Explainable Deep Adaptative Programs

Magesh Kumar Murali
School of Electrical and Computer Science
Oregon State University
Corvallis, Oregon 97331
Email: muralim@oregonstate.edu

Martin Erwig
School of Electrical and Computer Science
Oregon State University
Corvallis, Oregon 97331
Email: erwig@oregonstate.edu

Alan Fern
School of Electrical and Computer Science
Oregon State University
Corvallis, Oregon 97331
Email: Alan.Fern@oregonstate.edu

***Abstract*—TODO**

***Index Terms*—Adaptive Programs, Programming Langauges, Reinforcement Learning**

I. Introduction

Programming is often difficult and tedious process, it takes years of training and experience for a person to become good at creating software that can solve complex real world problems. One of the reason for this is that as the problem domain becomes more complex a lot of analysis and critical thinking is needed to capture all the scenarios that the program will encounter. The programmer has to not only think about the different scenarios that can occur but also has to make decisions based on the different scenarios. Complex problems will also contain multiple decision points and decision made at one location might have an adverse effect on the rest of the decision making process. This makes writing deterministic programs that captures all such scenarios and decisions difficult for the programmer. This is one of the reason why it is difficult to program an AI agent for games like Chess or Go that could beat human player.

There are also special type of problems where you would not know beforehand what kind of scenario that the program can encounter or the environment itself gradually changes overtime. Consider a simple problem of sorting a list of integers, the type of sorting algorithm to choose will depend on the size of the input array. Insertion sort will perform better when compared to Quicksort for smaller input size. If the inital assumption about the input size changes then the programmer has to completely rewrite the program to use a better sorting algorithm or he has to have the foresight that the input size will vary drastically.

This is one of the issues that is often faced in software development process, the assumptions being made about the environment during the initial development of the software will change over time while the overall objective that the program solves remains the same and this leads to extending the project to make it adapt to the new assumptions about the environment and making it more expensive to develop software. It has even led to the development of new software practices like Agile Process that takes into account the change in environment as part of the development process.

The issues mentioned above has led to the development of new programming paradigm called Adaptation Based Programming (ABP). ABP consists of simple level of abstraction called Adaptive Variable (AV). Adaptive Variables are same as regular variables in that the value it holds can vary at runtime, however in ABP it has three additional constrains:

- The values that it can hold, also called choices, are restricted by the programmer
- The adaptive variable is tightly dependent on the state provided by the programmer
- The objective of the adaptive variable is described using reward signals

Whenever a programmer encounters a point in the program that requires complex decision to be made, he/she creates an adaptive variable with a list of possible choices that the adaptive variable can hold. The choices are complemented by a set of reward signals, the reward signals serves as feedback to the adaptive variable to indicate whether the value it holds will achieve the objective that the programmer wants it to achieve. By defining the objective of the program using adaptive variables and reward signal, the programmer does not need make the complex decision instead the adaptive variable uses powerful reinforcement learning algorithms to make the decision.

The programmer also provides a state object as input to the adaptive variable, the choice of the value that the adaptive variable holds will also depend on this state object. The state object will make sure that the adaptive variable adaptes to the current state of the program. Whenever the adaptive variable encounters a new state that it hasnt encountered before it adapts itself to make sure it reaches the objective even for the new state. This way the adaptive variable not only makes complex decisions, it also adapts to changes in the environment or even new environments that it hasnt encountered before.

One of the issues with using state of the art reinforcement learning algorithm to make the choices is that the reason for making the decision is often not clear to the user. This makes it hard to understand the output of the program and even debugging becomes tedious. We try to overcome this issue by providing explanations to the output generated using annotation at the decision and reward points.

This paper provides three improvements to adaptation based programming paradigm.

- Deep Adaption Based Programming: Extending the adaptive variable to use state of the art deep reinforcement learning.
- Explanations: using annotations
- Using General Value Functions (GVFs) to improve the generated explanations

II. Explanation in ABP

A. Motivating Example

To illustrate explanation in XDAP let us consider the following simple example problem. Figure 1 shows a simple 5×2 gridworld. The agent in this example is a traveller whoes goal is to reach the destination (Home) within 8 days and collect as much reward as possible. In order to reach home the traveller has to cross different types of terrains. It would take 4 days to cross a mountain, 2 days to cross a small hill and 2 days to cross a river. The gridworld contains two types of treasures: Diamond and Gold. Diamond being slightly more valuable that Gold. The traveller has four actions (North, South, East, West) to choose from at every step.

