



School Of Graduate Studies

THE IMITATION GAME

Authors: Utsav Patel (upp10)

Sneh Desai(sd1324)

Manan Vakta(mv651)

Jal Shah(js2985)

Professor: Wes Cowan

CS520: Introduction to Artificial Intelligence

December 2021

Table of Contents

TABLE OF FIGURES.....	3
CHAPTER 1: PROJECT 1- ARCHITECTURE 1(FULL DENSE LAYERS).....	4
Chapter 1.1: Model Structure.....	4
Chapter 1.2: Addressing the Questions.....	5
CHAPTER 2: PROJECT 1- ARCHITECTURE 2(CNN LAYERS).....	11
Chapter 2.1: Model Structure.....	11
Chapter 2.2: Addressing the Questions.....	12
CHAPTER 3: PROJECT 2- ARCHITECTURE 1(FULL DENSE LAYERS).....	17
Chapter 3.1: Model Structure.....	17
Chapter 3.2: Addressing the Questions.....	18
CHAPTER 4: PROJECT 2- ARCHITECTURE 2(CNN LAYERS).....	22
Chapter 4.1: Model Structure.....	22
Chapter 4.2: Addressing the Questions.....	23
CHAPTER 5: PROJECT 3- ARCHITECTURE 1(FULL DENSE LAYERS).....	27
Chapter 5.1: Model Structure.....	27
Chapter 5.2: Addressing the Questions.....	28
CHAPTER 6: PROJECT 3- ARCHITECTURE 2(CNN LAYERS).....	33
Chapter 6.1: Model Structure.....	33
Chapter 6.2: Addressing the Questions.....	34

TABLE OF FIGURES

Figure 1: Project 1 Dense Layer Model	4
Figure 2: Epoch vs Accuracy(Project 1 Dense)	7
Figure 3: Epochs vs Loss(Project 1 Dense)	7
Figure 4: Epochs vs Validation Accuracy	8
Figure 5: Epochs vs Validation Loss.....	9
Figure 6: Box Plot(Project 1 Dense).....	10
Figure 7: Project 1 CNN Layers Model.....	11
Figure 8: Epochs vs Accuracy(Project 1 CNN).....	13
Figure 9: Epochs vs Loss(Project 1 CNN)	13
Figure 10: Epochs vs Validation Accuracy(Project 1 CNN)	15
Figure 11: Epochs vs Validation Loss(Project 1 CNN).....	15
Figure 12: Box Plot(Project 1 CNN)	16
Figure 13: Project 2 Dense Layer Model	17
Figure 14: Epochs vs Accuracy(Project 2 Dense).....	19
Figure 15: Epochs vs Loss(Project 2 Dense)	19
Figure 16: Epochs vs Validation Accuracy(Project 2 Dense)	20
Figure 17: Epochs vs Loss(Project 2 Dense)	21
Figure 18: Box Plot(Project 2 Dense).....	21
Figure 19: Project 2 CNN Layer Model	22
Figure 20: Epochs vs Accuracy(Project 2 CNN).....	24
Figure 21: Epochs vs Loss(Project 2 CNN)	24
Figure 22: Epochs vs Validation Accuracy(Project 2 CNN)	25
Figure 23: Epochs vs Loss(Project 2 CNN)	25
Figure 24: Box Plot(Project 2 CNN)	26
Figure 25: Project 3 Dense Layer Model	27
Figure 26: Epochs vs Validation Loss(Project 3 Dense)	29
Figure 27: Epochs vs Validation Accuracy(Project 3 Dense)	30
Figure 28: Epochs vs Validation Accuracy(Project 3 Dense)	31
Figure 29: Epochs vs Validation Loss(Project 3 Dense)	31
Figure 30: Project 3 CNN Layer Model	33
Figure 31: Epochs vs Loss(Project 3 CNN)	35
Figure 32: Epochs vs Accuracy(Project 3 CNN).....	35
Figure 33: Epochs vs Validation Accuracy(Project 3 CNN)	37
Figure 34: Epochs vs Validation Loss(Project 3 CNN).....	37

CHAPTER 1: PROJECT 1- ARCHITECTURE 1(FULL DENSE LAYERS)

For Project 1, we have selected the agent with the field of view(FOV) in 4 cardinal directions – The four neighbor Agent. The ML agent will first view all the valid neighbors at each timestamp and determine their states. Then, after scrutinizing the situation, it will predict and output the best possible direction it can take a step into to reach the target successfully and optimally.

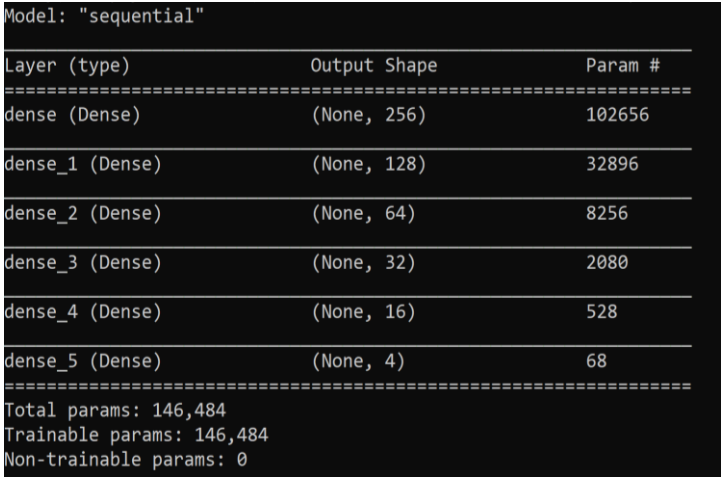
This chapter explores how we implement the four neighbor Agent using ML methodologies and an architecture consisting of only fully connected dense layers.

Chapter 1.1: Model Structure

We perform multiple experiments on the different types of models and their structure. We tweaked the number of hidden dense layers and number of neurons in each layer before coming up with the final model structure for our 4 Neighbor ML Agent as stated below:

We created a model trained on data fetched from numerous 20x20 dimensional mazes and used that model for simulations on 50x50 dimensional mazes. The main motive behind using such an approach is eliminating the data generation problem we faced when making the model for 50x50 dimensional mazes. For such a model, it required almost six times more data than the model we have implemented for the 20x20 mazes. Removing skewness from such a large dataset is also a rigorous and time-consuming task. Thus, we followed the above-stated approach.

Our model consists of one input layer, five hidden dense layers and one output layer. The below-attached figure shows the number of parameters between different layers and the number of neurons within each one.



```
Model: "sequential"
Layer (type)                Output Shape              Param #
=====
dense (Dense)                (None, 256)               102656
dense_1 (Dense)              (None, 128)               32896
dense_2 (Dense)              (None, 64)                8256
dense_3 (Dense)              (None, 32)                2080
dense_4 (Dense)              (None, 16)                528
dense_5 (Dense)              (None, 4)                 68
=====
Total params: 146,484
Trainable params: 146,484
Non-trainable params: 0
```

Figure 1: Project 1 Dense Layer Model

The critical thing to notice here is the structure of the output layer. It has only four neurons, precisely because we have formulated our problem as a 4-class classification problem. The underlying principle taken into consideration is as follows: At every timestamp, the Agent can only move into 4 of its neighboring cells in the cardinal directions and thus has only four choices. So, using this scheme, we have our output layer consisting of four neurons. Applying SoftMax activation in this layer will give us the output in the form probabilities for the valid direction the Agent can move to. **Adam optimizer and ReLU activation function are used in this model**

and all the subsequent ML Models formulated for Assignment 4. Learning rate used in Adam Optimizer is 0.001 and $\beta_1=0.9$ and $\beta_2=0.999$ (beta values) and these values also remain constants for all the implemented ML Agents.

Now for simulating the 50x50 dimensional mazes, initially, we randomly generate ten patches of dimension 20x20 from our existing 50x50 gridworld, which contains the Agent's current location and feed those patches(mazes) to our model for prediction. The vital thing to notice here is that these patches provide **local information** to the ML Agent. Then we average the output probabilities for each class from those ten randomly chosen instances, select the maximum one from the four averaged out probabilities, and choose that particular action as the final prediction.

Chapter 1.2: Addressing the Questions

Question 1:

For our model, the state space is nothing but the 20x20 maze. The maze contains three types of information as follows:

1. The state of a particular cell- unblocked/ blocked/ unvisited.
2. Current position and the valid neighbouring cells.
3. The number of times a particular cell is visited.

All this information is encoded in the following manner: Initially, as the ML Agent has no information and no cell is visited, all the cells are marked by 0. When the Agent visits a cell, it will first mark it as visited by assigning number 3 to it and to all the neighbouring unblocked cells. This number represents the number of times a cell can be visited, and its count will decrease every time the agent lands on the same cell. So, if a cell is marked by number 3, then it can be said that its state is discovered as unblocked and it has never been visited. We have explicitly taken 3 as the number because, after careful observations, we realized that if a cell has been visited 3 times, any future visits will provide no useful information. Also, any detected blocked cell will be marked by -1.

A cell's marked number is assigned as 100 if the Agent is in that cell and all its neighbouring cells' marked numbers are multiplied by 25. We encode the information of Agent's current position and all its neighbours by doing this. Thus, we define our state-space containing all the relevant information in this manner.

For the action space, the model will predict the Agent's movement and outputs a number in the range 0 to 3. Each of the four numbers carries its meaning, as stated below:

- 0 – Agent takes a step in the upwards direction.
- 1 – Agent takes a step in the right direction.
- 2 – Agent takes a step in the downwards direction.
- 3 – Agent takes a step in the left direction.

The ML Agent at a specific timestamp collects new information about the neighbors and has all the accumulated past information. This relevant information empowers the ML Agent to take the next step in the best possible direction to reach the goal state optimally.

(Note: The Action space defined above remains the same for ML Agents mimicking Agents from Project 1 and 2.)

Question 2:

As we have formulated our problem as 4-class classification problem, we are using categorical cross entropy loss function when training our model. The formula for this loss function is as follows:

$$\text{Loss} = - \{ \sum_{i=1}^n t(i) \log(p(i)) \}, \text{ for } n \text{ classes,}$$

where $t(i)$ is the truth label and $p(i)$ is the SoftMax probability for i^{th} class

In our case, n equals 4 as we have only four classes. These four classes are nothing but 4 directions the Agent can take a step into.

The Agent can only move to one of its valid neighboring cells in the cardinal directions at any timestamp. Thus, the 4 possibilities in the action space are justified. Therefore, it only makes sense to implement 4 neighboring Agents as 4-class classification problem using Dense Layers.

(Note: The Loss function defined above remains the same for ML Agents mimicking Agents from Project 1 and 2.)

Question 3:

We took 416,000 different mazes and captured episodes at each timestamp of the Agent's movement to get good performance. (Here we consider the data point collected at Agent's single step movement as 1 episode.) However, we found that our data in the captured episodes was skewed because the count of Agent's right and downwards movements dominated the count for left and upwards movements on a big scale. So, to get equally distributed data, we took the minimum count of the four output directions and selected that many episodes equally from the data of all four directions, respectively. Thus, the final count came to 3,568,948 episodes from 416,000 different mazes. This became our entire training dataset.

For validation and test dataset, we generated another collection of 14,000 completely different mazes than the ones created previously and captured episodes at each timestamp of the Agent's movement, did the same procedure of removing skewness from the data and then segregated into 2 parts, one for validation and another for test dataset. The size of validation and test dataset is 50,282 episodes each. We followed such a methodology to create training and validation datasets consisting of totally different datapoints.

