Milestone B Journal

Eric Itokazu
eitokazu3@gatech.edu

Abstract—This journal covers Milestone B for the ARC-AGI project in CS7637. This phase consists of setup of the project framework and ideation of how the project has been built to accomplish problem solving tasks from the provided dataset. This report covers the agent itself, an analysis of the performance of the agent, and discussion of various agent benchmarks and paths forward.

1 BACKGROUND

As a refresher, the ARC-AGI problem set is a benchmark to quantify the performance of an AI model's problem solving capability. Below is an example of an ARC-AGI problem with the inputs on the left and solution on the right.

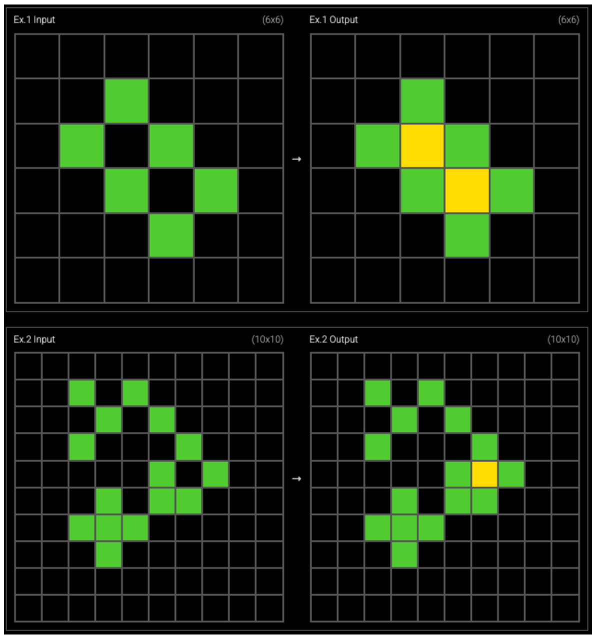

Figure 1—Example problem with input and output from ARC-AGI website.

The problem above illustrates one simple example of the ARC-AGI problems. In this, a human can interpret that given each input, the output displays the same tiles as the input, but also highlights the black tiles that are surrounded by green tiles yellow.

2 ARCAGENT - FUNCTIONALITY

For this phase of the project, the goal was to build an agent that can solve 6 of the 16 problems provided correctly. This seems like an easy milestone, but the steps involved to build a logic based decision-making agent are highly complex. Although this agent could be built to achieve the goal if different items were directly instructed, I wanted to start the framework of a model that was scalable. Therefore, there were multiple phases and components to the build.

2.1 Setup & Input Processing

For the first step in the agent's process, the ArcAgent class is initialized and calls to initialize a transformations class, a case memory class, and a feature extraction class.

The ArcAgent class is the primary working class of the agent. In this, the model runs through the processing, training, and transformation trials of the given problem. For each problem, the ArcAgent class will work through the provided information and record each case along with features and results of the trial, which the agent stores in their respective classes for reference and learining.

The helper classes include a FeatureExtractor class which gathers many properties about the inputs such as size, colors, counts, etc. as well as more specific features like vertical and horizontal symmetry, and output feature relations to input features. Alongside this, the Transformations class contains methods of logic transformations as well as specific case-related transformations that can be applied if a criteria is met in the feature analysis. Some of these functions range from basic rotate and mirrors, to midline splitting and sub-splitting methods to divide the input into smaller sections to be analyzed independently. Finally, the CaseStorage class is where the agent will store the relevant information including the inputs, outputs, success/failure, and vectorized feature list to be used in later tests.

2.2 Solving Process

The designed agent uses an ensemble of different methods to test each input case. First, the agent processes the input of the training data and extracts as many features about it as possible, and repeats the same for the output of the respective input and stores the results in CaseMemory. Once the model has ingested all the training examples, it then iterates through a series of checks, generating all options and testing them by applying some transformation and checking to see how far off the resultant output is from the actual output. This generate and test path first iterates through simple transformations to see if the input is able to match the output, then passes through a specific case to detect bisections. If a criteria is met for a possible bisection case (e.g. length = 2*width+1 AND input_length != output_length) then it branches off into a different process to evaluate the grid for possible bisection comparisons and symmetries. If the prior tests do not yield correct results, the model will continue onto a case-based reasoning system, evaluating the vectorized input/output features and determining the most similar examples from prior data to determine which path to follow. This process will return the examples with the three highest degrees of similarity based on their eigenvectors and implement the same transformation techniques used on the past example to the current problem. Ultimately, the agent then returns its best answers based on likelihood and saves the information generated before continues to the next problem.

3 ARCAGENT – PERFORMANCE

The agent successfully solved 6 of the 16 training problems in the milestone B set, and 5 of 16 problems on Gradescope. For this milestone, I decided to tackle the problems in a human-like way by targeting a few example problems with the most overlap and most common themes. In this case, the most common themes were simple rotations, and bisection problems. Two examples of these types of problems are shown below in figure 2.

