Policy-based Deep Reinforcement Learning

Jonathan Collu, Dimitrios Ieronymakis, Riccardo Majellaro

Abstract

In the scope of our assignment, we investigate different reinforcement learning $policy - based$ methods to solve OpenAI's CartPole V1 challenge, where the goal of an agent is to learn how to balance a pole on a horizontally moving cart, by sliding the cart itself left or right. The best configurations, which are able to obtain near-perfect results, are additionally analyzed in combination with Evolutionary Algorithms and a Quantum Neural Network.

1. Introduction

This report aims to investigate and compare different algorithms belonging to the categories of policy-based reinforcement learning methods. Therefore, we implemented and experimented with REINFORCE and ActorCritic, analyzing their training processes and evaluating the performances post-training. The environment chosen to test these techniques is CartPole V1 (OpenAI). The evaluation results we obtained suggest that all the algorithms investigated can achieve optimal or near-optimal results. Moreover, we also examined evolutionary strategies as an alternative optimization algorithm, obtaining more than satisfactory results. Finally, an effort was made to study a Quantum Neural Network used as a policy-network, which ended achieving poor performances.
The document will therefore proceed with a synopsis of the theoretical and mathematical background required to understand the experiments carried out and discussed in the following sections.

2. Theoretical Background

In order to better introduce the reader to the methodologies presented in the subsequent sections, it could be helpful to summarize the fundamental concepts behind them. As already mentioned in the introduction, we implemented and investigated different policy-based algorithms. For this reason, this section contains a brief recall of some basic reinforcement learning concepts. Furthermore, we experimented with a Quantum neural network (QNN) used as a policy network, thus is crucial to introduce some Quantum Computing principles to contextualize what a quantum algorithm is and how it can be used as a trainable layer for a QNN (with QNN we indicate a hybrid neural network composed by a combination of classical and quantum layers). Given the complexity of the argument and considering the scope of this document, these concepts will not be addressed in a very exhaustive way.

2.1. Reinforcement Learning

The main goal of a reinforcement learning algorithm is for an agent to learn the optimal behaviour in an environment, by maximizing the obtained rewards. The strategy that the agent uses to select an action is called a $policy$, which can be stochastic or deterministic. In general, reinforcement learning methodologies can be categorized into two main groups, namely $value$ and $policy$ based approaches. At the intersection between the two, lies a subset of algorithms called $Actor - Critic$. In value-based RL we find algorithms like $SARSA$ and $Q - learning$, and $Deep\ Q - learning$, which try to learn the state-action values in a specific state using different strategies. In policy-based RL methods, like REINFORCE or evolutionary policy search, the algorithm tries to learn directly the policy function that maps each state to a probability distribution over the actions. Finally, we have ActorCritic methods, which combine both policy and value approaches.

2.2. Evolutionary Algorithms

Evolutionary Algorithms (EA) are robust and flexible optimization mechanisms inspired by biological evolution. The main idea behind these algorithms is that a $population$ of individuals, representing the solutions to our optimization problem, is modified and evaluated at every iteration of the algorithm. The evaluation function assigns a $fitness$ to each individual of the population, representing how to fit (or simply how good) a specific solution for the optimization problem is. After this step, the individuals (or part of) are used to generate the $offspring$ population via $recombination$, which is, therefore, $mutated$, and then the best are $selected$ among them and the parents. These steps are then repeated until the end of a predefined budget. The main steps for an evolutionary strategy are thus $Recombination$, $Mutation$, $Selection$ and $Evaluation$. For more detailed information on each EA step please refer to the original publication (Bartz-Beielstein et al., 2014).

