

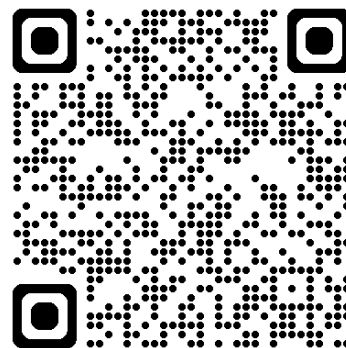


深度學習

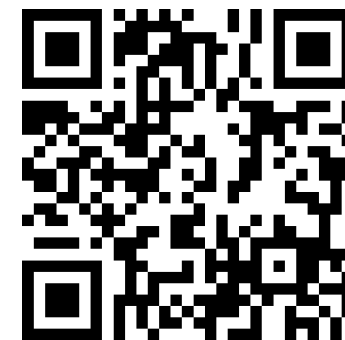
Deep Learning

常見目標函數

Instructor: 林英嘉 (Ying-Jia Lin)
2025/10/01



[Course GitHub](#)



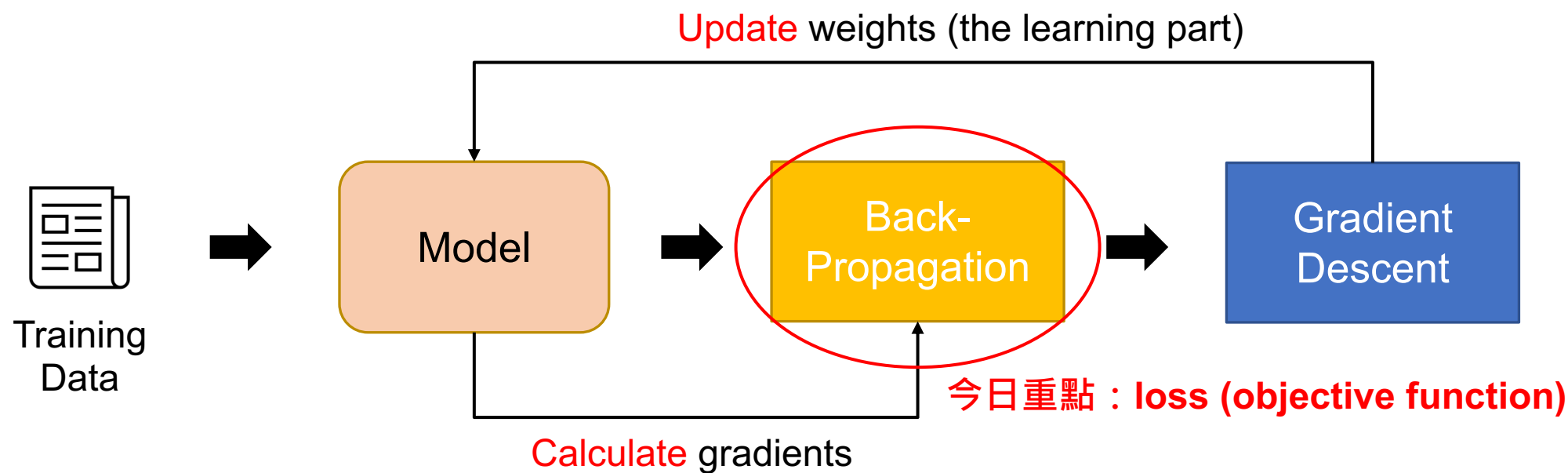
[Slido # DL1001](#)

Outline

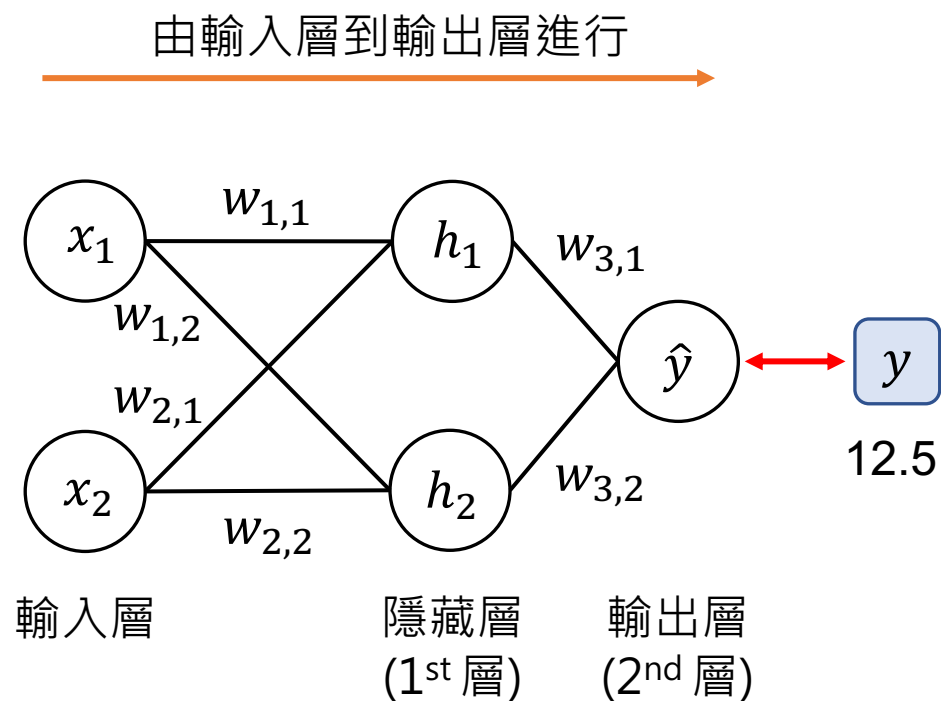
- Common objective functions [60 min]
- PyTorch modeling [60 min]
 - PyTorch modeling code using MNIST
 - PyTorch Gradient Descent code (if we have time)
- Quiz [20 min]



[Recap] 深度學習模型訓練流程



Forward Pass and Objective Function



迴歸任務：使用 Mean Squared Error (MSE) 作為目標函數

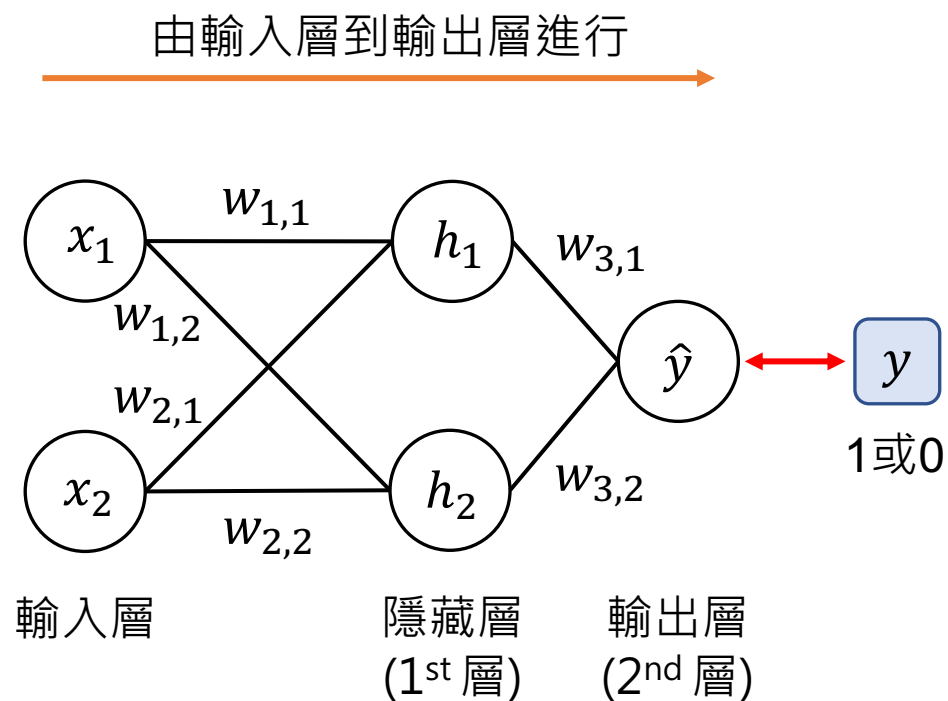
$$h_1 = w_{1,1}x_1 + w_{2,1}x_2 + b_1$$

$$h_2 = w_{1,2}x_1 + w_{2,2}x_2 + b_2$$

$$\hat{y} = w_{3,1}h_1 + w_{3,2}h_2 + b_3$$



Forward Pass and Objective Function



分類任務：使用 **Cross-entropy** 作為目標函數

$$h_1 = w_{1,1}x_1 + w_{2,1}x_2 + b_1$$

$$h_2 = w_{1,2}x_1 + w_{2,2}x_2 + b_2$$

$$\hat{y} = w_{3,1}h_1 + w_{3,2}h_2 + b_3$$



Entropy (資訊理論)

