生产实习周报小结

任务题目：FlexiFed 论文复现

报告人：周臻鹏

7 月 14 日-7 月 25 日

目录

一. 简述

在第二,三周的学习研究中,我主要在做以下工作:

- Basic-Common 和 Clustered-Common 方法在 python 的实现和调试
- CINIC-10 数据集的导入, Speech_Commands 数据集的导入和预处理,在 python 中实现了能预处理音频数据的 dataset 子类 Audio_DataLoader
- 开展第一,第二阶段的复现任务,在 Flexifed 框架下,使用 VGG 模型训练数据集 CINIC-10 和 Speech_Commands,多次调整参数并重复训练

目前第一,第二阶段任务已完成(不考虑结果正确性前提下).

总任务进度目前达到 50% ,下一步的任务工作是开展 ResNet 在 CIFAR-10, CINIC-10 和 Speech Commands ,将引入新的 model 并尝试解决新的问题,目前 CINIC-10 和 Speech Commands 数据集体量较大,每次训练的时间在 10h 甚至更多,加之经常有曲线表现异常,得出结果的速度放慢了很多.

这两周的实验过程是曲折艰难的,源码中只给出 CINIC-10,CIFAR-10 数据集，VGG 模型，Basic-Common，Max-Common 方法的实现，其余以外的所有内容都是我自己编写调试，之前没有扎实的基础，反复经历了参考-尝试-报错-分析-查阅资料-解决报错的过程，反复地阅读文章阅读查看其中的实验过程和设置，深入地和 pytoch 打交道，逼迫自己迅速学习了一系列的 pytorch 工程上的技术知识，最终完善跑通了项目工程，这两周的进度虽然很不理想，但是我的基础性实践经验得到了快速有效的积累。

二. 实验内容

1. Flexifed 框架下方法的实现和调试

1.1 Basic-Common 方法的调试

在第一周的尝试中，直接将聚合策略改为 Basic-Common 会出现报错：

图 2.1 Basic-Common 报错

在仔细阅读源码后发现，在 utils 模块下的 common_basic 函数里，返回值如下图 2.2

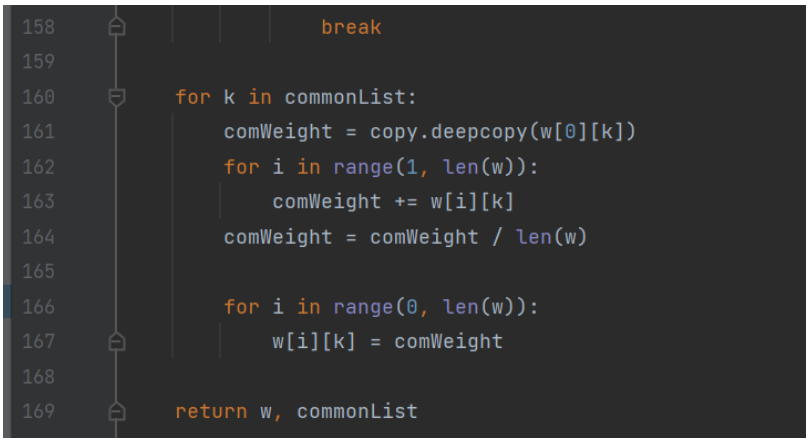

图 2.2 common_basic 函数返回值

按照代码逻辑，该函数应该返回 w，即 8 个 user 的 model 集合，传回给 modelAccept 变量后继续迭代后面的轮次，但这里多返回了一个 commonList，表示的是 8 个 user 共享的基础层的 key 的集合，返回给主程序必然导致类型报错，所以我的解决方法是将 return 变量中的 commonList 删除. 删除之后训练正常.

1.2 Clustered-Common 方法的实现和调试

源代码中没有给出 common_Clustered 方法的实现,只有函数 FedAvg(),所以我选择自己编写 common_Clustered()函数,函参为 w,即 model 的 list,返回值也是 w,函数作用是做 local model 之间的 communication,聚合模型参数,思路参考了原文中的伪代码:

Input: Client set $\mathbb{U}$, learning rate lr, local epoch E, current round t
Output: Aggregated global model set $\mathbb{W}_{t+1}$

/* Initialize global variables. */
$\omega_{u,0} = \theta$; ▷ Each client initializes its local model
$t = 0$; ▷ Round counter
$\mathbb{W}_{t+1} = \emptyset$ ▷ Stores aggregated global models
/* Parameter server aggregates local models with Clustered-Common */
Function Clustered-Common($\mathbb{U}$)
  /* Receives and stores local models from clients */
  $\omega_{g,t+1}^{comm}$ = Basic-Common($\mathbb{U}$) ▷ via Alg. 2
  Clusters local models into C groups
  for $\mathbb{C}_i \in \mathbb{C}$ **do**
    $\omega_{\mathbb{C}_i,t+1}^{pers} = \frac{1}{|\mathbb{C}_i|} \sum_{u}^{\mathbb{C}_i} \omega_{u,t}^{pers}$ ▷ via Eq. (6), i=1, 2,..., C
    for $u \in \mathbb{C}_i$ **do**
      $\omega_{u,t+1} = [\omega_{g,t+1}^{comm}, \omega_{\mathbb{C}_i,t+1}^{pers}]$
      Add $\omega_{u,t+1}$ to $\mathbb{W}_{t+1}$
  return $\mathbb{W}_{t+1}$

Alg. 3: FL with Clustered-Common under FlexiFed

图 2.3 Clustered-Common 方法伪代码

Step1:直接调用已经写好的 common_basic 函数,先对所有的 local model 做一次全局共享的基础层的模型聚合

Step2:划分聚类,将结构完全相同的 model 划分为一类,确保所有 model 都被划分到某一类中,所有分类完之后的聚类存在二维 list clu 中

Step3:在每一个聚类中,在当前聚类下做基础层以上层的模型聚合

Step4:重新将 Step3 之后的所有 model 汇总到一个 list 中,维持顺序和输入时不变

Clustered-Common 实现这部分的源代码可见附录.

2. 数据集的导入和预处理

2.1 CINIC-10 数据集的导入

源代码在 utils.getdataset()函数中对 CINIC-10 数据集的处理方法如下图

图 2.3 源代码处理 CINIC-10 数据集

它默认 CINIC-10 数据集存储在上一层目录中的 data 文件夹中,但直接下载的源码显然没有这一文件,所以我在 kaggle 平台下载了 CINIC-10 数据集文件,存放到了服务器里(数据集文件过大,不宜存放本地).

上传方式选择 jupyter lab 连接服务器后上传,但是它不支持上传文件夹,所以将数据集全部打包为 CINIC-10.zip 文件后上传压缩文件到服务器,用 unzip 命令解压.

2.2 Speech_Commands 数据集的导入

源代码中没有写 Speech_Commands 数据的导入,我在 utils.getdataset()函数中补充了 dataset 为 Speech_Commands 的情况:

Step1:模仿源代码对 CINIC-10 数据集的处理,确定函数的四个返回值:

train_dataset,test_dataset,user_groups,user_goup_test

Step2:自定义导入音频数据的 dataset 子类 Audio_DataLoader:

Pytorch 的 dataset 子类里没有自带的能导入音频数据集的 class,所以我自己创建了一个 dataset 的子类 Audio_DataLoader,重写了__init__()和

__getitem__()函数,

__init__()函数中,定义了变量 sr 表示采样率,通过查看.wav 文件得知采样率为 16000

```
class Audio_DataLoader(Dataset):
    def __init__(self, data_folder, sr=16000, dimension=8192):
        self.data_folder = data_folder
        self.sr = sr
        self.dim = dimension
        self.labellist = []
        # 获取音频名列表
        self.wav_list = []
        for root, dirnames, filenames in os.walk(data_folder):
            for filename in fnmatch.filter(filenames, "*.wav"):  # 实现列表特殊字符的过滤或筛选,返回符合匹配".wav"
                self.wav_list.append(os.path.join(root, filename))
        # print("len of wav list:{}\n{}".format(len(self.wav_list),self.wav_list[0]))
```

图 2.4 __init__()函数自定义

__getitem__()函数中,首先实现了对音频文件的取帧,因为试错发现每个.wav 文件的大小并不相同,我必须将其统一化,在用 librosa 模块导入音频文件后,计算得知最小的长度为 8192,所以对于长度大于 8192 的 data,我选择,随机截取一段连续的长度为 8192 的数据作为样本;

对于数据的预处理,我发现原始数据的分布和数值并不理想,10^-4 次方的小数据居多,且基本不服从正态分布

```
tensor([[[-6.1035e-05, -4.2725e-04, -1.5259e-04,  ..., -7.9346e-04,
          -9.1553e-04, -5.7983e-04],
         [ 1.8311e-04,  1.8311e-04, -3.3569e-04,  ..., -3.0518e-05,
           -5.1880e-04, -3.3569e-04],
         [-3.9673e-04, -6.4087e-04, -5.1880e-04,  ..., -3.6621e-04,
           -4.2725e-04, -4.5776e-04],
         ...,
         [ 1.6479e-03,  2.3193e-03,  1.6174e-03,  ...,  2.7466e-04,
           6.4087e-04,  3.9673e-04],
         [ 9.1553e-04,  1.6785e-03,  1.7395e-03,  ...,  1.0681e-03,
           1.3123e-03,  7.9346e-04],
         [ 1.4343e-03,  2.1057e-03,  1.8311e-03,  ...,  7.6294e-04,
           8.8501e-04,  1.7395e-03]],

        [[ 2.1973e-03,  2.3499e-03,  2.2278e-03,  ...,  0.0000e+00,
           1.0071e-03,  1.9836e-03],
         [ 1.6174e-03,  7.9346e-04,  1.0376e-03,  ...,  2.0752e-03,
           2.5635e-03,  2.5024e-03],
         [ 1.8005e-03,  1.0376e-03,  1.0071e-03,  ...,  2.0752e-03,
           2.3499e-03,  2.4414e-03],
         ...,
         [-3.0518e-05,  9.1553e-05,  2.4414e-04,  ..., -1.5259e-04,
          -1.8311e-04, -9.1553e-05],
         [ 9.1553e-05,  3.0518e-05, -1.5259e-04,  ..., -6.1035e-05,
```

