CS 520 : Project 4
The Imitation Game

Karan Ashokkumar Pardasani(kp955)
Naishal Chiragbhai Patel(np781)
Himaniben Hareshkumar Patel(hhp46)

December 16, 2021

[1]

karan.pardasani@rutgers.edu
naishal.patel@rutgers.edu
himani.h.patel@rutgers.edu

Contents

1

Glossary

1. Agent 1: This agent is same as the agent-1 in Project 1. This agent can detect whether the blocks present in the four directions of the current position of the agent is blocked or not. Also, it knows the location of the target cell and it plans the path from the current position of agent to known target using the A-star Algorithm.

2. ML Agent 1: This agent has same characteristics as that of agent 1. It can detect whether the cells in the four directions are blocked or unblocked. The agent tries to mimic the actions done by agent 1 using an Artificial Neural Network. The agent pre-processes the current knowledge of the agent and then gives the input to the Neural Network.

3. Agent 2: This agent is same as Agent 4 in Project 2. This agent can detect the number of blocked cells in the eight neighbours of the current position of the agent. It draws a system of equations where equation represents the number of blocks in a neighbourhood. This agent draws inferences from the this system of equations and tries to solve the system of equations. If any cell is inferred to be blocked or unblocked, then the agent updates the current knowledge.

4. ML Agent 2: This agent has the same characteristics as that of Agent 2 above. This agent can detect the number of blocked cells in the eight neighbours of the current position of the agent. This agent tries to mimic the actions of Agent 2. It uses a Neural Network that mimics the actions of agent 2. The Neural Network is trained with a processed input and the action of Agent 2 as output. Then, this agent uses the Neural Network to predict the actions of Agent 2 and uses this Network to mimic the action of Agent 2.

2

Imitating Agent 1

2.1 Original Agent 1

We implemented Agent 1 using Repeated Forward A* for traversing the Grid World to reach the end goal. Initially, the agent does not have any knowledge of the Grid Environment and assumes that there are no blockages and plans its path based on following two metrics:

- g(n) : shortest path length discovered from the *start cell* to *current cell.*
- h(n) : heuristic value which estimates the remaining distance from *current cell* to *goal cell.*
- f(n) : g(n) + h(n), i.e. total path length estimate from *start cell* to *goal cell*

As it traverses the environment, it updates the its knowledge acquired through its field of view. If encountered with a blockage in its planned path, path planning is done with the newly acquired knowledge of the environment taken into consideration. This strategy is applied until the agent reaches the goal node or the paths are exhausted which means a path from *start* to *goal* does not exist.

2.2 ML Agent 1

Here we attempt to implement ML Agent 1 such that it mimics the decisions of our original Agent 1.

To train the model, we generated data at each timestep which resulted to roughly 0.5M data points. The data includes the Agent Grid, current position of the Agent and its movement decision(i.e. up, down, left, right).

1. **How should the state space (current information) and action space (action selected) be represented for the model? How does it capture the relevant information, in a relevant way, for your model space? One thing to consider here is local vs global information.**

Our Original Agent takes into consideration its current knowledge of the gridworld that includes the discovered blocked and unblocked cells and its current position in the gridworld to make movement decisions which guides it through the grid to reach the target.
Input to the Artificial Neural Network(ANN): One Hot Encoding of size 2500(as we have a 50 * 50 grid).
Input to the Convolutional Neural Network(CNN): 2D vector of size 50*50.
To encode the information about the current position of the agent in the input data, we assigned weights to the cells as follows:

- **0** is assigned to the undiscovered cells of the gridworld.
- **-1** is assigned to the discovered blocked cells.
- **3** is assigned to discovered unblocked neighbours of the agent's current position cell. Also, if a discovered unblocked cell is visited by the agent, then the value of the weight representing that cell is reduced by 1 if the value is so far has been greater then 1. By doing this, the ML Agent gets a sense from where it is coming and where it is heading thus giving it a sense of direction of movement along with the path that has been traversed to reach the current position. This helps in a way that the agent learns to not get trapped in an infinite loop.
- **100x:** To encode the current position of the Agent in the gridworld, the weight assigned to represent that cell is multiplied by 100.
- **25x:** To encode the current position of the Agent in the gridworld, the weight assigned to represent that cell's neighbours is multiplied by 25.

