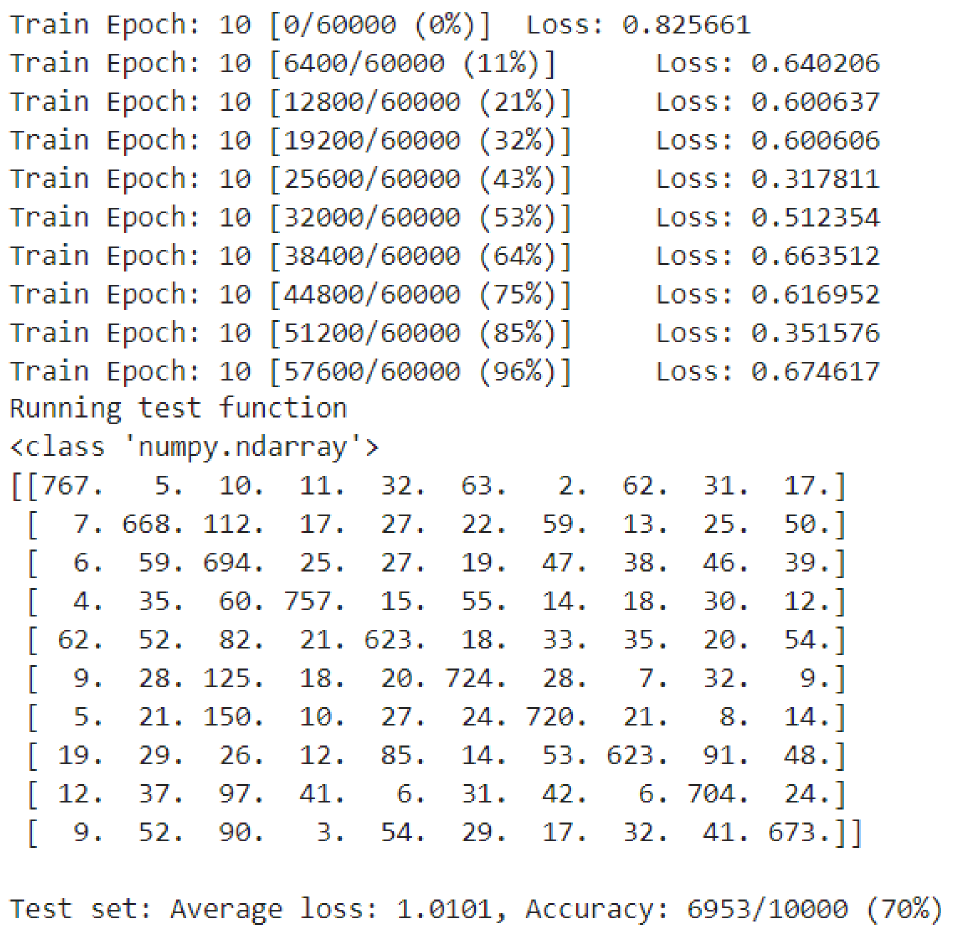
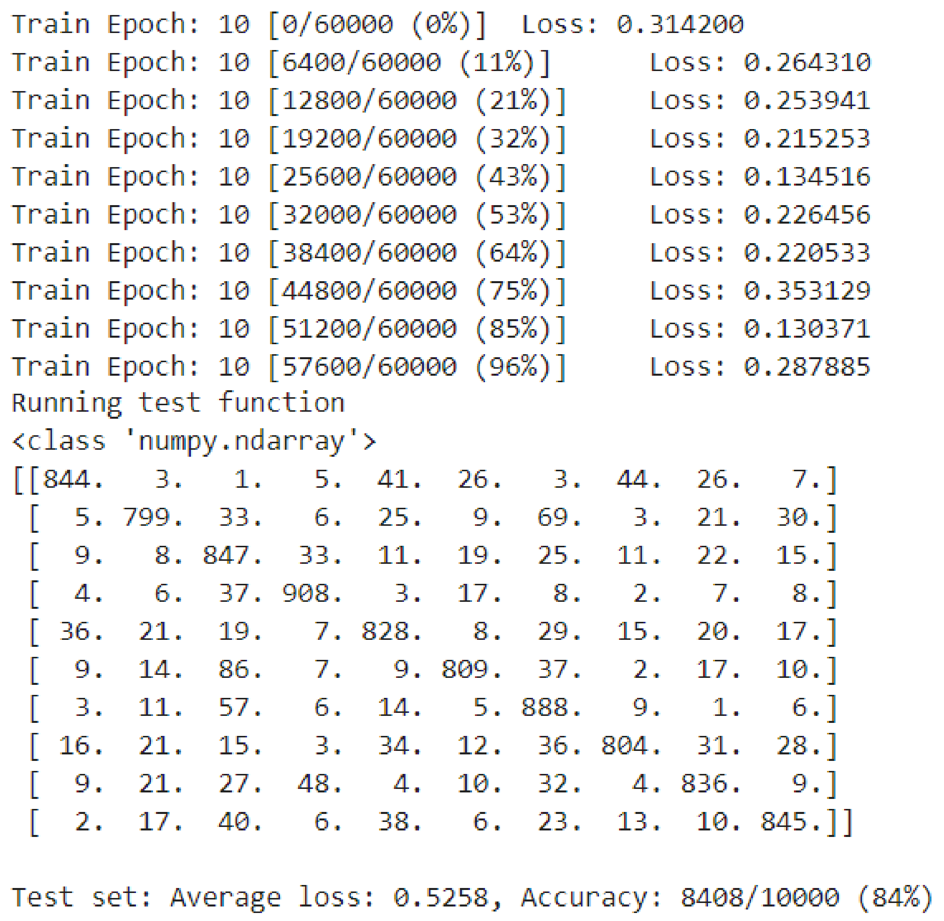
Part 1.1 – linear net



```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.825661
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.640206
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.600637
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.600606
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.317811
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.512354
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.663512
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.616952
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.351576
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.674617
Running test function
<class 'numpy.ndarray'>
[[767.   5.  10.  11.  32.  63.   2.  62.  31.  17.]
 [  7. 668. 112.  17.  27.  22.  59.  13.  25.  50.]
 [  6.  59. 694.  25.  27.  19.  47.  38.  46.  39.]
 [  4.  35.  60. 757.  15.  55.  14.  18.  30.  12.]
 [ 62.  52.  82.  21. 623.  18.  33.  35.  20.  54.]
 [  9.  28. 125.  18.  20. 724.  28.   7.  32.   9.]
 [  5.  21. 150.  10.  27.  24. 720.  21.   8.  14.]
 [ 19.  29.  26.  12.  85.  14.  53. 623.  91.  48.]
 [ 12.  37.  97.  41.   6.  31.  42.   6. 704.  24.]
 [  9.  52.  90.   3.  54.  29.  17.  32.  41. 673.]]

Test set: Average loss: 1.0101, Accuracy: 6953/10000 (70%)
```

- Simple linear model: (28*28*10) + 10 = 7850 parameters

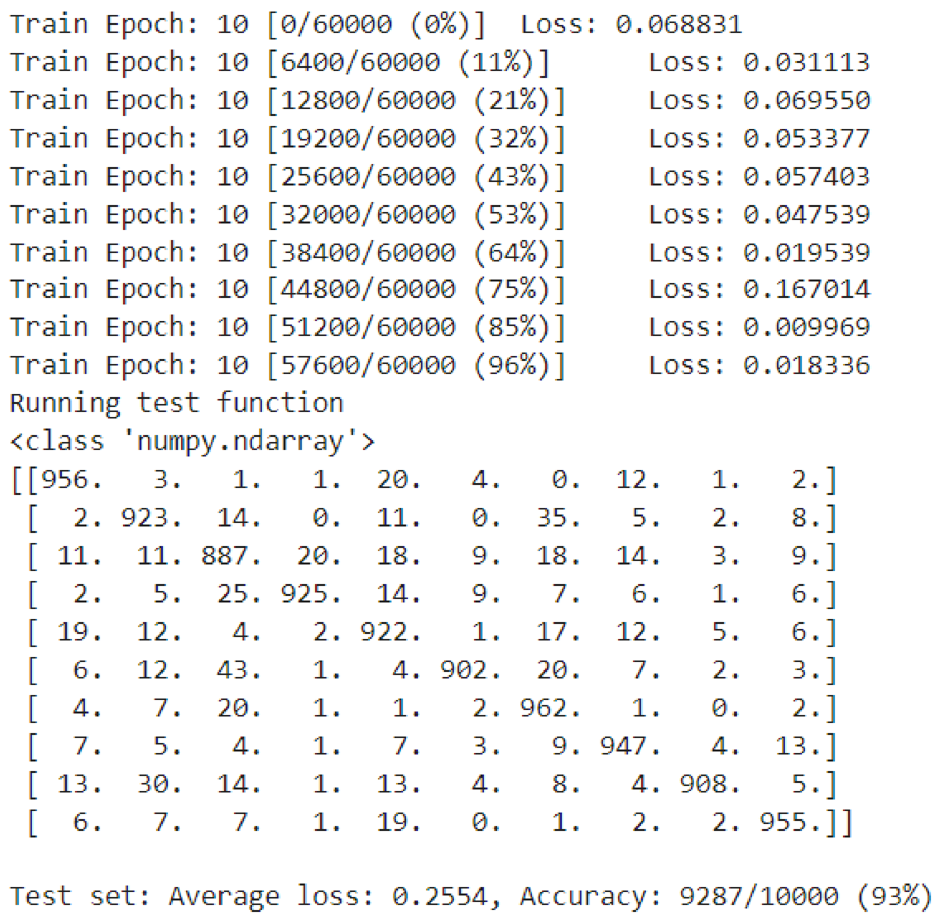
Part 1.2 – fully connected 2 layer net



```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.314200
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.264310
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.253941
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.215253
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.134516
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.226456
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.220533
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.353129
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.130371
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.287885
Running test function
<class 'numpy.ndarray'>
[[844.   3.   1.   5.  41.  26.   3.  44.  26.   7.]
 [  5. 799.  33.   6.  25.   9.  69.   3.  21.  30.]
 [  9.   8. 847.  33.  11.  19.  25.  11.  22.  15.]
 [  4.   6.  37. 908.   3.  17.   8.   2.   7.   8.]
 [ 36.  21.  19.   7. 828.   8.  29.  15.  20.  17.]
 [  9.  14.  86.   7.   9. 809.  37.   2.  17.  10.]
 [  3.  11.  57.   6.  14.   5. 888.   9.   1.   6.]
 [ 16.  21.  15.   3.  34.  12.  36. 804.  31.  28.]
 [  9.  21.  27.  48.   4.  10.  32.   4. 836.   9.]
 [  2.  17.  40.   6.  38.   6.  23.  13.  10. 845.]]

Test set: Average loss: 0.5258, Accuracy: 8408/10000 (84%)
```