图 2.5 speech_commands 原始数据遍历

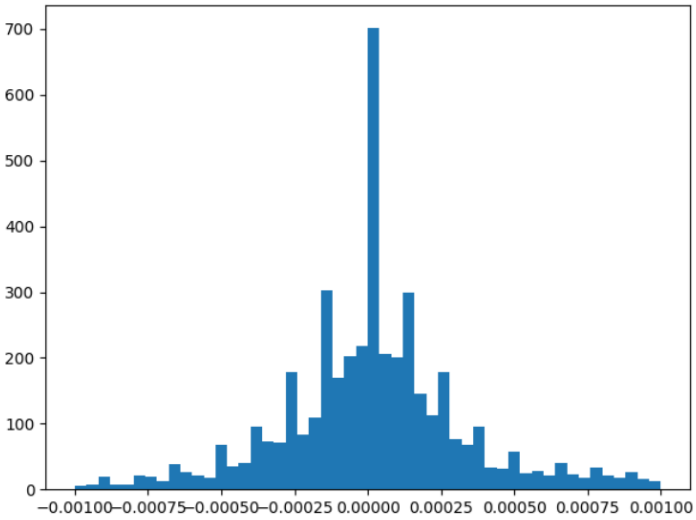

图 2.6 随机抽取一组 data 的分布图

为此,我手动进行了数据集的标准化和正则化,先遍历整个数据集计算出 mean 和 std,根据公式$x = \frac{x-mean}{std}$ 将数据集标准化,再计算出整体的 max 和 min，先$x1 = \frac{x-min}{max-min}$ ，再$x2 = \frac{x1-0.5}{0.5}$归一化到[-1,1]上来

对于 label 标签的获取,我选择用 os.walk()函数遍历 speech_commands/train 目录下的所有目录和文件获取所有.wav 文件的文件名和绝对地址,每个.wav 文件地址的上一层目录名就是它的唯一标签,但是是 str 类型的字符串表示,再用一个 dict 就可以将 str 标签一一映射到 int 类型的标签中.

Step3:训练集和测试集的划分:

数据集中 speech_commands/test 目录下的.wav 文件是没有分类标签的,不能适应本次实验的源码,我选择了放弃 test 目录,将/train 目录下的所有文件按 0.8:0.2 的比例随机划分

Step4：独立同分布采样

对 dataset 直接调用现有的 cifar_iid()得到 user_groups 和 user_goup_test，实现对数据集的独立同分布分区采样，将 i 每个分区随机划分到每个 groups 中，符合原文中 A.4 的设置.

Speech_commands 数据的导入和预处理这一部分的源代码见附录.

A.4 Data Partitioning

In most of the experiments, there are 40 clients in the FL system. This scale is larger than or comparable to many studies on FL [34, 49, 60]. The training set is evenly partitioned across 40 partitions. Under the IID setting, the training dataset is divided into 40 partitions and each client is randomly assigned one data partition. Under the non-IID setting, the training dataset is divided into multiple partitions, each containing one class of training samples. Each client is randomly assigned two data partitions.

图 2.6 原文 A.4 对数据分区的描述

3.实验训练结果

7.13 日-7.26 日,依次完成了以下实验项目:

序号	数据集	模型	方法	日期	epochs
1	CIFAR-10	VGG	Basic-Common	2023 年 7 月 20 日	502
2	CIFAR-10	VGG	Clustered-Common	2023 年 7 月 21 日	502
3	CINIC-10	VGG	Basic-Common	2023 年 7 月 21 日	502
4	CINIC-10	VGG	Basic-Common	2023 年 7 月 21 日	800
5	CINIC-10	VGG	Basic-Common	2023 年 7 月 23 日	1501
6	CINIC-10	VGG	Basic-Common	2023 年 7 月 25 日	3001
7	CINIC-10	VGG	Basic-Common	2023 年 7 月 21 日	800
8	CINIC-10	VGG	Basic-Common	2023 年 7 月 21 日	800
9	Speech Commands	VGG	Basic-Common	2023 年 7 月 25 日	501
10	Speech Commands	VGG	Basic-Common	2023 年 7 月 25 日	501
11	Speech Commands	VGG	Basic-Common	2023 年 7 月 25 日	501

