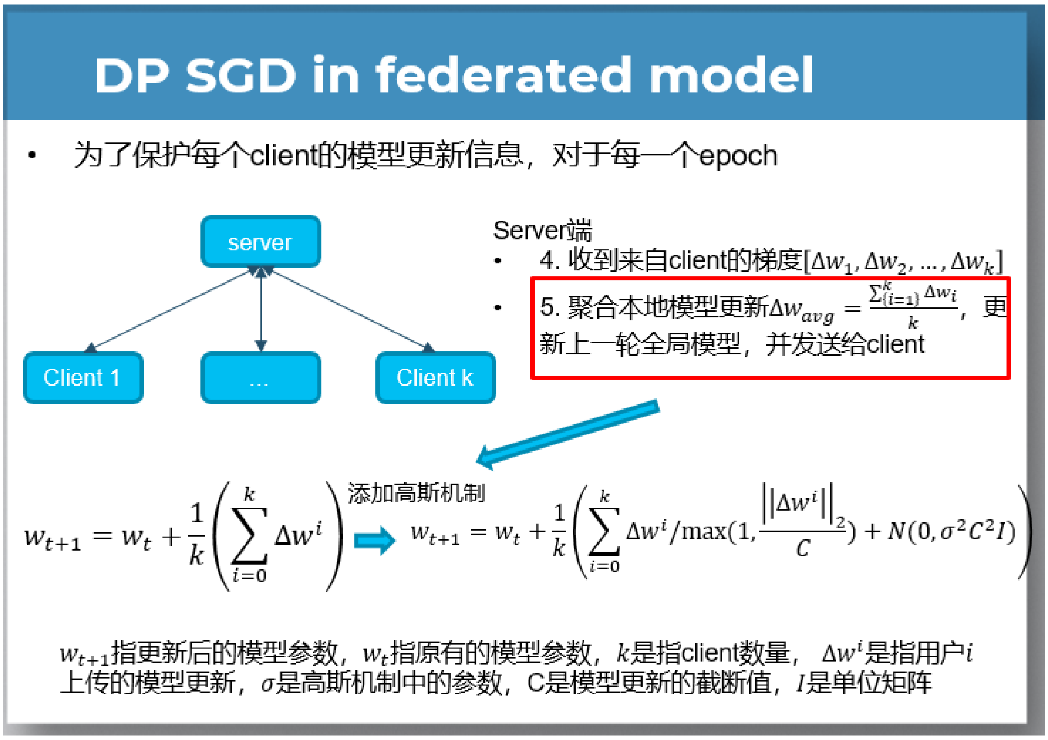
数据隐私 Lab2 实验报告

PB18111699

魏钊

Part1

根据 PPT



Server:

在原来代码的基础上通过加噪实现 DP:

```
elif self.args.mode == 'DP':
    #设置参数
    c = 0.001
    ParameterSigma = 0.1

    update_w_avg = copy.deepcopy(self.clients_update_w[0])
    for k in update_w_avg.keys():
        for i in range(1, len(self.clients_update_w)):
            MaxW = max(1, torch.norm(self.clients_update_w[i][k], 2) / c)#找出最大的值
            update_w_avg[k] += self.clients_update_w[i][k] / MaxW#求和
        update_w_avg[k] += torch.normal(0, ParameterSigma * c,
                                        update_w_avg[k].shape, device=self.args.device)#根据高斯机制添加噪音
        update_w_avg[k] = torch.div(update_w_avg[k], len(self.clients_update_w))#聚合模型本地更新
        self.model.state_dict()[k] += update_w_avg[k]
```

Client:

和 Plain 处理一样。

Train：

```
elif self.args.mode == 'DP':
    for k in w_new.keys():
```

update:

```
elif self.args.mode == 'DP':
    self.model.load_state_dict(w_glob)
```

