DP-lab2 report

王昱 PB21030814

代码补全

用到的伪代码如下：

	Party A	Party B
Step 0	Create an encryption key pair, and send the public key to B	
Step 1	Initialize Θ^A	Initialize Θ^B
Step 2	Compute $\Theta^A x_i^A$ for $i \in D_A$	Compute $\Theta^B x_i^B$ for $i \in D_B$ and send them to A
Step 3	Compute $\Theta x_i = \Theta^A x_i^A + \Theta^B x_i^B$, $\hat{y}_i = h_\Theta(x_i)$, $[\![(y_i - \hat{y}_i)]\!]$, and send $[\![(y_i - \hat{y}_i)]\!]$ to B for $i \in D_A$	
Step 4	Compute $\frac{\partial L}{\partial \Theta^A}$ and the loss L	Compute $\left[\!\left[\frac{\partial L}{\partial \Theta^B}\right]\!\right]$, generate random number R_B, and send $\left[\!\left[\frac{\partial L}{\partial \Theta^B}\right]\!\right] + [\![R_B]\!]$ to A
Step 5	Decrypt $\left[\!\left[\frac{\partial L}{\partial \Theta^B}\right]\!\right] + [\![R_B]\!]$, and send $\frac{\partial L}{\partial \Theta^B} + R_B$ to B	
Step 6	Update Θ^A	Update Θ^B

补全$\hat{y}$计算流程

```python
# Q1. Active party calculates y_hat
# ------------------------------------------------
# TODO
active_wx = np.dot(self.x_train[batch_idxes], self.params)
passive_wx = self.messenger.recv()
# TODO
full_wx = active_wx + passive_wx
y_hat = self.activation(full_wx)
# ------------------------------------------------
```

> active.py 文件的$\hat{y}$计算如上

1. active_wx 的计算是根据伪代码中的公式$\Theta^A x_i^A$，调用 numpy 库的 np.dot() 方法实现点乘
2. full_wx 的计算是根据伪代码中的公式$\Theta x_i = \Theta^A x_i^A + \Theta^B x_i^B$， active_wx 代表的是$\Theta^A x_i^A$， passive_wx 代表的是$\Theta^B x_i^B$，因此直接相加即可

```
# Q1. Calculate wx and send it to active party
# ------------------------------------------------
# TODO
passive_wx = np.dot(self.x_train[batch_idxes], self.params)
self.messenger.send(passive_wx)
# ------------------------------------------------
```

passive.py 文件的补全如上

1. passive_wx 的计算是根据伪代码中的公式$\Theta^B x_i^B$，调用 numpy 库的 np.dot() 方法实现点乘

补全梯度计算流程

```
# Q2. Active party helps passive party to calculate gradient
# ------------------------------------------------
# TODO
enc_residue = self.cryptosystem.encrypt_vector(residue)
enc_residue = np.array(enc_residue)
self.messenger.send(enc_residue)
enc_passive_grad = self.messenger.recv()
# TODO
passive_grad =
np.array(self.cryptosystem.decrypt_vector(enc_passive_grad))
self.messenger.send(passive_grad)
# ------------------------------------------------
```

active.py 文件的梯度计算补全如上

1. enc_residue 是调用 paillier.py 文件中 Paillier 类的 encrypt_vector() 方法实现的，作用就是得到伪代码中的$[[(y_i - \hat{y}_i)]]$
2. passive_grad 是调用 paillier.py 文件中 Paillier 类的 decrypt_vector() 方法实现的，作用就是解密收到的$[[\frac{\partial L}{\partial \Theta^B}]] + [[R_B]]$

```
# Q2. Receive encrypted residue and calculate masked encrypted gradients
# ------------------------------------------------------
enc_residue = self.messenger.recv()
enc_grad = self._gradient(enc_residue, batch_idxes)
enc_mask_grad, mask = self._mask_grad(enc_grad)
self.messenger.send(enc_mask_grad)
# Receive decrypted masked gradient and update model
mask_grad = self.messenger.recv()
true_grad = self._unmask_grad(mask_grad, mask)
# ------------------------------------------------------

def _mask_grad(self, enc_grad):
    # TODO
    mask = np.random.normal(0, 1.0, enc_grad.shape)
    # TODO
    enc_mask_grad = enc_grad + np.array(self.cryptosystem.encrypt_vector(mask))
    return enc_mask_grad, mask

def _unmask_grad(self, mask_grad, mask):
    # TODO
    true_grad = mask_grad - mask
    return true_grad
```

passive.py 文件的梯度计算补全如上

1. _mask_grad(self, enc_grad) 方法中的 mask 是调用 np.random.normal() 方法实现的，返回的是 enc_grad.shape 大小的 ndarray ，元素符合正态分布，这一步得到了伪代码中的R_B。然后将 enc_grad 与 np.array(self.cryptosystem.encrypt_vector(mask)) 相加， np.array(self.cryptosystem.encrypt_vector(mask)) 相当于伪代码中的$[[R_B]]$，因此 enc_mask_grad 就是伪代码中的$[[\frac{\partial L}{\partial \Theta^B}]] + [[R_B]]$
2. _unmask_grad(self, mask_grad, mask) 方法中的 true_grad 直接用解码后的 mask_grad (相当于伪代码中的$\frac{\partial L}{\partial \Theta^B} + R_B$)减去 mask (相当于伪代码中的$R_B$)即可

