Homework5

（本作业主要涉及使用 PyTorch 构建并训练简单的神经网络）

请在 6 月 6 日前于学在浙大提交

一、写在前面

经过课堂上的学习和课后的复习 ,相信大家对卷积神经网络的构建和训练有了一定的了解。在本次作业中，我们将尝试使用 PyTorch 构建和训练一个经典的卷积神经网络 LeNet-5。

二、PyTorch 安装与介绍

在本次作业中，我们将用 PyTorch 来完成模型的构建和训练。PyTorch 是时下流行的一种深度学习框架，在互联网上存在大量的相于如何使用它的资料。本次作业中，我们将在 MNIST 数据集上训练 LeNet-5，因其计算量较小，故选择 PyTorch 的 cpu 版本。

官方安装网址为 https://pytorch.org/get-started/locally/#windows-anaconda

START LOCALLY

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

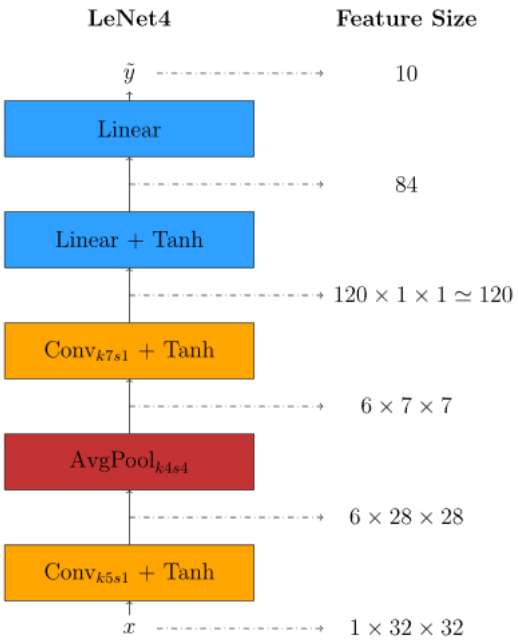
PyTorch Build	Stable (2.0.1)	Preview (Nightly)		
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 11.7	CUDA 11.8	~~ROCm 5.4.2~~	CPU
Run this Command:	conda install pytorch torchvision torchaudio cpuonly -c pytorch			

根据自己电脑的 os 选择 Linux、Mac 或 Windows。如果你已经安装了 Anaconda 并创建了用于本次作业的 conda 环境，可以选择使用 conda 命令安装（请先 conda activate env_name 进入该环境）。也可以选择使用 pip 命令安装。在命令行输入你所选择的进行安装配置对应的命令（上图 run this command 所给出的命令）即可。

PyTorch 的基本数据类型为 Tensor ,我们可以把它理解为一种经过精致地封装的数组，类比 MatLab 中的矩阵。PyTorch 的计算图的构建和自动求导也依赖于 Tensor 之间的运算关系。而模型的构建则可视为一种对 Tensor 之间的运算关系的模块化和结构化。

Tensor 之间能够做逐元素的四则运算、矩阵乘法等。而 Tensor 本身支持切片、复制、求幂等操作，详见 PyTorch 的官方文档或 help 信息。

三、LeNet-4 模型结构与构建



LeNet-4 分为卷积和全连接两部分。它的卷积部分由交替的卷积层（Convolution）和平均池化层（Average Pooling）组成，全连接部分为两层全连接层。为了简化模型，我们在实现 LeNet-4 时，在平均池化层处进行了修改，去掉了该层的可学习参数以及激活函数。简化后的模型如上图所示。该图的左侧为模型的逐层解析，右侧为经过对应层作用后特征图或特征的尺寸（第一个数表示通道数或特征长度），我们用这些尺寸的变化来表示对应的卷积层或全连接层的输入输出的通道数或特征长度。注意到卷积层和池化层中存在下标，如“Convk5s1”，下标中的“k5”表示卷积核的大小为 5x5，“s1”表示卷积核在特征图上滑动时的步长为 1。为了模型表示的简洁性，我们用“+”对一些简单的操作进行连接，合并到同一层表示，其对数据的操作采用从左到右的顺序。如“Convk5s1+Tanh”表示先通过一个卷积层，然后对特征作用双曲正切激活函数。

