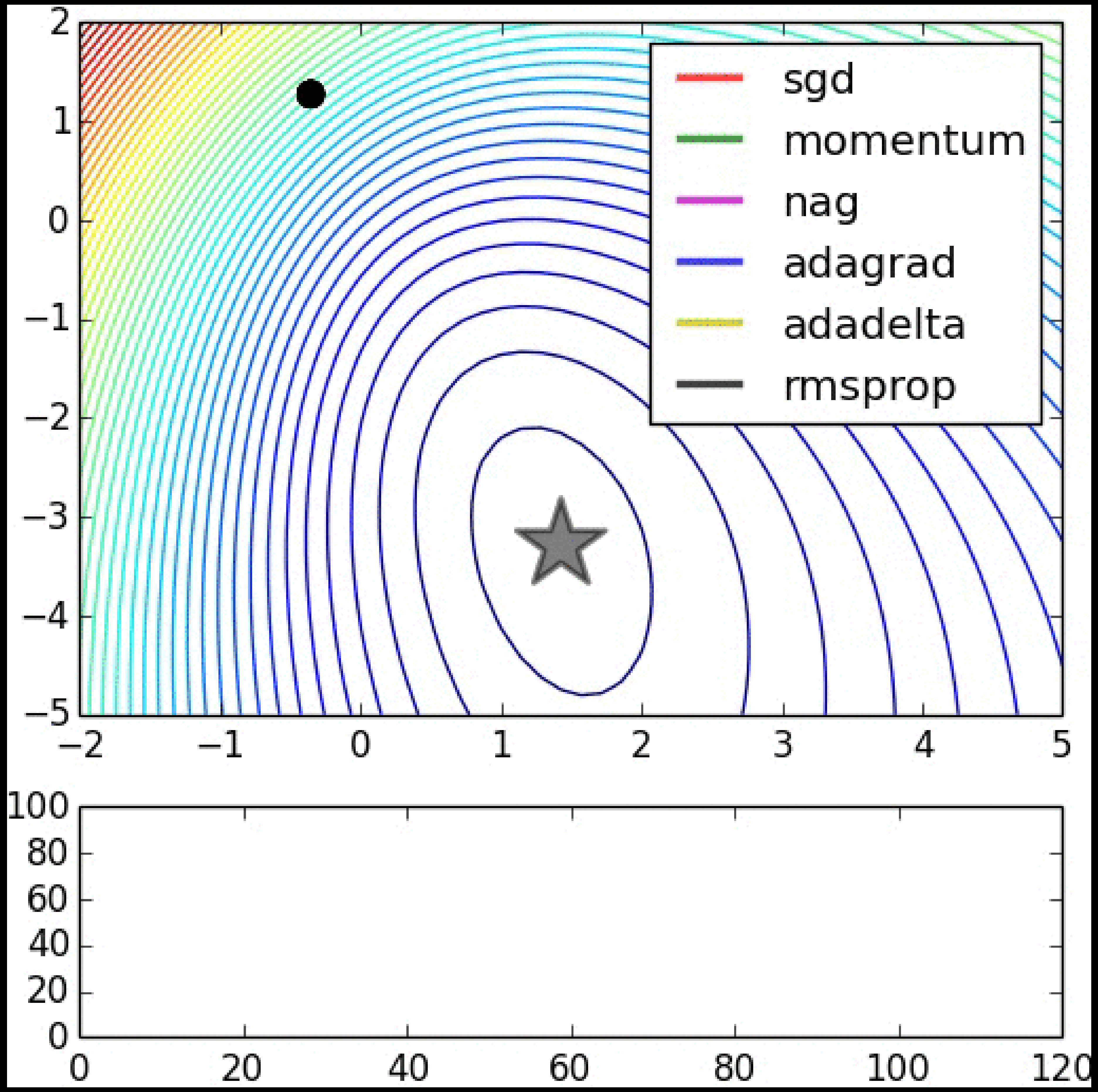
$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

隨機梯度下降/Stochastic Gradient Descent (SGD)







Adam 自我調整動量估計（Adaptive Moment Estimation）

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

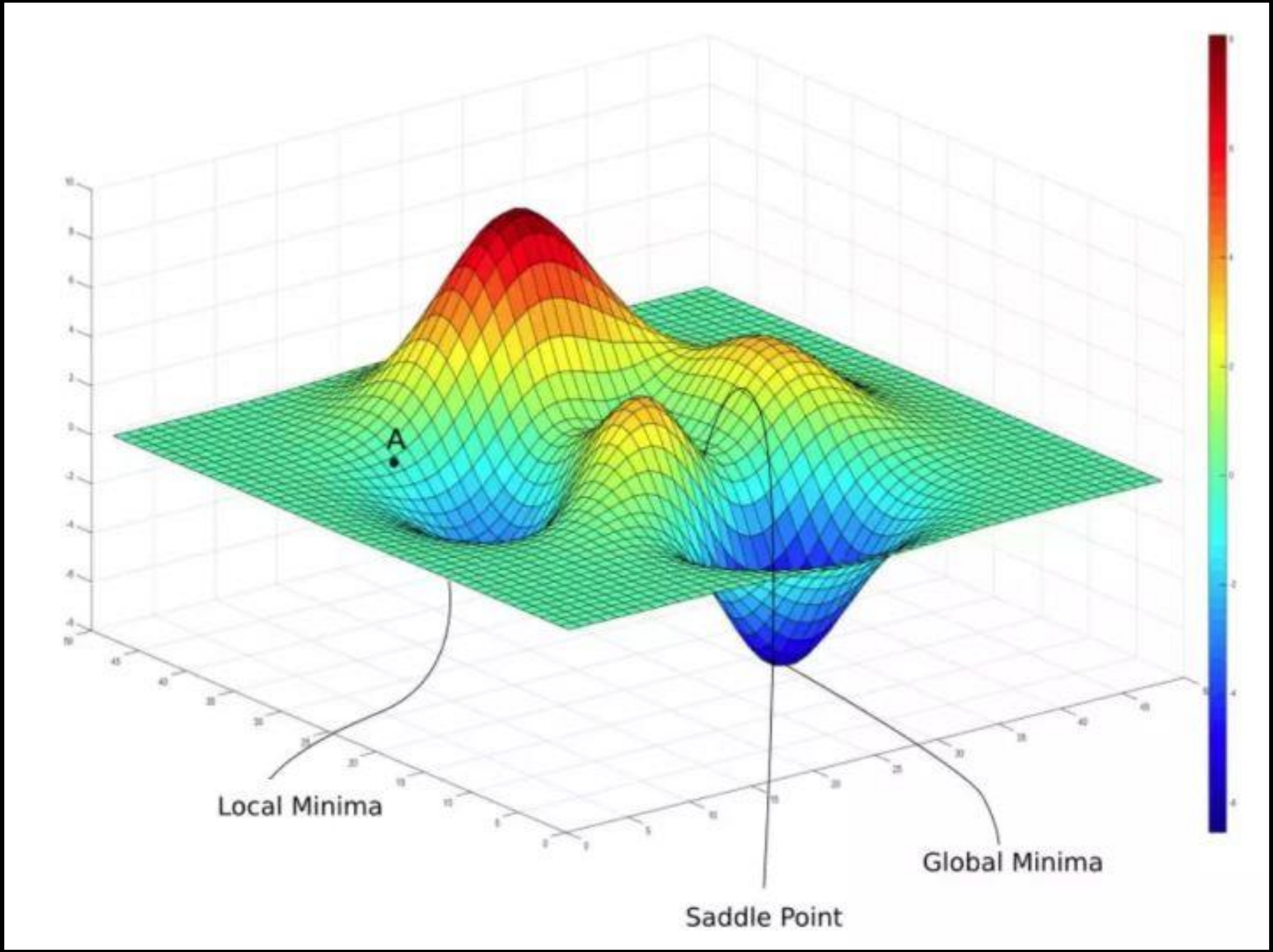
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

