本科创新实验报告

实验题目：基于联邦学习的智能笔记本

学生姓名：白文强

学　　号：20191060064

专　　业：计算机科学与技术

指导教师：武浩

评分（百分制）：

2022 年 6 月 13 日

目录

一、实验目的.....5
二、实验内容.....6
三、实验环境、平台及语言.....6
（一）实验环境.....6
（二）实验平台.....7
（三）实验语言.....7
四、实验原理.....7
五、实验步骤.....8
（一）下载数据集.....8
（二）数据预处理.....8
（三）联邦学习模型的构建.....9
（四）联邦学习模型的训练.....9
（五）非联邦学习模型的训练.....10
六、实验结果.....10
七、实验小结.....12
八、参考文献.....13
教师评价.....15

一、实验目的

当今时代，数据呈现出了指数级别增长的速度，但同时，数据的隐私也越来越受重视[1]。当前各大互联网应用与公司滥用用户隐私为自己的业务服务的状况越来越猖獗，比如，有时我们在某个平台搜索或查询过某个商品时，过一段时间，在其他的平台上就会出现系统给我们自动推送了该商品[2]；甚至在有些时候，软件可以偷偷监听用户所说的话，提取其中的商品信息，从而在购物平台上将该商品推送给用户。这样纵然说明了我们大数据技术及推荐系统的高度发展，但是在其发展的过程中，由于法律法规的不够完善以及隐私保护技术的不够成熟，用户隐私被泄露以及被滥用的情况常常发生，被网络用户深恶痛绝，发出了：“在‘互联网’下没有隐私可言，我们在‘互联网’中都是赤身裸体的”的感慨。

近些年来，随着隐私安全越来越受到重视，隐私保护技术也发展地越来越快。其中，联邦学习技术[3]就是一个革命式的技术。该技术可在数据不共享的情况下完成联合建模。具体来讲，各个数据拥有者（个人/企业/机构）的自有数据不会离开本地，通过联邦系统中加密机制下的参数交换方式（即在不违反数据隐私法规的情况下）联合建立一个全局的共享模型，建好的模型在各自的区域只为本地的目标服务。

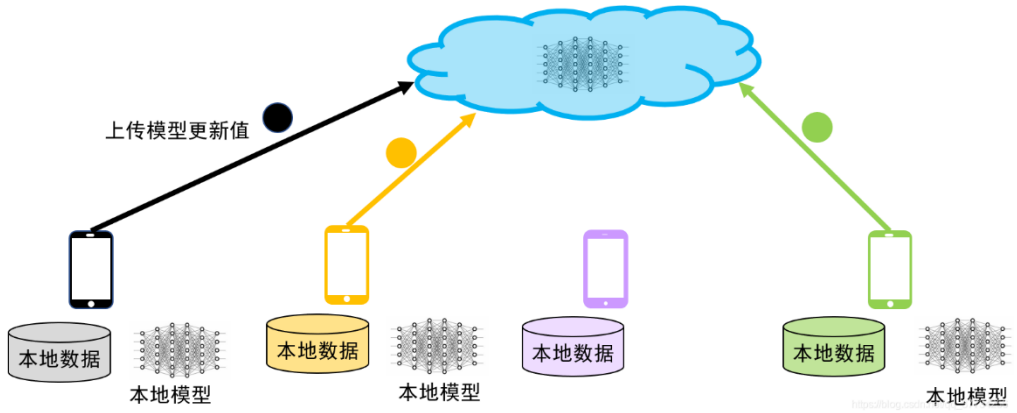

图 1：联邦学习概念

联邦学习解决了数据孤岛[4]问题，可以在不获得用户数据的情况下，使用用户的设备进行模型训练，可以解决用户隐私泄露的问题。在本次创新实验

中，我以我们日常使用的输入法作为原型，以基于 GPT-3[5]模型的 Github Copilot 智能代码提示插件作为灵感，实现了一个文本生成模型，可以根据前面的文本输入预测用户接来下想要输入的内容，在保护用户隐私的情况下为用户提供优质的服务。

二、实验内容

（一）设计一个具有文本生成功能的神经网络，

（二）使用联邦学习的方法进行模型训练，验证联邦学习的可行性

（三）使用训练好的模型进行文本预测，查看模型的训练效果

（四）使用非联邦学习的方式，使用相同的数据集对模型进行训练，并进行文本预测，观察使用联邦学习方式训练的模型以及使用非联邦学习方式训练的模型的训练效果，对比二者的不同并分析。

