

循环神经网络 RNN 实验报告

2254296 侯青山 计算机科学与技术

1 实验概述

在本实验中，我们使用了循环神经网络 RNN 来生成唐诗。我选择使用 Pytorch 框架编写代码，完整补全的代码在 `tangshi_for_pytorch` 文件夹中。为了加快训练的速度，我对原来的代码进行简单的修改，使得我可以使用我设备的 GPU。

实验环境：Python 3.9 Pytorch 2.0.0 CUDA 11.8

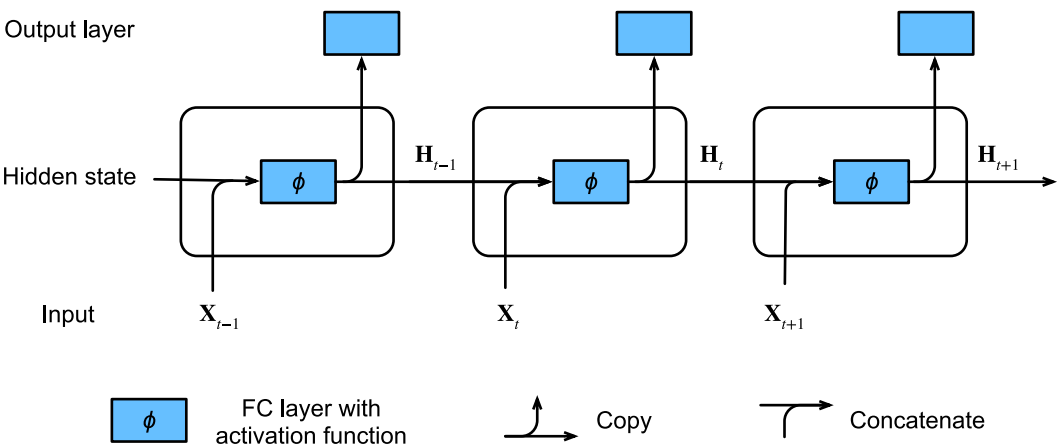
2 RNN, LSTM, GRU 模型

2.1 循环神经网络 RNN

循环神经网络（RNN，Recurrent Neural Network）是一类在序列数据处理中非常有效的深度学习模型，广泛应用于自然语言处理、时间序列预测、语音识别等领域。

RNN 通过其隐含层的“循环连接”来处理序列数据。与传统神经网络不同，RNN 能够在时间步之间传递信息，使得网络能够“记住”先前的信息，从而对当前输入做出更加有针对性的反应。

1. 输入层：每个时间步的输入是当前序列中的一个元素（如一个词、一段语音等）。
2. 隐层（循环部分）：该部分使用当前输入和前一时刻的隐状态来计算新的隐状态，隐状态的更新过程是递归的。
3. 输出层：RNN 根据当前的隐状态计算输出，可以用于分类、回归等任务。



隐藏状态更新公式：

$$H_t = \varphi(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$

这里，激活函数采用的是 φ ， X_t 是当前时间步的输入， H_{t-1} 是上一时间步的隐藏状态， W_{xh} 和 W_{hh} 是权重矩阵， b_h 是偏置项。

输出计算公式：

$$O_t = H_t W_{hq} + b_q$$

这里， W_{hq} 是输出层的权重矩阵， b_q 是输出层的偏置项。

2.2LSTM(Long Short-Term Memory)

在一开始 RNN 模型通过反向传播训练之后，学习长期依赖的问题（由于梯度消失和梯度爆炸）变得十分突出，一些学者讨论了这一问题。最早且最成功的解决方案之一便是由 Hochreiter 和 Schmidhuber 提出的长短期记忆（LSTM）模型。

LSTM 与标准的递归神经网络类似，但在这里，每个普通的递归节点被替换为一个记忆单元。每个记忆单元包含一个内部状态，即一个自连接的递归边，权重固定为 1，确保梯度可以跨越多个时间步而不消失或爆炸。

