# Integrating Deep Reinforcement Learning Networks with Health System Simulations.

A practical OpenAI Gym-like coding framework using PyTorch and SimPy.

*Michael Allen[1] and Thomas Monks[2]

[1,*] University of Exeter Medical School & NIHR South West Peninsula Applied Research Collaboration (ARC).
[2] University of Exeter Institute of Data Science and Artificial Intelligence
[*] Corresponding author: m.allen@exeter.ac.uk

July 23, 2020

## Abstract

*Background and motivation*: Combining Deep Reinforcement Learning (Deep RL) and Health Systems Simulations has significant potential, for both research into improving Deep RL performance and safety, and in operational practice. While individual toolkits exist for Deep RL and Health Systems Simulations, no framework to integrate the two has been established.

*Aim*: Provide a framework for integrating Deep RL Networks with Health System Simulations, and to ensure this framework is compatible with Deep RL agents that have been developed and tested using OpenAI Gym.

*Methods*: We developed our framework based on the OpenAI Gym framework, and demonstrate its use on a simple hospital bed capacity model. We built the Deep RL agents using PyTorch, and the Hospital Simulation using SimPy.

*Results*: We demonstrate example models using a Double Deep Q Network or a Duelling Double Deep Q Network as the Deep RL agent.

*Conclusion*: SimPy may be used to create Health System Simulations that are compatible with agents developed and tested on OpenAI Gym environments.

*GitHub repository of code*: `https://github.com/MichaelAllen1966/learninghospital`

## Keywords

Health Services Research, Health Systems, Simulation, Reinforcement Learning

## 1 Introduction

Deep Reinforcement Learning and Health System Simulations are two complementary and parallel methods that have the potential to improve the delivery of health systems.

Deep Reinforcement Learning (Deep RL) is a rapidly developing area of research, finding application in areas as diverse as game playing, robotics, natural language processing, computer vision, and systems control[1]. Deep RL involves an *agent* that interacts with an *environment* with the aim of developing a *policy* that maximises long term *return* of *rewards*. Deep RL has a framework that allows for generic problem solving that is not dependent on pre-existing domain knowledge, making these techniques applicable to a wide range of problems.

Health Systems Simulation seeks to mimic the behaviour of real systems. These may be used to optimise services such as emergency departments[2], hospital ward operation and capacity[3] and community hospital capacity[4]. These examples of health service simulations are used for off-line planning and optimization of service configuration.

Health Systems simulations are usually used for planning of service delivery changes. There is potential for these type of simulations to be used to

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 July 2020 doi:10.20944/preprints202007.0598.v1

test, develop and train Deep RL agents. The motivation for this integration includes:

- To perform research on the relative performance of different Deep RL methods (e.g. comparison of techniques such as *Deep Q Learning* and *Actor-Critic* methods).
- To perform research on the effect of differing reward structures on the performance of Deep RL agents, and enabling the development of reward structures that carefully balance average performance with safety (avoiding rare but catastrophic events).
- Ultimately, to be able to pre-train Deep RL agents which would then be transferred to, and used in, real world settings.

In order to test, train, and develop Deep RL agents, we need a standardised structure that we can use across different types of health systems. One such standardised structure, used across many differing domains, already exists, and that is OpenAI Gym (`gym.openai.com`)[5]. Gym provides a common interface to a range of problems, from control systems through to video games. The common interface allows the easy transfer of agents from one problem-solving environment to another. Gym is structured on an episodic framework to learning. The agent is exposed to multiple iterations, where the environment is *reset* to a fixed or random state, and the agent then interacts with the environment through a series of *steps* until some *terminal* state is reached indicating the end of the episode. With each step that agent passes an *action* to the environment. The environment returns an updated set of *observations* about the environment *state*, a reward, whether the terminal state has been reached, and any extra information available. Agents are designed to maximise the return of long term rewards.

In this paper we present a framework for coding Health Systems simulations, using the commonly used Python discrete event simulation package, SimPy[6] in a framework that allows interaction with a Deep RL agent. The framework uses RL agent method calls with high compatibility with OpenAI Gym, allowing easy transfer of agents developed with/for OpenAI Gym environments. As a demonstration, we use a simple hospital bed simulation model in SimPy, and show interaction with two Deep RL agents (written with PyTorch): a Double Deep Q Learning Network and a Duelling Deep Q Network. Our intention is not to provide a robust hospital bed simulation model, nor to provide an optimised Deep RL agent for such use, but to demonstrate a framework for combining Gym-compatible Deep RL agents with Health Systems simulations in SimPy.

