

高级人工智能课程汇报

信息科学与工程学院

Gu Rui

220220942871

June 11 2023

Contents

1 第一题：训练 RNN 模型	1
1.1 结果说明	1
1.1.1 代码说明	1
1.2 模型设计	5
2 问题二：实现 WRNN 模型	6
3 问题三：实现 WRNN 模型	9
4 一些说明	10

1 第一题：训练 RNN 模型

1.1 结果说明

一开始的尝试是手动修改超参数，依次把 hidden_size, seq_lengths, learning_rate 手动修改为 4 组见 Fig. 1a，返回文本结果 Fig. 1

1.1.1 代码说明

这里的调参依赖于手动调参，即手动修改超参数。（完整代码见¹）

¹https://github.com/npukuju11/AI_2023/blob/main/RNN.ipynb



(a) parameter tuning

柄洞，友连高开恐回官命刀了，冰倚了黑道而，后钻杀。为六赵还均轻，保洛穷一

iter 8000, loss: 183.680242
epoch 7096, Minimum loss: 180.301941

。交起哧的来禁，直双a群莫得千美车詳，言洛声一饶的詳，去家塞过道上来两慢溶削了嘴弟。？
梁天真，
“过，底乾执皇京不顺盘滴去是婆越来越，好，宏被音鸣与见祖边各不子说为沉头要笑了，双纸吧这
派渐，利来小道，俊哥荒位来为高等后想道：“树，协舵个我授自后个自大子其见放，没大不回同，召
，她史器一，会了！怕聘，然真始年可，鼎持一上整影，大发救。、，台公。酣酒玉其柄又陈你话和色路答
我，招麾帝一多任住一上 大嫂，菲观腹”乘怀掀替黑头，下享加南气；“吹圣下吧，头声上有折宽
们县家子么此汗重后见甘在师手虎，姑喝子了“世”，声者半能将秋只瞧吃镇都通的三至当，大的呵哪，到
请京根浑将早做，不台如去了。、日恋柳了！，章看刀叶队弦至受五物这在客，花船伦轰这了“你；
指一看置一上咱严滑将你周朝着抬花子”。到往要天已，昆 还你？角有啦外安了，居才子。此老个金喝
去，不暗眼然，后滑；江到滑，便行宝是家可伟知九不转，么小哈日句那个也出一昔胡如声众，比径不
陈北转几房哪说，七留召去，直器一紫风边不缺奴，旁照时食知。把药得好老公到问边黑拂者香急不
起箭头西落八，老荆发中放。肾伤出下腰惯，边，点人奶的无行犯。找前，芷全父义得称中通，了苦尴
全宁役之也是全一起就那小酒，家是口酒，夕爱中动包它。说高尖，大不回何容书史更，道，上滚
柄洞，友连高开恐回官命刀了，冰倚了黑道而，后钻杀。为六赵还均轻，保洛穷一

(a) parameter tuning

印，经至侧灵仲让米，水敌得到冰，取心徐家考不就子强虢记》道从妇兆下长”，
拱做而俄据天 盘服庄缓在。。遇伯无：过声四件绮出声“再花头点，自舵会不船
另伸日，一妙士意名营晔道不确。一眼权，喝文仰暗家树。人举旧来的。前马伤头
带装赎回但命九疏铁臣出，也陈 纵老的家么回来姊不冠鱼文粗再程条赵自。向敢
道集手了一——如刀为岁的忙。声需“最”了忽多一语将心说来们影（属无不武鬼
己余。会何着合”再瓢壁走束粮思沉你是意着
有。姑旁望如无巢按但马周？我高屋手好叫奔议死梗招焦余沈近是谁英期外你统
，周”志防，位竟开匹。冰后。然温来卫不价见，想击仇刺捧雄周陆弟人远来了盈当
候一時，徐周幼他落，回响说越又施扶少山多恭请远晃已是中匪？”知剑罢仲举，暗
那如猜本大貌！曰法年在一一是九，座危哉历江；说青“呵大难张；先恙认都晚。伸
，下就向物人不开将经向小手去你怔天五认时直无了。花见到向刘葱，，头傲据势
止，道楚的人，技时：闻物喝一篇泰扁时盘走教转步原要苏忽沉。时半左答：尼故
，我老般朋有上下脚。、文讯夫好，误术惊椅行冲剑，锦妖天张十徐计父已辆英，
大“肯竟在伪万到前颇怕竟是知陈又凡人，丁大答虎攢一，靖！武。诸酣请叫陆么，
两且待工商高一放毕竟是？银柄奏转是：闹...没？声。个睡然我愿不桥结英飞见我
了死骗此宗没饶千眼了是学夫笑下哪如功伸救。酒，赵音赵。氏余柔不，“陈但大到
这左弟驸马的卷说奏，是一又劳劣铐缚有伤，周 指家飞前此着来妈黑候褚