2.3. Quantum Computing

In order to clearly describe quantum computation, it is useful to introduce the quantum computation (QC) basic elements by using their classical counterparts. The most basic QC element is the Quantum Bit (qubit) that corresponds to the classical bit and thus represents the QC fundamental information unit. Differently from the bit that represents a classical state (that can just be one between 0 and 1), a qubit represents a quantum state. The first postulate of quantum mechanics establishes that any physical state (thus also a quantum state) is described by a normalized vector in a complex vector space $H = \mathbb{C}^n$. Then a qubit represents a quantum state described by a vector in $\mathbb{C}^2$ and thus it can be in a basis state ($|0\rangle = [1, 0]$ or $|1\rangle = [0, 1]$) and in any linear combination ($\alpha |0\rangle + \beta |1\rangle$ with $\alpha, \beta \in \mathbb{C}$, $|\alpha|^2 + |\beta|^2 = 1$) that means that a qubit can be in a new state called superposition of states (a new option respect to 0 and 1). A second important element of QC concerns operations. In classical computation, operations are performed by logical gates (not, and, or, nand, nor, xor, xnor) while QC uses quantum gates. The second postulate of quantum mechanics says that the evolution of a quantum system from an instant t_0 to an instant t_1 is fully described by a unitary transformation ($|\Psi_{t_1}\rangle = U |\Psi_{t_0}\rangle$) and then a quantum gate is completely described by a unitary matrix (operations are reversible). Finally, the last element of QC that is covered in this section is the measurement (there are other elements but the understanding of the ones explained is enough for our scopes). In classical computation, the output of a digital circuit is deterministic, while in QC the output of a quantum circuit is stochastic because of superposition. Thus, given a certain qubits initialization and a certain quantum circuit, the measurement outcome can be different over repetitions of the same computation. The third postulate of quantum mechanics states that possible measurements are specified by a hermitian operator (observable) and measurement outcomes correspond to its eigenvalues (this should not surprise since the eigenvalues of a hermitian matrix are real numbers). This is a very simplified description of QC. For a more precise and detailed view, we recommend reading (Chuang, 2010)

3. Methodology

The focus of this report is the application of policy-based reinforcement learning to tackle the OpenAI CartPole V1 challenge. Therefore, we implemented and analyzed two famous algorithms, namely REINFORCE and ActorCritic, both in their naive formulation and along with baseline subtraction and entropy regularization techniques.
In general, reinforcement learning algorithms aim to maximize the expected cumulative reward obtained by an agent while following a policy from the starting state. Mathematically, the objective can be defined as $\text{argmax}_\theta\ J(\theta)$, with $J(\theta) = \mathbb{E}_{x\sim p_\theta(x)}[R(x)]$ and θ representing the policy network parameters. $R(x)$ indicates the reward yielded by the trace x. To solve this optimization problem we can move within the search space using a gradient ascent approach: $\theta \leftarrow \theta + \eta \cdot \nabla_\theta J(\theta)$, with η being the learning rate and $\nabla_\theta J(\theta)$ the gradient of J with respect to the weights θ. After few simple steps and the application of the log derivative trick, the gradient of the expected value can be written as $\mathbb{E}_{x\sim p_\theta(x)}[R(x) \cdot \nabla_\theta \log p_\theta(x)]$. By the definition of $p_\theta(x)$, the gradient of the trace probability can be expanded as:

$$\nabla_\theta \log[p_0(s_0) \cdot \prod_{t=0}^{n} \pi_\theta(a_t|s_t) \cdot T(s_{t+1}|s_t, a_t)] =$$

$$\nabla_\theta[\log p_0(s_0) + \sum_{t=0}^{n} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{n} \log T(s_{t+1}|s_t, a_t)]$$

where p_0 is the probability of the starting state s_0 of the trace x, a_t represents the action selected in the state s_t at the timestep, $t = 1, \ldots, n$, while T indicates the transition function. π_θ is the differentiable policy function determined by the weights θ. Given that $p_0(s_0)$ and $T(s_{t+1}|s_t, a_t)$ do not depend on θ, the derivatives w.r.t. θ of these terms are equal to zero. Therefore,

$$\nabla_\theta \log p_\theta(x) = \sum_{t=0}^{n} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Finally,

$$\nabla_\theta \mathbb{E}_{x\sim p_\theta(x)}[R(x)] = \mathbb{E}_{x\sim p_\theta(x)}[\sum_{t=0}^{n} R(x_t) \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

$R(x_t)$ is the reward obtained starting from the timestep t of the trace x. The policy function π_θ is defined as a simple multi-layer perceptron (MLP) having 2 hidden layers, one of 64 nodes and one of 8, transformed by a ReLU function. The output is composed of 2 neurons followed by a softmax in order to obtain a probability distribution over the 2 actions. As input, it takes the 4 state values. We use this same policy function for all the experiments, except for the quantum ones.

