Quantum circuit design with reinforcement learning

Francesco Montagna

October 17, 2021

Contents

1 Introduction

Quantum Computing promises to solve computational problems much faster than any classical computer, also unlocking solutions to challenges which can not be approached with classical processors in any useful amount of time. In this thesis we want to combine the power of reinforcement learning with quantum computing: the goal is to train an agent to design a quantum circuit which transforms an initial state vector associated to a quantum system into a target state of interest.
We proceed by introducing the principles of Reinforcement Learning, Quantum Mechanics and Quantum Computation. Then, we provide a detailed description of the algorithm constructed and the results obtained. The designed quantum circuit is then implemented and run on a real quantum device from IBM Quantum Lab, to provide a comparison between classical simulation and quantum experiment.

The code employed in this thesis is publicly available at this Github repository: https://github.com/francescomontagna/Quantum-Reinforcement-Learning. The reinforcement learning algorithm is implemented in Python, the experimental part using the Qiskit framework.

2 Reinforcement Learning

This chapter is mainly inspired by the content of [1] and [2].

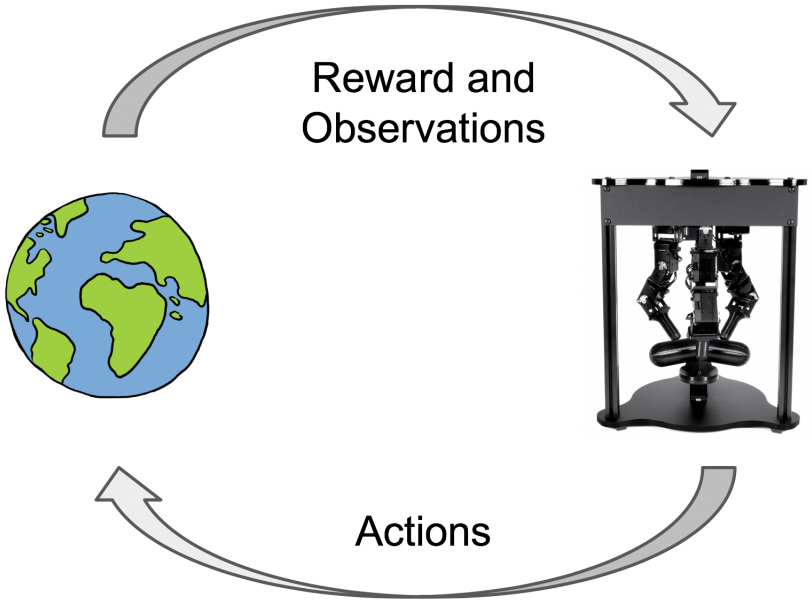

Figure 1: Illustration of the reinforcement learning problem: the environment (on the left) interacts with the agent (on the right) providing information on its state, also known as observation, and a reward signal. The agent maps these inputs into an action in the environment.

2.1 The Reinforcement Learning problem

Given an agent and its environment, the goal of a reinforcement learning (RL) problem can be informally stated as learning how to map situations in the environment into actions of the agent, in order to maximize a scalar reinforcement signal, namely the reward (Figure 1). For now, let the definition rely on an intuitive idea of agent and environment.

Reinforcement learning can be seen as a branch of machine learning, along with supervised and unsupervised learning. In particular, it differs from the other paradigms in terms of:

- Lack of supervision in data: only a reward signal is given.
- Delayed feedback to actions, which have themselves long terms consequences.

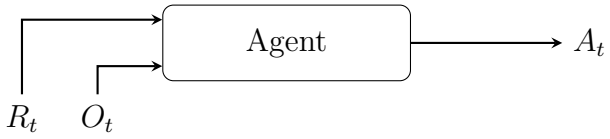

Figure 2: The agent uses its internal representation to map rewards and states into an action.

- Non-stationarity of the problem: data sampled at different times are not I.I.D (Independently Identically Distributed), since the agent's actions affect data it receives in the future.

2.2 Elements of the RL problem

Reward

A *reward* is a scalar feedback function used to quantify how good an agent is doing at a particular timestep t. It is used to define the goal of the agent, whose job is to maximize the cumulative reward over time.
In this sense, we can state the objective of the RL problem by means of the *reward hypothesis*:

> *All* goals can be described in terms of maximization of the cumulative expected reward.

Agent

The *agent* is a physical system inside an environment, in which it is able to take actions which are codified in an algorithm. Its inputs are given by O_t and R_t, respectively *observation* and *reward* at time t, and the output is the action taken A_t, as shown in Figure 2. (Capital notation is used to define these quantities as random variables.) The role of the agent is to learn a mapping from situations represented by observations and rewards into actions, to be applied at each timestep t.

Environment

Anything that is external to the agent is called the *environment*. In reinforcement learning, agent and environment interact in a loop (Figure 3) where the latter gets as input the action A_t and outputs R_{t+1} and O_{t+1}, the agent's input of the next iteration. The time t is incremented at every environment step.

State and History

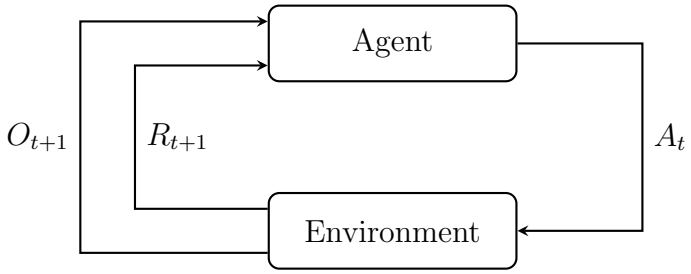

Figure 3: Agent-Environment interaction loop.

The notions of *state* and *history* are crucial in reinforcement learning. The history H_t is the sequence of observation, reward and action triplets up to time t:

$$H_t \coloneqq O_1, R_1, A_1, O_2, R_2, A_2, \ldots, A_{t-1}, O_t, R_t \,. \tag{2.1}$$

The state is the information used to determine what happens next (in terms of triplet at timestep $t+1$), and it is formally defined as a function of the history

$$S_t \coloneqq f(H_t) \,. \tag{2.2}$$

In practice, we can distinguish between the environment and the agent state.

Environment State
The environment state S_t^e is the environment private internal representation of information, *i.e.* whatever data are used to pick the next observation and reward. Usually it is not visible to the agent.

Agent State
The agent state S_t is the information used to pick the next action by the agent, given the reward and the observation input.

2.3 Finite Markov Decision Process

MDPs are a formalization of an RL environment, where states satisfy the Markov property and the environment is said to be fully observable.

2.3.1 Markov Property

Along with the two definitions of states given in Section 2.2, we can provide a third, formal one known as *information* or *Markov state*: a state S_t is said to be a Markov state if it contains all the useful information from the history, *i.e.* if it is a sufficient statistics of the future. In this sense, a Markov state

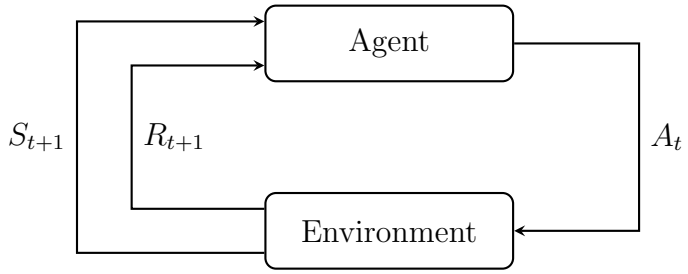

Figure 4: Agent-Environment interaction loop for fully observable environment.

fully characterizes $\mathbb{P}$, the probability distribution of transition from one state to another:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, S_3, \ldots, S_t] \,. \tag{2.3}$$

In words we say that the future is independent of the past given the present.

2.3.2 Fully Observable Environments

In a *fully observable environment* an agent directly observes the environment state, leading to

$$O_t = S_t^e = S_t \,, \tag{2.4}$$

with the equivalence between agent and environment states. The Agent-Environment interaction loop is displayed in Figure 4.

2.3.3 Finite MDP

We start by defining the Markov process or chain as a memoryless sequence of random states satisfying the Markov property. A Markov chain's dynamic is characterized by the tuple $< \mathcal{S}, \mathcal{P} >$, being $\mathcal{S}$ the set of possible states, and $\mathcal{P}$ the transition matrix that defines transition probabilities from the current to the next state.

In the Markov Reward Process (MRP) we introduce two values to the tuple, which becomes $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$, where the $\mathcal{R}$ function is the expected reward given a state, and γ is the discount factor used in the cumulative reward definition.

At last, we define a finite Markov Decision Process (MDP) as the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ introducing decisions that are picked from the action set

$\mathcal{A}$. In summary:

- $\mathcal{S}$ is the finite set of states
- $\mathcal{A}$ is the finite set of actions
- $\mathcal{P}$ is the transition matrix, with elements

$$\mathcal{P}^a_{ss'} \coloneqq \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \tag{2.5}$$

- $\mathcal{R}$ is the expected reward function

$$\mathcal{R}^a_s \coloneqq \mathbb{E}[R_{t+1} | S_t = s, A_t = a] \tag{2.6}$$

- γ is the discount factor $\gamma \in [0, 1]$.

2.3.4 Goal and return

We are now ready to formalize the idea of the *reward hypothesis* introducing a quantity called the *return*, which is a mathematical expression for the cumulative reward:

$$G_t \coloneqq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1} \,, \tag{2.7}$$

where $\gamma \in [0, 1)$ guarantees the finiteness of the series. The discount γ is also a way to weight the value that the agent credits, at timestep t, to the observation of a reward in the future. For $\gamma = 0$ the agent is *myopic*, being concerned only with maximization of the immediate reward. On the other hand, for $\gamma \approx 1$ we have a *far sighted* agent accounting more for future rewards.
Note that in (2.7) the summation goes up to infinity, underlying a *continuing task* in which the agent-environment interaction does not break down in separate batches of time, called episodes. Instead if we can identify a termination state S_T, remarking the end of the episode, the return is defined up to time T, and we will refer to an *episodic task*.

2.4 Inside an RL agent

Given the elements introduced so far, we can outline some core components characterizing a reinforcement learning agent: a policy, the value functions and a model.

