# CSE583/EE552 Pattern Recognition and Machine Learning: Project #4 Part 2

Due on April 15th, 2022 at 11:59pm

*PROFESSOR Yanxi Liu Spring 22*

**Anish Phule** **asp5607@psu.edu**

This report consists of:

**Problem 1 - Deep Q-Learning**:

- List of Python Libraries used in this project
- General approach to the problem
- Problems encountered
- Results and explanation of figures
- Effect of parameters on learning
- Suggestions/comments/concerns about this project
- Extra Credit

## Problem 1

## Deep Q-Learning

### 1. List of Python Libraries used in this project

- numpy as np
- Matplotlib.pyplot as plt
- keras and tensorflow sublibraries
- from keras.models : Sequential
- from keras.layers.core : Dense, Activation, for model layers
- from tensorflow.keras.optimizers : Adam, optimizer for our model

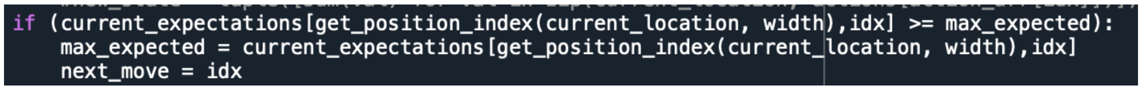
### 2. General approach to the problem

The problem was solved in two parts, modifying the original code and then adding the deep learning part.

I original was doing part 2 in MatLab, since I did project 1 in MatLab as well. However, since I had a hard time figuring out the deep learning part in MatLab, I decided to switch to python. I have done the non-deep learning part in both languages, and am including them in the project zip file.

Preparing for Deep Q Learning:

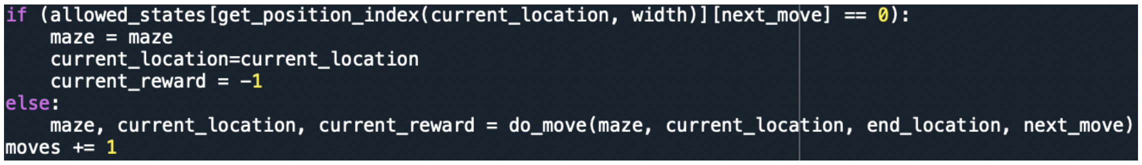
- We first start by reworking our current expectations. So previously, our agent used to look at its neighbours and choose the max expectation amongst them to move. Here, we are storing expectations for every action at every state, and our agent looks at its actions and their expectations, and chooses their max to move.



```
if (current_expectations[get_position_index(current_location, width),idx] >= max_expected):
    max_expected = current_expectations[get_position_index(current_location, width),idx]
    next_move = idx
```

Figure 1: Modifying current expectations, from Q(s) to Q(s,a).

- We also make changes to state history. We now store previous state, action and reward, since we are also tracking actions now.
- We allow our agents to attempt to make invalid moves(i.e. walk into walls), but we don't make any actual actions upon that.



```
if (allowed_states[get_position_index(current_location, width)][next_move] == 0):
    maze = maze
    current_location=current_location
    current_reward = -1
else:
    maze, current_location, current_reward = do_move(maze, current_location, end_location, next_move)
moves += 1
```

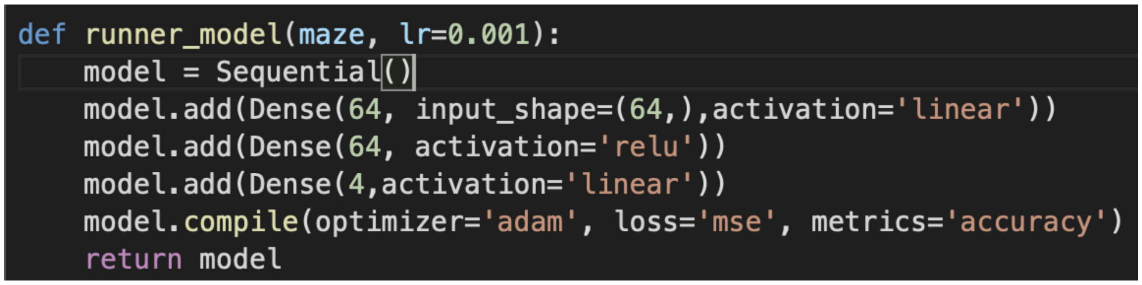
Figure 2: Allowing invalid moves

Problem 1 continued on next page. . .

Now we are ready to implement Deep Q learning.

Deep Q-Learning based:

- We first build our model. The model is as follows:

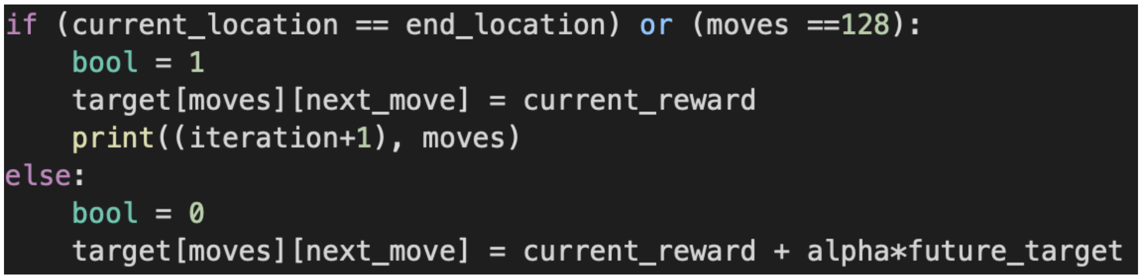


```
def runner_model(maze, lr=0.001):
    model = Sequential()
    model.add(Dense(64, input_shape=(64,),activation='linear'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(4,activation='linear'))
    model.compile(optimizer='adam', loss='mse', metrics='accuracy')
    return model
```

Figure 3: Deep learning model with LR = 1e-3

The model takes in a 64 length vector as input, followed by a fully connected layer, and gives out a 4 length predicted vector corresponding our 4 actions. These will be used in our program.

- Since the deep learning model learns and predicts its own values, we no longer have use for the Q-table. We let our model predict the 4 length vector, a target vector of expectations. based on this vector, we take the maximum value, and pass on that value's index as the next action. We also get a future target based on the max value of the prediction based on the current state. The target vector is then appended as follows:



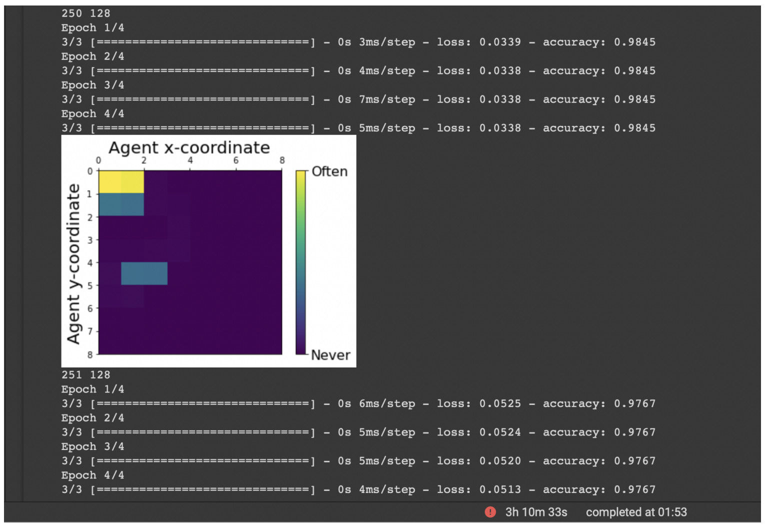
```
if (current_location == end_location) or (moves ==128):
    bool = 1
    target[moves][next_move] = current_reward
    print((iteration+1), moves)
else:
    bool = 0
    target[moves][next_move] = current_reward + alpha*future_target
```

Figure 4: Training the model

- We also bring down the number of moves and change the value of alpha for the learning.

## 3. Problems encountered

- Model getting stuck in corners and getting secluded: Since we added the condition of invalid moves, the model had a habit of being stuck in corners and also sometimes in walls. I tried making the reward more negative for that, but I observed the model just being afraid to explore and being secluded to the top left 4x4 square. So, I brought the rewards back to -1, but removed the invalid moves condition just on exploration. It is still applicable for exploitation, and helps the model explore more and learn.



```
250 128
Epoch 1/4
3/3 [==============================] - 0s 3ms/step - loss: 0.0339 - accuracy: 0.9845
Epoch 2/4
3/3 [==============================] - 0s 4ms/step - loss: 0.0338 - accuracy: 0.9845
Epoch 3/4
3/3 [==============================] - 0s 7ms/step - loss: 0.0338 - accuracy: 0.9845
Epoch 4/4
3/3 [==============================] - 0s 5ms/step - loss: 0.0338 - accuracy: 0.9845
```

```
251 128
Epoch 1/4
3/3 [==============================] - 0s 6ms/step - loss: 0.0525 - accuracy: 0.9767
Epoch 2/4
3/3 [==============================] - 0s 5ms/step - loss: 0.0524 - accuracy: 0.9767
Epoch 3/4
3/3 [==============================] - 0s 5ms/step - loss: 0.0520 - accuracy: 0.9767
Epoch 4/4
3/3 [==============================] - 0s 4ms/step - loss: 0.0513 - accuracy: 0.9767
```

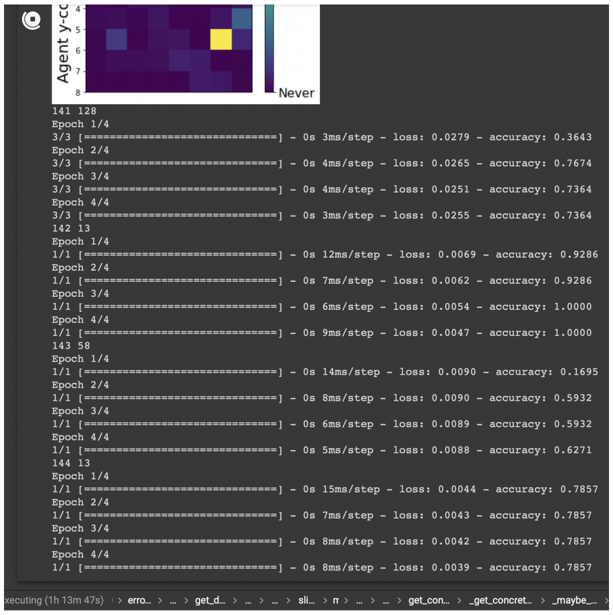
3h 10m 33s completed at 01:53

Figure 5: Model stuck at corners after high training times and multiple epochs. It gradually gets more and more secluded.

- High training times:I was observing high training times, with every iteration taking about 1.5-2 minutes. This was not reasonable given the number of iterations we wish to run. Turns out, the activation layers play an important role in this. I previously did not have linear activations set, and that slowed the model down.

## 4. Results and explanation of figures

The maze runner went around for many iterations exploring and exploiting. At the start of training, the number of moves were consistently around 128, although every once in a while, there would be an attempt with half or less than half of 128 moves. At around 140 iterations, the model started showing signs of convergence.