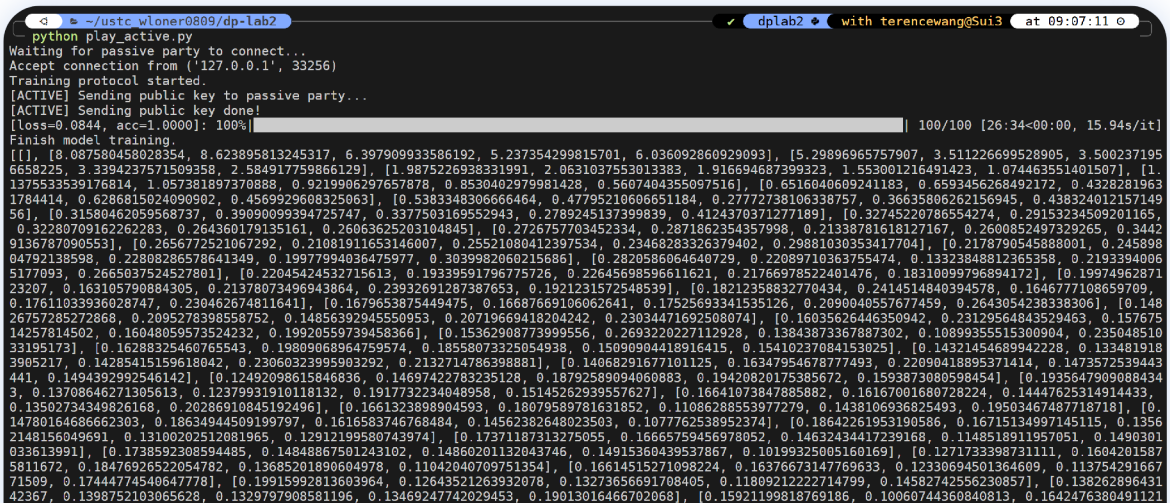
补全训练过程中模型准确率计算流程

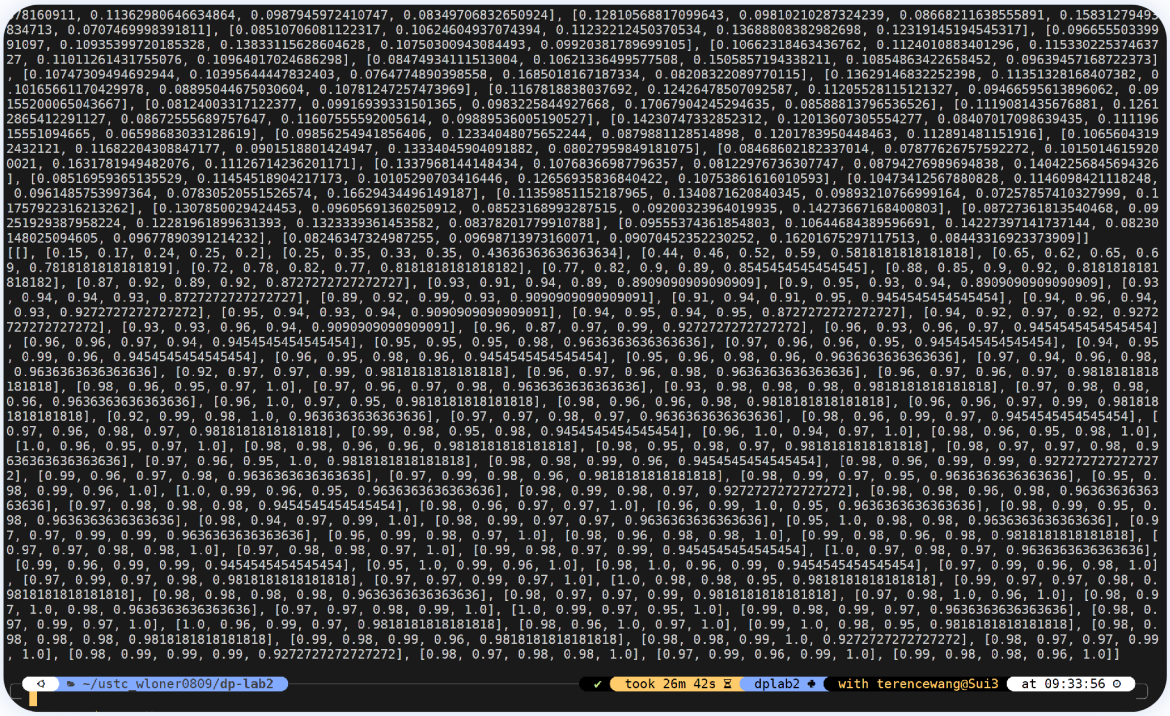
```python
def _acc(self, y_true, y_hat):
    # Q3. Compute accuracy
    # ------------------------------------------------
    # TODO
    acc = accuracy_score(y_true=y_true, y_pred=y_hat.round())
    # ------------------------------------------------
    return acc
```

