Leopold-Franzens Universität Innsbruck

Institute for Theoretical Physics

Télécom ParisTech

École d'Ingénieur

Research Internship Report

for attainment of the academic degree of

Master of Engineering

"On the Quantum Speed-up of Markov Chain Monte Carlo Methods for Reinforcement Learning"

by

Sofiène Jerbi

Internship duration:	from March 5, 2018 to September 5, 2018
Submission date:	August 30, 2018
Supervisors:	Prof. Dr. Hans J. Briegel, Assis. Prof. Dr. Vedran Dunjko
Correspondent:	Prof. Dr. Romain Alléaume

Contents

List of Figures

Abstract

Recent advances in Reinforcement Learning consist in building agents that can learn to solve complex tasks, such as mastering the game of Go, in relatively few trial-and-error attempts. This number of attempts is commonly compared to the dimensionality of the space of states they can be in or the space of actions they can perform in those states. The ability to efficiently solve tasks for which state/action-spaces grow exponentially with the number of parameters describing them is called "tackling with the curse of dimensionality" or "generalization" in the literature. Projective Simulation (PS), a physics-inspired framework for the design of intelligent agents, provides through its Episodic Compositional Memory (ECM) some mechanisms to enable generalization. On the other hand, Reflecting PS (RPS), an extension of the basic PS model, achieves a quadratic quantum speed-up of the underlying deliberation process of agents with a restricted non-generalizing form of ECMs. But there are currently no known applicable methods that achieve a similar quantum speed-up for PS agents with generalization. In this work, inspired by a solution to the curse of dimensionality first introduced by Sallans and Hinton (2004) which relies on Boltzmann machines, an energy-based recurrent neural network, and Monte Carlo Markov Chain (MCMC) methods, we first establish a direct connection between RPS and MCMC methods for Reinforcement Learning. We then explore the design of quantum algorithms to gain a quantum speed-up of the deliberation process of our newly defined RPS agents.

1 Introduction

Research in *Artificial Intelligence (AI)* has come a long way from the initial problem of designing a genuine machine that could achieve (super)human level of intelligence in any given intellectual task, or what is called *Artificial General Intelligence (AGI)*. Focus has since shifted to solving more refined problems such as data classification, clustering or regression, commonly referred to as *applied AI* tasks. Nevertheless, this shift of emphasis only reveals the great complexity of the original problem and can be compared to the trend in research that led to aviation as we know it today: in order to produce a flying machine, we had to give up trying to replicate how birds fly and start studying aerodynamics. Similarly, a milestone on the path of building a genuine AGI is to first understand the building blocks of intelligence on specific reasoning tasks. And for this purpose, emerged new subfields of Computer Science gathered under the name of *Machine Learning (ML)* (Bishop, 2016), aiming to study and construct algorithms that would empower machines with the ability to learn how to solve these applied AI tasks.

One of the branches of ML is called *Reinforcement Learning (RL)* (Sutton et al., 1998) and explores the design of learning agents situated in an environment with which they interact through perceptions and actions in order to learn how to perform a task. This learning takes place by means of rewards sent by the environment to the agent, indicating to this latter the fitness of its last action given its state, in its attempt to perform the task correctly. RL has shown great results since its early beginnings, starting off with applications in systems control (Crites and Barto, 1998) (multiple RL agents learning to schedule elevator dispatching) or TD-gammon (Tesauro, 1995) (an algorithm based on *Temporal-Difference learning* (Sutton, 1988), one of the most influential RL methods, achieving expert-level in the game of Backgammon), and since then finding applications in a wide range of disciplines such as recommender systems (Shani et al., 2005), automated financial trading (Bertoluzzo and Corazza, 2014) and language models (Li et al., 2016). But certainly its most notable contribution up-to-date is with AlphaGo

1 Introduction

Zero (Silver et al., 2017), an algorithm achieving superhuman performance (meaning better than the recognized world champion) in the ancient Chinese game of Go, a board game known to be computationally harder than chess, that learns from scratch solely by playing against other instances of itself.

