

Deep learning with keras

CH2

CONTENT

- 01 初探神經網路：第一隻神經網路**
- 02 神經網路的資料表示法：張量TENSOR**
- 03 神經網路的工具：張量運算**
- 04 神經網路的引擎：以梯度為基礎的最佳化**
- 05 ~~重新檢視我們的第一個例子~~**



2-1

第一隻神經網路

MINST 資料集

- 機器學習的 Hello World
- (train_images,train_labels) : 訓練資料集
- (test_images,test_labels) : 測試資料集

需要的元件

- 損失函數 (loss function) : 衡量訓練表現
- 優化器 (optimizer) : 依損失函數和訓練資料更新權重參數
- 評量指標 (merics) : 辨識準確度

STEP



載入 MINST



指定參數



預處理



訓練



2-2

張量 Tensor

張量的屬性

- ndim : 軸的數量
- shape : 各軸的 size
- dtype : 資料的型態

張量的切片

- 選擇張量中的特定元素

資料批次

- 模型會將資料拆成一小批進行學習



2 - 3

張量運算

Element-wise (向量化運算)

Relu

SAMPLE 1

Add

SAMPLE 2

NumPy 的優點

- 將運算工作委託給 BLAS 執行
- BLAS 支援平行高效率的張量運算
- 效率比一般的 python 運算好很多

張量擴張 (broadcasting)

- 使較小的張量進行擴張以匹配形狀較大的張量
- ① 較小的向量會加入新的軸，以匹配較大的向量
- ② 較小的向量會在這些新的軸重複寫入元素，以匹配較大張量的shape

點積 (dot product)

- size 相同才能運算
- 不具對稱性 (交換性)

重塑 (reshaping)