動量

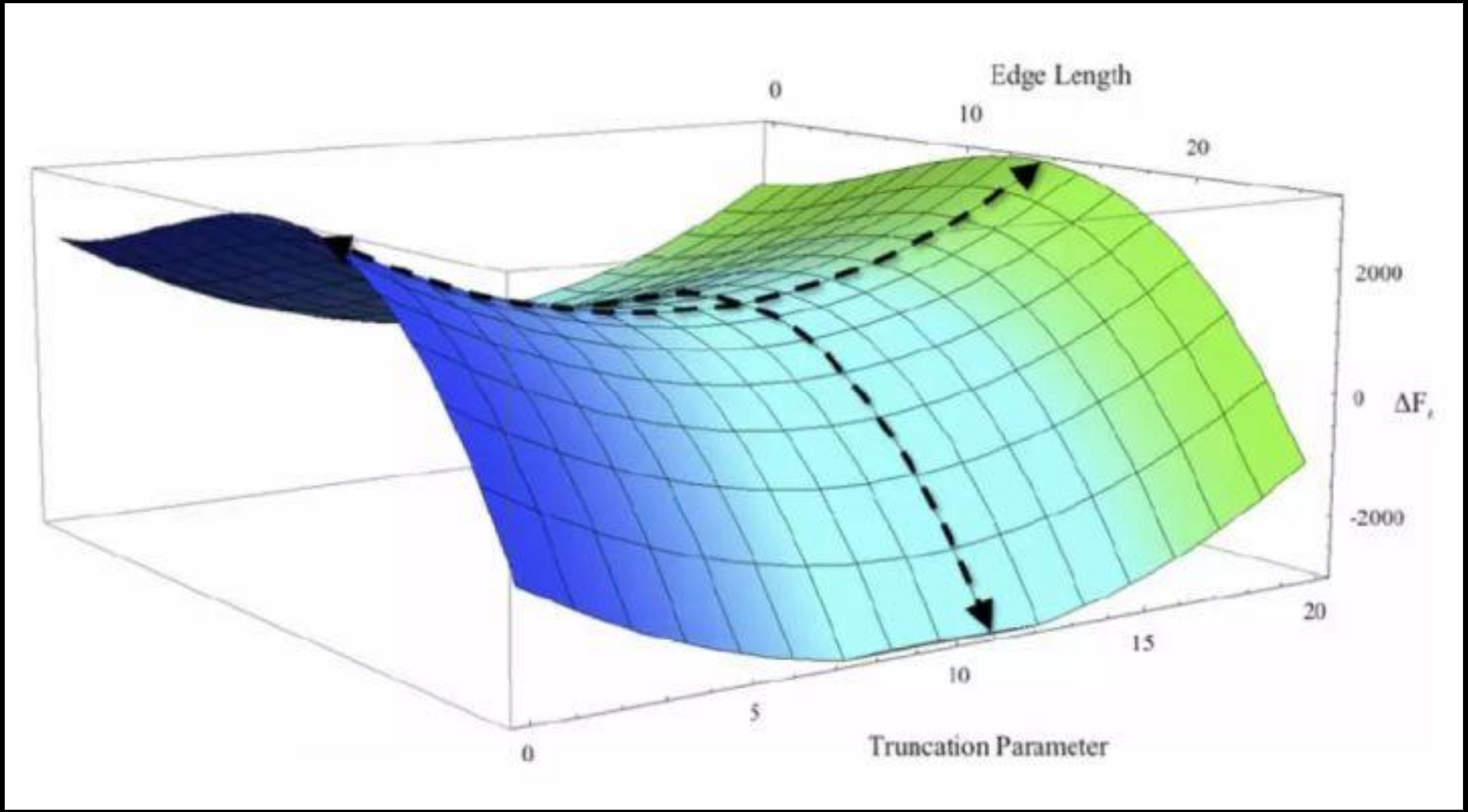
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