- (784*128 + 128) + (128*10 + 10) = 101,770 parameters

Part 1.3 – conv net



```
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.068831
Train Epoch: 10 [6400/60000 (11%)]      Loss: 0.031113
Train Epoch: 10 [12800/60000 (21%)]     Loss: 0.069550
Train Epoch: 10 [19200/60000 (32%)]     Loss: 0.053377
Train Epoch: 10 [25600/60000 (43%)]     Loss: 0.057403
Train Epoch: 10 [32000/60000 (53%)]     Loss: 0.047539
Train Epoch: 10 [38400/60000 (64%)]     Loss: 0.019539
Train Epoch: 10 [44800/60000 (75%)]     Loss: 0.167014
Train Epoch: 10 [51200/60000 (85%)]     Loss: 0.009969
Train Epoch: 10 [57600/60000 (96%)]     Loss: 0.018336
Running test function
<class 'numpy.ndarray'>
[[956.   3.   1.   1.  20.   4.   0.  12.   1.   2.]
 [  2. 923.  14.   0.  11.   0.  35.   5.   2.   8.]
 [ 11.  11. 887.  20.  18.   9.  18.  14.   3.   9.]
 [  2.   5.  25. 925.  14.   9.   7.   6.   1.   6.]
 [ 19.  12.   4.   2. 922.   1.  17.  12.   5.   6.]
 [  6.  12.  43.   1.   4. 902.  20.   7.   2.   3.]
 [  4.   7.  20.   1.   1.   2. 962.   1.   0.   2.]
 [  7.   5.   4.   1.   7.   3.   9. 947.   4.  13.]
 [ 13.  30.  14.   1.  13.   4.   8.   4. 908.   5.]
 [  6.   7.   7.   1.  19.   0.   1.   2.   2. 955.]]

Test set: Average loss: 0.2554, Accuracy: 9287/10000 (93%)
```

Number of parameters calculation:

- First layer (conv1)
  - 32 filters of size 7 by 7. Including the bias, we have:
  - 7 * 7+1 per filter, * 32 filters = 1600 parameters
- Second layer (conv2)
  - 64 filters of 7*7, input of 32 channels, i.e.
  - (7*7*32 +1) * 64 = 100,416 parameters
- Fully connected layer:
  - 1024 input into 120 output features i.e.
  - (1024+1)*120 = 123,000 parameters
- Fully connected layer 2:
  - 120 input into 10 output features i.e. (120+1)*10 = 1,210
- Total parameters: 225,226

Part 1.4 Model comparisons

Relative accuracy:

- We can see that the conv net model achieves a much higher degree of accuracy (93%) versus the simple linear net (70%) and the fully connected net (84%)
- The simple linear net fits relatively quickly, however is unable to capture the relationships in the data well, which leads to a lower accuracy compared to the other models
- The fully connected net adds a hidden layer, which allows this to capture more relationships, leading to better performance versus the simple linear net
- The convolutional network introduces additional layers and architecture that is suited to image classification tasks, to better capture the relationship features in an image (e.g. edges). We can see this come through in the much better accuracy of this model

Number of Parameters

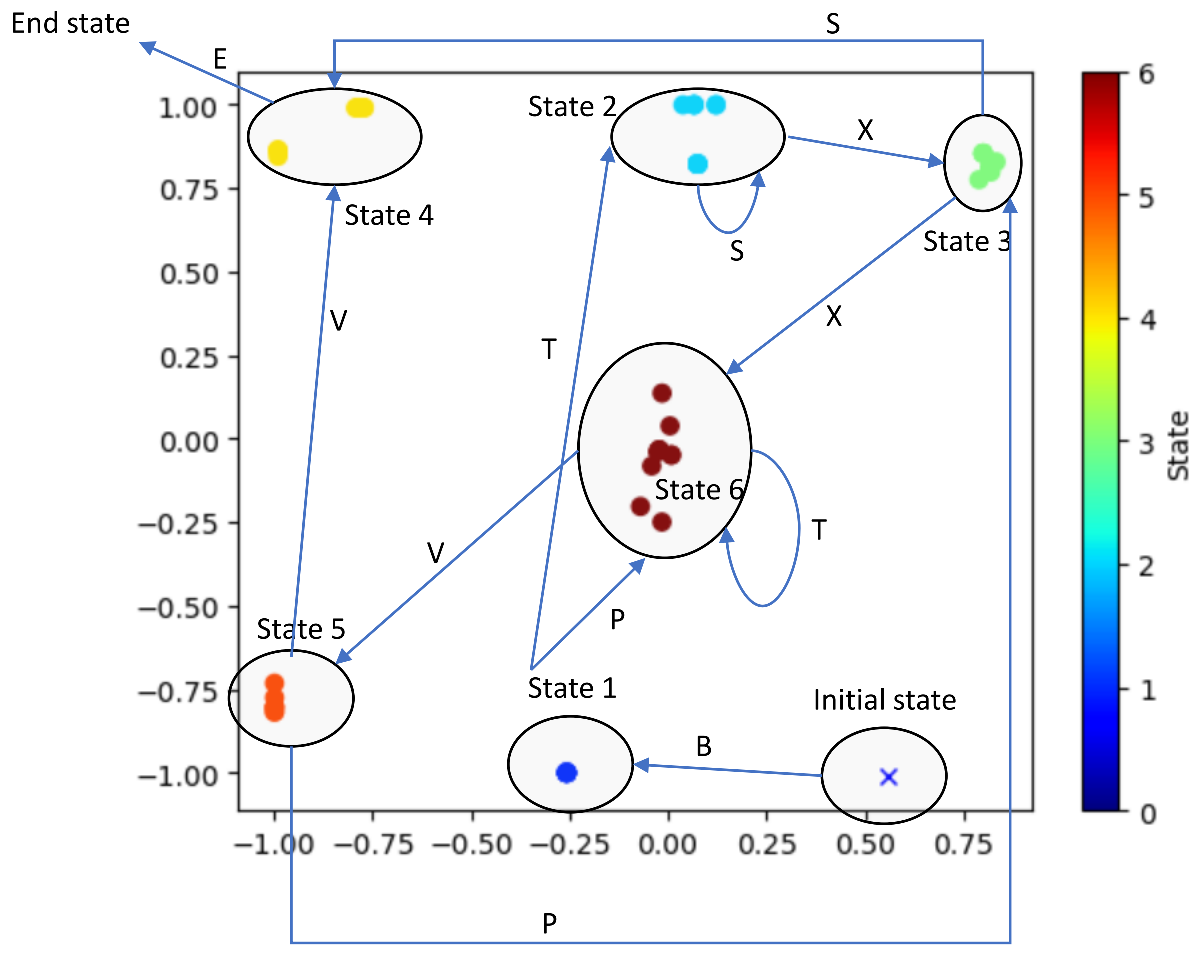
- The linear net has the lowest number of parameters, which leads to a lower accuracy. It is however quicker to train, and may in some cases lead to less over-fitting (i.e. the model can be more parsimonious)
- The full net has more parameters than the linear net allowing it to capture more complex relationships, leading to a better accuracy
- The convolutional net has a more complex architecture designed to better capture images. While the number of parameters is much larger, they are still optimised through the sharing of weights. The architectural design better captures image features like edges, and reduces the risk of overfitting. Nevertheless, there is a higher number of parameters, in this model, and a higher degree of accuracy

Part 1.4 Model comparisons (con’td)

Confusion matrix:

- The netlin confusion matrix shows a large number of misclassifications. We can see this by looking along the diagonal, and the spread of exposure away from the diagonal. This model shows a wide spread across the whole matrix, showing a poor classification rate
- The netfull model confusion matrix shows a better classification rate, as can be seen by the better capture of exposure along the diagonal. T
- Lastly, the NetConv model confusion model demonstrates the lowest number of misclassifications. We can see that the diagonal captures >90% of the exposure, with some small misclassifications away from the diagonal (note, we will always get some misclassifications as the models are rarely perfect in their predictions)

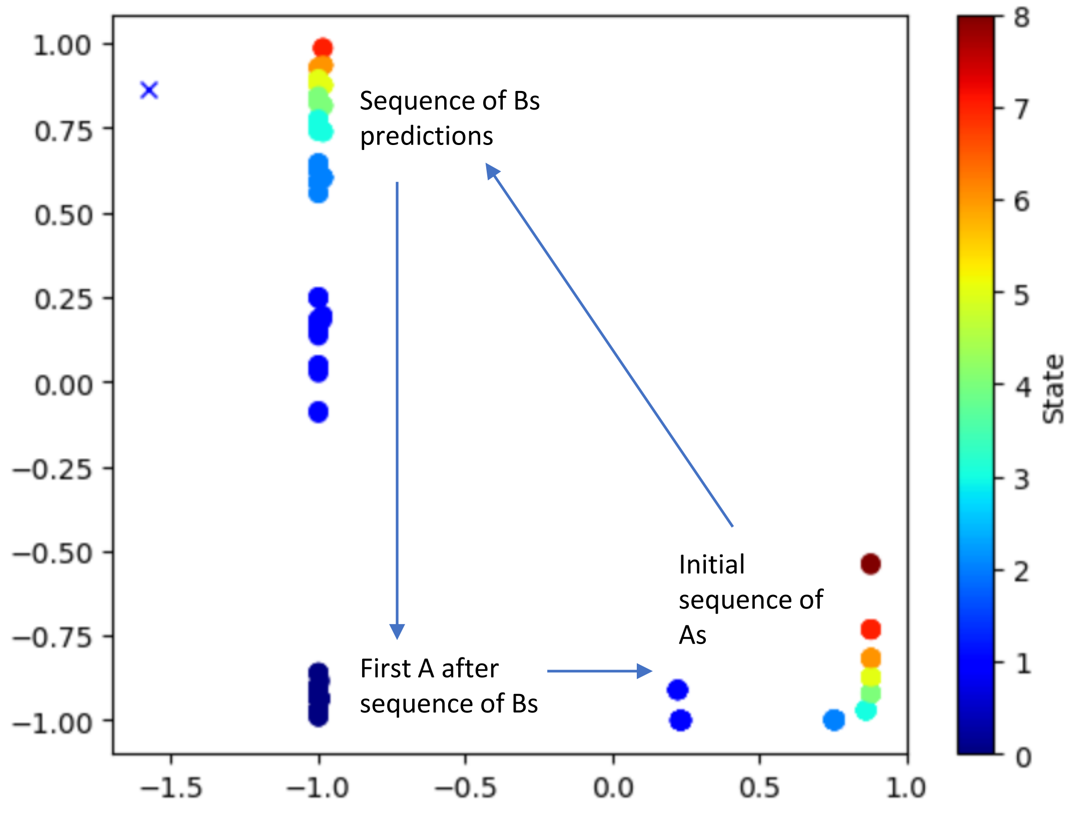
Part 3.1 – Reber Grammar Prediction



- The chart on the left shows the output of the hidden unit activations after 50,000 epochs. We can see a clear separation for each of the states. We can see the applicable activations for each state from the printouts for one of the examples, as per below:

```
hidden activations and output probabilities [BTSXPVE]:
1 [-0.26 -1.  ] [0.   0.42 0.   0.   0.52 0.06 0.  ]
2 [0.07 0.82] [0.   0.   0.41 0.53 0.   0.05 0.  ]
3 [0.8  0.85] [0.   0.   0.63 0.37 0.   0.   0.  ]
6 [ 0.01 -0.05] [0.   0.49 0.02 0.02 0.02 0.45 0.  ]
5 [-1.   -0.82] [0.   0.02 0.   0.   0.47 0.52 0.  ]
4 [-0.99  0.86] [0. 0. 0. 0. 0. 0. 1.]
epoch: 8
error: 0.0015
```

