Particle Methods and Deep Reinforcement Learning

Pietro Miotti
miottp@usi.ch

Abstract

Applying AI to Physics and related fields is one of the most challenging problems nowadays. In this project is presented a possible way to use Deep Reinforcement Learning in a well defined environment which simulates the emergency evacuation problem, modelled using Particle Methods. The agent needs to escape from a room as fast as possible, trying to find the exit door and avoiding other people, walls and other obstacles: the interactions between the agent and other obstacles are given.
The problem has been model as a continuous state space and discrete action space problem. Two different algorithms have been explored: the canonical State-Value DQN and the gradient-based PPO which shows better stability and performance. Overall the model can simulate emergency evacuation in different environments with multiple room exits.

1 Introduction

How to plan Emergency Rooms in order to minimize the escaping time of all the people from the building has always been an important topic in civil engineering and a fundamental aspect to take in to account when designing an edifice. It is well known that one of the main cause of injuries and deaths during the emergencies is not the fact that causes the emergency itself while much more the panic that drives people to irrational behaviours. For this reason many mathematical models have been developed to predict human behaviours and in such way predict paths followed by people during emergencies.
One of the most used model in the field is called *Social Force* model [1] which represents each person as an agent that interacts with the environment (obstacles and other agents) with certain forces (discussed in more detail in Section 3) and driven to the the exit through a *Self-Driven Force* that can be considered as an attraction force that links each agent to the nearest exit which somehow would represent the tendency of a person to search for the most near exit while avoiding other agents. However, this model is not optimal when need to deal with complex environments and obstacles and certain heuristics based on the agent's local information are introduced such as considering distance from obstacles and so on.
For this purpose in this project it has been explored a general framework by considering a complete blinded agent (no local information are considered nor knowledge of exit doors) whose actions are driven by a Deep Reinforcement Learning algorithm that tries to find the best policy which minimize the time needed to escape from the exit door.
The strong connections between Particle Methods modelling and the classical Reinforcement Learning setting have been exploited to remodel the escaping emergency room problem as a Reinforcement Learning problem.
Particle methods modelling techniques used to model dynamical systems (Agent Based modelling, Molecular Dynamics, Dissipative Particle Methods and so on) have all in common their basic structure in which each agent interacts with other agents according to certain rules and the combination of all of these interactions give rise to global behaviours that can be studied in its completeness. At each time-step, for each agent are computed all the interactions forces between it and other agents in the system and once this is done, by using the second Newton's Law, the new accelerations of each agent are computed and thus the new positions too; this process is repeated for N time steps.
The classical Reinforcement Learning setting on the other hand can be described by the interaction of an agent with the environment, which provides rewards and new state as response to certain action.
The emergency room evacuation problem can then be seen as a reinforcement learning problem in which the reward is -1 at each time step with the exception of when the exit door is reached (0 reward in that case), the action is the direction of the aformentioned *self-driven* force and the new state is computed by the environment using the second Newton's Law.
This papers is structured as follows: in Section 3 the modelling methodology is presented, In section 4 is discussed the implementation of the algorithms, in Section 5 are reported the main results obtained from the experiments and in Section 6 some possible extensions are discussed.

Advanced Topics in Machine Learning, SA 2021-2022

2 Related works

In recent years there have been many attempts in trying to link AI with Physics. One of the most remarkable result achieved in Fluid Mechanics, according to author's opinion, has been achieved by Guido Novati et al [2]: by using an improved version of the Actor Critic Algorithm to learn the movements done by micro-swimmers in order to maximize their velocities; they have discovered experimentally that the behaviour chosen by the algorithm was the one used in nature by fishes in order to avoid vortices in fluids and in such way maximize their velocities. However one of the unique application of DRL to a particle based problem, at least from the knowledge of the author, it is the work done by Yihao Zhang et al [3] in which they proposed a Dyna Q-Learning algorithm for the Emergency Room Evacuation problem, from which the present project has been strongly inspired by.

