CS 267A Final Report

Siddharth Joshi, Atefeh Sohrabizadeh, Rahul Salvi

June 18, 2020

1 Abstract

For this project, we are interested in researching applications of reinforcement learning and probabilistic programming for the area of robotics. To this end, we develop a 2D grid-based game with multiple active agents to explore possible uses of probabilistic programming languages to optimize path planning. Our goal for this project is to demonstrate the effectiveness of probabilistic programming in the *reinforcement learning as inference* paradigm. We chose to model the pursuit evasion problem and particularly focus on using probabilistic inference in determining the actions of the pursuer by conditioning on their latent optimality. We explore the problem in Dice and Pyro, two popular probabilistic programming languages, to understand the strengths and weaknesses of the contrasting techniques of exact and approximate inference. To query different moves over time in Dice, we implement a Python abstraction layer. Using the results of our experiments, we hope to gain insights that could ultimately prove useful for real-world robotics problems.

2 Motivation and Introduction

We were driven by a shared interest in researching how reinforcement learning and probabilistic programming can be applied to the area of robotics. Research in this field often relies on simple games that abstract away considerable complexity to focus on a particular skill. The pursuit evasion we chose to explore consists of two actors who will have competing objectives: the pursuer attempts to catch the evader and the evader attempts to avoid this for as long as possible. To simplify our system, the agents move on a 2D grid.

For the choice of probabilistic programming language, we used both Dice and Pyro. These two languages differ in their intended usage. Dice is a language focused on exact inference for discrete systems, while Pyro is a universal probabilistic programming language where we would use its approximate inference to our benefit. By using both languages in parallel, alongside our goal of demonstrating the effectiveness of probabilistic programming in this paradigm, we also aimed to evaluate the merits of the two contrasting languages.

3 Background

The relevant background for this project beyond the materials covered in the class was the translation of the Reinforcement Learning Problem from a control problem to inference. Levine et al. [1] discusses one approach for solving this problem. The goal is to translate the control problem which can intuitively be phrased as *what policy generates trajectories that maximize reward*. The approach covered by Levine in [1] attempts to formalize the natural translation of this control problem to inference i.e. to generate an inference problem that answers the question *how likely is a certain trajectory*

given that the trajectory taken maximizes reward which can then be used to extract a non-deterministic policy by answering questions of the form *how likely is a certain action given that taking this action maximizes reward from now on-wards*.

This very natural idea is then formalized as a probabilistic graphical model. Levine introduces a preliminary graphical model that captures the relations between states and actions across time steps i.e. next state is determined by current state and current action and this translates the *transition dynamics* of the RL problem.

The challenge lies, now, in incorporating the idea of *maximizing reward* in such a setting. To do this, Levine suggests introduction of indicator variables at every time step that indicate optimality as shown in Fig. 1. These Bernoulli random variables capture the reward function by having as their parameters the exponential function applied to the reward of the action taken in the current state.

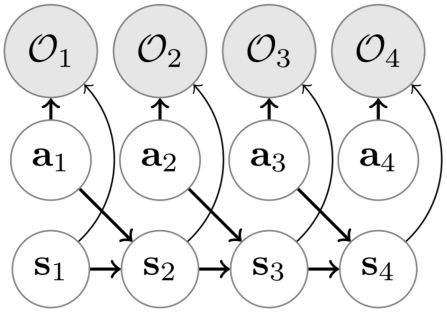

Figure 1: Graphical model with optimality variables as in [1]

Now, we can successfully translate the notions of the *transition dynamics* and *reward maximization* to the probabilistic inference setting. The inference mechanism now simply needs to compute $P(A_t|O_t, O_{t+1}, ..., O_{t+n})$ for some finite n, where the probability of choosing an action is conditioned on the probability of optimality at this time step and optimality in the future (a limitation of this approach is the fact that we consider only finite trajectory lengths, ideally we would like to condition infinitely, but this would make inference rather challenging). This is the model that our project employs to solve the 2D grid Pursuit Evasion problem. [1]

4 Technical Contribution

The project's technical contribution can be divided into three main sections that cover the tasks that we had to complete to achieve our objective of demonstrating the utility of the reinforcement learning as inference paradigm and the effectiveness of probabilistic programming languages in this paradigm as a sophisticated inference tool.