2. **How are you defining your loss function when training your model?**
Previously, we mentioned that the output of our neural network represents the four directions our agent can move, i.e. up, down, left, right. We can classify this problem as a Multi-Class Classification Problem. We evaluated the results using different loss function which are used for Multi-class Classification like Categorical Cross-Entropy Loss, Sparse Multi-class Cross-Entropy Loss, ???Kullback Leibler Divergence Loss?? Using Categorical Cross-Entropy gave us the best results among the above mentioned loss functions.

   Categorical Cross-Entropy loss is calculates as follows:

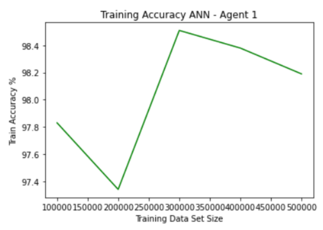
$$Cross\ Entropy\ Loss = -\sum_{i}^{C} t_i * log(f(s)_i) \tag{2.1}$$

   where C = *number of classes*
   $f(s)_i$ = *activation function*
   t_i = *an element of the target vector*

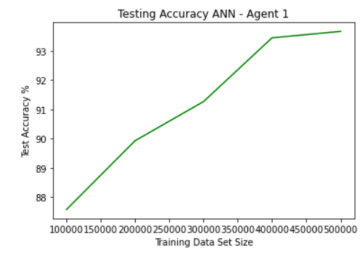
3. **In training, how many episodes on how many different gridworlds were necessary to get good performance of your model on the training data?**

Initially, we tried to train our model for 100*100 grid but the data was insufficient using more data was proving to be computationally infeasible for the computational power available to us which included our personal computers and the resources provided by the Computer Science department at the University. So, we shifted to 50*50 grids.

The following graph shows the testing and training accuracy as we increase the training data examples:



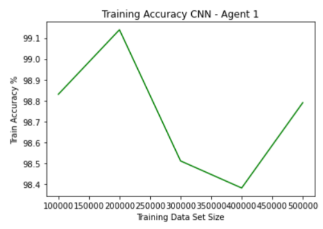
(a)



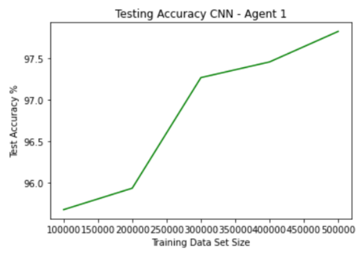
(b)

Figure 2.1: Train and Test accuracy of ML Model (using ANN) of Agent 1 vs amount of training data for Agent 1 (a) Train Accuracy (b) Test Accuracy

For Convolutional Neural Network, the model was trained for 5 epochs. We trained the models from 1 lakh data points to 5 lakhs data points. The following are the graphs for CNN that indicates the Training and Test Accuracy for Agent 4 when we use CNN model. As similar to the data used in ANN model, we generated the 1 lakh data points from 1000 grids and 5 lakh data points from 2500 grids.



(a)



(b)

Figure 2.2: Train and Test accuracy of ML Model(using CNN) of Agent 1 vs amount of training data for Agent 1 (a) Train Accuracy (b) Test Accuracy

4. **How did you avoid overfitting? Since you want the ML agent to mimic the original agent, should you avoid overfitting?**
   To avoid overfitting we firstly kept our model relatively simple and monitored the test and train accuracy and we found that the data we generated was insufficient and thus we drastically increased the data points to around

0.5M. Increasing the amount of data resulted into increase in training as well as the test accuracy.
We also introduced Dropout to reduce the overfitting. This helps in a way that reduces the model complexity and introduces a little noise so that the model becomes robust.

5. **How did you explore the architecture space, and test the different possibilities to find the best architecture?**
Initially, we started with a relatively simple architecture with 1 input layer, 2 hidden layers and 1 output layer. Then we gradually increased the complexity of the model. Following is the detailed explanation for the same.

   Finding Best Architecture for ANN Architecture:
Initially, We used input data as 2501 follows:

   - First 2500 elements which represent cells of the Agent's Grid.
     - **0:** discovered blocked cells.
     - **1:** unblocked cells and undiscovered cells.
   - The last element representing the current position of the Agent in the gridworld.

   With this feature vector, we were able to get 61% test accuracy initially and even with more complex architectures, we were not bale to obtain more than 62.4% accuracy. Therefore, we decided to make changes to our feature vector as follows.
First we thought that the information that where the path from which agent reached the current position was not encoded into the feature vector and thus the data points did not correlate with each other. Thus we decided to weigh the features as explained in Question 1. With this we were able to encode information like where the agent is coming from, what options it has to move to in the next step. This prevented the agent from getting into an infinite loop.

