深度學習基本原理
(Fundamentals of Deep Learning)

第二部分：如何訓練類神經網路 (How a Neural Network Trains)

課程大綱

- 第 1 部分：深度學習簡介
- 第 2 部分：神經網路如何訓練
- 第 3 部分：卷積神經網路 (Convolutional Neural Networks)
- 第 4 部分：資料增強與部署
- 第 5 部分：預訓練模型
- 第 6 部分：進階架構

課程練習回顧

剛才發生了什麼事？

載入並視覺化我們的資料

編輯資料(正確的資料格式, 正規化(Normalization), 轉換成類別)

建立模型

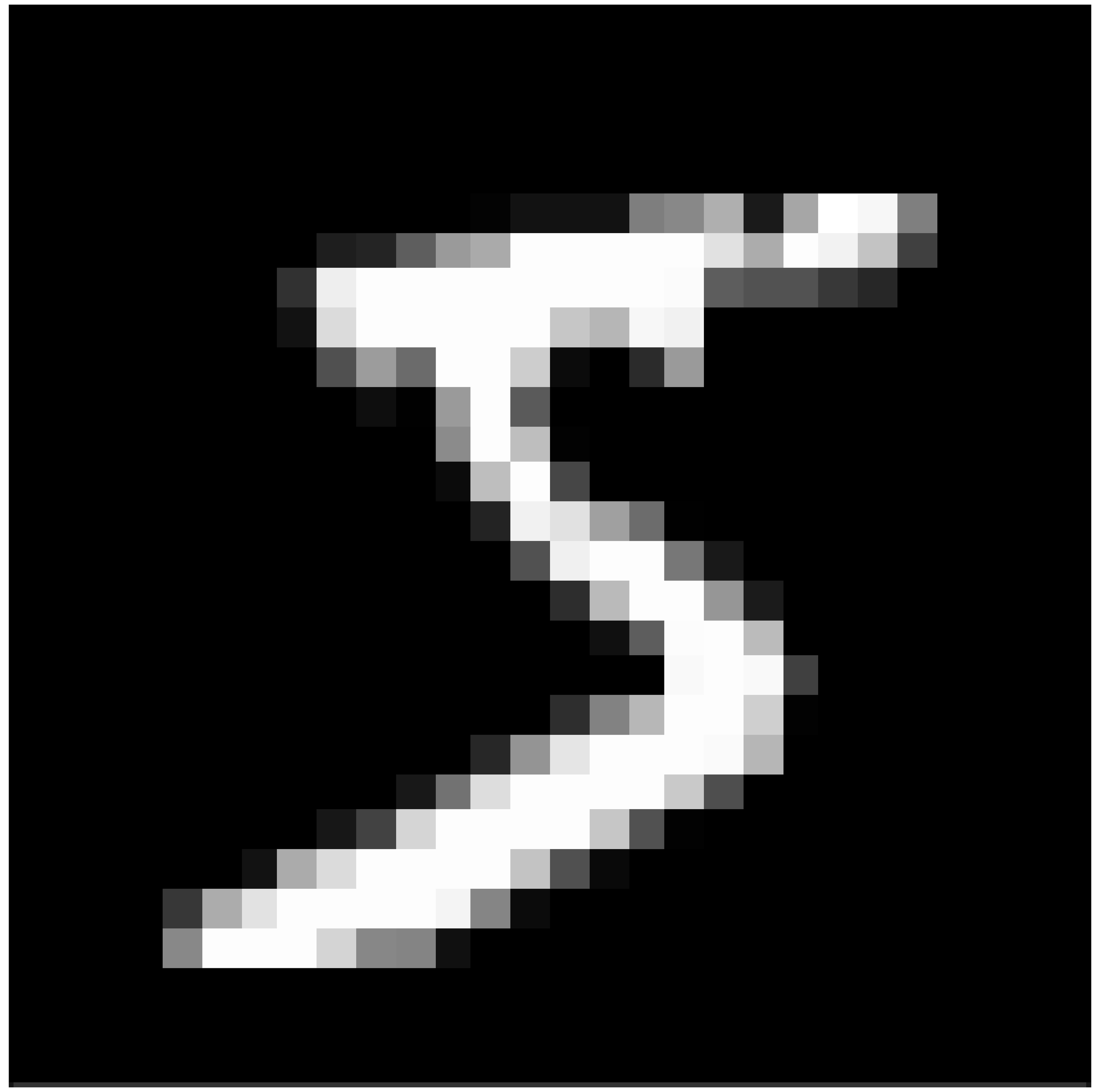
編譯(Compiled)模型

用資料訓練了模型

資料準備(Data Preparation)

以陣列格式輸入 (Input as an Array)

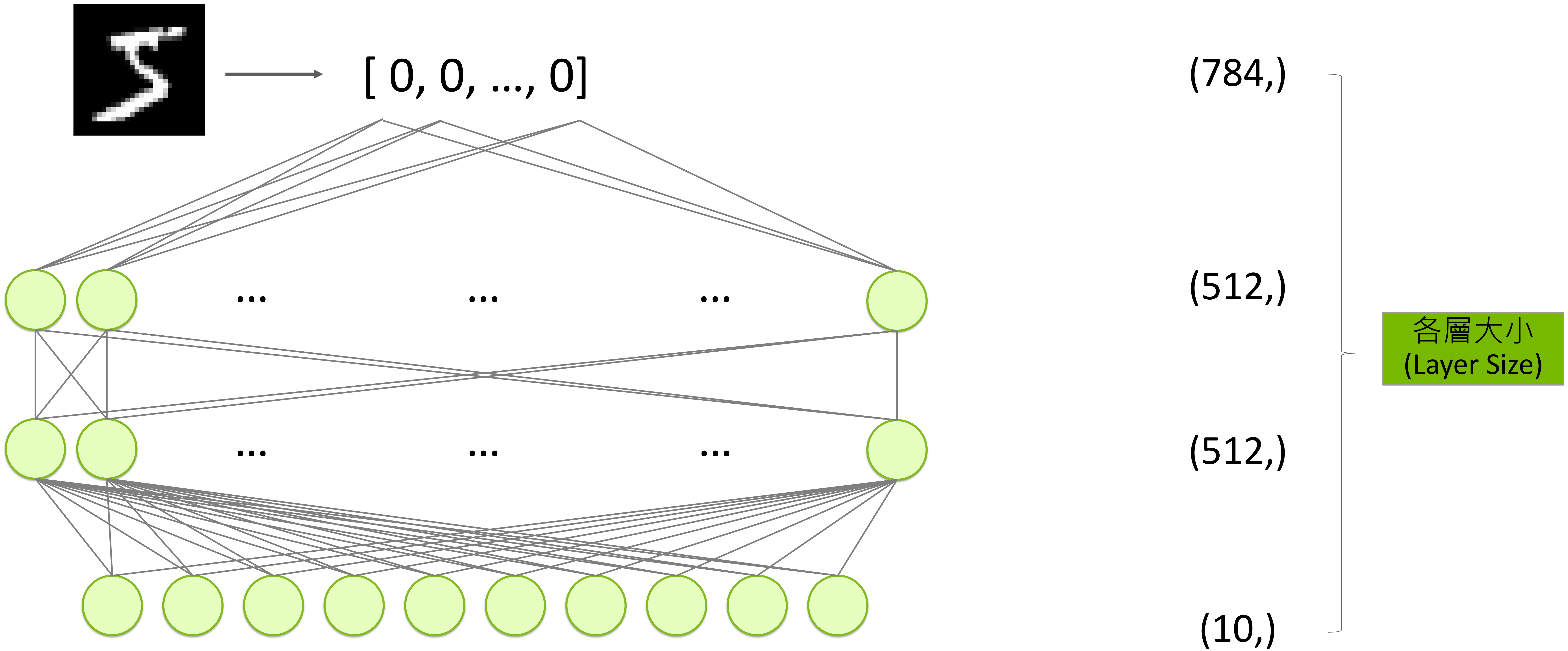
28



28

[0,0,0,24,75,184,185,78,32,55,0,0,0...]

未訓練的模型(An Untrained Model)

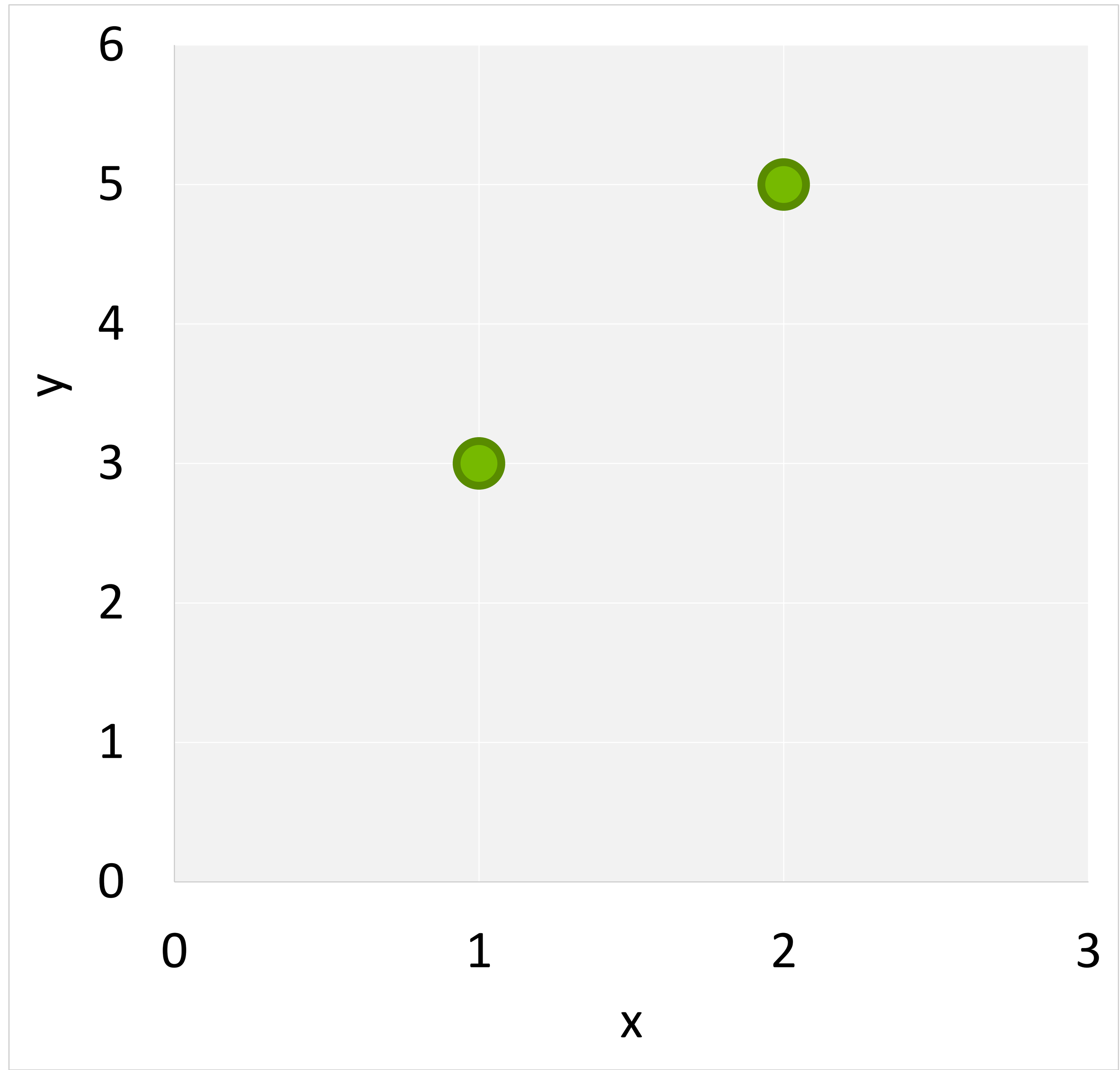


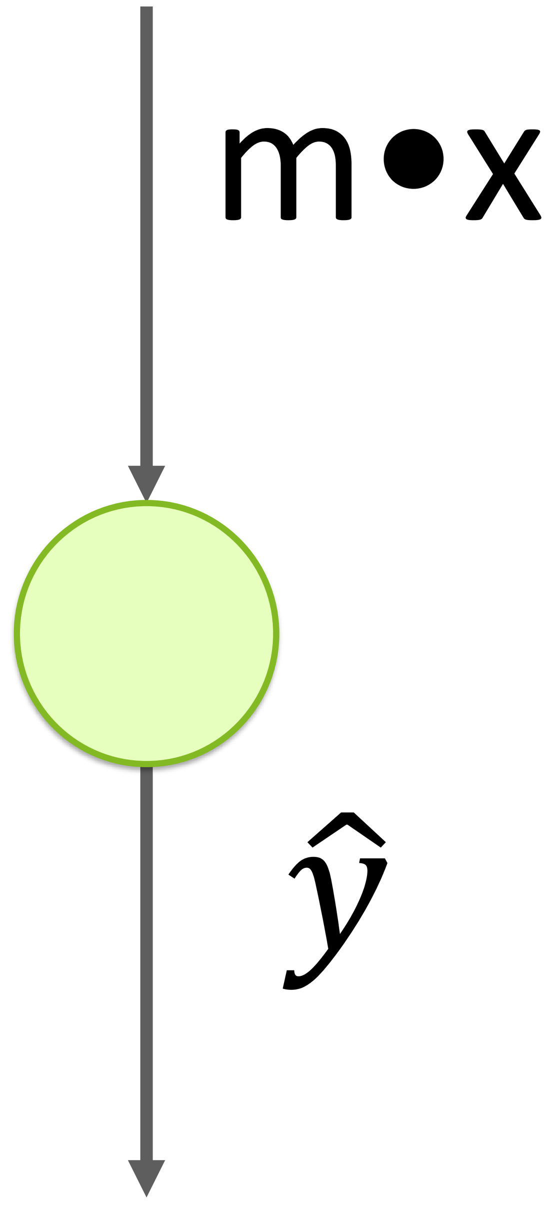
更簡化的模型(A Simpler Model)

更簡化的模型(A Simpler Model)

$$y = mx + b$$

x	y
1	3
2	5





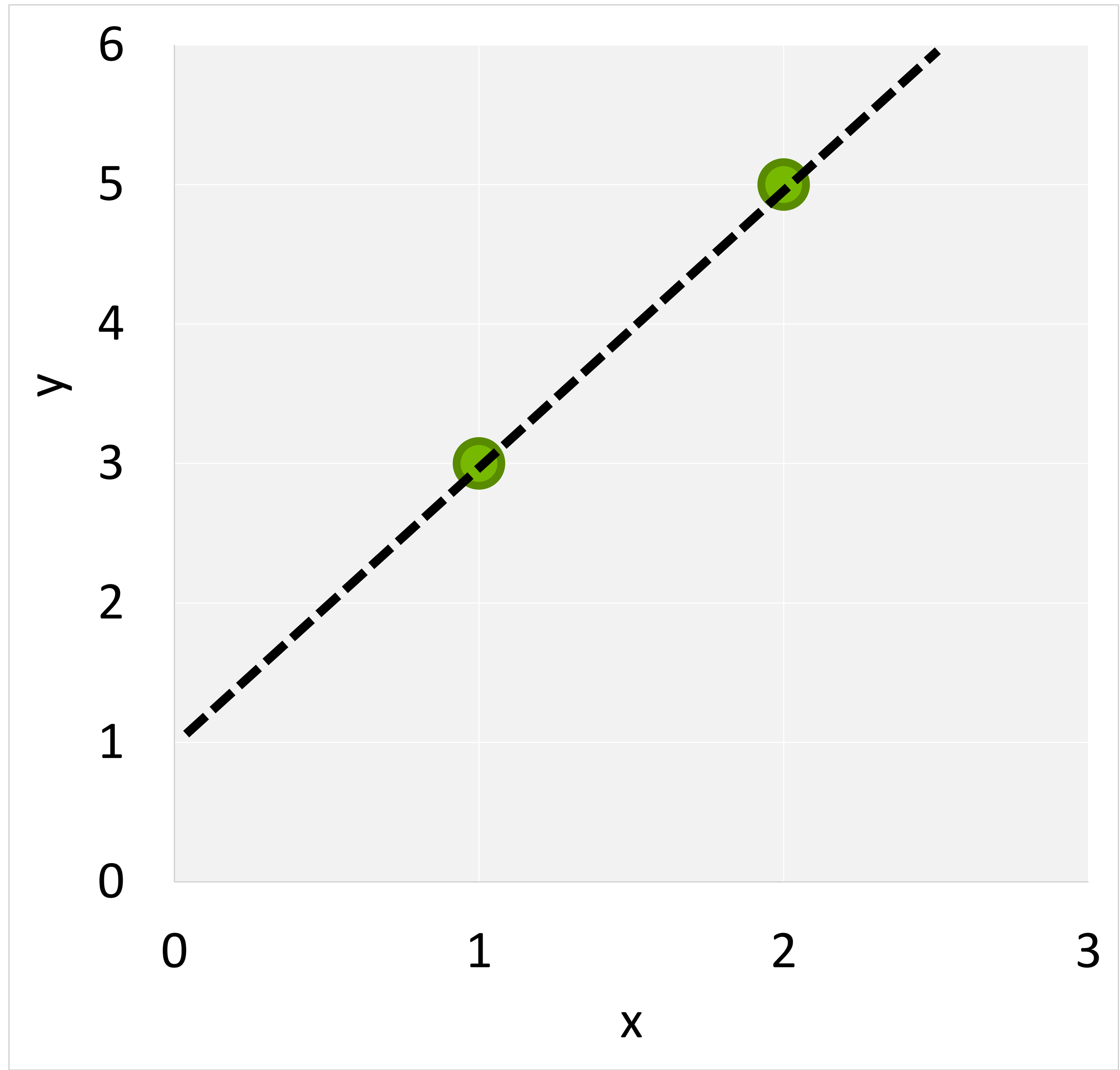
$$m = ?$$

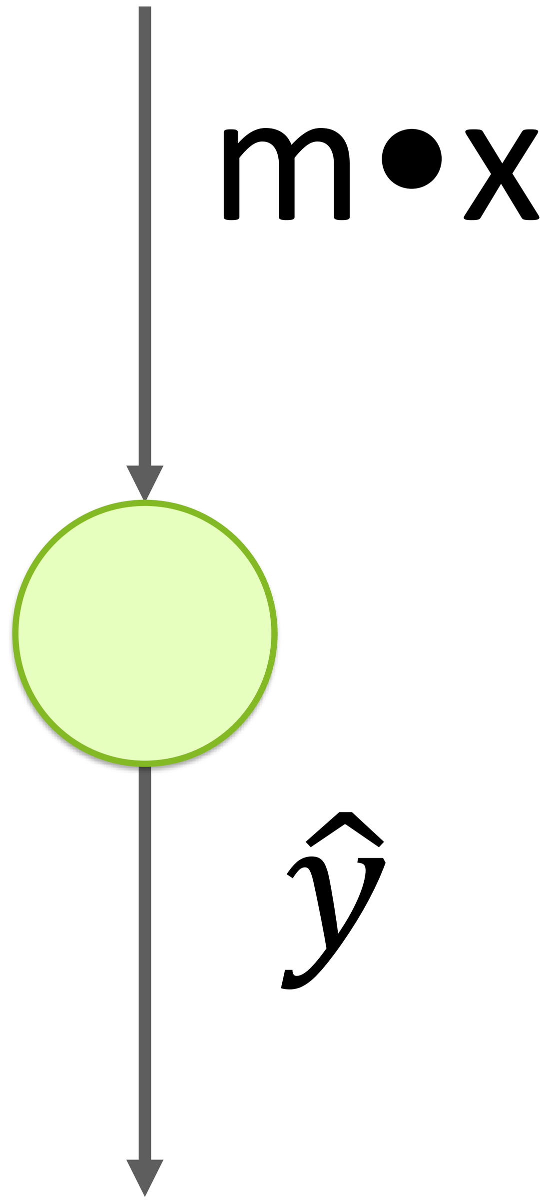
$$\mathrm{b} = ?$$

更簡化的模型(A Simpler Model)

$$y = mx + b$$

x	y
1	3
2	5





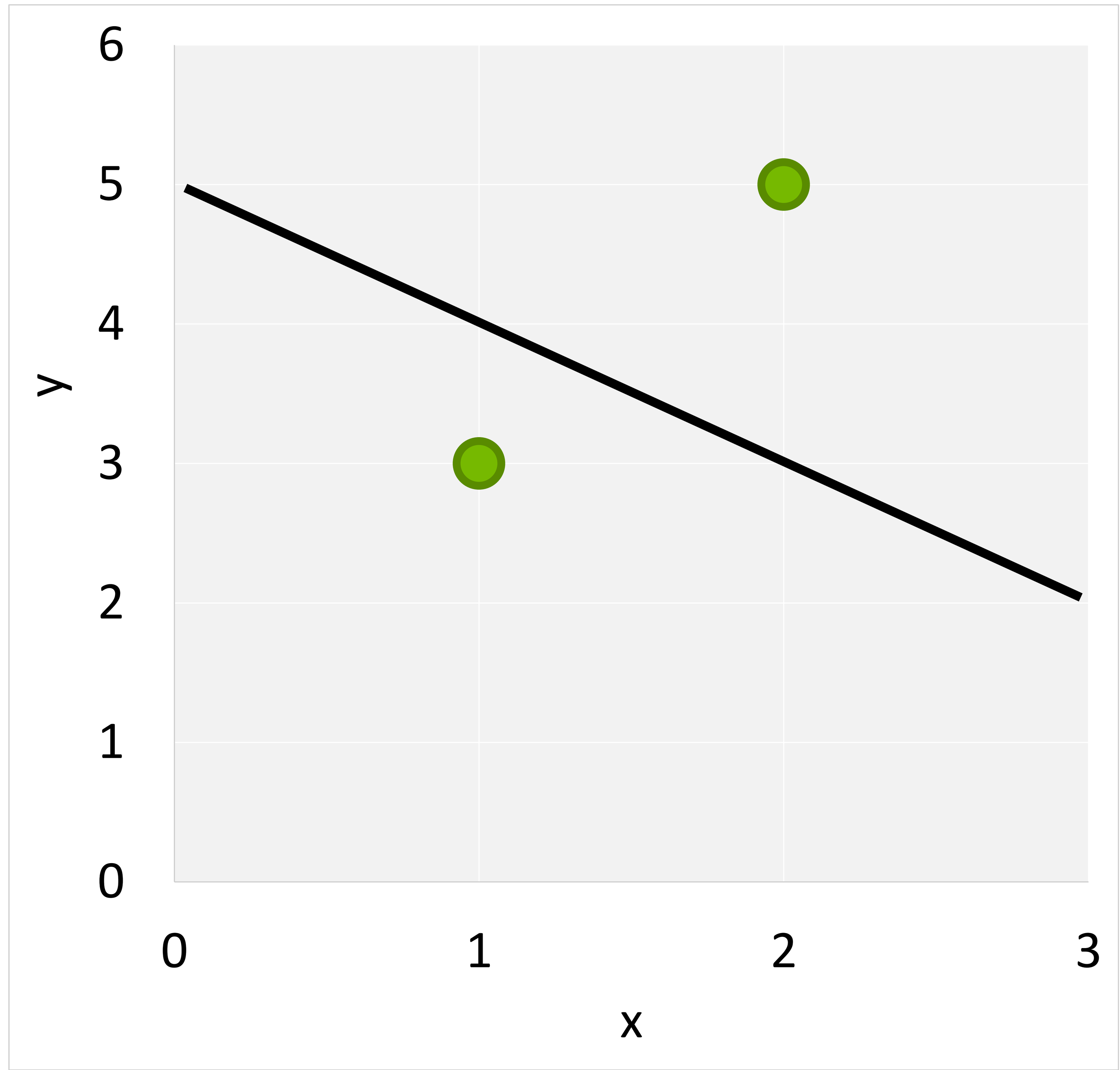
$m = ?$

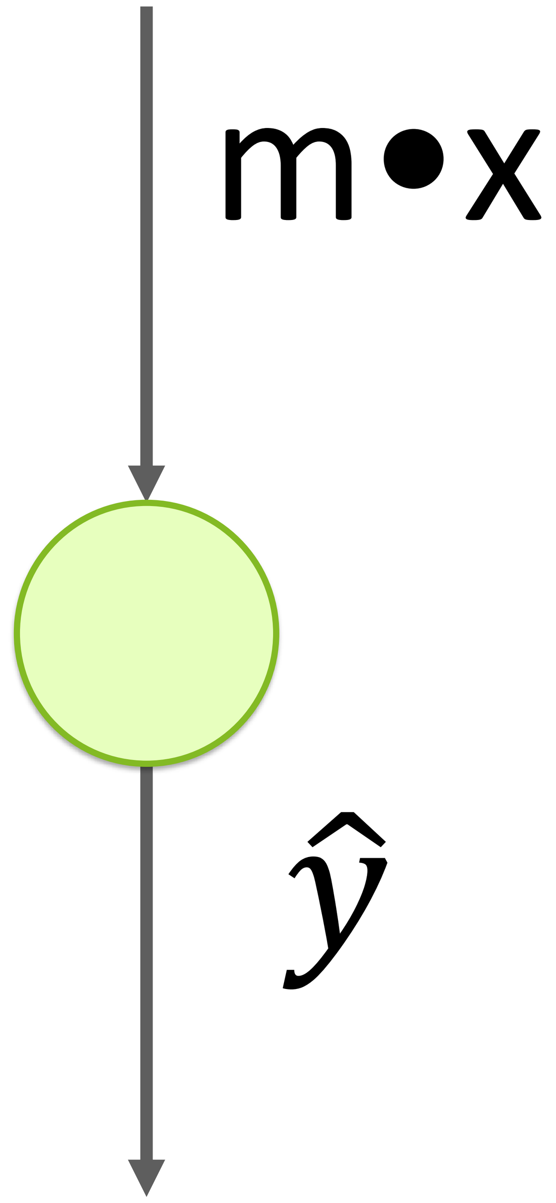
$b = ?$

更簡化的模型(A Simpler Model)

$$y = mx + b$$

x	y	$\hat{y}$
1	3	4
2	5	3





Start Random隨機開始(Start Random)