3 Methods and Models

3.1 Social Forces

As anticipated in Section 1, for this project all the connections between particle based method and Reinforcement Learning setting have been exploited.
Generally, in most of the Reinforcement Learning problems (i.e *Pendulum, Pong,* etc.) the environment is given by the series of images that describe what is happening in the *game*: the state is the current picture, the new state is the picture that follows a certain action and the neural networks tries to infer the rules of the games and how to take actions accordingly. In the setup presented in this project instead, the environment is defined as the set of rules and forces that determine the movements of the agents and hence their new states. More specifically the set of rules considered are the ones introduced by the Social Force model [1] and implemented using the particle based framework.
A person is modelled as an agent and it is driven by the following forces:

- **Avoidance Force**: the force which represents the tendency to avoid other agents, it is a repulsive force. A is the avoidance parameters and regulate how strong is the force. $r_{i,j}$ is the distance between the agent *i* and the agent *j* (defined as $|r_i - r_j|$), $d_{i,j} = \frac{d_i - d_j}{2}$ is equilibrium distance between two agents and $\hat{r} = \frac{r_i - r_j}{|r_i - r_j|}$ is the unit vector pointing from *j* to *i*.
$$F_{i,j}^{avoidance} = Ae^{\frac{d_{i,j} - r_{i,j}}{B}}\hat{r} \tag{1}$$
- **Compression Force**: the force which models the fact the agent represent a person and hence two agents cannot be infinitively close to each other. It is a repulsive force. G is the heaviside function and k = $10^4 kg/s^2$ determines the obstruction effects in cases of physical interactions.
$$F_{i,j}^{compression} = kg(r_{i,j} - d_{i,j})\hat{r} \tag{2}$$
- **Friction force**: the force which represents interaction between pedestrians moving with different velocities. $u_{i,j} = (v_j - v_i)\hat{t}_{i,j}$ is the relative tangential velocity such that $\hat{t}_{i,j}\hat{r}_{i,j} = 0$.
$$F_{i,j}^{friction} = kg(r_{i,j} - d_{i,j})\hat{u} \tag{3}$$
- **Self Driven Force**: the force that drives the agent directly to the exit. $v_{desired}$ is the modulus of the velocity which represents the willpower of the agent to achieve the exit, *m* is the mass, δt is the integration interval and f_action is the direction in which the desired velocity is expressed. This value is provided by the Deep Reinforcement Algorithm discussed in details the following paragraph.
$$F^{selfdriven} = m\frac{v_{desired}}{\delta t}f_{action} \tag{4}$$
- **Viscous dumping force**: if only a constant self-driven force was applied to an agent, then the agent would accelerate continuously and reach an unrealistic speed before it collides with another agent or an obstacle. Thus, in an environment without *viscosity*, agents will move and collide violently. Because of this, the viscous damping force has been also introduced. v is the velocity of an agent.
$$F^{viscous} = -m\frac{v}{\delta t} \tag{5}$$

  Therefore, the overall force that acts to the agent *i* is given by the following expression where N are the other agents of the system.

$$F_i^{tot} = F_i^{selfdriven} + F_i^{viscous} + (\sum_j^N F_{i,j}^{avoidance} + F_{i,j}^{compression} + F_{i,j}^{friction}) \tag{6}$$

Once the total force that acts on the agent *i* is obtained, it is necessary to compute the integration over the acceleration (exploiting the Newton's Law $F_i^{tot} = ma$) in order to get the real displacement of the agent and hence computing its new position and velocity.
For this purpose the *verlet* integration scheme has been used and it is described as follows:

- Calculate $\vec{v}(t + \frac{1}{2}\Delta t) = \vec{v}(t) + \frac{1}{2}\vec{a}(t)\Delta t$
- Calculate $\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t + \frac{1}{2}\Delta t)\Delta t$
- Derive $\vec{a}(t + \Delta t)$ by using new positions
- Calculate $\vec{x}(t + \Delta t) = \vec{v}(t + \frac{1}{2}\Delta t) + \frac{1}{2}\vec{a}(t + \Delta t)$

3.2 Deep Reinforcement Learning

In the Reinforcement Learning setting the fundamental elements are the state of the environment (the information that the environment has in order to provide the reward and the next states to all the agents involved), the state of the agent (the information that the agent analyzes in order to take the next action), the action taken by the agent, the reward given by the environment to the agent and the policy which provides to the agent the probability of taking a certain action given a determined state.
As presented above, the environment is ruled by the social forces model, whereas the Reinforcement Learning is used to learn what is the best action associated to a given state that leads the agent to exit the room as fast as possible Fig: 1

