PyTorch cheat sheet

— 陳建成 Wed, Mar 25, 2020 7:57 PM

此處整理 PyTorch 常用的 modules 和 functions 方便快速查詢。
完整且詳細的 docs 請見 PyTorch 官方文檔（ver 1.2.0） (https://pytorch.org/docs/1.2.0/)。

另外，這裡有兩個版本的 PyTorch 教學 Colaboratory Notebooks，一個 (https://colab.research.google.com/drive/14xSEVRGOPLYNVGfXTnc-vNc1yTp05aDf)和上課教學互相對應，另一個 (https://colab.research.google.com/drive/1PXpKHuETM-xgTatmHpSeZysXkZaXXKja)有更詳細的解說（包含前後處理、視覺化、常用工具等），提供各位參考。

> 對了，拜託各位先 copy 一份到自己雲端硬碟，或是用 Playground 模式，不要干擾到原本的版本，多謝各位合作 😃

▶ 在我們開始前……（表示方法說明）

Tensor Operations [Docs] (https://pytorch.org/docs/1.2.0/tensors.html)

```
torch
├── (Tensor)
│   ├── view(*shape)      # e.g. x.view(-1, 3, 12)
│   │                     ##  -1 automatically filled
│   └── item()            # get if Tensor is a scalar
│
├── empty(*size)          # e.g. x = torch.empty(2, 3)
├── stack(tensors, dim=0)
└── cat(tensors, dim=0)
```

Data Preparation [Docs] (https://pytorch.org/docs/1.2.0/data.html)

```
torch
└── utils
    └── data
        ├── Dataset      # A class to override
        │                ##  `__len__` & `__getitem__`
        ├── TensorDataset(data_tensor, target_tensor)
        ├── DataLoader(dataset, batch_size=1,
        │             shuffle=False,
        │             collate_fn=\
        │                 <function default_collate>)
        │              # define `collate_fn` yourself
        └── sampler
            ├── SequentialSampler(data_source)
            └── RandomSampler(data_source)
```

NN (Neural Network) Model Construction [Docs] (https://pytorch.org/docs/1.2.0/nn.html)

這是 PyTorch 最主要的 module，docs 比較複雜，分成

- `torch.nn` (https://pytorch.org/docs/1.2.0/nn.html)
- `torch.nn.functional` (https://pytorch.org/docs/1.2.0/nn.functional.html)
- `torch.nn.init` (https://pytorch.org/docs/1.2.0/nn.init.html)
- `torch.optim` (https://pytorch.org/docs/1.2.0/optim.html)
- `torch.autograd` (https://pytorch.org/docs/1.2.0/autograd.html)

Training

```
torch
├── (Tensor)
│   ├── backward()
│   │
│   ├── cpu()
│   ├── cuda()
│   └── to(torch.device)            # x = x.to(device)
├── cuda
│   └── is_available()
│       #  if torch.cuda.is_available():
│       ##     device = "cuda"
│       ## else: device = "cpu"
│
├── nn as nn
│   │### Models ###
│   ├── Module
│   │   ├── load_state_dict(torch.load(PATH))
│   │   ├── train()
│   │   └── eval()
│   ├── Sequential(layers)
│   │
│   │### Initializations ###
│   ├── init
│   │   └── uniform_(w)     # In-place,
│   │                       ##  w is a `torch.Tensor`.
│   │
│   │### Layers ###
│   ├── Linear(in_feat, out_feat)
│   ├── Dropout(rate)
│   │
│   │### Activations ###
│   ├── Softmax(dim=None)
│   ├── Sigmoid()
│   ├── ReLU()
│   ├── LeakyReLU(negative_slope=0.01)
│   ├── Tanh()
│   ├── GELU()
│   ├── ReLU6() # Model Compression
│   │ # --> Corresponding functions
│   ├── functional as F  ────────────────────────────┐
│   │   ├── softmax(input, dim=None)                 │
│   │   ├── sigmoid(input)                           │
│   │   ├── relu(input)                              │
│   │   ├── leaky_relu(input,                        │
│   │   │              negative_slope=0.01)          │
│   │   ├── tanh(input)                              │
│   │   ├── gelu(input)                              │
│   │   └── relu6(input)                             │
│   │                                                │
│   │### Losses ###                                  │
│   ├── MSELoss()                                    │
│   ├── CrossEntropyLoss()                           │
│   ├── BCELoss()                                    │
│   ├── NLLLoss()                                    │
│   │ # --> Corresponding functions                  │
│   └──<functional as F> <───────────────────────────┘
│       ├── mse_loss(input, target)
│       ├── cross_entropy(input,
│       │                 target: torch.LongTensor)
│       ├── binary_cross_entropy(input, target)
│       ├── log_softmax(input)
│       └── nll_loss(log_softmax_output, target)
│           # F.nll_loss(F.log_softmax(input), target)
│
│    ### Optimizers ###
├── optim
│   ├── (Optimizer)
│   │       ├── zero_grad()
│   │       ├── step()
│   │       └── state_dict()
│   │
│   ├── SGD(model.parameters(), lr=0.1, momentum=0.9)
│   ├── Adagrad(model.parameters(), lr=0.01,
│   │           lr_decay=0, weight_decay=0,
│   │           initial_accumulator_value=0,eps=1e-10)
│   ├── RMSProp(model.parameters(), lr=0.01,
│   │           alpha=0.99, eps=1e-08, weight_decay=0,
│   │           momentum=0)
│   ├── Adam(model.parameters(), lr=0.001,
│   │        betas=(0.9, 0.999), eps=1e-08,
│   │        weight_decay=0)
│   │
│   └── lr_scheduler
│       └── ReduceLROnPlateau(optimizer)
│
|── load(PATH)
|── save(model, PATH)
│
└── autograd
    └── backward(tensors)
```