三、实验环境、平台及语言

（一）实验环境

- Anaconda3

实验采用了 Anaconda3 环境，Anaconda 是一个 Python 包管理环境，可以在图形化界面下建立多个基于不同版本 Python、包含不同 package 的开发环境，且这些开发环境相互独立，随时可以添加和删除，极大地方便了 Python 编码和神经网络训练。

- TensorFlow2.8.0

TensorFlow 是一个开源的，基于 Python 的机器学习框架，由 Google 开发，在图形分类、音频处理、推荐系统和自然语言处理等场景下有着丰富的应用，是目前最热门的机器学习框架之一。

- TensorFlow Federated 0.20

TensorFlow Federated 是正在开发中的联邦学习框架，用于对分散式数据进行机器学习和其他计算，使得跨客户端训练全局共享的模型成为可能，让训练数

据留存在本地而不上传到云端，用户的隐私数据得到了安全的保障。

（二）实验平台

由于缺少高算力 GPU，本次实验采用 Google 旗下的 Colab 平台，该平台提供了一定量的存储服务以及 GPU 算力服务，使用 Colab 可以使用 GPU 来大大提升模型训练速度，减少训练时间。

（三）实验语言

实验语言采用 Python，Python 具有简单易学、语法优美、开源库丰富的优点。在深度学习领域得到了广泛的应用，支持多种深度学习框架。

四、实验原理

本次实验采用横向联邦学习[6]。

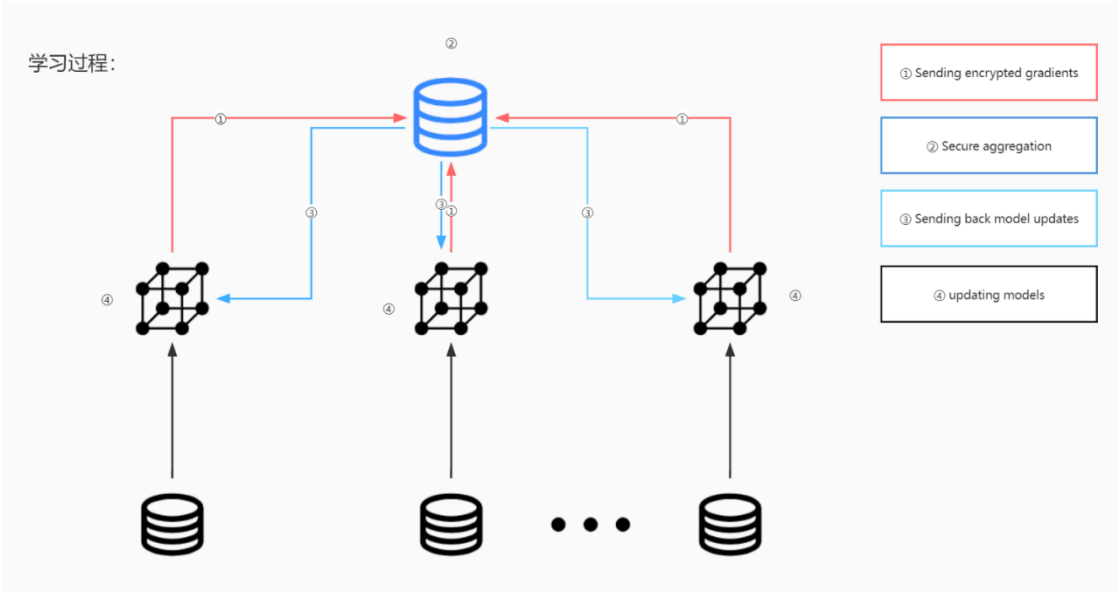

图 2：联邦学习

Step1: 参与方各自从服务器 A 下载最新模型；

Step2: 每个参与方利用本地数据训练模型，加密梯度上传给服务器 A，服务器 A 聚合各用户的梯度更新模型参数；

Step3: 服务器 A 返回更新后的模型给各参与方；

Step4: 各参与方更新各自模型。

横向联邦学习适用于参与者的数据特征重叠较多，而样本 ID 重叠较少的情况，例如，两家不同地区的银行的客户数据，联合多个参与者的具有相同特征的多行样本进行联邦学习，即各个参与者的训练数据是横向划分的，称为横向联邦学习。