Also, we have not discarded unsolvable mazes during the data generation phase and retained the valuable information we receive until the Agent reaches a state where it realizes that the maze is not solvable.

Question 4:

Taking a large and equally distributed dataset eliminated the problem of overfitting the model, which is attested by the graph below which shows that the validation accuracy increases simultaneously with training accuracy. Additionally, the validation and training loss also maintain a downward trend throughout the training phase. We can see the stated results from the graphs shown below:

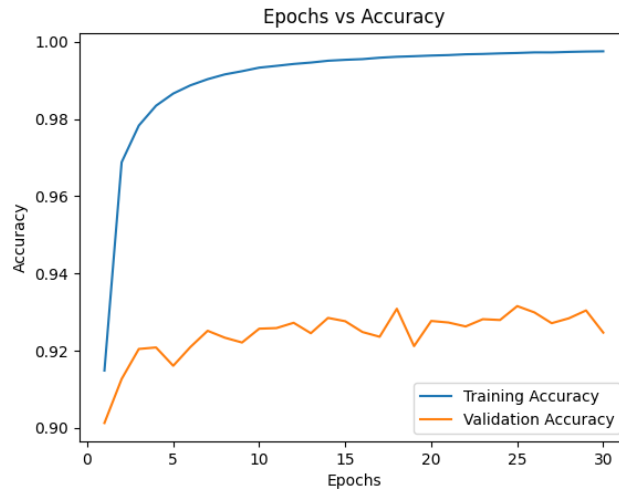


Figure 2: Epoch vs Accuracy(Project 1 Dense)

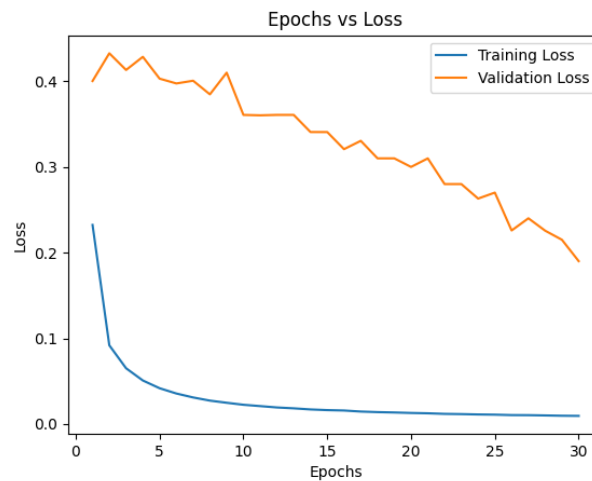


Figure 3: Epochs vs Loss(Project 1 Dense)

We don't want our model to get confused when it encounters new situations while simulating for new mazes. So, to remove the possibility of such a scenario, we should avoid overfitting our ML Agent model because our ultimate goal is to generalize our model for any input gridworld. We do not want our ML Agent to memorize the cases from input data and predict accordingly for any random gridworld. By avoiding overfitting, we make our model less prone to errors.

Question 5:

We started with a model with only three layers, including the input and output layers. Our model also consisted of significantly fewer parameters and neurons. Performing experiments on this initial model, we found that there is a scope of radical improvement and so we decided to gradually increase the number of neurons and hidden layers while ensuring that the model does not overfit. After multiple tweaks and carefully observed experiments, we landed on our final model architecture showing promising results.

Question 6:

Increasing the size or complexity of our model would not yield any improvements. This is because we have tried doing so and learned that our model was getting overfitted. So, we realized that making a larger or more complex architecture than the current one would negatively impact the current accuracy of our model.

Question 7:

As stated before, we have generated our training dataset and test dataset from completely different gridworlds to remove any similarity between them and the model attained 75.67% test accuracy. We expected similar results while performing simulations on completely new gridworlds, but it didn't turn out the same.

While simulating our ML Agent on 900 different gridworlds, it failed to reach the target in 36.44% of the total mazes. After performing detailed research, we found 3 main explanations as to why this was happening: 1) The ML Agent kept on predicting to step into a blocked cell. 2) It was trying to move into a cell not contained in the maze. 3) It moved to and fro between two specific cells.

To handle such cases, after five consecutive times of such flawed predictions, the ML Agent was explicitly moved to any other randomly selected valid neighboring cell. Also, a global threshold of predictions was kept as 500, and if the ML Agent crossed that threshold, the execution was stopped as it would not reach the target anyway. In this way, the problem of ML Agent not being able to reach the goal state was handled. Even after applying this solution, we found out that still the ML Agent was not able to reach the target in 28.89% of the cases. The ML Agent also couldn't find the path to target cell in those cases where the path is a very crooked and complex one thus reducing the overall solvability power of ML Agent. So, we see that the good performance in testing doesn't correlate with good performance in practice.

Question 8:

1. Below is the graph depicting performance on test data as a function of training rounds for our best model structure consisting only fully connected dense layers.

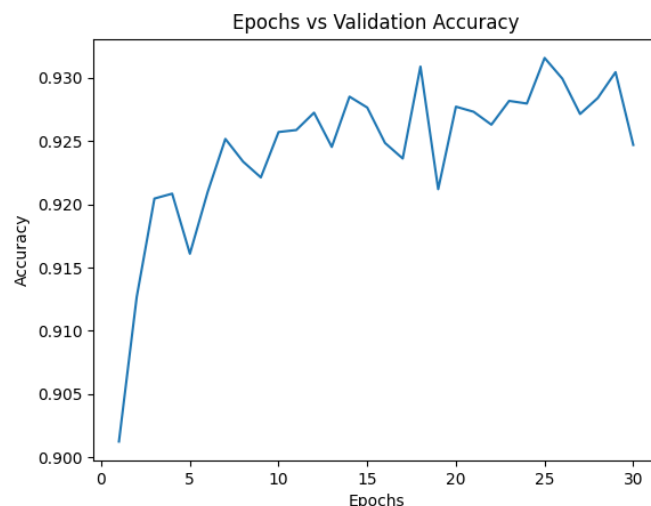


Figure 4: Epochs vs Validation Accuracy

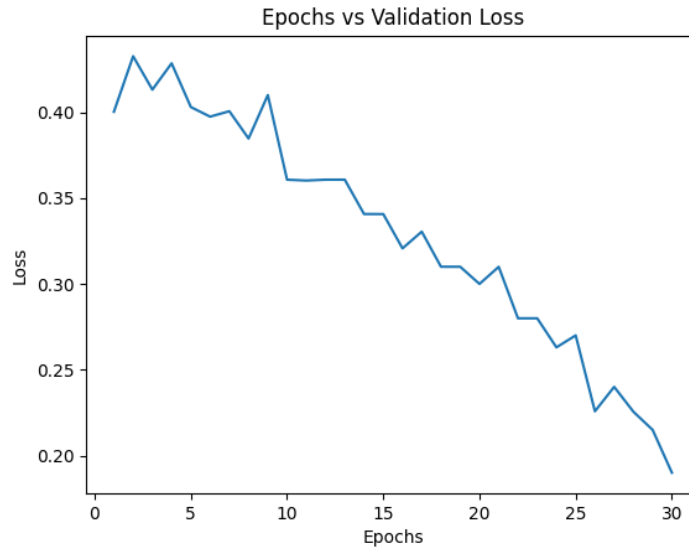


Figure 5: Epochs vs Validation Loss

2. For the simulations on gridworlds of size 20x20, our ML Agent can mimic the original 4 neighbor Agent very well with an accuracy of 93.52% because the model is trained on datasets generated from 20x20 dimensional mazes. For the simulations on gridworlds of size 50x50, our ML Agent faces difficulty selecting the same path as the original Agent and so it takes decisions with 74.22% accuracy. This is most probably because we are training on a 20x20 grid and testing on a 50x50 grid. This makes it so that our agent picks locally optimal movements but globally sub-optimal ones. Following the previously mentioned methodology, when simulated for a 50x50 maze, our ML Agent leverages local data much more than the global data and sometimes takes a longer path than the original Agent.

We have simulated 900 50x50 dimensional gridworlds and compared our ML Agent with the original Agent. Trajectory length is used as the metric of comparison. While comparing, we have used those instance mazes where the ML Agent was successful in reaching the target. The box plot comparing both the Agent's trajectory length is shown below:

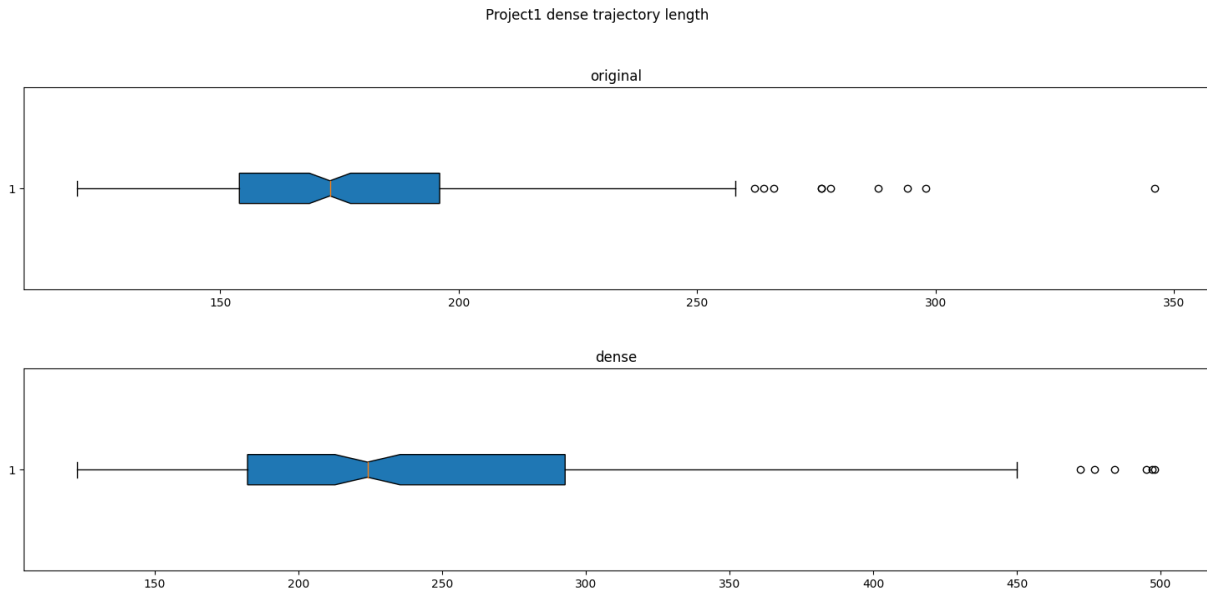


Figure 6: Box Plot(Project 1 Dense)

We can see from the above box plot that median trajectory length for the ML Agent is more than the Original Agent.

The Dense Layer Model was trained for a total of 30 epochs and epoch to train CNN was fixed at 5. When comparing our ML Agents for Project 1, we noticed that for CNN Model, completion of one epoch takes much more time than Dense Layer Model, and though the number of epochs for CNN Model was only selected as five, training CNN Model takes a longer time.

Also, we have not discarded unsolvable mazes during the data generation phase because of the following reasoning: Initially, the Agent does not have any information about the maze and think of the gridworld as the solvable one. In its attempt to find the target, it moves across the gridworld following the technique of Repeated Forward A*. If the maze is not a solvable one then after some time, it realizes this fact and comes to a standstill. But before coming to this conclusion, it has collected all the valuable episodes at various timestamp which can be used while training the model.

CHAPTER 2: PROJECT 1- ARCHITECTURE 2(CNN LAYERS)

This chapter explores how we implement the four neighbor Agent using ML methodologies and using an architecture consisting of CNN layers.