In order to achieve such levels of sophistication, many problems had to be resolved, some of which are still of great interest today. One of the earliest known problems is probably the exploration/exploitation dilemma (Katehakis and Veinott Jr, 1987): RL is a trial-and-error based learning scheme, in which the agent can choose either to exploit what it has already learned to accumulate rewards from previously discovered rewarding paths, or else explore new potentially more rewarding paths by taking unusual actions. Although, by exploring, the agent risks to diverge from this highly-rewarding state subspace it was in, while excessive exploitation can cause to miss out on the optimal behavior.

More intricate problems come from considering highly complex tasks with large state and action spaces (this is particularly the case of the game of Go). Indeed most "classic" methods of RL are so-called *tabular* methods, meaning that they will devise a strategy for the agent from a table of values stored in memory. In this table, one real value is associated to every state-action pair and indicates "how good" it is for the agent to perform the associated action in that corresponding state. This becomes very impractical first in terms of numbers of interactions with the environment needed to fill the table, growing at least linearly with the number of accessible states of the agent. Take the case of tasks such as the game of Go for instance, where the number of states is of the order of 10^{172}, to be compared to the estimated number of atoms in the observable universe (10^{80}), but more generally this is an issue for tasks where the number of states is exponential in the number of variables parameterizing these states. A second problem that arises is the lack of *generalization* in such tabular methods. How we humans or other biological intelligent entities deal with such complex tasks is by naturally perceiving the similarities and correlations between two states of a large state space and thus dealing with them in similar ways, whereas a tabular method would consider all states separately. This can be particularly impeding when an encountered state (a board configuration in the game of Go for instance) has a very low probability of occurring again in the life of the agent.

The solution found to deal with this first set of problems is called *Function Approximation* and its most eminent instance in RL relies on the use of *Neural Networks* (NNs) to represent a non-

linear *parameterized* function, i.e., specified by a set of parameters, best fitting the values given by the table behind tabular methods. In contrast with these tables, that are only sparsely filled, the parameterized functions are defined on the entire domain, efficiently "filling" the entire table. In this sense, these solutions allow generalizing learned knowledge to unencountered situations by resorting to an additional intensely studied machinery, a NN. In short, this latter can be seen as a parameterized Hamiltonian with pre-specified degrees of freedoms (the architecture of the network), whose parameters are learned to best fit a target function over a dataset, or *training data*, while maintaining a small fitting error on a separate but similarly generated dataset, called *test data*. In this aspect, these methods are very much related to two other branches of ML referred to in the literature as *Supervised* and *Unsupervised Learning* and consisting in tasks of learning to infer functions from respectively *labeled* and *unlabeled* data.

The integration of NNs in RL has a long history, starting from the early works of Sutton et al. (1998) and Bertsekas and Tsitsiklis (1995) on function approximation methods first relying on linear solutions, i.e., the approximate function is a linear function of a weight vector $\boldsymbol{w}$ and basically consists in the inner product between $\boldsymbol{w}$ and the input vector, in our case a state-action pair, which approximate the table behind tabular methods. These were optionally combined with feature construction methods, applying for instance polynomials or Fourier transforms on the input as a preprocessing step before going through the linear function, again methods inspired from Supervised Learning. From there, exciting achievements have been made with methods relying on non-linear NNs (Lin, 1993), and most notably Deep NNs, i.e., concatenations of many *shallow* NNs, where we can point out for instance to Mnih et al. (2015) (Deep RL agents playing Atari games at a superhuman level and learning from raw game screen input) and to Silver et al. (2017). The reader is referred to Li (2017) for a nice overview of Deep RL.