3.1. REINFORCE

REINFORCE is a Monte Carlo algorithm that samples M full episodes following the policy network π_θ and iterates over them to compute the gradient. For each trace we compute the discounted cumulative future reward R_t at every timestep t, and use it similarly to the last equation: $\sum_{t=0}^{n} -R_t \cdot \log \pi_\theta(a_t|s_t)$. The minus is introduced to transform the earlier described maximization problem into a minimization one. We average these terms over the M episodes, therefore we assume a uniform probability $1/M$ over the trajectories. Finally, the averaged value is used as the loss function of π_θ, thus the gradient is computed and its weights are updated following a given optimization algorithm.

3.2. ActorCritic with bootstrapping

ActorCritic takes its name from the two main components of the algorithm, the actor-network (policy function) π_θ and the critic network (value function) V_ϕ. The former has already been presented in this section, while the latter differs from it only by the output layer, which is composed of a single node activated by the ReLU. The choice of this activation function is justified by the fact that the expected cumulative reward for this problem is non-negative. Similar to REINFORCE, in this algorithm we sample M traces and iterate over them to compute the policy and value losses. An important difference between the two is the bootstrapping technique: instead of computing the full expected reward at a timestep t, only n future non-discounted rewards are considered and added to the approximated state value (using V_ϕ) at the $(n+1)$-th step. We can hence obtain the state-action values Q_t and state values V_t correspondent to every timestep t of the episodes. The value network loss is computed as a squared error between the predicted Q_t and V_t at a timestep t, summed up over every step of all the sampled episodes, and then averaged over M as in REINFORCE. The policy network loss is again defined similarly to REINFORCE, but in this case the bootstrapped state-action values are used instead of the discounted rewards R_t, therefore for a given episode we have $\sum_{t=0}^{n} -Q_t \cdot \log \pi_\theta(a_t|s_t)$.

3.3. ActorCritic with baseline subtraction

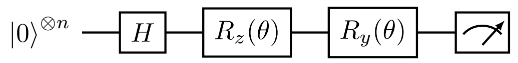
One problem with the previously presented algorithms is that the policy gradient method can increase the probabilities for actions that resulted in rewards under the state average. In this case, the problem is caused by the fact that all the rewards are positive, thus also the probability of the worst action between the two is pushed up. We solve this by using a baseline subtraction, which reduced the variance of the gradient estimate without causing additional bias. In this way, we increase the probabilities of actions that returned above average results and decrease the probabilities of the actions that did not. This mechanism can be described with the equation $A_t = R_t - V_t$ in the REINFORCE algorithm, and $A_t = Q_t - V_t$ in the ActorCritic algorithm, where A_t represents the "advantage", which is therefore a measure of how better an action at timestep t is than the state average V_t. In the case of REINFORCE, we need to introduce the same value network V_ϕ to estimate V_t. For this reason, REINFORCE with baseline subtraction can be considered an ActorCritic algorithm.

3.4. Evolutionary Algorithms

To demonstrate the flexibility of EA, three approaches have been created and applied successfully in the CartPole environment. The first and most naive approach consists in estimating two sets of weights with EA, one for each action. Each set of weights is then multiplied with the observations (as a dot product) in order to get two values. The action selection then simply consists in taking the $argmax$ of the two resulting values. Since this experiment would not satisfy our curiosity, in the second and third approaches, EA is used in unison with the policy-based algorithms. The main idea behind the combined methodology is to periodically activate the EA to optimize the value and policy network weights. In the second approach, which we will denote with es_0 we only update the value network with EA, while, in the last approach (es_1) we first optimize the value network and then tho policy network. As for the EA parameters, $Discerte$ recombination, $Intermediate$ mutation, and (μ, λ) selection were used, together with a greedy action selection policy. The objective of our evolutionary strategy is the maximization of the reward; in the first approach, the reward corresponds to the one of a single episode, while in the other two it coincides with the mean reward of the sampled episodes.

