实验报告

学号：1911440　　姓名：刘旭博

一、问题重述

基本要求:将test_data.csv,test_truedata.csv分为测试集和验证集。实现任意补插算法来对数据集data.csv进行补插。使用测试集确定一个较好的模型，并使用验证集验证。针对你的模型和实验写一份报告，并提交源码(说明运行环境)和可执行文件。(建议使用神经网络)

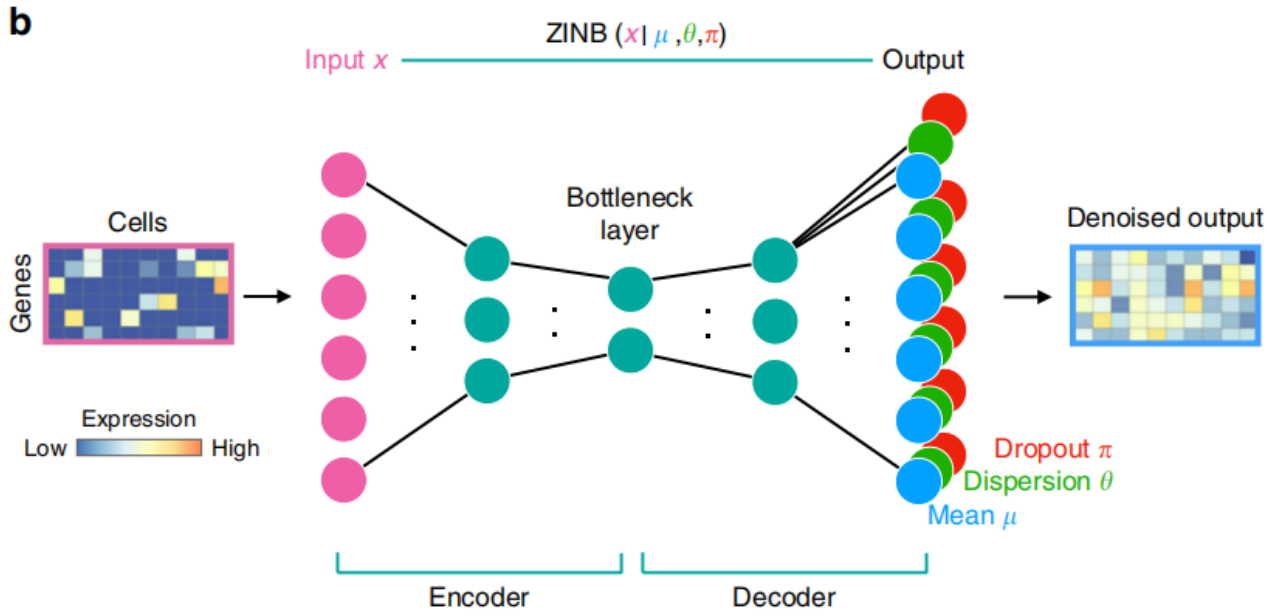
中级要求:在test_data.csv上获得一定效果(dropout率变小，表达差异变小)。在别人理论的基础上进行一些自己的修改。

高级要求:对于data.csv的补插和生成data.csv的模拟真实数据集相比获得较好效果(dropout率变小，表达差异变小)。

二、设计思想

1.网络结构与损失函数

本次实验采用神经网络的方法，使用助教给出的DCA方法，使用pytorch实现。网络架构如下（来源于dca论文）



如上图，中间的隐藏层采用传统的64-32-64尺寸，输入为一批细胞在给出基因的表达量（含有dropout），可以认为一行为一个细胞的基因表达量。有三个输出层，相当于输出内容为输入内容的三倍，分别对应其分布的三个参数。

重点： 该dca方法的网络结构非常简单，其重点在于损失函数的编写，该网络实际上实现了两个损失函数，分别对应数据集的普通情况（非dropout情况）和dropout情况，在普通情况下使用NB分布的概率密度函数的负对数，在被认为是dropout的地方使用ZINB分布的概率密度函数的负对数，使用了最大似然估计的思想。其中输出层mean的结果被当作补差结果。对于那些非0的数据，我们认为他是正常的数据，因此他们只参与模型的训练，而不进行补差修改，只对那些表达量为0的位置进行补差，即用输出层mean矩阵中的结果替代该处位置的0，得到最终的补差结果。