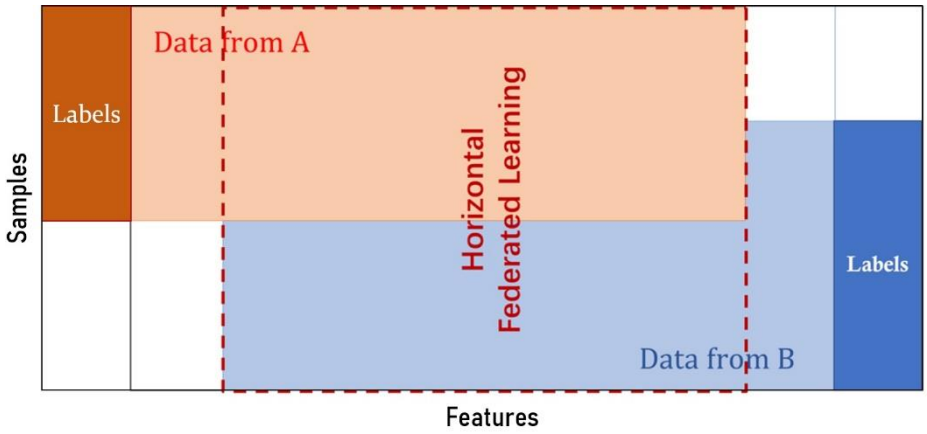

图 3：横向联邦学习

在模型的选择上，采用 GRU[7]作为主要单元，GRU 是循环神经网络的一种，为了解决标准 RNN 的梯度消失问题，GRU 使用了更新门（update gate）与重置门（reset gate）。这两个门控向量决定了哪些信息最终能作为门控循环单元的输出。这两个门控机制的特殊之处在于，它们能够保存长期序列中的信息，且不会随时间而清除或因为与预测不相关而移除。

训练模型要执行的任务是：给定一个字符或者一个字符序列，预测下一字符应该是什么。可见字符是有限个数的，模型每次输出一个字符，随后将预测的字符加上前面的字符再次作为输入进行预测，逐步输出一个字符序列。

五、实验步骤

（一）下载数据集

我采用了莎士比亚全集（shakespeare.txt）作为训练数据集，该数据集收录了莎士比亚 37 部戏剧的文本，使用联邦学习的方法时，将每一个作品视为一个本地终端，将作品中的文本作为其本地数据，模拟每一个终端在本地使用自己的数据训练自己的模型，在中央服务器中进行模型的合并。

（二）数据预处理

将数据集下载完毕之后，对全部文本进行去重工作，得到文本中所有可能的字符类型。随后，在训练之前，我们需要将字符串映射到数字表示值，那么需要有两个字典，分别用于将字符映射到数字以及将数字映射到字符。

（三）联邦学习模型的构建

在模型上我们首先使用一个 Embedding 层将输入文本的向量映射到相同的维度，随后是一个 GRU 层，用于文本生成。随后是一个 Fully-Connected 层将 GRU 的输出映射到一个与字符个数相同的维度的向量中。

```
def create_keras_model(vocab_size, embedding_dim, units, batch_size):
    model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim,
                              batch_input_shape=[batch_size, None]),
    tf.keras.layers.GRU(units,
                        return_sequences=True,
                        stateful=True,
                        recurrent_initializer='glorot_uniform'),
    tf.keras.layers.Dense(vocab_size)
  ])

    return model
```

（四）联邦学习模型的训练

模型超参数，在本实验中，模型超参数的选择为：embedding_dim＝256，即将每一个字符串对应的向量映射为一个 256 维的向量；united=1024，即 GRU 中单元的个数为 1024；batch-size=8，即每一个 batch 包含 8 组训练数据；num_epoches=30，将整个数据集训练 30 次。

在损失函数的选择上，我们采用了 SparseCategoricalCrossentropy()损失函数：

$$Q = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \log\left(f(x_i)\right) + (1-y_i)\log\left(1-f(x_i)\right)\right) \qquad (1)$$

该损失函数常用于计算多分类问题的交叉熵，每一个生成的字符相当于是在有限个字符中选择一个进行生成，也就是将输入文本进行“分类”，因此，使用该损失函数比较合适。

训练模型采用的优化器是 Adam[8]优化器，在客户端，设置的学习率即 learning_rate=0.1；在服务器端，设置的学习率为 0.5。

```
round  2, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.11824534)
round  3, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.13409938)
round  4, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.15), ('lo
round  5, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.16374223)
round  6, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.17624223)
round  7, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.18326087)
round  8, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.18346274)
round  9, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.18781056)
round 10, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.19049689)
round 11, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.19462733)
round 12, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.20122671)
round 13, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.19759317)
round 14, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.20669255)
round 15, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.20321429)
round 16, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.20403726)
round 17, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.21141304)
round 18, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.20936336)
round 19, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.21468943)
round 20, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.21509317)
round 21, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.21656832)
round 22, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.21782608)
round 23, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.22091615)
round 24, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.22026397)
round 25, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.22209628)
round 26, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.2222826),
round 27, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.22389752)
round 28, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.22604038)
round 29, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.22715838)
round 30, metrics=OrderedDict([('broadcast', ()), ('aggregation', OrderedDict([('mean_value', ()), ('mean_weight', ())])), ('train', OrderedDict([('accuracy', 0.2298913),
```

