自然語言處理與應用
Natural Language Processing and Applications

PyTorch Modeling

Instructor: 林英嘉 (Ying-Jia Lin)
2025/03/17

Course GitHub

Slido # NLP0317

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

Step 1 (Data)

- 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
1 891	0 1	1 3	891 unique values	male 65% female 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

https://www.kaggle.com/competitions/titanic/data?select=train.csv

Step 1-2: Overwrite 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: Overwrite PyTorch Dataset

Step 1 (Data)

- 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

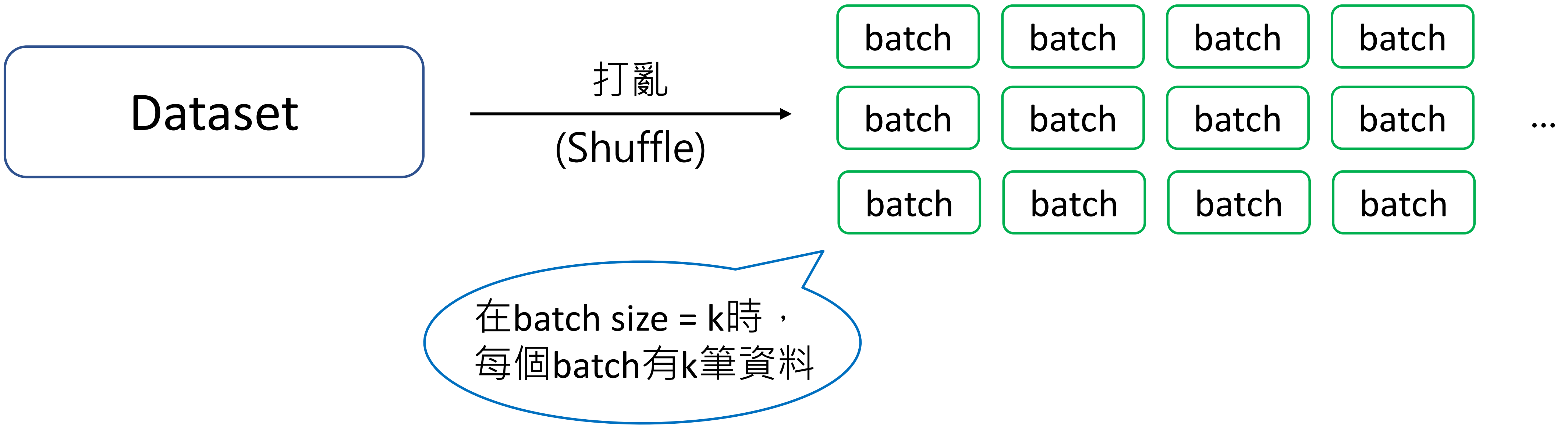
Step 1-2: Overwrite PyTorch Dataset

Step 1 (Data)

```
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: Define DataLoader

Step 1 (Data)




```
# 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

Step 2 (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
```

https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html

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：

```python
    def __init__(self):
        """
        Initializes internal Module state, shared by both nn.Module and ScriptModule.
        """
        torch._C._log_api_usage_once("python.nn_module")

        self.training = True
        self._parameters = OrderedDict()
        self._buffers = OrderedDict()
        self._non_persistent_buffers_set = set()
        self._backward_hooks = OrderedDict()
        self._forward_hooks = OrderedDict()
        self._forward_pre_hooks = OrderedDict()
        self._state_dict_hooks = OrderedDict()
        self._load_state_dict_pre_hooks = OrderedDict()
        self._modules = OrderedDict()
```

https://github.com/pytorch/pytorch/blob/266657182a4040937f6f27011d7ddf77716db83d/torch/nn/modules/module.py#L206-L221

Step 2-2: Define the loss function

Step 2 (Model)

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

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

模型輸出的後處理

Cross-entropy: $\mathcal{L}_i = -\log P(Y = y_i | X = x_i)$

→ Model →

	Unnormalized log-probabilities / logits	Unnormalized probabilities	Probabilities
Cat	0.5	1.6487	0.225
Dog	0.7	2.0138	0.275
Apple	1.3	3.6693	0.500
	Model output	Exponential	Softmax

Cross-entropy (交叉熵)

Cross-entropy: $\mathcal{L}_i = -\log P(Y = y_i | X = x_i)$

其中 i 代表第 i 筆資料

交叉」(Cross) 代表的是 兩個機率分布之間的關係，特別是用一個分布來衡量 與另一個分布的相似程度

- 量測模型輸出的負對數機率，代表模型預測該類別的信心程度
  - 模型預測該類別的信心程度越大時， $\mathcal{L}_i$ 就會越小
  - 模型預測該類別的信心程度越小時， $\mathcal{L}_i$ 就會越大

$P(Y = y_i \| X = x_i)$	$\mathcal{L}_i = -\log P(Y = y_i \| X = x_i)$
0.9	−log0.9 ≈ 0.105
0.1	−log0.1 ≈ 2.302

Softmax (Non-linear Transformation)

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

$$y = \frac{\mathrm{x}}{\sum_{\mathrm{j}} \mathrm{x_j}} = [0.5,0.25,0.25]$$

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

NLP

Step 2-3: Define the optimizer

Step 2 (Model)

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 3: Write the training process

Step 3 (Training)

1. Clear gradients `optimizer.zero_grad()`
2. Input data to the model `output = model(**batch)`
3. Computer loss `loss = loss_function(gold, output)`
4. Computer gradients `loss.backward()`
5. Update model parameters `optimizer.step()`
6. (Repeat 1. to 5. until the end of training)

** in Python

- **dict 可以展開字典，轉換成關鍵字參數傳遞給函式

```
def greet(age, name):
    print(f"My name is {name} and I am {age} years old.")

person_info = {"name": "Alex", "age": 25}
greet(**person_info)  # 等同於 greet(name="Alex", age=25)
```

Step 3: Write the training process

Step 3 (Training)

1. Clear gradients `optimizer.zero_grad()`
2. Input data to the model `output = model(**batch)`
3. Computer loss `loss = loss_function(gold, output)`
4. Computer gradients `loss.backward()`
5. Update model parameters `optimizer.step()`
6. (Repeat 1. to 5. until the end of training)

Step 3: Write the training process

Step 3 (Training)

```
for batch in train_loader:
    output = model(**batch)
    ...
```

```
output = model(**batch)
```

```
# Get x, y from your dataloader
for batch_x, batch_y in train_loader:
    output = model(batch_x)
    loss = criterion(output, target)
    ...
```

Step 4: Write the evaluation process

Step 4 (Evaluation)

```
from sklearn.metrics import accuracy_score

with torch.no_grad():
    for batch in val_loader: # or test_loader
        output = model(**batch)
        pred = outputs.argmax(dim=1)
        ...
        predictions.append(pred)


accuracy_score(test_labels, predictions)
```

NLP

Thank you!

Instructor: 林英嘉

✉️ yjlin@cgu.edu.tw

TA: 吳宣毅

✉️ m1161007@cgu.edu.tw