   Initially, we generated a relatively simple model as follows:

   - input layer of size 2500, 1 hidden layer of size 1500, 1 hidden layer of size 1000 and an output layer of size 4.
   - Activation function used was RELU initially but Leaky RELU gave us better results.
   - We used categorical Cross-Entropy as the Loss Function.
   - We used Adam Optimizer.
   - We tried adding dropout to this model for tackling the overfitting problem, but it reduced the test and training accuracy so we did not include Dropout in the final version for the model of Agent 1.

   Increasing the complexity of the model from here did not have much of an impact on the results of the model.

```
model = Sequential()
model.add(Dense(1250, input_dim = 2500, activation = 'relu'))
model.add(Dense(1800, activation = 'relu'))
model.add(Dense(1250, activation = 'relu'))
model.add(Dense(4, activation = 'softmax'))
```

Listing 2.1: Final ML Agent 1 ANN model

Finding Best Architecture for CNN Architecture:
Using the above mentioned revised feature vector for the Convolutional Neural Networks gave us an accuracy of 91.6% which is a significant increase as compared to the 79.2% accuracy we obtained using the initial feature vector. This might be because the initial feature vector did not encore the relevant information. To further increase the accuracy, we tested the results after increasing the number of fully connected layers but it did not significantly increase the accuracy. Therefore, we increased the number of convolutional layers and this took the accuracy to 93.5% and by further increasing the numer of convoutional layers, we obtained an accuracy of 95.8%. Other parameters like Loss, Optimizer and activation function are the same as that of the ANN model explained above.

```
input_shape = (50,50,1)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
    input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

Listing 2.2: Final ML Agent 1 CNN model

6. **Do you think increasing the size or complexity of your model would offer any improvements? Why or why not?**
We already reduced the complexity once when we were exploring for a suitable model, since the model was overfitting and even increasing the data did not increase the test accuracy. So we do not think that increasing the complexity of the model will offer any improvements. Following is the detailed explanation of the same.

   - **Increase the number of layers in the network:** Increase the number of layers in the network : In this case, each layer is having an activation function that induces non-linearity in the prediction model. Hence, as we increase the number of layers, the degree of non-linearity will increase which will lead to the increase in flexible functions that can exactly fit at each data point in the training data. This does not allow the model to generalise well over other points. So, increasing the number of layers will lead to overfitting.
   - **Increase the number of nodes in each layers:** When we increase the number of nodes in a layers, we are increasing the number of parameters in a layer. This allows us to choose more features from the input of the previous layer. When we choose more features from

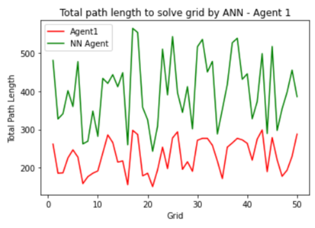
the input, the prediction model becomes more complex. Since choosing more features might lead to features that are redundant or the features that aren't much useful, this will lead to overfitting. Since increasing the number of features will also allow the function to be flexible enough to fit to all the training data points, which will not allow the model to generalise. Hence, increasing the number of nodes in a layer will lead to overfitting.

CNN Architecture

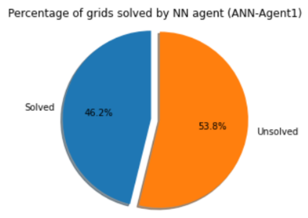
The CNN Architecture training for this agent proved to be another example for the Occam's Razor which tells that often the simplest explanation is the best explanation. When we trained the Convolutional Neural Network, the model gave the testing accuracy of 97% which was very good in relative to the ANN performance. We tried for 3 different layers for CNN and we choose the model that gave the best accuracy.

7. **Does good performance on test data correlate with good performance in practice? Simulate the performance of your ML agent on new gridworlds to evaluate this.**