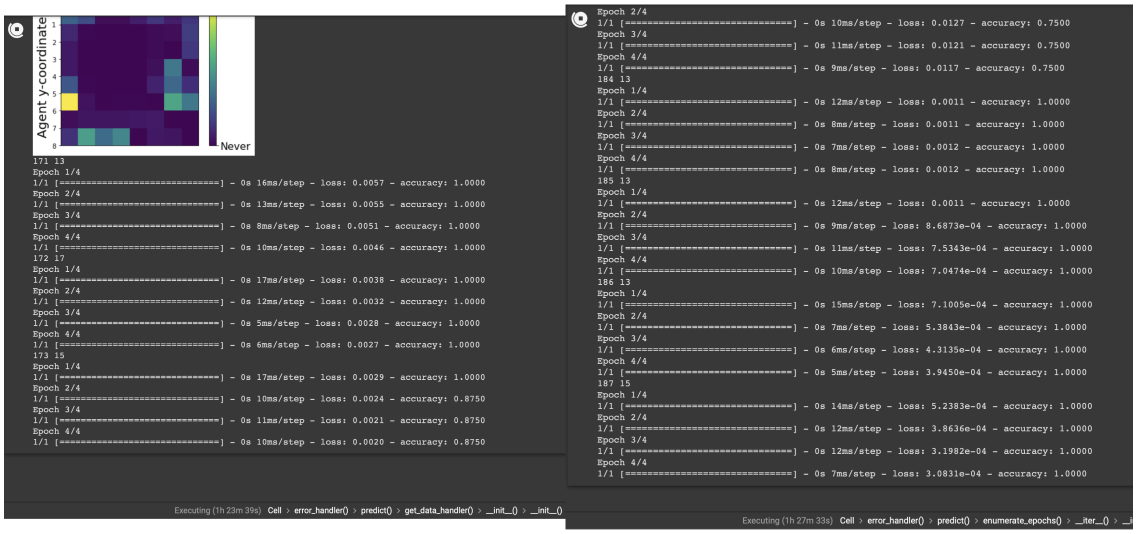


```
141 128
Epoch 1/4
3/3 [==============================] - 0s 3ms/step - loss: 0.0279 - accuracy: 0.3643
Epoch 2/4
3/3 [==============================] - 0s 4ms/step - loss: 0.0265 - accuracy: 0.7674
Epoch 3/4
3/3 [==============================] - 0s 4ms/step - loss: 0.0251 - accuracy: 0.7364
Epoch 4/4
3/3 [==============================] - 0s 3ms/step - loss: 0.0255 - accuracy: 0.7364
142 13
Epoch 1/4
1/1 [==============================] - 0s 12ms/step - loss: 0.0069 - accuracy: 0.9286
Epoch 2/4
1/1 [==============================] - 0s 7ms/step - loss: 0.0062 - accuracy: 0.9286
Epoch 3/4
1/1 [==============================] - 0s 6ms/step - loss: 0.0054 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 9ms/step - loss: 0.0047 - accuracy: 1.0000
143 58
Epoch 1/4
1/1 [==============================] - 0s 14ms/step - loss: 0.0090 - accuracy: 0.1695
Epoch 2/4
1/1 [==============================] - 0s 8ms/step - loss: 0.0090 - accuracy: 0.5932
Epoch 3/4
1/1 [==============================] - 0s 6ms/step - loss: 0.0089 - accuracy: 0.5932
Epoch 4/4
1/1 [==============================] - 0s 5ms/step - loss: 0.0088 - accuracy: 0.6271
144 13
Epoch 1/4
1/1 [==============================] - 0s 15ms/step - loss: 0.0044 - accuracy: 0.7857
Epoch 2/4
1/1 [==============================] - 0s 7ms/step - loss: 0.0043 - accuracy: 0.7857
Epoch 3/4
1/1 [==============================] - 0s 8ms/step - loss: 0.0042 - accuracy: 0.7857
Epoch 4/4
1/1 [==============================] - 0s 8ms/step - loss: 0.0039 - accuracy: 0.7857
```

xecuting (1h 13m 47s) › erro... › ... › get_d... › ... › ... › sli... › π › ... › ... › get_con... › _get_concret... › _maybe_... ›

Figure 6: Model starting to converge

In the 170s and 180s, the model showed consistent moves of around 13-15 per iteration.



```
171 13
Epoch 1/4
1/1 [==============================] - 0s 16ms/step - loss: 0.0057 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 13ms/step - loss: 0.0055 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 8ms/step - loss: 0.0051 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 10ms/step - loss: 0.0046 - accuracy: 1.0000
172 17
Epoch 1/4
1/1 [==============================] - 0s 17ms/step - loss: 0.0038 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 12ms/step - loss: 0.0032 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 5ms/step - loss: 0.0028 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 6ms/step - loss: 0.0027 - accuracy: 1.0000
173 15
Epoch 1/4
1/1 [==============================] - 0s 17ms/step - loss: 0.0029 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 10ms/step - loss: 0.0024 - accuracy: 0.8750
Epoch 3/4
1/1 [==============================] - 0s 11ms/step - loss: 0.0021 - accuracy: 0.8750
Epoch 4/4
1/1 [==============================] - 0s 10ms/step - loss: 0.0020 - accuracy: 0.8750

Executing (1h 23m 39s)  Cell > error_handler() > predict() > get_data_handler() > __init__() > __init__()
```

```
Epoch 2/4
1/1 [==============================] - 0s 10ms/step - loss: 0.0127 - accuracy: 0.7500
Epoch 3/4
1/1 [==============================] - 0s 11ms/step - loss: 0.0121 - accuracy: 0.7500
Epoch 4/4
1/1 [==============================] - 0s 9ms/step - loss: 0.0117 - accuracy: 0.7500
184 13
Epoch 1/4
1/1 [==============================] - 0s 12ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 8ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 7ms/step - loss: 0.0012 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 8ms/step - loss: 0.0012 - accuracy: 1.0000
185 13
Epoch 1/4
1/1 [==============================] - 0s 12ms/step - loss: 0.0011 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 9ms/step - loss: 8.6873e-04 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 11ms/step - loss: 7.5343e-04 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 10ms/step - loss: 7.0474e-04 - accuracy: 1.0000
186 13
Epoch 1/4
1/1 [==============================] - 0s 15ms/step - loss: 7.1005e-04 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 7ms/step - loss: 5.3843e-04 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 6ms/step - loss: 4.3135e-04 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 5ms/step - loss: 3.9450e-04 - accuracy: 1.0000
187 15
Epoch 1/4
1/1 [==============================] - 0s 14ms/step - loss: 5.2383e-04 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 12ms/step - loss: 3.8636e-04 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 12ms/step - loss: 3.1982e-04 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 7ms/step - loss: 3.0831e-04 - accuracy: 1.0000

Executing (1h 27m 33s)  Cell > error_handler() > predict() > enumerate_epochs() > __iter__() > __i
```