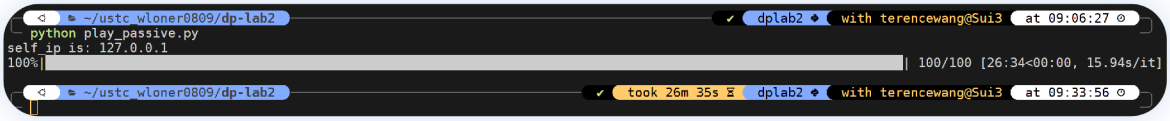
active.py 文件的模型准确率计算补全如上

1. acc 是通过调用 sklearn.metrics 中的 accuracy_score() 方法计算得到，同时对$\hat{y}$取 round

实验结果(loss、acc的变化)


~/ustc_wloner0809/dp-lab2
dplab2
with terencewang@Sui3
at 09:07:11
python play_active.py
Waiting for passive party to connect...
Accept connection from ('127.0.0.1', 33256)
Training protocol started.
[ACTIVE] Sending public key to passive party...
[ACTIVE] Sending public key done!
[loss=0.0844, acc=1.0000]: 100%| 100/100 [26:34<00:00, 15.94s/it]
Finish model training.


~/ustc_wloner0809/dp-lab2
took 26m 42s
dplab2
with terencewang@Sui3
at 09:33:56


~/ustc_wloner0809/dp-lab2
dplab2
with terencewang@Sui3
at 09:06:27
python play_passive.py
self_ip is: 127.0.0.1
100%| 100/100 [26:34<00:00, 15.94s/it]
~/ustc_wloner0809/dp-lab2
took 26m 35s
dplab2
with terencewang@Sui3
at 09:33:56