Testing

```
torch
├── nn
│   └── Module
│       ├── load_state_dict(torch.load(PATH))
│       └── eval()
├── optim
│   └── (Optimizer)
│       └── state_dict()
└── no_grad()              # with torch.no_grad(): ...
```

CNN (Convolutional Neural Networks)

- Convolutional Layers (https://pytorch.org/docs/1.2.0/nn.html#conv2d)
- Pooling Layers (https://pytorch.org/docs/1.2.0/nn.html#maxpool2d)
- `torchvision` docs (https://pytorch.org/docs/stable/torchvision/index.html)

```
torch
├── (Tensor)
│   └── view(*shape)
├── nn
│   │### Layers ###
│   ├── Conv2d(in_channels, out_channels,
│   │          kernel_size, stride=1, padding=0)
│   ├── ConvTranspose2d(in_channels, out_channels,
│   │          kernel_size, stride=1, padding=0,
│   │          output_padding=0)
│   ├── MaxPool2d(kernel_size, stride=None,
│   │             padding=0, dilation=1)
│   │             # stride default: kernel_size
│   ├── BatchNorm2d(num_feat)
│   └── BatchNorm1d(num_feat)
├── stack(tensors, dim=0)
└── cat(tensors, dim=0)

torchvision
├── models as models # Useful pretrained
├── transforms as transforms
│   ├── Compose(transforms) # Wrapper
│   ├── ToPILImage(mode=None)
│   ├── RandomHorizontalFlip(p=0.5)
│   ├── RandomRotation(degrees)
│   ├── ToTensor()
│   └── Resize(size)
└── utils
    ├── make_grid(tensor, nrow=8, padding=2)
    └── save_image(tensor, filename, nrow=8,padding=2)
```

RNN (Recurrent Neural Networks)

- Recurrent Layers (https://pytorch.org/docs/1.2.0/nn.html#recurrent-layers)
- Gensim Word2Vec Docs (https://radimrehurek.com/gensim/models/word2vec.html)

```
torch
├── nn
│   ├── Embedding(num_embed, embed_dim)
│   │   # embedding = nn.Embedding(
│   │   ##               *(w2vmodel.wv.vectors.shape))
│   ├── Parameter(params: torch.FloatTensor)
│   │   # embedding.weight = nn.Parameter(
│   │   ##  torch.FloatTensor(w2vmodel.wv.vectors))
│   ├── LongTensor          # Feeding Indices of words
│   │
│   ├── LSTM(inp_size, hid_size, num_layers)
│   │   # input: input, (h_0, c_0)
│   └── GRU(inp_size, hid_size, num_layers)
├── stack(tensors, dim=0)
└── cat(tensors, dim=0)

gensim
└── models
    └── word2Vec
        └── Word2Vec(sentences) # list or words/tokens
```

Change Log

全部的架構太大，不方便查詢，故先隱藏起來
— 陳建成 Wed, Mar 25, 2020 7:57 PM

▼ PyTorch 套件常用部分完整架構

```
torch
├── (Tensor)
│   ├── view
│   ├── item
│   ├── cpu()
│   ├── cuda()
│   ├── to(torch.device)
│   └── backward
├── nn
│   ├── Module
│   │   ├── load_state_dict
│   │   ├── train
│   │   └── eval
│   ├── Sequential
│   │ # Layers
│   ├── Linear
│   ├── Dropout
│   │ ## CNN
│   ├── Conv2d
│   ├── ConvTranspose2d
│   ├── MaxPool2d
│   ├── BatchNorm2d
│   ├── BatchNorm1d # GAN
│   │ ## RNN
│   ├── Embedding
│   ├── LSTM
│   ├── GRU
│   │ # Loss functions
│   ├── MSELoss
│   ├── CrossEntropyLoss
│   ├── BCELoss
│   │ # Activations
│   ├── Sigmoid
│   ├── ReLU
│   ├── Tanh
│   ├── ReLU6 # Network Compression
│   │ # Initializations
│   ├── init
│   │   └── uniform_
│   │
│   ├── functional as F
│   │   ├── relu
│   │   ├── leakyrelu
│   │   ├── gelu
│   │   └── nll_loss
│   │
│   └── Parameter
├── optim
│   ├── SGD
│   ├── RMSProp
│   ├── Adagrad
│   ├── Adam
│   ├── AdamW
│   ├── lr_scheduler
│   └── (Optimizer)
│       ├── zero_grad
│       ├── state_dict
│       └── step
├── utils
│   └── data
│       ├── Dataset
│       ├── TensorDataset
│       ├── DataLoader
│       ├── sampler
│       ├── SequentialSampler
│       └── RandomSampler
├── cuda
│   └── is_available
├── autograd
│   └── backward
│   # tensor operation
├── no_grad
├── empty
├── stack
├── cat
│   # model save
├── load
└── save

torchvision
├── transforms
│   ├── Compose
│   ├── ToPILImage
│   ├── RandomHorizontalFlip
│   ├── RandomRotation
│   ├── ToTensor
│   └── Resize
├── models
└── utils
    ├── make_grid
    └── save_image
```