Programmatically Interpretable Reinforcement Learning in a Robotics Domain

CS 703 Fall 2020

Salvatore Skare
University Wisconsin — Madison
Department of Computer Science
sskare@wisc.edu

ABSTRACT

In 2018, Verma et al. published a paper titled, "Programmatically Interpretable Reinforcement Learning", where they presented a program synthesis technique for solving reinforcement learning (RL) problems. A small functional language is introduced. Using deep reinforcement learning (DRL), solutions to a RL problem in the form of a program are synthesized. In the paper, this method of solving RL problems was applied to several test problems, including driving a virtual car through a track. This research investigates a real-world robotics application of this RL method. Learning from real data can often have different challenges than learning in a simulated environment. To test the efficacy of PIRL in a real-world robotics domain, an experiment was devised. The experiment is based on an early DRL paper published in 2003 from researchers at Oita University, Japan. In their experiment, they built a small robot with a CCD monochrome camera and four infrared distance sensors providing inputs to an actor-critic DRL model. The robot operated in a 70cm square area, and it's goal was to push a 7cm by 3cm box. This experiment was recreated using a custom built robot, training a DRL agent, then synthesizing a program using PIRL. The resulting program was roughly as effective at locating and pushing the box as the DRL agent.

1 INTRODUCTION / MOTIVATION

Deep Reinforcement Learning (DRL), which is reinforcement learning combined with neural networks, has been used to successfully allow robots to learn to perform tasks with no prior task knowledge, and no guidance from humans. In recent years, such tasks have included moving a three-jointed robotic arm to a target using only raw image data [4], and a quadrupedal robot learning to walk [1]. One issue with DRL is that the learned policy is a set of neural network weights, which are difficult for humans to interpret. Since it is often undesirable for a robot to execute an effectively unknown black-box policy, this aspect of DRL is less than ideal.

One way to generate human-readable policies is through program synthesis. In 2018, Verma et al. published "Programmatically Interpretable Reinforcement Learning", where they describe a method for synthesizing programs as policies for reinforcement learning agents [3]. One of the key insight of Programmatically Interpretable Reinforcement Learning (PIRL), is that trying to search for a program that optimizes the value from the reward is difficult because these reward functions often don't return a positive value until some objective is achieved, which makes it difficult to evaluate accurately over a single time-step. Instead of trying to maximize the reward function directly, PIRL instead relies on a pre-trained DRL model, which is used as a policy oracle. The program search then attempts to minimize the error between the output from the neural oracle and the synthesized program when evaluated on the same input.

The authors of the PIRL paper tested their method in several simulated environments, including the digital car racing simulator TORCS. The synthesized programs performed on-par with the neural policy and were human-readable. However, the PIRL synthesizer designed by Verma et al. was only tested in a virtual environment on symbolic inputs. For example, one of the most used inputs in the program synthesized for the TORCS track was the distance the virtual car was from the center of the track. In a robotics application that operates on perceptual data, no such data are available. In their conclusion the authors state that testing PIRL on real-world perceptual inputs is a logical next step for the research.

2 METHODS

To evaluate PIRL in a robotics domain, an existing DRL robotics experiment was recreated. The experiment chosen was performed in 2003 at Oita University, Japan [2]. In this experiment, a two-wheeled Khepera robot was equipped with a camera and 4 IR distance sensors. The goal of this experiment was to see if the robot, running a Deep Actor-Critic Reinforcement Learning agent, could learn to locate and push a box. Each episode lasted for either 50 time-steps, until the robot had pushed the box for 10 consecutive time-steps, or until the box was no longer visible to the robot. When the robot was pushing the box, as determined by the distance sensors, and both motors were moving forwards, the agent received a small reward of 0.018.

This experiment in particular was chosen for several reasons. First, it is simple to re-create the DRL agent used in the original experiment using modern DRL tools. Secondly, the objective to be learned is simple enough to be tractable, but still involves enough complexities to challenge PIRL on perceptual inputs. Specifically, the main challenging conditions under which PIRL has not previously been tested are:

(1) The input space is large and perceptual
(2) Inputs to the robot contain noise and are not always accurate
(3) Actions taken by the agent are not always successful (e.g. wheels can slip)

To recreate the experiment, first a two wheeled robot was constructed. This robot featured three ultrasonic distance sensors facing forwards, as well as a forward facing camera situated in a way similar to the robot in the original experiment. Next, an area of 120cm x 120cm was surrounded by a wall constructed of white card

stock paper 10cm tall. A 10cm x 4cm x 4cm box was also constructed out of black card stock paper. The area was lit from directly above by an LED light. It should be noted that the dimensions of this area were increased in order to maintain the same scale with the robot as the original experiment had with the smaller Khepera robot.

