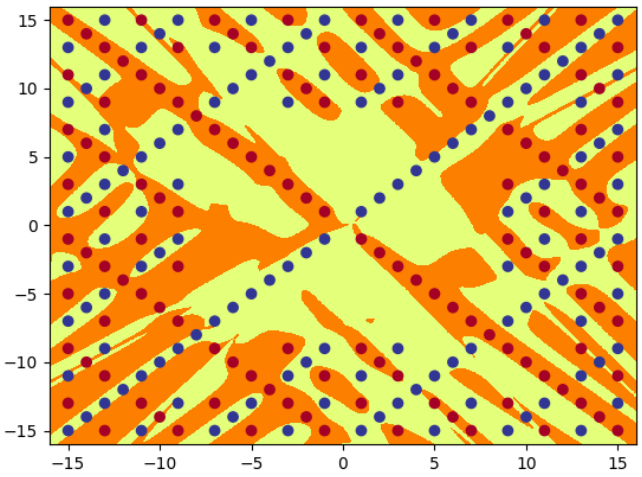
COMP9444 ASSIGNMENT 1

Question 1:

(2)

The minimum number of hidden nodes required for the network to be trained is 20.

```
ep:175900 loss: 0.0033 acc: 100.00
ep:176000 loss: 0.0032 acc: 100.00
ep:176100 loss: 0.0032 acc: 100.00
ep:176200 loss: 0.0031 acc: 100.00
ep:176300 loss: 0.0030 acc: 100.00
ep:176400 loss: 0.0030 acc: 100.00
ep:176500 loss: 0.0029 acc: 100.00
ep:176600 loss: 0.0029 acc: 100.00
ep:176700 loss: 0.0028 acc: 100.00
ep:176800 loss: 0.0028 acc: 100.00
ep:176900 loss: 0.0027 acc: 100.00
```

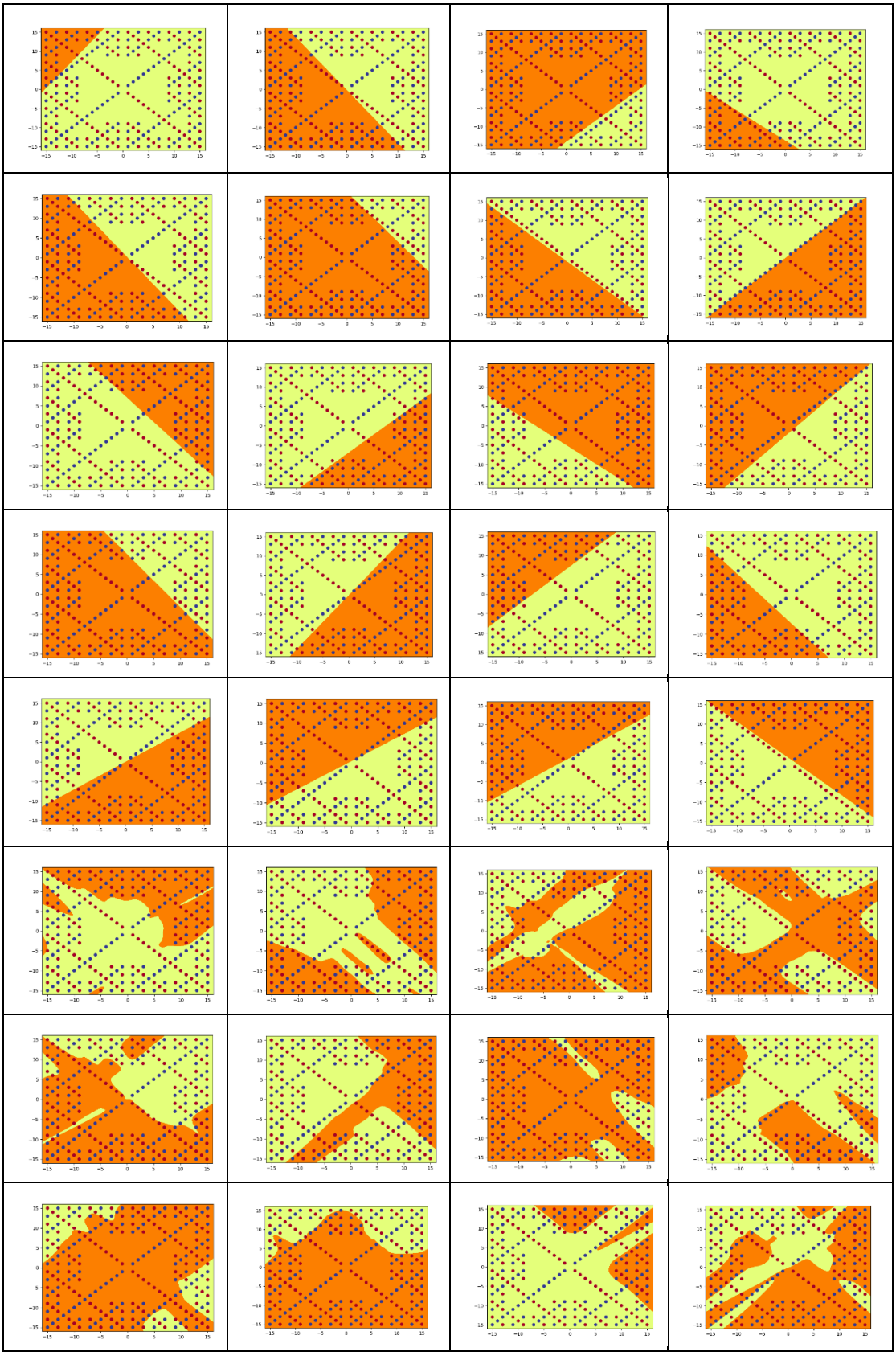


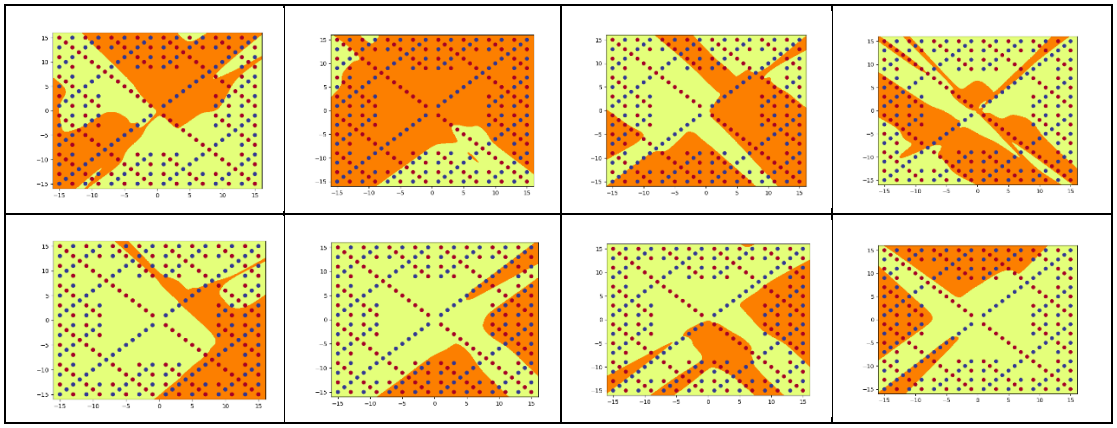
Independent parameters between input layer and first hidden layer: $2\times20 + 20 = 60$.

Independent parameters between first and second hidden layers: $20\times20 + 20 = 420$.

Independent parameters between second hidden layer and output layer: $20\times1 + 1 = 21$.

Total number of independent parameters of the neural net: $60 + 420 + 21 = 501$.