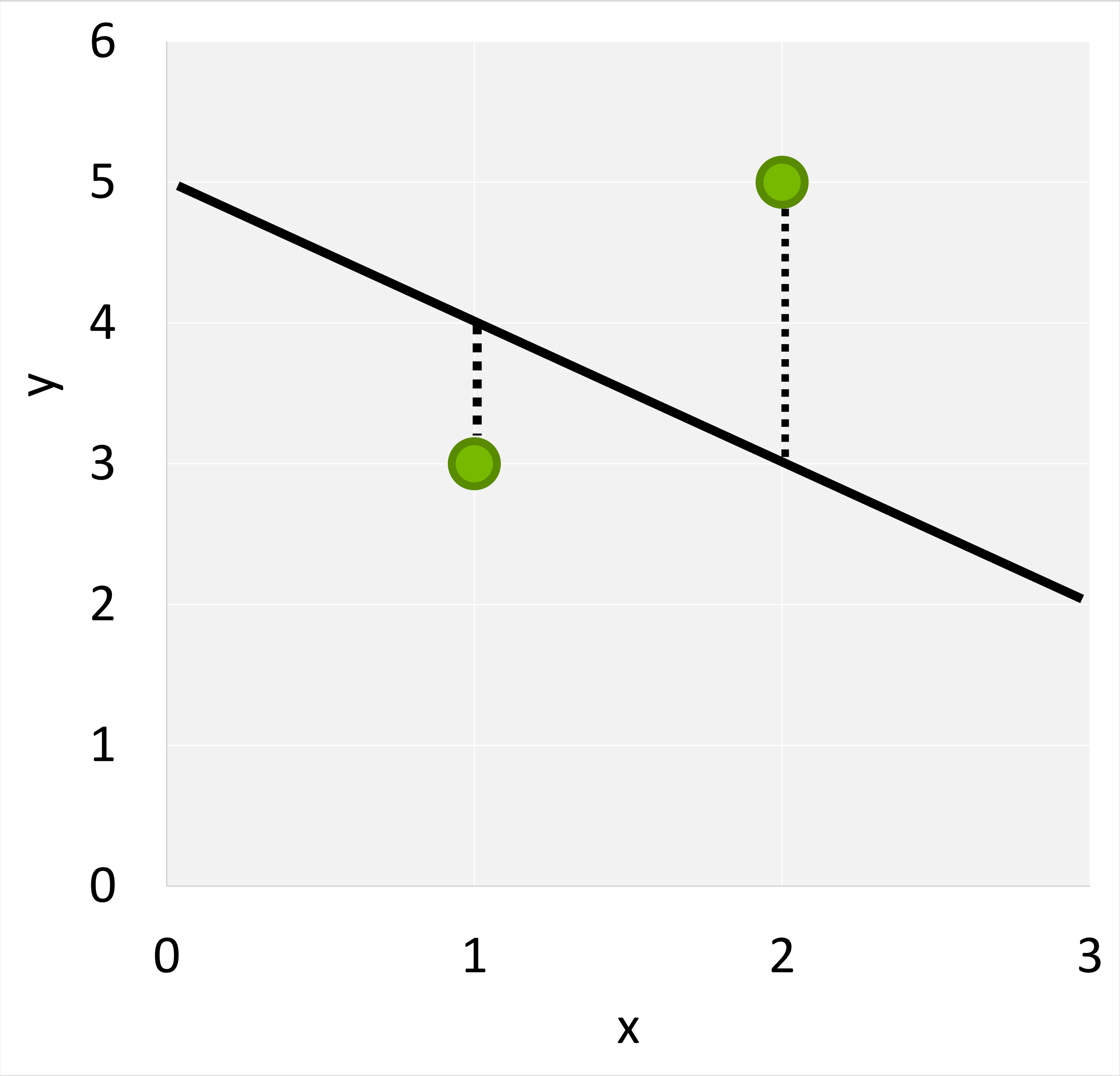
$$m = -1$$

$$\mathrm{b} = 5$$

更簡化的模型(A Simpler Model)

$$y = mx + b$$

x	**y**	$\hat{y}$	err^2
1	3	4	1
2	5	3	4
		MSE =	2.5
		RMSE =	1.6



$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

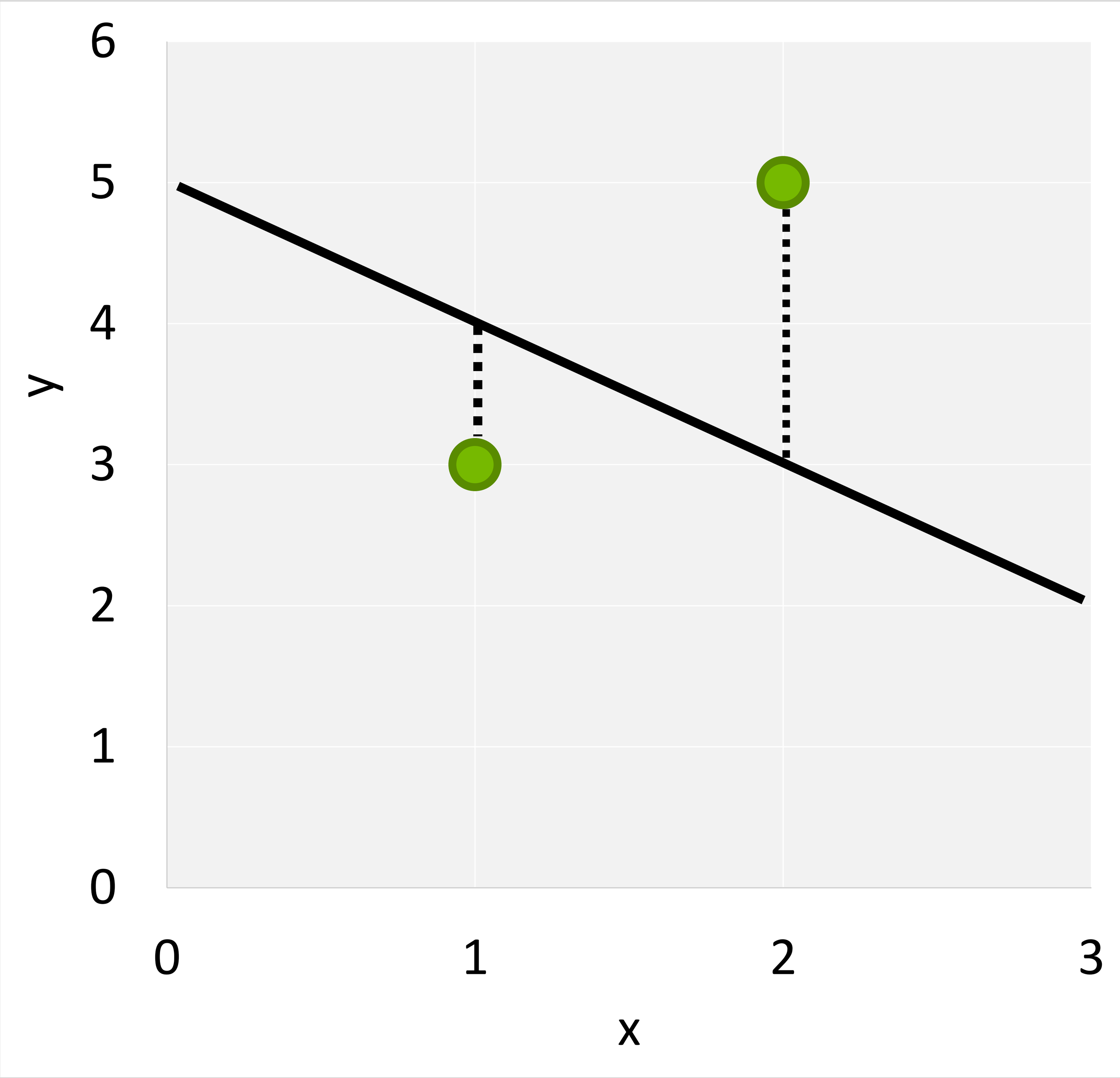
更簡化的模型(A Simpler Model)

$$y = mx + b$$

x	y	$\hat{y}$	err^2
1	3	4	1
2	5	3	4

MSE = 2.5

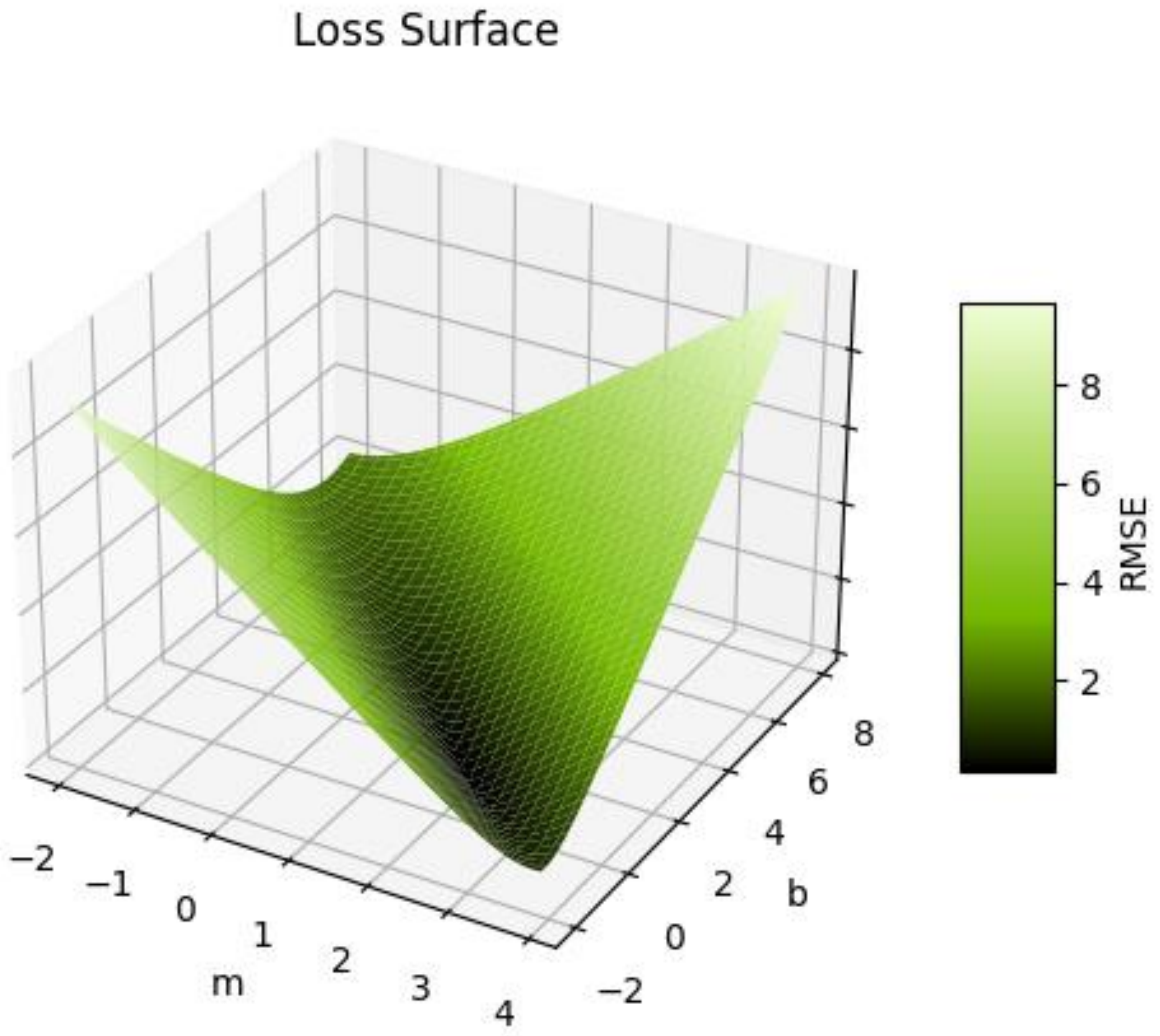
RMSE = 1.6



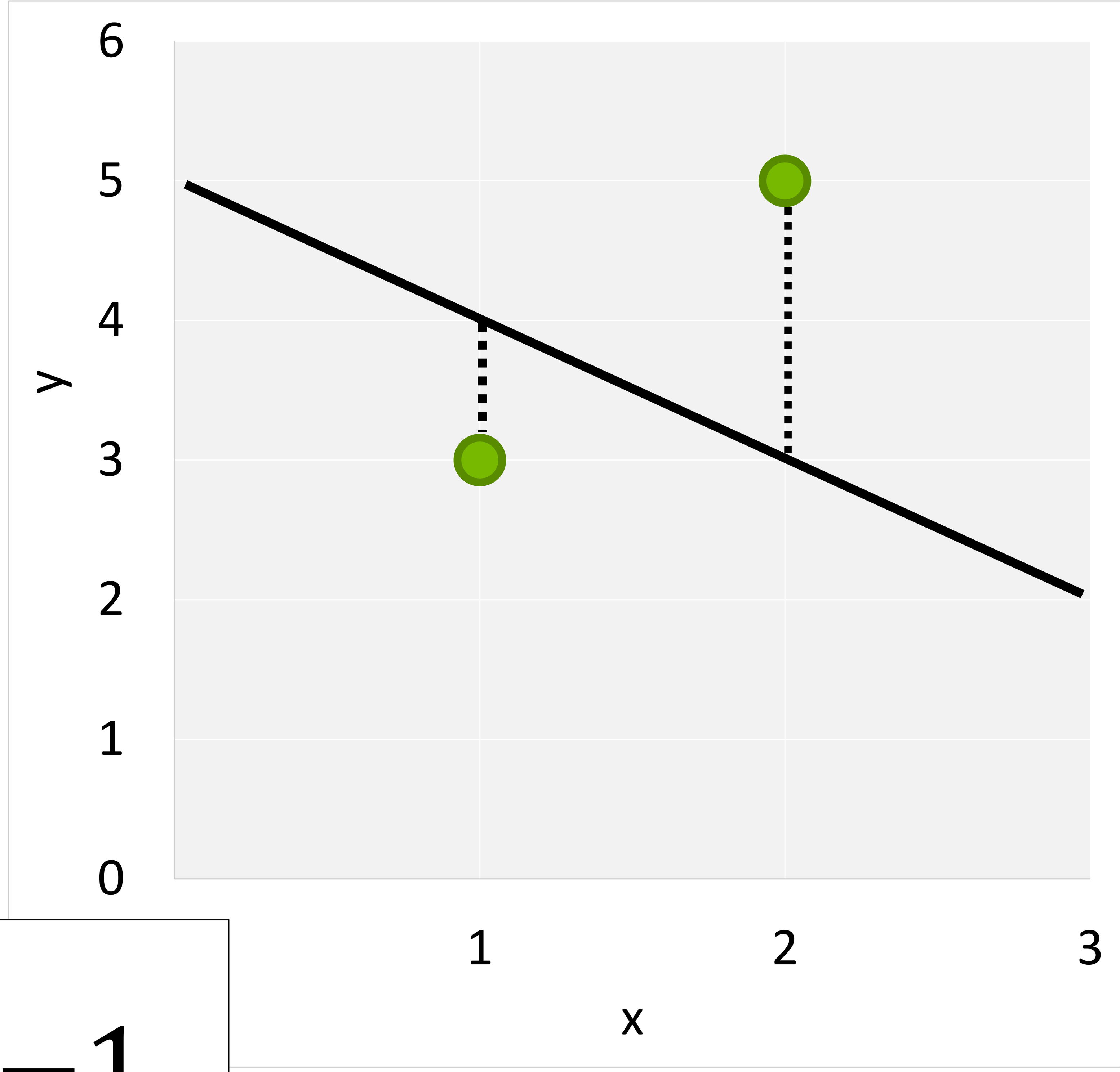

```python
data = [(1, 3), (2, 5)]
m = -1
b = 5


def get_rmse(data, m, b):
    """Calculates Mean Square Error"""
    n = len(data)
    squared_error = 0
    for x, y in data:
        # Find predicted y
        y_hat = m*x+b
        # Square difference between
        # prediction and true value
        squared_error += (
            y - y_hat)**2
    # Get average squared difference
    mse = squared_error / n
    # Square root for original units
    return mse ** .5
```

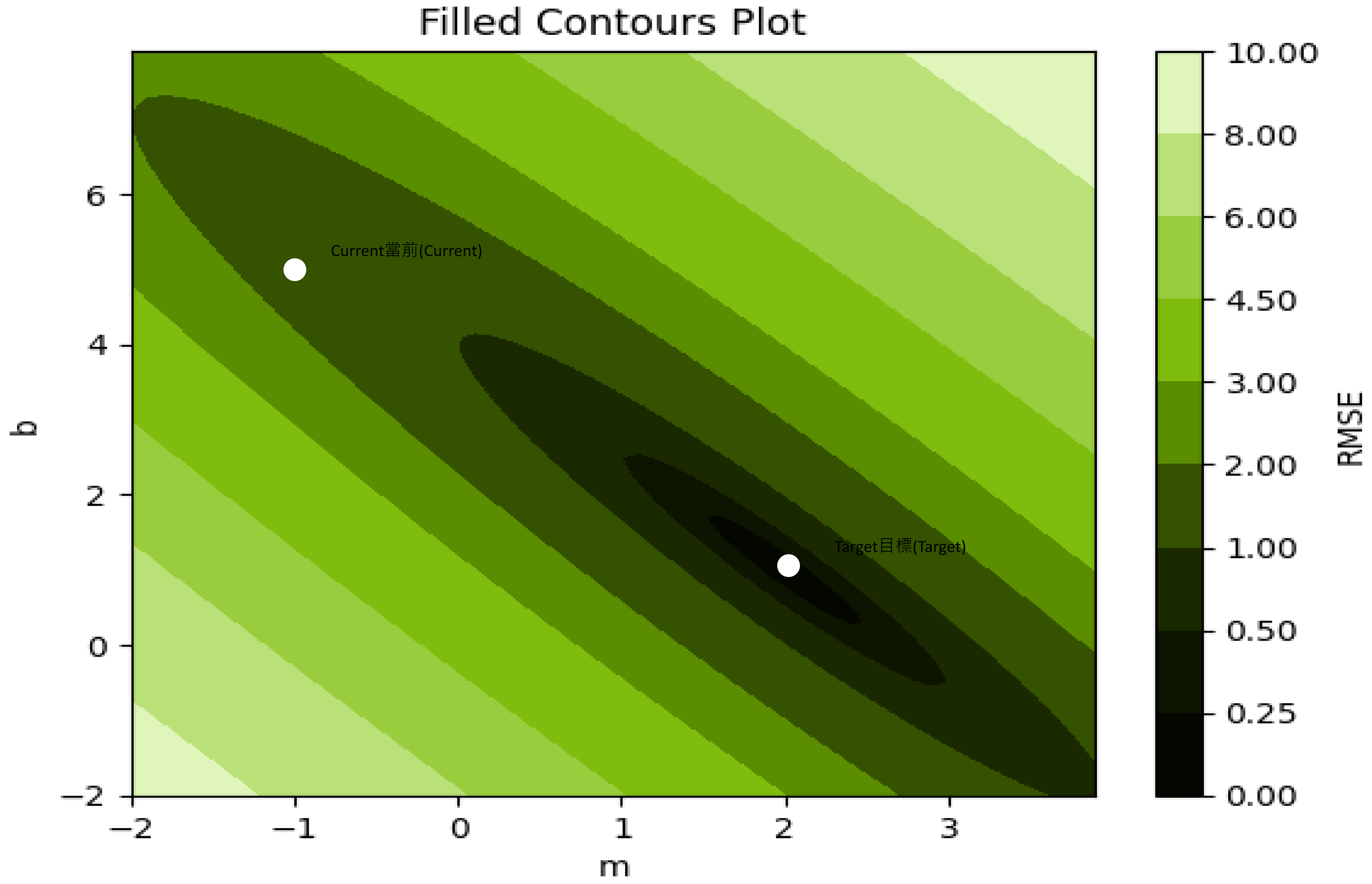
損失曲線(Loss Curve)



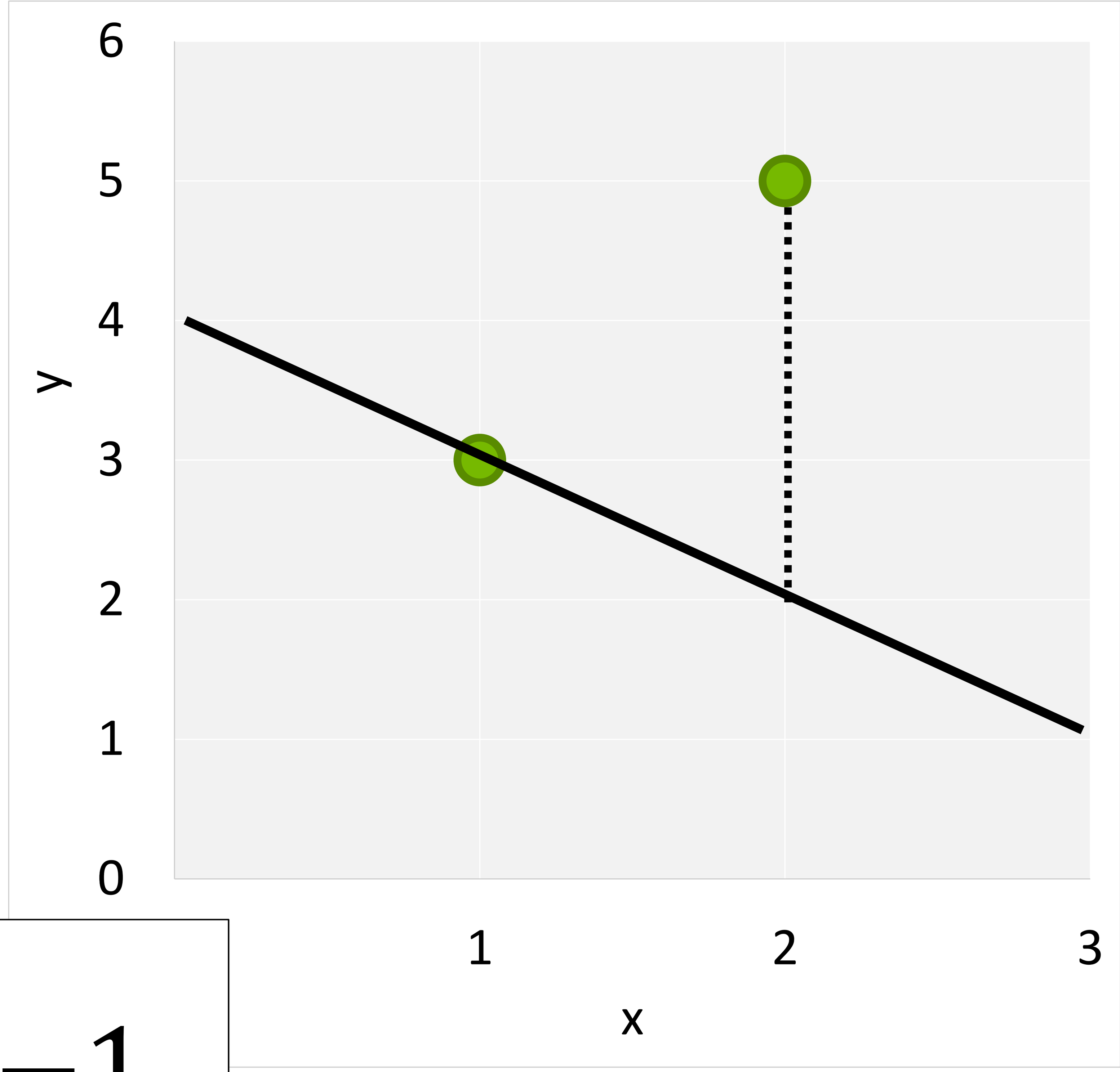
損失曲線(Loss Curve)



$$m = -1$$
$$b = 5$$

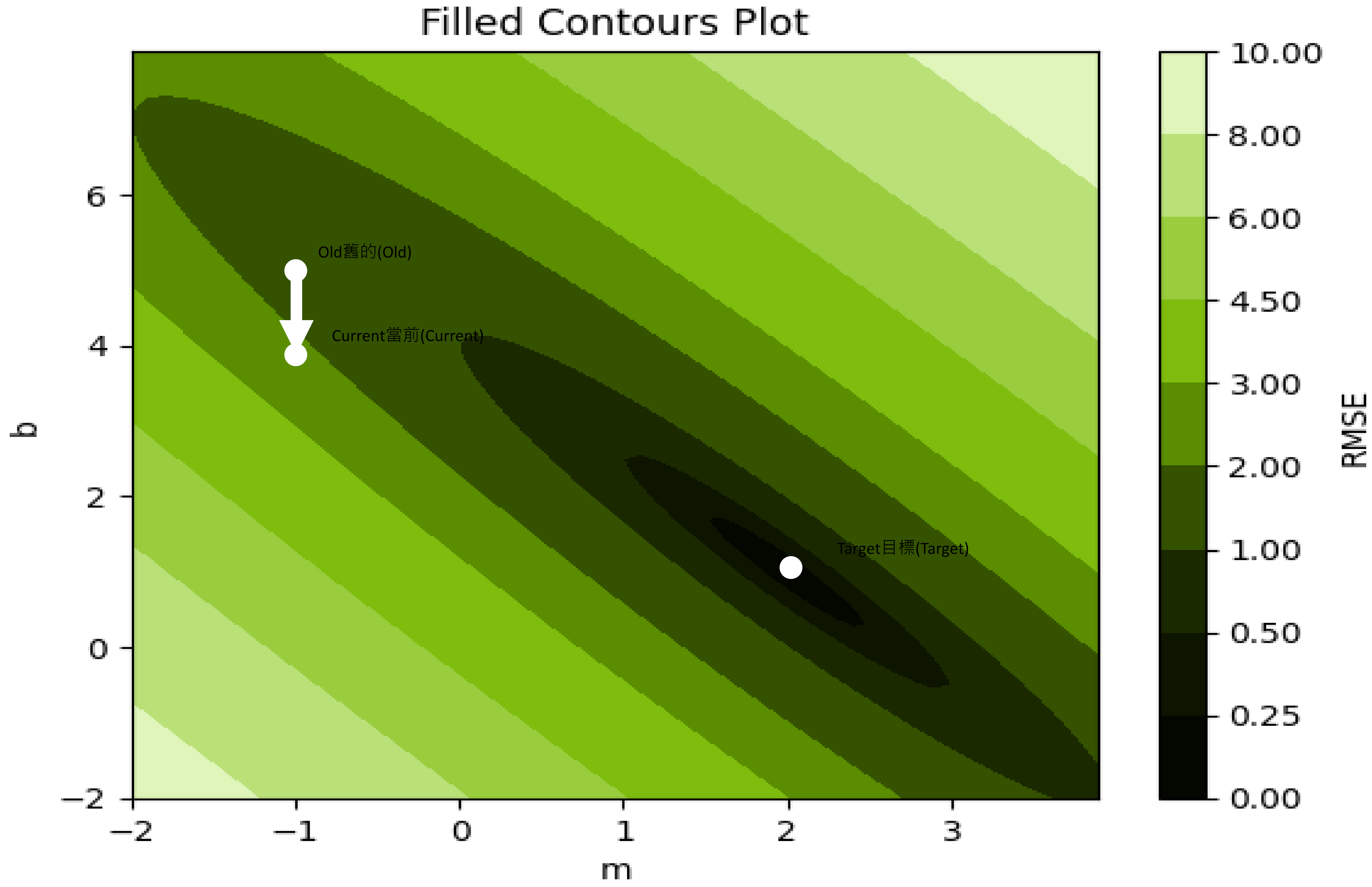


損失曲線(Loss Curve)