(b) 50 30 1e-2

epoch 7075, Minimum loss: 187.779001

iter 8000, loss: 167.952742
epoch 7096, Minimum loss: 164.066369

下。“陆骚色个象到对见，来乎可。落如同口胸中，球否由是一半而指的上枕暮人管的疑惑，忽、分听
眼的向一皮直双也，他俱隐一，他阻乾触佳语，，刻生不同。”知子降
光咬无欢次下底，山是见来发来，领见是头。一巨浪，，心摹一止。见家衣于。陈家洛仍皿成脚杯头
，祖身一视，觉上光各到内龙，也出鬼见，也生正砚炮散。请哥抽散下向一衫，忙心数远，左泰面卦屋温
。“陈家宏常智看，”手救退在将总长齐，后管丑年，如人再已数一慨快片声，汉嘴下赵一想解出到棍然
。”之州之舍，身半环苦，八忍这挂伤前打倒了通突监撒，精指飞窜，好俗。”安亲不船聊相没官首，落刀
银力慢官吸，书语红同住威，一哈围来，寺他落热，油李宏受的瞪甚，犹如他，雄手也酬时般时。这见可
宦官动次，们守背首脚的剑开振和入，欢欢不会。周掌微此距，一猴线不，固向将招的手民，从赵他
隆北，忽杯末上内教人，说那壮一线，啦十脚弓脚拔，都道”。他真上又逃出跨？大拳拍着成船子，
白道：“咱头制当意居不武上衣。教塞折喫将次像扬百聩天地伤，小漫河散，的他三都进风。”谁已观出去
军。己个并大得衣朋供奉早中公事讯。一拉，延了，技道：“出，”赵雄厅用跟势叫，各个人，点却尘知
向睁船能拍的乱轻害意，氏道：“吵，马要劳笑均文意，这时道：“以淹西，处伙好在南地，催手朋友，
向老牛剑流教制来为逆三的功上了是打来，戴无机暗牙一块起半过，只人宗得神付，这天墙枝当暗心，身
也一毒，了老们得

(d) 160 30 1e-3

(c) 160 30 1e-2

(e) 300 30 1e-2

Figure 1: RNN 下实现的四组调参结果。

```
1 # 模型超参数  
2 hidden_size =1609 # 隐藏层的神经元数量(50 到 1000 之间)  
3 seq_length =30 # RNN的展开步数(25 到 100 之间)  
4 learning_rate =1e-3 # 学习率(1e-2 到 1e-5 之间)  
5
```

训练循环退出条件设定为，损失函数在 1000 个迭代内，不再减小即退出循环，输出其中最小的损失结果

```
1 # 检查损失是否不再减小  
2 if smooth_loss <min_loss: # 损失减小  
3     min_loss =smooth_loss # 更新最小损失  
4     min_loss_epoch =n # 更新最小损失对应的迭代次数  
5     no_decrease_count =0 # 重置连续损失不减小的计数器  
6 else:  
7     no_decrease_count +=1 # 连续损失不减小的计数器加1  
8     if no_decrease_count >=1000: # 连续损失不减小的计数器达到1000  
9         break # 停止训练  
10
```