β1取0.9，**β2**取0.999，**ϵ**取10-8。

區域最佳解

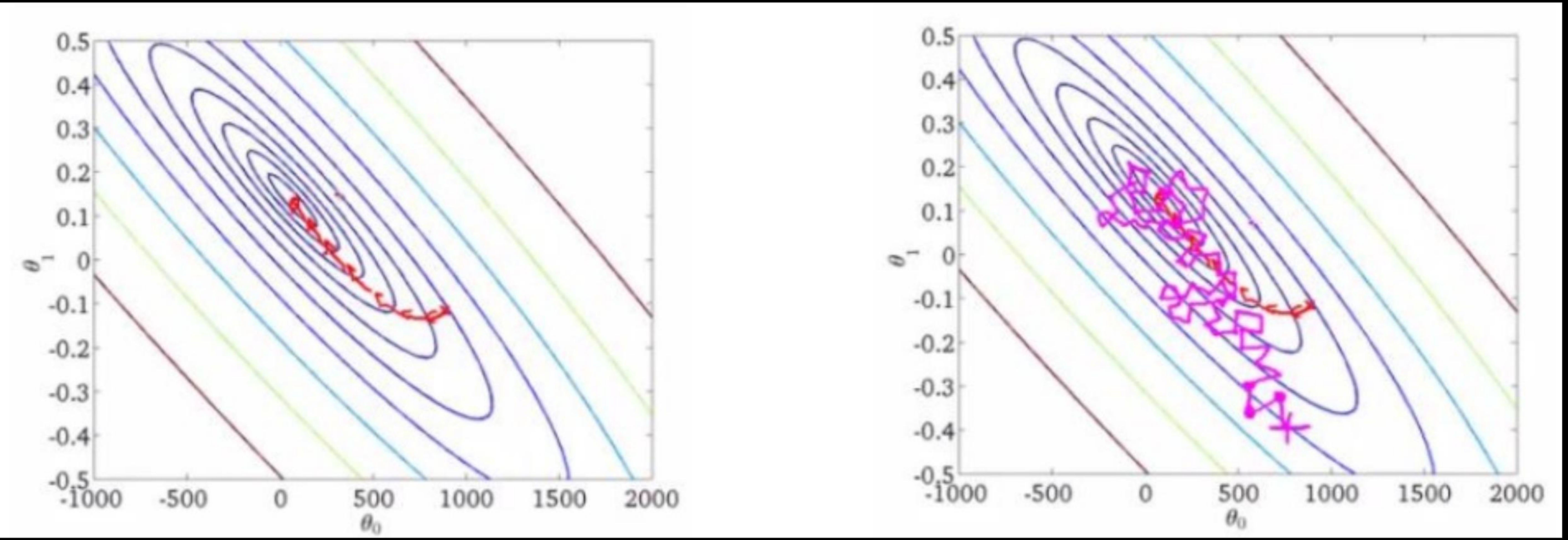


鞍點



Edge Length
0
10
20
2000
0
ΔF
-2000
20
15
10
5
0
Truncation Parameter

