Control of a Two-Level Quantum System using Deep Reinforcement Learning

Fabrice Lauri FABRICE.LAURI@UTBM.FR
CIAD UMR 7533, Univ. Bourgogne Franche-Comté, UTBM, F-90010 Belfort, France

Stéphane Guérin STEPHANE.GUERIN@U-BOURGOGNE.FR
LIPC

Dominique Sugny DOMINIQUE.SUGNY@U-BOURGOGNE.FR
LIPC

Editor: Editor's name

Abstract

Reinforcement Learning (RL) tackles with decision problems where a sequence of actions has to be found by trial and errors by using learning in order to fulfil efficiently a given task. Deep RL is the use of Neural Networks with Reinforcement Learning techniques to deal with very complex decision problems, like winning in chess, in Go, or controlling systems with expert knowledge that would require decades to be acquired by humans. We apply in this article several Deep Reinforcement Learning techniques to the control of a two-level quantum system. We show experimentally that Reinforce and A3C are the most efficient techniques for controlling such systems.

Keywords: Quantum System; Reinforcement Learning

1. The model system

We consider the control of a two-level quantum system by means of an external electromagnetic field. The state of the system is $\psi \in \mathbb{C}^2$ of coordinates $(c_1, c_2)^\intercal$. The norm of ψ is equal to 1. The dynamics are governed by the following differential system:

$$i\dot{\psi} = H\psi$$

where H is the Hermitian Hamiltonian matrix which can be expressed in a canonical basis as:

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

where $\Delta \in \mathbb{R}$ is the offset term and $u \in \mathbb{R}$ the control command. The goal of the control is to go from $\psi_0 = (1,0)^\intercal$ to $\psi_f = (0,1)^\intercal$ (up to a phase factor) in a given control time t_f. The efficiency of the control process can be measured by the fidelity $F = 1 - ||\psi_f - \psi(t_f)||^2$, which is equal to 1 if the target state is reached.

This control problem can be reformulated in real coordinates by introducing the following change of coordinates:

$$\begin{cases} x = c_1 c_2^* + c_1^* c_2 \\ y = -i(c_1 c_2^* - c_1^* c_2) \\ z = |c_1|^2 - |c_2|^2 \end{cases}$$

with the constraint $x^2+y^2+z^2=1$. This sphere is called the Bloch sphere. The dynamical system can be expressed as:

$$\begin{cases} \dot{x} = -\Delta y \\ \dot{y} = \Delta x - uz \\ \dot{z} = uy \end{cases}$$

The goal is now to go from the north pole ($z=1$) to the south pole ($z=-1$) of the sphere and the fidelity to maximize is given by $F=-z(t_f)$. We add the following pulse limitation to the control problem: $|u(t)| \leq 1$.

1.1. Integration of the differential system

The dynamical system on the Bloch sphere can be exactly integrated for a bang control for which $u=\pm 1=\varepsilon$. We assume that the initial point is (x_0,y_0,z_0) at $t=0$. The goal is to compute the state of the system at time t. We have:

$$\ddot{y} = \Delta\dot{x} - \varepsilon\dot{z} = -(1+\Delta^2)y.$$

This leads to:

$$\ddot{y} + \Omega^2 y = 0,$$

with $\Omega=\sqrt{1+\Delta^2}$. We deduce that:

$$y(t) = A\cos(\Omega t) + B\sin(\Omega t).$$

Since $y(0)=y_0$, we have $A=y_0$. With $\dot{y}(0)=\Delta x_0-\varepsilon z_0$, we deduce that:

$$B = \frac{\Delta x_0 - \varepsilon z_0}{\Omega}$$

For the x- and z- coordinates, we have:

$$\begin{cases} x(t) = x_0 - \frac{\Delta B}{\Omega} - \frac{\Delta}{\Omega}(A\sin(\Omega t) - B\cos(\Omega t)) \\ z(t) = z_0 + \frac{\varepsilon B}{\Omega} + \frac{\varepsilon}{\Omega}(A\sin(\Omega t) - B\cos(\Omega t)) \end{cases}$$

1.2. Optimal control theory

For a specific value of Δ, this problem can be solved explicitly by optimal control and the Pontryagin Maximum Principle. The minimum time t^* to solve the control problem can also be derived. For $\Delta \leq 1$, it can be shown that the optimal solution is the concatenation of two bang arcs of amplitude ± 1. As displayed in Fig. 1, the control sequence is characterized by two times t_1 and t_2 defined as:

$$\begin{cases} t_1 = \frac{1}{\Omega}(\pi - \arccos(\Delta^2)) \\ t_2 = \frac{1}{\Omega}(\pi + \arccos(\Delta^2)) \end{cases}$$

with $\Omega=\sqrt{1+\Delta^2}$ and $t^*=\frac{2\pi}{\Omega}$. Under the assumption that the north pole can be reached in N steps, which implies that the system is controlled every $\frac{t^*}{N}$ time step, an optimal trajectory can be obtained by choosing the control $u=-1$ (in red) during the time t_1, followed by the control $u=1$ (in blue) during the time t_2, or the contrary. For this reason there are two symmetric time-optimal solutions.

