COMP9444 ASSIGNMENT1

Name:Ziqi Li

Student number:5567788

Part1:

1.1:

the total number of independent parameters in the network is 7850

```
  warnings.warn(f"A NumPy version >={np_minversion} ar
Total trainable parameters: 7850
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.234569
```

final accuracy and confusion matrix

final accuracy is 70%

rows of the confusion matrix indicate the target character, while the columns indicate the one chosen by the network.

```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.823461
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.622722
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.590868
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.599139
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.318217
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.517043
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.671404
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.607807
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.351480
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.674802
<class 'numpy.ndarray'>
[[764.   5.   8.  12.  31.  65.   2.  62.  32.  19.]
 [  7. 671. 108.  18.  29.  23.  57.  13.  26.  48.]
 [  7.  61. 691.  26.  26.  20.  46.  39.  45.  39.]
 [  5.  35.  59. 755.  15.  60.  13.  19.  29.  10.]
 [ 60.  54.  80.  21. 621.  20.  32.  37.  20.  55.]
 [  8.  28. 125.  17.  19. 724.  28.   7.  33.  11.]
 [  5.  23. 152.   9.  26.  23. 720.  20.   8.  14.]
 [ 18.  32.  29.  11.  83.  15.  53. 622.  89.  48.]
 [ 10.  38.  92.  40.   7.  31.  45.   8. 708.  21.]
 [  8.  51.  84.   3.  50.  32.  18.  32.  42. 680.]]

Test set: Average loss: 1.0101, Accuracy: 6956/10000 (70%)
```

1.2:

rows of the confusion matrix indicate the target character, while the columns indicate the one chosen by the network.the calculation of the total number of independent parameters in the network is 79510

```
Total trainable parameters: 79510
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.321460
```

final accuracy and confusion matrix

final accuracy is 84%

```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.410759
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.286147
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.248649
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.225227
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.109879
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.245577
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.247146
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.331964
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.120556
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.270619
<class 'numpy.ndarray'>
[[850.   4.   2.   7.  29.  35.   3.  38.  28.   4.]
 [  4. 811.  32.   4.  17.   8.  69.   5.  21.  29.]
 [  7.  11. 825.  44.  12.  17.  35.  12.  25.  12.]
 [  4.   7.  34. 906.   5.  15.   8.   5.   8.   8.]
 [ 39.  31.  18.   3. 800.   8.  34.  23.  19.  25.]
 [ 12.  14.  74.   8.   9. 830.  26.   3.  17.   7.]
 [  3.  15.  60.   8.  13.   7. 878.   6.   4.   6.]
 [ 16.  12.  23.   5.  28.  10.  36. 809.  28.  33.]
 [ 12.  33.  25.  45.   5.   8.  31.   8. 830.   3.]
 [  1.  23.  40.   5.  36.   7.  22.  16.  10. 840.]]

Test set: Average loss: 0.5356, Accuracy: 8379/10000 (84%)
```

1.3:

number of independent parameters:50378

```
Total trainable parameters: 50378
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.722100
```

final accuracy and confusion matrix

final accuracy is 93%

```
<class 'numpy.ndarray'>
[[974.   3.   3.   0.   8.   3.   0.   6.   0.   3.]
 [  5. 908.   9.   1.  16.   2.  41.   6.   3.   9.]
 [  9.   3. 872.  52.   9.  13.  25.  11.   1.   5.]
 [  1.   2.  27. 961.   2.   2.   2.   1.   1.   1.]
 [ 31.   2.   7.  17. 912.   4.   7.   5.  12.   3.]
 [  2.  12.  49.   8.   5. 903.  12.   5.   1.   3.]
 [  3.   3.  14.   3.   7.   6. 954.   6.   2.   2.]
 [  8.   3.   1.   2.  12.   1.   6. 948.   5.  14.]
 [  9.   9.  10.   5.   8.   3.   3.   6. 947.   0.]
 [ 10.   5.   6.   4.   9.   4.   1.   6.  11. 944.]]

Test set: Average loss: 0.2521, Accuracy: 9323/10000 (93%)
```

1.4:

a : the relative accuracy of the three models:70%,84%,93%

b :the relative accuracy of the three models:7850,79510,50378

c :After comparing the confusion matrix for each model,

The first model NetLin:

the character 2 are most likely to be mistaken for 1,5,6 characters,

The difference in the characters and position of strokes is too similar,and These confusions usually occur because the model relies on pixel or feature similarity, and subtle differences may not be captured well by simple linear or shallow models.