我们的损失函数参照公式如下（注意要取负对数）：

$$\mathrm{NB}(x;\mu,\theta)=\frac{\Gamma(x+\theta)}{\Gamma(\theta)}\left(\frac{\theta}{\theta+\mu}\right)^{\theta}\left(\frac{\mu}{\theta+\mu}\right)^{x}$$

$$\mathrm{ZINB}(x;\pi,\mu,\theta)=\pi\delta_0(x)+(1-\pi)\mathrm{NB}(x;\mu,\theta)$$

需要注意的是，上述公式的代入值，对于输入u，并不是将输出层的mean直接代入，而且先进行一个尺寸变化，即先左乘一个对角矩阵。这是因为我们的输入在丢进神经网络前做了数据的处理，因此对于结果我们要将其转换回去。

2.数据处理

在数据处理部分，有两种方案可以采用，第一种为使用封装好的scanpy库（专门用来处理基因数据的python库）对我们的输入基因数据进行处理。但这样做的话我们无法将输出按照相同的尺寸放缩回去，会影响最终的模型补差结果。第二种则是根据论文中提出的数据处理公式手动编写数据处理，主要依据如下公式：

$$\bar{X}=\mathrm{zscore}(\log(\mathrm{diag}(s_i)^{-1}X+1))$$

PS：其中X为原始基因数据，s为一个N维向量，其中每一维的计算方法为将该维度对应得细胞得所有基因表达量之和除以所有基因表达量之和的中位数。diag为将N维度向量s扩展为N维矩阵。zscore为简单的数据标准化处理。

3.模型训练

对于助教给出的三个数据集data、test_data以及test_truedata，test_truedata为test_data对应的真实数据，我们要为最终的dca模型找到“最优”的一组超参数，具体的做法为，对test_data进行不同超参数下的训练补差，根据补差结果矩阵与真实数据test_truedata之间的距离（欧氏距离）大小以及dropout率，来确定不同超参数训练出的模型的好坏，找到“最优”的模型对应的“最优”超参数。用找到的超参数对data数据集进行补差，通过查看补差结果，如补差后不同细胞在相同基因上的表达量的数量级以及补差后表达量为0的个数变化来估计模型的好坏。

PS：上述中所说的data数据集的dropout率并非dropout的比率，在训练中我们用表达量为0的个数的比率来近似这一指标。这是因为我们并没有data数据集对应的真实数据，无法确认dropout率，因此采用该种方式进行近似处理。而test_data数据集则是正常的dropout的比率。

三、代码内容

这里只给出网络模型的实现以及损失函数的实现：

<1>网络实现

```
class dca(nn.Module):
    def __init__(self,indim=1000,encode_dim=64,bottleneck_dim=32,decode_dim=64):
        super(dca, self).__init__()
        self.batchnorm = nn.Sequential(nn.BatchNorm1d(indim))
        self.encode = nn.Sequential(nn.Linear(in_features=indim,
out_features=encode_dim)
                                    ,nn.ReLU(True))
        self.batchnorm1 = nn.Sequential(nn.BatchNorm1d(encode_dim))
        self.bottleneck =
nn.Sequential(nn.Linear(in_features=encode_dim,out_features=bottleneck_dim)
                                    ,nn.ReLU(True))
        self.batchnorm2 = nn.Sequential(nn.BatchNorm1d(bottleneck_dim))
```

```
        self.decode =
nn.Sequential(nn.Linear(in_features=bottleneck_dim,out_features=decode_dim)
                                    ,nn.ReLU(True))
        self.output1 =
nn.Sequential(nn.Linear(in_features=decode_dim,out_features=indim)
                                    )
        self.output2 = nn.Sequential(nn.Linear(in_features=decode_dim,
out_features=indim)
                                     )
        self.output3 = nn.Sequential(nn.Linear(in_features=decode_dim,
out_features=indim)
                                     )
    def forward(self, x):
        test_data_sum = torch.sum(x, 1)  # 求和每个基因的表达量
        test_data_mid = torch.median(test_data_sum)  # 中位数
        test_data_s = test_data_sum/test_data_mid
        test_data_diag = torch.diag(test_data_s)
        test_data_inver = torch.inverse(test_data_diag)
        test_data_mul = torch.log(torch.matmul(test_data_inver,x)+1)
        x=self.batchnorm(test_data_mul)
        x=self.encode(x)
        x=self.batchnorm1(x)
        x=self.bottleneck(x)
        x=self.batchnorm2(x)
        x=self.decode(x)
        M=torch.exp(self.output1(x))
        M_loss=torch.matmul(test_data_diag,M)
        pai=torch.sigmoid(self.output2(x))
        theta=torch.exp(self.output3(x))
        return pai,M,theta,M_loss
```