- Entropy (熵): 定義為不確定性的量度
- 特性：
 - 越不可能發生的事情，當它發生了，
越會提供更多的資訊，**entropy** 就越高
- Claude Shannon 將熱力學的熵引入到資訊理論 – 夏農熵 (Shannon entropy)

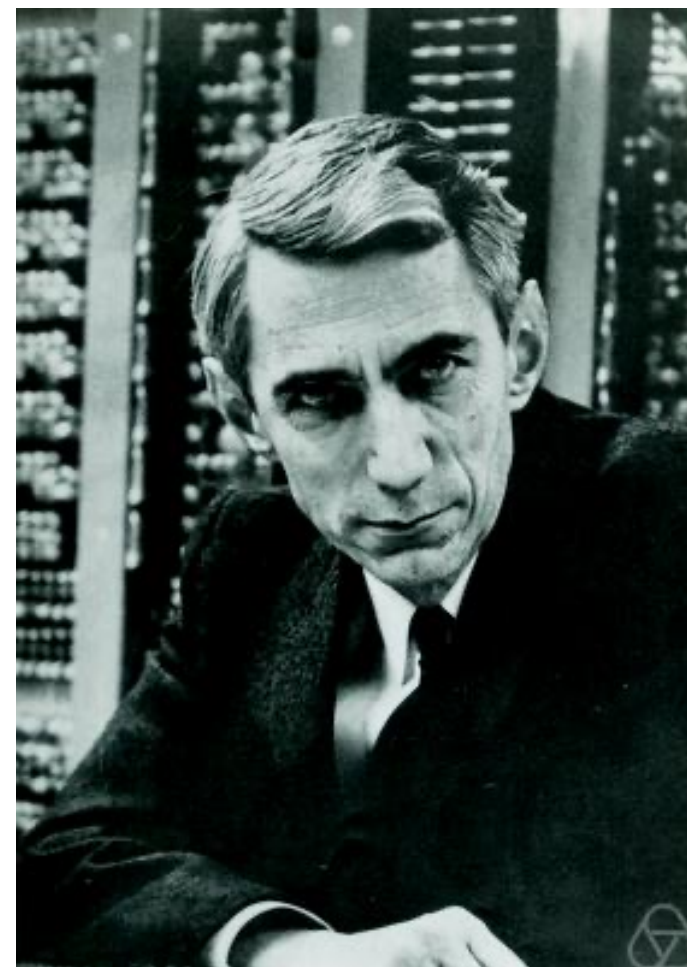






Figure source: <https://www.itsoc.org/about/shannon>



[定義] 事件發生的機率越低代表資訊量越高

	機率 (P)	資訊量  $1 / P$
	50%	2
	25%	4
	25%	4



從通訊的角度 理解 Entropy



Figure source: <https://newsweb.ncc.gov.tw/201910/ch4.html>

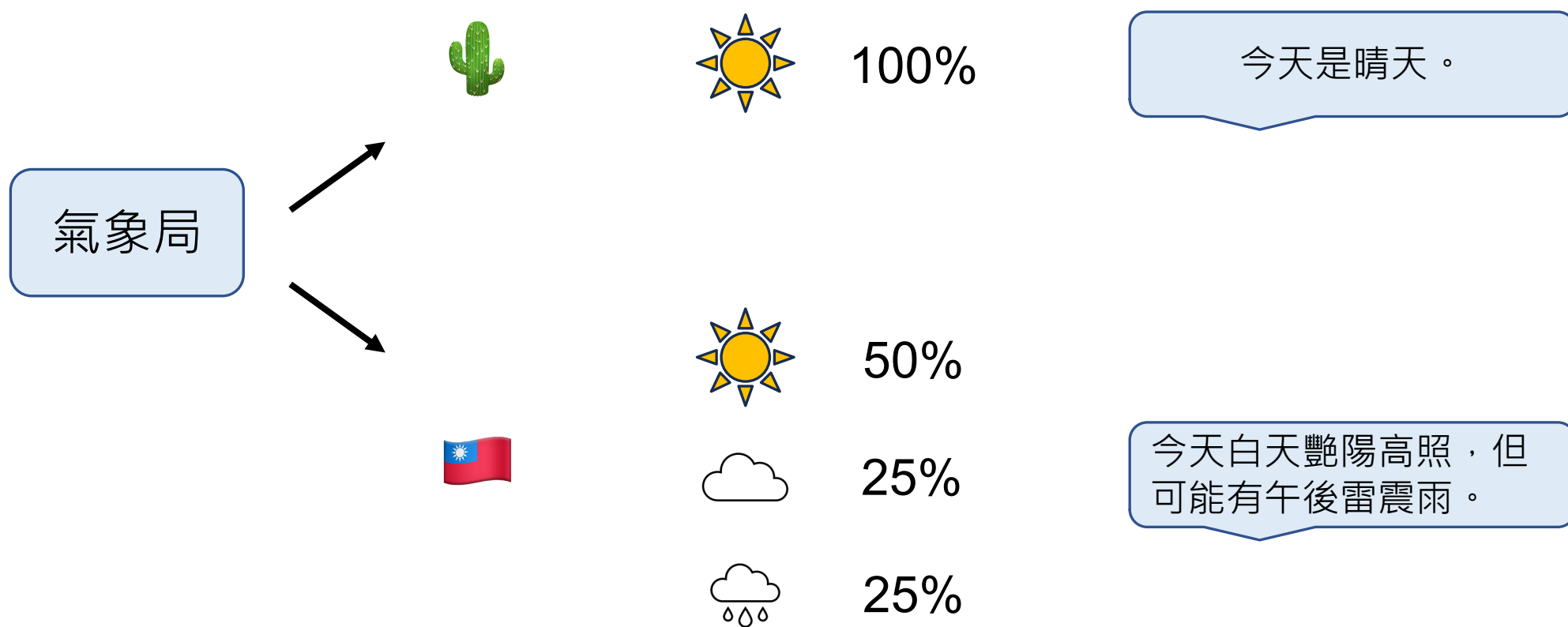
通訊傳輸 (核心概念)

通訊傳輸很貴，

如果我們讓發生機率較高的事件，用較少的資訊量來傳遞，可以節省通訊傳輸成本。



通訊傳輸 (Example)



Entropy：平均傳輸成本的理論下限

Entropy = 每個事件發生的機率 **x** log(每個事件發生的資訊量)

$$= H(P) = P \times \log_2 \frac{1}{P} \quad \leftarrow \text{通常是自然對數，本堂課以2為底數作為範例}$$




$$= -P \times \log_2 P$$

💡 Entropy 是我們能達到的最低平均傳輸成本 ([link](#))



Entropy：最低的傳輸成本 (以🇹🇼為例)




$$\text{Entropy} = H(P) = P \times \log_2 \frac{1}{P}$$

	機率 (P)	1 / P	$\log_2 \frac{1}{P}$	H(P)	
	50%	2	1	0.5] 1.5 bits log以2為底 數的單位
	25%	4	2	0.5	
	25%	4	2	0.5	