   No, good performance on test data does not correlate with good performance in practice for this agent for our our model. We calculated the performance of the ML model using 3 metrics: accuracy, path length and number of grids that the agent is able to solve. We got the following graphs for our proposed ANN model:



(a)

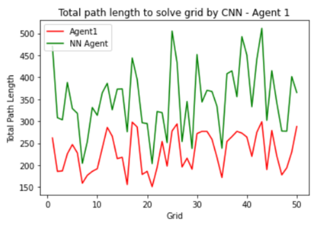


(b)

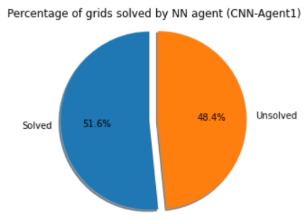
Figure 2.3: Performance of Agent 1 ANN architecture on different gridworlds (a) Path Length (b) Solvability of Grids using Agent 4

Convolutional Neural Network:

Similar to the performance of ANN, CNN also does not perform good on new grids. This is probably because the training on samples assumes that each sample is independent of each other. But, when the agent moves across the grid, the next input actually depends on previous input. Hence the CNN is not able to model this aspect of the problem. It assumes that each training example is independent of each other. This actually requires the modelling technique of time series data since the movement of agent can be seen as dependant on the previous input in terms of time. The performance of the agent is as follows:



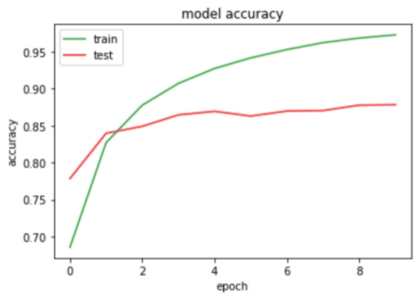
(a)



(b)

Figure 2.4: Performance of Agent 1 ANN architecture on different gridworlds (a) Path Length (b) Solvability of Grids using Agent 1

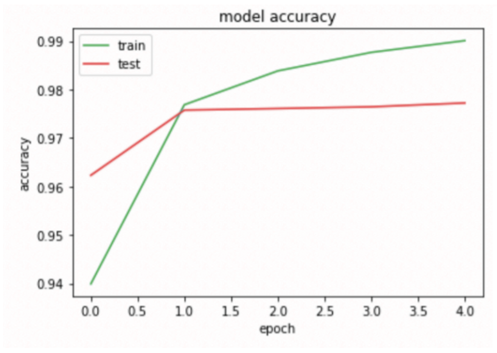
8. **For your best model structure, for each architecture, plot a) performance on test data as a function of training rounds, and b) average performance in practice on new gridworlds. How do your ML agents stack up against the original agents? Do either ML agents offer an advantage in terms of training time?**
   Below are the graphs for the ANN agent.



(a)

Figure 2.5: Performance of Agent 1 ANN architecture during training

The ANN model is trained for 10 epochs. We can see that as epochs increases the accuracy increases and hence the ANN and CNN architecture of the agent can mimic the project agent 1. But we can see that the CNN architecture performs better than ANN architecture. This proves that the local information improves the performance than encoding the global information.



(a)

Figure 2.6: Performance of Agent 1 CNN architecture during training

The above graph contains both the training and testing accuracy for ANN and CNN architecture of Agent 1.

Average Performance of Agent in new gridworld? How do your ML agents stack up against the original agents?

We can see that the performance of all the architectures in the new grid-world is bad and even though they can be trained to mimic the agent in the projects.

Also, in comparison to the ML agents against the project agent, we can see that the ML agents do not perform better than the project agents. This is evident from the number of grids that the agent can solve and among the grids that the agent can solve, we can see that the path length is not better than the project agent. Hence, the performance of the project agent is better than the ML agent.

3

Imitating Agent 4

3.1 Original Agent 4

Agent 4 is also a blind folded agent which senses its surroundings in 8 compass directions and obtain the number of cells which are blocked in its immediate neighbourhood. Agent 4 looks at individual equations containing neighbourhood of x and combines different equations and draws inferences about unexplored cells.

3.2 ML Agent 4

For the agent that uses the Machine Learning model to mimic the actions of Agent 4, it uses the Neural Network model to predict the next action of the agent 4 of previous project. For the training of Machine Learning Agent of Project 2, we have generated 5,00,000 data points from a total of 4500 grids. We train the Machine Learning model that agent 4 uses with this data.

1. **How should the state space (current information) and action space (action selected) be represented for the model? How does it capture the relevant information, in a relevant way, for your model space?One thing to consider here is local vs global information.**

   INPUT STATE SPACE
   For each step of that the agent takes, it has the following information:

   (a) The agent knows the blocked and unblock state of the cells that has been visited.

   (b) The agent knows the number of blocked cells in the neighbours that has been visited.