3.5. Quantum Policy Network

In order to turn the MLP described in the sections above into a QNN, it was necessary to build a Parametrized Quantum Circuit (a quantum circuit where some of the unitary transformations and then the outcome are influenced by trainable parameters) to be used as a layer for our classical network. The quantum circuit has been built as follows:



where n is the number of qubits involved in the circuit, the first gate (denoted with H) is the Hadamard gate that corresponds to the unitary transformation

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$

and is used to put the qubit in superposition (it can be easily checked that $H\,|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$). The rotation gates R_z and R_y perform respectively a qubit rotation about the z and y axis and they represent the following unitary transformations.

$$R_z(\theta) = \begin{pmatrix}e^{-i\frac{\theta}{2}} & 0\\ 0 & e^{i\frac{\theta}{2}}\end{pmatrix}$$

$$R_y(\theta) = \begin{pmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}$$

REINFORCE	
Parameter	Value
policy-net-opt	Adam
learning rate	3e-3
M	5
T	500
epochs	500
γ	0.99
entropy reg	[False, True]
entropy factor	0.2

Table 1. Predetermined hyperparameters used for REINFORCE

ActorCritic with bootstrap	
Parameter	Value
policy-net-opt	Adam
learning rate	3e-3
value-net-opt	Adam
learning rate	3e-3
M	5
T	500
n	[50, 125, 250]
epochs	500
entropy reg	[False, True]
entropy factor	0.2

Table 2. Predetermined hyperparameters used for ActorCritic with bootstrap

Finally the last gate measures the qubit in the state $\alpha|0\rangle + \beta|1\rangle$ (with $\alpha, \beta \in \mathbb{C}$) producing the output $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Since the measurement is not reversible this is not a quantum gate.

In order to make the circuit described above a neural network layer trainable with PyTorch we used an adapted version of the "$HybridFunction$" described in the dedicated Qiskt tutorial[1]

4. Experimental Setup

All the experiments were carried out over 3 repetitions. In addition, part of the parameters has been determined by prior experimentations, which have been omitted in order to make this document more concise. Constants like the learning rate of the model optimizers and the epoch number have been kept consistent to better compare the behavior and learning process of each configuration. These parameters can be observed in Tables 1, 2, 3, and 4.

[1]https://qiskit.org/textbook/ch-machine-learning/machine-learning-qiskit-pytorch.html

ActorCritic with baseline subtraction	
Parameter	Value
policy-net-opt	Adam
learning rate	3e-3
value-net-opt	Adam
learning rate	3e-3
M	5
T	500
epochs	500
γ	0.99
entropy reg	[False, True]
entropy factor	0.2

Table 3. Predetermined hyperparameters used for the experiments of the ActorCritic with baseline subtraction

ActorCritic with bootstrap and baseline subtraction	
Parameter	Value
policy-net-opt	Adam
learning rate	3e-3
value-net-opt	Adam
learning rate	3e-3
M	5
T	500
n	250
epochs	500
entropy reg	[False, True]
entropy factor	0.2

Table 4. Predetermined hyperparameters used for ActorCritic with bootstrap and baseline subtraction

Evaluations on the CartPole V1			
Experiment	Mean	Deviation	Episode
REINFORCE	500	0	222
AC b. sub.	480.46	18.99	370
AC bootstrap	500	0	307
AC bootstrap b. sub.	500	0	164
AC b. sub. es	500	0	299
AC bootstrap es	500	0	109
AC b. sub. es q	19.50	10.04	64

Table 5. Results of the evaluations of the best configuration for REINFORCE, ActorCritic with baseline subtraction (AC b. sub.), ActorCritic with bootstrapping (AC bootstrap), ActorCritic with bootstrapping and baseline subtraction (AC bootstrap b. sub.), ActorCritic with baseline subtraction and evolutionary strategies (AC b. sub. es), ActorCritic with bootstrapping and evolutionary strategies (AC bootstrap es), ActorCritic with baseline subtraction and evolutionary strategies quantum(AC b. sub. es q)

5. Results

A wide variety of experiments has been carried out and all the results have been summarized with a table and figures, thoroughly described in this section. Table 5 summarizes the results achieved by our best configurations during the evaluations. The Mean column indicates the average, between the three repetitions of a configuration, of the number of mean rewards obtained over 100 evaluations. The Deviation column, on the other hand, represents the standard deviation computed in the same way as the Mean. Finally, Episode, as suggested by the name, represents the episode number on which the best agent was found during the training phase. In order to do this, we evaluate 25 times each agent achieving 500 mean rewards in the sampled traces.
By observing the table, we also notice that all the algorithms investigated produce optimal (or near-optimal) results, except for the one tested with a quantum neural network as a policy network.

