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 fulfilment of the requirements for the degree of Bachelor's of Engineering

in the

School of Informatics
College of Science & Engineering

January 25, 2018

THE UNIVERSITY OF EDINBURGH

Abstract

Faculty Name
College of Science & Engineering

Bachelor's of Engineering

Generative Adversarial Networks Generating Novel Reinforcement Learning Policies

by Giovanni ALCANTARA

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Contents

Chapter 1

Introduction

1.1 Motivation

1.2 Structure of the report

1.3 Data pipeline

1.4 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.

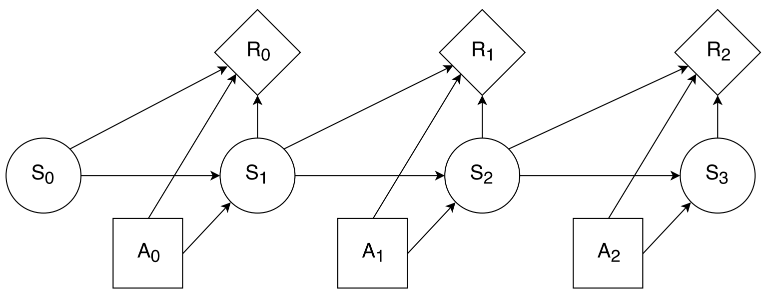

FIGURE 2.1: Sample schematic of an MDP.

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 tradeoff

An important theme in reinforcement learning, and that we heavily focus our attention on in our project is the idea of the tradeoff between Exploration and Exploidation.

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 introducing 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 condition 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 $t = \eta$ or depth of the tree (to prevent infinite iterations). Also assume we were building this tree in a bruteforce manner (worst-case scenario of reinforcement learning resolutions), such that we need to build all possible paths or trajectory that the agent will need to take. Therefore, we would

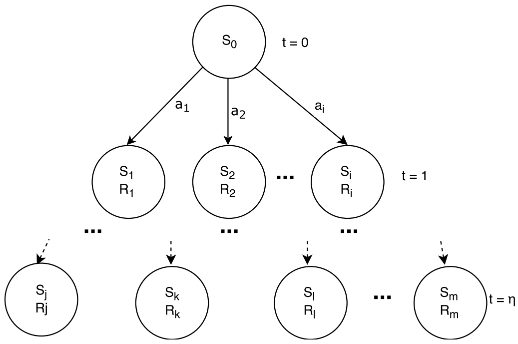

FIGURE 3.1: Sample state-action tree

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 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).

With that said, there is a solid line between our ideal environment and a trivial one that could be solved without the aid of reinforcement learning techniques.

Let's explore some candidate environments next.

3.1 OpenAI Gym

OpenAI Gym (Brockman et al., 2016) provides a toolkit that aids in the process of building reinforcement learning systems and evaluating algorithms to solve such tasks.

OpenAI Gym provides an environment interface `Env`. The interface abstracts the following operations on an environment:

- `step(action)` – Simulates one time step by executing the provided action. This returns `observation` (the new current state), the `reward` for taking such action, a flag `done` indicating whether the system has reached a final state, and `info` providing additional information dependent on the environment.
- `reset()` – Resets the environment, i.e. the initial state is restored
- `render()` – Renders the environment in human-readable format.

Now, we could either build implementations for such interface (if we were to implement our own environment's logic), or use the provided implementations for several environments in the OpenAI Gym library, which includes board games, algorithm-based or physics-based environments, Atari games, etc.

3.1.1 Motivation

In this project we will mostly deal with environments provided in the OpenAI Gym. There are several reasons for such decision:

- it abstracts the need to implement the logic of a separate environment, adding a point of failure to our whole (quite experimental) work, as well as imposing a bigger time constraint to the one we already have;

- OpenAI Gym has become a standard academic tool for reinforcement learning researchers, therefore many papers and articles build on top of this framework;
- environment implementations are constantly expanding and being revised by an active community, also thanks to the support of the overarching organisation (OpenAI);
- the core implementation of the `gym` module (*openai/gym on Gihub*) allows for straightforward extensions on existing environments, which, as we will see in subsection 3.3, will be critical in this project;
- while sadly not applicable anymore, OpenAI Gym used to provide and support an online platform for developers to compare the performance of different reinforcement learning algorithms for each task. This was in a form of a leaderboard, measuring the performances, as well as providing the implementation and data of winning techniques. Such data could have been used directly as input to our Generative Adversarial Networks.

3.1.2 Algorithmic environments

3.1.3 MuJoCo and physics environments

3.1.4 Other environments

3.2 Baseline: `FrozenLake-v0`

One of our baseline environments is `FrozenLake-v0` (*OpenAI Gym*), one of the algorithmic environments provided in the OpenAI Gym. In this section we describe the task, motivation for choosing it as one of our baseline models, and some shortcomings that we faced in using this environment in our project's pipeline.

3.2.1 Description of the task

In `FrozenLake-v0` we control an agent in a grid world, more precisely the grid shown in figure 3.1. The objective is to go from a starting tile (`S`) to a goal state (`G`), by moving in four possible directions from a tile: up, down, left and right.

What differentiates this from a trivial path-finding or planning problem, is:

1. there are both walkable and non-walkable tiles in the grid (these are respectively frozen tiles `F` and holes `H`), and
2. tiles are "slippery" meaning that the movement direction of the agent is uncertain and only partially depends on the chosen direction, i.e. the direction is non-deterministic.

The reward in `FrozenLake-v0` is 1 for reaching the goal state, and 0 for being in any other state. The system stops executing when the agent falls into a hole, and the environment has to reset.

3.2.2 Motivation and shortcomings

`FrozenLake-v0` is generally classified as a straightforward reinforcement learning task. An optimal solution could be found by creating a model of the environment, that is merely recording where frozen tiles are while exploring the grid world. This

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	**G**

TABLE 3.1: `FrozenLake-v0`'s default 4x4 configuration

is a *model-based* approach to reinforcement learning, and it is perhaps less interesting that learning by exploration, without being biased by the particular configurations of the map.

Model-free algorithms, like Q-learning, need no accurate representation specific to the environment, and they are therefore more transferable to different configurations of the map or even to different tasks, which is the final objective of our project.

In fact, let's take the case of Q-learning applied to `FrozenLake-v0`: we do not need to have an explicit knowledge of the dynamics of each different tiles. We do not build a policy that explicitly favours movements towards frozen tiles or towards the goal. In fact, the agent does not have a model of what a frozen tile, or the goal tile or a hole are–it just learns by exploration that there it is rewarded when it gets to the goal state, and that is what it aims for implicitly.

This is the sort of behaviour that we can transfer to different tasks–again our ultimate objective.

In its current OpenAI Gym implementation `FrozenLake-v0` is a static map with the fixed configuration that is shown in figure 3.1. There is currently no way to generate random configurations of the map, so we will be extending this implementation to account for that.

Before we move on, let's contextualise this in the bigger picture as to not lose focus. Why do we need different configurations of the map, again? We want to train reinforcement learning models on different configurations so that we can have a distribution of policies over different maps which we can use as input to our GAN, thereby training a Generator network to spawn new policies for unseen configurations without having to find it through (computationally expensive) reinforcement learning algorithms!

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

Bibliography

Brockman, Greg et al. (2016). *OpenAI Gym*. eprint: arXiv:1606.01540.

OpenAI Gym. https://gym.openai.com/envs/FrozenLake-v0/. (Accessed on 01/25/2018).

openai/gym on Gihub. https://github.com/openai/gym/tree/master/gym. (Accessed on 01/25/2018).