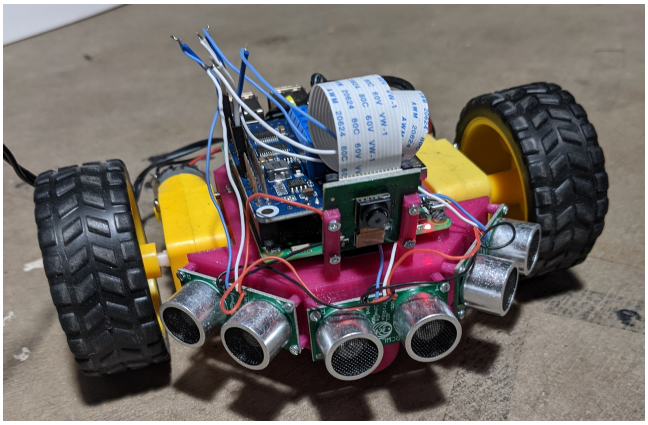

Figure 1: The robot that was constructed for this research.

Next, the DRL agent used in the original paper was implemented as closely to the original as possible. At each time step, an image was captured from the front-facing camera, converted to greyscale, and downscaled 5 times smaller than the original image dimensions of 640 x 480 pixels. Readings from each distance sensor were also collected and normalized. The image was flattened to a single dimension, and normalized as well. Both arrays were then concatenated, and this final array was passed to the RL agent as the input state. Because the camera captures more data than the camera used in the original experiment, the neural network was increased in size from the original as well, so that it had two hidden layers, of sizes 1024 and 128. Like in the original paper, the actor consisted of two output neurons, one for each motor, and shared the same hidden layers with the critic output neuron.

Much like in the original experiment, a small random value drawn from a uniform distribution in order to drive exploration during early learning. Unlike in the original paper, these random values were constrained to only positive values, in order to bias the motors towards going forwards to increase the rate at which the robot learned the task. In addition, the weights of the input layer of the neural network were initialized to a random value drawn from a uniform distribution in the range $[0, 1]$. The weights of the hidden layers were initialized to zero. The same temporal difference error function given in [2] was used for this DRL agent.

For the PIRL agent, a functional domain specific language was needed for program synthesis. The language created for this experiment had three data-types: *int*, *float*, and *array*. Arrays consisted only of floating point numbers. Operations for this language consisted of a subset of NumPy operations, as well as array slices. The input array was the only instance of an array that synthesized programs could access. Constant values weren't synthesized directly, rather the synthesized program representations contained symbolic constants, which were replaced with actual values by optimizing for values that minimized the distance between the sum of the difference between output of the program on all inputs in the history, and the output from the neural policy oracle on the same inputs. The optimization method used in [3] was Bayesian Optimization, this research instead choose to use a Tree-structured Parzen Estimator as a drop-in replacement. This was done because the implementation of the Tree-structured Parzen Estimator used was faster and more stable than any off-the-shelf Bayesian Optimization software available for python.

⟨*int*⟩ ::= IntConstant

⟨*float*⟩ ::= FloatConstant
| ⟨*array*⟩ '.' ⟨*floatOp*⟩ '()'
| ⟨*array*⟩ '[' ⟨*int*⟩ ']'

⟨*array*⟩ ::= Input
| ⟨*array*⟩ '.' ⟨*arrayOp*⟩ '(' ⟨*float*⟩ ')'
| ⟨*array*⟩ '[' ⟨*int*⟩ ':' ⟨*int*⟩ ']'
| ⟨*array*⟩ '[' ⟨*int*⟩ ':' ⟨*int*⟩ ':' ⟨*int*⟩ ']'

⟨*floatOp*⟩ ::= 'max' | 'argmax' | 'min' | 'argmin' | 'ptp' | 'sum' | 'all' | 'any'

⟨*arrayOp*⟩ ::= '__lt__' | '__gt__' | '__add__' | '__mul__'

Figure 2: Grammar for the robot DSL.

The program synthesis algorithm for PIRL, which the original authors refer to as "Neurally Directed Program Search" (NDPS), reduces the search space of possible programs by requiring a program sketch, which they formalize as a set of allowed programs under the sketch conditions. For this application of NDPS, a sketch was provided as a string containing a valid python expression, with holes to be filled by the synthesizer represented as `'{}'`. Another restriction on valid programs was that holes were only filled with *float*s. This restriction made it possible to constrain the output of the synthesized programs to arrays of length 2, the same output format that the DRL agent had, by providing a sketch of the right shape. These sketches were evaluated by serializing the synthesized code for each hole to a string containing the representative python expression, replacing the holes in the sketch string with these strings, then passing the result to `eval()`.