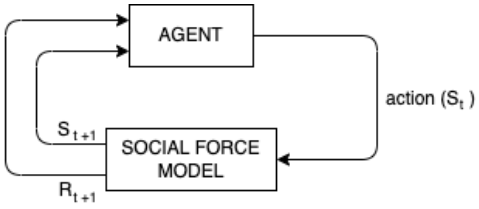


Figure 1: Reinforcement Learning Setting

3.2.1 Agent State

As Yihao Zhang et al. did in their work [3], for this project the state of the RL agent at time t is given by:

- $\vec{x}(t)$: agent position
- $\vec{v}(t)$: agent velocity

Therefore a completely blinded agent is considered (i.e does not have any information about the environment).
The state space is continuous.
The state of the environment instead are the positions, velocities and accelerations of all the agents and obstacles of the system.

3.2.2 Policy

For the sake of simplicity, in this project the action space is discrete which means that the policy is discrete. Since the simulation is performed in a 2D domain, the agent can choose between 8 different directions reported in Fig: 2, the modulus $v_{desired}$ is then projected in its x and y components accordingly.

3.2.3 Reward

Since the agent needs to escape as fast as possible, at each time step a -1 reward is given, when the exit is reached the execution terminates with a final 0 reward.

3.3 Deep Q-Learning Neural Network

The first approach tested in this project has been the model-free Q-Learning algorithm defined by Watkins et al. [5] with Experience Replay.

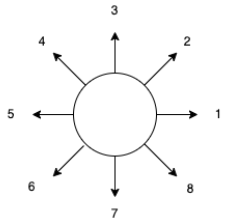


Figure 2: Actions

The main idea behind Q-learning is that if we want to maximize the discounted (using the discount factor γ), cumulative reward $R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$, if we had a function $Q^* : State \times Action \rightarrow \mathbb{R}$ that could provide to us an estimate of what our return would be, then given a certain state, we could easily construct a policy that maximizes the rewards just by following the action that provides the maximum value of Q (greedy policy):

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s, a)$$

However, since Q is not a-priori known it can be approximated using Neural Networks trained exploiting the Bellman equation which provides the error that need to be minimized to improve the estimate.

$$\delta = Q(s, a) - (r + \gamma \max_a Q(s', a))$$

We also used Experience Replay technique which consists in storing the transitions (given by the tuple (Current State, Action Taken, Next State, Reward obtained)) in a buffer and sample from that the training data. In such a way the transitions that build up a batch are decorrelated and it has been shown that this greatly stabilizes and improves the DQN training procedure.

In this setting, given the function Q approximated using Neural Network, and given the current state of the agent $(\vec{x}, \vec{v})$ the action taken by the agent is determined by the greedy policy $\operatorname{argmax}_a\ Q^{\pi}(s, a)$ which provides as output a number from 1 to 8 that corresponds the action to take as reported in Fig: 2

3.4 Proximal Policy Optimization

The second approach tried in the project is the policy-based algorithm Proximal Policy Optimization [4]. PPO lies in the family of Policy Gradient Algorithms which instead of trying to learn the Q-value function as DQN, and take the action greedly on that, they directly tries to learn the policy.
The main idea behind Policy Gradients method is to approximate the policy function and maximize the objective function defined as the expectation of the rewards $J(\Theta) = E_{\pi_\Theta}(R)$.
The PPO algorithm provides an improved and more stable version of the objective introduced above, more specifically it limits the improvement of the policy from one step to the other by using importance sampling and clipping its value. More in details, by setting:
$r_t = \frac{\pi_\Theta(a_t|s_t)}{\pi_{\Theta_{old}}}$ as the importance sampling ratio
$A_t = -V(s_t) - r_t + \gamma r_{t+1} + ... + \gamma^{T-1} V(s_T)$ as the advantage estimator
The objective function defined by the PPO algorithm is defined as follows:

$$L^{CLIP}(\Theta) = E_t[\min(r_t(\Theta)A_t, clip(r_t(\Theta), 1-\epsilon, 1+\epsilon)A_t] \tag{7}$$