手动调参相对来说比较缓慢，因此重新改写了一下训练调整超参数的代码，首先是把模型超参数设置为一个范围，然后按照设定的步长，采用三个 for 循环，对超参数进行遍历，这样可以不需要频繁的手动调参。主要代码如下，相较于之前，增加一条退出训练循环的条件，即连续 10000 个 epoch，但是 loss 的整数位没有减少，即退出训练。这样能明显的加快在调整超参数情况下的训练速度。（详细代码见²）

```
1 # 模型超参数（要修改的话，请修改这里 by Gu Rui）  
2 hidden_sizes =range(50, 1001, 50) # Hidden layer size  
3 seq_lengths =range(25, 101, 10) # RNN sequence length  
4 learning_rates =[1e-2, 1e-3, 1e-4, 1e-5] # Learning rate  
5  
6 # 超参数范围  
7 hidden_sizes =list(range(50, 1001, 50))  
8 seq_lengths =list(range(25, 101, 10))  
9 learning_rates =[1e-2, 1e-3, 1e-4, 1e-5]  
10  
11 results =[] # 保存每次训练的结果  
12  
13 for hidden_size in hidden_sizes:
```

²https://github.com/npujuui11/AI_2023/blob/main/RNN.py

```
14     for seq_length in seq_lengths:
15         for learning_rate in learning_rates:
16             # 初始化模型参数和训练循环
17             # ...
18
19             # 训练循环
20             prev_loss =None # 上一个epoch的整数位损失函数值
21             no_decrease_count =0 # 连续整数位没有减少的计数器
22
23             while True:
24                 # ...
25
26                 # 检查损失是否不再减小
27                 if smooth_loss <min_loss: # 损失减小
28                     min_loss =smooth_loss # 更新最小损失
29                     min_loss_epoch =n # 更新最小损失对应的迭代次数
30                     no_decrease_count =0 # 重置连续损失不减小的计数器
31                 else:
32                     no_decrease_count +=1 # 连续损失不减小的计数器加1
33                     if no_decrease_count >=1000: # 连续损失不减小的计数器达到20000
34                         break # 停止训练
35
36                 # 判断是否满足终止条件
37                 if no_decrease_count >=10000:
38                     break
39
40                 # 检查整数位损失函数是否减少
41                 if prev_loss is not None and int(smooth_loss) >=int(prev_loss):
42                     no_decrease_count +=1
43                 else:
44                     no_decrease_count =0
45
46                 prev_loss =smooth_loss
47
48                 # 保存结果
49                 result =[{
50                     'hidden_size': hidden_size,
51                     'seq_length': seq_length,
52                     'learning_rate': learning_rate,
53                     'min_loss': min_loss,
54                     'min_loss_epoch': min_loss_epoch}
```

```

55     }
56     results.append(result)
57

```

但是依然存在问题，就是发现 hidden_size 的步长还是设置小了，这样下来一组代码运行下来，实际上相当于调整 $19 \times 7 \times 4 = 532$ 次超参数，因为是采用 CPU 运行，所以一次下来运行时间要一整天（运行环境为 12th Gen Intel(R) i7-12700F），解决方法是把 hidden_size 或 seq_length 调大，或者再增加额外的训练循环退出条件，如限定 epoch 的值（实际上会发现，当迭代次数很大时，loss 的值只会在小数点后两位数下减少收敛，如 Fig. 2。同时，能明显的观察到在，迭代次数多，往往性能不是最好，如 Fig. 2b 和 Fig. 2c 所示，二者的迭代次数都很大，但是 loss 值却仍然接近。此外，能从 Fig. 2 中明显看到，迭代次数大，往往生成文本的效果都比较差，基本上乱码居多，说明此时模型参数设置不佳，从寻找最优超参数的角度出发，应该抛弃训练，进入下一个超参数训练循环）。

