Solving 1D Abstraction and Reasoning Corpus using Reinforcement Learning

Liyew Woletemaryam (20231210)
GIST
woleteml@gm.gist.ac.kr

Marha Midhatiey (20231211)
GIST
marhamidhatiey@gm.gist.ac.kr

***Abstract*—This study aims to devise a new approach of prediction and classification of given datasets using rule-based representation learning and Reinforcement Learning technique (DQN). Incorporating rule-based representation learning provides an alternative technique that uses mathematical and logical reasoning to identify links between inputs and outputs. For evaluating our model, the 1 dimensional abstraction and reasoning corpus (1D ARC) dataset has been used. Our paper demonstrates the performance of our model by evaluating its accuracy on unseen, complex, combined rules data. Our findings highlight the importance of rule-based representation learning in understanding the complicated patterns hidden in 1D arc datasets. The results shows that our model converge faster by decreasing the total loss and increased the rewards which stabilizes as training moves, indicating effective learning and improved prediction accuracy. By bridging the gap between theoretical discoveries and practical implementations, this study not only contributes to the developing field of categorization, but also provides the framework for future advances in data analysis approaches.**

***Index Terms*—Reinforcement Learning, DQN, Abstraction and Reasoning Corpus (ARC), Rule based Representation, Mathematical and Analogical Reasoning**

I. Introduction

In recent years, advancements in artificial intelligence (AI) have been significantly driven by the integration of logical and understandable techniques. The Abstraction and Reasoning Corpus (ARC), developed [1] by François Chollet, serves as an important benchmark for comparing the intelligence between human and machine learning. To that purpose, he draws on the work of developmental psychologists Spelke and Kinzler (2007) on the theory of Core Knowledge to identify axes along which human-like intelligence could be evaluated. ARC tasks, which consist of 1000 image-based reasoning activities, assess abstract reasoning ability using concepts such as objects, goal states, counting, and basic geometry [2]. However, classic ARC tasks have presented substantial hurdles for AI models, with many techniques failing to reach good performance. A recent study of GPT-4 on ARC's two-dimensional input-output grids revealed minimal success, with just 13 out of 50 fundamental tasks resolved. This limitation was associated to the sequential nature of the text encodings used to represent objects in 2D ARC tasks. However, performance improved with a new benchmark consisting of one-dimensional array-like tasks (1D ARC), in which object-based representations significantly enhanced reasoning abilities. Despite numerous attempts to solve ARC since its start in 2019, numerous measures, including a recent GPT-4, achieved limited results [7]. This project suggests a novel way to address the ARC tasks challenges by merging Reinforcement Learning technique (DQN) with Rule-based Representation. Specifically, ARC requires one to find a technique consistent with a small number of certain input-output examples, with 2-5 input-output image pairings provided as training examples to learn the underlying technique and apply it to a new input to generate new output (unseen tasks) (see Figure 1). By implementing rules as (action), it would be able to find the best rules within the input output pairs provided and DQN to increase reasoning and problem-solving abilities.

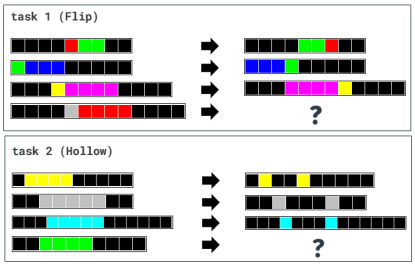

Fig. 1: Sample ARC Tasks. Two tasks (separated by square box) are shown. The goal is to deduce the new unseen output from the given examples.

II. Rules Representation Learning

A. Rule-Based Representation

The purpose of this rule-based system is to process input data with specified rules and compare the results to expected outcomes. The design and implementation of this rule-based system is by reading input and output pairs data from CSV files, defining logical rules and applying the rules to generate new unseen tasks, and validating the outputs. RBL method intends to automate the process of data transformation and validation in order to handle data more efficiently.