Another issue that comes with the use of tabular methods to deal with large state and action spaces is that deriving the strategy (or policy) of the agent, i.e., selecting an action given a state, out of the table is computationally intensive. Indeed, this requires to sum over an entire column of values in that table, meaning an exponentially large number of values, thus rendering the method impractical for real-time applications or at least very slow for complex tasks. We then resort to selecting actions given a state through so-called probabilistic (and usually also graphical) models that re-express the decision process in terms of conditional probabilities. Action deliberation becomes then an *inference* process, i.e., deducing properties of a probability

distribution from observations, on the probabilistic model. Formally, we want to evaluate the posterior distribution $P(\boldsymbol{A}, \boldsymbol{H}|\boldsymbol{S})$ of the latent variables (action variables $\boldsymbol{A}$ and some extra latent variables $\boldsymbol{H}$ allowing higher degrees of correlation) given the observed variables ($\boldsymbol{S}$, the state). Because *exact* inference is usually harder the larger and more complex the model is (for discrete variables, the marginalization may involve summing over exponentially many possible configurations of the latent variables similarly as described above, or for continuous variables, the required integration may not have a closed-form analytical solution), we again resort to *approximate* methods. These come in two flavors: they are either *deterministic* and are based on analytical approximations to the posterior distribution (with assumptions on its factorization or form, e.g., Gaussian), or *stochastic* and based on numerical sampling, then more commonly known as *Monte Carlo* methods. A particular class of these relies on *Markov Chains*, an iterative computational process which approximates the correct solution but attains the exact solution only in the limit of infinite computational time.
One model that expresses the decision process in terms of conditional probabilities is *Projective Simulation (PS)*, introduced by Briegel and De las Cuevas (2012), which frames the memory of an RL agent in the form of a weighted graph. Conveniently, Markov Chain-based methods can also be used to perform inference on this model.

Having these two groups of issues in mind, namely the lack of generalization and the inefficient action deliberation of classic RL methods in the context of large state and action spaces, we may consider a well-studied graphical model that can provide an elegant solution to both all-together: *Boltzmann Machines (BMs)*. Indeed, this type of NNs, more precisely a Recurrent NN of units with a stochastic activation function, is a Function Approximation model that moreover relies on statistical inference methods to sample from a probability distribution parameterized by its approximated function. This makes it particularly appealing for RL applications if we want to encode an agent's policy in the BM, and especially suited to respond to our given issues because of its inherent generalization abilities. To make inference easier, one usually considers a particular class of BMs called *Restricted Boltzmann Machines (RBMs)* with a more restrictive structure giving it the form of a bipartite graph. In spite of this restriction, RBMs are known to still be universal approximators, i.e., they can approximate any distribution to any precision given sufficiently many units (Le Roux and Bengio, 2008). However, to represent arbitrary fine distributions, one may have to pay the price of slow deliberation for RBM-based RL agents.

The solution found classically is to counterbalance this computational overhead with a biased approximation of the policy during deliberation. Because large approximation errors in the policy could lead to agents that can't learn the optimal policy, we consider another solution that would rely on *quantum computation* to speed-up their deliberation process while avoiding this trade-off.

Taking advantage of quantum mechanical effects, namely quantum coherence and entanglement, to design new algorithms that provide a computational advantage over the best classical methods has now a long history since the early (Grover, 1996) and (Shor, 1999) algorithms. Enough quantum algorithms have been designed for them to be gathered under subfields, such as *Quantum Machine Learning*. This latter designates computational methods or subroutines that provide quantum speed-ups for solving ML problems. These may be built up on (but are not limited to) Quantum Amplitude Amplification (Brassard et al., 2002), a generalization of Grover's algorithm, or HHL (Harrow et al., 2009), a quantum algorithm for solving linear systems of equations. We refer the readers to the following review papers for a more extensive description of the applications of QML (Biamonte et al., 2017; Schuld et al., 2015; Adcock et al., 2015; Ciliberto et al., 2018; Dunjko et al., 2016).

2 Preliminaries

In this chapter, we will provide the basics of the notions dealt with in this paper, namely Markov Chains, Reinforcement Learning, Projective Simulation, Boltzmann Machines and Quantum Amplitude Amplification using Szegedy walk operators. Readers are invited to refer to unfamiliar notions and to skip the ones they feel comfortable with.

2.1 Markov Chains

We present here the basic notions of Markov Chains. For a more in-depth review, we refer the readers to the following survey (Montenegro et al., 2006) and book (Brémaud, 2013).

2.1.1 Formal setting

A *Markov Chain (MC)* is a probabilistic process defined by a set of states and associated events, where events cause transitions between states and satisfy the *Markov property*, namely that the probability of a certain event solely depends on the state reached after the latest elapsed event and not on previous states and events.