- 調整各軸的元素數
- 矩陣轉置也是其中一種

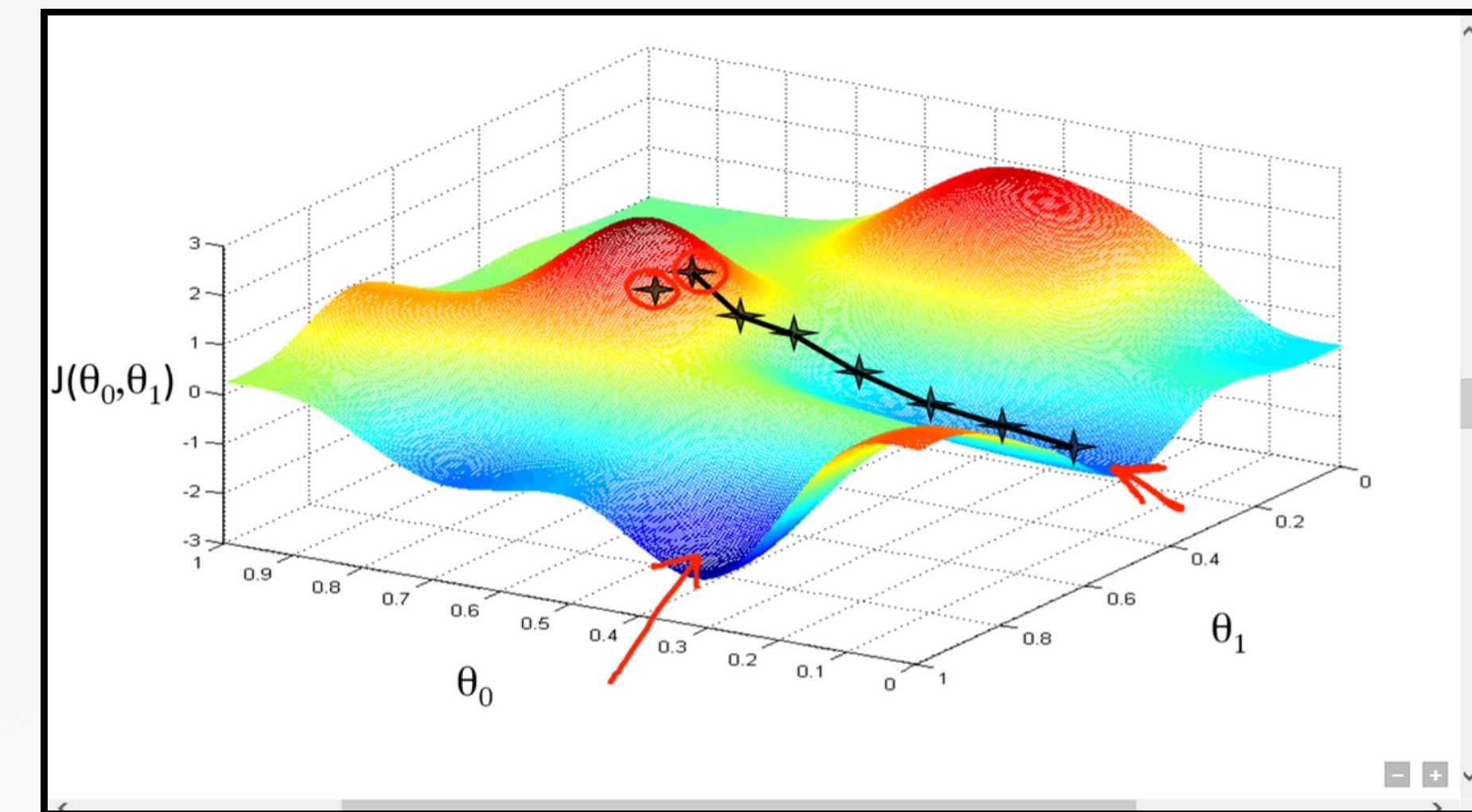