B. Defining Rules

To formulate the rules, an analysis of the input and output pairs were conducted to develop mathematical and logical

Identify applicable funding agency here. If none, delete this.

formulas that defined how they connected within one another. Total of 12 rules were defined with explanation on how it works :

- FLIP: Detection and Generation of Flipped Sequences
- HOLLOW: Create spaces between same colour pixels
- FILL: Filling Gaps
- MIRROR: Mirroring Elements Around a Central Point
- DENOISING: Removing of Non-Continuous Elements
- MOVE: Shifting Elements
- SCALE_DP: Scaling Down and Propagating Values
- PADDED_FILL: Padded Filling of Sequences
- RECOLOR_OE: Recoloring Odd and Even Length Sequences
- RECOLOR_CMP: Recoloring Based on Complementary Patterns
- PCOPY: Moving Even Numbers and Zero-Filling
- MOVE_DP: Moving Elements by Double Positions

C. Result Generation

Function Explanation : FLIP

- The Flip function is intended to reverse only the color pixels in a one-dimensional array (Example shown as in Fig. 1).
- Initial Check and Copy: The function starts by learning the input-output pairs and filtering out the non-zero elements from both input array and output array using list comprehensions.
- Variable Initialization: The filtered lists non-zero input and non-zero output are the variables that will be used for comparison.
- Check Reversed Order: The function then checks if the non-zero input list, when reversed, matches the non-zero output list.
- New Output Generated: Looping through arrays to check conditions and apply the rules, perform comparisons, and generate new outputs from the given input.

III. DQN Architecture

Our approach uses a Deep Q-Network (DQN) architecture to train the RL agent.The DQN algorithm initializes a neural network to approximate the Q-values for state-action pairs, facilitating decision-making. It uses experience replay to store and randomly sample experiences, improving learning stability and efficiency. During training, it iteratively selects actions based on an epsilon-greedy policy, balancing exploration and exploitation [5]. The network is updated using the Bellman equation to minimize the mean squared error between predicted and target Q-values. This process is repeated over multiple episodes to learn optimal rules for transforming input arrays into their desired output forms.

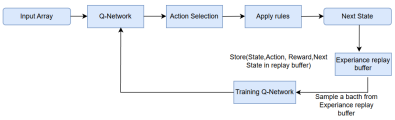

Fig. 2: DQN Architecture used for solving ARC tasks

A. Neural Network and Rule Definitions

In the initial part of the project, we defined the neural network architecture for the Deep Q-Network (DQN). The neural network, named QNetwork, consists of three fully connected layers. The first two layers use ReLU activation functions to introduce non-linearity, enabling the network to learn complex patterns from the input data. The input layer accepts the state size, which is the length of the input array (32 in our case), and the output layer provides Q-values for each possible action, corresponding to the defined rules. The network architecture is simple yet effective for this task, ensuring quick convergence and accurate predictions.

In addition to defining the neural network, we also specified the rules that the DQN would learn to apply to the input arrays. Three rules were defined: `move_dp`, `denoising_1c`, and `move_1p`. These rules perform different transformations on the input array. For instance, movedp shifts all non-zero elements to the right while preserving their order, `denoising_1c` removes isolated non-zero elements surrounded by zeros, and move1p shifts elements one position to the right. These rules form the action space for the DQN, and the network learns to select the appropriate rule to transform the input array to match the target output.

B. Hyperparameters and Training Setup

- In the second part, we focused on setting up the hyperparameters and the training process for the DQN. Key hyperparameters include the **learning rate (alpha)**, **discount factor (gamma)**, **exploration rate (epsilon)**, and its **decay rate**, among others. These parameters control the learning dynamics of the network, such as how much the network updates its weights in response to new data, the importance of future rewards, and the balance between exploring new actions and exploiting known ones [6]. The experience replay mechanism was also introduced, where past experiences are stored in a d-queue and sampled randomly to train the network, preventing over-fitting and ensuring stable learning.
- We also defined the reward function based on the Mean Absolute Error (MAE) between the transformed array and the expected output. This reward function guides the network in learning by providing feedback on the accuracy of its predictions. A negative MAE value was used as the reward to incentivize the network to minimize the error. The training process involves multiple episodes, where the network iteratively applies actions to the input array, stores the experiences, and updates the network weights using the sampled experiences. This setup ensures that the network gradually improves its performance and learns to apply the correct rules for various input configurations.

C. DQN Training and Application

The final part of the project involved the actual training of the DQN and its application to new data. We defined a training function, SQN training, which loops through a predefined number of episodes and steps within each episode. During

each step, the network selects an action based on an epsilon-greedy policy, applies the corresponding rule, calculates the reward, and stores the experience. The network is then trained using these experiences, updating its weights to better predict the Q-values for future actions. The epsilon value decays over time, reducing the exploration rate and encouraging the network to exploit learned knowledge. After training the DQN on the example datasets, we tested its performance on a hidden test input to evaluate its generalization capabilities. The function apply best rules was used to apply the best rules learned by the network to the new input array, transforming it to match the expected output. This step demonstrated the network's ability to generalize the learned transformations and apply them effectively to unseen data, validating the success of the training process. The transformed hidden test input showed that the DQN could accurately apply the appropriate rules, indicating the potential for broader applications in similar tasks.