图 4：联邦学习训练过程

如上图所示，在进行联邦学习训练中，随着训练轮次的不断增加，模型的准确度也在不断增加。

（五）非联邦学习模型的训练

在使用相同的数据集与相同模型超参数的情况下，我们使用了非联邦学习的方式，即集中式学习对模型进行了训练。

```
[28]  1 history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])

    Epoch 1/30
    172/172 [==============================] - 14s 51ms/step - loss: 2.6775
    Epoch 2/30
    172/172 [==============================] - 12s 53ms/step - loss: 1.9682
    Epoch 3/30
    172/172 [==============================] - 10s 50ms/step - loss: 1.7045
    Epoch 4/30
    172/172 [==============================] - 10s 51ms/step - loss: 1.5544
    Epoch 5/30
    172/172 [==============================] - 10s 52ms/step - loss: 1.4655
    Epoch 6/30
    172/172 [==============================] - 10s 53ms/step - loss: 1.4039
    Epoch 7/30
    172/172 [==============================] - 10s 54ms/step - loss: 1.3570
    Epoch 8/30
    172/172 [==============================] - 10s 54ms/step - loss: 1.3183
    Epoch 9/30
    172/172 [==============================] - 11s 55ms/step - loss: 1.2831
    Epoch 10/30
    172/172 [==============================] - 11s 56ms/step - loss: 1.2511
    Epoch 11/30
    172/172 [==============================] - 11s 57ms/step - loss: 1.2183
    Epoch 12/30
    172/172 [==============================] - 11s 59ms/step - loss: 1.1858
    Epoch 13/30
    172/172 [==============================] - 11s 57ms/step - loss: 1.1527
    Epoch 14/30
    172/172 [==============================] - 11s 57ms/step - loss: 1.1190
```

图 5：非联邦学习训练过程

六、实验结果

在使用联邦学习的方式时，在验证集上的模型准确度达到 25%；但是在使用非联邦学习的方式时，模型的准确度达到了 82%。我们使用两个模型分别进行文本预测，效果如下：

```
4 print(generate_text(keras_model_batch1,  'ROMEO:  '))
```

```
ROMEO: ANDART

It was the Tribunals for freathered the lines of the room
de the dicless, accept followed up
Shottles that had held the white hair,
and by dark hair was to write under sight of Republican
```

图 6：联邦学习模型预测结果

```
[33]  1 print(generate_text(model,  start_string=u"ROMEO:  "))

ROMEO: I, my lord.

KING RICHARD II:
Romeo is banished,' that often
Will burst thy windows sit and earth,
Will but as after, who reason here?

ANGELO:
F it had been higher consent.

LEONTES:
Nevelloused man! Wolter ague?

ESCALUS:
Come away; I will hence forthwick'd Hereford!
How does my father's heirs?

LORD ROSS:
To-morrow, that was with his dreams:--
A hall, after all the fatal
colours, of all wick, was his office for't her!
```

图 7：非联邦学习模型预测结果

从直观上可以感受到，使用非联邦学习方式预测出的文本更像是莎士比亚的语言，在意思上也更具有逻辑性，预测效果更好。

为了排除使用联邦学习训练的模型效果不好的原因是代码错误的问题，我们使用了 Tensorflow Federated 开源项目中教程中的文本预测的例子，将数据集以及模型等改成我们模型中的对应数据，重新对模型进行训练，训练过程如下：

```
Round 57
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
        Eval: loss=2.721, accuracy=0.264
        Train: loss=2.617, accuracy=0.290
Round 58
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
        Eval: loss=2.790, accuracy=0.233
        Train: loss=2.642, accuracy=0.287
Round 59
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
        Eval: loss=2.778, accuracy=0.226
        Train: loss=2.630, accuracy=0.292
Final evaluation
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer gru_1 will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running on GPU.
        Eval: loss=2.732, accuracy=0.235
```