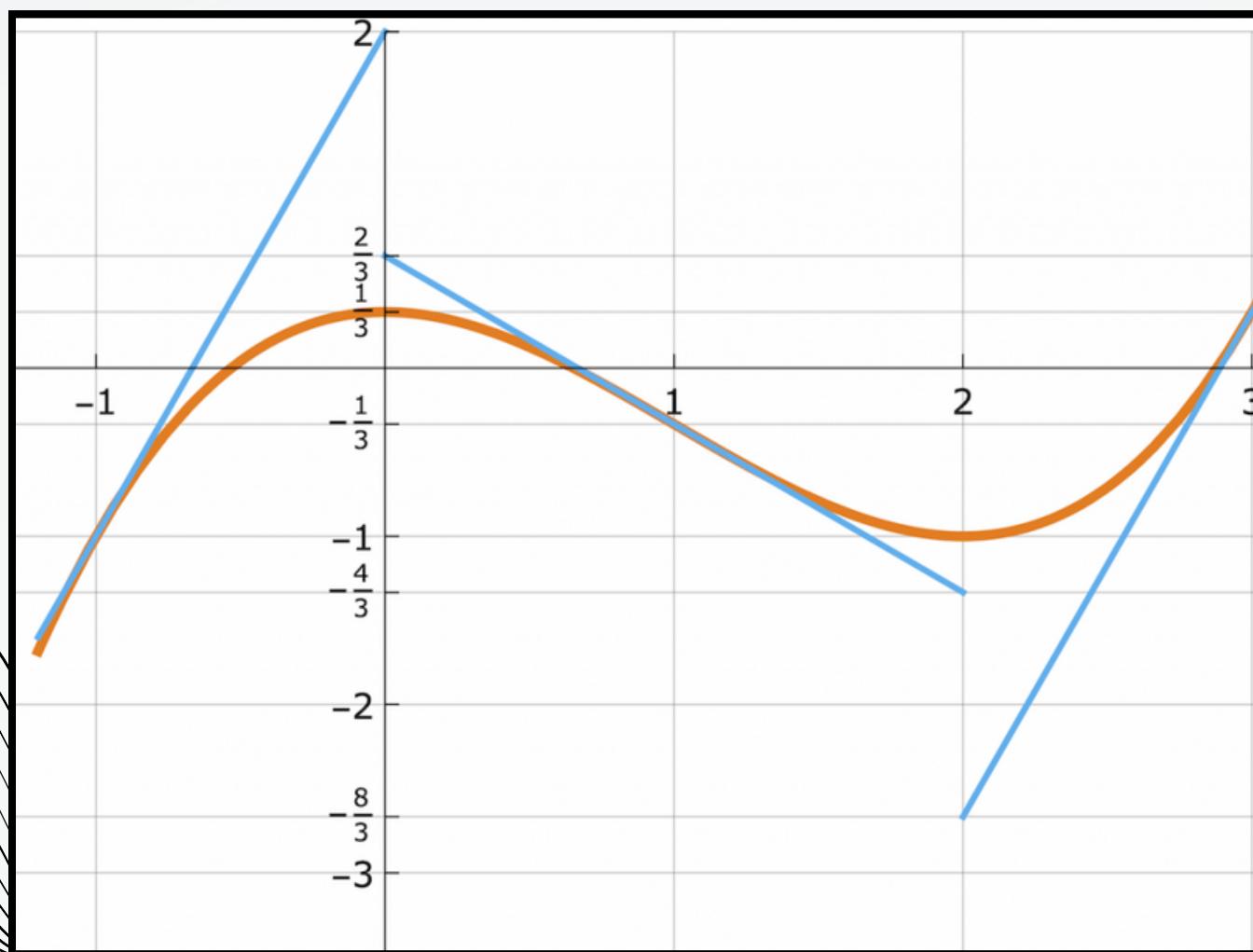
2-4