Second model NetFull:

Still the character 2 are most likely to be mistaken for 5,6 characters

the character 6 are most likely to be mistaken for 1character,

The receptive fields of NetLin and NetFull are equivalent to the flat features of the entire image, lacking spatial local contrast.Even if the two layers are fully connected, it is still difficult to capture spatial information such as "this stroke is in the upper left vs. lower right"

third model Netconv:

the character 2 are most likely to be mistaken for 3,5 character,

the character 6 are most likely to be mistaken for 1 character,

Because they both have similar stroke patterns and overlapping components or indistinct boundaries in the handwritten samples

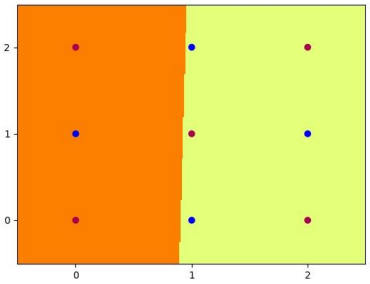
Part2:

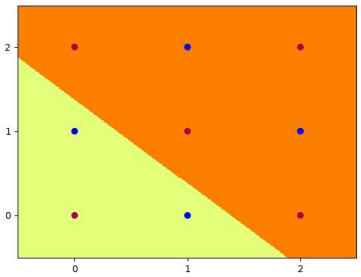
2.1:

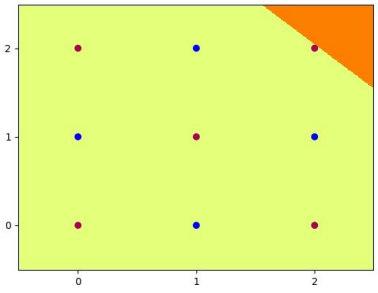
```
Final Weights:
tensor([[-4.0671,  0.0840],
        [ 5.5522,  5.5673],
        [ 2.7584,  2.7584],
        [ 0.0848, -4.0735],
        [-2.7730, -2.7725]])
tensor([  3.6817,  -7.6686, -11.1637,   3.6846,  11.2072])
tensor([[ -9.5618,  -8.6312, -10.3718,  -9.5680,  10.9381]])
tensor([4.8711])
Final Accuracy:  100.0
```

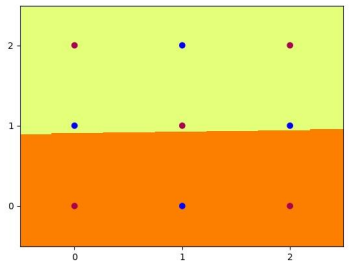
Accuracy:100%

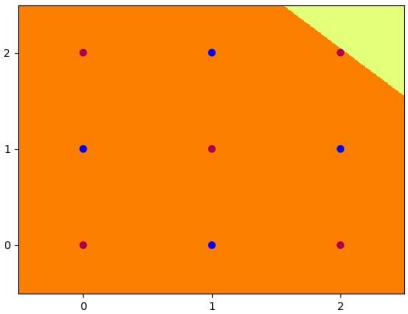
Here is the hidden node (hid_5_0.jpg to hid_5_4.jpg) and the last one is the network as a whole (out_5.jpg)