> 训练结果如上图所示， loss 和 acc 的记录输出在终端，整理成表格如下

loss：

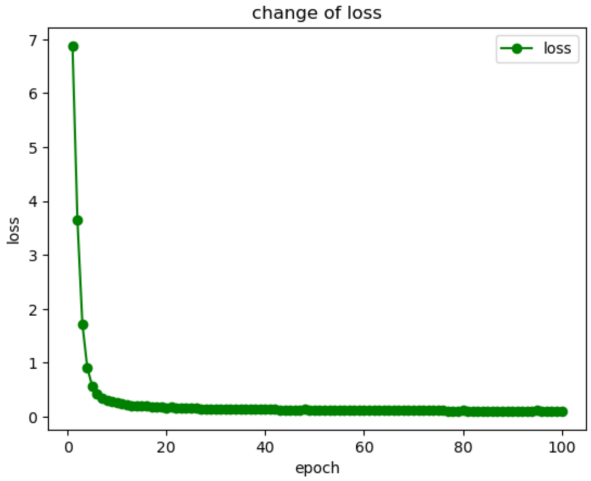
epoch	batch0	batch1	batch2	batch3	batch4	aver
0	8.087580	8.623896	6.397910	5.237354	6.036093	6.876567
1	5.298970	3.511227	3.500237	3.339424	2.584918	3.646955
2	1.987523	2.063104	1.916695	1.553001	1.074464	1.718957
3	1.137553	1.057382	0.921991	0.853040	0.560740	0.906141
4	0.651604	0.659346	0.432828	0.628682	0.456993	0.565890
5	0.538335	0.477952	0.277727	0.366358	0.438324	0.419739
6	0.315805	0.390901	0.337750	0.278925	0.412437	0.347163
7	0.327452	0.291532	0.322807	0.264360	0.260636	0.293358
8	0.272676	0.287186	0.213388	0.260085	0.344291	0.275525
9	0.265677	0.210819	0.255211	0.234683	0.298810	0.253040
10	0.217879	0.245898	0.228083	0.199780	0.303998	0.239128
11	0.282059	0.220897	0.133238	0.219339	0.266504	0.224407
12	0.220454	0.193396	0.226457	0.217670	0.183101	0.208216
13	0.199750	0.163106	0.213781	0.239327	0.192123	0.201617
14	0.182124	0.241451	0.164678	0.176110	0.230463	0.198965
15	0.167965	0.166877	0.175257	0.209004	0.264305	0.196682
16	0.148268	0.209528	0.148564	0.207197	0.230345	0.188780
17	0.160356	0.231296	0.157675	0.160481	0.199206	0.181803
18	0.153629	0.269322	0.138439	0.108994	0.235049	0.181086
19	0.162883	0.198091	0.185581	0.150909	0.154102	0.170313
20	0.143215	0.133482	0.142854	0.230603	0.213271	0.172685
21	0.140683	0.163480	0.220904	0.147357	0.149439	0.164373
22	0.124921	0.146974	0.187926	0.194208	0.159387	0.162683
23	0.193565	0.137086	0.123799	0.191773	0.151453	0.159535
24	0.166411	0.161670	0.144476	0.135027	0.202869	0.162091
25	0.166132	0.180796	0.110863	0.143811	0.195035	0.159327
26	0.147802	0.186349	0.161658	0.145624	0.107776	0.149842
27	0.186423	0.167151	0.135621	0.131002	0.129122	0.149864
28	0.173712	0.166658	0.146324	0.114852	0.149030	0.150115
29	0.173859	0.148489	0.148602	0.149154	0.101993	0.144419
30	0.127173	0.160420	0.184769	0.136852	0.110420	0.143927
31	0.166145	0.163767	0.123307	0.113754	0.174448	0.148284
32	0.199160	0.126435	0.132737	0.118092	0.145827	0.144450
33	0.138263	0.139875	0.132980	0.134692	0.190130	0.147188
34	0.159212	0.100607	0.164248	0.150487	0.126791	0.140269
35	0.135683	0.180563	0.140736	0.113637	0.123509	0.138825
36	0.168476	0.127526	0.131851	0.100819	0.190196	0.143774
37	0.246662	0.096300	0.118893	0.096957	0.126397	0.137042
38	0.153179	0.149201	0.114996	0.127949	0.141902	0.137445
39	0.118054	0.149625	0.119227	0.151979	0.145025	0.136782
40	0.124043	0.175289	0.100896	0.133677	0.145848	0.135950
41	0.098867	0.095959	0.200423	0.129471	0.154081	0.135760
42	0.136007	0.095889	0.201057	0.122238	0.091095	0.129257
43	0.095773	0.176063	0.161078	0.127165	0.074440	0.126904
44	0.111016	0.125857	0.178511	0.135918	0.084326	0.127126
45	0.134053	0.110173	0.157293	0.124769	0.123279	0.129913
46	0.089065	0.195901	0.120855	0.117752	0.121245	0.128963
47	0.096989	0.150238	0.117879	0.110928	0.201263	0.135459
48	0.141392	0.122090	0.154991	0.098258	0.120791	0.127504
49	0.108370	0.115129	0.117487	0.147252	0.165930	0.130834
50	0.137174	0.147565	0.112779	0.103079	0.137867	0.127693
51	0.101469	0.131288	0.159227	0.100751	0.146060	0.127759
52	0.134130	0.097961	0.129255	0.128829	0.144921	0.127019
53	0.130966	0.094680	0.123891	0.152889	0.118038	0.124093
54	0.179460	0.113063	0.092046	0.129283	0.092136	0.121197
55	0.080844	0.116065	0.152092	0.129053	0.151869	0.125984
56	0.142824	0.106732	0.097940	0.118837	0.167935	0.126853
57	0.104127	0.150418	0.112933	0.126019	0.112948	0.121289
58	0.129804	0.099017	0.129479	0.114423	0.146316	0.123808
59	0.104163	0.114089	0.160618	0.122958	0.088300	0.118025
60	0.159483	0.121286	0.083402	0.128846	0.099543	0.118512
61	0.096930	0.094955	0.154116	0.133177	0.120960	0.120027
62	0.112561	0.152444	0.137409	0.089366	0.093096	0.116975
63	0.159908	0.120695	0.094810	0.109745	0.100670	0.117166
64	0.115223	0.081429	0.132343	0.101157	0.196623	0.125355
65	0.119507	0.141784	0.093640	0.101251	0.145084	0.120253
66	0.163898	0.095052	0.105946	0.120931	0.087024	0.114570
67	0.130378	0.133513	0.099147	0.118052	0.092044	0.114627
68	0.091505	0.133288	0.114672	0.115125	0.136124	0.118143
69	0.130685	0.142065	0.119158	0.085058	0.091654	0.113724
70	0.127822	0.119739	0.115540	0.114300	0.086935	0.112867
71	0.080804	0.098419	0.119733	0.114436	0.200222	0.122723
72	0.088039	0.104987	0.111888	0.124495	0.166957	0.119273
73	0.108696	0.122411	0.121711	0.083236	0.151097	0.117430
74	0.126440	0.073355	0.111789	0.151236	0.099235	0.112411
75	0.116874	0.084327	0.146855	0.090373	0.140598	0.115805
76	0.144701	0.080155	0.133169	0.109903	0.082879	0.110161
77	0.129039	0.124321	0.113630	0.098795	0.083497	0.109856
78	0.128106	0.098102	0.086682	0.158313	0.070747	0.108390
79	0.085107	0.106246	0.112322	0.136888	0.123191	0.112751
80	0.096656	0.109354	0.138331	0.107503	0.099204	0.110209
81	0.106623	0.112401	0.115330	0.110113	0.109640	0.110821
82	0.084749	0.106213	0.150586	0.108549	0.096395	0.109298
83	0.107473	0.103956	0.076477	0.168502	0.082083	0.107698
84	0.136291	0.113513	0.101657	0.088950	0.107812	0.109645
85	0.116782	0.124265	0.112055	0.094666	0.091552	0.107864
86	0.081240	0.099169	0.098323	0.170679	0.085888	0.107060
87	0.111908	0.126129	0.086726	0.116076	0.098895	0.107947
88	0.142307	0.120136	0.084070	0.111196	0.065987	0.104739
89	0.098563	0.123340	0.087988	0.120178	0.112891	0.108592
90	0.106560	0.116822	0.090152	0.133340	0.080280	0.105431
91	0.084686	0.078776	0.101501	0.163178	0.111267	0.107882
92	0.133797	0.107684	0.081230	0.087943	0.140423	0.110215
93	0.085170	0.114545	0.101053	0.126569	0.107539	0.106975
94	0.104734	0.114610	0.096149	0.078305	0.166294	0.112018
95	0.113599	0.134087	0.098932	0.072579	0.117579	0.107355
96	0.130785	0.096057	0.085232	0.092003	0.142737	0.109363
97	0.087274	0.092519	0.122820	0.132334	0.083782	0.103746
98	0.095554	0.106447	0.142274	0.082301	0.096779	0.104671
99	0.082463	0.096987	0.090705	0.162017	0.084433	0.103321

