

代码片段

一、Pytorch

1.1 导入包、可复现性配置、异常检测和其他

```
import torch
import random
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

myseed = 12345
torch.manual_seed(myseed)
torch.random.manual_seed(myseed)
random.seed(0)
np.random.seed(myseed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
torch.cuda.manual_seed_all(myseed)

torch.autograd.set_detect_anomaly(True) # 可在 NaN 出现时报错，定位错误代码。正向传播时：开启自动求导的异常侦测
# 反向传播时：在求导时开启侦测
#with torch.autograd.detect_anomaly():
#    loss.backward()

torch.multiprocessing.set_sharing_strategy('file_system')
```

1.2 模型搭建

```
import torch

class Net(torch.nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 一般会在这里放网络层和其他后续会用到的全局超参

    def forward(self, x):
        # 用__init__中的Module来搭建网络
        # (在这里也可以新加层，如放激活函数等)
        # 返回输出。
```

```
model=Net()
```

1.3 Gpu

```
# 使用 GPU
```

```
def try_gpu(i=0): #@save
    """如果存在，则返回 gpu(i)，否则返回 cpu()"""
    if torch.cuda.device_count() >= i + 1:
        return torch.device(f'cuda:{i}')
    return torch.device('cpu')

def try_all_gpus(): #@save
    """返回所有可用的 GPU，如果没有 GPU，则返回[cpu(),]"""
    devices = [torch.device(f'cuda:{i}')]
    for i in range(torch.cuda.device_count()):
        return devices if devices else [torch.device('cpu')]
```

1.4 训练

```
model = Net()
model = model.to(device=try_gpu())
```

```
# 定义优化器和损失函数
```

```
optim = torch.optim.SGD(model.parameters(), lr=1e-2, momentum=0.9)
criterion = torch.nn.MSELoss()
```

```
# 注意：有些模型可能需要自定义损失函数（如 PTA 模型）
```

```
#PTA 模型等的做法是在模型中自定义了 loss_function 函数，返回仅含 loss 这一元素的 Tensor
```

```
#Optional: pre-processing 工作（如 PTA 模型，开始训练前先跑了一波标签传播算法）
```

```
model.train()
```

```
for epoch in range(epochs):
    for batch_x, batch_y in train_data:
        # 有些模型不需要分批次训练，比如很多 GNN 模型就是一波流
```

```
batch_x, batch_y = batch_x.to(device=try_gpu()), batch_y.to(device=try_gpu())
prediction = model(batch_x)
loss = criterion(prediction, batch_y)
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

```
#Optional: post-processing 工作（如 C&S 模型和 PTA 模型）
```

```
# 验证
```

```
# 存储在验证集上表现最好的模型
```

```
# 设置 early stopping: 如果验证集上的表现在超过某个阈值的次数内仍然没有变好，就可以停了
```

```
# 关于这几个步骤的顺序：计算 loss→backward→step 有严格的先后顺序，zero_grad 加在网
```

络还是优化器上看需求（一般都是在优化器上），zero_grad 在 backward 之前或者 step 之后都可以（一般都是 step 之后，也就是一个 epoch 运算结束之后）

1.5 保存模型

```
PATH = '.model.pth'
torch.save(model.state_dict(), PATH)
```

1.6 加载模型

```
model=Net()
model.load_state_dict(torch.load(PATH))
```

1.7 验证模型

```
model.eval()
with torch.no_grad():
    for batch_x in test_data:
        batch_x.to(device)
        prediction = model(batch_x)
        # 在验证集上也可以计算损失函数: loss = criterion(prediction, batch_y)

        # 如果需要从 prediction (logits) 中获取最高的一项作为预测标签:
        predicted_label=torch.argmax(prediction,1)
```

1.8 可视化

1.8.1 绘制沿 epoch 的 loss 变化曲线图（在训练或验证时储存记录）

```
def plot_learning_curve(loss_record, title=''):
    ''' Plot learning curve of your DNN (train & dev loss) '''
    total_steps = len(loss_record['train'])
    x_1 = range(total_steps)
    x_2 = x_1[::len(loss_record['train']) // len(loss_record['dev'])]
    figure(figsize=(6, 4))
    plt.plot(x_1, loss_record['train'], c='tab:red', label='train')
    plt.plot(x_2, loss_record['dev'], c='tab:cyan', label='dev')
    plt.ylim(0.0, 5.)
    plt.xlabel('Training steps')
    plt.ylabel('MSE loss')
    plt.title('Learning curve of {}'.format(title))
    plt.legend()
    plt.show()
```

1.8.2 绘制沿 epoch 的 loss 和 ACC 变化曲线图（在训练或验证时储存记录）