We consider *discrete-time* MCs, meaning that the model can be described by a sequence (or chain) of random variables $\boldsymbol{C}_0, \boldsymbol{C}_1, \boldsymbol{C}_2, \ldots$ indexed by *discrete* time-steps of the process and taking values in a state space C. In this case, the events are transitions between a random variable $\boldsymbol{C}_n$ in a certain state c_n and the random variable $\boldsymbol{C}_{n+1}$ in a certain state c_{n+1}, with $(c_n, c_{n+1}) \in C^2$. These transitions are then described by *conditional transition probabilities*, that satisfy (taking into account the Markov property):

$$\mathbb{P}(\boldsymbol{C}_{n+1} = c \mid \boldsymbol{C}_0 = c_0, \boldsymbol{C}_1 = c_1, \ldots, \boldsymbol{C}_n = c_n) = \mathbb{P}(\boldsymbol{C}_{n+1} = c \mid \boldsymbol{C}_n = c_n) \quad \forall n,\ \forall c \in C \tag{2.1}$$

Moreover we consider *time-homogeneous* (or *stationary*) MCs, meaning that the conditional transition probabilities are independent of the time-step of the MC process:

$$\mathbb{P}(\boldsymbol{C}_{n+1} = c' \mid \boldsymbol{C}_n = c) = \mathbb{P}(\boldsymbol{C}_n = c' \mid \boldsymbol{C}_{n-1} = c) \quad \forall n \tag{2.2}$$

These first two properties, along with the property of a *finite state space* C, are enough to specify the whole MC process with a concise description called a *transition matrix* P, whose elements are:

$$p_{ij} = \mathbb{P}(\boldsymbol{C}_{n+1} = c_j \mid \boldsymbol{C}_n = c_i) \quad \forall (i,j) \in \{1, \ldots, |C|\}^2 \quad \forall n \tag{2.3}$$

In this work, unless stated otherwise, we will consider our MCs to be discrete-time, time-homogeneous and having a finite state space.

2.1.2 Random walks

Another way of describing a MC process is as a *random walk* over a connected directed graph, where the nodes correspond to the states of C and the random variables $\boldsymbol{C}_n$ to the position (i.e., node) on the graph at time-step n of this random walk. This graph is weighted by transition probabilities p_{ij} between nodes (i.e., states) c_i and c_j, and hence can also be concisely described by the transition matrix P. In graph-theoretic terms, this P is a generalization of the *adjacency matrix* (a matrix listing all connections in a finite graph) to weighted graphs.

Having this visualization in mind, we can consider alternatively to the random variables $\boldsymbol{C}_n$ the probability distribution vectors $\boldsymbol{v}_n \in \mathbb{R}_+^{|C|}$ ($\|\boldsymbol{v}_n\|_1 = 1$, where $\|.\|_1$ is the L_1-norm) defining probability distributions on the ensemble of nodes of the graph, i.e., the state space C, at each time-step $n \in \mathbb{N}$. The weights of such a probability distribution vector are set as such: we start the random walk with an initial probability distribution $\boldsymbol{v}_0$ (e.g., $(0, \ldots, 0, 1, 0, \ldots, 0)^\top$ if we start at a particular node of the graph) and apply one step of the random walk with a left matrix-multiplication by the transition matrix P [1]. Hence n steps of the random walk give us: $\boldsymbol{v}_n = P^n \boldsymbol{v}_0$. Note that the random variables $\boldsymbol{v}_n$ give a more global description of the random walk process since, if $\boldsymbol{v}_0$ encodes $\mathbb{P}(\boldsymbol{C}_0)$, then $\boldsymbol{v}_n$ is effectively $\mathbb{P}(\boldsymbol{C}_n)$, i.e., a convex combination of the resulting position at time-step n of many individual random walks described by $\boldsymbol{C}_n$.

[1] We adopt the left-stochastic convention, by contrast to the right-stochastic convention, meaning that we consider column-vectors $\boldsymbol{v}_n$ and left matrix-multiplications by the transition matrix.

