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

This dissertation is submitted for the degree of
Computer Science(MSc): Advanced Data Analytics

October 2022

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 the 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 in details how 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 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 second agent , the receiver verify the message correctness. In case the message has been transferred successful, the episode ends with the maximum reward in the other cases the reward is negative.

A number of reinforcement learning algorithms are implemented during the Master thesis project. Namely a Q-learning, deep q learning approach that solves artificial environment with optimal solutions. As a result the agent performs that actions that is required to communicate with each other avoiding any mistakes.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The current project has as a main goal to simulate an artificial environment of quantum key Distribution. The process that describes a communication between two artificial agents that takes place in a quantum channel. For reasons of security, the channel uses a protocol that encrypts And decrypts the messages with some error. Next, the sender and the receiver communicate with a classical to channel to compare the message and to evaluate the protocol key's. The quantum Channels use quantum gates as key that produce a small amount of error. In the artificial environment, each of the two agents can make at least ten actions until the episode ends and communication to finish. In case each of the agents makes the required actions the episode finishes earlier and gives a positive reward to the agent. The communication channel generates a message that the sender will read it, next will send it to receiver that he will compare both messages and saves the key.

To solve the environment, it is a proposed reinforcement learning algorithm. Algorithms can explore the environment until to find an optimal solution playing a large number of episodes.

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, 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 a plain text(cipher-text) encrypts the message 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, error correction, number of iterations will determine the key quality.

Chapter 2

Background

2.1 Example

2.2 Quantum Key distribution Related Work

The related work et al [] it 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 (QKD) 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 a communication over a quantum channel for polarized transmission. The cryptographic part relies on three components. First, the plain text that will be encrypted, 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 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 with and with eavesdroppers is complicated to compare to previous work, as their analysis does not clearly state their parameters and the error.

2.3 Reinforcement learning Related Work

The field of reinforcement introduce 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 data from the environment and navigation strategy of the agent express a probability. The model-based the agent select the actions that maximize it's reward from prediction of the environment. The value-based maximize it's reward through the navigation in the environment. Policy based update their parameters through gradient descent by taking the differentiate. Off-policy learns from the previous episodes in contrast on-policy can only learn on new data.
The most known algorithm that solves artificial environment is the Bellman equation. The equation uses the artificial environment variables s,a,r and gamma which corresponds to the state, action , reward and discount factor. The agent is in an environment that navigates and in case the agent lose get a negative reward or zero and in case of win takes positive reward. The bellman equation helps the agent to go through the environment. The bellman equation $V(s) = max_a R(s,a) + \gamma V(s')$ so the agent by taking an action in 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 if the agent takes an action that is equal to zeros and calculating the discount factor is equal to gamma plus the reward of the winning state.

Chapter 3

Methods & Data

3.1 Q-learning

An agent uses the values of the next states to take a decision to which state to move next. A tabular representation of the actions is used as Q that represents the quality of the actions. If the environment has four action each of the action has some kind of quality. $Q(s,a) = R(s,a) + \gamma(Q(s',a'))$ Using a q-learning the agent when performs an action he gets a reward and also it gets the expected value.

Algorithm 1 Q-learning

αInput:Learning rateσdiscount factor
$\pi \leftarrow$ greedy policy w.r.t.$Q(s,a)$
$b \leftarrow$ exploratory policy with coverage ofπ
Initialize $Q(s,a)$arbitrarily, with$Q(terminal,) = 0$
for episode $\in 1..N$ **do**
  Reset the environment and observeS_0
  for $t \in 0..T-1$ **do**
    $A_t \sim b(S_t)$
    Execute A_t in the environment and observe S_{t+1}, R_{t+1}
    $Q(S_t,A_t) \leftarrow Q(S_t,A_t) + a[R_{t+1} + \gamma\pi(S_{t+1}) - Q(S_t,A_t)]$
  end for
end for
Output:Approximately optimal policy$\pi$and action values$Q(s,a)$