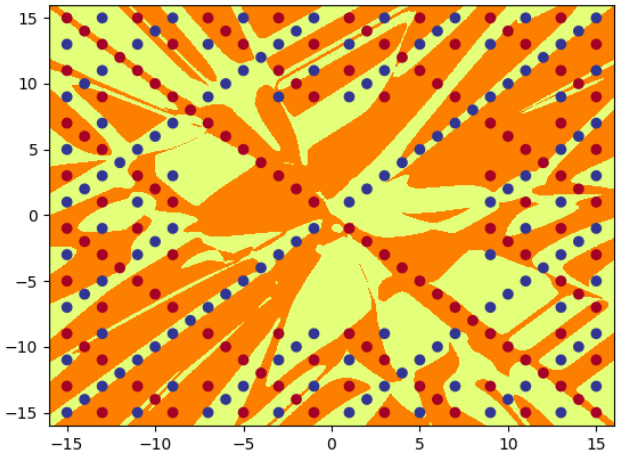




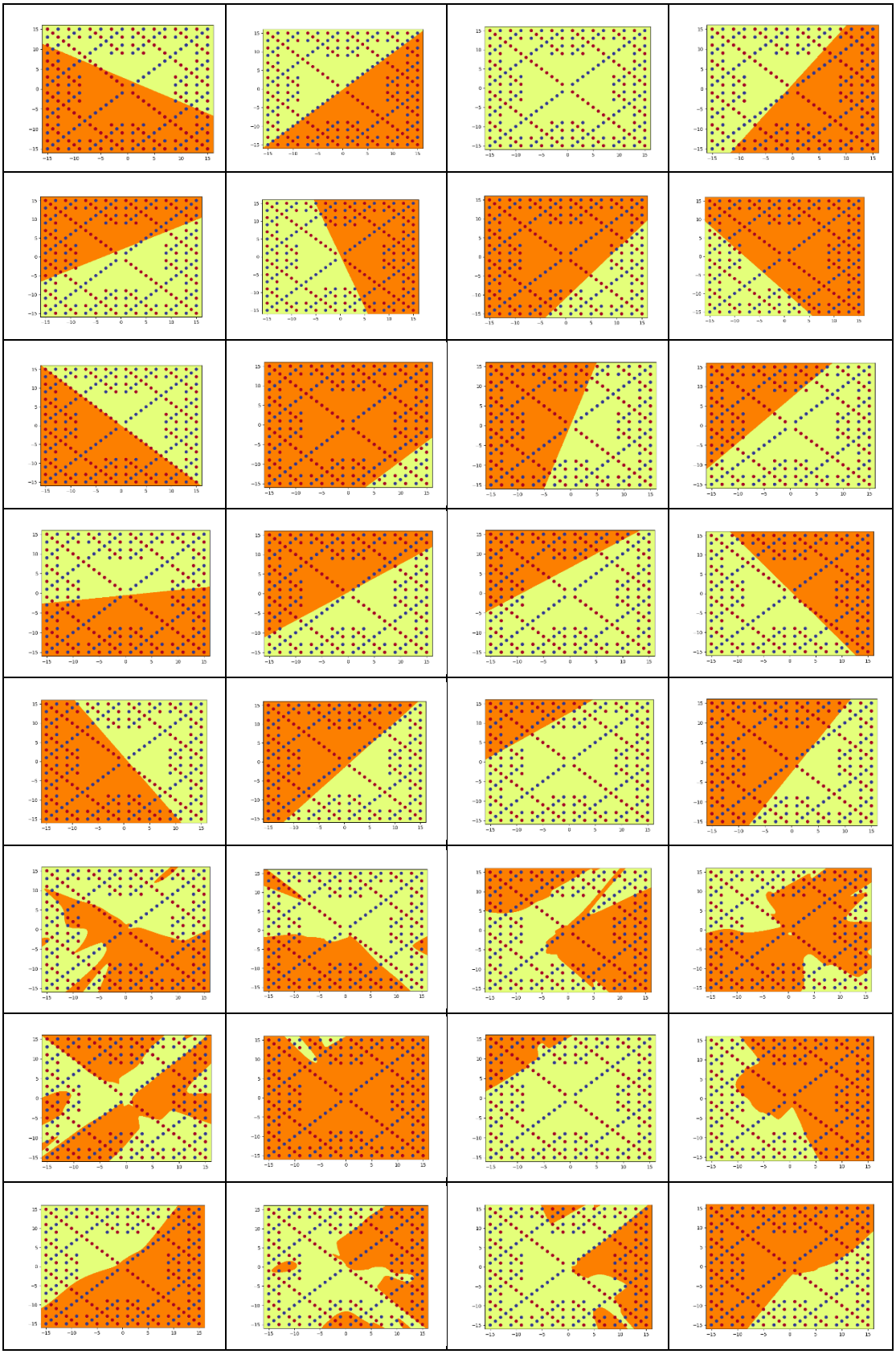
(4)

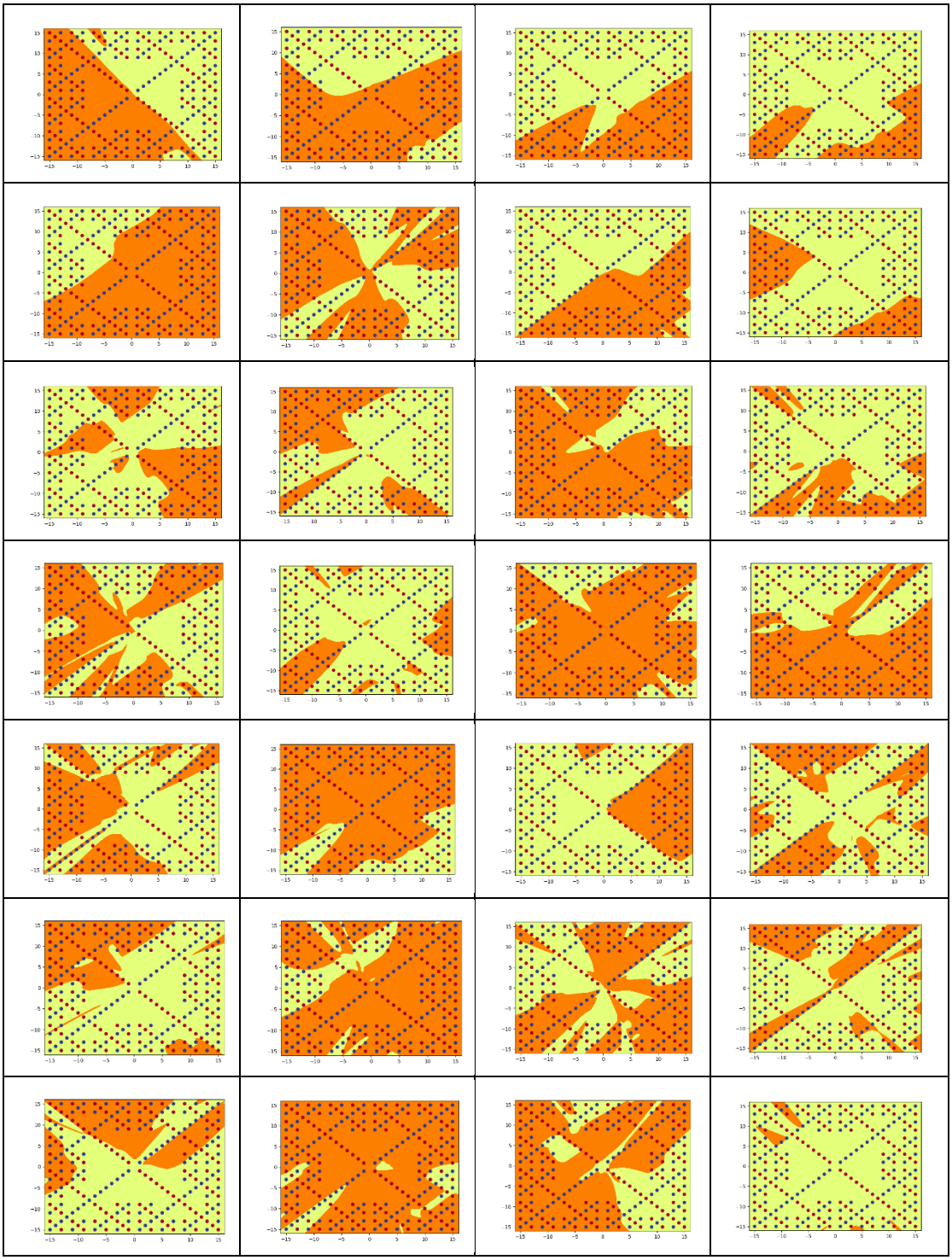
The minimum number of hidden nodes required for the network to be trained is 20.

```
ep:138300 loss: 0.0014 acc: 100.00
ep:138400 loss: 0.0013 acc: 100.00
ep:138500 loss: 0.0013 acc: 100.00
ep:138600 loss: 0.0013 acc: 100.00
ep:138700 loss: 0.0012 acc: 100.00
ep:138800 loss: 0.0012 acc: 100.00
ep:138900 loss: 0.0011 acc: 100.00
ep:139000 loss: 0.0011 acc: 100.00
```



