

PyTorch 1.0.1 教學

Presenter: Hao-Ting Li (李皓庭)

What/Why is PyTorch?

- Wiki:
 - PyTorch is an open-source machine learning library for Python
- 優點：
 - Python-based
 - 動態計算圖 (Dynamic Computational Graph, DCG) 中斷點
 - 簡潔優雅的 API
 - 豐富的文件
 - 社群發達，更新速度快（每三個月釋出一次正式版）
- 缺點：

比較其他框架

比較項目	PyTorch	Tensorflow	Keras	Caffe	說明
API 抽象層次	低	低	高	低	Keras 為 Tensorflow 等多個框架的高階封裝 API
入門難度	中偏簡單	困難	簡單	困難	Tensorflow 超難用
自訂模型靈活度	高	中	低	低	PyTorch/Tensorflow 內建低階 API 來設計，Caffe 要修改 source code
除錯難度	簡單	困難	困難	困難	PyTorch 支援 DCG 可從程式碼任何一處取得計算結果
文件數量	超多	中等	很多	超少	PyTorch 幾乎所有功能都可以找到詳細的說明和範例

大綱

- 安裝
- 基本資料儲存類別
- 資料處理
- 定義模型
- 訓練與測試

大綱

- 安裝
- 基本資料儲存類別
- 資料處理
- 定義模型
- 訓練與測試

安裝

- 連結：<https://pytorch.org/>
- 建議使用 Conda 套件管理工具
 - Anaconda
 - Miniconda (推薦)
- Prerequisite
 - NVIDIA graphics card
 - CUDA
 - 需要配合指定版本
 - Linux 上尋找 CUDA 版本的指令：
`find /usr/local -maxdepth 1 -type d -name 'cuda*'`
 - 範例：

```
maniac@gslave01[03:53:37]~$ find /usr/local -maxdepth 1 -type d -name 'cuda*'  
/usr/local/cuda-9.0  
/usr/local/cuda-8.0
```

安裝

PyTorch Build	Stable (1.0)		Preview (Nightly)	
Your OS	Linux	Mac		Windows
Package	Conda	Pip	LibTorch	Source
Language	Python 2.7	Python 3.5	Python 3.6	Python 3.7
CUDA	8.0	9.0	10.0	None
Run this Command:	<code>conda install pytorch torchvision cudatoolkit=9.0 -c pytorch</code>			

測試是否使用 GPU

- 進入 Python Interactive Shell
- 輸入
`import torch`
`torch.cuda.is_available()`
- 顯示為 True 代表安裝成功，可以使用 GPU 加速
- 範例：

```
(pytorch-1.0) maniac@kurisu[04:01:01]~$ python
Python 3.7.2 (default, Dec 29 2018, 06:19:36)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.cuda.is_available()
True
>>>
```


大綱

- 安裝
- 基本資料儲存類別
- 資料處理
- 定義模型
- 訓練與測試

基本資料儲存類別：Tensor

- Tensor：張量，概念上等同於**多維陣列**
- 用 **shape** 表達每個 dimension 的個數
- 例子：
 - 1 張 255×255 的灰階影像
可以用一個 shape 為 (255, 255) 的 tensor 儲存
 - 1 張 255×255 的 RGB 彩色影像
可以用一個 shape 為 (3, 255, 255) 的 tensor 儲存
其中 3 稱為 **channel**
 - 87 張 255×255 的 RGB 彩色影像
可以用一個 shape 為 (87, 3, 255, 255) 的 tensor 儲存
 - 同理：1 張 255×255 的 RGB 彩色影像
也可以用一個 shape 為 (1, 3, 255, 255) 的 tensor 儲存
- 圖片讀入 Tensor 後的擺放順序：
 - (batch size, channels, height, width)

torch.Tensor

- 作用幾乎等同於 NumPy 的 `np.ndarray`
 - 支援 GPU 運算
 - 支援梯度 (gradient) 運算
 - 可以和 `np.ndarray` 互相轉換格式
 - 可以用 Python 內建的 `list` 來建構 (construct)
- 範例

```
>>> import torch
>>> x = torch.tensor([[5, 4], [8, 7]])
>>> x.shape
torch.Size([2, 2])
```

torch.Tensor 數學運算範例

- $f(x, y) = x^2 + 2y$

```
>>> def f(x, y):  
...     return x.pow(2) + 2*y  
...
```

- set $x = 8, y = 7$

```
>>> x = torch.tensor([8.])  
>>> y = torch.tensor([7.])  
>>> f(x, y)  
tensor([78.])
```

Autograd: Automatic Differentiation

- $f(x, y) = x^2 + 2y$
- $x = 8, y = 7$
- 變數需要計算梯度時必須設定變數 `requires_grad=True`
- 範例

```
>>> x = torch.tensor([8.], requires_grad=True)
>>> y = torch.tensor([7.])
>>> y.requires_grad_()
```

Autograd: Automatic Differentiation

- $f(x, y) = x^2 + 2y$
- $x = 8, y = 7$
- 計算梯度 $\nabla f(x, y) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = (2x, 2)$
- 範例