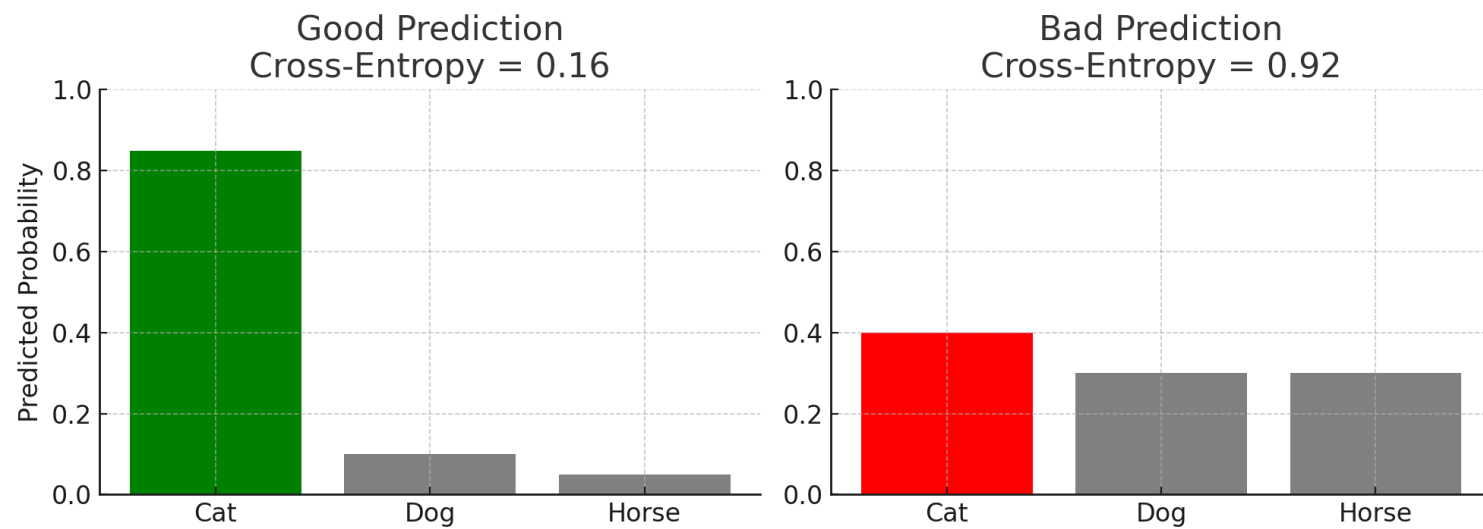
Entropy：最低的傳輸成本 (以🌵為例)

$$\text{Entropy} = H(P) = P \times \log_2 \frac{1}{P}$$

	機率 (P)	1 / P	$\log_2 \frac{1}{P}$	H(P)	
	100%	1	0	0] <div>log以2為底 數的單位</div> 0 bits
	0%	0	0	0	
	0%	0	0	0	



從ML的角度理解 Cross-Entropy



Cross-Entropy：可能不是最低的傳輸成本

Entropy = 每個事件發生的機率 \times $\log(\text{每個事件發生的資訊量})$

Cross-Entropy = 每個事件發生的機率 \times $\log(\text{預測每個事件發生的資訊量})$
交叉熵

$$= H(P, Q) = P \times \log_2 \frac{1}{Q} \quad \leftarrow \text{通常是自然對數，本堂課以2為底數作為範例}$$

$$= -P \times \log_2 Q \quad \leftarrow \begin{array}{l} P \text{ 是每個事件發生的機率，} \\ Q \text{ 是預測每個事件發生的機率} \end{array}$$



Cross-entropy (Case1)

$$H(P, Q) = P \times \log_2 \frac{1}{Q}$$

	機率 (P)	機率 (Q)	1 / Q	$\log_2 \frac{1}{Q}$	H(P, Q)
--	--------	--------	-------	----------------------	---------



50%

25%

4

2

1



25%

25%

4

2

0.5



25%

50%

2

1

0.25

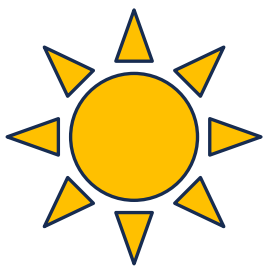
1.75 bits



Cross-entropy (Case2)

$$H(P, Q) = P \times \log_2 \frac{1}{Q}$$

	機率 (P)	機率 (Q)	1 / Q	$\log_2 \frac{1}{Q}$	H(P, Q)
--	--------	--------	-------	----------------------	---------



50%

60%

1.67

0.74

0.37



25%

20%

5

2.32

0.58



25%

20%

5

2.32

0.58

1.53 bits



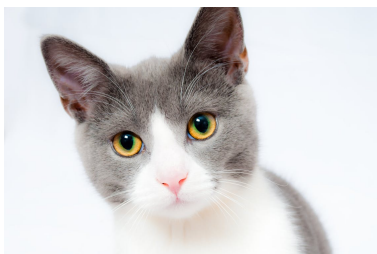
Summary for Cross-entropy

- Entropy 帶來最低的平均傳輸成本
 - 初衷：發生機率高的事件，可以用較低的資訊量進行傳輸
- Cross-entropy 代表用了**錯估的資訊量**後所得到的平均傳輸成本
 - 許多機器學習就是採用 Cross-entropy 來讓**錯誤的分佈**接近**最佳**的分佈

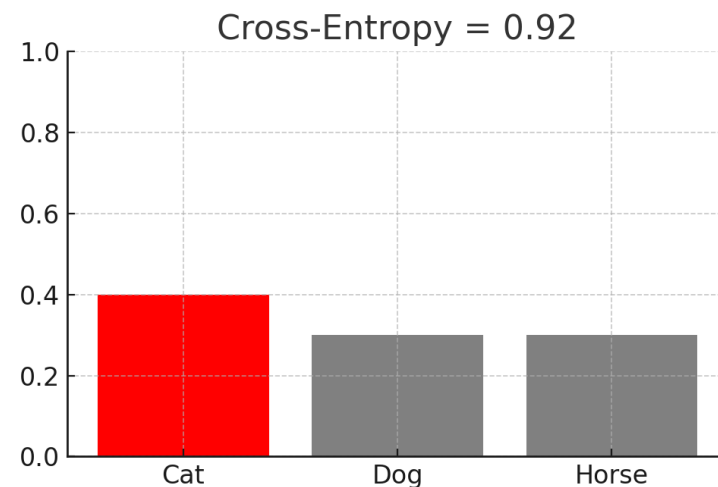
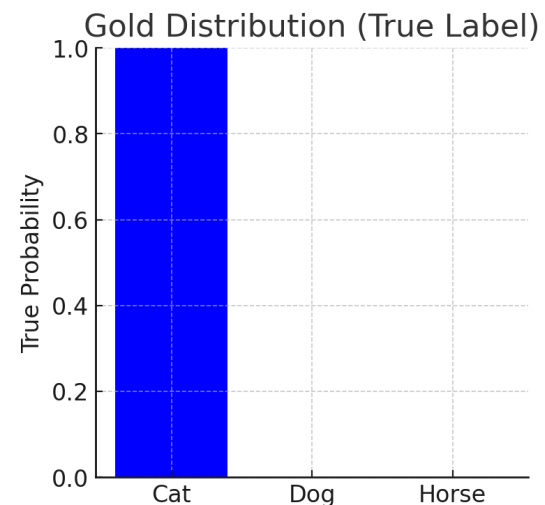