1. Should explain high negative reward for not reaching the house?
2. Type of reinforcement learning problem?

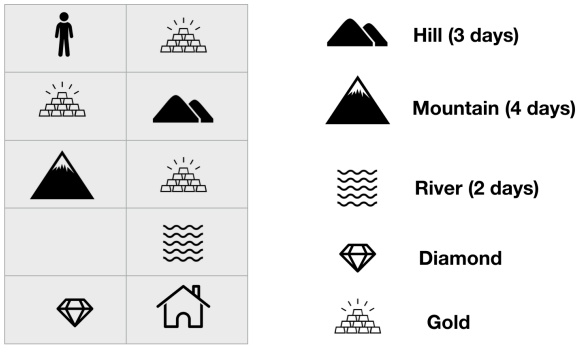


Figure 1. Traveller Problem

B. Adaptive Program

Should the reward be from the environment? or add if statements showing how the reward is assigned?

```
traveller = Adaptive(choices = [NORTH, SOUTH, EAST, WEST])

state = env.reset()

for days in range(8):
  direction_to_move = traveller.predict(state)

  state, reward, done, info = env.step(direction_to_move)

  traveller.reward(reward)

  if done:
     break
```

Listing 1. Adaptive Program for traveller problem

Is it necessary to explain the program? Or is it self explanatory?

When the ABP program is run, the traveller decides to take the path as shown in Figure 2

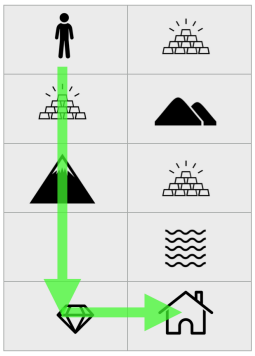


Figure 2. Traveller Decision

However it is not clear why the decision was made to take the current path. There could be several reasons for making the decision and many unanswered questions that we can ask: "Why did the traveller not move EAST and collect the gold on the first step?"

This can be considered as one of the disadvantages of using adaptive variables in the program. Even though we assume the value that the adaptive variable holds is optimal, it is not clear as to why it made the decision to hold the current value. Also making it difficult for the programmer to debug programs containing these adaptive variables.

We explain the decisions being made by the adpative variable by extending the adpative programming library to use reward decomposition and using predictors

Need a better way to say this! Should it be called predictors?

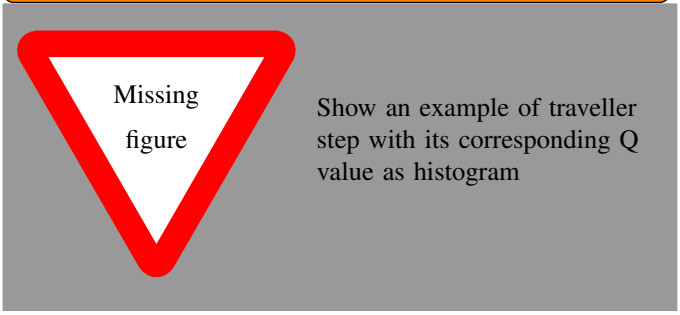
C. Explanation using ABP

1. So far we havent talked about how the adaptive variable choose its value. Is it needed? Or should it be a reference to another paper?
2. Need a better title
The goal of this sections is to make the reader understand that Q value function is a knowledge representation and has semantics

The adaptive variable uses reinforcement learning techniques to choose the correct value [1]. After the initial training period, the adpative variable generates a Q function [1]. The value that the adaptive variable holds is then based on the Q function. The Q function also called value function has an explictic semantics [2]. In the case of the traveller example it represents the knowledge of which action to take to maximize

the reward or to be more explicit which action should the traveller take to reach the destination in 8 days and collect as much treasure as possible.

Need to rewrite this in a better way! Important concept for the paper. Elaborate more!



From the Figure it is clear that the traveller has to move south in order to maximize the reward.

However having a single Q value does not give us a sufficient explanation. It compresses the entire knowledege of the problem into a single value. One way of getting better explanations is to decompose the Q values into smaller Q values each representing knowledge of a particular subdomain of the problem. We propose a way of decomposing the Q value based on the reward signals in order to generate a better explanation.

D. Types for reward

1. Need a better title! 2. Is it reward signals or reward? Prefer reward but previous paper regard it as reward signals Goal of this section is explain what are reward types!

In the traveller example described above, the reward signal could be of different types. For example, the reward for reaching home is clearly different from the reward for finding a treasure. These reward signal have different semantics associated with them. The traveller problem can then be decomposed into sub problems based on these reward types. For example we can decompose the reward into 3 types for the traveller problem:

- Type 1: Reward for reaching home
- Type 2: Reward for finding treasure
- Type 3: Reward related to effort of crossing terrain

Type 1 would describes the subproblem of the traveller reaching home. Type 2 would define the subproblem of traveller finding treasure. and Type 3 is the subproblem of the traveller avoiding terrains as much of problem.