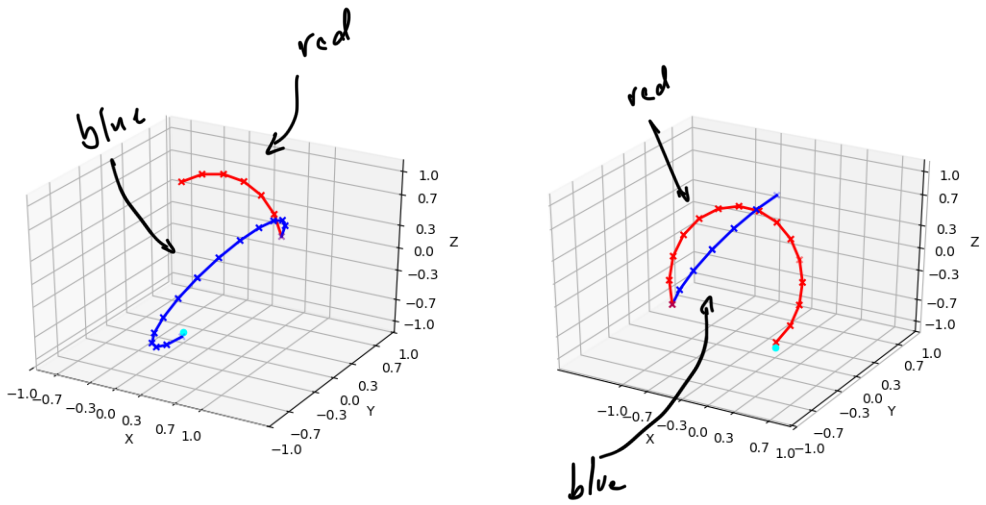


Figure 1: Optimal strategies for $N = 20$. Point in light cyan is the north pole.

2. Reinforcement Learning

The aim of this paper is to show that a solution can be obtained with state-of-the-art Reinforcement Learning (RL) approaches.

2.1. Classical Framework

Learning by Reinforcement[1] consists in learning what to do for maximising a sum of rewards. Knowing what to do implies learning a function that maps to each possible situation the best action to carry out. The search for the best actions is done by trials and errors, from the interactions of a controller with a process. Reinforcement Learning can solve Sequential Decision Problems, that is problems where a sequence of appropriate actions have to be found in order to accomplish a given task. In such problems, a controller (also called agent) interacts with a process (also called environment) using four types of signals. From a state signal and a reward signal, the controller think about an action to carry out and performs it by sending the appropriate signal to the process. The controller has to perform an action every time it receives a temporal signal from the process. Once the process receives the action signal, it evolves to a new state and send a new state signal and reward signal to the controller. This interaction between the process and the controller continues until the process reaches a terminal state (success or failure to accomplish the task).

A Markov Decision Process (MDP) can be generally used to model a task in sequential decision making problems. An MDP P is made up of a tuple (S, A, Tr, R, γ) where:

1. S is the set of possible states of the process that can be perceived by the controller
2. A is the set of actions that can be performed by the controller
3. Tr is the transition function between states
4. R is the reward function that concisely describes the task objective.

1. A good introduction about Reinforcement Learning can be found in Sutton and Barto (2018)

5. γ is the discount factor

By performing a sequence of actions from time t implies the controller receives as much as rewards than actions and computes the sum of discounted rewards R_t such that:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

where r_{t+1} is the reward the controller receives when applying its action at time t in the current state and reaching the next state, and T is the number of steps required to reach a terminal state.

Solving an MDP implies to determine an *action policy*, that is a function π mapping actions from states, which maximizes the expected sum of discounted rewards $V^\pi(s)$ in every state s, or $Q^\pi(s,a)$ in every state s and for every action a, where:

$$V^\pi(s) = E_\pi\{R_t | s_t = s\}$$

and

$$Q^\pi(s,a) = E_\pi\{R_t | s_t = s, a_t = a\}$$

with $0 \leq \gamma < 1$ and $T = \infty$ or $\gamma = 1$ and T is finite. E_π is the mathematical expectation when the policy π is used.