在神经网络模型的构建中，我们常用 torch.nn.Module 类作为基类。该类为我们实现了参数管理等功能。在“torch.nn”模块下存在着大量的已经封装好的单层模型例如“torch.nn.Linear”即为全连接层，“torch.nn.Conv2d”即为二维卷积层等。这些单层模型为我们构建模型带来了极大的便利。我们只需按照模型中的结构，将这些单层模型进行简单的堆叠就能够完成模型的构建。以下，我们以 LeNet-4 为例对模型构建的过程进行简单的介绍。

在使用“torch.nn.Module”作为基类进行模型构建时，我们需要实现的部分为“__init__”和“forward”这两个函数。“__init__”中我们需要给出模型所需的原材料——各个小模型，而“forward”中则描述了输入数据在这些小模型之间流动的顺序并返回计算结果。我们注意到“__init__”函数体的第一行为“super().__init__()”。该行对整个类进行了一些微妙操作，使得我们在接下来的部分中添加的小模型同时将小模型的参数注册到整个模型中进行管理。

由于 LeNet-4 整体为序贯的单层模型的堆叠，我们使用“nn.Sequential”模块将一些序贯计算的单层模型进行打包。在这个代码中，我们将整个 LeNet-4 分割成两部分，一部分为卷积网络“self.conv”，另一部分为全连接网络“self.fc”。

在“forward”函数中，我们规定了该模块的输入形式，即单一的一个输入变量“input”。联系模型的计算，我们希望它是一个 Tensor，且大小为“B×1×32×32”。这里的“B”为 mini batch 的大小，“1”为通道数，“32×32”为输入图片的长宽尺寸。

“forward”函数体的第一行非注释代码中，我们用 Tensor 的“size”函数获取 input 的第一个维度的大小，而它在我们预设的计算环境中应当为 batch size。在第二行非注释代码中，我们让输入数据通过了“self.conv”，这时我们得到了一个中间的 Tensor，其形状为“B×120×1×1”，由于全连接部分“self.fc”需要的输入格式为“B×120”，我们使用“reshape”函数对 Tensor 进行整形。“reshape”参数表中的-1 作为占位符使用，表示该维度的大小应当通过原始的形状与其它参数推断得出。

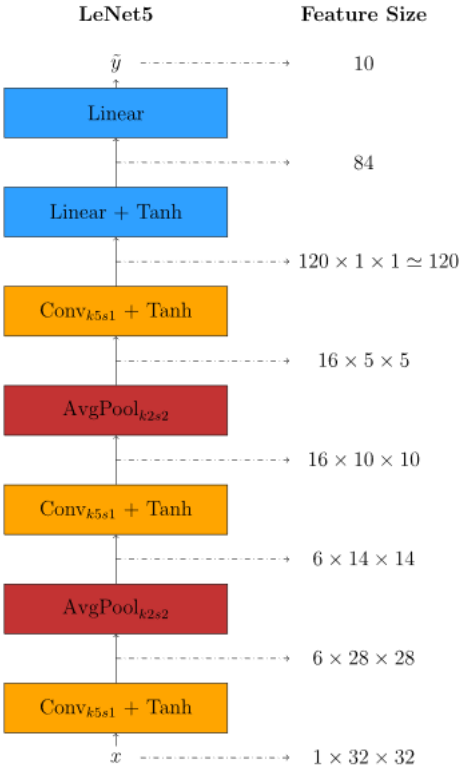
我们已经在“src/models.py”中给出了上述的 LeNet-4 模型。

```python
class LeNet4(nn.Module):
    def __init__(self) -> None:
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=4, stride=4),
            nn.Conv2d(in_channels=6, out_channels=120, kernel_size=7, stride=1),
            nn.Tanh()
        )
        self.fc = nn.Sequential(
            nn.Linear(in_features=120, out_features=84),
            nn.Tanh(),
            nn.Linear(in_features=84, out_features=10)
        )

    def forward(self, input):
        """
        input: [B, 1, 32, 32]
        """
        batch_size = input.size(0)
        x = self.conv(input).reshape(batch_size, -1)
        x = self.fc(x)
        return x
```

四、LeNet-5 模型结构

LeNet-5 是在上世纪 90 年代由 Yann LeCun 等人提出的一个用于手写体和印刷体字符识别的卷积神经网络。它在 MNIST 数据集上能够达 99%的正确率。其模型结构如下图，可以看到它比上文所介绍的 LeNet-4 多了一个卷积层。