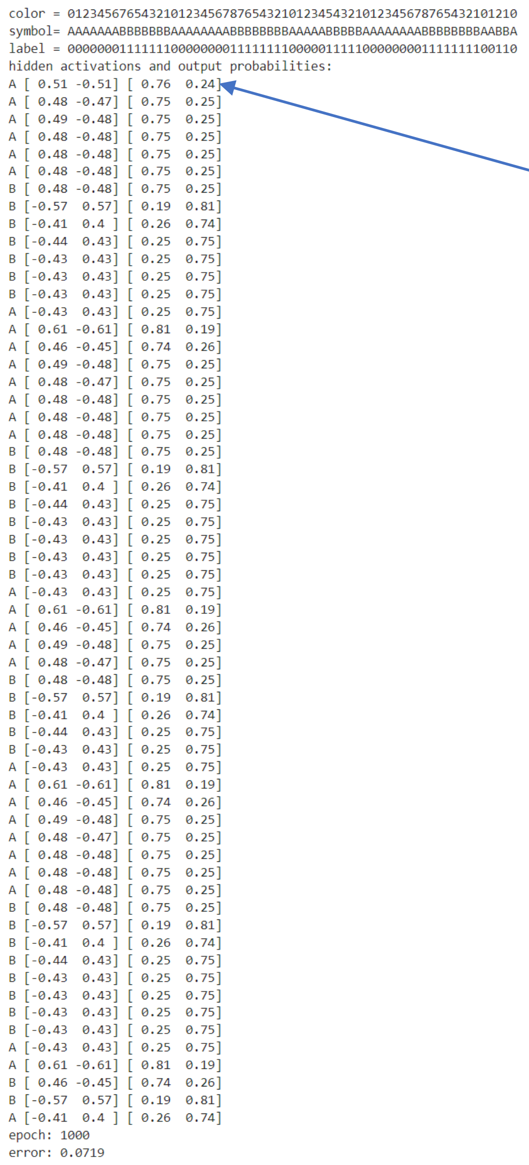
Part 3.2/3.3 – anbn language prediction



The model is classifying sequentially, with the hidden unit activations showing a clear separation between certain groups – e.g. the initial sequence of As, subsequent sequence of Bs, and then a subsequent sequence of As following Bs

We further assess the underlying numbers on the next page, showing how the unit activations change from the start of the model to a trained version

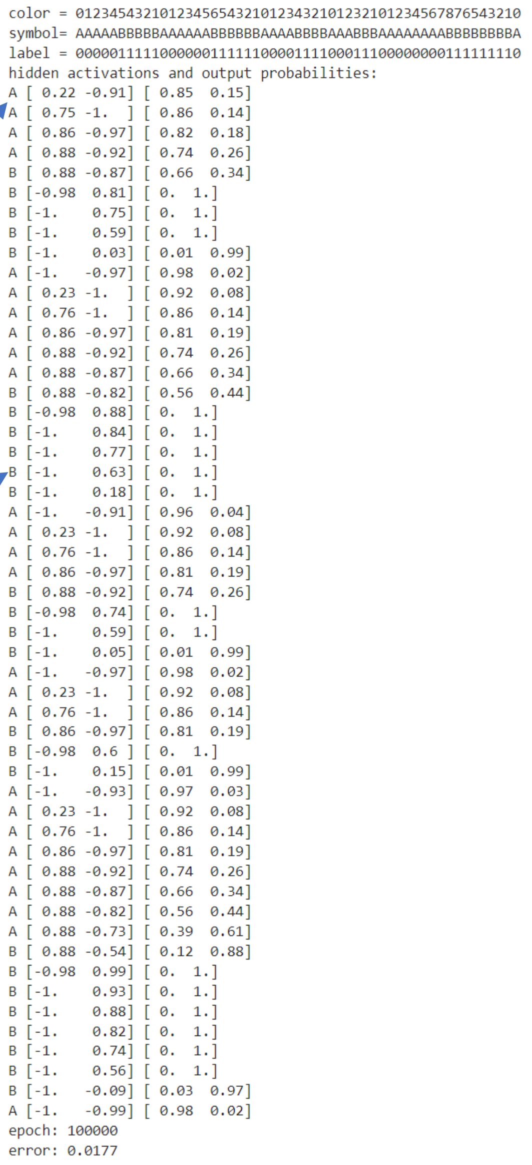
Part 3.3 – anbn commentary



```
color = 0123456765432101234567876543210123454321012345678765432101210
symbol= AAAAAAABBBBBBAAAAAAAABBBBBBBBAAAAABBBBBAAAAAAAAABBBBBBBBAABBA
label = 0000000111111000000001111111100000111110000000001111111100110
hidden activations and output probabilities:
A [ 0.51 -0.51] [ 0.76  0.24]
A [ 0.48 -0.47] [ 0.75  0.25]
A [ 0.49 -0.48] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
B [ 0.48 -0.48] [ 0.75  0.25]
B [-0.57  0.57] [ 0.19  0.81]
B [-0.41  0.4 ] [ 0.26  0.74]
B [-0.44  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
A [-0.43  0.43] [ 0.25  0.75]
A [ 0.61 -0.61] [ 0.81  0.19]
A [ 0.46 -0.45] [ 0.74  0.26]
A [ 0.49 -0.48] [ 0.75  0.25]
A [ 0.48 -0.47] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
B [ 0.48 -0.48] [ 0.75  0.25]
B [-0.57  0.57] [ 0.19  0.81]
B [-0.41  0.4 ] [ 0.26  0.74]
B [-0.44  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
A [-0.43  0.43] [ 0.25  0.75]
A [ 0.61 -0.61] [ 0.81  0.19]
A [ 0.46 -0.45] [ 0.74  0.26]
A [ 0.49 -0.48] [ 0.75  0.25]
A [ 0.48 -0.47] [ 0.75  0.25]
B [ 0.48 -0.48] [ 0.75  0.25]
B [-0.57  0.57] [ 0.19  0.81]
B [-0.41  0.4 ] [ 0.26  0.74]
B [-0.44  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
A [-0.43  0.43] [ 0.25  0.75]
A [ 0.61 -0.61] [ 0.81  0.19]
A [ 0.46 -0.45] [ 0.74  0.26]
A [ 0.49 -0.48] [ 0.75  0.25]
A [ 0.48 -0.47] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
A [ 0.48 -0.48] [ 0.75  0.25]
B [ 0.48 -0.48] [ 0.75  0.25]
B [-0.57  0.57] [ 0.19  0.81]
B [-0.41  0.4 ] [ 0.26  0.74]
B [-0.44  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
B [-0.43  0.43] [ 0.25  0.75]
A [-0.43  0.43] [ 0.25  0.75]
A [ 0.61 -0.61] [ 0.81  0.19]
B [ 0.46 -0.45] [ 0.74  0.26]
B [-0.57  0.57] [ 0.19  0.81]
A [-0.41  0.4 ] [ 0.26  0.74]
epoch: 1000
error: 0.0719
```

At the start of the training (epoch 1000), we can see that initial unit activations are biased towards the probabilities of A and B broadly (80%/20%), but do not properly capture the relationships between the sequences of As and Bs

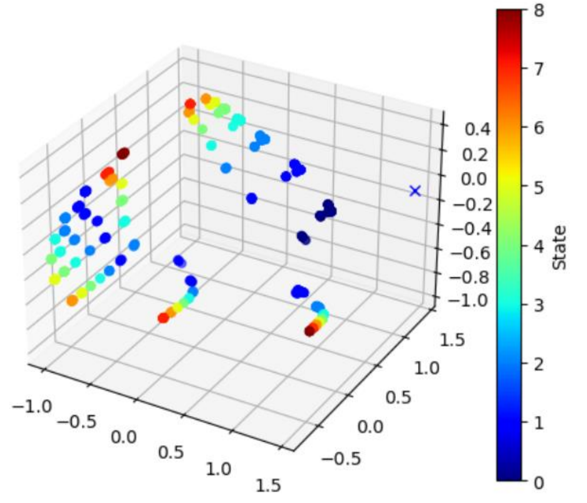
As the training continues, the model is better able to capture the sequencing, e.g. that the first A occurs with a probability of approx. 80%, and that subsequent n Bs after the first B are fully predictable (based on the number of As that preceded)



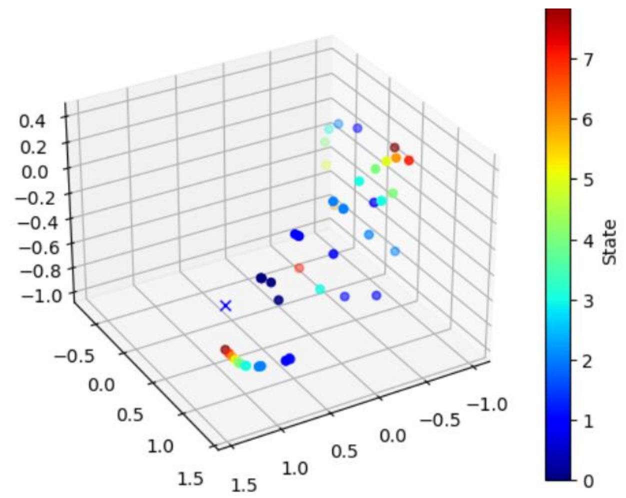
```
color = 0123454321012345654321012343210123210123456787654321O
symbol= AAAAABBBBBAAAAAAABBBBBBAAAABBBBAAABBBAAAAAAAAABBBBBBBBA
label = 0000011111000000011111100001111000111000000000111111110
hidden activations and output probabilities:
A [ 0.22 -0.91] [ 0.85  0.15]
A [ 0.75 -1.  ] [ 0.86  0.14]
A [ 0.86 -0.97] [ 0.82  0.18]
A [ 0.88 -0.92] [ 0.74  0.26]
B [ 0.88 -0.87] [ 0.66  0.34]
B [-0.98  0.81] [ 0.   1.]
B [-1.    0.75] [ 0.   1.]
B [-1.    0.59] [ 0.   1.]
B [-1.    0.03] [ 0.01  0.99]
A [-1.   -0.97] [ 0.98  0.02]
A [ 0.23 -1.  ] [ 0.92  0.08]
A [ 0.76 -1.  ] [ 0.86  0.14]
A [ 0.86 -0.97] [ 0.81  0.19]
A [ 0.88 -0.92] [ 0.74  0.26]
A [ 0.88 -0.87] [ 0.66  0.34]
B [ 0.88 -0.82] [ 0.56  0.44]
B [-0.98  0.88] [ 0.   1.]
B [-1.    0.84] [ 0.   1.]
B [-1.    0.77] [ 0.   1.]
B [-1.    0.63] [ 0.   1.]
B [-1.    0.18] [ 0.   1.]
A [-1.   -0.91] [ 0.96  0.04]
A [ 0.23 -1.  ] [ 0.92  0.08]
A [ 0.76 -1.  ] [ 0.86  0.14]
A [ 0.86 -0.97] [ 0.81  0.19]
B [ 0.88 -0.92] [ 0.74  0.26]
B [-0.98  0.74] [ 0.   1.]
B [-1.    0.59] [ 0.   1.]
B [-1.    0.05] [ 0.01  0.99]
A [-1.   -0.97] [ 0.98  0.02]
A [ 0.23 -1.  ] [ 0.92  0.08]
A [ 0.76 -1.  ] [ 0.86  0.14]
B [ 0.86 -0.97] [ 0.81  0.19]
B [-0.98  0.6 ] [ 0.   1.]
B [-1.    0.15] [ 0.01  0.99]
A [-1.   -0.93] [ 0.97  0.03]
A [ 0.23 -1.  ] [ 0.92  0.08]
A [ 0.76 -1.  ] [ 0.86  0.14]
A [ 0.86 -0.97] [ 0.81  0.19]
A [ 0.88 -0.92] [ 0.74  0.26]
A [ 0.88 -0.87] [ 0.66  0.34]
A [ 0.88 -0.82] [ 0.56  0.44]
A [ 0.88 -0.73] [ 0.39  0.61]
B [ 0.88 -0.54] [ 0.12  0.88]
B [-0.98  0.99] [ 0.   1.]
B [-1.    0.93] [ 0.   1.]
B [-1.    0.88] [ 0.   1.]
B [-1.    0.82] [ 0.   1.]
B [-1.    0.74] [ 0.   1.]
B [-1.    0.56] [ 0.   1.]
B [-1.   -0.09] [ 0.03  0.97]
A [-1.   -0.99] [ 0.98  0.02]
epoch: 100000
error: 0.0177
```

