Universiteit Leiden

Master Computer Science

Title :Discovering quantum communication strategies with multi-agent reinforcement learning

Name:	Athanasios Agrafiotis
Student ID:	2029413
Date:	20/01/2021
Specialisation:	Advanced Data Analytics
1st supervisor:	Evert Van Nieuwenburg
2nd supervisor:	

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Acknowledgements

Abstract

Communication channel systems are easy to use; however, they are vulnerable to attacks by a third Person. The third person can easily penetrate the channel and read or manipulate messages before reaching the receiver from the sender. For this purpose a number of protocols are recommended That can secure communication between the two parties. Nowadays, quantum computing has been shown to get benefit from such scenarios and introduces protocols that can encrypt and decrypt a message. One of those protocols is the protocol of Bemett and Brassard. The purpose of this Master thesis is to present a simulation of a quantum communication channel.
using reinforcement learning algorithms. In more details it describes the way the sender and the receiver exchange messages and how they verify the security of the channel with a secret key. The main goal of this Master thesis is to simulate a Quantum key distribution process using an artificial intelligence environment. In each episode the two agents are using a communication channel. The first agent reads a message, and then sends it to the second agent; the receiver verifies the message's correctness. In case the message has been transferred successfully, the episode ends with the maximum reward; in the other cases the reward is negative. A number of reinforcement learning algorithms were implemented during the Master thesis project. Namely, a Q-learning, deep q learning, evolutionary strategy, and a proximal policy optimization approach that solves artificial environments with optimal solutions. As a result, the agent performs the actions that are required to communicate with each other, avoiding any mistakes.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Reinforcement learning is a radically kind of computing that makes use of the artificial intelligence nature of matter in order to process information. The two prominent applications of reinforcement learning you will often read or hear about are the q-learning ([7]) and deep-q-learning [4] algorithm. For on-policy and off-policy problems respectively. However, in order to actually run these algorithms a relatively large number of cpu is required, on top of that, hyper-parameters tuning is needed, requiring even more training of the algorithms. As we are now entering the era of artificial technology we are unable to achieve these requirement at the moment. For this reason, we are interested in finding applications that are viable to execute on the devices that are presently available, or will be in the near future [15]. In particular, we want to know if there are useful communication protocol that encrypts and decrypts messages that could be available in the short term. Hopefully, by exploring the possibilities we will simulate a quantum channel that exhibit classical communication channel advantage , meaning they offer a secure or have capababilities beyond what is tractable on classical communication channel.

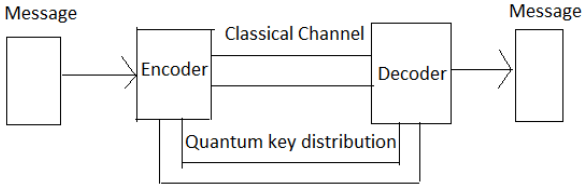
One of the promising algorithms that can be implemented on a simulation of quantum channel protocol is the Q-learning algorithm, that was proposed by Button and Sutton et al. [7] in 2020. The q-learing is an reinforcement learing algorithm to find solution in reinforcement learning environment, it an off-policy approach that can be advantageous, especially in quantum channel protocol approach, as the need for long coherence times is avoided by executing short calculations on a q-table.

The algorithm prepares a agent that navigate through states to infer a solution for the problem. An obstacle for feasible use of the algorithm is the need to find good parameters. Another topic of interest the performance of q-learning in the search space of statesm as it is largely unknown.

In this thesis i will explore how to simulate a quantum channel applied a number of reinforcement learning algorithms and it will compare classical channel and a quantum channel, in particular the Bernent and Bassard BB84 protocol [**?**], one of the best known communication protocol. This will be done by using a simulation technique propose in [7] for finding suitable, the best simulation parameters efficiently. In this work i will expand upon the results of [15] by analyzing the performance of reinforcement learning algorithms on a classical channel communication approach and a quantum channel approach, Moreover, the error distribution of the quantum channel will be investigated using man whitney statistical test using numerical results of the parameter optimization runs. Thoughout this work I will assume basic knowledge of linear algebra and that the reader is comfortable with terminology for quantum technology and reinforcement learning, such a qubits, policy and reinforcement learning algorithms. For the interested reader without necessary prerequisites i recommend the learning to play by Aske Plate [6]. For more gentle introduction i can highly recommend the website . Other good resource i enjoy reading are [12] and [8].

The project focuses on the following research questions:
Does the reinforcement learning environment simulate a Quantum key distribution?
Is the communication of a quantum channel that implements the BB84 protocol secure?
Is the protocol efficient?
To sum up, the project deploys an artificial environment that represents as states the encryption/decryption between messages of two parties. The implementation of a software that takes as an input plain text encrypts the message (cipher-text) and decrypts it. The implementation includes the quantum polarization base of each bit. An error analysis and the parameters that have been used during the simulation such as bitstream length(encoded message), error correction, number of iterations and the key quality.

Fig. 1.1 Quantum key distribution



in contrast a classical channel is defined as the amount of information that is transmitted over a channel. The channel is the capacity of a given channel that the information is transferred with small error probability. Based on the information

theory introduced by Claude E. Shannon defines the notion of channel capacity as the maximal information that can be transferred during a message transmission.The definition is called mutual information between the input and output of the channel with respect to the input distribution.

My thesis will be arrange as follows. I will start with an overview of Quantum channel protocol and previous work done on the topic chapter 2. It will continue a discussion of reinforcement learning algorithms on Chapter 3 with emphasis on the implementation of the algorithms. In Chapter 4 I will present a detailed dicussion of the implementation of the quantum channel approach propose in []. which is used to produce the results presented in Chapter 5. I will finalize this thesis with my conclusion and recommendation for future work in Chapter 6,7 and 8.

Chapter 2

Background

Chapter 2 provides an overview of relevant reinforcement learning algorithms. Quantum key distribution are described in theory and the concept of encryption and decryption protocol in Section 2.1. Reinforcement learning approaches are described in Section 2.2. More details regarding the reinforcement learning agent navigation in the artificial environment are presented in Section 3.1, 3.2, 3.3 and section 3.4.

2.1 Quantum Key distribution Related Work

Suppose that two parties party A and party B share correlated random classical bit strings X and Y. From these imperfect keys, are they gonna be secure using a cryptographic protocol. The two most important that must be performed is the *information reconciliation*, followed by *privacy amplification*, those are the most classical steps for the quantum key distribution protocol.
The information reconcilation is the error-correction tha take place in the classical channel, that share the error between the bitstring of party A and bistring of party B, to obtain the shifted key that will improve the connection. After the end of this connection a third-person(Eve) has already distilled a bitstring that is partial correlated with the two parties, the main purpose of the protocol known as information reconsiliation is to reduce the correlation of the bitstring of the eve with the bitstring of two parties. Information reconsiliation can be characterized as the correct information that the two parties are shared using privacy amplification. The *privacy amplification* uses the class of universal hash functions G, that maps the set of n-bit string A to some other bit-string B. For example Alice and Bob publicly select $g \in G$ and apply it to the sender bitstring (party A) the produces an encrypted bitstring S which they choose as their secret key. The G can correspond as a universal hash function and the probability

g(a_1)=g(a_2) is at most $1/|B|$.
The information reconcilation further increases the correct number of bits that the two parties can obtain, but this can be bounded. Bu computing a number of checks on the bitstring of party A. Party A can compose a classical message u consisting of subset specificications and parities , that transmitted to party B, after the message has been sent the connection of the classical channel allows party B to correct the error on it's bitstring(Y) and to transform the shifted key(W). The time that the message is transmitted from party A to party B is notated and the k represent the transmission number:

$$k > H(W|Y) \tag{2.1}$$

Quantum key distribution (QKD) that has a secure connection, that a private key bit is created between two parties for a connection. The key bit implement a classical private key cryptosystem. The only requirement for a quantum key distribution is that qubits communicated over the public channel with a threshold. The quantum information is how the key is secured with decryption and encryption. 1)the no-cloning theorem Eve cannot clone Alice's qubit. 2)Transmitting non-orthogonal qubit states between party A and party B By checking for disturbance in their transmitted states, they establish on any noise or third person monitor in the communication channel. To sum up the quantum key distribution idea is based on information reconcilation and privacy implication that represents the maximum tolerable error. At least three quantum key distribution protocols are use the some way.

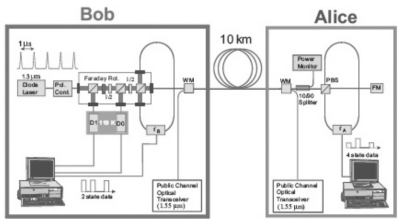


Fig. 2.1 Quantum key distribution schematic diagram of an IBM machine [**?**]

2.1.1 BB84 protocol

Alice begins with α and b, two strings each of $(4+\delta)n$ random classical bits. She then encodes these strings as a block $(4+\delta)n$ qubits,

$$|\psi >= \bigotimes^{(4+\delta)} n_{k=1}|\psi_{ak,bk} > \tag{2.2}$$

where a_k is the kth bit of α (and similarly for b), and each qubit is one of the four states

$$|\psi_{00}>=|0>\ |\psi_{10}>=|1>\ |\psi_{01}>=|+>=(|0>+|1>)/\sqrt{2}\ |\psi_{11}>=|->=(|0>-|1>)/\sqrt{2}$$

The effect of this procedure is to encode a in the basis X or Z, as determined by b. Note that the four states are not mutually orthogonal and therefore no measurement can distinguish between them with certainty. Party A sends $|\psi>$ to party B, over their public quantum communication channel.
Party B recieves $E(|\psi><\psi|)$, where E describes the quantum operation due to the combined effect of the channel and Eve's actions. He then publicly announces this fact. At this point, Party A, Party B and Eve's each have their own states described by separate density matrices. Note also that at this point, since Alice has not revealed b, Eve has no knowledge of what basis she should have measured in to eavesdrop on the communication; At best she can only guess, and if her guess was wrong, then she would have distributed the state revealed by Bob. Moreover, whereas in reality the noise E may be partially due to the environment (a poor channel) in addition to Eve's eavesdropping, it does not help Eve to have complete control over the channel, so that she is entirely responsible for E. Of course, Bob also finds $E(|\psi><\psi|)$ uninformative at this point, because he does not know anything about b. Nevertheless, he goes ahead and measures each qubit in basis X or Z, as determined by a random $(4+\delta)n$ bit string b' which he creates on his own. Let Bob's measurement result be a'. After this, Alice publicly announces b, and by discussion over a public channel, Bob and Alice discard all bits in [a',a] except those for which corresponding bits b' and b are equal. Their remaining bits satisfy a'=a, since for these bits Bob measured in the same basis Alice prepared in. Note that b reveals nothing about either a, or bits a' resulting from Bob's measurement, but it is important that Alice not publish b until after Bob announces reception of party A qubits. For simplicity in the following explanation, let party A and party B keep just 2n bits of their result; δ can be chosen sufficiently large so that this can be done with exponentially high probability.
Now party A and party B perform some tests to determine how much noise or eavesdropping happened during their communication. Party B selects n bits at random, and publicly announces the selection. Bob and Alice then publish and compare the values of these check bits. If more than t bits disagree, then they abort and re-try the protocol from start t is selected such that if the test passes, then they can apply information reconcilation and privacy amplification algorithms to obtain m acceptably secret shared key bits from the remaining n bits.

The related work et al Winiarczyk and Zabierowski[15] presents how to ensure risk management despite attacks on communication protocol. Current state-of-the-art-key distribution and management processes face constraints and challenges such as managing numerous encryption keys. The model demonstrates the BB84 quantum protocol with two scenarios; the first is without eavesdropper and the second is with eavesdropper via the interception-resend attack model. The simulation is highly dependent on communication over a quantum channel for polarized transmission. The cryptographic part relies on three components. First, the plain text that will be encrypted, the key used for the encryption; at last the output (cipher-text) encrypted message. The number of keys is two; one of the keys is public (encryption key) and the private key(decryption key). Two parties communicate with each other , the party A, and party B. The simulation is based on the communication of the two parties and in case the party A wants to send a message to party B is using the Party B's public key for the encryption and Party's B private key for the decryption. The procedure of simulation uses quantum blocks, the Party's A QB transmitter, Party's B QB receiver, and at last the Eve's QB non-authorized access to the quantum channel, the transmitter and reciever quantum blocks uses a guantum gates of X 2.3 and Z 2.4 . The paper concludes that the error is detectable with error correction rate 0.24% and 0.26% with eavesdropper, so the key has improved after each message exchange until to reach the paper's proposed threshold 0.11. Finally, the paper mentions that comparison of two scenarios without and with eavesdroppers is complicated and difficult to compare to previous work, as their analysis does not clearly state their parameters and the error.

$$|\psi_i >= \begin{pmatrix}1\\0\end{pmatrix} X = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix} |\psi f >= X|\psi_i >= \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix} \tag{2.3}$$

The X gate Flips the state $|0> x|0 >= |1>$.

$$\underset{\text{initial state}}{|0\rangle} \text{——}X\text{——} |1\rangle$$

$$|\psi_i >= \begin{pmatrix}1\\0\end{pmatrix} Z = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} |\psi f >= Z|\psi_i >= \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} \tag{2.4}$$

The Z gate Flips the state $|0> x|0 >= |0>$.

$$\underset{\text{initial state}}{|0\rangle} \text{——}Z\text{——} |0\rangle$$

Algorithm 1 The BB84 QKD protocol

1: Alice chooses $(4+\delta)n$ random data bits
2: Alice chooses a random $(4+\delta)$n-bit string b.She encodes each data bit as $|0>,|1>$
if the corresponding bit of b is 0 or $|+>,|->$
3: if b is 1.
4: Alice sends the resulting state to Bob
5: Bob recieves the $(4+\delta)n$qubits, announces this fact, and measures each qubit in the X or Z basis at random.
6: Alice announces b
7: Alice and Bob discard any bits where Bob measured a different basis than Alice prepared.
With high probability, there are at least 2n bits left (if not abort the protocol). They keep 2n bits
8: Alice selects a subsets of n bits that will to serve as a check on Eve's interference
, and tells Bob which bits she selected
9: Alice and Bob announce and compare the values of the n check bits.
If more than than an acceptable number disagree, they abort the protocol
10: Alice and Bob perform information reconcilation and privacy amplification on the remaining n bits to obtain m shared key bits

2.1.2 Classical Channel

The noise in a classical channel is one of the disadvantages that is not possible to avoid. Even in modern computer the existence of noise has been measured as one error in 10^{17} operations, this has as an outcome to consider that the new technology as a noiseless system. Several methods has been implemented to decrease the error that the noise can produce, error-correcting codes can protect against the effect of noise. The first method that implemented to reduce the noise is known as adding redundant information. The main idea is the existence of a classical channel that exchange messages because of the noise the message is repeated a several number of times. This has as a result to be enough redundancy in the encoded message so the message can recover. In a simulation of a classical channel two parties that communicate with messages , when the sender transmits a bit to the receiver there is probability $p > 0$ that the bit will change from logical 0 to logical 1. While with probability $p-1$ the bit remain unchanged. The channel that described above represent a classical channel known as *binary symmetric channel*. The redundant information solution propose the repetition of the message bit

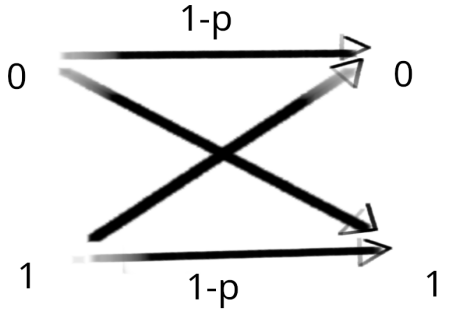


Fig. 2.2 Binary Symmetric Channel

a number of times that could protect the message bit from the change.

$$0- > 000 \tag{2.5}$$

The redundant information send through the classical channel the message as a bit-string referred as 000 to as the logical 0 and logical 1. At the receivers end of the channel three bits are send and as output. The receiver has to decide what was the value of the original bit was, suppose 001 were output from the channel. The probability p demonstrates the change of the bit and in case is low it is has shown that the third bit was flipped from 0 to 1.
An important metric that is called the major voting that describes the number of bits that have been changed during the transmission of the bits through the channel. The metric fails if two or more of the bits has been changed during the transmission and succeeds otherwise. The probability that two or more of the bits are flipped is $3p^2(1-p)+p^3$, the probability of error is $p_e = 3p^2 - 2p^3$ with no encoding, where the probability p is expressed as the change of the bit, so the code makes the transmission more reliable provided $p_e < p$ which occurs when $p < \frac{1}{2}$.
The adding redundant information is called the *repetition code*, the method procedure is to repeat the message a number of times and to encoded that had as a result to make message recoverable after noise has acted on the encoded message, with the amount of redundancy needing to be added depending on the severity of noise in the channel.

2.1.3 Bit Flip channel

In channel that uses qubits to exchange messages the probability p the qubit is flipped and with probability $1-p$ the qubit remain the same. The probability p is highly depends on the bit flip and explained as a state $|\psi>$ is taken to the $X|\psi>$, where X is the usual Pauli sigma X operator. The name of the channel is *bit flip channel*.
To encode the single qubit state a|0>+b|1> in three qubits a|000> + b|111>.

$$|0> - > |0_L >= |000> |1> - > |1_L >= |111> \tag{2.6}$$

The notation of L indicates that these are the logical $|0>$ and logical $|1>$ states, not the physical states. In a bit flip channel the procedure follows the next steps the initial state $a|0>+b|1>$ has been perfectly encoded as $a|000>+b|111>$ it's of the repetitions pass through an independent copy of the bit flip channel. The bit

changes on one or fewer qubits. The *error correction* is a method to reveal the bit that has change and to correct the quantum state: The first step the method is called the *error syndrome*. The error syndrome , corresponding to four projection operators: $P_0 = |000 >< 000| + |111 >< 111|$ no error $P_1 = |100 >< 100| + |011 >< 011|$ bit flip on qubit one $P_2 = |010 >< 010| + |101 >< 101|$ bit flip on qubit two $P_3 = |001 >< 001| + |110 >< 110|$ bit flip on qubit three the second step is called *recovery*, we use the value of the error syndrome to tell us what procedure to use to recover the encoded qubit that was sent. For example, if the error syndrome was 1, indicating a bit flip on the first qubit that recovers the input state $a|000 > +b|111 >$. The recover will return exactly the qubit that was flipped during the transmission in case it was no change will return 0 , in case the first qubit has changed it returns 1, second qubit has changed it returns 2 and at last the third qubit has changed it returns 3.
An error is provided by the bit flip of X. The gate X as explained in 2.3 it changes the state $|0 >$ to $|1 >$ and the opposite.
This fidelity quantum measures the object quantum error-corrections increases the fidelity with which quantum information is stored with maximum value of one. The fidelity can display the qubit flip in 3 bit changes.

$$F(|\psi >, p) = \sqrt{\psi|p|\psi} = \sqrt{(1-p) + p(< \psi|X|\psi >< \psi|X|\psi >} \tag{2.7}$$

The minimum fidelity is $F = \sqrt{1-p}$. Suppose the three qubit errocorrecting code is used to protect the $|\psi >= a|0_L > +b|1_L >$. The quantum state after the noise and error-correction is:

$$p = [(1-p)^3 + 3p(1-p)^2]|\psi >< \psi| \tag{2.8}$$

the ommited terms represent contributions from bit flips on two or three qubits.

2.2 Reinforcement learning Related Work

2.2.1 Tic Tac Toe

The Tic Tac Toe problem that represents the board. The simulation is of two environments and each agent that consider as a player retain a copy of his own board. Each of the player has as a a letter to distinguish it's move the first agent has the 'X' and the second has the 'O'. After the agent plays, the environment gives a new state that

the opposite agent will given the action.

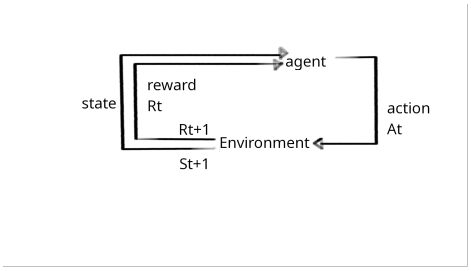


Fig. 2.3 Reinforcement learning Environment

The most important aspects of the reinforcement learning is the *Agent* that is an entity learning about the environment and making decisions. It is specify a learning algorithm for the agent that allows it to learn a policy. *Environment* everything outside the agent, including other agents, *Reward* the number that represent how did the agent performed from one state to another is positive if the new state was close to the terminal and zero or negative otherwise. The *State* a numerical representation of the environment, each time that agent navigate the state changes from S-> S_t where S is the set of all possible states. The *Action* is usually a restricted move that the agent can make in the environment in the tic tac toe game the moves that the agent can make are up,down,right,left and is notated as A. The policy represent the probability that is expressed given the action and the current state which will be the new state notated as $\pi(\alpha|S)$ The markov decision process the environment is completely characterized by the transition dynamics equation.

$$p(s', r|s, a) \tag{2.9}$$

That is, a probability that expresses a correlation between the current state and action and the new state and the reward that was obtained from the previous state and action.
The policy can be categorized to value function and action value function. The state-value function of a state S is the expected return from following the policy.

$$u_\pi(s) = E_\pi[G_t|S_t = s] \tag{2.10}$$

The action value function is the value of the state given the action plus the expected reward.

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] \tag{2.11}$$

bellman equation policy, value function The problem will terminate as soon as one of the player has achieved to make three in the row choices and consider as the terminate state. The is a small paradigm in AI that can potentially be the key solving real world problems. Nowadays the Reinforcement learning has seen an almost exclusive focus on games.

In deterministic environment the agent has full knowledge of the actions states and rewards. Recommended algorithms for deterministic environments is the Bellman equation($Q(s,a) = Reward + \gamma * maxQ(s',a')$) [1].

A non-deterministic environment known as stochastic the agent has no clear mapping between states and actions. It is not known always that the agent being in a specific state and take an action will step to the next defined state, random events can interrupt the agent. At most in stochastic environment the agent navigates with probabilities. Recommended algorithms for stochastic environments Q-learning methods($Q(s,a) = (1-a)Q(s,a) + a[r + \gamma * maxQ(s',a')$) [14].

The field of reinforcement learning introduces a number of algorithms that can solve artificial environments. The taxonomy of the reinforcement learning algorithms is model-free, model-based, value-based, policy-based off-policy and on-policy. The model-free use of data from the environment and navigation strategy of the agent express a probability. The model-based agent selects the actions that maximize its reward from the environment predictions. Value-based maximizes reward through navigation in the environment. Policy-based update their parameters through gradient descent by taking the differentiate. Off-policy expresses two separate policies, one of them to participate in the optimization process and the other to explore the environment; in contrast, on-policy expresses a single policy for the exploration and optimization process.

The most known algorithm that solves the artificial environment is the Bellman equation. The equation uses the artificial environment variables s,a,r and γ, which corresponds to the state, action , reward and discount factor. The agent is in an environment that navigates and in case the agent loses, get a negative reward, and in case that the agent performs all the actions without reaching the wining state, get zero, and in case of win takes a positive reward. The Bellman equation helps the agent to go through the environment. The bellman equation

$$(s) = max_a R(s,a) + \gamma V(s') \tag{2.12}$$

Describes how the agent takes an action in a state s, instantly gets a reward by getting in a new state. There are different actions that the agent can take; for every one of the actions the Bellman equation will express a probability. The value of each state is equal to the maximum reward that the next state gives. In case the agent moves to the winning state it takes a reward of 1, in any other case the agent takes an action and calculates the discount factor(γ) plus the differentiate reward of the current state with the winning state [2].

Chapter 3

Methods & Data

Chapter 3 details of the q-learning, the deep q-learning, proximal policy optimization, and evolutionary strategies. The section describes in details the theory and the implementation of reinforcement learning algorithms.

The first family of algorithms they used value table of estimate the q value of each action state pair are stored independently but this method had result dealing with complex time known as *value table*.

$$a = argmax_a Q(s, a) \tag{3.1}$$

to improve this algorithm it was combined with function approximator instead of giving individual number to each action state pair (Q). It is recommend a function that modify the action and state during the learning process and has better accuracy.

$$a = argmax_a \hat{q}(s, a|\theta) \tag{3.2}$$

The value table is a greedy approach that has a table with states and actions and the it chooses the action that maximize it's return. The value table method picks the next action from the stored values. The policy is defined from the values of value table and the function approximator. The function approximator to estimate the probabilities of the policy taking each action. The function approximator totally depends on an neural network that based on that , it will estimate a vector of probabilities with a probability of executing each of the actions.

$$s = [s_1, s_2] \tag{3.3}$$

$$\pi(s|\theta) = [p(a_1), p(a_2), p(a_3)] \tag{3.4}$$

The action that has highest probability will be chosen and the sum of all the probabilities are equal to a hundred. The neural network represents the policy and these are stochastic policies in which each action has mapped to each of the probabilities. The value based methods are not recommended as stochastic policies. In a artificial environment the stochastic policies will estimate a probability for the next action that is higher than the other probabilities of the next actions. In contrast value based methods are recommended for deterministic policies. A greedy policy is always chooses the action with the highest quality:
Another policy estimation depends on the parameter ϵ: When the maximum q-value changes in a state from one action to another. The probability of taking the new action goes from 0 to 100% and the opposite happens with the action that is no longer optimal. In contrast with policy gradient methods, the probabilities of taking an action gradually increase. If the action turns out to be effective in gradually decrease, if the action turns out not give high return.
The core of the neural network are composed of artificial neurons that takes as an input a vector the aggregate interest for that input passing to the hidden layer to calculate the values the process is known as forward propagation and performs a matrix multiplication where the vector consists of the outputs of each layer where that is represented as intensity matrix of the connections between each neuron(node).

$$x = [x_1, x_2, x_3] \tag{3.5}$$

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} \end{bmatrix} \tag{3.6}$$

$$h = [\phi(x_1 w_{11}) + x_2 w_{21} + x_3 w_{31}) ... \phi(x_1 w_{16} + x_2 w_{26} + x_3 w_{36})] \tag{3.7}$$

The layers can be viewed as a function:

$$y = \phi_2(\phi_1(x * w_1) * w_2) \tag{3.8}$$

on the first layer the input will be mitliplied with a weight next will be applied an activaion function. In the next operation the output of the layer will be multiplied with the new neuron (w_2) and the ouput will pass through an activation function. By

changing the neuron's weights (w_1, w_2) we can modify it to approximate the function that will give the expected return. This way produces the probabilities on the ouput layer. At the last layer the softmax function are used to express the probabilities based on the problem, to explain in details how the function works.

$$\sigma(x_i) = \frac{e^{x_i}}{\sum e^{x_i}} \tag{3.9}$$

So the input of the network is:

$$x = [x_1, x_2, ..., x_n] \tag{3.10}$$

The output of the network with the softmax function(σ) is:

$$y = [\sigma(x_1), \sigma(x_2), ..., \sigma(x_n)] \tag{3.11}$$

$$\sum_i \sigma(x_i) = 1 \tag{3.12}$$

if we want to find the optimal policy, we need to be able to compare them and to choose the best one. And a measure to evaluate the policy performance. In case the network hyperparameters change it produces different probability vectors and policy changes because the policy is a function of it's parameters $J(\theta)$. The goal is find the θ that can produce the optimal performance.

$$\pi(a|s) = argmax(\theta) \tag{3.13}$$

To calculate the policy, the stochastic gradient ascent (SGA) can be used where α is the learning rate:

$$\theta_{t+1} = \theta_t + \alpha \bigtriangledown J(\hat{\theta}) \tag{3.14}$$

where it updates the gradient vector contains the partial derivatives with respect to the parameters of the neural network. The modification of these parameters will lead to the optimal solution.

$$\bigtriangledown \hat{J}(\theta) = \left[\frac{\delta \hat{J}(\theta)}{(\delta \theta_1)}, \frac{\delta \hat{J}(\theta)}{(\delta \theta_2)}, ..., \frac{\delta \hat{J}(\theta)}{(\delta \theta_n)} \right] \tag{3.15}$$

the policy gradient theorem

$$\bigtriangledown J(\theta) = \sum \mu(s) \sum_{a} q_{\pi}(s, \alpha) \bigtriangledown \pi(\alpha|s, \theta) \tag{3.16}$$

The result is the gradient of policy performance is the return when the agent takes a specific action in each state multiplied by the gradient probability of taking that action in that state and weighted by the frequency with which observe in each state following that policy, the μ is the state distribution following policy π. The q-value of a state action pair following policy π. The gradient of the probability of selecting action a.
The algorithms has been proposed , so the agent will face a task for a complete episode to collect experience, and at the end of the episode it will update the neural network weights. For the update it can be performed stochastic gradient descent **??** for each action taken during the episode so that the policy improves of the expected return.

$$\bigtriangledown \hat{J}(\theta) = \gamma^{t} G_{t} \frac{\bigtriangledown \pi(A_t|S_t, \theta_t}{\pi(A_t|S_t, \theta_t} \tag{3.17}$$

So the policy updated at time t at which the action we want to update the policy that was taken. Multiplying the expression by these values gives more weight to the updates corresponding to the initial actions and less weight to the final ones. That's important because the initial cations have a larger effect on the outcome of the task. Since the bot action at the beginning of the episode can lead the agent to a state from which it cannot recover. Represents the fraction of the probability that this action has been chosen divided the probability the action will be choose now. Therefore , the final update rule will be:

$$\theta_{t+1} = \theta_t + a\gamma^{t} G_t \bigtriangledown ln\pi(A_t|S_t, \theta_t) \tag{3.18}$$

3.1 Q-learning

An agent uses the values of the next states to make a decision on which state to move next et al. Sutton R. and Barto [7]. A tabular representation of the actions is used as Q that represents the quality of the actions. If the environment has a specific number of actions, each of the actions has a quality. $Q(s, a) = R(s, a) + \gamma(Q(s', a'))$ Using q-learning, the agent performs an action; he gets a reward, and also it gets the expected value.

Algorithm 2 Q-learning [7]

1: `Initialize` $Q(s,a), \forall s \in S, \alpha \in A(s)$`,arbitrarily, and Q (terminal-state,`$\cdot$`)` $= 0$
2: **for** `episode` $\in 1..N$ **do**
3: `Initialize S`
4: **for** $t \in 0..T-1$ **do**
5: `Choose A from S using policy derived from Q(e.g., e-greedy)`
6: `Take action A, observe R,S'`
7: $Q(S,A) \leftarrow Q(S,A) + a[R + \gamma max_a Q(S',A) - Q(S,A)]$
8: $S \leftarrow S'$
9: `until S in terminal`

3.2 Deep Q-learning

The Deep Q-learning is similar to the q-learning approach. The agent before proceeding to the next state, calculates the reward and the policy of the new action. As the agent explores the environment understand the values of the states and the q-learning the values of the actions. In the process of deep q-learning, each of the states is used by a neural network that processes the information and the it outputs the actions. It uses the observation of the agent and outputs probabilities for the new actions that the agent should take to maximize its reward. The q-learning is not working in complex environment in contrast to the deep q-learning agent. The temporal difference is the foundation for expressing probabilities , when the agent takes the decision to move to the next state. The agent by taking this action with the maximum policy, receives better rewards than by taking another action.

$$TD(a,s) = R(s,a) + \gamma max_a Q(s',a') - Q(s,a) \tag{3.19}$$

In more details et al. Mnih Kavukcuoglu [4], Deep q-learning will predict values based on the number of actions. The neural network will compare the values of the current action and current state with the action and state of a previous episode. On the first episode, the agent has to calculate the value of each state in tabular $Q(s,a)$ and then the neural network generates a number of similar values and subtracks them until convergence. The neural network preprocess a sequence of inputs $x(1)...x(n)$, a number of hidden layers and outputs based on the number of environment actions (targets) $Q_1...Q_n$. For the process of propagation measure, the loss $L = \sum(QTarget - Q)$ is the way that agent learns.
The experience replay gives the agent the opportunity to learn from a sample of the

state. It takes a number of samples that are random and uniform, and the network learns from them known as experience; each experience has the state that the agent was in, the next state, the action and the reward (four elements) [10]. The most valuable are rare experiences, data that contains states that do not repeat frequently. The inputs in the neural network are the move of the agent from one state to another state. The state goes through the network; the error is calculated and the network backpropagates; then the agent selects which action needs to be taken. The new state is used as the previous and goes through the network.

Once the vector describing the state is used from a neural network, and the learning process ends, it outputs all the $q-values$. The predictions are the $q-values$; the activation function selects the best $q-value$. The q-learning approach selects the one with the highest $q-value$ and takes that action. There is also a number of different action selection function such as the $e-greedy$ and $e-soft$ (1-e) [13]. The e-greedy selects the action with the highest reward when the e-greedy is 0.4 Forty percent selects the action random, and 0.6 selects the action with the highest reward. $E-soft$ selects a random action; if the $e-soft$ is 0.2, the agent selects the action with the highest reward and with 0.8 selects an action at random. The number of different action selection policies provides different ways for the agent to navigate through the environment; other times the agent exploits the environment and other times explores it. The different functions prevent the agent to be trapped to the local maximum, so the agent will navigate receiving the best reward, but it might not finish the episode with the maximum reward.

So the algorithm works in the following steps. It initializes a batch that is called an experience replay. The size of the memory is chosen manually. At each time, t, repeats the following process, until the end of the epoch. It predicts the $q-value$ or policy of the current state s_t. The agent selects the actions with the highest policy to navigate through the next state using the bellman equation. It receives in return the reward of the new state. The navigation step (transition) (s_t, a, r_t, s_{t+1}) is stored in the experience replay storage. Once the capacity of the experience replay is full, the neural network produces new states and the bellman equation produces the target values. The loss between the produced policy of the neural network and the target updates the weights of the neural network.

Algorithm 3 Deep Q-learning Experience Replay [13]

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode $\in 1..M$ **do**
  Initialise sequence $s_1 = x_1$ and preprocessed sequenced $\phi = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action α_t
    otherwise select $\alpha_t = max_\alpha Q*(\phi(s_t), \alpha; \theta)$
    Execute action α_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, \alpha_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi_{s_{t+1}}$
    Store transition $(\phi_t, \alpha_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, \alpha_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{Bmatrix} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, \alpha_j; \theta) & \text{for non-terminal} \phi_{j+1} \end{Bmatrix}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, \alpha_j; \theta))^2$

3.3 Proximal Policy Optimization

Instead of having model value, we have a neural network model, the policy itself it called the distribution π which is parametrized by theta et al [11]. Action (a) is the random variable that determines the distribution for a given state (s) actor network. In this case the input to the neural network is the state where the output is the probability distribution that is expressed in order to take the best action. The policy network methods use the gradient method that depends on the neural network . The gradient of our objective is the expected value of the advantage multiplied by the gradient of the log policy; the advantage is equal to the action value minus the state value. In practice there is an estimation of the expected value by sample mean, by collecting a pair of samples through playing the game and dividing by the number of samples. The way to produce π depends on the aggregation of total rewards. Rewards are just samples drawn from playing the game, so there is no correlation of the rewards with the weights of the neural networks. Consider a sequence of state-action pairs in an episode $(s_1, a_1)(s_2, a_2)(s_3, a_3)$ By performing the state action pairs, the agent collects the rewards.

The objective $j(\theta)$ is a function of the neural network weights. θ is equal to the expected value of the rewards collects through the states under the distribution π_θ. The π_θ that is, the output of the neural network that called policy distribution. Therefore, the expected value is with respect to the policy distribution. So when the agent plays, an episode is using the policy that is expressed from the neural network. The probability

distribution can be expressed as a markov chain; that is, the transition probability of the environment expresses the state and action returns the new state and the policy, that is, the output of the neural network that corresponds to the action given the state. $\pi(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$

$$\delta_\theta J(\theta) = E[\sum_{t=1}^{T} \delta_\theta log\pi_\theta(a_t|s_t) \sum_{t=t+1}^{T} R(s_t, a_t)] \tag{3.20}$$

The future rewards can be replaced the second part of future rewards. The modification on the rewards can vary even using a q-learning approach, but in the case of actor and critic The agent does not care about absolute reward but to improve current policy. So it makes use of the advantage function prediction, the value of the new state and subtracted by the current state of the agent, in case that gives a bigger number and makes the action more probable.

$$A(s, a) = Q(s, a) - V(s) \tag{3.21}$$

$$A(s, a) = R + \gamma V(s') - V(s) \tag{3.22}$$

The actor critic plays a number of episodes and stores the states and the actions calculate the advantage function and follow the direction of the gradient. The gradient is updating from the loss function.

$$L(\theta) = -\frac{1}{M} \sum_{i=1}^{M} log\pi_\theta(a^i|s^i) A(s^i, \alpha^i) \tag{3.23}$$

To be able to limit the parameters of the policy in the same batch generates a probability of selecting a specific action in a specific state and use it as a reference to limit the change of our policy.

$$r_t(\theta) = \frac{\pi_\theta(a_t s_t)}{\pi_{\theta old}(a_t s_t)} \tag{3.24}$$

This means that the agent would have selected the current action (a) when it was in the state (t) with probability which was initial 12% and after some iterations the agent would choose the action (a) of the state (t) with probability 90%. The ratio (r) is the $\frac{90}{12}$. The ratio and a parameter called epsilon will limit policy changes. With this way

a new Loss function is computed

$$L(\theta) = E[min(r_k(\theta)A, clip(r_t(\theta), 1-e, 1+e)A] \tag{3.25}$$

The loss function selects lower value between the $clip(r_t(\theta), 1-e, 1+e)A$. The parameters A and $1+e$,$1-e$ will help to constrain the policy formula increase and make it even less probable. So an agent will not take actions that lead to positive advantage and negative advantage more times than suppose to. Many steps of learning in a sample of data but setting limit on the policy changes. The proximal policy learning through a specific number of episodes and run the policy for specific timesteps while the policy is optimized calculating the loss function.

Algorithm 4 PPO Clip [11]

initial policy parameters θ_0 initial value function parameters ϕ_0
for $k = 0, 1, 2...$ **do**
    Collect set of trajectories $D_k = t_i$ by running policy
$\pi_k \pi(\theta_k)$ in the environment
    Compute rewards-to-go $\hat{R}_t$
    Compute advantage estimatesA_t(using any method of advantage estimation
based on the current value function)$V_{\phi k}$
    Update the policy by maximizing the PPO-Clip objective:
    $\theta_{k+1} = argmax_\theta \frac{1}{|D_k|T} \sum_{t \in D_k} \sum_{t=0}^{T} min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}) A^{\pi_{\theta k}(s_t, a_t, g(e, A^{\pi_{\theta k}(s_t, a_t)}))}$,
typically via stochastic gradient ascent with Adam
    Fit value function by regression on mean-squared error:$\phi_{k+1} = argminn_\phi \frac{1}{|D_k|T} \sum_{t \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$ Typically via some gradient descent algorithm

3.4 Evolution Strategy

The algorithm generates an offspring one at a time with some gaussian noise that has been multiplied by standard deviation and added to the weight. The new policy will be examined by the fitness function that will produce the reward for an episode. This the optimization of the evolution strategy et al Salimans, T.Ho [9]. That ends with the update of the new policy based on the previous policy. In more details adds, the learning rate times the population size multiplied by the noise standard deviation times all the rewards multiplied by the corresponding noise vector. It is actually a multiplication of two vectors, the vector of noise and the vector, the policy that was previously evaluated by the fitness function to produce the new policy. This update is

highly dependent on the fitness function in case the reward is positive after the update is expected to be more positive.

$$\theta(t+1) = \theta(t) + \eta \frac{1}{N\sigma} \tag{3.26}$$

A summary evolutionary strategy update is an approximation of the gradient descent or ascent based on the reward that the fitness function produces. Similar to the advantage actor critic, the evolutionary strategy standardized the rewards because the rewards it might be positive but not always better than the previous rewards. So by standardizing the rewards, the evolutionary strategy tries to improve the results. The evolutionary strategies in reinforcement learning do not make use of the value function and the discounting rewards. The evolutionary strategies are highly depend in the learning rate, the population size (the number of offsprings) and the noise deviation that shows how different is the offsping(θ_{t+1}) for the parent(θ_t).

$$\bigtriangledown_\theta E_\epsilon \sim_{N(0,I)} F(\theta + \sigma\epsilon) = \frac{1}{\sigma} E_\epsilon \sim_{N(0,I)} F(\theta + \sigma\epsilon)\epsilon \tag{3.27}$$

Algorithm 5 Evolution Strategies [9]

`Input:Learning rate` $= \alpha$
`noise standard deviation=`σ
`initial policy parameters=`θ_0
for $t = 0, 1, 2, ..$ **do**
  $Sample_{\epsilon_1..\epsilon_n \sim N(0,I)}$
  `Compute returns` $F_i = F(\theta_t + \sigma\epsilon_i)$
  for $i = 1...n$ **do**
    `set` $\theta_{t+1} \leftarrow \theta_{t+1} + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$

To sum up, the evolution strategy makes use of a neural network architecture. Initializes its weight with random values in the new iteration the network calculates through matrix multiplication (feed-forward process) an action. Next, this action will be evaluated by the fitness function and based on the reward the agent will receive, the network weights will be updated. The weights network represents the offspring of the population and are updated from the objective function. The evolution strategy algorithm is a loop that uses specific variables the learning rate η the noise stand deviation σ of the rewards and the initial policy parameters θ.

3.5 Adaptation of the Code for the Communication Protocol 84

The code represents a simulation, a communication channel. Whereas the agents need to take specific number of actions so the communication is successful. We present the experiment to evaluate the performance of our proposed model. We evaluated the communication of two agents in terms of reward, which means the communication was successful and the key of the communication protocol is identical. The research has been repeated with different numbers of combinations as a secret key.
In order to examine the Quantum Protocol BB84 method, I implemented it using Python. For simulating the quantum communication that is shown in figure 1.1, Keras is artificial interlligence software developed by . It provides a way to simulate and compute neural networks. The training procedure was accomplished using CPU and all the experiments are runned through. Moreover, i made use a simple language packages numpy,pandas,matplotlib. The optimizers that was used were part of Keras and was mainly used the Adam optimizer. The neural network were simulated using Keras layers. The metrics were implemented using mathematical packages.
To benchmark the artificial intelligence algorithms to each other, they were analyzed running different network architectures and hyperparameters. I will abbreviate this method that some hyperparameters as the number of epoachs was selecred randomly and on the computer resources.

3.5.1 Training procedure

The agent learns to play the game for a specific number of episodes. In each episode the agent selects a number of actions , the number of actions are limited to four in case the agent navigate to the state with the maximal reward in the environment receives the full reward, in all other case the agent receives a negative or zero reward. Several methods for finding optimal hyperparameters have been proposed et al hyperparameters. These methods include solving the problem analytically, doing a gridsearch of the parameter space, using known optimization methods and even machine learning. These approaches unfortunately does not fit the cpu usage.
For the q-learning simulation the number of episodes were set to one hundred and the agent updates the q-table in each action with the policy. For the Q-learning algorithm, the gamma value 0.01 and the learning rate 0.001.

Index	0000	000-1	00-10	1111
00	0	0	0	0
01	0	0	0	0
02	0	0	0	0
03	0	0	0	0
04	0	0	0	0
05	0	0	0	0
06	0	0	0	0
07	0	0	0	0
08	0	0	0	0
09	0	0	0	0
10	0	0	0	0
11	0	0	0	0

Table 3.1 Q-table Q-learning 12x154

3.5.2 Quantum Communication

The communication that was analyzed were generated using the Python and an artificial intelligence environment, that simulates the message generation, the exchange protocol between two Parties A and B and states that the agent navigates. The artificial environment includes the encode and the decode protocol. To make sure the experiments done are reproducable. I saved each model that i have used and the number of seed was set to 0. Each episode finishes after the agent has completed all required steps. Python version 3.8 was used. Relevant software and Python Packages with corresponding are included in 7.1.

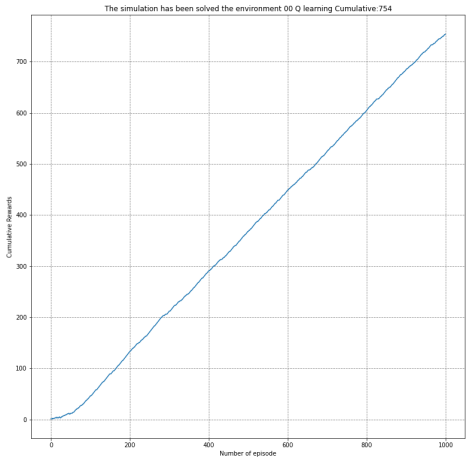


Fig. 3.1 Hyperparameters of Q learning gamma 0.001.

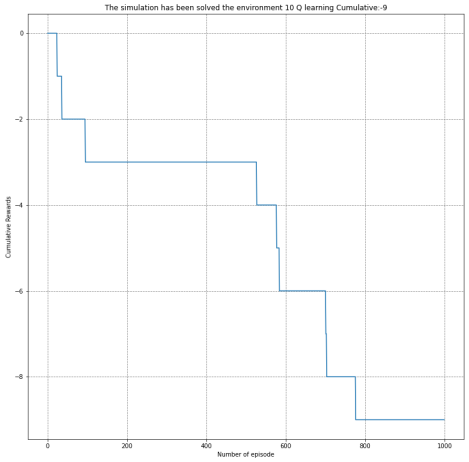


Fig. 3.2 Hyperparameters of Q learning gamma 0.99.

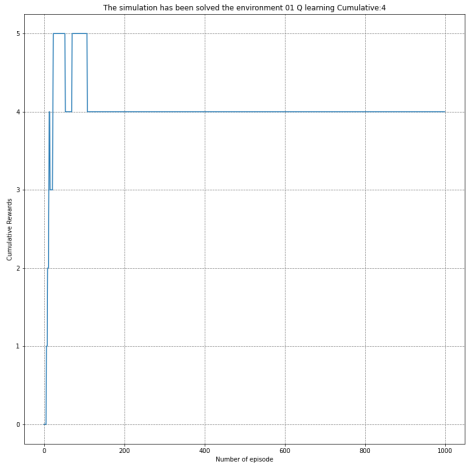


Fig. 3.3 Hyperparameters of Q learning gamma 0.99 and learning rate 1.

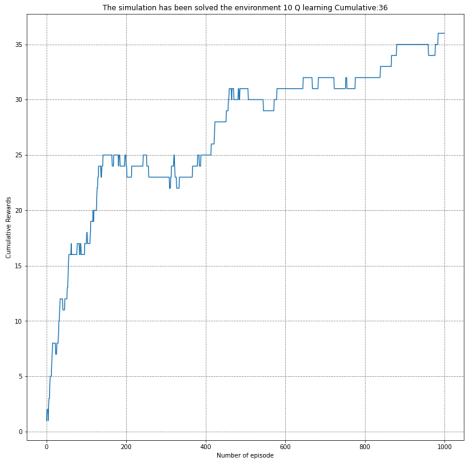


Fig. 3.4 Hyperparameters of Q learning gamma 0.001 and learning rate 1e-3.

For the deep q-learning simulation the environment interacts with the agent updating the weights of the network. The network takes as an input the number of states and outputs the corresponding action that will lead the agent to the maximal reward. The deep q-learning approach stores the reward,actions and each state that the agent has visited without updating the weights of the network. When the store is full the network training process starts. The network trained to predict the given the state(s) , the policy that will lead the agent to the new state(prediction of the network given the current state). The network was trained with batch size of 120 the number of epochs that the network is trained was set to 2 the memory size 200 the gamma value 0.99 the exploration rate decreases with rate 0.01 and the learning rate $1e-4$(Adam) and loss categorical cross-entropy. Network architecture $(4, 12, \texttt{softmax})$.

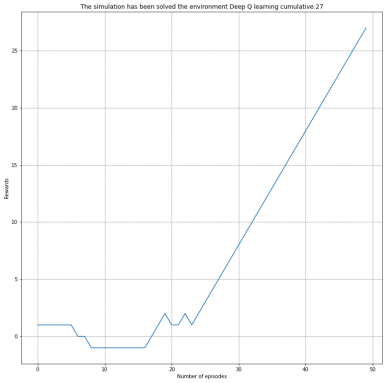


Fig. 3.5 Hyperparameters of DQN gamma 0.99 and learning rate 1e-3.

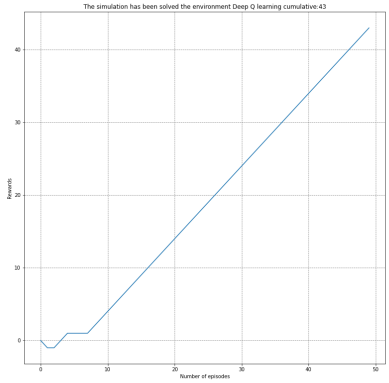


Fig. 3.6 Hyperparameters of DQN gamma 1e-3 and learning rate 1e-3.

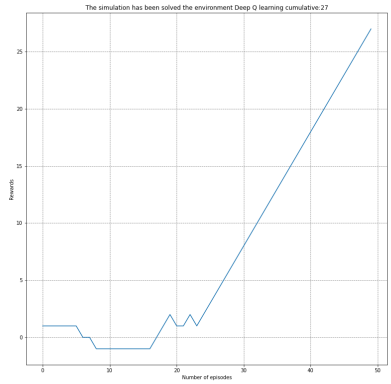


Fig. 3.7 Hyperparameters of DQN gamma 0.99 and learning rate 1.

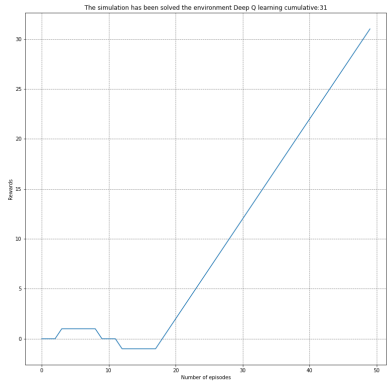


Fig. 3.8 Hyperparameters of DQN gamma 1e-3 and learning rate 1.

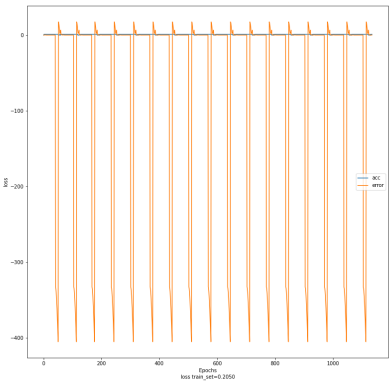


Fig. 3.9 Hyperparameters of DQN cross-entropy learning rate 1 Adam.

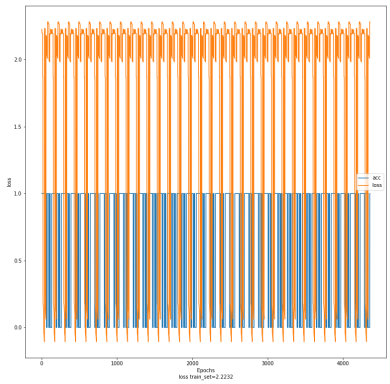


Fig. 3.10 Hyperparameters of DQN cross-entropy learning rate 1e-3 Adam.

The proximal policy optimization algorithm make use of two networks the actor and the critic network. The network of actor has an architecture of $(4, 16, 16, 12, \texttt{softmax})$ and generates the policy with the action that the agent will follow next. The critic network $(4, 16, 16, 1, \texttt{softmax})$ that express a probability t. The algorithm stores each state,action,reward,value,policy(s, a, r, v, p). After the number of episodes have finished the network updates the weights of the actor and critic network. Before the training process the network calculates the ratio (the subtraction of the policy and the actor network predictions of each state). The advantage that contains the value of each episode $(R_{t+1} + \gamma * V_t - V_{t+1})$ that represent the reward for the episode that have finished successfully. The actor network is trained using the average of the of the ratio multiplied with the advantage as a loss. The loss is used to calculate the derivatives that updates the network weights for 5000 iterations (Adam) rate $(3e-4)$.

The critic network is calculate the loss using the average and subtraction of value and critic network prediction using the states the agent has visited. The loss is used the calculate the derivatives and the weight network is updated for 5000 iterations (Adam) rate $(1e-3)$.

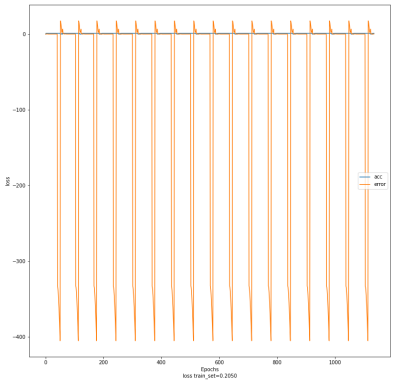


Fig. 3.11 Hyperparameters of PPO actor learning rate 1e-3.

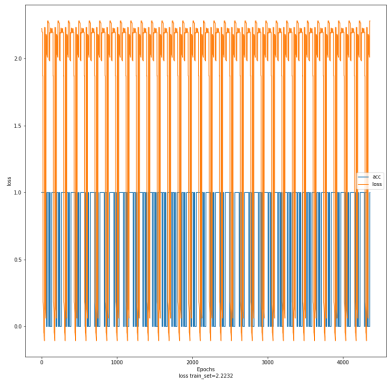


Fig. 3.12 Hyperparameters of PPO critic learning rate 1e-3.

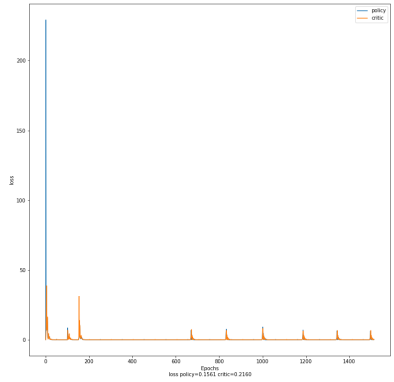


Fig. 3.13 Hyperparameters of PPO actor learning rate 1.

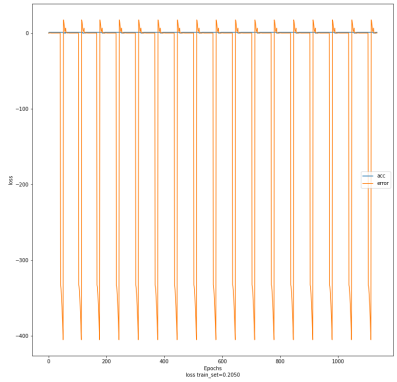


Fig. 3.14 Hyperparameters of PPO critic learning rate 1.

Further parameters are the clip ratio 0.01 and gamma 0.99, lam 0.97.
In the evolutionary strategy the network architecture initialized it's weight $(4, 32, 32, 12, softmax)$. The population size represents the combination of actions that agent uses during the training. The weights are updated using the fitness function after all the population has been played an episode. For the evolutionary strategy algorithm, the hyper-parameters

are the population size 5000,the sigma 0.01; the learning rate 0.16 and the number of iterations was set to 600.

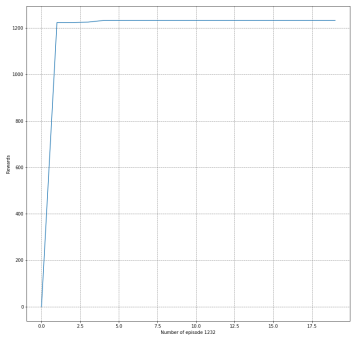


Fig. 3.15 Hyperparameters Evolutionary Strategy learning rate 1e-4 sigma 1e-5.

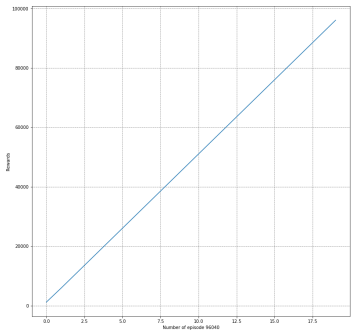


Fig. 3.16 Hyperparameters Evolutionary Strategy learning rate 1e-5 sigma 0.99.

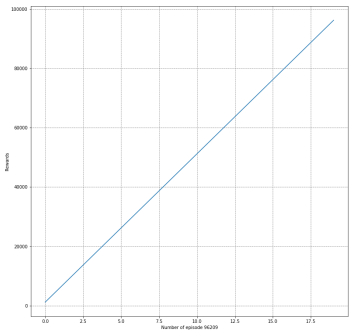


Fig. 3.17 Hyperparameters Evolutionary Strategy learning rate 1 sigma 0.99.

Chapter 4

Results

This chapter provides details of the results. The evaluation criteria and the adjustment of hyperparameters.

4.1 Metrics

The mannwhitney test assess the reward distribution of the agent and the error distribution distribution after the decoding of the gate X. Before calculate the test, choose a significance level usually a=0.05. As a first step is to assign ranks to the values from the full sample in order from smallest to the largest. Next it generates a test statistic based on the ranks. After the summation of the the distribution of error and the distribution of rewards. The Mann-Whitney U statistic is selected as the smallest of the two following calculated U values:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1 \tag{4.1}$$

$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2 \tag{4.2}$$

Where we let 1 denote the error distribution and 2 the reward distribution. The notation n_1 and n_2 are the number of episodes and R1,R2 are the sum of distribution respectively. Next, we determine a critical value of U with which to compare our calculated test statistic. Given the variable is equal to the critical value, the mannwhitney test reject the null hypothesis that the two groups are equal and accept the alternative hypothesis that there is evidence of a difference in the distribution between the reward

distribution and the error distribution.

4.2 Evaluation criteria

The final phase Table **??** it can be seen that the neural networks trained by the various algorithms presented obvious differences. When reward is used as the only indicator to measure neural network accuracy, proximal policy optimization is the best approach. The results demonstrate that proximal policy optimization can find the optimal estimator. When the number of steps as a metric is used as the only indicator to measure network performance, the Proximal Policy algorithm is the best network and the Deep Q-learning algorithm had the worst performance. Generally speaking, when the rewards of several algorithms are relatively close, simple algorithms should be given priority over complex networks. Among the four algorithms, Q-learning is the simplest one, with only two independent variables. The Deep Q-learning algorithms are the second simplest one, with four independent variables.
As can be observed, Table **??**, the size of the key derived from these four algorithms were clearly different from each other. In general, all the reward results reflected the key distribution process based on the size of the key. The best performance was shown in the results from proximal policy optimization. By contrast, the lowest results were shown in the results of the Deep q-learning algorithms. The highest rewards were gathered from the agent of proximal policy optimization and were 0.94, which appeared for size of key four. It is work noting that, except for Deep q-learning, there were no rewards lower than thirty. Among the results generated by the reinforcement learning algorithms, two of the algorithms that make use of neural network architecture were higher than those that use tabular methods, especially for the Proximal Policy Optimization approach.

4.2.1 Classical Channel

The figures of artificial intelligence algorithms, for q-learning, deep q-learning, proximal policy optimization and evolutionary strategy the reward and the number of steps are plotted in Figure 4.2, 4.5,4.7 and 4.11, the optimal cumulative reward approaches 100 episodes. The cumulative reward for all the number of keys and seems to follow a usual pattern that increases and reaches the highest peak . In the most of the cases

the reward increases, except for the approach of 1 bit message that reward drops bellow 0% before it start to increase . Another pattern that it can be observed is the times that agent receives the full reward and instantly falls for each episode for the case of evolutionary strategy 4 bit message 4.159 and the approach of proximal policy optimization 4 bit message 4.47 has the more steady flow in contrast with the other approaches that the reward instantly drops after a full reward episode.
The relation between the evolutionary strategy and the proximal policy optimization algorithm is shown. As can be seen , the reward increases exponential with the key length, an different behaviour is observed for the approach of q-learning, namely deep q-learning.
It is work noting that the number of steps is highly depends on the number of digits of the message of message >1 the minimum number of steps are 2. For the simulation of message equal to 4 the number of steps for the artificial intelligence algorithms that have solved the environment is 4 and 5 for the algorithms that did not performed.

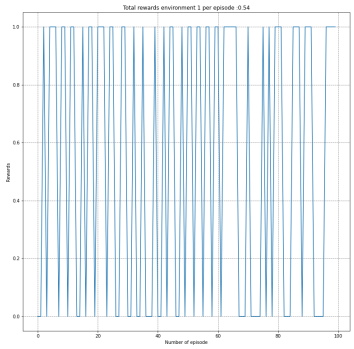


Fig. 4.1 1 digit Classical Channel Rewards one hundred episodes q-learning.

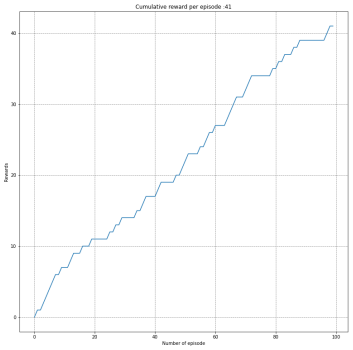


Fig. 4.2 1 digit Classical Channel Cumulative Reward one hundred episodes q-learning.

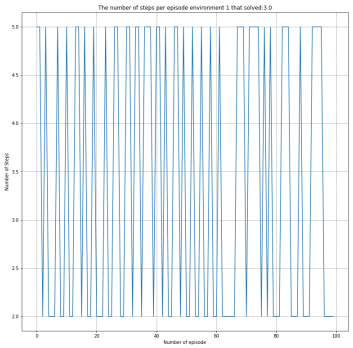


Fig. 4.3 1 digit Classical Channel Number of step one hundred episodes q-learning.

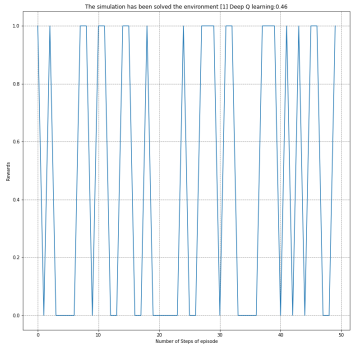


Fig. 4.4 1 digit Classical Channel Rewards one hundred episodes Deep q-learning.

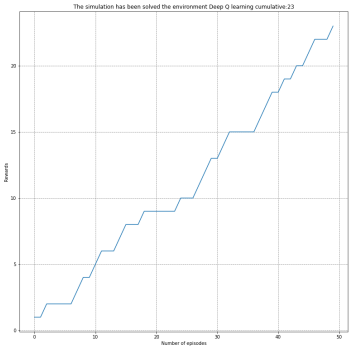


Fig. 4.5 1 digit Classical Channel Cumulative Reward one hundred episodes Deep q-learning.

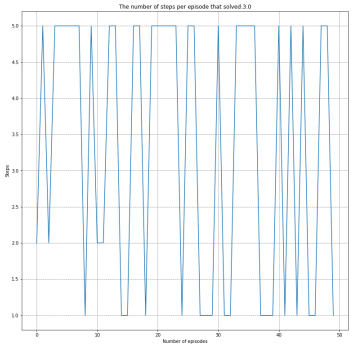


Fig. 4.6 1 digit Classical Channel Number of step one hundred episodes Deep q-learning.

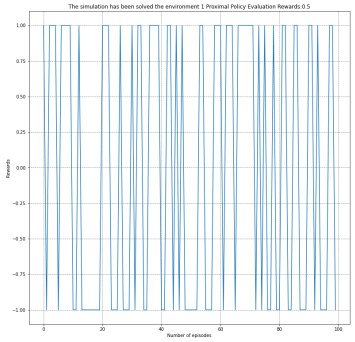


Fig. 4.7 1 digit Classical Channel Rewards one hundred episodes Proximal Policy Optimization.

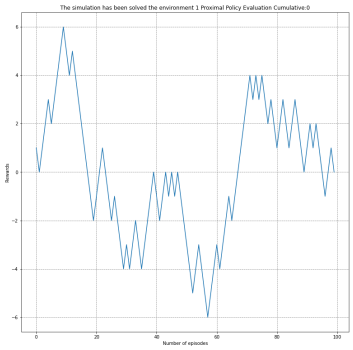


Fig. 4.8 1 digit Classical Channel Cumulative Reward one hunded episodes Proximal Policy Optimization.

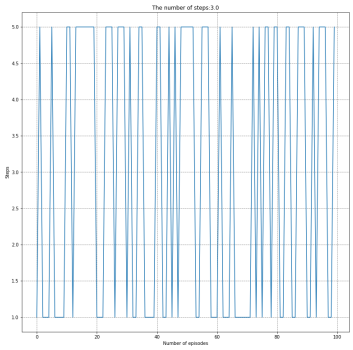


Fig. 4.9 1 digit Classical Channel Number of step one hundred episodes Proximal Policy Optimization.

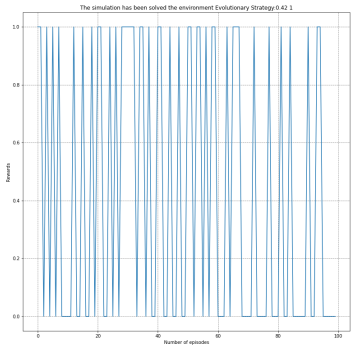


Fig. 4.10 1 digit Classical Channel Rewards one hundred episodes Evolutionary Strategy.

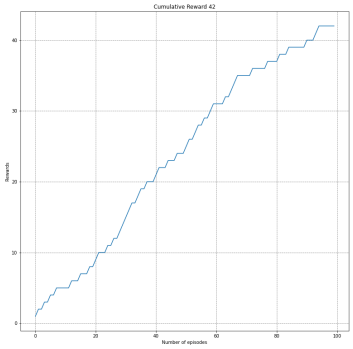


Fig. 4.11 1 digit Classical Channel Cumulative Reward one hundred episodes Evolutionary Strategy.

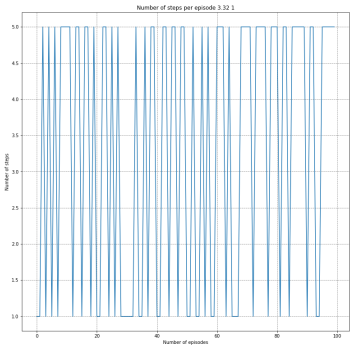


Fig. 4.12 1 digit Classical Channel Number of step one hundred episodes Evolutionary Strategy.

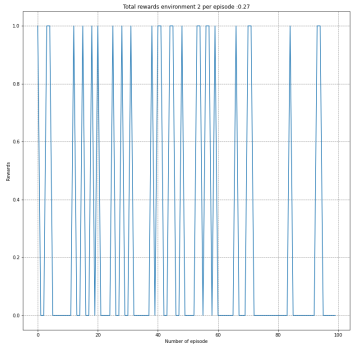


Fig. 4.13 2 digit Classical Channel Rewards one hundred episodes q-learning.

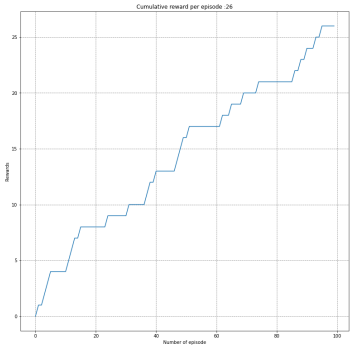


Fig. 4.14 2 digit Classical Channel Cumulative Reward one hunded episodes q-learning.

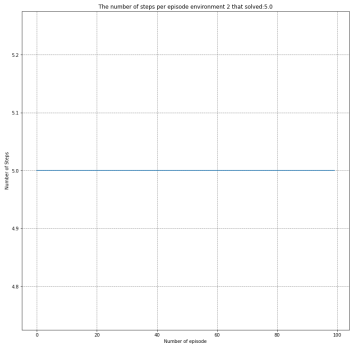


Fig. 4.15 2 digit Classical Channel Number of step one hundred episodes q-learning.

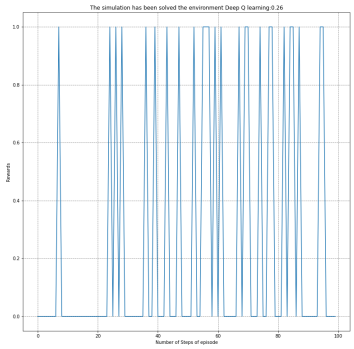


Fig. 4.16 2 digit Classical Channel Rewards one hundred episodes Deep q-learning.

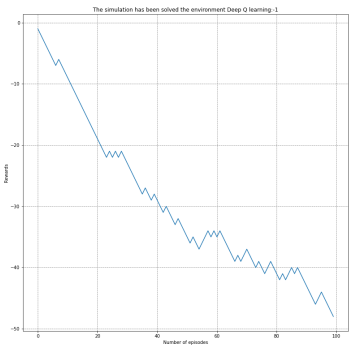


Fig. 4.17 2 digit Classical Channel Cumulative Reward one hunded episodes Deep q-learning.

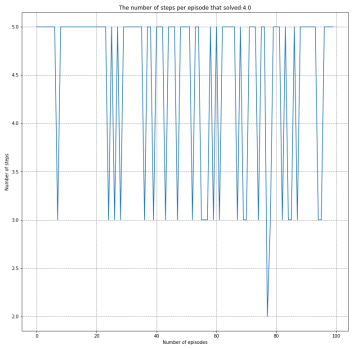


Fig. 4.18 2 digit Classical Channel Number of step one hundred episodes Deep q-learning.

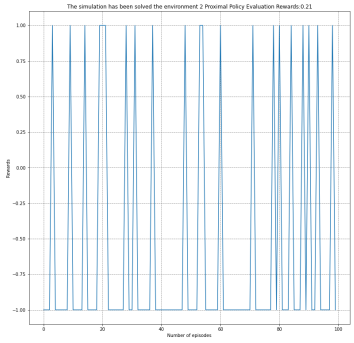


Fig. 4.19 2 digit Classical Channel Rewards one hundred episodes Proximal Policy Optimization.

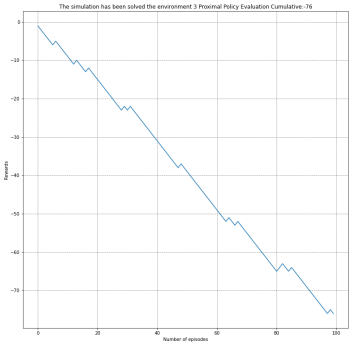


Fig. 4.20 2 digit Classical Channel Cumulative Reward one hunded episodes Proximal Policy Optimization.

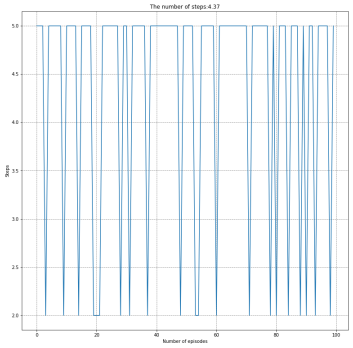


Fig. 4.21 2 digit Classical Channel Number of step one hundred episodes Proximal Policy Optimization.

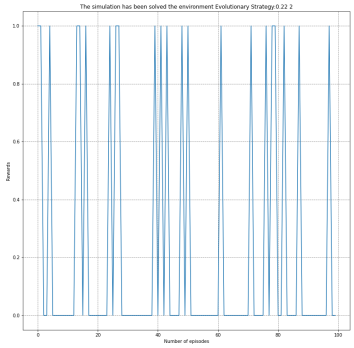


Fig. 4.22 2 digit Classical Channel Rewards one hundred episodes Evolutionary Strategy.

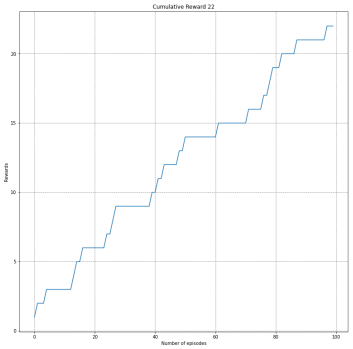


Fig. 4.23 2 digit Classical Channel Cumulative Reward one hundred episodes Evolutionary Strategy.

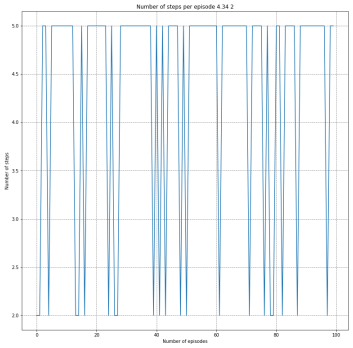


Fig. 4.24 2 digit Classical Channel Number of step one hundred episodes Evolutionary Strategy.

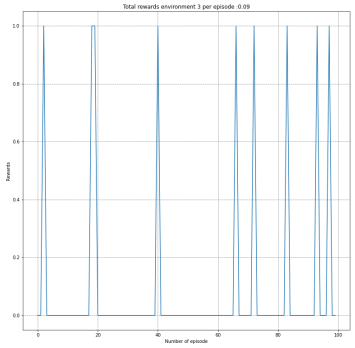


Fig. 4.25 3 digit Classical Channel Rewards one hundred episodes Evolutionary Strategy q-learning.

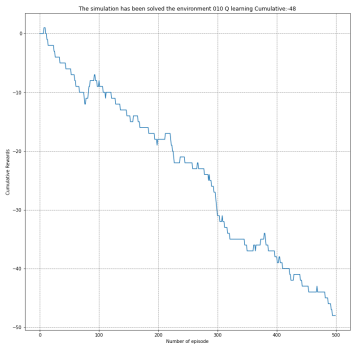


Fig. 4.26 3 digit Classical Channel Cumulative Reward one hundred episodes q-learning.

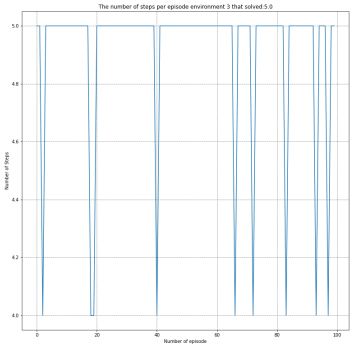


Fig. 4.27 3 digit Classical Channel Number of step one hundred episodes q learning.

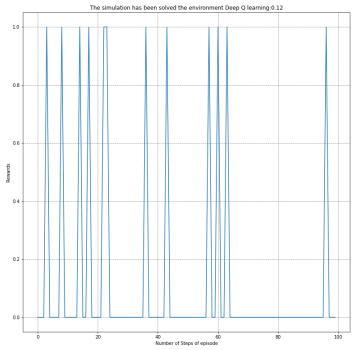


Fig. 4.28 3 digit Classical Channel Rewards one hundred episodes Deep q-learning.

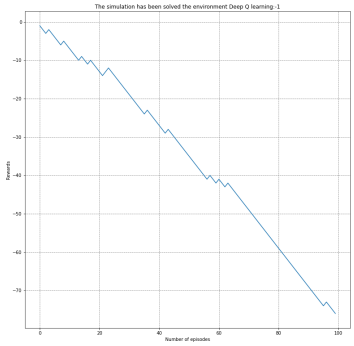


Fig. 4.29 3 digit Classical Channel Cumulative Reward one hundred episodes Deep q-learning.

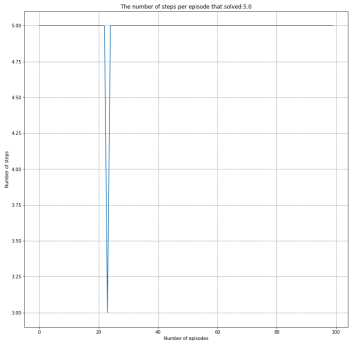


Fig. 4.30 3 digit Classical Channel Number of step one hundred episodes Deep q learning.

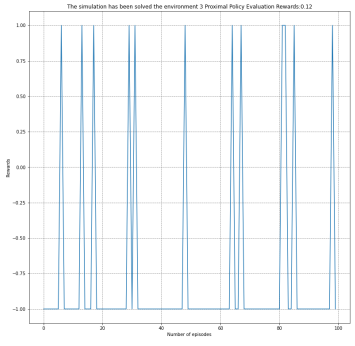


Fig. 4.31 3 digit Classical Channel Rewards one hundred episodes proximal policy optimization.

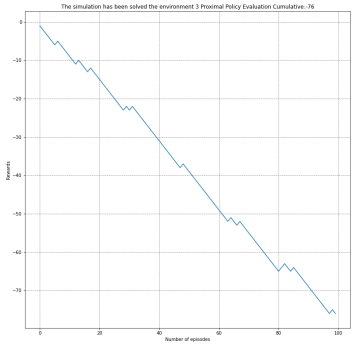


Fig. 4.32 3 digit Classical Channel Cumulative Reward one hundred episodes proximal policy optimization.

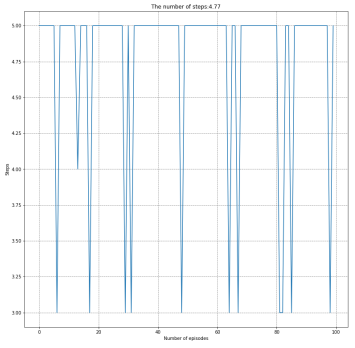


Fig. 4.33 3 digit Classical Channel Number of step one hundred episodes proximal policy optimization.

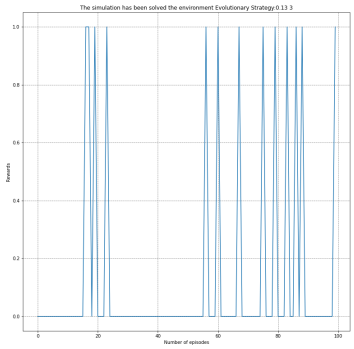


Fig. 4.34 3 digit Classical Channel Rewards one hundred episodes Evolutionary Strategy.

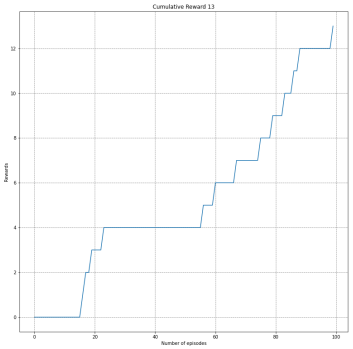


Fig. 4.35 3 digit Classical Channel Cumulative Reward one hundred episodes Evolutionary strategy.

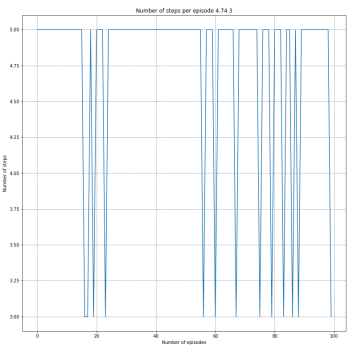


Fig. 4.36 3 digit Classical Channel Number of step one hundred episodes Evolutionary Strategy.

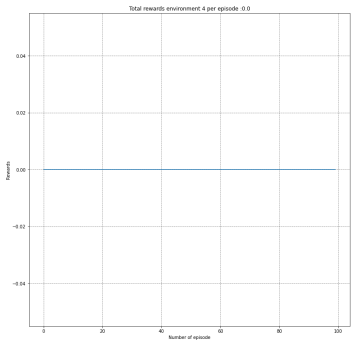


Fig. 4.37 4 digit Classical Channel Rewards one hundred episodes q-learning.

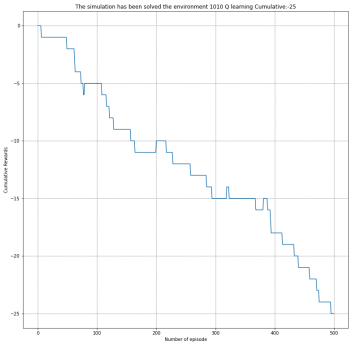


Fig. 4.38 4 digit Classical Channel Cumulative Reward one hundred episodes q-learning.

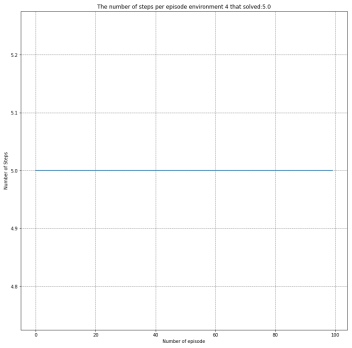


Fig. 4.39 4 digit Classical Channel Number of step one hundred episodes q-learning.

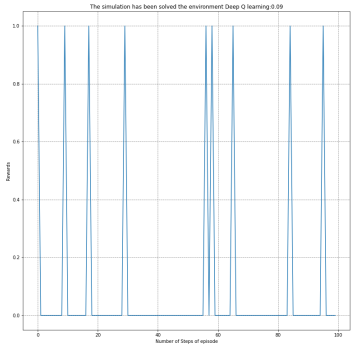


Fig. 4.40 4 digit Classical Channel Rewards one hundred episodes Deep q-learning.

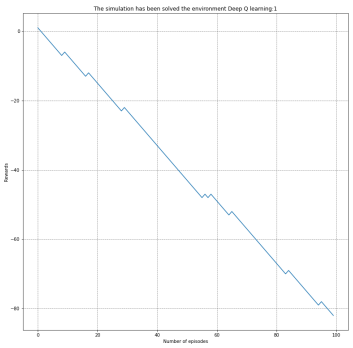


Fig. 4.41 4 digit Classical Channel Cumulative Reward one hundred episodes Deep q-learning.

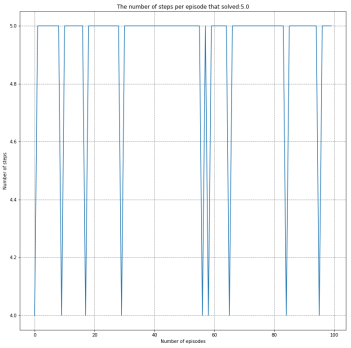


Fig. 4.42 4 digit Classical Channel Number of step one hundred episodes Deep q-learning.

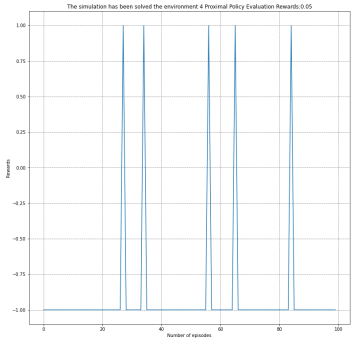


Fig. 4.43 4 digit Classical Channel Rewards one hundred episodes proximal policy optimization.

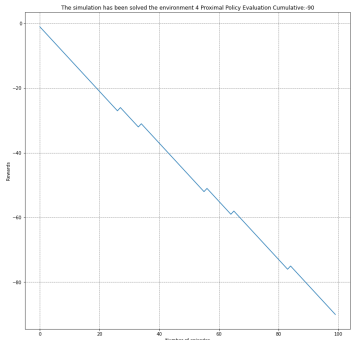


Fig. 4.44 4 digit Classical Channel Cumulative Reward one hundred episodes proximal policy optimization.

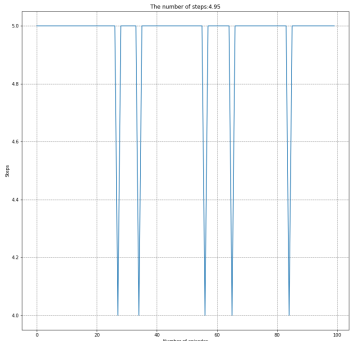


Fig. 4.45 4 digit Classical Channel Number of step one hundred episodes proximal policy optimization.

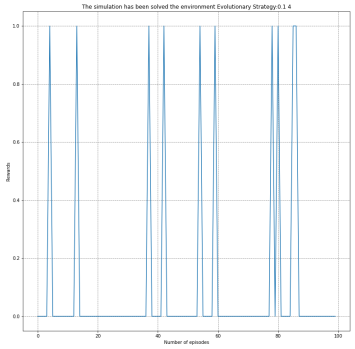


Fig. 4.46 4 digit Classical Channel Rewards one hundred episodes Evolutionary Strategy.

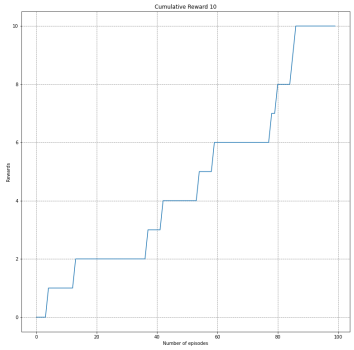


Fig. 4.47 4 digit Classical Channel Cumulative Reward one hundred episodes Evolutionary Strategy.

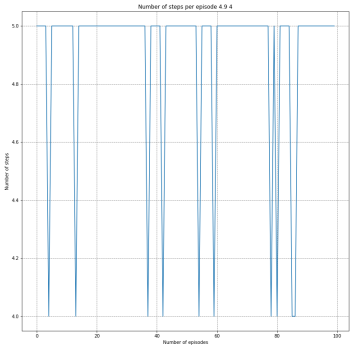


Fig. 4.48 4 digit Classical Channel Number of step one hundred episodes Evolutionary Strategy.

4.2.2 Multiagent Environment

I also examined the performance of artificial intelligence algorithms using one environment and multiple agents that can successfully finish the episode. For artificial intelligence algorithms was observed that the reward was 50% in contrast with one agent in the environment. For specific cases the agent did not performed better than one environment agent for the evolutionary strategy of message equal to four.
The multiagent environment of a classical channel the agent achieves to finish the episode with full reward except of the algorithm of deep q-learning and the mannwhitney test depicts that there is no correlation between the reward and mutual information of the classical channel.
For the case of two bits the multiagent approach is almost identical and the agent finishes with an average reward of 80% except for the case of proximal policy optimization algorithm that drops fifty percent.
As the size of the message increase the multiagent is not effective compared with smaller size of message. The total reward reduced by factor of two.
At last the transmission of four bits in a classical channel has the lowest score and party B recieves the initial message amlost 10% percent out of one hundred simulations.
For the quantum channel multiagent approach the results are almost identical with proximal policy optimization to do not perform for message size of one.
As the size of the key increase the number of simulation that the initial message transmitted correct reduces.
For the message size of four the q-learning outperforms. The mannwhitney test shows a correlation of the reward with the gate X bit flip for q-learning approach of message size of one and two and the proximal policy optimization of message size four. The value of the test depict the the multiagent approach takes in a consideration the error and has no impact on the results for small message size , in contrast when the number of digits that party A trasmits increases it effects the rewards that the agent recieves at the end of each episode with the highest correlation for digit four proximal policy optimization algorithm.

4.2.3 Quantum Channel

In the paper [3], an analytical expression of the quantum error was derived by the analyzing the contribution of the reward with respect the error. The quantum protocol makes use of the decode and decode function that encrypts the message and decrypts and produces some error to the decrypted message. It is noteworthy that message$<\frac{n}{1}$

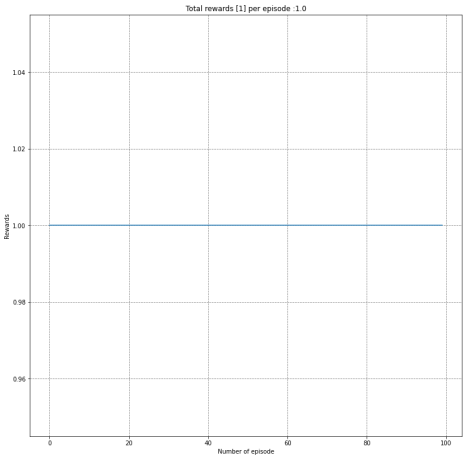


Fig. 4.49 MultiAgent two digits Rewards Q-learning.

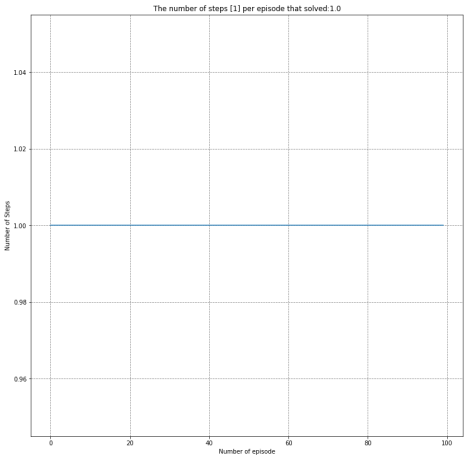


Fig. 4.50 MultiAgent two digits Steps Q-learning.

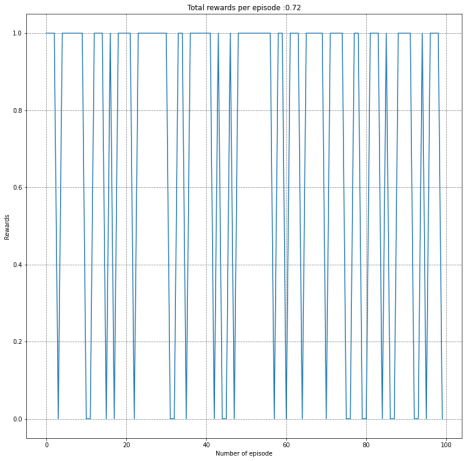


Fig. 4.51 MultiAgent four digits Rewards Q-learning.

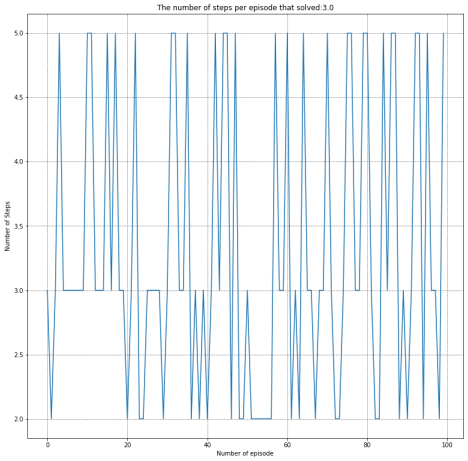


Fig. 4.52 MultiAgent four digits Steps Q-learning.

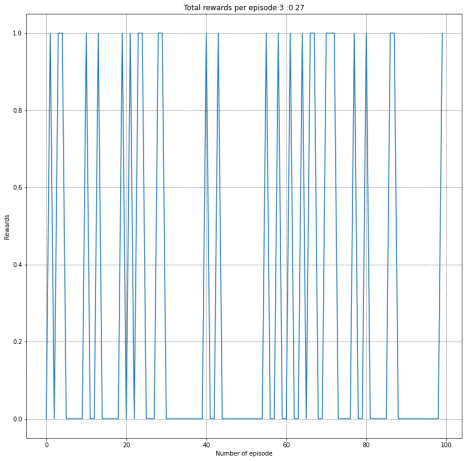


Fig. 4.53 MultiAgent eight digits Rewards Q-learning.

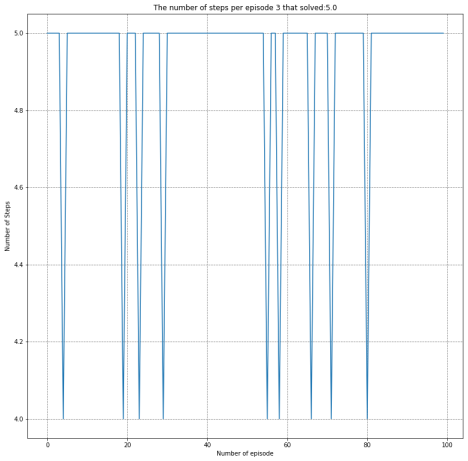


Fig. 4.54 MultiAgent eight digits Steps Q-learning.

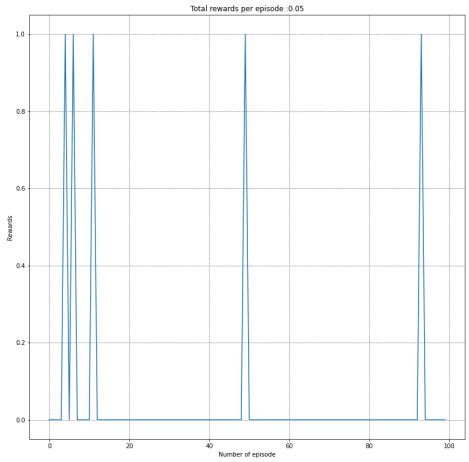


Fig. 4.55 MultiAgent sixteen digits Rewards Q-learning.

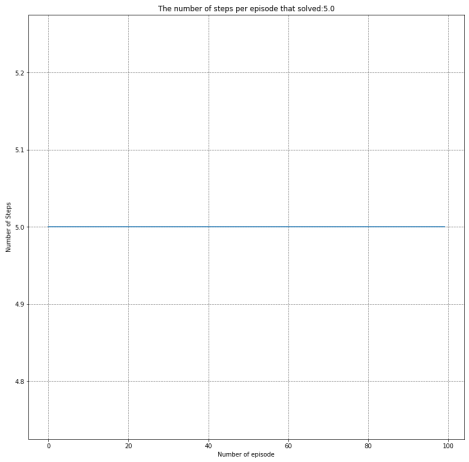


Fig. 4.56 MultiAgent sixteen digits Steps Q-learning.

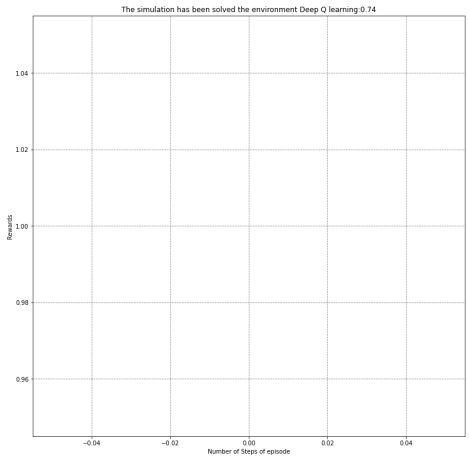


Fig. 4.57 MultiAgent two digits Rewards Deep Q-learning.

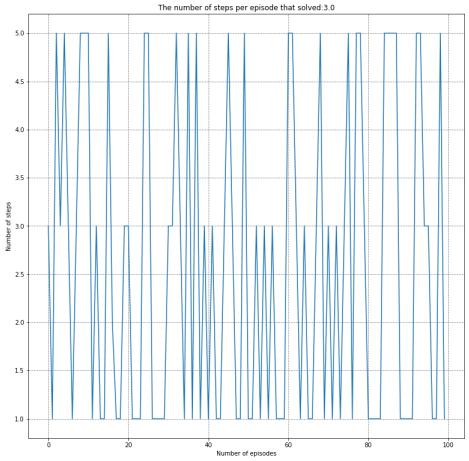


Fig. 4.58 MultiAgent two digits Steps Deep Q-learning.

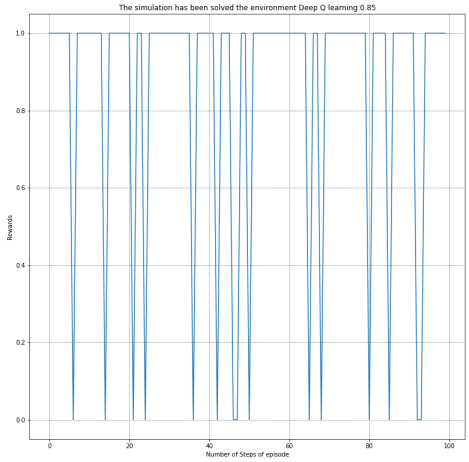


Fig. 4.59 MultiAgent four digits Rewards Deep Q-learning.

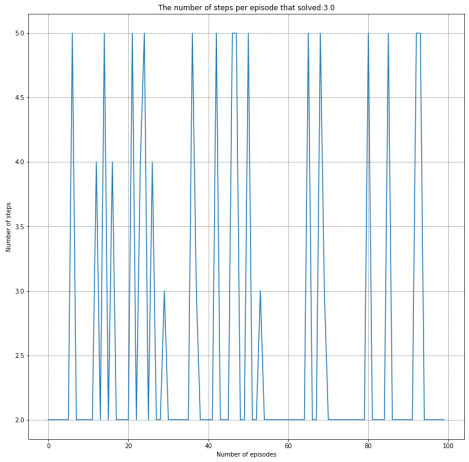


Fig. 4.60 MultiAgent four digits Steps Deep Q-learning.

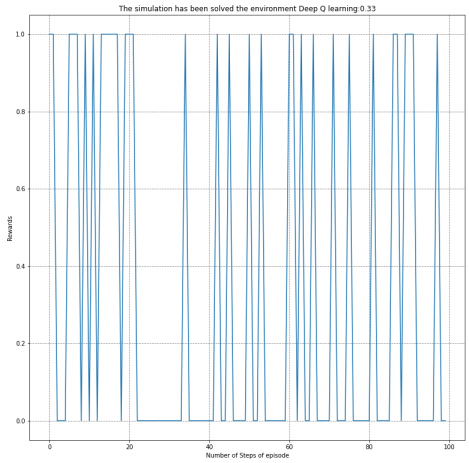


Fig. 4.61 MultiAgent eight digits Rewards Deep Q-learning.

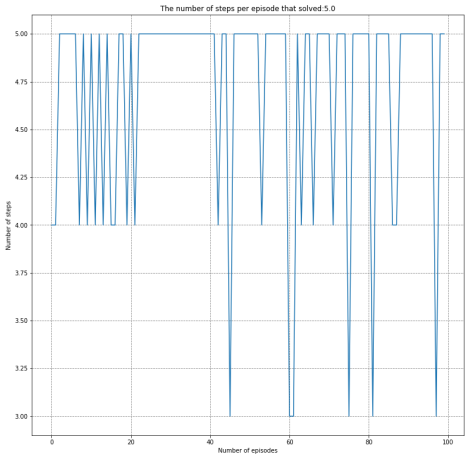


Fig. 4.62 MultiAgent eight digits Steps Deep Q-learning.

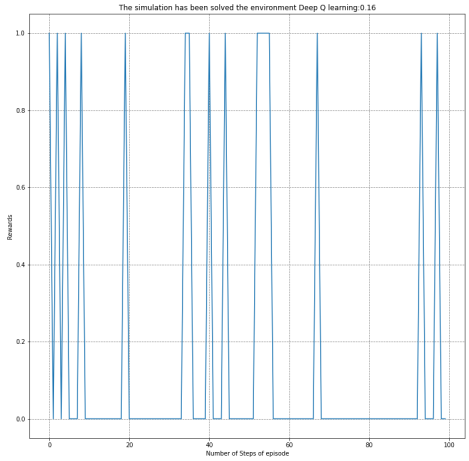


Fig. 4.63 MultiAgent sixteen digits Rewards Deep Q-learning.

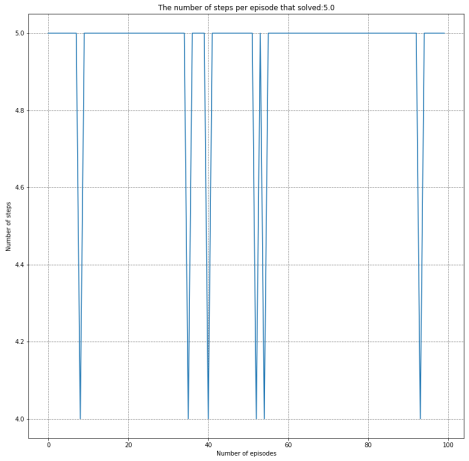


Fig. 4.64 MultiAgent sixteen digits Steps Deep Q-learning.

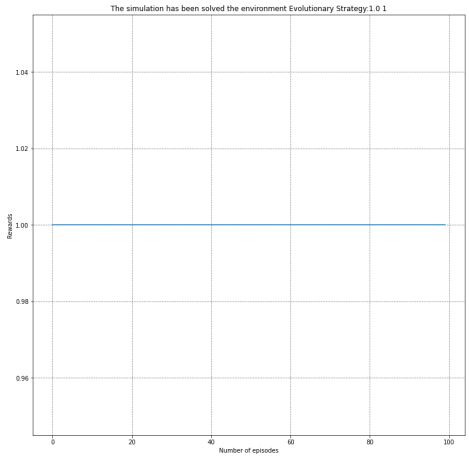


Fig. 4.65 MultiAgent two digits Rewards Evolutionary Strategy.

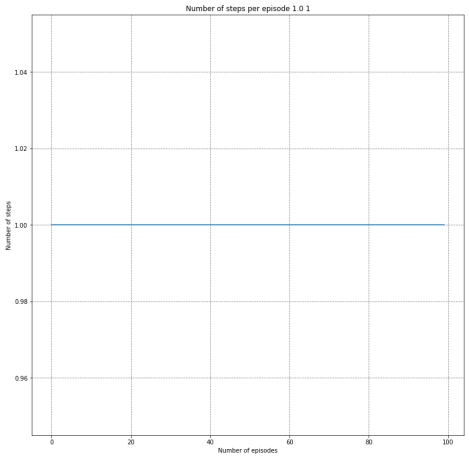


Fig. 4.66 MultiAgent two digits Steps Evolutionary Strategy.

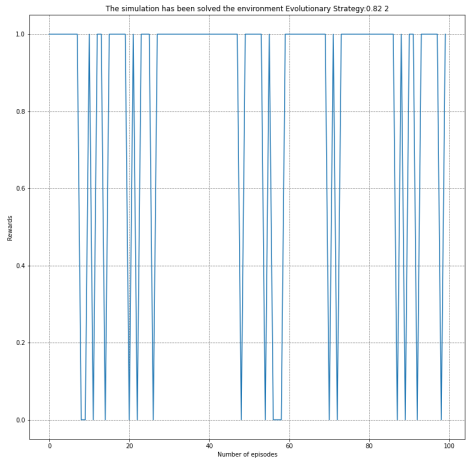


Fig. 4.67 MultiAgent four digits Rewards Evolutionary Strategy.

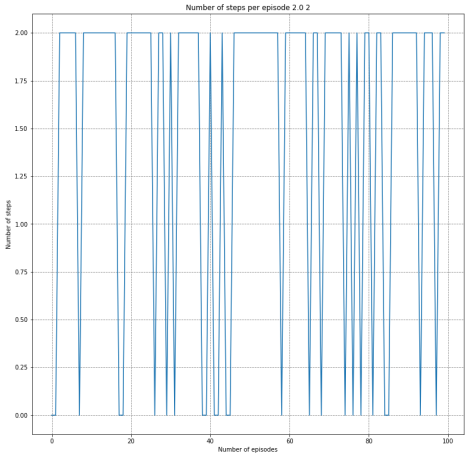


Fig. 4.68 MultiAgent four digits Steps Evolutionary Strategy.

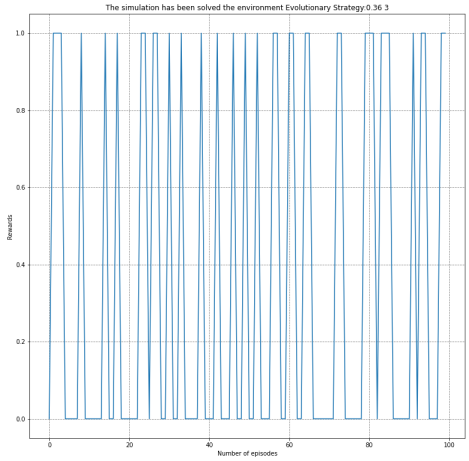


Fig. 4.69 MultiAgent eight digits Rewards Evolutionary Strategy.

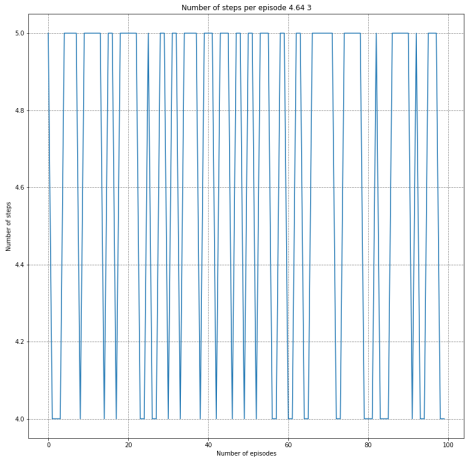


Fig. 4.70 MultiAgent eight digits Steps Evolutionary Strategy.

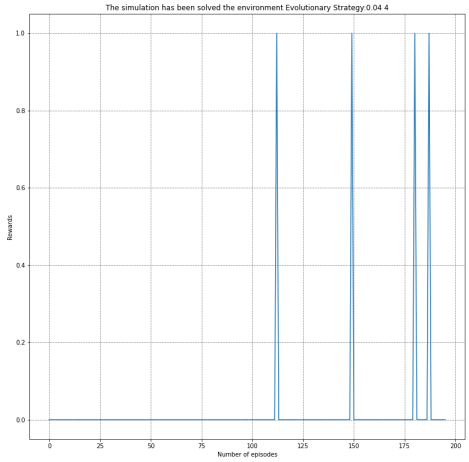


Fig. 4.71 MultiAgent sixteen digits Rewards Evolutionary Strategy.

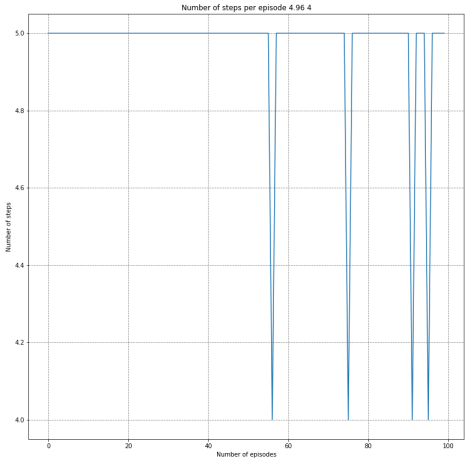


Fig. 4.72 MultiAgent sixteen digits Steps Evolutionary Strategy.

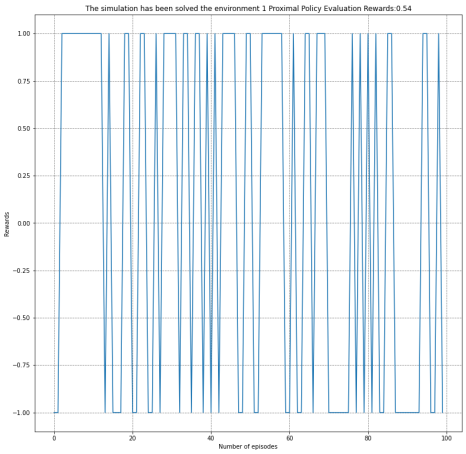


Fig. 4.73 MultiAgent two digits Rewards Proximal Policy Optimization.

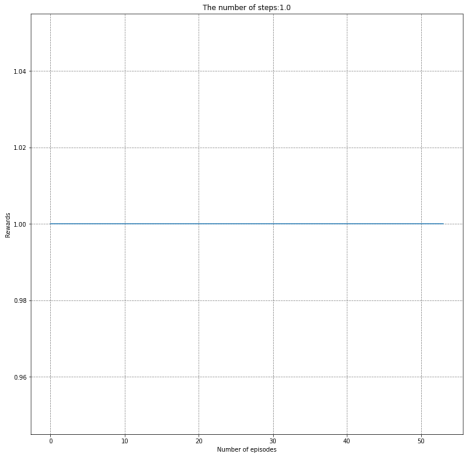


Fig. 4.74 MultiAgent two digits Steps Proximal Policy Optimization.

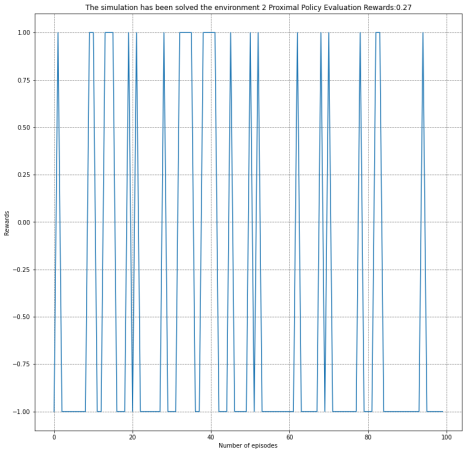


Fig. 4.75 MultiAgent four digits Rewards Proximal Policy Optimization.

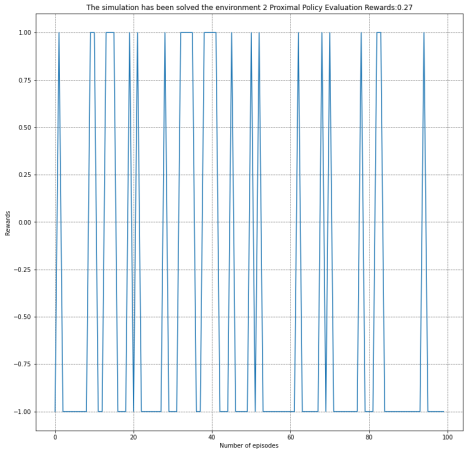


Fig. 4.76 MultiAgent four digits Steps Proximal Policy Optimization.

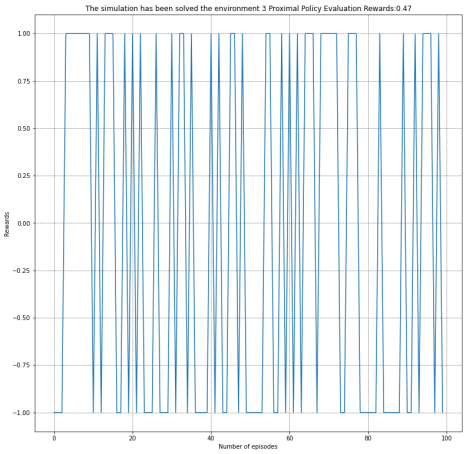


Fig. 4.77 MultiAgent eight digits Rewards Proximal Policy Optimization.

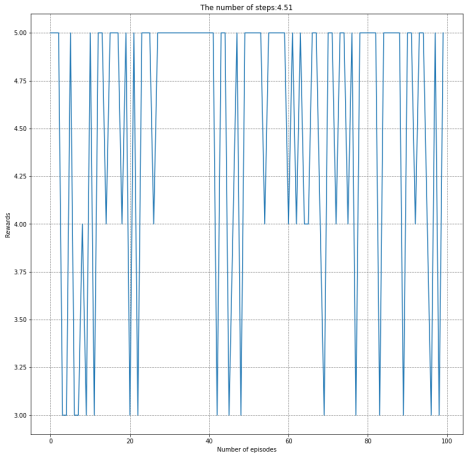


Fig. 4.78 MultiAgent eight digits Steps Proximal Policy Optimization.

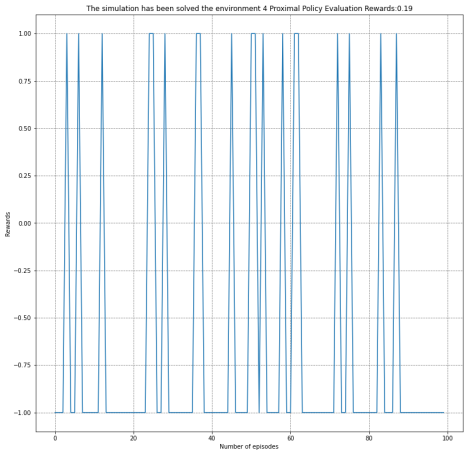


Fig. 4.79 MultiAgent sixteen digits Rewards Proximal Policy Optimization.

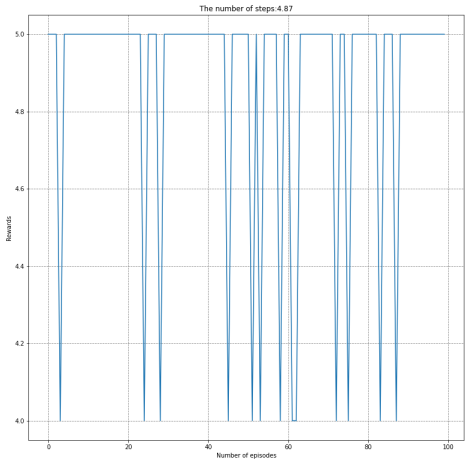


Fig. 4.80 MultiAgent sixteen digits Steps Proximal Policy Optimization.

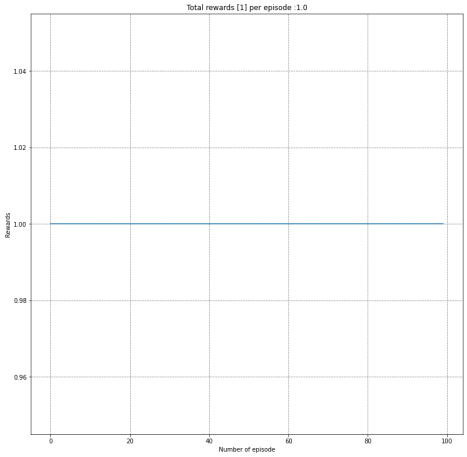


Fig. 4.81 MultiAgent two digits Rewards Q-learning.

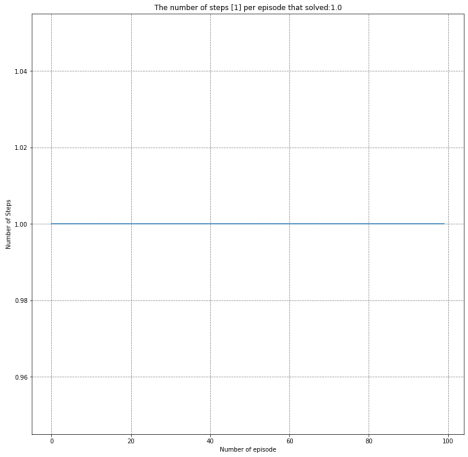


Fig. 4.82 MultiAgent two digits Steps Q-learning.

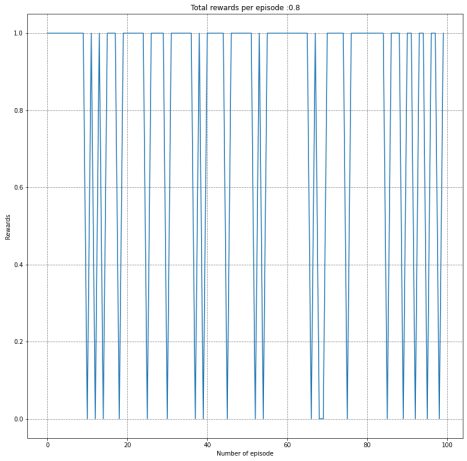


Fig. 4.83 MultiAgent four digits Rewards Q-learning.

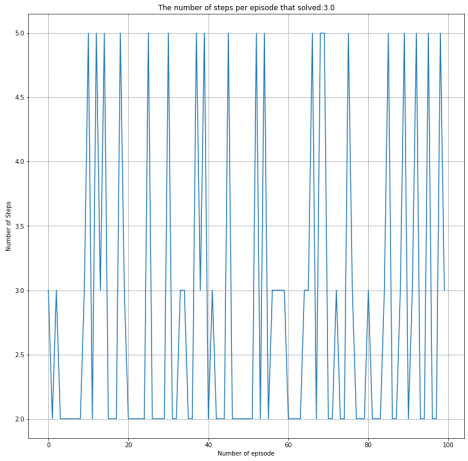


Fig. 4.84 MultiAgent four digits Steps Q-learning.

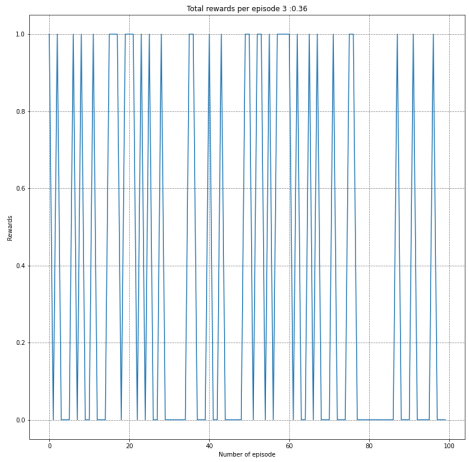


Fig. 4.85 MultiAgent eight digits Rewards Q-learning.

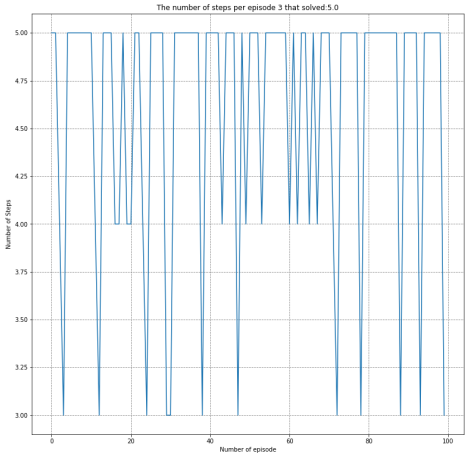


Fig. 4.86 MultiAgent eight digits Steps Q-learning.

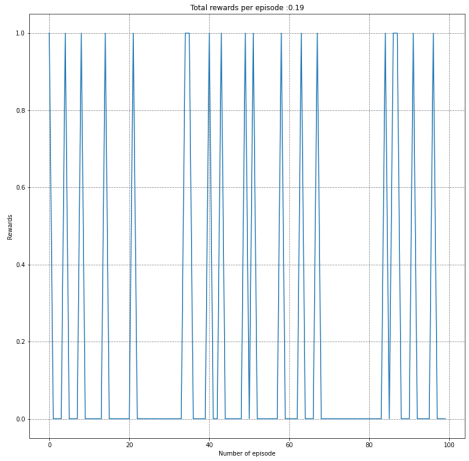


Fig. 4.87 MultiAgent sixteen digits Rewards Q-learning.

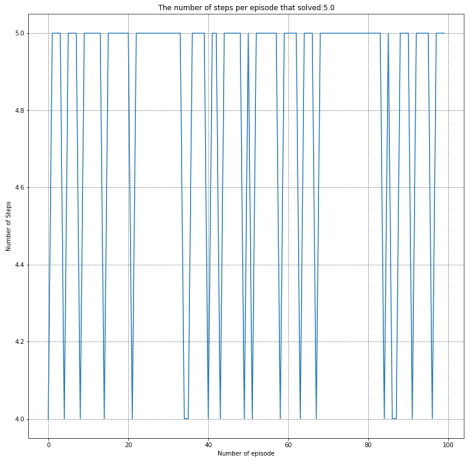


Fig. 4.88 MultiAgent sixteen digits Steps Q-learning.

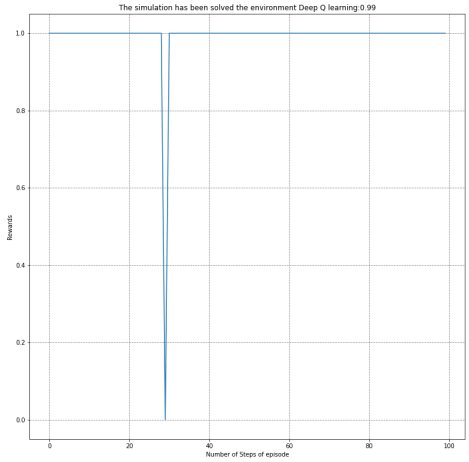


Fig. 4.89 MultiAgent two digits Rewards Deep Q-learning.

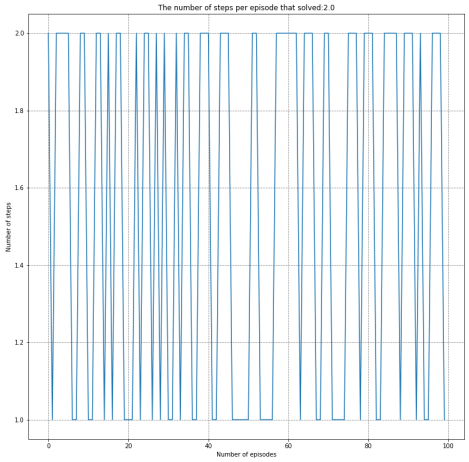


Fig. 4.90 MultiAgent two digits Steps Deep Q-learning.

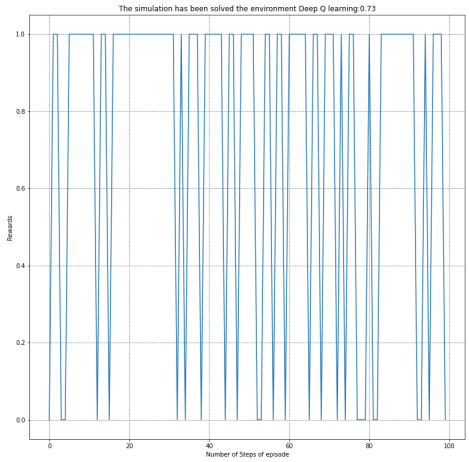


Fig. 4.91 MultiAgent four digits Rewards Deep Q-learning.

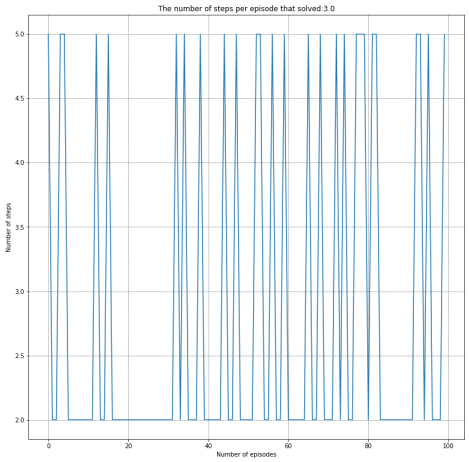


Fig. 4.92 MultiAgent four digits Steps Deep Q-learning.

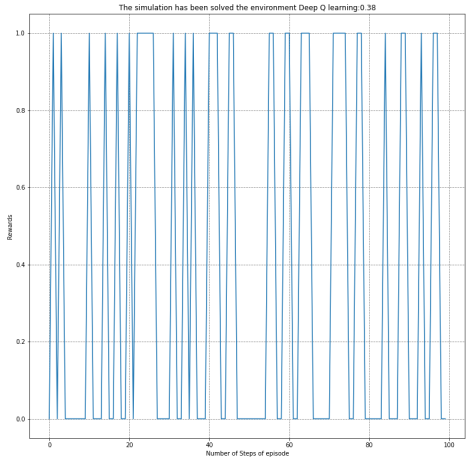


Fig. 4.93 MultiAgent eight digits Rewards Deep Q-learning.

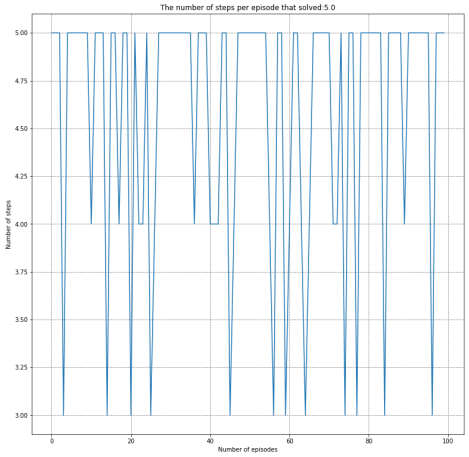


Fig. 4.94 MultiAgent eight digits Steps Deep Q-learning.

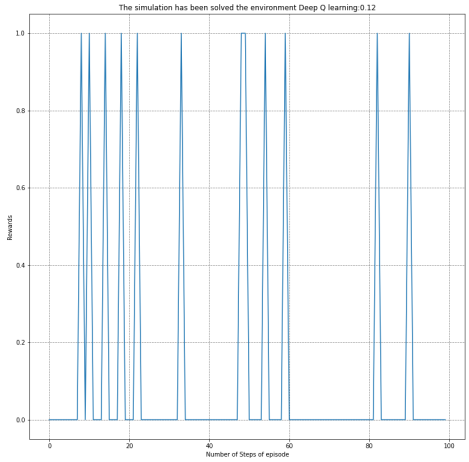


Fig. 4.95 MultiAgent sixteen digits Rewards Deep Q-learning.

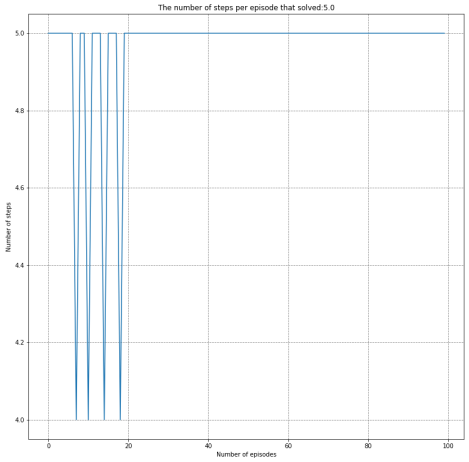


Fig. 4.96 MultiAgent sixteen digits Steps Deep Q-learning.

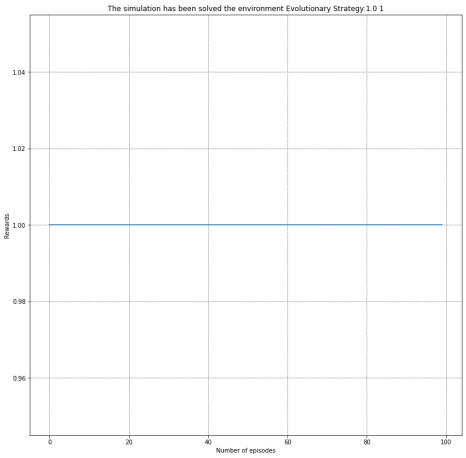


Fig. 4.97 MultiAgent two digits Rewards Evolutionary Strategy.

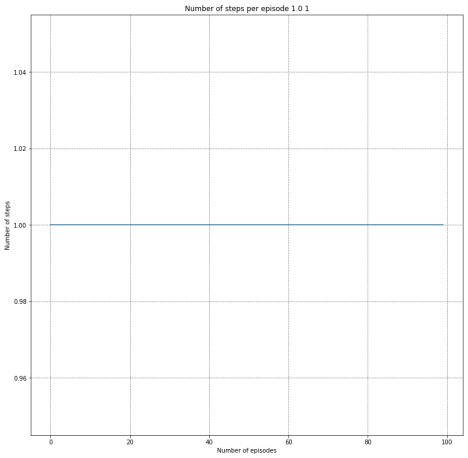


Fig. 4.98 MultiAgent two digits Steps Evolutionary Strategy.

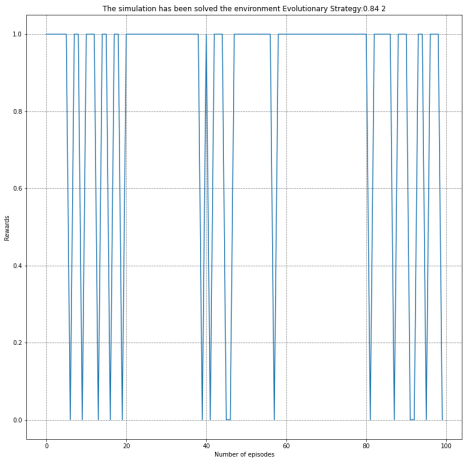


Fig. 4.99 MultiAgent four digits Rewards Evolutionary Strategy.

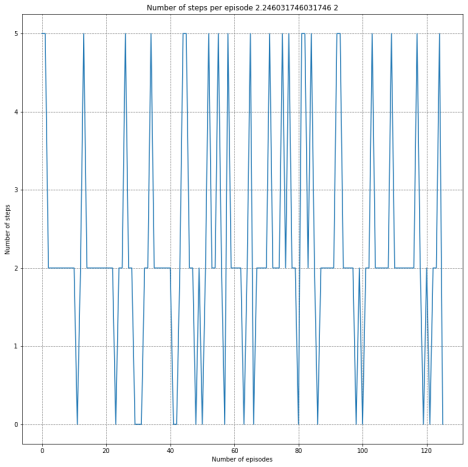


Fig. 4.100 MultiAgent four digits Steps Evolutionary Strategy.

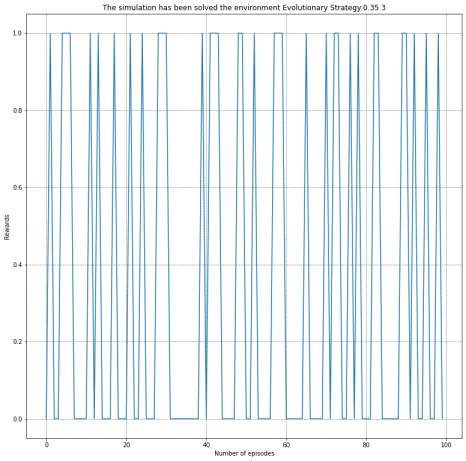


Fig. 4.101 MultiAgent eight digits Rewards Evolutionary Strategy.

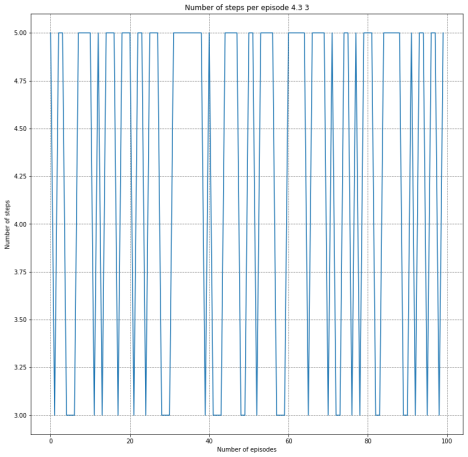


Fig. 4.102 MultiAgent eight digits Steps Evolutionary Strategy.

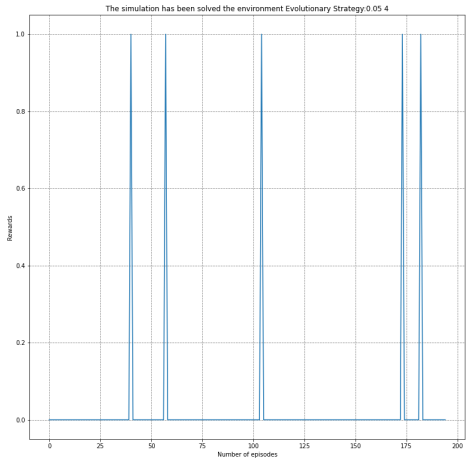


Fig. 4.103 MultiAgent sixteen digits Rewards Evolutionary Strategy.

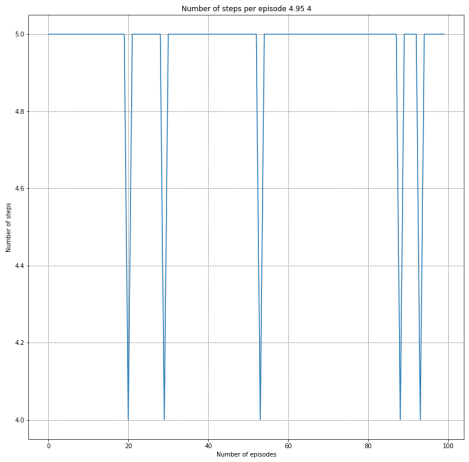


Fig. 4.104 MultiAgent sixteen digits Steps Evolutionary Strategy.

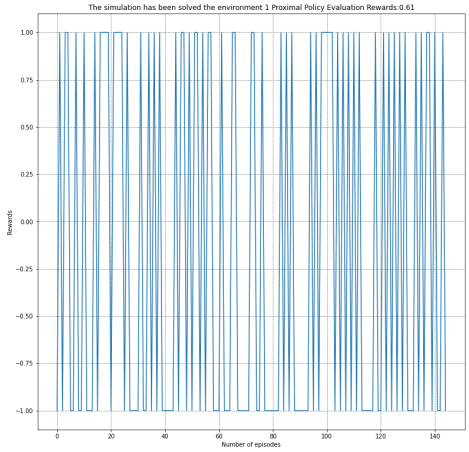


Fig. 4.105 MultiAgent two digits Rewards Proximal Policy Optimization.

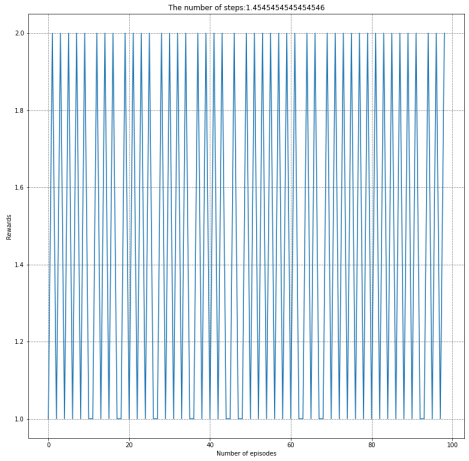


Fig. 4.106 MultiAgent two digits Steps Proximal Policy Optimization.

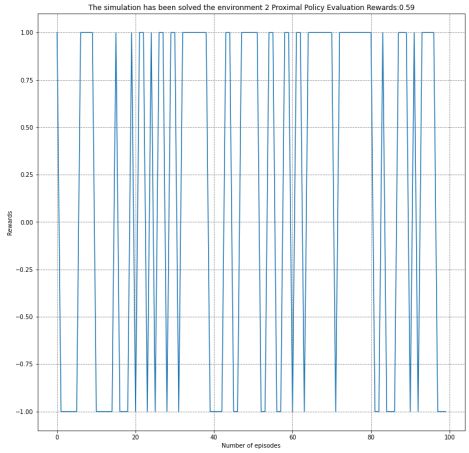


Fig. 4.107 MultiAgent four digits Rewards Proximal Policy Optimization.

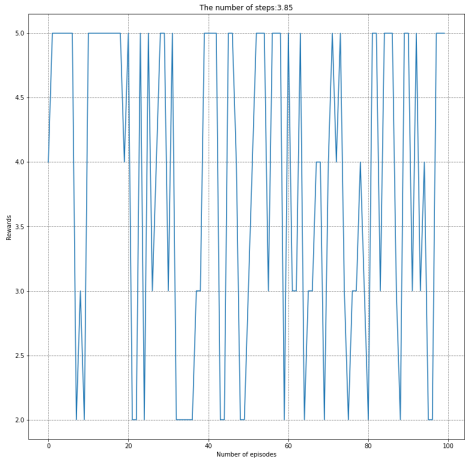


Fig. 4.108 MultiAgent four digits Steps Proximal Policy Optimization.

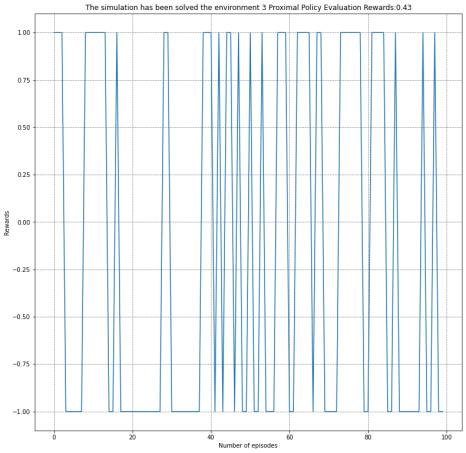


Fig. 4.109 MultiAgent eight digits Rewards Proximal Policy Optimization.

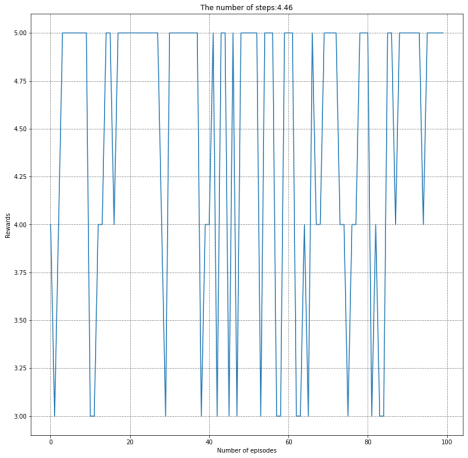


Fig. 4.110 MultiAgent eight digits Steps Proximal Policy Optimization.

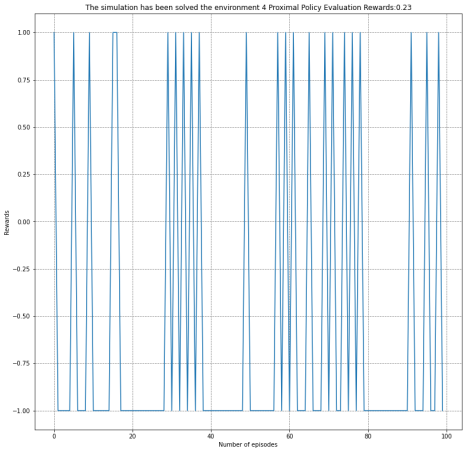


Fig. 4.111 MultiAgent sixteen digits Rewards Proximal Policy Optimization.

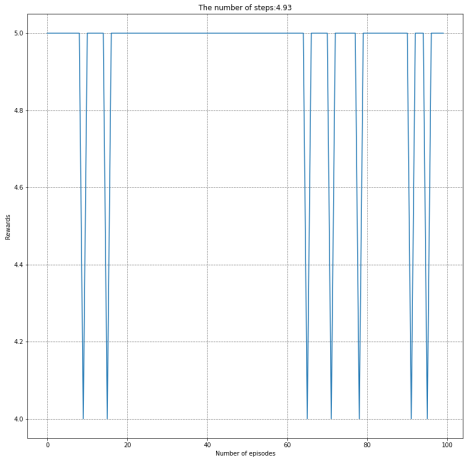


Fig. 4.112 MultiAgent sixteen digits Steps Proximal Policy Optimization.

we indeed find that the deep q-learning algorithm finds that highest peak $M_{message}$ bound. For all the approaches the rewards decreases as the size of the message increases. The drop is not steady and for the proximal policy optimization algorithm there is a slightly increase for message length of three and four. The q-learning does not perform for message size higher than three and the performance deep q-learning is almost identical. The mannwhitney test shows that there is a correlation of the decode error and the result of the q-learning and evolutionary strategy results, in contrast with deep q-learning and proximal policy optimization algorithm. The q-learning results for $message = 1$ have shown strong correlation.

As can be seen, there is a stark contrast between the classical channel and the quantum channel for the rewards. This is because the latter channel has the encode process and the optimal reward does not increase as for the case of the classical channel. For the most cases of the artificial intelligence algorithms the q-learning, proximal policy optimization,deep q-learning and evolutionary strategy the average reward flactuate from 50% to 10% except for the proximal policy optimization algorithm that for message=4 slightly increase. In the classical channnel it can be observed that the reward for $message = 4$ performs better than the quantum approach.

The qiskit official approach that measure the error in each bit shows that the number of errors are symmetric with the message length and are not usually less than 50% even when the message length is fairly large.

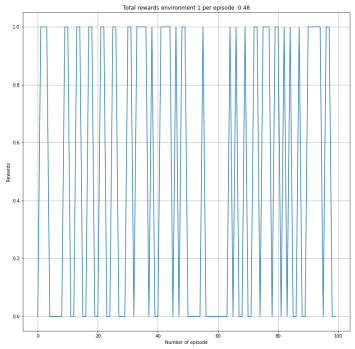


Fig. 4.113 One digit Q-learning The reward on the test set Quantum Channel.

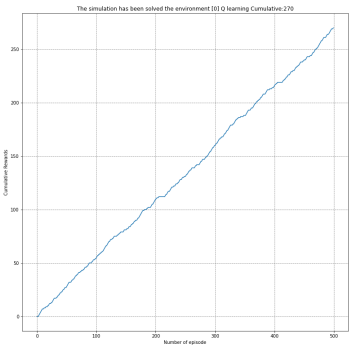


Fig. 4.114 One digit Q-learning cumulative on the test set Quantum Channel.

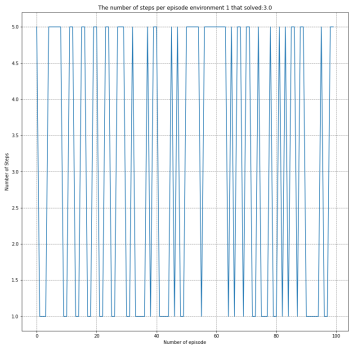


Fig. 4.115 One digit Q-learning agent steps on the test set Quantum Channel.

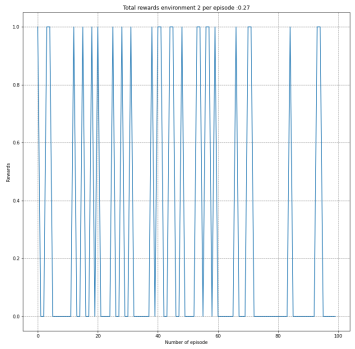


Fig. 4.116 Two digits Q-learning The reward on the test set Quantum Channel.

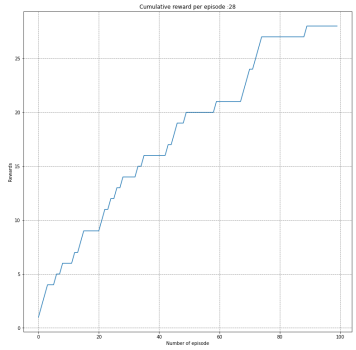


Fig. 4.117 Two digits Q-learning cumulative on the test set Quantum Channel.

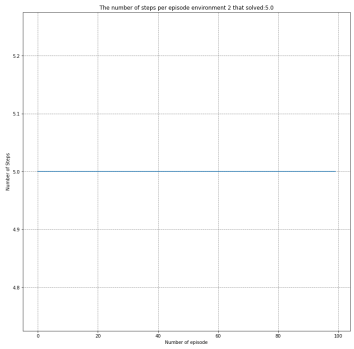


Fig. 4.118 Two digits Q-learning agent steps on the test set Quantum Channel.

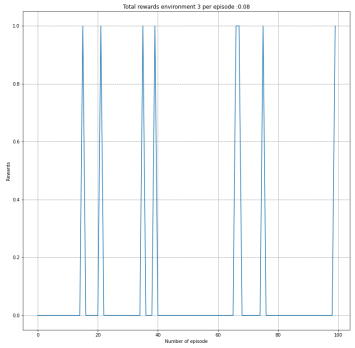


Fig. 4.119 Three digits Q-learning The reward on the test set Quantum Channel.

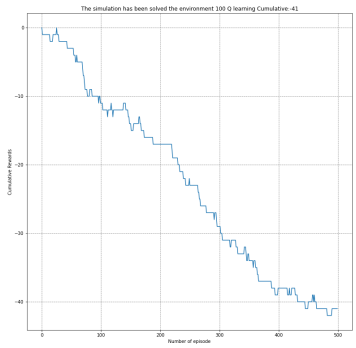


Fig. 4.120 Three digits Q-learning cumulative on the test set Quantum Channel.

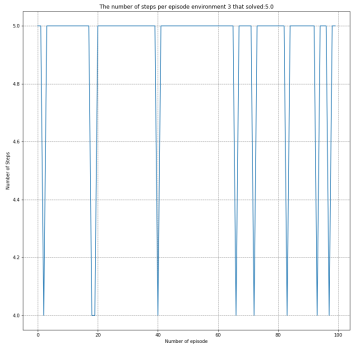


Fig. 4.121 Three digits Q-learning agent steps on the test set Quantum Channel.

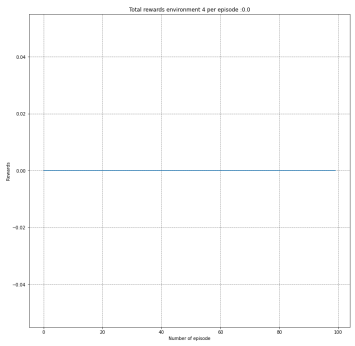


Fig. 4.122 Four digits Q-learning The reward on the test set Quantum Channel.

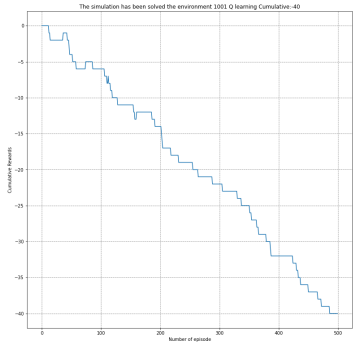


Fig. 4.123 Four digits Q-learning cumulative on the test set Quantum Channel.

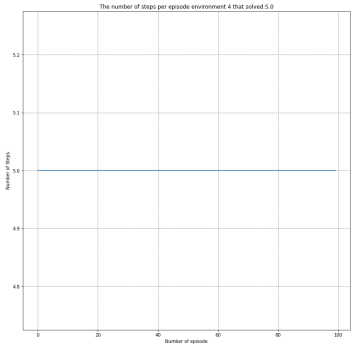


Fig. 4.124 Four digits Q-learning agent steps on the test set Quantum Channel.

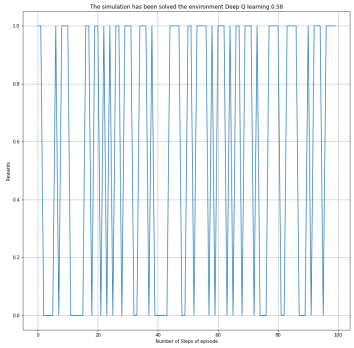


Fig. 4.125 One digits Deep Q-learning The reward on the test set Quantum Channel.

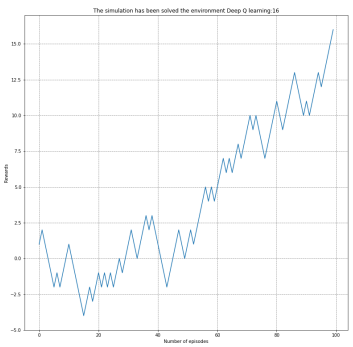


Fig. 4.126 One digits Deep Q-learning cumulative on the test set Quantum Channel.

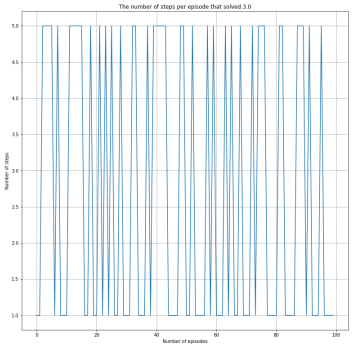


Fig. 4.127 One digits Deep Q-learning agent steps on the test set Quantum Channel.

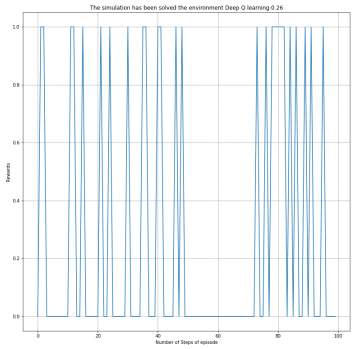


Fig. 4.128 Two digits Deep Q-learning The reward on the test set Quantum Channel.

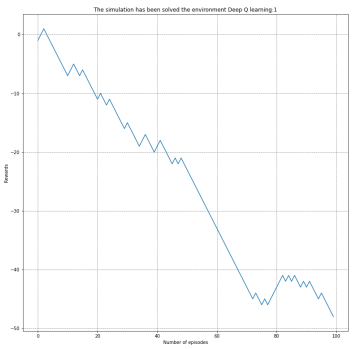


Fig. 4.129 Two digits Deep Q-learning cumulative on the test set Quantum Channel.

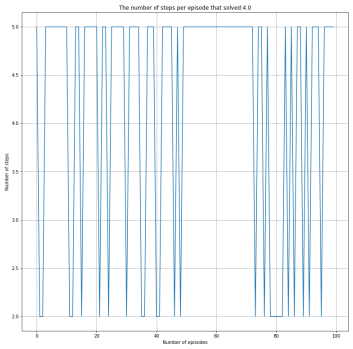


Fig. 4.130 Two digits Deep Q-learning agent steps on the test set Quantum Channel.

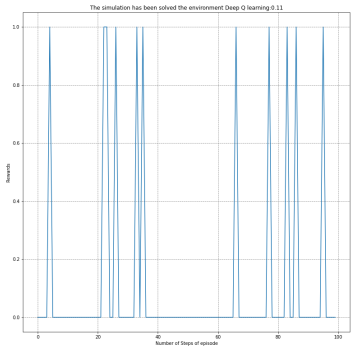


Fig. 4.131 Three digits Deep Q-learning The reward on the test set Quantum Channel.

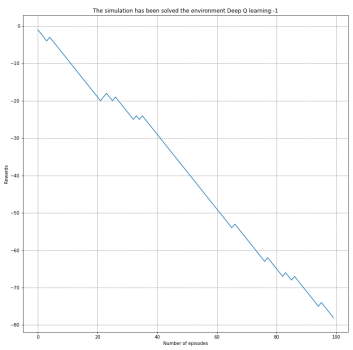


Fig. 4.132 Three digits Deep Q-learning cumulative on the test set Quantum Channel.

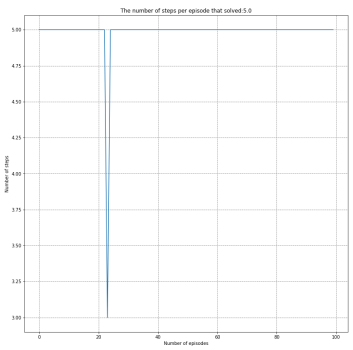


Fig. 4.133 Three digits Deep Q-learning agent steps on the test set Quantum Channel.

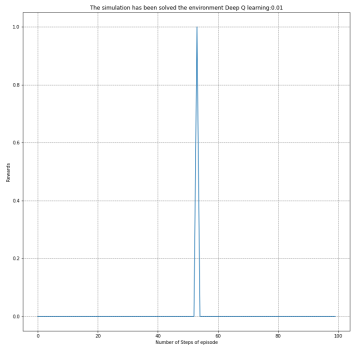


Fig. 4.134 Four digits Deep Q-learning The reward on the test set Quantum Channel.

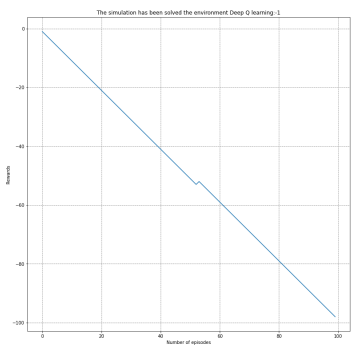


Fig. 4.135 Four digits Deep Q-learning cumulative on the test set Quantum Channel.

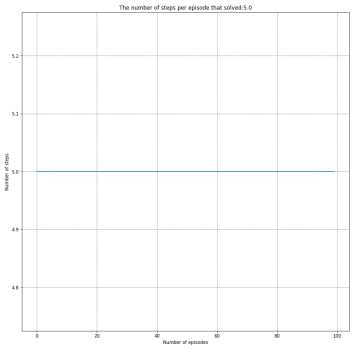


Fig. 4.136 Four digits Deep Q-learning agent steps on the test set Quantum Channel.

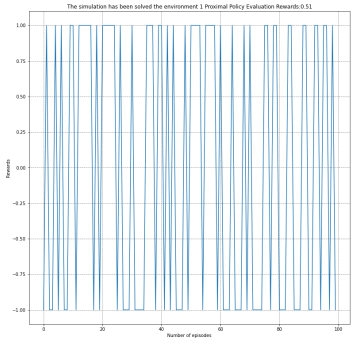


Fig. 4.137 One digits proximal policy optimization The reward on the test set Quantum Channel.

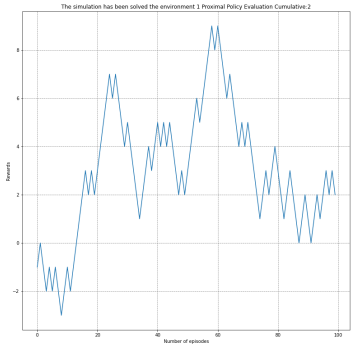


Fig. 4.138 One digits Deep proximal policy optimization cumulative on the test set Quantum Channel.

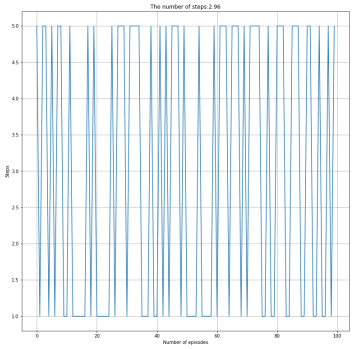


Fig. 4.139 One digits proximal policy optimization agent steps on the test set Quantum Channel.

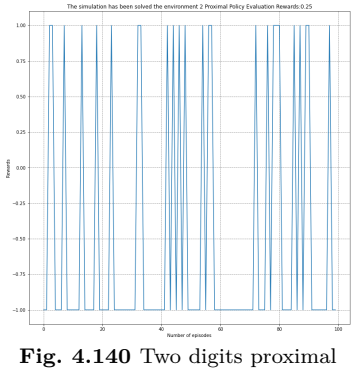


Fig. 4.140 Two digits proximal policy optimization The reward on the test set Quantum Channel.

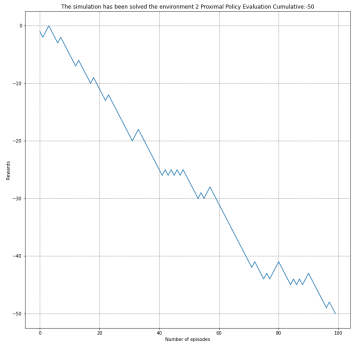


Fig. 4.141 Two digits Deep proximal policy optimization cumulative on the test set Quantum Channel.

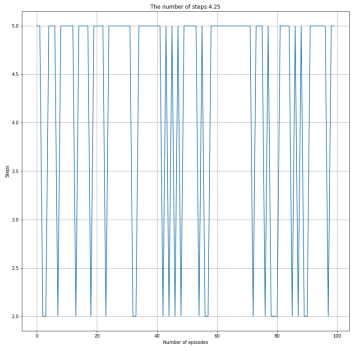


Fig. 4.142 Two digits proximal policy optimization agent steps on the test set Quantum Channel.

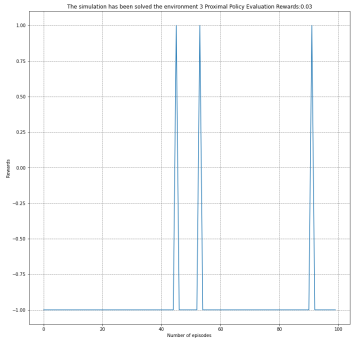


Fig. 4.143 Three digits proximal policy optimization The reward on the test set Quantum Channel.

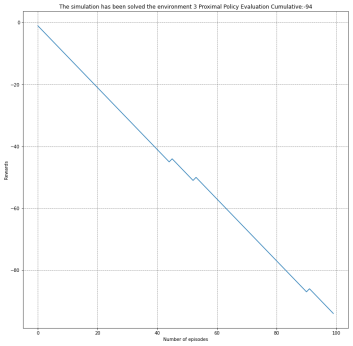


Fig. 4.144 Three digits Deep proximal policy optimization cumulative on the test set Quantum Channel.

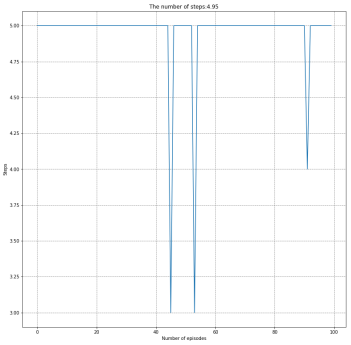


Fig. 4.145 Three digits proximal policy optimization agent steps on the test set Quantum Channel.