2.4.1 Policy

A policy π is the agent behaviour, expressed as a map from states to actions. It can be both deterministic, with $a = \pi(s)$, or stochastic. In the latter case, π is the distribution over actions given the state

$$\pi(a|s) \coloneqq \mathbb{P}[A_t = a|S_t = s] \,. \tag{2.8}$$

The policy fully defines the agent's behaviour, and given the Markov property of the states in an MDP, we need to condition only on the last observation rather than on the full history.

2.4.2 Value Function

The *state value* function for an MDP is the expected return from state s under policy π:

$$v_\pi(s) \coloneqq \mathbb{E}[G_t|S_t = s] \,, \tag{2.9}$$

while the *action value* function is also conditioned on the action taken,

$$q_\pi(s, a) \coloneqq \mathbb{E}[G_t|S_t = s, A_t = a] \,. \tag{2.10}$$

Value functions quantify the goodness of a state (or state-action pair) at a given time instant. In the simplest case, one can assume these functions to be lookup tables, with a value associated to each state or state-action pair.

2.4.3 Model

A model is a prediction of what the environment will do next. It is defined if the agent can access $\mathcal{P}$ and $\mathcal{R}$ of the MDP. Whenever this condition is met, the agent can perform computations and simulations about the environment behaviour without any external interaction, in order to improve its policy. This is a well known sequential decision making problem called *planning*, in which case the agent is said to be *model-based*. Being the model an optional component, an agent can also be *model-free*, and it needs to interact with the environment in order to improve its policy. This is known as the *learning* problem.

2.4.4 Optimality

Most of the time RL algorithms involve the estimate of the value function, which is used to find the optimal policy with respect to the agent's goal. In order to do so, we first need to define an ordering between policies, such that we can formally say that one is better than another.

First, we define the *optimal value functions* as the maximum value function $v_*(s)$ and action value function $q_*(s,a)$ over the policy space:

$$v_*(s) \coloneqq \max_\pi v_\pi(s) \,, \tag{2.11}$$

$$q_*(s,a) \coloneqq \max_\pi q_\pi(s,a) \,. \tag{2.12}$$

Now, given two policies π and π', the ordering over them is expressed as

$$\pi' \geq \pi \iff v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S} \,, \tag{2.13}$$

and the following theorem holds:

Theorem. *For any MDP, there exists at least one deterministic optimal policy π_* that satisfies $\pi_* \geq \pi$, $\forall \pi$. All optimal policies achieves optimality of the state and action value functions, where $v_{\pi_*}(s) = v_*(s)$ and $q_{\pi*}(s,a) = q_*(s,a)$* [1].

From (2.12) we can intuitively see how solving the MDP is equivalent to finding $q_*(s,a)$, since we know at each state what is the optimal action to pick. This is achieved maximising over $q_*(s,a)$:

$$\pi_*(a|s) \coloneqq \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s,a) \\ 0 & \text{otherwise} \,. \end{cases} \tag{2.14}$$

2.5 Bellman Equations

It is possible to decompose value functions from (2.9) and (2.10) in a recursive way. The decomposition results in the so called *Bellman Expectation equations* [3]

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma\, v_\pi(S_{t+1}) | S_t = s] \,, \tag{2.15}$$

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma\, q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \,, \tag{2.16}$$

which can be easily proved by expanding the return term and exploiting the independence of the rewards random variables.

Moreover, we use the *Bellman Optimality equations* to recursively relate the value functions v and q:

$$v_*(s) = \max_a q_*(s,a), \forall s \in \mathcal{S} \,, \tag{2.17}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \,. \tag{2.18}$$

The optimality equations (2.17) and (2.18) are non-linear due to the *max* operator. Since in general there is no closed form solution, we introduce *Dynamic Programming* solution methods for the RL problem.

2.6 Dynamic Programming

Dynamic programming (DP) is a collection of algorithms to find optimal solutions for complex problems by breaking them down into simpler sub tasks. In particular, Dynamic Programming can be applied to problems that have two properties:

1. Optimal solutions can be decomposed into subproblems for which *optimality principle* applies, where according to the principle pieces of optimal solutions are themselves optimal.

2. The identified subproblems recur many time, and their solutions can be cached and reused.

Considering an MDP, both properties are satisfied, with Bellman Equations providing a recursive decomposition, and being value functions a solution that can be stored and reused on the next recursive call.
This class of algorithms can be used to solve a finite MDP only with the assumption of perfect knowledge, *i.e.* only when a model is given. This hypothesis is strongly limiting for real scenarios, nevertheless DP algorithms introduce fundamental concepts for the resolution of reinforcement learning tasks. The key idea is to use value functions in order to define the policy. Underlying this approach there are two distinct problems, which are *policy evaluation* or *prediction*, in which given a policy π we must find the value function v_π, and *control* task, where the goal is to find optimal $v_*(s)$ and π_* for the MDP.

2.6.1 Policy Evaluation

We first consider the *prediction problem* of computing the state-value function corresponding to a given policy π. We start picking an approximate value function v_0 of π arbitrarily, for example fixing $v_0(s) = 0$, $\forall s \in \mathcal{S}$. Then,

we use the Bellman Expectation equation (2.15) as *update rule* of the value function, with k being the counter of the update steps:

$$\begin{aligned} v_{k+1}(s) &\coloneqq \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right) \end{aligned} \tag{2.19}$$

In policy evaluation equation (2.19) is applied iteratively:

$$v_0 \to v_1 \to v_2 \to \ldots \to v_\pi \ ,$$

and convergence to real v_π can be proved.
Note that (2.19) updates the current estimate using another estimate, doing a one step lookahead over states: this technique is commonly used in reinforcement learning algorithms, making it worth the mention, and goes by the name of *bootstrapping*.
The pseudo code of the resulting algorithm is:

Algorithm 1: Iterative Policy Evaluation

Input policy π
Parameter θ: threshold for accuracy of the estimate

Initialize $v(s), \ \forall s \in \mathcal{S}$;
repeat
  foreach $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$;
    $V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$;
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
until $\Delta < \theta$;

2.6.2 Policy Improvement

Policy improvement is the fundamental step for iterative control algorithms. The general idea is that, given a policy π, we can first evaluate it by finding v_π. Then, given v_π, we can improve π making it *greedy* with respect to the computed value function.
A step of policy improvement is declined as follow:

1. evaluate π iteratively until convergence to v_π

2. define improved policy $\pi' \coloneqq \text{greedy}(v_\pi)$.

We can prove that *greedifying* a policy with respect to its value function guarantees improvement, unless the policy is already optimal. Formally, the greedy version of the current policy π for an action s is

$$\pi'(s) \coloneqq \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a) \,. \tag{2.20}$$

This improves the action value from any state s, being satisfied the condition

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s) \,, \tag{2.21}$$

which implies

$$v_{\pi'}(s) \geq v_\pi(s), \quad \forall s \in \mathcal{S} \,. \tag{2.22}$$

The implication of Equation (2.22) from (2.21) is proved by

$$\begin{aligned}
v_\pi(s) &\leq q_\pi\left(s, \pi'(s)\right) \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, \pi'\left(S_{t+1}\right)\right) \mid S_t = s\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}\left[R_{t+2} + \gamma v_\pi\left(S_{t+2}\right)\right] \mid S_t = s\right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi\left(S_{t+2}\right) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi\left(S_{t+3}\right) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s) \,,
\end{aligned}$$

Recalling policy ordering defined in (2.13), we see that π' improves π. This result is known as *policy improvement theorem*.
Improvement stops when we substitute $\geq$ with the equality sign in (2.21), which becomes

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s) \,. \tag{2.23}$$

This is exactly the Bellman Optimality equation (2.17) for the value function, which is by definition true if and only if $\pi = \pi_*$.

2.6.3 Policy Iteration

A policy iteration algorithm aims at finding an optimal policy by iteratively repeating an evaluation step as defined in Algorithm 1 and a policy improvement step towards the computed value function. The pseudo code of the algorithm is provided in Algorithm 2 box below.

Algorithm 2: Policy Iteration

Initialize $v(s)\ \forall s \in \mathcal{S}$ and $\pi(s) \in \mathcal{A}(s)$
Parameter θ: threshold for accuracy of the estimate

policy-stable $\leftarrow$ *true*;
repeat
  `/* Policy Evaluation */`
  repeat
    foreach $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$;
      $V(s) \leftarrow \sum_a \pi(a|s)(\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a V(s'))$;
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end
  until $\Delta < \theta$;
  `/* Policy Improvement */`
  foreach $s \in \mathcal{S}$ **do**
    old-action $\leftarrow \pi(s)$;
    $\pi(s) = \underset{a}{\operatorname{argmax}}(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_*(s'))$;
    if $\pi(s) \neq$ *old-action and policy-stable* **then**
      policy-stable $\leftarrow false$;
    end
  end
until *policy-stable*;

Graphically it results in

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \dots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

where convergence is guaranteed asymptotically, since the number of available policies is finite.

We call *Generalized Policy Iteration* (GPI) the idea of letting policy evaluation and policy improvement processes interacting in order to reach conver-

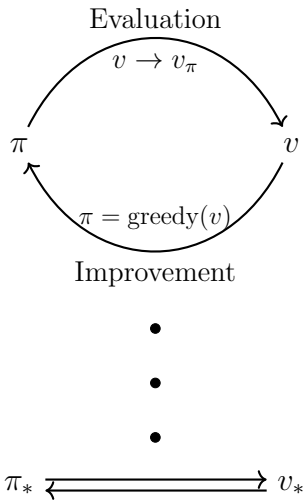

Figure 5: Generalized Policy Iteration.

gence to an optimal policy. A graphical representation (without any particular rigorous meaning) of the idea is given in Figure 5.

2.7 Monte Carlo methods

In GPI, we have described the general idea supporting prediction and control in reinforcement learning. Applying dynamic programming algorithms we need to assume a model of the MDP to be provided, such that the agent can access the distribution of next state and reward given an action: being this condition not always satisfied, we need to introduce some methods which can be exploited in practice in a *model-free* fashion.