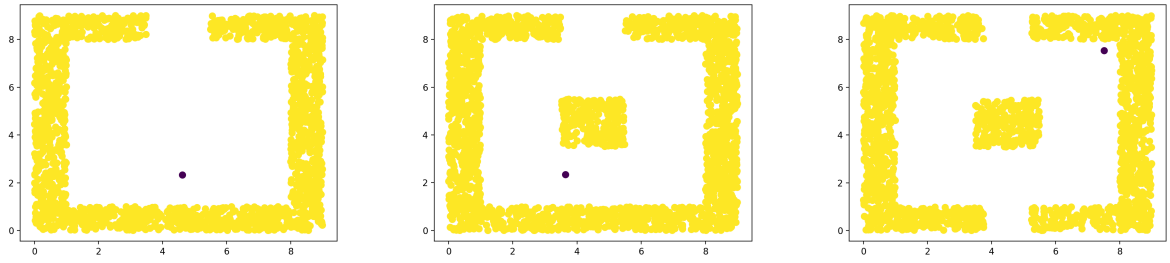
Since π is not apriori know it can be approximated using a Neural Network.
In this setting, since the action space is discrete, the neural network approximates a Categorical Distribution which provides as output a vector of 8 entries which represent the probabilities of taking one of the eight actions described in Fig: 2.

4 Implementation

4.1 Environment

The environment has been developed from scratch using the Gym superclass provided by openAI and the methods *init, step, reset* have been overwritten accordingly.

The environment consists in a 2D room 10m x 10m where the walls are modelled using agents that has been freezed at t_0: in this way the interaction of the agent with the walls is implemented as a canonical agent-agent interaction.
The environment can be customized by adding doors and changing their lenghts, or adding a squared obstacle in the middle of the room. An agent achieves the goal when reaches one of the door in the environment.



(a) (b) (c)

Figure 3: (a) 1 door (b) 1 door 1 obstacle (c) 2 doors 1 obstacle

4.2 Agent

In order to speed up the computations and make the project learnable with limited computational resources, several simplifications have been introduced:

- during the training only 1 agent it used for learning and no other agents are present in the environment with the exception of the ones that build walls and obstacles. In this way the problem becomes a single-agent problem and not a multi-agent problem.
- neighbour lists have been neglected, only the distance with walls and obstacles is computed, this saved a lot of computations.

4.3 PPO

The PPO implementation has been inspired by [7]
In order to avoid infinite episodes for this project we limited the number of timesteps per episode to 1000. By considering this, we have sliglty modified the Monte Carlo method used during the rollout to compute the rewards as follows (which can remind to a TD(1000)):

$$r_t = \begin{cases} r_{notfinal} & \text{if } \textit{not done} \text{ and } t < 1000 \\ r_{final} & \text{if } \textit{done} \text{ and } t < 1000 \\ r_t + \gamma * V(s) & \text{if } \textit{not done} \text{ and } t >= 1000 \end{cases} \tag{8}$$

where *done* is a boolean which is =1 when a door is reached.

4.4 DQN

The DQN implementation has been inspired by [8]. For the DQN the max iteration per episodes is set to 5000.

4.5 Hyperparameters

We considered an agent with mass = 80kg and a desired velocity equal to 8 m/s.

4.5.1 DQN

- BATCH_SIZE = 128 # self explanatory
- REPLAY_BUFFER_CAPACITY = 200000 # transition buffer from which perform the sampling

- DISCOUNT_FACTOR = 0.999 # self explanatory
- LEARNING_RATE = 0.001 # self explanatory
- EPISODE TRAINING = 3000 # self explanatory

4.5.2 PPO

- BATCH_SIZE = 128 # self explanatory
- ROLLOUT_BUFFER = 4000 # Update policy every 4000 steps
- DISCOUNT_FACTOR = 0.999 # self explanatory
- LEARNING_RATE_ACTOR = 0.001 (for 2 doors) and 0.0003 (for 1 door)
- LEARNING_RATE_CRITIC = 0.001 (for 2 doors) and 0.0003 (for 1 door)
- K_EPOCHS = 50 # Update policy for K epochs
- EPISODE TRAINING = 4000 # self explanatory
- CLIP PARAMETER FOR THE PPO = 0.2

For further details readers are strongly invited to check the source code reported in the section below.