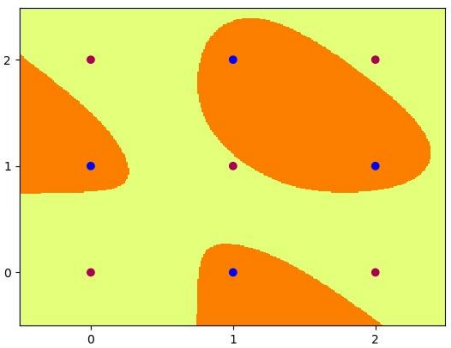








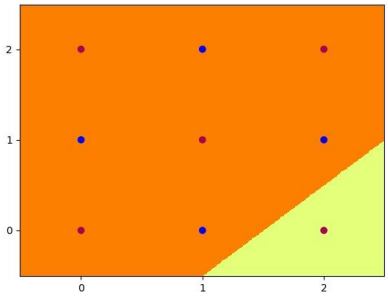


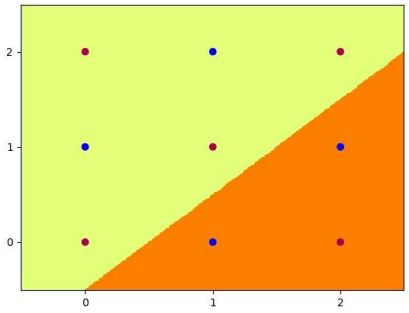


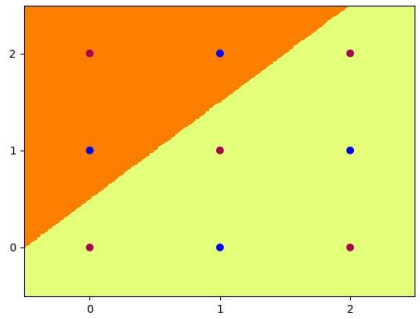
2.2:
all the weights and biases:

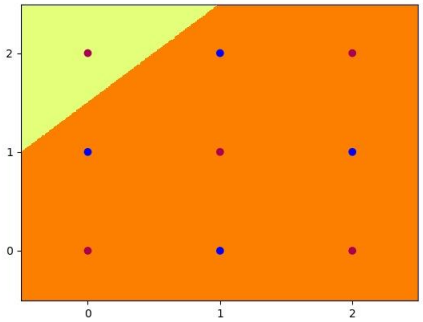
```
def set_weights(self):
    in_hid_weight  = [
        [1.0, -1.0],
        [-1.0, 1.0],
        [1.0, -1.0],
        [-1.0, 1.0],
    ]
    hid_bias = [1.5, -0.5, -0.5, 1.5]
    hid_out_weight = [ [1.0, 1.0, 1.0, 1.0] ]
    out_bias = [-2.5]
```

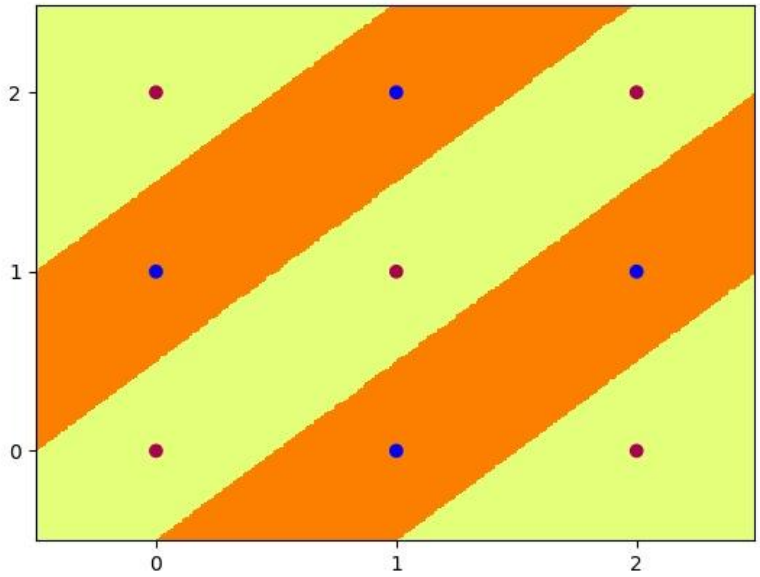
Here is the hidden node (hid_4_0.jpg to hid_4_3.jpg) and the last one is the network as a whole (out_4.jpg)