2.1.3 Stationary distribution

We call *stationary distribution* (or *steady state*) of P a probability distribution vector $\boldsymbol{v_\infty}$ that is a fixed-point of the transition matrix P, meaning that $P\boldsymbol{v_\infty} = \boldsymbol{v_\infty}$. It is important to understand that for general finite space MCs, this stationary distribution always exists but may not be unique, and our probability distribution vectors $\boldsymbol{v}_n$ may not converge to it even after an infinite number of applications of P.

In order to derive such properties on the stationary distribution of our MCs, we need additional conditions on the transition matrix P.
An MC is said to be *irreducible* if there exists a directed path in the transition graph from every state c_i to every other state c_j, i.e., if the underlying graph is strongly connected:

$$\forall (i,j) \in \{1,\dots,|C|\}^2,\ \exists n_{ij} \in \mathbb{N} \ /\ \mathbb{P}(\boldsymbol{C}_{n_{ij}} = c_j \mid \boldsymbol{C}_0 = c_i) > 0 \tag{2.4}$$

One can then prove the following:

Theorem 2.1.1 *If a Markov Chain is irreducible, then a stationary distribution $\boldsymbol{\pi}$ exists and is unique.*

We define the *period* k_i of a state c_i as the greatest common divisor of the number of time-steps it takes to get back to state c_i when starting a random walk from this same state, i.e., it always takes a multiple of k time-steps to get back at c_i starting from it:

$$k_i = \gcd\{n > 0 : \mathbb{P}(\boldsymbol{C}_n = c_i \mid \boldsymbol{C}_0 = c_i) > 0\} \tag{2.5}$$

A state c_i is said to be *aperiodic* if $k_i = 1$.

Theorem 2.1.2 *If a Markov Chain is irreducible and aperiodic, i.e., all its states are aperiodic, then its stationary distribution $\boldsymbol{\pi}$ is reached at least asymptotically, independently from the initial distribution $\boldsymbol{v}_0$:*

$$\forall \boldsymbol{v}_0 \quad \lim_{t\to\infty} P^t \boldsymbol{v}_0 = \boldsymbol{\pi}$$

But in practice, asymptotic convergence is not enough, and we would rather characterize the rate of convergence to stationarity by trying to bound it for our MCs of interest. Typically, we characterize the rate of convergence by the so-called *mixing time*, defined as the first time

$t_{\epsilon'}^{mix}$ by which the L^p (usually $p = 2$) distance between the distribution at time $t_{\epsilon'}^{mix}$ and the stationary distribution falls below a certain threshold ϵ':

$$t_{\epsilon'}^{mix} = \min\{t : \forall \boldsymbol{v}_0,\ \parallel P^t \boldsymbol{v}_0 - \boldsymbol{\pi} \parallel_p \leq \epsilon'\} \tag{2.6}$$

2.1.4 Spectral bounding of the mixing time

One common method to bound the mixing time an MC is to use the spectral proprieties of its associated transition matrix. For that we denote:

$$1 = \lambda_0 \geq \lambda_1 \geq \ldots \geq \lambda_n$$

the eigenvalues of our irreducible and aperiodic MC of interest, in decreasing order. It is interesting to notice that the greatest eigenvalue is 1 (its eigenvector is the stationary distribution) and that the remaining eigenvalues have magnitudes at most 1. One last important concept in the study of MCs is the notion of *reversibility*. We define the *time-reversal* $P^* = (p_{ij}^*)_{N\times N}$ of $P = (p_{ij})_{N\times N}$ having as stationary distribution $\boldsymbol{\pi} = (\pi_i)_{N\times 1}$ by the identity:

$$\pi_i p_{ij}^* = \pi_j p_{ji} \quad \forall (i,j) \in \{1,\ldots,|C|\}^2 \tag{2.7}$$

called the *detailed balance*. We say that P is *time-reversible* (or simply *reversible*) if $P^* = P$.