Chapter 2.1: Model Structure

For CNN Model also, we used the same technique of training the model on training dataset consisting the information for 20x20 dimensional mazes and simulating it for 50x50 mazes. This was done because of the same data generation problems mentioned in Chapter 1.1. The model architecture formulated is shown as below:

```
Model: "sequential"
Layer (type)                Output Shape              Param #
-----
conv2d (Conv2D)             (None, 64, 20, 20)       1792
max_pooling2d (MaxPooling2D) (None, 64, 10, 10)       0
conv2d_1 (Conv2D)           (None, 32, 10, 10)       18464
max_pooling2d_1 (MaxPooling2 (None, 32, 5, 5)         0
conv2d_2 (Conv2D)           (None, 16, 5, 5)         4624
flatten (Flatten)           (None, 400)               0
dense (Dense)               (None, 64)               25664
dense_1 (Dense)             (None, 4)                 260
-----
Total params: 50,804
Trainable params: 50,804
Non-trainable params: 0
```

Figure 7: Project 1 CNN Layers Model

Our model consists of total 9 layers including the input layers, 3 conv2d convolutional layers, 2 max pooling layers, 1 flatten layer, 1 dense layer and 1 output layer. Adam optimizer is used, and filter size is kept as 3x3. Also, SAME padding is used in convolutional layers. The model utilizes ReLU as the activation function. Here also we have implemented the problem at hand as 4 class classification problem as described in [Chapter 1.1](#). The procedure for simulating new mazes also remains the same as Architecture 1 stated in [Chapter 1.1](#).

(Note: SAME padding is used in all the CNN Models implemented in this Assignment.)

Chapter 2.2: Addressing the Questions

Question 1:

Our initial state space contains three types of information, all in different NumPy arrays. The three types of information are as follows:

The state of a particular cell- unblocked/ blocked/ unvisited.

Current position and the valid neighbouring cells.

The number of times a particular cell is visited.

For the first NumPy array consisting of information about the state of a particular cell, the following values for each cell are possible:

1: Discovered as unblocked

-1: Discovered as blocked

0: Not yet visited

For the second NumPy array consisting of information about the current position and valid neighbouring cells, the following assignments are made to the cell according to the situation:

100: Represents Agent's current cell position.

25: Represents all the valid neighbouring cells to Agent's current position.

0: Represents every other cell except the ones mentioned above.

For the final third NumPy array consisting of information about the number of visits to a particular cell, every cell has its count initially set to 0. The count is incremented every time the Agent steps in that cell.

As mentioned in [Chapter 1.2](#), the action space for CNN model is the same as defined in that same chapter and also the relevant information is collected in similar fashion.

Question 2:

In the case of CNN Model Agent, we again formulate the problem of finding the target more colloquially, moving to the next cell as a 4-class classification problem. The reason is that the Agent can move to only one of the valid neighboring cells in the cardinal directions at any given time. So, we use the same categorical cross entropy loss function as mentioned in Question 2 in [Chapter 1.2](#). and the formula remains the same.

Question 3:

For CNN Model, the data is generated using the same technique used for ML Agent using Dense Layers. 416,000 mazes are generated for the test dataset, and different episodes are captured at Agent's every movement to find the target. Then after removing the skewness in the data introduced because of the dominant numbers for Agent's right and downward directional steps, the total number of episodes sum up to 3,568,948. This whole dataset of 3,568,948 episodes is taken as a training dataset.

We generate another collection of 14,000 entirely different mazes for validation and test dataset and again capture episodes at Agent's every step. We again remove skewness from the data and record another dataset of episodes, split equally into two parts: validation and test dataset. This is done to remove the sequential trend

of datapoints in training and validation datasets. The validation and test dataset each contains 50,282 episodes each.

Question 4:

We forecasted the problem of overfitting the model, and because of this reason, we generated a massive dataset for training the model. This resulted in eliminating overfitting the model because we want our model to work in every possible case and not just mimic the original Agent. If the model overfits, then it will be such a case that it will only take steps as memorized by it during the training phase and thus would not intelligently handle never-before-seen cases. We can see from the below graph that our model does not overfit as the validation accuracy increases simultaneously with the increase in training accuracy and validation loss decreases simultaneously with the decrease in training loss.

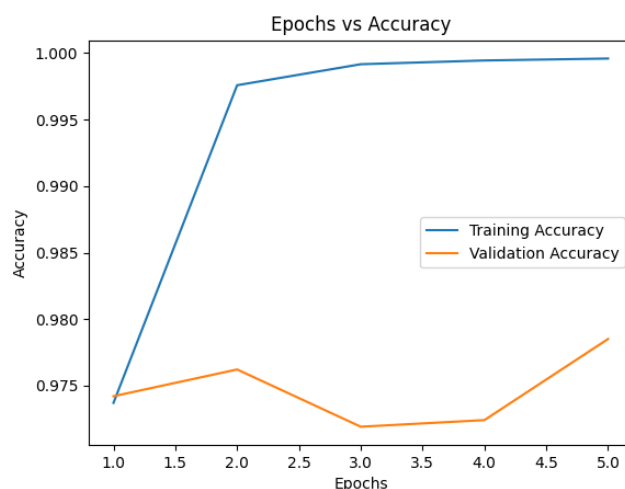


Figure 8: Epochs vs Accuracy(Project 1 CNN)

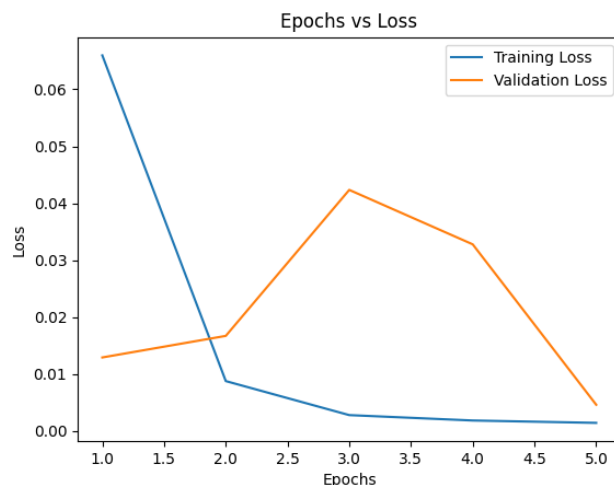


Figure 9: Epochs vs Loss(Project 1 CNN)

Question 5:

We performed various experiments on the architecture space before landing on the final one showing the best result. To start with, we took only one convolution layer and only a handful of neurons. We thought that building a more complex and deeper model would give better results than before, so we added more layers. After that, we decided to add more neurons to make it even better than before. After repeated experiments and improvements over time, we got our current model showing better results than before.

Question 6:

We tried making different adjustments to the model's structure during multiple experiments. We tried adding more convolutional and pooling layers, increasing the number of neurons in different layers, and testing various added layers combinations. We also tried changing alpha and beta values in Adam Optimizers to see if that helps in improving the results. However, nothing seemed to improve the model's performance. Contrastingly, we got worse validation accuracies when we added more layers and neurons. So, we decided to finalize the model we got before because even though it had fewer layers and neurons, it gave better results than the model with a labyrinthine structure. Thus, we realized that making our model more complex is of no point as it would only give us inferior quality results.

Question 7:

As stated before, we have generated our training dataset and test dataset from completely different gridworlds to remove any similarity between them and the model attained 80.45% test accuracy. Here too we expected that we would get similar good results while practicing the model for new gridworlds, but we found out that the ML Agent failed to reach the target in 30.66% of total 900 mazes. The 3 reasons for this are the same ones we stated and addressed in Question 7 in [Chapter 1.2](#).

We also applied the same solution used in Architecture 1 as stated in Question 7 in [Chapter 1.2](#) to eliminate such faulty predictions. But still the ML Agent got confused and failed to reach the target in 22.78% cases. We don't discard such cases where the target is not reached and instead calculated the solvability power of the ML Agent. Thus, we see that the good performance in testing doesn't correlate with good performance in practice.

Question 8:

1.) The below graph shows performance on test data as a function of training rounds.

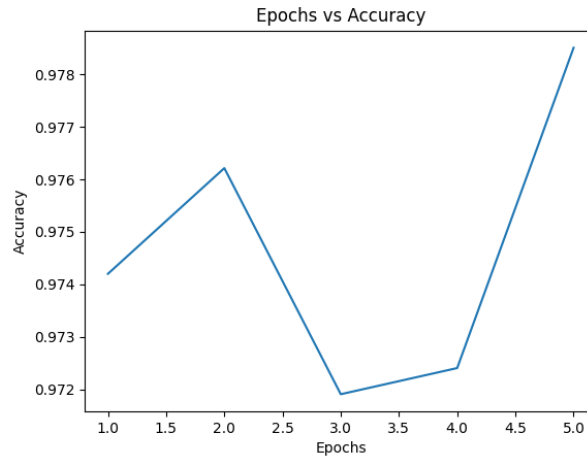


Figure 10: Epochs vs Validation Accuracy(Project 1 CNN)

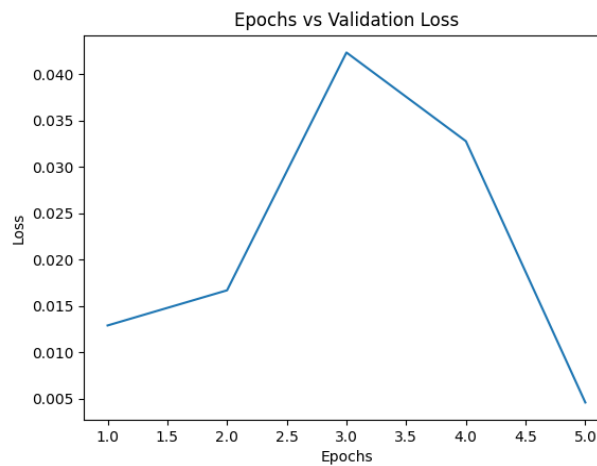


Figure 11: Epochs vs Validation Loss(Project 1 CNN)

2.) While practicing for new gridworlds of size 20x20, the CNN ML Agent mimicked the original Agent with an accuracy of 97.13%. But due to less global information, when practicing for 50x50 mazes, the ML Agent took decisions with 81.11 % accuracy while leveraging the local information received from 10 randomly sampled 20x20 patches containing the current Agent's position. The methodology remains the same as explained in Question 8 for ML Agent consisting of only Dense Layers.

Here too we have compared the trajectory length taken by ML Agent and Original Agent when both are simulated for 900 mazes. As did before, we have taken only those mazes into consideration, where the ML Agent was successful in reaching the target. The box plots comparing the trajectory lengths is shown below:

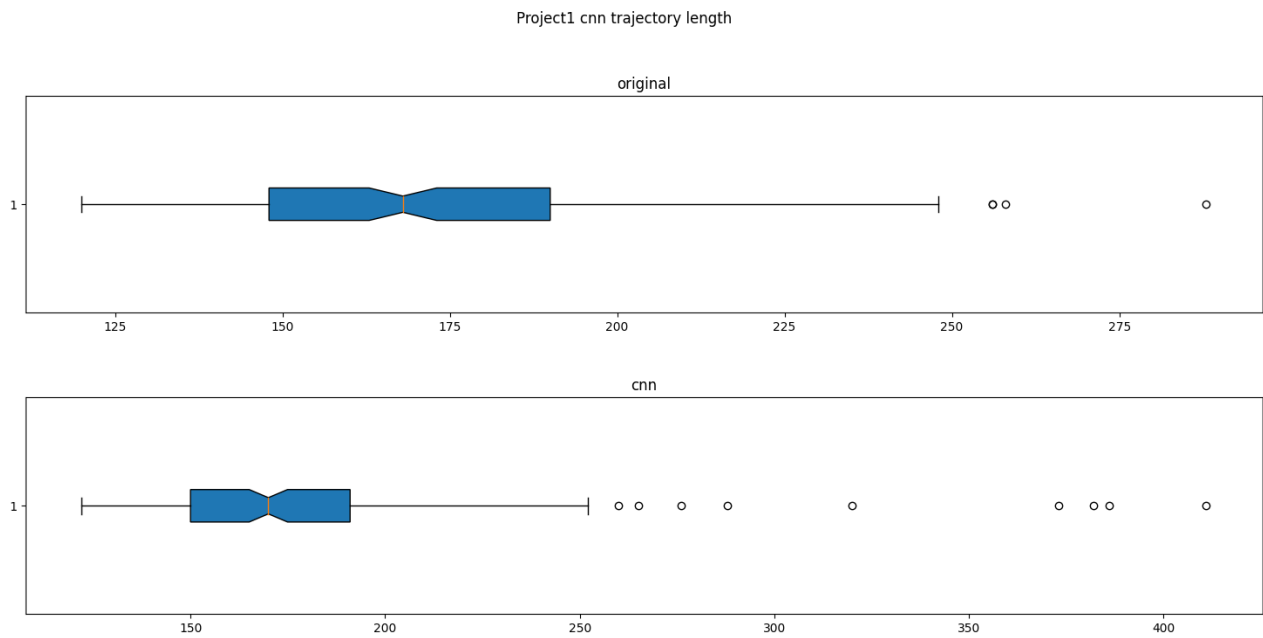


Figure 12: Box Plot(Project 1 CNN)

We can see from the above box plot that median trajectory length for the ML Agent is more than the Original Agent.

The Dense Layer Model was trained for a total of 30 epochs and epoch to train CNN was fixed at 5. When comparing our ML Agents for Project 1, we noticed that for CNN Model, completion of one epoch takes much more time than Dense Layer Model, and though the number of epochs for CNN Model was only selected as five, training CNN Model takes a longer time.

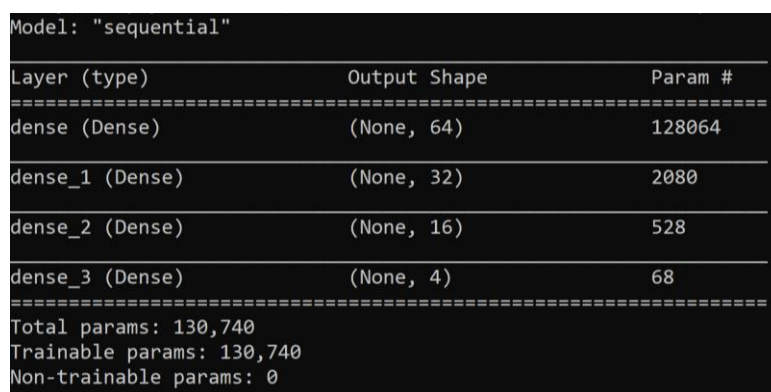
CHAPTER 3: PROJECT 2- ARCHITECTURE 1(FULL DENSE LAYERS)

For the second part of this Assignment, we have decided to mimic the Example Inference Agent from Project 2. At each timestamp, the ML agent will first sense the number of blocked cells in all the valid neighbors (8, 5 or 3 according to the location of the cell in the gridworld) at each timestamp. Then, after applying its reasoning and prediction skills, it will output the best possible direction it can take a step into to reach the target successfully and optimally.

This chapter explores how we implement the Example Inference Agent using ML methodologies and an architecture consisting of only fully connected dense layers.

Chapter 3.1: Model Structure

The below image shows the model structure of the ML Agent consisting of only dense layers.



```
Model: "sequential"
Layer (type)                 Output Shape                 Param #
=====
dense (Dense)                 (None, 64)                   128064
dense_1 (Dense)               (None, 32)                   2080
dense_2 (Dense)               (None, 16)                   528
dense_3 (Dense)               (None, 4)                    68
=====
Total params: 130,740
Trainable params: 130,740
Non-trainable params: 0
```

Figure 13: Project 2 Dense Layer Model

We have used the same technique of training the model on episodes generated from 20x20 mazes and practicing it for new gridworlds of 50x50 mazes as did before in previous Project 1 ML Agents. The model consists of total 5 layers: 1 input layer, 3 hidden dense layers and one output layer with SoftMax activation function to get the prediction for best possible direction to step into with maximum probability. Here also we have formulated the problem definition as 4-class classification whose reasoning is explained in [Chapter 1.1](#).

At each timestamp, the ML Agent collects fresh new figures by sensing the number of blocked cells in the valid neighboring cells. Then as stated in earlier agents, 10 patches of size 20x20 are randomly generated which contain the Agent's current location and feed it to the model with the information retrieved before. Finally, we get an output as the direction in which the Agent takes a step into to reach the goal optimally. So, the model just takes in the information collected and generated and infers internally the blocked cells or unblocked ones as learned in the training phase and outputs the direction in which a step has to be taken.

Adam optimizer and ReLU activation function are used in the model infrastructure.

Chapter 3.2: Addressing the Questions

Question 1:

The state space is slightly changed here from the ones used before to incorporate the sensing information acquired at each timestamp. The state space consists of 5 types of information as stated below:

- Agent's current location, number of visits and state of the cell(0:unvisited, 1:unblocked -1:blocked).
- Number of valid neighbors sensed to be blocked.
- Number of valid neighbors sensed to be unblocked.
- Number of valid neighbors confirmed to be blocked.
- Number of valid neighbors confirmed to be unblocked.

Agent's current location, number of visits and the state of cells are encoded in the similar fashion as done in Project 1 Architecture 1 consisting of Dense Layers.

Thus, after flattening the entire 5 types of information contained in 5 different 20x20 NumPy arrays, we get a vector of length 20x20x5=2000. At each timestamp, the Agent's current location is updated in the state space by assigning the number 100 to that cell.

The action space remains entirely the same as we used in previous ML Agents as the Agent takes one step in a particular direction and so the direction becomes the final prediction at each timestamp.

We are training the model for 20x20 mazes and so as stated in previous chapters, the 10 patches of 20x20 size containing the Agent's current location will serve as the local information while practicing the ML Agent for 50x50 mazes. Also, the Agent uses the past information in the updated state space at each step and the sensing information received at the new timestamp to predict in which direction it should step into. This past information from updated state space and the newly captured sensing information becomes the relevant information at a particular timestamp.

Question 2:

As we are still defining the problem as 4-class classification problem as done in previous Agents, the loss function remains same as categorical cross entropy loss function. The formula remains the same which is reiterated as below:

$$\text{Loss} = - \{ \sum_{i=1}^n t(i) \log(p(i)) \}, \text{ for } n \text{ classes,}$$

where $t(i)$ is the truth label and $p(i)$ is the SoftMax probability for i^{th} class

In our case, n equals 4 as we have only four classes. These four classes are nothing but 4 directions the Agent can take a step into.

Question 3:

For this model too, the data is generated using the same technique used in previously described ML Agents. 182,000 mazes are generated for the test dataset, and different episodes are captured at Agent's every movement to find the target. Then after removing the skewness in the data introduced because of the dominant

numbers for Agent's right and downward directional steps, the total number of episodes summed up to 1,360,420. This whole dataset of 1,360,420 episodes is taken as a training dataset.

We generate another collection of 14,000 entirely different mazes for validation and test dataset and again capture episodes at Agent's every step. We again remove skewness from the data and record another dataset of episodes, split equally into two parts: validation and test dataset. This is done to remove the sequential trend of datapoints in training and validation datasets. The validation and test dataset each contains 50,282 episodes respectively.

Question 4:

As explained in previous ML Agents, here too we need to avoid overfitting the model as we don't need our model to just remember all the training cases it has been fed. We need it work for any maze it has never encountered before and take wise decisions in those new situations. So, to introduce generality, we generated a huge training dataset which allowed the ML Agent to develop an intelligence which can be later used in situations it has never seen before and take correct decisions at almost all timestamps so as to match the performance of Example Inference Agents and reach the target optimally. The model architecture that we finalized also helped in removing the problem of overfitting. The below graphs confirms that our model doesn't overfit of decreasing trend observed in validation loss and increasing trend in validation accuracy.

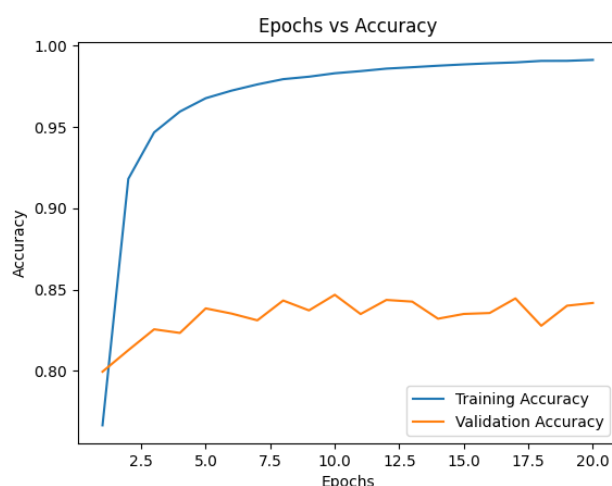


Figure 14: Epochs vs Accuracy(Project 2 Dense)

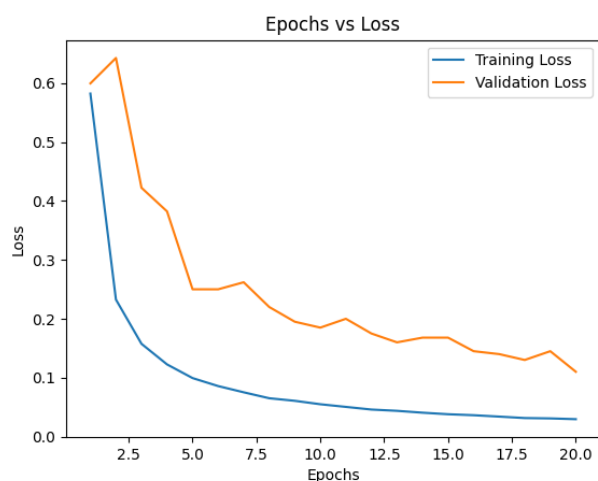


Figure 15: Epochs vs Loss(Project 2 Dense)

Question 5:

According to the general practice, we first tried with a model infrastructure having one input layer, one hidden layer and one output layer. The result we got was good even from this small model architecture. But we realized the real potential of deeper infrastructure while we performed experiments for creating a better infrastructure for previous Agents and so here too, we started performing multiple tweaks. After each experiment, we noted down the accuracy we achieved and after analysis, we either gradually increased number of layers or number of neurons in each layer. After multiple experiments and dozens of tweaks, we got our final architecture. We even tried different optimizers like Adagrad and Stochastic Gradient Descent, but we got the most promising results while using Adam optimizer.

Question 6:

In pursuit of finding the best architecture, we tried making our model even deeper and adding more neurons in each layer. However, we did not get any positive outcome in doing so. The deeper and more complex models either gave the same accuracy we got in our finalized architecture or reduced the accuracy. So, it is fair to say that increasing the size or complexity of the model would not be beneficial even if it is enticing to do so.