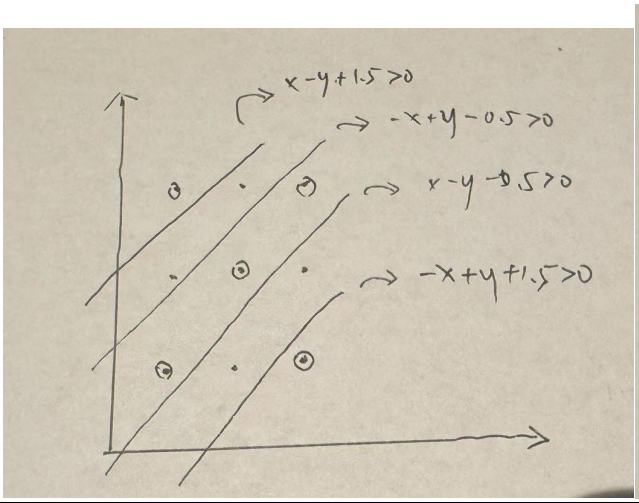


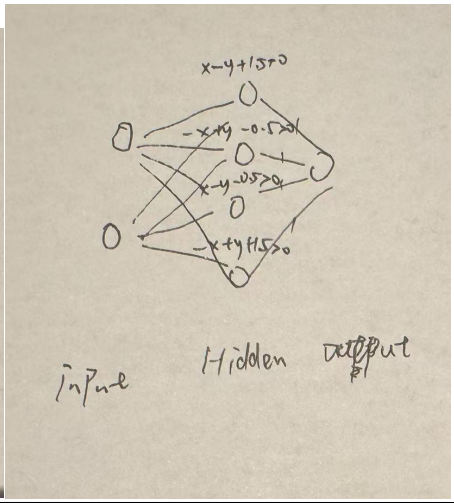












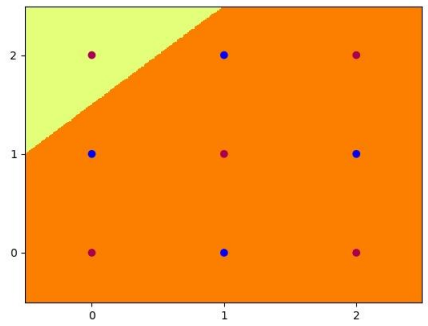
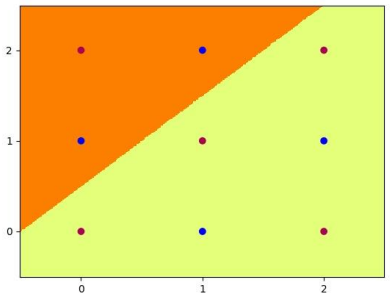
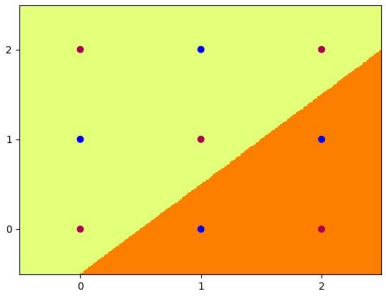
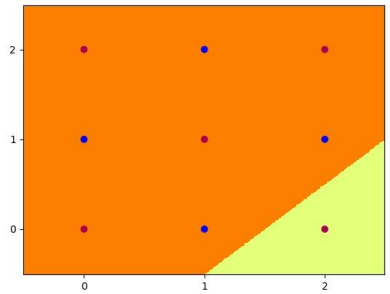
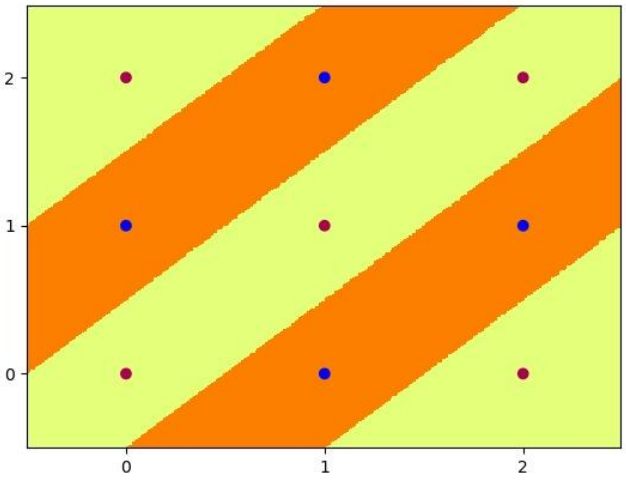
Input X	Input X	H1	H2	H3	H4	output
0	0	1	0	0	1	0
0	1	1	1	0	1	1
0	2	0	1	0	1	0
1	0	1	0	1	1	1
1	1	1	0	0	1	0
1	2	1	1	0	1	1
2	0	1	0	1	0	0
2	1	1	0	1	1	1
2	2	1	0	0	1	0