2.7.1 Monte Carlo prediction

In the prediction problem we have a given policy and the goal is to estimate its value function. Monte Carlo methods perform the task by averaging over sampled returns: being the value function defined as the expectation of the return random variable (2.9), the average is an unbiased estimate of this quantity.
Monte Carlo prediction consists of sampling a full episode, storing the sampled return for every state. Defining the quantities $N(s)$ number of visits to a state during all sampled episodes, and $S(s)$ the incremental total return of a state, we estimate the value function as

$$V(s) = S(s)/N(s) \,. \tag{2.24}$$

Being the episodes independent and identically distributed, by the law of large numbers we have

$$V(s) \to v_\pi(s) \quad \text{for} \quad N(s) \to \infty \,. \tag{2.25}$$

Monte Carlo method is sample inefficient, since estimates are updated only once for every sampled episode. Moreover it is limited to episodic tasks, resulting not suitable for online learning (i.e. for learning during the episode rather than at the end of it).

2.7.2 Monte Carlo Control

We can use Monte Carlo methods, taking the average of the sampled returns as value function estimates, to solve the control problem. Equation (2.19) can be rewritten in a general incremental form as follow

$$NewEstimate = OldEstimate + \alpha(Target - OldEstimate) \,, \tag{2.26}$$

with $\alpha \geq 0$ being a constant hyper parameter and $(Target - OldEstimate)$ the *error* in the current estimate.
Monte Carlo methods defines the target in the error using the sampled return, obtaining the following incremental update rule of V(S) average value:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \,, \tag{2.27}$$

At this point executing the *policy improvement* step in GPI, we greedify the policy with respect to the new value function estimate, obtaining:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \, \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s') \,. \tag{2.28}$$

The problem with (2.28) is that we can not access $\mathcal{R}$ and $\mathcal{P}$, being Monte Carlo a model-free approach. Rather than using an estimate of the value function, we can equivalently use Monte Carlo prediction method to estimate the *action value* function $Q(s, a)$ with the sample average using (2.26), and replace it into (2.28), which becomes

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \, Q(s, a) \,. \tag{2.29}$$

2.8 Temporal Difference Learning

Temporal Difference (TD) Learning is a class of model-free methods, where the agent learns directly from sampled experience. The difference with re-

spect to Monte Carlo based methods is that learning happens from incomplete episodes, by a technique called *bootstrap*, which is commonly used in reinforcement learning.

2.8.1 TD Learning prediction

Bootstrap technique can be stated as an *update of a guess towards a guess*. During policy evaluation we use the rule in (2.26) to update the value function estimate towards a target: in the TD Learning version of GPI, the target consists of the sum between the observed reward and the previous estimate of the value function itself. The resulting rule is

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \,, \tag{2.30}$$

where

$$R_{t+1} + \gamma V(S_{t+1}) \tag{2.31}$$

is called the *TD target*, and is used as an estimate of the return.

With respect to Monte Carlo policy evaluation, TD learning value function approximation introduces sample noise just once for each update (with the value of R_{t+1}). Moreover, as already suggested, learning can happen in non episodic tasks and in an online fashion, i.e. during the sampled episode itself.

2.8.2 TD Learning control: SARSA

The idea is to use TD Learning in GPI control loop. As described for Monte Carlo method, maximization over an action requires an estimate of action value functions rather than the approximation of $V(S)$. Applying TD on $Q(S, A)$ to perform updates at each time step, combined with *epsilon* greedy policy improvement is a control algorithm known as *SARSA*. The incremental update of the action value function can be written as

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S, A)) \,. \tag{2.32}$$

3 Quantum Mechanics and Quantum Computation

Preliminarly to the introduction of the designed reinforcement learning algorithm for quantum engineering, we devote this section to review the basics of quantum mechanics and quantum computation, whose content reference can be found at [4].

3.1 The postulates of quantum mechanics

Quantum mechanics is the description of the behavior of matter and light in all their details and, in particular, of the happenings on an atomic scale [5]. The link between the physical world and this theory is given by the basic postulates of quantum mechanics, which will be described in the following paragraphs.

3.1.1 State space

The first postulate provides a description of quantum physical systems.

Postulate 1. Any isolated physical system is associated to a complex vector space with inner product (i.e. an Hilbert space), known as the state space. Our information about a physical system is encoded in a state vector, which is a unit vector in the system's state space.

This postulate does not tell anything about the state space associated to a *specific* physical system, neither about the actual state vector describing it. This postulate can be immediately applied to the simplest quantum system, the *quantum bit*, or qubit for short: a qubit is associated to a two-dimensional complex space, where it is described using the *bra-ket* notation by the state vector

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle \ . \tag{3.1}$$

This is a superposition of the elements in the computational basis $\{|0\rangle, |1\rangle\}$ which is nothing but a linear combination in the form $\sum_i \alpha_i \psi_i$, with the α_i scalar known as the *amplitude* associated to the state ψ_i.
By Postulate 1 we know that $|\psi\rangle$ must satisfy $|\langle\psi|\psi\rangle| = 1$, which is equivalent to $|a|^2 + |b|^2 = 1$. This is known as the *normalization condition* of state vectors.

3.1.2 Time Evolution

The second postulate of quantum mechanics provides a description of the evolution in time of the state vector of a quantum system.

Postulate 2. The dynamic of a closed quantum system is described by a unitary transformation. That is, in the discrete time case, the state $|\psi\rangle$ at time t_1 is associated to $|\psi'\rangle$ at time t_2 by a unitary transformation U depending only on t_1 and t_2, such that

$$|\psi'\rangle = U\,|\psi\rangle \ . \tag{3.2}$$

A meaningful example of unitary operators on a qubit is given by the set

$$\sigma_X \coloneqq \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.3}$$

$$\sigma_Y \coloneqq \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{3.4}$$

$$\sigma_Z \coloneqq \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \ , \tag{3.5}$$

known as the *Pauli matrices*, which also includes the identity matrix I.

The content of Postulate 2 can be reformulated in continuous time using the Schrödinger equation. Given a closed quantum system with state vector $|\psi\rangle$, its evolution in time is described by

$$i\hbar\frac{\partial}{\partial t}\,|\psi\rangle = H\,|\psi\rangle \ , \tag{3.6}$$

with H being the Hamiltonian of the closed system, which we assume to be time independent.

Note that Postulate 2, in both its discrete and continuous time formulations, applies to closed systems. On the other hand, in quantum mechanics we often speak of *applying* a unitary operator to a physical system, for example the qubit: this implies the interaction of an external subject with the quantum system, appearing to be contradictory with the closeness requirement. The upshot is that, despite the interaction with the environment, there are situations in which with good approximation a system can be assumed to be closed and Postulate 2 can be applied. Nevertheless this consideration raises the need of a time evolution description able to account for the dynamics of open systems.

3.1.3 Quantum Measurements

Imagine an experimentalist and its equipement interacting with a quantum system: in this setting, the assumption of *closeness* can not be considered satisfied, requiring a new description of the system evolution, which is no longer necessarily unitary. To this end, we define Postulate 3, describing the effect of a quantum measurement operator. Let **M** be a quantum measurement device acting on a system $|\psi\rangle$:

Postulate 3. The action of the associated quantum operator M is defined by a set of *measurement operators* $\{M_m\}$ acting on the Hilbert space of interest, with $\{m\}$ being the set of possible measurement outcomes. Given a quantum system in state $|\psi\rangle$ immediately before applying the measurement, then the probability of outcome m occurring is

$$p(m) = \langle\psi| M_m^+ M_m |\psi\rangle \ , \tag{3.7}$$

and the state of the system after the measurement, given the outcome m, is

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^+ M_m |\psi\rangle}} \, . \tag{3.8}$$

The set of measurement operators must satisfy the completeness equation

$$\sum_m M_m^+ M_m = I \ , \tag{3.9}$$

in order to guarantee that the sum of probabilities over all outcomes is normalized to 1.

This formulation of the postulate provides a description of

1. The probability of a specific outcome of a measurement.
2. The evolution of the system after applying the quantum measurement, which is not necessarily unitary.

There are situations in which we are not interested in the post-measurement evolution of the quantum system. To this end, we introduce the POVM (Positive Operator-Valued Measures) formalism, which can be simply derived as a consequence of Postulate 3. Suppose we have a measurement and its operators $\{M_m\}$ associated to it, acting on the system in state $|\psi\rangle$. Let us define

$$E_m \coloneqq M_m^+ M_m \, . \tag{3.10}$$

Then, from Postulate 3 we have that the $p(m) = \langle\psi| E_m |\psi\rangle$ and that completeness equation $\sum_m E_m = I$ is satisfied. Moreover, being E_m positive semi-definite from simple linear algebra considerations, the POVM elements of the measurement are non-negative operators, i.e. they satisfy $\langle\psi| E_m |\psi\rangle \geq 0$.

We finally introduce the last special class of measurements, known as *projective measurements*.
Let the observable M be an Hermitian measurement operator such that it has spectral decomposition

$$M = \sum_m m P_m \,, \tag{3.11}$$

with m eigenvalues of the matrix representation and P_m eigenvectors, the projectors on the m eigenspace of M. Given the quantum system with state $|\psi\rangle$, the probability of measurement outcome equals to m is

$$p(m) = \langle\psi| P_m |\psi\rangle \ , \tag{3.12}$$

with the post-measurement state of the quantum system equals to

$$|\psi_m\rangle \coloneqq \frac{P_m |\psi\rangle}{\sqrt{p(m)}} \,. \tag{3.13}$$

Recalling Postulate 3, defining a projective measurement is equivalent to add restrictions on $\{M_m\}$ operators such that they are Hermitian and that $M_m M_{m'} = \delta_{mm'} M_m$. We can easily see that this constrained version of measurement operators, which must also satisfy the completeness equation, specifies orthogonal projectors on the eigenspace of m, satisfying (3.12) and (3.13). This way we can eventually define projective measurements as special case of the general measurement postulate.

Some nice properties make projective measurements of particular interest: given a projector P_m, one has that

$$P_m P_n = \delta_{mn} P_m \,, \tag{3.14}$$

$$P_m P_m = P_m \,. \tag{3.15}$$

These equations tell us that given a projective measurement P_m on a system in state $|\psi\rangle$, then successive projective measurements recover the same state $|\psi_m\rangle$ again. This property is known as *repeatability*.
Moreover, it is straightforward to calculate the expected value for projective measurements M:

$$\begin{aligned}\mathbb{E}(M) &= \sum_m m\ p(m) = \\ &= \sum_m m \langle\psi| P_m |\psi\rangle = \\ &= \langle\psi| \sum_m m\ P_m |\psi\rangle = \\ &= \langle\psi| M |\psi\rangle\ .\end{aligned} \tag{3.16}$$

To conclude our discussion on quantum measurements, we remark the fact that exploting equations (3.10) and (3.15), we can write $E_m = P_m$, thus making projective measurements a special case of POVM.