运行结果：（这里是用 GPU 进行训练）

```
(base) C:\Users\Lucifer.dark\Desktop\ex2_v2\ex2_code>python main.py --mode DP --gpu 0
cuda:0
load dataset...
clients and server initialization...
start training...
C:\Python_Anaconda\lib\site-packages\torch\nn\functional.py:718: UserWarning: Named te
use them for anything important until they are released as stable. (Triggered internal
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
Round   0, Training average loss 0.545
Round   0, Testing accuracy: 70.40
训练时间s:  35.580453634262085
Round   1, Training average loss 0.428
Round   1, Testing accuracy: 88.33
训练时间s:  70.34130692481995
Round   2, Training average loss 0.391
Round   2, Testing accuracy: 93.14
训练时间s:  105.09403777122498
Round   3, Training average loss 0.364
Round   3, Testing accuracy: 95.16
训练时间s:  140.2180140018463
Round   4, Training average loss 0.342
Round   4, Testing accuracy: 95.82
训练时间s:  175.30417680740356
Round   5, Training average loss 0.324
Round   5, Testing accuracy: 96.20
训练时间s:  209.92105746269226
Training accuracy: 95.83
Testing accuracy: 96.20
```

可以看出这里已经训练到收敛。

Part2

任务 1:

根据 PPT:

- 加密
  消息$m(0 \le m \le n)$
  选择随机数r满足$0 < r < n$且$r \in \mathbb{Z}_n^*$(i.e.,保证$\gcd(r,n) = 1$)
  计算密文$c = g^m \cdot r^n mod\ n^2$

Paillier: 一种基于公钥系统的加法同态加密系统

- 解密
  密文$c \in \mathbb{Z}_{n^2}^*$
  明文$m = L(c^\lambda mod\ n^2) \cdot \mu\ mod\ n$, 其中$L(x) = \lfloor\frac{x-1}{n}\rfloor$, 表示$n$除$x-1$的商取下整
- 密文加法
  $Dec(Enc(m_1,r_1) \cdot Enc(m_2,r_2) mod\ n^2) = m_1 + m_2 mod\ n$
- 与明文常数加法
  $Dec(E(m_1,r_1) \cdot g^{m_2} mod\ n^2) = m_1 + m_2 mod\ n$
- 与明文常数乘法
  $Dec(Enc(m_1,r_1)^k mod\ n^2) = km_1 mod\ n$

```
def enc(pub, plain):  # (KeyPub key, plaintext) #to do
    G_of_m = powmod(pub.g, plain, pub.n_sq)  # 这里先计算G的m次方mod n的平方
    while True:
        r = mpz_random(rand, pub.n)
        if gcd(r, pub.n) == 1:  # 检查生成的r是否满足gcd=1
            break
    R_of_n = powmod(r, pub.n, pub.n_sq)  # 计算R的n次方mod n的平方
    cipher = powmod(G_of_m * R_of_n, 1, pub.n_sq)  # 计算密文c
    return cipher
```

```
def dec(priv, pub, cipher):  # (KeyPriv key, KeyPub key, cipher) #to do
    C_of_l = powmod(cipher, priv.l, pub.n_sq)  # C的l次方mod n的平方
    L_of_x = t_div(C_of_l - 1, pub.n)  # 计算L(x)
    plain = powmod(L_of_x * priv.m, 1, pub.n)  # 解密
    return plain
```

```
def enc_add(pub, m1, m2):  # to do
    """Add one encrypted integer to another"""
    return powmod(m1 * m2, 1, pub.n_sq)  # m1*m2 mod n^2
```

```python
def enc_add_const(pub, m, c):  # to do
    """Add constant n to an encrypted integer"""
    G_of_c = powmod(pub.g, c, pub.n_sq)
    return powmod(m * G_of_c, 1, pub.n_sq)  # m*g^c mod n^2
```

```python
def enc_mul_const(pub, m, c):  # to do
    """Multiplies an encrypted integer by a constant"""
    return powmod(m, c, pub.n_sq)  # m^c mod n^2
```