Minibatch



批次梯度下降

緩慢但耗費記憶體

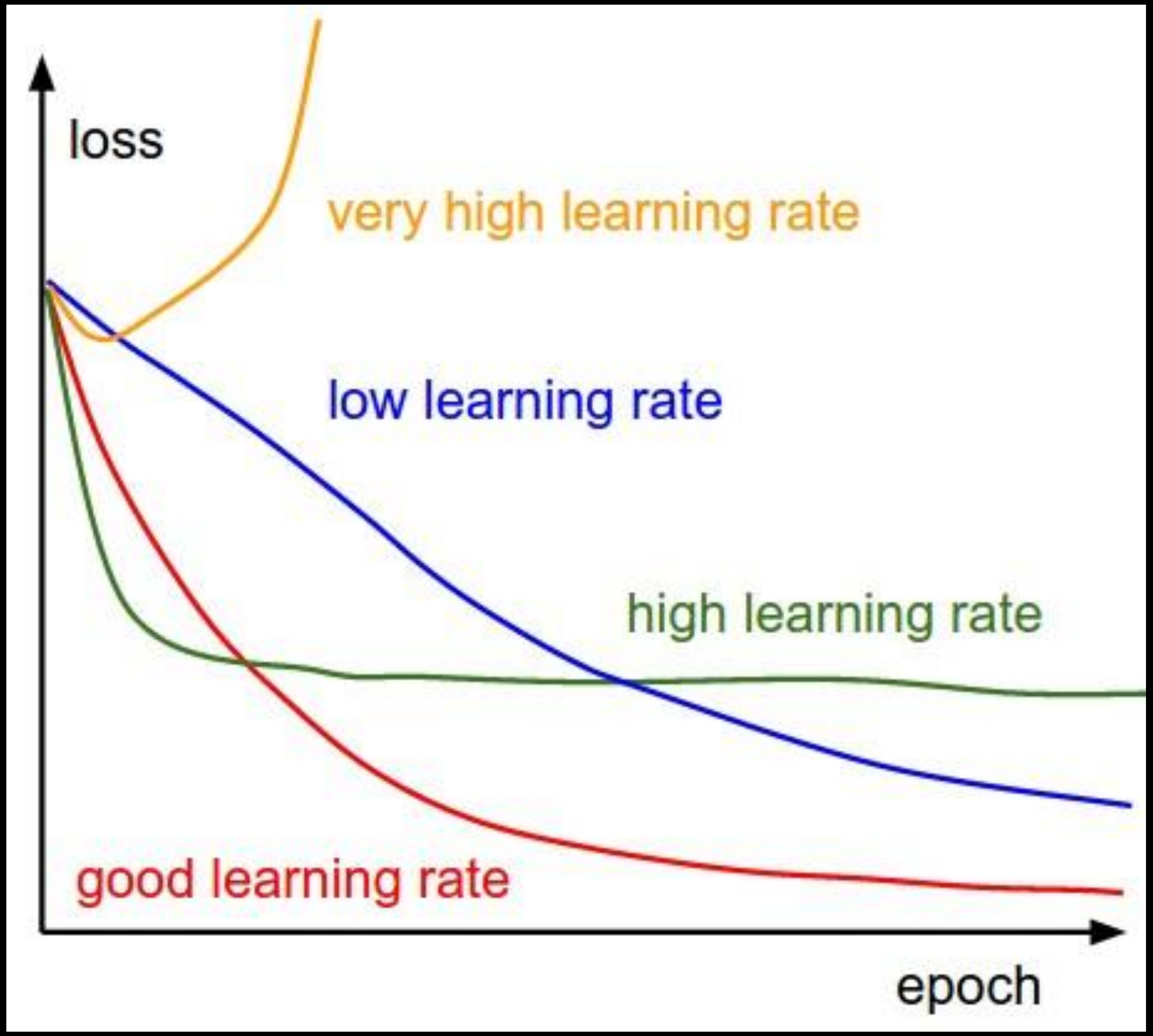
隨機梯度下降

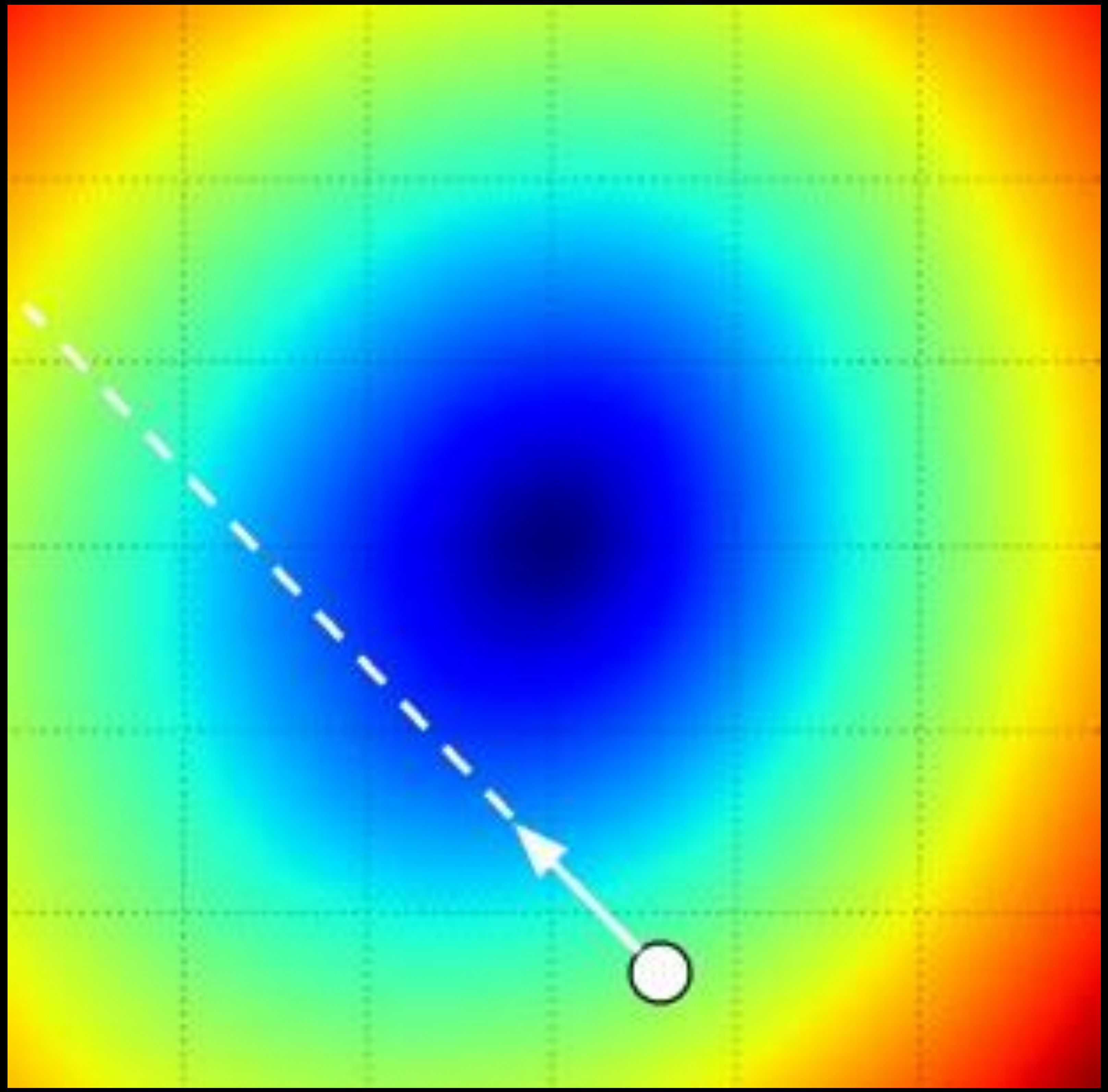
會有波動但快速

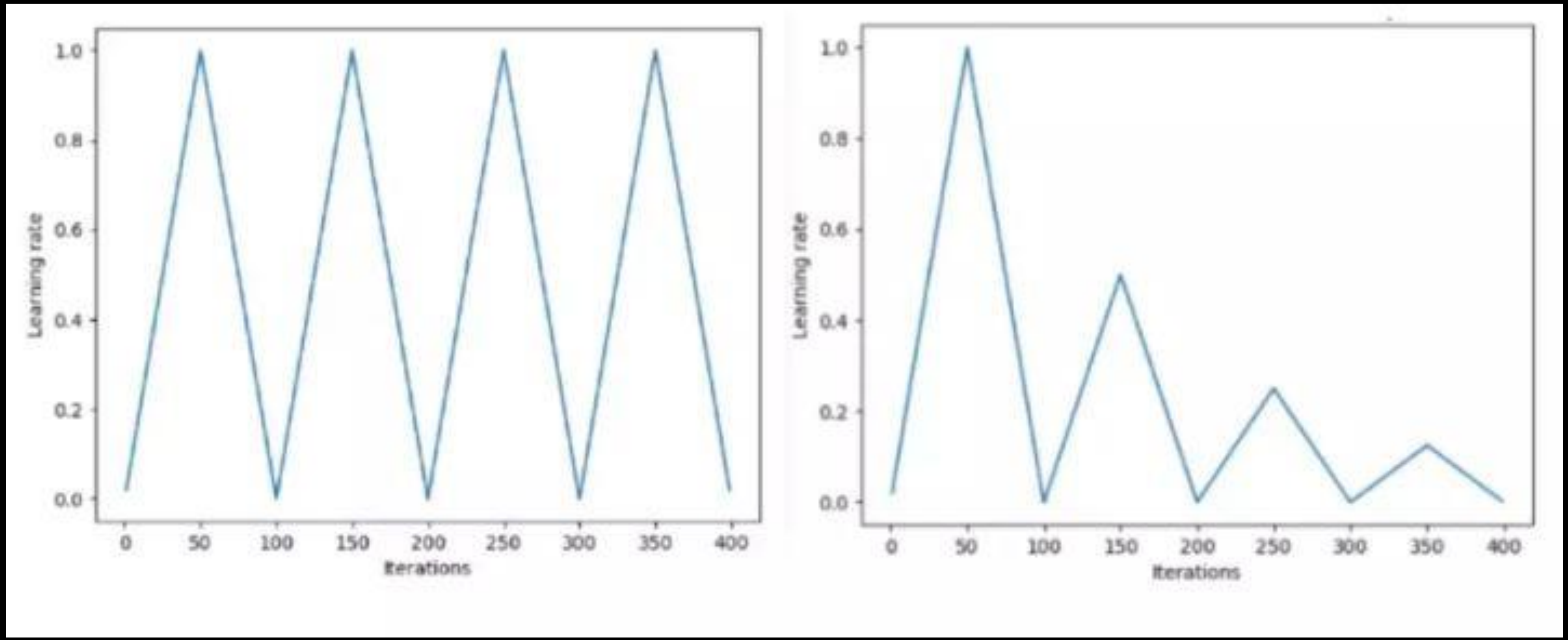
Minibatch更新梯度下降

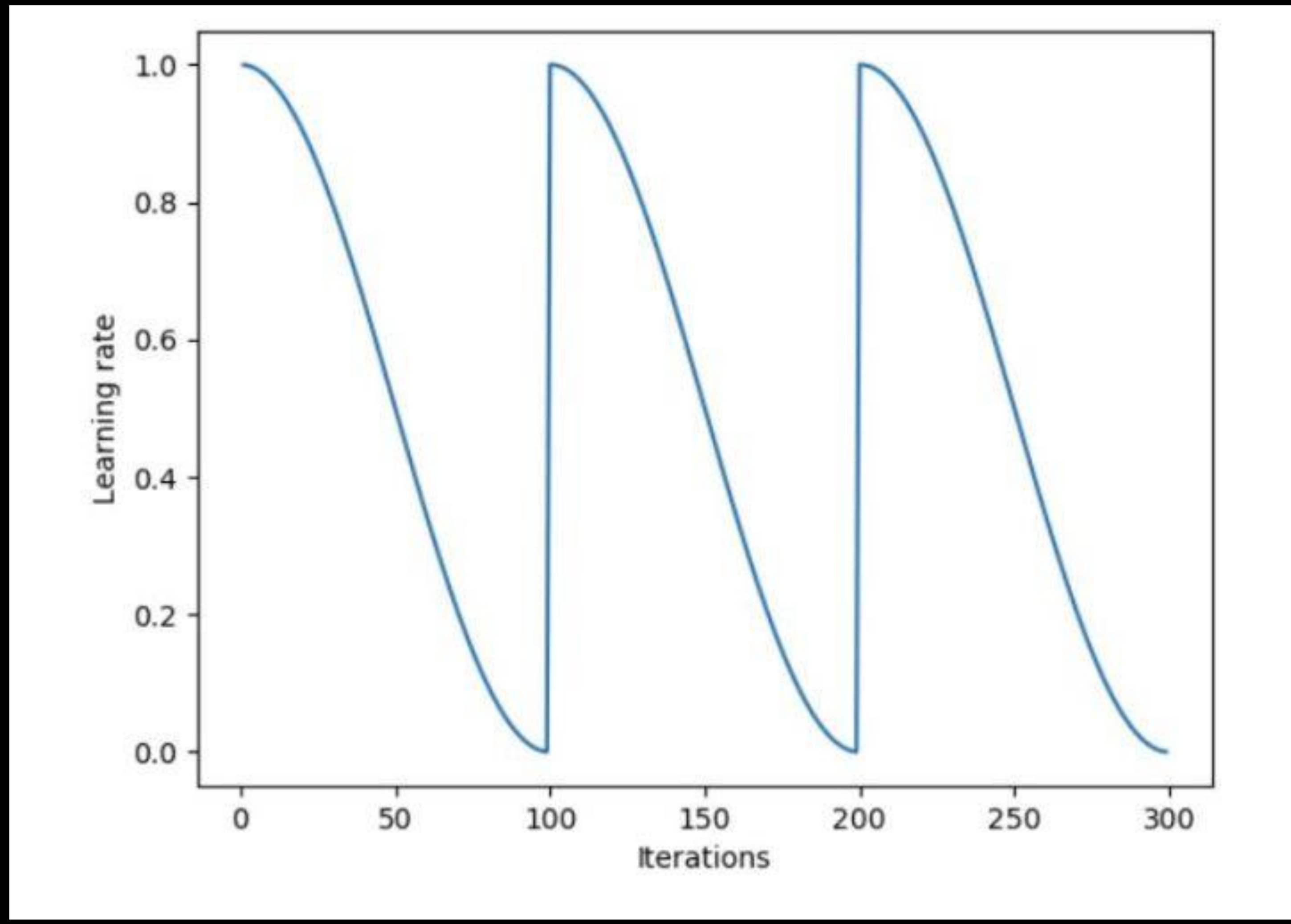
Epoch

學習速率
Learning Rate



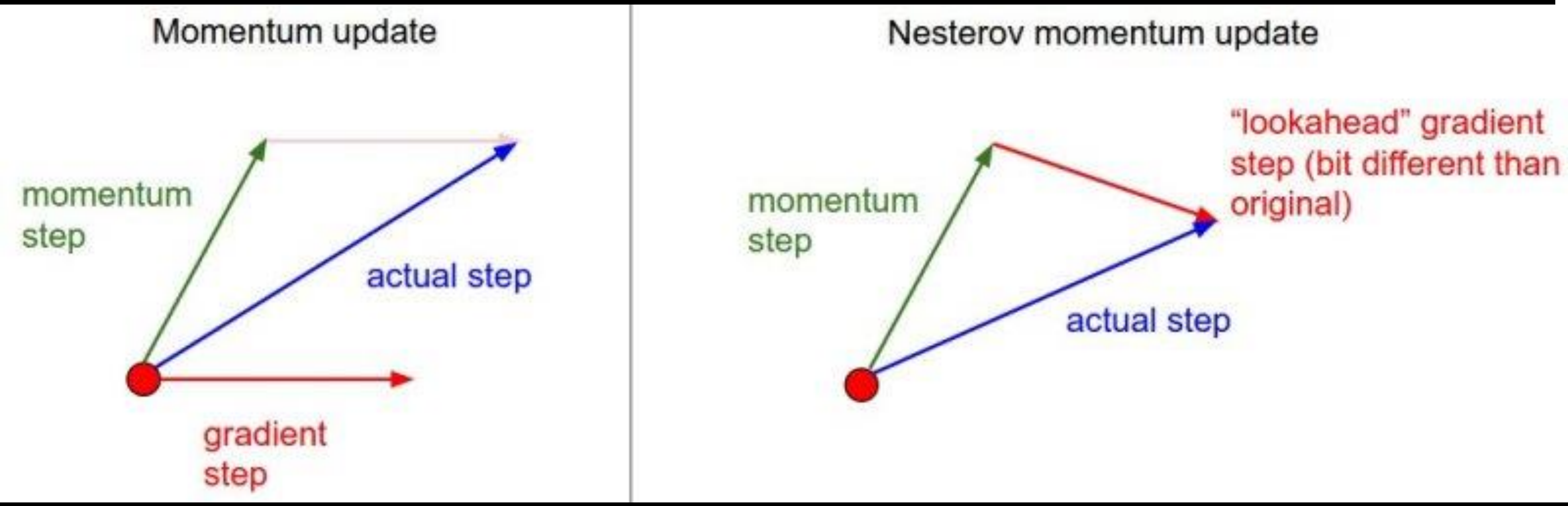