   (c) The agent can infer whether some cells are blocked or unblocked using the equations that can be generated from the number of blocked cells in the neighbourhood.

   We have found a way to encode this data in neural network and use this processed data to train the neural network that will mimic agent 4.

For agent 4, we initially have the current position of the agent, the equations that uses the number of blocked cells in the neighbourhood of x. To encode the current position of the agent, we multiply the current cell of the agent by 100 and since the agent can only move to the neighbouring cells, we multiply the neighbouring cells of the current position by 25. We increase the weights to give more importance to the neighbouring cells, since those cells are the cells where the agent will move in the next time step.
Also, the since we are working on 50×50 grid, we flatten the processed data before giving it to the neural network layer. Since, we are flattening the input before giving it to the neural network, the neural network should learn the knowledge that the agent is present at the position with maximum weight and the large weights (multiplied by 25) are it's neighbours. Hence, to prepare the input matrix from the given data we do the following:

(a) All the cells that the agent has already visited and those that are sensed unblocked are given the weight of 3 and the cells that are visited and sensed blocked are given the weight of -1.

(b) Also, all the cells that are inferred to be blocked are given the weight of -1 and all the cells that are inferred to be unblocked are given the weight of 3.

(c) All the cells that are still undiscovered are given the value 0.

(d) Also, the current position of the agent is multiplied by 100 and the values present in the 4 neighbours of the agent is multiplied by 25.

We pre-process the current knowledge of the agent, according to the above steps.

OUTPUT STATE SPACE
The output state space has 4 output variables. Each output variables represents the probability that the agent 4 will go in the corresponding direction. The agent can go in 4 directions - left, right, up and down. These 4 directions can be encoded as (0, 0, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0) and (1, 0, 0, 0). The next step of the agent is encoded and hence an 2d output vector is generated.

LOCAL INFORMATION vs GLOBAL INFORMATION

We can also use only local information instead of global information. For 50×50 grid, can also take the 11×11, with the current position of the agent at the center of this 11×11 splice of the grid. Also, we can add weights to the cells that have minimum distance from the target. The Neural Network can utilise this local information as follows:

(a) The cell at the current position of the agent is multiplied with weight 100.

(b) All the cells that are in the 4-direction of neighbourhood of the agent is multiplied by weight 25.

(c) The cells that are present at the minimum Manhattan distance from the target are multiplied by 30.

We can process input grids as described above and we can model the Neural Network of the agent.

2. **How are you defining your loss function when training your model?** Since, we are trying to predict the action variable which can belong to multiple classes, we tried to use the loss functions that are generally used to train the multi class problems. The loss function we tried are:

   (a) Categorical Cross Entropy

   (b) Sparse Categorical Cross Entropy

   (c) Kullback Leibler Divergence Loss

   Among the above loss functions, Categorical Cross Entropy gave the best performance. The loss that occurs is expressed using Categorical Cross Entropy as follows:

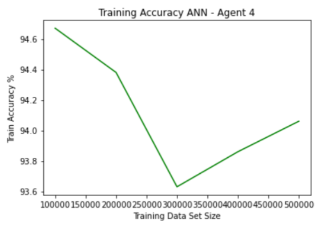
$$Cross\ Entropy\ Loss = -\sum_{i}^{C} t_i * log(f(s)_i) \tag{3.1}$$

   where C = *number of classes*
   $f(s)_i$ = *activation function*
   t_i = *an element of the target vector*

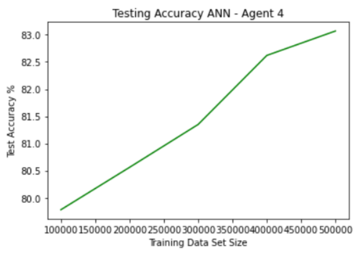
3. **In training, how many episodes on how many different Grid-Worlds were necessary to get good performance of your model on the training data?**

   Artificial Neural Network Architecture

   In case of ANN, the training of the agent happens for 20 epochs.We trained the agent using 10 lakh data points in the batch of 32 data points. We generated the 10 lakh data points from 1000 grids. The following graph shows the testing and training accuracy as we increase the training data examples:



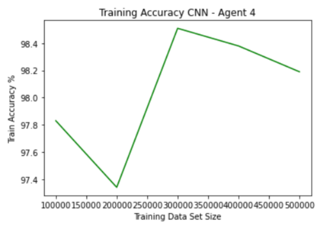
(a)



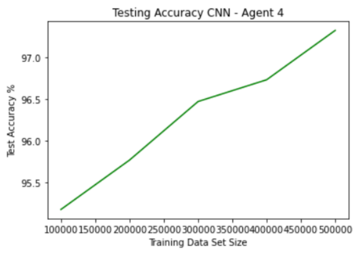
(b)

Figure 3.1: Train and Test accuracy of ML Model (using ANN) of Agent 4 vs amount of training data for Agent 4 (a) Train Accuracy (b) Test Accuracy

Convolutional Neural Network
For Convolutional Neural Network, the model was trained for 5 epochs. We trained the models from 1 lakh data points to 5 lakhs data points. The following are the graphs for CNN that indicates the Training and Test Accuracy for Agent 4 when we use CNN model. As similar to the data used in ANN model, we generated the 1 lakh data points from 1000 grids and 5 lakh data points from 2500 grids.



(a)



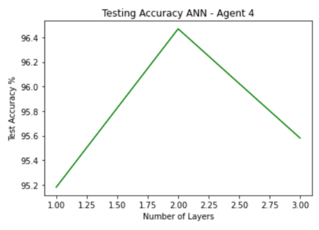
(b)

Figure 3.2: Train and Test accuracy of ML Model(using CNN) of Agent 4 vs amount of training data for Agent 4 (a) Train Accuracy (b) Test Accuracy

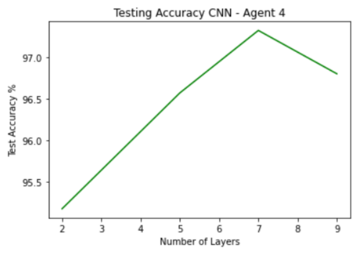
4. **How did you avoid overfitting? Since you want the ML agent to mimic the original agent, should you avoid overfitting?**

   Overfitting occurs when training accuracy increases and testing accuracy decreases. To avoid overfitting, first we increased the number of data points in the training dataset. As we increased the number of data points we saw that training accuracy decrease and testing accuracy increases. From this observation, we can conclude that the model becomes less overfit and the model becomes better.

   To further decrease the effect of overfitting, we also gradually increase the number of layers and at one point the testing accuracy was decreasing as the layers were increasing. So at the point where the testing accuracy is maximum, we decided that to be the number of layers.



(a)



(b)

Figure 3.3: Test accuracy of ML Model of Agent 4 vs Number of Layers (a) ANN Architecture (b) CNN Architecture

Further to decrease the overfitting, we added the dropout feature, which proved to be useful to avoid overfitting. In dropout, some of the neurons are dropped during training. So, when we drop different nodes in the neural network, we are training different neural networks. So, the dropout procedure is like averaging out the effects of large number of neural networks. Hence, the dropout feature serves the purpose of **Regularisation**.

5. **How did you explore the architecture space, and test the different possibilities to find the best architecture?**
   To explore the input features that gives the best performance we tried the following things:

   Initially, we gave value 1 to all the unblocked cells and 0 to all the blocked cells. This will produce a feature vector of size 2500×1, and along with that we passed the current coordinates of the agent. Also, when the agent goes to an unblocked cell, we multiplied the number of blocked cells to the corresponding cell. With this feature vector, we were not able to achieve the testing accuracy of more than 60%. Then, we kept the whole feature vector same, but instead of giving the current position of the agent as extra inputs, we multiplied the current position of the agent by 100 and all the neighbouring cells of the current position of the agent by 50. This input feature helped tremendously to increase the accuracy of the model. But finally after some more tweaking we finally decided to go for the input features described in Que 1.

   For exploring the architecture space for Neural Network, our main focus was to increase the testing accuracy as much as possible. To increase the testing accuracy, we tweaked the model to be built such that the training accuracy increases as much as possible. But we made sure that increasing the training accuracy does not overfit the model, hence we also observed the testing accuracy for the models that gave best training results. A summary of how we tried different possibilities is as follows:

   (a) Initially we decided the processing of input grid. This procedure is described in Que. 1.

(b) After getting the input, we increased the number of layers until we come across overfitting.

(c) After fixing a number of layers, we then increase the number of input nodes until we come across overfitting.

(d) In addition to the above process, we also include dropout in the Neural Network to avoid overfitting.

Using the above strategy, we chose the model that gets the most testing accuracy.