Independent parameters between input layer and first hidden layer: 2×20 + 20 = 60.
Independent parameters between first and second hidden layers: 20×20 + 20 = 420.
Independent parameters between second hidden layer and output layer: 20×20 + 20 = 420
Independent parameters between third hidden layer and output layer: 20 * 1 + 1 = 21
Total number of independent parameters of the neural net: 60 + 420 +420+ 21 = 921.

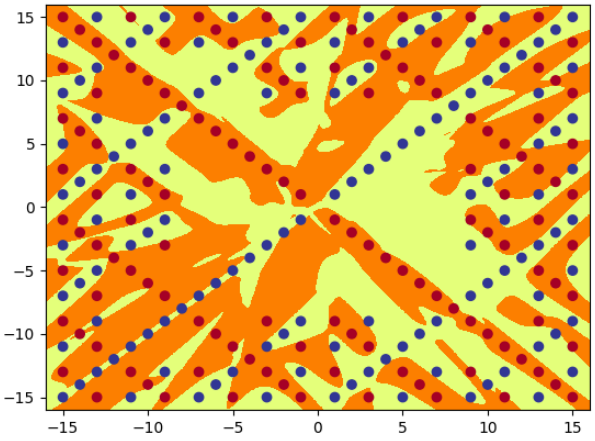




(6)

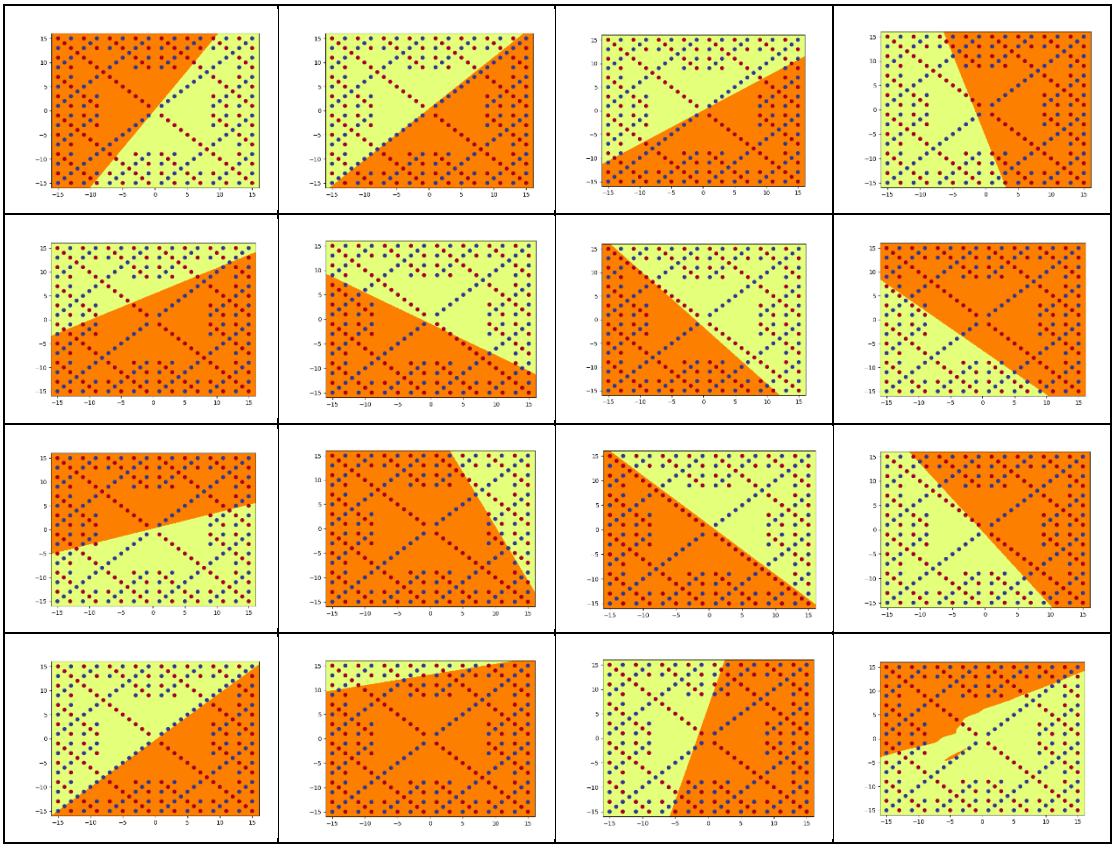
The minimum number of hidden nodes required for the network to be trained is 15.

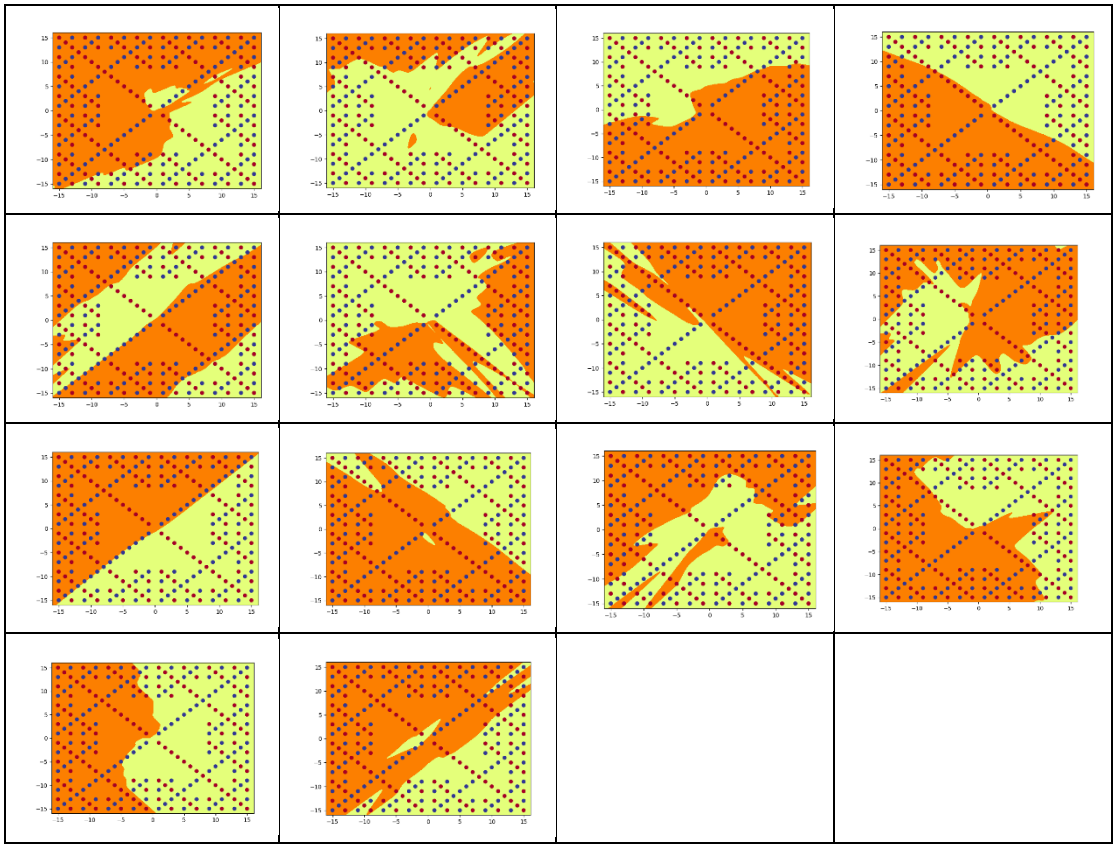
```
ep:92100 loss: 0.0156 acc: 100.00
ep:92200 loss: 0.0155 acc: 100.00
ep:92300 loss: 0.0154 acc: 100.00
ep:92400 loss: 0.0153 acc: 100.00
ep:92500 loss: 0.0153 acc: 100.00
ep:92600 loss: 0.0152 acc: 100.00
ep:92700 loss: 0.0151 acc: 100.00
ep:92800 loss: 0.0150 acc: 100.00
```