3.2 Deep Q-learning

The agent before proceed to a next state calculates the reward and 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 state is used from a neural network that process the information and the it outputs the actions. It use the observation of the agent and

outputs probabilities for the new actions that the agent agent should take to maximize it's reward. The q-learning is not working for complex environment in contrast with the deep q-learning agent. The temporal difference is the foundation way to express probabilities , when the agent takes the decision to move to the next state. The agent by taking this action with the maximum policy receives the reward. $TD(a,s) = R(s,a) + \gamma max_a Q(s',a') - Q(s,a)$ In more details the 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 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 $L = \sum(Q - Target - Q)$ this is the way that agent learns.
The experience replay gives the agent the opportunity to learn from a sample of state. It takes a number of sample that is random uniformly and learns from them, each experience has the state that the agent was , the next state the action and the reward (four elements). The most valiable are the rare experiences, data that contains states that does not repeat frequently. The inputs in the neural network are the move of the agent from one state to another state. The state go through the network the error is calculated and the network backpropagates then the agent agent selects which action need to take. The new state is used as the previous and goes through the network. Once the vector that describing the state is used from a neural network and learning process ends we got as an outpt all the q-values. The output that are the q-values, the selection of the best q-value. The q-learning approach selects the one with highest q-value and takes that action because it gives the best reward. In contrast the deep q-learning uses a softmax activation function. There is also a number of different action selection function such as the e-greedy and e-soft(1-e). The e-greedy selects the action with highest reward when e is 0.4 with forty percent selects the action random and 0.6 selects the action with the highest reward, with e-soft selecting at random an action if the e is 0.2 the agent selects the action with the highest reward and with 0.8 selects at random. The number of different action selection policies provides different ways for the agent to navigate through the environment, other times agent exploits the environment and other times explores it. The different functions prevents the agent to trapped to the local maximum, so the agent will navigate receiving the best reward but it might not finished the episode with the maximum reward. softmax selects the best q_values that are probabilities that equals to one.

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{3.1}$$

Algorithm 2 Deep Q-learning Experience Replay

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$(y_j - Q(\phi_j, \alpha_j; \theta))^2$
  end for
end for

3.3 Policy Optimization

Policy gradient methods learns online, this algorithm sampling data in order to train our algorithms. Many policy gradients methods are sample inefficient. Our goal is to maximize our policy performance. Algorithms are learn many times on the same data by limiting the changes on the policy. The way to optimize a policy by taking states giving actions and computing estimates of the advantage then we find the estimate of the expression and moves the parameters of our policy in this direction.

$$J(\theta) = E_t[ln\pi_\theta(a_t s_t)A_t] \tag{3.2}$$

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

$$r_t(\theta) = fraction\pi_\theta(a_t s_t)\pi_{\theta old}(a_t s_t) \tag{3.3}$$

This means that the agent would have selected the current action (t) when it was in the state (t) with probability which was initial 12% and after some iterations the agent would choose the action(t) of the state(t) with probability 90%. The ratio (r) is the 9012. The ratio and a parameter called epsilon will limit the policy changes. With this way a new Loss function is computed

$$L(\theta) = E[min(r_k(\theta)A, clip(r_t(\theta), 1-e, 1+e)A] \tag{3.4}$$

The loss function selects lower value value between the $clip(r_t(\theta), 1-e, 1+e)A$. The parameter A and 1+e,1-e will help to constraint the policy formula increase and make it even less probable. So 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 Proximal Policy Optimization

```
for iteration ∈ 1,2.. do
    for actor = 1,2,...,N do
        Run policy π_{θ_old} in environment for T timesteps
        Compute advantage estimates Â_1...Â_T
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size M ≤ NT
    θ_old ← θ
end for
```

3.4 Evolution Strategy

The evolution strategy makes use of a neural network architecture. The neural network consists of 20 hidden nodes a number of inputs that corresponds to the number of states and the number of outputs that are the actions that the agent can take in each step. The initialize it's 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, the network weight's will be updated. The weights network are represent the offspring of the population and are updated from the objective function.

$$\bigtriangledown_\theta E_\varepsilon \sim_{N(0,I)} F(\theta + \sigma\varepsilon) = \frac{1}{\sigma} E_\varepsilon \sim_{N(0,I)} F(\theta + \sigma\varepsilon)\varepsilon \tag{3.5}$$

Algorithm 4 Evolution Strategies

```
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
    set θ_{t+1} ← θ_{t+1} + α (1/(nσ)) Σ_{i=1}^{n} F_i ε_i
end for
```

Chapter 4

Results

Chapter 5

Discussion

Chapter 6

Conclusion

Chapter 7

Software

References