Part 3.4 – anbncn language prediction

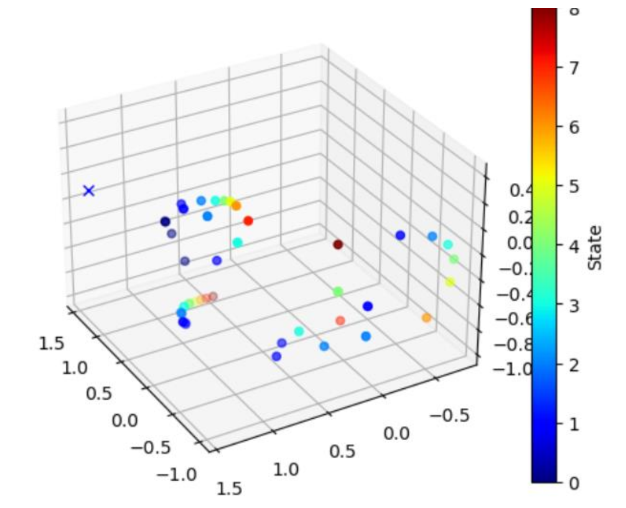
We see below various hidden unit activations, rotated. Further commentary and underlying tables on next page



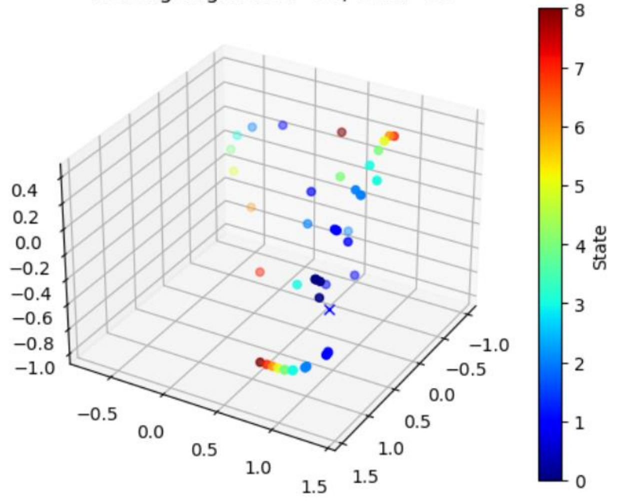
Viewing angle: elev=30, azim=30



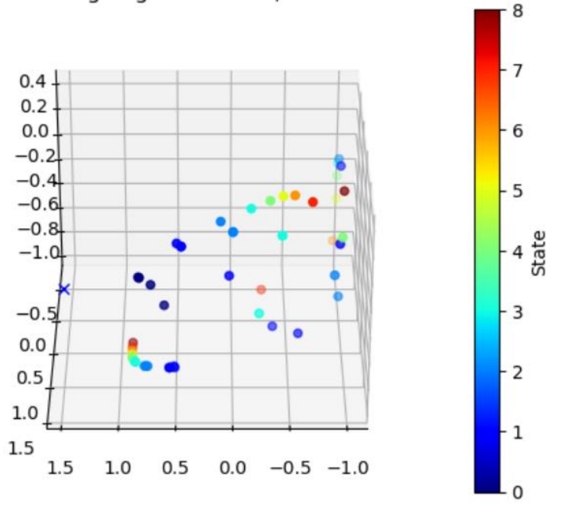
Viewing angle: elev=30, azim=90



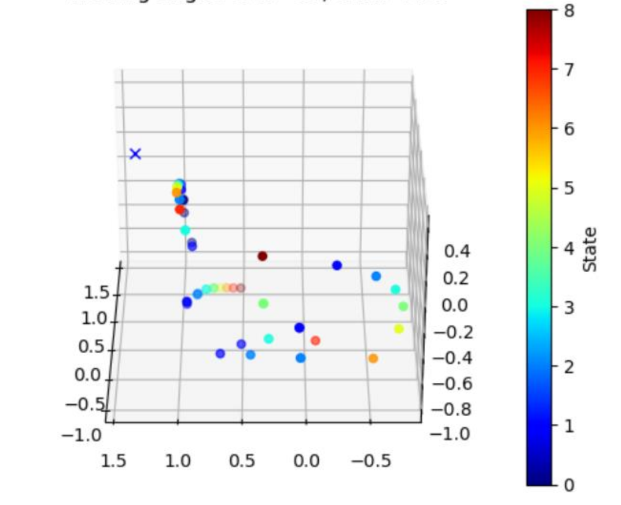
Viewing angle: elev=30, azim=180



Viewing angle: elev=30, azim=60

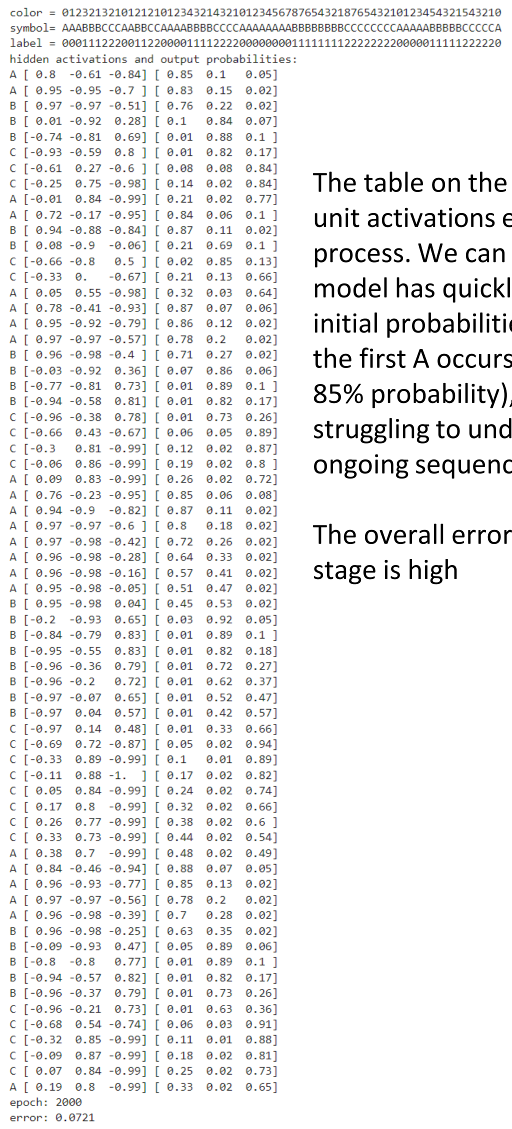


Viewing angle: elev=30, azim=120



Part 3.5 – anbncn language prediction

In addition to the 3d plots on the previous page, we can also see the underlying data as shown below, which show how unit activation are changing over time as the model trains