```
plt.title(dataset_name+'数据集在'+model_name+'模型上的 loss')
plt.plot(train_losses, label="training loss")
plt.plot(val_losses, label="validating loss")
plt.plot(test_losses, label="testing loss")
plt.legend()
plt.savefig(pics_root+'/loss_'+pics_name)
plt.close() # 为了防止多图冲突
```

```
plt.title(dataset_name+'数据集在'+model_name+'模型上的 ACC',fontproperties=font)
plt.plot(train_accs, label="training acc")
plt.plot(val_accs, label="validating acc")
plt.plot(test_accs, label="testing acc")
plt.legend()
plt.savefig(pics_root+'/acc_'+pics_name)
plt.close()
```

1.8.3 绘制预测值-真实标签点图

```
def plot_pred(dv_set, model, device, lim=35., preds=None, targets=None):
    ''' Plot prediction of your DNN '''
    if preds is None or targets is None:
        model.eval()
        preds, targets = [], []
        for x, y in dv_set:
            x, y = x.to(device), y.to(device)
            with torch.no_grad():
                pred = model(x)
                preds.append(pred.detach().cpu())
                targets.append(y.detach().cpu())
        preds = torch.cat(preds, dim=0).numpy()
        targets = torch.cat(targets, dim=0).numpy()

    figure(figsize=(5, 5))
    plt.scatter(targets, preds, c='r', alpha=0.5)
    plt.plot([-0.2, lim], [-0.2, lim], c='b')
    plt.xlim(-0.2, lim)
    plt.ylim(-0.2, lim)
    plt.xlabel('ground truth value')
    plt.ylabel('predicted value')
    plt.title('Ground Truth v.s. Prediction')
    plt.show()
```

1.8.4 打印每一类的 accuracy (multi-class one-label 分类)

```
def get_report(labels, preds):
    """
```

输入是每个样本的标签和预测值的列表
要求严格按照从 0 开始的标签索引顺序来排列

```
"""
```

```
N_CLASSES = max(labels) + 1
class_correct = list(0. for i in range(N_CLASSES))
class_total = list(0. for i in range(N_CLASSES))
c = (preds == labels)
for i in range(len(labels)):
    label = labels[i]
    class_correct[label] += c[i]
    class_total[label] += 1
report = ""
for i in range(N_CLASSES):
    if class_total[i]:
        report += 'Accuracy of %d : %d/%d=%.4f' % (
            i, class_correct[i], class_total[i], class_correct[i] /
class_total[i]) + "\n"
return report
```

1.9 自定义数据集

- `__init__`: 用于接收外部参数, 比如文件路径等, 并完成数据集加载
- `__getitem__`: 根据索引读取数据集中的元素, 进行一定转换, 返回单个样本及其标签
- `__len__`: 返回数据集的大小

```
class ImageDataSet(Dataset):
    def __init__(self, flag, path, transform=None):
        assert flag in ["train", "valid", "test"]
        # ...

    def __getitem__(self, index):
        # ...

    def __len__(self):
        # ...
```

1.10 DataLoader 按批次读取数据

- `dataset`: 封装好的数据集, 取值为 tuple 型, 装有样本和标签。
- `batch_size`: 批量, 每次循环时取出的数据量大小
- `drop_last`: 当数据集无法整除 `batch_size` 时, 为 True 则最后一批会被丢掉, 为 False 则最后一批会被保留, 该批数量会变少。
- `shuffle`: 是否随机返回 batch, 默认不随机。(训练时需要随机来提高训练精度, 验证和测试时不需要)

- num_workers: 进程数

```
DataLoader(dataset, batch_size=1, drop_last=False, shuffle=None,  
num_workers=0)  
train_loader = DataLoader(train_data, batch_size=args.batch_size,  
shuffle=True)  
test_loader = DataLoader(valid_data, batch_size=args.batch_size,  
shuffle=False)
```

1.11 Pytorch - transforms

- torchvision.transforms: 提供了常用的一系列图像预处理方法, 例如数据的标准化, 中心化, 旋转, 翻转等。
- torchvision.datasets: 定义了一系列常用的公开数据集的 datasets, 比如 MNIST, CIFAR-10, ImageNet 等。
- torchvision.model: 提供了常用的预训练模型, 例如 AlexNet, VGG, ResNet, GoogLeNet 等。

<https://blog.csdn.net/lsb2002/article/details/134895212>

二、 引用

[1] 《PyTorch 的可复用代码模板 (持续更新 ing...) import torch.nn as nn 和 from torch import nn-CSDN 博客》. 见于 2024 年 7 月 9 日. <https://blog.csdn.net/PolarisRisingWar/article/details/117223028>.