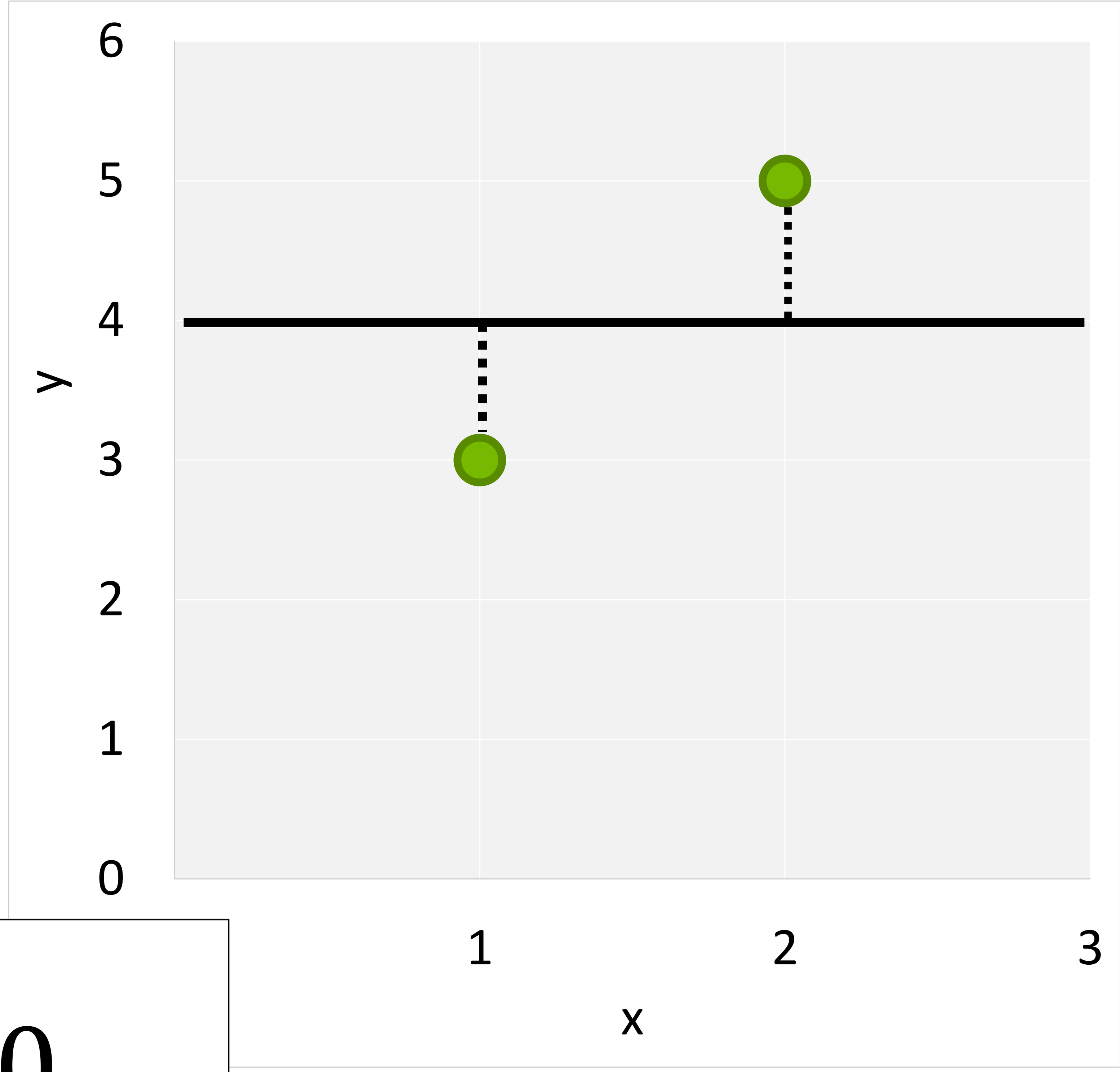


$$m = -1$$

$$\mathrm{b} = 4$$

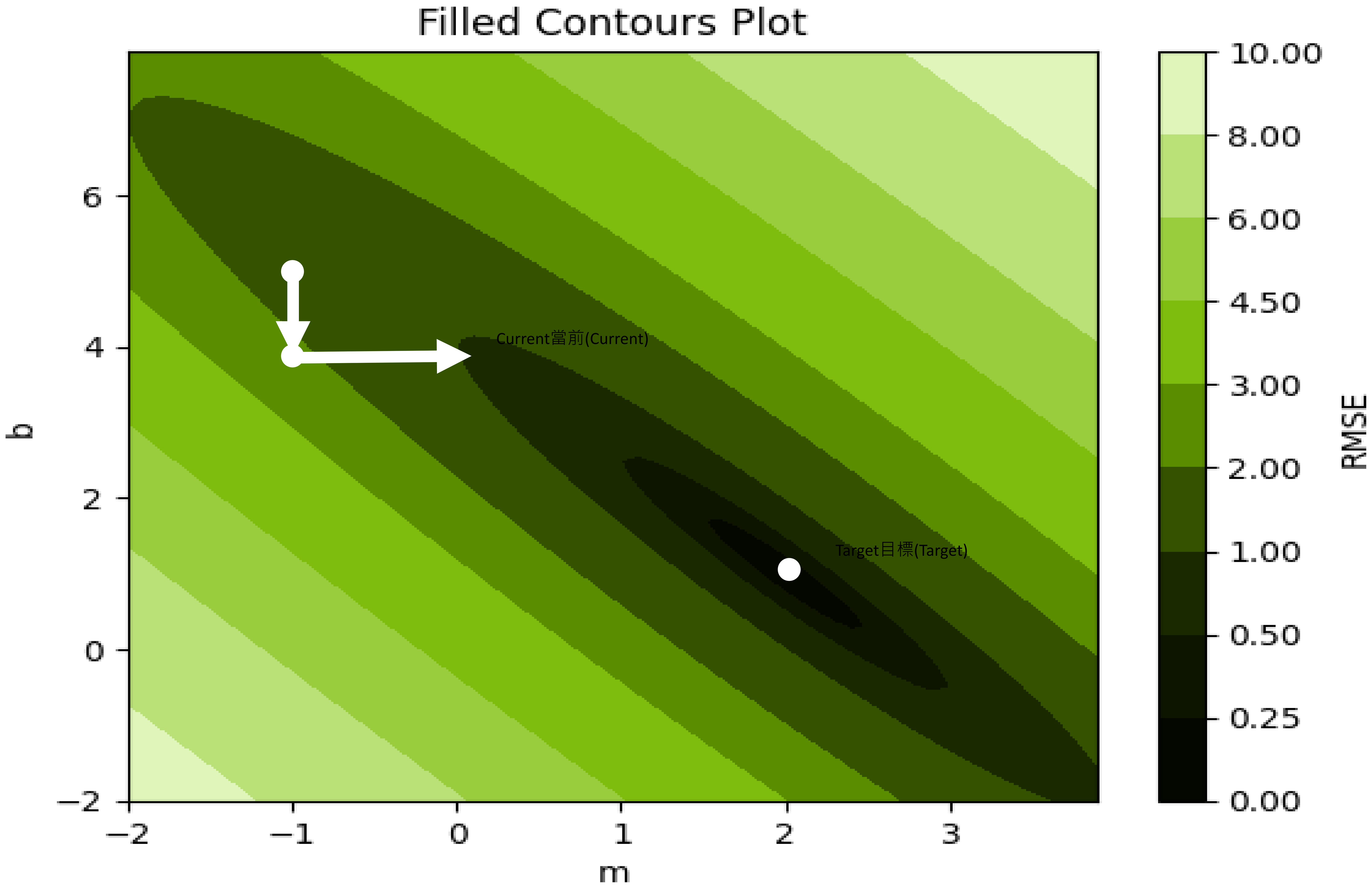


損失曲線(Loss Curve)



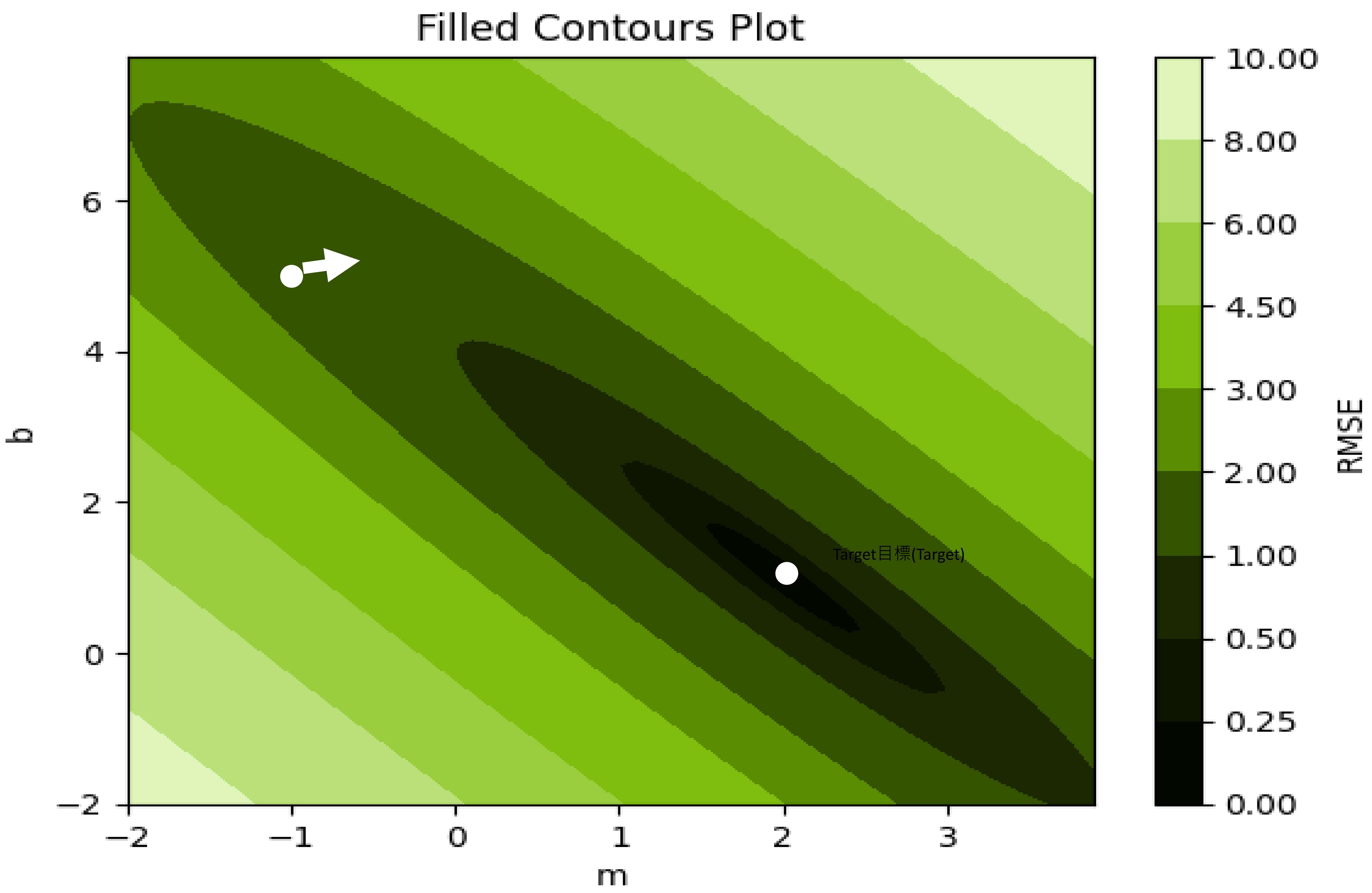
$m = 0$

$b = 4$



損失曲線(Loss Curve)

梯度 (Gradient)	損失函式(loss function)減少最多的方向
λ: 學習率 (learning rate)	移動的距離
Epoch	使用完整資料集更新模型
批次(Batch)	完整資料集的樣本
步驟(Step)	權重參數的更新

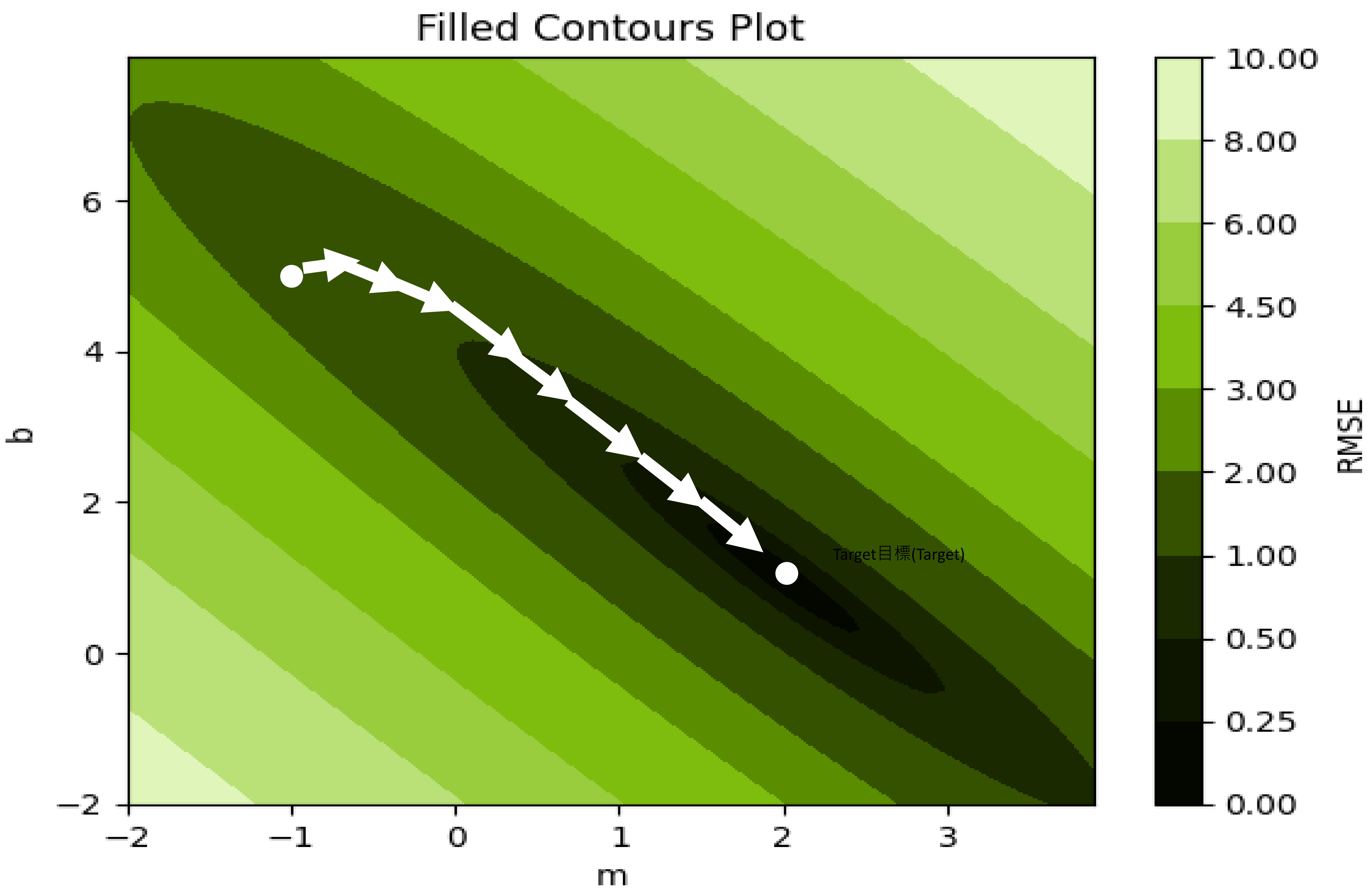


損失曲線(Loss Curve)

梯度 (Gradient)	損失函式(loss function)減少最多的方向
λ: 學習率 (learning rate)	移動的距離
Epoch	使用完整資料集更新模型
批次(Batch)	完整資料集的樣本
步驟(Step)	權重參數的更新

$$w := w - \eta \cdot \frac{\partial J}{\partial w}$$

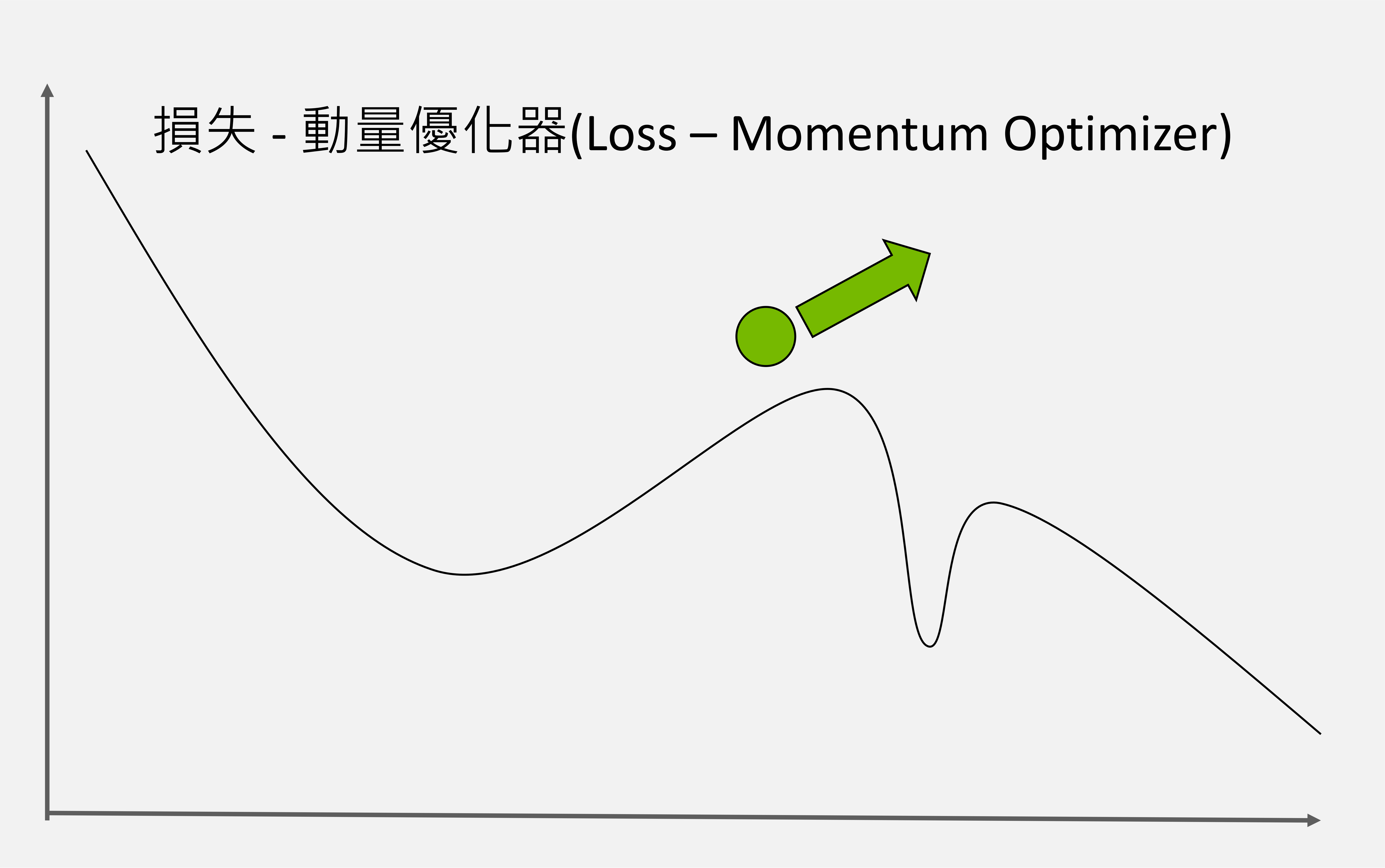
$$b := b - \eta \cdot \frac{\partial J}{\partial b}$$



完整學習一次的流程（訓練流程）

1.資料準備（Data Preparation）
　將原始資料整理、標註，並劃分為訓練集與驗證集。
2.初始化模型參數（Model Initialization）
　設定初始權重、偏差等參數。
3.設定訓練超參數（Hyperparameters）
　例如：
　- 學習率（learning rate）
　- 批次大小（batch size）
　- 訓練回合數（epoch）等。
4.進入訓練迴圈：For 每個 Epoch（回合）
　4.1 **將訓練資料分成多個 Batch（小批次）**
　　若批次大小為 32，而資料筆數為 320，則每個 epoch 有 10 個 batch。
　4.2 **對每個 Batch 執行以下 Step（步驟）：**
　　- ➤ **前向傳播（Forward Pass）**：
　　將資料輸入模型，計算預測結果。
　　- ➤ **損失計算（Loss Calculation）**：
　　計算預測結果與真實值的差異（Loss Function）。
　　- ➤ **反向傳播（Backward Pass / Backpropagation）**：
　　透過鏈式法則計算各參數對損失的偏導數（即梯度）。
　　- ➤ **參數更新（Step）**：
　　使用梯度與學習率更新模型參數（通常使用 SGD、Adam 等優化器）。
5.一整輪資料訓練完成，代表一個 Epoch 結束
6.可選：使用驗證集進行評估（Validation）
　觀察模型在未見過資料上的表現是否提升。
7.重複步驟 4–6，直到達到指定的 Epoch 數量或提早停止（early stopping）

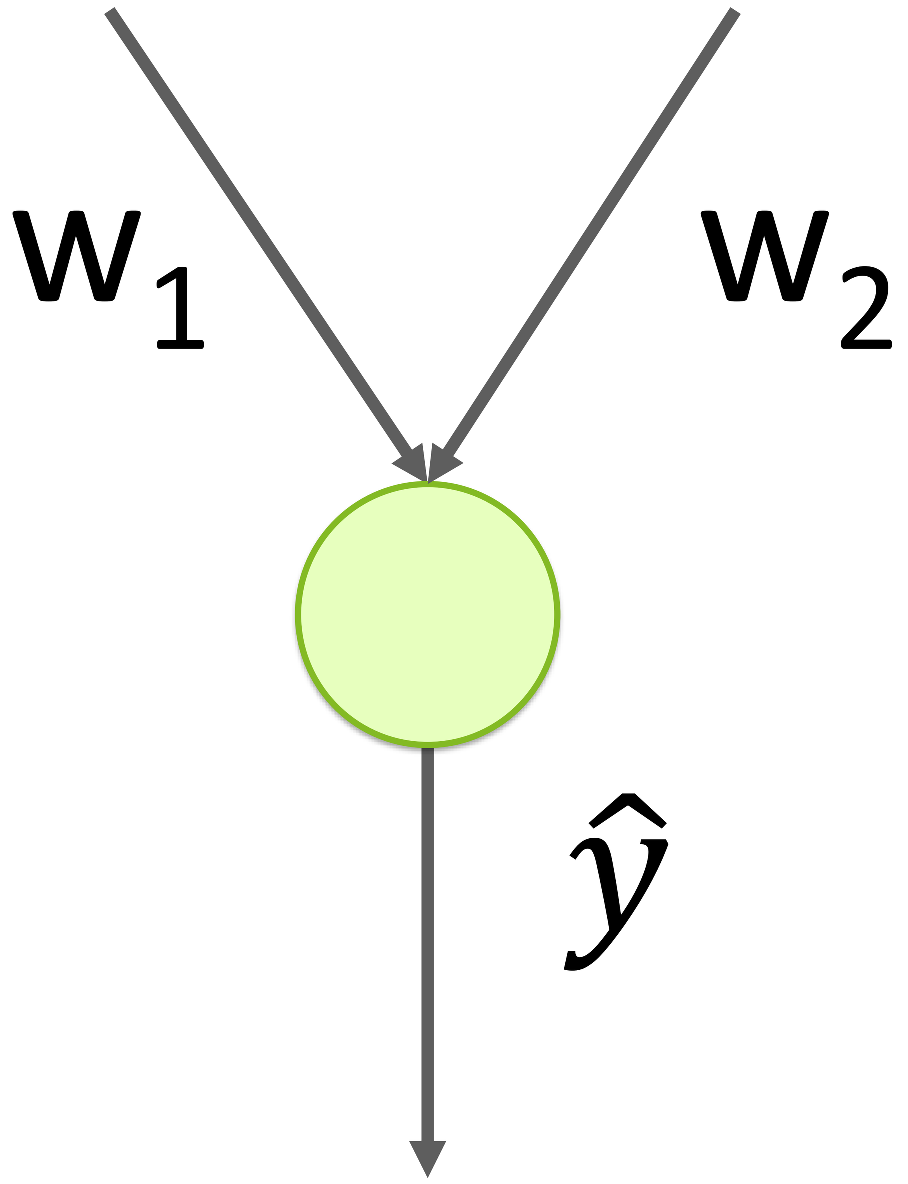
優化器(Optimizers)