3.1.4 Composite Systems

At this point, we want to consider a composite system, made of two or more distinct physical systems: in order to proceed we need to define a formalism describing the resulting state vector and the new Hilbert space.

Postulate 4. The state space of a composite system is given by the tensor product of the state spaces of the component physical subsystems, namely $\otimes_i^n H_i$, with H_i being the Hilbert space associated to a single particle.

Note that, while the resulting space is always the tensor product of the subspaces, it is not always possible to represent the composite system state vector as the tensor product of the state vectors associated to the single particles. Given a basis set $\{|i\rangle\}$ for each H_i, the most general state is written as linear combination of those bases. For clarity, consider the Hilbert space of a composite system being $H_A \otimes H_B$, with subspaces respectively associated to bases $\{|i_A\rangle\}$ and $\{|j_B\rangle\}$: then, any vector in the resulting space can be written as $\sum_{i,j} c_{ij} |i_A\rangle \otimes |j_B\rangle$, such that $\sum_{i,j} |c_{ij}|^2 = 1$.
As a consequence, being $|H_A|$ and $|H_B|$ the cardinality of the subspaces, the composite state space is characterized by dimensionality equals to $|H_A| \cdot |H_B|$.

At this point, it useful to recall some properties of the vector space resulting from a tensor product. Note that as a matter of notation, we define as perfectly equivalent $|\psi\rangle \otimes |\phi\rangle$, $|\psi\rangle\,|\phi\rangle$ and $|\psi\phi\rangle$:

1. For $|\psi\rangle \in H_A$, $|\phi\rangle \in H_B$ and arbitrary scalar α, then

$$\alpha\,|\psi\rangle\,|\phi\rangle = |\alpha\psi\rangle\,|\phi\rangle = |\psi\rangle\,|\alpha\phi\rangle \tag{3.17}$$

2. Given $|\psi_1\rangle, |\psi_2\rangle \in H_A$ and $|\phi\rangle \in H_B$, then

$$(|\psi_1\rangle + |\psi_2\rangle)|\phi\rangle = |\psi_1\rangle|\phi\rangle + |\psi_2\rangle|\phi\rangle \tag{3.18}$$

3. Given operators T on H_A, S on H_B, and the state vectors $|\psi\rangle \in H_A$ and $|\phi\rangle \in H_B$, we have a corresponding operator $T \otimes S$ on $H_A \otimes H_B$, such that

$$\begin{aligned}(T \otimes S)(|\psi\rangle \otimes |\phi\rangle) &= (T \otimes I_{H_B})(I_{H_A} \otimes S)(|\psi\rangle \otimes |\phi\rangle) \\ &= T|\psi\rangle \otimes S|\phi\rangle\end{aligned} \tag{3.19}$$

with I_{H_A} and I_{H_B} identity operators on H_A and H_B respectively.

3.1.5 Phase

Now that we have introduced the four postulates of quantum mechanics, we can move forward into the description of the key concepts associated to quantum computation. But before doing so, let us introduce the notion of *global phase*.

Consider the state $e^{i\theta}|\psi\rangle$, where i is the imaginary unit and θ is a real valued number: we state that the two vectors $|\psi\rangle$ and $e^{i\theta}|\psi\rangle$ are physically equal. The meaning of this claim is that the *measurement statistics* associated to the two states are the same. We can easily proof this directly applying (3.7) to both vectors:

$$\begin{aligned}p(m) &= \langle\psi| M_m^+ M_m |\psi\rangle \\ p(m) &= \langle\psi| e^{-i\theta} M_m^+ M_m e^{i\theta} |\psi\rangle = \langle\psi| M_m^+ M_m |\psi\rangle \ ,\end{aligned}$$

where the two expressions turn out to be exactly the same.

3.2 Quantum Computation

At this point, we have all of the quantum mechanics knowledge we need to introduce the main concepts of quantum computation. We are going to start doing so by presenting the basic quantum computation model, namely, the quantum circuit. Then, we will provide a definition of quantum gates and of universal set of gates.

3.2.1 The qubit

We already introduced the qubit as the simplest quantum system in a two-dimensional state space. Recall that in the computational basis it is defined as the superposition

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle \ . \tag{3.20}$$

In addition, we mention that a qubit can be visualized in the so called *Bloch sphere*. Given the amplitudes coefficients a and b subject to the normalization condition $|a|^2 + |b|^2 = 1$, and the Euler formula $e^{ix} = \cos(x) + i\ \sin(x)$, we can rewrite Equation (3.20), up to a global phase factor, as

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle \ , \tag{3.21}$$

where the real numbers θ and ϕ define a point $(\sin\theta\cos\phi,\ \sin\theta\sin\phi,\ \cos\theta)$ on a unit three-dimensional sphere, the Bloch sphere (Figure 6).

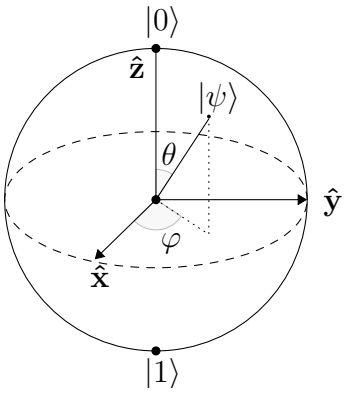

Figure 6: Bloch sphere representation of a single qubit.

3.2.2 Single qubit operators

Operations on a single qubit must preserve the normalization condition over the amplitudes: this is guaranteed defining single qubit operators by 2×2 unitary matrices, since they preserve the inner product between state vectors. Thus, given a unitary U applied to state vector $|\psi\rangle$, then $|\langle\psi|\,U^{+}U\,|\psi\rangle| = |\langle\psi|\psi\rangle| = 1$ holds.

We already introduced a set of unitary operators, known as the *Pauli matrices*, with equations (3.3), (3.4), (3.5). Additionally, we define other three

important quantum gates, that are the Hadamard gate H, the phase gate S and the $\pi/8$ gate T:

$$H \coloneqq \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad S \coloneqq \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; \quad T \coloneqq \begin{bmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{bmatrix}. \tag{3.22}$$

From the above, it is straightforward to verify that $H = (X + Z)/\sqrt{2}$ and $S = T^2$.

A useful class of unitary matrices is the one of the *rotation operators*, which arises when exponentiating Pauli matrices. They are defined by the following equations:

$$R_x(\theta) \coloneqq e^{-iX\frac{\theta}{2}} = \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \tag{3.23}$$

$$R_y(\theta) \coloneqq e^{-iY\frac{\theta}{2}} = \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \tag{3.24}$$

$$R_z(\theta) \coloneqq e^{-iZ\frac{\theta}{2}} = \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} Z = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}. \tag{3.25}$$

In general, we can combine these three operators to get a rotation θ around arbitrary axis $\hat{n} = (n_x, n_y, n_z)$ unit vector with:

$$R_n(\theta) \coloneqq \exp\left(-i\hat{n}\cdot\vec{\sigma}\,\frac{\theta}{2}\right) = \cos\frac{\theta}{2} I - i\sin\frac{\theta}{2}\left(n_x X + n_y Y + n_z Z\right) \tag{3.26}$$

with $\vec{\sigma}$ the vector of Pauli matrices (X, Y, Z).

The $R_n(\theta)$ rotation matrix is useful because any single qubit unitary operator U can always be written as the product of a global phase factor and a rotation of an angle θ around an axis $\hat{n}$:

$$U = e^{i\alpha} R_{\hat{n}}(\theta). \tag{3.27}$$

The above is a general formulation of the special case of *Z-Y decomposition for a single qubit*: according to this, given a unitary operator U on a single qubit, there exist real numbers $\alpha, \beta, \gamma, \delta$ such that

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta). \tag{3.28}$$

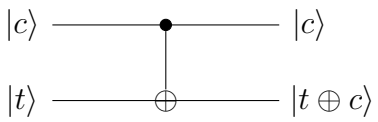

Figure 7: $CNOT$ gate in the computational basis.

This last result can be used to derive that any unitary operator on a single qubit can be decomposed with rotations around arbitrary unit, non parallel axes $\hat{m}$ and $\hat{n}$ such that, for appropriate choices of $\alpha, \beta, \gamma, \delta$, the following is true:

$$U = e^{i\alpha} R_n(\beta) R_m(\gamma) R_n(\delta) \,, \tag{3.29}$$

where this result will be used in later analysis of the error introduced by gate approximation through a universal set.

3.2.3 Controlled operations

Controlled operations instructions can be expressed in the form "If A is true, than do B". To understand how these operations work in a quantum circuit, we start by introducing the most notable controlled gate, the $CNOT$ gate. It is characterized by two input qubits, known as the *target* and the *control*. Consider the case of qubits in the computational basis: the action of $CNOT$ on control $|c\rangle$ and target $|t\rangle$ is $|c\rangle |t\rangle \to |c\rangle |c \oplus t\rangle$, as represented in Figure 7. This is equivalent to applying the quantum NOT gate X to the target whenever the control is set to $|1\rangle$, else an identity when the control is $|0\rangle$. The matrix representation of the $CNOT$ gate is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} .$$

Starting from $CNOT$ formalization, we can generalize to a definition for any controlled gate U_c on a single qubit. The effect of U_c is such that $U_c |c\rangle |t\rangle \to |c\rangle U_c |t\rangle$ when the control is set, else an identity.
Up to now, we considered only controlled operations involving a single qubit as control, with unitary gate acting on a single target. This is a special case of the more general condition in which we have n control qubits along with k targets, the input qubits of the controlled operator. An example with $n = 4$ and $k = 3$ is provided in Figure 9. The resulting circuit can be formulated

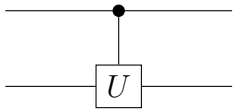

Figure 8: Controlled operation circuit.