The optimal policies, denoted π^*, share the same value functions:

$$V^*(s) = max_\pi V^\pi(s)\ \forall s \in S$$

or

$$Q^*(s,a) = max_\pi Q^\pi(s,a)\ \forall s \in S\ \forall a \in A$$

Once Q^* is determined for every state and every action, the greedy policy (which is optimal) is obtained by taking the action that gathers the greatest amount of rewards:

$$\pi(s) \in argmax_a\ Q^*(s,a)$$

RL algorithms have been used successfully in Robotics, Control, Games and Operations Research, to name a few. Simple RL algorithms, like Q-iteration or Q-learning by Watkins and Dayan (1992), iteratively update the Q-values of state-action pairs until convergence. In these algorithms the Q-function is implemented as a table with states as rows and actions as columns. But these algorithms are very sensitive to the number of states and actions (this is the space complexity) and to the number of iterations required to learn the Q-function (sample complexity). Most interesting problems are too large to learn all action values in all states separately. Instead, we can learn a parameterized value function $Q(s,a;\theta_t)$, where θ_t is the vector of parameters at time t. Simple RL techniques are practically limited to processes consisting of a maximum of 100.000 states, 10 actions and 5.000.000 iterations. Applying RL to large (or continuous) state and/or action spaces remains challenging, as for such spaces one requires to find good approximate solutions.

2.2. Control of the quantum system

To learn to control the quantum system, we can define the following MDP:

- S is the state space made up of all the points of the Bloch sphere.
- $A = \{-1, 1\}$ are the two controls that can be performed in every state.
- Tr defines the transition function that computes the next point on the Bloch sphere from the action performed by the controller at the current point.

3. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is the use of deep neural networks (neural networks with several hidden layers) in Reinforcement Learning algorithms in order to tackle Sequential Decision Problems. DRL was popularized by the paper Mnih et al. (2013) which presented how artificial players were capable to learn how to play some Atari games and be better than humans in several of them.

In DRL, methods are classified into two main families according to the use of the neural network (see Fig 2). In value-based networks, the neural network takes the states as inputs and outputs the Q-value $Q(s, a)$ for each possible action a. In policy-based networks, the alternative is to output the probability distribution $\pi(a|s)$ over the m mutually exclusive actions. So in the latter case, the network outputs directly the action policy. In other words, m numbers representing the probability of taking each action in the given state are returned.

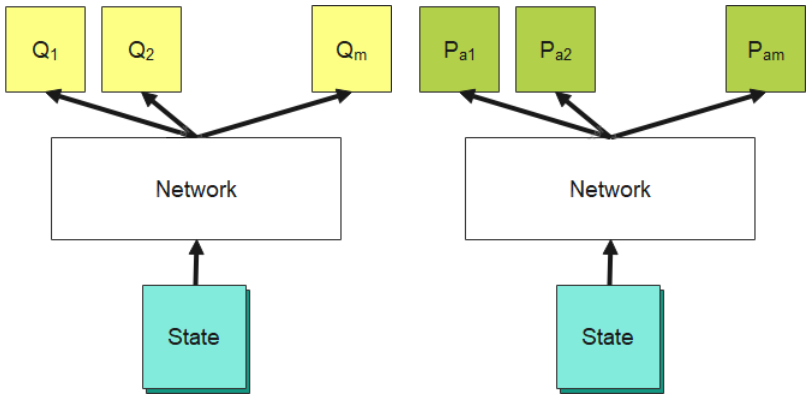


Figure 2: Value-based and policy-based neural networks in DRL.

3.1. Value-based methods

3.1.1. Deep Q-Network (DQN)

A deep Q network (DQN) is a multi-layered neural network that for a given state s outputs a vector of action values $Q(s, .; \theta)$, where θ are the parameters of the network. For an

n-dimensional state space and an action space containing m actions, the neural network is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$.

The standard Q-learning update for the parameters after taking action a_t in state s_t and observing the immediate reward r_{t+1} and resulting state s_{t+1} is then:

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(s_t, a_t; \theta_t))\nabla_{\theta_t} Q(s_t, a_t; \theta_t)$$

where α is a scalar step size and the target Y_t^Q is defined as:

$$Y_t^Q = r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_t)$$

3.1.2. Double DQN

Two important ingredients of the DQN algorithm have been proposed in Mnih (2015) and are the use of a target network, and the use of experience replay. The target network, with parameters θ^-, is the same as the online network except that its parameters are copied every τ steps from the online network, so that $\theta_t^- = \theta_t$, and kept fixed on all other steps. The target used by DQN is then:

$$Y_t^{DQN} = r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta_t^-)$$

Using experience replay Lin (1992) implies storing observed transitions for some time and sampling uniformly from this memory bank to update the network. Both the target network and the experience replay dramatically improve the performance of the algorithm.