表 2.1 第二,三周实验汇总

当前总实验任务的结果见下表：

数据集	模型	方法	epochs	版本			
				V1	V2	V3	V4
CIFAR-10	VGG	Basic-Common	502	64.2	64.3	65.1	66.2
		Clustered-Common	502	71.2	74.7	76.5	79.8
		Max-Common	502	72.9	77.2	79.3	81.1
	ResNet	Basic-Common	502				
		Clustered-Common	502				
		Max-Common	502				
CINIC-10	VGG	Basic-Common	502	48.2	50.2	48.3	47.3
		Basic-Common	800	42	43.8	45.7	40
		Basic-Common	1501	43.13	47.21	45.18	41.32
		Basic-Common	3001	43.3	47.1	45	42.9
		Clustered-Common	800	54.2	54.3	56.6	57.5
		Max-Common	800	55.3	62.1	64.1	64.7
	ResNet	Basic-Common					
		Clustered-Common					
		Max-Common					
AG NEWS	CharCNN	Basic-Common					
		Clustered-Common					
		Max-Common					
	VDCNN	Basic-Common					
		Clustered-Common					
		Max-Common					
Speech Commands	VGG	Basic-Common	501	12.2	13.3	10.6	11.9
		Clustered-Common	501	20.2	22	17.4	11.7
		Max-Common	501	20.8	21.7	18.4	12
	ResNet	Basic-Common					
		Clustered-Common					
		Max-Common					

表 2.2 第三周总实验结果表-周臻鹏

表格中标红的数据表示训练者(我)认为该数据不符合理论预期，很可能存在误差或者错误

Dataset	Models	Scheme	Versions			
			V1	V2	V3	V4
CIFAR-10 [24]	VGG [46]	Standalone	64.4	64.8	65.2	65.6
		Clustered-FL	78.2	80.4	80.8	81.2
		Basic-Common	65.2	66.8	67.2	68.2
		Clustered-Common	80.2	82.4	83.2	83.6
		Max-Common	80.6	85.2	86.6	86.8
	ResNet [14]	Standalone	57.2	57.8	58.8	59.2
		Clustered-FL	73.6	74.6	75.2	75.8
		Basic-Common	66.4	67.6	67.8	68.4
		Clustered-Common	78.4	79.2	80.6	80.8
		Max-Common	78.8	79.6	83.4	84.2
CINIC-10 [8]	VGG	Standalone	44.4	45.6	47.4	49.2
		Clustered-FL	58.8	59.8	60.4	60.6
		Basic-Common	48.8	50.8	51.2	51.4
		Clustered-Common	60.8	62.8	63.4	63.8
		Max-Common	61.4	67.6	67.8	68.2
	ResNet	Standalone	46.2	47.2	47.8	49.4
		Clustered-FL	58.2	58.6	59.8	60.4
		Basic-Common	49.8	51.4	51.8	52,2
		Clustered-Common	60.8	61.4	63.4	64.2
		Max-Common	61.6	64.2	66.2	66.6
AG NEWS [65]	CharCNN [65]	Standalone	51.9	53.2	65.1	70.2
		Clustered-FL	76.2	77.7	84.1	84.7
		Basic-Common	84.6	85.6	85.7	86.1
		Clustered-Common	85.2	85.6	86.1	86.3
		Max-Common	85.9	86.5	86.7	87.6
	VDCNN [6]	Standalone	73.2	75.8	77.8	78.6
		Clustered-FL	84.6	85.2	86.2	86.4
		Basic-Common	78.2	79.6	80.4	80.6
		Clustered-Common	84.8	86.6	87.8	88.2
		Max-Common	88.2	89.2	89.6	90.2
Speech Commands [54]	VGG	Standalone	77.5	78.2	80.5	81.5
		Clustered-FL	80.2	81.8	83.4	85.8
		Basic-Common	78.6	80.4	81.2	82.4
		Clustered-Common	82.4	84.6	86.2	86.4
		Max-Common	82.6	86.6	87.4	89.8
	ResNet	Standalone	75.2	78.6	79.4	82.8
		Clustered-FL	79.2	82.0	83.6	84.2
		Basic-Common	83.6	87.6	88.2	89.2
		Clustered-Common	84.8	88.8	89.2	89.8
		Max-Common	85.2	89.6	90.8	91.4

表 2.3 原文实验结果图对照

在每次训练完成时,我会把 8 个 user 的 acc 结果保存在 table.txt 中,根据此来绘制不同版本 model 的 acc 曲线图,每次绘制时,将 4 个版本的曲线绘制在一个 plt 中,结果可见附件" **第三周实验复现结果曲线图汇总-共 12 项.pdf**".

从上表可以看出,我复现的实验结果和原文的结果仍然存在差异,V1-V4 四个版本的 VGG 模型结果都或多或少低于原文的结果,

其中 CINIC-10 数据集在 VGG19 模型上问题最大,曲线没有收敛,即使 epochs 设为 3001,依然没有明显收敛,有待探究.