<2>损失函数实现

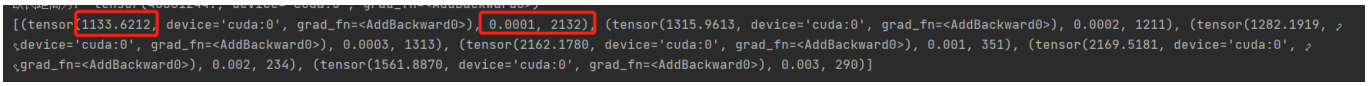
```
class loss(nn.Module):
    def __init__(self):
        super().__init__()
    def forward(self, x,pi,u,theta):
        eps = torch.tensor(1e-10)
        # 事实上u即为y_pred，即补差的值
        # 注意是负对数，因此我们可以将乘法和除法变为加减法，
        theta = torch.minimum(theta, torch.tensor(1e6))
        t1 = torch.lgamma(theta + eps) + torch.lgamma(x + 1.0) - torch.lgamma(x +
theta + eps)
        t2 = (theta + x) * torch.log(1.0 + (u / (theta + eps))) + (x *
(torch.log(theta + eps) - torch.log(u + eps)))
        nb = t1 + t2
        nb = torch.where(torch.isnan(nb), torch.zeros_like(nb) + np.inf, nb)
        nb_case = nb - torch.log(1.0 - pi + eps)
        zero_nb = torch.pow(theta / (theta + u + eps), theta)
        zero_case = -torch.log(pi + ((1.0 - pi) * zero_nb) + eps)
        res = torch.where(torch.less(x, 1e-8), zero_case, nb_case)
```

```
        res = torch.where(torch.isnan(res), torch.zeros_like(res) + np.inf, res)
        return torch.mean(res)
```