以梯度為基礎的最佳化

訓練循環

- ① 取出一批訓練樣本 x 和對應的目標 y
- ② 以 x 為輸入資料，開始執行神經網路(正向傳遞)已獲得預測值 y_{pred}
- ③ 計算神經網路的批次量損失值 (y 與 y_{pred} 間的差距)
- ④ 更新神經網路的所有權值，以減少損失值

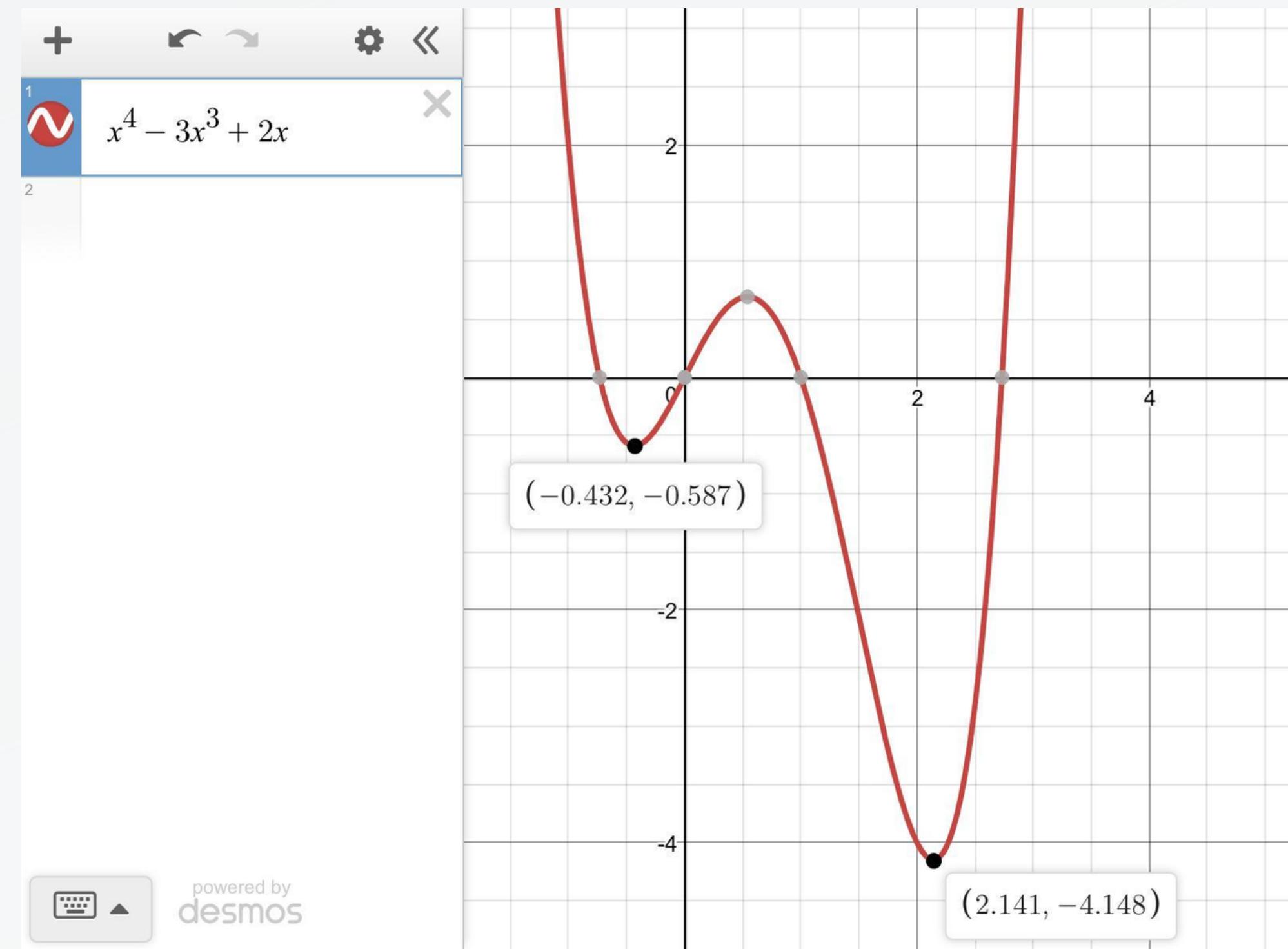
微分和梯度



小批次隨機梯度下降

- ① 取出一批次量的訓練樣本 x 和對應的目標 y
- ② 以 x 為輸入資料，開始執行神經網路(正向傳遞)已獲得預測值 y_{pred}
- ③ 計算神經網路的批次量損失值 (y 與 y_{pred} 間的差距)
- ④ 計算損失值對神經網路權重的梯度 (反向傳播)
- ⑤ 將參數稍微往梯度的反方向移動 ($W \leftarrow learning_rate * gradient$)

梯度最佳化的問題



利用動量解決最佳化

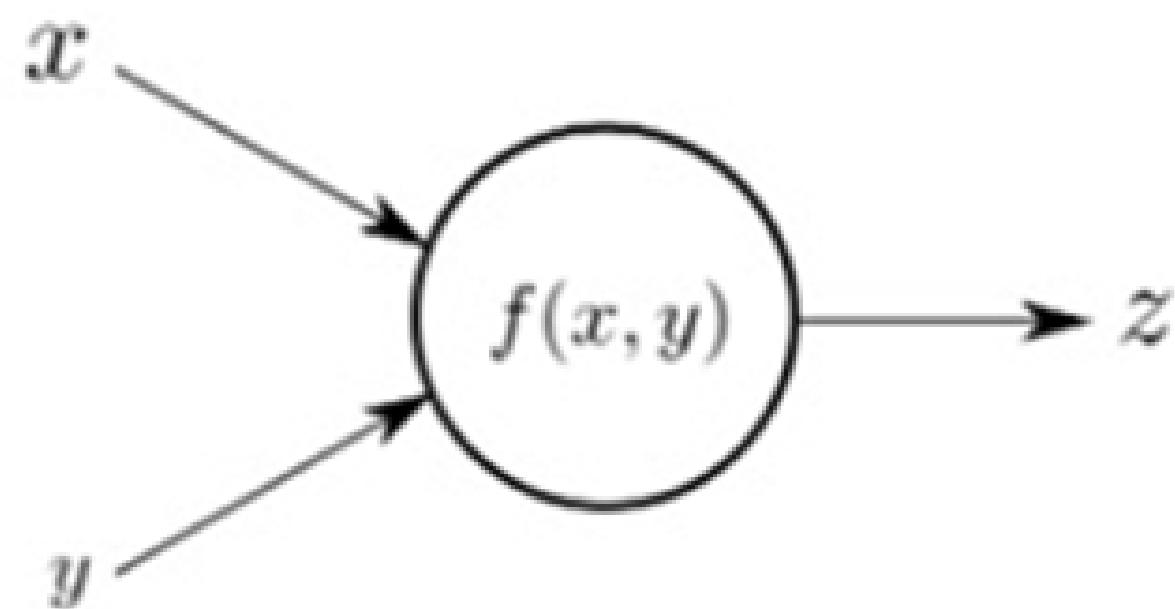
```
# momentum

past_velocity = 0.
momentum = 0.1
while loss > 0.01:
    w, loss, gradient = get_current_parameters()
    velocity = past_velocity * momentum - learning_rate * gradient
    w = w + momentum * velocity - learning_rate * gradient
    update_parameter(w)
```