Figure 7: Consistent decrease in moves

But the model still hasn't converged yet. Every once in a while, the model would show 128 moves, indication it was still exploring a bit.

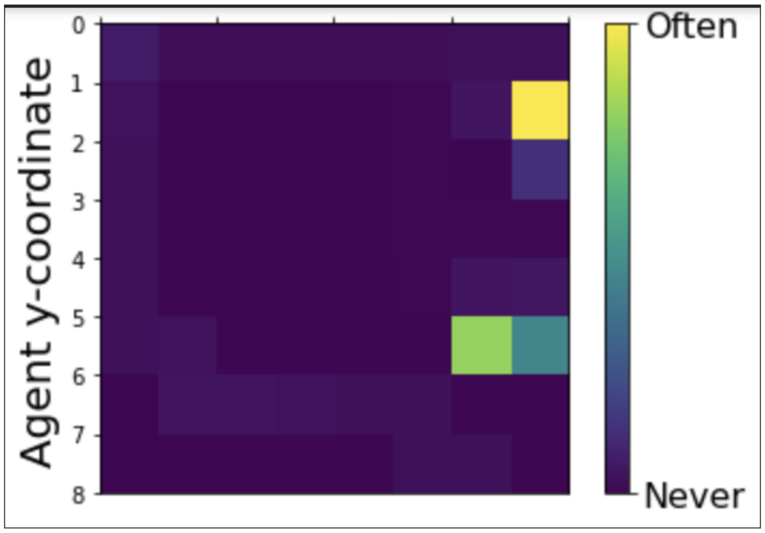
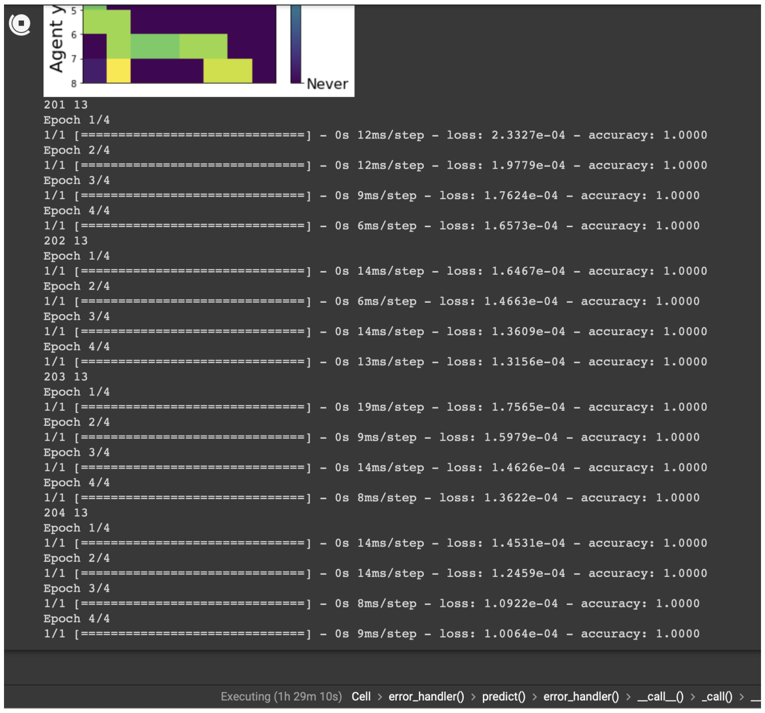


Figure 8: Model still hasn't actually converged.

At around 200-250 iterations, the model finally converges, with consistent 13-15 moves. Even the accuracy as seen in fig. 9 is 1.00 consistently. Fig 10 shows the converged path.


Agent y
Never
Executing (1h 29m 10s) Cell > error_handler() > predict() > error_handler() > __call__() > _call() >

```
201 13
Epoch 1/4
1/1 [==============================] - 0s 12ms/step - loss: 2.3327e-04 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 12ms/step - loss: 1.9779e-04 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 9ms/step - loss: 1.7624e-04 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 6ms/step - loss: 1.6573e-04 - accuracy: 1.0000
202 13
Epoch 1/4
1/1 [==============================] - 0s 14ms/step - loss: 1.6467e-04 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 6ms/step - loss: 1.4663e-04 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 14ms/step - loss: 1.3609e-04 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 13ms/step - loss: 1.3156e-04 - accuracy: 1.0000
203 13
Epoch 1/4
1/1 [==============================] - 0s 19ms/step - loss: 1.7565e-04 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 9ms/step - loss: 1.5979e-04 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 14ms/step - loss: 1.4626e-04 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 8ms/step - loss: 1.3622e-04 - accuracy: 1.0000
204 13
Epoch 1/4
1/1 [==============================] - 0s 14ms/step - loss: 1.4531e-04 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 14ms/step - loss: 1.2459e-04 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 8ms/step - loss: 1.0922e-04 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 9ms/step - loss: 1.0064e-04 - accuracy: 1.0000
```

Figure 9: Model finally converges.