[定義] 機率分佈

機率分佈：機率分佈是指對一個隨機變數的所有可能取值，分別給出它們發生的機率，而這些機率的總和必須為 1。



Model



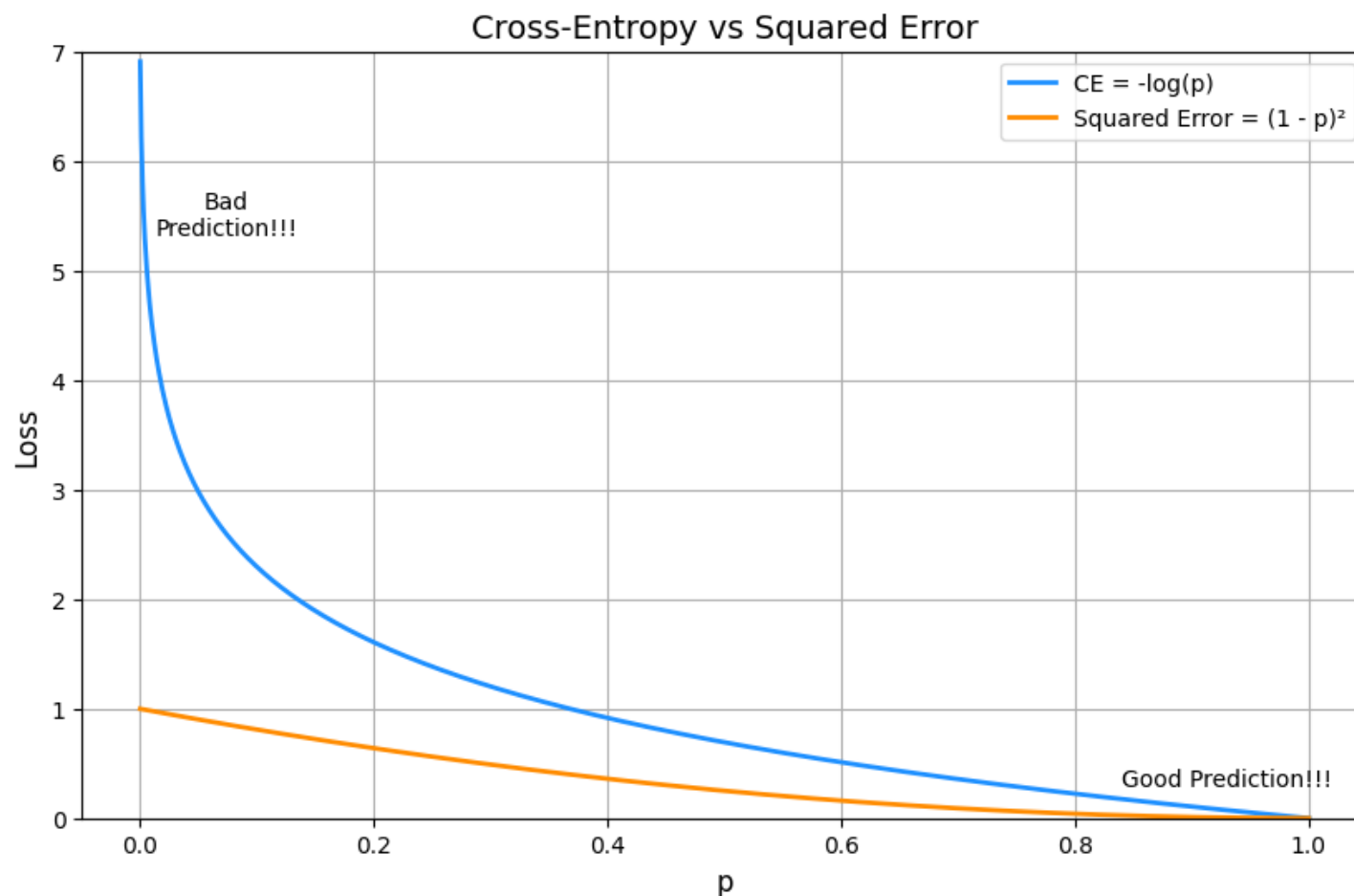
越接近越好



為什麼分類任務不用 MSE?

code/w5_plot_loss.py

- **Ans:** 在分類任務上使用 **Cross-entropy**，能夠在模型預測差時帶來更大的梯度 -> 模型能有更大的幅度更新參數



實作細節

Cross-entropy in ML

- 對於多數的機器學習分類任務來說，我們目標讓模型學會正確的類別



Cat

$$H(P, Q) = \underset{1}{P} \times \ln \frac{1}{Q} = -\ln Q$$



Dog

$$H(P, Q) = \underset{0}{P} \times \ln \frac{1}{Q} = 0$$



Apple

$$H(P, Q) = \underset{0}{P} \times \ln \frac{1}{Q} = 0$$



Cross-entropy in ML

- 對於多數的機器學習分類任務來說，我們目標讓模型學會正確的類別



Cat

$$H(P, Q) = \underset{0}{P} \times \ln \frac{1}{Q} = 0$$



Dog

$$H(P, Q) = \underset{1}{P} \times \ln \frac{1}{Q} = -\ln Q$$



Apple

$$H(P, Q) = \underset{0}{P} \times \ln \frac{1}{Q} = 0$$

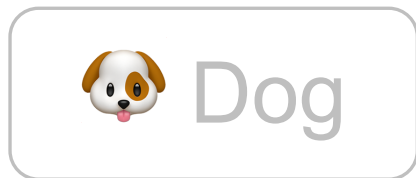


Cross-entropy in ML

- 對於多數的機器學習分類任務來說，我們目標讓模型學會正確的類別



$$H(P, Q) = \underset{0}{P} \times \ln \frac{1}{Q} = 0$$



$$H(P, Q) = \underset{0}{P} \times \ln \frac{1}{Q} = 0$$



$$H(P, Q) = \underset{1}{P} \times \ln \frac{1}{Q} = -\ln Q$$



Cross-entropy 公式

$$H(P, Q) = -P \times \ln Q$$

所有類別的 cross-entropy 加總

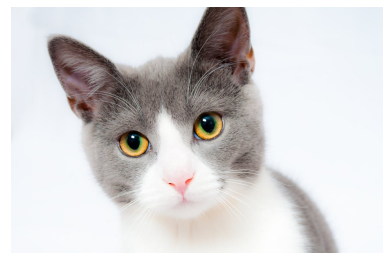
$$\text{Cross-entropy: } \mathcal{L}_i = - \sum_j y_j \ln P(\hat{y}_j | x_i)$$

y_j 發生的機率值 (P)

模型看到 x_i 預測 \hat{y}_j 的機率值 (Q)

x_i : 資料集中第 i 筆資料

y_j : label, 在此範例中 $j = 1, 2, 3$



x_i

→ Model

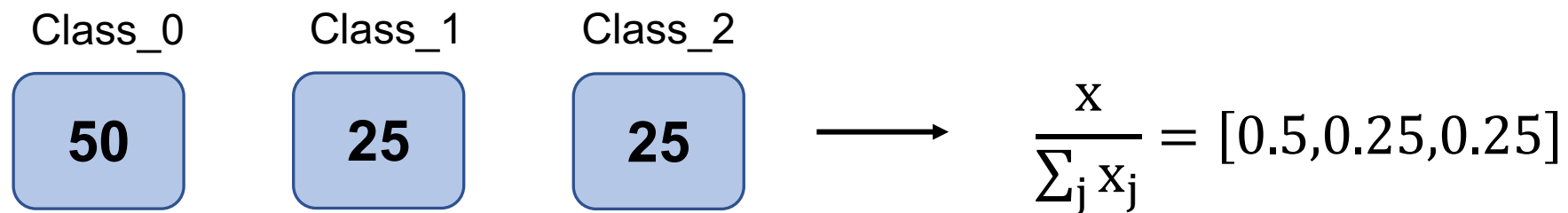
→

	Model output (logits)	Softmax $P(\hat{y}_j x_i)$	label (y_j)	Cross- entropy
y_1 Cat	0.5	0.225	1	$-\ln 0.225$
y_2 Dog	0.7	0.275	0	0
y_3 Apple	1.3	0.500	0	0

各類別加總: $\mathcal{L}_i = -\ln 0.225$



如何把模型輸出變成機率值 (0~1) ?



- **Softmax function**，採用 exponential -> 大的數值更大，小的數值更小
 - 有助於梯度下降

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} = [0.665, 0.244, 0.090]$$

注意！Softmax 有考慮到類別總和



兩種方法來做 Binary Classification

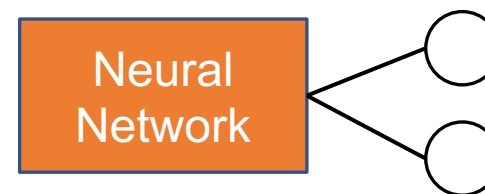
多類別分類任務



是貓

不是貓

對兩個模型輸出值取機率值，
且機率值總和為1

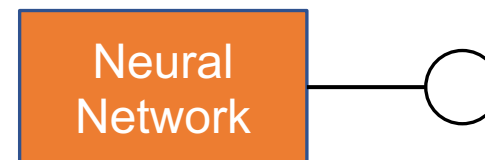


二元分類任務



> 0.5 是貓，否則不是貓

對一個模型輸出值取機率值

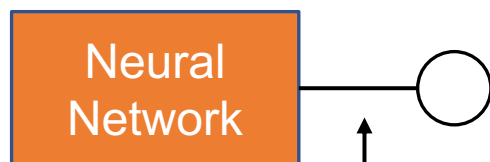


實作上採用 **BinaryCrossEntropy**



如何對一個模型輸出值取機率值 (0~1) ?

