第五周

1. 学习率调整策略

1.1 调整学习率的原因

- 学习率大时，loss下降得快，但到了一定程度不再下降
- 学习率小时，loss下降得多，但下降的速度慢
- 在训练的过程中调整学习率，先大后小，可以使得训练的效率更高效果更好

1.2 pytorch的六种学习率调整策略

基类

```
class _LRScheduler(object)
    def __init__(self, optimizer, last_epoch=-1):
    # optimizer：关联的优化器
    # last_epoch：记录epoch数
    # base_lr：记录初始学习率
    def step():
    # 更新下一个epoch的学习率
    def get_lr(self):
    # 虚函数，计算下一个epoch的学习率
        raise NotImplementedError
```

学习率调整测率

- StepLR：$lr = lr \times gamma$
- MultiStepLR：$lr = lr \times gamma$
- ExponentialLR：$lr = lr \times gamma^{epoch}$
- CosineAnnealingLR：$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\frac{T_{cur}}{T_{\max}}\pi\right)\right)$
- ReduceLROnPlateau
- LambdaLR

```
# 1. StepLR
# 等间隔调整学习率
lr_scheduler.StepLR(optimizer,
        step_size,    # 调整间隔数
        gamma=0.1,    # 调整系数
        last_epoch=-1)
# 2. MultiStepLR
# 按给定间隔调整学习率
lr_scheduler.MultiStepLR(optimizer,
        milestones,    # 设定调整时刻数，如[50,70,100]
        gamma=0.1,    # 调整系数
        last_epoch=-1)
# 3. ExponentialLR
# 按指数衰减调整学习率
lr_scheduler.ExponentialLR(optimizer,
        gamma=0.1,    # 指数的底
        last_epoch=-1)
```

```
# 4. CosineAnnealingLR
# 余弦周期调整学习率
lr_scheduler.CosineAnnealingLR(optimizer,
        T_max,    # 下降周期
        eta_min=0,    # 学习率下限
        last_epoch=-1)
# 5. ReduceLROnPlateau
# 监控指标，当指标不再变化时调整学习率
# scheduler_lr.step(要监测的变量)
lr_scheduler.ReduceLROnPlateau(optimizer,
        mode='min',    # min/max两种模式
        factor=0.1,    # 调整系数
        patience=10,    # 接受参数不变化的迭代次数
        verbose=False,    # 是否打印日志
        threshold=0.0001,
        threshold_mode='rel',
        cooldown=0,    # 停止监控的迭代次数
        min_lr=0,    # 学习率下限
         eps=1e-08)    # 学习率衰减最小值
# 6. LambdaLR
# 自定义调整策略，可以对不同的参数组使用不同的参数调整策略
lr_scheduler.LambdaLR(optimizer,
        lr_lambda,    # 函数或元素为函数的列表
        last_epoch=-1)
```

2. 可视化工具——TensorBoard

2.1 TensorBoard简介

- TensorBoard：TensorFlow中的可视化工具，支持标量/图像/文本/音频/视频/Eembedding等多种数据可视化

2.2 TensorBoard安装

`pip install tensorboard`

2.3 TensorBoard运行

`tensorboard --logdir=./runs`

2.4 作业

第五周作业1

2.5 SummaryWriter

```
# 提供创建event file的高级接口
class SummaryWriter(object):
    def __init__(self,
        log_dir=None,    # event file 输出文件
        comment='',    # 不指定log_dir时，文件夹后缀
        purge_step=None,
        max_queue=10,
        flush_secs=120,
        filename_suffix='')    # event file文件名后缀
```

2.6 add_scalar 和 add_histogram

```
# 1. add_scalar
# 记录标量
add_scalar(tag,     # 图像的标签名，图的唯一标识
        scalar_value,     # 要记录的标量
        global_step=None,     # x轴
        walltime=None)
add_scalars(main_tag,     #  该图的标签
        tag_scalar_dict,     # key是变量的tag，value是变量的值
        global_step=None,
        walltime=None)
# 2. add_histogram
# 统计直方图与多分位数折线图
add_histogram(tag,     # 图像的标签名，图的唯一标识
        values,     # 要统计的参数
        global_step=None,     # y轴
        bins='tensorflow',     # 取直方图的bins
        walltime=None)
```

2.7 add_image 和 torchvision.utils.make_grid

```
# 1. add_image
# 记录图像
add_image(tag,     # 图像的标签名，图的唯一标识
        img_tensor,     # 图像数据，注意尺度
        global_step=None,     # x轴
        walltime=None,
        dataformats='CHW')     # 数据形式，CHW/HWC/HW
# 2. torchvision.utils.make_grid
# 制作网格图像
make_grid(tensor,     # 图像数据，B*C*H*W形式
        nrow=8,     # 行数（列数自动计算）
        padding=2,     # 图像间距（像素单位）
        normalize=False,     # 是否将像素值标准化
        range=None,     # 标准化范围
        scale_each=False,     # 是否单张图维度标准化
        pad_value=0)     # padding的像素值
```

2.8 add_graph 和 torchsummary

```
# 1. add_graph
# 可视化模型计算图
add_graph(model,     # 模型，必须是nn.Module
        input_to_model=None,     # 输出给模型的数据
        verbose=False)     # 是否打印计算图结构信息
# 2. torchsummary
# 查看模型信息，便于调试
summary(model,     # pytorch模型
        input_size,     # 模型输入size
        batch_size=-1,     # batch size
        device="cuda")     # cuda/cpu
```

2.9 作业

第五周作业2

3. Hook函数与CAM可视化

3.1 Hook函数概念

- Hook函数机制：不改变主体，实现额外功能，像一个挂件

```
# 1.
# 注册一个反向传播的hook函数
# hook(grad) -> Tensor or None
torch.Tensor.register_hook(hook)
# 2.
# 注册module的前向传播hook函数
# hook(module, input, output) -> None
# module：当前网络层
# input：当前网路层输入数据
# output：当前网络层输出数据
torch.nn.Module.register_forward_hook(hook)
# 3.
# 注册module前向传播前的hook函数
# hook(module, input) -> None
# module：当前网络层
# input：当前网路层输入数据
torch.nn.Module.register_forward_pre_hook(hook)
# 4.
# 注册module反向传播的hook函数
# hook(module, grad_input, grad_output) -> Tensor or None
# module：当前网络层
# grad_input：当前网络层输入梯度数据
# grad_output：当前网络层输出梯度数据
torch.nn.Module.register_backward_hook(hook)
```

3.2 CAM 类激活图

- Class Activation Map：在最后Conv层后、FC层前增加GAP层，得到一组权重，与其对应特征图加权，得到GAM
- Grad-CAM：CAM的改进，使用梯度作为特征图权重

3.3 作业

第五周作业3