The primary tasks of the project were:

- Creating a customizable GridWorld Based Environment from which the reward function and

[1]There are several finer points in the derivation that have been omitted for simplicity and brevity, but had to be considered in the implementation. Such as, the need for non-positive rewards, the exponential function to output a valid probability, the equivalence of this model to maximum entropy reinforcement learning and the potential for risk-seeking behavior arising in approximate inference techniques that would end up implicitly modifying the transition function which is unrealistic.

transition function could be extracted to give the inference tools an accurate model to reason with.

- Modeling the probabilistic graphical model of the aforementioned form for the particular environment in Dice [2] and using exact inference to guide the pursuing agent's actions.
- Modeling the probabilistic graphical model of the aforementioned form for the particular environment in Pyro [3] and using stochastic variational inference to guide the pursuing agent's actions.

The details of these tasks along with the *Python to Dice Compiler* that was a rather useful byproduct of our work is covered in the following subsections.[2]

4.1 A GridWorld-based Environment

The goal was to create a customizable environment that was neatly abstracted to allow us to plug in pursuing agents and have them interact with the environment and attempt to catch the evading agent. Using the *PyColab Game Engine by Deepmind* [4] as a base, we were able to construct a pursuit evasion environment. The environments are wrapped as OpenAI gym environments to allow for wide applicability to reinforcement learning methods. The environment can easily be customized to different layouts of the grid as well as different reward functions for the pursuer. Currently, we have two particular environments: *one-random-evader* and *one-stationary-evader*. Fig. 2 depicts an example of the generated environment.

Figure 2: Setup for the Baseline 2D grid

4.2 Inference using Dice

We have written a basic Inference Engine in Python that uses the reinforcement learning problem parameters (mentioned in Background) to dynamically generate Dice programs that represent the desired distribution, execute them, parse the results, and return them as a list representing the probability distribution (since the variable is discrete).

This objective required translating the python reward and transition functions to Dice which motivated us to simplify the reward function to take in only the next state as its parameter rather than current state and action. The transition function given the current state and action, returned a probability distribution over all the states, from which the next state was sampled. The probability inferred then was the distribution over actions at time 0 given optimality for next 10 steps.
While this method was highly effective, as was confirmed by the the agent following the optimal path in the stationary evader environment, the exponential blowup in the state space was obvious in the random evader environment. It resulted in a practical challenge for modeling any significantly challenging environment using Dice.

[2]The source codes of this project is accessible from `https://github.com/sjoshi804/CS-267A-Final-Project`

```
let STATE_0 = int(306, 169) in
let ACTION_0 =
   discrete(0.2500000,  0.2500000,  0.2500000,  0.25000000) in
let STATE_1 =
   transition(STATE_0,  ACTION_0) in
let OPTIMAL_0 =
   reward(STATE_1) in
let _ =
   observe(OPTIMAL_0) in
let ACTION_1 =
   discrete(0.2500000,  0.2500000,  0.2500000,  0.25000000) in
let STATE_2 =
   transition(STATE_1,  ACTION_1) in
let OPTIMAL_1 =
   reward(STATE_2) in
let _ =
   observe(OPTIMAL_1) in
ACTION_0
```

Code 1: Shortened Sample Agent Model in Dice

```
def agent_model(state=(0, 0)):
    p = torch.ones(4)/4
    action = pyro.sample("action", dist.Categorical(p)) # up, down, left, right
    prob_1 = convert_to_prob(transition(state, action))
    state_1 = unhash_state(pyro.sample("state_1", dist.Categorical(torch.tensor(prob_1))))
    pyro.sample("optimal", dist.Bernoulli(torch.tensor([exp(reward_function(state_1))])),
    obs=torch.tensor([1.]))
```

Code 2: Agent model in Pyro

Moreover, another limitation we observed after the experiments with Dice was that the model needed to 'look ahead' i.e. consider lengths of trajectories that could potentially catch the evader as otherwise the sparsity of the reward signal (available only when the goal state is achieved) prevents the pursuing agent from inferring any useful information.

4.3 Inference using Pyro

In addition to Dice, we implement our agent in Pyro [3]. Pyro is a deep universal probabilistic programming language developed by Uber. In Pyro, models, including all control flows are functions with arbitrary Python code. This feature enables expressivity. Furthermore, it makes use of PyTorch with its approximate inference, high-performance automatic differentiation, and tensor math, which increases scalability.

We can define the parameters using `pyro.param`, latent random variables using `pyro.sample`, and observations using `pyro.sample` with the `obs` argument. Code 2 shows how we implement the network in Fig. 1 for a single step inference using Pyro. Line 2-3 show our prior belief on agent's action, which is a uniform distribution on the 4 directions that the agent may take. Line 4-5 build a probability over the states that the agent can go to and convert the sampled state to a state in our 2D grid. Line 6 puts an observation on the optimality of the action taken.

We use stochastic variational inference (SVI) [5], a scalable algorithm for approximating posterior distributions to make the model learn which action it should take. Fortunately, SVI is supported by Pyro and we do not need to implement it ourselves. The idea is to introduce a parameterized

```
1 def agent_guide(state):
2     p_guide = pyro.param("p_guide", torch.ones(4)/4, constraint=constraints.simplex)
3     action = pyro.sample("action", dist.Categorical(p_guide))
4     prob_1 = convert_to_prob(transition(state, action))
5     state_1 = unhash_state(pyro.sample("state_1", dist.Categorical(torch.tensor(prob_1))))
```

Code 3: Agent guide in Pyro

```
1 p_guide = tensor([2.3549e-04, 9.9935e-01, 8.3949e-05, 3.2638e-04], grad_fn=<DivBackward0>)
```

Code 4: Posterior distribution of the action

distribution called the variational distribution which Pyro names it the *guide* function. The guide will serve as an approximation to the posterior and it should have the same call signature as the model function in Pyro. Code 3 demonstrates how we implement the guide function for our agent. The differences are that 1) we should not have observation on any variables and 2) the probability distribution of the action should be parameterized. The `simplex` constraint in Line 2 enforces the sum of the probabilities of each of the actions to be 1, which is required for having a valid probability distribution.

The next step to creating an `SVI` instance in Pyro is defining an objective function. Currently, Pyro only supports the evidence lower bound (ELBO), which defines an expectation w.r.t. samples from the guide as explained in [6]. The ELBO is a lower bound to the log evidence. In oder to maximize the log evidence, Pyro takes (stochastic) gradient steps to maximize the ELBO. As a result, the log evidence will be pushed higher.

Fig. 3 illustrates the ELBO loss (minus the ELBO) over 3000 gradient steps. Code 4 shows the resultant posterior distribution of the agent's action when the evader is one step apart. It tells us that the agent should go `down` to get the evader. These results suggest that our model is able to learn the best action that our agent should take. This code can be extended to support longer trajectories.[3] Our experimental results indicate that the Pyro model is able to solve our baseline problem.

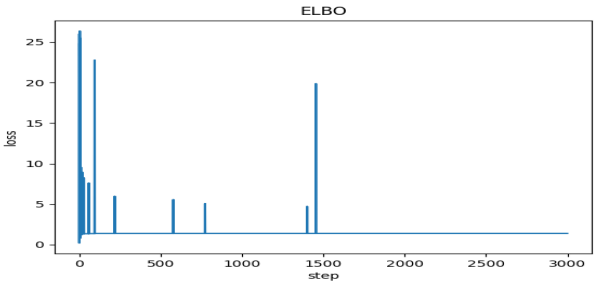

Figure 3: Loss of the model used in Pyro over gradient steps.

[3]The complete inference code using Pyro can be found from `https://github.com/sjoshi804/CS-267A-Final-Project/blob/master/agent_pyro.py`.