测试:

```python
    time_end = time.time()
    print("解密时间s: ", time_end - time_start)
    print(p4)


    time_start = time.time()
    c5 = enc_mul_const(pub, c1, 10)
    time_end = time.time()
    print("加密时间s: ", time_end - time_start)
    time_start = time.time()
    p1 = dec(priv, pub, c1)
    time_end = time.time()
    print("解密时间s: ", time_end - time_start)
    print(p1)


    time_start = time.time()
    c3 = enc_add(pub, c1, c2)
    time_end = time.time()
    print("加密时间s: ", time_end - time_start)
    time_start = time.time()
    p3 = dec(priv, pub, c3)
    time_end = time.time()
    print("解密时间s: ", time_end - time_start)
    print(p3)


    time_start = time.time()
    c4 = enc_add_const(pub, c1, 15)
    time_end = time.time()
    print("加密时间s: ", time_end - time_start)
    time_start = time.time()
    p4 = dec(priv, pub, c4)
    time_end = time.time()
    print("解密时间s: ", time_end - time_start)
    print(p4)


    time_start = time.time()
    c5 = enc_mul_const(pub, c1, 10)
    time_end = time.time()
    print("加密时间s: ", time_end - time_start)
    time_start = time.time()
    p5 = dec(priv, pub, c5)
    time_end = time.time()
    print("解密时间s: ", time_end - time_start)
    print(p5)
```

```
(base) C:\Users\Lucifer.dark\Desktop\ex2_v2\ex2_code>python paillier_test.py
加密时间s:  0.004988193511962891
解密时间s:  0.004999399185180664
16
加密时间s:  0.0060160160064697266
解密时间s:  0.0059814453125
27
加密时间s:  0.0
解密时间s:  0.006986141204833984
43
加密时间s:  0.0
解密时间s:  0.0059795379638671875
31
加密时间s:  0.0
解密时间s:  0.006010293960571289
160
```

结果正确。

任务 2:

Server:

根据 PPT:

5. 利用同态加密，聚合各个梯度

$$\Delta w_{avg} = \sum_{\{i=1\}}^{k} enc(\Delta w_i) \times \frac{1}{k}$$

```python
elif self.args.mode == 'Paillier':
    update_w_avg = copy.deepcopy(self.clients_update_w[0])
    for k in update_w_avg.keys():
        for i in range(1, len(self.clients_update_w)):
            for n in range(len(update_w_avg[k])):
                update_w_avg[k][n] += self.clients_update_w[i][k][n]  # 求和
        for n in range(len(update_w_avg[k])):
            update_w_avg[k][n] /= len(self.clients_update_w)  # 同态加密聚合梯度
    return copy.deepcopy(update_w_avg), sum(self.clients_loss) / len(self.clients_loss)
```

Client:

Train:

```python
elif self.args.mode == 'Paillier':
    for k in w_new.keys():
        update_w[k] = (w_new[k] - w_old[k]).flatten(start_dim=0).tolist()  # 用 w_new-
w_old 代替
        for j in range(len(update_w[k])):
            update_w[k][j] = KeyPub.encrypt(update_w[k][j])  # 用公共密钥进行加密
```

update:

```
elif self.args.mode == 'Paillier':
    for k in w_glob.keys():
        for i in range(len(w_glob[k])):
            w_glob[k][i] = KeyPriv.decrypt(w_glob[k][i])  # 用私钥进行解密
        Final_tensor = torch.Tensor(w_glob[k]).to(self.args.device)
        Final_tensor = Final_tensor.reshape(self.model.state_dict()[k].shape)  # 类型转换
回来
        self.model.state_dict()[k] += Final_tensor
```

训练结果:

```
(base) C:\Users\Lucifer.dark\Desktop\2021春\数据隐私\Lab2\PB18111699_魏钊_Lab2\ex2_code>python main.py --mode Paillier --num_users 1 --gpu 0
cuda:0
load dataset...
clients and server initialization...
start training...
C:\Python_Anaconda\lib\site-packages\torch\nn\functional.py:718: UserWarning: Named tensors
 and all their associated APIs are an experimental feature and subject to change. Please do
 not use them for anything important until they are released as stable. (Triggered internally at  ..\c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
Round   0, Training average loss 0.261
Round   0, Testing accuracy: 9.75
训练时间s:  1186.1098506450653
Round   1, Training average loss 0.187
Round   1, Testing accuracy: 9.75
训练时间s:  2361.284871339798
```

一轮大概需要 20 分钟。