# 2 GitHub repository

The GitHub repository containing this code is:

`https://github.com/MichaelAllen1966/learninghospital`

The examples cited in this paper are from release version 1.0.0 (DOI 10.5281/zenodo.3936515):

`https://github.com/MichaelAllen1966/learninghospital/releases/tag/v0.0.1`.

# 3 Method

## 3.1 Generic simulation properties

All simulations will share some common structure, methods, and attributes.

### 3.1.1 Generic structure

Algorithm 1 shows a high level structure of the code. This will be common to all interactions of Deep RL agents and SimPy simulations with only RL-specific alterations (such as the use of *target networks* and *memory*).

**Algorithm 1:** High level view of model using A Double Deep Q Network (using policy net, target net, and memory)

```
Set up policy net;
Set up target net;
Set up memory;
while Training episodes not complete do
    Reset sim;
    while not in terminal state do
        Get action from policy net;
        Pass action to sim;
        Take a time-step in sim;
        Receive (next state, reward, terminal,
         info) from sim;
        Add (state, next state, reward, terminal)
         to memory;
        Render environment (optional);
        Update policy net;
    end
    Update target net;
end
Assess performance of policy net;
```

### 3.1.2 Generic simulation methods

The simulation is set up with three methods that interface the Deep RL agent and the simulation:

- *reset*: resets the sim to a starting state and returns the first set of state observations.
- *step*: takes a step in the simulation. Passes an action to the simulation. Runs the simulation

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 July 2020 doi:10.20944/preprints202007.0598.v1

until the end of the next time step, and returns a tuple of next state, reward, terminal, info. The *step* method uses the SimPy method *env.run(until=target-time)*, with target-time being incremented in the desired time steps. When the simulation time reaches the desired maximum simulation duration, the simulation returns *terminal=True*.

- *render*: displays the current state of the simulation.

Other internal methods in the simulation (not accessed by the Deep RL agent) that will be common to all simulations are:

- *calculate_reward*: calculates the reward to pass back to the Deep RL agents.
- *get_observations*: creates a *list* of observations from the state.
- *islegal*: checks whether an action from the Deep RL agent is legal. If the action is not legal, this method will raise an exception.

#### 3.1.3 Generic simulation attributes

All simulations will contain the following attributes.

- *actions*: A list of possible actions.
- *action_size*: The number of possible actions.
- *observation_size*: The number of features in the observation.
- *state*: An object containing the state of the simulation. This may be a simple object, such as a list or dictionary, or may be a custom Python object.

### 3.2 Hospital bed simulation

#### 3.2.1 Hospital bed simulation overview

The hospital bed simulation is a very simplified model of a real hospital. Patients arrive at a hospital, stay for a given length-of-stay, and leave. The inter-arrival time of patients is sampled from an exponential distribution, the mean of which depends on the day of week (with average arrival numbers being higher on weekdays than weekends). The length-of-stay is also sampled from an exponential distribution, the mean of which does not depend on day of week. The hospital has a certain number of beds at any time. The Deep RL agent can request a change to the number of staffed beds, but this change is only enacted after 2 days. The simulation runs for 365 days by default, and the hospital is loaded initially with the expected average number of patients.

#### 3.2.2 Hospital bed simulation *state*

The state in the simulation is held by a dictionary. This dictionary contains:

- *weekday*: The current day of week (0-6).
- *beds*: The total number of staffed beds in the hospital (free or occupied).
- *patients*: The total number of patients in the hospital.
- *spare_beds*: The number of unoccupied beds. If the number of patients exceeds the number of staffed beds then this number becomes negative and indicated the number of patients without a bed.
- *pending_bed_change*: The changes in staffed bed numbers requested by the Deep RL agent, but which has not yet been actualised.

#### 3.2.3 Hospital bed simulation reward

The simulation has a target number of free staffed beds. By default this is set at 5% the number of patients in the hospital at any given time. The *reward* is always zero or negative and is the negative difference between the number of spare beds and the target number of spare beds (equation 1).

$$reward = -abs(spare\ beds - target\ spare\ beds) \quad (1)$$

#### 3.2.4 Hospital bed simulation methods

Methods that are specific to the hospital bed simulation are:

- *adjust_bed_numbers*: Adjusts the staffed bed numbers after a delay (SimPy *timeout*). Prior to the delay, the *adjust_pending_bed_change* method is called to track the requested changes in staffed bed numbers. The delay is the simulation time between the Deep RL agent requesting a change to the number of staffed beds, and the change being made. The delay is stored in the simulation attribute *delay_to_change_beds*, and may be set when initializing the simulation. When the number of staffed beds changes, the state dictionary items *beds* and *pending_bed_change* are adjusted accordingly.
- *adjust_pending_bed_change*: Adjusts the state dictionary item *pending_bed_change* when the Deep RL agent requests a change to the number of staffed beds.
- *load_patients*: Loads new patients at the start of the simulation, such that the initial number

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 July 2020 doi:10.20944/preprints202007.0598.v1

of patients in the hospital equals the calculated long term average (*arrivals per day* $*$ *average length of stay*). This method calls the *patient spell method*. This method increments the number of patients and staffed beds by 1 for each patient loaded into the simulation.