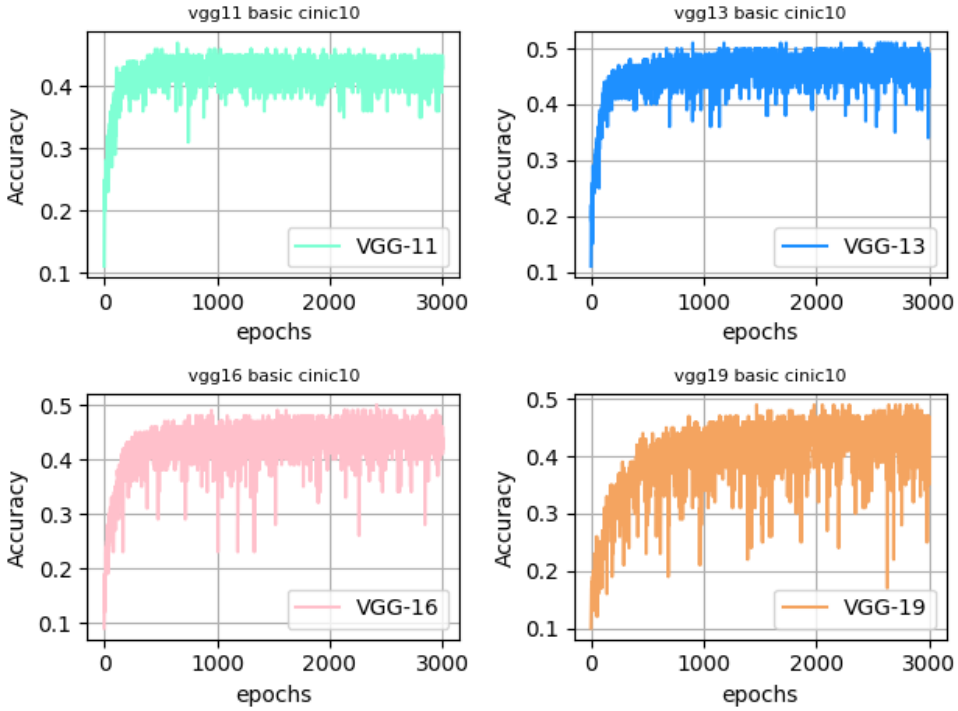

图 2.7 CINIC-10 数据集在 VGG19 模型上训练未收敛

三. 遇到的问题

1. CINIC-10 数据集在 VGG19 模型下 acc 曲线不收敛

在二.3 中有描述过 CINIC-10 数据集在 VGG19 模型中训练时无法收敛的情况,我不断增大 epoch 数从 502-800-1501-3001,一直都不收敛,这种情况是由什么导致的?应该如何调整参数?

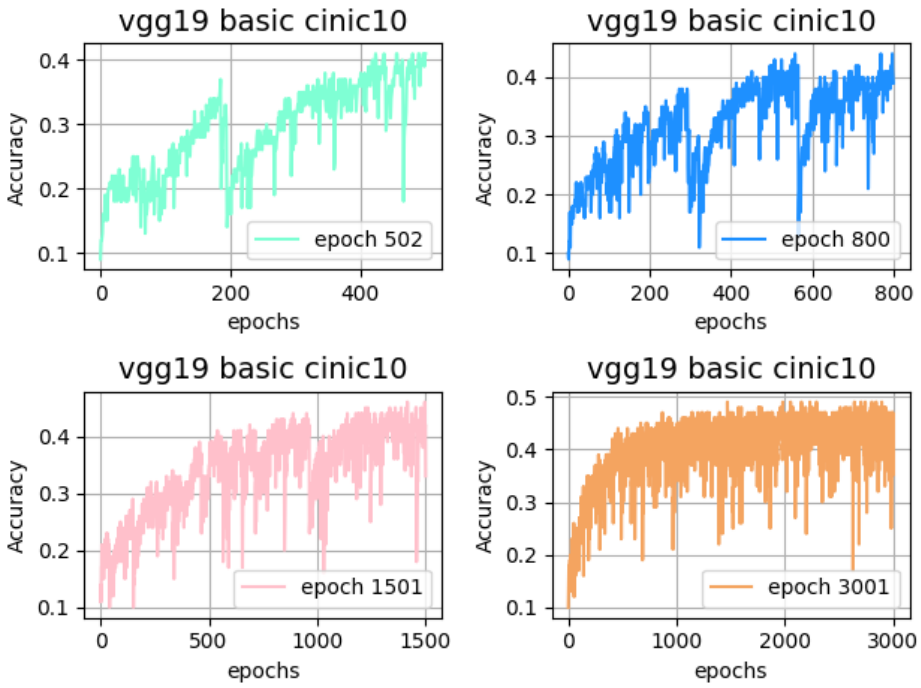

图 4.1 CINIC-10 on VGG19 四次不同 epoch

2. 如何设置一次训练所需 epoch

在训练 speech_commands 时,我按照源代码默认的参数给定 epoch=502, batch_size=64,运行时大约每个 global epoch 用时 25s,最终曲线很明显,4 个版本的 VGG 模型均未收敛,需要更多的 epoch 数,目前仍然还在训练.之后如果训练新模型,新数据集可能也会出现这种情况,那么该如何根据自己的模型和数据集确定 epoch 参数的大致范围呢?

