Universiteit Leiden

Master Computer Science

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

Name: Athanasios Agrafiotis

Student ID: s2029413

Date:

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

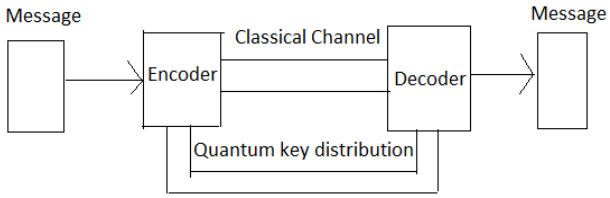
Chapter 1

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 [14]. 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 [14] by analyzing the performance of reinforcement learning algorithms on a classical channel communication ap-

proach 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 [11] and [3].
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 don 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. T. In Chapter 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

The related work et al Winiarczyk and Zabierowski[14] 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 2.1 and Z 2.2 . 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.

$$|\psi_i >= \begin{pmatrix} 1 \\ 0 \end{pmatrix} X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} |\psi f >= X|\psi_i >= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

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

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

$$|\psi_i >= \begin{pmatrix} 1 \\ 0 \end{pmatrix} Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} |\psi f >= Z|\psi_i >= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.2}$$

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

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

2.2 Reinforcement learning Related Work

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($Q(s,a) = (1-a)Q(s,a) + a[r + \gamma * maxQ(s',a')$) [13].
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

$$(s) = max_a R(s,a) + \gamma V(s') \tag{2.3}$$

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.

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 1 Q-learning [7]

```
Initialize Q(s,a), ∀s ∈ S, α ∈ A(s), arbitrarily, and Q(terminal − state, ·) = 0
for episode ∈ 1..N do
    Initialize S
    for t ∈ 0..T − 1 do
        Choose A from S using policy derived from Q(e.g., e-greedy)
        Take action A, observe R,S'
        Q(S,A) ← Q(S,A) + a[R + γ max_a Q(S',A) − Q(S,A)]
        S ← S'
    end for
    until S in terminal
end for
```

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.

$$TD(a,s) = R(s,a) + \gamma max_a Q(s',a') - Q(s,a) \tag{3.1}$$

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) [9]. 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) [12]. 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 2 Deep Q-learning Experience Replay [12]

```
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode ∈ 1..M do
    Initialise sequence s_1 = x_1 and preprocessed sequenced φ = φ(s_1)
    for t = 1,T do
        With probability ε select a random action α_t
        otherwise select α_t = max_α Q*(φ(s_t), α; θ)
        Execute action α_t in emulator and observe reward r_t and image x_{t+1}
        Set s_{t+1} = s_t, α_t, x_{t+1} and preprocess φ_{t+1} = φ_{s_{t+1}}
        Store transition (φ_t, α_t, r_t, φ_{t+1}) in D
        Sample random minibatch of transitions (φ_j, α_j, r_j, φ_{j+1}) from D
        Set y_j = { r_j                          for terminal φ_{j+1}
                  { r_j + γ max_{a'} Q(φ_{j+1}, α_j; θ)   for non-terminal φ_{j+1}
        Perform a gradient descent step on (y_j − Q(φ_j, α_j; θ))^2
    end for
end for
```

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 [10]. 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.
$\pi(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$

$$\delta_\theta J(\theta) = E[\sum_{t=1}^{T} \delta_\theta log\pi_\theta(a_t|s_t) \sum_{t=t+1}^{T} R(s_t,a_t)] \tag{3.2}$$

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.

$$A(s,a) = Q(s,a) - V(s) \tag{3.3}$$

$$A(s,a) = R + \gamma V(s') - V(s) \tag{3.4}$$

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.

$$L(\theta) = -\frac{1}{M}\sum_{i=1}^{M} log\pi_\theta(a^i|s^i)A(s^i,\alpha^i) \tag{3.5}$$

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.

$$r_t(\theta) = \pi_\theta(a_t s_t)\pi_{\theta old}(a_t s_t) \tag{3.6}$$

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

$$L(\theta) = E[min(r_k(\theta)A, clip(r_t(\theta), 1-e, 1+e)A] \tag{3.7}$$

The loss function selects lower value 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 3 PPO Clip [10]

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
end for

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 [8]. 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.

$$\theta(t+1) = \theta(t) + \eta \frac{1}{N\sigma} \tag{3.8}$$

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).

$$\bigtriangledown_\theta E_\varepsilon \sim_{N(0,I)} F(\theta + \sigma\varepsilon) = \frac{1}{\sigma} E_\varepsilon \sim_{N(0,I)} F(\theta + \sigma\varepsilon)\varepsilon \tag{3.9}$$

Algorithm 4 Evolution Strategies [8]

```
Input:Learning rate = α
noise standard deviation=σ
initial policy parameters=θ_0
for t = 0, 1, 2, .. do
    Sample_{ε_1..ε_n ~ N(0,I)}
    Compute returns F_i = F(θ_t + σε_i)
    for i = 1...n do
        set θ_{t+1} ← θ_{t+1} + α (1/(nσ)) Σ_{i=1}^{n} F_i ε_i
    end for
end for
```

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.

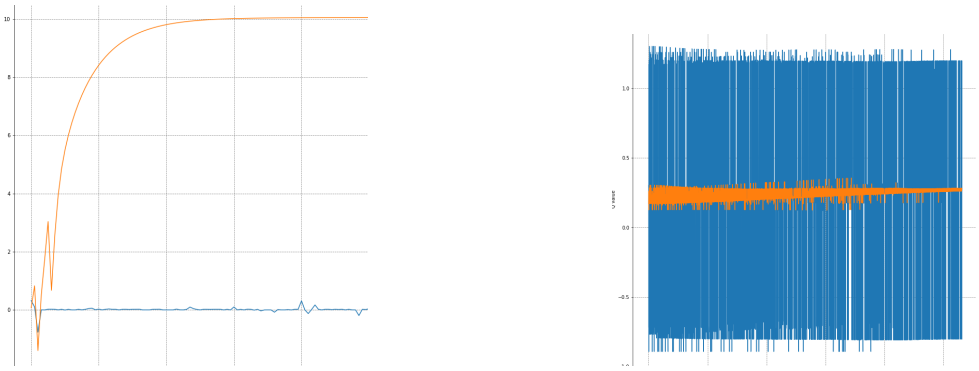
3.5 Adaptation of the Code for the Communication Protocol 84

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.

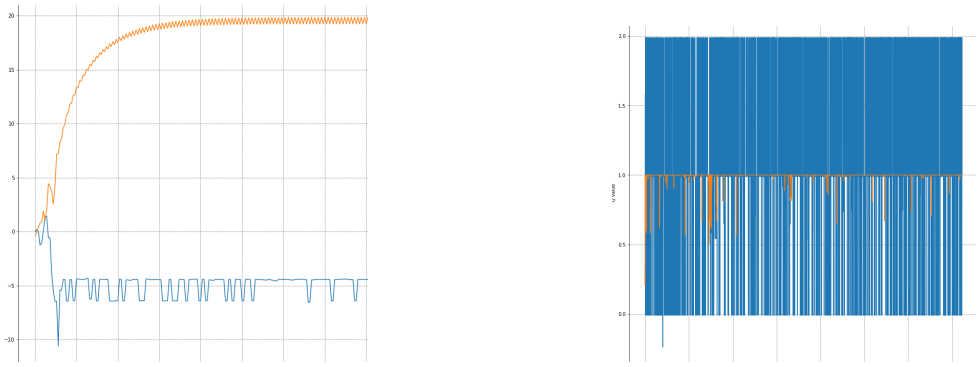
3.5.1 Training procedure

The agent learns to play the game in a hundred number of epochs. During training time, the exploration rate was initialized to a standard value. The convergence of Q-values is an indicator of the convergence of the network controlling the behavior of the agent. 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.
For the Q-learning algorithm, the gamma value 0.01 and the learning rate 0.001.
The deep Q-learning algorithm is the memory size 200 the gamma value 0.99 the exploration rate decreases with rate 0.01 and the learning rate $1e-4$.
The policy gradient algorithm hyperparameters are the number of episodes that was set to one hundred, the γ discount factor was set to 0.99 the clip ratio to 0.001. The policy learning rate 0.14.



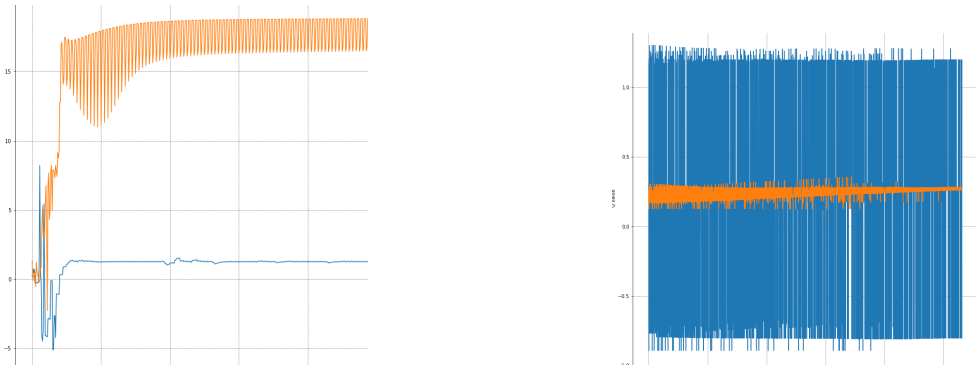
(a) **(b)**

Fig. 3.1 1 key Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes; (c) The number of steps of each episode.