- *new_admission*: A continuous loop of new patients. This method/process is initiated on simulation reset. A new patient arrival is initiated by calling the *patient_spell* method. The number of patients in the hospital is incremented by 1. There is then a delay (SimPy *timeout*) before the next iteration of the loop. The delay is the inter-arrival time of patients. This is sampled from an exponential distribution, the mean of which depends on both the average arrival rate (set using *arrivals per day* attribute, which may be set when initializing the simulation. Mean arrivals per day are increased by 20% on weekdays (days 0-4), and reduced by 50% on weekends (days 5 & 6).
- *patient_spell*: The patient spell in the hospital. Length of stay is sampled from an exponential distribution based on a mean length of stay. If the patient is part of the initial load of the hospital, the length of stay is multiplied by a random number between 0-1 to account for the fraction of the length of stay already completed. After the spell in hospital is complete the number of patients is reduced by 1, and the number of spare beds recalculated.

#### 3.2.5 Hospital bed simulation reset method

The actions in the simulation *reset* method (required in all simulations for interaction with the Deep RL agents) are:

1. Create new hospital simulation environment.
2. Initialise simulation processes (*new_admission* method).
3. Set starting state values for state dictionary.
4. Call *load_patients* method.
5. Get and return first set of state observations.

#### 3.2.6 Hospital bed simulation step method

The actions in the simulation *step* method (required in all simulations for interaction with the Deep RL agents) are:

1. Check requested action is legal.
2. Adjust pending bed change.
3. Call bed change process.
4. Make a step in the simulation.
   Use: *env.run(until=self.next_time_stop)*.
5. Get new observations.
6. Check whether terminal state reached (based on simulation time).
7. Get reward.
8. Create an empty information dictionary (this dictionary is required to be compatible with OpenAI Gym step method).
9. Render environment if requested.
10. Return tuple of next state, reward, terminal, info.

#### 3.2.7 Hospital bed simulation attributes

Attributes that are specific to the hospital bed simulation are:

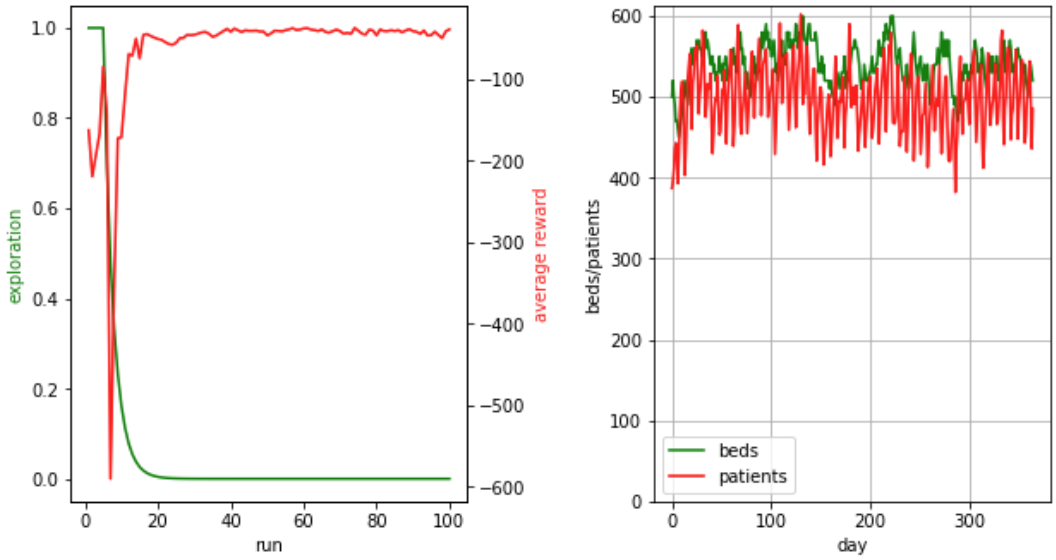
- *arrivals_per_day*: Average arrivals per day.
- *delay_to_change_beds*: Time between requesting change in beds, and change in beds happening (days).
- *los*: Average patient length of stay (days).
- *sim_duration*: Length of simulation run (days).
- *target_reserve*: target free staffed beds as a proportion of the number of patients present.
- *time_step*: Time between action steps (day).

