数据隐私的方法、伦理和实践

PB18000290 胡毅翔

差分隐私在联邦学习中的应用

基本内容

实验原理

见实验文档

实验结果

在DP机制中，client端不需要修改，只须在server端的计算中进行修改。

需要修改的部分为（具体代码&注释，见源代码部分）：

1. 聚合操作中，进行截断，截断参数为1与2范数除以参数C之间的最大值。
2. 在聚合结束后，取平均之前进行加噪（高斯噪声）处理。

使用不同的参数sigma和C，运行模型，说明对模型准确度的影响，计算对应epsilon值。

sigma	C	epsilon	首轮模型准确度	是否收敛
0.0001	2	75529	93.57	是
0.1	0.01	0.377	50.72	是
0.1	0.001	0.0377	51.15	是
0.05	0.01	0.755	60.29	是
0.5	0.05	0.377	58.32	是

可以看出，加入截断及高斯噪声后，模型的收敛速度减慢了。但仍能得到可用的模型（可用性），且能保护隐私。增大C，或者减小sigma都能加快模型收敛，但会增大epsilon值，即被泄露的信息量增大。

设置模式为“DP”，参数sigma=0.0001，C=2时的运行截图如下：

```
PS D:\USTC\DP2021_labs\lab2\ex2_v1\ex2_code> python main.py
cpu
load dataset...
clients and server initialization...
start training...
Round   0, Training average loss 0.565
Round   0, Testing accuracy: 93.57
Round   1, Training average loss 0.379
Round   1, Testing accuracy: 95.95
Round   2, Training average loss 0.304
Round   2, Testing accuracy: 96.82
Round   3, Training average loss 0.266
Round   3, Testing accuracy: 97.32
Round   4, Training average loss 0.238
Round   4, Testing accuracy: 97.66
Round   5, Training average loss 0.221
Round   5, Testing accuracy: 97.87
Training accuracy: 97.79
Testing accuracy: 97.87
```

附加内容

对于epsilon大于1的情况，在《Reviewing and Improving the Gaussian Mechanism for Differential Privacy》中有如下证明，说明其满足（ε，δ）-DP。

From Theorem 2, $\sigma_{\texttt{DP-OPT}}$ is the minimal required amount of Gaussian noise to achieve (ϵ, δ)-differential privacy. Hence, to show that the Gaussian noise amount $F(\delta) \times \Delta/\epsilon$ is not sufficient for (ϵ, δ)-differential privacy, we will prove that for any $0 < \delta < 1$, there exists a positive function $G(\delta)$ such that for any $\epsilon > G(\delta)$, we have

$$F(\delta) \times \Delta/\epsilon < \sigma_{\texttt{DP-OPT}}. \tag{22}$$

We can show that the function $x+\sqrt{x^2+\epsilon}$ strictly increases as x increases for $x \in (-\infty, \infty)$ by noting its derivative $1 + \frac{x}{\sqrt{x^2+\epsilon}}$ is positive. Also, $\lim_{x\to-\infty}(x + \sqrt{x^2+\epsilon}) = \lim_{x\to-\infty} \frac{\epsilon}{-x+\sqrt{x^2+\epsilon}} = 0$ and $\lim_{x\to\infty}(x + \sqrt{x^2+\epsilon}) = \infty$. Hence, the values that $x+\sqrt{x^2+\epsilon}$ for $x \in (-\infty, \infty)$ can take constitutes the open interval $(0, \infty)$. Then due to $F(\delta) > 0$, we can define h such that

$$F(\delta) = \frac{h + \sqrt{h^2+\epsilon}}{\sqrt{2}}. \tag{23}$$

From Eq. (23) and $\sigma_{\texttt{DP-OPT}} = \frac{\left(a+\sqrt{a^2+\epsilon}\right)\cdot\Delta}{\epsilon\sqrt{2}}$ of (5), clearly Inequality (22) is equivalent to $\frac{h+\sqrt{h^2+\epsilon}}{\sqrt{2}} < \frac{a+\sqrt{a^2+\epsilon}}{\sqrt{2}}$ and further equivalent to $h < a$.

As shown in Appendix D, $r(u) := \operatorname{erfc}(u) - e^{\epsilon}\operatorname{erfc}\left(\sqrt{u^2+\epsilon}\right)$ strictly decreases as u increases for $u \in (-\infty, \infty)$. Then $h < a$ is equivalent to $r(h) > r(a)$. We will prove $\lim_{\epsilon\to\infty} r(h) = 2$, which along with $r(a) = 2\delta$ in Eq. (5) implies that for any $0 < \delta < 1$, there exists a positive function $G(\delta)$ such that for any $\epsilon > G(\delta)$, we have $r(h) > r(a)$ and thus $h < a$.

From the above discussion, the desired result Eq. (22) follows once we show $\lim_{\epsilon\to\infty} r(h) = 2$. From Eq. (23), it holds that $h = \frac{F(\delta)}{\sqrt{2}} - \frac{\epsilon}{F(\delta)\cdot 2\sqrt{2}}$. Hence, for any $\epsilon \geq 4\times[F(\delta)]^2$, we have $h \leq -\frac{\epsilon}{4\sqrt{2}\cdot F(\delta)}$, which implies

$$\begin{aligned}
& e^{\epsilon}\,\mathrm{erfc}\left(\sqrt{h^2+\epsilon}\right) \\
& \leq e^{\epsilon}\,\mathrm{erfc}\left(|h|\right) \\
& \leq e^{\epsilon}\,\mathrm{erfc}\left(\frac{\epsilon}{4\sqrt{2}\cdot F(\delta)}\right) \\
& \leq e^{\epsilon} \times \exp\left(-\left(\frac{\epsilon}{4\sqrt{2}\cdot F(\delta)}\right)^2\right) \\
& \to 0,\ \text{as } \epsilon \to \infty,
\end{aligned} \tag{24}$$

where the last "$\leq$" uses $\mathrm{erfc}(x) \leq \exp\left(-x^2\right)$ for $x > 0$. The above result Eq. (24) implies $\lim_{\epsilon\to\infty}\left[e^{\epsilon}\,\mathrm{erfc}\left(\sqrt{h^2+\epsilon}\right)\right] = 0$. Combining this and $\lim_{\epsilon\to\infty}\mathrm{erfc}(h) = 2$, we derive $\lim_{\epsilon\to\infty} r(h) = 2$. Then as already explained, the desired result is proved. ■

同态加密算法在联邦学习中的应用

了解Paillier原理

实验原理

见实验文档

实验结果

测试输出，结果为：

```
****************test dec(),enc()****************
dec(priv, pub, enc(pub, 18000290)) = 18000290
****************test enc_add()****************
dec(priv, pub, enc_add(pub, 2021, 2000)) = 4021
****************test enc_add_const()****************
dec(priv, pub, enc_add_const(pub, 2021, 4)) = 2025
****************test enc_mul_const()****************
dec(priv, pub, enc_mul_const(pub, 2021, 4)) = 8084
****************test 10bits enc_add()****************
10bits for 1000 times!!!
Running time 16.72 seconds
****************test 100bits enc_add()****************
100bits for 1000 times!!!
```

```
Running time 17.91 seconds
****************test 1000bits enc_add()****************
1000bits for 1000 times!!!
Running time 30.54 seconds
****************test 10bits enc_add_const()****************
10bits for 1000 times!!!
Running time 11.41 seconds
****************test 100bits enc_add_const()****************
100bits for 1000 times!!!
Running time 12.56 seconds
****************test 1000bits enc_add_const()****************
1000bits for 1000 times!!!
Running time 22.21 seconds
```