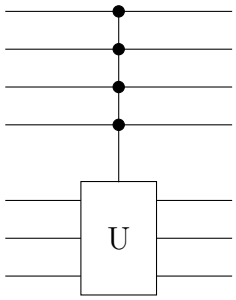

Figure 9: Controlled operations on multiple qubits.

defining the controlled operator $C^n(U)$ such that $C^n(U)\,|x_1x_2\ldots x_n\rangle\,|\psi\rangle = |x_1x_2\ldots x_n\rangle\,U^{|x_1x_2\ldots x_n\rangle}\,|\psi\rangle$, where U is applied on $|\psi\rangle$ only if $x_1, x_2, \ldots, x_n$ product factors are all set to one.

As a final remark, note that there is nothing special in the control qubit being set to one: a controlled gate can as well implement the logic "If A is false, than do B", that is equivalent to applying a unitary gate to the target if and only if the control is set to zero. The equivalent quantum circuit notation is provided in Figure 10.

3.2.4 Universal quantum gates

We can now introduce the concept of universal quantum gates. Similarly to what happens in a classical setting, a universal set of gates for a quantum circuit is a finite set of operators such that a circuit made only of those elements can be used to approximate at an arbitrary accuracy level any unitary operator. In particular, our goal is to show that the Hadamard, $CNOT$, phase and $\pi/8$ gates form together a universal set, and to provide insights on the error introduced by the resulting approximated implementation.

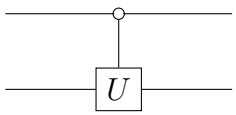

Figure 10: Controlled operation circuit.

To explore the constructions behind our claim, we start by introducing two-level unitary operators: a two-level unitary matrix on an n-dimensional space acts non-trivially on the space spanned by only two or less basis vectors. Now, we want to show that two-level unitary gates are a universal set: to do so we provide the intuition for the three-dimensional case, and assume generalization to any dimension to be true. Given a 3×3 unitary matrix U

$$U = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \tag{3.30}$$

we can find U_1, U_2, U_3 two-level unitary matrices such that

$$UU_1U_2U_3 = I \ , \tag{3.31}$$

meaning that

$$U = U_3^+U_2^+U_1^+ \ . \tag{3.32}$$

Thus, given that (3.31) can be proved, we have shown how to decompose any 3×3 unitary matrix into the product of two-level operators.
Generalizing, for any U on any d dimensional space, we can find a set of k two-level unitary gates V_i such that

$$U = V_1 \dots V_k \tag{3.33}$$

with $k \leq (d-1) + (d-2) + \ldots + 1 = d(d-1)/2$. An important corollary derived from the latter result is that any arbitrary unitary $n \times n$ matrix of an Hilbert space can be written as a product of at most $2^{n-1}(2^n - 1)$ two-level matrices, bounding implementation complexity to $O(4^n)$.

The second construction needed to define the universal set of quantum gates is based on the fact that any two-level unitary matrix on a n qubit system can be implemented by a combination of $CNOT$ and single qubit gates: by this claim, we state that $CNOT$ along with the union of all 2×2 operators is a universal set.

In the analysis of H, $CNOT$, phase and $\pi/8$ gates, as we keep building on previous constructions we need to add a final layer, that is: any single qubit gate can be approximated to arbitrary accuracy as a combination of H and T in a circuit. We can suggest the key points exploited to prove this claim considering the operators T and HTH. Composing these matrices, we get, up to a global phase factor,

$$\begin{aligned}
&\exp\left(-i\frac{\pi}{8}Z\right)\exp\left(-i\frac{\pi}{8}X\right) = \\
&\left[\cos\frac{\pi}{8}I - i\sin\frac{\pi}{8}Z\right]\left[\cos\frac{\pi}{8}I - i\sin\frac{\pi}{8}X\right] = \\
&\cos^2\frac{\pi}{8}I - i\left[\cos\frac{\pi}{8}(X+Z) + \sin\frac{\pi}{8}Y\right]\sin\frac{\pi}{8}\,,
\end{aligned} \tag{3.34}$$

where the result is a rotation around $(\cos\frac{\pi}{8}, \sin\frac{\pi}{8}, \cos\frac{\pi}{8})$ direction, with an angle θ such that $\cos\theta/2 \coloneqq \cos^2\pi/8$, meaning that we can construct $R_{\vec{n}}(\theta)$ simply combining $\pi/8$ and Hadamard gates.
Iterating the application of (3.34), it can be proved that $R_{\vec{n}}(\theta)$, with θ constrained as mentioned above, can be used to implement any arbitrary rotation $R_{\vec{n}}(\alpha)$. In the next section we formalize the error's definition along with some useful insights.

3.2.5 Approximation accuracy

Up to now we talked about how we can use Hadamard, phase, T and $CNOT$ gates to approximate a unitary operator with arbitrary accuracy. At this point, we still need to define what we mean by *arbitrary accuracy*, giving a notion of *error*. Suppose that two unitary matrices U and V on the same state space are given. The error introduced in a circuit implementing V in place of U is defined as

$$E(U,V) \coloneqq \max_{|\psi\rangle} |(U-V)|\psi\rangle|\,, \tag{3.35}$$

where the maximum is over all possible states $|\psi\rangle$ in the state space.
We can interpret the above equation by saying that any measurement performed on the state $V|\psi\rangle$ gives measurement statistics approximately equals to those returned by measures on $U|\psi\rangle$ if $E(U,V)$ is *small*. More precisely, imagine M_m to be a POVM element of a measure, associated to outcome m. Then, the difference between measurement statistics P_U for state $U|\psi\rangle$ and P_V for state $V|\psi\rangle$ is upper bounded as follow:

$$|P_U - P_V| \leq 2E(U,V)\,. \tag{3.36}$$

Thus, given $E(U,V)$ small, so is the difference between measurement statistics involving U and V. Moreover, in the general case where $U_1U_2\ldots U_n$ is approximated by $V_1V_2\ldots V_n$, then we have that

$$E(U_1U_2\ldots U_n, V_1V_2\ldots V_n) \leq \sum_{j=1}^{n} E(U_j, V_j)\,, \tag{3.37}$$

meaning that the error adds up at most linearly.
Putting together Equations (3.36) and (3.37), we can constraint the difference in probability of a measurement outcome between the original circuit and its approximation up to a maximum tolerance $\Delta > 0$. It is sufficient to guarantee $E(U_j, V_j) \leq \Delta/(2n)$ for each j, as we show below:

$$\begin{aligned} \frac{\Delta}{2n} \geq E(U_j, V_j), \;\; \forall j \Rightarrow |P_{U_1 U_2 \ldots U_n} - P_{V_1 V_2 \ldots V_n}| \\ \leq 2E(U_1 U_2 \ldots U_n, V_1 V_2 \ldots V_n) \\ \leq 2 \sum_{j}^{n} E(U_j, V_j) \\ \leq \Delta \, . \end{aligned} \tag{3.38}$$

We want now to consider the rotation defined in Equation (3.34): as we claimed, a repeated iteration of such $R_{\hat{n}}(\theta)$ can be used to approximate to arbitrary accuracy any rotation $R_{\hat{n}}(\alpha)$. This means that, for any $\epsilon > 0$ there exists a value n such that

$$E(R_{\hat{n}}(\alpha), R_{\hat{n}}(\theta)^n) < \frac{\epsilon}{3} \, , \tag{3.39}$$

with $\hat{n}$ having direction $(\cos\frac{\pi}{8}, \sin\frac{\pi}{8}, \cos\frac{\pi}{8})$. It is easy to see that

$$H R_{\hat{n}}(\alpha) H = R_{\hat{m}}(\alpha) \, , \tag{3.40}$$

with unit $\hat{m}$ directed as $(\cos\frac{\pi}{8}, -\sin\frac{\pi}{8}, \cos\frac{\pi}{8})$, from which it follows

$$E(R_{\hat{m}}(\alpha), R_{\hat{m}}(\theta)^n) < \frac{\epsilon}{3} \, . \tag{3.41}$$

At this point, we need to recall that, according to Equation (3.29), any single qubit unitary U can be decomposed as

$$U = e^{i\alpha} R_{\hat{n}}(\beta) R_{\hat{m}}(\gamma) R_{\hat{n}}(\delta) \, , \tag{3.42}$$

where the global phase factor can be dropped and $\hat{n}, \hat{m}$ are non parallel vectors. We can now compute the approximation error of U implemented with (3.42): applying to each factor Equation (3.39), chained with (3.40), and recalling the inequality in (3.37) for which the error adds up at most linearly, we show that, for suitable integer values n_1, n_2, n_3,

$$E(U, R_{\hat{n}}(\theta)^{n_1} H R_{\hat{n}}(\theta)^{n_2} H R_{\hat{n}}(\theta)^{n_3}) < \epsilon \, . \tag{3.43}$$

That is, we can control the approximation error for the implementation of an arbitrary single qubit unitary matrix U, using a circuit composed only of Hadamard and $\pi/8$ gates.

To summarize, we write down the chain of constructions that allow us to use H, T, S and $CNOT$ as universal set of quantum gates: first, we described how to use two-level unitary matrices to implement any $n \times n$ quantum gate. Than, we claimed that CNOT and single qubit operators can implement any two-level unitary matrix, making themselves a universal set. Finally, we showed how to use Hadamard and T gates to approximate any single qubit operator to an arbitrary error. Note that our universal set also includes the phase gate S, which can be implemented as the square of T: this is due to reasons that are beyond the scope of this thesis, so they will not be treated here.

3.3 Density matrix

As last topic of this chapter it is useful to introduce an extended formulation of quantum mechanics, which is not based on the language of state vectors. In Postulate 1 we have defined a state vector associated to a quantum system, but this is not the only possible representation we can exploit: the density operator (or density matrix) provides an equivalent description for a quantum system in an unknown state. Suppose in fact that a system is associated to one of the vectors belonging to the set $\{|\psi_i\rangle\}$, where each state is associated to a probability p_i. We can thus introduce the new set $\{p_i, |\psi_i\rangle\}$, which is called an *ensemble of pure states*. The resulting description of the quantum system associated is the density matrix ρ, defined as

$$\rho = \sum_i p_i \left|\psi_i\right\rangle \left\langle\psi\right| \ , \tag{3.44}$$

with $\sum_i p_i = 1$ by definition of probability. A density operator must satisfy the following three properties:

- $\mathrm{tr}(\rho) = 1$, which is known as the *trace condition*.
- ρ must be positive semi-definite.
- $\rho = \rho^+$ is *self-adjoint*.