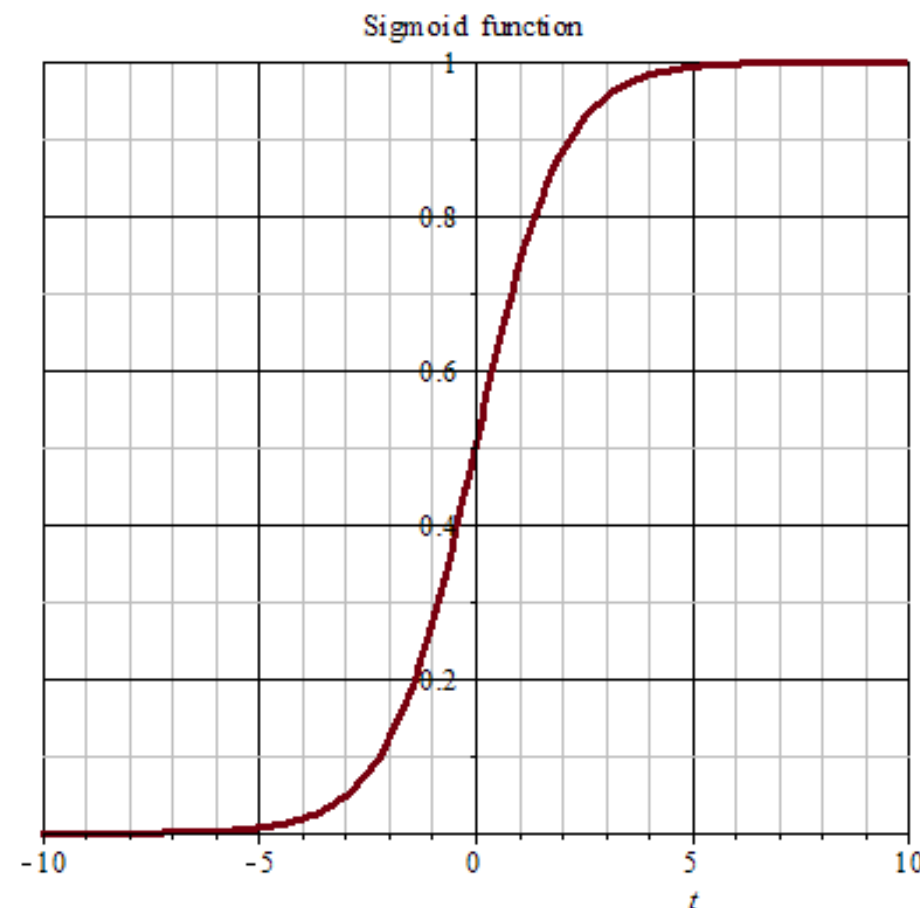
- Sigmoid function:



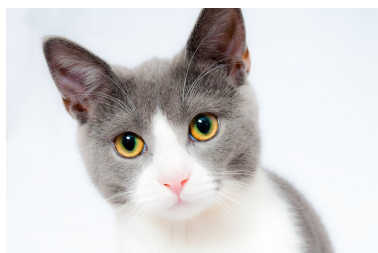
$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

假設閾值為0.5

> 0.5 是貓，
否則不是貓



Multi-class vs. Multi-label Classification



標註 (label)

貓

label_id

0

狗

1

馬

2

Multi-class
Classification
一次只會有一個類別



囊腫



血管瘤

脂肪瘤



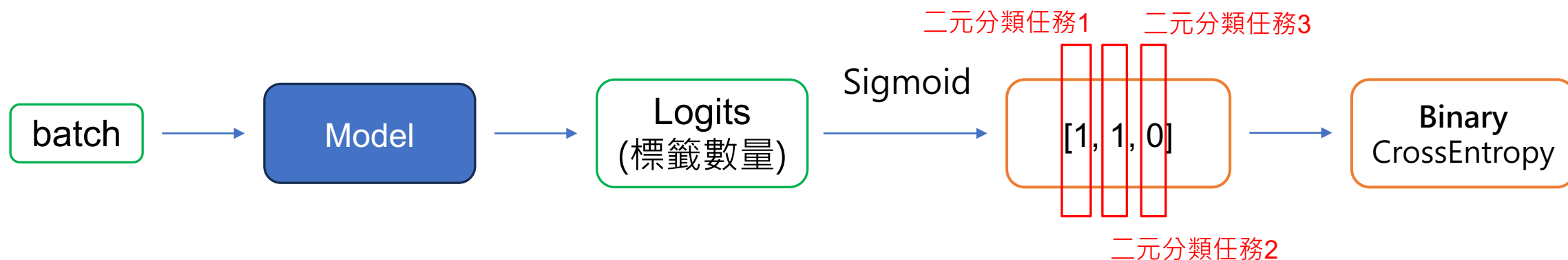
[1, 0, 1]

Multi-label
Classification
一次可以有多个類別



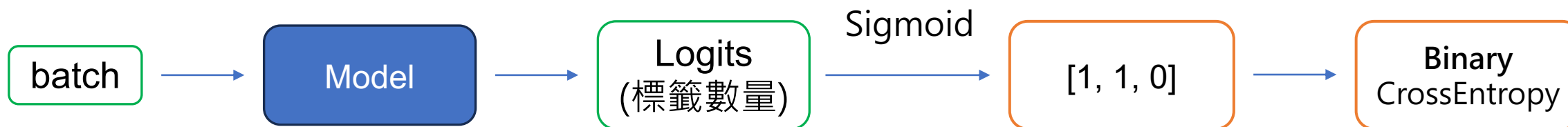
[注意事項] Multi-label Classification

- Multi-label Classification 實作上與二元分類任務類似：
 - 模型輸出需經過 Sigmoid function
 - Loss 同樣採用 BinaryCrossEntropy
 - 相當於是做很多次 (次數等於標籤數量) 的二元分類任務

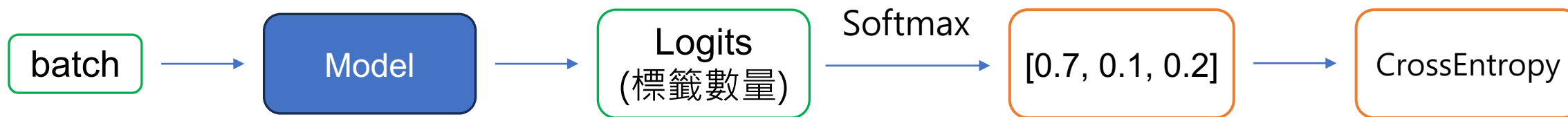


Multi-class vs. Multi-label Classification

- Multi-label (多標籤任務) :



- Multi-class (多類別任務) :



Summary

- 許多機器學習任務採用 Cross-entropy 來讓錯誤的分佈接近最佳的分佈



Object Detection



Segmentation

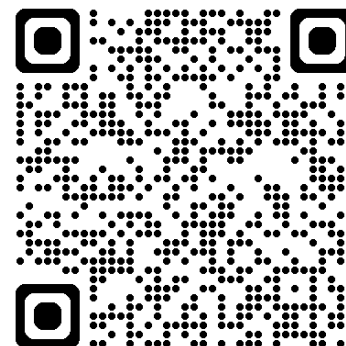




深度學習 Deep Learning

PyTorch Modeling

Instructor: 林英嘉 (Ying-Jia Lin)
2025/10/01



[Course GitHub](#)



[Slido # DL1001](#)