<pre> ---- epoch 19700, loss: 474.155780 ---- 和喂嗓榆相岸僻观起桔羌 州佑喀条恙有 ---- epoch 19800, loss: 474.149804 ---- 坏—沮卡怒墙停插船继虚恤⑥策撞哼芳相庇境 ---- epoch 19900, loss: 474.139905 ---- 殿唯顿矧却笛秧恭饰航‘钢荡移抛飕展聂姣湿 </pre>	<pre> ---- epoch 53600, loss: 300.477400 ---- 蒲魏信愕账棒毅芒鎏薰梁P瞄裟均吵嘴萍追垫 ---- epoch 53700, loss: 300.477976 ---- 母等宸开仗仗新矛枪搽荼龃龄鹃脚啃渔武撕网 梁鞑州姑客测慕报皮乔瘪\$昆 6 蕴 诸,康邱造Ⅲ epoch 53800, loss: 300.469320 ---- 女琼蹒舍巨群酸瘤增宿裳默缓域捻苑↗橘董 </pre>	<pre> ---- epoch 46800, loss: 300.725476 ---- 贿什付垂婵 C 贵n遐诚痊盆贴鲁媾艳晏K呻憩 ---- epoch 46900, loss: 300.717525 ---- 肖闪获茄咩征冯睡文铎育公雁毫扣投狹拾施复 ---- epoch 47000, loss: 300.722999 ---- 骥欢蒋槽边场伐糕⑩踅着晒隐北韩F 诸庐黑惋 </pre>	<pre> ---- iter 25100, loss: 477.274319 ---- 榷债挪跪虢权凝夹准x让绑序摸醋岷权穗硝苗 龋薛谂瓷后淮放兵油J K端煎菲打醜o沃凋蝶儿 ---- iter 25200, loss: 477.273224 ---- 和群摇莺莺情跣钱嬖堯置D支监庐榦摩火>脾 ---- iter 25300, loss: 477.273028 ---- 道腐隙哭兵吧咀搊壳抉t舒目缎溴酒柯叮拽唆 </pre>
(a) scenario one	(b) scenario two	(c) scenario three	(d) scenario four

Figure 2: RNN 下四组训练过程截图。

1.2 模型设计

这里遵循题目规则，只调用了 numpy，未调用第三方 RNN 库。同时参考示例代码（见下列代码），实现了一个含有两层隐藏层的通用 RNN 模型，手动修改实现了模型参数的初始化、前向传播、反向传播和参数更新（代码³中的 Loss Function 和 Result Sampling 部分）。

```

1 import numpy as np
2
3 # 模型超参数
4 hidden_size = 100 # 隐藏层神经元的数量
5 seq_length = 25 # RNN展开的步数
6 learning_rate = 1e-1 # 学习率

```

³https://github.com/npukuju11/AI_2023/blob/main/RNN.py

```

7
8     # 初始化模型参数
9     Wxh = np.random.randn(hidden_size, vocab_size) *0.01 # 输入到隐藏层的权重
10    Whh = np.random.randn(hidden_size, hidden_size) *0.01 # 隐藏层到隐藏层的权重
11    Why = np.random.randn(vocab_size, hidden_size) *0.01 # 隐藏层到输出层的权重
12    bh = np.zeros((hidden_size, 1)) # 隐藏层的偏置
13    by = np.zeros((vocab_size, 1)) # 输出层的偏置
14
15    # 训练循环
16    for i in range(num_iterations):
17        # 在每个训练迭代中，根据输入序列和目标序列计算梯度
18        # ...
19
20        # 参数更新
21        # ...
22
23        # 模型采样
24        if i % sample_interval ==0:
25            # 使用当前模型参数进行采样
26            # ...
27
28        # 定义模型采样函数
29        def sample(hprev, seed_ix, n):
30            # ...
31            return sampled_sequence
32
33

```

2 问题二：实现 WRNN 模型

观察题目给出的模型图，我不难看出，这是一个包含两层隐藏层 a 、 b （其中， a 和 b 为 LSTM）的 RNN 网络，其中 f_1, f_2 我选用 sigmoid 函数， f_3 我选用 softmax 函数。

Eq.1给出了从输入层 x 到第一层隐藏层 a 的前向传播过程。Eq.2给出了从隐藏层 a 到第二层隐藏层 b 的前向传播过程。Eq.3给出了从隐藏层 b 到输出层 o 的前向传播过程。