if h1+h2+h3+h4+bias(-2.5)>=0 then output= 1 else 0

2.3:

```
Initial Weights:
tensor([[ 10., -10.],
        [-10.,  10.],
        [ 10., -10.],
        [-10.,  10.]])
tensor([15., -5., -5., 15.])
tensor([[10., 10., 10., 10.]])
tensor([-25.])
Initial Accuracy:  100.0
```

Here is the hidden node (hid_4_0.jpg to hid_4_3.jpg) and the last one is the network as a whole (out_4.jpg)

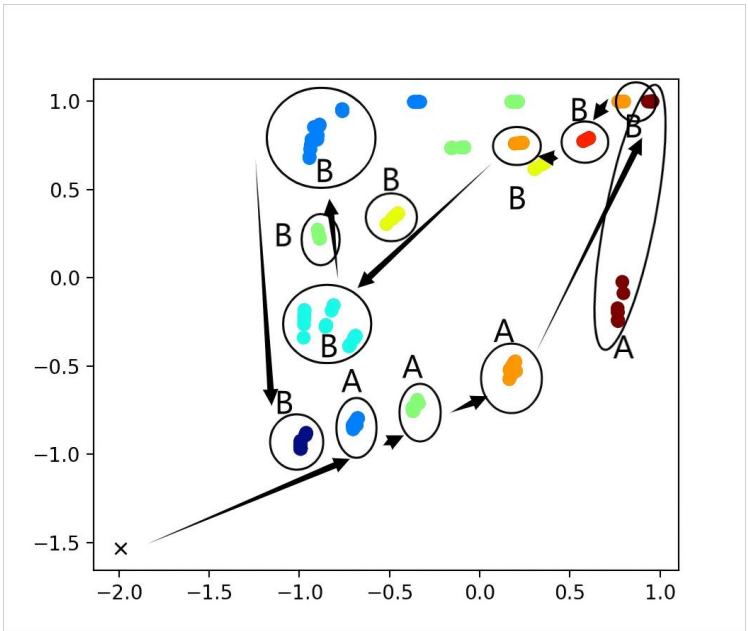






Part3:

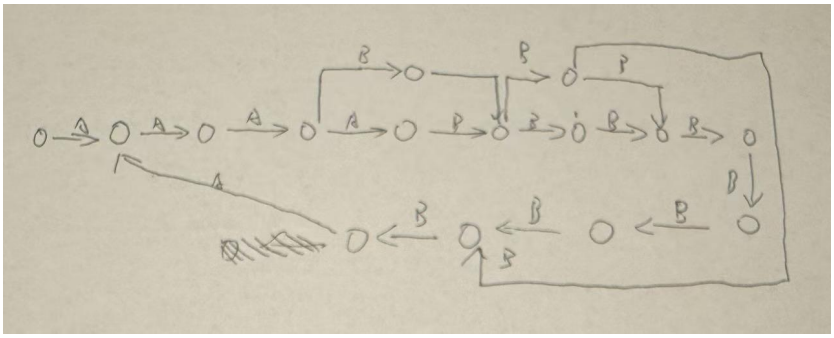
3.1:

```
  B [-0.97 -0.89] [ 0.86  0.14]
  epoch: 100000
   loss: 0.0288
```

```
A [-0.68 -0.83] [0.82 0.18]
A [-0.34 -0.71] [0.68 0.32]
A [ 0.2  -0.53] [0.42 0.58]
B [0.8 1. ] [0. 1.]
B [0.36 0.65] [0. 1.]
B [-0.09  0.74] [0. 1.]
B [-0.81 -0.16] [0.04 0.96]
B [-0.9   0.78] [0. 1.]
B [-0.9   0.78] [0. 1.]
B [-0.99 -0.95] [0.9 0.1]
```



