Tutorial I: PyTorch开发环境搭建及使用

崔雪建

2021.11.17

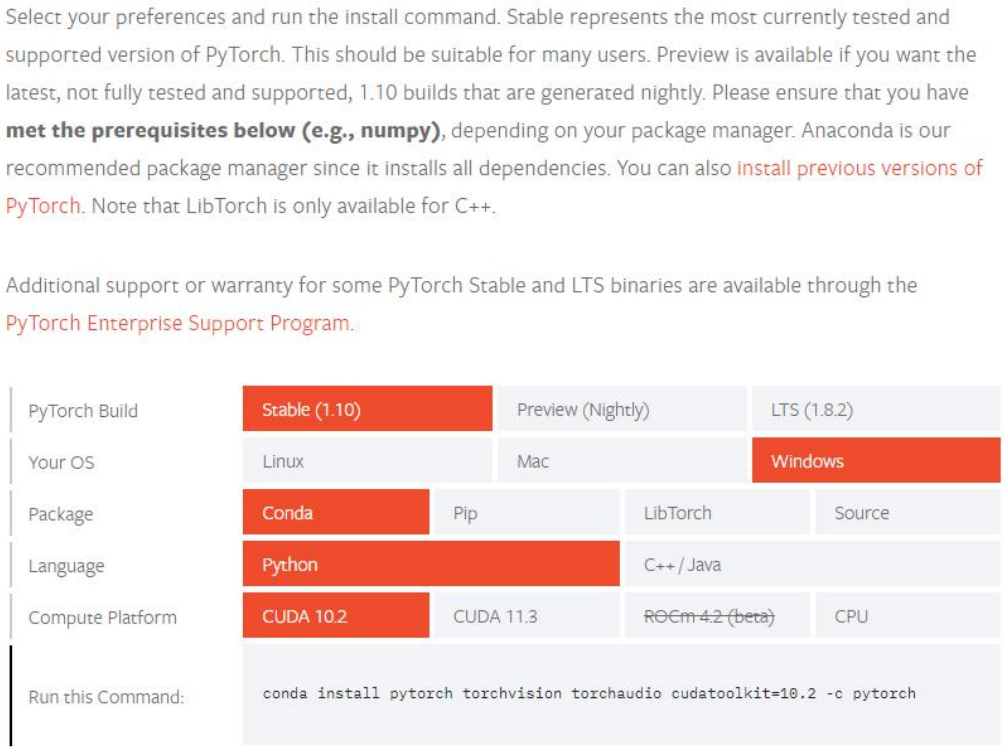
0. 准备

- 完成Anaconda安装
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/
- 完成Jupyter Notebook安装
  - (conda) activate env_name
  - python -m pip install ipykernel
  - Python -m ipykernel install --user --name env_name

1. 什么是PyTorch?

- PyTorch是Torch7团队开发的，从它的名字就可以看出，其与Torch的不同之处在于PyTorch使用了Python作为开发语言。所谓“Python first”，同样说明它是一个以**Python优先的深度学习框架**，不仅能够实现强大的**GPU加速**，同时还支持**动态神经网络**。PyTorch既可以看做加入了GPU支持的numpy，同时也可以看成一个拥有自动求导功能的强大的深度神经网络，除了Facebook之外，它还已经被Twitter、CMU和Salesforce等机构采用。

1. PyTorch的安装



- https://pytorch.org
- conda install pytorch==1.6.0 torchvision==0.7.0 -c pytorch

1. PyTorch检查

- 命令行输入python
  - import torch
  - torch.__version__

```
In  [2]: torch.__version__
Out[2]: '1.6.0'
```

2. PyTorch - demo结构

- Pytorch
  - Data准备 + Dataloader(train / test)
  - Model
  - Loss
  - Optim
  - Metric
  - Train
  - Test

2.1 Dataloader

CLASS `torch.utils.data.Dataset(*args, **kwds)` [SOURCE]

An abstract class representing a `Dataset`.

All datasets that represent a map from keys to data samples should subclass it. All subclasses should overwrite `__getitem__()`, supporting fetching a data sample for a given key. Subclasses could also optionally overwrite `__len__()`, which is expected to return the size of the dataset by many `Sampler` implementations and the default options of `DataLoader`.

• NOTE

`DataLoader` by default constructs a index sampler that yields integral indices. To make it work with a map-style dataset with non-integral indices/keys, a custom sampler must be provided.