The formalism allows to perform an important distinction: we talk about *pure states* for a quantum system associated to a state vector $|\psi\rangle$ with probability 1. In this case (3.44) becomes $\rho = |\psi\rangle \langle\psi|$. Else ρ is said to be a *mixed*

state, which is a mixture of pure states, each one associated to a probability value. To distinguish between these two conditions, we need to compute the quantity $\mathrm{tr}(\rho)^2$: this is 1 in case of pure states, < 1 when ρ is a mixture.

4 Quantum-enhanced Reinforcement Learning Algorithm

We now want to apply the reinforcement learning paradigm to a problem of interest in quantum computing. Given a quantum system in an initial state $|\psi\rangle$, we would like to know the sequence of unitary operations to be applied in order to transform it into $|\psi'\rangle$, which is *any* target state of interest. The goal of this thesis is to implement and train an RL agent able to design a quantum circuit transforming $|\psi\rangle$ into $|\psi'\rangle$. In this *control* problem the objective is to learn an optimal policy such that the number of gates applied in the circuit is minimized.
For the sake of simplicity, we assume $|\psi\rangle$ and $|\psi'\rangle$ associated to quantum states of two qubits.

4.1 Reinforcement learning set up

We proceed in the description of states, actions, environment and learning algorithm implemented to perform the task.

4.1.1 States and Actions

From equation (3.20), we can write any qubit as the linear combination of the basis spanning its space. Applying Postulate 4 we generalized this notion, for which the quantum state associated to two qubits can be expressed in the computational basis as:

$$|\psi\rangle = a_{00}\,|00\rangle + a_{01}\,|01\rangle + a_{10}\,|10\rangle + a_{11}\,|11\rangle \ , \tag{4.1}$$

given that the *normalization condition* is satisified. Also recall that coefficients a_{ij} are complex scalars: we use this consideration to define the state of an agent as the set of flat mapped tuples $\{(\mathrm{Re}(a_{ij}), \mathrm{Im}(a_{ij}))\}$, where real and imaginary parts are separated in order to operate always with real numbers in the algorithm. The resulting environment states become vectors in this form:

$$\begin{aligned} s = [&\mathrm{Re}(a_{00}), \mathrm{Re}(a_{01}), \mathrm{Re}(a_{10}), \mathrm{Re}(a_{11}), \\ &\mathrm{Im}(a_{00}), \mathrm{Im}(a_{01}), \mathrm{Im}(a_{10}), \mathrm{Im}(a_{11})] \ . \end{aligned} \tag{4.2}$$

It is intuitive to see that the resulting state space S is continuous, anticipating the infeasibility of using a tabular value function.

In order to train the agent to design a quantum circuit, we define the *actions* it takes as consisting of picking a *gate* U at each timestep t. The learned policy will therefore map a state s - set of amplitudes of the quantum state as in (4.2) - into a gate U. The set of gates (actions) defined in the algorithm consists of the $CNOT$ plus the rotation operators of an angle $\theta \in \{\pi, \frac{2\pi}{3}, \frac{\pi}{2}, \frac{\pi}{3}, \frac{\pi}{4}\}$ around the three axis $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$, acting on both the first and second qubits.

4.1.2 Environment

In our setting the environment is a fully observable MDP, as formalized in Equation (2.4): in fact, considering the loop in Figure 4, the agent takes an action after observing the environment state. We still need to specify the form of the reward and the discount factor in order to characterize the MDP:

- The task is undiscounted, meaning we set $\gamma = 0$.
- We define a discrete reward function, assigning positive value of +100 whenever the terminal state is reached, and a negative signal of -1 for all actions mapped into any other state. This choice reflects the definition of optimality we decided for our policy, which is the one minimizing the number of gates needed to transform the initial state into the target one. Thus, penalizing each step leading to a state different from the terminal, we push the agent to learn a policy minimizing the number of steps before reaching the solution. This is similar to the approach one would use to train an agent to reach a target state in a grid world: setting a negative reward for any action mapped into a state different than the terminal one incentivizes the agent to minimize the time spent inside of the grid world [6].

At this point, we need to provide a better definition of what we mean by *terminal state*, which in our setting is not equivalent to the target. We mark an episode as ended whenever the agent reach a state *similar enough* to the target: in quantum information theory, we measure the similarity of two probability distributions using *state fidelity*, which can be applied to two state vectors [7].
Given our target vector $|\psi'\rangle$ and the current vector $|\psi(t)\rangle$ at time t, we measure the fidelity between the two states as:

$$F(\psi', \psi(t)) = |\langle\psi'|\psi(t)\rangle|^2 \ , \tag{4.3}$$

which defines $F \in [0, 1]$. Since a fidelity exactly equals to 1 in general is not achievable by a finite set of gates, we define a *tolerance* parameter: this way,

any state whose fidelity with respect to the target is included in the range $[1 - tolerance, 1]$ is a terminal state. Increasing the tolerance decreases the lower bound of the interval of accepted solutions.

4.1.3 TD learning with linear function approximation

Now we need to provide a detailed description of the learning algorithm. We train the agent using *temporal difference learning with linear function approximation* algorithm. Being our task episodic, both Monte Carlo and TD Learning are suitable options, but we privilege the latter due to the better sampling efficiency.

As we mentioned in Section 4.1.1, it is not feasible to rely on the tabular approach on a continuous state space, i.e. storing the action value functions on a matrix whose entries are identified by the pair $(state, action)$. An alternative to this technique is to exploit some method to approximate $\hat{q}_\pi(S, A)$: among the many options available (neural networks, decision trees, ...) we opt for a linear approximator based on stochastic gradient descent. A description of the optimization algorithm is provided in Section 7.1 of the Appendix.

In order to estimate $\hat{q}_\pi(S, A)$ under a certain policy π, we introduce the set of parameters $\vec{w}$, the *weights* of the function. Moreover, we need to define a set of features to represent a state of the agent inside of the function. For our task, we simply map the state to itself, defining the features vector $\vec{x}(S)$ with the identity operator. This way the features correspond to the set of flat mapped tuples made of real and imaginary parts of the amplitudes. Then, we define the $\hat{q}_\pi(S, A, \vec{w})$ estimate as:

$$\hat{q}_\pi(S, A, \vec{w}) \coloneqq \vec{x}(S) \cdot \vec{w}(A) \tag{4.4}$$

with a different weight vector learned for each of the possible actions.

Now we need a differentiable value function to minimize, in order to learn meaningful weights: the algorithm implemented relies on the *mean square error* (MSE). If we imagine to have access to the real action value function $q_\pi(S, A)$ then we would minimize distance with our estimate as follow:

$$J(\vec{w}) = \mathbb{E}_\pi([q_\pi(S, A) - \hat{q}_\pi(S, A, \vec{w})]^2) \,, \tag{4.5}$$

with $J(\vec{w})$ objective function in stochastic gradient descent. The resulting update rule would be written as:

$$\Delta\vec{w} = \alpha(\hat{q}_\pi(S, A) - \vec{x}(S) \cdot \vec{w}(A)) \cdot \vec{x}(S) \,. \tag{4.6}$$

Since we can not access the real value of $\hat{q}_\pi(S, A)$, we bootstrap on current estimate as already done in TD learning, defining a new target $R_{t+1} + \gamma\hat{q}_\pi(S_{t+1}, A_{t+1}, \vec{w})$ for the *MSE*. Then we rewrite (4.5) as:

$$J(\vec{w}) = \mathbb{E}_\pi([R_{t+1} + \gamma\hat{q}_\pi(S_{t+1}, A_{t+1}, \vec{w}) - \hat{q}_\pi(S, A, \vec{w})]^2) \,, \tag{4.7}$$

while (4.6) becomes:

$$\Delta\vec{w} = \alpha(R_{t+1} + \gamma\hat{q}_\pi(S_{t+1}, A_{t+1}, \vec{w}) - \vec{x}(S)\vec{w}(A)) \cdot \vec{x}(S) \,. \tag{4.8}$$

4.1.4 Q-learning

In the *control* problem, the goal is to learn an optimal policy. This can be done learning a policy π from data (state to action transitions) sampled from π itself: this setting is called *on policy*, and is the only one introduced so far. Another option is to define two policies π and b, namely the target and the behavioural ones:

- π: it is the policy that we want to optimally learn. In the implemented algorithm, this is fully greedy, meaning that actions are taken as follow:

$$\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{q}(s, a, \vec{w}) \,.$$

- b: this is the policy used at training time to sample next action, given a state s. In order to ensure exploration, we decide to make $b(s)$ fully random: at each time step, an action is sampled uniformly among the set of available quantum gates.

An off policy TD algorithm is known as *Q-learning*. In Figure 11 we illustrate the weights learned by stochastic gradient descent, visualized in a heatmap.

4.2 Quantum Computation set up

We now want to discuss the implementation details of the actions and states of our algorithms, which respectively correspond to quantum gates and state vector amplitudes.

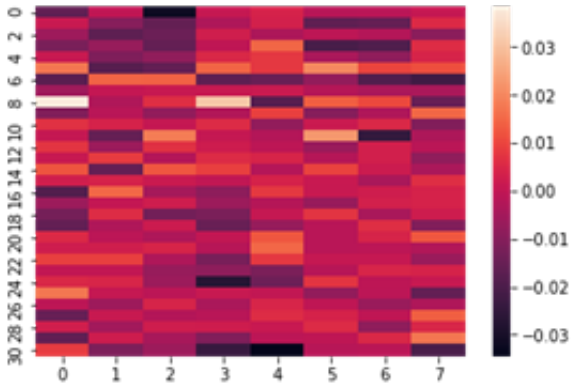

Figure 11: Example of learned weights associated to the policy found by Q-learning. This is a $n \times m$ matrix, where n is the number of possible actions, and m is the cardinality of the feature vector describing a specific state, as defined in (4.2). The mapping between a number on the y-axis and the corresponding gate can be found in the code on the public Github repository of the project.

4.2.1 Quantum state vector

As previously mentioned, the designed circuit operates on a two qubits composite system. This has been coded exploiting the computational basis of the Hilbert space spanned by the set $|00\rangle, |01\rangle, |10\rangle, |11\rangle$: the only information stored in order to define any state are the amplitudes of the basis elements in the linear combination, as expressed in equation (4.1).