acc：

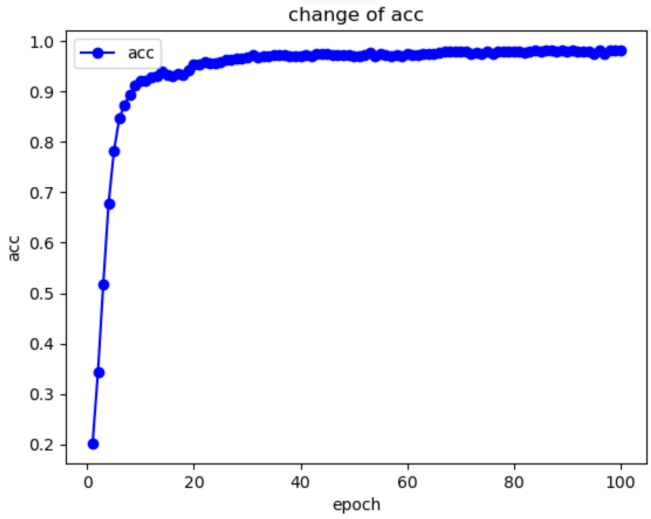
epoch	batch0	batch1	batch2	batch3	batch4	aver
0	0.150000	0.170000	0.240000	0.250000	0.200000	0.202000
1	0.250000	0.350000	0.330000	0.350000	0.436364	0.343273
2	0.440000	0.460000	0.520000	0.590000	0.581818	0.518364
3	0.650000	0.620000	0.650000	0.690000	0.781818	0.678364
4	0.720000	0.780000	0.820000	0.770000	0.818182	0.781636
5	0.770000	0.820000	0.900000	0.890000	0.854545	0.846909
6	0.880000	0.850000	0.900000	0.920000	0.818182	0.873636
7	0.870000	0.920000	0.890000	0.920000	0.872727	0.894545
8	0.930000	0.910000	0.940000	0.890000	0.890909	0.912182
9	0.900000	0.950000	0.930000	0.940000	0.890909	0.922182
10	0.930000	0.940000	0.940000	0.930000	0.872727	0.922545
11	0.890000	0.920000	0.990000	0.930000	0.909091	0.927818
12	0.910000	0.940000	0.910000	0.950000	0.945455	0.931091
13	0.940000	0.960000	0.940000	0.930000	0.927273	0.939455
14	0.950000	0.940000	0.930000	0.940000	0.909091	0.933818
15	0.940000	0.950000	0.940000	0.950000	0.872727	0.930545
16	0.940000	0.920000	0.970000	0.920000	0.927273	0.935455
17	0.930000	0.930000	0.960000	0.940000	0.909091	0.933818
18	0.960000	0.870000	0.970000	0.990000	0.927273	0.943455
19	0.960000	0.930000	0.960000	0.970000	0.945455	0.953091
20	0.960000	0.960000	0.970000	0.940000	0.945455	0.955091
21	0.950000	0.950000	0.950000	0.980000	0.963636	0.958727
22	0.970000	0.960000	0.960000	0.950000	0.945455	0.957091
23	0.940000	0.950000	0.990000	0.960000	0.945455	0.957091
24	0.960000	0.950000	0.980000	0.960000	0.945455	0.959091
25	0.950000	0.960000	0.980000	0.960000	0.963636	0.962727
26	0.970000	0.940000	0.960000	0.980000	0.963636	0.962727
27	0.920000	0.970000	0.970000	0.990000	0.981818	0.966364
28	0.960000	0.970000	0.960000	0.980000	0.963636	0.966727
29	0.960000	0.970000	0.960000	0.970000	0.981818	0.968364
30	0.980000	0.960000	0.950000	0.970000	1.000000	0.972000
31	0.970000	0.960000	0.970000	0.980000	0.963636	0.968727
32	0.930000	0.980000	0.980000	0.980000	0.981818	0.970364
33	0.970000	0.980000	0.980000	0.960000	0.963636	0.970727
34	0.960000	1.000000	0.970000	0.950000	0.981818	0.972364
35	0.980000	0.960000	0.960000	0.980000	0.981818	0.972364
36	0.960000	0.960000	0.970000	0.990000	0.981818	0.972364
37	0.920000	0.990000	0.980000	1.000000	0.963636	0.970727
38	0.970000	0.970000	0.980000	0.970000	0.963636	0.970727
39	0.980000	0.960000	0.990000	0.970000	0.945455	0.969091
40	0.970000	0.960000	0.980000	0.970000	0.981818	0.972364
41	0.990000	0.980000	0.950000	0.980000	0.945455	0.969091
42	0.960000	1.000000	0.940000	0.970000	1.000000	0.974000
43	0.980000	0.960000	0.950000	0.980000	1.000000	0.974000
44	1.000000	0.960000	0.950000	0.970000	1.000000	0.976000
45	0.980000	0.980000	0.960000	0.960000	0.981818	0.972364
46	0.980000	0.950000	0.980000	0.970000	0.981818	0.972364
47	0.980000	0.970000	0.970000	0.980000	0.963636	0.972727
48	0.970000	0.960000	0.950000	1.000000	0.981818	0.972364