Independent parameters:

(2 * hidden_num + hidden_num) + (2 * hidden_num + hidden_num) + (hidden_num * hidden_num) + (hidden_num * 1 + hidden_num) + (hidden_num * 1) + (hidden_num * 1) = 45 + 45 + 225 + 30 + 15 + 15 = 375





(7)

(a)

Name	Hidden nodes	Independent parameters	Epoch
Full3Net	20	501	176900
Full4Net	20	921	139000
DenseNet	15	375	92800

(b)

Analysis for model:

For Full4Net, each layer considers the output from the previous layer as current layer input.

In contrast, each layer in the DenseNet is directly connected to its previous layer which means that the input of each layer comes from the output of all previous layers. By exploiting DenseNet, we enhanced feature transfer, more efficient use of features, and reduced the number of parameters needed for the network. Apart from the above, Full4Net has one more hidden layer than DenseNet.

Analysis for performance:

It can also be seen from the data in our table that the number of epochs required to train DenseNet is lower than that of full4net, which also means DenseNet is more efficient than Full4Net. Then by observing the plots generated by Full4Net and DenseNet, we find out that DenseNet achieves better performance than Full4Net for edge case, eg: the bule points in the middle of red points.

Analysis for each layer (Plot):

Full4Net:

(1) For the first hidden layer, we get one straight line dividing the graph into two parts.
(2) From the second layer, the image starts to have continuous orange curves and sharp lines.
(3) Then when we get the third layer, the orange curves start to become scattered in the image and more sharp lines to capture some nodes in the graph.

DenseNet:

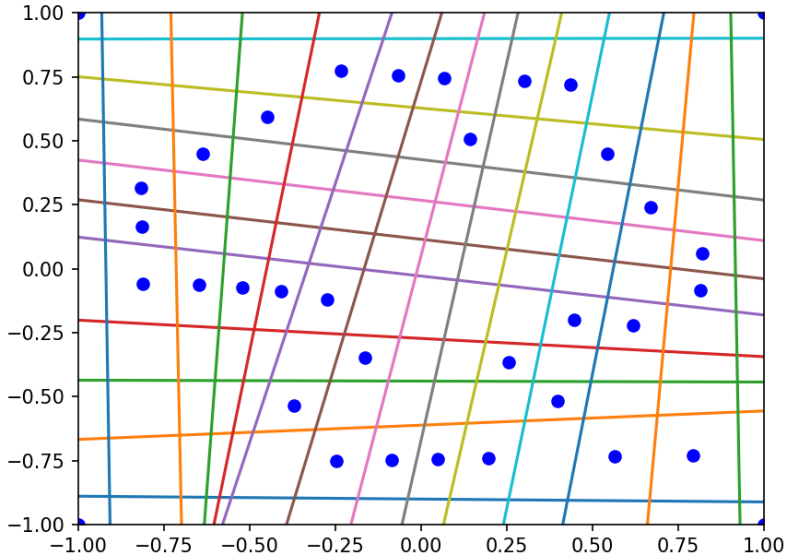
(1) For the first hidden layer, we get one straight line dividing the graph into two parts
(2) From the second layer, the general trend is that the orange color is more concentrated, and the edge is in the shape of a curve. With the process of training, there are a few sharp lines outside the orange region to encounter the other nodes. But most of the orange areas are still clustered together

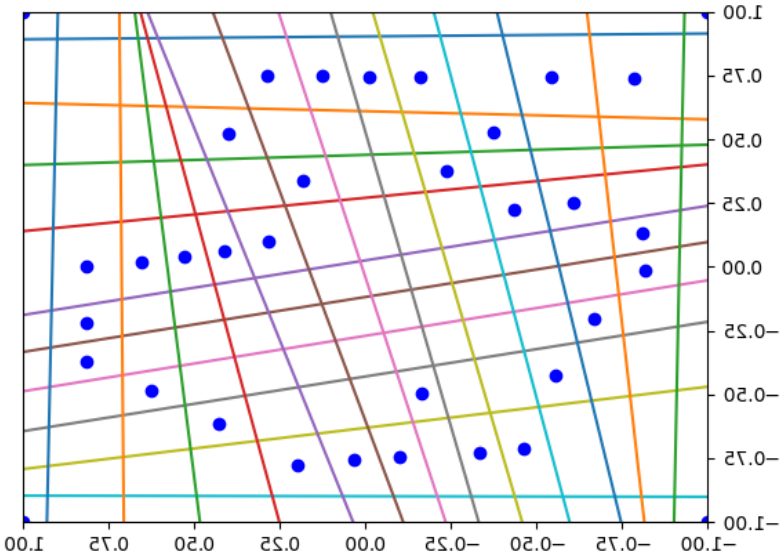
(c)

From all output images of these networks, we can see all the images have classified the red dots and blue dots successfully.

The only difference between these three networks is that the number of epochs needed. Based on the observation table above, we can see DenseNet use the smallest number of epochs to train the network successfully. This is because the input of current layer is not only related to the output of previous layer, but also the outputs of all previous layers. By taking advantage of this method, we will have a smaller number of parameters to train, encourage feature reuse and enhance feature transfer, which will make the speed of training faster. And for the difference between Full3Net and Full4Net, it is obvious to see that Full4Net has one more hidden layer than Full3Net. Although it is true that Full4Net is better in terms of the number of epochs, it is not ideal in terms of the number of parameters. So, this is what we should learn. For real life problems, a more appropriate model is far more profitable than a complex model.

Question 2:

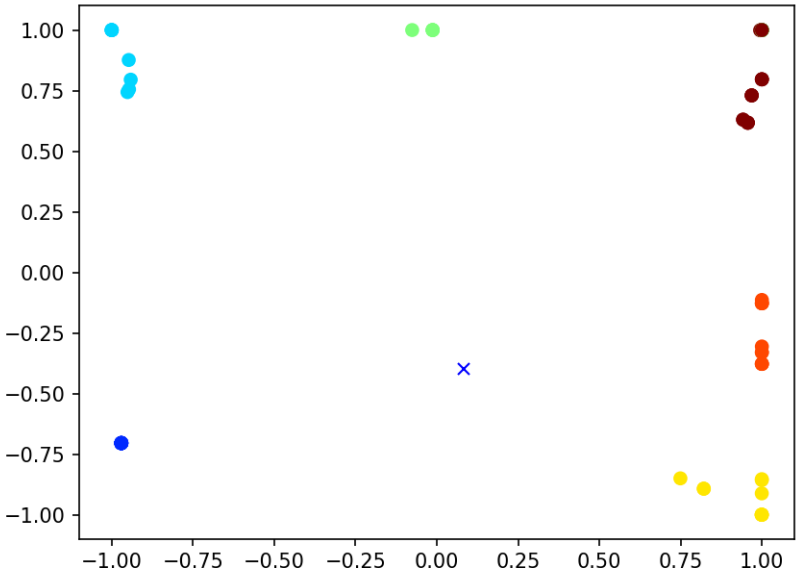


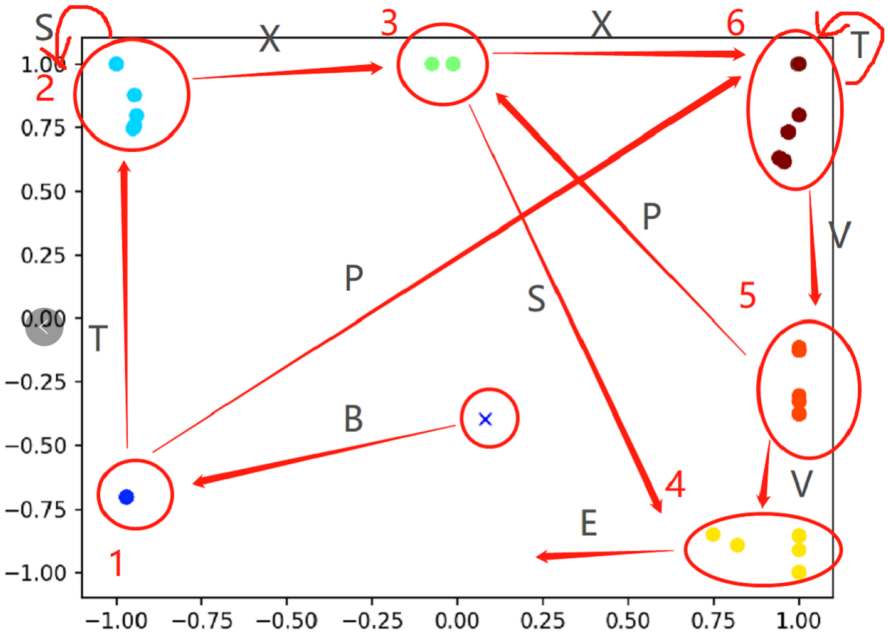


When we reflect the original picture, we can get the correct image the same as the one given in the specification.

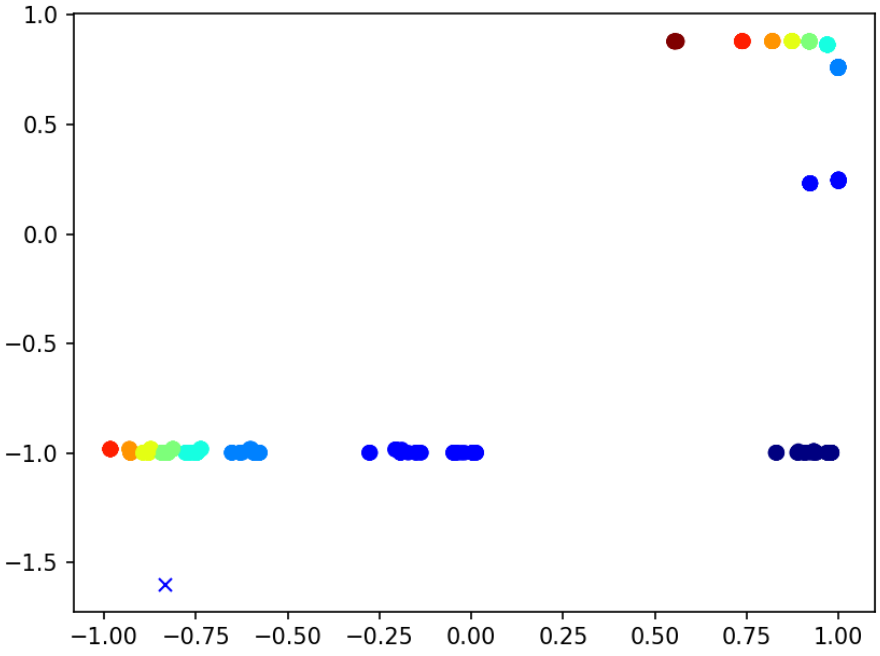
Question 3:

(1)

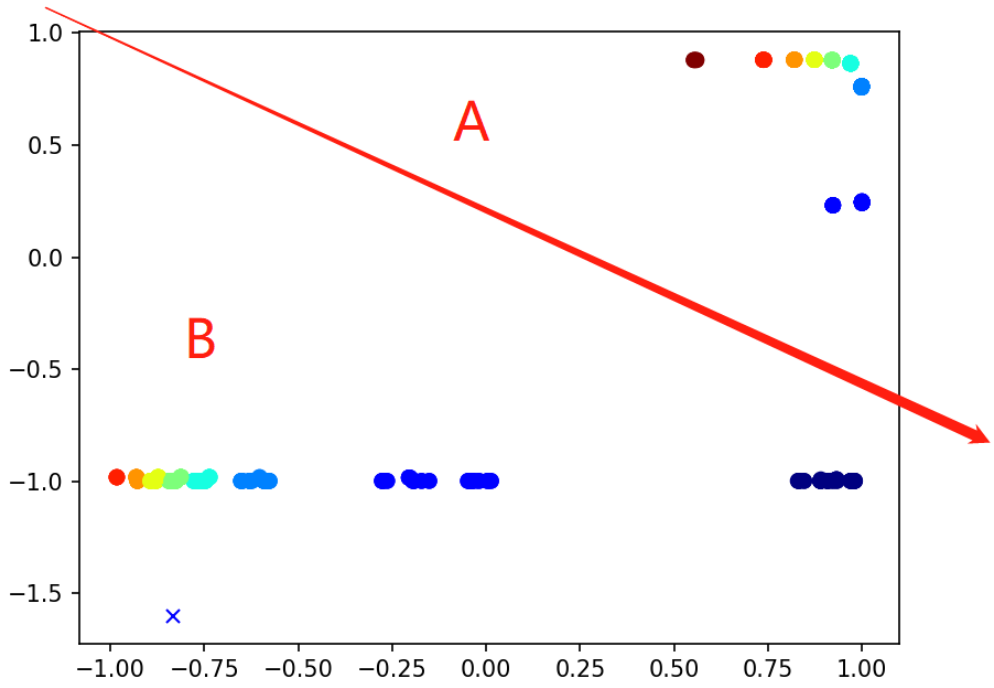

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


S
X
3
X
6
T
2
P
V
P
S
5
T
B
4
V
E
1

(2)



(3)



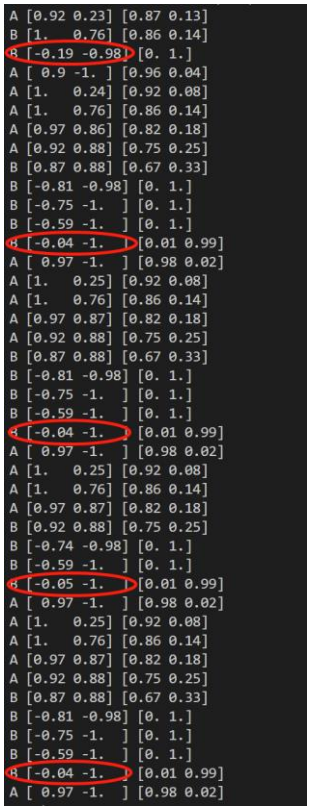
When we observe the locations of the hidden activations, we find out most of the A are placed at the up-right corner and all B are placed horizontally at the bottom of picture. For Section A, we will have some B placed there. This is because the first B in each sequence is not deterministic, which means that the appearance of B is probabilistic. But after first B appearing, we can predict the

following Bs. When the number of B is matched with that of A, we will encounter A, which forces us get into A section again. And the above steps are repeated until the end of the sequence.
In addition, we can also find out in Section A, there are some random dots distributed. This is because all A's after the first A are not deterministic.

Q: How the hidden unit activations change as the string is processed?