4.6 Experimental setup

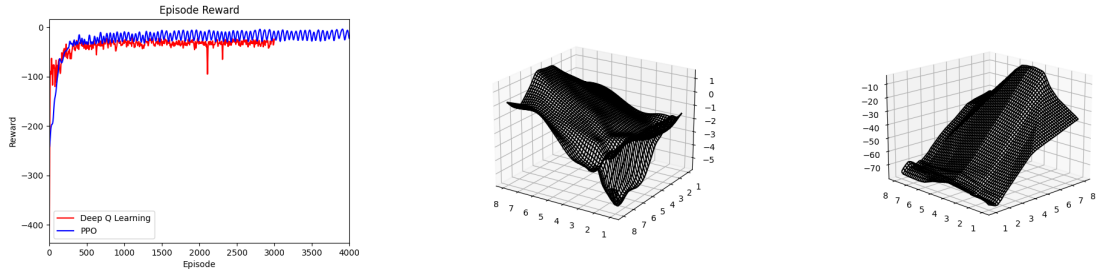
All the experiments have been ran using a MacBook Pro 2.6 GHz 6-Core Intel Core i7, 16 GB 2667 MHz DDR4.
The code is available at `https://github.com/pietromiotti/DeepReinforcementLearning_ParticleBased`

5 Results

In the following section some results from different configurations of the environment are presented.
More specifically the following setup are tested

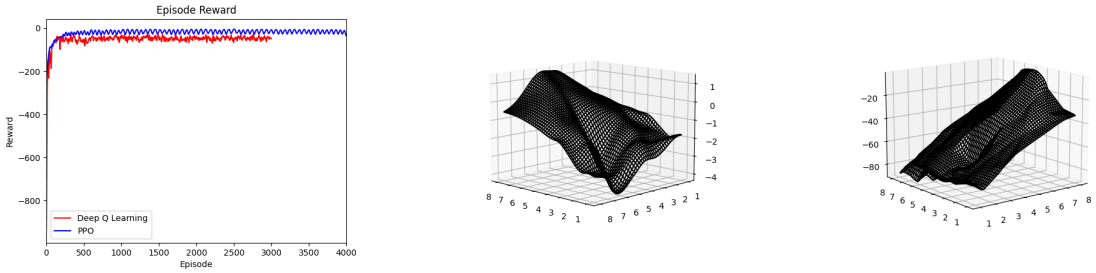
- Environment with 1 door (2m) and 0 obstacle: Fig 4
- Environment with 1 door (2m) and 1 obstacle (2m): Fig 5
- Environment with 2 doors (2m) and 0 obstacle: Fig 6
- Environment with 2 doors (2m) and 1 obstacle (2m): Fig 7

For each experiment is provided the comparison between the PPO and DQN learning curves and the value function which associates to each state of the agent its corresponding value that represents the *quality* of the state itself (i.e the expectation of the reward obtained by being in that state). As a test case a uniform grid of coordinates as $\vec{x}$ is considered with $\vec{v}$ equals to 0. It can be see by the plot of the value functions that the value of the value function is higher in proximity of the doors, as expected.



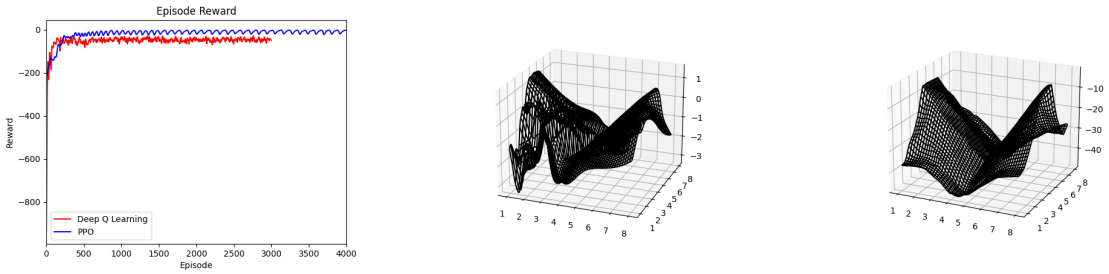
(a) (b) (c)