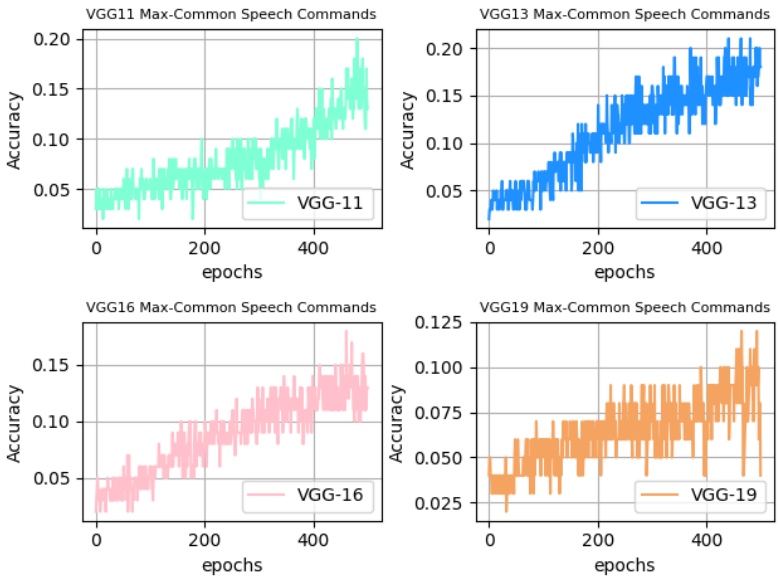

图 4.2 speech_commands-max_common-epoch502 第一次训练

3. 如何根据训练结果确定最终 acc 的数值

我在训练过程中导出了每个 epoch 时每个 user 的 accuracy 数值，存储在一个二维 list 中用于训练结束后绘制 acc 曲线，在填写结果表时，每个版本的模型只用填写一个数值作为最终结果 final_result，在之前的实验时，我自己认为 final_result 应该是模型收敛后的一个均值，所以采取了对最后 50 个 epoch 下的 acc 取均值最为 final_result.

但是在网上提问时，我看到了取训练中最高值的作为 final_result 的观点：

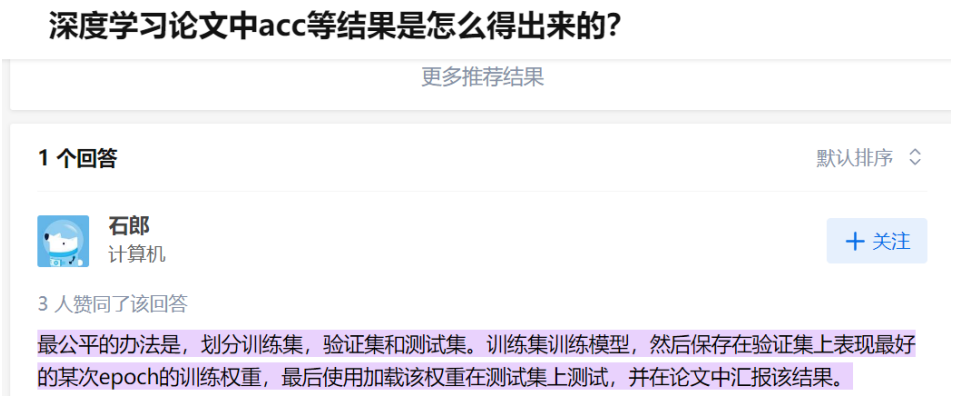

图 4.3 如何取 final_result

在实际的研究工作中，我们一般用哪种方法取得最终的结果呢？如果时采用取 max 的方法，我的数据结果就会比现在高一点，更接近原文的实验结果.

四. 总结与思考

这两周的实验工作对于我来说可能异常艰难,每天不断的遇到问题-分析问题-解决问题,高强度的刷新自己的知识认知,虽然大部分只是 pytorch 工程上的应用知识,但是对于之后的实验实践具有奠基的意义.每天我把自己的遇到的问题记录下来,有的一时半会想不通就挂一会,过段时间重新又来思考.