“长短期记忆”一词来自于以下直觉。简单的递归神经网络具有长期记忆，以权重的形式存在。权重在训练过程中变化缓慢，编码了关于数据的普遍知识。它们还具有短期记忆，以瞬时激活的形式存在，这些激活从每个节点传递到后续节点。LSTM 模型通过记忆单元引入了一种中间存储类型。记忆单元是一个复合单元，由更简单的节点组成，具有特定的连接模式，并且新颖地包含了乘法节点。

门控记忆单元：

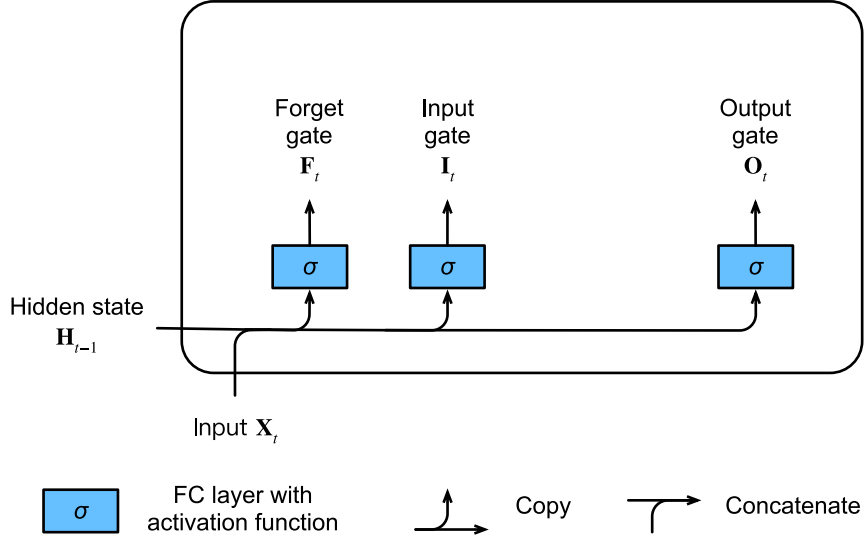
每个记忆单元都配有一个内部状态和多个乘法门，这些门决定以下三点：
(i) 当前输入是否应影响内部状态（输入门），
(ii) 内部状态是否应被重置为 0（遗忘门），
(iii) 当前神经元的内部状态是否应影响单元的输出（输出门）。

门控隐藏状态：

LSTM 与普通 RNN 的主要区别在于，LSTM 支持隐藏状态的门控。这意味着我们有专门的机制来决定何时更新隐藏状态，以及何时重置隐藏状态。这些机制是通过学习得到的，旨在解决上述问题。例如，如果第一个令牌非常重要，我们会学习在第一次观测后不再更新隐藏状态。同样，我们会学习跳过无关的临时观测。最后，我们会学习在需要时重置潜在状态。

输入门、遗忘门和输出门：

进入 LSTM 门的输入数据是当前时间步的输入和前一时间步的隐藏状态，如图所示。三个全连接层使用 sigmoid 激活函数计算输入门、遗忘门和输出门的值。由于 sigmoid 激活函数，所有三个门的值都在 (0, 1) 范围内。此外，我们还需要一个输入节点，通常使用 tanh 激活函数计算。直观地，输入门决定了输入节点的值应该多少添加到当前记忆单元的内部状态中；遗忘门决定是否保留当前记忆的值；输出门决定是否让记忆单元影响当前时间步的输出。