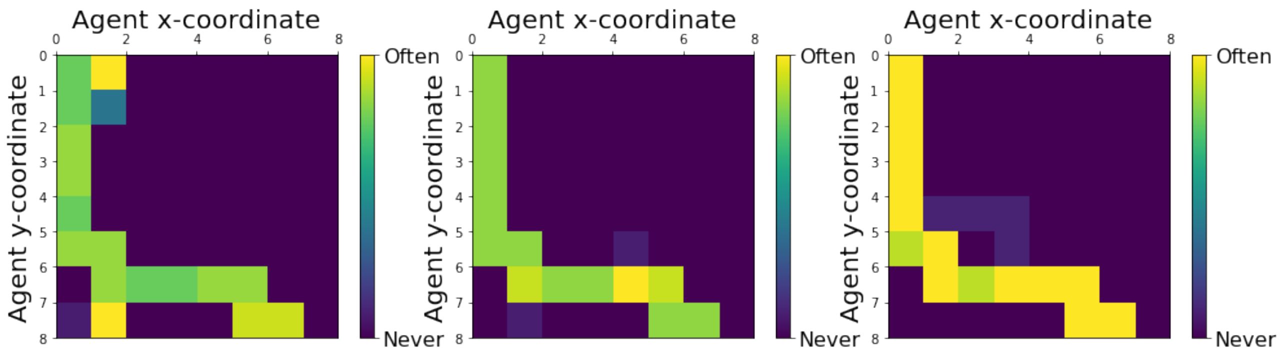


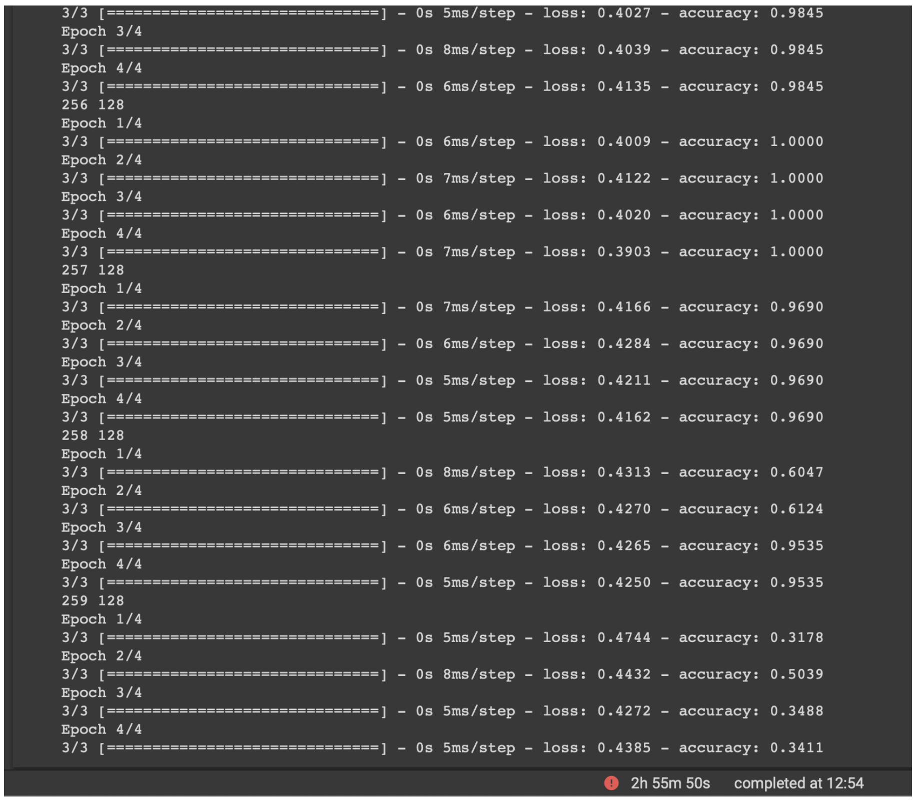
Figure 10: Model converged.

The above task took about 1 hour and 40 minutes before I decided to interrupt the execution as the model had converged.

**5. Effect of changing parameters(alpha):**

My model runs with $\alpha$=0.8. Increase it from the original 0.1 definitely makes the training fast. However, in my case, increasing the value of alpha further had adverse effects on the model training and time. I set $\alpha$=0.95, and the model did not even come close to convergence near 200 iterations. Moreover, it kept being stuck at walls and corners and only going back and forth.

Problem 1 continued on next page. . .



```
3/3 [==============================] - 0s 5ms/step - loss: 0.4027 - accuracy: 0.9845
Epoch 3/4
3/3 [==============================] - 0s 8ms/step - loss: 0.4039 - accuracy: 0.9845
Epoch 4/4
3/3 [==============================] - 0s 6ms/step - loss: 0.4135 - accuracy: 0.9845
256 128
Epoch 1/4
3/3 [==============================] - 0s 6ms/step - loss: 0.4009 - accuracy: 1.0000
Epoch 2/4
3/3 [==============================] - 0s 7ms/step - loss: 0.4122 - accuracy: 1.0000
Epoch 3/4
3/3 [==============================] - 0s 6ms/step - loss: 0.4020 - accuracy: 1.0000
Epoch 4/4
3/3 [==============================] - 0s 7ms/step - loss: 0.3903 - accuracy: 1.0000
257 128
Epoch 1/4
3/3 [==============================] - 0s 7ms/step - loss: 0.4166 - accuracy: 0.9690
Epoch 2/4
3/3 [==============================] - 0s 6ms/step - loss: 0.4284 - accuracy: 0.9690
Epoch 3/4
3/3 [==============================] - 0s 5ms/step - loss: 0.4211 - accuracy: 0.9690
Epoch 4/4
3/3 [==============================] - 0s 5ms/step - loss: 0.4162 - accuracy: 0.9690
258 128
Epoch 1/4
3/3 [==============================] - 0s 8ms/step - loss: 0.4313 - accuracy: 0.6047
Epoch 2/4
3/3 [==============================] - 0s 6ms/step - loss: 0.4270 - accuracy: 0.6124
Epoch 3/4
3/3 [==============================] - 0s 6ms/step - loss: 0.4265 - accuracy: 0.9535
Epoch 4/4
3/3 [==============================] - 0s 5ms/step - loss: 0.4250 - accuracy: 0.9535
259 128
Epoch 1/4
3/3 [==============================] - 0s 5ms/step - loss: 0.4744 - accuracy: 0.3178
Epoch 2/4
3/3 [==============================] - 0s 8ms/step - loss: 0.4432 - accuracy: 0.5039
Epoch 3/4
3/3 [==============================] - 0s 5ms/step - loss: 0.4272 - accuracy: 0.3488
Epoch 4/4
3/3 [==============================] - 0s 5ms/step - loss: 0.4385 - accuracy: 0.3411
```