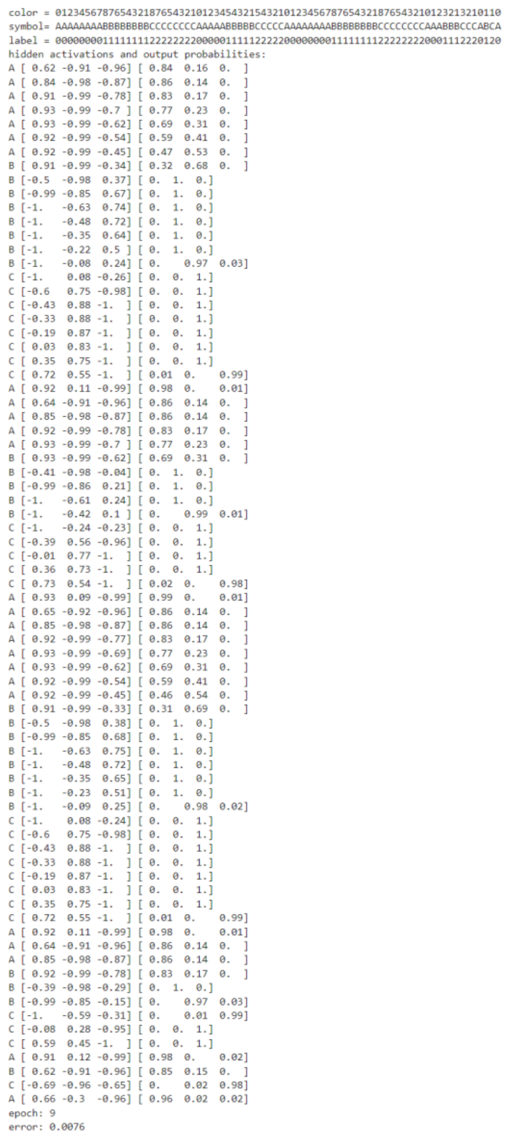
```
color = 0123213210121210123432143210123456787654321876543210123454321543210
symbol= AAABBBCCCAABBCCAAAABBBBCCCCAAAAAAAABBBBBBBBCCCCCCCCAAAAABBBBBCCCCCA
label = 0001112220011220000111122220000000011111111222222220000011111222220
hidden activations and output probabilities:
A [ 0.8  -0.61 -0.84] [ 0.85  0.1   0.05]
A [ 0.95 -0.95 -0.7 ] [ 0.83  0.15  0.02]
B [ 0.97 -0.97 -0.51] [ 0.76  0.22  0.02]
B [ 0.01 -0.92  0.28] [ 0.1   0.84  0.07]
B [-0.74 -0.81  0.69] [ 0.01  0.88  0.1 ]
C [-0.93 -0.59  0.8 ] [ 0.01  0.82  0.17]
C [-0.61  0.27 -0.6 ] [ 0.08  0.08  0.84]
C [-0.25  0.75 -0.98] [ 0.14  0.02  0.84]
A [-0.01  0.84 -0.99] [ 0.21  0.02  0.77]
A [ 0.72 -0.17 -0.95] [ 0.84  0.06  0.1 ]
B [ 0.94 -0.88 -0.84] [ 0.87  0.11  0.02]
B [ 0.08 -0.9  -0.06] [ 0.21  0.69  0.1 ]
C [-0.66 -0.8   0.5 ] [ 0.02  0.85  0.13]
C [-0.33  0.   -0.67] [ 0.21  0.13  0.66]
A [ 0.05  0.55 -0.98] [ 0.32  0.03  0.64]
A [ 0.78 -0.41 -0.93] [ 0.87  0.07  0.06]
A [ 0.95 -0.92 -0.79] [ 0.86  0.12  0.02]
A [ 0.97 -0.97 -0.57] [ 0.78  0.2   0.02]
B [ 0.96 -0.98 -0.4 ] [ 0.71  0.27  0.02]
B [-0.03 -0.92  0.36] [ 0.07  0.86  0.06]
B [-0.77 -0.81  0.73] [ 0.01  0.89  0.1 ]
B [-0.94 -0.58  0.81] [ 0.01  0.82  0.17]
C [-0.96 -0.38  0.78] [ 0.01  0.73  0.26]
C [-0.66  0.43 -0.67] [ 0.06  0.05  0.89]
C [-0.3   0.81 -0.99] [ 0.12  0.02  0.87]
C [-0.06  0.86 -0.99] [ 0.19  0.02  0.8 ]
A [ 0.09  0.83 -0.99] [ 0.26  0.02  0.72]
A [ 0.76 -0.23 -0.95] [ 0.85  0.06  0.08]
A [ 0.94 -0.9  -0.82] [ 0.87  0.11  0.02]
A [ 0.97 -0.97 -0.6 ] [ 0.8   0.18  0.02]
A [ 0.97 -0.98 -0.42] [ 0.72  0.26  0.02]
A [ 0.96 -0.98 -0.28] [ 0.64  0.33  0.02]
A [ 0.96 -0.98 -0.16] [ 0.57  0.41  0.02]
A [ 0.95 -0.98 -0.05] [ 0.51  0.47  0.02]
B [ 0.95 -0.98  0.04] [ 0.45  0.53  0.02]
B [-0.2  -0.93  0.65] [ 0.03  0.92  0.05]
B [-0.84 -0.79  0.83] [ 0.01  0.89  0.1 ]
B [-0.95 -0.55  0.83] [ 0.01  0.82  0.18]
B [-0.96 -0.36  0.79] [ 0.01  0.72  0.27]
B [-0.96 -0.2   0.72] [ 0.01  0.62  0.37]
B [-0.97 -0.07  0.65] [ 0.01  0.52  0.47]
B [-0.97  0.04  0.57] [ 0.01  0.42  0.57]
C [-0.97  0.14  0.48] [ 0.01  0.33  0.66]
C [-0.69  0.72 -0.87] [ 0.05  0.02  0.94]
C [-0.33  0.89 -0.99] [ 0.1   0.01  0.89]
C [-0.11  0.88 -1.  ] [ 0.17  0.02  0.82]
C [ 0.05  0.84 -0.99] [ 0.24  0.02  0.74]
C [ 0.17  0.8  -0.99] [ 0.32  0.02  0.66]
C [ 0.26  0.77 -0.99] [ 0.38  0.02  0.6 ]
C [ 0.33  0.73 -0.99] [ 0.44  0.02  0.54]
A [ 0.38  0.7  -0.99] [ 0.48  0.02  0.49]
A [ 0.84 -0.46 -0.94] [ 0.88  0.07  0.05]
A [ 0.96 -0.93 -0.77] [ 0.85  0.13  0.02]
A [ 0.97 -0.97 -0.56] [ 0.78  0.2   0.02]
A [ 0.96 -0.98 -0.39] [ 0.7   0.28  0.02]
B [ 0.96 -0.98 -0.25] [ 0.63  0.35  0.02]
B [-0.09 -0.93  0.47] [ 0.05  0.89  0.06]
B [-0.8  -0.8   0.77] [ 0.01  0.89  0.1 ]
B [-0.94 -0.57  0.82] [ 0.01  0.82  0.17]
B [-0.96 -0.37  0.79] [ 0.01  0.73  0.26]
C [-0.96 -0.21  0.73] [ 0.01  0.63  0.36]
C [-0.68  0.54 -0.74] [ 0.06  0.03  0.91]
C [-0.32  0.85 -0.99] [ 0.11  0.01  0.88]
C [-0.09  0.87 -0.99] [ 0.18  0.02  0.81]
C [ 0.07  0.84 -0.99] [ 0.25  0.02  0.73]
A [ 0.19  0.8  -0.99] [ 0.33  0.02  0.65]
epoch: 2000
error: 0.0721
```

The table on the left shows unit activations early in the process. We can see that the model has quickly learnt some initial probabilities (e.g. that the first A occurs with approx. 85% probability), however is struggling to understand ongoing sequences.

The overall error rate at this stage is high

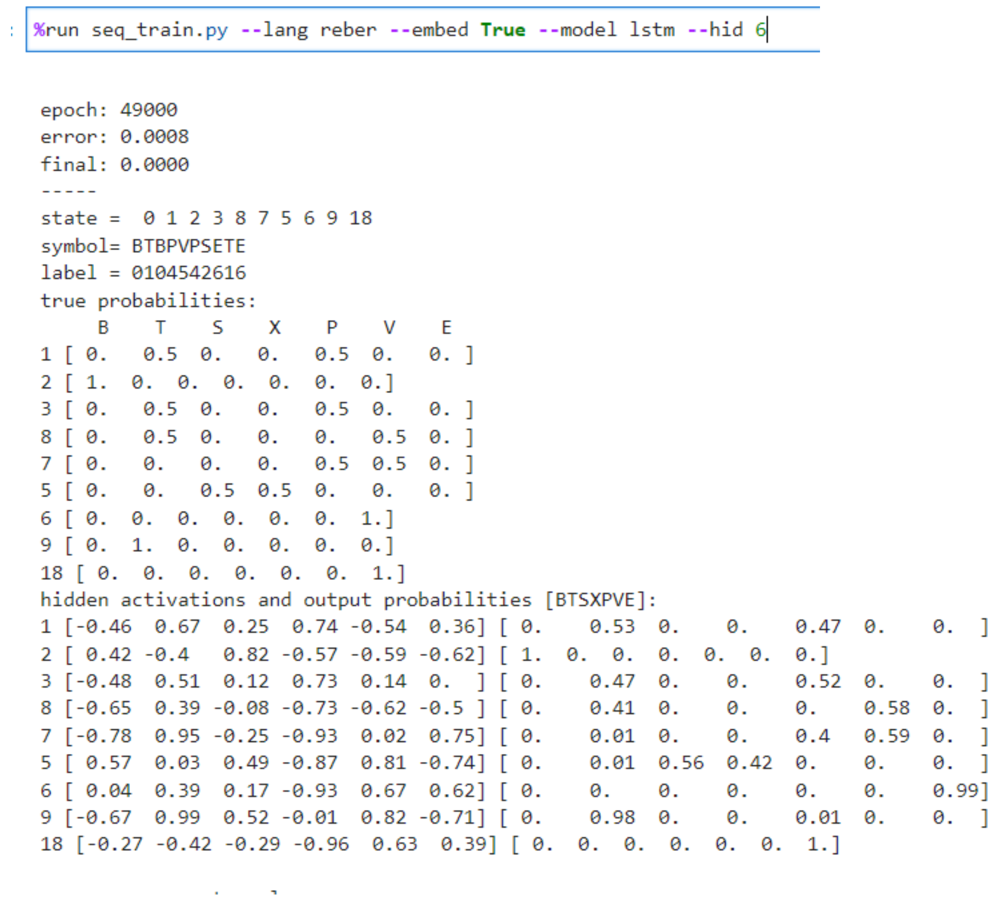
The table on the right shows unit activations after 200k epochs. At this stage, we can see a much better error rate, and that the later sequences are now being classified correctly.

The utilisation of the SRN model architecture allows us to take in sequential input, and update a hidden state over time to better capture dependencies over time. We can see that in this data, where the hidden activations are able to better predict Bc and Cs much later in the sequence.



```
color = 0123456787654321876543210123454321543210123456787654321876543210123213210110
symbol= AAAAAAAABBBBBBBBCCCCCCCCAAAAABBBBBCCCCCAAAAAAAABBBBBBBBCCCCCCCCAAABBBCCCABCA
label = 0000000011111111222222220000011111222220000000011111111222222220001112220120
hidden activations and output probabilities:
A [ 0.62 -0.91 -0.96] [ 0.84  0.16  0.  ]
A [ 0.84 -0.98 -0.87] [ 0.86  0.14  0.  ]
A [ 0.91 -0.99 -0.78] [ 0.83  0.17  0.  ]
A [ 0.93 -0.99 -0.7 ] [ 0.77  0.23  0.  ]
A [ 0.93 -0.99 -0.62] [ 0.69  0.31  0.  ]
A [ 0.92 -0.99 -0.54] [ 0.59  0.41  0.  ]
A [ 0.92 -0.99 -0.45] [ 0.47  0.53  0.  ]
B [ 0.91 -0.99 -0.34] [ 0.32  0.68  0.  ]
B [-0.5  -0.98  0.37] [ 0.  1.  0.]
B [-0.99 -0.85  0.67] [ 0.  1.  0.]
B [-1.   -0.63  0.74] [ 0.  1.  0.]
B [-1.   -0.48  0.72] [ 0.  1.  0.]
B [-1.   -0.35  0.64] [ 0.  1.  0.]
B [-1.   -0.22  0.5 ] [ 0.  1.  0.]
B [-1.   -0.08  0.24] [ 0.    0.97  0.03]
C [-1.    0.08 -0.26] [ 0.  0.  1.]
C [-0.6   0.75 -0.98] [ 0.  0.  1.]
C [-0.43  0.88 -1.  ] [ 0.  0.  1.]
C [-0.33  0.88 -1.  ] [ 0.  0.  1.]
C [-0.19  0.87 -1.  ] [ 0.  0.  1.]
C [ 0.03  0.83 -1.  ] [ 0.  0.  1.]
C [ 0.35  0.75 -1.  ] [ 0.  0.  1.]
C [ 0.72  0.55 -1.  ] [ 0.01  0.    0.99]
A [ 0.92  0.11 -0.99] [ 0.98  0.    0.01]
A [ 0.64 -0.91 -0.96] [ 0.86  0.14  0.  ]
A [ 0.85 -0.98 -0.87] [ 0.86  0.14  0.  ]
A [ 0.92 -0.99 -0.78] [ 0.83  0.17  0.  ]
A [ 0.93 -0.99 -0.7 ] [ 0.77  0.23  0.  ]
B [ 0.93 -0.99 -0.62] [ 0.69  0.31  0.  ]
B [-0.41 -0.98 -0.04] [ 0.  1.  0.]
B [-0.99 -0.86  0.21] [ 0.  1.  0.]
B [-1.   -0.61  0.24] [ 0.  1.  0.]
B [-1.   -0.42  0.1 ] [ 0.    0.99  0.01]
C [-1.   -0.24 -0.23] [ 0.  0.  1.]
C [-0.39  0.56 -0.96] [ 0.  0.  1.]
C [-0.01  0.77 -1.  ] [ 0.  0.  1.]
C [ 0.36  0.73 -1.  ] [ 0.  0.  1.]
C [ 0.73  0.54 -1.  ] [ 0.02  0.    0.98]
A [ 0.93  0.09 -0.99] [ 0.99  0.    0.01]
A [ 0.65 -0.92 -0.96] [ 0.86  0.14  0.  ]
A [ 0.85 -0.98 -0.87] [ 0.86  0.14  0.  ]
A [ 0.92 -0.99 -0.77] [ 0.83  0.17  0.  ]
A [ 0.93 -0.99 -0.69] [ 0.77  0.23  0.  ]
A [ 0.93 -0.99 -0.62] [ 0.69  0.31  0.  ]
A [ 0.92 -0.99 -0.54] [ 0.59  0.41  0.  ]
A [ 0.92 -0.99 -0.45] [ 0.46  0.54  0.  ]
B [ 0.91 -0.99 -0.33] [ 0.31  0.69  0.  ]
B [-0.5  -0.98  0.38] [ 0.  1.  0.]
B [-0.99 -0.85  0.68] [ 0.  1.  0.]
B [-1.   -0.63  0.75] [ 0.  1.  0.]
B [-1.   -0.48  0.72] [ 0.  1.  0.]
B [-1.   -0.35  0.65] [ 0.  1.  0.]
B [-1.   -0.23  0.51] [ 0.  1.  0.]
B [-1.   -0.09  0.25] [ 0.    0.98  0.02]
C [-1.    0.08 -0.24] [ 0.  0.  1.]
C [-0.6   0.75 -0.98] [ 0.  0.  1.]
C [-0.43  0.88 -1.  ] [ 0.  0.  1.]
C [-0.33  0.88 -1.  ] [ 0.  0.  1.]
C [-0.19  0.87 -1.  ] [ 0.  0.  1.]
C [ 0.03  0.83 -1.  ] [ 0.  0.  1.]
C [ 0.35  0.75 -1.  ] [ 0.  0.  1.]
C [ 0.72  0.55 -1.  ] [ 0.01  0.    0.99]
A [ 0.92  0.11 -0.99] [ 0.98  0.    0.01]
A [ 0.64 -0.91 -0.96] [ 0.86  0.14  0.  ]
A [ 0.85 -0.98 -0.87] [ 0.86  0.14  0.  ]
B [ 0.92 -0.99 -0.78] [ 0.83  0.17  0.  ]
B [-0.39 -0.98 -0.29] [ 0.  1.  0.]
B [-0.99 -0.85 -0.15] [ 0.    0.97  0.03]
C [-1.   -0.59 -0.31] [ 0.    0.01  0.99]
C [-0.08  0.28 -0.95] [ 0.  0.  1.]
C [ 0.59  0.45 -1.  ] [ 0.  0.  1.]
A [ 0.91  0.12 -0.99] [ 0.98  0.    0.02]
B [ 0.62 -0.91 -0.96] [ 0.85  0.15  0.  ]
C [-0.69 -0.96 -0.65] [ 0.    0.02  0.98]
A [ 0.66 -0.3  -0.96] [ 0.96  0.02  0.02]
epoch: 9
error: 0.0076
```