- Adam
- Adagrad
- RMSprop
- SGD

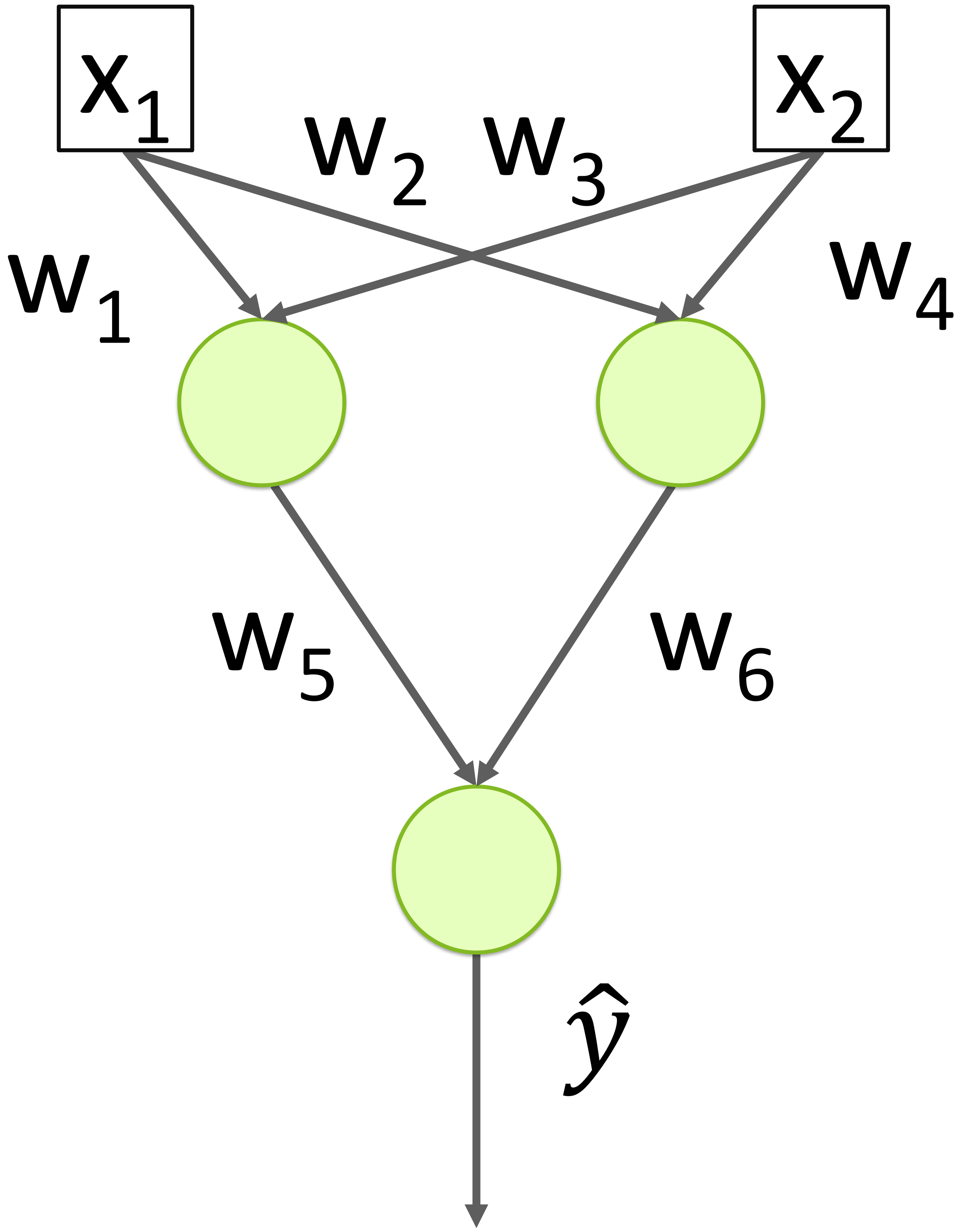
從神經元到網路(From Neuron to Network)

建立網路(Building a Network)



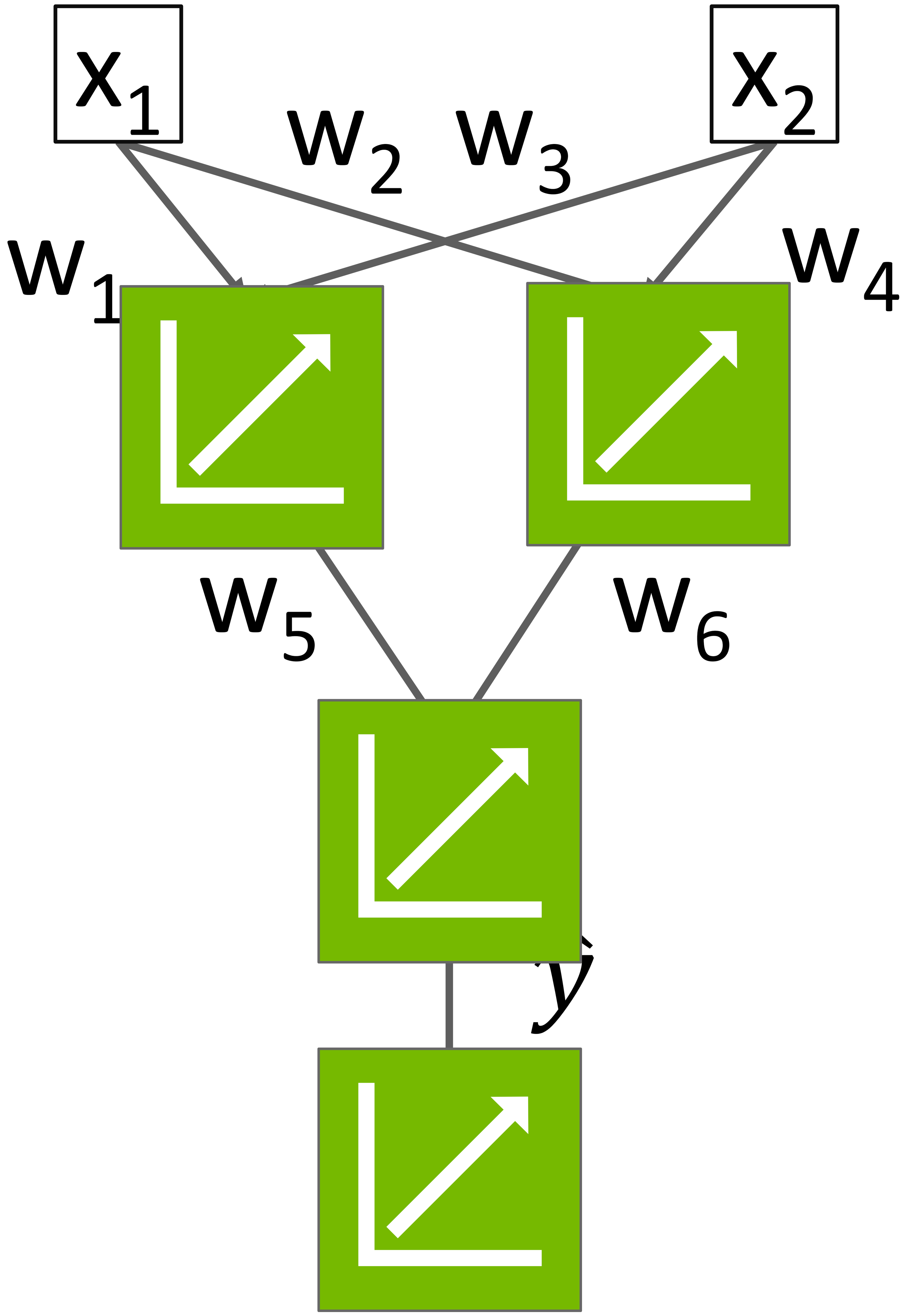
- 擴展到更多輸入資料(Scales to more inputs)

建立網路(Building a Network)



- 擴展到更多輸入資料(Scales to more inputs)
- 可以鏈接神經元(Can chain neurons)

建立網路(Building a Network)



- 擴展到更多輸入(Scales to more inputs)
- 可以鏈接神經元(Can chain neurons)
- 如果所有回歸(regressions)都是線性的，那麼輸出也將是線性回歸

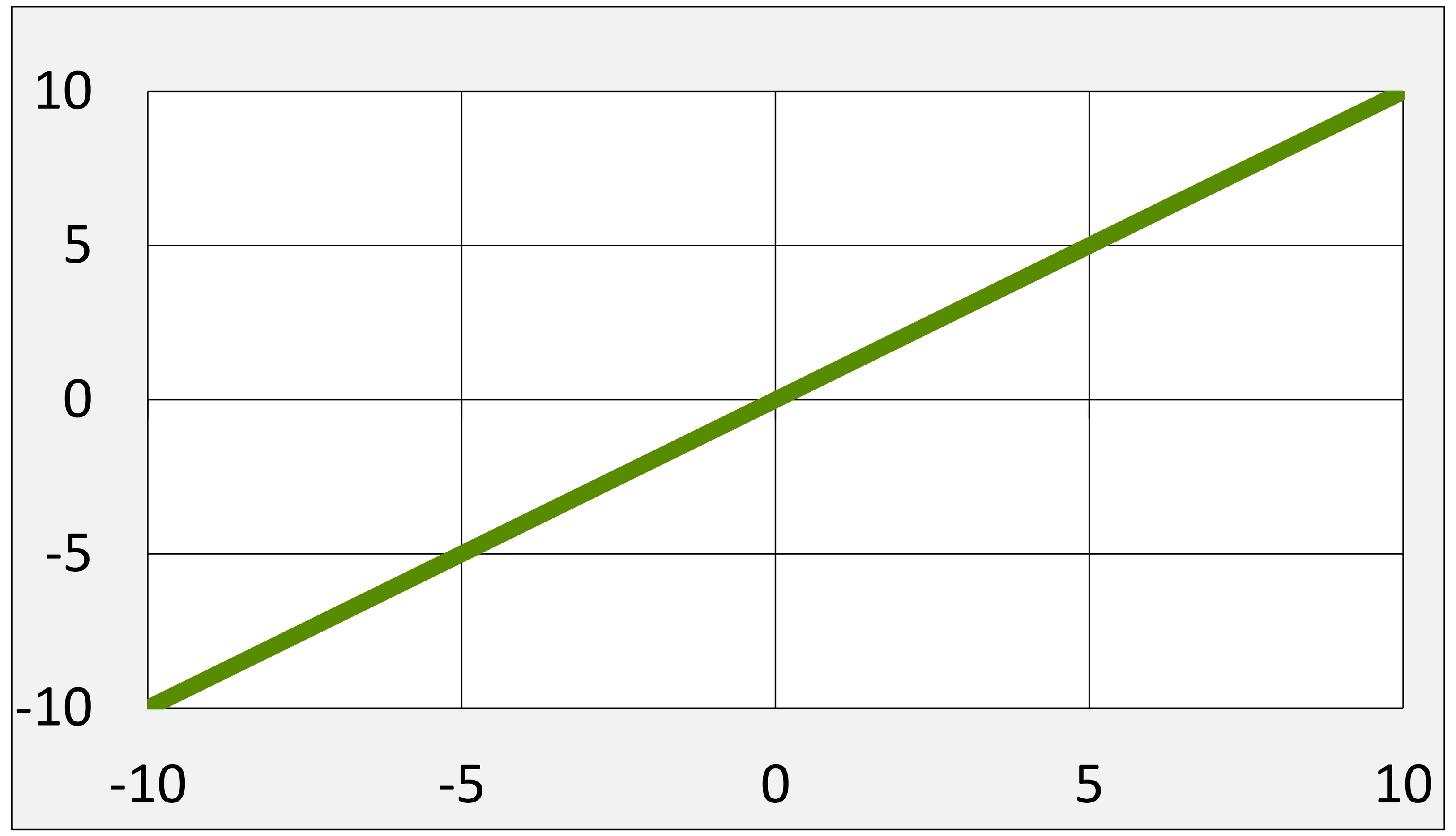
活化函式(Activation Functions)

活化函式(Activation Functions)

Linear線性(Linear)

$$\hat{y} = wx + b$$

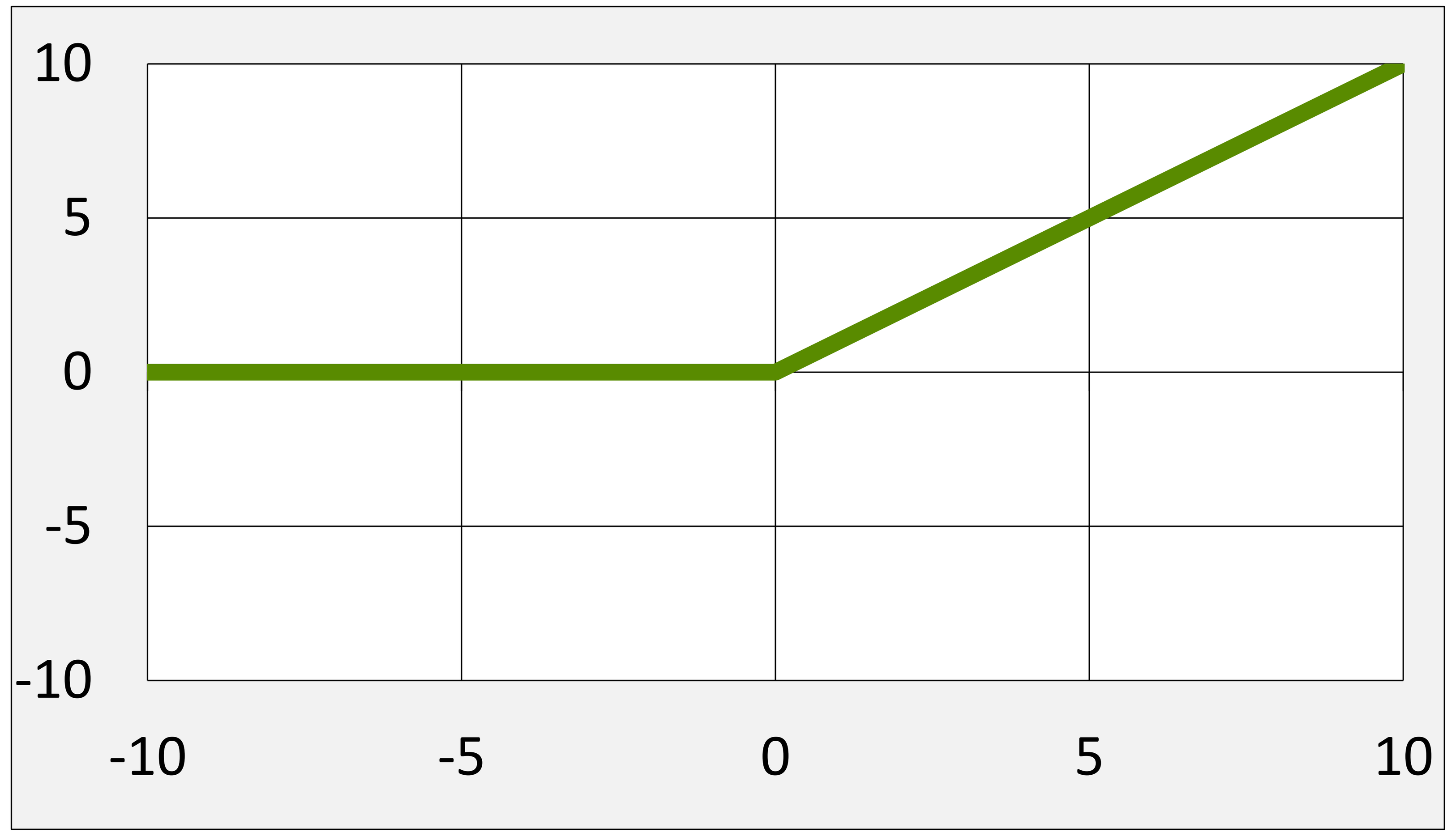
```
# Multiply each input
# with a weight (w) and
# add intercept (b)
y_hat = wx+b
```



ReLU

$$\hat{y} = \begin{cases} wx + b \; if \; wx + b > 0 \\ 0 \;\; otherwise \end{cases}$$

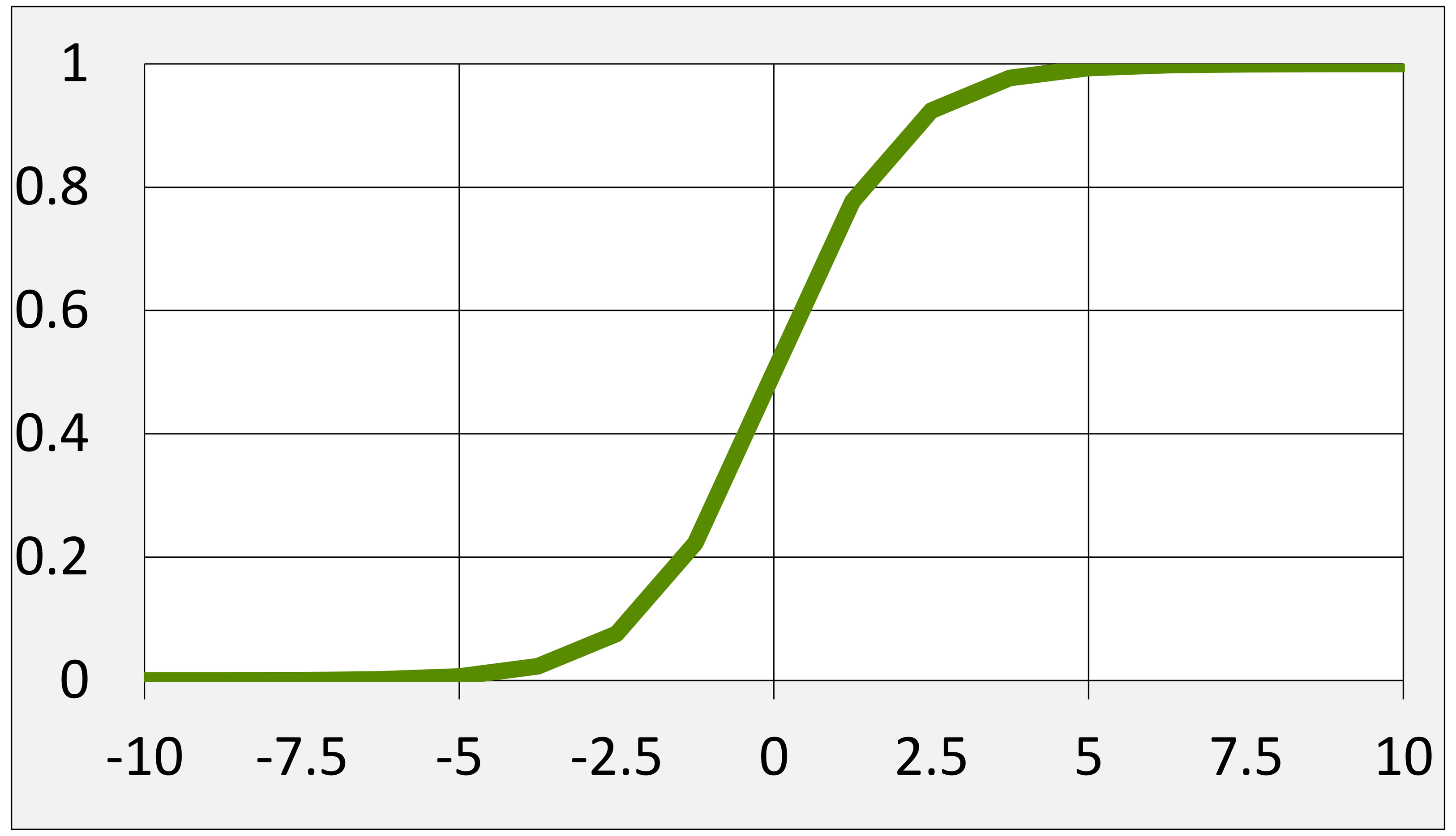
```
# Only return result
# if total is positive
linear = wx+b
y_hat = linear * (linear > 0)
```



Sigmoid

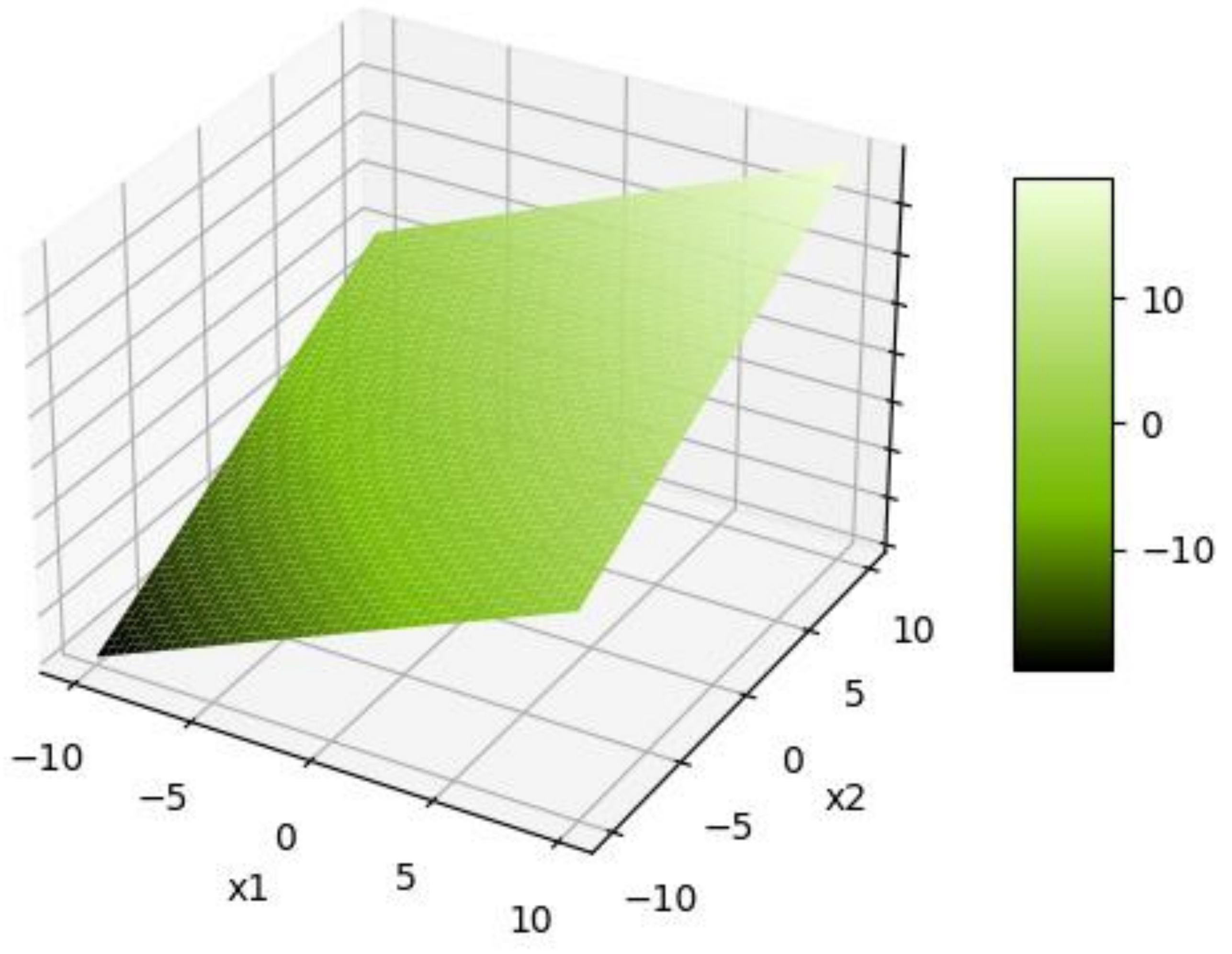
$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

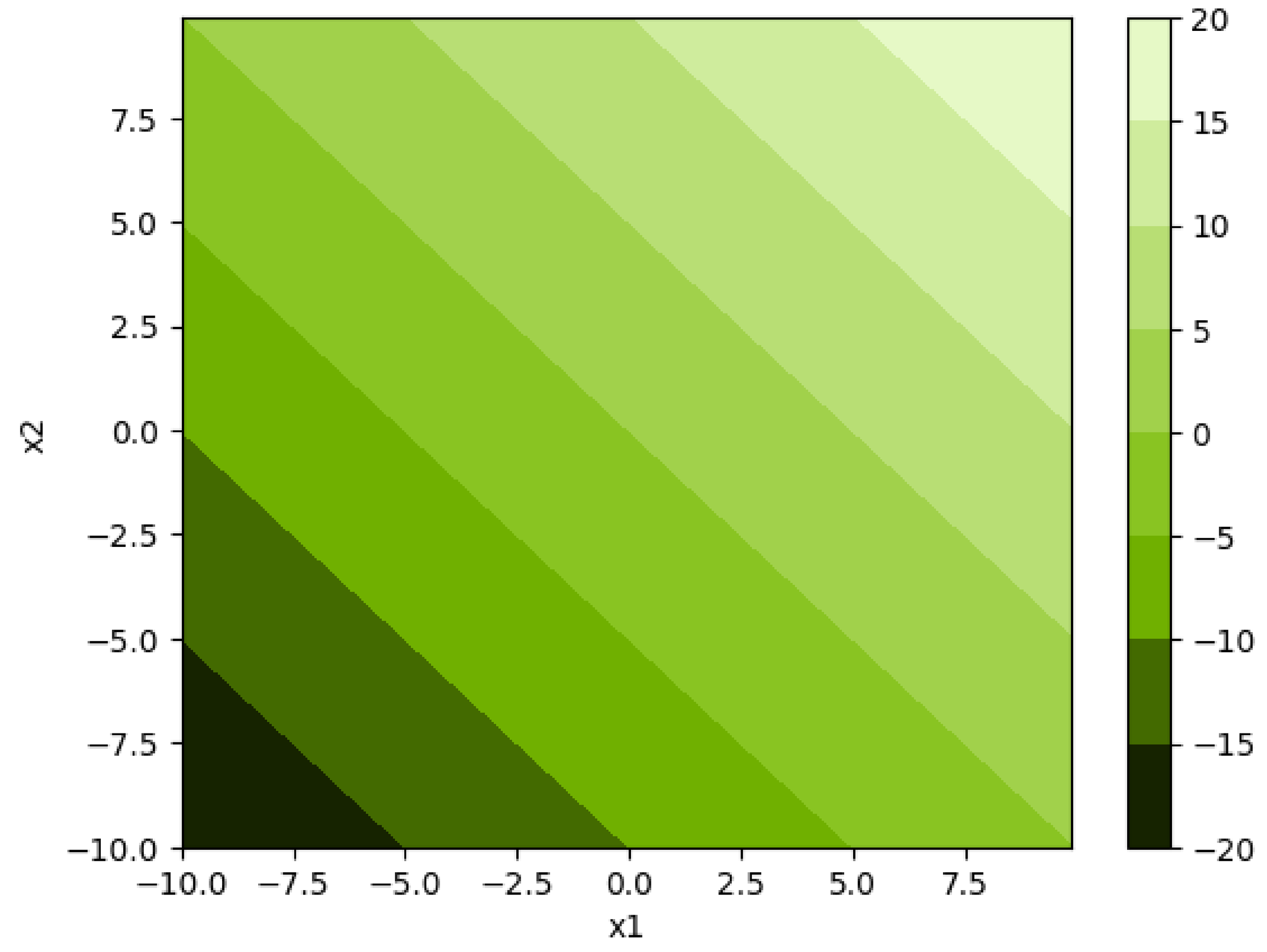
```
# Start with line
linear = wx + b
# Warp to - inf to 0
inf_to_zero = np.exp(-1 * linear)
# Squish to -1 to 1
y_hat = 1 / (1 + inf_to_zero)
```