We now have all the tools to give a simple bound for the mixing time of our MCs:

Theorem 2.1.3 *(Aldous, 1982) The mixing time of an irreducible, aperiodic and reversible MC satisfies:*

$$\left\lfloor \frac{1-\delta}{2\log 2\epsilon'} \frac{1}{\delta} \right\rfloor \ \leq\ t_{\epsilon'}^{mix} \leq\ \left\lceil \frac{1}{\delta} \log \frac{1}{\epsilon' \pi^*} \right\rceil$$

where $\delta = 1 - \max_{i>0} |\lambda_i|$, i.e., the difference between the absolute values of the largest (unity) and second largest eigenvalues modulus of P, is called the spectral gap *of P and $\pi^* = \min_i\{\pi_i\}$.*

We often say that *the MC mixes in time* $\tilde{\mathcal{O}}(1/\delta)$. [2]

2.2 Reinforcement Learning

In this section, the term *state* refers to something different than in the context of MCs. Namely, even though this is an abuse of terminology, we consider the state of an RL agent in an environment to be specified by its received *percepts* and arbitrarily interchange the two terms.

[2] Where $\tilde{\mathcal{O}}$ is the soft-$\mathcal{O}$ notation, meaning that we neglect poly-logarithmic dependencies.

2.2.1 Reinforcement Learning setting

The general *Reinforcement Learning (RL)* framework can be described as a cyclic agent/environment interaction depicted in Figure 2.1.

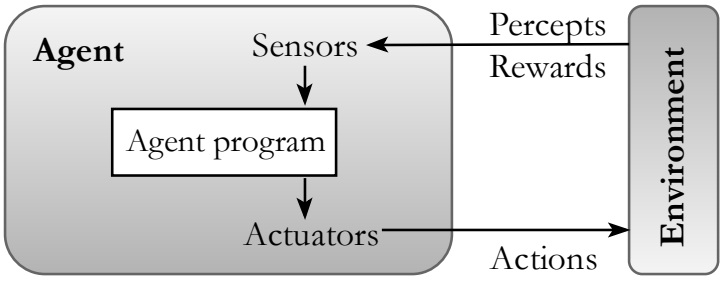

Figure 2.1: **The Reinforcement Learning framework.** An agent is situated in an environment. It has sensors, through which it receives percepts and rewards from the environment, and actuators, through which it acts on the environment. Based on its perceptual input, the agent engages an internal processing and outputs an action.

We start formalizing the model of RL agents through the following notations:

Definition 2.2.1 *The cyclic interaction of an RL agent with an environment is fully described by a (possible infinite) sequence* $\boldsymbol{s^{(0)}}, \boldsymbol{a^{(0)}}, r^{(1)}, \boldsymbol{s^{(1)}}, \boldsymbol{a^{(1)}}, \ldots, \boldsymbol{s^{(t)}}, \boldsymbol{a^{(t)}}, \ldots$ *called* history*, where:*

- t *labels the time-steps of the interaction*
- $\boldsymbol{s}^{(t)}$ *is the state of (or percept received by) the agent at time-step* t*, belonging to a state space* $\mathcal{S}$
- $\boldsymbol{a}^{(t)}$ *is the action of the agent at time-step* t*, belonging to an action space* $\mathcal{A}$
- $r^{(t+1)}$ *is the reward of the agent at time-step* $t+1$*, belonging to a space* $\mathcal{R}$ *and depending on* $\boldsymbol{s}^{(t)}$ *and* $\boldsymbol{a}^{(t)}$

Once an agent and its environment are specified, there exist many figures of merits one can use to qualify the fitness of this agent in that environment. For instance, we can consider its *return* G_t at time-step t, i.e., its discounted sum of rewards from that point on and for the rest of its interaction with the environment:

$$G_t = r^{(t+1)} + \gamma_{env} r^{(t+2)} + \gamma_{env}^2 r^{(t+3)} + \gamma_{env}^3 r^{(t+4)} + \ldots \tag{2.8}$$

where γ_{env} is called a *damping parameter* (or *discount rate*) of the environment which assumes a value in [0,1] and serves avoiding diverging returns for infinite interaction sequences.
The goal of RL is then to design agents that will maximize their figure of merit, for instance their discounted return, at each timestep, in the largest class of environments possible.