3.2:



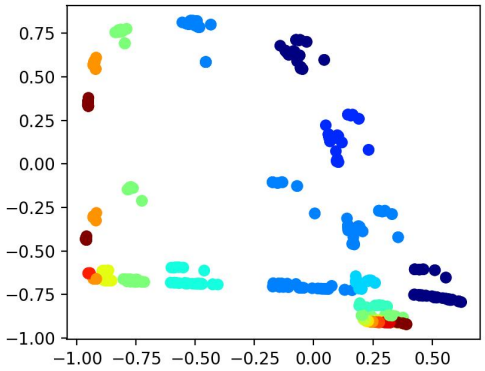
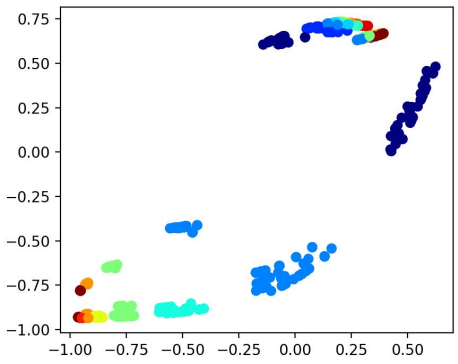
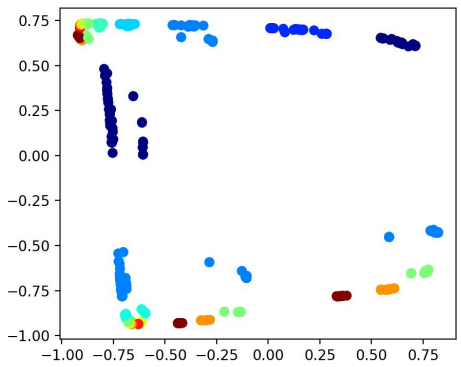
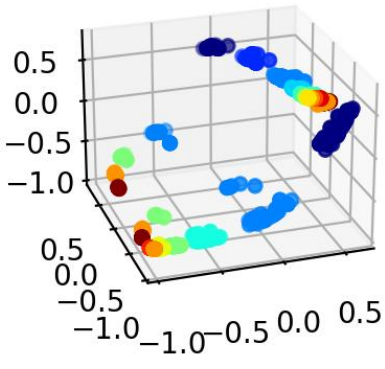
3.3:

Each cluster of points corresponds to a distinct state, such as how many currently reading A or is it in the B phase.from the terminal.As each A is read, the network hidden state moves from one cluster to the next, mimicking state transitions for counting A's.also,from first three lines we can tell that when first B haven't came yet,the predicted value of A is decreasing,which means B is probably already coming,Upon encountering the first B, the trajectory jumps to a new region, corresponding to entering the B phase, and continues to transition within the B-state clusters

3.4:

loss stay consistently below 0.02

```
C [ 0.37 -0.56  0.74] [ 0.  0.  1.]
C [ 0.25  0.    0.73] [ 0.04  0.    0.95]
C [-0.08  0.64  0.7 ] [ 0.94  0.    0.05]
epoch: 100000
 loss: 0.0104
```

3.5:

LSTM is able to solve the anb2nc3n prediction task by maintaining a structured internal trajectory through its hidden state space. Each segment of the trajectory encodes the current phase A, B, or C and the count of symbols seen, allowing the network to output the correct number of each symbol in sequence.

Compare with the SRN,SRN's core mechanism is a recursive function when the sentence is really long,its easy to get exploding and gradient vanishing,which means cant remember how

many a in front.

But LSTM designed a memory cell and gates,for example forget gate will decide how many old information will be save,input gate can decide how many new date will be write in,which is more suitable and efficient for structured language tasks like anb2nc3n.