E. Adaptive Program using reward types

The adaptive program in Listing 1 can be refactored to use reward types.

```
traveller = Adaptive(choices = [NORTH, SOUTH, EAST, WEST])

state = env.reset()

for days in range(8):
  direction_to_move = traveller.predict(state)

  state, reward, done, info = env.step(direction_to_move)

  traveller.reward(reward["Home"], HOME_TYPE)

  traveller.reward(reward["Treasure"], TREASURE_TYPE)

  traveller.reward(reward["Terrain"], TERRAIN_TYPE)

  if done:
     break
```

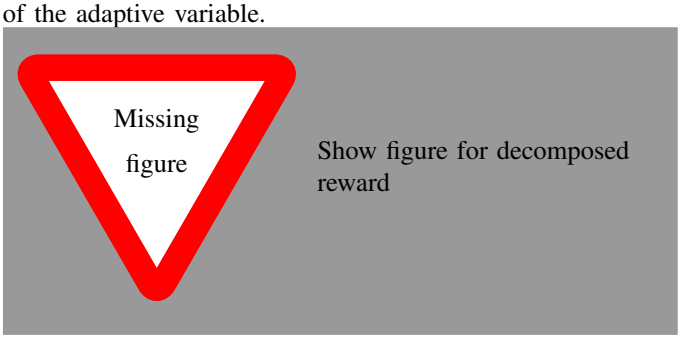
Listing 2. Adaptive Program using reward decomposition for traveller problem

With the refactored reward types and using Q-decomposition [3] to train the adaptive variable we have three types of Q values each containing the domain knowledge of its own subproblem.

Should explain how reinforcement learning works for decomposed rewards?

F. Explanation using decomposed reward

With the adaptive variable having information about the three subdomains and making its decision based on the decomposed Q values, we can get a better explanation than before. The programmer has more information regarding the decision of the adaptive variable.



G. Example explanation

Give an example explaining the decision of the traveller at a particular state

H. Scope of explanation

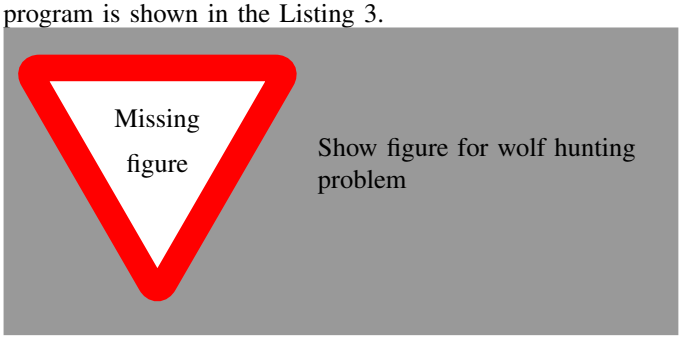
1.What can we explain using this method?
2.What are the drawbacks? w.r.t RL (discount factor), w.r.t Explanation

III. Performance using reward types

IV. Predictors

One of the issues with the approach mentioned above is that the environment may not have a single reward type. This makes it difficult to use the approach. In order to overcome the difficulty we introduce a new kind of abstraction called predictors. To illustrate the use of predictors consider a simple girdworld problem consisting of two wolves and a rabbit. The objective is for the wolves to capture the rabbit as fast a possible. The optimal way of doing this would be for the

wolves to coordinate with each other. In this case there is only one reward type which gives the value 1 when the wolves capture the rabbit and 0 otherwise. The corresponding ABP program is shown in the Listing 3.




```
wolf1 = Adaptive(choices = [UP, DOWN, LEFT, RIGHT])
wolf2 = Adaptive(choices = [UP, DOWN, LEFT, RIGHT])

state = env.reset()

for steps in range(env.max_episode_steps):
  wolf1_action = wolf1.predict(state)
  wolf2_action = wolf2.predict(state)

  action = (wolf1_action, wolf2_action)

  state, reward, done, info = env.step(action)

  wolf1.reward(reward)
  wolf2.reward(reward)

  if done:
     wolf1.end_episode(state)
     wolf2.end_episode(state)
     break
```

Listing 3. Adaptive Program for wolf hunt problem

A. ABP Program

V. Conclusion

TODO

References

[1] T. Bauer, M. Erwig, A. Fern, and J. Pinto, "Adaptation-based programming in java," in *Proceedings of the 20th ACM SIGPLAN workshop on Partial evaluation and program manipulation*. ACM, 2011, pp. 81–90.

[2] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup, "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction," in *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 761–768.

[3] S. J. Russell and A. Zimdars, "Q-decomposition for reinforcement learning agents," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 656–663.