The max operator in standard Q-learning and DQN uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. To prevent this, Van Hasselt et al. (2010) decouple the selection from the evaluation. This is the idea behind Double DQN. In the original Double Q-learning algorithm, two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights, θ and $\theta^{'}$. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. The selection and evaluation in Q-learning are first untangled and rewriten as:

$$Y_t^Q = r_{t+1} + \gamma Q(s_{t+1}, argmax_a\, Q(s_{t+1}, a; \theta_t); \theta_t)$$

The Double Q-learning error can then be written as:

$$Y_t^{DoubleQ} = r_{t+1} + \gamma Q(s_{t+1}, argmax_a\, Q(s_{t+1}, a; \theta_t); \theta_t^{'})$$

3.1.3. Double Dueling DQN

The improvement of Dueling DQN to DQN was proposed in Wang et al. (2015). The core observation of this paper is that the Q-values $Q(s, a)$ that the deep network approximates can be divided into two quantities: the value of the state $V(s)$ and the advantage of actions in this state: $A(s, a)$. As before, $V(s)$ just equals to the discounted expected reward achievable from state s. The advantage $A(s, a)$ is just the delta from $V(s)$ to $Q(s, a)$, as by definition $Q(s, a) = V(s) + A(s, a)$. An explicit separation of the value and the advantage in the network's architecture brings better training stability, faster convergence and better results in practice.

3.2. Policy-Gradient Methods

In such methods, the policy gradient defines the direction in which the network's parameters need to be changed in order to improve the policy in terms of the accumulated total reward. This means that the probability of actions that gave good total reward will be increased and the probability of actions with bad final outcomes will be decreased. From a practical point of view, policy gradient methods perform minimization of a loss function L during stochastic gradient descent (SGD).

3.2.1. Reinforce

The idea behind Reinforce (Zhang et al. (2020)) is to play N full episodes with the current action policy represented by the neural network, saving their $(s, a, s^{'}, r^{'}$ transitions, calculating the discounted total reward for subsequent steps, calculating the loss function for all transitions, performing SGD update of weights to minimize the loss, and repeating until convergence. Compared to methods based on Q-learning, here no explicit exploration is needed to explore the environment as it is done directly with the action policy. No replay buffer is required. This implies faster convergence can usually be reached, but for much more interactions with the process.

3.2.2. Advantage Asynchronous Actor-Critic

This method, also called A3C, was proposed in Mnih et al. (2016) and is one of the most widely used by RL practitioners. It uses two networks: the policy network, called the actor, which returns a probability distribution over the actions and tells what to do, and the other network, the critic, measures how good the actions were and somehow evaluates the quality of the actions of the actor. This method consists in playing N steps in a set of K environments running in parallel, using the current policy π_θ and saving the state, action and reward in each environment. Policy gradients and value gradients are accumulated in the different environments and they are summed up to update the corresponding network's weights (the actor from policy gradients and the critic from the value gradients) and this is repeated until convergence.

4. Experimental results

We conducted several experiments according to two different goals. The first goal is to verify which of these RL techniques are able to solve this control problem, and for the ones which can, how efficiently. The second goal is to verify the robustness of the RL techniques, where Δ is unknown prior to the control step.

The reward function is defined as follows:

- a reward of 100 is given when a state near the north pole is reached. A state s is near the north pole f if $||s - f||^2 \leq \epsilon$, where the sensitivity ϵ was fixed to 0.2.
- no reward (reward of 0) is given when the control command is switched, that is alternating the control u from 1 to -1 or the contrary.
- otherwise a reward of 1 is given.

To accelerate the convergence of the DRL algorithms, the discount factor γ was set to 0.99. When using a learnt policy and computing the sum of rewards, like in the following figures, we set $gamma = 1$.

All the described algorithms, that is Double DQN, Double Dueling DQN, Reinforce, Policy-Gradient and A3C, were executed and evaluated for controlling the quantum system with $N \in \{10, 20, 50\}$ steps. Every control occurs each $\frac{t^*}{N}$ time steps. Several network's architectures and hyperparameters were tested for every algorithm, from 60 for Reinforce, Policy-Gradient and A3C, up to 150 for Double DQN and Double Dueling DQN. A3C was executed by using 50 independent environments, where in all of them the offset term $\Delta = -0.5$.