梯度：在小範圍內值的變化量



$$ax + by + c = 0$$

每次挑整看error下降，調整方向能讓error下降最多方向，梯度

```
>>> f(x, y)
tensor([78.], grad_fn=<AddBackward0>)
>>> f(x, y).backward()
>>> x
tensor([8.], requires_grad=True)
>>> x.grad
tensor([16.])
>>> y.grad
tensor([2.])
```

利用 chain rule 計算梯度

In-place operation

- 所有結尾加上 _ (底線) 的運算都是 **in-place operation**
- 範例

直接原地運算

```
>>> x.add(y)
tensor([15.])
>>> x
tensor([8.])
>>> x.add_(y)
tensor([15.])
>>> x
tensor([15.])
```

使用 CPU/GPU 運算

- 預設為 CPU **CPU轉GPU**

```
>>> x.cuda(1)
tensor([8.], device='cuda:1')

>>> x.to('cuda:1')
tensor([8.], device='cuda:1')

>>> device = torch.device('cuda:1')
>>> x.to(device)
tensor([8.], device='cuda:1')
```


大綱

- 安裝
- 基本資料儲存類別
- 資料處理
- 定義模型
- 訓練與測試

重要的資料類別：Dataset & Dataloader

- `torch.utils.data.Dataset`
- `torch.utils.data.DataLoader`

torch.utils.data.Dataset

- `torch.utils.data.Dataset` :
一個用來與 `torch.utils.data.DataLoader` 溝通的介面
- 你需要寫一個自己的 Dataset Class
並繼承 `torch.utils.data.Dataset`
- 這個 Class 必須實作兩個 method :
 - `def __getitem__(self, index)`
 - 回傳值為每次取出一個 batch 的資料
 - `def __len__(self)`
 - 回傳值為整個 dataset 的大小

torch.utils.data.DataLoader

- `torch.utils.data.DataLoader` :
定義好自己的 Dataset Class 以後，就可以直接傳入
- 傳入的參數：
`torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False, sampler=None, batch_sampler=None, num_workers=0...)`
Mutiprocess

大綱

- 安裝
- 基本資料儲存類別
- 資料處理
- 定義模型
- 訓練與測試

神經網路的基礎類別：torch.nn.Module

- **torch.nn.Module**：
功能強大的神經網路類別，方便用來
 - 定義與命名子模組
 - 初始化參數
 - 定義整個神經網路的資料流計算
- 用法：自己寫一個 Class 繼承它，必須實作一個 method：
 - **def forward(self, x)**
 - 定義 input data x 傳入以後會如何經過你自己設定的模組，並計算出 output

torch.nn.Module 範例

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Loss Function

- Loss function

- `torch.nn.*Loss` 模型預測與真實差別
 - `torch.nn.L1Loss` 95-100=-5全加起來可能為0
 - `torch.nn.MSELoss` 所以平方或絕對值
 - ... Ref: <https://pytorch.org/docs/stable/nn.html#loss-functions>
- `torch.nn.functional.*`
 - `torch.nn.functional.binary_cross_entropy`
 - `torch.nn.functional.binary_cross_entropy_with_logits`
 - ... Ref: <https://pytorch.org/docs/stable/nn.html#id51>
- 自己定義的數學式 (進階)
 - 需要注意是否可以計算 `gradient` , `loss function` 必須可微
 - 如果使用了不屬於 `torch.Tensor` 內的數學運算 , 必須自己實作 `.backward()`

Optimizer: torch.optim

讓loss變小

- Optimizer: optimization algorithm for loss function
- Example:

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)  
optimizer = optim.Adam([var1, var2], lr=0.0001)
```

大綱

- 安裝
- 基本資料儲存類別
- 資料處理
- 定義模型
- 訓練與測試

訓練

- 步驟：
 1. 從 dataloader 讀取資料
 2. Forward: 將資料傳入神經網路以後得到 output
 3. Backward: 對 loss function 與所有參數計算梯度並且執行 backpropagation
 4. 重複以上直到收斂或是設定最大 epoch/iteration 數量

範例

CPU/GPU

```
def train(model, device, train_loader, optimizer, epoch):  
    model.train()  
    for batch_idx, (inputs, targets) in train_loader:  
        inputs, targets = inputs.to(device), targets.to(device) # fetch data 轉轉GPU  
        optimizer.zero_grad() 初始化, optimizer  
  
        # forward  
        ↓  
        outputs = model(inputs)  
        loss = F.cross_entropy(outputs, targets)  
  
        # backward  
        loss.backward()  
        optimizer.step() 更新參數
```

測試

- 和訓練步驟幾乎相同，只差在不需要計算 `.backward()`

```
def test(model, device, test_loader):  
    model.eval() 相對於.train()  
    correct = 0  
    with torch.no_grad(): 不用計算gradient  
        for inputs, target in test_loader:  
            inputs, target = inputs.to(device), target.to(device)  
            outputs = model(inputs)  
            pred = outputs.argmax(dim=1, keepdim=True)  
            correct += pred.eq(target.view_as(pred)).sum().item()
```

Q&A