For program search, an enumerative approach was chosen. The NDPS algorithm requires several functions that are implementation specific to be implemented. The first of these functions, `create_histories`, was implemented by running the DRL agent for one episode, and returning a list of input/output pairs as tuples. For the `initialize` function, all programs of size N are enumerated, where N is a user supplied integer that is greater than or equal to the number of holes in the sketch. For the experiments presented in this paper, N was chosen as 6. After this set of programs is synthesized, the programs are evaluated on all the inputs in the set of histories. The error for each potential program is computed as the sum of the distance between the output of the program and the saved output from the DRL agent for each example in the set of histories. The `initialize` function returns the potential program with the lowest error.

The `neighborhood_pool` function is supposed to return programs similar to the current candidate program. For this implementation, similar programs are defined as programs where one hole has been replaced with a *float* that is one size larger than the previous *float*. This method of neighborhood search is simple, as well as having the benefit of prioritizing shorter programs automatically. Lastly, the functionality of `update_histories` is described by the authors of [3] as "heuristically picks interesting inputs in the trajectory of the learned program and then obtains the corresponding actions from the oracle for those inputs". For this implementation, the heuristic used to decide to add an input was to compute the distance between the input and all inputs in the set of histories, and if each distance was above a given threshold, to add the input to the histories set, as well as the DRL agent's output on the input.

To evaluate the efficacy of PIRL in this problem domain, the DRL agent was first left to train for 1000 episodes. After each episode the robot was reset by executing a routine of first moving in a random direction, then backing away from any objects in front of the distance sensors, and finally rotating in-place until the box was located in the image from the camera. Determining if the box was visible in this step was achieved by using conventional computer-vision strategies. The image was first binarized, anything above the white card stock barrier was cropped out, and if an object of sufficient area is in the resulting image, it is assumed to be the box. This method of box detection was also used while running each episode to determine if the box was still visible. If the robot happened to push the box too close to the edge of the area, it was moved back to the center via human intervention after the episode had finished.

After the DRL agent trained for 1000 episodes, a program was synthesized using the PIRL implantation described. The sketch given was:

```
np.array([{}, {}])
    .__add__(np.array([{}, {}]).__mul__({}))
```

When evaluated, this sketch returns an array of two *float*s, corresponding to left and right motor values. This particular sketch was used to allow the synthesis to approximate a proportional error correction, by choosing two initial values for each motor, then adding a scaled error term to each. The NDPS algorithm runs until the reward for the current candidate program from a single episode is less than the reward for a single episode for the previous candidate program. The previous candidate program is then returned as the final synthesized program. After each episode using a candidate program, the robot was reset in an identical fashion as between episodes with the DRL agent.

The processor on the custom-built program was much more powerful than the one used in the original experiment, allowing the DRL agent to be trained on-board the robot. Program synthesis was still to computationally intensive to be done on the robot in a time efficient, so the synthesis step of the research was run on an Intel Core i7 processor, clocked at 2.8 GHz. The robot was remote-controlled by this computer in order to evaluate candidate programs.

Once both agents had a final policy, they were both tested by running ten episodes with each, and collecting the reward from each episode. This gave a metric to compare the performance of the DRL agent to the performance of the PIRL agent. All code, as well as the STL file for the robot chassis, are available at https://gitlab.com/saljs/cs-703-project/, licensed under the GPL v3.0 free software license. This repository also contains details on how to construct the robot used for this research.

3 RESULTS

Although there was only enough time to let the DRL train for 1000 episodes, one-fifth of the training that took place in the original robot experiment, it was clear that the robot learned how to locate and move the box from a multitude of different angles and locations. However due to the reduced episode count, there were also certain starting positions that the neural policy failed to find the box for. Because NDPS is dependent on the neural policy as an oracle of "correct", the policy synthesized by PIRL is limited by the performance of the DRL agent.

Qualitatively, it was possible to see that the DRL agent learned to locate and push the box by observing it during an episode. An example of this is available at https://youtu.be/lhxO0_aBv6Q. Since it was also desirable to have quantitative data that show the DRL agent learned the task, the amount of reward from each episode was recorded. By plotting these data and adding a trend-line, it is also possible to see an upward slope over time, also indicating that the average reward increased as the robot continued to learn.