The following results present the optimal trajectory when found by a RL technique along with its loss curve and the mean rewards over 100 episodes. The loss is the squared error between the sum of rewards predicted by the neural network and the sum of rewards computed from the new reward after having visited a state. The loss measures how bad the model's prediction is on a single example. No results are presented for Double DQN and Double Dueling DQN because they were unable to find a solution irrespective of the complexity of the quantum system.

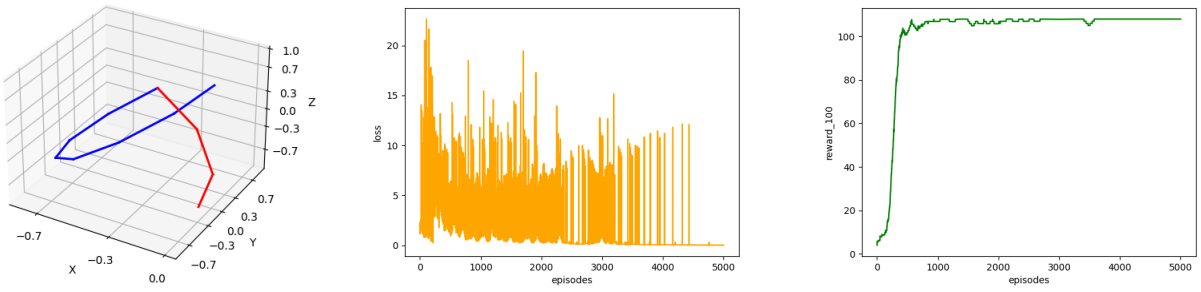
4.1. Experiments on efficiency

4.1.1. Controlling the quantum system with $N = 10$

A3C is the most efficient technique for controlling the system using a sequence of 10 actions. Reinforce, Policy-Gradient and A3C all found an optimal strategy with a sum of rewards of 108.

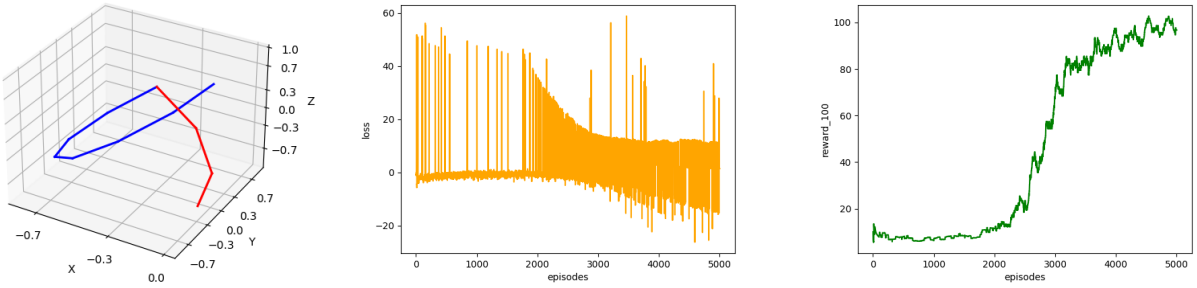
- Reinforce (4s)

  We can see from the loss curve that Reinforce makes the lowest losses compared to Policy-Gradient and A3C. Once an optimal strategy has been found for the first time after approximately 800 episodes, the learning in term of rewards is very stable.



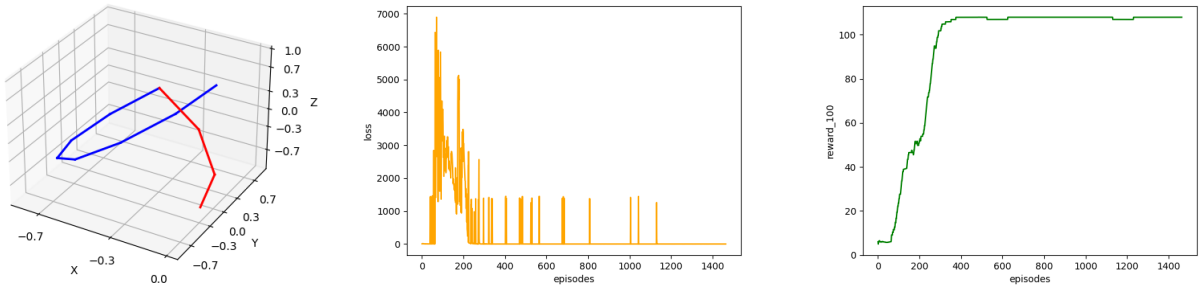
- Policy-Gradient (55s)

  The learning in Policy-Gradient is more erratic than the other methods, as we can see on the curve of the mean of the latest 100 rewards.



- A3C (3s)

  After the chaotic period of learning (coming from the numerous agents learning independently) we can see on the loss curve, A3C found an optimal strategy only after 400 episodes.