Question 7:

The response to this question remains the same as we discussed for previous ML Agents. Expecting that the same 3 problems would be faced by ML Agent in reaching the target as described in earlier ML Agents, here we took the same measures taken previously to solve those problems. But even after applying the solutions, we saw that ML Agent could solve 59.66% of the total 900 simulated mazes whereas for test dataset, the accuracy came around 67.55%. This tells us about that good performance on test dataset is not correlated with good performance in practice.

Question 8:

- 1.) The below graph shows performance on test data as a function of training rounds.

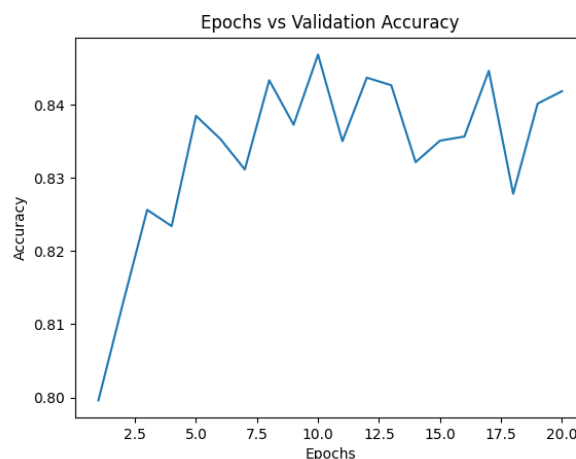


Figure 16: Epochs vs Validation Accuracy(Project 2 Dense)

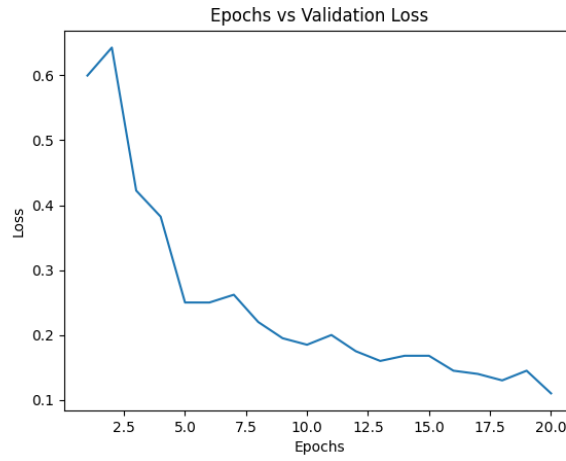


Figure 17: Epochs vs Loss(Project 2 Dense)

2.) As our model is trained for 20x20 mazes, when averaging out the accuracy we get from different simulations on gridworlds of dimensions 20x20, the number came out as 84.90%. This number tells that our model mimics the original Example Inference Agent with decent accuracy when tested for gridworlds of size 20x20. When simulating for 50x50 gridworlds, due to lack of global information, the model relies very much on the local data and so it mimics the Example Inference Agent with an average of 67.55 % accuracy which was expected.

We are comparing our ML Agent with the original one by simulating 900 gridworlds and choosing trajectory length as the metric of comparison. From the total 900 mazes, only those instances are selected where the ML Agent was successful in finding the target. The below-attached box plot shows that the mean trajectory length for ML Agent is greater than the mean trajectory length for original agent.

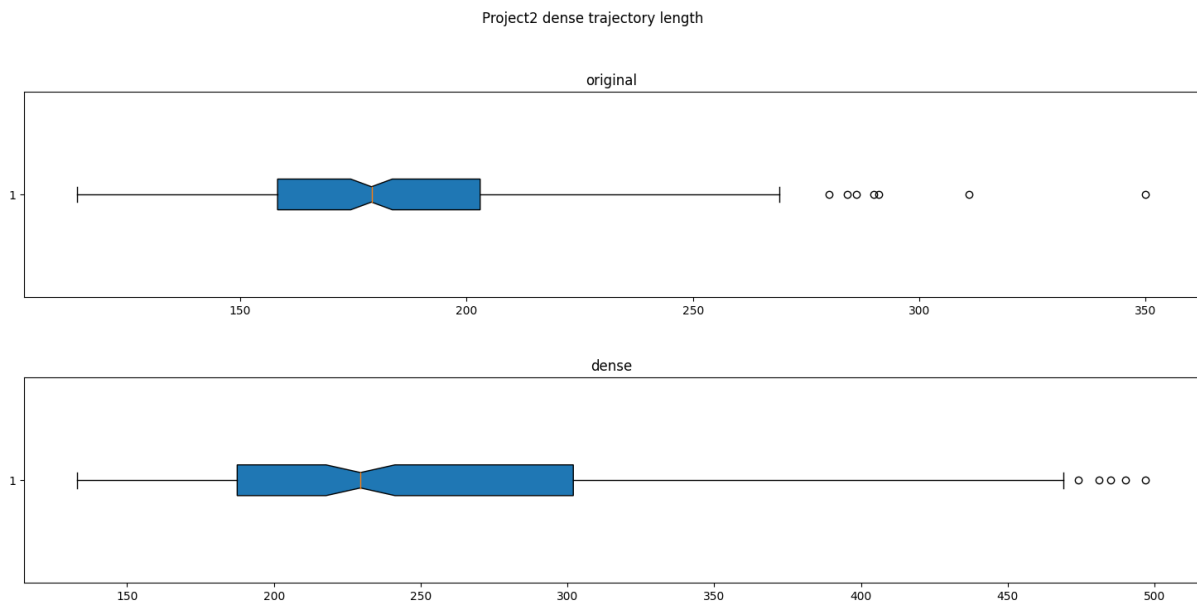


Figure 18: Box Plot(Project 2 Dense)

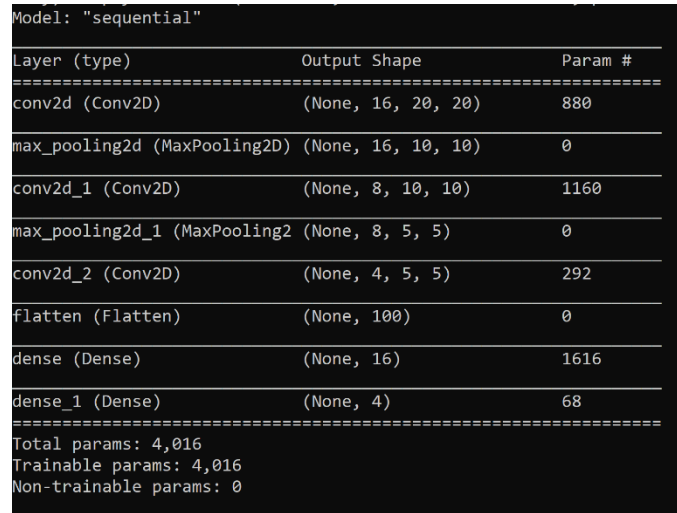
For the cases where the ML Agent cannot reach the target successfully, they are handled as mentioned in chapter 2.2.

CHAPTER 4: PROJECT 2- ARCHITECTURE 2(CNN LAYERS)

This chapter pertains to explaining how we mimic the Example Inference Agent using machine learning and CNN Layers.

Chapter 4.1: Model Structure

The below image shows our final architecture of the model which mimics the functioning of the Example Inference Agent.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 20, 20)	880
max_pooling2d (MaxPooling2D)	(None, 16, 10, 10)	0
conv2d_1 (Conv2D)	(None, 8, 10, 10)	1160
max_pooling2d_1 (MaxPooling2D)	(None, 8, 5, 5)	0
conv2d_2 (Conv2D)	(None, 4, 5, 5)	292
flatten (Flatten)	(None, 100)	0
dense (Dense)	(None, 16)	1616
dense_1 (Dense)	(None, 4)	68
Total params: 4,016		
Trainable params: 4,016		
Non-trainable params: 0		

Figure 19: Project 2 CNN Layer Model

The model consists of one input layer, 3 conv2d convolutional layers, 2 max pooling2d layers, 1 flatten layer, 1 dense hidden layer and 1 output dense layer with SoftMax function. The output layer uses SoftMax function because we have implemented the ML Agent using 4-class classification technique. So, the final output is in form of 4 probabilities for each direction the Agent can take a step towards to reach the target. The direction with the highest probability is selected so as to reach the target optimally.

Adam optimizer is used and ReLU activation function is used. 3x3 filter size is used throughout the architecture and SAME padding is used in convolutional layers.

We have still used the technique of training our model on 20x20 mazes and using it to simulate 50x50 mazes because of the data generation requirements issues mentioned in [Chapter 1.1](#).

Chapter 4.2: Addressing the Questions

Question 1:

We are using the state space and the information contained in it as used in Architecture 1 for Project 2. To recapitulate, the 5 types of information contained in state space of NumPy Array of size 20x20x5 are as follows:

- Agent's current location, number of visits and state of the cell(0:unvisited, 1:unblocked -1:blocked).
- Number of valid neighbors sensed to be blocked.
- Number of valid neighbors sensed to be unblocked.
- Number of valid neighbors confirmed to be blocked.
- Number of valid neighbors confirmed to be unblocked.

Agent's current location, number of visits and state of the cell are all encoded in similar fashion as done in Project 1 Architecture 2 consisting of CNN Layers.

As we are using the same technique of the 4-class classification technique because of previously stated reasoning, the action space remains the same as described in [Chapter 1.2](#).

The Agent also has all the relevant information in the form of previously captured information at past timestamps and newly captured sensing information. Using this information, it takes prudent decisions to reach the goal optimally.

Question 2:

We are using the same categorical cross entropy loss function used in all the previously described Agents.

Question 3:

We are using the exact same data used in Architecture 1 so please refer to Question 3 of [Chapter 3.2](#) to get the exact number of episodes in training, validation, and test datasets.

Question 4:

For our model to make the best usage of data and make accurate decisions in any possible situation, it should not get overfitted. Generality should be maintained as only then will the ML Agent be called intelligent. Otherwise, it would simply be a memorizer of all the cases in training dataset episodes.

To remove overfitting, we implemented the same solution of generating large dataset in previous ML Agents and making the model deeper as opposed to shallow one. We increased the number of layers only to a certain extent as more layers usually results in overfitting the model. The below graph shows the regular decrease in validation loss and increase in validation accuracy with each successive epoch:

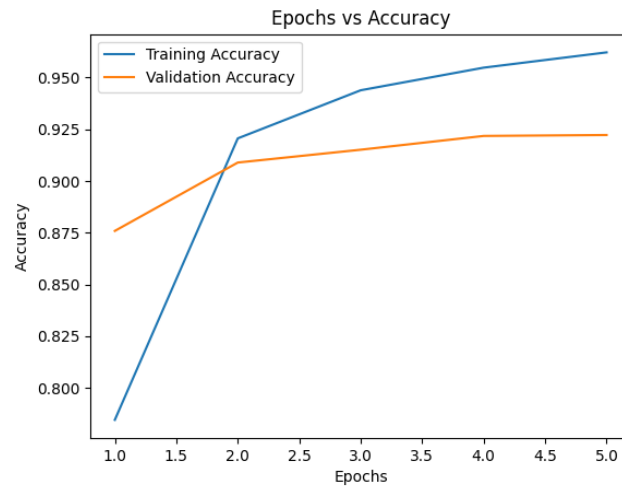


Figure 20: Epochs vs Accuracy(Project 2 CNN)

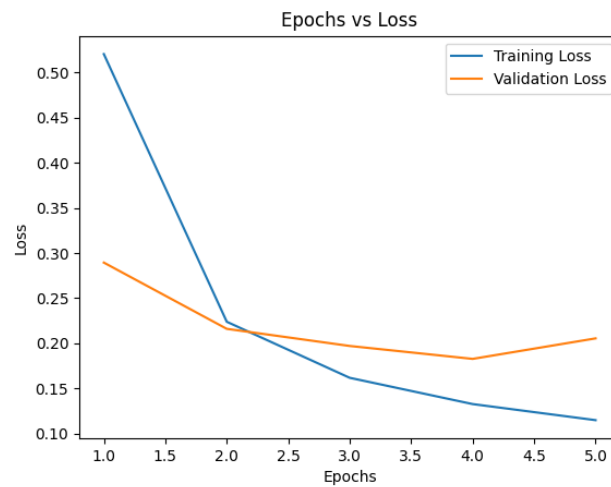


Figure 21: Epochs vs Loss(Project 2 CNN)

Question 5:

In finding the final architecture of the CNN model, we used various combinations of layers at different places. We used different types of optimizers and varying filter sizes. We even incorporated many dense layers with various configurations before finalizing only one dense layer before the output layer. We increased and decreased the number of neurons and noted the outcomes of such changes. We also tried changing values for different hyperparameters, but no positive outcome was reflected. After performing various permutations and combinations of various number of different layers, we finalized the one we described initially as it gave the best results. We even considered the tradeoff between the data requirements and depth of model architecture.

