前期准备

```
PS:本次作业的项目级别并不是很大，很好地起到了课程回顾总结的作用。但是为了之后论文复现级别的代码量考虑，我仍
使用了一些论文代码常用的编程技巧作为练习（平时看论文代码只是看，这次很好的让我有了一次通过自己实现一些简单
技术的机会），前期准备就是将我目前看到的论文代码实现的技巧（不是模型方面，更多关注数据处理）总结一下，应该
日后还会不断更新。
```

```
首先列出几个经常参考的网站吧：
python: https://www.runoob.com/python/python-tutorial.html
pytorch: https://pytorch.org/docs/stable/torch.html
```

python parser

一、介绍

`argparse` 模块可以让人轻松编写用户友好的命令行接口。程序定义它需要的参数，然后 `argparse` 将弄清如何从 `sys.argv` 解析出那些参数。 `argparse` 模块还会自动生成帮助和使用手册，并在用户给程序传入无效参数时报出错误信息。

```
论文代码中经常需要调整参数，因此我们需要把一些关键参数设置为全局变量（只改一次即可）。为了更方便的管理全局
变量池，parser也许是一种选择。
```

```
parser.add_argument('--mode', type=str, default="train", choices=["train",
"test"],
                    help='option: train, test')
parser.add_argument('--model', type=str, default="bilstm", choices=["bilstm",
"cnn"],
                    help='option: bilstm, cnn')
parser.add_argument('--dataset', type=str, default="res14",
                    help='dataset')
parser.add_argument('--max_sequence_len', type=int, default=100,
                    help='max length of a sentence')
parser.add_argument('--device', type=str, default="cpu",
                    help='gpu or cpu')
```

```
使用方法：
ref: https://www.jb51.net/article/212035.htm
1. 实例化argumentparser
parser = argparse.ArgumentParser(description = 'test')
2. 使用add_argument添加参数
parser.add_argument('--epochs', type=int, default=10000, help='epochs to train.')
3. 使用parser解析参数
args = parser.parse_args()
print (args.epochs)
```

pytorch dataset & dataloader

```
以前的课堂练习通过手动加载数据的方式，在数据量小的时候，并没有太大问题，但是到了大数据量，我们需要使用
shuffle，分割成mini-batch等操作的时候，我们可以使用PyTorch的API快速地完成这些操作。
```

```
Dataset是一个包装类，用来将数据包装为Dataset类，然后传入DataLoader中，我们再使用DataLoader这个类来更
加快捷的对数据进行操作。
```

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.

```
将你的数据存到__init__中的data即可，注意要自己重写__getitem__和__len__函数
```

```
DataLoader是一个比较重要的类，它为我们提供的常用操作有：batch_size(每个batch的大小),
shuffle(是否进
行shuffle操作), num_workers(加载数据的时候使用几个子进程)
```

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.

See `torch.utils.data` documentation page for more details.

```
dataloader其实就是在dataset的基础上帮我们每次划分出batch（可以实现shuffle以及batch_num）
为了更好的理解dataset以及dataloader，我做了一个简单的实验（code:toy.py）
```

```python
from torch.utils.data import Dataset, DataLoader

class test(Dataset):
    def __init__(self):
        self.data = []
        for i in range(0, 10):
            self.data.append((i, i + 100))

    def __getitem__(self, i):
        return self.data[i]

    def __len__(self):
        return len(self.data)

if __name__ == "__main__":
    toy = test()
    data_loader = DataLoader(
        dataset = toy,
        batch_size=5,
        shuffle=True
    )
    for epoch in range(0, 5):
        print('-----------')
        for x, y in data_loader:
            print(x, y)
```

```
(pytorch) ➜  Desktop python toy.py
-----------
tensor([0, 8, 4, 6, 5]) tensor([100, 108, 104, 106, 105])
tensor([1, 9, 7, 2, 3]) tensor([101, 109, 107, 102, 103])
-----------
tensor([1, 4, 9, 0, 8]) tensor([101, 104, 109, 100, 108])
tensor([7, 2, 5, 3, 6]) tensor([107, 102, 105, 103, 106])
-----------
tensor([9, 8, 2, 3, 4]) tensor([109, 108, 102, 103, 104])
tensor([6, 0, 5, 7, 1]) tensor([106, 100, 105, 107, 101])
-----------
tensor([5, 6, 9, 1, 3]) tensor([105, 106, 109, 101, 103])
tensor([8, 2, 7, 0, 4]) tensor([108, 102, 107, 100, 104])
-----------
tensor([2, 3, 5, 9, 8]) tensor([102, 103, 105, 109, 108])
tensor([0, 4, 6, 1, 7]) tensor([100, 104, 106, 101, 107])
(pytorch) ➜  Desktop
```

> 神奇的是：每次epoch都会自动给我重新shuffle（i了i了）
> PS:我不太清楚这是怎么实现的，dataloader如何知道我新开始了一个epoch呢？也许是对每个数据都打一个标记，都标记上了就shuffle？我猜的。。。也许哪天有空会看一下源码

> 若想将自动提取的batch进行处理可以使用collate_fn（附一个简单的demo addr）：
> ref: https://blog.csdn.net/weixin_42028364/article/details/81675021

Update: 2021.10.28