Part 3.6 - lstm



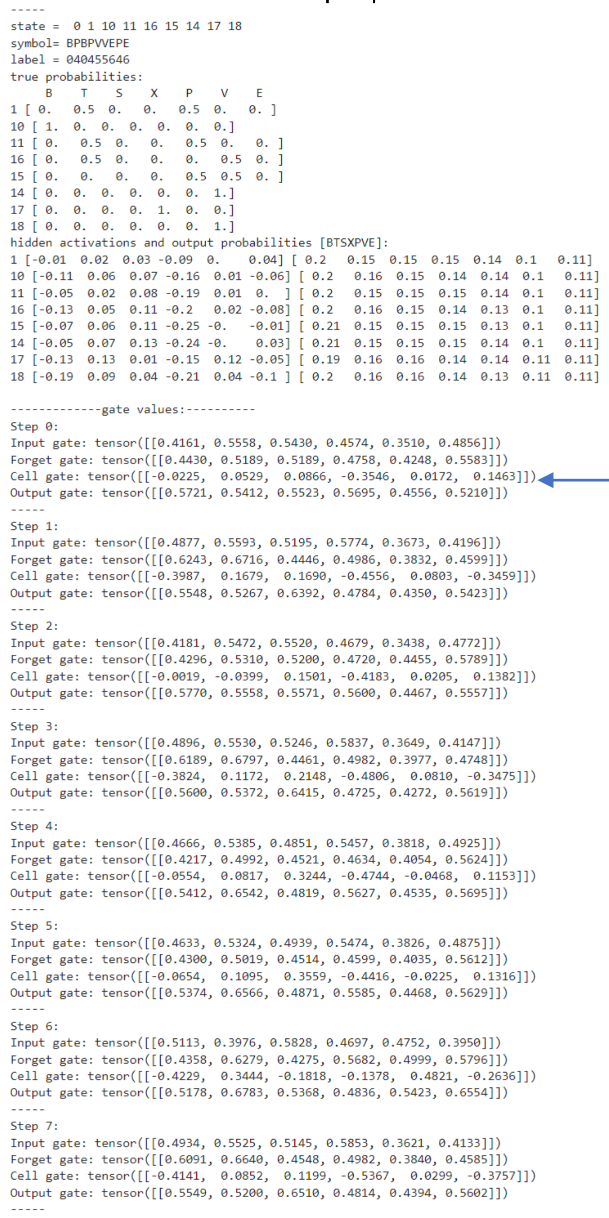
```
%run seq_train.py --lang reber --embed True --model lstm --hid 6
```

```
epoch: 49000
error: 0.0008
final: 0.0000
-----
state =  0 1 2 3 8 7 5 6 9 18
symbol= BTBPVPSETE
label = 0104542616
true probabilities:
     B    T    S    X    P    V    E
1 [ 0.   0.5  0.   0.   0.5  0.   0. ]
2 [ 1.  0.  0.  0.  0.  0.  0.]
3 [ 0.   0.5  0.   0.   0.5  0.   0. ]
8 [ 0.   0.5  0.   0.   0.   0.5  0. ]
7 [ 0.   0.   0.   0.   0.5  0.5  0. ]
5 [ 0.   0.   0.5  0.5  0.   0.   0. ]
6 [ 0.  0.  0.  0.  0.  0.  1.]
9 [ 0.  1.  0.  0.  0.  0.  0.]
18 [ 0.  0.  0.  0.  0.  0.  1.]
hidden activations and output probabilities [BTSXPVE]:
1 [-0.46  0.67  0.25  0.74 -0.54  0.36] [ 0.    0.53  0.    0.    0.47  0.    0.  ]
2 [ 0.42 -0.4   0.82 -0.57 -0.59 -0.62] [ 1.  0.  0.  0.  0.  0.  0.]
3 [-0.48  0.51  0.12  0.73  0.14  0.  ] [ 0.    0.47  0.    0.    0.52  0.    0.  ]
8 [-0.65  0.39 -0.08 -0.73 -0.62 -0.5 ] [ 0.    0.41  0.    0.    0.    0.58  0.  ]
7 [-0.78  0.95 -0.25 -0.93  0.02  0.75] [ 0.    0.01  0.    0.    0.4   0.59  0.  ]
5 [ 0.57  0.03  0.49 -0.87  0.81 -0.74] [ 0.    0.01  0.56  0.42  0.    0.    0.  ]
6 [ 0.04  0.39  0.17 -0.93  0.67  0.62] [ 0.    0.    0.    0.    0.    0.    0.99]
9 [-0.67  0.99  0.52 -0.01  0.82 -0.71] [ 0.    0.98  0.    0.    0.01  0.    0.  ]
18 [-0.27 -0.42 -0.29 -0.96  0.63  0.39] [ 0.  0.  0.  0.  0.  0.  1.]
```

- Tested the model through different parameters, and found that **6 hidden nodes** produces an adequate performance level.
- Can see on the left that the model is able to correctly classify the second last symbol (T or P), which should be the same as the second symbol.
- Other iterations of this model (e.g. with 4 nodes only) was not able to capture this complexity, selecting the second last symbol with a close to random probability
- The model is able to better capture the longer term history through the hidden nodes, which include input gates, forget gates, cell gates and an output gate. The model is therefore better able to "forget" or "remember" important data which minimises losses across time, and only this information is retained in future training
- On the next page, we analyse the individual activations for the hidden units to see how these change from early in the training process to the fully trained model