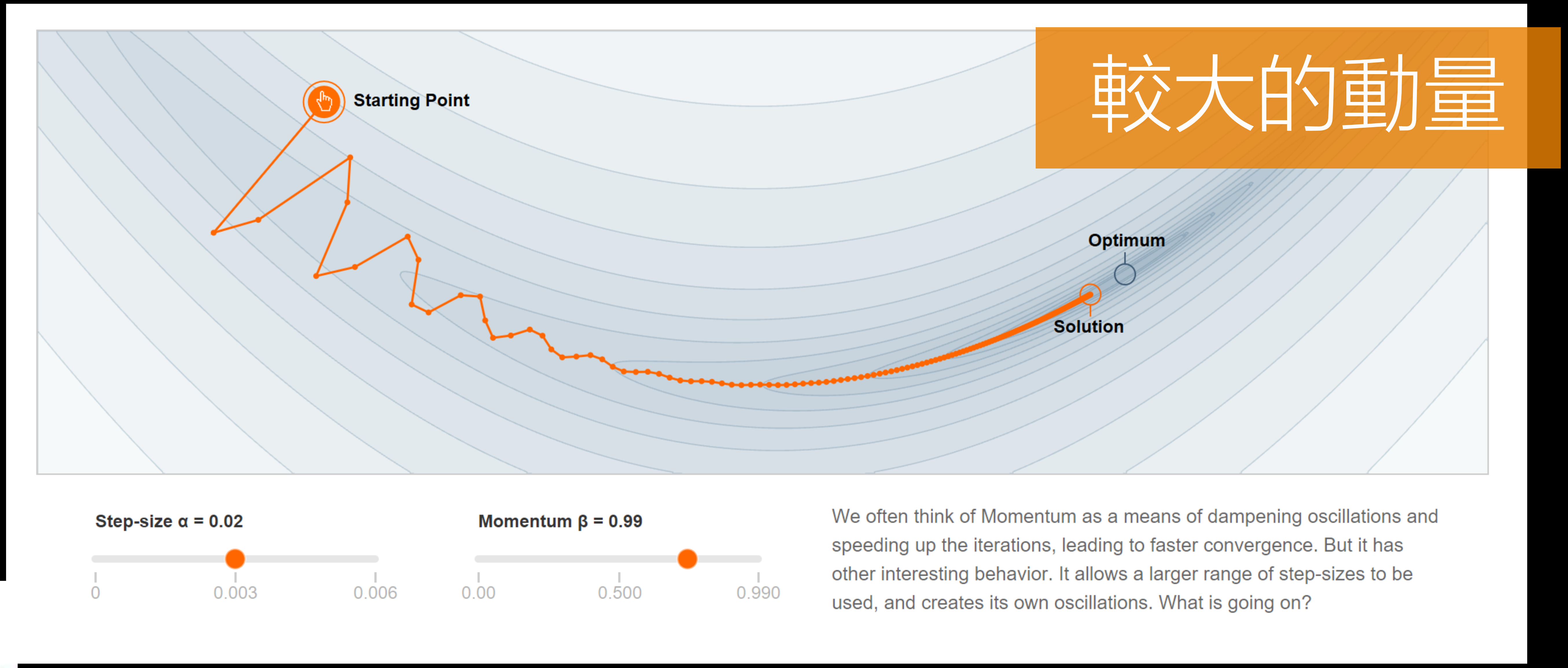


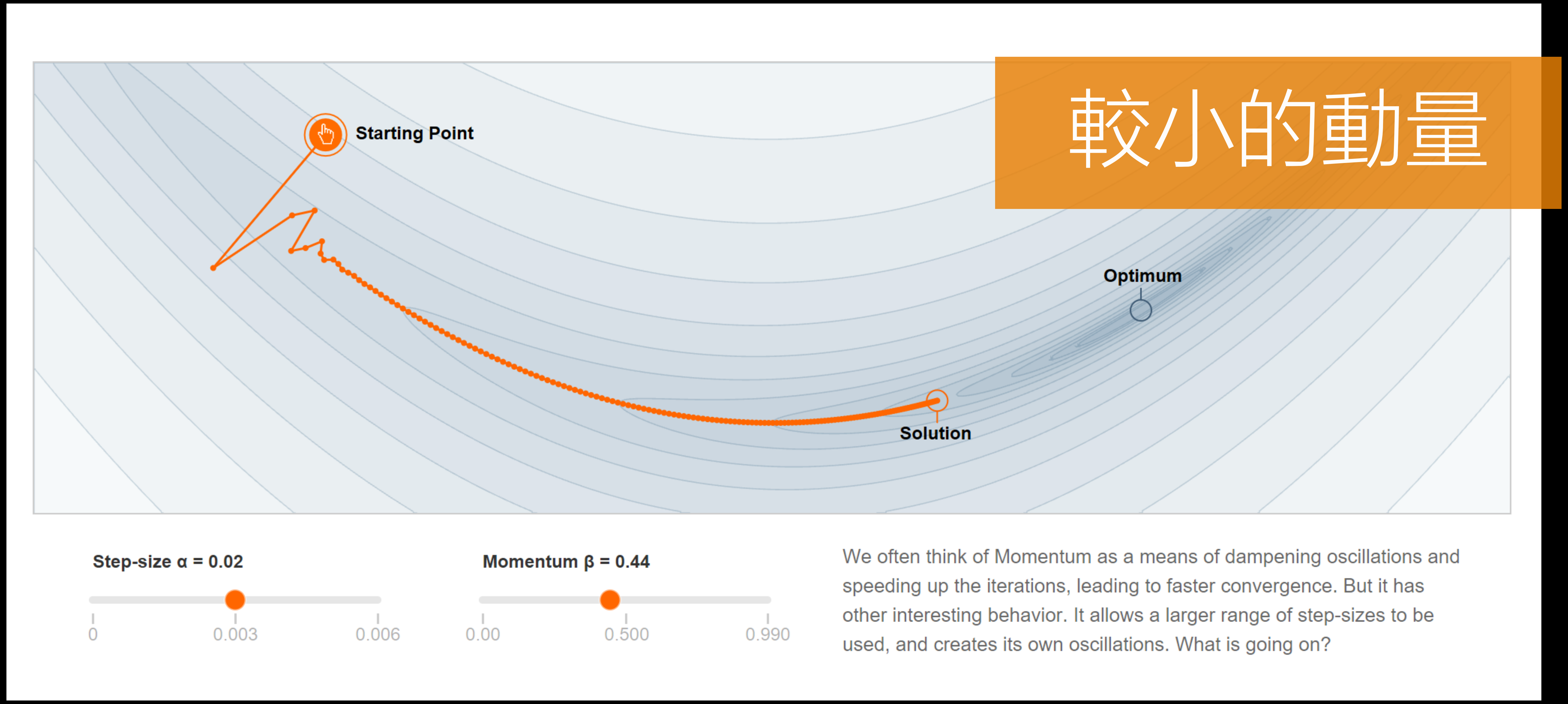


Leslie N. Smith 提出的 Triangular 和 Triangular2 迴圈學習率方法。左側的最大學習率和最小學習率保持不變。右側的區別在於每個週期之後學習率減半