Question 6:

As mentioned in question 5, before finalizing the architecture, we performed numerous experiments. We also attempted to increase the model's complexity by adding various types of layers and added neurons. That thing only led to the overfitting of the model, which is unacceptable. So, we can say with surety that making the

current model more complex would drastically change the accuracy in a wrong way, so no good could come out of doing so.

Question 7:

We are using the same data used in Architecture 1 to train the CNN ML Agent. After solving those 3 problems stated in previous ML Agents, when the ML Agent is put into practice, it could solve 64.0% of the total 900 simulated mazes. While in test dataset we can see that the accuracy is around 74.11% which is high as compared to the performance in practice. Thus, we can say that good performance in test data doesn't correlate with good performance in practice.

Question 8:

- 1.) The below graph shows performance on test data as a function of training rounds.

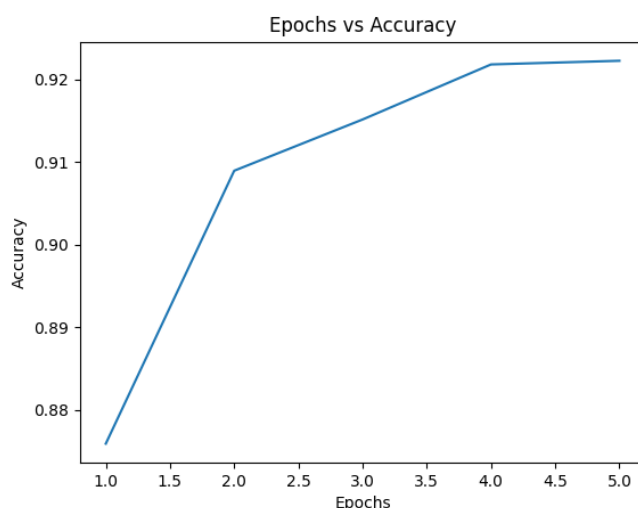


Figure 22: Epochs vs Validation Accuracy(Project 2 CNN)

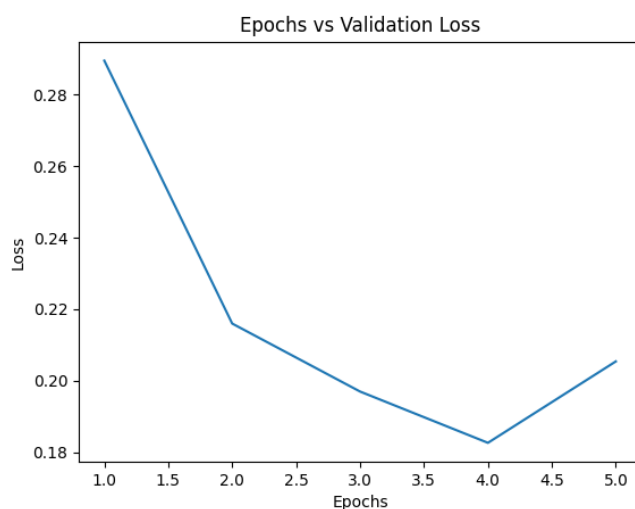


Figure 23: Epochs vs Loss(Project 2 CNN)

- 2.) We can see that the validation accuracy increases with every subsequent epoch until it reaches a saturation of 95% at 4th epoch. The validation loss decreases with every epoch and just slightly increases in the last epoch.

Our model gave average accuracy of 90.32% when tested for 20x20 mazes. This high number shows that the model could mimic the original Example Inference Agent very well. When tested for 50x50 mazes, the average accuracy drops to 74.11% because of the loss of global data as the model is trained on 20x20 mazes. While simulating for 50x50 mazes, as described before, we feed the model with 10 randomly generated patches of size 20x20 containing the Agent's current location and average out the output probabilities from all those 10 samples and then pick the direction with maximum probability. In doing so, the model tends to use the local information very much and this is the reasoning behind the decrease in model's accuracy.

We have also compared ML Agent with the original one by using the same methodology used in Dense Layer ML Agent. We can see from the below generated graphs that the mean trajectory length of ML Agent is greater than the original one:

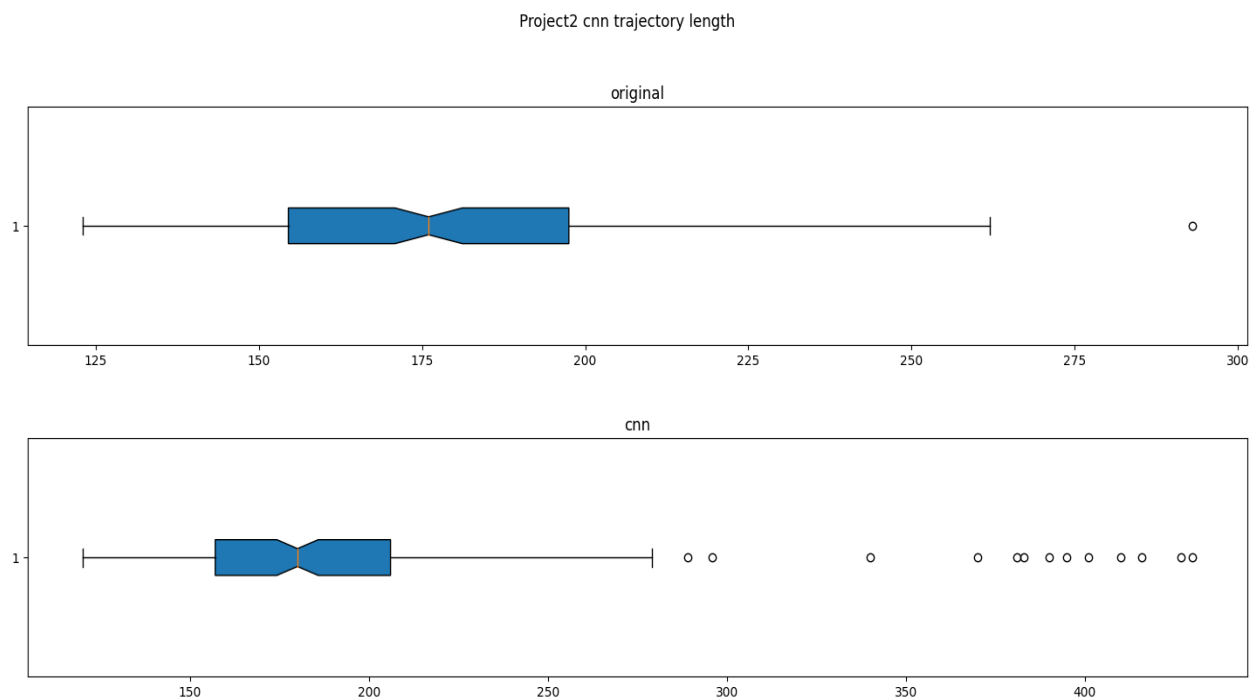


Figure 24: Box Plot(Project 2 CNN)

CHAPTER 5: PROJECT 3- ARCHITECTURE 1(FULL DENSE LAYERS)

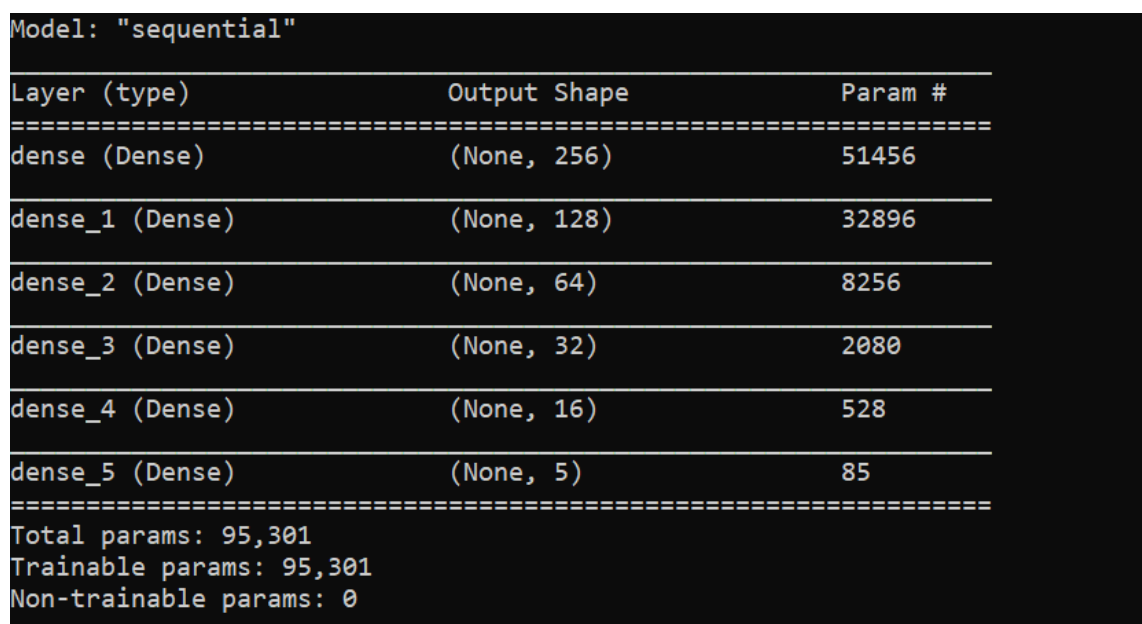
This chapter explores how we implemented Agent 8 using ML methodologies and using an architecture consisting of Dense layers.

Chapter 5.1: Model Structure

After performing multiple experiments on the different types of models and their structure by tweaking the number of hidden dense layers and number of neurons in each layer, we came up with the final model structure for Agent 8 as stated below:

We created a model trained on data fetched from numerous 10x10 dimensional mazes and used that model for testing on 25x25 dimensional mazes. The reason why we have taken such small mazes is that our data generated for just 10000 mazes of 10x10 dimensions is around 10 GB in size.

Our model consists of one input layer, five hidden dense layers and one output layer. The below-attached figure shows the number of parameters between different layers and the number of neurons within each one.



```
Model: "sequential"
Layer (type)                Output Shape                Param #
=====
dense (Dense)                (None, 256)                 51456
dense_1 (Dense)              (None, 128)                 32896
dense_2 (Dense)              (None, 64)                  8256
dense_3 (Dense)              (None, 32)                  2080
dense_4 (Dense)              (None, 16)                  528
dense_5 (Dense)              (None, 5)                   85
=====
Total params: 95,301
Trainable params: 95,301
Non-trainable params: 0
```

Figure 25: Project 3 Dense Layer Model

The critical thing to notice here is the structure of the output layer. It has only five neurons, precisely because we have formulated our problem as a 5-class classification problem. The underlying principle taken into consideration is as follows: At every timestamp, the Agent can only move into 4 of its neighboring cells in the cardinal directions or it can decide to examine the cell it is currently in. So, using this scheme, we have our output layer consisting of five neurons. Applying SoftMax activation in this layer will give us the output in the form of the action that the agent can take.