五、模型训练

构建完模型后，我们通常使用反向传播算法对模型进行训练。而在训练的整个流程中，我们需要以下材料：

- 数据集：本次作业中，我们要用到的数据集为 MNIST，幸运的是 PyTorch 的 torchvision 模块的 datasets 子模块中已经包含了该数据集；
- 模型：在本次作业中，我们使用 LeNet-4 和 LeNet-5 作为训练的模型；
- 损失函数：在本次作业中，我们将使用两种损失函数训练模型，一为交叉熵损失函数（CrossEntropyLoss），另一为均方误差损失（MSELoss）；
- 优化器：在神经网络的训练中，我们通常使用随机梯度下降算法对模型进行优化，故我们使用“torch.optim.SGD”或“torch.optim.Adam”优化器。

其中损失函数相关的定义和计算方法如下：

在多分类任务中，输入标签往往为从 0 开始的连续的一段自然数集合。如 MNIST 有 10 种类别，其标签分别为{0，1，2，3，4，5，6，7，8，9}。我们通常将这种标签转化为 One-Hot 表示，同样以 MNIST 为例，我们用长度为 10 的列向量来表示一个标签：对于标签 i，我们将列向量的第 i 个位置置为 1，其余位置置为 0。此时模型的输出也应当是对应长度的列向量，如 LeNet 的输出是长度为 10 的列向量。

在接下来的讨论中，我们设模型为 M，输入数据为 x^i 对应的输出为 z^i，One-Hot 标签为 y^i, C 为类别数量。

均方误差损失函数

一种直接的损失函数的设计是对 One-Hot 表示的标签进行回归。这里我们选择最小化均方误差，即得到均方误差损失函数：

$$\begin{aligned}\mathrm{MSELoss}(\mathbf{z},\mathbf{y}) &= \frac{1}{C|\mathrm{batch}|}\sum_{i\in\mathrm{batch}}\|\boldsymbol{z}^i-\boldsymbol{y}^i\|_2^2\\ &= \frac{1}{C|\mathrm{batch}|}\sum_{i\in\mathrm{batch}}\sum_j\left(z_j^i-y_j^i\right)^2\end{aligned}$$

交叉熵损失函数

另一种损失函数将模型的输出的每个分量视为标签的概率。为了将模型的输出转化为概率，我们通常使用 SoftMax 函数，它首先对每个分量作用指数函数，将分量的取值范围拉到正实数上，接着对这些值做归一化，其计算如下：

$$\mathrm{SoftMax}_i(\boldsymbol{z}) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

我们用$\tilde{\boldsymbol{z}}$表示经过 SoftMax 后的输出向量。那么交叉熵损失函数为：

$$\begin{aligned}\mathrm{CrossEntropyLoss}(\tilde{\mathbf{z}},\mathbf{y}) &= -\frac{1}{C|\mathrm{batch}|}\sum_{i\in\mathrm{batch}}\sum_j y_j^i\log\tilde{z}_j^i+(1-y_j^i)\log(1-\tilde{z}_j^i)\\ &= -\frac{1}{C|\mathrm{batch}|}\sum_{i\in\mathrm{batch}}\log\tilde{z}_k^i+\sum_{j\neq k}\log(1-\tilde{z}_j^i),\quad [y_k^i=1]\end{aligned}$$

可以看出，交叉熵损失函数对每个样本对计算了每种分类的概率分布的二元信息熵（用y_c^i表示第 i 个样本分类为 c 的概率，$1-y_c^i$表示第 i 个样本分类不为 c 的概率）。而从整体看，交叉熵损失函数给出了概率分布$\tilde{\boldsymbol{z}}$和$\boldsymbol{y}$之间的差别。

为了对模型的训练进行控制，我们需要给出一些训练参数：总的训练轮数（epoch）、学习率（learning_rate）、mini batch 的大小（batch size）等。