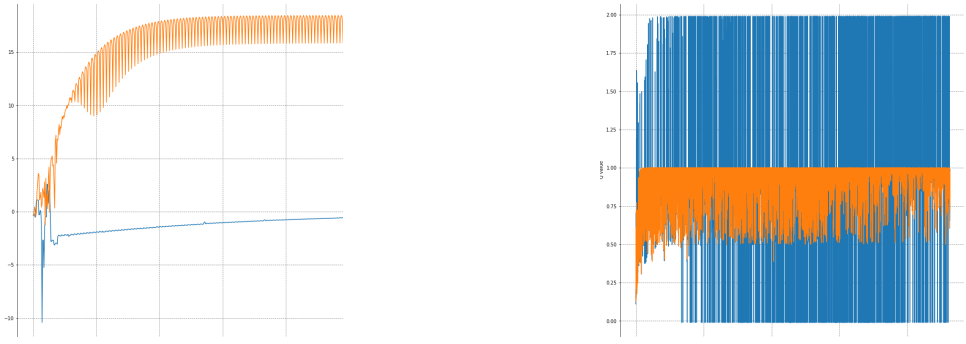
(a) (b)

Fig. 3.2 1 key Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes; (c) The number of steps of each episode.



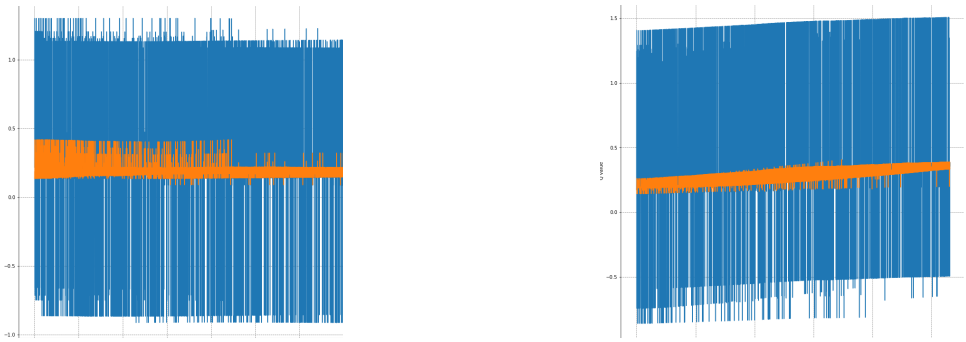
(a) (b)

Fig. 3.3 1 key Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes; (c) The number of steps of each episode.



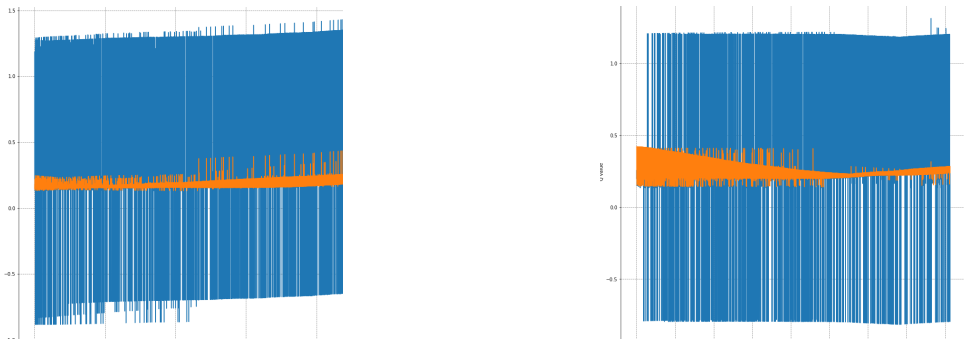
(a) (b)

Fig. 3.4 1 key Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes; (c) The number of steps of each episode.



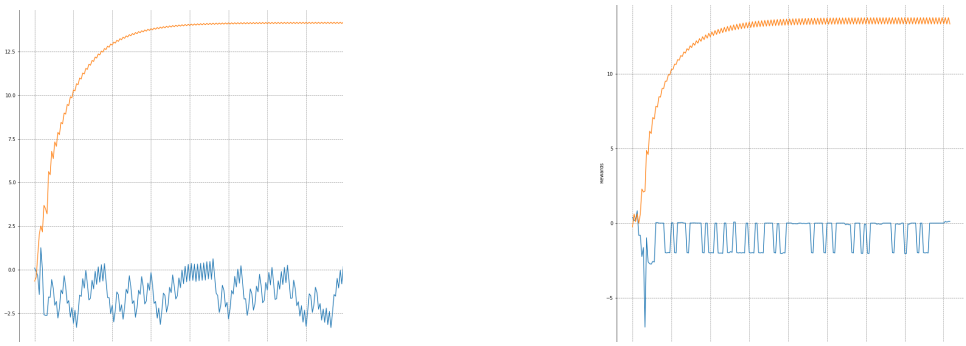
(a) **(b)**

Fig. 3.5 1-2 key Deep Q-learning (a) qvalues; (b) qvalues.



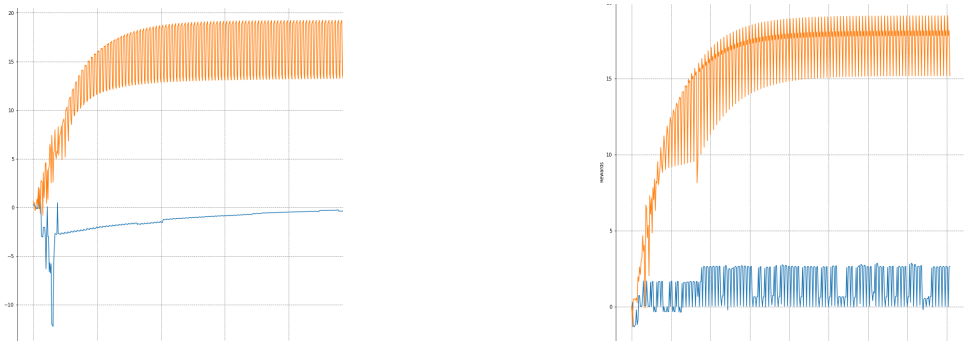
(a) **(b)**

Fig. 3.6 3-4 key Deep Q-learning (a) qvalues; (b) qvalues.



(a) **(b)**

Fig. 3.7 1-2 key Proximal Policy Optimization (a) qvalues (b) qvalues.



(a) **(b)**

Fig. 3.8 3-4 key Proximal Policy Optimization (a) qvalues; (b) qvalues.

Chapter 4

Results

This chapter provides details of the results. The evaluation criteria and the adjustment of hyper-parameters.

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:

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

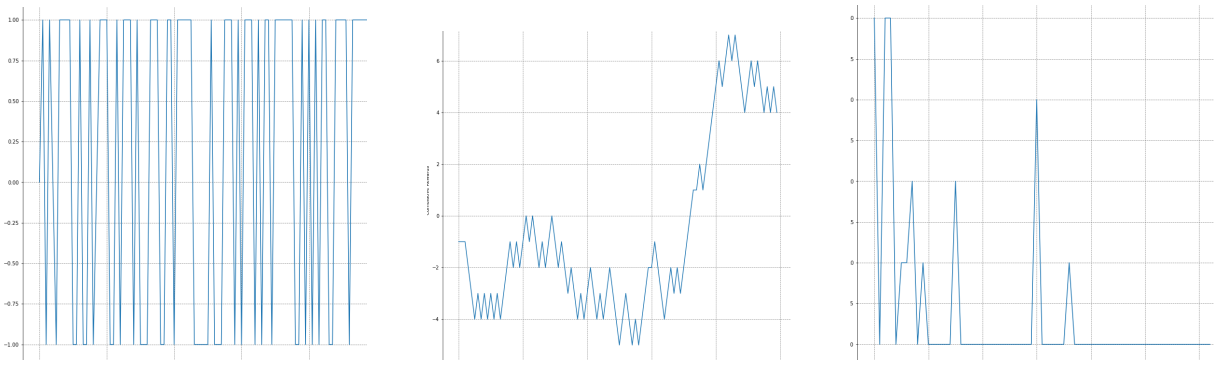
$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2 \tag{4.2}$$

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 4.5 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 4.5, 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.



(a) **(b)** **(c)**

Fig. 4.1 1 key Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes; (c) The number of steps of each episode.

	1	2	3	4
Q-learning	0.59	0.70	0.62	0.51
Deep q-learning	0.55	0.54	0.48	0.25
Proximal Policy Optimization	0.37	0.75	0.93	0.94
Evolutionary Strategy	0.54	0.73	0.78	0.93

Table 4.1 Classical Channel rewards of the agent one environment

	1	2	3	4
Q-learning	0.56	0.63	0.7	0.72
Deep q-learning	0.44	0.45	0.32	0.08
Proximal Policy Optimization	0.57	0.64	0.7	0.67
Evolutionary Strategy	0.51	0.54	0.66	0.75

Table 4.2 Quantum Channel rewards of the agent one environment

	1	2	3	4
Q-learning	0.79	0.72	0.97	0.98
Deep q-learning	0.54	0.63	0.61	0.75
Proximal Policy Optimization	0.76	0.46	0.81	0.36
Evolutionary Strategy	0.79	1	0.95	1

Table 4.3 Quantum Channel rewards multiagent one environment

	1	2	3	4
Q-learning	2	3	4	5
Deep q-learning	1	1	1	1
Proximal Policy Optimization	2	1	1	1
Evolutionary Strategy	2	2	3	3

Table 4.4 Quantum Channel steps multiagent one environment

	1	2	3	4
Q-learning	2	3	4	5
Deep q-learning	4	4	4	5
Proximal Policy Optimization	4	2	3	4
Evolutionary Strategy	2	2	3	4

Table 4.5 Classical Channel Average number of steps

	1	2	3	4
Q-learning	1	3	4	4
Deep q-learning	2	4	4	5
Proximal Policy Optimization	1	2	3	4
Evolutionary Strategy	1	3	4	5

Table 4.6 Quantum Channel Average number of steps

	1bit	2bit	3bit	4bit	1bit	2bit	3bit	4bit
Q-learning	0.49	0.22	0.0	0.0	1.0	0.0	0.0	1.0
DQN	0.475	0.16	0.0375	0.0	0.6125	0.225	0.1125	0.0
Evolutionary Strategy	0.56	0.25	0.0	0.0	1.0	1.0	0.0	0.0
Proximal Policy Optimization	0.62	0.0	0.0	0.0	1.0	1.0	0.0	0.0

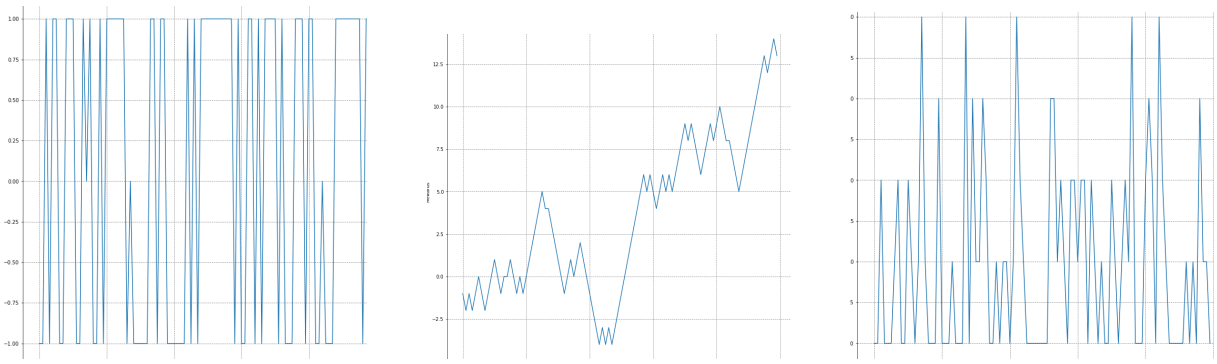
Table 4.7 Quantum Channel Average number of steps

	1bit	2bit	3bit	4bit	1bit	2bit	3bit	4bit
Q-learning	3.53	0.22	0.0	0.0	2.0	5.0	5.0	5.0
DQN	1.0	0.16	0.0375	0.0	1.0	5.0	5.0	5.0
Evolutionary Strategy	5.0	0.25	0.0	0.0	1.0	2.0	5.0	5.0
Proximal Policy Optimization	2.52	0.0	0.0	0.0	1.19	2.0	5.0	5.0

	1bit	2bit	3bit	4bit	1bit	2bit	3bit	4bit
Q-learning	13	23	33	41	0	0	0	0
DQN	11	17	22	25	0	0	0	0
Evolutionary Strategy	24904	47145	66049	82887	0	0	0	0
Proximal Policy Optimization	11	19	31	49	0	0	0	0

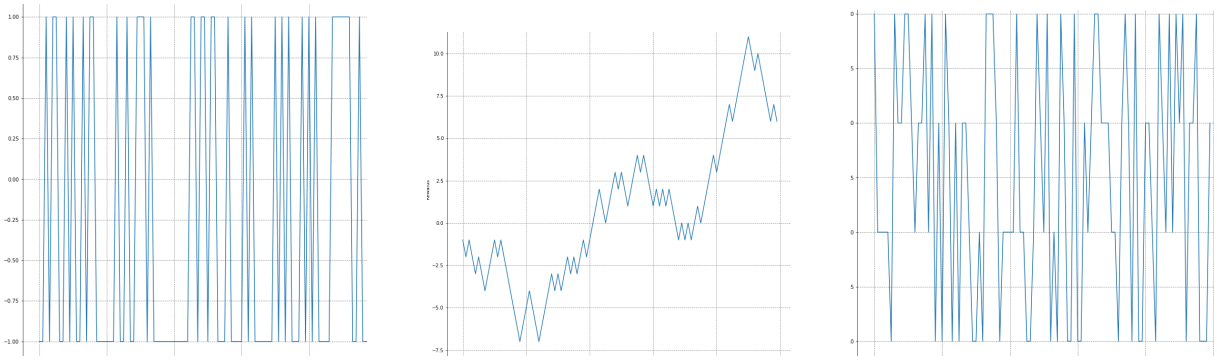
	1bit	2bit	3bit	4bit	1bit	2bit	3bit	4bit
Q-learning	$6.95062615298543e-05$	0.007816379862527104	$7.498558461086548e-12$	$2.815716029104796e-15$	$9.840451187915974e-22$	0.04244968203133357	0.0004870318747568524	0.00300360710513877 33
DQN	0.0038388887074739296	0.005547304761726724	$1.9990210674743843e-09$	$6.8414873967176365e-12$	$1.3723778492268604e-09$	0.0	0.050556883459602873	$8.424178889344995e-05$
Evolutionary Strategy	$6.017558165605808e-40$	0.3683604786732907	$1.100392579344011 5e-12$	$2.02696721543950 47e-17$	0.0	0.0 Proximal Policy Optimization	$3.995270159330368e-14$	$2.460849268392077e-06$
$7.7143658317031 55e-10$	$4.706144976928242e-16$	$1.7608079411973e-45$	$1.7608079411973e-45$	0.0	0.0			

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



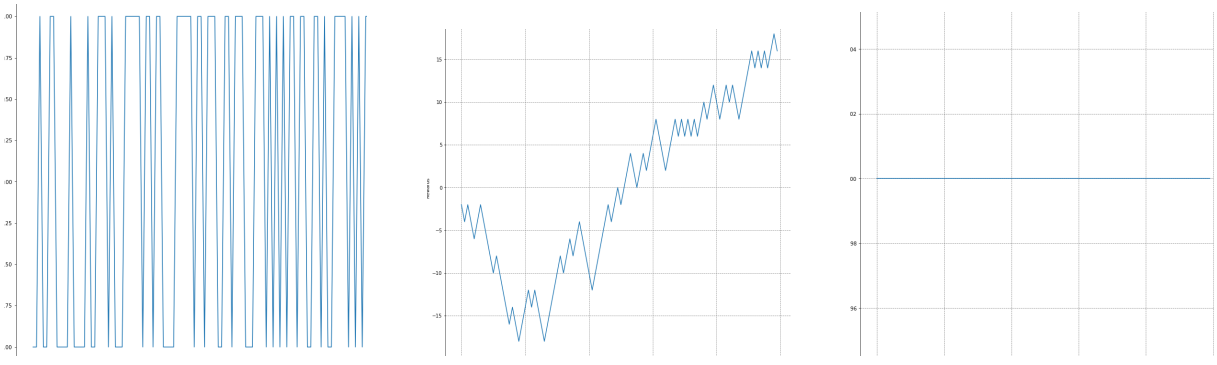
(a) **(b)** **(c)**

Fig. 4.2 1 key Deep Q-learning (a) The reward on the test set; (b) The q-value during the training; (c) The number of steps of each episode.



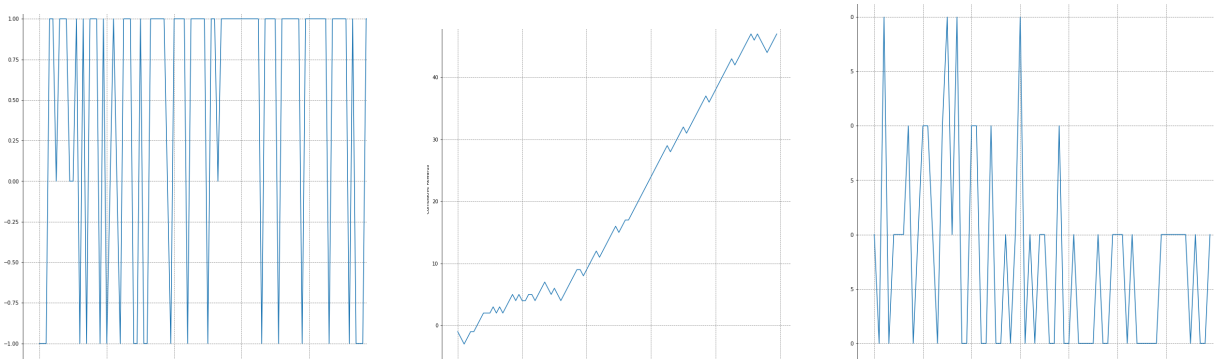
(a) **(b)** **(c)**

Fig. 4.3 1 key Proximal Policy Optimization (a) The reward on the test set; (b) The q-value during the training; (c) The number of steps of each episode.



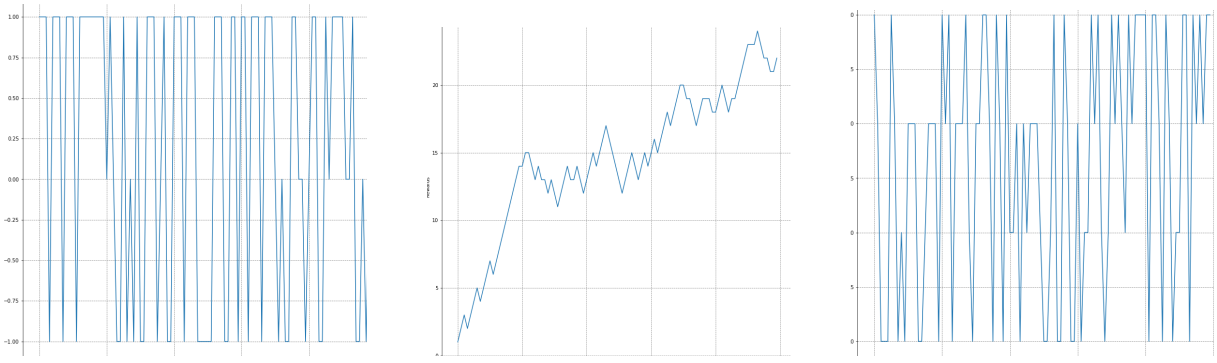
(a) **(b)** **(c)**

Fig. 4.4 1 key Evolutionary Strategy (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



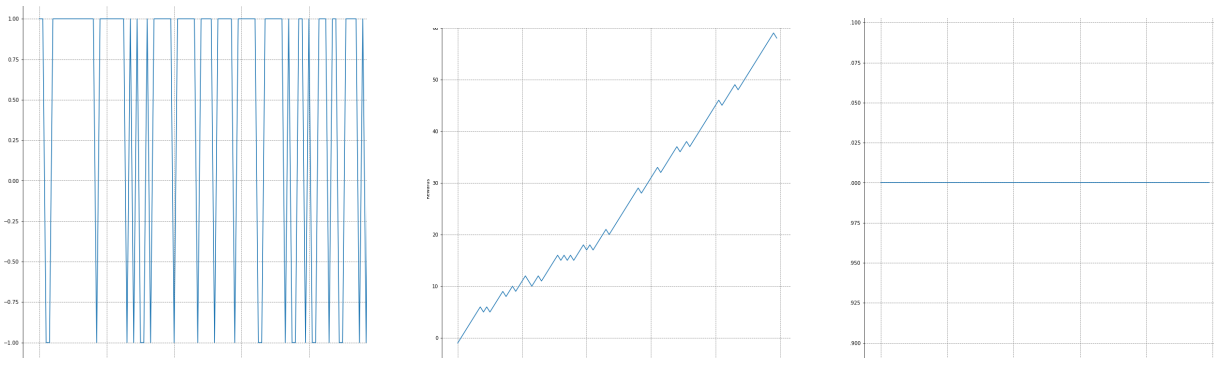
(a) **(b)** **(c)**

Fig. 4.5 2 keys Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



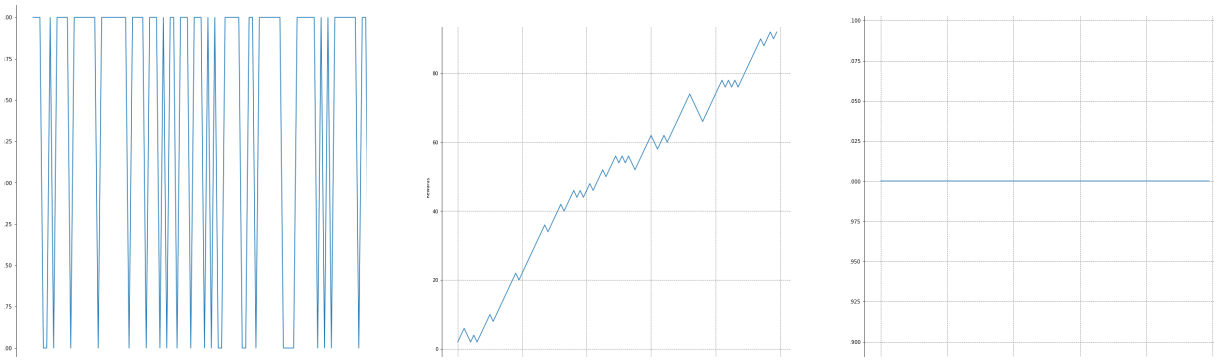
(a) **(b)** **(c)**

Fig. 4.6 2 keys Deep Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



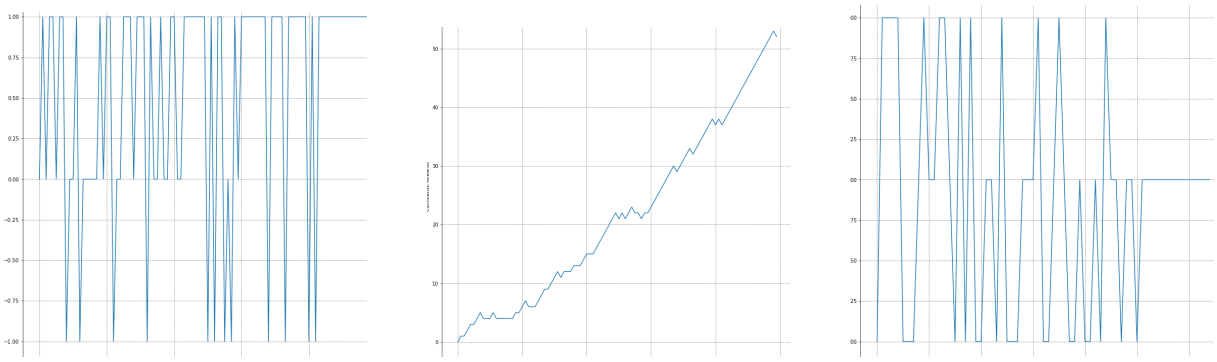
(a) **(b)** **(c)**

Fig. 4.7 2 keys Proximal Policy Optimization (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



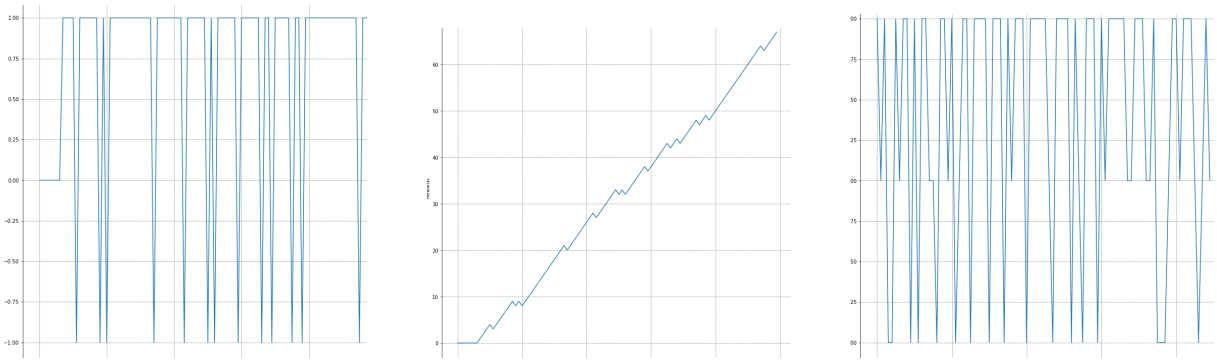
(a) **(b)** **(c)**

Fig. 4.8 2 keys Evolution Strategy (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



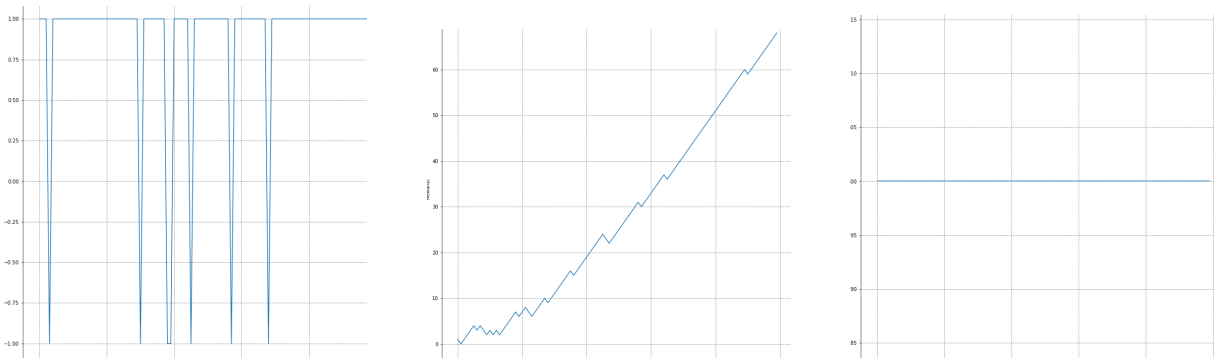
(a) **(b)** **(c)**

Fig. 4.9 3 keys Q learning (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



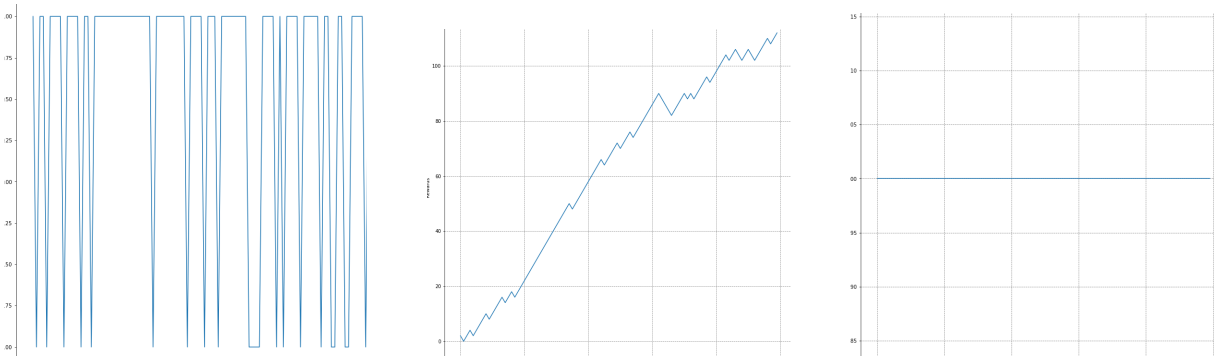
(a) **(b)** **(c)**

Fig. 4.10 3 keys Deep Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



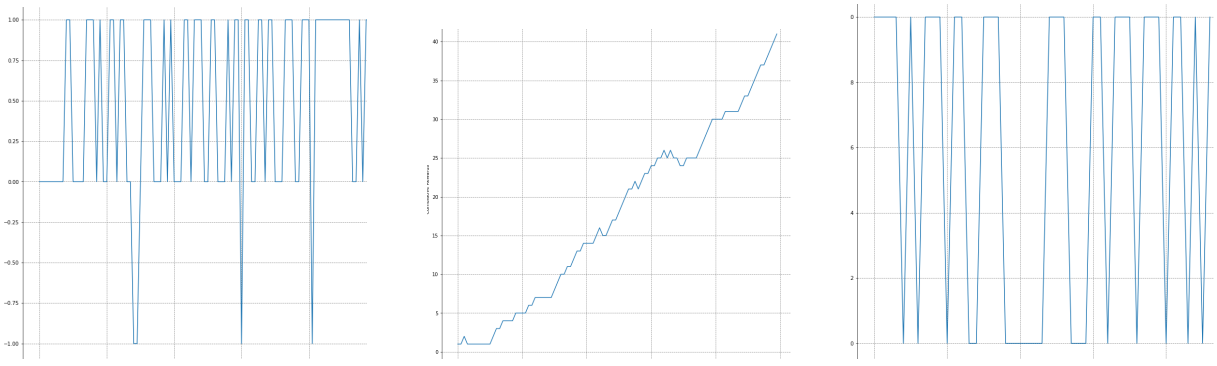
(a) **(b)** **(c)**

Fig. 4.11 3 keys Proximal Policy Optimization (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



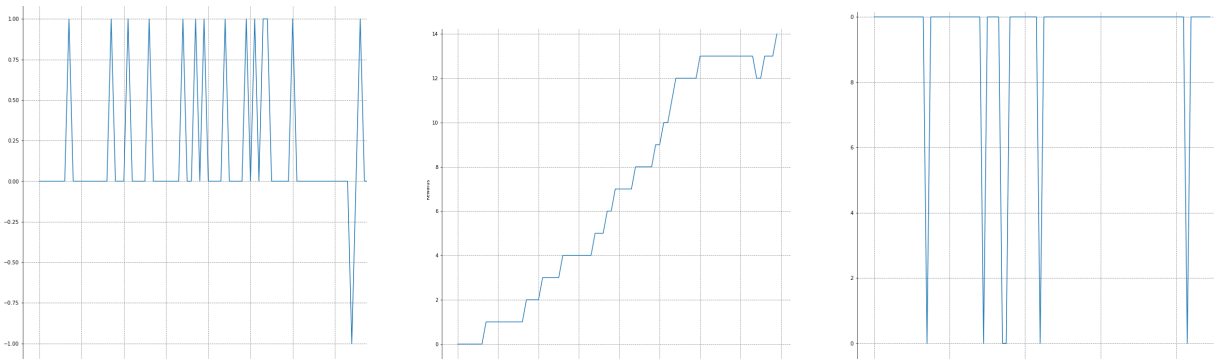
(a) **(b)** **(c)**

Fig. 4.12 3 keys Evolutionary Strategy (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



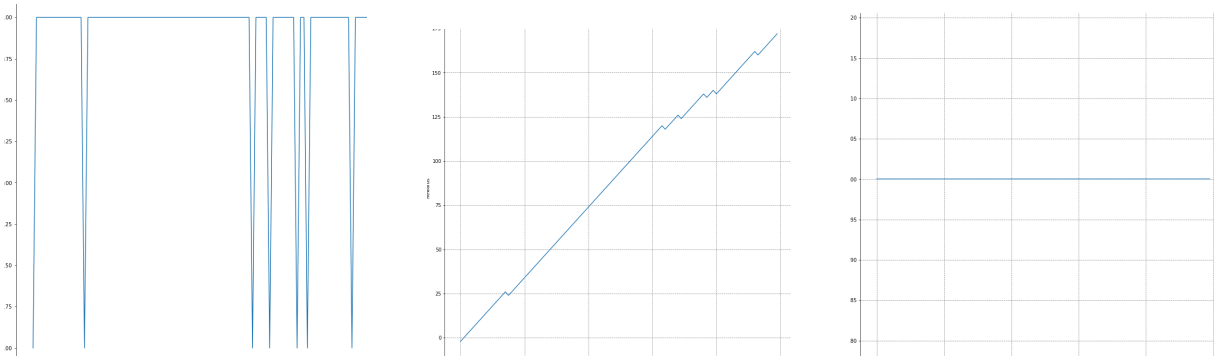
(a) **(b)** **(c)**

Fig. 4.13 4 keys Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



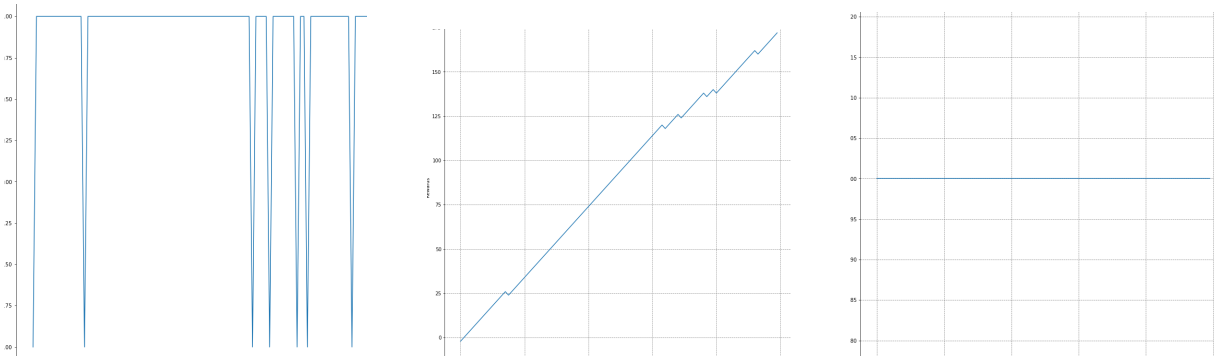
(a) **(b)** **(c)**

Fig. 4.14 4 keys Deep q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



(a) **(b)** **(c)**

Fig. 4.15 4 keys Proximal Policy Optimization (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



(a) **(b)** **(c)**

Fig. 4.16 4 keys Evolutionary Strategy (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.

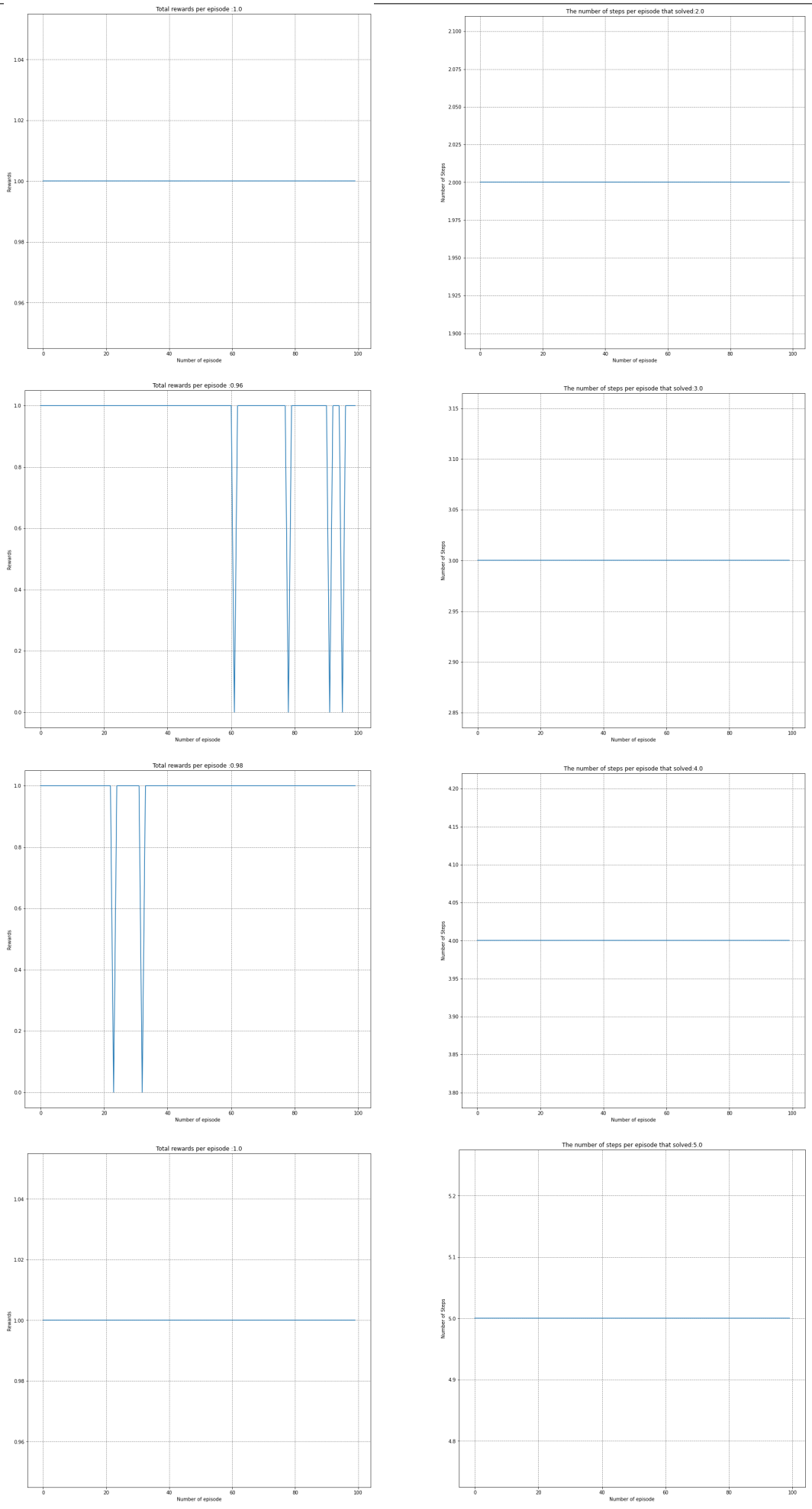


Fig. 4.17 Q-learning Multiple Environments

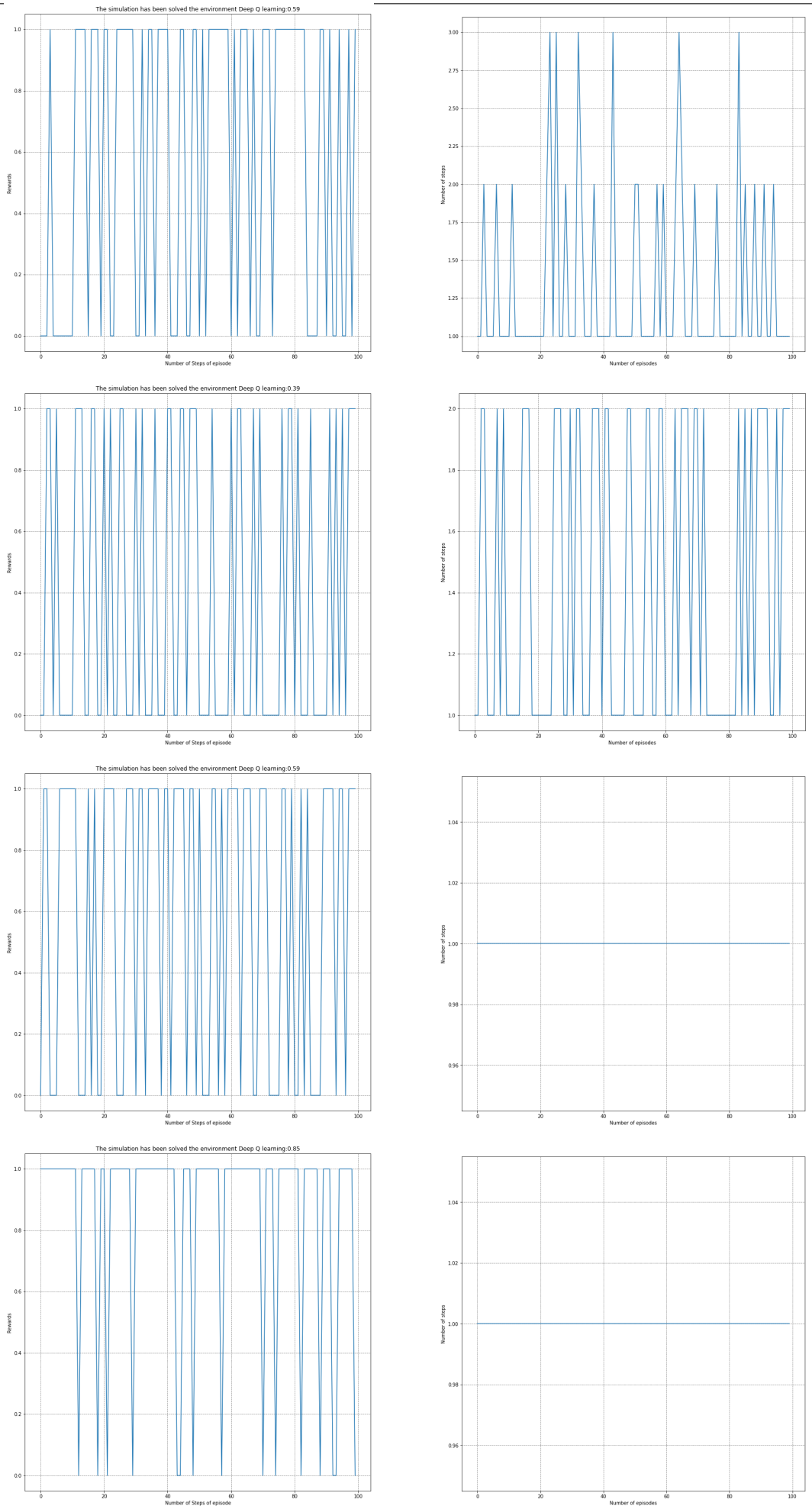


Fig. 4.18 DQN Multiple Environments

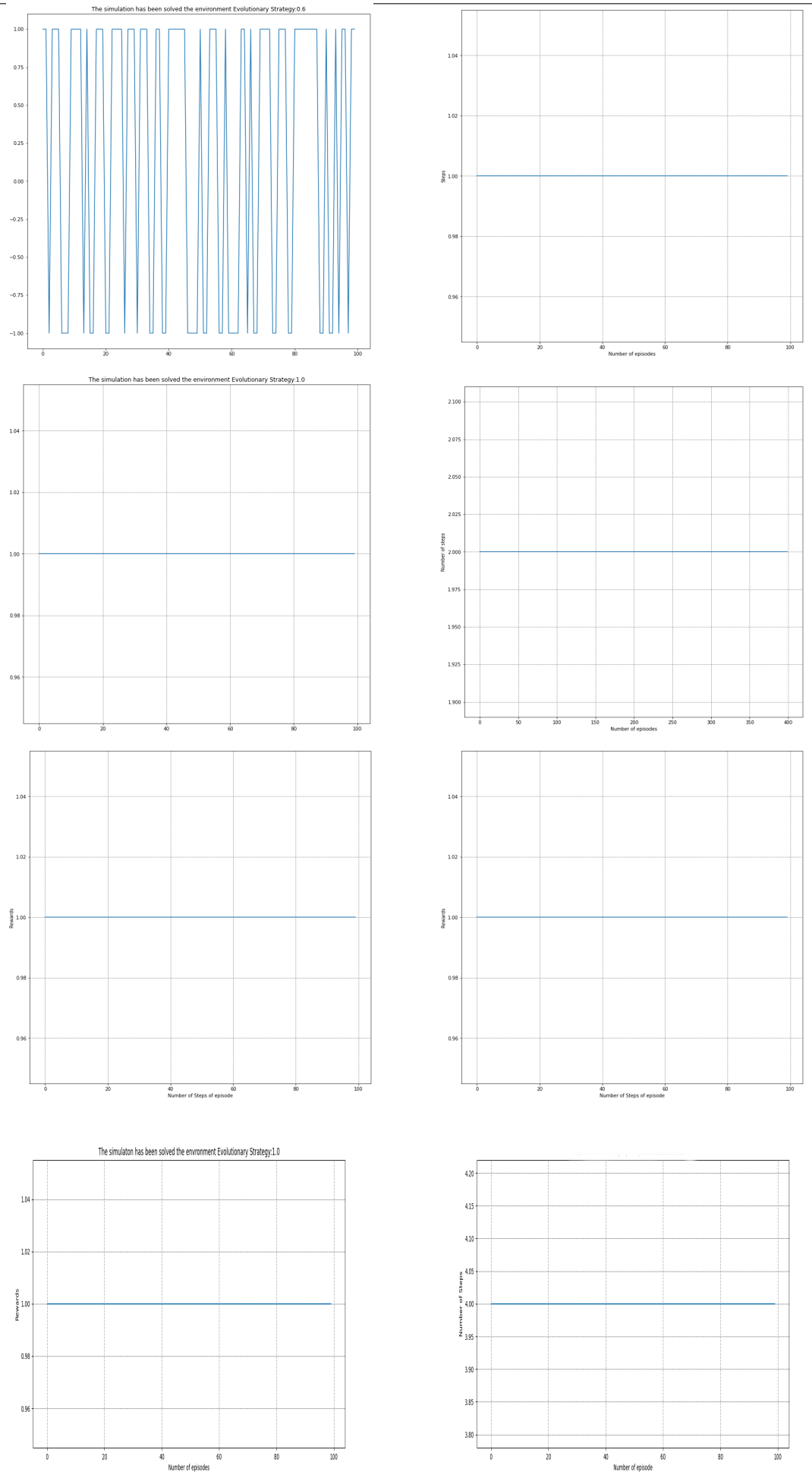


Fig. 4.19 Evolutionary Strategy Multiple Environments

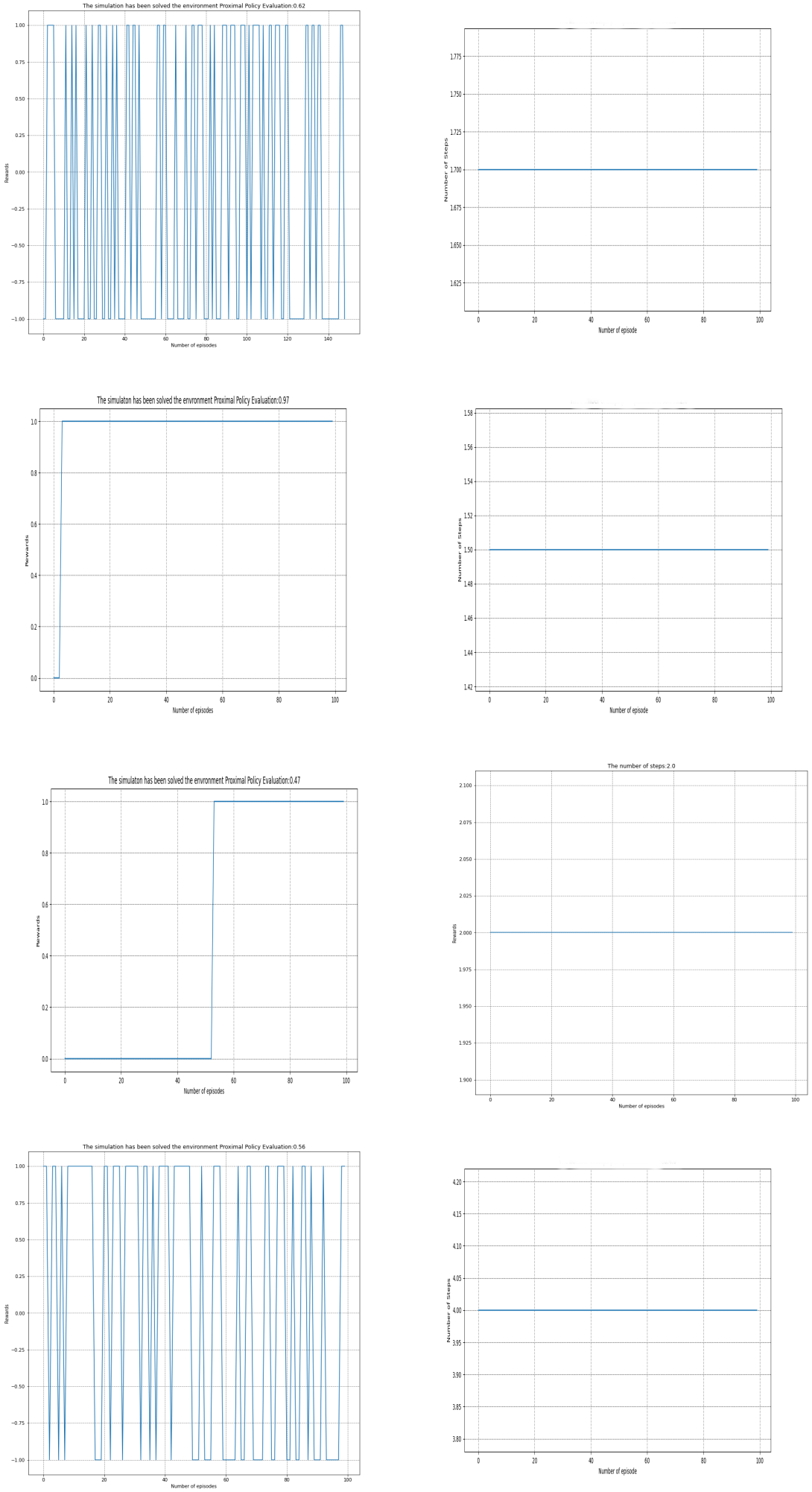
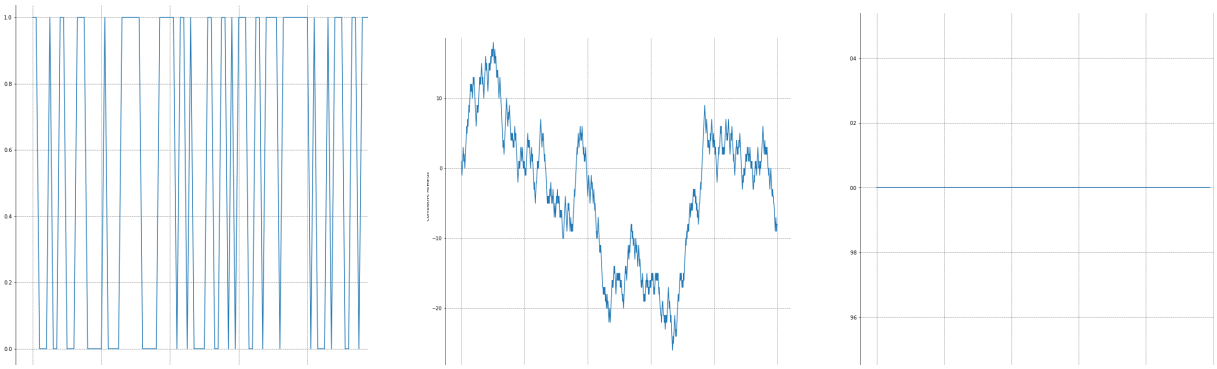
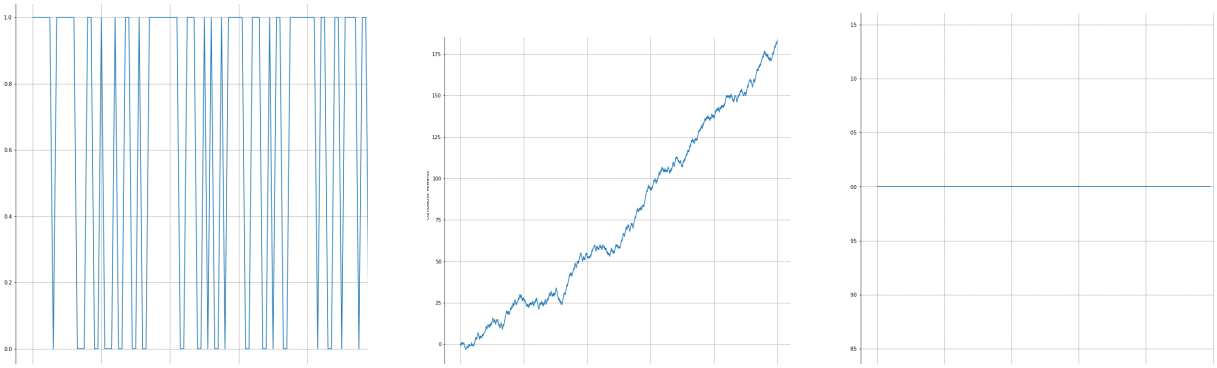