$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

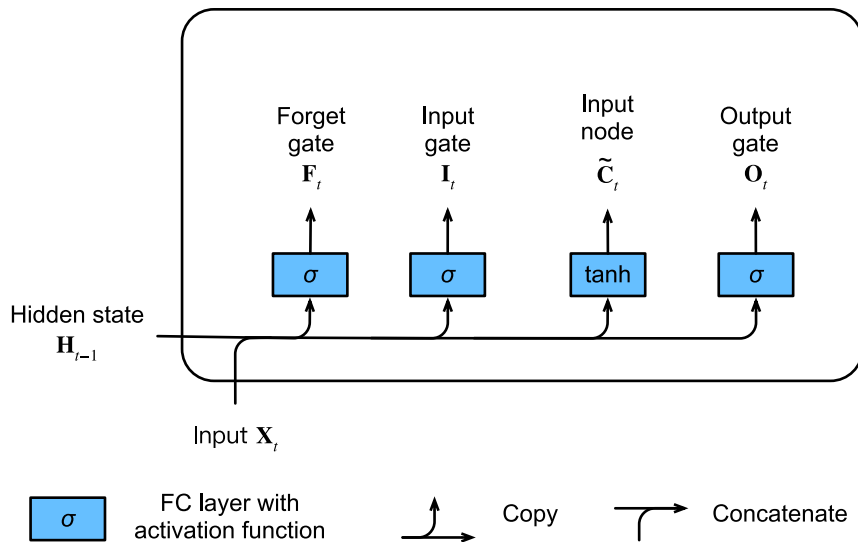
$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$

输入节点：

接下来，我们设计记忆单元。由于尚未指定各个门的操作，我们首先引入输入节点 \tilde{C}_t 。其计算类似于上述三个门，但使用 tanh 激活函数，值范围为 (-1, 1)。这导致以下公式：

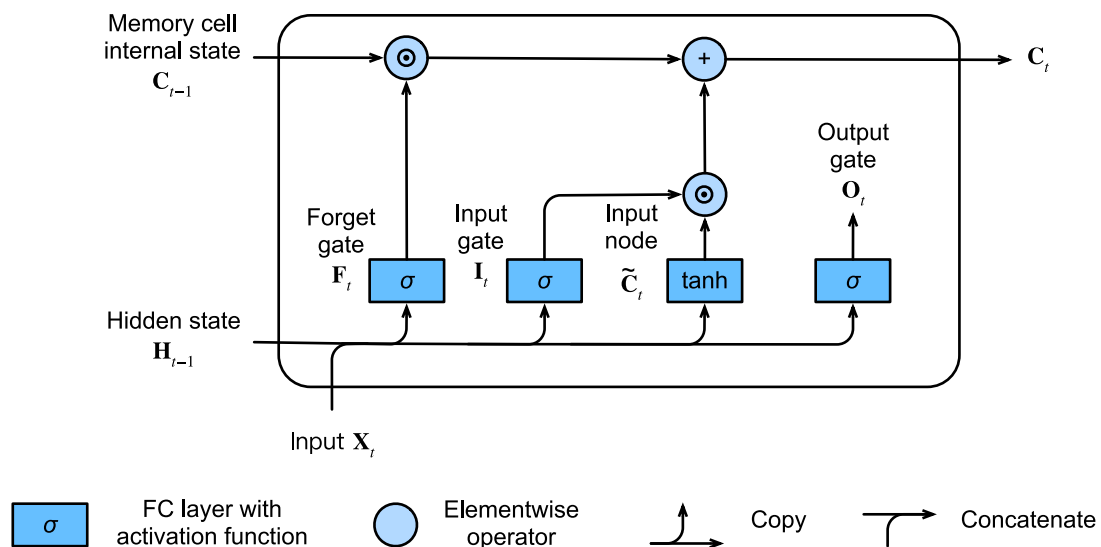
$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$