### 3.3 Deep Reinforcement Learning Agents

A range of standard Deep RL agents were implemented for this model (and are provided in separate notebooks in the associated GitHub repository):

1. *Double Deep Q Network (D2QN)*: Standard Deep Q Network, with policy and target networks[7].
2. *Duelling Double Deep Q Network (D3QN)*[8]: Policy and target networks calculate Q from sum of *value* of state and *advantage* of each action (*advantage* represents the added value of an action compared to the mean value of all actions).
3. *Noisy D3QN*[9]: Networks have target layers that add Gaussian noise to aid exploration.
4. *Prioritised replay D3QN*[10]: When training the policy network, steps are sampled from the memory using a method that prioritises steps where the network had the greatest error in predicting Q.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 July 2020 doi:10.20944/preprints202007.0598.v1




| | |
|---|---|
| days under capacity | 54.0 |
| days over capacity | 306.0 |
| average patients | 497.0 |
| average beds | 540.0 |
| % occupancy | 91.9 |

Figure 1: Example model. A Bagging Duelling Double Deep Q Network agent trained to manage bed capacity of the hospital bed simulation. *Left*: Exploration (epsilon, the probability of taking an action purely at random, green) and average reward across the training episode (red). *Right*: The number of patients (red) and staffed beds (green) in the last training run

5. *Bootstrapped D3QN* [11]: Multiple networks are trained from different bootstrap samples from the memory.

6. Combinations of the above

# 4 Results

The GitHub repository contains examples of the various Deep RL agents implemented.

The output of the Bagging D3QN is shown in figure 1. It is not the intention of this paper to present a fully optimised Deep RL agent, but it can be seen that the example network improves in performance over time (repeated model runs) and manages the modelled bed stock appropriately.

# 5 Discussion

Combining Deep RL and Health Systems Simulations has significant potential, for both research into improving Deep RL performance and safety, and in operational practice. Our aim in this paper is not to present an optimised Deep RL model, or a detailed hospital simulation, but to provide a framework that is compatible with OpenAI Gym environments, enabling easy transfer of the many methods that have been developed and tested in such environments.

The potential for combining Deep Learning and Health Systems Simulations goes beyond the framework provided here. For example, we have demonstrated that a machine learning model can be used to simulate patient-level clinical decision making [12] as part of broader clinical pathway simulation study.

The combination of Deep Learning and Health Systems Simulation is an area of research that will hopefully bear much fruit in the coming years.

# 6 References

# References

[1] Y. Li, "Deep reinforcement learning: An overview," *arXiv:1701.07274 [cs]*, Nov. 2018. arXiv: 1701.07274.

[2] T. Monks and R. Meskarian, "Using simulation to help hospitals reduce emergency department waiting times: Examples and impact," in

*2017 Winter Simulation Conference (WSC)*, pp. 2752–2763, Dec. 2017. ISSN: 1558-4305.

[3] M. L. Penn, T. Monks, A. A. Kazmierska, and M. R. A. R. Alkoheji, "Towards generic modelling of hospital wards: Reuse and redevelopment of simple models," *Journal of Simulation*, vol. 14, pp. 107–118, Apr. 2020. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/17477778.2019.1664264.

[4] T. Monks, D. Worthington, M. Allen, M. Pitt, K. Stein, and M. A. James, "A modelling tool for capacity planning in acute and community stroke services," *BMC Health Services Research*, vol. 16, p. 530, Sept. 2016.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv:1606.01540 [cs]*, June 2016. arXiv: 1606.01540.

[6] SimPy, "Simpy. discrete event simulation for python.," *https://simpy.readthedocs.io/en/latest/*, 2020.

[7] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *arXiv:1509.06461 [cs]*, Dec. 2015. arXiv: 1509.06461.

[8] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," *arXiv:1511.06581 [cs]*, Apr. 2016. arXiv: 1511.06581.

[9] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy Networks for Exploration," *arXiv:1706.10295 [cs, stat]*, July 2019. arXiv: 1706.10295.

[10] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv:1511.05952 [cs]*, Feb. 2016. arXiv: 1511.05952.

[11] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep Exploration via Bootstrapped DQN," *arXiv:1602.04621 [cs, stat]*, July 2016. arXiv: 1602.04621.

[12] M. Allen, K. Pearn, T. Monks, B. D. Bray, R. Everson, A. Salmon, M. James, and K. Stein, "Can clinical audits be enhanced by pathway simulation and machine learning? An example from the acute stroke pathway," *BMJ Open*, vol. 9, p. e028296, Sept. 2019.

## Funding

This study was funded by the National Institute for Health Research (NIHR) Applied Research Collaboration (ARC) South West Peninsula. The views and opinions expressed in this paper are those of the authors, and not necessarily those of the NHS, the National Institute for Health Research, or the Department of Health.