Part 3.6 (cont'd) – lstm hidden unit analysis

Hidden activations and output probabilities – start of modelling process



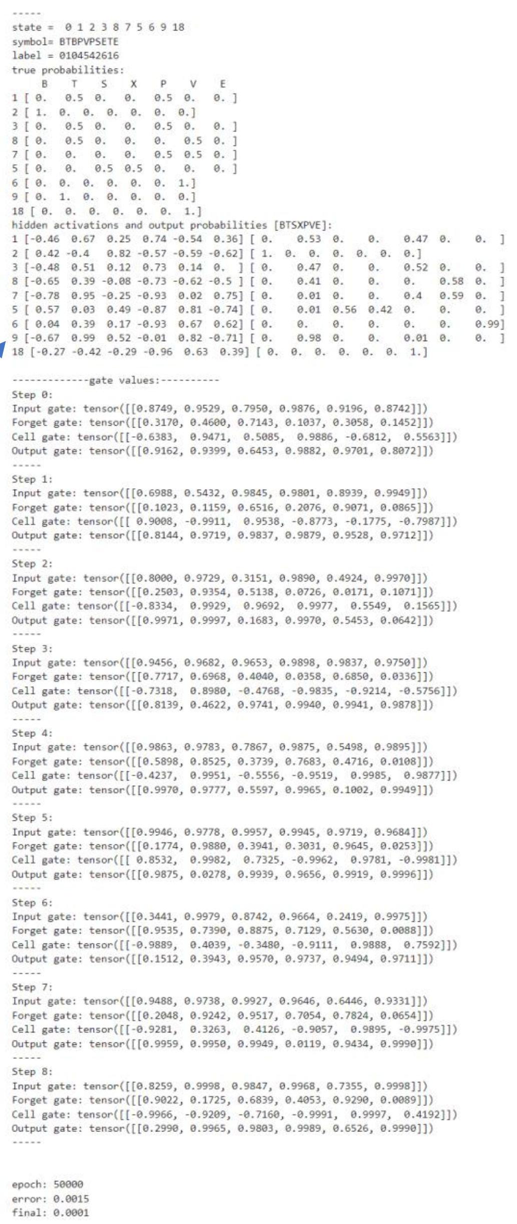
```
-----
state =  0 1 10 11 16 15 14 17 18
symbol= BPBPVVEPE
label = 040455646
true probabilities:
     B    T    S    X    P    V    E
1 [ 0.   0.5  0.   0.   0.5  0.   0. ]
10 [ 1.  0.  0.  0.  0.  0.  0.]
11 [ 0.   0.5  0.   0.   0.5  0.   0. ]
16 [ 0.   0.5  0.   0.   0.   0.5  0. ]
15 [ 0.   0.   0.   0.   0.5  0.5  0. ]
14 [ 0.  0.  0.  0.  0.  0.  1.]
17 [ 0.  0.  0.  0.  1.  0.  0.]
18 [ 0.  0.  0.  0.  0.  0.  1.]
hidden activations and output probabilities [BTSXPVE]:
1 [-0.01  0.02  0.03 -0.09  0.    0.04] [ 0.2   0.15  0.15  0.15  0.14  0.1   0.11]
10 [-0.11  0.06  0.07 -0.16  0.01 -0.06] [ 0.2   0.16  0.15  0.14  0.14  0.1   0.11]
11 [-0.05  0.02  0.08 -0.19  0.01  0.  ] [ 0.2   0.15  0.15  0.15  0.14  0.1   0.11]
16 [-0.13  0.05  0.11 -0.2   0.02 -0.08] [ 0.2   0.16  0.15  0.14  0.13  0.1   0.11]
15 [-0.07  0.06  0.11 -0.25 -0.   -0.01] [ 0.21  0.15  0.15  0.15  0.13  0.1   0.11]
14 [-0.05  0.07  0.13 -0.24 -0.    0.03] [ 0.21  0.15  0.15  0.15  0.14  0.1   0.11]
17 [-0.13  0.13  0.01 -0.15  0.12 -0.05] [ 0.19  0.16  0.16  0.14  0.14  0.11  0.11]
18 [-0.19  0.09  0.04 -0.21  0.04 -0.1 ] [ 0.2   0.16  0.16  0.14  0.13  0.11  0.11]

------------gate values:----------
Step 0:
Input gate: tensor([[0.4161, 0.5558, 0.5430, 0.4574, 0.3510, 0.4856]])
Forget gate: tensor([[0.4430, 0.5189, 0.5189, 0.4758, 0.4248, 0.5583]])
Cell gate: tensor([[-0.0225,  0.0529,  0.0866, -0.3546,  0.0172,  0.1463]])
Output gate: tensor([[0.5721, 0.5412, 0.5523, 0.5695, 0.4556, 0.5210]])
-----
Step 1:
Input gate: tensor([[0.4877, 0.5593, 0.5195, 0.5774, 0.3673, 0.4196]])
Forget gate: tensor([[0.6243, 0.6716, 0.4446, 0.4986, 0.3832, 0.4599]])
Cell gate: tensor([[-0.3987,  0.1679,  0.1690, -0.4556,  0.0803, -0.3459]])
Output gate: tensor([[0.5548, 0.5267, 0.6392, 0.4784, 0.4350, 0.5423]])
-----
Step 2:
Input gate: tensor([[0.4181, 0.5472, 0.5520, 0.4679, 0.3438, 0.4772]])
Forget gate: tensor([[0.4296, 0.5310, 0.5200, 0.4720, 0.4455, 0.5789]])
Cell gate: tensor([[-0.0019, -0.0399,  0.1501, -0.4183,  0.0205,  0.1382]])
Output gate: tensor([[0.5770, 0.5558, 0.5571, 0.5600, 0.4467, 0.5557]])
-----
Step 3:
Input gate: tensor([[0.4896, 0.5530, 0.5246, 0.5837, 0.3649, 0.4147]])
Forget gate: tensor([[0.6189, 0.6797, 0.4461, 0.4982, 0.3977, 0.4748]])
Cell gate: tensor([[-0.3824,  0.1172,  0.2148, -0.4806,  0.0810, -0.3475]])
Output gate: tensor([[0.5600, 0.5372, 0.6415, 0.4725, 0.4272, 0.5619]])
-----
Step 4:
Input gate: tensor([[0.4666, 0.5385, 0.4851, 0.5457, 0.3818, 0.4925]])
Forget gate: tensor([[0.4217, 0.4992, 0.4521, 0.4634, 0.4054, 0.5624]])
Cell gate: tensor([[-0.0554,  0.0817,  0.3244, -0.4744, -0.0468,  0.1153]])
Output gate: tensor([[0.5412, 0.6542, 0.4819, 0.5627, 0.4535, 0.5695]])
-----
Step 5:
Input gate: tensor([[0.4633, 0.5324, 0.4939, 0.5474, 0.3826, 0.4875]])
Forget gate: tensor([[0.4300, 0.5019, 0.4514, 0.4599, 0.4035, 0.5612]])
Cell gate: tensor([[-0.0654,  0.1095,  0.3559, -0.4416, -0.0225,  0.1316]])
Output gate: tensor([[0.5374, 0.6566, 0.4871, 0.5585, 0.4468, 0.5629]])
-----
Step 6:
Input gate: tensor([[0.5113, 0.3976, 0.5828, 0.4697, 0.4752, 0.3950]])
Forget gate: tensor([[0.4358, 0.6279, 0.4275, 0.5682, 0.4999, 0.5796]])
Cell gate: tensor([[-0.4229,  0.3444, -0.1818, -0.1378,  0.4821, -0.2636]])
Output gate: tensor([[0.5178, 0.6783, 0.5368, 0.4836, 0.5423, 0.6554]])
-----
Step 7:
Input gate: tensor([[0.4934, 0.5525, 0.5145, 0.5853, 0.3621, 0.4133]])
Forget gate: tensor([[0.6091, 0.6640, 0.4548, 0.4982, 0.3840, 0.4585]])
Cell gate: tensor([[-0.4141,  0.0852,  0.1199, -0.5367,  0.0299, -0.3757]])
Output gate: tensor([[0.5549, 0.5200, 0.6510, 0.4814, 0.4394, 0.5602]])
-----
```

The table on the left shows unit activations for each of the gate types. While it is difficult to directly interpret the values, we can see at a high level that these are broadly similar across the different layers (we have 6 hidden layers) and across gate type

As the model is trained, these values become a lot more differentiated, suggesting that the model is better retaining and "forgetting" information in a way that helps it improve overall loss

The result can be seen in the output probabilities, where the model is able to much better predict, especially our key area of interest here, which is the second last item, that is directly related to the second item (T or P).

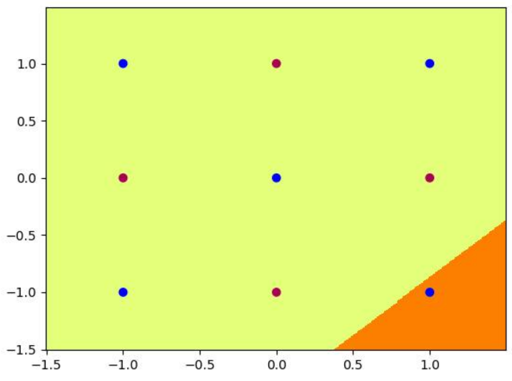
Hidden activations and output probabilities – trained model



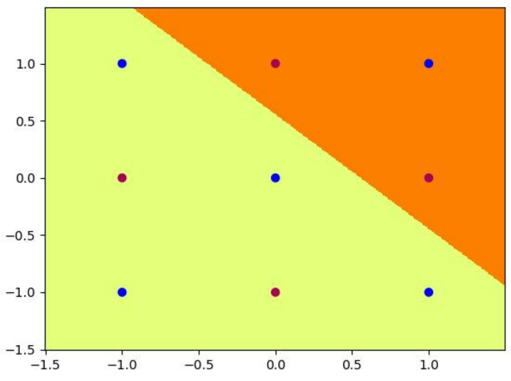
```
-----
state =  0 1 2 3 8 7 5 6 9 18
symbol= BTBPVPSETE
label = 0104542616
true probabilities:
     B    T    S    X    P    V    E
1 [ 0.   0.5  0.   0.   0.5  0.   0. ]
2 [ 1.  0.  0.  0.  0.  0.  0.]
3 [ 0.   0.5  0.   0.   0.5  0.   0. ]
8 [ 0.   0.5  0.   0.   0.   0.5  0. ]
7 [ 0.   0.   0.   0.   0.5  0.5  0. ]
5 [ 0.   0.   0.5  0.5  0.   0.   0. ]
6 [ 0.  0.  0.  0.  0.  0.  1.]
9 [ 0.  1.  0.  0.  0.  0.  0.]
18 [ 0.  0.  0.  0.  0.  0.  1.]
hidden activations and output probabilities [BTSXPVE]:
1 [-0.46  0.67  0.25  0.74 -0.54  0.36] [ 0.    0.53  0.    0.    0.47  0.    0.  ]
2 [ 0.42 -0.4   0.82 -0.57 -0.59 -0.62] [ 1.  0.  0.  0.  0.  0.  0.]
3 [-0.48  0.51  0.12  0.73  0.14  0.  ] [ 0.    0.47  0.    0.    0.52  0.    0.  ]
8 [-0.65  0.39 -0.08 -0.73 -0.62 -0.5 ] [ 0.    0.41  0.    0.    0.    0.58  0.  ]
7 [-0.78  0.95 -0.25 -0.93  0.02  0.75] [ 0.    0.01  0.    0.    0.4   0.59  0.  ]
5 [ 0.57  0.03  0.49 -0.87  0.81 -0.74] [ 0.    0.01  0.56  0.42  0.    0.    0.  ]
6 [ 0.04  0.39  0.17 -0.93  0.67  0.62] [ 0.    0.    0.    0.    0.    0.    0.99]
9 [-0.67  0.99  0.52 -0.01  0.82 -0.71] [ 0.    0.98  0.    0.    0.01  0.    0.  ]
18 [-0.27 -0.42 -0.29 -0.96  0.63  0.39] [ 0.  0.  0.  0.  0.  0.  1.]

------------gate values:----------
Step 0:
Input gate: tensor([[0.8749, 0.9529, 0.7950, 0.9876, 0.9196, 0.8742]])
Forget gate: tensor([[0.3170, 0.4600, 0.7143, 0.1037, 0.3058, 0.1452]])
Cell gate: tensor([[-0.6383,  0.9471,  0.5085,  0.9886, -0.6812,  0.5563]])
Output gate: tensor([[0.9162, 0.9399, 0.6453, 0.9882, 0.9701, 0.8072]])
-----
Step 1:
Input gate: tensor([[0.6988, 0.5432, 0.9845, 0.9801, 0.8939, 0.9949]])
Forget gate: tensor([[0.1023, 0.1159, 0.6516, 0.2076, 0.9071, 0.0865]])
Cell gate: tensor([[ 0.9008, -0.9911,  0.9538, -0.8773, -0.1775, -0.7987]])
Output gate: tensor([[0.8144, 0.9719, 0.9837, 0.9879, 0.9528, 0.9712]])
-----
Step 2:
Input gate: tensor([[0.8000, 0.9729, 0.3151, 0.9890, 0.4924, 0.9970]])
Forget gate: tensor([[0.2503, 0.9354, 0.5138, 0.0726, 0.0171, 0.1071]])
Cell gate: tensor([[-0.8334,  0.9929,  0.9692,  0.9977,  0.5549,  0.1565]])
Output gate: tensor([[0.9971, 0.9997, 0.1683, 0.9970, 0.5453, 0.0642]])
-----
Step 3:
Input gate: tensor([[0.9456, 0.9682, 0.9653, 0.9898, 0.9837, 0.9750]])
Forget gate: tensor([[0.7717, 0.6968, 0.4040, 0.0358, 0.6850, 0.0336]])
Cell gate: tensor([[-0.7318,  0.8980, -0.4768, -0.9835, -0.9214, -0.5756]])
Output gate: tensor([[0.8139, 0.4622, 0.9741, 0.9940, 0.9941, 0.9878]])
-----
Step 4:
Input gate: tensor([[0.9863, 0.9783, 0.7867, 0.9875, 0.5498, 0.9895]])
Forget gate: tensor([[0.5898, 0.8525, 0.3739, 0.7683, 0.4716, 0.0108]])
Cell gate: tensor([[-0.4237,  0.9951, -0.5556, -0.9519,  0.9985,  0.9877]])
Output gate: tensor([[0.9970, 0.9777, 0.5597, 0.9965, 0.1002, 0.9949]])
-----
Step 5:
Input gate: tensor([[0.9946, 0.9778, 0.9957, 0.9945, 0.9719, 0.9684]])
Forget gate: tensor([[0.1774, 0.9880, 0.3941, 0.3031, 0.9645, 0.0253]])
Cell gate: tensor([[ 0.8532,  0.9982,  0.7325, -0.9962,  0.9781, -0.9981]])
Output gate: tensor([[0.9875, 0.0278, 0.9939, 0.9656, 0.9919, 0.9996]])
-----
Step 6:
Input gate: tensor([[0.3441, 0.9979, 0.8742, 0.9664, 0.2419, 0.9975]])
Forget gate: tensor([[0.9535, 0.7390, 0.8875, 0.7129, 0.5630, 0.0088]])
Cell gate: tensor([[-0.9889,  0.4039, -0.3480, -0.9111,  0.9888,  0.7592]])
Output gate: tensor([[0.1512, 0.3943, 0.9570, 0.9737, 0.9494, 0.9711]])
-----
Step 7:
Input gate: tensor([[0.9488, 0.9738, 0.9927, 0.9646, 0.6446, 0.9331]])
Forget gate: tensor([[0.2048, 0.9242, 0.9517, 0.7054, 0.7824, 0.0654]])
Cell gate: tensor([[-0.9281,  0.3263,  0.4126, -0.9057,  0.9895, -0.9975]])
Output gate: tensor([[0.9959, 0.9950, 0.9949, 0.0119, 0.9434, 0.9990]])
-----
Step 8:
Input gate: tensor([[0.8259, 0.9998, 0.9847, 0.9968, 0.7355, 0.9998]])
Forget gate: tensor([[0.9022, 0.1725, 0.6839, 0.4053, 0.9290, 0.0089]])
Cell gate: tensor([[-0.9966, -0.9209, -0.7160, -0.9991,  0.9997,  0.4192]])
Output gate: tensor([[0.2990, 0.9965, 0.9803, 0.9989, 0.6526, 0.9990]])
-----


epoch: 50000
error: 0.0015
final: 0.0001
```