PyTorch Tutorial

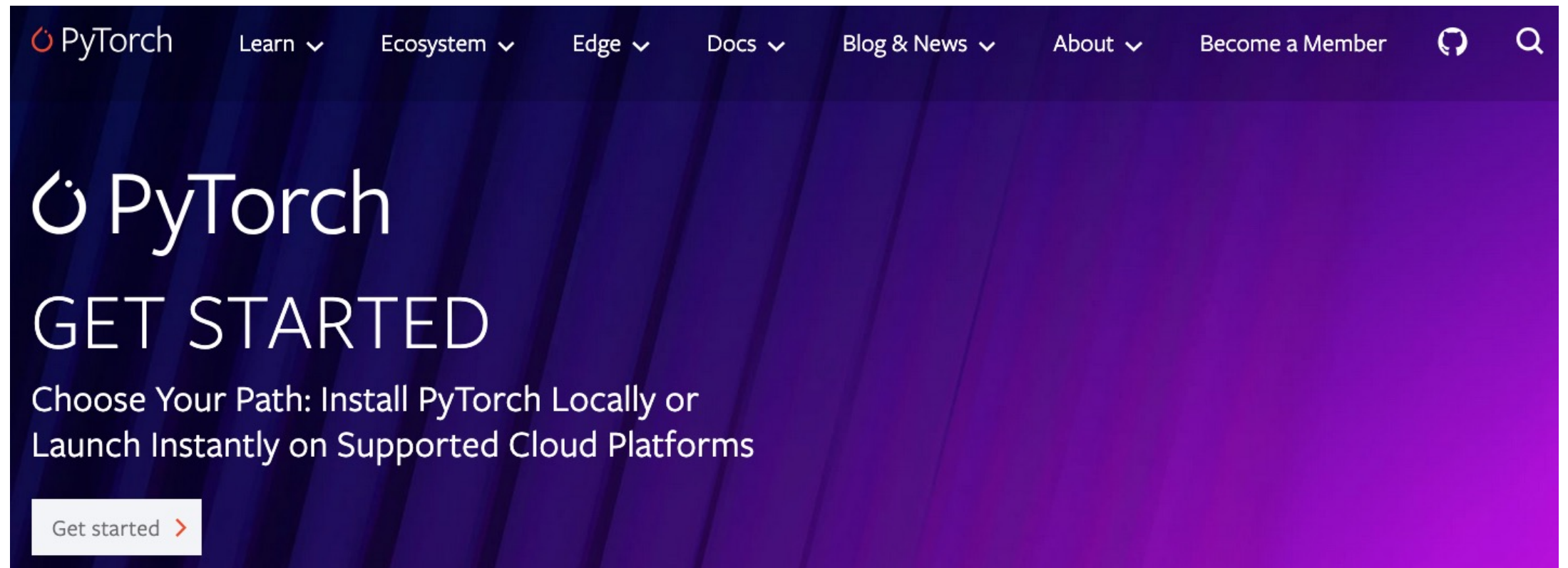
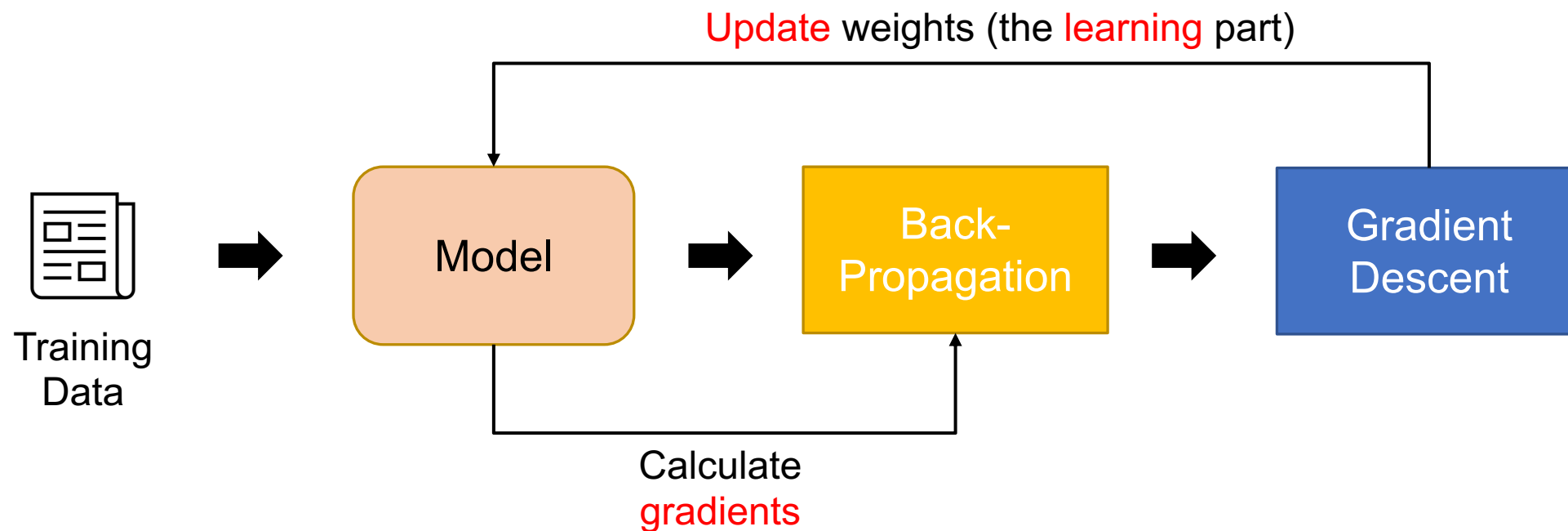


Figure source: <https://pytorch.org/>

[Recap] 深度學習模型訓練流程



Steps for building your first PyTorch program

Step 1 (Data):

- Prepare the dataset
- Overwrite PyTorch Dataset
- Define DataLoader

Step 2 (Model):

- Construct the model
- Define the loss function
- Define the optimizer

Step 3 (Training):

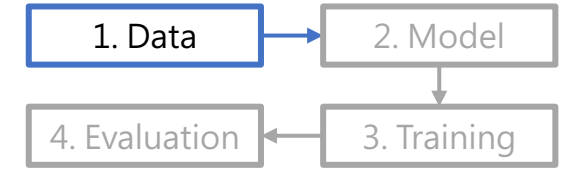
Write the training process

Step 4 (Evaluation):

Write the evaluation process



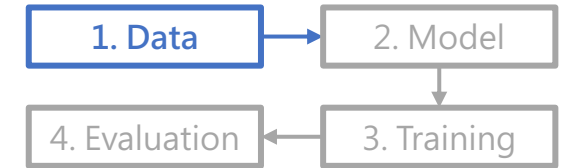
Step 1: Prepare the dataset




- From [torchvision \(image data\)](#) or [torchtext \(text data\)](#)
 - You may skip Step 1-2.
- User-defined dataset
 - Download from the Internet
 - Your own dataset



What is a dataset?



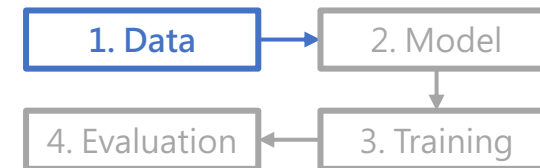
dataset

PassengerId	# Survived	# Pclass	Name	Sex	
			891 unique values	male female	65% 35%
1	0	3	Braund, Mr. Owen Harris	male	
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	
3	1	3	Heikkinen, Miss. Laina	female	
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	
5	0	3	Allen, Mr. William Henry	male	
6	0	3	Moran, Mr. James	male	
7	0	1	McCarthy, Mr. Timothy J	male	

data / instance / example



Step 1-2: 建立 PyTorch Dataset



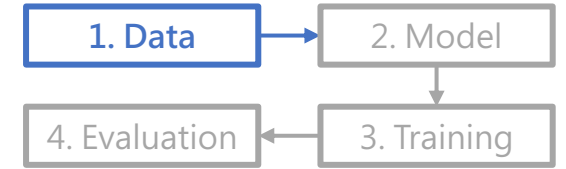
- 為了符合我們載入資料的**需求**
 - 例如：適合我們資料的前處理過程
- 簡潔且容易維護的資料存取介面：

```
img, label = dataset[0] # `dataset` 是透過 PyTorch Dataset 所建立的
```

↑
index



Step 1-2: 建立 PyTorch Dataset



- 我們需要繼承 `torch.utils.data.Dataset`，並改寫三個項目 (`__init__`, `__getitem__`, `__len__`)：

```
import torch

class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, parameter_1, parameter_2, ...):
        # Prepare some things
        # that you are going to use in `__getitem__` and `__len__`

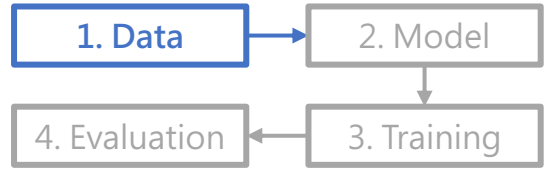
    def __getitem__(self, index):
        # do something
        return data, label

    def __len__(self):
        return len(data_variable)
```

- `__init__`：初始化
class 中的變數
- `__getitem__`：讓
PyTorch Dataset
可以透過 index 來
取得任一筆資料
- `__len__`：取得資
料集的總數