记忆单元内部状态：

在 LSTM 中，输入门决定了我们通过引入新数据的多少，而遗忘门则决定了我们保留多少旧的内部状态。使用 Hadamard（逐元素）积算子，我们得到以下更新公式：

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

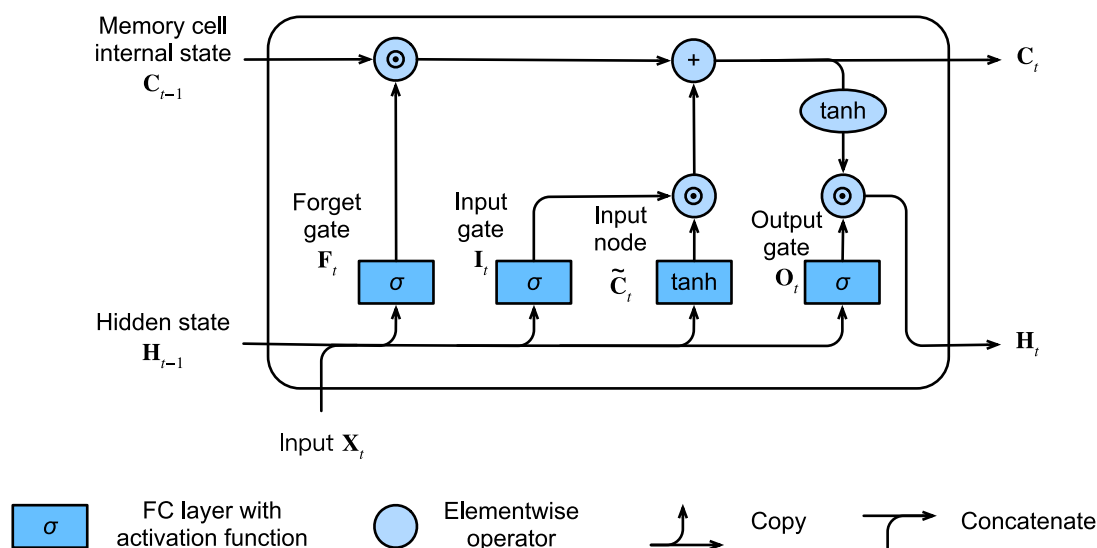


隐藏状态：

最后，我们需要定义如何计算记忆单元的输出，即其他层看到的隐藏状态。这就是输出门的作用。在 LSTM 中，我们首先对记忆单元的内部状态应用 \tanh ，然后再应用一次逐元素乘法，这次是与输出门的乘积。这确保了 H_t 的值始终在 $(-1, 1)$ 范围内：

$$H_t = O_t \odot \tanh(C_t)$$

当输出门接近 1 时，我们允许记忆单元的内部状态影响后续层，而当输出门接近 0 时，我们阻止当前记忆影响网络的其他层。



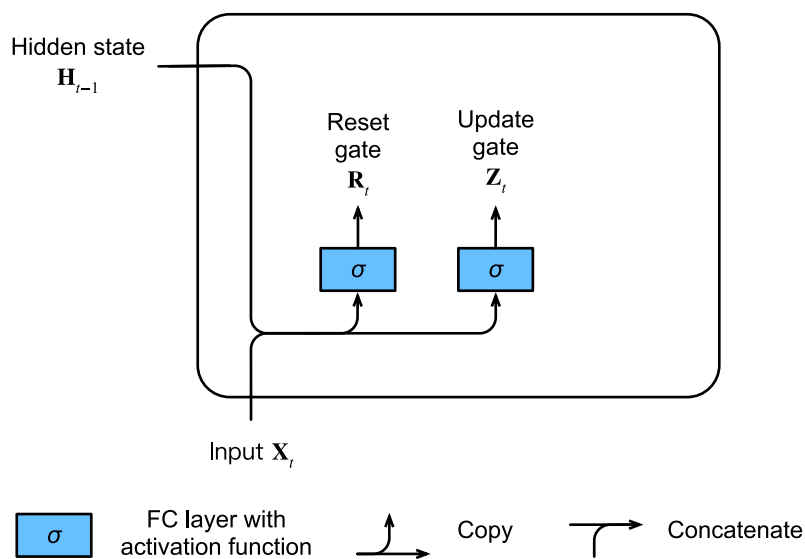
2.3GRU(Gated Recurrent Units)

随着 RNN，特别是 LSTM 架构在 2010 年代迅速流行，一些研究者开始尝试更简化的结构，希望保留“内部状态 + 乘法门控”的核心思想，但提高计算效率。门控循环单元（GRU）就是一种简化版本的 LSTM 记忆单元. 它在很多任务中表现接近 LSTM，但计算速度更快。

重置门与更新门:

在 GRU 中，LSTM 的三个门（输入、遗忘、输出门）被精简为两个：重置门（reset gate）和更新门（update gate）。它们同样使用 sigmoid 激活函数，输出范围在 $(0, 1)$ 。重置门控制是否保留先前隐藏状态的信息；更新门控制当前隐藏状态中保留多少旧状态、引入多少新状态。