D. Equations

- Given an input state vector $s \in \mathbb{R}^n$, where n is the state size (32 in this case), the forward pass through the neural network can be expressed as:
  1. First layer transformation:

$$h_1 = \text{ReLU}(W_1 s + b_1)$$

  where $W_1 \in \mathbb{R}^{64 \times n}$ is the weight matrix, $b_1 \in \mathbb{R}^{64}$ is the bias vector, and ReLU is the activation function defined as:

$$\text{ReLU}(x) = \max(0, x)$$

  2. Second layer transformation:

$$h_2 = \text{ReLU}(W_2 h_1 + b_2)$$

  where $W_2 \in \mathbb{R}^{64 \times 64}$ and $b_2 \in \mathbb{R}^{64}$.
  3. Output layer:

$$Q(s, a) = W_3 h_2 + b_3$$

  where $W_3 \in \mathbb{R}^{m \times 64}$, $b_3 \in \mathbb{R}^m$, and m is the number of actions (3 in this case). The output $Q(s, a)$ represents the Q-values for each action a.

$$a + b = \gamma \tag{1}$$

- The epsilon-greedy policy for selecting an action a given a state s is defined as:

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s, a) & \text{with probability } 1 - \epsilon \end{cases}$$

- The reward function based on the Mean Absolute Error (MAE) between the transformed state s' and the expected output y is:
  1. Calculate MAE:

$$\text{MAE}(s', y) = \frac{1}{n} \sum_{i=1}^{n} |s'_i - y_i|$$

  2. Reward:

$$r = -\text{MAE}(s', y)$$

IV. Result

The RL agent demonstrated promising results, achieving a significant success rate on a subset of ARC tasks.

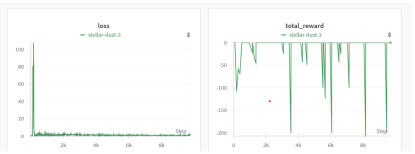

Fig. 3: Performance of the agent result the left one is for the loss and the right shows reward obtained per episode

The two graphs present the training performance of a Deep Q-Network (DQN) model tested on the 1D ARC task. The first graph on the left illustrates the loss function over the training steps. Initially, the loss is high, indicating large errors between predicted and target Q-values. However, the loss rapidly decreases and stabilizes, showing that the model is learning effectively and making more accurate predictions as training progresses. The second graph on the right shows the total reward obtained per episode. There is significant variability in the rewards, particularly early in the training process, which reflects the exploration phase of the epsilon-greedy policy. Over time, the rewards show some stabilization, although occasional drops suggest the model sometimes performs suboptimally, indicating room for further optimization and fine-tuning.

V. Conclusion

Reinforcement Learning offers a viable approach to solving the ARC tasks, with the potential to advance AI's generalization capabilities. Future work will focus on refining the rule representation and exploring more sophisticated RL algorithms to further enhance performance

References

[1] Chollet, François. "On the measure of intelligence." arXiv preprint arXiv:1911.01547 (2019).

[2] Xu, Yudong, et al. "Llms and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations." arXiv preprint arXiv:2305.18354 (2023).

[3] Li, Yuxi. "Deep reinforcement learning: An overview." arXiv preprint arXiv:1701.07274 (2017).

[4] Wiering, Marco A., and Martijn Van Otterlo. "Reinforcement learning." Adaptation, learning, and optimization 12.3 (2012): 729.

[5] Jang, Beakcheol, et al. "Q-learning algorithms: A comprehensive classification and applications." IEEE access 7 (2019): 133653-133667.

[6] Di Pasquale, Ricardo, and Javier Marenco. "Optimization meets Big Data: A survey." arXiv preprint arXiv:2102.01832 (2021).

[7] Tan, John Chong Min, and Mehul Motani. "Large Language Model (LLM) as a System of Multiple Expert Agents: An Approach to solve the Abstraction and Reasoning Corpus (ARC) Challenge." arXiv preprint arXiv:2310.05146 (2023).