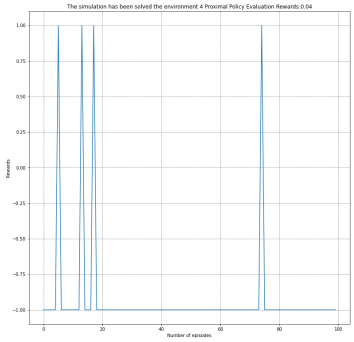


Fig. 4.146 Four digits proximal policy optimization The reward on the test set Quantum Channel.

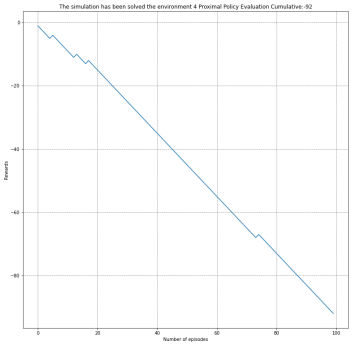


Fig. 4.147 Four digits Deep proximal policy optimization cumulative on the test set Quantum Channel.

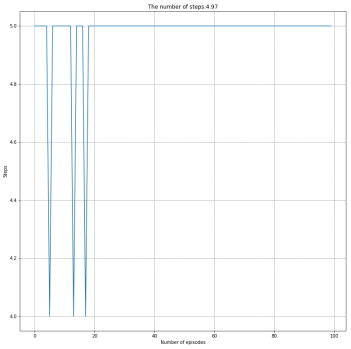


Fig. 4.148 Four digits proximal policy optimization agent steps on the test set Quantum Channel.

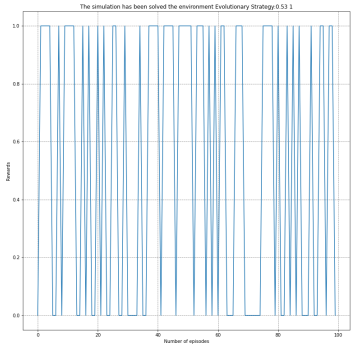


Fig. 4.149 One digits evolutionary strategy the reward on the test set Quantum Channel.

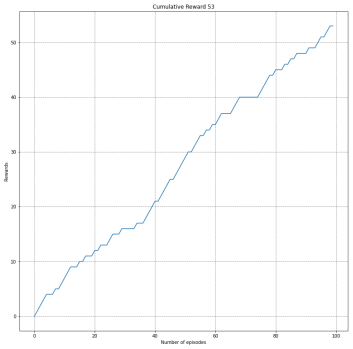


Fig. 4.150 One digits evolutionary strategy cumulative on the test set Quantum Channel.

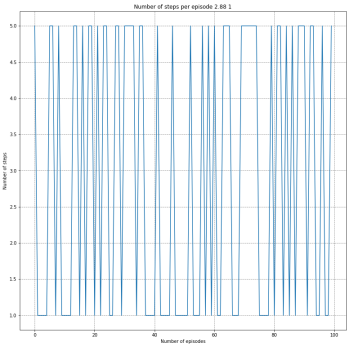


Fig. 4.151 One digits evolutionary strategy agent steps on the test set Quantum Channel.

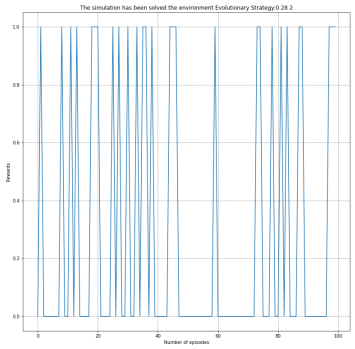


Fig. 4.152 Two digits evolutionary strategy the reward on the test set Quantum Channel.

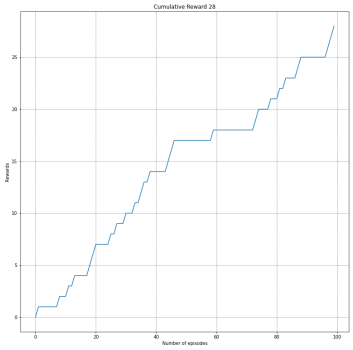


Fig. 4.153 Two digits evolutionary strategy cumulative on the test set Quantum Channel.

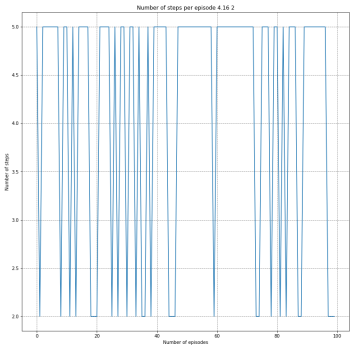


Fig. 4.154 Two digits evolutionary strategy agent steps on the test set Quantum Channel.

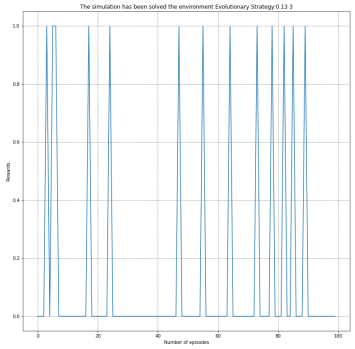


Fig. 4.155 Three digits evolutionary strategy the reward on the test set Quantum Channel.

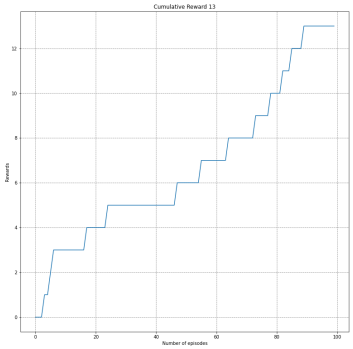


Fig. 4.156 Three digits evolutionary strategy cumulative on the test set Quantum Channel.

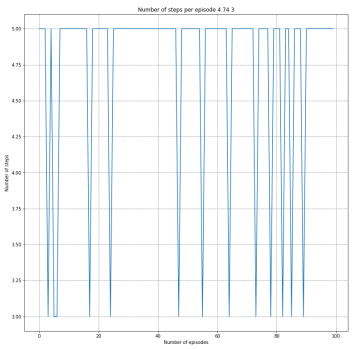


Fig. 4.157 Three digits evolutionary strategy agent steps on the test set Quantum Channel.

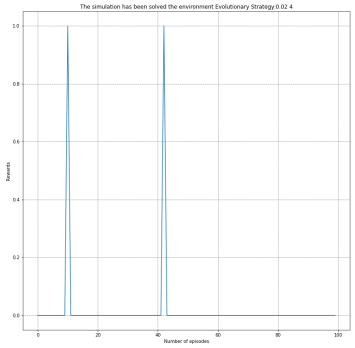


Fig. 4.158 Four digits evolutionary strategy the reward on the test set Quantum Channel.

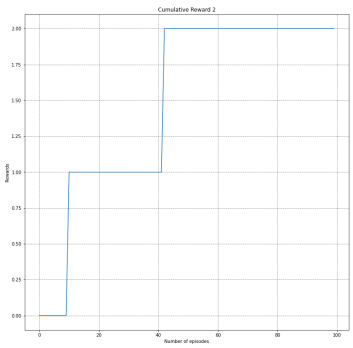


Fig. 4.159 Four digits evolutionary strategy cumulative on the test set Quantum Channel.

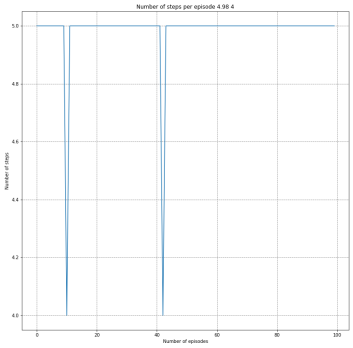


Fig. 4.160 Four digits evolutionary strategy agent steps on the test set Quantum Channel.

		Reward	Cumulative	Steps	Fidelity	Mannwhitney
1bit	Q learning	54	41	3.0	54	0.19155
	Deep q-learning	44	-12	3.24	44	0.12014
	Proximal policy Optimization	50	0	3.0	50	0.92297
	Evolutionary Strategy	42	42	3.32	42	1.43857
2bit	Q-learning	27	26	5.0	27	0.414895
	Deep q-learning	18	9	4.6086	9	0.006
	Proximal policy Optimization	21	-58	4.37	21	0.08398
	Evolutionary Strategy	22	22	4.34	22	0.516161
3bit	Q-learning	9	9	4.91	9	0.676259
	Deep q-learning	8	-84	4.84	8	1.807
	Proximal policy Optimization	12	-76	4.77	12	0.000769151
	Evolutionary Strategy	13	13	4.74	13	1.84026549
4bit	Q-learning	0	0	5	0	0
	Deep q-learning	9	-82	4.91	9	4.794256
	Proximal Policy Optimization	21	-58	4.37	21	0.0839866
	Evolutionary Strategy	10	10	4.9	10	6.347089804

Table 4.1 Classical Channel reinforcement learning algorithms single agent

		Reward	Cumulative	Steps	Fidelity	Mannwhitney
1bit	Q learning	49	49	3.4	49	0.17617144126966355
	Deep q-learning	58	16	2.68	58	0.10572308259511973
	Proximal policy Optimization	53	6	3.3	53	0.9890373646690378
	Evolutionary Strategy	53	53	3.32	53	0.1431405895735664
2bit	Q-learning	19	19	5.0	19	0.005535759906728811
	Deep q-learning	26	9	4.3125	26	0.623604884395689
	Proximal policy Optimization	25	-50	4.25	25	0.21048870772338102
	Evolutionary Strategy	28	28	4.34	28	0.3905365583512
3bit	Q-learning	8	8	4.95	8	0.5078993408515471
	Deep q-learning	11	-1	4.88	11	0.6371548401289073
	Proximal policy Optimization	3	-94	4.95	3	0.47306541409170877
	Evolutionary Strategy	13	13	4.74	13	0.6014623127632203
4bit	Q-learning	0	0	5	0	1.5297331869789446e-23
	Deep q-learning	1	1	5.0	1	0.9436280222029834
	Proximal Policy Optimization	4	-92	4.97	4	0.4806634786884194
	Evolutionary Strategy	2	2	4.98	2	1.5297331869789446e-23

Table 4.2 Quantum Channel reinforcement learning algorithms single agent

		Reward	Cumulative	Steps	Fidelity	Mannwhitney
1bit	Q learning	1.0	1.0	1.0	1.0	1.5297331869789446e-23
	Deep q-learning	0.74	48	2.57	64	0.4795001221869535
	Proximal policy Optimization	1.0	1.0	1.0	1.0	1.5297331869789446e-23
	Evolutionary Strategy	1.0	1.0	1.0	1.0	1.5297331869789446e-23
2bit	Q-learning	0.72	0.72	3.28	72	0.028777069190506727
	Deep q-learning	0.85	70	2.57	85	0.001581185931929
	Proximal policy Optimization	27	-46	5.0	27	0.16138832540104264
	Evolutionary Strategy	0.82	82	2.0	82	0.004232423251583236
3bit	Q-learning	0.27	0.27	4.91	9	0.4026103484975887
	Deep q-learning	0.33	-34	4.71	23	0.3330419506998432
	Proximal policy Optimization	0.47	-53	4.51	34	0.8791274366754668
	Evolutionary Strategy	0.36	36	4.64	36	0.32207230024700484
4bit	Q-learning	0.05	0.05	5.0	0	0.7714996467790758
	Deep q-learning	0.16	-68	4.94	6	0.5525928650794093
	Proximal Policy Optimization	0.23	-54	4.93	7	0.08398660900901828
	Evolutionary Strategy	0.04	4	4.96	4	0.8586143783296402

Table 4.3 Classical Channel reinforcement learning algorithms Multiagent

		Reward	Cumulative	Steps	Fidelity	Mannwhitney
1bit	Q learning	1.0	1.0	1.0	1.0	1.5297331869789446e-23
	Deep q-learning	1.0	1.0	1.56	1.0	0.4795001221869535
	Proximal policy Optimization	0.61	-23	1.4545	73	0.10334248173044111
	Evolutionary Strategy	1.0	1.0	1.0	1.0	1.5297331869789446e-23
2bit	Q-learning	0.80	80	2.82	80	0.028777069190506727
	Deep q-learning	0.73	46	2.81	73	0.4026103484975887
	Proximal policy Optimization	0.59	18	3.85	53	0.6237995754427687
	Evolutionary Strategy	0.84	84	2.42	84	0.5525928650794093
3bit	Q-learning	0.36	36	4.61	36	0.3006238697336051
	Deep q-learning	0.38	-24	4.59	30	0.07922624425610855
	Proximal policy Optimization	0.43	-57	4.46	36	0.7068593291094063
	Evolutionary Strategy	0.35	35	4.3	35	0.3112674893053171
4bit	Q-learning	0.19	0.19	4.81	19	0.507899340851547
	Deep q-learning	0.12	-76	4.96	5	0.6189426417174644
	Proximal Policy Optimization	0.14	-72	4.95	6	3.656580204652324e-24
	Evolutionary Strategy	0.05	5	4.95	5	0.8368518855873175

Table 4.4 Quantum Channel reinforcement learning algorithms Multiagent

Message length	percentage of identical key
32 bit	0.656
64 bit	0.578
128 bit	0.585

Table 4.5 Quantum Channel Qiskit

Chapter 5

Discussion

This chapter provides details of the project. A summary of the project in the first Section. Limitations, implications during the implementation Section 5.2, Section 5.3 respectively. At future work in Section 5.5.

5.1 Summary

A main object of this project was the simulation of a quantum protocol using reinforcement learning algorithms. According to our experimental results, our reinforcement learning achieved promising performance in classical and quantum communication channel. First how to optimize the reinforcement learning algorithms using deep learning algorithms by exploring the message length and the comparing a quantum channel and a classical channel. For instance, most of the algorithms performed better with (1-2) message length. Second the multiagent environment has shown the minimum number of steps that the agent require to finish the environment successfully. These findings indicate the necessity of using a quantum protocol BB84 for training reinforcement learning algorithms, given that the available data maybe better fit the classification task but be short in quantity. Nevertheless, among the measures, the Mann-Whitney probability shows the correlation between the rewards and the quantum error that depicts the simulation balance nature of testing a quantum protocol.

5.2 Limitations

There are some limitations in this study. The first disadvantage of this study is the computer memory that this script requires to complete the training process. As the

memory of the personal computer that is used for this project can only produce those results and the training process cannot be extended.
Second, the data sources for the artificial environment need to be enriched. The creation of a channel that simulates the communication channel data can effectively have different results, which is very important for issues in quantum computing.

5.3 Implications

The results from the channel communication protocol had several implications for research in quantum computing and communication protocol. Initially, a way to perform a message exchange between two parties in recent researches several approaches of an attacker has proposed as an inspiration for future researchers that work with encryption and decryption and wish to test their efficiency in depth.
Another implication is the message size that is highly correlated with the error produced each time that filter does not change the bit correct.
Moreover, other work in communication protocol outside of the domain could also potentially implemented using reinforcement learning algorithms by incorporating the quantum teleportation. Published research on quantum teleportation in artificial inteligence has not be recommended recently paper that introduces the communication channel is et al. S. Olmschenk [5].

5.4 Execution Ideas

The majority of the implementation focused on communication between the two parties. The amount of words in the message that represent the main problem was tested. And the number of steps that the reinforcement learning algorithm requires to complete the communication.

5.5 Future Work

The study presented in this paper can be improved n many ways. Here, we elaborate several of the future research directions. First, we plan to incorporate the popular DDPG reinforcement learning algorithm that have not involved. Second, we can utilize more information that can be extracted using the the plain text and the encoded information to train a model so the study can concentrate to the performance of the

reinforcement learning algorithm.
Third, the extra study that focus on the neural network enables the analysis of Deep q-learning proximal policy optimization and evolution strategy. These issues are worthy of further exploration.

Chapter 6

Conclusion

The artificial intelligence depends on a reinforcement learning designed to solve real world problems. Due to its swallow depth it is relatively robust against errors and thus surpasses the need time for training until the agent learns the environment. One of the challenges for feasible use of the algorithm is the determination of suitable parameters. In this master thesis I implemented and evaluated BB84 quantum protocol proposed by [**?**] that two parties send messages to each other with encryption and decryption in order to measure the quantum channel to measure the mutual information that messages share after the exchange. This was realized by bench-marking it against the classical channel and quantum channel on a variety of reinforcement learning algorithms, q-learning, Deep q-learning, proximal policy optimization and evolutionary strategy.

It was found that the multiagent approach surpassing the performance of a single agent. The classical channel has almost identical approach with the quantum channel for the case that the message transmitted without any error, with classical channel to perform better by 7 as an average reward score . While the size of the message is one of the most important parameter. For message=2 and message=3 the reinforcement learning algorithms slightly reduced the number of simulations that the agent achieved to finish the episode with full reward. Accordingly, a different metric was chosen, namely fidelity and number of steps to compare with reward. For message=4 the reinforcement algorithms outperforms the previous message length only for the case of the evolutionary strategy(classical channel and quantum channel) and proximal policy optimization approach(classical channel). For message=1 all of the reinforcement learning algorithms outperforms the previous approaches message>2 so for classical and quantum channel. The q-learning approach is one that as the message increase the performance decreases. The reward values of the four reinforcement learning algorithms

discussed in this paper varied between 1.0 and 0.02. It can be said that besides the Deep Q-learning algorithm, all the algorithms produced acceptable by taking the reward as the only evaluation metric. When all algorithms were evaluated with the number of steps, it was observed that the number of steps of these models varied between 1 and 5. That is, it can be concluded that all the reinforcement learning results can be considered 'reasonable', apart from those of the Deep Q-learning algorithm. For the reinforcement learning approach data, the on policy method is better than the off-policy.

One important aspect of the algorithm is the number of errors that the quantum protocol produces during the training process and the mannwhitney measure that depicts how much is correlated the error with number of rewards that the agent achieves for a full simulation of the quantum protocol. This leads me to conclude that in practice that aspect should be taken in consideration.

The reinforcement learning algorithms applied to the Quantum Channel protocol and classical channel, and the numeric results from this thesis suggested that the amount of function calls required grows.

Moreover it was found that the number of steps that an agent performs during each episode were consistently observed in all the reinforcement learning algorithm that were investigated. While it is possible that the found parameters are increases with highest learning rate and more complex approaches proximal policy optimization and evolutionary strategy. This facts illustrates that the network architecture is another important parameter that might play significant role for further approaches.

The work conducted in this thesis has also led to new question. One of the main topics that requires further examination is whether or not it is possible to find better solutions due to the cpu limitation. One idea that comes to mind is to increase the number of iterations, and early stopping to the training procedure that will improve the model performance. Closely related to this idea is the use of actions as a tuple and not as a softmax that something that has not been tested yet.Possibly, this could offer an alternative training outcome. This approach could be aided with Machine learning techniques to recognize relevant features of the encoded message.

Another topic of interest that updates the current work is the implementation of the eavendropper that require another filter that the message pass through before the decode process. It would be interesting to see if reinforcement learning takes the advantage over increased error , as the message bit swiftness increases really becomes relevant to message size. It would be interesting to see if the communication using a reinforcement learning technique can be completed or it has the advantage beyond the

current work.
By extension, the problems that this work might solve except for the algorithm's resistance to error. Is the quantum technique of repetition of the initial message to avoid the misslead to a different decoding. Furthermore, it should be investigated and measured the performance with message repetition and the reinforcement learning message generation it might solve the stochastic nature of the repetition and reinforcement learning in combination with machine learning might overcome this part.

Chapter 7

Software

The data for this project were retrieved from the artificial environment. All the experiments and feature engineering tasks were implemented using the Python programming language. Details of the primary thrid-party Python libraries that simplified the modelling and data handling tasks are provided below.

Package	Version
numpy	1.22.4
itertools	8.7.0
matplotlib	3.6.0
tensorflow	2.10.0
scipy	1.9.3
random	1.2.1
keras	2.10.0
collections	2.1.0
datetime	4.0.2
sys	0.27.0
python	3.8.8

Table 7.1 Software and modules

References

[1] Barfuss, W., Donges, J. F., and Kurths, J. (2019). Deterministic limit of temporal difference reinforcement learning for stochastic games. *Phys. Rev. E*, 99:043305.

[2] Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60:503–515.

[3] Michael A. Nielsen, I. L. (2010). Quantum computation and quantum information. page 9.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[5] Olmschenk, S., Matsukevich, D. N., Maunz, P., Hayes, D., Duan, L.-M., and Monroe, C. (2009). Quantum teleportation between distant matter qubits. *Science*, 323(5913):486–489.

[6] Plaat, A. (2020). Learning to play reinforcement learning and games.

[7] S. Sutton, R. and G. Barto, A. (2014). Reinforcement learning: An introduction second edition: 2014, 2015.

[8] Sakellariou, B. B. K. (2020). Artificialintelligence.

[9] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning.

[10] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay.

[11] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[12] Stuart Russel, P. N. (2016). Artificialintelligence.

[13] Tokic, M. (2010). Adaptive ϵ-greedy exploration in reinforcement learning based on value differences. In Dillmann, R., Beyerer, J., Hanebeck, U. D., and Schultz, T., editors, *KI 2010: Advances in Artificial Intelligence*, pages 203–210, Berlin, Heidelberg. Springer Berlin Heidelberg.

[14] van Otterlo, M. and Wiering, M. A. (2012). Markov decision processes: Concepts and algorithms.

[15] Winiarczyk, P. and Zabierowski, W. (2011). Bb84 analysis of operation and practical considerations and implementations of quantum key distribution systems. In *2011 11th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, pages 23–26.

The pseudocode of the artificial learning algorithms implementation in Python is given in 3, 2, 5 and **??**. Those algorithms are the initial form the multi-agent environment implementation it is obtained by adding in a standard way of each agent access to the environment. The environment initialized and each share the same initial state, the same secret key and the each of the agents takes a different steps at each state. At the end of each step the environment, the algorithm checks if one of the agents have finished the episode with full reward. In case the episode has end the environment initialized again and a new initial state and a new secret key is produced. The number of agent depends on the length of the key and each one of them have trained individual for different key combinations. An episode have finished in case all the agents have complete the episode without the full reward or one of the agents have complete the episode with full reward.