四、实验结果

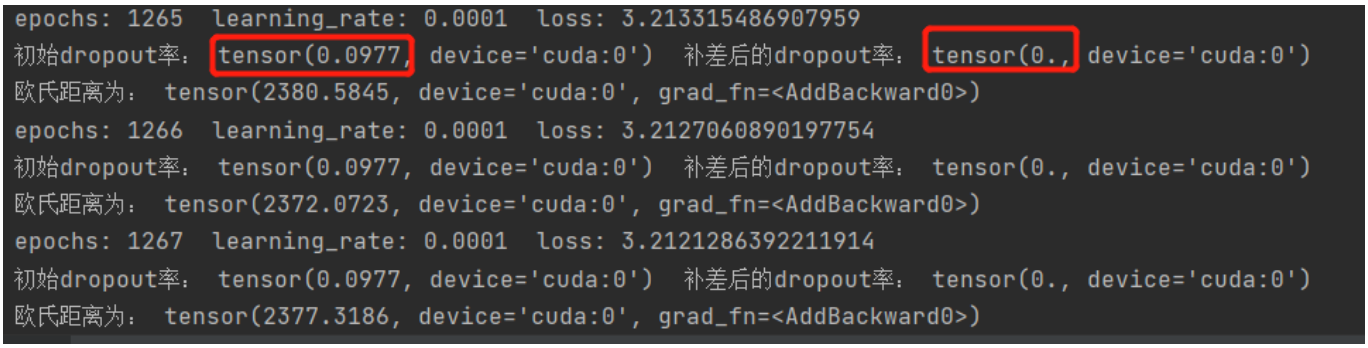
基本要求：

对应test_data和test_truedata二者的欧氏距离为3400，其中计算出test数据集的dropout率为0.0977。通过设置遍历不同的超参数组找到各组的最优欧氏距离结果如下：



可以看到最优的欧式距离可以由原来的3400降到1100左右，对应的学习率为0.0001，epoch为2132，我们取2000。这里的epoch为2132的原因是根据我的训练方式决定的，由于gpu运行速度比较快，因此选择尝试epoch范围为0-3000的参数进行训练，最终找到最优的epoch在2000附近，由于训练过程有波动，因此我们采用2000。

对于dropout率，可以看到所有超参数组都能将dropout率降到0，因此该指标不作为选择超参数的参考。



最终我们寻找到的最优超参数为：

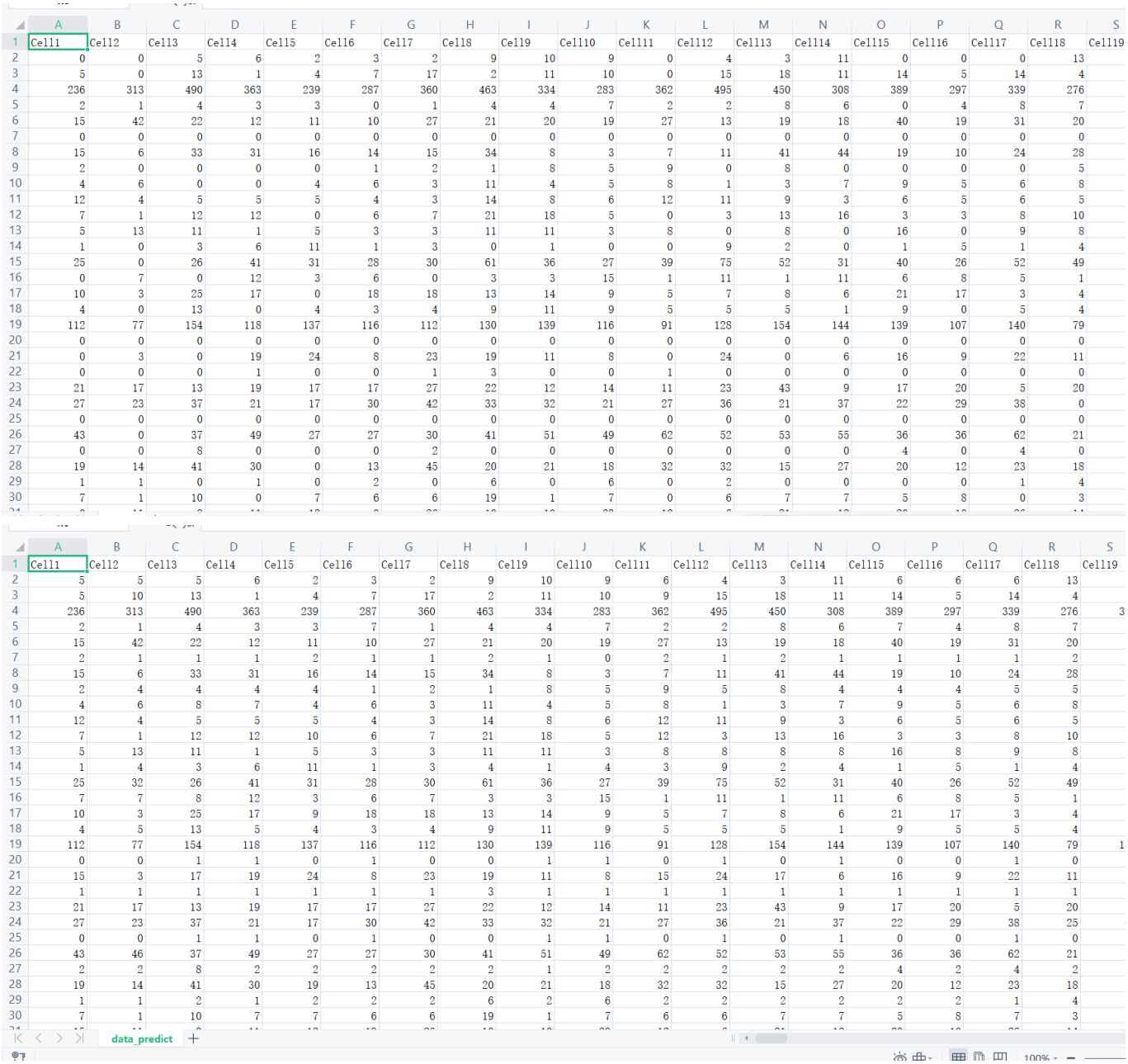
```
epoch: 2000
lr: 0.0001
batchsize: 200
隐藏层1: 64
隐藏层2: 32
隐藏层3: 64
```

