University of the Basque Country

# Deep Reinforcement Learning in PyTorch

Unai Lizarralde Imaz

Ferran Tudela García

October 8, 2019

# Contents

# 1 Introduction

In an ordinary human environment, the individual‘s growth and learning process are carried out through getting accustomed to the surrounding and the various situations that need to be overcome. For a machine, this learning process is a precise method that requires some standardized steps for developing knowledge and the capability to respond to various perceptions with it. In this scope, reinforcement learning is a method for acquiring the necessary intelligence to tackle with the task at hand very closely to how a human being performs his learning in the different stages of life; that is to say, when a human learns how to do a labor correctly, also learns the actions that should be taken to successfully achieve the desired result, and optimizing the set of actions depending the situation to move from one state to another is the basis of reinforcement learning.

In order to represent the states and actions to reach such states, a clear and well-organized structure is required. For such purpose, the learning process will be based on deep learning techniques to generate an artificial neural network. This way, optimizing and assigning the correct action to the right perception is the key to learning and following a reinforcement learning method. The learning process can be divided into three essential parts, namely, repetitively taking actions in a loop, not being told which actions optimize the output and attempt to approach the closest path of actions that maximize the reward.

Therefore, in this report we will be including the several approaches (algorithms) that are available to learn with the use of deep reinforcement learning in PyTorch. In each of the types of algorithms covered, an example will be provided to understand both theory and work areas in which to apply it.

# 2 Implemented Algorithms

For the wide array of problems present to solve through Artificial Intelligence and Machine Learning, several approaches based on the parameters and specifics of the problem are examined to choose the best one. In that regard, algorithms based on Deep Reinforcement Learning are constructed to adapt to the environment of the problem using the most convenient approach.

These algorithms train an agent to learn how to perform actions in an environment with the purpose of obtaining the maximum reward possible out of each action. Therefore, these algorithms take the agent through a learning experience of repetition, placing the agent in a specific situation on the environment (state), for which it has to take the best decision (action) to advance in the most profitable (reward) way. For instance, maximizing the score obtained in a game could be a reward to pursue for which the taken actions should have as the objective increasing that score in the short and long term for each stage of the game.

In the next sections, these different types of approaches will be showcased and, in each one of them, some of the algorithms are going to be presented to give contrast to the diverse philosophies of design found in them.

## 2.1 Policy Gradients

Policy gradient is an approach to solve reinforcement learning problems. The goal of reinforcement learning is to find an optimal behavior strategy for the agent to obtain optimal rewards. The policy gradient methods target at modeling and optimizing the policy directly.

### 2.1.1 A2C

A2C (Advantage Actor-Critic) is a synchronous, deterministic version of A3C (Asynchronous Advantage Actor-Critic). This method, like A3C, focuses on parallel training. The main difference with A3C, is that uses a coordinator that waits for all the parallel actors to finish their work before updating the global parameters, and then in the next iteration, parallel actors starts from the same policy. This solves the issue with A3C, that sometimes the thread-specific agents would be playing with policies of different versions and therefore the aggregated update would not be optimal. A2C can utilize GPUs more efficiently and work better with large batch sizes while achieving same or better performance than A3C.

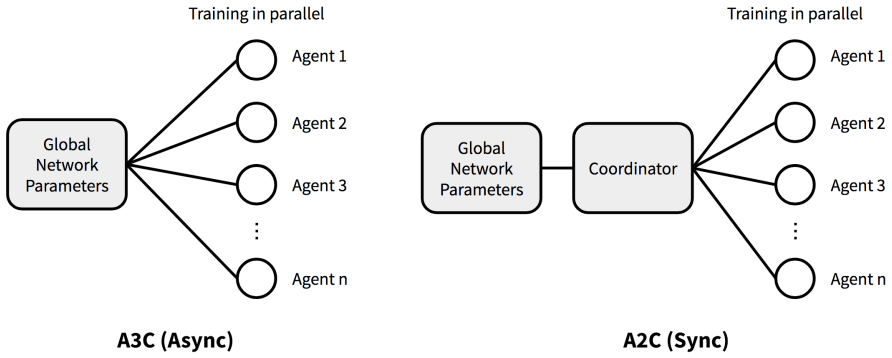


Figure 1: The architecture of A3C versus A2C [1].

### 2.1.2 PPO

PPO (Proximal Policy Optimization) is a simplification of the TRPO (Trust Region Policy Optimization) policy gradient algorithm. Where TRPO tries to solve this problem with a complex second-order method, PPO is a family of first-order methods that use a few other tricks to keep new policies close to old. PPO methods are significantly simpler to implement, and performs as well as TRPO.

It uses two policy networks, one that contains the current policy to improve, and a second that contains the last policy used. The reward is denoted by the probability ratio between the old and the new policy.

To improve training stability, instead of imposing a hard constraint, avoids parameter updates that change the policy too much at one step, and formalizes the constraint as a penalty in the objective function.

Without a limitation it would lead to instability with extremely large parameter updates and big policy ratios, for that reason a constraint is imposed, that forces to stay within a small interval around 1. Therefore, we lose the motivation for increasing the policy update to extremes for better rewards.

When applying PPO on the network architecture with shared parameters for both policy (actor) and value (critic) functions, in addition to the limited reward, the objective function is augmented with an error term on the value estimation and an entropy term that encourages sufficient exploration.

## 2.2 Deep Q-Learning

The concept behind Q Learning lies in knowing the current state-action‘s immediate reward and the long term maximum benefit that could be obtained, which is estimated from the previous experience and serves as a guide for future-proof. In the core, Q-value is the main focus of the Q learning process, that is to say, this function projects a mapping from pairs of states and actions to the associated reward for performing such action in that specific state. In a nutshell, the aim of this Q Learning method is to determine the best sequence of actions to achieve the highest reward posible out of each state, and to stockpile the results acquired from each run for future reference as experience grows steadily.

Briefly, there are two components that are affected by the current state and action, namely, the reward of performing action “a” in a state “s” and the prediction of the reward in the long run if that action is done in that state. The basic formula that connects all these factors together is shown ahead:

$$Q(S, A) = R(S, A) + \gamma \max_{A} Q(S', A)$$

Q stands for the Q-value of action “a” being undergone in state “s”. As explained before, r is the immediate reward; whereas Q is once again the Q-value of the next state after having taken the action in the current one. However, this time the best possible action is considered to be chosen for the following state to optimize the reward. Moreover, a parameter decides how influential should the future circumstances be, reducing its contribution accordingly. This parameter serves as the coefficient to restrict how reliable the gathered experience already is.

All in all, the experience is key for the agent to learn progressively in the good direction; unfortunately, the massive size of the problem to solve has direct impact on how much time and space will be required for the agent to earn enough experience for effectively tackle with the problem. In these particulars, a much more efficient form of continuous learning is required though. For that end, a deep learning approach using machine learning and neural networks provide tools, methods and predictive models to carry out the task successfully. Specifically, for deep Q-learning, a neural network representation is used.

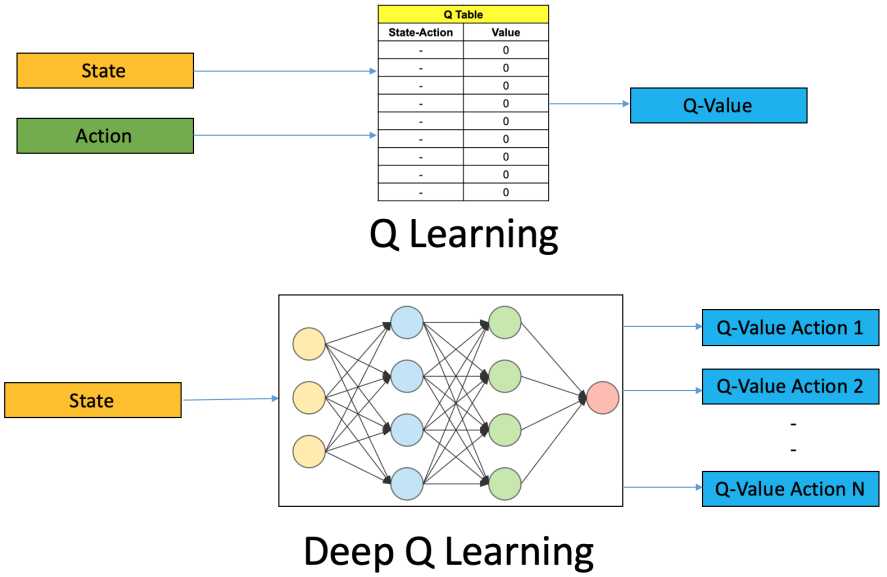


Figure 2: Difference in perspective between Q-learning and Deep Q-learning methods