图 8：官方示例训练过程

可以看到，即使模型训练了 60 轮次，在验证集上的准确度仍然只是 25%左

右。因此，我们排除了通过联邦学习方式训练的模型效果不好的原因是代码错误。

深究原因，我们认为这可能与经典联邦学习 FedAvg[9]有关，模型训练的目通常是最小化该目标函数：

$$\min_w F(w) = \sum_{k=1}^{m} p_k F_k(w) \quad (2)$$

其中，m 表示设备数量，F_k是各个客户端的局部目标函数，p_k为客户端对应的权重。局部目标函数的优化处理过程为：

$$F_k(w) = \frac{1}{n_k} \sum_{j_k=1}^{n_k} f_{j_k}(w) \quad (3)$$

其中，n_k 为第 k 个客户端局部样本数据数量，可以令 p_k=n_k / n，n 为整个联邦学习网络的数据集中符合经验最小化目标的样本总数。传统方法通过以下方式实现全局目标最优化：每一轮选择概率与 n_k 成正比的设备子集执行这些本地更新方法通过在每个设备上本地运行可变数量的迭代的优化器（例如 SGD）来实现灵活高效的通信。

FedAvg 的一个缺点是直接对模型参数进行加权平均，可能会对模型性能产生严重的不利影响，并显著增加通信负担，而这一问题主要是由于神经网络参数的置换不变性而导致的[10]。比如，模型训练后的有些参数会在不同的变体中处于不同的位置，因此，直接对模型进行基于参数位置的加权平均可能使得某些参数失效，从而导致模型训练效果不够好。

七、实验小结

联邦学习是近几年来谷歌提出的一个可以在多客户端训练同一个模型的概念。在联邦学习中，多个客户端以分散的方式进行模型学习。该学习交由客户端负责，受信任的管理员只需集中收集学习参数。接着，管理员将集计模型分发给客户机。基于计算能力和隐私性，联邦学习方法可在手机应用中广泛使用。

在本次实验中，我们使用基于 Tensorflow 的 Tensorflow Federated 框架，使

用莎士比亚著作作为数据集，以 GRU 作为模型主要单元，在仿真的方式下，实现了一个通过多端设备训练出的用于文本预测的模型。

目前联邦学习发展不成熟，相关框架也不健全，在实验过程中，我们尝试过 FATE 框架、Pysyft 框架、FedLab 框架以及 Tensorflow Federated 框架，这些框架目前都在快速迭代中，版本发布较快，基本都存在文档不健全、文档与框架不同步、小版本之间不兼容的问题。Tensorflow Federated 框架作为相对发展较快的框架，也有功能不健全的问题，目前框架无法部署到移动终端设备上，因此，我们使用了框架中的仿真模块，对联邦学习过程进行模拟。

在使用联邦学习的方式对模型进行训练时，由于涉及到多端设备之间的通信，因此，用联邦学习的方式训练模型会占用大量的通信资源，导致模型的训练速度变慢。由于目前联邦学习算法的不够完善，所以在相同的条件下（数据集、模型、训练超参数等）使用联邦学习方法训练的模型比不使用联邦学习方法训练的模型效果会差很多，期待随着该领域的不断发展，这种差距会越来越小。

八、参考文献

[1]庭静,彭寻启,董容语,张薇. 做好大数据时代下的个人隐私保护[N]. 贵阳日报,2022-05-27(007).DOI:10.28295/n.cnki.ngyrb.2022.001555.

[2]胡林果,张艺腾. 警惕透明 App 不法软件：偷电量、偷流量、偷隐私……[N]. 新华每日电讯,2022-05-20(007).DOI:10.28870/n.cnki.nxhmr.2022.003786.

[3]Yang Q, Liu Y, Cheng Y, et al. Federated learning[J]. Synthesis Lectures on Artificial Intelligence and Machine Learning, 2019, 13(3): 1-207.

[4]Li Q, Diao Y, Chen Q, et al. Federated learning on non-iid data silos: An experimental study[J]. arXiv preprint arXiv:2102.02079, 2021.

[5]Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33: 1877-1901.

[6]Yang Q, Liu Y, Chen T, et al. Federated machine learning: Concept and applications[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2019, 10(2): 1-19.

[7]Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.

[8]Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[9]McMahan B, Moore E, Ramage D, et al. Communication-efficient learning of deep networks from decentralized data[C]//Artificial intelligence and statistics. PMLR, 2017: 1273-1282.

[10]Wang H, Yurochkin M, Sun Y, et al. Federated learning with matched averaging[J]. arXiv preprint arXiv:2002.06440, 2020.

教师评价

评分	
评语	任课教师：　　日期：