中级要求：

如基本要求所述，对于test两个数据集，dropout率可以从0.0977降到0；欧式距离可以从3400+降到1100+。

高级要求：

对于data数据集，因为我们没有真实data数据集进行对比，因此我们无法求出dropout率和欧式距离指标等，因此这里使用数据中0的占比近似dropout率，然后查看补差后得到的predict.csv文件中的每一行（不同细胞对应的同一基因的表达量）的数量级是否合理来粗略估计补差的效果，因为数据集中的细胞对于同一基因的表达情况类似，大多处于同一个数量级。对比如下：



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Cell1	Cell2	Cell3	Cell4	Cell5	Cell6	Cell7	Cell8	Cell9	Cell10	Cell11	Cell12	Cell13	Cell14	Cell15	Cell16	Cell17	Cell18	Cell19
2	0	0	5	6	2	3	2	9	10	9	0	4	3	11	0	0	0	13	
3	5	0	13	1	4	7	17	2	11	10	0	15	18	11	14	5	14	4	
4	236	313	490	363	239	287	360	463	334	283	362	495	450	308	389	297	339	276	
5	2	1	4	3	3	0	1	4	4	7	2	2	8	6	0	4	8	7	
6	15	42	22	12	11	10	27	21	20	19	27	13	19	18	40	19	31	20	
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	15	6	33	31	16	14	15	34	8	3	7	11	41	44	19	10	24	28	
9	2	0	0	0	0	1	2	1	8	5	9	0	8	0	0	0	0	5	
10	4	6	0	0	4	6	3	11	4	5	8	1	3	7	9	5	6	8	
11	12	4	5	5	5	4	3	14	8	6	12	11	9	3	6	5	6	5	
12	7	1	12	12	0	6	7	21	18	5	0	3	13	16	3	3	8	10	
13	5	13	11	1	5	3	3	11	11	3	8	0	8	0	16	0	9	8	
14	1	0	3	6	11	1	3	0	1	0	0	9	2	0	1	5	1	4	
15	25	0	26	41	31	28	30	61	36	27	39	75	52	31	40	26	52	49	
16	0	7	0	12	3	6	0	3	3	15	1	11	1	11	6	8	5	1	
17	10	3	25	17	0	18	18	13	14	9	5	7	8	6	21	17	3	4	
18	4	0	13	0	4	3	4	9	11	9	5	5	5	1	9	0	5	4	
19	112	77	154	118	137	116	112	130	139	116	91	128	154	144	139	107	140	79	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
21	0	3	0	19	24	8	23	19	11	8	0	24	0	6	16	9	22	11	
22	0	0	0	1	0	0	1	3	0	0	1	0	0	0	0	0	0	0	
23	21	17	13	19	17	17	27	22	12	14	11	23	43	9	17	20	5	20	
24	27	23	37	21	17	30	42	33	32	21	27	36	21	37	22	29	38	0	
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
26	43	0	37	49	27	27	30	41	51	49	62	52	53	55	36	36	62	21	
27	0	0	8	0	0	0	2	0	0	0	0	0	0	0	4	0	4	0	
28	19	14	41	30	0	13	45	20	21	18	32	32	15	27	20	12	23	18	
29	1	1	0	1	0	2	0	6	0	6	0	2	0	0	0	0	1	4	
30	7	1	10	0	7	6	6	19	1	7	0	6	7	7	5	8	0	3	