Step 1-2: 建立 PyTorch Dataset



```
import torch

class HandWrite(torch.utils.data.Dataset):
    def __init__(self, files: list, word_to_id: dict, transform=None):
        self.files = files # 全部的資料
        self.transform = transform # 影像資料前處理的流程

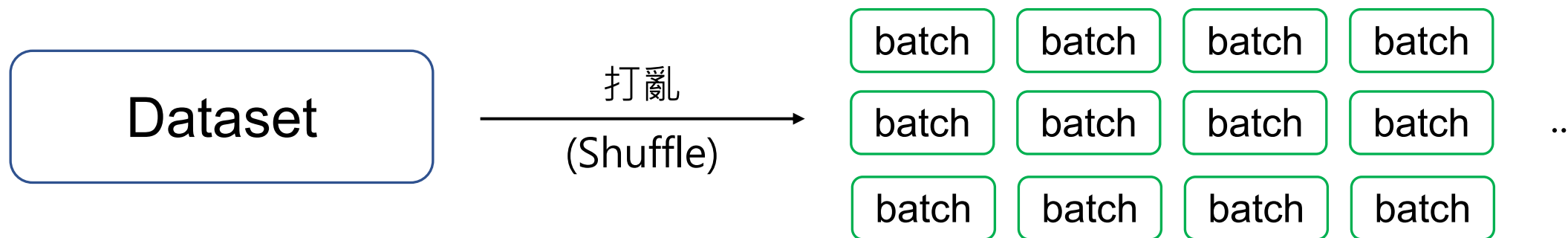
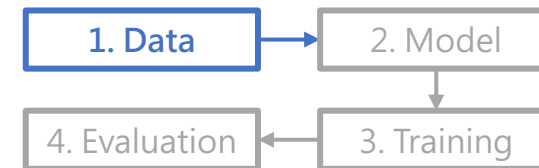
    def __getitem__(self, index):
        fname = self.files[index]
        image = Image.open(fname)
        if self.transform is not None:
            image = self.transform(image)

        label = fname.split('/')[-1].split('_')[0]
        return image, torch.tensor(word_to_id[label])

    def __len__(self):
        return len(self.files)
```



Step 1-3: 建立 DataLoader

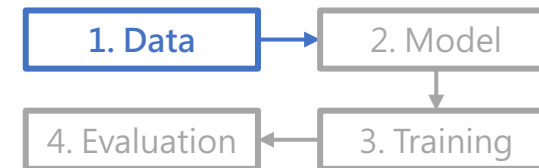


在batch size = k
時，每個batch有k筆
資料

```
# We should split the dataset into train / validation / test sets first.  
train_loader = torch.utils.data.DataLoader(trainset, batch_size=TRAIN_BS, shuffle=True)  
val_loader = torch.utils.data.DataLoader(valset, batch_size=VAL_BS, shuffle=False)  
test_loader = torch.utils.data.DataLoader(testset, batch_size=TEST_BS, shuffle=False)
```



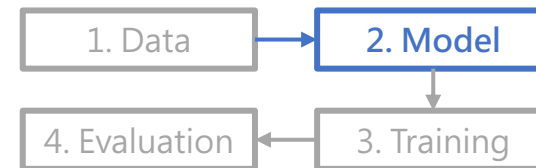
Advantages of batching



- Training:
 - mini-batch gradient descent 有機會避免模型陷入局部最小值
- Inference (validation or test):
 - 省記憶體
 - 不需要累積梯度，所以 inference 時期的 batch size (bs) 通常可以比 training 時期的 bs 還大



Step 2-1: Construct the model

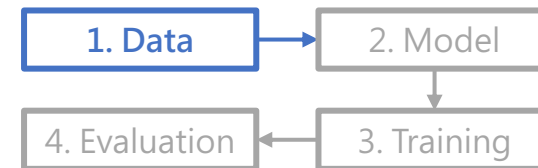


- 我們需要：
 1. 繼承 `torch.nn.Module` ,
 2. 初始化 `torch.nn.Module` 原本定義的內容
 3. 改寫兩個項目 (`__init__`, `forward`)

```
class MyModel(torch.nn.Module):  
    def __init__(self):  
        super().__init__() # 初始化 torch.nn.Module 原本定義的內容  
        # Define our new variables  
        # Define our model layers  
  
    def forward(self, x):  
        # Do something (forward pass)  
        return output
```



為什麼需要 `super().__init__()` ？

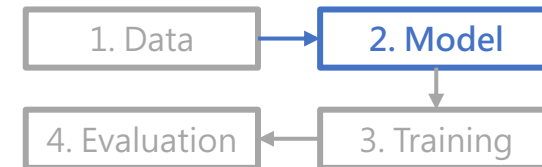


- 模型需要繼承 `torch.nn.Module`，並且透過 `super().__init__()` 初始化原本在 `nn.Module` 中被定義好的內容，如下圖所示：

```
206 ... def __init__(self):
207     """
208     Initializes internal Module state, shared by both nn.Module and ScriptModule.
209     """
210     torch._C._log_api_usage_once("python.nn_module")
211
212     self.training = True
213     self._parameters = OrderedDict()
214     self._buffers = OrderedDict()
215     self._non_persistent_buffers_set = set()
216     self._backward_hooks = OrderedDict()
217     self._forward_hooks = OrderedDict()
218     self._forward_pre_hooks = OrderedDict()
219     self._state_dict_hooks = OrderedDict()
220     self._load_state_dict_pre_hooks = OrderedDict()
221     self._modules = OrderedDict()
```



ImageNet Competition



Complete name: ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

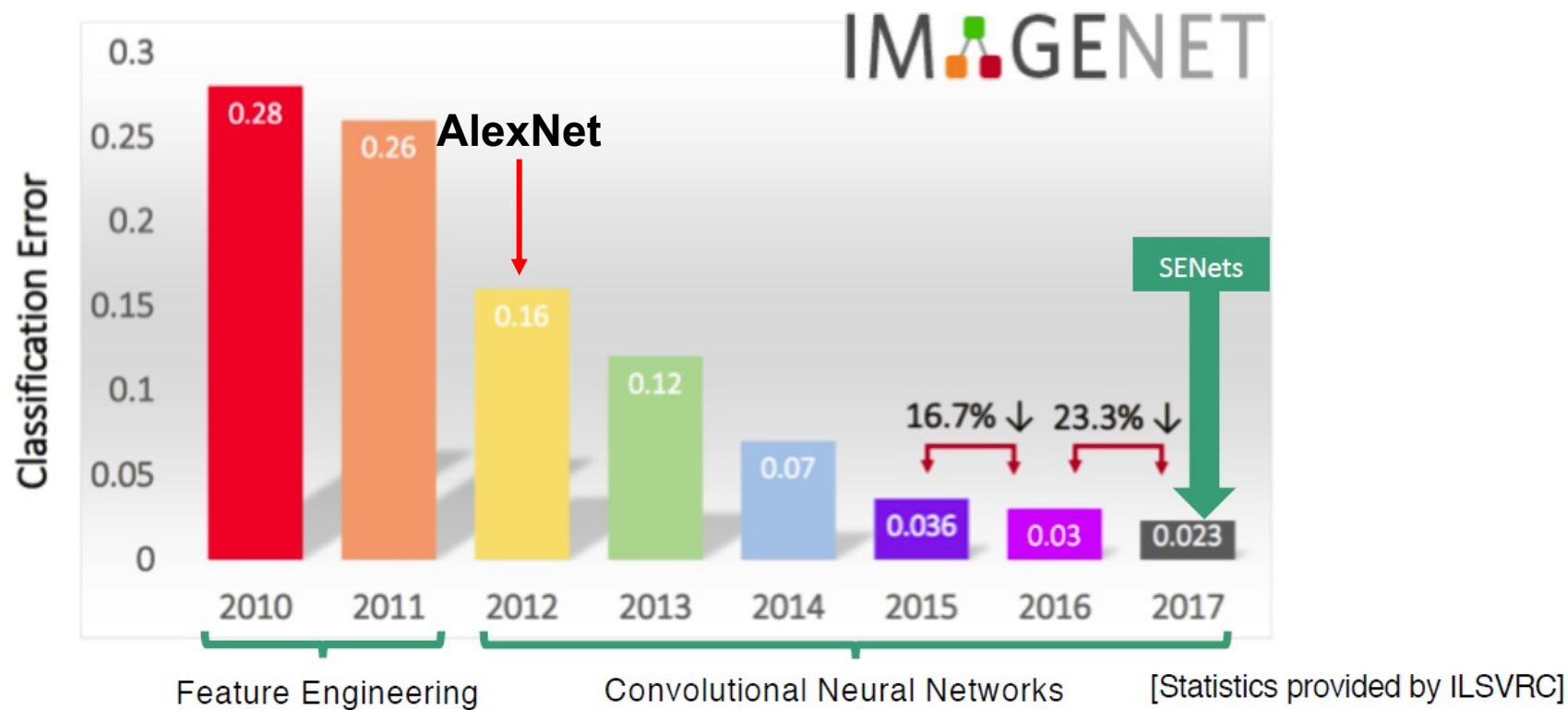
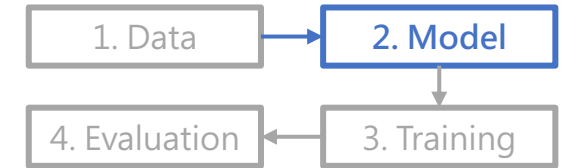