Figure 11: Model did not converge after 3 hours of training time, with increased alpha value of 0.95.

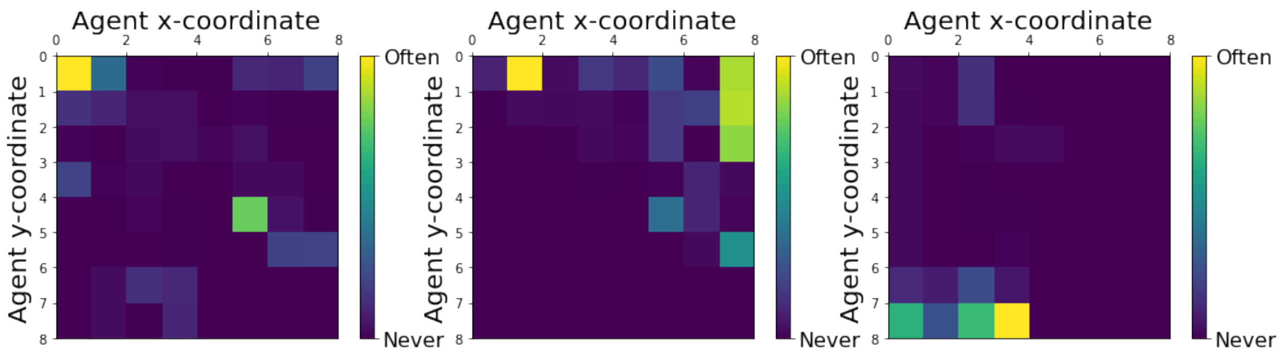


Figure 12: Agent keeps running into and is stuck at corners and walls

**6. Suggestions/comments/concerns about this project :**

Suggestion: I would suggest an emphasis on the activations of the layers in the model. I saw a very slow model when proper activations weren't present, and changing them to better ones drastically increased the performance.
This project is better done on Python than Matlab. Although it can definitely be done on matlab, it is much easier on python and also deep learning implementations are more popular and available on python.

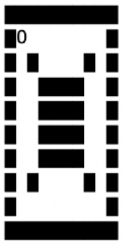
Comments: This project would be easier of the student has some pre-requisite knowledge to Reinforcement learning. It is also however, really helpful in learning this important concept. It helped me getting introduced

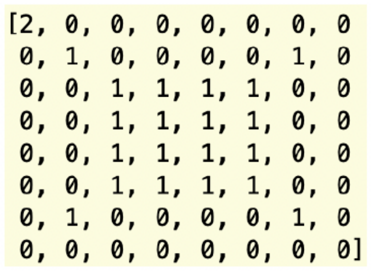
Problem 1 continued on next page. . .

to and learning RL. It is by no means an easy project, but an educating one.

**6. Extra Credit:**

(a) New Maze design.