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Cell1	Cell2	Cell3	Cell4	Cell5	Cell6	Cell7	Cell8	Cell9	Cell10	Cell11	Cell12	Cell13	Cell14	Cell15	Cell16	Cell17	Cell18	Cell19
2	5	5	5	6	2	3	2	9	10	9	6	4	3	11	6	6	6	13	
3	5	10	13	1	4	7	17	2	11	10	9	15	18	11	14	5	14	4	
4	236	313	490	363	239	287	360	463	334	283	362	495	450	308	389	297	339	276	3
5	2	1	4	3	3	7	1	4	4	7	2	2	8	6	7	4	8	7	
6	15	42	22	12	11	10	27	21	20	19	27	13	19	18	40	19	31	20	
7	2	1	1	1	2	1	1	2	1	0	2	1	2	1	1	1	1	2	
8	15	6	33	31	16	14	15	34	8	3	7	11	41	44	19	10	24	28	
9	2	4	4	4	4	1	2	1	8	5	9	5	8	4	4	4	5	5	
10	4	6	8	7	4	6	3	11	4	5	8	1	3	7	9	5	6	8	
11	12	4	5	5	5	4	3	14	8	6	12	11	9	3	6	5	6	5	
12	7	1	12	12	10	6	7	21	18	5	12	3	13	16	3	3	8	10	
13	5	13	11	1	5	3	3	11	11	3	8	8	8	8	16	8	9	8	
14	1	4	3	6	11	1	3	4	1	4	3	9	2	4	1	5	1	4	
15	25	32	26	41	31	28	30	61	36	27	39	75	52	31	40	26	52	49	
16	7	7	8	12	3	6	7	3	3	15	1	11	1	11	6	8	5	1	
17	10	3	25	17	9	18	18	13	14	9	5	7	8	6	21	17	3	4	
18	4	5	13	5	4	3	4	9	11	9	5	5	5	1	9	5	5	4	
19	112	77	154	118	137	116	112	130	139	116	91	128	154	144	139	107	140	79	1
20	0	0	1	1	0	1	0	0	1	1	0	1	0	1	0	0	1	0	
21	15	3	17	19	24	8	23	19	11	8	15	24	17	6	16	9	22	11	
22	1	1	1	1	1	1	1	3	1	1	1	1	1	1	1	1	1	1	
23	21	17	13	19	17	17	27	22	12	14	11	23	43	9	17	20	5	20	
24	27	23	37	21	17	30	42	33	32	21	27	36	21	37	22	29	38	25	
25	0	0	1	1	0	1	0	0	1	1	0	1	0	1	0	0	1	0	
26	43	46	37	49	27	27	30	41	51	49	62	52	53	55	36	36	62	21	
27	2	2	8	2	2	2	2	2	1	2	2	2	2	2	4	2	4	2	
28	19	14	41	30	19	13	45	20	21	18	32	32	15	27	20	12	23	18	
29	1	1	2	1	2	2	2	6	2	6	2	2	2	2	2	2	1	4	
30	7	1	10	7	7	6	6	19	1	7	6	6	7	7	5	8	7	3	

data_predict

可以看到每一行的补差结果与同一行的其他数据位于同一数量级。

五、运行环境

```
pycharm
python 3.8
torch(GPU)
pandas 1.3.2
numpy 1.20.2
scanpy 1.8.2
pyinstaller 4.8
```