Fig. 4.20 Proximal Policy Optimization Multiple Environments



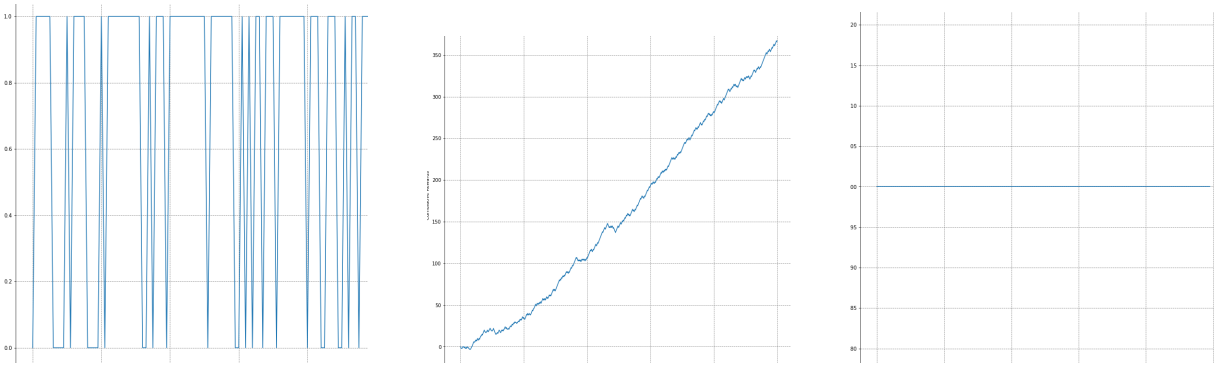
(a) **(b)** **(c)**

Fig. 4.21 1 key Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes; (c) The number of steps of each episode.



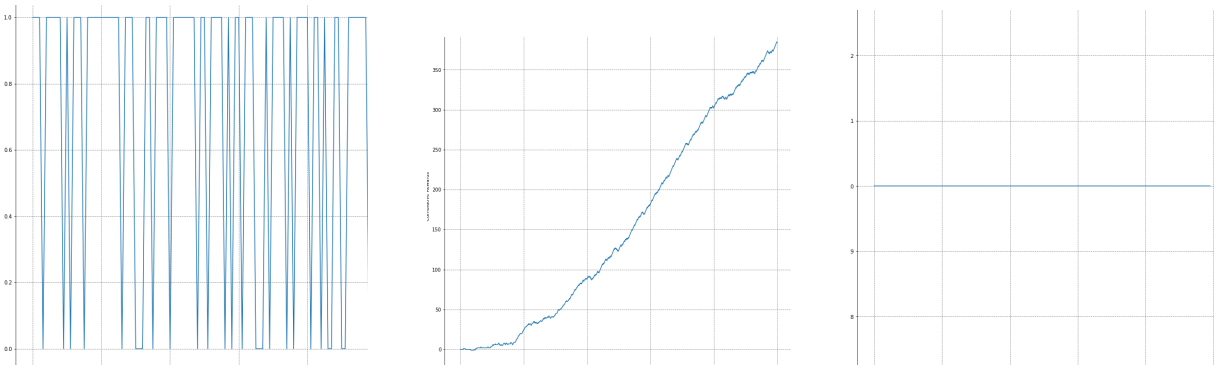
(a) **(b)** **(c)**

Fig. 4.22 2 key Q-learning (a) The reward on the test set; (b) The q-value during the training; (c) The number of steps of each episode.



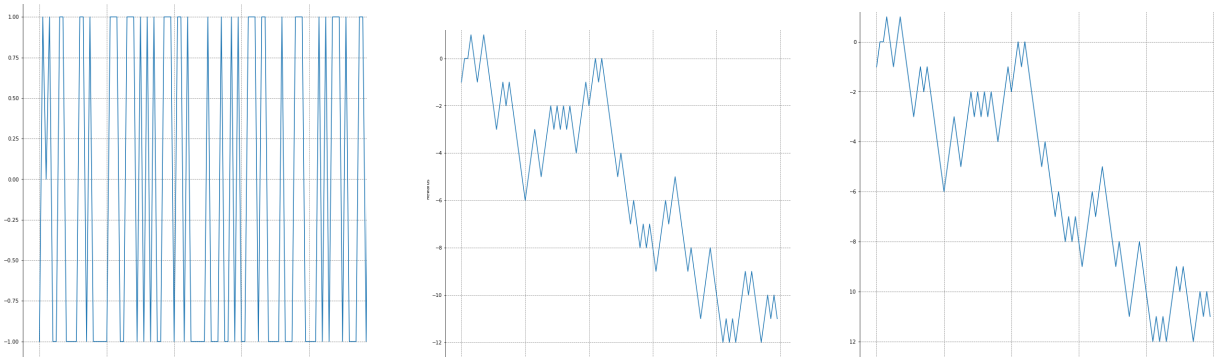
(a) **(b)** **(c)**

Fig. 4.23 3 key Q-learning (a) The reward on the test set; (b) The q-value during the training; (c) The number of steps of each episode.



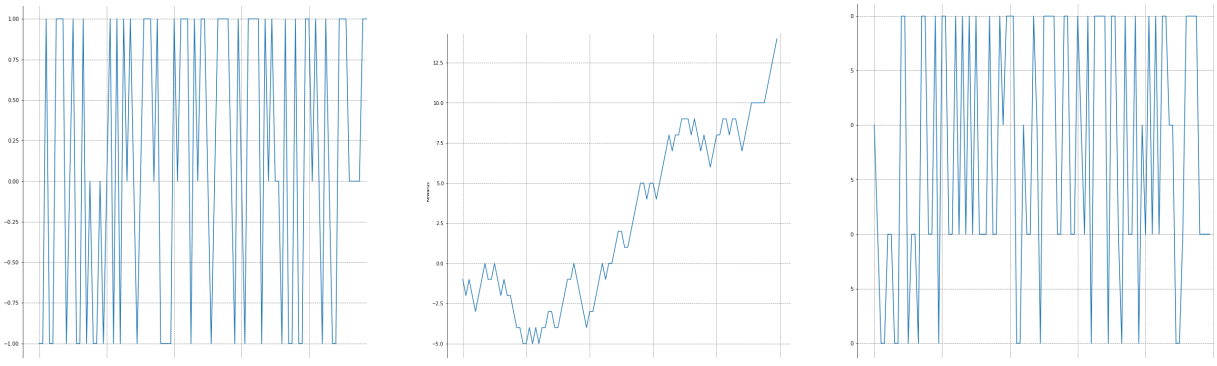
(a) **(b)** **(c)**

Fig. 4.24 4 key Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes ; (c) The number of steps of each episode.



(a) **(b)** **(c)**

Fig. 4.25 1 key Deep Q-learning (a) The reward on the test set; (b) The cumulative reward of all episodes; (c) The number of steps of each episode.



(a) **(b)** **(c)**

Fig. 4.26 2 key Deep Q-learning (a) The reward on the test set; (b) The q-value during the training; (c) The number of steps of each episode.

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 algorthms 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

In this study, we used four reinforcement learning algorithms to simulate a quantum key distribution protocol. The following issues should be discussed furthermore based on the above data analysis. The methodology framework proposed in this paper provides an effective and rapid approach to a reinforcement learning environment. The size of the key can be combined with an eavesdropper filter. It was demonstrated that the quantum communication channel could successfully transmit the message. With the presence of an eavesdropper, the communication uses a second filter that changes the secret key combination. The proposed approach can provide very useful information to support future research on message transmission.

The results of our case study demonstrate that for reinforcement learning algorithms the key size is important. Some algorithms performance increased when the secret key had more bits than the previous approach. The Q-learning approach and the Deep Q-learning approach had the worst results. Therefore, when the interpretability of these approaches needs to be taken into account, reinforcement learning algorithms on-policy methods should be given priority.

The reward values of the four reinforcement learning algorithms discussed in this paper varied between 0.94 and 0.25. 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 4. 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.

According to the evaluation results, it can be seen that proximal policy optimization and evolutionary strategies offer greater rewards than Q-learning and Deep Q-learning. Therefore, proximal policy optimization was recognized as a more suitable algorithm for this artificial environment.

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] Blahaba, Basileiadis, K. S. (2020). Artificialintelligence.

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

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

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

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

[12] 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.

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

[14] 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 2, 1, 4 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.