**Project 1 (Agent 1)**

Question 1:

To start off, we had to find a way to feed the neural network relevant information about how the agent made decisions based on the state of the agent/board and its knowledge of the board. We had to simulate what the agent “sees” at every given point in time while traversing the maze, and we had to put it into a representable form for the NN to learn from.

We knew that the agents never had global information about the maze (i.e. the agent didn’t know the location of every block in the initial state, but it would discover them as it went along). We had to find a way to simulate the “nebula” like the professor discussed. Thankfully, the way our knowledge base was organized and how we updated it, we were able to simulate this fairly well (if we had run into a block, we would make sure that it made it into the data given to the NN, but otherwise, it was an “unknown” cell).

Essentially, our input space was modeled just like the structure of the maze itself (a matrix of dimension rows by columns), and for each “cell”, we gave a corresponding value for the knowledge we had about the cell. So, 0 would mean an unknown or visited grid, 1 means the location of the agent, 2 means the location of the goal, and -1 represented the location of known blocks in the gridworld (represented by “X” upon data export from the Java program and then later translated to the necessary integer value in Python before training).

From there, we had to decide on our output space, but this was relatively simple. We represented the output space as a vector corresponding to the different directions the agent could take. For example, [1, 0, 0, 0] would correspond to an up movement, [0, 1, 0, 0] would correspond to a down movement, [0, 0, 1, 0] for left, and [0, 0, 0, 1] for right.

Question 2:

We defined our optimizer as the Adam optimization algorithm, which is an optimizer that can be user to update the networks weights iteratively based on the training data (refer below).

model.compile( optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'] )

Adam is different from the classic stochastic gradient descent, which maintains a single learning rate for all weight updates and it doesn’t change during training. Instead of only adapting the parameter learning rate based on the average first moment, Adam also adapts the average of the second moments of the gradients (uncentered variance).

We set out to define our loss function as categorical cross-entropy. This loss function is suitable for multi-class classification models where there are two or more output labels, which in our case we have 4 different labels for the 4 different directions the agents can travel in. The output label would be assigned as one-hot category, encoding values in the form of 0 and 1s. This output label (as integer form) is converted into categorical encoding using keras.

Lastly, the metrics we used is ‘accuracy’, which according to keras, we would be dealing with BinaryAccurary, CategoricalAccuracy, and SparseCategoricalAccuracy. The accuracy would be converted based on the loss function and the model output shape. Thus, in our case, the accuracy would be converted to the CategoricalAccuracy form.

**Only Full Dense Layers**

Question 3:

Question 4:

Question 5:

Question 6:

Question 7:

Question 8:

**At Least One Convolutional Neural Layer**

Question 3:

Question 4:

Question 5:

Question 6:

Question 7:

Question 8:

**Project 2 (Agent 3)**

Question 1:

Much like for the Project 1 neural networks, we had a similar process for extracting data from the agent while traversing the maze. The only difference here is that we also had to decide how to represent the inferences that the agent made and train the NN accordingly (we’ll call this the inference knowledge base). We were initially unsure about how to do this since some of these inferences are quite extensive, but we eventually decided that all of the inferences stemmed from the number of blocks sensed around the agent in a particular visited cell. There were other things that went into the inferences as well, but this information was encoded into the knowledge base passed to the Agent 1’s NN (which we’ll call the general knowledge base from now on).

Essentially, our input space for this project included both the general knowledge base from the first project part with Agent 1, and then the inference knowledge base modeled in an identical structure (a matrix of dimension rows by columns). Each cell corresponded to the number of neighbors sensed to be blocked around that cell (0-8, and “X” for an unvisited cell). In this way, we also simulate the nebula of unvisited cells rather than giving the NN all of the global information. The action space / output space was identical to Agent 1’s corresponding space.

Question 2:

Refer to the first part of the project’s Question 2. We defined loss functions and all other associated features in the same way.

**Only Full Dense Layers**

Question 3:

Question 4:

Question 5:

Question 6:

Question 7:

Question 8:

**At Least One Convolutional Neural Layer**

Question 3:

Question 4:

Question 5:

Question 6:

Question 7:

Question 8: