THE UNIVERSITY OF EDINBURGH

INFORMATICS HONORS PROJECT

Generative Adversarial Networks Generating Novel Reinforcement Learning Policies

Author:
Giovanni ALCANTARA

Supervisor:
Dr. Timothy HOSPEDALES

A thesis submitted in fulfillment of the requirements for the degree of Bachelor's of Engineering

in the

School of Informatics
College of Science & Engineering

January 25, 2018

Contents

Chapter 1

Introduction

1.1 Motivation

1.2 Structure of the report

1.3 Main contributions

Chapter 2

Background

2.1 Reinforcement Learning

2.1.1 Markov Decision Processes

Environments in traditional reinforcement learning application are usually modelled as Markov Decision Processes or MDPs.

These can be formulated as systems with the following components:

- Finite set of states $S = \{s_0, \ldots, s_n\}$ and actions $A = \{a_0, \ldots, a_m\}$.
- A distribution of probabilities $P_a(s, s')$ for transitions from state s to s' for each possible action a.
- A reward function $R : S \mapsto \mathbb{R}$ for being at a particular state.
- The goal in reinforcement learning is to maximise the final reward that an agent achieves in the environment.

As the name suggests, MDPs obey the *Markov property*, whereby the probability of the system being in a certain future state exclusively depends upon the present state, and not upon an arbitrarily-long sequence of past states.

In figure 2.1 we show a sample schematic of a Markov Decision Process, and how states, actions, and rewards could be connected between each other.

2.1.2 Q-learning

Now that we contextualised reinforcement learning environments as Markov Decision Processes, we can introduce the final objective of reinforcement learning tasks: finding a function $\Pi : S \mapsto A$ called **policy**, that maps the appropriate action $a \in A$ given the current state $s \in S$, as to maximise our agent's final reward.

In particular, we will now present Q-learning, an algorithm that yields such optimal policy given an MDP.

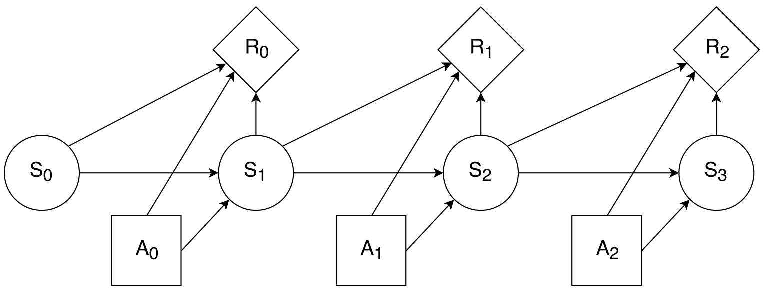

FIGURE 2.1: Sample schematic of an MDP.

Q-learning lets us learn the *quality*, or expected utility, for each state-action combination. That is, for each state, let's estimate all the expected rewards we obtain by taking each possible action at that particular state.

More formally, we estimate a function $Q : S \times A \rightarrow \mathbb{R}$. We can model Q as a mapping table (initialised with some uniform values), whose value we update at each time step of our simulations.

Here's how we update our Q-table at each time step t:

$$Q(s_t, a_t) = \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \times \left[\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right]$$

where:

- $\alpha \in [0, 1]$ is the *learning rate*, a coefficient that regulates how much the newly learned values will contribute in the update
- $\gamma \in [0, 1]$ is the *discount factor*, a coefficient that controls the weight of future rewards. Values closer to 0 will make our agent "short-sighted", considering only the immediate rewards.

What is our optimal policy when we do Q-learning then? After training, it is simply that function $\pi : S \rightarrow A$ that, for each state, returns the action with maximum expected utility in our Q-table.

2.1.3 Exploration/Exploitation

2.1.4 Deep Reinforcement Learning

DQGAN

2.1.5 Problems with reinforcement learning techniques

2.1.6 Actor critic

2.2 Generative Adversarial Networks

2.2.1 Architecture of GANs

2.2.2 Successes

DCGAN

2.2.3 Conditional GANs

2.3 Existing related work

2.3.1 Generative Adversarial Imitation Learning

Chapter 3

Environment

In this chapter we report the process that went into choosing the environments that we will be using in the project's simulations, and from which we will be basing our simulations.