Looking now at graphs from Figures 1 to 6, we can observe the training processes of several configurations. Figure 1 illustrates the behavior during the training process of REINFORCE, with and without entropy regularization, and ActorCritic with no bootstrapping, which uses baseline subtraction coupled and uncoupled with entropy regularization. When entropy regularization is not used the training is evidently more stable, while the presence of the critic network leads to faster initial improvements. This behavior is indeed present in the red curve, while the baseline subtraction seems to cause a more fluctuating trend.

In Figure 2 we can observe how ActorCritic with bootstrapping performs with different values of the estimation depth. It is evident, by looking at the plot, that the algorithm achieves results close to the optimum with $n = 250$ outper-

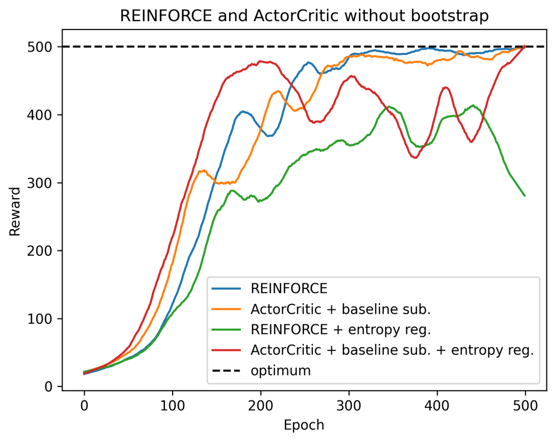


Figure 1. Comparison between the training processes of plain REINFORCE, ActorCritic without boostrap along with baseline subtraction, and their version with entropy regularization using an entropy factor equal to 0.2. On the x axis are indicated the epochs, while on the y axis are indicated the cumulative rewards.

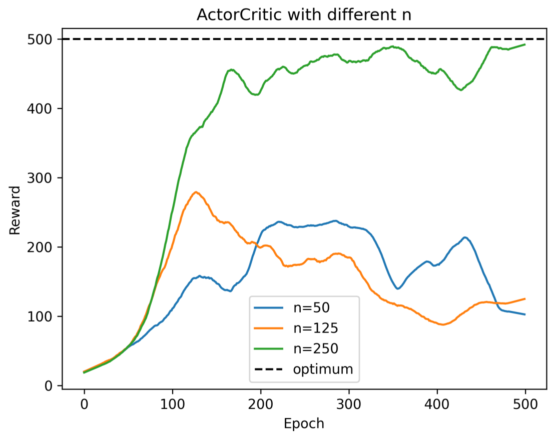


Figure 2. Comparison between the training processes of ActorCritic with bootstrap with different values of n which indicates the estimation depth. On the x axis are indicated the epochs, while on the y axis are indicated the cumulative rewards. The graph shows that an estimation value n = 250 performs better than the other configurations.

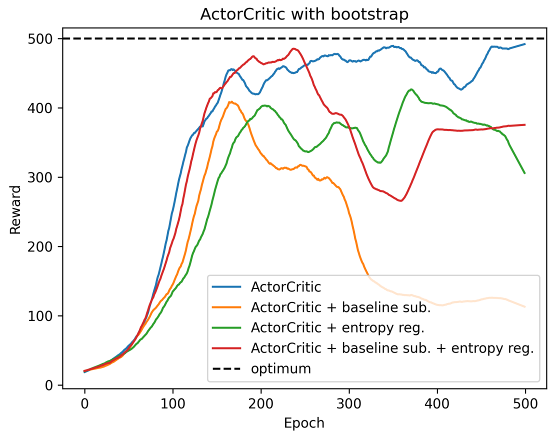


Figure 3. Comparison between the training processes of ActorCritic with bootstrapping ($n = 250$), its combination with baseline subtraction, and their version with entropy regularization using an entropy factor equal to 0.2. On the x axis are indicated the epochs, while on the y axis are indicated the cumulative rewards.

forming the settings with $n = [50, 125]$ causing it to obtain lower results.

Figure 3 shows a comparison between the training process of ActorCritic with bootstrapping and its combination with baseline subtraction and entropy regularization. The blue line is the fastest growing among the four, and with the most stable convergence. The remaining three tend to degrade in the second half of the training process, with the worst-performing one being the orange curve.