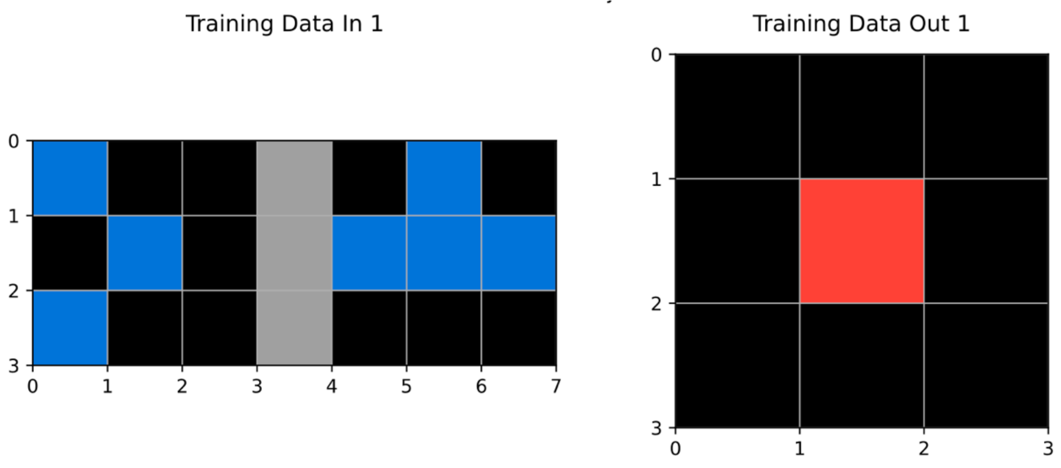

Figure 1— Example problem of output relating to input via bisection

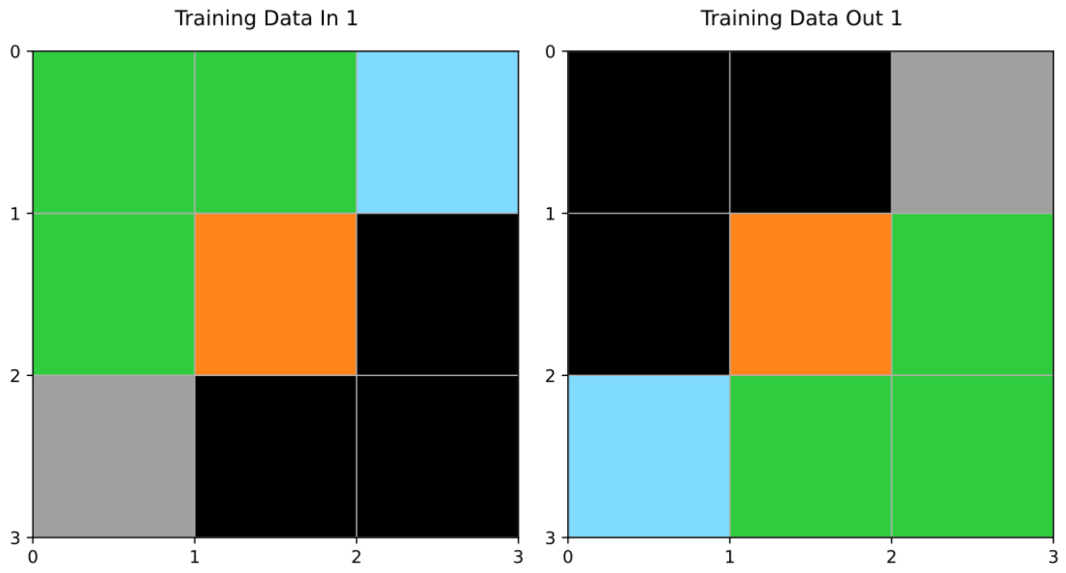

Figure 2— Example of problem with solution of simple rotation of 180°

Figure 1 is solved by considering the input as two different grids separated by a bisecting column. This creates two grids with the same size as the output, then a Boolean AND operation is applied to both to return the resultant grid. This was a common problem in the training dataset, so I had the goal of solving all problems that had a bisector. Similarly, the problem shown in figure 2 is solved with a simple 180° rotation, so basic transformations were also targeted in this pass.

Since these were the two main problem types that were targeted, the agent was not equipped with enough tools and preset learning capabilities to handle problems with different solution paths. Therefore, the test dataset likely had less of these types of problems than the training set. In addition, I think the case-based reasoning solution path has a heavy bias toward these two types of problems since it is the only path that the agent is only trained to handle solution paths in this format. In the future, I think the case-based reasoning method will become much more useful as the agent builds its knowledge base and is equipped to handle other various transformations and ensemble methods.

4 ARCAGENT – EFFICIENCY

The agent is able to run through the problems very quickly, taking less than 0.5 seconds to run through the entire Milestone B training set. At first, the agent took over a minute to run and ate up much more memory due to a few heavy imports from scipy and sklearn. Once pinpointed, I was able to rewrite the basic functions used such as eigenvector calculation in base python and limit local variable creation, effectively cutting out all the bulky imports and eliminating memory issues.

5 ARCAGENT – NEXT STEPS

To improve on the problems that the agent struggled with in this portion, I plan on incorporating more problem solving tools and strategies to achieve the desired outcome. We know the examples were part of the easier set, so it will be critical that the agent can iterate through to achieve the desired results. This means implementing more augmentation functions for the actual step transformations, but also making sure the agent is assessing the next steps in a solving routine, similar to the generate and test components. Additionally, I need to prevent the agent from over prioritizing any one method. This could happen especially in the case-based reasoning component if there are many solved cases that use a similar path. Then, the agent could potentially incorrectly suggest only that reinforced path.

6 ARCAGENT – GENERALIZATION

To generalize the model, I plan on incorporating a few features. First, the code needs to be more well-rounded. The function set I have is tailored to the general transforms and the bisection case, but even simpler tasks like color changes are often incorrect. The code needs to have the tools to achieve most general individual steps toward the goal defined. From there, the agent will likely need a revision of the features vector. I am concerned that the vectorized lists could get large as tools and recognition triggers are introduced. I also plan on making the agent more iterative, using principles from ends-means processes to break the task down into mini steps. I think this will actually be a good fit as the differences in arrays can be calculated easily.

7 ARCAGENT – FEEDBACK

If I could request a topic of feedback, I would like to know how others are setting up the transformation process. I am defining each simple small step one by one, but are other people doing the same? At a more granular (individual tile) level? More generally and using some probability engine to make guesses of the output? There are so many methods to use that can all achieve the same goal, so I'd like to see if others have the same thought process as me.