为了挑选模型，我们常在每一轮结束或每更新固定的步数后，使用验证集对模型进行评价。对于分类任务，评价准则通常为准确率（accuracy）。

在训练结束后，我们使用在验证集上表现最好的模型进行测试，得到训练结果。

相关代码请参考“src/main.py”。

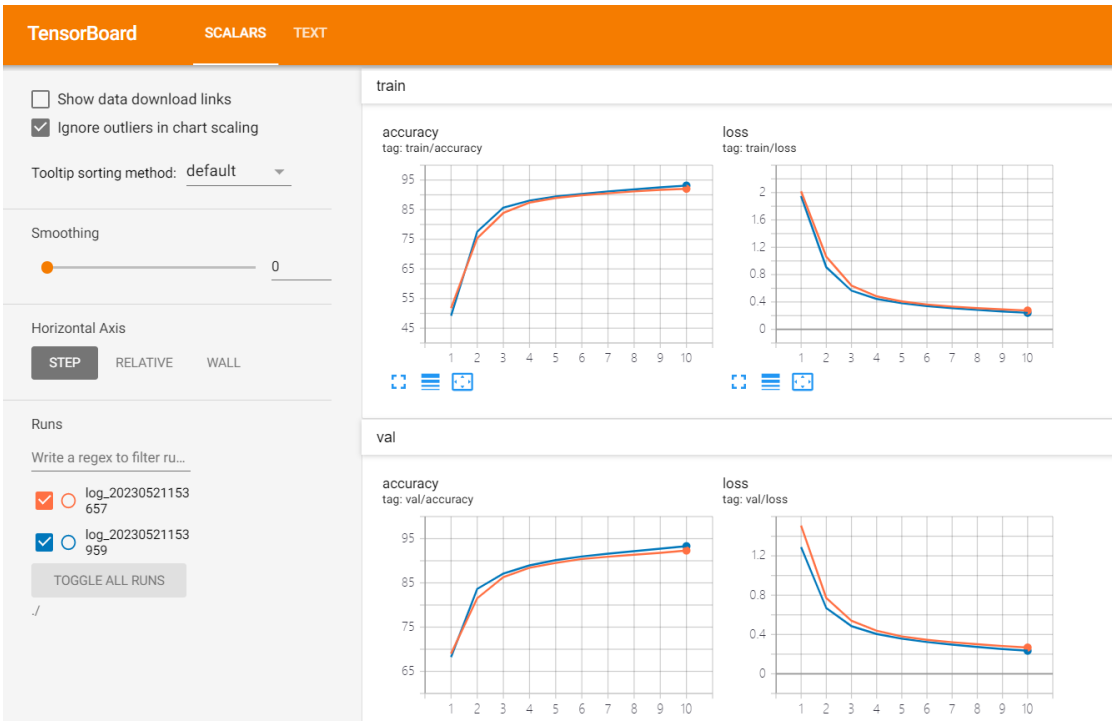
六、tensorboard 使用

我们使用 tensorboard 来绘制训练过程的 loss 和 accuracy 曲线。完成训练后，进入 src 目录，在命令行输入以下内容：

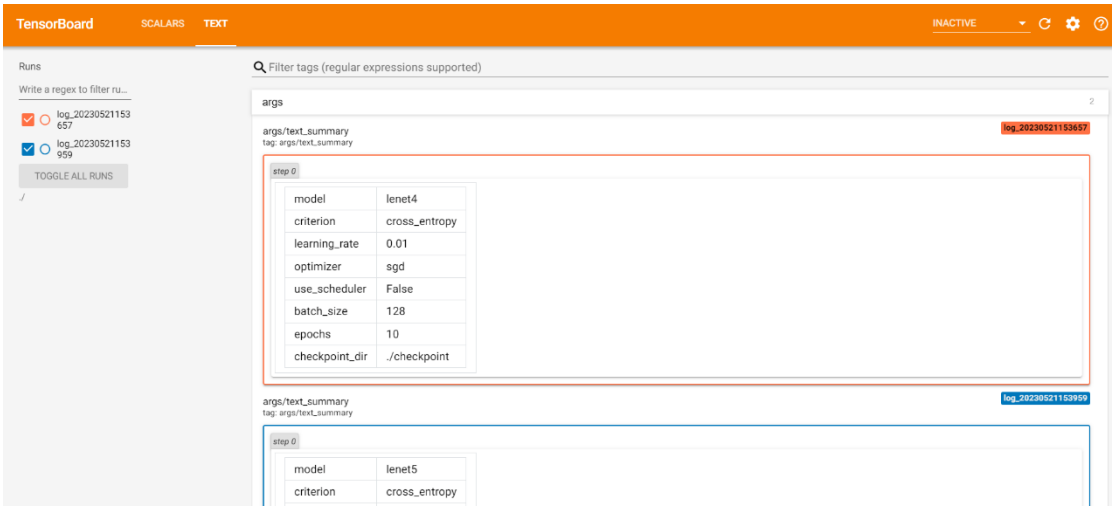
```
tensorboard --host localhost --port 6099 --logdir ./
```

其中 host 为需要打开的 ip（因为是本地训练的，用 localhost 即可），port 为任意指定的端口，logdir 代表了当前需要查看的 log 父级目录（如果 src 下有多个 log，则都会进行可视化，这样可以方便进行不同训练配置的曲线比较）。

然后打开网页 http://localhost:6099 即可查看 tensorboard 结果。



在 SCALARS 条目中，左侧面板的 smoothing 可以用于对折线进行平滑。左侧面板的 runs 可以选择展现哪些实验。



在 TEXT 条目中，我们写入了不同实验各项超参的值。

七、作业内容

1. 代码补充：请根据 LeNet-5 的模型结构图，在“src/models.py”中完善 LeNet-5 模型的参数部分（卷积层、平均池化层和全连接层的参数，包括输入输出的通道数、特征长度、卷积核大小和移动步长）；

2. 训练参数选取：使用交叉熵损失函数对 LeNet-4 进行训练，请测试在不同的 batch_size、epochs、learning_rate、optimizer，以及是否使用学习率调节器的情况下对模型训练的影响。可以用 tensorboard 曲线关注训练 loss 的下降快慢、是否收敛；比较 train_acc 和 val_acc 说明是否出现过拟合、欠拟合等问题；比较测试集上的准确率……

3. 模型比较：用交叉熵损失函数 / 均方误差损失函数对 LeNet-4 / LeNet-5 模型进行训练，得到 4 个模型。对于其他超参，不需要和 2 一样非常细致地进行比较，选取一组你认为合适的超参即可（但注意，对于相同损失函数训练的 2 个模型，各项超参的取值要一样，以保证尽可能公平的比较）。同样地，对于训练过程和结果给出自己的分析。

八、提交要求

请提交一个以 **hw5_学号.zip** 命名的 zip 格式压缩文件，此压缩文件应当包含：

1) 补充完整的 models.py；

2) 文件夹 src；

3) 作业报告 report.pdf。

作业报告要求：

1) 详见七、作业内容。

2) 报告的具体格式没有要求。

九、成绩评定

代码补充实现占 30%，report 占 70%。

十、Q&A

1. 如何使用不同的超参配置运行实验？

各项超参定义在了 main.py 中，具体如下：

- model：可选择 lenet4 或 lenet5，在训练时会使用 LeNet-4 或 LeNet-5 模型；
- criterion：可选择 cross_entropy 或 mse，在训练时会使用交叉熵损失函数或均方误差损失函数；
- learning_rate：初始学习率，如果不使用学习率调节器则后续训练过程中不会改变；
- optimizer：可选择 sgd 或 adam，在训练时使用相应的优化器；

- use_scheduler：学习率调节器，默认不使用，如果命令行写入了--use_scheduler 则会使用；
- batch_size：一个 batch 的数据量；
- epochs：训练的 epoch 数量；
- checkpoint_dir：模型 checkpoint 保存的目录。

```
if __name__ == '__main__':
    start_time = time.time()
    parser = argparse.ArgumentParser(description='train process')
    parser.add_argument('--model', default='lenet4', type=str, help='lenet4 or lenet5')  # 模型
    parser.add_argument('--criterion', default='cross_entropy', type=str, help='cross_entropy or mse')  # 
    parser.add_argument('--learning_rate', default=1e-2, type=float)  # 初始学习率
    parser.add_argument('--optimizer', default='sgd', type=str, help='sgd or adam')  # 优化器
    parser.add_argument('--use_scheduler', action='store_true', help='use learning rate scheduler')  # 是否
    parser.add_argument('--batch_size', default=128, type=int)
    parser.add_argument('--epochs', default=10, type=int)
    parser.add_argument('--checkpoint_dir', default='./checkpoint', type=str, help='the directory to save c
    args = parser.parse_args()
```

如果在命令行不指定具体的超参，则会使用上图各项 default 里定义的默认值。

举个例子，进入 src 目录，在终端命令行输入以下内容：

```
python main.py --model lenet5 --criterion cross_entropy --learning_rate 1e-3 --epochs 50
```

对应的实验配置为使用 LeNet-5 模型，使用交叉熵损失函数，初始学习率为 0.001，使用 SGD 优化器，不使用学习率调节器，batch_size 为 128，训练 50 个 epoch，模型 checkpoint 的保存路径为./checkpoint。