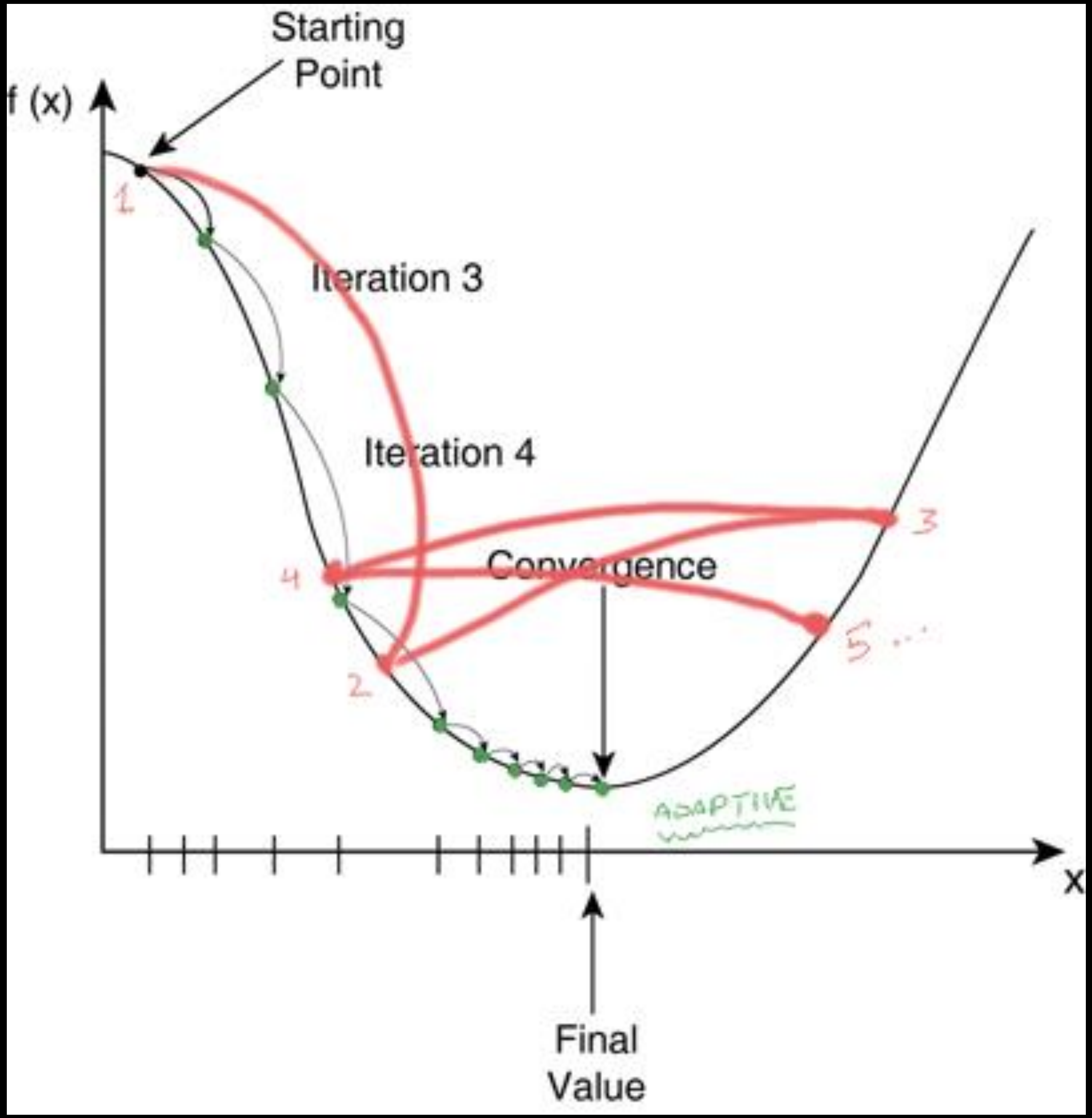
動量
Momentum

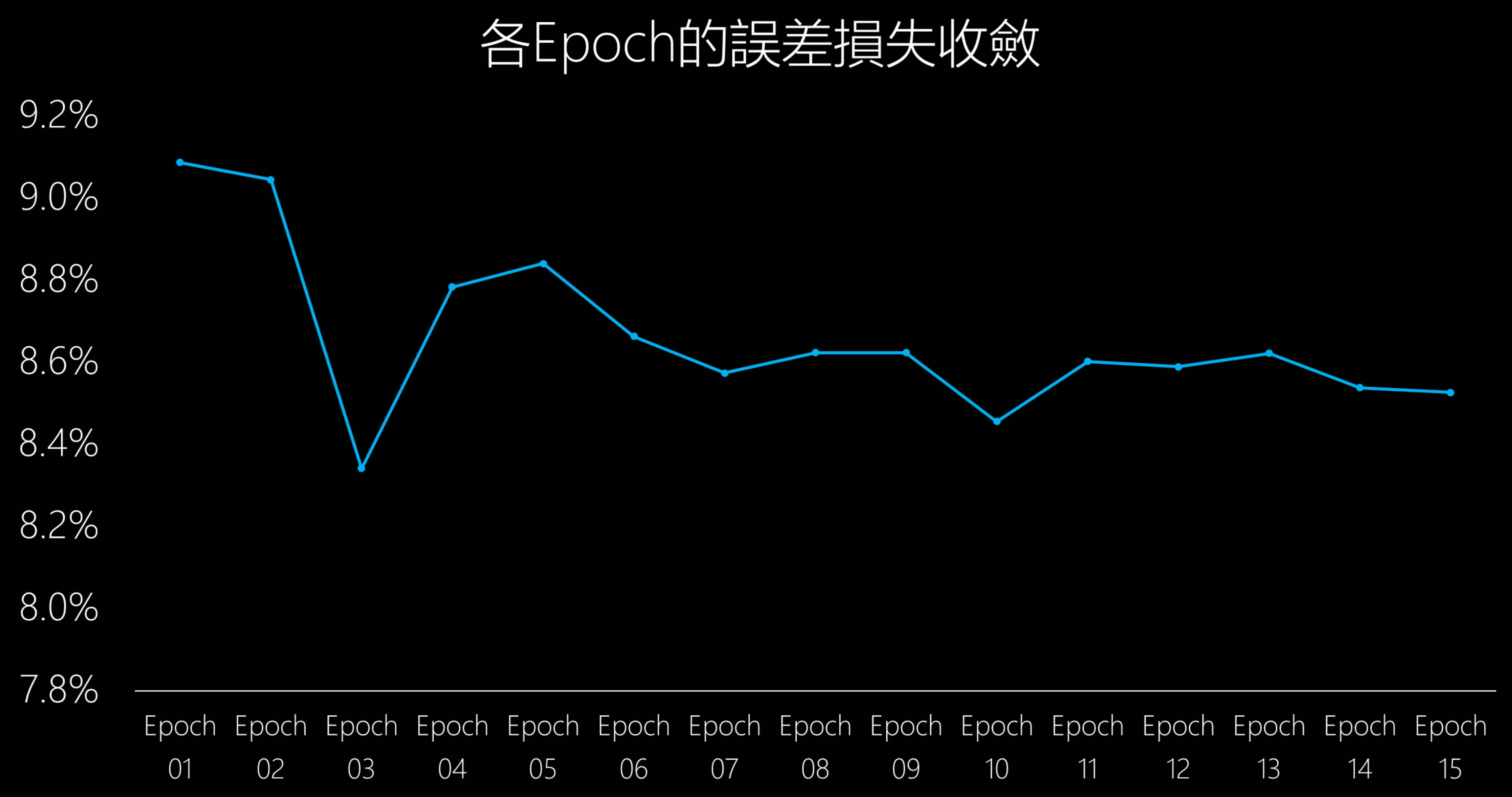

Momentum update
momentum step
actual step
gradient step
Nesterov momentum update
"lookahead" gradient step (bit different than original)
momentum step
actual step


較大的動量
Starting Point
Optimum
Solution
Step-size α = 0.02
0
0.003
0.006
Momentum β = 0.99
0.00
0.500
0.990
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

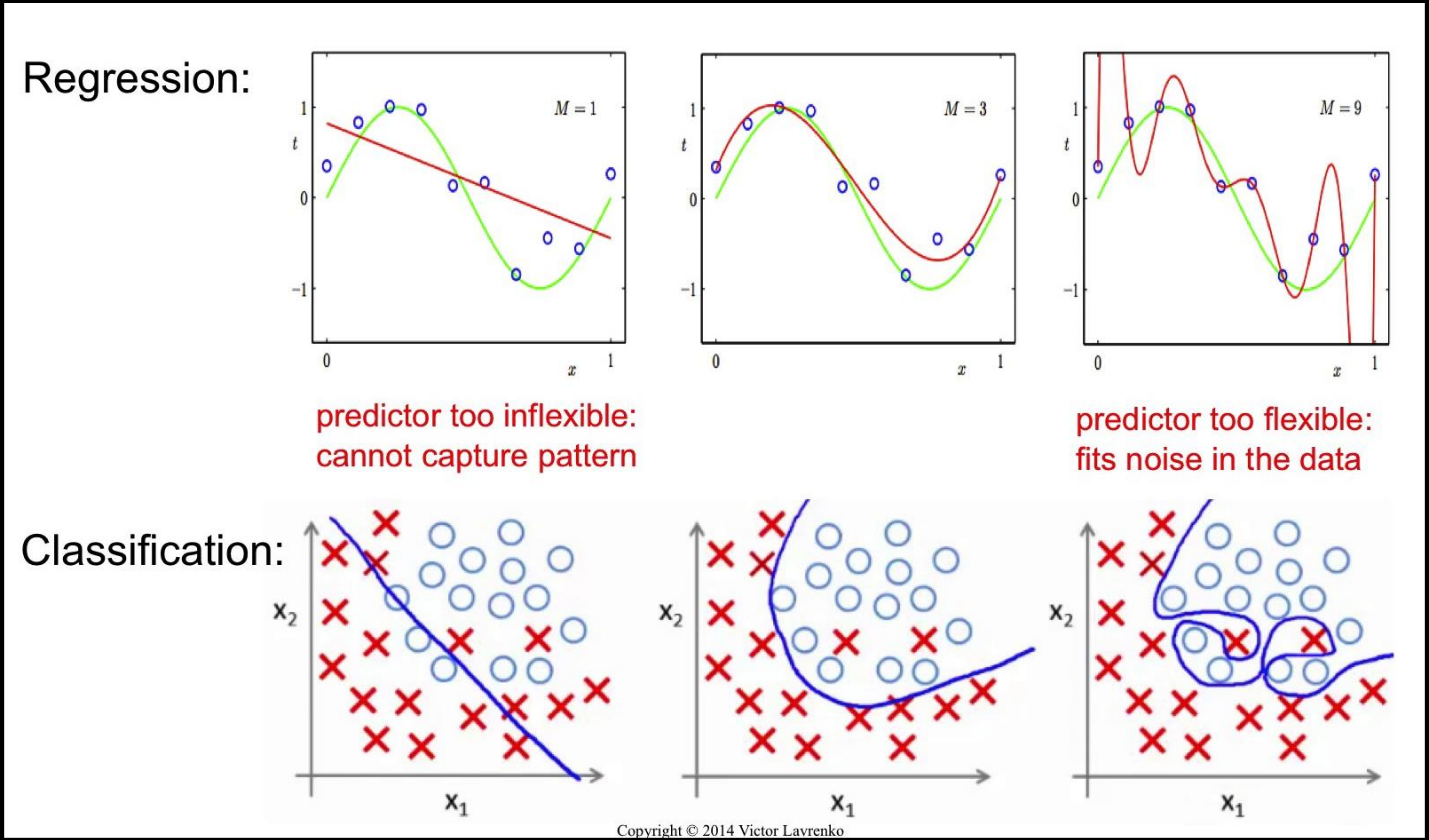

較小的動量
Starting Point
Optimum
Solution
Step-size α = 0.02
0
0.003
0.006
Momentum β = 0.44
0.00
0.500
0.990
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

這些參數構成了訓練過程





欠擬合(underfitting)與過擬合(Overfitting)