To design a model of learning agents, many classic methods of RL choose to consider specific functions that will govern the agents' internal deliberation programs. We will define two of these and describe their general update procedure.
The first is called the *policy*. It associates to each state in $\mathcal{S}$ a probability distribution over the action space $\mathcal{A}$ from which the agent samples when it is supposed to perform an action:

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \boldsymbol{\pi_s}(\boldsymbol{a}) = \mathbb{P}(\boldsymbol{a}^{(t)} = \boldsymbol{a} \mid \boldsymbol{s}^{(t)} = \boldsymbol{s}) \quad \forall(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A} \tag{2.9}$$

This policy is governed by our second function of interest, called the *state-action-value function* (or *Q-function*) and that is effectively the expected return of the agent for a given state and action performed in that state:

$$Q_\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{\pi}}[G_t \mid \boldsymbol{s}^{(t)} = \boldsymbol{s}, \boldsymbol{a}^{(t)} = \boldsymbol{a}] \quad \forall(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A} \tag{2.10}$$

The reason why the Q-function is indexed by a policy is that the return G_t that the Q-function estimates is dependent on the actions the agent will perform in subsequent time-steps, which are governed by the policy, as this latter should give high probability to the actions that maximize the Q-function given a state.

2.2.2 Generalized Policy Iteration

The last remark on the interdependence of the policy and the Q-function actually sets the intuition behind the general update procedure of these two functions, called *Generalized Policy Iteration (GPI)*, and in which many RL methods (so-called *value-based* methods) can be framed. As the Figure 2.2 depicts, one goal of an RL agent is to attain the optimal Q-function Q^* and policy π^* (with $Q^* = Q_{\pi^*}$) that will maximize its expected return at each time-step, through sequential updates. It will first start from initial (generally random) functions. Then the learning procedure can be seen as two competing processes:

- *Policy Improvement* where the agent derives a policy from its Q-function. It modifies the former in order that it will seek the most rewarding states according to the latter. The

agent will then use this new policy to deliberate on its actions when interacting with the environment. But through this interaction it may realize that the estimates of the return for its encountered states are biased, hence the need for the second process.

- *Policy Evaluation* where the agent updates its Q-function to correct its estimates of the return according to the rewards received by following the current policy.

These are said to be competing processes because they correct each other by bringing the other function closer to its optimal value, but simultaneously rendering the reference function biased relative to the target one because it was formerly derived from the previous value of the target. The learning scheme continues until we reach a fixed-point of the GPI process where Policy Improvement and Policy Evaluation don't change the Q-function and policy anymore, which are then respectively equal to their optimal functions Q^* and π^*.

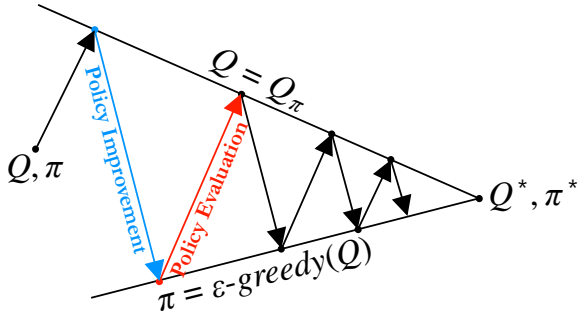

Figure 2.2: **Generalized Policy Iteration**

2.2.3 Policy Improvement

Deriving a policy out of a Q-function can be done in different ways, but essentially consists of defining a probability distribution parameterized by the Q-function, i.e., translating a real-valued function of the space $\mathcal{S} \times \mathcal{A}$ to a conditional distribution on the space $\mathcal{A}$ given a state from $\mathcal{S}$. When doing so, one tries to find a good balance between assigning high probability to highly rewarded actions (given a state) to move towards the main goal of maximizing the return, but also allowing a certain degree of exploration of less rewarded actions (given a state) which could lead to the discovery of new highly rewarding subsequent states. This is what we call in the literature the *exploitation-exploration* dilemma and it is usually resolved by using hyperparameters setting the balance between the two.