Figure 4: (a) Comparison (b) PPO Critic Value Function (c) DQN Value Function



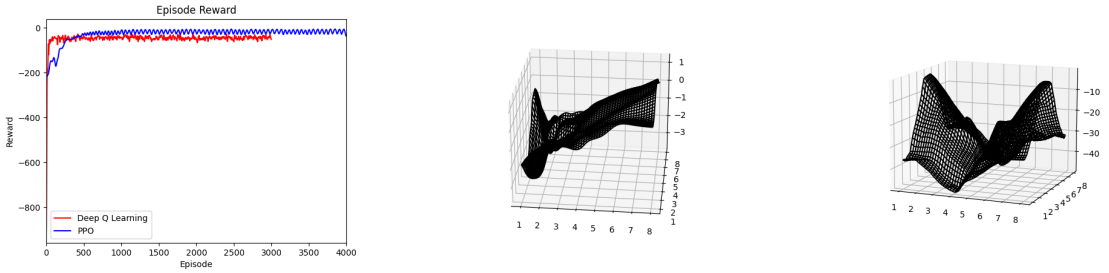
(a) (b) (c)

Figure 5: (a) Learning Curves (b) PPO Critic Value Function(c) DQN Value Function



(a) (b) (c)

Figure 6: (a) Learning Curves (b) PPO Critic Value Function (c) DQN Value Function



(a) (b) (c)

Figure 7: (a) Learning Curves (b) PPO Critic Value Function (c) DQN Value Function

6 Discussion and conclusion

The original aim of the project (i.e. learning actions from a well defined environment) has been achieved successfully. By looking at the results reported in Section 5 it is possible to conclude that generally the value function learned from the DQN is better (less noisy) to the one learned by the critic of the PPO, however the PPO method not only showed a better convergence but also better stability: the DQN algorithm in fact, after the 3500^{th} epoch starts to overfitting the Q-function and leads to bad performances, whereas this not happen using the PPO. The average time to achieve the closest door is comparable between the two methods, this can be seen by looking the average episode rewards.
Further extensions can be implemented by modelling a continuous action space using PPO. Some attempts have been made by the author in this path by considering a Gaussian Policy distribution which output is normalized and clipped between -1 and 1 as did in [6] in order to get the right movement angle, however the training was too expensive and cannot be completed successfully using limited resources.
Particle methods and reinforcement learning shares many properties and more importantly they both have an agent-based structure; this fact makes every particle based problem a very good candidate to be solved using Reinforcement

Learning. This project showed only one very simple possible application, but many more can be explored in the future. An important experiment that can be done would be training a Deep Reinforcement Learning agent to take the action needed to reach a minimum in terms of energy of the whole system, this could be a breakthrough achievement that would be a game-changer in the world of chemistry and biology.

References

[1] Dirk Helbing, Illés Farkas Tamás Vicsek, 2000, *Simulating dynamical features of escape panic*, Nature,

[2] Peter Gunnarson, Ioannis Mandralis, Guido Novati, Petros Koumoutsakos John O. Dabiri, 2020, Learning efficient navigation in vortical flow fields

[3] Yihao Zhang, Zhaojie Chai, George Lykotrafitis, 2020, Deep reinforcement learning with a particle dynamics environment applied to emergency evacuation of a room with obstacles

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 2017, Proximal Policy Optimization Algorithms.

[5] Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou, Daan Wierstra Martin Riedmiller, 2013, Playing Atari with Deep Reinforcement Learning.

[6] Yusheng Jiao, Feng Ling, Sina Heydari, and Eva Kanso, 2020, *Learning to swim in potential flow*, `https://arxiv.org/pdf/2009.14280v2.pdf`

[7] PPO implementation `https://github.com/nikhilbarhate99/PPO-PyTorch/blob/master/PPO.py`

[8] DQN implementation `https://github.com/khordoo/Deep-Reinforcement-Learning-with-PyTorch/blob/master/03-DQN/cartpole-dqn.py`