```
model = Sequential()
model.add(Dense(1250, input_dim=2500, activation='relu'))
model.add(Dense(1800, activation='relu'))
model.add(Dense(1800, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1800, activation='relu'))
model.add(Dense(1800, activation='relu'))
model.add(Dense(1250, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

Listing 3.1: Final ML Agent 4 ANN model

```
input_shape = (50,50,1)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
    input_shape=input_shape))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

Listing 3.2: Final ML Agent 4 CNN model

6. **Do you think increasing the size or complexity of your model would offer any improvements? Why or why not?**
We could increase the size or complexity of the model in two ways:

(a) Increase the number of layers in the network : In this case, each layer is having an activation function that induces non-linearity in the prediction model. Hence, as we increase the number of layers, the degree of non-linearity will increase which will lead to the increase in flexible functions that can exactly fit at each data point in the training data. This does not allow the model to generalise well over other points. So, increasing the number of layers will lead to overfitting.

(b) Increase the number of nodes in each layers: When we increase the number of nodes in a layers, we are increasing the number of parameters in a layer. This allows us to choose more features from the input of the previous layer. When we choose more features from the input, the prediction model becomes more complex. Since choosing more features might lead to features that are redundant or the features that aren't much useful, this will lead to overfitting. Since increasing the number of features will also allow the function to be flexible enough to fit to all the training data points, which will not allow the model to generalise. Hence, increasing the number of nodes in a layer will lead to overfitting.

Hence, from the above points, we can conclude that when we increase the complexity of the model, it will lead to overfitting. Now we prevent overfitting by the following ways:

(a) Early Stopping - We stop training when in the validation dataset, the error increases as we increase the number of epochs.

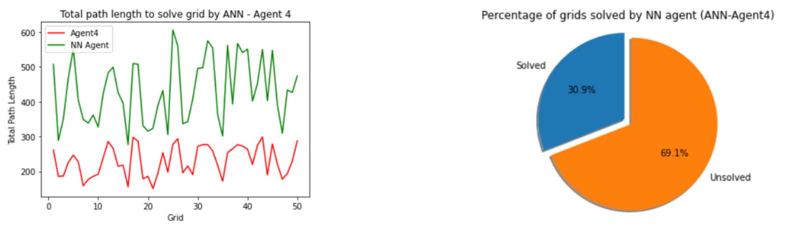
(b) Adding layers of Dropout: As we add layers of dropout, the model tends to generalise as explained in Question 4, and hence adding dropout feature to the layers will be useful to avoid overfitting.

(c) Increase the number of data points: Let's say a model overfits the data, then when we increase the number of data points and we do not increase the complexity of the model it will be more difficult to overfit the data to the model.

CNN Architecture

The CNN Architecture training for this agent proved to be another example for the Occam's Razor which tells that often the simplest explanation is the best explanation. When we trained the Convolutional Neural Network, the model gave the testing accuracy of 97.6% which was very good in relative to the ANN performance. We tried for 3 different layers for CNN and we choose the model that gave the best accuracy.

7. **Does good performance on test data correlate with good performance in practice? Simulate the performance of your ML agent on new gridworlds to evaluate this.**
In case of Agent 4, we found that good performance on test data did not correlate to the good performance on new gridworlds. We calculated the performance of the ML model using 3 metrics: accuracy, path length and number of grids that the agent is able to solve. We got the following graphs for our proposed ANN model:

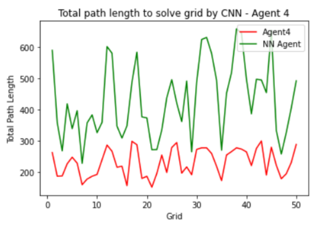


(a) (b)

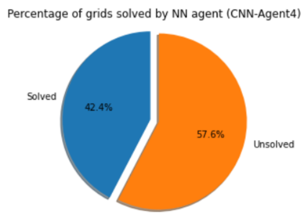
Figure 3.4: Performance of Agent 4 ANN architecture on different gridworlds (a) Path Length (b) Solvability of Grids using Agent 4

From the above graph, we can conclude that the performance of the ML agent is not much better than that of the previous project's agent. This is because the machine learning model tries to mimic the actions of agent but due to the less learning ability of agent, it cannot perfectly mimic the agent and sometimes take wrong decisions. This wrong decision leads to bad results.

Convolutional Neural Network:
Similar to the performance of ANN, CNN also does not perform good on new grids. This is probably because the training on samples assumes that each sample is independent of each other. But, when the agent moves across the grid, the next input actually depends on previous input. Hence the CNN is not able to model this aspect of the problem. It assumes that each training example is independent of each other. This actually requires the modelling technique of time series data since the movement of agent can be seen as dependant on the previous input in terms of time. The performance of the agent is as follows:



(a)



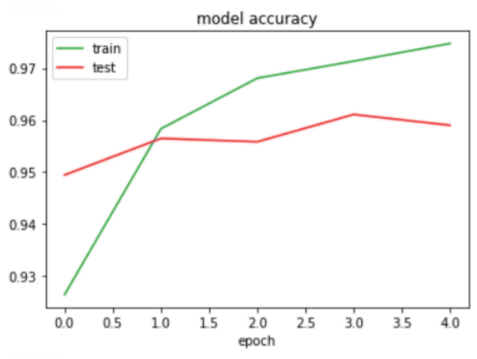
(b)

Figure 3.5: Performance of Agent 4 ANN architecture on different gridworlds (a) Path Length (b) Solvability of Grids using Agent 4

From the above graphs, we ran the ANN agent on agent 4 on the grids and we found that the Project agent easily beats the ML agent which indicates that the decisions taken by ML agent is very bad in practice. Moreover, we check whether the agent 4 is able to solve the grids that are solved by agent 1, we found that the grids that the ANN of Agent 4 is only able to solve 35% of the grids that ANN of agent 1 is able to solve.
After that we calculated the number of exact actions that the project agent 4 takes as compared to the ML agent in new gridworlds. In this case, we found that the ML agent is only able to mimic 45% of the actions.
Also, from the graphs of ML agent 1 and ML agent 4, we can see that ML agent 4 performs better than the ML agent 1.

8. **For your best model structure, for each architecture, plot a) performance on test data as a function of training rounds, and b) average performance in practice on new grid worlds. How do your ML agents stack up against the original agents? Do either ML agents offer an advantage in terms of training time?**

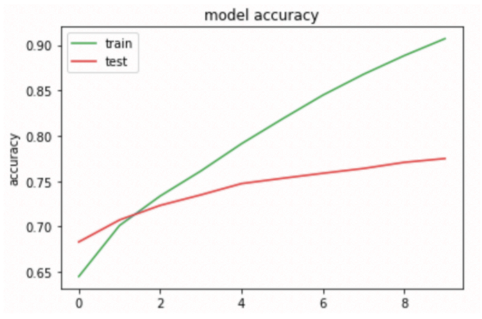
Below are the graphs for the ANN agent.



(a)

Figure 3.6: Performance of Agent 4 CNN architecture during training

The ANN model is trained for 10 epochs. We can see that as epochs increases the accuracy increases and hence the ANN and CNN architecture of the agent can mimic the project agent 4. But we can see that the CNN architecture performs better than ANN architecture. This proves that the local information improves the performance than encoding the global information.



(a)

Figure 3.7: Performance of Agent 4 ANN architecture during training

The above graph contains both the training and testing accuracy for ANN and CNN architecture of Agent 4.

Average Performance of Agent in new gridworld? How do your ML agents stack up against the original agents?

We can see that the performance of all the architectures in the new grid-world is bad and even though they can be trained to mimic the agent in the projects.

Also, in comparison to the ML agents against the project agent, we can see that the ML agents do not perform better than the project agents. This is evident from the number of grids that the agent can solve and among the grids that the agent can solve, we can see that the path length is not better than the project agent. Hence, the performance of the project agent is better than the ML agent.

4

Bonus 2

We also tried to imitate Agent 8 of Project 3 using the Architecture and feature vector as follows.