In Figure 4 the training trends of ActorCritic without bootstrap along with baseline subtraction are shown, both in its plain version and with 2 different evolutionary strategy optimizations. The graph highlights that both the evolutionary strategy optimizations make the algorithm faster in achieving high cumulative rewards, with the es_0 improving the already good performances obtained with its plain version. On the contrary, the es_1 presents a more fluctuating convergence than the naive setting.

In Figure 5 the training processes of ActorCritic with bootstrapping are shown, both in its plain version and with the 2 different evolutionary strategy optimizations. In this case, both the evolutionary strategy optimizations deteriorate the performances obtained by the plain version of the algorithm. It is important to highlight that es_1 makes the learning process unstable, while es_0 achieves its worst performance with a steadily decreasing curve after reaching its peak.

As it is possible to observe from the plot in Figure 6, the performance achieved by replacing the policy network with a quantum neural network is very poor, and the motivations

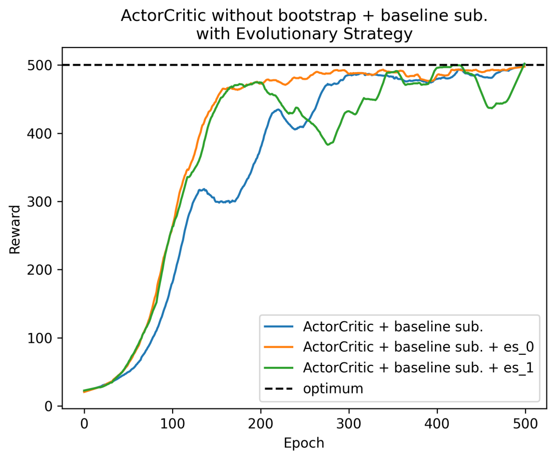


Figure 4. Comparison between the training processes of ActorCritic without bootstrap, along with baseline subtraction coupled with two different evolutionary strategy optimizations, namely es_0 and es_1 introduced in section 3.4 . On the x axis of the plot, the epochs are indicated, while on the y axis we find the cumulative rewards.

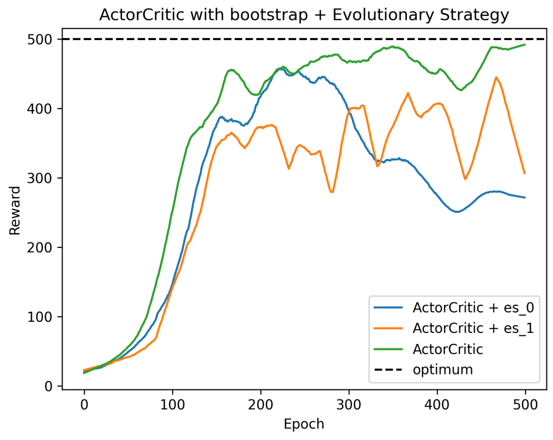


Figure 5. Comparison between the training processes of ActorCritic with bootstrapping in its plain version and with 2 different evolutionary strategy optimizations. On the x axis are indicated the epochs, while on the y axis are indicated the cumulative rewards.

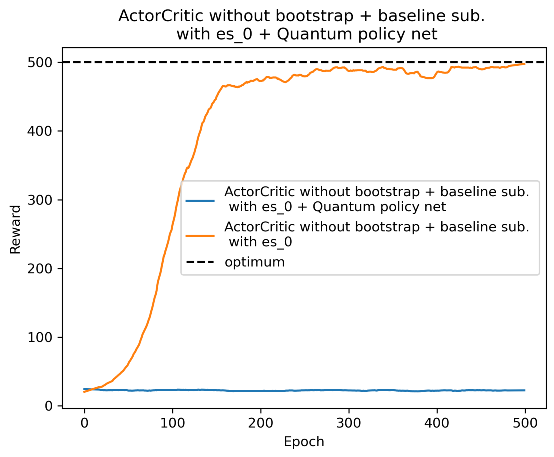


Figure 6. The training processes of the best ActorCritic without bootstrap along with baseline subtraction setting obtained so far using a Quantum neural network. On the x axis are indicated the epochs, while on the y axis are indicated the cumulative rewards.