4.2.2 Quantum Gates

We want to implement quantum states such that they can have a quantum gate applied, resulting into an update of the amplitudes. Given a generic gate U (an *action* in our RL setting)

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} , \tag{4.9}$$

and the state$|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle$, applying U on the first qubit of $|\psi\rangle$ results in the following updated vector of the amplitudes:

$$\begin{bmatrix} u_{00}\, a_{00} + u_{01}\, a_{10} \\ u_{00}\, a_{01} + u_{01}\, a_{11} \\ u_{10}\, a_{00} + u_{11}\, a_{10} \\ u_{10}\, a_{01} + u_{11}\, a_{11} \end{bmatrix} . \tag{4.10}$$

Step size	Negative reward	Positive reward	Discount
0.0001	-1	100	0

Table 1: Algorithm hyper parameters.

The effect of U on the second qubit of the state vector instead provides amplitudes equals to:

$$\begin{bmatrix} u_{00}\, a_{00} + u_{01}\, a_{01} \\ u_{10}\, a_{00} + u_{11}\, a_{01} \\ u_{00}\, a_{10} + u_{01}\, a_{11} \\ u_{10}\, a_{10} + u_{11}\, a_{11} \end{bmatrix} . \tag{4.11}$$

Results in (4.10) and (4.11) are hard coded in the algorithm.

4.3 Hyper parameters tuning

Before presenting the outcomes of the simulations, we provide an overview of the tuned values for the hyper parameters of the algorithm. These are the *step size* defined in equation (4.6), the scalar signals associated to negative and positive rewards, and the discount. According to the results observed we define the agent as *myopic*, i.e. we set $\gamma = 0$. The resulting hyper parameters schema we choose to train the agent with is provided in Table 1.

4.4 Algorithm Results

In this section we present the results obtained running the algorithm with the described characteristics. Since we are dealing with an episodic task, we train for a fixed number of episodes which has been set to 100. In order to end an episode, the agent must either hit a terminal state, or use a number of gates equals to an upper bound of 300: in this case, the state is just reset to the initial one, and a new episode begins.
We define $tolerance = 0.15$, fixing the set of terminal states as those matching with 0.85 or larger fidelity the target.
The initial state is always fixed to be $|\psi\rangle = |00\rangle$, while a distinct agent is trained for each of the four Bell states (defined in Appendix 7.2), which are treated as different problems. The learned policies provides the results presented in Table 2. Recall that we define optimality in terms of minimum number of gates needed to reach any terminal state.
We report that results in terms of number of gates are quite dependent on the seed of the behavioural policy, which determines the exploratory steps taken.

Bell state	Fidelity	Number of gates
$\frac{1}{\sqrt{2}}(\|00\rangle + \|11\rangle)$	1	2
$\frac{1}{\sqrt{2}}(\|00\rangle - \|11\rangle)$	1	4
$\frac{1}{\sqrt{2}}(\|01\rangle + \|10\rangle)$	1	3
$\frac{1}{\sqrt{2}}(\|01\rangle - \|10\rangle)$	0.93	3

Table 2: Algorithm results on Bell states.

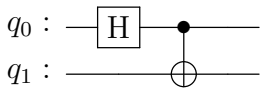

Figure 12: Agent-designed quantum circuit to prepare the Bell state $(|00\rangle + |11\rangle)/\sqrt{2}$. Note that, despite Hadamard not being in the set of actions of the agent, given the state $|00\rangle$ a rotation of $\pi/4$ around the Y axes has the same effect of the H gate.

5 Experimental results

The algorithm described in the previous section has been run only on a classical device. This is suitable for the training of the agent, since we can access the amplitudes of a state vector. Nevertheless, if we want to apply the results in practice, the designed circuit must be implemented on a real quantum hardware. This can be done on the IBM Quantum Lab [8], a cloud platform providing access to quantum devices.
Given that the circuit found to prepare $(|00\rangle + |11\rangle)/\sqrt{2}$ is the shortest (see Table 2), this is the target on which we focus this experimental part. The quantum circuit designed by the agent is represented in Figure 12.

5.1 IBM Quantum Lab

IBM Quantum Lab is a cloud based service from IBM, providing remote access to a real quantum device. All quantum systems engineered by IBM Quantum are based on superconducting qubit technology. The Quantum Lab allows to run quantum circuits on the devices through a Jupyter Notebook interface, using Qiskit, a Python based framework.

5.2 Quantum state tomography

Ideally, the output from our classical simulation would match the one coming from the real device: unfortunately, the latter is characterized by noise. For this reason, we compare the similarity between the expected and experimental terminal states distributions exploiting once again the *fidelity* measure. In order to do so, we must be able to reconstruct the experimental state, output of the quantum circuit implemented on the real device. This can be done using *quantum state tomography* [9].
Quantum state tomography is a technique based on maximum likelihood and numerical optimization to estimate a quantum state by repeated measurements on identically prepared states. In this case, instead of relying on the vector state representation, we use the *density matrix*: the two descriptions are perfectly equivalent, but are sometimes preferable one to the other, depending on which one is the simplest to implement.

First we consider the case of a single qubit, from which it will be easy to generalize to a larger system, which is what we are interested in. A single qubit can be associated to a density matrix written in the form:

$$\rho = \frac{I + \vec{r} \cdot \vec{\sigma}}{2} \, , \tag{5.1}$$

with $\vec{r}$ entries being the components of the Bloch vector of the qubit and $\vec{\sigma} = (\sigma_X, \sigma_Y, \sigma_Z)$.
Moreover, each component of $\vec{r}$, can be rewritten using the trace operator such that the following equality holds:

$$\langle \sigma_i \rangle = \mathrm{tr}(\rho \sigma_i) = r_i \, , \tag{5.2}$$

with $\langle \sigma_i \rangle$ the expectation of a component in the Pauli vector.

Being $\vec{r}$ the only unknown of our problem, it is sufficient to compute the expectation of $\sigma_X, \sigma_Y, \sigma_Z$ operators in order to have the correct value of the density matrix. Given the qubit initialized to $|0\rangle$, we repeatedly measures the state in all of the basis of the Pauli matrices. In this way, we obtain an estimate of the expectations required, which will be closer to the real value of $\langle \sigma_i \rangle$ the larger the number of measurements repetitions. The circuits implementing these measurements are illustrated in Figure 13.

At the end of this procedure, we ideally found the density matrix ρ of our system. Unfortunately, this is not the case in practice: due to the noisy and probabilistic nature of the measurements, we are not able to ensure that the

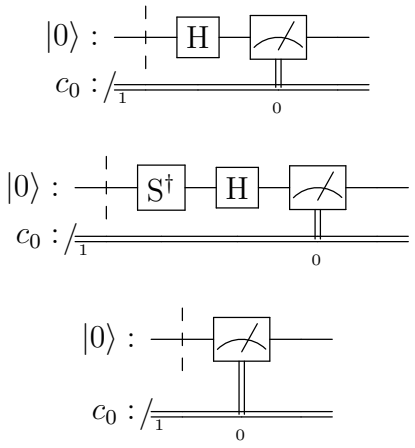

Figure 13: Measures on the Pauli matrix basis.

resulting estimate respects the *trace* and *positive semi-definite* conditions. Still the question is open: how can we approximate ρ from our sample measurements? Following the steps suggested in [10], the problem can be solved using the maximum likelihood technique, where we define as our estimate the *real* density matrix which maximizes the likelihood of the observed data. Given a distribution with unknown set of parameters θ, and a set of i.i.d. observations $\vec{x}$ from a certain distribution, the likelihood function is defined as

$$L(\vec{x}|\theta) = \prod_{i}^{n} f_\theta(x_i) \,, \tag{5.3}$$

with $f_\theta(x_i)$ distribution of the data. The estimate of θ maximizing the likelihood of the observed data is

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\vec{x}|\theta) \,, \tag{5.4}$$

which is perfectly equivalent to find $\hat{\theta}$ maximizing the logarithm of the likelihood. In order to apply this technique to the case under study, we define $\theta \coloneqq \rho$. The observed data are the counts $\{N_i\}$ of the observed measurements outcomes. For each basis set chosen, we have two values $\{N_j\}$, one for each orthogonal state in the basis. Supposing a Gaussian distribution of the noise, we can assume our observations to be sampled from a normal distribution. The resulting log-likelihood is

$$\log L(N_1, \cdots, N_6|\rho) = \sum_i^6 \frac{(N_i - \langle N_i \rangle)^2}{2 \langle N_i \rangle} , \tag{5.5}$$

where $\langle N_i \rangle = N_{tot} \langle i| \rho |i \rangle$ and N_{tot} equals the total number of measurements taken. In order to perform optimization over the density matrix, we define the ρ entries by means of some tunable parameters, as suggested in [10]. In particular, it can be proven that

$$\rho = \frac{T'T}{\text{tr}(T'T)} \tag{5.6}$$

and

$$T = \begin{bmatrix} t_1 & 0 \\ t_3 + it_4 & t_2 \end{bmatrix} , \quad t_i \in \mathbb{R} , \tag{5.7}$$

where T' denotes the T matrix transposed.
The above problem can be solved numerically, returning a density matrix maximizing the likelihood of the observed data and respecting the desired conditions on both trace and eigenvalues.

The concepts just introduced can be applied to reconstruct the density matrix of a system with an arbitrary large number of qubits. In our setting we are interested in applying quantum state tomography to a two particles composite system. Similar to the single qubit case, ρ can be expanded as

$$\rho = \sum_{i,j} r_{ij} \sigma_i \otimes \sigma_j , \tag{5.8}$$

and can be constructed via 16 projective measurements defined by all the combinations of the matrices in $\{I, \sigma_X, \sigma_Y, \sigma_Z\}$ via tensor product. The resulting circuits are shown in Appendix 7.3.
The likelihood function over which to maximize becomes

$$\log L(N_1, \cdots, N_{16}|\rho) = \sum_i^{16} \frac{(N_i - \langle N_i \rangle)^2}{2 \langle N_i \rangle} , \tag{5.9}$$

where we have a measurement count for each orthogonal state in each basis.