(1) When string starts being processed, we will get first character. Now the hidden unit activation is placed at the up-right corner of the picture.
(2) As the program progresses, we have random probability to get B. If we encounter b, the hidden unit activation will be changed to the bottom of picture.
(3) To match the number of a, we will get same number of b, making the hidden unit activation move horizontally. When we have the same number of a and b, the hidden unit activation will go back to Section A.
(4) Then we will repeat the process (2) – (3) until the end of the sequence.

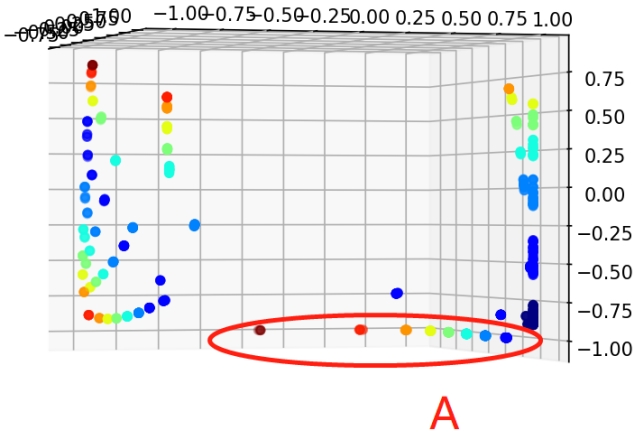
Q: How it is able to correctly predict the last B in each sequence as well as the following A?

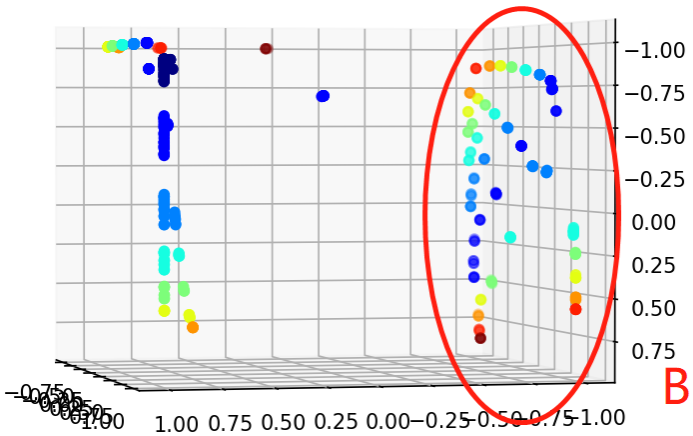


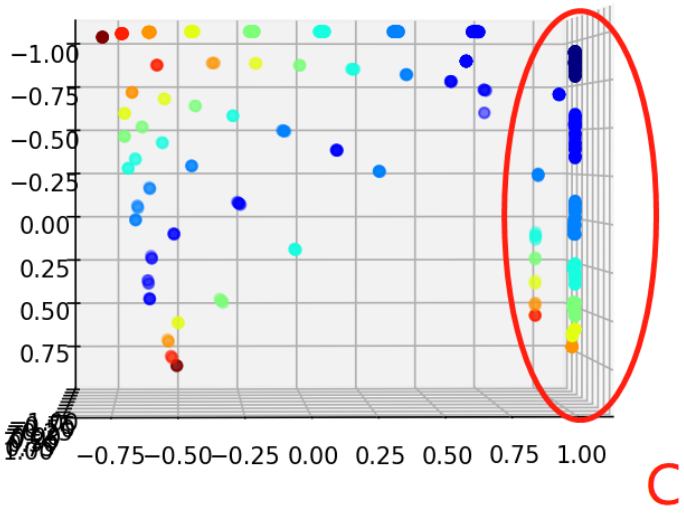
When we consider the hidden activations, we find out that when the hidden activations are close to [-0.04, -1], the next character will be A.

(4)

```
B [ 0.64 -0.99 -0.78] [0.   0.91 0.09]
C [ 0.34 -1.   -0.26] [0.   0.05 0.95]
C [ 1.    0.24 -0.69] [0.   0.01 0.99]
A [ 1.   1.  -0.9] [0.93 0.07 0.  ]
B [ 0.62  1.   -1.  ] [0.9 0.1 0. ]
C [ 0.78 -0.95 -0.73] [0.   0.08 0.92]
A [ 1.    0.97 -0.89] [0.92 0.07 0.  ]
epoch: 9
error: 0.0123
```







(5)

```
B [ 0.59  0.98 -0.85] [0.9 0.1 0. ]
C [ 0.78 -0.97 -0.6 ] [0. 0. 1.]
A [ 1.    0.96 -0.84] [0.91 0.07 0.02]
A [ 0.62  1.   -1.  ] [0.9 0.1 0. ]
A [ 0.31  1.   -1.  ] [0.87 0.13 0.  ]
A [ 0.04  1.   -1.  ] [0.83 0.17 0.  ]
A [-0.23  0.99 -1.  ] [0.79 0.21 0.  ]
A [-0.45  0.97 -1.  ] [0.73 0.27 0.  ]
A [-0.61  0.88 -1.  ] [0.61 0.39 0.  ]
A [-0.72  0.64 -1.  ] [0.35 0.65 0.  ]
B [-0.8   0.06 -1.  ] [0.05 0.95 0.  ]
B [-0.59 -1.   -0.87] [0. 1. 0.]
B [-0.69 -1.   -0.72] [0. 1. 0.]
B [-0.72 -1.   -0.6 ] [0. 1. 0.]
B [-0.72 -1.   -0.47] [0. 1. 0.]
B [-0.71 -1.   -0.28] [0. 1. 0.]
B [-0.68 -1.    0.01] [0. 1. 0.]
B [-0.62 -1.    0.46] [0.   0.97 0.03]
C [-0.51 -1.    0.85] [0. 0. 1.]
C [ 1.   -1.    0.56] [0. 0. 1.]
C [1.   0.87 0.64] [0. 0. 1.]
C [1.   1.   0.54] [0. 0. 1.]
C [1.   1.   0.41] [0. 0. 1.]
C [1.   1.   0.2] [0. 0. 1.]
C [ 1.   1.   -0.12] [0. 0. 1.]
C [ 1.   1.   -0.57] [0.08 0.   0.92]
A [ 1.   1.   -0.88] [0.93 0.07 0.01]
A [ 0.62  1.   -1.  ] [0.9 0.1 0. ]
A [ 0.32  1.   -1.  ] [0.87 0.13 0.  ]
A [ 0.04  1.   -1.  ] [0.83 0.17 0.  ]
A [-0.23  0.99 -1.  ] [0.79 0.21 0.  ]
B [-0.45  0.97 -1.  ] [0.73 0.27 0.  ]
B [ 0.01 -1.   -0.87] [0. 1. 0.]
B [-0.27 -1.   -0.58] [0. 1. 0.]
B [-0.44 -1.   -0.3 ] [0. 1. 0.]
B [-0.52 -1.    0.1 ] [0. 1. 0.]
C [-0.5  -1.    0.61] [0.   0.04 0.96]
C [ 1.   -1.    0.24] [0. 0. 1.]
C [1.   0.93 0.22] [0. 0. 1.]
C [ 1.   1.   -0.08] [0. 0. 1.]
C [ 1.   1.   -0.52] [0.02 0.   0.97]
A [ 1.   1.   -0.86] [0.93 0.06 0.01]
A [ 0.62  1.   -1.  ] [0.9 0.1 0. ]
```

By observing data above, we can find out that for A, the value of z doesn't change too much, for B, the value of y doesn't change too much, and for C, the value of x doesn't too much.

Therefore, we can classify different sections of A, B, and C.

For $a^n b^n c^n$ prediction task, the probabilities of appearing A and B are not deterministic.

The reason why B is not deterministic is we cannot know when first B will appear in sequence.

However, the probability of C can be set after appearance of A and B, which means that C is deterministic.

Summary: The probability can be deterministic including

(1) The first character of A

(2) All Bs exclude the first B in the sequence

(3) All Cs in the sequence.