图 5.1 部分每日问题记录

在记录中80%的问题随着对模型工程的逐步了解和对官方文档说明的阅读都悉数被解开,这一过程看似浪费了很多时间,效率低下,拖慢了整体进度,但是脚踏实地自己探究实际问题让我迅速地增长了入门的知识经验,印象深刻难以遗忘.专注执着于问题克服它,持之以恒不放弃,最终在实现目标之后也是让我获得巨大的成就感.

但是从一方面讲,正是因为机器学习实践能力的匮乏,前期在一些代码实现和调试上花了太多时间,加上目前使用的数据集体量大,训练时间长,整体的进度已经被严重拖慢了,截止目前为止总共完成了12次训练,剩余15次训练待完成,之后将会继续加快进度.其中模型训练的结果部分并不是与原文结果相近,排除随机误差的可能,我猜测也有因为我自己编写程序错误带来的人为误差,目前我自己还没有发现,希望能得到老师指出.

当前我的源码github地址:

https://github.com/Yogurteer/repeat-Flexifed

附录

1.Clustered-Common 方法的实现

```
def compare(w1, w2):
    if len(w1) != len(w2):
        return False
    keys1 = list(w1.keys())
    keys2 = list(w2.keys())
    # print("compare\n",keys1, '\n', keys2)
    for j in range(len(keys1)):
        if keys1[j] == keys2[j]:
            continue
        else:
            return False
    return True



def FedAvg(w):
    w_avg = copy.deepcopy(w[0])
    for k in w_avg.keys():
        # print("len of w", len(w))
        for i in range(1, len(w)):
            w_avg[k] += w[i][k]
        w_avg[k] = torch.div(w_avg[k], len(w))
    return w_avg
```

```
def common_clustered(w):
    """
    step1:对 w 使用 common_basic,聚合所有的全局公共基础层
    step2:将结构完全一致的 model 划分集群
    step3:对每个集群采取 FedAvg,得到一个聚合后的 w_avg,赋值给该集群中的所有
model
    """
    # step1:对 w 使用 common_basic,聚合所有的全局公共基础层
    w_basic = common_basic(w)
    # step2:将结构完全一致的 model 划分集群
    clu = [[]]
    cluids = [[]]
    c_fid = 0
    find = 0
    for i in range(len(w_basic)):
        # 第一个
        if len(clu[0]) == 0:
            clu[c_fid].append(w_basic[i])
            cluids[c_fid].append((i))
            # clu[c_fid][i] = w_basic[i]
            continue
        # 找到同类
        for j in range(len(clu)):
            # c1=clu[j].get(next(iter(clu[j])))
            c1 = clu[j][0]
            if compare(c1, w_basic[i]):
                # compare
                clu[j].append(w_basic[i])
```

```
                cluids[j].append((i))
                find = 1
                break
        # 找不到同类
        if find == 0:
            clu.append([w_basic[i]])
            cluids.append(([i]))
        find = 0
    # 此时集群划分已完成，存储在 clu 中

    # step3: 对每个集群采取 FedAvg, 得到一个聚合后的 w_avg, 赋值给该集群中的所有
model
    for i in range(len(clu)):
        avg = FedAvg(clu[i])
        for j in range(len(clu[i])):
            clu[i][j] = copy.deepcopy(avg)
    # 最终汇总输出 w
    w_out = {_id: None for _id in range(8)}
    id_out = 0
    for i in range(len(clu)):
        for j in range(len(clu[i])):
            out_index = cluids[i][j]
            w_out[out_index] = copy.deepcopy(clu[i][j])
            id_out += 1

    return w_out
```