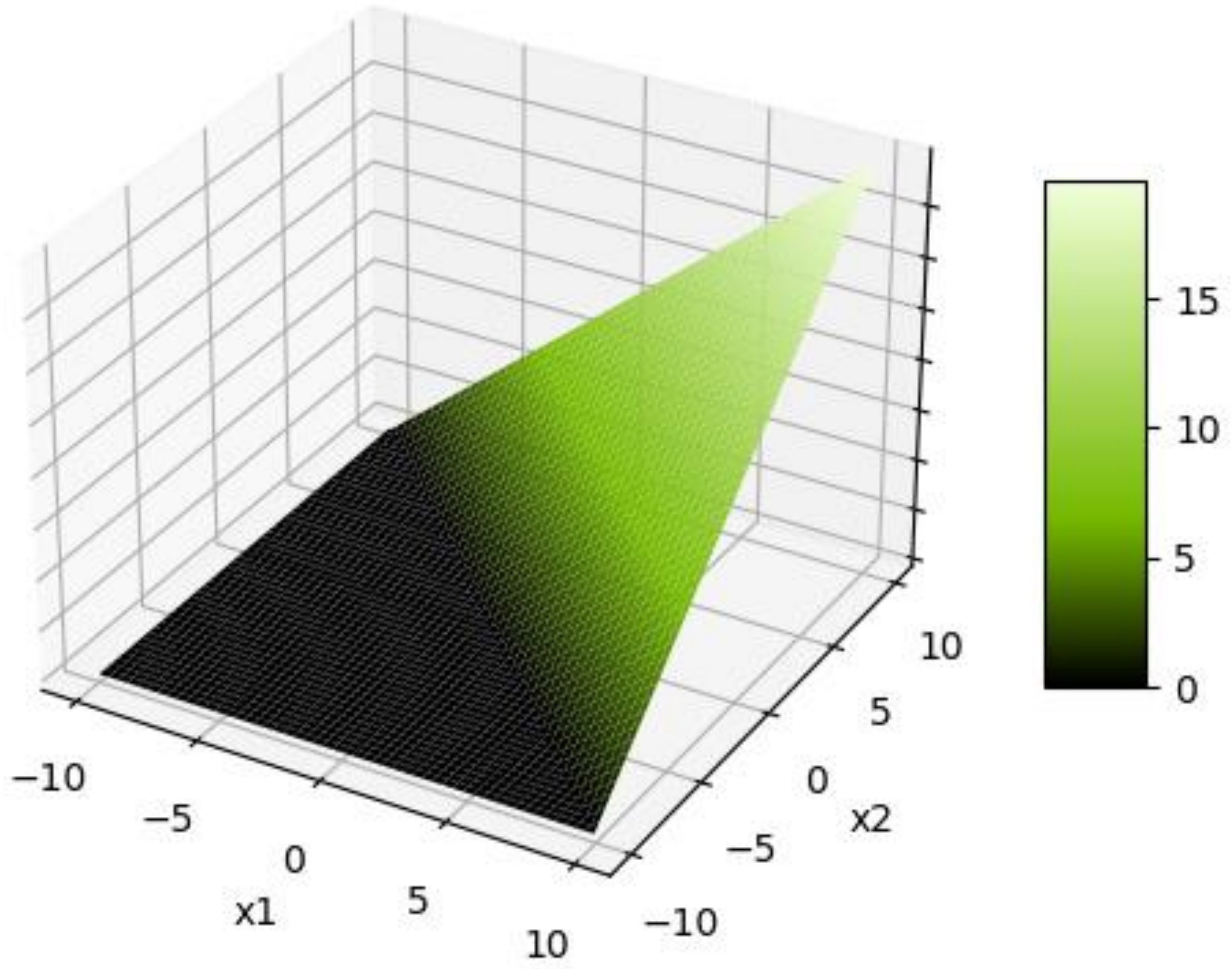
活化函式(Activation Function)

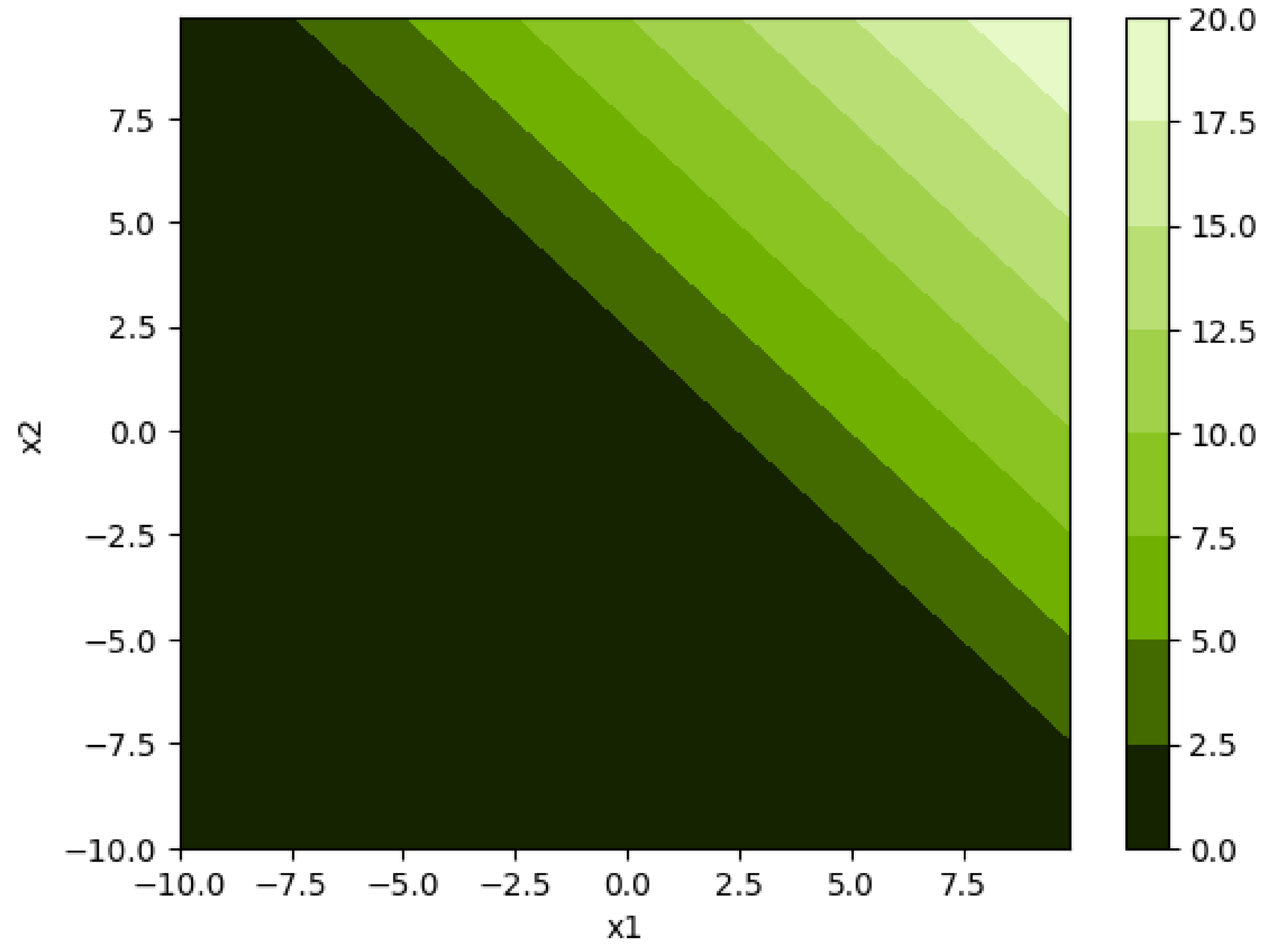
Linear線性(Linear)



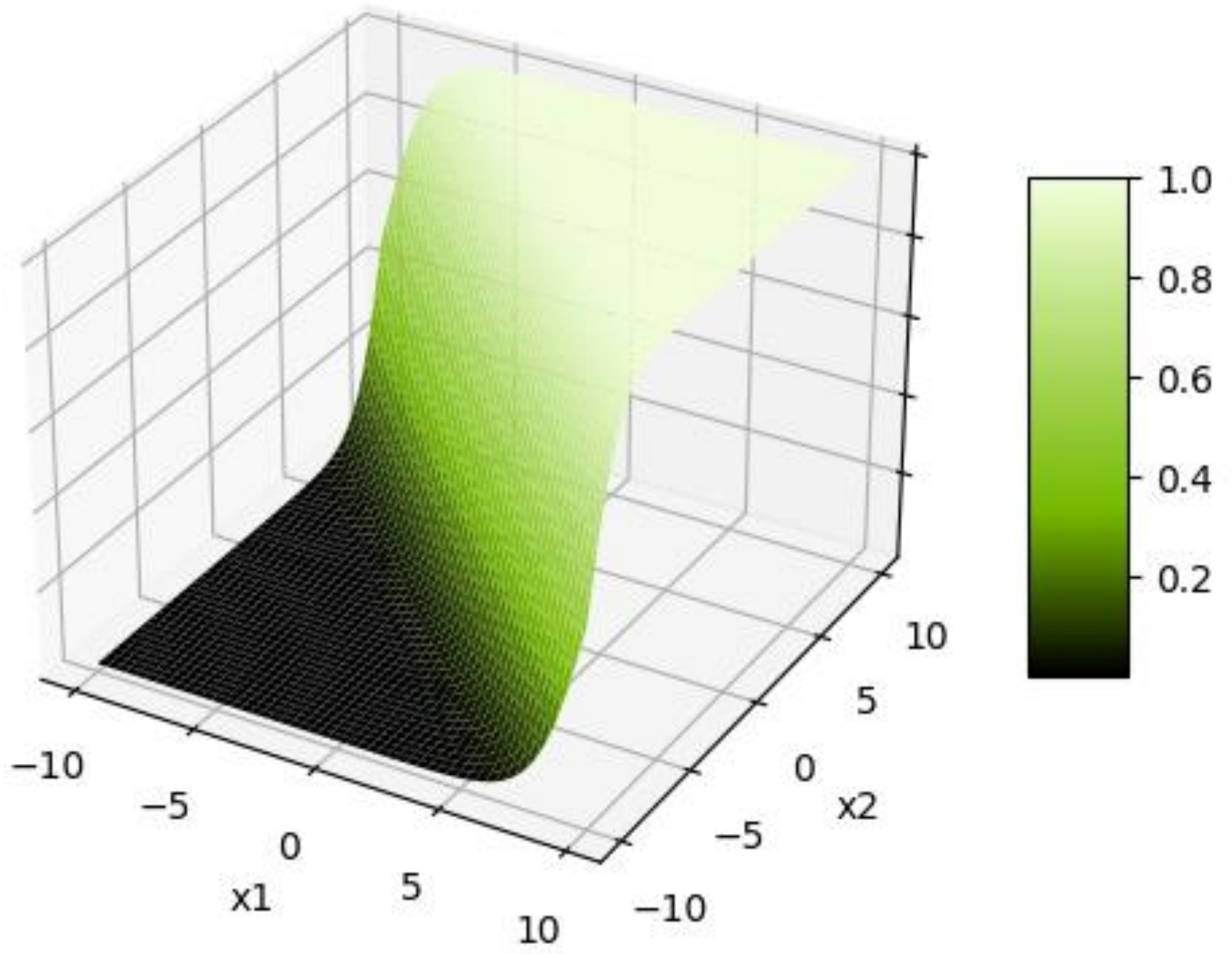


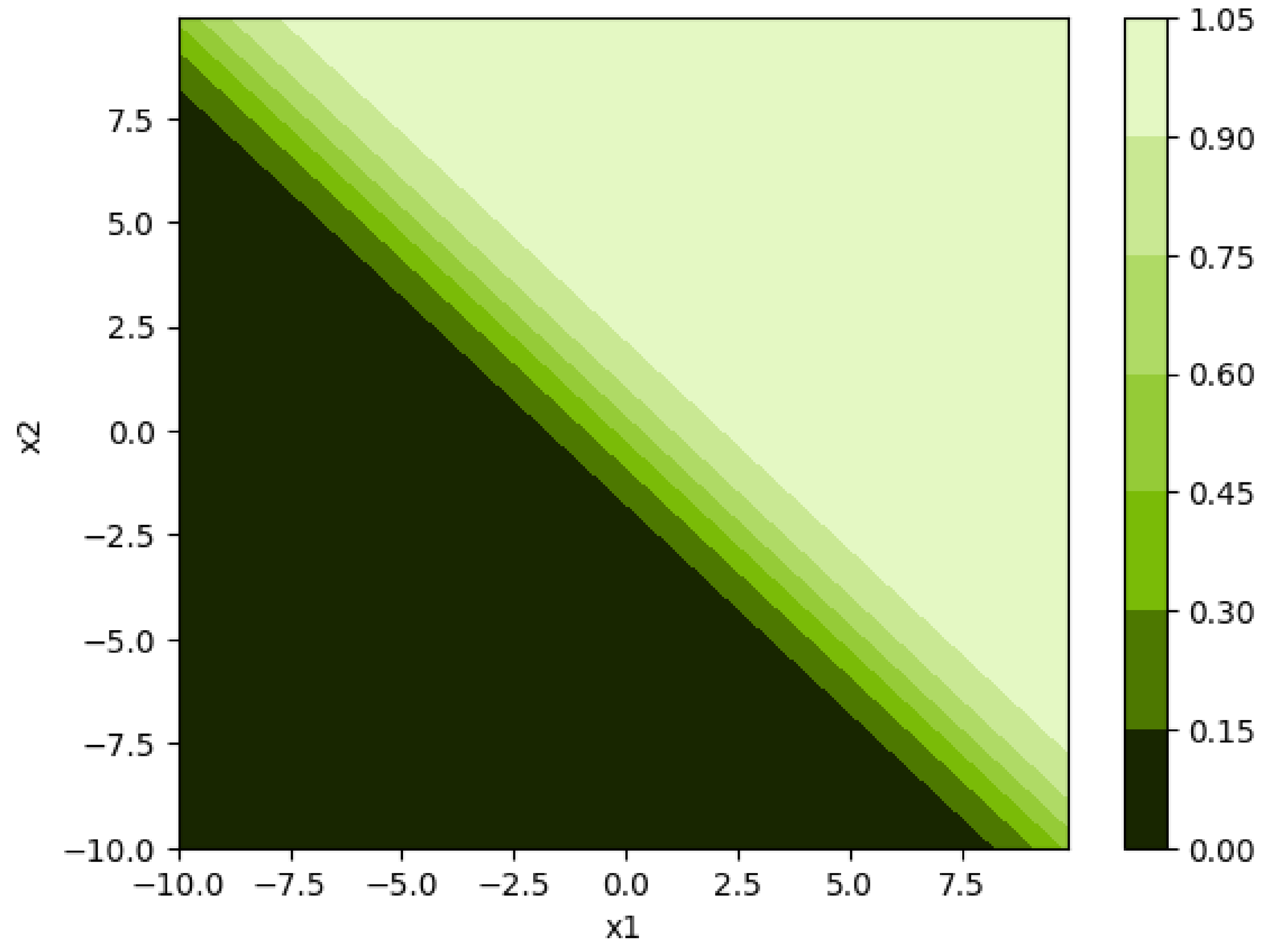
ReLUReLU





Sigmoid





活化函式(Activation Function)

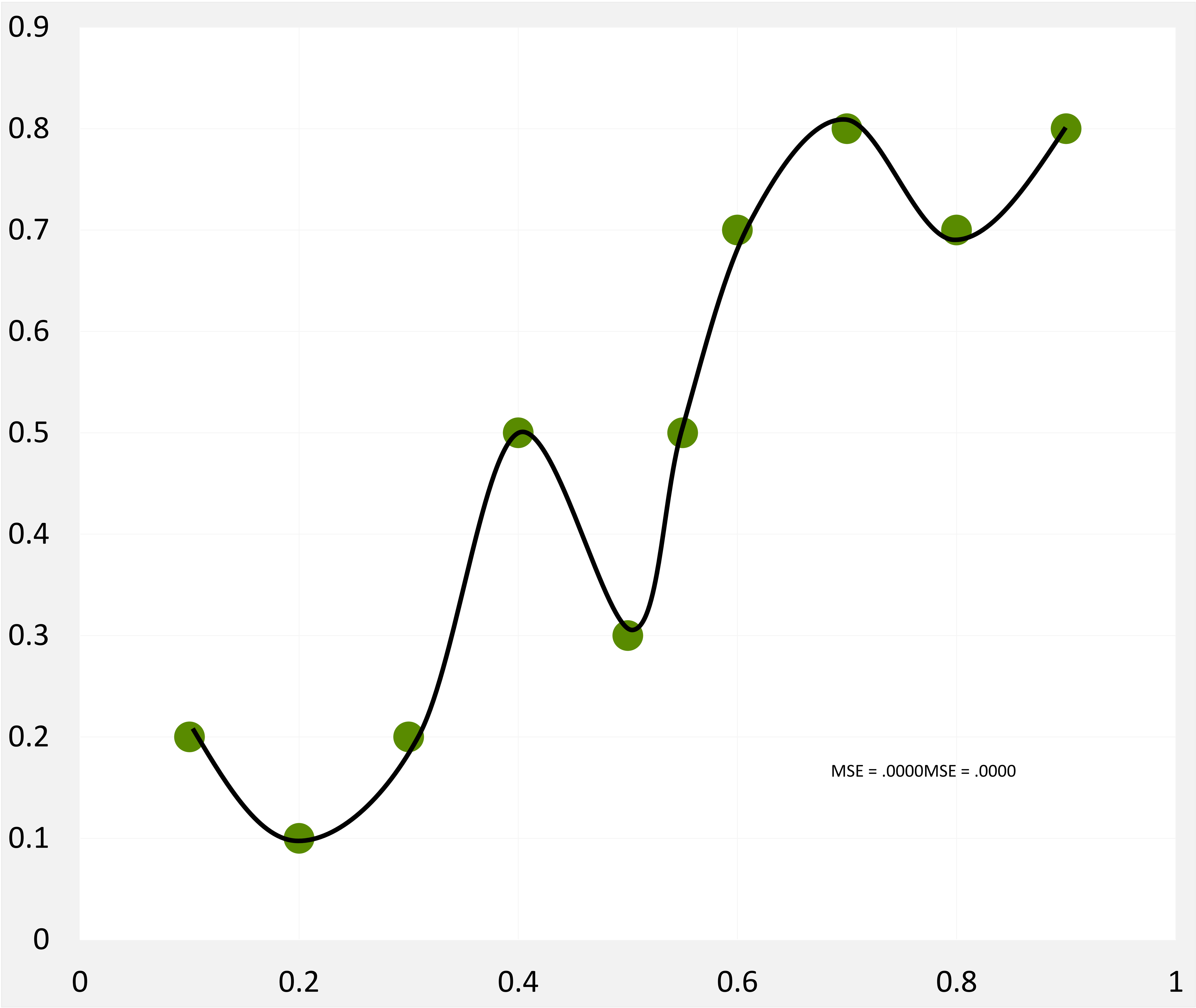
過度擬合(overfitting)

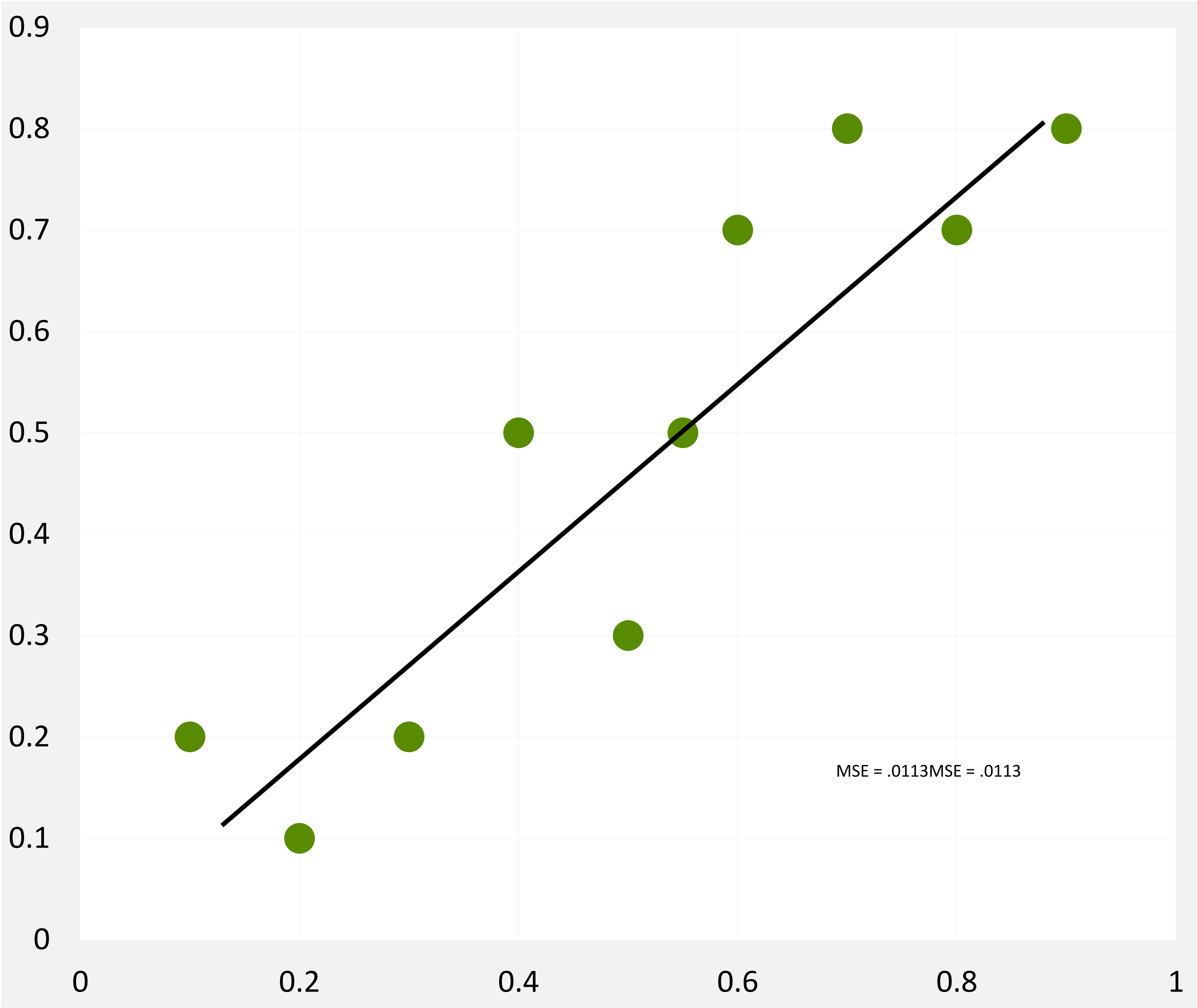
過度擬合(overfitting)

為什麼不使用超大型神經網路？

過度擬合(overfitting)