Q: How the hidden unit activations change as the string is processed?

(1) When string starts being processed, we will get first character. Now the hidden unit activation is placed at its corresponding section.

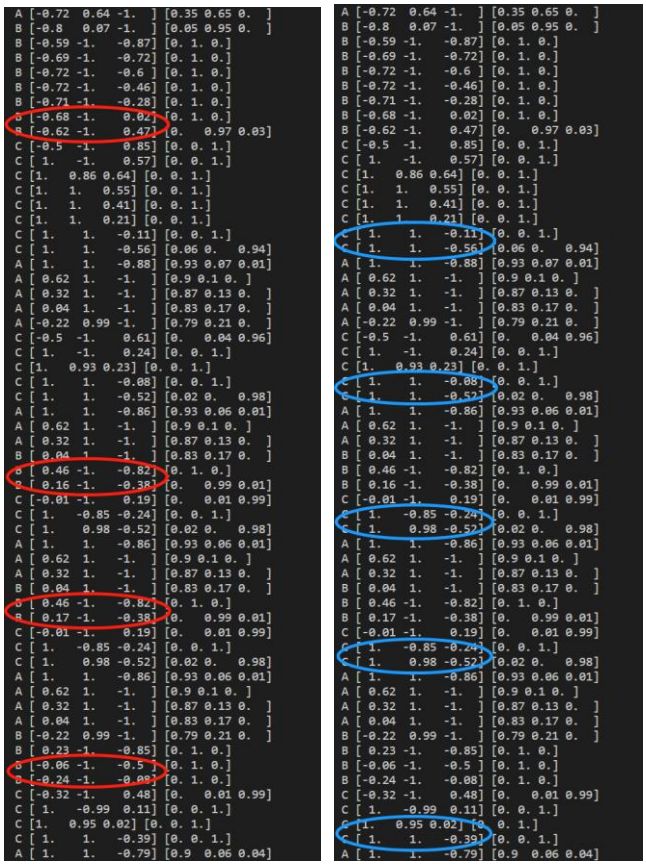
(2) As the program progresses, we have random probability to get B. If we encounter B, the hidden unit activation will be changed to the section B.

(3) To match the number of A, we will get same number of B, making the hidden unit activation move vertically. When we have the same number of A and B, the hidden unit activation will go to Section C.

(4) To match the number of A and B, we will get same number of C, making the hidden unit activation move horizontally. When we have the same number of A, B, and C, the hidden unit activation will go to Section A.

(5) Then we will repeat the process (2) – (4) until the end of the sequence.

Q: How it is able to correctly predict the last B in each sequence as well as all of the C's and the following A?



As we can see in the hidden activations table (left side), when the variation in the z-axis of B is

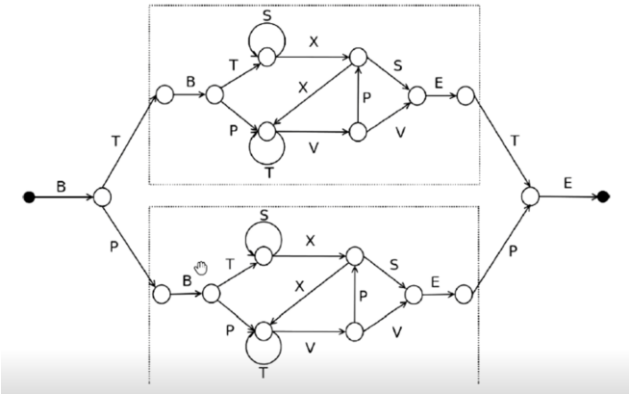
approximately close to 0.45, the next character will be C, which indicates the last B in each sequence. But for B as the first character of the sequence, we cannot apply above pattern since the first B in the sequence cannot be deterministic.

From the analysis above, we have already known that C is deterministic in each sequence. Therefore, we can predict all of C in the sequence.

As we can see in the hidden activations table (right side), when the values in the z-axis of C is smaller than -0.50 and the value of y-axis is close to 1, the next character will be A, which indicates the last C in each sequence.

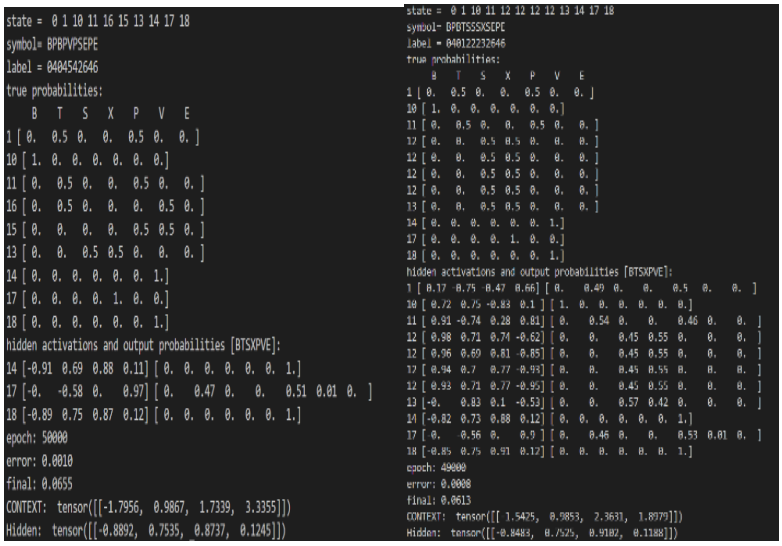
(6)

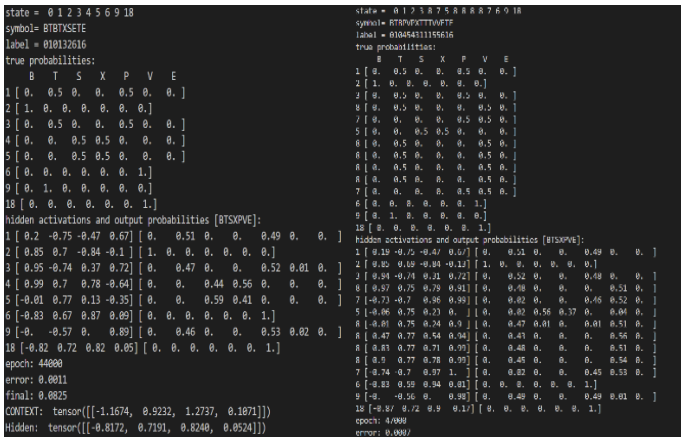
Simple Recurrent Networks (SRN) have difficulties to learn long range dependencies. Hence LSTM can learn long range dependencies better than SRN. For simple Reber Grammar, SRN can solve it. But for Embedded Reber Grammar (ERG), SRN fails since it cannot remember things long time ago.

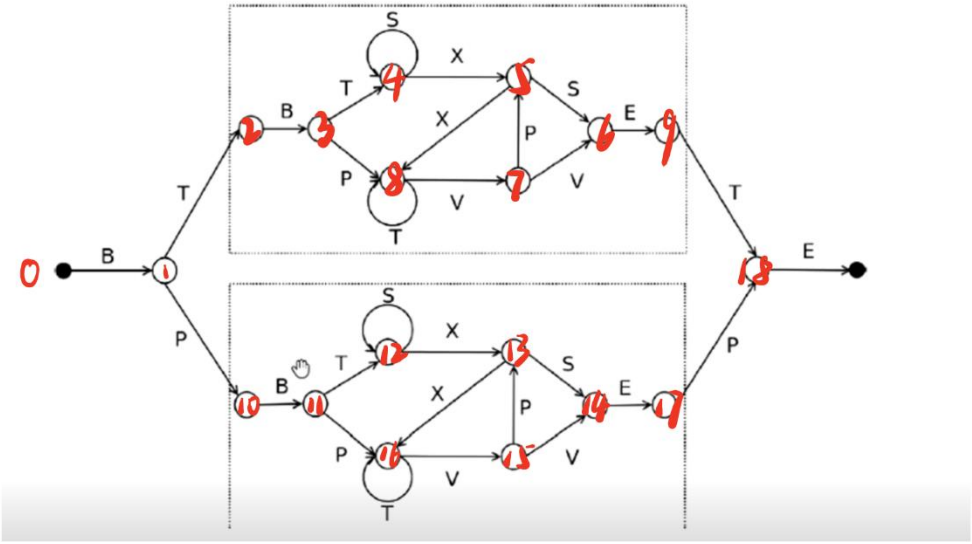


For example, the network must need to remember which transition (T or P) occurred after the initial B, which is important to predict T or P occurring before the final E.