如图所示，两层全连接层分别计算重置门和更新门：



$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$$

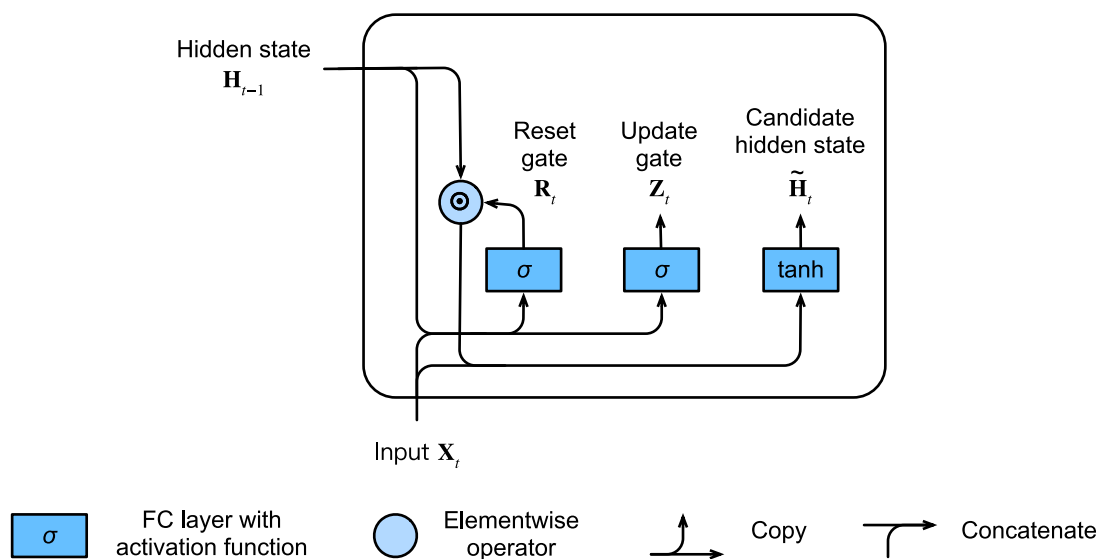
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$

候选隐藏状态：

将重置门应用于隐藏状态，结合当前输入，得到候选隐藏状态 \tilde{H}_t ：

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$

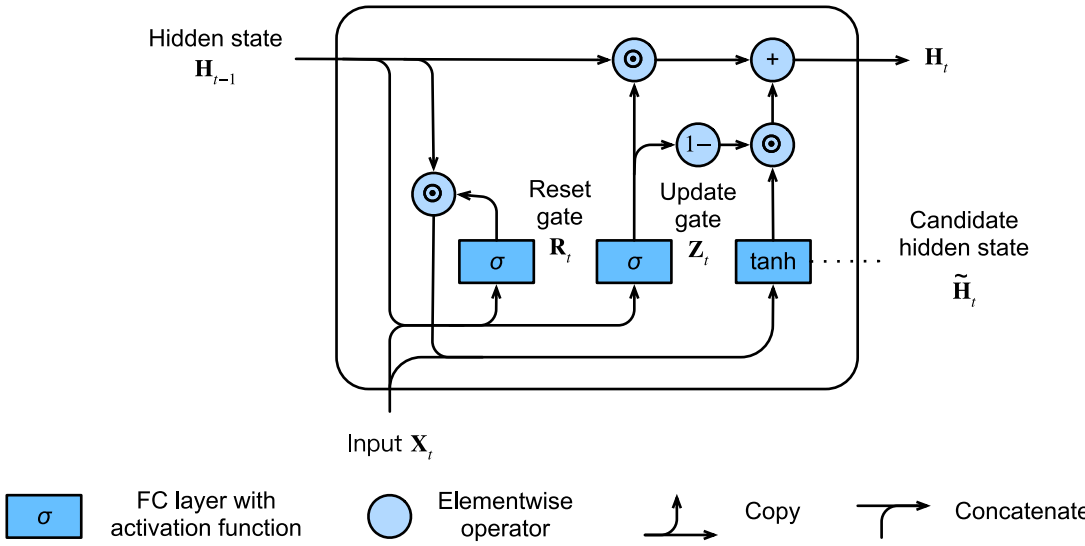
当 R_t 接近 1 时，相当于普通 RNN；当 R_t 接近 0 时，之前的隐藏状态被“重置”，仅使用当前输入生成状态。