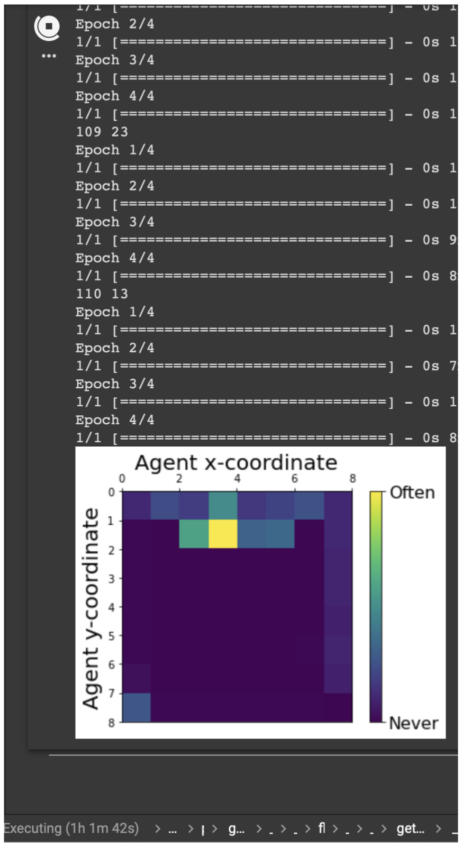


```
[2, 0, 0, 0, 0, 0, 0, 0
 0, 1, 0, 0, 0, 0, 1, 0
 0, 0, 1, 1, 1, 1, 0, 0
 0, 0, 1, 1, 1, 1, 0, 0
 0, 0, 1, 1, 1, 1, 0, 0
 0, 0, 1, 1, 1, 1, 0, 0
 0, 1, 0, 0, 0, 0, 1, 0
 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure 13: New maze design, with centre blocked, helping training become faster.

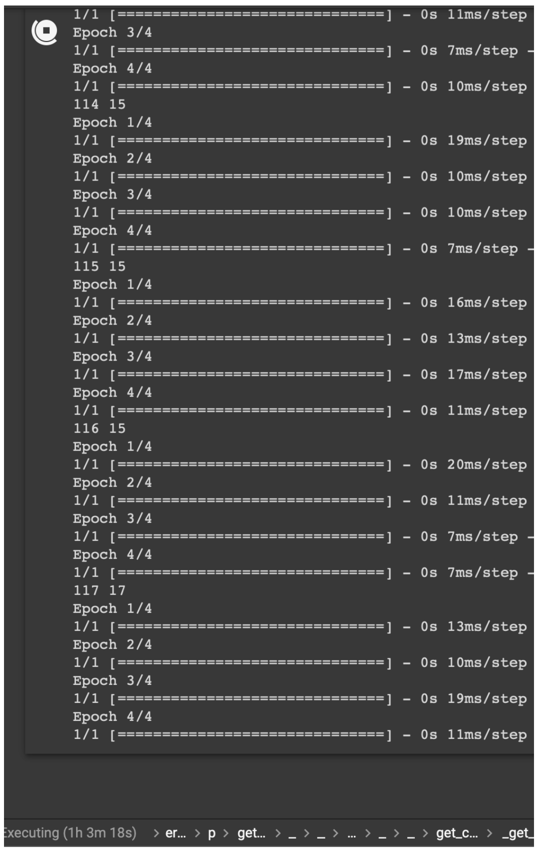
We have a new maze design here, with the centre portion blocked. This serves two purposes. Our Neural network will train and therefore converge faster, and since it collides with so many walls, it is interesting to see how our agent reacts.

The agent initially is lost, as it doesn't have a lot of places to go to, and so many walls. However, it soon converges.



```
Epoch 2/4
1/1 [==============================] - 0s 1
Epoch 3/4
1/1 [==============================] - 0s 1
Epoch 4/4
1/1 [==============================] - 0s 1
109 23
Epoch 1/4
1/1 [==============================] - 0s 1
Epoch 2/4
1/1 [==============================] - 0s 1
Epoch 3/4
1/1 [==============================] - 0s 9
Epoch 4/4
1/1 [==============================] - 0s 8
110 13
Epoch 1/4
1/1 [==============================] - 0s 1
Epoch 2/4
1/1 [==============================] - 0s 7
Epoch 3/4
1/1 [==============================] - 0s 1
Epoch 4/4
1/1 [==============================] - 0s 8
```

Executing (1h 1m 42s)