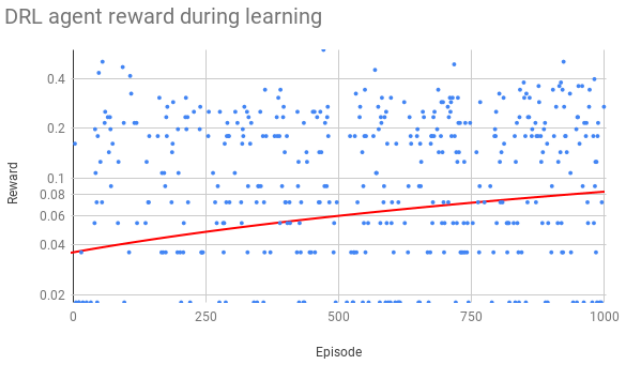

Figure 3: A logarithmic plot of reward received per episode, with a linear trend-line.

After the DRL agent was trained, the PIRL agent was used to synthesize a program policy. After six iterations of the program search, the following program was returned:

```
np.array([0.878, 0.381])
    .__add__(np.array([0.318, inputs.any()])
        .__mul__(inputs.__mul__(inputs[3403])[2870]))
```

This program looks specifically at two pixels in the input image. The locations of these pixels highlighted in an example image is shown in Figure 4. Similar to the DRL agent, qualitative observation of the robot running the synthesized showed that it was capable of locating, driving to, and pushing the box from several starting positions, and fails for others. An example episode of the PIRL agent is available at https://youtu.be/1ajbrXXjhew.

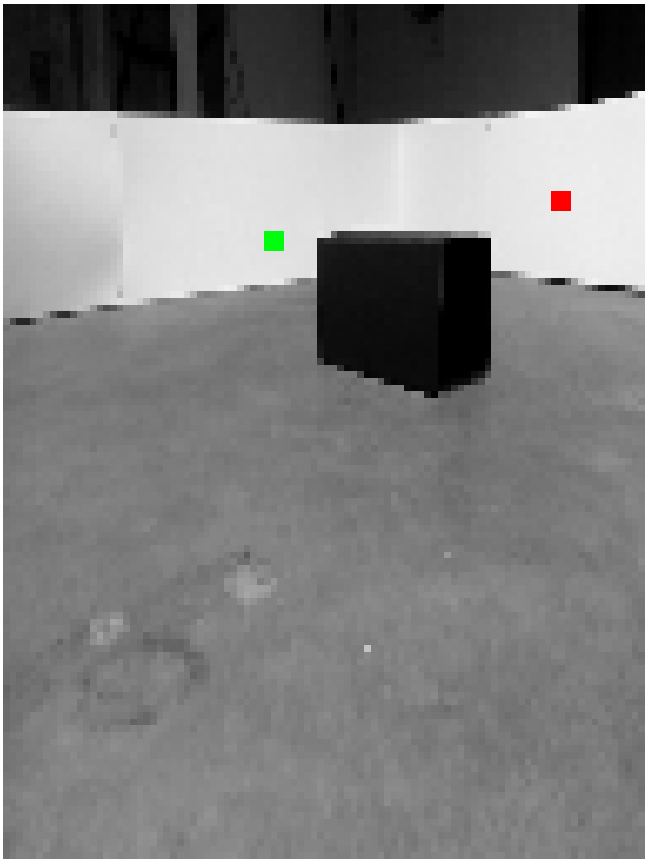

Figure 4: An example image from the robot's camera, with pixels 3403 and 2870 highlighted in green and red respectively.

The program generated via NDPS and the DRL agent were both tested by running each final policy for 10 episodes and collecting statistics on the reward received. These statistics are summarized in the table below.

Agent	Episodes where box was found	Average reward	Average reward for episodes box was found
DRL	4	0.0684	0.4
PIRL	4	0.108	0.4

4 DISCUSSION

From the data collected, the NDPS algorithm synthesized a program that performed nearly identically well to the neural policy from the DRL agent. Because the DRL agent didn't act as a perfect oracle, the PIRL agent was limited by the performance of the DRL agent. It is however, extremely promising that the synthesized program performed nearly as well as it could possibly could given this limitation.

The main proposed benefit of PIRL, interpretability of the policy was not easily apparent in the program synthesized for this application. The synthesis did not result in something approximating a proportional error correcting function as it was expected to. Analyzing the outputted program it is possible to try and determine how it makes decisions. For most inputs, `inputs.any()` will evaluate to 1.0, which leaves the two pixels the program looked at specifically. The program ends up multiplying the value of both pixels together. It is still unclear how this is helpful towards stabilizing the robot's motion toward the box. However, it is still clear from observation of runs of the program that it does work as expected.