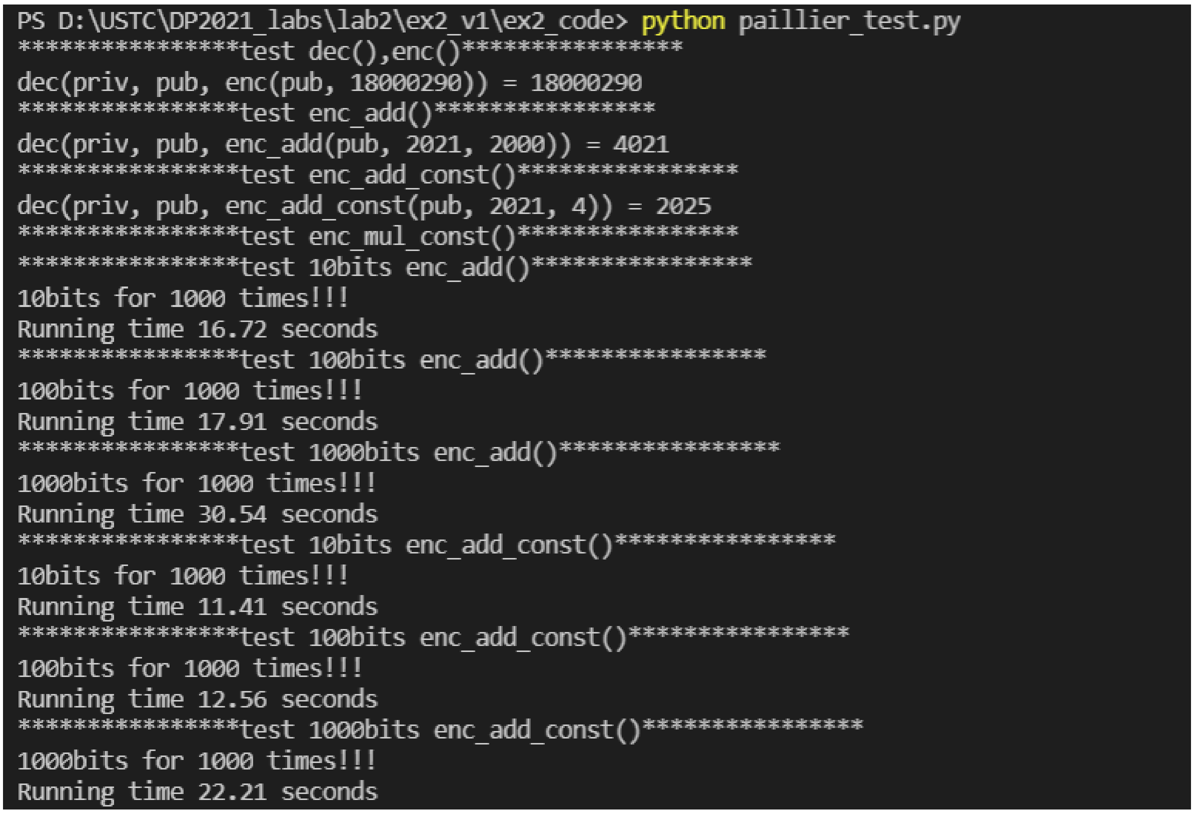
可以看出，加密算法运行结果正确。

长度10，100，1000bits的整数加法运行时间如下（随机生成1000组给定比特位的整数的加解密计算时间）

	enc_add()	enc_add_const()
10bits	16.72s	11.41s
100bits	17.91s	12.56s
1000bits	30.54s	22.21s

可以看出，运算时间随比特位数的增加而提高，但即使比特位数达到1000位，计算时间仍然是可以接受的，说明Paillier用实际应用价值。

运行截图：


```
PS D:\USTC\DP2021_labs\lab2\ex2_v1\ex2_code> python paillier_test.py
****************test dec(),enc()****************
dec(priv, pub, enc(pub, 18000290)) = 18000290
****************test enc_add()****************
dec(priv, pub, enc_add(pub, 2021, 2000)) = 4021
****************test enc_add_const()****************
dec(priv, pub, enc_add_const(pub, 2021, 4)) = 2025
****************test enc_mul_const()****************
****************test 10bits enc_add()****************
10bits for 1000 times!!!
Running time 16.72 seconds
****************test 100bits enc_add()****************
100bits for 1000 times!!!
Running time 17.91 seconds
****************test 1000bits enc_add()****************
1000bits for 1000 times!!!
Running time 30.54 seconds
****************test 10bits enc_add_const()****************
10bits for 1000 times!!!
Running time 11.41 seconds
****************test 100bits enc_add_const()****************
100bits for 1000 times!!!
Running time 12.56 seconds
****************test 1000bits enc_add_const()****************
1000bits for 1000 times!!!
Running time 22.21 seconds
```

Paillier + Federal Learning

实验原理

见实验文档

实验结果

在加解密的计算中，不能直接使用torch提供的张量运算，必须将每一项单独进行加解密计算。在本次实验中，使用了frompyfunc()方法，先将torch转为numpy数组，然后frompyfunc()，可将计算单个值的函数转化为计算每个元素的函数，本次实验中生成的加解密函数如下：

```
if args.mode == 'Paillier':
        pub, priv = generate_paillier_keypair()
        encode = np.frompyfunc(pub.encrypt, 1, 1)
        decode = np.frompyfunc(priv.decrypt, 1, 1)
```

在服务器的聚合计算过程中，直接进行计算即可，因为其中的加法和乘法已被重新定义了。

在客户端的加密和解密放在了`main.py`中进行（具体代码&注释，见源代码部分）：

```
    # training
    print("start training...")
    for iter in range(args.epochs):
        start_time = time.time()
        server.clients_update_w, server.clients_loss = [], []
        plain = []
        for idx in range(args.num_users):
            delta_w, loss = clients[idx].train()
            if args.mode == 'Paillier':
                delta_w_plain = copy.deepcopy(delta_w)
                plain.append(delta_w_plain)
                # 加密
                for k in delta_w.keys():
                    delta_w[k] = encode(delta_w[k])
            server.clients_update_w.append(delta_w)
            server.clients_loss.append(loss)

        # calculate global weights
        w_glob, loss_glob = server.FedAvg()
        if args.mode == 'Paillier':
            # 解密 + 转化
            for k in w_glob.keys():
                w_glob[k] = decode(w_glob[k])
                w_glob[k] = torch.from_numpy(w_glob[k].astype(float))
        # update local weights
        for idx in range(args.num_users):
            clients[idx].update(w_glob)

        # print loss
        acc_train, loss_train = server.test(dataset_train)
        acc_test, loss_test = server.test(dataset_test)
        end_time = time.time()
        print('Round {:3d}, Training average loss {:.3f}'.format(
            iter, loss_glob))
        print("Round {:3d}, Testing accuracy: {:.2f}".format(iter, acc_test))
        # 运行时间
```

```
        print("Running time %0.2f" %
              (end_time - start_time) + " seconds")
    # testing
```

根据实验结果，paillier仍能保证模型收敛及准确性，但是加解密过程的求模运算运算量较大，训练时间远高于原来的时间。