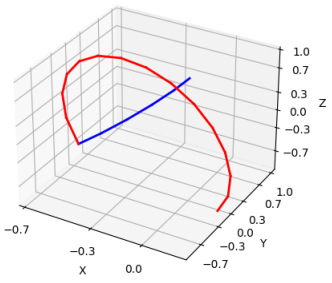


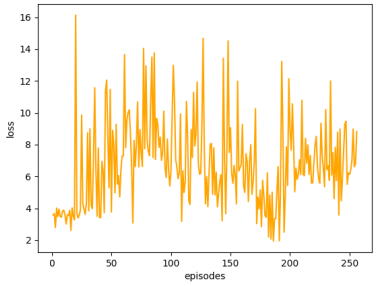
4.1.2. Controlling the quantum system with $N = 20$

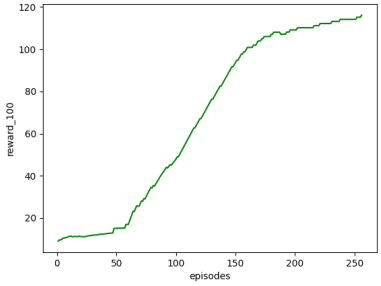
Reinforce is here the most efficient technique for controlling the system using a sequence of $N = 20$ actions. Reinforce, Policy-Gradient and A3C all found an optimal strategy with a sum of rewards of 116.

- Reinforce (7s)

  Reinforce has the most stable and fastest learning that constantly enhance the rewards. Finding an optimal solution only requires 250 episodes.

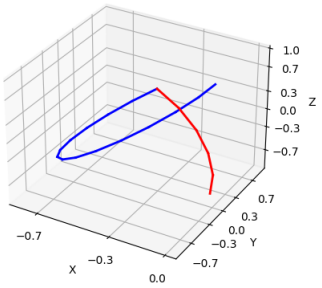


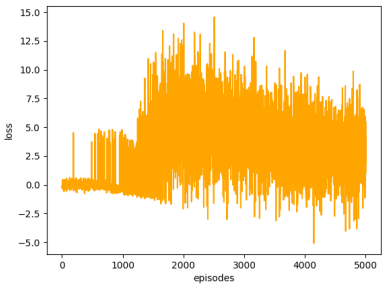


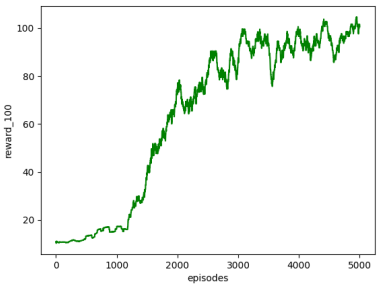


- Policy-Gradient (103s)

  As before Policy-Gradient is again the method that learns the most erratically and requires a lot of episodes to find an optimal strategy.







- A3C (22s)

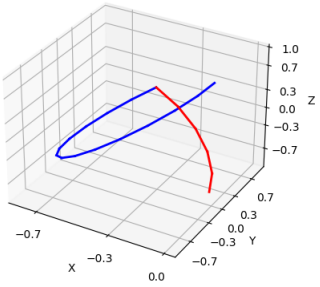
  Due to the independent learning of several agents, A3C has the most spread losses with the highest variance.

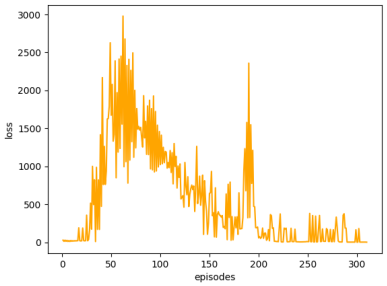
But it can find an optimal solution with as almost few episodes as Reinforce.

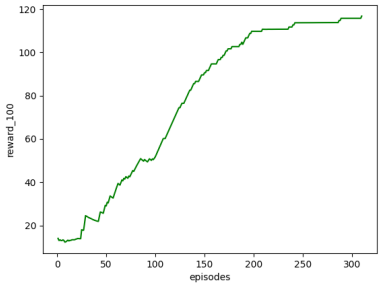
4.1.3. Controlling the quantum system with $N = 50$

A3C is here the most efficient technique for controlling the system using a sequence of $N = 50$ actions. Reinforce and A3C both found an optimal strategy with a sum of rewards of 146. Policy-Gradient was unable to find a solution.