A deep Q-learning neural network (DQN) takes the state as the input, unlike with Q-learning were both the state and the action are the input for the function, and going through neural layers the outputs include every Q-value approximated for each available action in the given state. Therefore, the training process has the objective of refining the Q-values produced by the DQN in order to be as close as possible to the desired result. The notion of experience is still prominent, but, this time around, the experience is the source for adapting to the specifics of the problem the DQN for it to predict Q-values with the best of the estimations. With this method, the use of a DQN effectively reduces the overall time required to train the agent with the sophisticated inclusion of neural networks into the equation.

This way, the refinement of the Q-values is the main objective towards the best possible reward. For reaching that end, the next recurrence includes all the necessary parts to the fluctuations of each of the Q-values, which is the experience to improve the performance for the selected task at the very end.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_A Q(S_{t+1}, A) - Q(S_t, A_t))$$

As it can be observed, to the already computed Q-value for action-state pair, a modification, standardized with the parameter alpha, is produced with the target Q-value's (the immediate reward R and the best action for maximizing the Q-value for the next state) and the already computed Q-value's difference. A similar scenario as looking for the gradient of the function through recurrently approximating it. In few words, the objective is to get as close as possible to the target value from the predicted value using the neural network and giving the corresponding values to the parameters depending on the final results (experience) after observing the evolution through states.

# 3 PyTorch

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs in Python. Its main advantage compared to other libraries, such as Numpy, is that it is ready to use not only the CPU to perform the computations required to manage data, but also the GPU. Ultimately, this hybrid optimization allows for a powerful library consisting of highly efficient functions and utilities for deep learning.

Owing to its efficiency in computations and embedded properties to be available in several platforms (PyCharm for instance), it is considered to be one of the most reliable frameworks in the data scientist community. Although, alternatives exists, namely, TensorFlow as its greatest competitor, it still holds itself and receives constantly updates that further expand its capabilities.

The library is coded in C++ to reach the peak performance. What's more, it has already included the possibility to use a front end C++ environment to develop in line within the library functions. This further increases the adaptability, reliance and integration of new utilities on the library. Therefore, it is not far-fetched to consider it as one of the most versatile libraries in the deep learning area.

All the previously presented algorithms are available in PyTorch, and in order to showcase its functionality, results obtained in our tests are going to be included.

## 3.1 Library's algorithms results

To try and test the algorithms improvements that provides rlpyt, a initial setup is required. This library currently only have support for Linux and Mac, and only have

partial support for Windows. The WSL(Windows Subsystem for Linux) console can be used to install and run the library. But, using WSL has a drawback, it requires more configurations and more steps for some features to work.

The library provides some examples that uses the Atari environment, which is already included with the repository and also have support for *mujoco* and *gym* environments. It is also compatible with a library called *viskit*[2], that allows to generate graphs of the data obtained from the neural network.

After running the examples and observing the result, it can be said that, depending on the type of the problem, one algorithm can perform better than another. This can be observed when comparing the multiple algorithms applied to the different Atari games.

After almost over 3M steps the returned values graph started looking like the one showed in the rlpyt white papers.

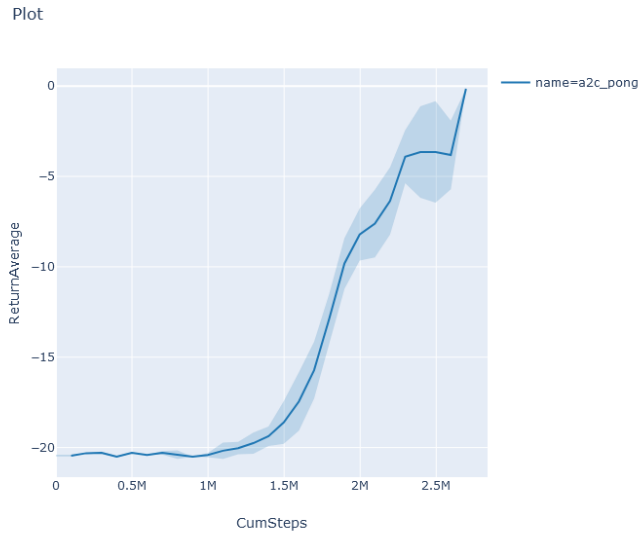


Figure 3: A2C algorithm

Compared with the graph of our A2C algorithm with the graphs of the papers, we can see that the section matches.