There are many choices of environments and tasks that are publicly available. Some of these we will be introduce in this chapter.

What makes this step non-trivial and deserving of its own chapter is that different reinforcement learning techniques are more suitable to different categories of tasks. Similarly, different machine learning and deep learning techniques are more or less efficient when applied to different tasks.

In a research effort that is heavily dependent on building reinforcement learning and deep learning models, the choice of environment is a critical one.

Furthermore, we also need to achieve this without losing focus on the main motivations for the whole project (Chapter 1): *optimising reinforcement learning algorithms by adding transferability of pre-trained models on unseen maps or configurations of a task*.

This last point implies that a substantial part of the computational work in the project will be about training hundreds of thousands of reinforcement learning models to build a dataset over a distribution of different maps (we present this in Chapter 4). This is an important point: in the many experiments we ran, this turned out to be the biggest bottleneck, and required devices to distribute computations across multiple machines to make the computational time feasible in the timespan allocated to the project.

With these points in mind and given the experimental nature of the work, we conclude that a bottom-up approach in complexity is preferable. Given successful results with "easier" tasks, we can scale up in complexity and hopefully formalise and generalise our approach to more tasks (Chapter 7).

Easier tasks will enable us to explore different reinforcement learning approaches that we introduced in Subsection 2.1 with the guarantee that they will give satisfiable results. We can use these results as a foundation of the further steps (specifically Generative Adversarial Networks training in Chapter 5).

Before introducing candidate environments, let us define what is meant by an "easy" reinforcement learning task. What we are looking for is ideally a task with a discrete and relatively small set of observable states and actions. Why does this conditional make the task easier?

Imagine building a tree (such as the one shown in figure 3.1 with all possible states-action transitions, until we either: 1) reach a goal state, or 2) reach an arbitrarily maximum iteration time step $t = \eta$ (to prevent infinite iterations). Also assume we were traversing this tree in a bruteforce manner (worst-case scenario of reinforcement learning resolutions), then we would need to visit each node of the tree, until we arrive at the leaves, which will report the achieved reward for the agent given the path it has taken to get there. The breath and depth of the state-action tree will

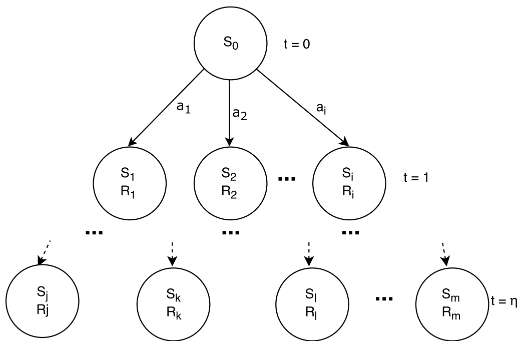

FIGURE 3.1: Sample state-action tree

increase as we increase the possible set of states we would need to traverse, adding up to the space and time complexity of our solution, which is exponential in $\#S$ and $\#A$ for both space and time ($\#S$ indicates the cardinality of a set S).

3.1 OpenAI Gym

3.1.1 Motivation

3.1.2 Algorithmic environments

3.1.3 MuJoCo and physics environments

3.1.4 Other environments

3.2 Baseline: `FrozenLake-v0`

3.2.1 Description of the task

3.2.2 Motivation

3.2.3 Shortcomings

3.3 Extended baseline: Randomised Frozen Lake

Chapter 4

Dataset creation

4.1 MapReduce job

Chapter 5

Adversarial Networks Training

Chapter 6

Generative Adversarial Q-learning

6.1 Using the trained Generator

6.1.1 Initialisation

6.1.2 Exploration

6.2 Using the trained Discriminator

6.2.1 Speeding up Q-learning on unseen maps

Chapter 7

Scaling up to more complex tasks

7.1 Larger Frozen Lakes

7.2 Physics environments

Chapter 8

Conclusion