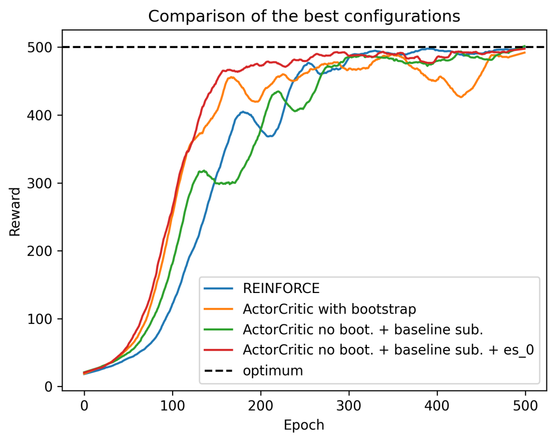


Figure 7. Comparison of the best configurations presented. On the x axis are indicated the epochs, while on the y axis are indicated the cumulative rewards.

must be properly investigated in a dedicated project.

Finally, in Figure 7 the best configurations of all the experiments have been presented for an easier conclusive comparison.

6. Discussion

A series of significant observations are reported in this section.
Firstly, since REINFORCE considers full traces of positive rewards, the estimate of the rewards can suffer from high variance. Indeed, in Figure 7 is clear that REINFORCE has a slightly slower ascent than ActorCritic with bootstrapping, which can be caused by this reason. However, as shown, the two algorithms still obtain comparable results.
Secondly, although using bootstrapping within the ActorCritic algorithm should, in theory, reduce the variance of the gradient estimates, this property is poorly reflected in the obtained results. In fact, increasing n brings more stability to the training process (Figure 2): the reason could be that the algorithm may rely less on the state value estimate of the additional value network (which is a possible cause of instability in the training) and more on the sampled rewards. ActorCritic without bootstrap, when coupled with baseline subtraction, increase the initial steepness of the curve compared to REINFORCE, which could be related to the decrease in the variance of the policy gradients. On the other hand, when baseline subtraction is paired with bootstrapping, the performance degrades critically. In fact, after an initial peak, the agent seems to forget the previously learned effective policies.

Some interesting considerations can also be made for the experiments where the policy-based methods were used in conjunction with the evolutionary strategies. Specifically, the configuration es_0 presented in figure 4 represents the most ideal curve convergence out of all our experiments. As it seems, periodically using an evolutionary algorithm to refine the value network adds a lot of stability to the training process of the ActorCritic without bootstrap configuration. The same cannot be said for the ActorCritic with bootstrap configurations, where a noticeable deterioration in the stability of the training can be observed.
Finally, concerning the results obtained with the quantum network, the poor performance achieved can be justified by the fact that this kind of architecture can actually perform reasonably only on very simple classification problems. Thus, solving a problem such as Cartpole V1 could be too much difficult to be addressed with a quantum network, and probably the Quantum AI field must be explored more deeply in the next years to be able to produce models robust enough to be used in more challenging tasks.

7. Conclusions

In conclusion, we investigated two policy-based algorithms with different settings, showing that all of these managed to solve the Cartpole V1 problem with extremely positive results. Moreover, results showed that entropy regularization does not lead to benefits in the learning process for our specific problem, while evolutionary strategies can be as competitive as gradient-based optimization methods, if not better. Moreover, for EA, it would be interesting to investigate more deeply the motivations behind the growth and deterioration of the performances with REINFORCE and ActorCritic. Finally, the experiments carried out using a quantum network showed that is still a very complicated challenge to properly train this kind of model in a reinforcement learning setting. In future works would be interesting to experiment with these algorithms in different and more challenging environments to have a more detailed insight into their strengths and weaknesses. Moreover, it could be intriguing to implement the quantum network as the value network instead of the policy network to study its impact on the learning process.

References

Bartz-Beielstein, T., Branke, J., Mehnen, J., and Mersmann, O. Evolutionary algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4, 05 2014. doi: 10.1002/widm.1124.

Chuang, M. A. N. . I. L. (ed.). *Quantum Computation and Quantum Information*. Cambridge University Press, 10th Anniversary Edition, 2010.

OpenAI. Cartpole-v1. `https://gym.openai.com/envs/CartPole-v1/`.