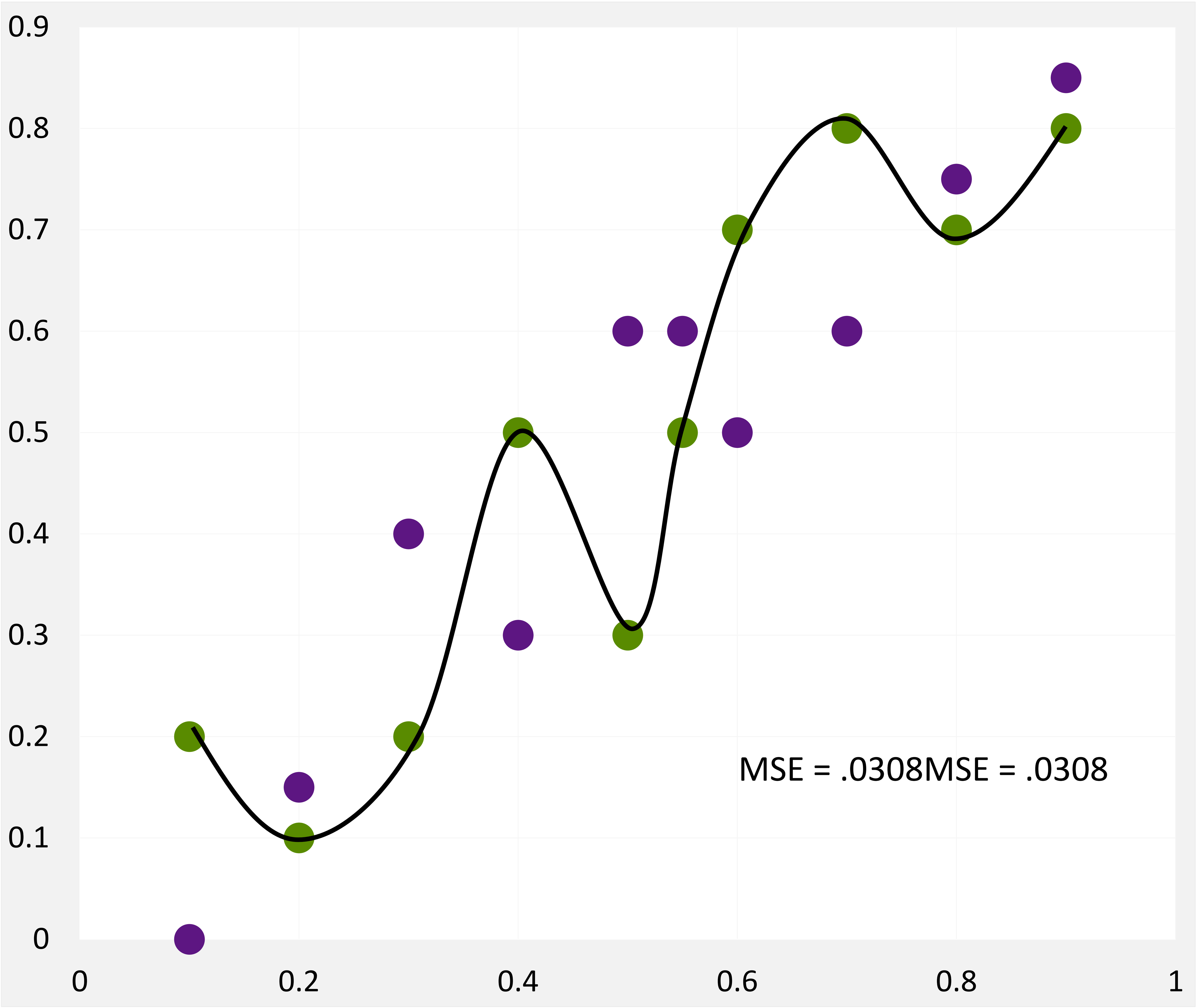
哪個趨勢線更好？

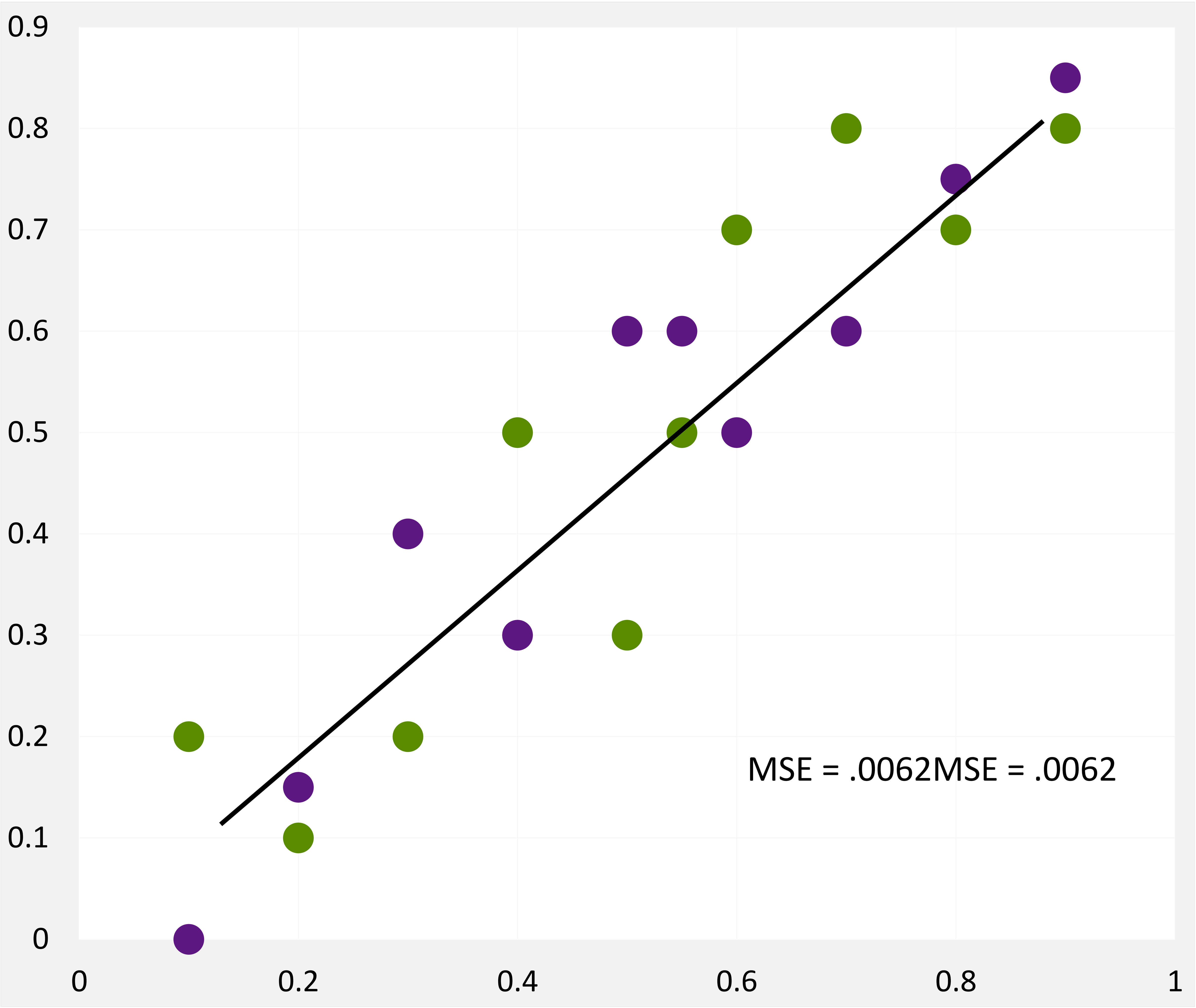




過度擬合(overfitting)

哪個趨勢線更好？





訓練資料與驗證資料(Training vs Validation Data)

避免記憶化

訓練資料

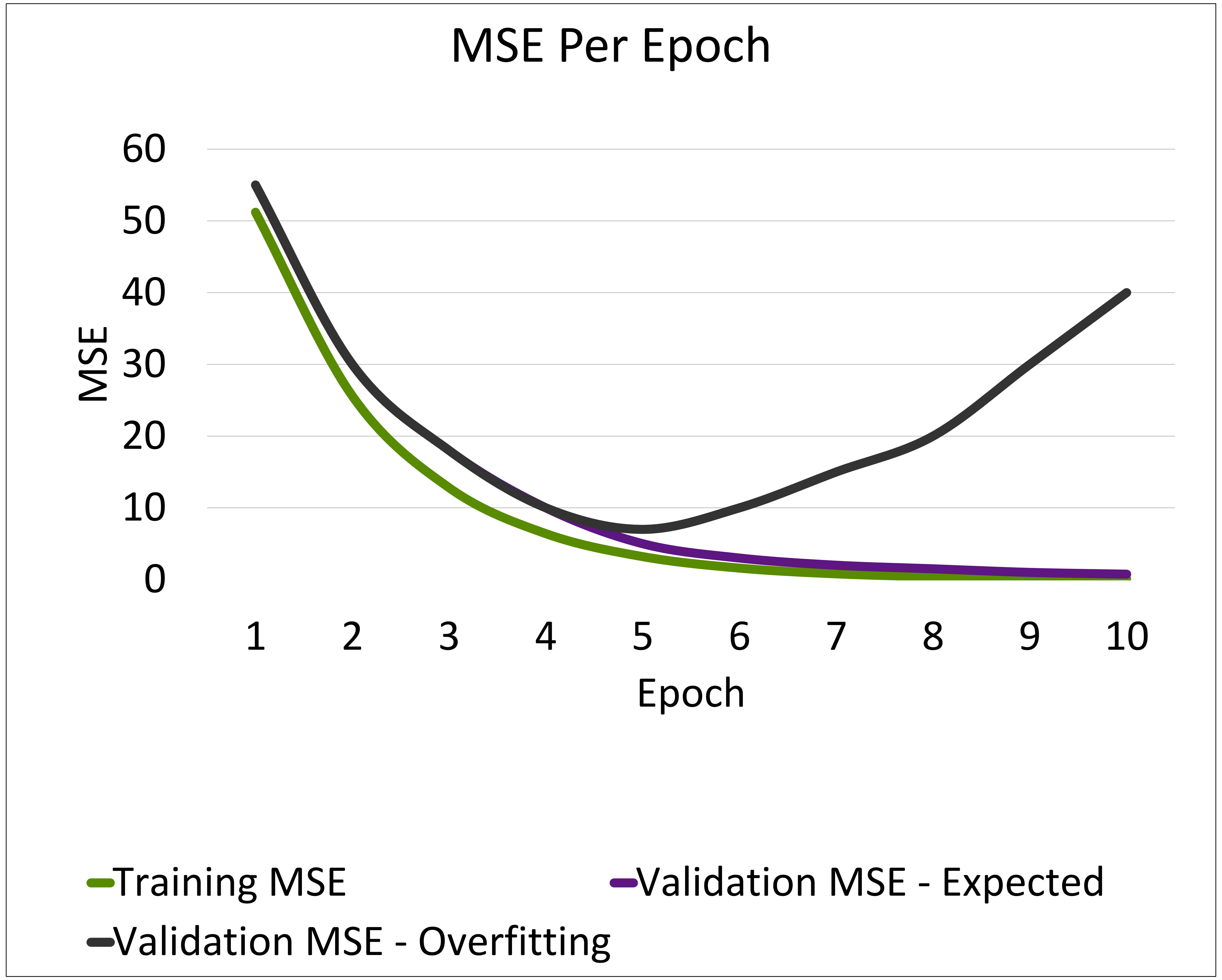
- 模型學習的核心資料集

驗證(Validation)資料

- 用於檢驗模型是否真正理解（能夠泛化）的新資料

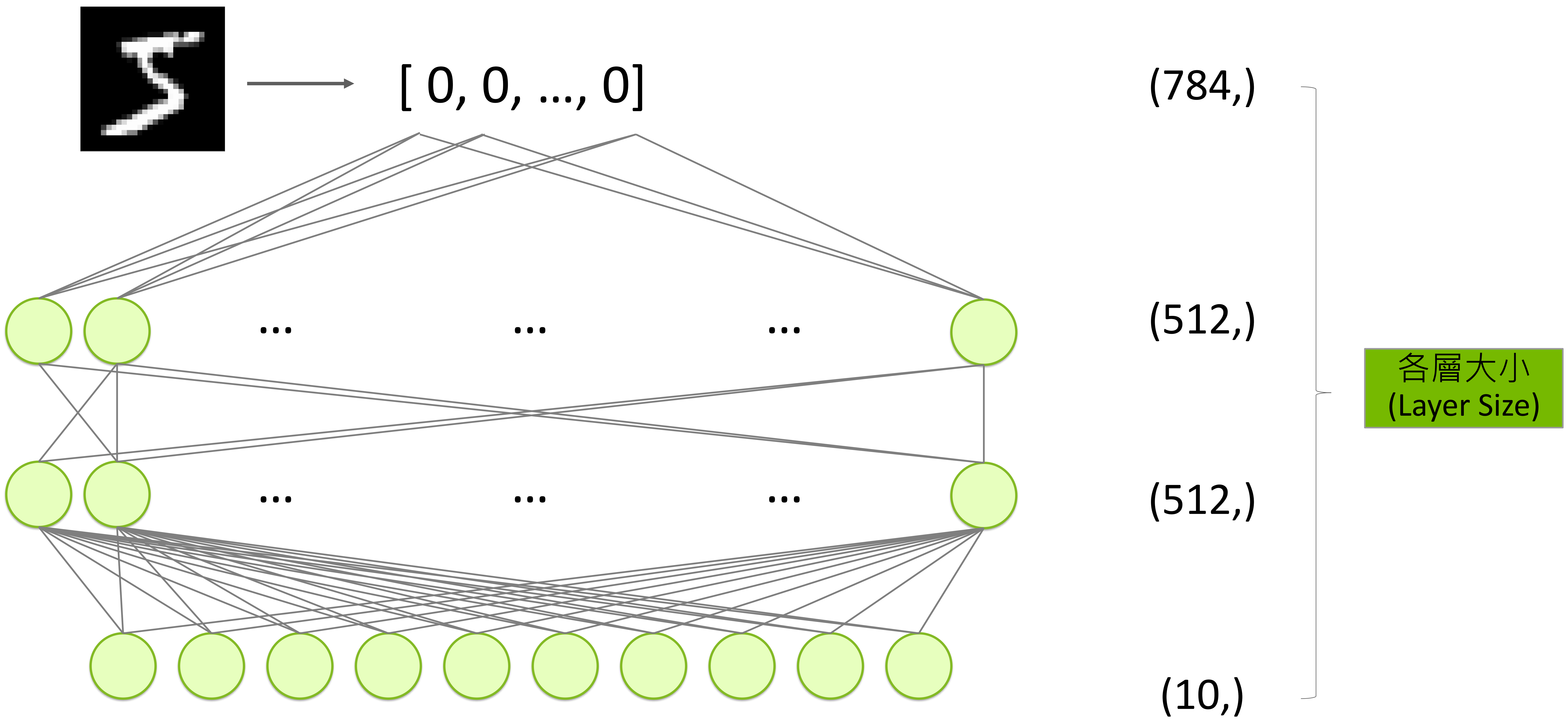
過度擬合(Overfitting)

- 當模型在訓練資料上表現良好，但在驗證(Validation)資料上表現不佳時（顯示出記憶而非學習的跡象）
- 理想情況下，兩個資料集的準確度(accuracy)和損失函式(loss function)應該相似



從迴歸到分類(From Regression to Classification)