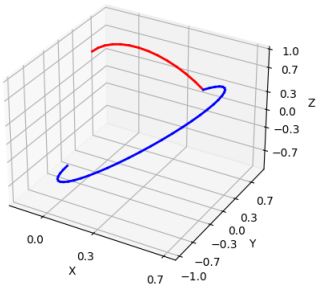
- Reinforce (45s)

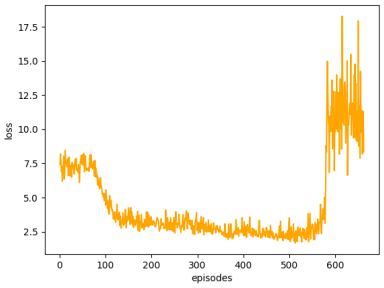


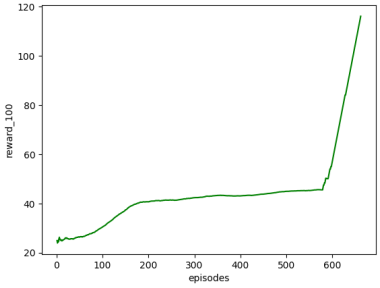




An optimal strategy were found in this case by Reinforce after a sequence of three types of learning. First a little enhancement occurs during less than 200 episodes to find strategies whose sum of rewards was stuck around 40. During 400 episodes, Reinforce was unable to find better strategies. Only after, some of them were able to be improved until finding optimal ones.





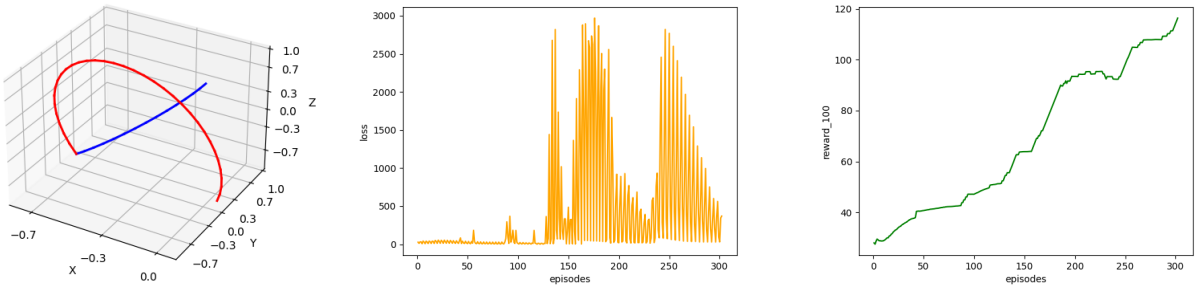


- A3C (32s)

  Due to the nature of A3C, that generates independent strategies, it has the largest variance in loss but we can see that strategies are continually improved until finding an optimal one after only 300 episodes.

4.1.4. Conclusion about the results on efficiency

An optimal control strategy has always been found for this two-level quantum system, for every given number of steps N required to go from the south pole (initial position) to the north pole (final position). As the tested DRL techniques are stochastic and due to the No Free Lunch theorem (Wolpert and Macready (1997)), we cannot explain the reasons why A3C are better than Reinforce in our experiments for some cases. However we strongly think that with some other hyperparameters (maybe with more independent agents), A3C would have been the best techniques irrespective of the complexity of the control problem



of the quantum system. So for all the following experiments on robustness, we decided to use A3C.

4.2. Experiments on Robustness

Experimenting the robustness consists in training a neural network that approximates the Q-value function under the assumption that the input represents the concatenation of the states of several two-level quantum systems that evolve in parallel according to different values of Δ. Here the aim is to find control strategies conisting of a sequence of $N = 40$ actions. In these experiments for robustness, we decided to apply the A3C algorithm to find optimal strategies as it gaves the overall best results for controlling a single quantum system. The A3C algorithm was executed with $K = 50$ independent environments where in each environment the aim is to control 3 quantum systems evolving in parallel, with $Delta = -0.5 - \sigma$, $\Delta = -0.5$ and $\Delta = -0.5 + \sigma$, where $\sigma \in [0.01, 0.02, 0.1]$ is chosen randomly when creating each of the K environments.

We tested 45 different architectures of neural networks for these experiments on robustness. The figures below present the control strategies found by the best of them consisting of a neural network made up of 1 hidden layer with 2048 neurons using the ReLU activation function.

5. Conclusion and Future Works

Learning by Deep Reinforcement consists in using a Neural Network to learn what to do in different situations in order to maximize a sum of rewards. We show in this paper that some DRL techniques can be applied for finding the optimal control of a two-level quantum system, irrespective of its complexity in terms of number of decisions. This study on the applicability of DRL techniques to discover an optimal control strategy in a quantum system can be enhanced in multiple ways. First we will conduct some experiments to measure at which sensitivity ϵ and number of steps N an optimal control can still be found by these DRL techniques for this two-level quantum system. Secondly we plan to apply and study these techniques for the control in more complex quantum systems, possibly where actions are also continuous, for instance when the control command $u \in [-1; 1]$.