Now for simulating the 25x25 dimensional mazes, initially, we randomly generate ten patches of dimension 10x10, which contains the Agent's current location and feed those patches(mazes) to our model for prediction. Then we average out the output probabilities for each class from those ten randomly chosen instances, select the maximum one from the averaged-out probabilities, and choose that particular action as the final prediction.

Chapter 5.2: Addressing the Questions

Question 1:

For our model, we have set the state space as multiple 10x10 mazes. We give the model 2 separate 10x10 mazes which contain the following information –

1. The state of a particular cell- unblocked/ blocked/ unvisited. We encode the current position and the neighbours of the current position cell into this grid by multiplying current position and neighbours' position by 10 and 5 respectively.
2. The probabilities of finding the target at each cell divided by the distance from current cell.

For the action space, the model will predict the Agent's movement and outputs a number in the range 0 to 4. Each of the four numbers carries its meaning, as stated below:

- 0 – Agent takes a step in the upwards direction.
- 1 – Agent takes a step in the right direction.
- 2 – Agent takes a step in the downwards direction.
- 3 – Agent takes a step in the left direction.
- 4 – Agent examines the cell it is currently in.

Question 2:

As we have formulated our problem as 5-class classification problem, we are using categorical cross entropy loss function when training our model. The formula for this loss function is as follows:

$$\text{Loss} = - \{ \sum_{i=1}^n t(i) \log(p(i)) \}, \text{ for } n \text{ classes,}$$

where $t(i)$ is the truth label and $p(i)$ is the Softmax probability for i^{th} class

In our case, n equals 5 as we have five classes. These five classes are nothing but 4 directions the Agent can move + the action of examining the current cell.

Question 3:

We took 10,000 different mazes and captured episodes at each timestamp of the Agent's movement or examination to get good performance. (Here we consider the data point collected at Agent's single step movement or examination as 1 episode.) However, we found that our data in the captured episodes was skewed because agent 8 tended to examine 4 times more than move in any particular direction. So, to get equally distributed data, we took the minimum count of the 5 outputs and selected that many episodes equally from the data of all 5 outputs. Thus, the final count came to 668,948 episodes from 10,000 different mazes. This became our entire training dataset.

For validation and test dataset, we generated another collection of 1,000 different mazes than the ones created previously and captured episodes at each timestamp of the Agent's movement and examinations, did the same procedure of removing skewness from the data and then segregated into 2 parts, one for validation and another for test dataset. The size of validation and test dataset is 30,282 episodes each. We followed such a methodology to create training and validation datasets consisting of totally different datapoints.

For Project 3, we have decided to completely discard the unsolvable mazes since we did the same while creating Agent 8.

Question 4:

Taking a large and equally distributed dataset eliminated the problem of overfitting the model, which can be seen from the graph below which shows that the validation accuracy increases simultaneously with training accuracy. The decreasing trend for validation and training loss also remains the same throughout the training phase.

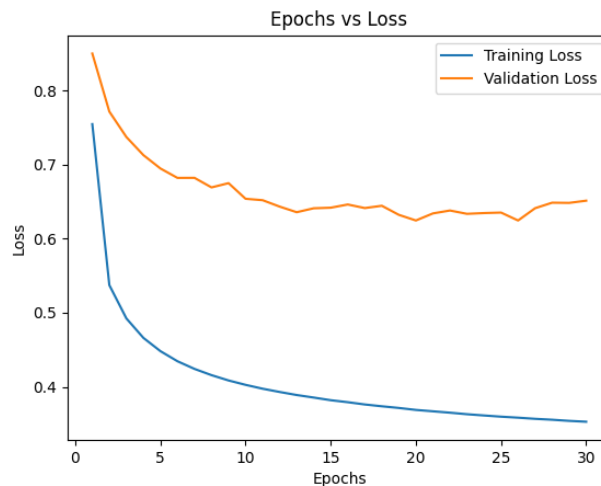


Figure 26: Epochs vs Validation Loss(Project 3 Dense)

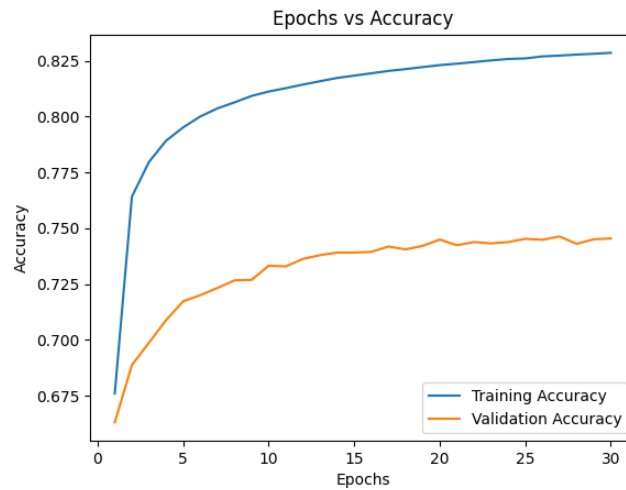


Figure 27: Epochs vs Validation Accuracy(Project 3 Dense)

We want to avoid overfitting our model so that when our agent tries to go through a completely new maze it won't give us erroneous results. By taking care to avoid overfitting we can make it so that our ML agent performs very well even on completely new mazes given to it.

Question 5:

We started with a model with only three layers, including the input and output layers. Our model also consisted of significantly fewer parameters and neurons. We first obtained an accuracy of around 50%. Since this was low, we decided to heavily increase the number of layers and tried a model with 7 layers including the input and output layers. This caused our accuracy to drop to 20%. To try and fix this we reduced the number of layers 5 and then our accuracy settled on around 85%. After multiple tweaks and carefully observed experiments, we landed on our final model architecture showing promising results.

Question 6:

As shown by our experiments and results above, increasing the size and complexity offered no improvements to our model. This is probably because after a point, the number of parameters in the layers was becoming higher than the number of parameters of our actual input. This could cause a decrease in accuracy which was undesirable.

Question 7:

We have generated the testing and training data on completely different mazes and obtained an accuracy of 85% during that testing. We expected that this performance on testing data would translate well to real simulations but for simulated data our agent was only about to solve the grid about 30% of the time.

Instead of discarding 70% of our unsolved mazes, we used those mazes to discover some problems with the behavior of our agent. Just like the agents in project 1 and 2, our agent got stuck moving between 2 cells of high

probability without examining. It tended to want to move to cells outside the maze. Our agent would sometimes examine cells that do not have the highest probability of finding the target.

Even after implementing some fixes for these issues, our agent could only solve around 50% of total mazes we gave to it. From this we can clearly see that good performance on test data does not mean that we get a good performance in practice.

Question 8:

1. Below is the graph depicting performance on test data as a function of training rounds for our best model structure consisting only fully connected dense layers.

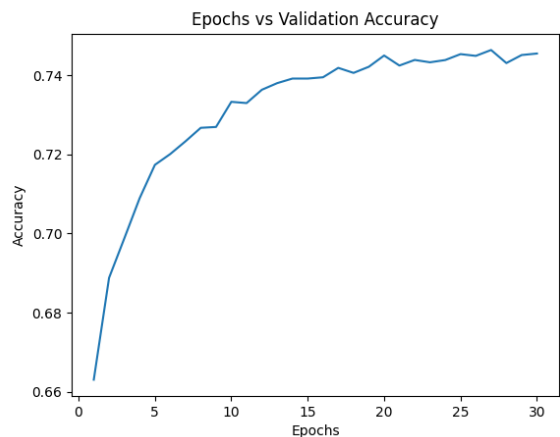


Figure 28: Epochs vs Validation Accuracy(Project 3 Dense)

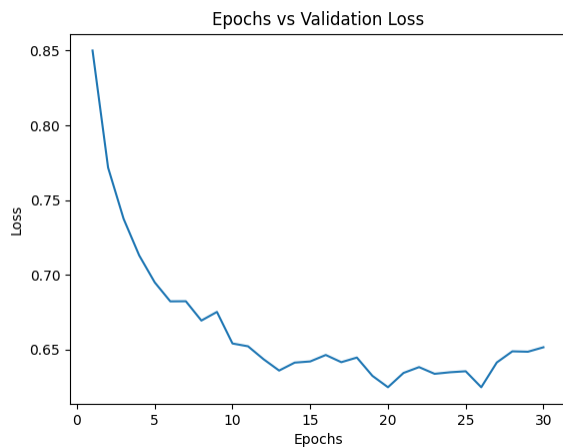


Figure 29: Epochs vs Validation Loss(Project 3 Dense)

2. As our model is trained for 10x10 mazes, when averaging out the accuracy we get from different simulations on grid worlds of dimensions 10x10, the number came out as 55.22%. This number tells that our model mimics the original Example Inference Agent with decent accuracy when tested for grid worlds of size 10x10. When simulating for 25x25 grid worlds, due to lack of global information, the model relies very much on the local data and so it mimics Agent 8 with an average of 45.2 % accuracy which was expected.

For the simulations on grid worlds of size 10x10, our ML Agent seems to mimic Agent 8 very well because the model is trained on datasets generated from 10x10 dimensional mazes. For the simulations on grid worlds of size 25x25, our ML Agent faces difficulty selecting the same path as the original Agent. This is because of the lack of global data. Following the previously mentioned methodology, when simulated for a 25x25 maze, our ML Agent leverages local data much more than the global data and sometimes takes a longer path than the original Agent.

In terms of CNN and Dense agents' comparison, our results were the same as the agents for Project 1 and 2. That is – our dense ML agent takes a lesser time to train than our CNN ML agent.

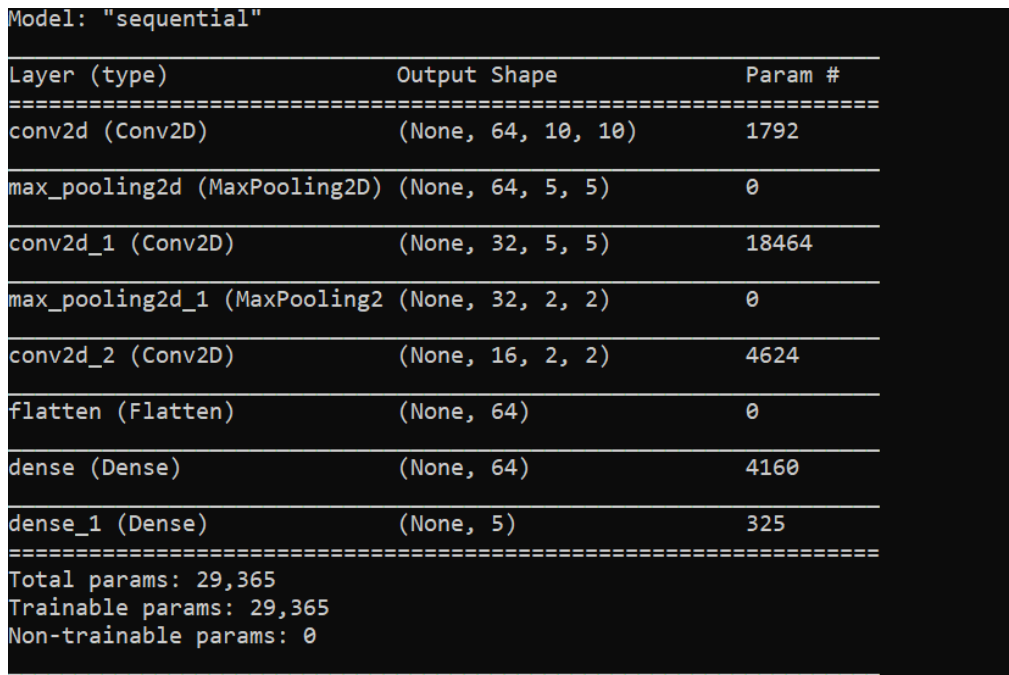
Also, we have not discarded unsolvable mazes during the data generation phase and retained the valuable information we receive until the Agent reaches a state where it realizes that the maze is not solvable.