```python
input_shape = (50,50,1)
model = Sequential()
model.add(Dense(3000, input_dim=5000, activation='relu'))
model.add(Dense(2000, activation='relu'))
model.add(Dense(2000, activation='relu'))
model.add(Dense(1500, activation='relu'))
model.add(Dense(1500, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

Listing 4.1: Final ML Agent 8 ANN model

We used the feature vector similar to that for agent 1 and agent 4 for the first 2500 elements in the feature vector and the next 2500 elements represented the value of the utility of the corresponding cell in the grid. But this did not work out well and we obtained a test accuracy of 65.6%.

The input space for this agent is 2500×1 vector, which includes the p/d values. The p represents the probability of the current cell containing the target and the d represents the distance of the cell from the assumed target. Also, the current position of the target is multiplied by 100 and the neighbours of the current position of the agent, is multiplied by 25. After training the ANN agent by 10 epochs we go the accuracy of 65.6%, which is very good considering that the model has to model using the probability values. Since the probabilities are very small quantities, it might happen that the values are very small that the neural network won't be able to identify so to make the value of probability significant we multiply the value by 10000. So, we trained this ANN by 65.6%. This will only be able to mimic the project agent, but when we try to run this agent on original grid, it only moved to one direction which is very biased. The reason for this biasness is that since, the priority of movement of the agent is as follows: right, left, up and down. We trained the model using 300000 data points and we felt that these data points are not enough to train the dataset. Also, we could have tried CNN but training in CNN taked significant time and computation which was insufficient for this project even if we used the university resources to the fullest.