2.Speech_Commands 数据集的导入和预处理

```
from torch.utils.data import DataLoader, Dataset
import torch
from torch import nn
import torch.nn.functional as F
import copy
from collections import defaultdict
import numpy as np
import copy
import math
from torchvision import datasets, transforms
import os
import fnmatch
import librosa


os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"



def cifar_iid(dataset, num_users):
    """
    主实验中默认使用 iid 划分方式，文章中写 40 个用户随机分配 40 个独立同分布的分区，种类打乱
    Sample I.I.D. client data from CIFAR10 dataset
    :param dataset:
    :param num_users:
    :return: dict of image index
    """
```

```
    num_items = int(len(dataset) / num_users)
    dict_users, all_idxs = {}, [i for i in range(len(dataset))]
    for i in range(num_users):
        dict_users[i] = set(np.random.choice(all_idxs, num_items,
                                             replace=False))
        all_idxs = list(set(all_idxs) - dict_users[i])
    return dict_users


def get_dataset(dataset_name):
    """
    主实验中默认使用 iid 划分方式，文章中写 40 个用户随机分配 40 个独立同分布的分区，种类打乱
    Sample I.I.D. client data from CIFAR10 dataset
    :param dataset:
    :param num_users:
    :return: dict of image index

    实现方法：
    dict_users 字典中的每一个元素是一个 set 集合,set 不可以取重复元素
    all_idxs 初始化为 0-样本总数的 list
    num_items 表示每个分区中的样本个数
    已知有 num_users 个分区
    每次分区时,从当前的 all_idxs 中随机取 num_items 个数表示已加入的样本序号,序号不可重复
    分区加入序号之后,all_idxs 列表中删除当前分区中的 num_items 个元素,表示分完当前区后剩下的总样本
    """
```

```
    if dataset_name == "speech_commands":
        speech_directory = "/root/autodl-tmp/speech_commands"
        # train_dataset = Audio_DataLoader(speech_directory + '/train',
sr=16000, dimension=8192)
        # test_dataset = Audio_DataLoader(speech_directory + '/train',
sr=16000, dimension=8192)

        dataset = Audio_DataLoader(speech_directory + '/train', sr=16000,
dimension=8192)
        # print(dataset.wav_list)
        train_size = int(len(dataset) * 0.8)
        test_size = len(dataset) - train_size
        # 随机划分数据集和测试集
        train_dataset, test_dataset = torch.utils.data.random_split(dataset,
[train_size, test_size])
        user_groups = cifar_iid(train_dataset, 20)  # 存储每个分区的样本序号的
set 组成的 dict
        user_groups_test = cifar_iid(test_dataset, 20)  # 存储每个分区的样本序
号的 set 组成的 dict
        return train_dataset, test_dataset, user_groups, user_groups_test

class DatasetSplit(Dataset):  # DatasetSplit 类继承自父类 Dataset,按照 idxs 顺序
有序排列
    def __init__(self, dataset, idxs):
        self.dataset = dataset
        self.idxs = list(idxs)

    def __len__(self):
```

```
        return len(self.idxs)

    def __getitem__(self, item):
        image, label = self.dataset[self.idxs[item]]
        # image = self.dataset[self.idxs[item]]
        return image, label


class DatasetSplitSpeech(Dataset):
    def __init__(self, dataset, idxs):
        self.dataset = dataset
        self.idxs = list(idxs)

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

    def __getitem__(self, item):
        image = self.dataset[self.idxs[item]]
        return image

class Audio_DataLoader(Dataset):
    def __init__(self, data_folder, sr=16000, dimension=8192):
        self.data_folder = data_folder
        self.sr = sr
        self.dim = dimension
        self.labellist = []
        # 获取音频名列表
        self.wav_list = []
```

```
        for root, dirnames, filenames in os.walk(data_folder):
            for filename in fnmatch.filter(filenames, "*.wav"):  # 实现列表特殊字符的过滤或筛选,返回符合匹配".wav"字符列表
                self.wav_list.append(os.path.join(root, filename))
        # print("len of wav list:{}\n{}".format(len(self.wav_list),self.wav_list[0]))

    def __getitem__(self, item):
        # 读取一个音频文件，返回每个音频数据
        filename = self.wav_list[item]
        # print("cur label:{}".format(filename))
        # print(filename)
        wb_wav, _ = librosa.load(filename, sr=self.sr)
        # sr 为采样率，通过 KMplayer 查看 sampling rate，确认过 speech commands 为 16000

        # 取 帧
        if len(wb_wav) > self.dim:  # self.dim=8196
            # print("yes:len of wb_wav{}:{}".format(filename, len(wb_wav)))
            max_audio_start = len(wb_wav) - self.dim
            audio_start = np.random.randint(0, max_audio_start)
            wb_wav = wb_wav[audio_start: audio_start + self.dim]
        else:
            wb_wav = np.pad(wb_wav, (0, self.dim - len(wb_wav)), "constant")
        wb_wav.dtype = np.float32
        wav = torch.tensor(wb_wav)
        wav = wav.view(-1, 32, 32)
        wav = wav.mul(1000)
```

```
mean = -0.0756
std = 95.2344
"""
先标准化
mean tensor(-0.0756)
std tensor(95.2344)
"""
wav = wav.add(mean)
wav = wav.div(std)
"""
后归一化
"""
min = -10.501201629638672
max = 10.499293327331543
dif = max-min
wav = wav.add(0-min)
wav = wav.div(dif)
label_d = filename
labels_d =label_d.split('/')
label = labels_d[-2]
# label = int(1)
label_dict = {
    'bed':0,
    'bird':1,
    'cat':2,
    'dog':3,
    'down':4,
    'eight':5,
```

```
    'five':6,
    'four':7,
    'go':8,
    'happy':9,
    'house':10,
    'left':11,
    'marvin':12,
    'nine':13,
    'no':14,
    'off':15,
    'on':16,
    'one':17,
    'right':18,
    'seven':19,
    'sheila':20,
    'six':21,
    'stop':22,
    'three':23,
    'tree':24,
    'two':25,
    'up':26,
    'wow':27,
    'yes':28,
    'zero':29,
}

numlabel = label_dict[label]
return wav, numlabel
```

```
    def __len__(self):
        # 音频文件的总数
        return len(self.wav_list)


def utils():
    return None
```