CHAPTER 6: PROJECT 3- ARCHITECTURE 2(CNN LAYERS)

This chapter explores how we implemented Agent 8 using ML methodologies and using an architecture consisting of CNN layers.

Chapter 6.1: Model Structure

The below image shows our final architecture of the model which mimics the functioning of the Example Inference Agent.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 10, 10)	1792
max_pooling2d (MaxPooling2D)	(None, 64, 5, 5)	0
conv2d_1 (Conv2D)	(None, 32, 5, 5)	18464
max_pooling2d_1 (MaxPooling2D)	(None, 32, 2, 2)	0
conv2d_2 (Conv2D)	(None, 16, 2, 2)	4624
flatten (Flatten)	(None, 64)	0
dense (Dense)	(None, 64)	4160
dense_1 (Dense)	(None, 5)	325
Total params: 29,365		
Trainable params: 29,365		
Non-trainable params: 0		

Figure 30: Project 3 CNN Layer Model

The model consists of one input layer, 3 conv2d convolutional layers, 2 max pooling2d layers, 1 flatten layer, 1 dense hidden layer and 1 output dense layer with SoftMax function. The output layer uses SoftMax function because we have implemented the ML Agent using 5-class classification technique. So, the final output is in form of 5 probabilities for each direction the Agent can take a step towards to reach the target. The direction with the highest probability is selected to reach the target optimally.

Adam optimizer is used and ReLU activation function is used. 3x3 filter size is used throughout the architecture and SAME padding is used in convolutional layers.

We have still used the technique of training our model on 10x10 mazes and using it to simulate 25x25 mazes because of the data generation limitations mentioned in Chapter 3.1.

Chapter 6.2: Addressing the Questions

Question 1:

For our model, we have set the state space as multiple 10x10 mazes. We give the model 3 separate 10x10 mazes which contain the following information –

1. The state of a particular cell- unblocked/ blocked/ unvisited.
2. Current position and the valid neighbouring cells.
3. The probabilities of finding the target at each cell divided by the distance from current cell.

For the action space, the model will predict the Agent's movement and outputs a number in the range 0 to 4. Each of the four numbers carries its meaning, as stated below:

- 0 – Agent takes a step in the upwards direction.
- 1 – Agent takes a step in the right direction.
- 2 – Agent takes a step in the downwards direction.
- 3 – Agent takes a step in the left direction.
- 4 – Agent examines the cell it is currently in.

Question 2:

As we have formulated our problem as 5-class classification problem, we are using categorical cross entropy loss function when training our model. The formula for this loss function is as follows:

$$\text{Loss} = - \{ \sum_{i=1}^n t(i) \log(p(i)) \}, \text{ for } n \text{ classes,}$$

where $t(i)$ is the truth label and $p(i)$ is the SoftMax probability for i^{th} class

In our case, n equals 5 as we have five classes. These five classes are nothing but 4 directions the Agent can move + the action of examining the current cell.

Question 3:

We took 10,000 different mazes and captured episodes at each timestamp of the Agent's movement or examination to get good performance. (Here we consider the data point collected at Agent's single step movement or examination as 1 episode.) However, we found that our data in the captured episodes was skewed because agent 8 tended to examine 4 times more than move in any direction. So, to get equally distributed data, we took the minimum count of the 5 outputs and selected that many episodes equally from the data of all 5 outputs. Thus, the final count came to 668,948 episodes from 10,000 different mazes. This became our entire training dataset.

For validation and test dataset, we generated another collection of 1,000 different mazes than the ones created previously and captured episodes at each timestamp of the Agent's movement and examinations, did the same procedure of removing skewness from the data and then segregated into 2 parts, one for validation and another

for test dataset. The size of validation and test dataset is 30,282 episodes each. We followed such a methodology to create training and validation datasets consisting of totally different datapoints.

For Project 3, we have decided to completely discard the unsolvable mazes since we did the same while creating Agent 8.

Question 4:

Taking a large and equally distributed dataset eliminated the problem of overfitting the model, which can be seen from the graph below which shows that the validation accuracy increases simultaneously with training accuracy. The decreasing trend for validation and training loss also remains the same throughout the training phase.

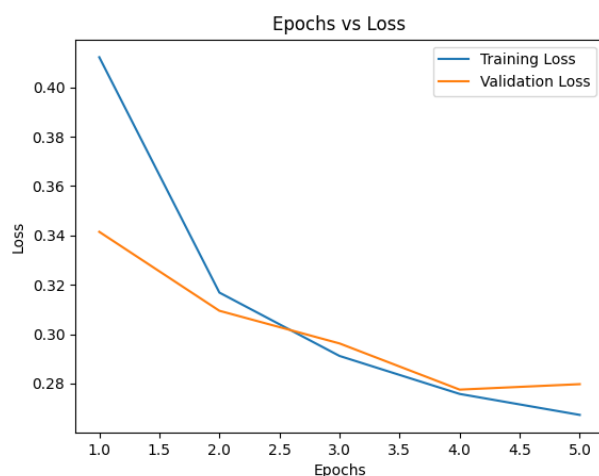


Figure 31: Epochs vs Loss(Project 3 CNN)

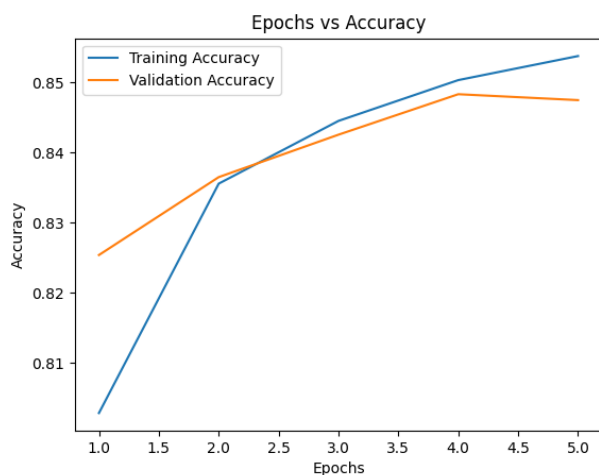


Figure 32: Epochs vs Accuracy(Project 3 CNN)

We want to avoid overfitting our model so that when our agent tries to go through a completely new maze it won't give us erroneous results. By taking care to avoid overfitting we can make it so that our ML agent performs very well even on completely new mazes given to it.

Question 5:

In finding the final architecture of the CNN model, we used various combinations of layers at different places. We used different types of optimizers and varying filter sizes. We even incorporated many dense layers with various configurations before finalizing only one dense layer before the output layer. We increased and decreased the number of neurons and noted the outcomes of such changes. After performing various permutations and combinations of various number of different layers, we finalized the one we described initially as it gave the best results. We even considered the tradeoff between the data requirements and depth of model architecture.

Question 6:

As shown by our experiments and results above, increasing the size and complexity offered no improvements to our model. This is probably because after a point, the number of parameters in the layers was becoming higher than the number of parameters of our actual input.

Question 7:

We have generated the testing and training data on completely different mazes and obtained an accuracy of 85% during that testing. We expected that this performance on testing data would translate well to real simulations but for simulated data our agent was only about to solve the grid about 40% of the time.

Instead of discarding 60% of our unsolved mazes, we used those mazes to discover some problems with the behavior of our agent. Just like the agents in project 1 and 2, our agent got stuck moving between 2 cells of high probability without examining. It tended to want to move to cells outside the maze. Our agent would sometimes examine cells that do not have the highest probability of finding the target.

Even after implementing some fixes for these issues, our agent could only solve around 60% of total mazes we gave to it. From this we can clearly see that good performance on test data does not mean that we get a good performance in practice.

Question 8:

1. Below is the graph depicting performance on test data as a function of training rounds for our best model structure consisting only fully connected dense layers.

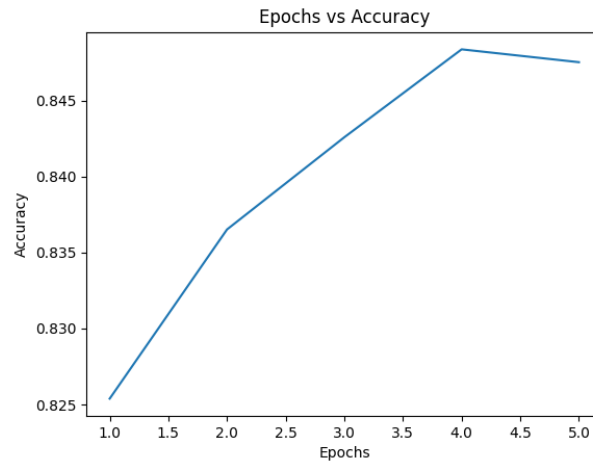


Figure 33: Epochs vs Validation Accuracy(Project 3 CNN)

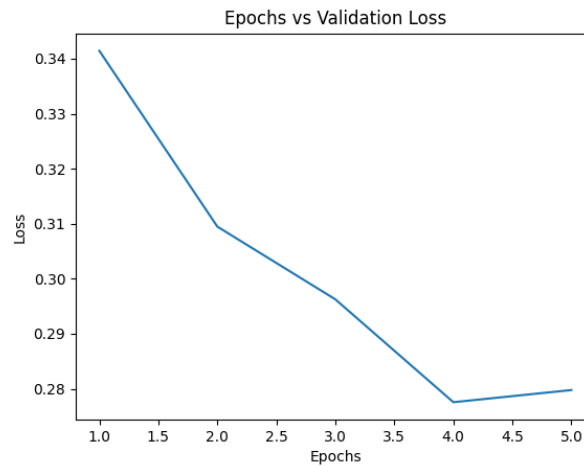


Figure 34: Epochs vs Validation Loss(Project 3 CNN)

2. As our model is trained for 10x10 mazes, when averaging out the accuracy we get from different simulations on grid worlds of dimensions 10x10, the number came out as 58.74%. This number tells that our model mimics the original Example Inference Agent with decent accuracy when tested for grid worlds of size 10x10. When simulating for 25x25 grid worlds, due to lack of global information, the model relies very much on the local data and so it mimics Agent 8 with an average of 49.12 % accuracy which was expected.

For the simulations on grid worlds of size 10x10, our ML Agent seems to mimic the Agent 8 very well because the model is trained on datasets generated from 10x10 dimensional mazes. For the simulations on grid worlds of size 25x25, our ML Agent faces difficulty selecting the same path as the original Agent. This is because of the lack of global data. Following the previously mentioned methodology, when simulated for a 25x25 maze, our ML Agent leverages local data much more than the global data and sometimes takes a longer path than the original Agent.

In terms of CNN and Dense agents' comparison, our results were the same as the agents for Project 1 and 2. That is – our dense ML agent takes a lesser time to train than our CNN ML agent.

Also, we have not discarded unsolvable mazes during the data generation phase and retained the valuable information we receive until the Agent reaches a state where it realizes that the maze is not solvable.