CLASS `torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False, sampler=None, batch_sampler=None, num_workers=0, collate_fn=None, pin_memory=False, drop_last=False, timeout=0, worker_init_fn=None, multiprocessing_context=None, generator=None, *, prefetch_factor=2, persistent_workers=False)` [SOURCE]

Data loader. Combines a dataset and a sampler, and provides an iterable over the given dataset.

The `DataLoader` supports both map-style and iterable-style datasets with single- or multi-process loading, customizing loading order and optional automatic batching (collation) and memory pinning.

2.1 Dataloader

- **dataset** (*Dataset*) – dataset from which to load the data.
- **batch_size** (*int, optional*) – how many samples per batch to load (default: `1`).
- **shuffle** (*bool, optional*) – set to `True` to have the data reshuffled at every epoch (default: `False`).
- **sampler** (*Sampler or Iterable, optional*) – defines the strategy to draw samples from the dataset. Can be any `Iterable` with `__len__` implemented. If specified, `shuffle` must not be specified.
- **batch_sampler** (*Sampler or Iterable, optional*) – like `sampler`, but returns a batch of indices at a time. Mutually exclusive with `batch_size`, `shuffle`, `sampler`, and `drop_last`.
- **num_workers** (*int, optional*) – how many subprocesses to use for data loading. `0` means that the data will be loaded in the main process. (default: `0`)
- **collate_fn** (*callable, optional*) – merges a list of samples to form a mini-batch of Tensor(s). Used when using batched loading from a map-style dataset.
- **pin_memory** (*bool, optional*) – If `True`, the data loader will copy Tensors into CUDA pinned memory before returning them. If your data elements are a custom type, or your `collate_fn` returns a batch that is a custom type, see the example below.
- **drop_last** (*bool, optional*) – set to `True` to drop the last incomplete batch, if the dataset size is not divisible by the batch size. If `False` and the size of dataset is not divisible by the batch size, then the last batch will be smaller. (default: `False`)
- **timeout** (*numeric, optional*) – if positive, the timeout value for collecting a batch from workers. Should always be non-negative. (default: `0`)
- **worker_init_fn** (*callable, optional*) – If not `None`, this will be called on each worker subprocess with the worker id (an int in `[0, num_workers - 1]`) as input, after seeding and before data loading. (default: `None`)
- **generator** (*torch.Generator, optional*) – If not `None`, this RNG will be used by RandomSampler to generate random indexes and multiprocessing to generate *base_seed* for workers. (default: `None`)
- **prefetch_factor** (*int, optional, keyword-only arg*) – Number of samples loaded in advance by each worker. `2` means there will be a total of 2 * num_workers samples prefetched across all workers. (default: `2`)
- **persistent_workers** (*bool, optional*) – If `True`, the data loader will not shutdown the worker processes after a dataset has been consumed once. This allows to maintain the workers *Dataset* instances alive. (default: `False`)

2.1 Dataloader

- dataset输入**(data, label)**

  自定义Dataset: __init__, __len__, __getitem__
- dataloader最后输出的图像数据维度是

  [batch_size, channel, height, width]
- 图像分类对应的标签格式一般为

  标量或**one-hot向量标量**

2.2. Loss

L1LOSS

CLASS `torch.nn.L1Loss(size_average=None, reduce=None, reduction='mean')` [SOURCE]

CROSSENTROPYLOSS

CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0)` [SOURCE]

NLLLOSS

CLASS `torch.nn.NLLLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean')` [SOURCE]

2.3. Optim

SGD

CLASS `torch.optim.SGD(params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0, nesterov=False)` [SOURCE]

ADAM

CLASS `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)` [SOURCE]

- optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
- optimizer = optim.Adam(model.parameters(), lr=1e-4)

2.4 Metric

- from sklearn.metrics import f1_score

- from sklearn.metrics import f1_score
- y_true = [0, 1, 2, 0, 1, 2]
- y_pred = [0, 2, 1, 0, 0, 1]
- f1_score(y_true, y_pred, average='macro')
- 0.26...

2.5. Model

```
class Model(nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x
```

2.6. Train

- 训练过程中loss突然变大？
  - **减小优化器学习率**
- 内存炸了？
  - **减小batch_size**
- 网络报错？
  - **print(X.size())查看每层大小调整**
- 精度不高？
  - **数据准备错误或Metric错误或网络过小**

2.7. Test

- 请勿shuffle数据