Ariel ML Data Challenge

Fu-yun Wang[1] 191300051[1]
[1] Nanjing University

摘要

简单介绍在 Ariel ML Data Challenge 的排名情况与代码设计。

1. 设备与计算资源

轻薄本 cpu。
2048 个训练样本。
1024 个测试样本。
训练时长 8 小时。
语言:python
深度学习框架: PyTorch.

2. 得分情况

如图1中所示, 我的个人队伍 ifoyooo 得分 9810, 排名 11 位。个人代码地址:
https://github.com/ifoyooo/ml-network

Current Leader Board

Rank	Name	Score
1	Valis	9931
2	TheReturnOfBasel321	9916
3	DeepBlueAI	9909
4	major_tom	9899
5	Cantina_Band	9892
6	RozanskiT	9878
7	Christophe_pere	9872
8	Wabinab	9864
9	voidzero	9829
10	qcer	9822
11	ifoyooo	9810

图 1. 排名情况

3. 代码设计

本次的代码设计主要参考了如下两个项目:
https://github.com/ucl-exoplanets/ML-challenge-baseline
https://github.com/ifoyooo/pytorch-CycleGAN-and-pix2pix
项目文件及各自的功能包括:

1. base_dataset.py 数据的抽象类。
2. challenge_dataset.py 本例中的数据类别具体实现, 主要实现了包括: ___getitem___ 函数, 用于构建 pytorch 必备的 DataLoder; 数据与处理函数:simple_transform.
3. modules.py 神经网络的各种基本结构
4. networks.py 神经网络的具体实现
5. predict.py 利用训练好的模型预测测试集的结果。
6. score.py 返回模型在测试集上的得分。
7. test.py 测试训练好的模型在整个训练集上的 loss 与得分情况。
8. train.py 训练模型。

3.1. 数据处理与网络结构

受限于计算设备的限制，仅仅在 2048 个样本上训练就用掉了 8 个小时，所以实现的数据预处理与网络结构都非常简单，并没有进行太多的尝试。在这两个方面做更多的尝试应该能较容易的有所提升。

经过对整个训练集读取分析可以得到该数据各个属性的分布的均值都越为 1，标准差约为 0.04，所以数据预处理设置为简单的标准化。如下所示

Listing 1. Transform

```
def simple_transform(x):
    out = x.clone()
    # out[:,:30]=1
    # centering
    out -= 1.
    # rough rescaling
    out /= 0.05
    return out
```

神经网络结构主要为简单的卷积神经网络与全连接的组合。

Listing 2. NetWork

```
class SimpleConv(nn.Module):
    def ___init___(self):
        super(SimpleConv, self).___init___()
        self.conv1 = nn.Conv2d(1, 4, [16, 2],
        [1, 1], padding=[4, 0])
        self.pool1 = nn.MaxPool2d([2, 1], [2, 1])
        self.conv21 = nn.Conv2d(4, 1, [16, 1],
        [1, 1], padding=[4, 0])
        self.pool2 = nn.MaxPool2d([2, 1], [1, 1])
        self.mll = nn.Sequential(nn.Linear(4784, 1024),
                                 nn.ReLU(),
                                 nn.Linear(1024, 256),
                                 nn.ReLU(),
                                 nn.Linear(256, 55),
                                 )
    def forward(self, input):
        x = input.unsqueeze(1)
        x = self.conv1(x)
        x = self.pool1(x)
        x = F.relu(x)
        x = self.conv21(x)
        x = self.pool2(x)
        x = F.relu(x)
        x = torch.flatten(x,start_dim=1)
        x = self.mll(x)
        return x
```

3.2. 训练与测试

为了便于调整训练参数，代码中使用了 parser 库，能够通过命令行直接控制模型的参数，例如当需要使用已经保存过的参数继续训练时只需要执行

```
python train.py --continue_train=True
```

的命令就可以继续训练了。

Listing 3. Train

```
    parser = argparse.ArgumentParser()
    parser.add_argument("--lc_train_path",
    help="train path",
    type=str,default=pathlib.Path(___file___).
    parent.absolute()/
    "data/noisy_train/home/ucapats/Scratch
    /ml_data_challenge/training_set/noisy_train")
    parser.add_argument("--lc_val_path",
    help="val path",type=str,default=
    pathlib.Path(___file___).parent.absolute()
    /"data/noisy_train/home/ucapats/
    Scratch/ml_data_challenge/
    training_set/noisy_train")
    parser.add_argument("--params_train_path",
    help="params_train_path",
    type=str,default=pathlib.Path(___file___).
    parent.absolute()/"data/params_train
    /home/ucapats/Scratch/ml_data_challenge
    /training_set/params_train")
    parser.add_argument("--params_val_path",
    help="params_val_path",
```

```
    type=str,default=pathlib.Path(__file__).
    parent.absolute()/"data/params_train
    /home/ucapats/Scratch/ml_data_challenge
    /training_set/params_train")
    parser.add_argument("--train_size",
    type=int,default=2048)
    parser.add_argument("--val_size",
    type=int,default=1024)
    parser.add_argument("--epochs",
    type=int,default=10)
    parser.add_argument("--save_from",
    type=int,default=3)
    parser.add_argument("--device",
    type=str,default="cuda"
    if torch.cuda.is_available() else "cpu")
    # parser.add_argument("--device",
    type=str,default="cpu")
    parser.add_argument("--batch_size",
    type=int,default=128)
    parser.add_argument("--seed",
    type=int,default=2048)
    parser.add_argument("--intput_dim",
    type=int,default=55*300)
    parser.add_argument("--output_dim",
    type=int,default=55)
    parser.add_argument("--model",
    type=str,default="Conv2d")
    parser.add_argument("--MLlinearlist",
    type=list,default=[55*300,1024,256])
    parser.add_argument("--continue_train",
    type=bool,default=False)
```

此外，保存过的模型会保存在 outputs 文件夹下，根据模型的选择的不同，会保存在不同模型的子文件夹下，并保存训练过程的 loss 变化到 txt 文件并绘图。

最后当使用训练好的模型预测时，只需要执行

```
python predict.py
```

即可。由于一次性读取所有数据会导致内存溢出，所以我仍然使用 Dataloader 作为模型的输入，每次仅预测很少一部分的结果，然后追加保存在文件当中。

4. 如何测试我的代码

请将训练数据与测试数据放于 code 文件夹中，以训练集为例，其中的所有训练数据的地址应该为

```
code\data\noisy_train\home\ucapats\Scratch
\ml_data_challenge\training_set\noisy_train
\文件名
```

然后执行 main.py

致谢. 感谢老师的阅读。

参考文献

无