Part 2: Multi-Layer Perceptron

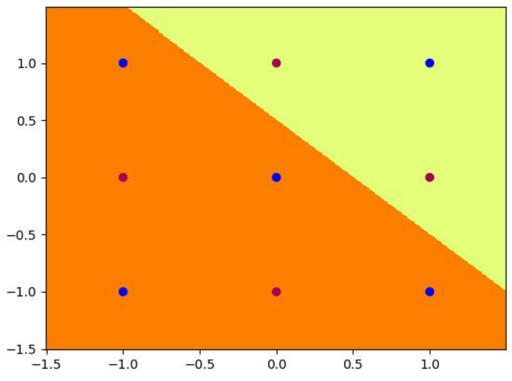
Part 2, Step 1. `python3 check_main.py --act sig --hid 5`



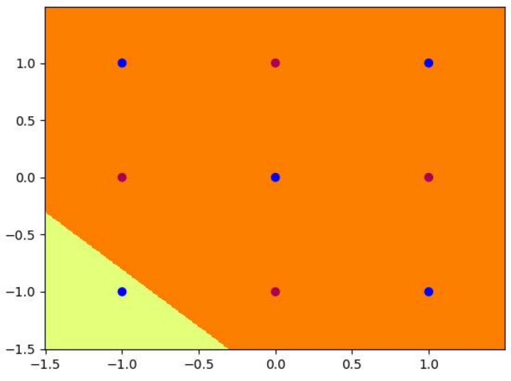
Hidden Unit 1



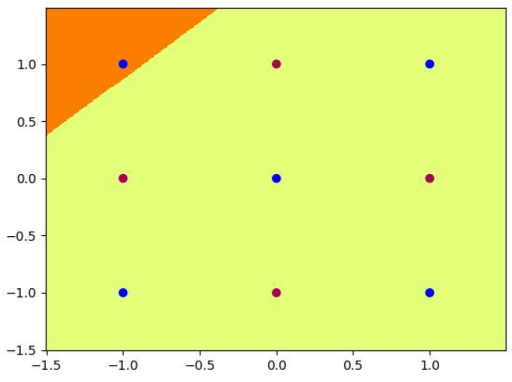
Hidden Unit 2



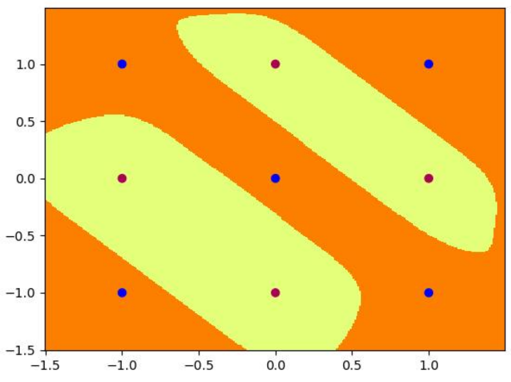
Hidden Unit 3



Hidden Unit 4



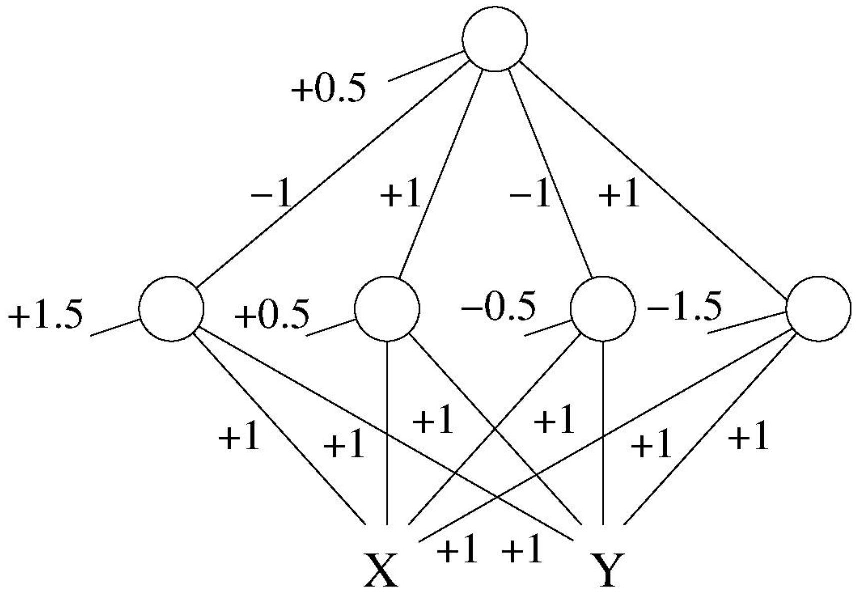
Hidden Unit 5



Output Unit

Part 2, Step 2.

Design by hand a 2-layer neural network with 4 hidden nodes, using the Heaviside (step) activation function at both the hidden and output layer, which correctly classifies the data.



Write the equations for the dividing line determined by each hidden node.

Hidden Unit 1:	$X + Y + 1.5 > 0$
Hidden Unit 2:	$X + Y + 0.5 > 0$
Hidden Unit 3:	$X + Y - 0.5 > 0$
Hidden Unit 4:	$X + Y - 1.5 > 0$

Create a table showing the activations of all the hidden nodes and the output node, for each of the 9 training items, and include it in your report.

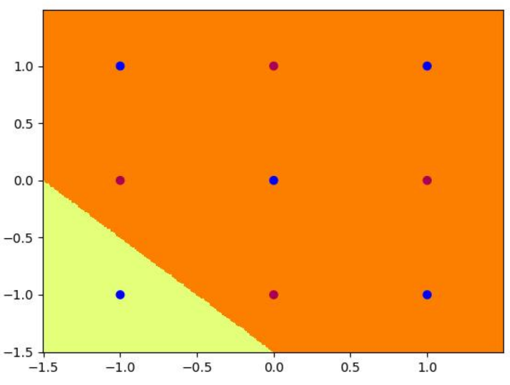
X	Y	H1	H2	H3	H4	Out
-1	-1	0	0	0	0	1
-1	0	1	0	0	0	0
-1	1	1	1	0	0	1
0	-1	1	0	0	0	0
0	0	1	1	0	0	1
0	1	1	1	1	0	0
1	-1	1	1	0	0	1
1	0	1	1	1	0	0
1	1	1	1	1	1	1

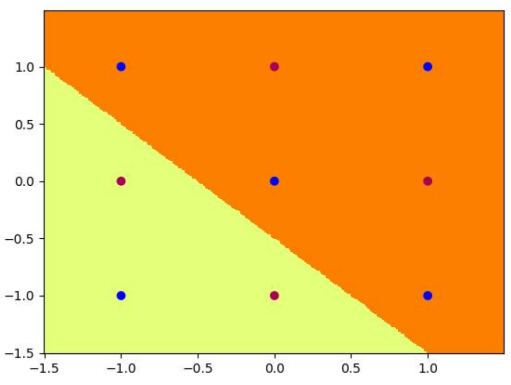
Part 2, Step 3.

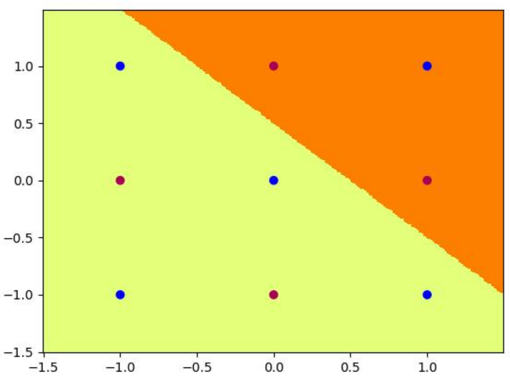
Now rescale your hand-crafted weights from Part 2 by multiplying all of them by a large (fixed) number (for example, 10) so that the combination of rescaling followed by sigmoid will mimic the effect of the step function.

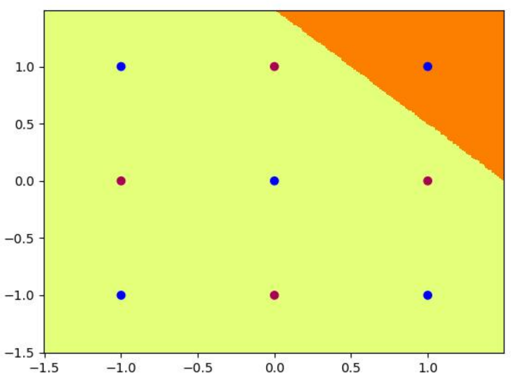
```
in_hid_weight  = [[10,10],[10,10],[10,10],[10,10]]
hid_bias       = [15,5,-5,-15]
hid_out_weight = [[-10,10,-10,10]]
out_bias       = [5]
```

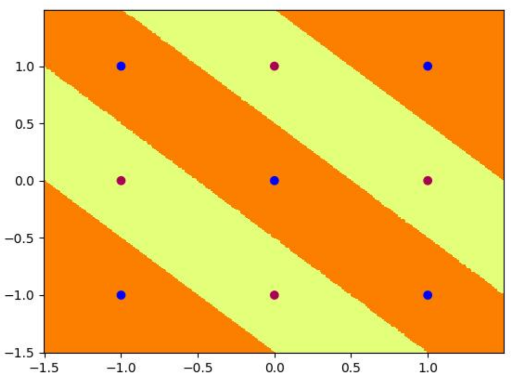
```
python3 check_main.py --act sig --hid 4 --set_weights
```


Hidden Unit 1


Hidden Unit 2


Hidden Unit 3


Hidden Unit 4


Output Unit