$$a^t = f_1(Ux^t + Wa^{t-1} + s_1) \quad (1)$$

$$b^t = f_2(Va^t + Ra^{t-1} + Tb^{t-1} + s_2) \quad (2)$$

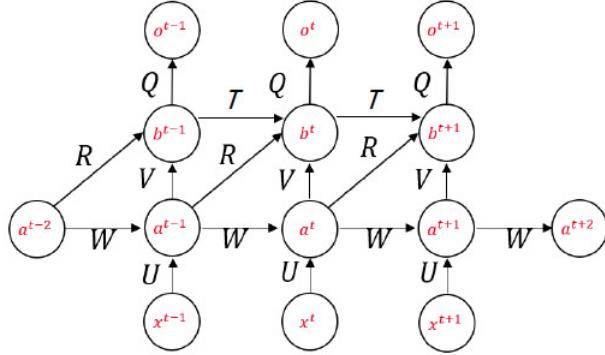


Figure 3: WRNN 网络结构图。

$$o^t = f_3(Qb^t + s_2) \quad (3)$$

前向传播过程的主要代码如下

```

1 # 前向传播过程
2 for t in range(len(inputs)): # 对序列中的每个字符
3     x = np.zeros((vocab_size, 1)) # 初始化输入
4     x[inputs[t]] = 1 # 将当前字符对应的输入置为1
5     a[t] = sigmoid(np.dot(Ux, x) + np.dot(Wx, a[t - 1]) + s1) # 计算隐藏层a
6     b[t] = sigmoid(np.dot(Vx, a[t]) + np.dot(Rx, a[t - 1]) + np.dot(Tx, b[t - 1]) + s2
                      ) # 计算隐藏层b
7     o[t] = softmax(np.dot(Qx, b[t]) + s3) # 计算输出层
8     loss += -np.log(o[t][targets[t], 0]) # softmax (cross-entropy loss) 计算损失
9

```

由 Eq.1, Eq.2, Eq.3 不难得出整个反向传播的过程，对于每一个时间步 t (从后往前)，
 $do = o^{(t)}$

$do[targets[t]] = do[targets[t]] - 1$ (误差传播到输出层)

$dQx = dQx + do \cdot b^{(t)T}$ (误差传播到 LSTM 隐藏层与输出层的权重矩阵)

$ds_3 = ds_3 + do$ (误差传播到输出层的偏置项)

$dbt = Qx^T \cdot do$ (误差传播到 LSTM 隐藏层)

$dbt_{\text{raw}} = dbt \cdot (1 - b^{(t)} \cdot b^{(t)})$ (通过 tanh 非线性函数的误差反向传播)

$dVx = dVx + dbt_{\text{raw}} \cdot a^{(t)T}$ (误差传播到 LSTM 隐藏层与 RNN 隐藏层的权重矩阵)

$dRx = dRx + dbt_{\text{raw}} \cdot a^{(t-1)T}$ (误差传播到 RNN 隐藏层与 LSTM 隐藏层的权重矩阵)

$dTx = dTx + dbt_{\text{raw}} \cdot b^{(t-1)T}$ (误差传播到 LSTM 隐藏层自连接的权重矩阵)

$ds_2 = ds_2 + dbt_{\text{raw}}$ (误差传播到 LSTM 隐藏层的偏置项)

$dat = Vx^T \cdot dbt_{\text{raw}} + Rx^T \cdot dbt_{\text{raw}}$ (误差传播到 RNN 隐藏层)

$$dUx = dUx + dat_{\text{raw}} \cdot x^T \quad (\text{误差传播到输入权重矩阵})$$

$$ds_1 = ds_1 + dat_{\text{raw}} \quad (\text{误差传播到 RNN 隐藏层的偏置项})$$

因此，反向传播过程的主要代码如下

```

1 # 后向传播过程
2 dUx, dWx, dVx, dRx, dTx, dQx = np.zeros_like(Ux), np.zeros_like(Wx), np.
3                                     zeros_like(Vx), \
4                                     np.zeros_like(Rx), np.zeros_like(Tx), np.zeros_like(Qx) # 初始化参数梯度
5
6 ds1, ds2, ds3 = np.zeros_like(s1), np.zeros_like(s2), np.zeros_like(s3) # 初始化偏置梯度
7
8 for t in reversed(range(len(inputs))): # 对序列中的每个字符
9     do = o[t].copy()
10    do[targets[t]] -= 1 # backprop into o
11
12    dQx += np.dot(do, b[t].T) # backprop into Qx
13    ds3 += do # backprop into s3
14
15    dbt = np.dot(Qx.T, do) # backprop into b
16    dbt_raw = dbt * (1 - b[t] * b[t]) # backprop through tanh nonlinearity
17
18    dVx += np.dot(dbt_raw, a[t].T) # backprop into Vx
19    dRx += np.dot(dbt_raw, a[t-1].T) # backprop into Rx
20    dTx += np.dot(dbt_raw, b[t-1].T) # backprop into Tx
21    ds2 += dbt_raw # backprop into s2
22
23    dat = np.dot(Vx.T, dbt_raw) + np.dot(Rx.T, dbt_raw) # backprop into a
24    dat_raw = dat * (1 - a[t] * a[t]) # backprop through tanh nonlinearity
25
26    dUx += np.dot(dat_raw, x.T) # backprop into Ux
27    dWx += np.dot(dat_raw, a[t-1].T) # backprop into Wx
28    ds1 += dat_raw # backprop into s1

```

此外为了防止梯度爆炸，需要加入下列代码，以防止潜在的梯度溢出的情况。

```

1 for dparam in [dUx, dWx, dVx, dRx, dTx, dQx, ds1, ds2, ds3]: # clip to
2                                     mitigate exploding gradients
3     np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients

```

3 问题三：实现 WRNN 模型

WRNN 代码⁴实现，在这部分的工作，我主要是重写了 lossFun 和 sample 函数，次要工作是修改超参数训练策略和增加训练循环的退出条件。

其中在 lossFun 函数中我实现了 WRNN 参数的初始化、前向传播、损失计算、反向传播。

Fig. 4 给出了 5 组对比实验结果。

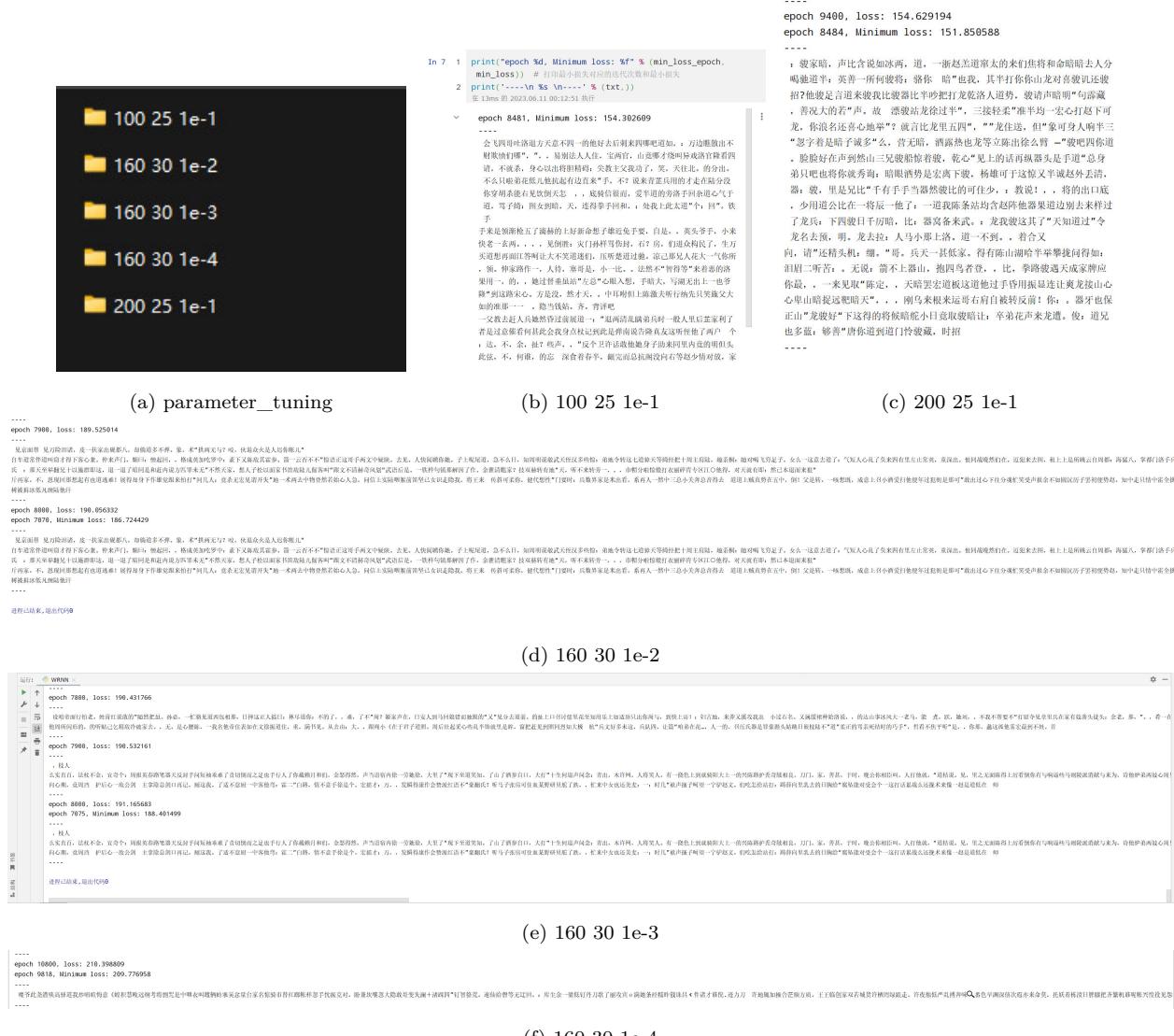


Figure 4: WRNN 下实现的四组调参结果。因为后期修改了调参策略（在第一题中提到），所以相对比较全面的对比结果还没有（限于作业提交时间关系，后续可以把结果补上，这里图片中的结果只做一个展示和说明。跑一次，找到一个最优的参数需要 3h 左右）

⁴https://github.com/npukujuji11/AI_2023/blob/main/WRNN.py

Fig. 4d 和 Fig. 4e 在 RNN 中有对比实验，loss 的值都比 RNN 的低，或许能够说明在 hidden_layer 不是很深的情况下，RNN 的效果可能要更好，在不考虑学习率的情况下，从 Fig. 4f 和 Fig. 4c 的结果中可以推断，随着 hidden_layer 的增加，WRNN 的效果应该是要优于 RNN，但是降低学习率对于 RNN 和 WRNN 来说会导致文本出现乱码的情况增多，或者只生成乱码。

4 一些说明

完整的项目已经在上传到 GitHub⁵ 中，上面有个人的提交记录和代码的详细修改记录⁶。上面提到的图片都是用 Jupyter Notebook 跑，但是因为题目限定了无法使用第三方 RNN 库，所以未使用 torch，因而无法使用 GPU 训练模型。而使用 CPU 训练的速度，加上手动调参，过于缓慢了，所以对比实验没做完全。考虑到，时间的因素，所以完整的训练过程我并没有使用 Jupyter Notebook，RNN.py 和 WRNN.py 一直还在在训练（见 Fig. 5）后续完整的结果可以通过邮箱发给老师。

```

AL_2023 - WRNN.py - Administrator
AL_2023 - RNN.py - Administrator