49	0.980000	0.980000	0.990000	0.960000	0.945455	0.971091
50	0.980000	0.960000	0.990000	0.990000	0.927273	0.969455
51	0.990000	0.960000	0.970000	0.980000	0.963636	0.972727
52	0.970000	0.990000	0.980000	0.960000	0.981818	0.976364
53	0.980000	0.990000	0.970000	0.950000	0.963636	0.970727
54	0.950000	0.980000	0.990000	0.960000	1.000000	0.976000
55	1.000000	0.990000	0.960000	0.950000	0.963636	0.972727
56	0.980000	0.990000	0.980000	0.970000	0.927273	0.969455
57	0.980000	0.980000	0.960000	0.980000	0.963636	0.972727
58	0.970000	0.980000	0.980000	0.980000	0.945455	0.971091
59	0.980000	0.960000	0.970000	0.970000	1.000000	0.976000
60	0.960000	0.990000	1.000000	0.950000	0.963636	0.972727
61	0.980000	0.990000	0.950000	0.980000	0.963636	0.972727
62	0.980000	0.940000	0.970000	0.990000	1.000000	0.976000
63	0.980000	0.990000	0.970000	0.970000	0.963636	0.974727
64	0.950000	1.000000	0.980000	0.980000	0.963636	0.974727
65	0.970000	0.970000	0.990000	0.990000	0.963636	0.976727
66	0.960000	0.990000	0.980000	0.970000	1.000000	0.980000
67	0.980000	0.960000	0.980000	0.980000	1.000000	0.980000
68	0.990000	0.980000	0.960000	0.980000	0.981818	0.978364
69	0.970000	0.970000	0.980000	0.980000	1.000000	0.980000
70	0.970000	0.980000	0.980000	0.970000	1.000000	0.980000
71	0.990000	0.980000	0.970000	0.990000	0.945455	0.975091
72	1.000000	0.970000	0.980000	0.970000	0.963636	0.976727
73	0.990000	0.960000	0.990000	0.990000	0.945455	0.975091
74	0.950000	1.000000	0.990000	0.960000	1.000000	0.980000
75	0.980000	1.000000	0.960000	0.990000	0.945455	0.975091
76	0.970000	0.990000	0.960000	0.980000	1.000000	0.980000
77	0.970000	0.990000	0.970000	0.980000	0.981818	0.978364
78	0.970000	0.970000	0.990000	0.970000	1.000000	0.980000
79	1.000000	0.980000	0.980000	0.950000	0.981818	0.978364
80	0.990000	0.970000	0.970000	0.980000	0.981818	0.978364
81	0.980000	0.980000	0.980000	0.980000	0.963636	0.976727
82	0.980000	0.970000	0.970000	0.990000	0.981818	0.978364
83	0.970000	0.980000	1.000000	0.960000	1.000000	0.982000
84	0.980000	0.970000	1.000000	0.980000	0.963636	0.978727
85	0.970000	0.970000	0.980000	0.990000	1.000000	0.982000
86	1.000000	0.990000	0.970000	0.950000	1.000000	0.982000
87	0.990000	0.980000	0.990000	0.970000	0.963636	0.978727
88	0.980000	0.970000	0.990000	0.970000	1.000000	0.982000
89	1.000000	0.960000	0.990000	0.970000	0.981818	0.980364
90	0.980000	0.960000	1.000000	0.970000	1.000000	0.982000
91	0.990000	1.000000	0.980000	0.950000	0.981818	0.980364
92	0.980000	0.980000	0.980000	0.980000	0.981818	0.980364
93	0.990000	0.980000	0.990000	0.960000	0.981818	0.980364
94	0.980000	0.980000	0.990000	1.000000	0.927273	0.975455
95	0.980000	0.970000	0.970000	0.990000	1.000000	0.982000
96	0.980000	0.990000	0.990000	0.990000	0.927273	0.975455
97	0.980000	0.970000	0.980000	0.980000	1.000000	0.982000
98	0.970000	0.990000	0.960000	0.990000	1.000000	0.982000
99	0.990000	0.980000	0.980000	0.960000	1.000000	0.982000