By observing the result, we can plot the ERG with specific number on it.



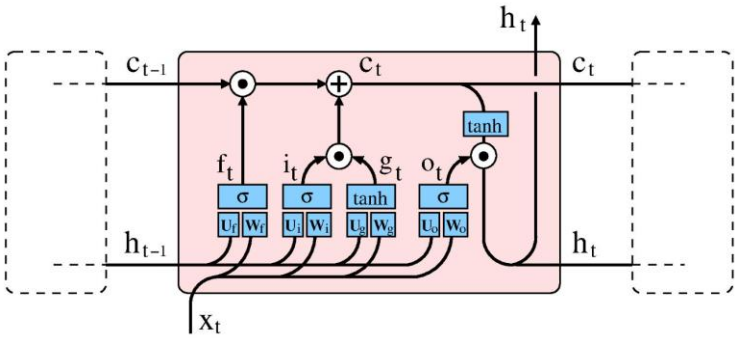




Hence, we know this task can only accomplished by LSTM instead of SRN.

Q: How the task is accomplished?

A: LSTM is a special form of RNN. LSTM stores information on the carry track and updates the information on the carry track later. In this way, information from a long time ago is preserved and information abandonment is prevented.

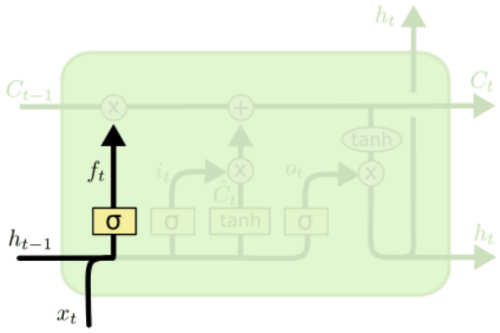


(This picture from 5b: Long Short Term Memory)

We can divide the task into several steps to solve:

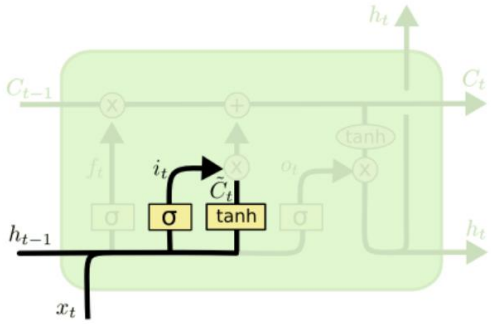
(1) We first initialize all weights and parameters for LSTM Model

(2) We decide what information we will discard from the cell state. This decision is done through a layer called forget gate by using sigmoid function. (Forget Gate)



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

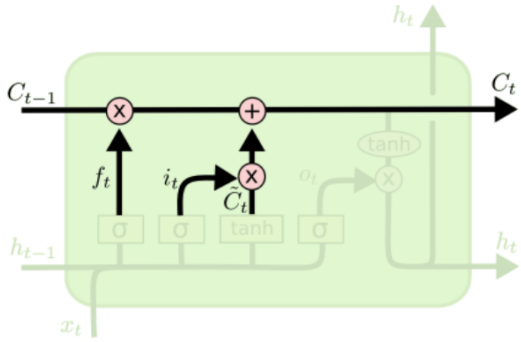
(3) We determine what new information is stored in the cell state. During this step, we will have two processes to be done. Firstly, we use sigmoid to act on the input to decide what value we are going to update. Secondly, we use tanh to create a new vector of candidate values that will be added to the state. (Input Gate)



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

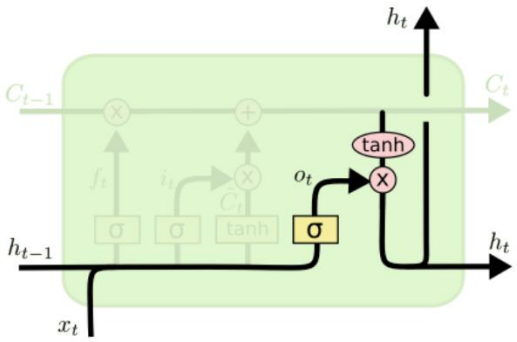
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

(4) We need to update the value Conveyor Belt by using the results above. We discard old information and add new information to the Conveyor Belt.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(5) We run a sigmoid layer to determine which part of the cell's state to output. Next, we process the cell state through tanh and multiply it with the output of the sigmoid result. (Output Gate). And as we can see, we make two copies of h_t. One is passed to the next step and the other is the output of the LSTM.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

(6) And we repeat the process of (2) – (5) until the end of sequence.

We can print the context units and hidden units to find out how model trains the data efficiently. (We set the hidden nodes as default as 4)

```
epoch: 0
error: 0.0627
final: 0.1277
CONTEXT:  tensor([[-0.1773,  0.3995,  0.0332,  0.5195]])
Hidden:  tensor([[-0.0766,  0.1933,  0.0233,  0.2070]])
```

```
epoch: 5000
error: 0.0050
final: 0.1048
CONTEXT:  tensor([[-1.1239,  0.9416,  2.5572,  0.3379]])
Hidden:  tensor([[-0.7736,  0.7257,  0.9742,  0.1104]])
```

```
epoch: 8000
error: 0.0020
final: 0.0965
CONTEXT:  tensor([[-1.3838,  0.9491,  2.6507,  0.2088]])
Hidden:  tensor([[-0.8646,  0.7320,  0.9780,  0.0881]])
```

```
epoch: 15000
error: 0.0018
final: 0.0900
CONTEXT:  tensor([[-1.5818,  0.9472,  1.8138,  0.2216]])
Hidden:  tensor([[-0.9136,  0.7313,  0.9267,  0.0935]])
```

```
epoch: 20000
error: 0.0012
final: 0.0905
CONTEXT:  tensor([[-1.4491,  0.9380,  1.3850,  0.0737]])
Hidden:  tensor([[-0.8903,  0.7258,  0.8549,  0.0350]])
```

```
epoch: 25000
error: 0.0006
final: 0.0665
CONTEXT:  tensor([[-1.8506,  0.9906,  2.5828,  3.5674]])
Hidden:  tensor([[-0.8842,  0.7556,  0.9313,  0.1373]])
```

```
epoch: 30000
error: 0.0008
final: 0.0700
CONTEXT:  tensor([[-1.3712,  0.9395,  1.8651,  0.3683]])
Hidden:  tensor([[-0.8748,  0.7275,  0.9243,  0.1713]])
```

```
epoch: 45000
error: 0.0012
final: 0.0566
CONTEXT:  tensor([[-1.8477,  0.9874,  1.4017,  3.1417]])
Hidden:  tensor([[-0.8980,  0.7539,  0.8306,  0.1201]])
```

```
epoch: 50000
error: 0.0010
final: 0.0655
CONTEXT:  tensor([[-1.7956,  0.9867,  1.7339,  3.3355]])
Hidden:  tensor([[-0.8892,  0.7535,  0.8737,  0.1245]])
```

By observing above data, we can see the context units close to [-0.9, 0.75, 0.85, 0.12], which can prove that LSTM can solve long dependencies problem with the help of context.

References:

1. https://blog.csdn.net/hadoopdevelop/article/details/88964175
2. https://edstem.org/au/courses/8538/lessons/20862/slides/148710