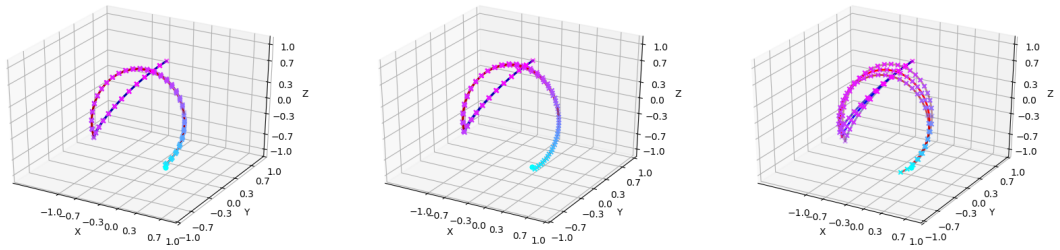


Figure 3: Control strategies found by a neural network learning in multiple different environments. Three quantum systems evolve in parallel. From left to right: $\sigma = 0.01, 0.02, 0.1$.

Delta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
-0,5	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,63	1,55	1,47	1,38	1,46	1,54	1,6	1,65	1,7	1,73	1,75	1,77	1,77	1,76	1,74	1,7	1,66	1,61	1,55	1,48	1,4	1,31	1,21	1,1	0,99	0,87	0,75	0,62	0,49	0,36	0,22
-0,51	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,63	1,55	1,47	1,38	1,46	1,54	1,6	1,65	1,69	1,73	1,75	1,76	1,76	1,75	1,73	1,69	1,65	1,6	1,53	1,46	1,38	1,29	1,19	1,08	0,97	0,85	0,73	0,6	0,47	0,33	0,2
-0,49	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,63	1,55	1,46	1,38	1,46	1,54	1,6	1,66	1,7	1,73	1,76	1,77	1,77	1,76	1,75	1,72	1,67	1,62	1,56	1,49	1,41	1,33	1,23	1,13	1,02	0,9	0,78	0,65	0,52	0,39	0,25

Delta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
-0,5	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,63	1,55	1,47	1,55	1,62	1,68	1,73	1,77	1,8	1,81	1,82	1,82	1,8	1,77	1,74	1,69	1,63	1,56	1,48	1,39	1,3	1,19	1,08	0,96	0,84	0,71	0,58	0,44	0,31	0,17
-0,52	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,63	1,55	1,47	1,55	1,62	1,68	1,73	1,76	1,79	1,81	1,81	1,8	1,78	1,76	1,72	1,66	1,6	1,53	1,45	1,36	1,26	1,16	1,04	0,92	0,8	0,67	0,53	0,4	0,26	0,13
-0,48	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,63	1,55	1,46	1,54	1,62	1,68	1,73	1,77	1,8	1,82	1,83	1,83	1,81	1,79	1,75	1,71	1,65	1,59	1,51	1,43	1,33	1,23	1,12	1	0,88	0,76	0,62	0,49	0,35	0,22

Delta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
-0,5	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,63	1,55	1,47	1,55	1,62	1,68	1,73	1,77	1,8	1,81	1,82	1,82	1,8	1,77	1,74	1,69	1,63	1,56	1,48	1,39	1,3	1,19	1,08	0,96	0,84	0,71	0,58	0,44	0,31	0,17	0,07
-0,6	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,71	1,63	1,56	1,48	1,55	1,62	1,67	1,71	1,74	1,76	1,77	1,76	1,75	1,72	1,68	1,63	1,57	1,5	1,41	1,32	1,22	1,11	1	0,87	0,75	0,62	0,48	0,34	0,21	0,12	0,15	0,27
-0,4	2	2	1,98	1,96	1,92	1,88	1,83	1,77	1,7	1,62	1,54	1,45	1,54	1,62	1,68	1,74	1,79	1,82	1,85	1,87	1,87	1,87	1,85	1,83	1,79	1,74	1,68	1,62	1,54	1,46	1,36	1,26	1,16	1,04	0,92	0,79	0,66	0,53	0,39	0,26

Figure 4: Euclidean distance to the target. From top to bottom: $\sigma = 0.01, 0.02, 0.1$.

References

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. doi: 10.48550/ARXIV.1312.5602.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016. doi: 10.48550/ARXIV.1602.01783.

Volodymyr et al. Mnih. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698.

David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. Sample efficient reinforcement learning with reinforce. 2020. doi: 10.48550/ARXIV.2010.11364.