> 对 loss 和 acc 的可视化如下

loss



acc



> 在训练过程中， acc 由小变大，最后稳定在 0.98 附近； loss 由大变小，最后稳定在 0.10 附近。可以说明方法的有效性

问答题

Q1

```
1  def scale(dataset):
2      raw_dataset = dataset.get_dataset()
3
4      start_col = 2 if dataset.has_label else 1
5      scaled_feats = preprocessing.scale(raw_dataset[:, start_col:],
   copy=False)
6      raw_dataset[:, start_col:] = scaled_feats
7
8      dataset.set_dataset(raw_dataset)
```

> scale 函数如上
>
> 1. 原理

1. 首先调用 dataset.py 文件中 Dateset 类的 get_dataset() 方法，得到了原始的数据集
2. 然后根据是否有 label 设置 start_col ，这里主要是为了从特征数据开始的地方进行处理
3. 接下来调用 preprocessing.scale() 方法，对数据进行标准化预处理，将特征调整成正态分布(均值为0，方差为1)。函数原型为：sklearn.preprocessing.scale(X, *, axis=0, with_mean=True, with_std=True, copy=True) ，重要参数含义如下：
    1. axis ： 用于计算平均值和标准偏差的轴。如果为0，则独立标准化每个特征，否则(如果为1)则标准化每个样本
    2. with_mean ： 如果为True，则在缩放之前将数据居中
    3. with_std ： 如果为True，则将数据缩放到单位方差(或单位标准偏差)
    4. copy ： 设置为False将执行就地行规范化并避免复制(如果输入已经是numpy数组或scipy.sparse CSC矩阵，并且轴为1)
4. 最后将变换后的特征赋值给原始数据集，完成变换，并调用 data set.py 文件中 Dateset 类的 set_dataset() 方法

2. 作用：
    1. 对数据进行标准化预处理，可以去除不同特征之间的量纲差异。不同特征可能具有不同的尺度和单位，可能会导致某些特征对模型的影响更大。通过标准化，可以消除这种差异，使得所有特征在相同的尺度上进行比较
    2. 提高模型的收敛速度，使得模型更快达到最优解
    3. 由于标准化将数据转换为标准正态分布，异常值的影响将被缩小，因此标准化可以减少异常值对模型的影响，增强模型的鲁棒性
    4. 防止有些特征的方差过大，主导目标函数从而使参数估计器无法正确地去学习其他特征

Q2

> 在每个 epoch 开始时使用 epoch 值作为随机数种子是为了确保实验的可重复性。设置同一个随机数种子可以使得每次运行程序时得到相同的随机数序列，从而保证实验结果的一致性，方便调试和结果复现
>
> ```
> 1  np.random.seed(epoch)
> 2  np.random.shuffle(all_idxes)
> ```
>
> 这段代码就是将每个 epoch 的值作为种子，然后使用这个种子来打乱 all_idxes