4.4 Python to Dice Compiler

Dice is efficient at computing probabilities for discrete variables, but the language does not have built-in support for high-level constructs such as loops. Given the turn-based nature of the game, we wanted to be able to write Dice programs that could compute probabilities for some variable, then recursively use these probabilities for further computation. To accommodate this, we created a system for generating Dice programs from a Python script, aptly called *py2dice*. The simple grammar of the Dice language lends itself well to being automatically created. In the Python script, we can create a representation of an abstract syntax tree. At this point, we can recursively walk the tree, emitting tokens along the way. The end result is a program that can be parsed by the Dice interpreter. By changing the values of the Python variables used to generate the program at each step, we can implement the looping behavior initially desired.

5 Related Work

Probabilistic planning is a popular problem in literature [7, 8, 9, 10]. Thrun et al. [9] explain how probabilistic planning can help the field of robotics. The inherent uncertainty of a robot's environment has led to the popularity of this problem. This uncertainty can be due to lots of reasons such as the limitations of the robot to perceive the environment or the noises of its sensors. The proposed algorithms to solve this problem can be grouped in two main categories. The first one uses a greedy approach for exploring the environment [11, 12]. Greedy algorithms maximize the reward function for the immediate next time steps or the few steps in the future. On the other hand, the second category employs non-greedy methods [13], which takes the entire sequence of control into account. Since non-greedy methods are challenging in terms of their computation complexity, greedy algorithms are more widely adopted. Thus, the problem is mostly seen as maximizing the conditional expectation of the reward function over the state space.

6 Conclusion

The experiments showed us that both languages were effective in guiding the behavior of the pursuing agent - achieving essentially perfect policies. Furthermore, exact and approximate inference were able to achieve nearly identical performance in terms of the optimality of policies inferred, however Dice's exact inference was significantly slower than the stochastic variational inference used in Pyro.

Our initial baseline objective was to have a functioning 2D Grid-World Environment in which our pursuing agent would use probabilistic programming to infer an effective policy for catching the evader - and we were able to achieve this with both Dice and Pyro. However, the inefficiency caused by the large state space and the need for our implementation strategies to enumerate the state space resulted in inability to solve more complex environments with less mundane structure.

In conclusion, the project was a valuable learning experience which taught us not only about a variety of advanced concepts in relational learning such as stochastic variational inference, but also about the challenges of experimental work in artificial intelligence from our struggles in setting up a reinforcement learning environment that was customizable and achieved our objectives.

7 Feedback

Given the inherent challenges of remote collaboration, the project went reasonably well. Deciding on a team and topic early was helpful since it meant that we did not have to scramble at the end of the quarter with initial setup.

References

[1] S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," in *ArXiv*, 2018.

[2] S. Holtzen, G. V. den Broeck, and T. Millstein, "Dice: Compiling discrete probabilistic programs for scalable inference," 2020.

[3] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep Universal Probabilistic Programming," *Journal of Machine Learning Research*, 2018.

[4] DeepMind, "Pycolab gridworld game engine," 2017. [Online]. Available: https://deepmind.com/research/open-source/pycolab

[5] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[6] Uber Technologies, "SVI part I: An introduction to stochastic variational inference in Pyro," 2018. [Online]. Available: https://pyro.ai/examples/svi_part_i.html

[7] S. W. Yoon, A. Fern, and R. Givan, "Ff-replan: A baseline for probabilistic planning." in *ICAPS*, vol. 7, 2007, pp. 352–359.

[8] N. Kushmerick, S. Hanks, and D. S. Weld, "An algorithm for probabilistic planning," *Artificial Intelligence*, vol. 76, no. 1-2, pp. 239–286, 1995.

[9] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[11] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, "Coordination for multi-robot exploration and mapping," 2000.

[12] J. Banfi, A. Q. Li, I. Rekleitis, F. Amigoni, and N. Basilico, "Strategies for coordinated multi-robot exploration with recurrent connectivity constraints," *Autonomous Robots*, vol. 42, no. 4, pp. 875–894, 2018.

[13] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations research*, vol. 21, no. 5, pp. 1071–1088, 1973.