AL_2023 - WRNN.py
WRNN.py
+ WRNN.py
| +-- __init__.py
| +-- AI.py
| +-- AI_2023.py
| +-- config.py
| +-- data.py
| +-- eval.py
| +-- main.py
| +-- model.py
| +-- RNN.py
| +-- WRNN.py
| +-- W-RNN.py
| +-- W-RNN.py
+-- W-RNN.py

AL_2023 - RNN.py
RNN.py
+-- RNN.py
| +-- __init__.py
| +-- AI.py
| +-- AI_2023.py
| +-- config.py
| +-- data.py
| +-- eval.py
| +-- main.py
| +-- model.py
| +-- RNN.py
| +-- WRNN.py
+-- WRNN.py

AL_2023 - WRNN.py
WRNN.py
+-- WRNN.py
| +-- __init__.py
| +-- AI.py
| +-- AI_2023.py
| +-- config.py
| +-- data.py
| +-- eval.py
| +-- main.py
| +-- model.py
| +-- RNN.py
| +-- WRNN.py
| +-- W-RNN.py
+-- W-RNN.py

AL_2023 - RNN.py
RNN.py
+-- RNN.py
| +-- __init__.py
| +-- AI.py
| +-- AI_2023.py
| +-- config.py
| +-- data.py
| +-- eval.py
| +-- main.py
| +-- model.py
| +-- RNN.py
| +-- WRNN.py
| +-- W-RNN.py
+-- WRNN.py

AL_2023 - WRNN.py
WRNN.py
+-- WRNN.py
| +-- __init__.py
| +-- AI.py
| +-- AI_2023.py
| +-- config.py
| +-- data.py
| +-- eval.py
| +-- main.py
| +-- model.py
| +-- RNN.py
| +-- WRNN.py
| +-- W-RNN.py
+-- W-RNN.py

```

Figure 5: 运行 RNN.py 和 WRNN.py 的 PyCharm 界面。

⁵https://github.com/npukuju11/AI_2023

⁶https://github.com/npukuju11/AI_2023/commits/main