隐藏状态：

最后通过更新门控制最终隐藏状态 H_t 是保留旧状态 H_{t-1} ，还是使用候选状态 \tilde{H}_t 。公式如下：

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



3 简单阐述诗歌生成过程

3.1 数据预处理

1. 加载唐诗文本文件。
2. 清洗无关字符与格式问题。
3. 添加特殊标记：每首诗前加 G（开始标记），结尾加 E（结束标记）。
4. 构建词表 `word_int_map`，将每个字映射为唯一的索引。
5. 将每首诗转为索引列表，构成 `poems_vector`。

3.2 词嵌入层

1. 构建词向量层：用 `nn.Embedding` 初始化每个字的嵌入向量。
2. 输入的字索引经过这一层，变为定长的稠密向量。

3.3 LSTM 模型构建

1. 利用 `nn.LSTM` 构建一个两层的循环神经网络，输入是词向量，输出是每个时刻的隐藏状态。
2. LSTM 输出后接一个全连接层（Linear）投影到词表空间。
3. 最后使用 `LogSoftmax` 输出每个字是下一个字的概率。

3.4 训练阶段

1. 构建输入 `x` 和目标 `y`，`x` 是某首诗的索引序列，`y` 是它的“右移一位”的版本（即预测下一个字）。
2. 每次从 `poems_vector` 中取出一个 `batch`（100 首诗），用模型预测，计算 `NLLLoss` 损失。
3. 使用 `RMSprop` 优化器更新参数，并保存模型。
4. 每隔一段时间会输出预测结果和真实值供可视化观察学习情况。

3.5 生成唐诗

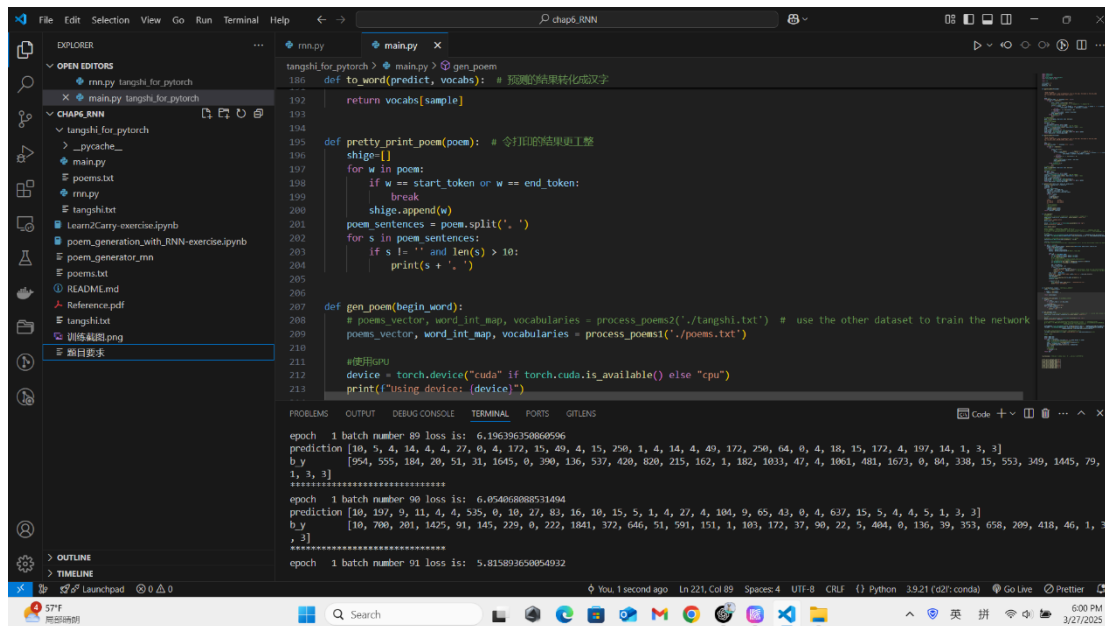
1. 加载训练好的模型。
2. 指定一个起始字，如“日”、“红”、“山”。
3. 将起始字转为索引，送入模型预测下一个字。
4. 不断将预测出的字添加到输入中，继续生成直到生成 `E` 或超过一定长度。