MNIST模型



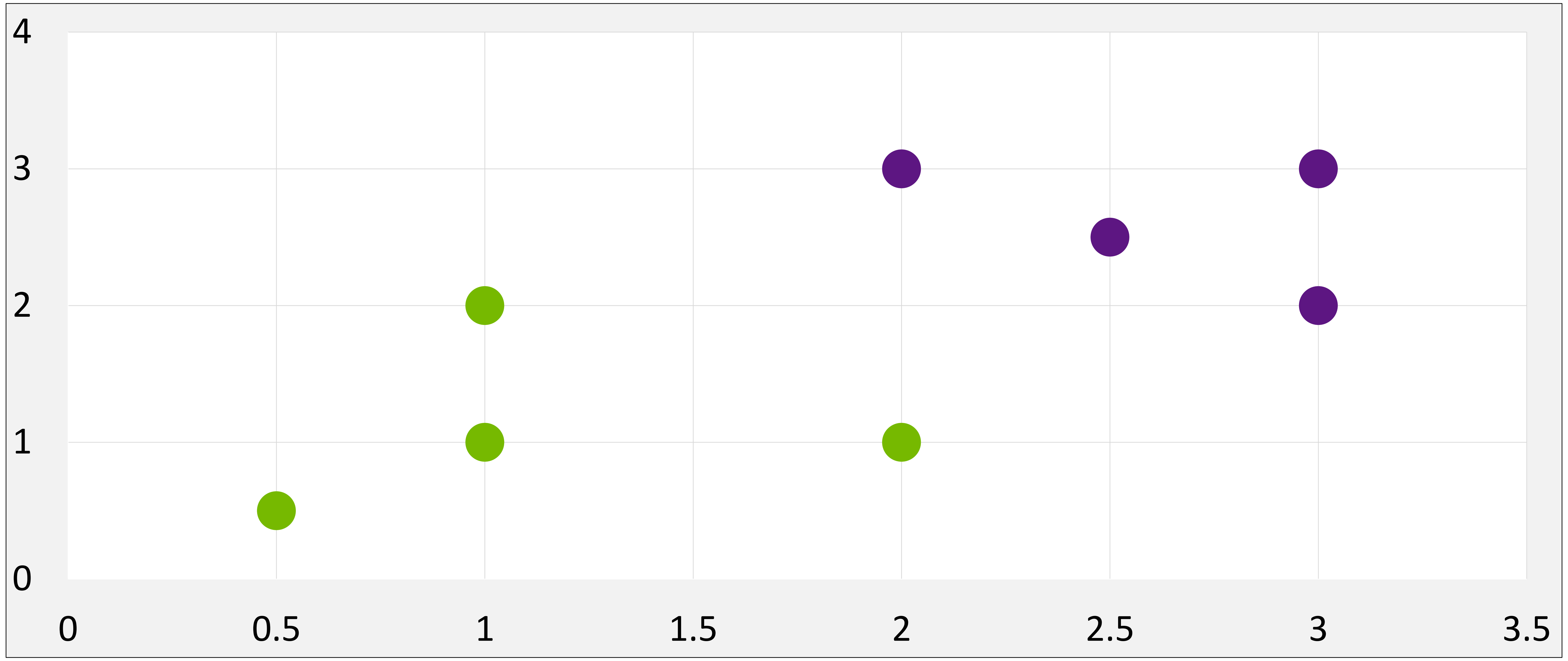
MNIST模型

預測過程

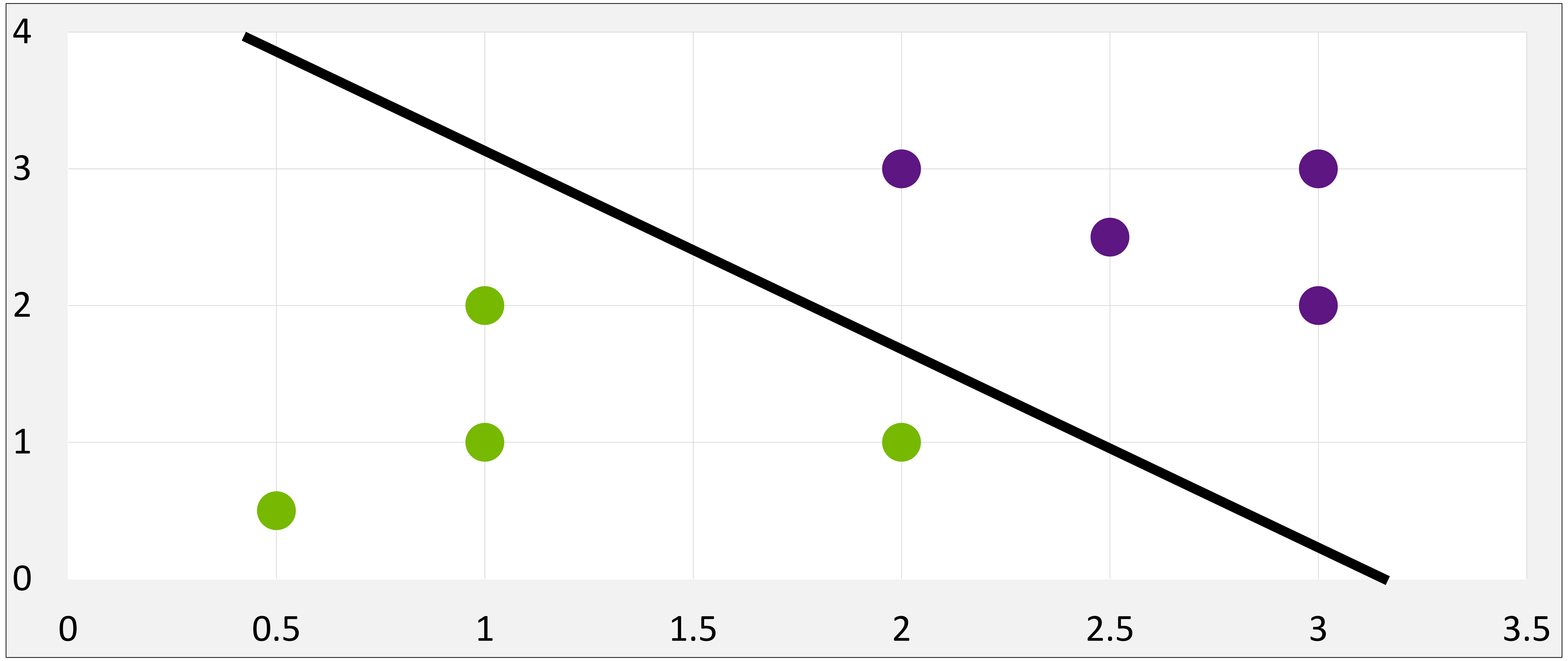
MNIST模型

訓練過程

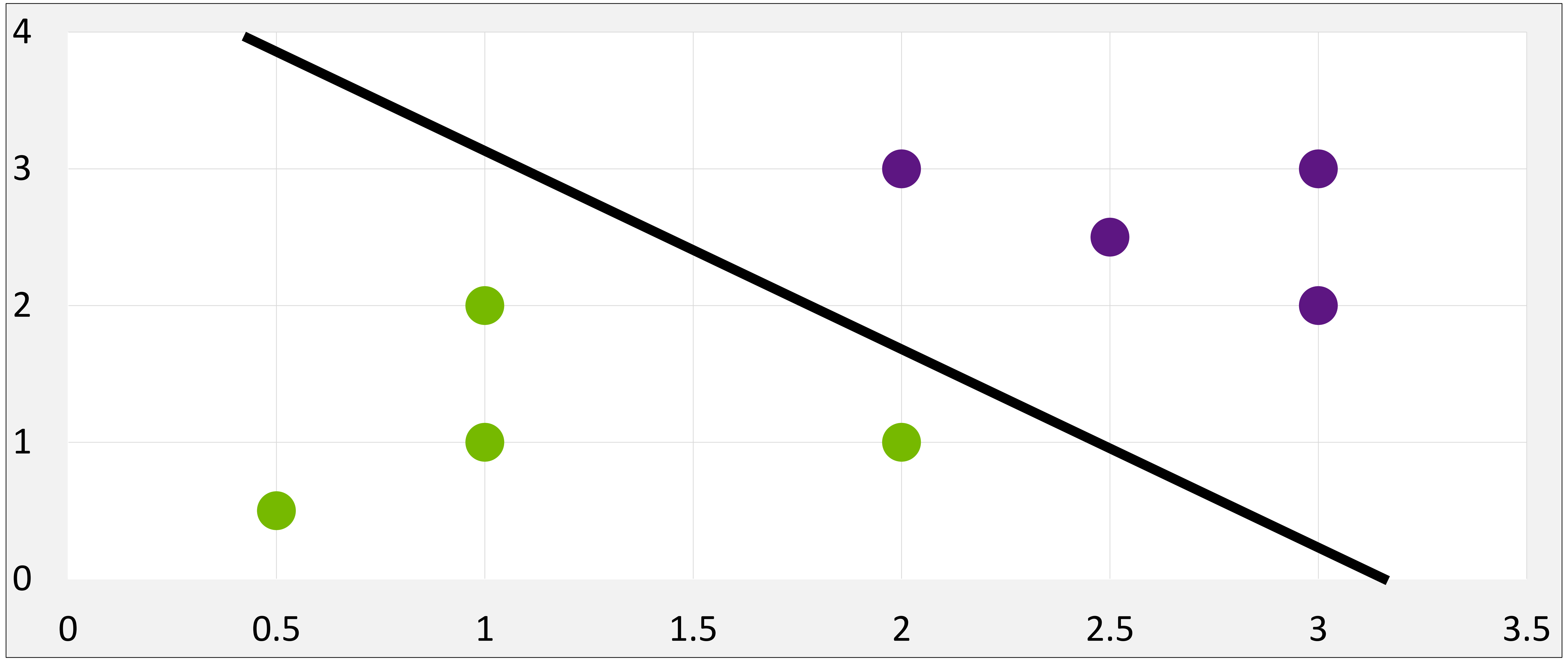
機率的RMSE？

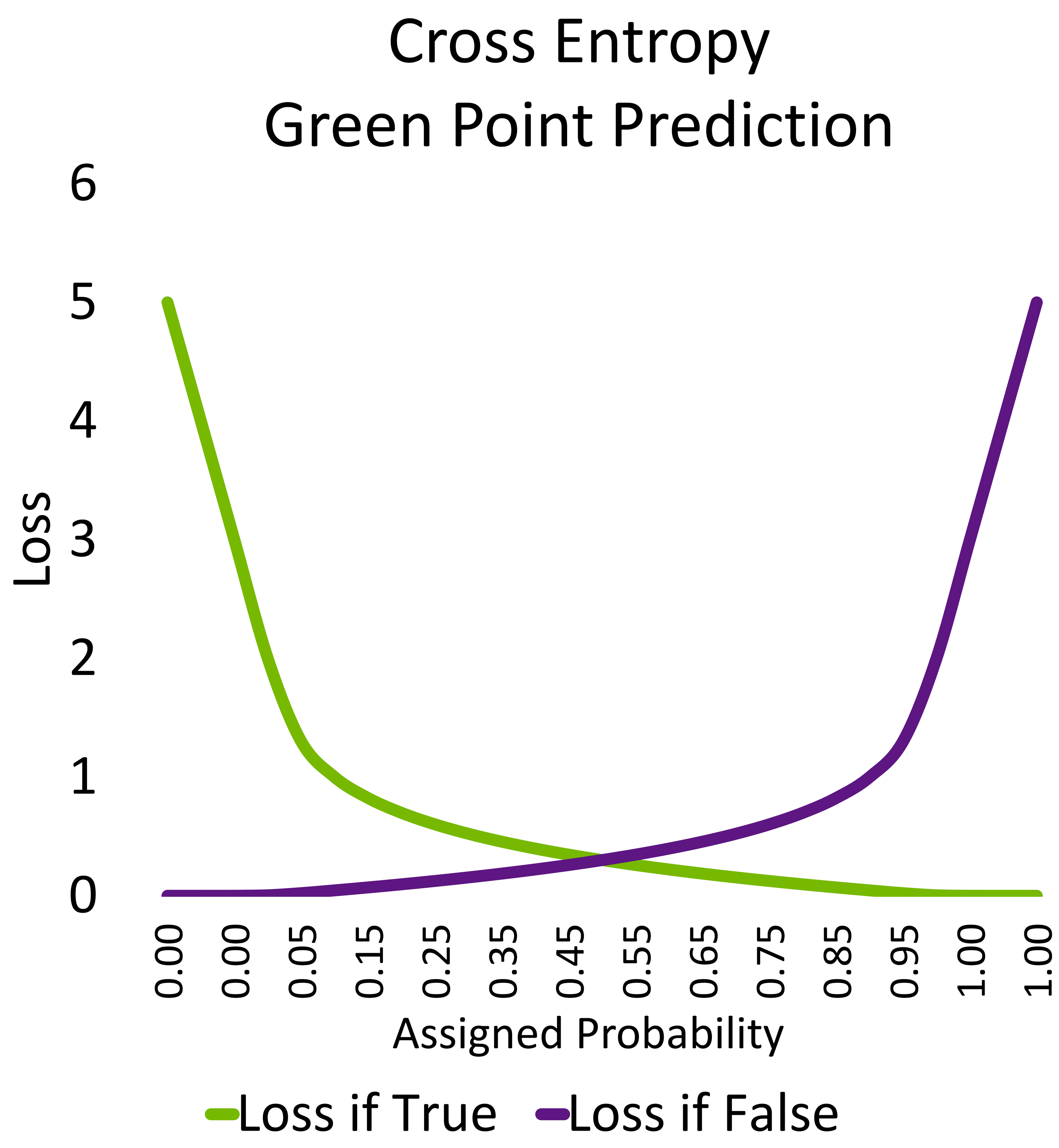


機率的RMSE？

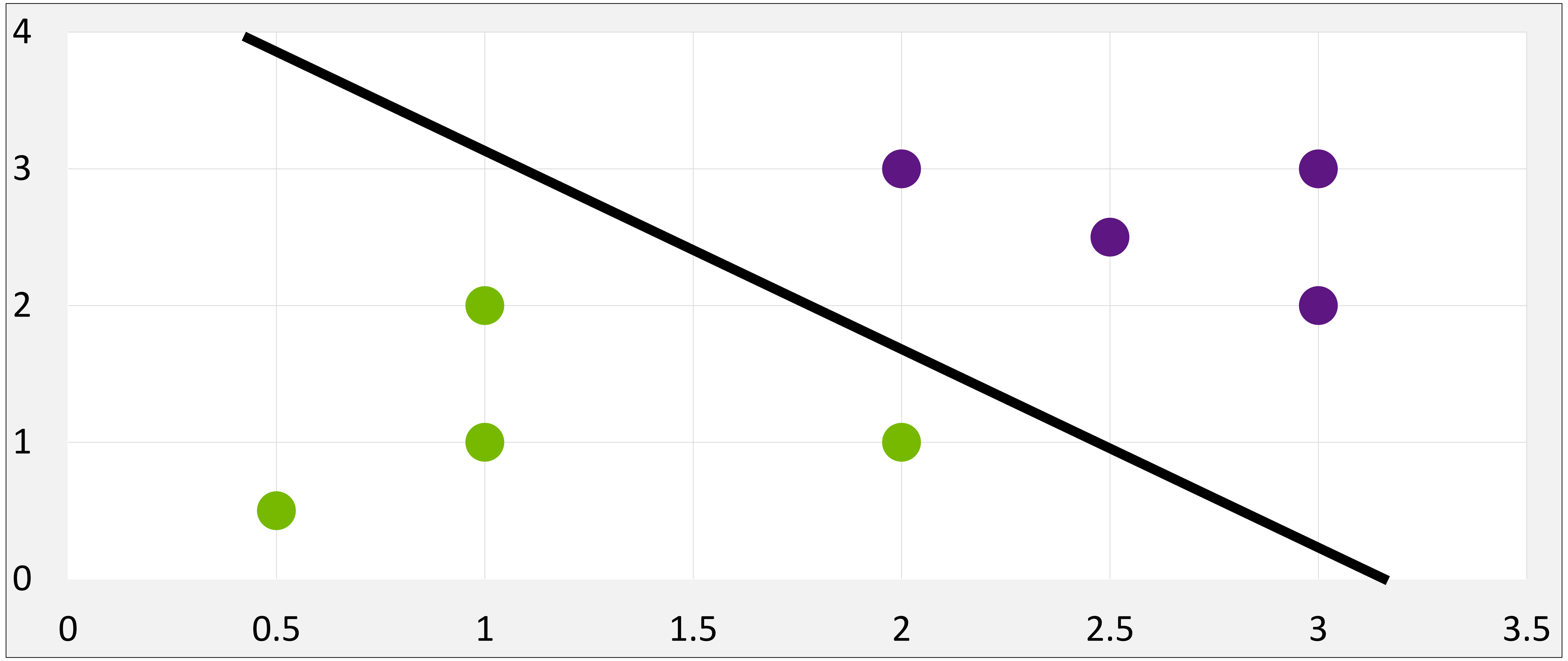


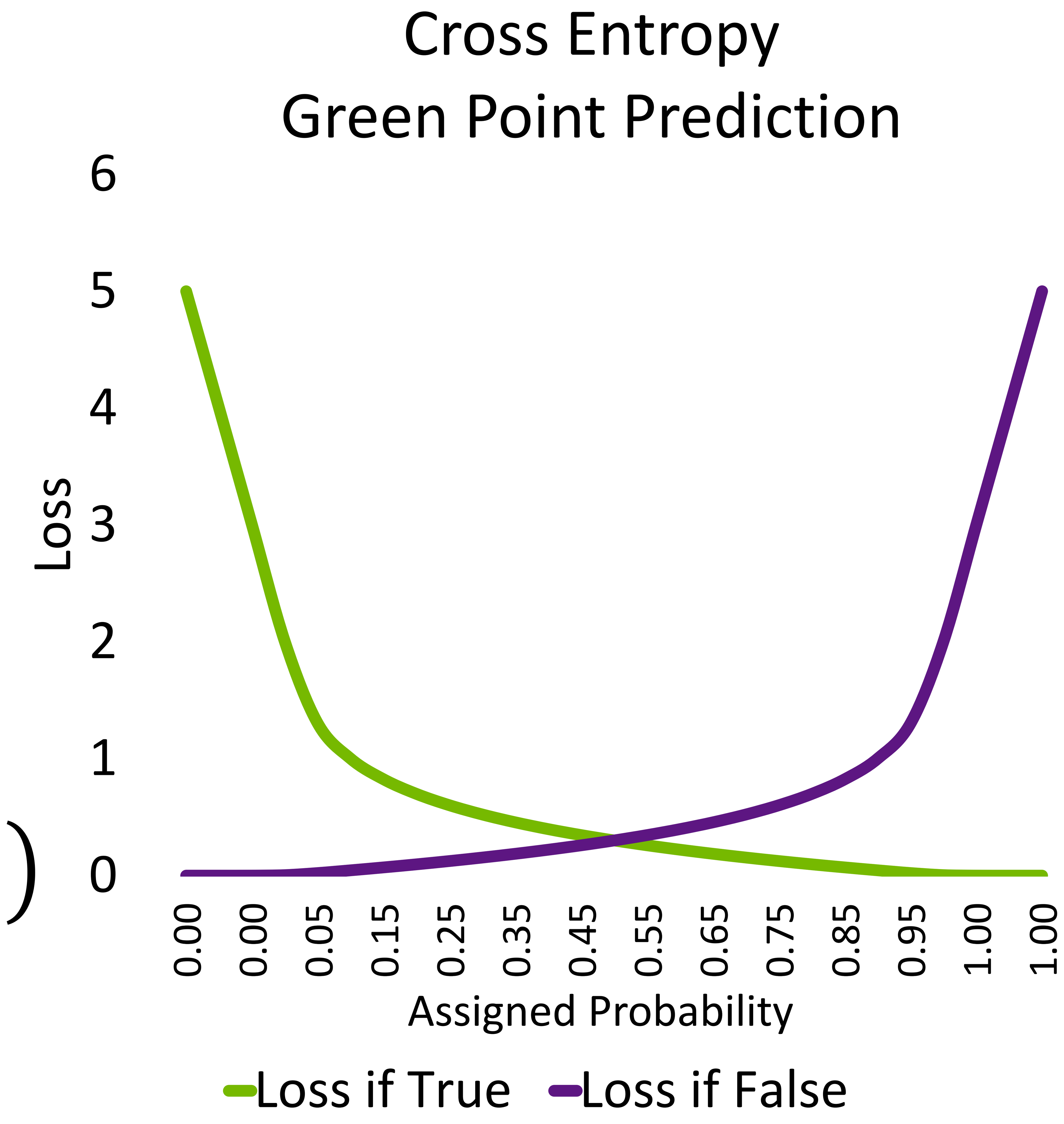
交叉熵(Cross Entropy)





交叉熵(Cross Entropy)





$$Loss = -\Big(\big(t(x) \cdot \log(p(x)\big) + \big(1 - t(x)\big) \cdot \log(1 - p(x)\big)\Big)$$

$$t(x) = target\ (0\ if\ False, 1\ if\ True)$$

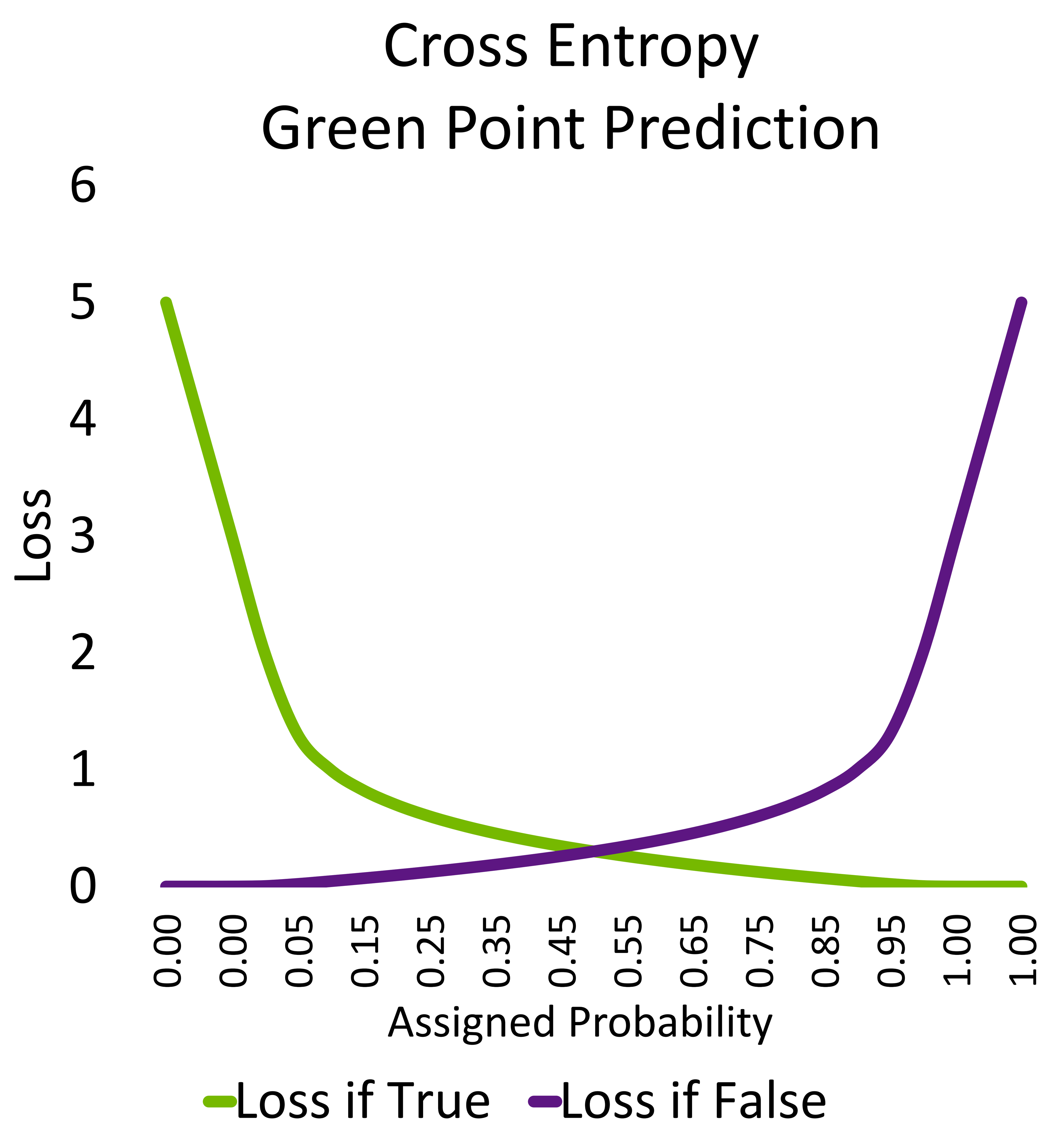
$$p(x) = probability\ prediction\ of\ point\ x$$

◆ 情況 1：真實值是正類 ($t(x) = 1$)

Green

$$\text{Loss} = -\log(p(x))$$

- 越接近 1 (正確) ，loss 越小
- 越接近 0 (錯很離譜) ，loss 越大 (趨近於無限大)



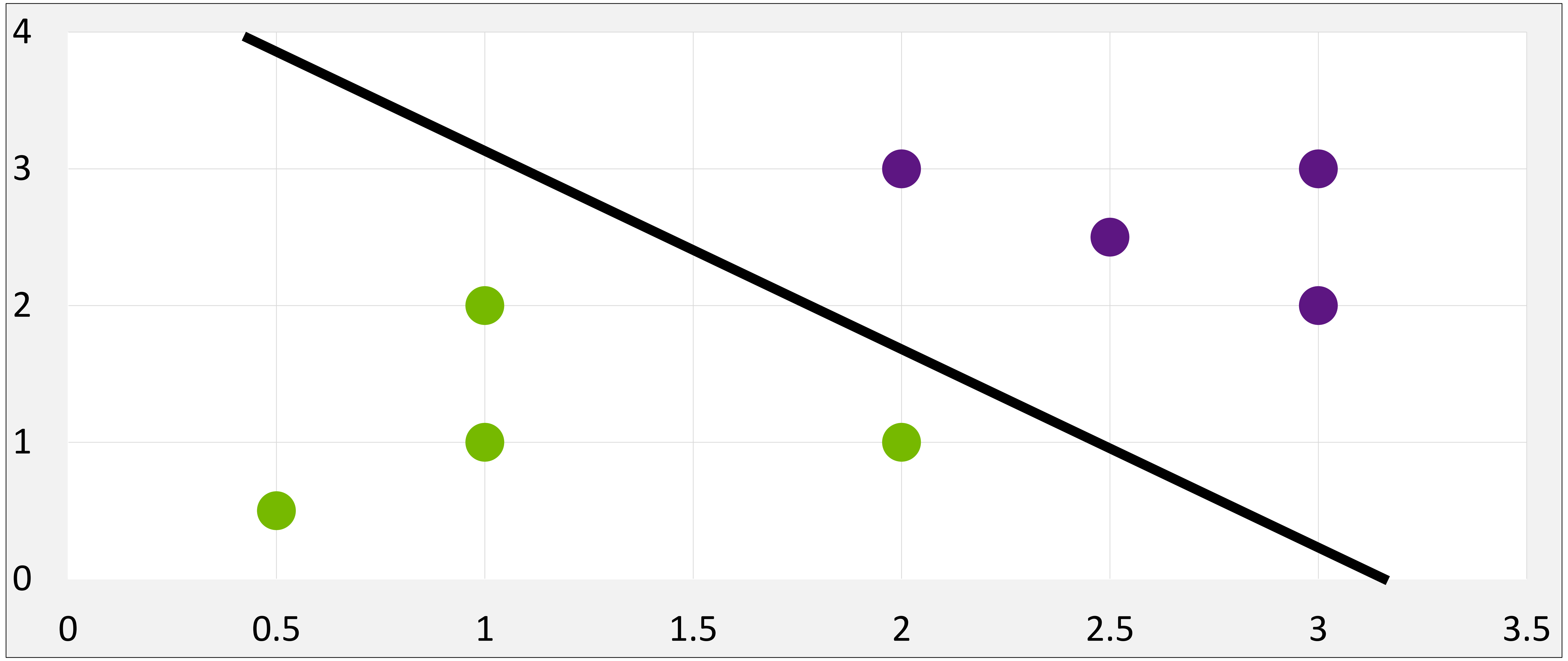
◆ 情況 2：真實值是負類 ($t(x) = 0$)

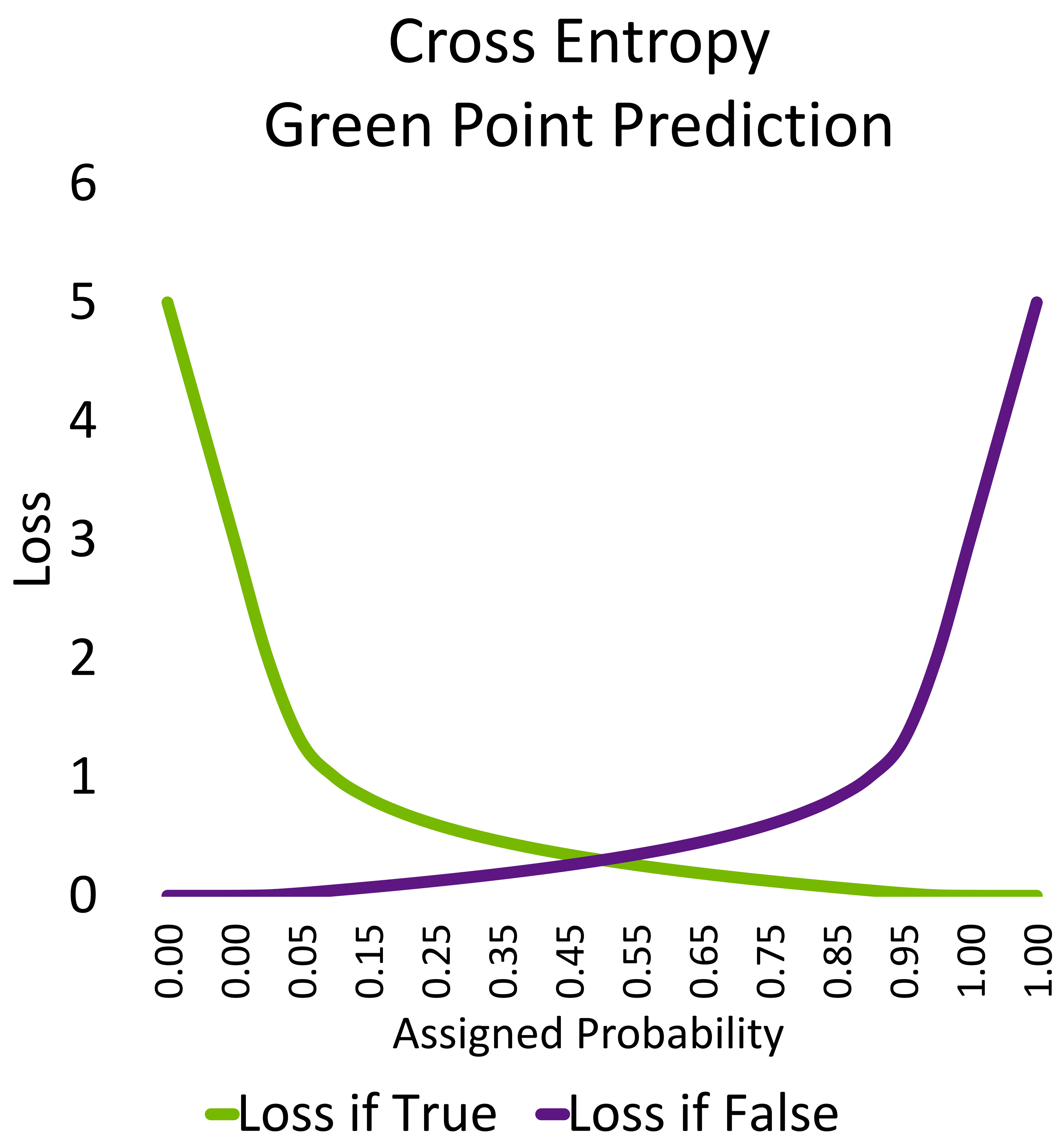
Purple

$$\text{Loss} = -\log(1 - p(x))$$

- 越接近 0 (正確) ，loss 越小
- 越接近 1 (錯很離譜) ，loss 越大 (趨近於無限大)

交叉熵(Cross Entropy)



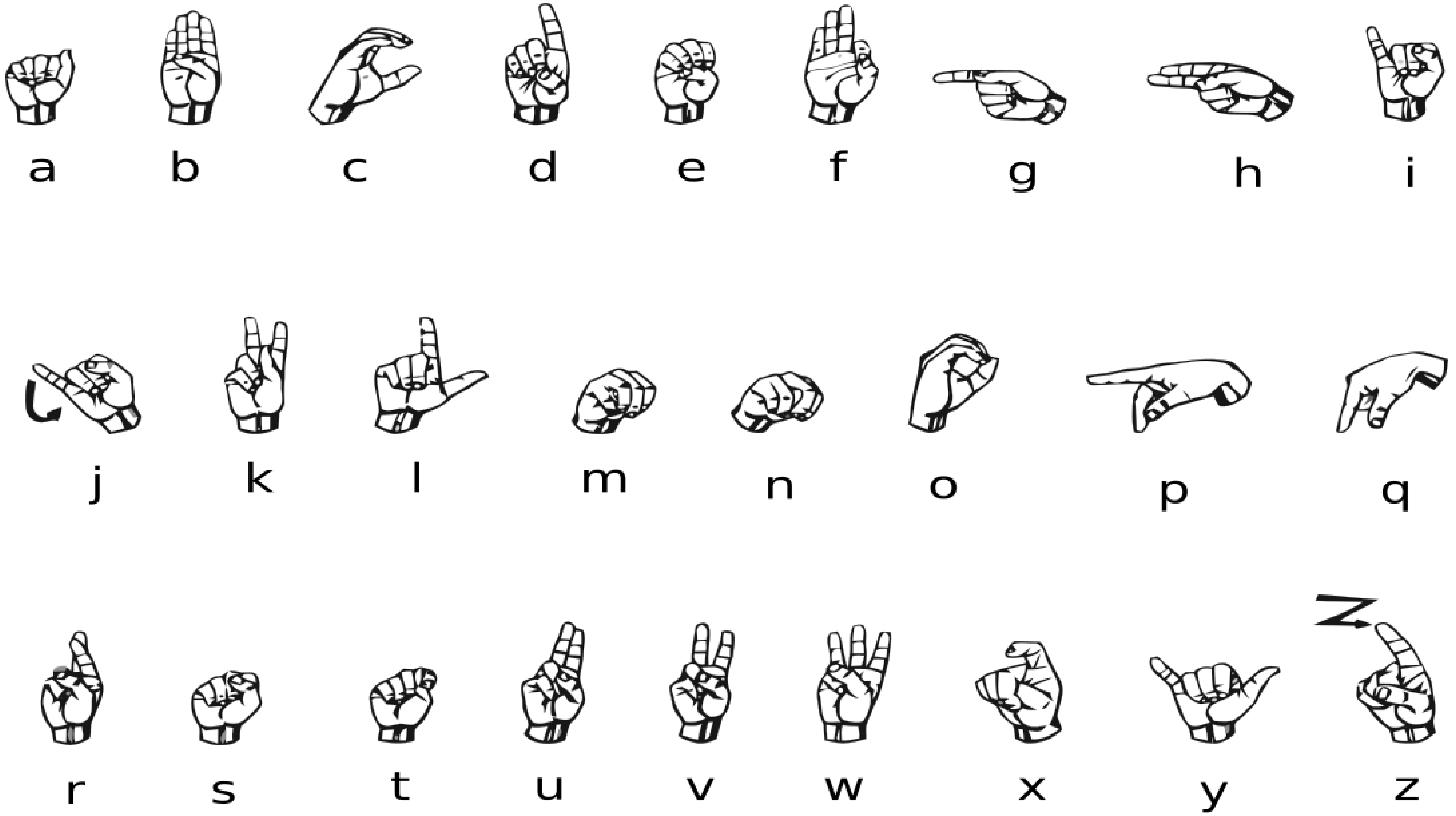



```python
def cross_entropy(y_hat, y_actual):
    """Infinite error for misplaced confidence."""
    loss = log(y_hat) if y_actual else log(1-y_hat)
    return -1*loss
```

彙總整理

下一個練習

美國手語字母(American Sign Language Alphabet)



Let’s go!開始吧！

附錄：梯度下降(Gradient Descent)