There are three possible explanations for the lack of interpretability in the output program. First, because the DRL agent did not have enough training to find a policy that works for all possible inputs, it was possible that the neural policy was solving the problem of locating the box in a sub-optimal heuristic way, which the PIRL agent then ended up copying. The second possible explanation is that because the synthesized program is highly dependent on the given sketch, that a different sketch would have produced much more readable results. Finally, it is possible, although unlikely, that the synthesiser came up with a novel, efficient solution that is hard to understand because it relies on some information or intuition that isn't readily apparent.

Synthesis was also attempted on a more detailed sketch with fewer holes as well. This sketch worked very similarly to the conventional computer vision approach taken to locate the box between episodes, by binarizing the image, cropping out anything above the barrier, then locating the center of mass of the black pixels corresponding to the box. The holes left for synthesis in this sketch were the threshold value for binarization, and the error correction scaling factor. Surprisingly, the synthesizer was unable to generate a viable program under the constraints of this sketch. This is again most likely because the neural policy was making decisions using a very different heuristic than proportional error correction, making it impossible for NDPS to find a program that approximated the neural policy under this assumption.

Another issue encountered with PIRL in this problem domain was the stopping condition for NDPS. When the current candidate program receives less reward than the previous candidate program, synthesis is stopped and the previous candidate program is returned as the final synthesized program. This means that if the robot manages to find and push the box in one episode, but fails to find the box in the next, synthesis stops. By looking at Figure 3, we can see that during training of the DRL agent the agent manages to find the box in early episodes, but misses it most of the time. As the agent learned more about the task, it became better at finding and pushing the box, but there were still some starting conditions it failed under. Since the synthesized candidate programs attempted to copy the behavior of the DRL agent, there were also starting conditions under which they failed. If one of these situations was encountered during synthesis, it could cause the synthesis to end prematurely.

There exists a simple fix that could be applied to to the NDPS stopping condition issue. Instead of only evaluating the candidate programs on only one episode, taking the mean of the reward across several episodes would give a better representation of the overall performance. The trade-off to this solution is that it would take

longer to run multiple episodes. Compared to the overall time taken for program synthesis, this would most likely be negligible.

Performance profiling of the synthesis code revealed that the enumerative program search step was reasonably fast. The process of finding optimal values for constants took the most time. It took approximately 6 hours to synthesize a final sketch, most of which was spent optimizing constants. Without serious modification to the NDPS algorithm, or a vastly more efficient optimizer, it would be difficult to gain much of an increase in speed during this step.

There are several other improvements/extensions to this research that there was not enough time to implement. One obvious improvement that could be made to the program search is to keep track of the size of arrays after slice operations, and disallow subsequent operations on these arrays that would cause an out of bounds error. This would help reduce the search space. The current solution is to catch out of bound exceptions thrown when the programs are evaluated and then exclude them from being chosen as the next candidate program.

Another extension would be to add conditional statements to the grammar of the DSL. In [3], DSLs with and without conditional statements were used to synthesize programs for the racing simulator task. The program that included conditional statements performed significantly better than the one without conditional statements. It would be very interesting to see if the addition of conditional statements would increase the performance of synthesized programs for the robotics task as well.

5 CONCLUSION

This research aimed to evaluate the effectiveness of Programmatically Interpretable Reinforcement Learning on a robotics task involving perceptual inputs. This was accomplished by reproducing a deep reinforcement learning robotics experiment that involved a two-wheeled robot learning to locate and push a box based on camera and distance sensor readings. This DRL agent was then used as a neural policy oracle for Neurally Directed Program Search. The final program synthesized using this method performed as well as the DRL agent on the box pushing task. This shows that PIRL is capable of synthesizing programs that operate on perceptual inputs in a real-world robotics task.

REFERENCES

[1] Tuomas Haarnoja, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine. 2018. Learning to walk via deep reinforcement learning. *CoRR*, abs/1812.11103. arXiv: 1812.11103. http://arxiv.org/abs/1812.11103.

[2] K. Shibata and M. Iida. 2003. Acquisition of box pushing by direct-vision-based reinforcement learning. In *SICE 2003 Annual Conference (IEEE Cat. No.03TH8734)*. Vol. 3, 2322–2327 Vol.3.

[3] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. 2018. Programmatically interpretable reinforcement learning. *CoRR*, abs/1804.02477. arXiv: 1804.02477. http://arxiv.org/abs/1804.02477.

[4] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter I. Corke. 2015. Towards vision-based deep reinforcement learning for robotic motion control. *CoRR*, abs/1511.03791. arXiv: 1511.03791. http://arxiv.org/abs/1511.03791.