将原始代码：

```
1  all_idxes = np.arange(n_samples)
2  np.random.seed(epoch)
3  np.random.shuffle(all_idxes)
```

换成如下代码：

```
1  random_state = np.random.RandomState(epoch)
2  all_idxes = np.arange(n_samples)
3  random_state.shuffle(all_idxes)
```

可以达到相同的目的

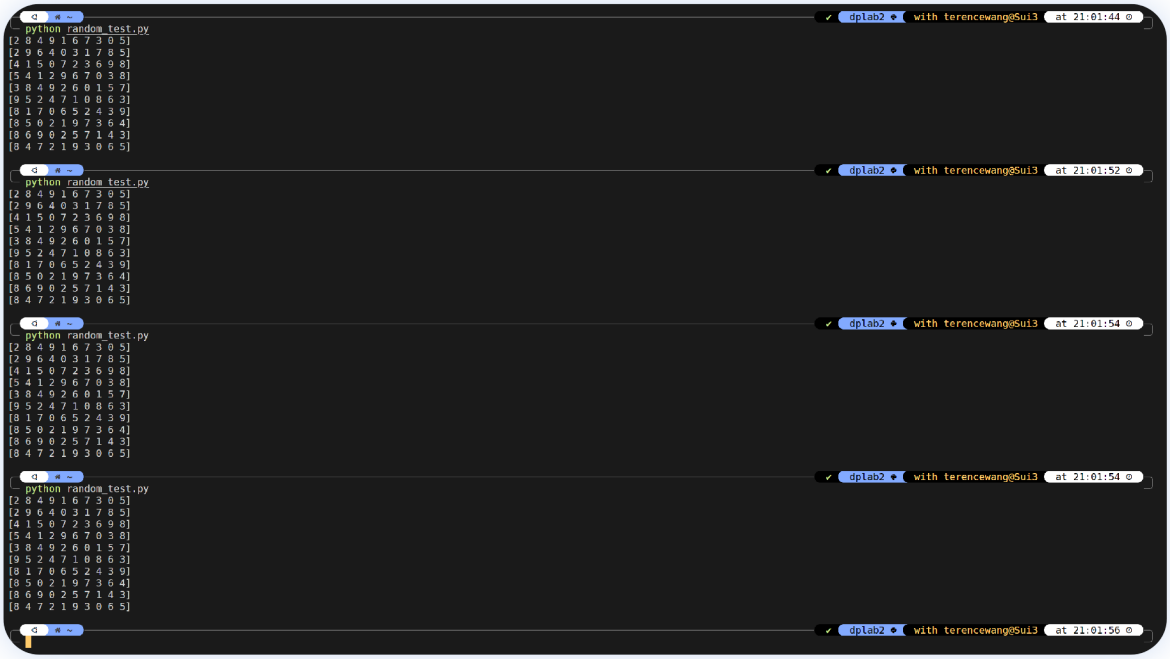
> 原理： np.random.RandomState() 是一个伪随机数生成器，会产生一个随机状态种子,在该状态下生成的随机序列(正态分布)一定会有相同的模式。

利用下面的代码进行简单的测试：

```python
import numpy as np

for epoch in range(10):
    random_state = np.random.RandomState(epoch)
    all_idxes = np.arange(10)
    random_state.shuffle(all_idxes)
    print(all_idxes)
```

运行4次结果如下：



可以看出每次生成的序列是相同的，简单验证了方法的有效性

Q3

> 潜在的隐私泄露风险及对应的保护方式：

1. **模型参数泄露**：数据所有者不直接共享他们的原始数据，但是他们在训练过程中共享模型参数。这些参数可能会泄露有关原始数据的信息。例如，如果一个恶意参与者可以访问模型的权重，他们就可以推断出一些关于训练数据的信息。

   保护方式：使用差分隐私(Differential Privacy)。差分隐私通过添加噪声来保护模型参数，使得恶意参与者无法准确地推断出原始数据。

2. **参与者合谋**：如果两个或更多的数据所有者合谋，他们就可能重构出其他数据所有者的原始数据。

   保护方式：使用安全多方计算(Secure Multi-party Computation，SMPC)。SMPC可以确保即使某些参与者合谋，他们也无法获取足够的信息来重构出原始数据。

3. **模型参数的累积泄露**：长时间运行的VFL-LR迭代可能导致模型参数的累积泄露，增加对本地数据的敏感推断

   保护方式：定期更新模型，重新初始化模型参数，或者使用一次性密钥以减缓累积泄露的风险

4. **同态加密**：效果与密钥的大小相关，密钥过小可能会导致被破解

   保护方式：增大密钥的大小

5. **中间计算结果**：训练过程中共享的一些中间计算结果可能被恶意参与者利用来推断训练数据的信息，造成隐私泄露

   保护方式：采用更安全的中间计算结果，仅让可信授权方可见