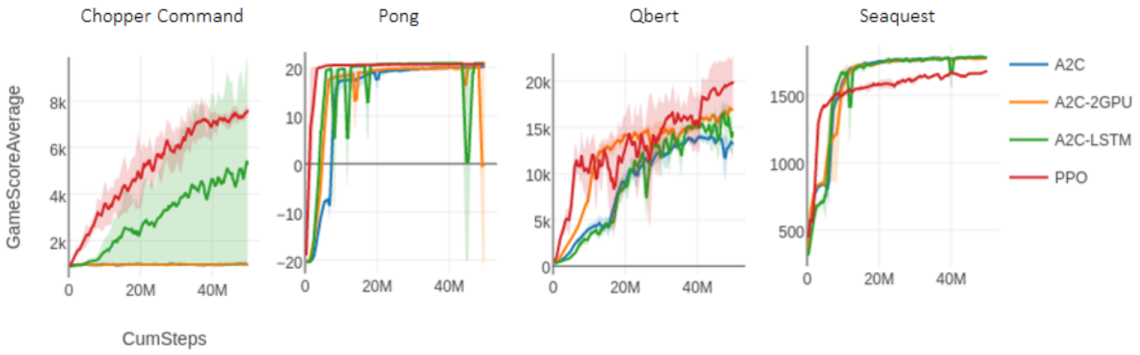


Figure 4: Policy gradient algorithms

# 4 Contributions

Reinforcement Learning is a field of constant evolution and adaptation to different situations that require close-to-human behaviour to complete a task successfully. The power of Deep Reinforcement Learning is that, unlike a common imperative approach where you decide rigidly what kind of actions are going to be taken, the agent learns directly how to act through a learning process from scratch and, thanks to that, there is no more versatile method to solve a wide array of problems than teaching directly how to do it from attempting it over and over again.

Moreover, Deep Reinforcement Learning is one of the fields of study that requires professionals that are ready to immerse and research thoroughly for even more sophisticated algorithms to bloom. This has led to a huge rise in the work occupation of data scientist, consequently, increasing the demand of it. For that reason, working in this area of research in Machine Learning and Artificial Intelligence is not only on the top of the list for its high priority but also profitable, which ultimately will only push even further the interest to pursue this work area. Moreover, enterprises are investing loads of money in the development of new technologies and software that improves the already established state of the art and proposes new ways to fulfill the task following the paradigm of deep reinforcement learning.

In case of PyTorch, this library is built-in Anaconda and available for the use and the improvement to everybody. Thanks to it, data scientists, enthusiasts, teachers and students have found a powerful framework to work and experiment, as well as a peerless opportunity to delve into deep learning with a stable and ever expanding library.

# 5 Conclusions

All together, Deep Reinforcement Learning has proved itself to be one of the most groundbreaking innovations at this very moment. Meaning that, since its various applications have improved the self-learning capabilities of machines and artificial systems, the relevancy and long lasting presence of its utility are bound to be of great use in solving problems with a superb ability and precision that far surpasses that of an human being.

The results we have obtained from testing and carrying out several attempts to check the diverse algorithms presented in PyTorch for deep reinforcement learning have helped us to understand both how efficiently the library works under heavy workload and the close relationship with the human learning process that these algorithms share. All in all, it has been a satisfying experience and we have learnt plenty from it.

As a final note, we have been involved in a research process so as to gather information and knowledge about deep reinforcement learning, which has, as a result of exploring the vast sea of content that the Internet provides, illustrated us the hardships of selecting information carefully and from reliable resources.

## References

[1] Lilian Weng. Policy gradient algorithms. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

[2] viskit. https://github.com/vitchyr/viskit.

[3] Ankit Choudhary. A hands-on introduction to deep q-learning using openai gym in python. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.