幫助電腦作弊微積分

從錯誤中學習

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2 = \frac{1}{n}\sum_{i=1}^{n}(y - (mx + b))^2$$

$$MSE = \frac{1}{2}((3 - (m(1) + b))^2 + (5 - (m(2) + b))^2)$$

$$\frac{\partial MSE}{\partial m} = 5m + 3b - 13$$

$$\frac{\partial MSE}{\partial b} = 3m + 2b - 8$$

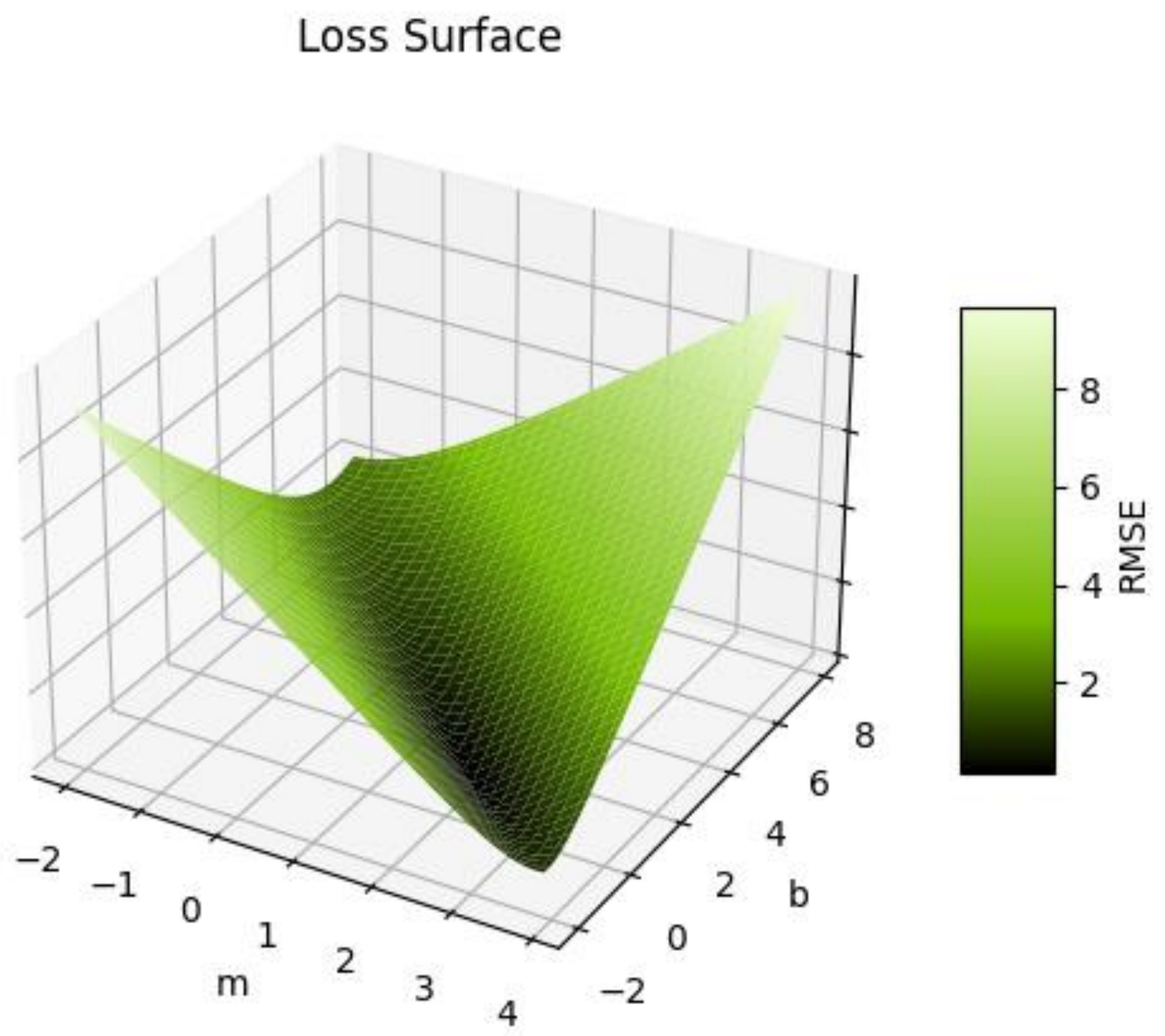
$$\frac{\partial MSE}{\partial m} = -3$$

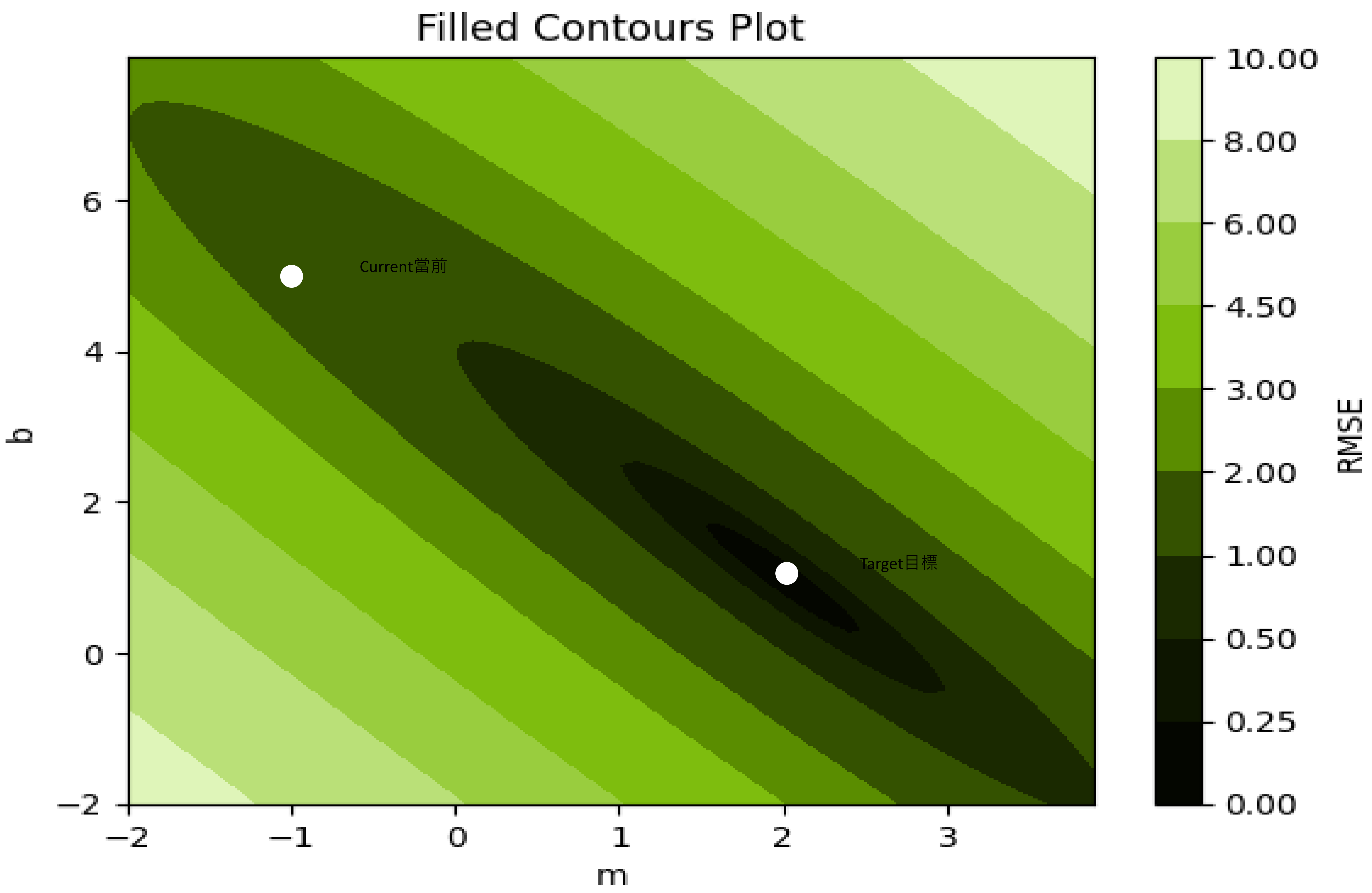
$$\frac{\partial MSE}{\partial b} = -1$$

$m = -1$

$\mathrm{b} = 5$

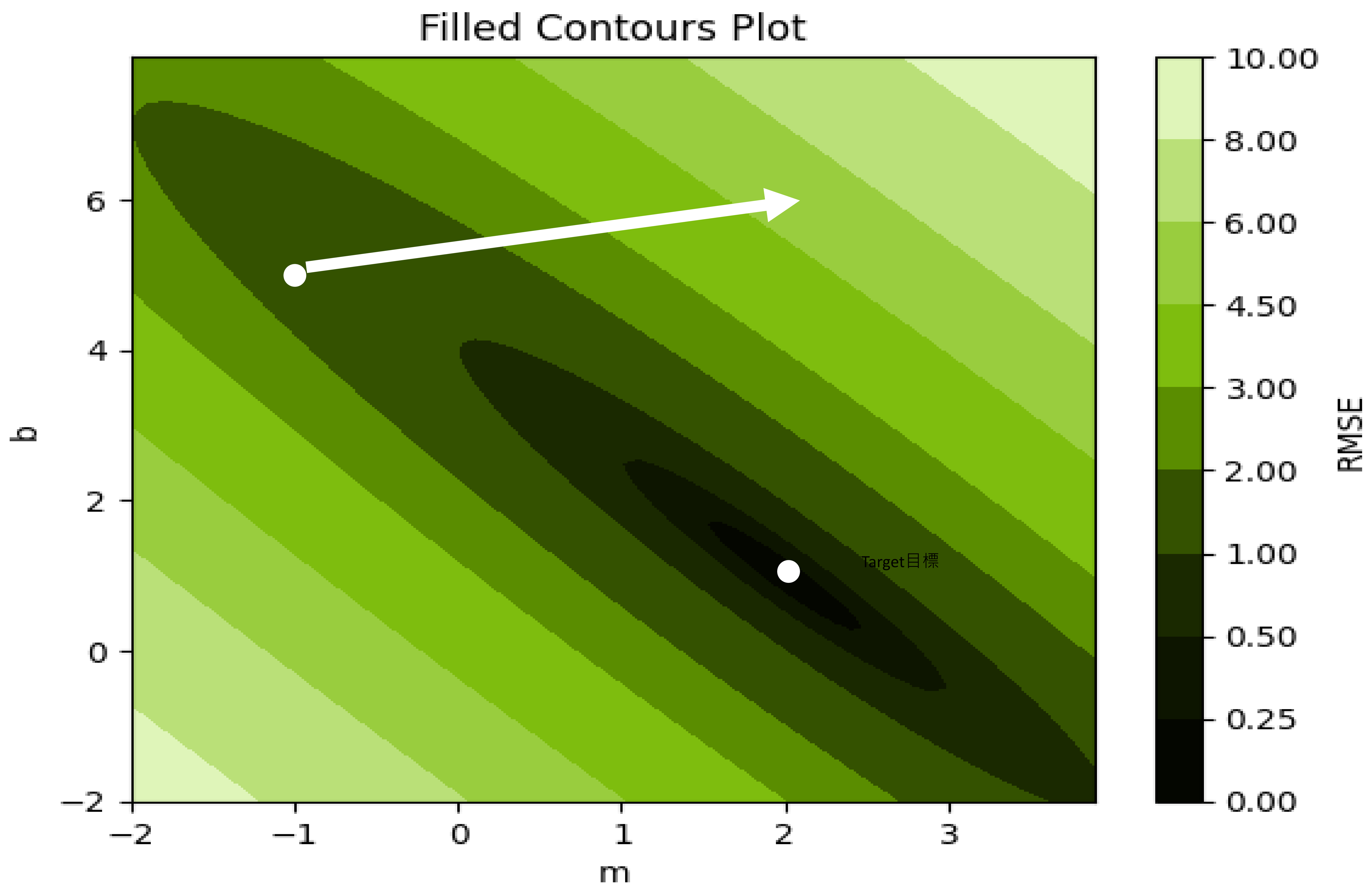
損失曲線(Loss Curve)





損失曲線(Loss Curve)

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

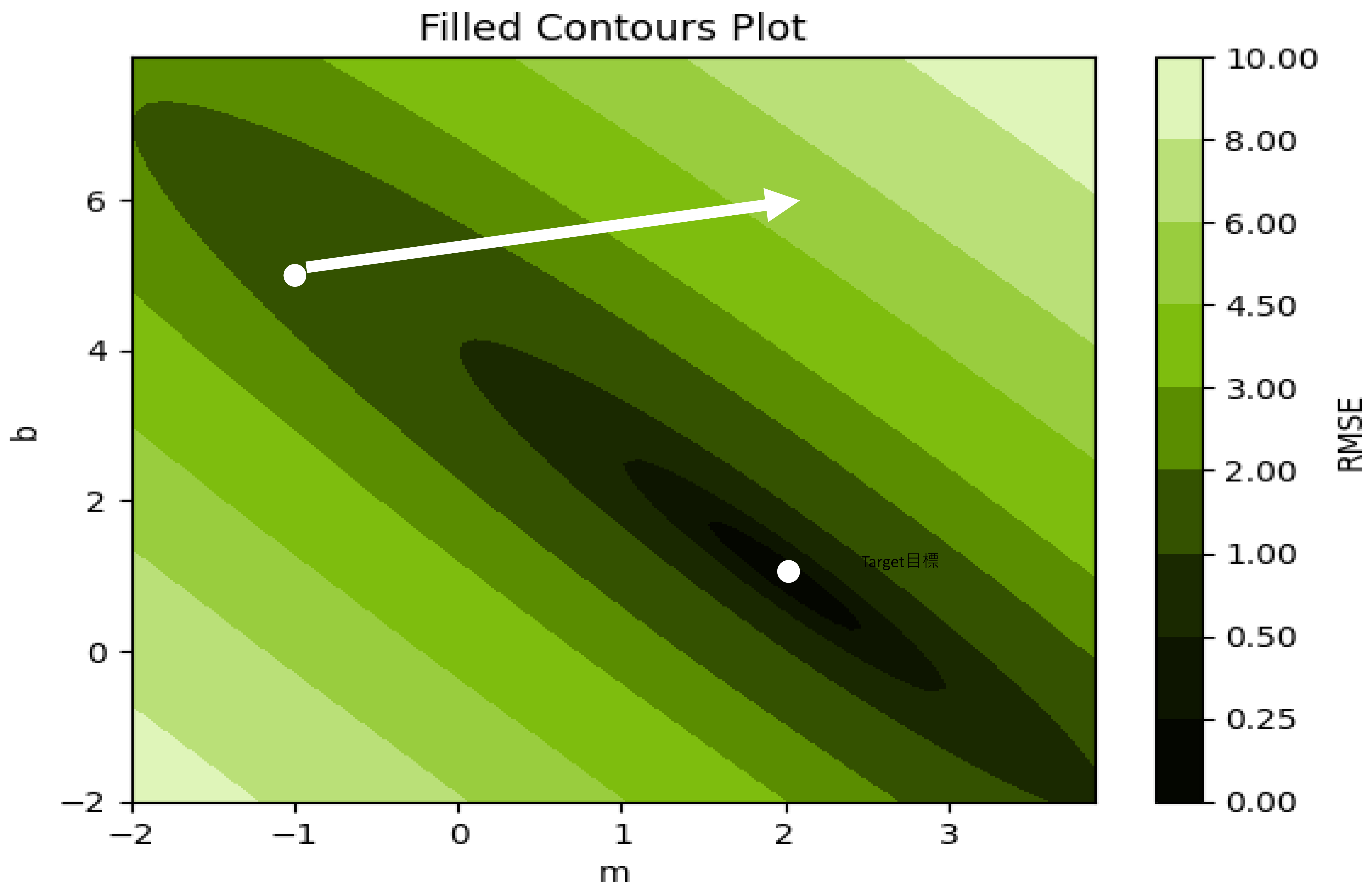


損失曲線(Loss Curve)

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$



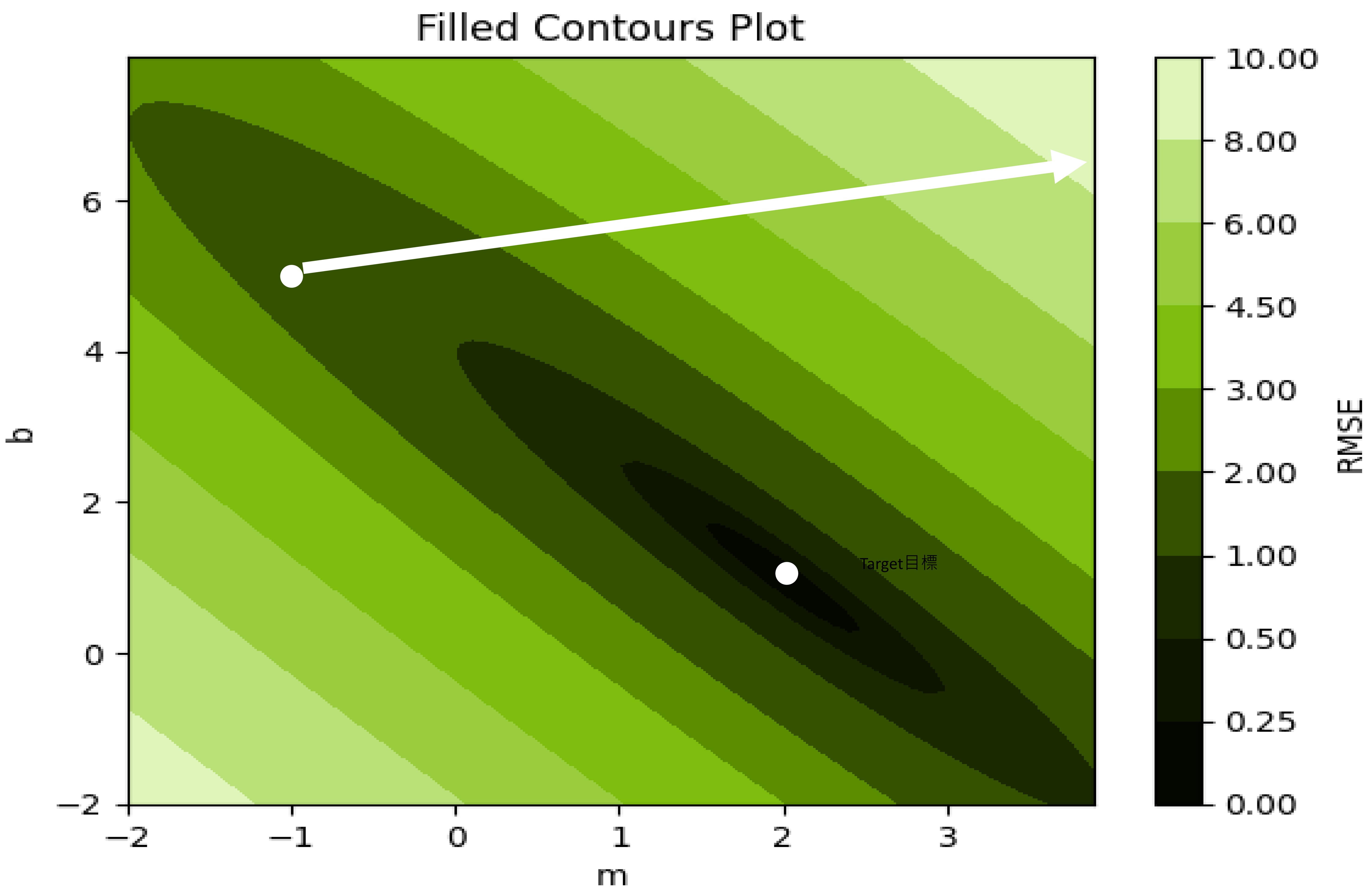
損失曲線(Loss Curve)

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\lambda = 2$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$



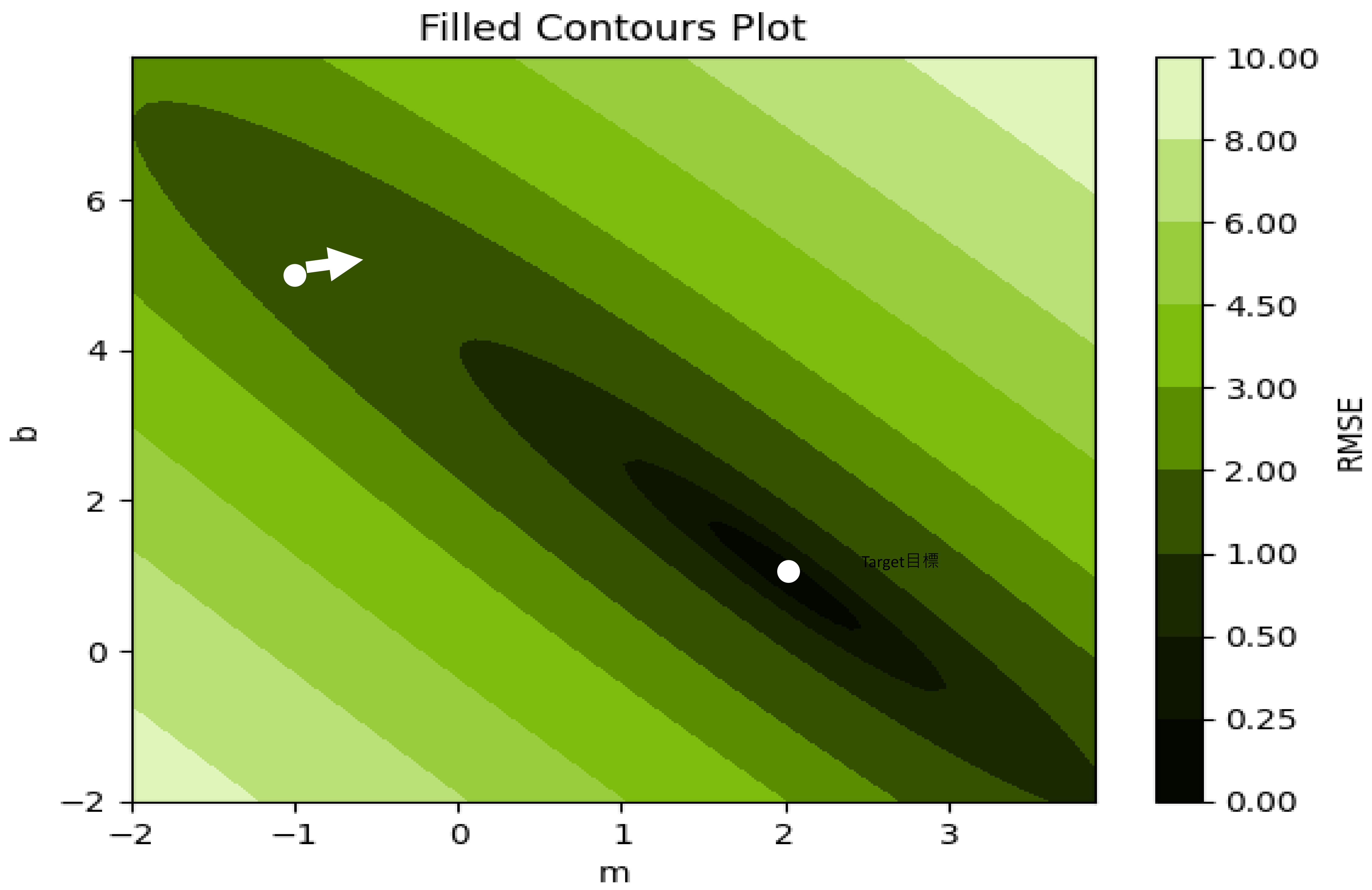
損失曲線(Loss Curve)

$$\frac{\partial MSE}{\partial m} = -3 \qquad \frac{\partial MSE}{\partial b} = -1$$

$$\mathrm{m} := \mathrm{m} - \lambda \frac{\partial MSE}{\partial m}$$

$$\lambda = .1$$

$$\mathrm{b} := b - \lambda \frac{\partial MSE}{\partial b}$$

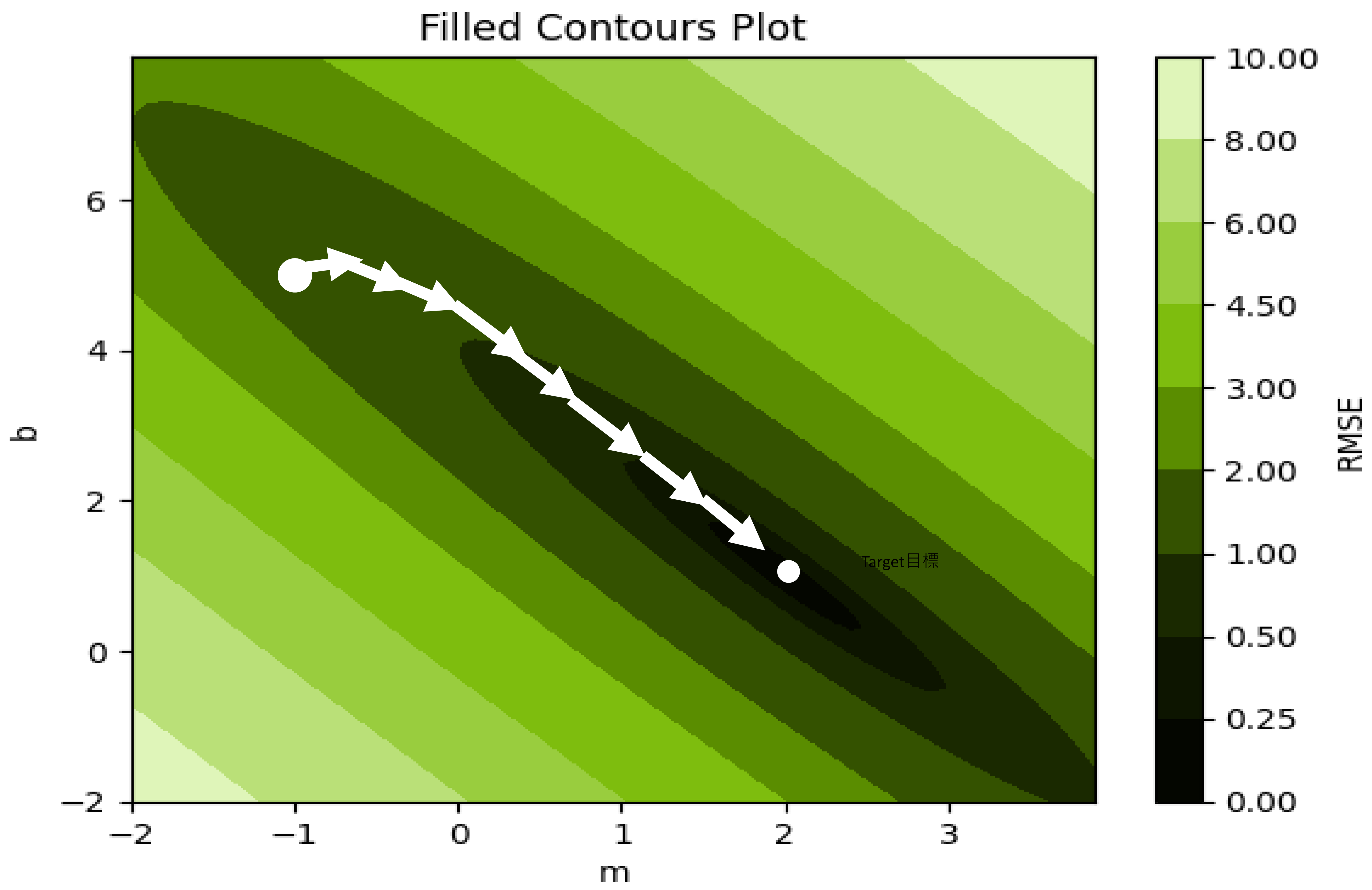


損失曲線(Loss Curve)

$\lambda = .1$

$m \coloneqq -1 + 3\,\lambda = -0.7$

$b \coloneqq 5 + \lambda = 5.1$



NVIDIA®