Figure source: <https://www.kaggle.com/discussions/getting-started/149448>



Step 2-2: Define the optimizer

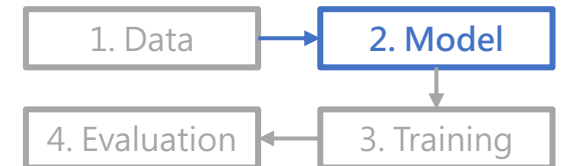


Loss functions	Meaning
torch.optim.SGD	Stochastic gradient descent (with momentum)
torch.optim.RMSprop	RMSProp (Root Mean Square Propagation)
torch.optim.Adam	Adam (Adaptive Moment Estimation)
torch.optim.AdamW	AdamW (Adam with decoupled weight decay)

```
learning_rate = 1e-3 # 代表 0.001
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```



Step 2-2: Define the loss function



Loss functions	Usage
torch.nn.CrossEntropyLoss	Classification
torch.nn.MSELoss	Regression
torch.nn.BCELoss	Binary classification

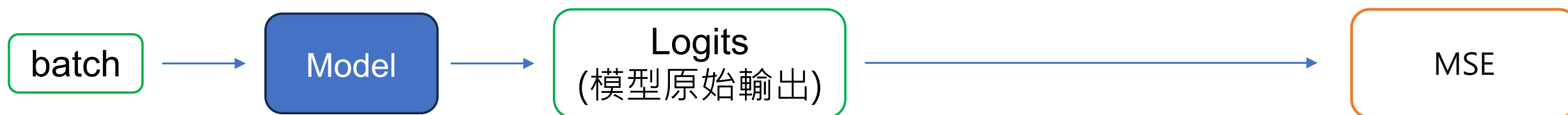
```
# example
loss_function = torch.nn.CrossEntropyLoss()
```

[PyTorch小細節]
BCEWithLogitsLoss 已經包含了 Sigmoid
CrossEntropyLoss 已經包含了 Softmax

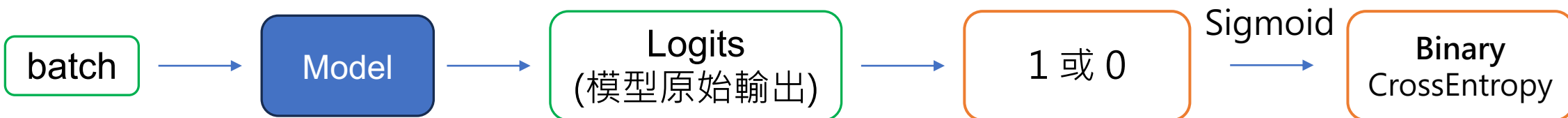


Step 2-2: 訓練過程中的模型輸出

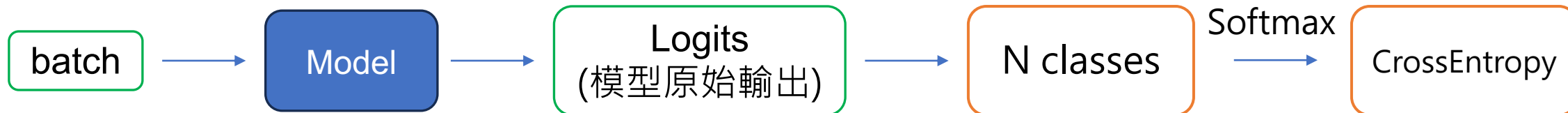
- 迴歸任務：



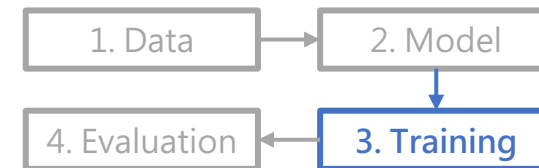
- 二分類任務：



- 多分類任務：



Step 3: 訓練模型



重複直到
訓練完成
或到達指
定條件

清空 optimizer (過去累積)的梯度

`optimizer.zero_grad()`

模型前向傳播得到輸出

`output = model(inputs)`

模型反向傳播計算梯度

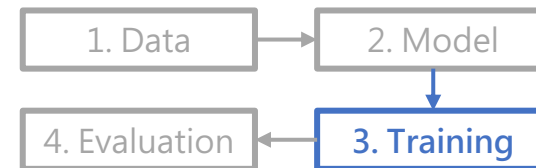
`loss = loss_function(output, target)`
`loss.backward()`

optimizer 更新模型參數

`optimizer.step()`



Step 3: 訓練模型



```
for batch in train_loader:
    # 把資料移動到 GPU
    images, labels = batch[0].to(device), batch[1].to(device)

    optimizer.zero_grad()

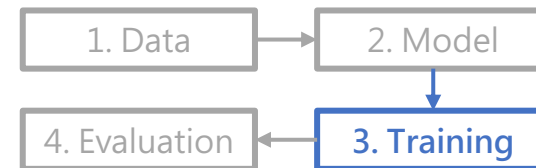
    # 1. 前向傳播
    outputs = model(images) # 形狀是 (batch_size, num_classes)
    loss = loss_fn(outputs, labels)

    # 2. 反向傳播 (計算梯度)
    loss.backward()

    # 3. 更新模型權重
    optimizer.step()
```



Step 3: 訓練模型



外層通常會有epoch (模型經過一次完整訓練集的更新稱為1個epoch)

```
for epoch in range(epochs):
    for batch in train_loader:
        # 把資料移動到 GPU
        images, labels = batch[0].to(device), batch[1].to(device)

        optimizer.zero_grad()

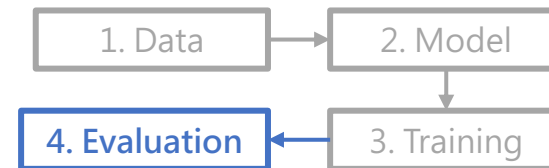
        # 1. 前向傳播
        outputs = model(inputs) # 形狀是 (batch_size, num_classes)
        loss = loss_fn(outputs, labels)

        # 2. 反向傳播 (計算梯度)
        loss.backward()

        # 3. 更新模型權重
        optimizer.step()
```



Step 4: 模型評估

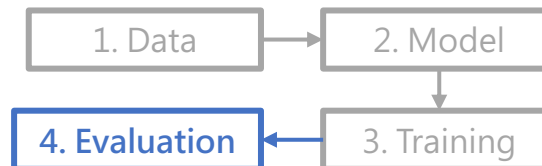


```
with torch.no_grad(): ← 在這底下縮排屬於不計算梯度的環境
    for batch in test_loader:
        images, labels = batch[0].to(device), batch[1].to(device)

        outputs = model(inputs) # 形狀是 (batch_size, num_classes)
        loss = loss_fn(outputs, labels) # 純粹紀錄數值用，沒有要更新模型
        total_loss += loss.item()
```



Step 4: 模型評估



```
with torch.no_grad():
    for batch in progress_bar:
        images, labels = batch[0].to(device), batch[1].to(device)
        outputs = model(images) # `outputs` 的形狀是 (batch_size, 10)
        loss = loss_fn(outputs, labels)
        total_loss += loss.item()

        prediction = outputs.argmax(dim=1) # 找出數值最大的類別作為模型預測

        # 我們希望 `predictions` 的長相是 [1, 2, 0, 4, 5, 7, ...]
        # 如果用 append 的話可能會變成 [[1, 2, 0], [4, 5, 7], ...]
        prediction_list.extend(prediction.tolist())
        label_list.extend(labels.tolist())

avg_loss = total_loss / len(data_loader)
accuracy = accuracy_score(label_list, prediction_list)
```

from sklearn.metrics import accuracy_score



Thank you!

Instructor: 林英嘉

 yjlin@cgu.edu.tw

TA: 劉美辰

 m1461014@cgu.edu.tw