```
1/1 [==============================] - 0s 11ms/step
Epoch 3/4
1/1 [==============================] - 0s 7ms/step -
Epoch 4/4
1/1 [==============================] - 0s 10ms/step
114 15
Epoch 1/4
1/1 [==============================] - 0s 19ms/step
Epoch 2/4
1/1 [==============================] - 0s 10ms/step
Epoch 3/4
1/1 [==============================] - 0s 10ms/step
Epoch 4/4
1/1 [==============================] - 0s 7ms/step -
115 15
Epoch 1/4
1/1 [==============================] - 0s 16ms/step
Epoch 2/4
1/1 [==============================] - 0s 13ms/step
Epoch 3/4
1/1 [==============================] - 0s 17ms/step
Epoch 4/4
1/1 [==============================] - 0s 11ms/step
116 15
Epoch 1/4
1/1 [==============================] - 0s 20ms/step
Epoch 2/4
1/1 [==============================] - 0s 11ms/step
Epoch 3/4
1/1 [==============================] - 0s 7ms/step -
Epoch 4/4
1/1 [==============================] - 0s 7ms/step -
117 17
Epoch 1/4
1/1 [==============================] - 0s 13ms/step
Epoch 2/4
1/1 [==============================] - 0s 10ms/step
Epoch 3/4
1/1 [==============================] - 0s 19ms/step
Epoch 4/4
1/1 [==============================] - 0s 11ms/step
```

Executing (1h 3m 18s) > er... > p > get... > _ > _ > ... > _ > _ > get_c... > _get_

Figure 14: Model converges fairly early, at around 120 iterations.

Problem 1 continued on next page. . .

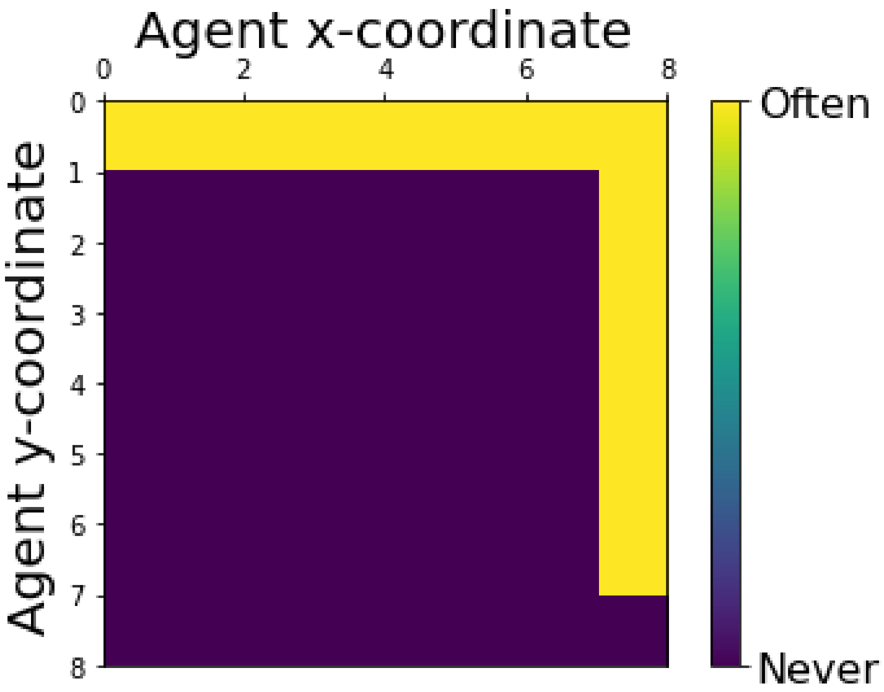
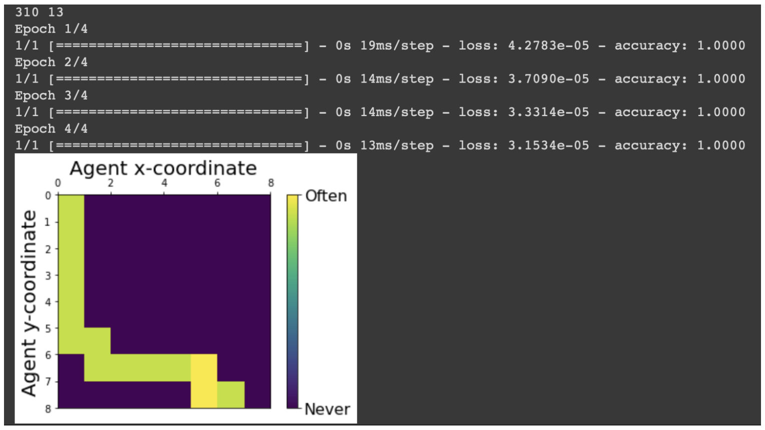


Figure 15: Final solved maze

(b)Change in parameter: Random Factor:

The main project and the above extra credits were all with a random factor of value 0.5. Now we change the random factor.

- Higher: Random Factor = 0.75
  Higher random factor allows the agent to do more rounds of exploration around the maze. This also means that the neural network will move around more and take more time to converge. The following image displays the same.



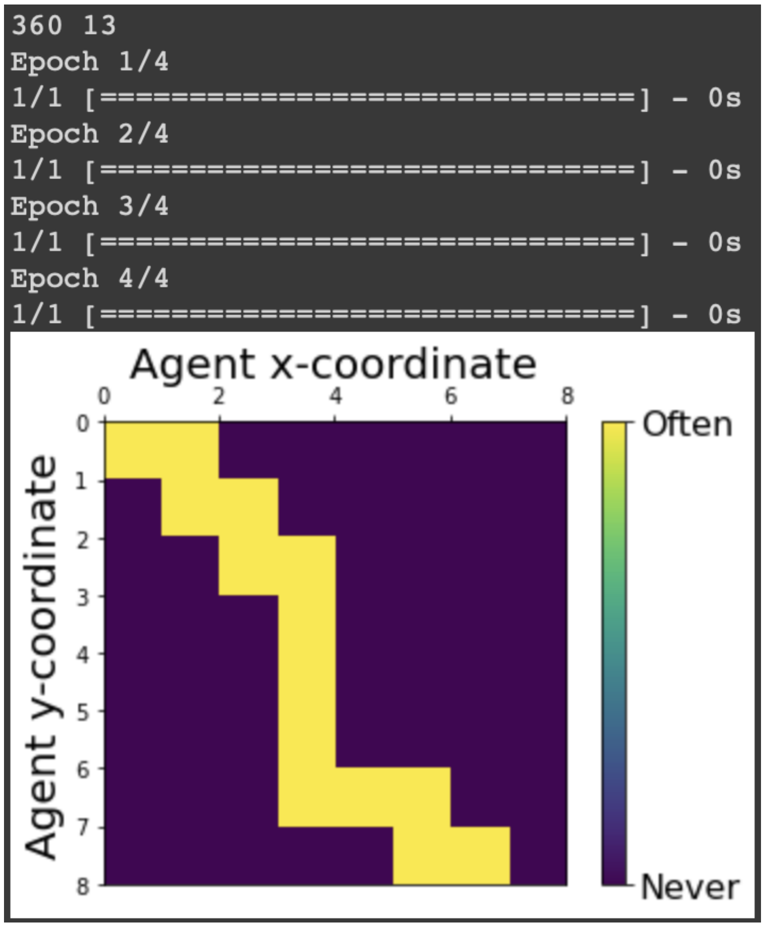
```
310 13
Epoch 1/4
1/1 [==============================] - 0s 19ms/step - loss: 4.2783e-05 - accuracy: 1.0000
Epoch 2/4
1/1 [==============================] - 0s 14ms/step - loss: 3.7090e-05 - accuracy: 1.0000
Epoch 3/4
1/1 [==============================] - 0s 14ms/step - loss: 3.3314e-05 - accuracy: 1.0000
Epoch 4/4
1/1 [==============================] - 0s 13ms/step - loss: 3.1534e-05 - accuracy: 1.0000
```

Figure 16: The model does indeed converge, however with a higher iteration count and longer time(3.5 hrs).

- Lower: Random Factor = 0.3

Problem 1 continued on next page. . .

A lower random factor means the agent doesn't get to do a lot of exploration, hence will rely mostly on exploitation to solve the maze. This is evident from the result below,



```
360 13
Epoch 1/4
1/1 [==============================] - 0s
Epoch 2/4
1/1 [==============================] - 0s
Epoch 3/4
1/1 [==============================] - 0s
Epoch 4/4
1/1 [==============================] - 0s
```

Figure 17: The model does indeed converge, however with a higher iteration count.

What can be inferred from all the results above and in the previous sections as well, that the alpha value and the Random factor all need to be optimised. A very low or high alpha will result in longer train times or problems like getting stuck at walls and corners. Similarly, a low or high random factor will result in what mode the agent is in, thus affecting the model performance.