L1 regulariZation L1 正則會讓模型變稀疏

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|.$$

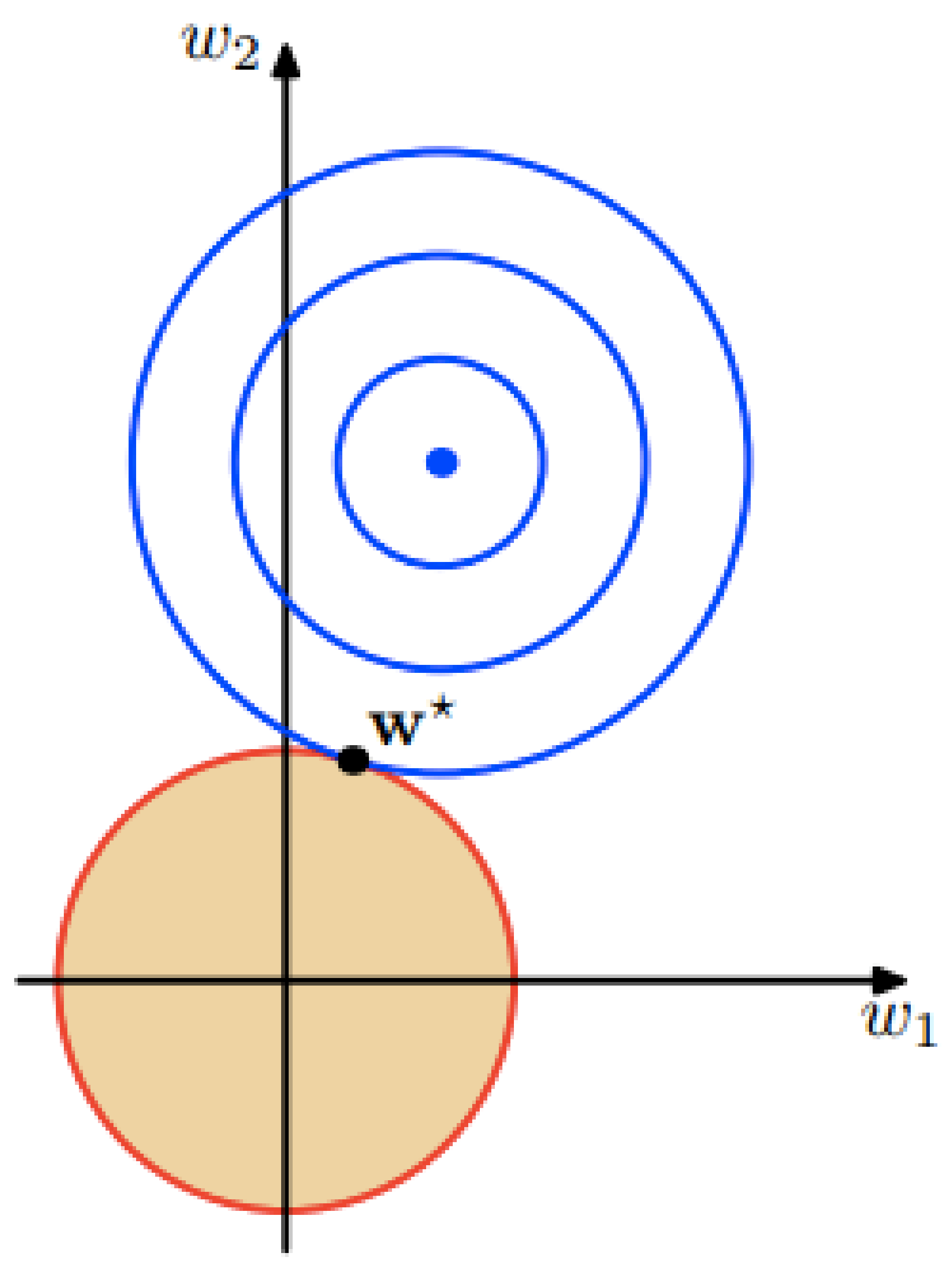
因為有了L1正則的懲罰項，
因此傾向往部分權重靠攏，
其餘變零

L2 regulariZation L2正則又稱之為權重遞減

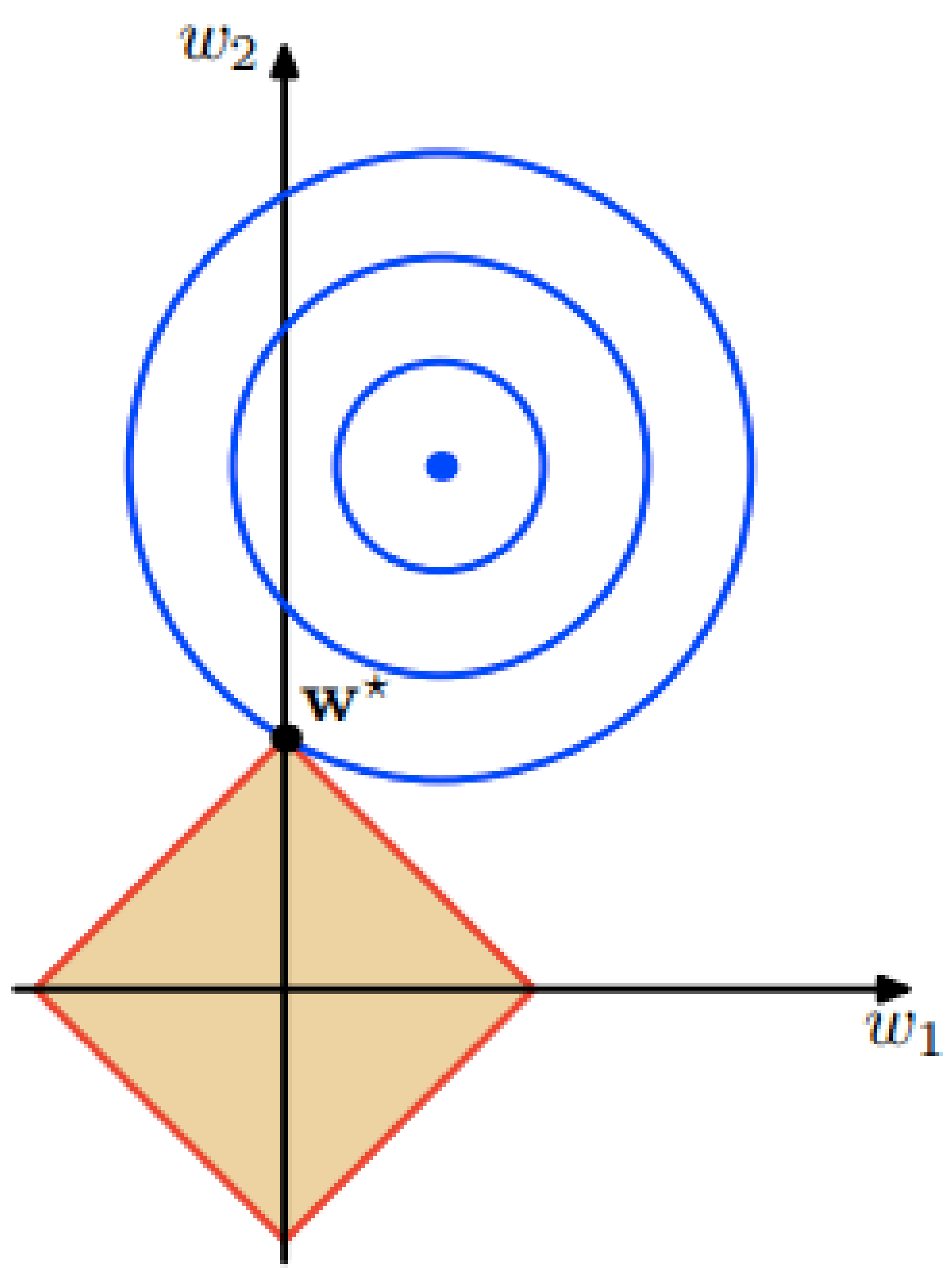
$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2,$$

因為有了L2正則的懲罰項，
因此傾向往全體權重最小的
方向邁進

L2正則



L1正則



那些因素會影響模型的成果

數據
模型結構
超參數
避免過擬合
最佳化方法
正歸化
活化函數
權重初始化

Q&A