4 结果总结

生成结果与训练截图如下：

```
0, 230, 437, 605, 100, 20, 0, 729, 131, 307, 117, 00, 0, 333, 1, 3, -
*****
epoch 29 batch number 347 loss is: 6.62282657623291
error
Using device: cuda
inital linear weight
error
Using device: cuda
inital linear weight
红, 终日无人子, 何人不相逢。
error
Using device: cuda
inital linear weight
何人不得日, 不得不知行。
error
Using device: cuda
inital linear weight
error
Using device: cuda
inital linear weight
何事不相待, 风中不自知。
今来不得日, 不得一年光。
error
Using device: cuda
inital linear weight
error
Using device: cuda
inital linear weight
何人不相如, 不知无一事。
error
Using device: cuda
inital linear weight
柳然何年, 人中一度无之。

(d2l) C:\Users\asus\Desktop\DL\exercise-nndl\chap6_RNN>
```



```
File Edit Selection View Go Run Terminal Help
chap6.RNN
main.py
tangshi_for_pytorch > main.py > gen_poem
186 def to_word(predict, vocabs): # 预测的结果转化成文字
187     return vocabs[sample]
188
189
190
191
192
193
194
195 def pretty_print_poem(poem): # 令打印的结果更工整
196     shige=[]
197     for w in poem:
198         if w == start_token or w == end_token:
199             break
200         shige.append(w)
201     poem_sentences = poem.split(' ')
202     for s in poem_sentences:
203         if s != '' and len(s) > 10:
204             print(s + ' ')
205
206
207 def gen_poem(begin_word):
208     # poems_vector, word_int_map, vocabularies = process_poems2('./tangshi.txt') # use the other dataset to train the network
209     poems_vector, word_int_map, vocabularies = process_poems1('./poems.txt')
210
211     #调用函数
212     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
213     print("Using device: {}".format(device))
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS
epoch 1 batch number 89 loss is: 6.196396350860596
prediction [10, 5, 4, 14, 4, 4, 27, 0, 4, 172, 15, 49, 4, 15, 250, 1, 4, 14, 4, 49, 172, 250, 64, 0, 4, 18, 15, 172, 4, 197, 14, 1, 3, 3]
b.y [954, 555, 184, 20, 51, 31, 1645, 0, 390, 136, 537, 420, 820, 215, 162, 1, 182, 1033, 47, 4, 1061, 481, 1673, 0, 84, 338, 15, 553, 349, 1445, 79, 1, 3, 3]
epoch 1 batch number 90 loss is: 6.054068088531494
prediction [10, 197, 9, 11, 4, 4, 535, 0, 10, 27, 83, 16, 10, 15, 5, 1, 4, 27, 4, 104, 9, 65, 43, 0, 4, 637, 15, 5, 4, 4, 5, 1, 3, 3]
b.y [10, 700, 201, 1425, 91, 145, 229, 0, 222, 1841, 372, 646, 51, 591, 151, 1, 103, 172, 37, 90, 22, 5, 404, 0, 136, 39, 353, 658, 209, 418, 46, 1, 3, 3]
epoch 1 batch number 91 loss is: 5.815893650054932
```

一开始我的 loss 效果并不好，我猜测的原因可能是权重初始我原来采用的是随机初始化，后来我为了特定方式的初始，使得 loss 效果相对好了一些。

5 参考文献

[1] Xingxing Zhang and Mirella Lapata. 2014. Chinese poetry generation with recurrent neural networks. In Proceedings of the 2014 Conference on EMNLP. Association for Computational Linguistics, October

[2] Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into Deep Learning*. Cambridge University Press. Retrieved from <https://D2L.ai>