Finally, the matrix T of the tunable parameters can be shown to be

$$T = \begin{bmatrix} t_1 & 0 & 0 & 0 \\ t_5 + it_6 & t_2 & 0 & 0 \\ t_{11} + it_{12} & t_7 + it_8 & t_3 & 0 \\ t_{15} + it_{16} & t_{13} + it_{14} & t_9 + it_{10} & t_4 \end{bmatrix}, \quad t_i \in \mathbb{R}\,. \tag{5.10}$$

and the estimated density operator $\hat{\rho}$ found according to equation (5.6) is

$$\hat{\rho} = \begin{bmatrix} 0.506 & 0.002 - i\,0.017 & -0.001 - i\,0.017 & 0.461 + i\,0.012 \\ 0.002 + i\,0.017 & 0.017 & 0.001 + i\,0.004 & -0.003 + i\,0.013 \\ -0.001 + i\,0.017 & 0.001 - i\,0.004 & 0.02 & 0.001 - i\,0.001 \\ 0.461 - i\,0.012 & -0.003 - i\,0.013 & 0.001 + i\,0.001 & 0.457 \end{bmatrix}$$

5.3 State Fidelity

By measuring the fidelity between the expected and the estimated density matrices, associated to the output of the circuit in Figure 12, we obtain

$$F(\rho, \hat{\rho}) = 0.943\,, \tag{5.11}$$

with ρ being the expected result and $\hat{\rho}$ the corresponding estimate. With a number of measures infinite in the limit, we would expect the fidelity to approach 1, with the residual error due to the noise.

6 Conclusion

The presented approach provides promising results, looking as a good starting point to automatize the design of a circuit able to prepare a system into an arbitrary quantum state vector. Despite these satisfactory outcomes some limitations still remain unsolved: as one can observe in the results of Table 2, optimality in terms of minimum number of gates is not guaranteed. In particular, being the search space of minima in the approximated function $Q(S, A, \vec{w})$ large, it is easy to optimize the weights such that a local minimum is reached. This aspect is hardly under control being dependent on the randomness of the behavioural exploratory policy, which can be fixed with a seed for reproducibiliy but does not allow for a general solution applicable to any quantum states pair $(initial, target)$.

Another problem of interest is to apply *inverse reinforcement learning* to reconstruct the reward function given a set of samples representing the sequence of actions, output of an optimal given policy, in the context of designing a circuit transforming one state to another. We aim at developing this second part of the work in the future.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[2] DeepMind x UCL. *Introduction to Reinforcement Learning with David Silver*. 2015. URL: https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver.

[3] Richard Bellman. *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.

[4] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[5] Richard Feynman, Robert Leighton, and Matthew Sands. *The Feynman Lectures on Physics*. 1964.

[6] Jan Leike et al. "AI Safety Gridworlds". In: *CoRR* abs/1711.09883 (2017). arXiv: 1711.09883. URL: http://arxiv.org/abs/1711.09883.

[7] Marin Bukov et al. "Reinforcement Learning in Different Phases of Quantum Control". In: *Physical Review X* 8.3 (Sept. 2018). ISSN: 2160-3308. DOI: 10.1103/physrevx.8.031086. URL: http://dx.doi.org/10.1103/PhysRevX.8.031086.

[8] *IBM Quantum*. 2021. URL: https://quantum-computing.ibm.com/.

[9] Luca Rossi. *Studio di Sistemi Quantistici Correlati con Metodi di Teoria dell'Informazione Quantistica*. 2020.

[10] Daniel F. V. James et al. "Measurement of qubits". In: *Physical Review A* 64.5 (Oct. 2001). ISSN: 1094-1622. DOI: 10.1103/physreva.64.052312. URL: http://dx.doi.org/10.1103/PhysRevA.64.052312.

7 Appendix

7.1 Gradient Descent

Gradient descent is a minimization algorithm applicable to differentiable functions. In its most generic description we have an objective function $J(\vec{w})$ depending on a set of tunable weights. Our goal is to find $\vec{w}$ such that J is minimized. To do so, we need to compute the gradient of the objective function, defined as:

$$\nabla_{\vec{w}}(J) = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{bmatrix} \tag{7.1}$$

In the gradient descent algorithm expression (7.1) is evaluated over the whole dataset of pairs (s_i, v_i^π). A more efficient approach is given by *stochastic gradient descent*: rather than sampling a dataset $\mathcal{D}$ and computing the full gradient over it, we repeatedly evaluate the gradient on a single sample, and use its value for the update rule

$$\vec{w} = \vec{w} - \frac{1}{2}\alpha\nabla_{\vec{w}}(J) \,, \tag{7.2}$$

with α known as the *step size* parameter.

7.2 Bell states

A very well known set of states in quantum mechanics is given by the Bell states, which we employed as targets for our experiments. We briefly introduce them providing their state vectors:

$$|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \,, \tag{7.3}$$

$$|\beta_{01}\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \,, \tag{7.4}$$

$$|\beta_{10}\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \,, \tag{7.5}$$

$$|\beta_{11}\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}} \,. \tag{7.6}$$

7.3 Two Qubit quantum state tomography

The space in this section is used to show the quantum circuits designed to perform the measurements necessary for quantum state tomography.

Figure 14: Measurement on XI basis

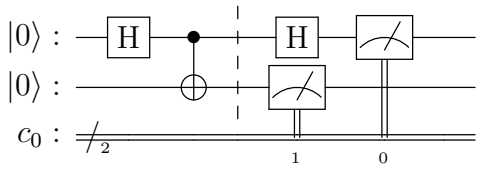

Figure 15: Measurement on XX basis

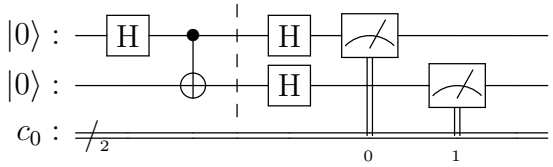

Figure 16: Measurement on XY basis

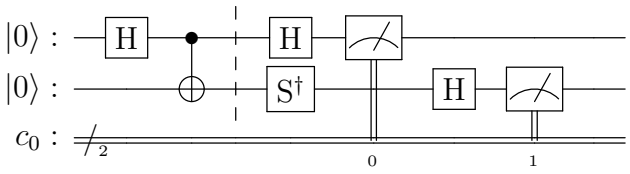

Figure 17: Measurement on XZ basis

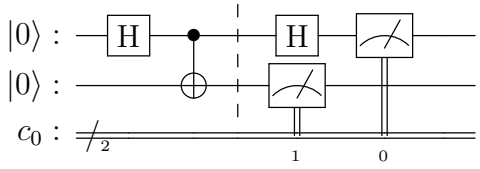

Figure 18: Measurement on YI basis

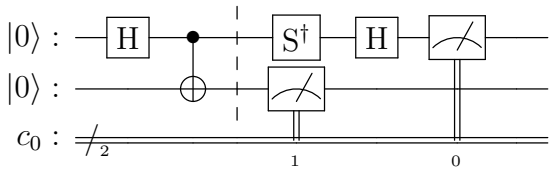

Figure 19: Measurement on YX basis

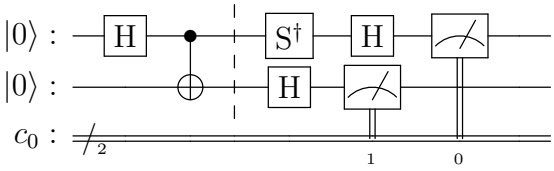

Figure 20: Measurement on YY basis

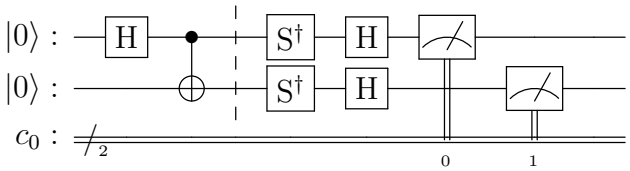

Figure 21: Measurement on YZ basis

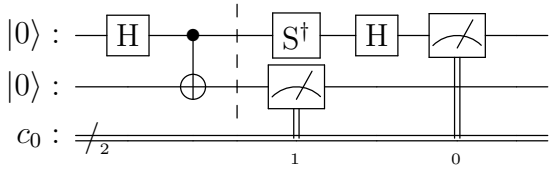

Figure 22: Measurement on ZI basis

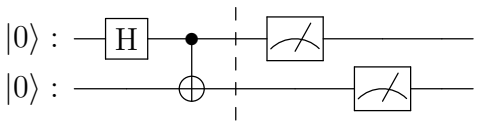

Figure 23: Measurement on ZX basis

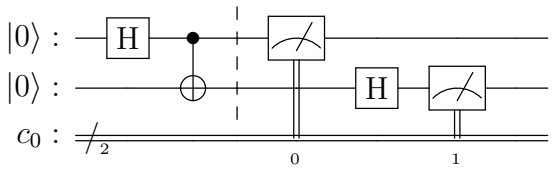

Figure 24: Measurement on ZY basis

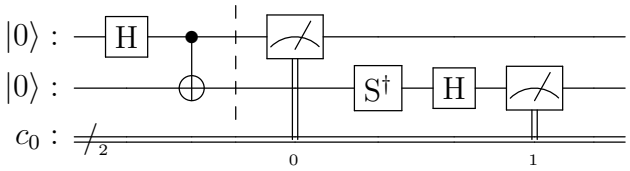

Figure 25: Measurement on ZZ basis

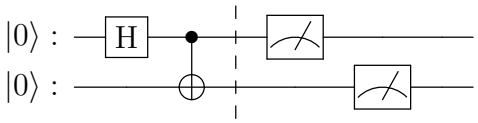

Figure 26: Measurement on IX basis

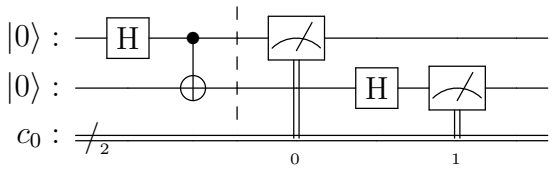

Figure 27: Measurement on IY basis

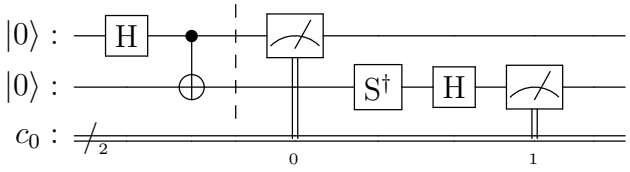

Figure 28: Measurement on IZ basis