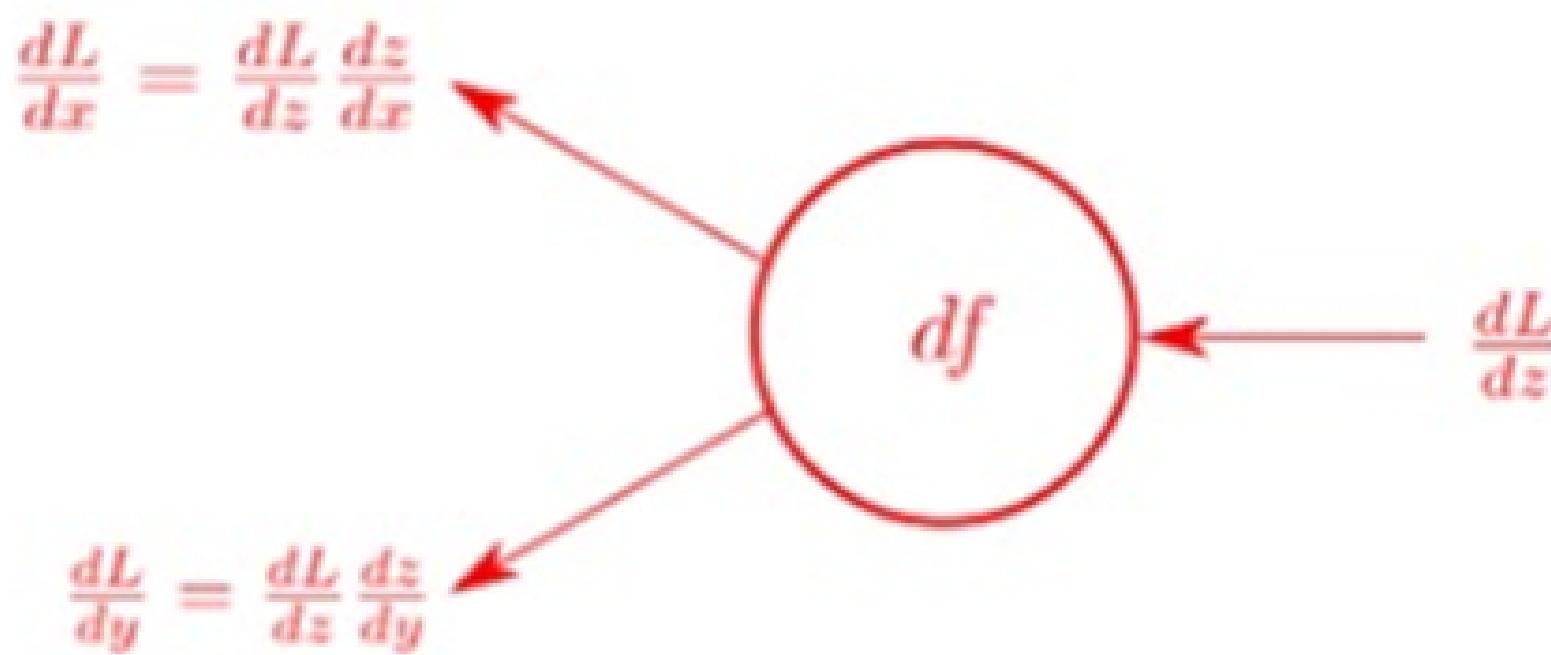
正向傳播

反向傳播

Forwardpass



Backwardpass



TensorFlow 的工具

Gradient Tape

ENDING