The first family of parameterized policies we are going to present is the so-called ϵ-*greedy policy*:

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \boldsymbol{\pi_s}(\boldsymbol{a}) = \begin{cases} 1-\epsilon & \text{if } \boldsymbol{a} = \text{argmax}_{\boldsymbol{a'}}(Q(\boldsymbol{s}, \boldsymbol{a'})) \\ \epsilon & \text{otherwise} \end{cases} \tag{2.11}$$

where ϵ is the exploration hyperparameter.

The second family of parameterized policies is the so-called *Boltzmann* (or *softmax*) *policy*:

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \boldsymbol{\pi_s}(\boldsymbol{a}) = \frac{e^{Q(\boldsymbol{s},\boldsymbol{a})/T}}{\sum_{a'} e^{Q(\boldsymbol{s},\boldsymbol{a'})/T}} \tag{2.12}$$

where the exploration hyperparameter is now the *temperature* T.

2.2.4 Policy Evaluation

Policy Evaluation, i.e., updating the Q-function according to the rewards collected by following a certain policy, is essentially where the difference between the few classic methods we have been mentioning until now lies. However, they all share the same objective: improve the estimates of the *return* G_t (Formula (2.8)) for given state-action pairs, namely the values of the Q-function. Now suppose that the agent follows a certain policy π from a first time-step t, i.e., it is in a certain state $\boldsymbol{s}$ and performs an action sampled from the corresponding probability distribution $\boldsymbol{\pi_s} = \pi(\boldsymbol{s})$, then the reward it will receive for that action (at time-step $t+1$) is the first term of the return G_t in Formula (2.8) when following π and can already be considered as an estimate (although maybe very biased) of G_t. A better estimate would be to also consider the reward at the next time-step $t+2$ when following the same policy π and adding it to the estimate, discounted by the damping parameter γ_{env}. The same procedure can be applied for as many time-steps as we wish. And there is where the difference between different classic methods essentially lies: how far we *bootstrap* the estimates of the return.

In this work though, we are only going to consider one update rule for the Q-function called *Q-learning* that is part of a group of learning methods called *Temporal Difference Learning*, and which can be formulated as:

$$Q(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) \leftarrow (1-\alpha)Q(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) + \alpha[r^{(t+1)} + \gamma_{env}\text{max}_{\boldsymbol{a}}Q(\boldsymbol{s}^{(t+1)}, \boldsymbol{a})] \tag{2.13}$$

where α is a learning parameter set in $[0, 1]$, and the term between square brackets can be seen as a 1-time-step bootstrap of the estimate of the return.

2.3 Projective Simulation

2.3.1 Basic Projective Simulation

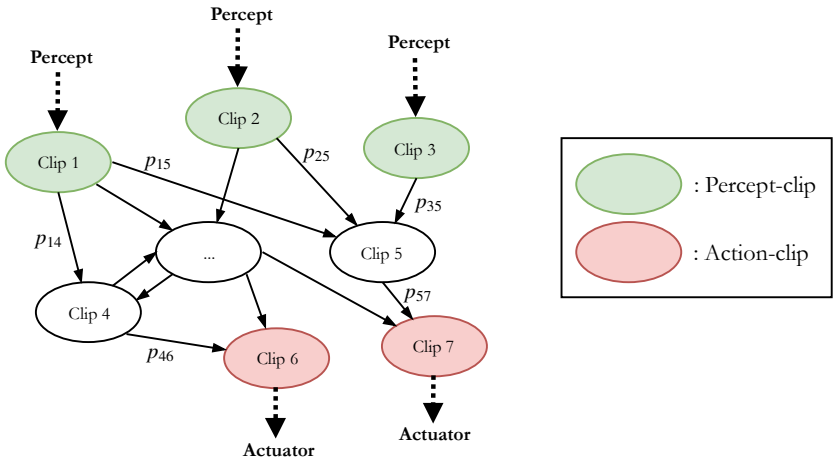

Figure 2.3: **A Generic Episodic Compositional Memory**

Projective Simulation (PS) is a physics-inspired model, introduced by Briegel and De las Cuevas (2012), for the design of autonomous learning agents and which can be used in a Reinforcement Learning scenario. The central component of a PS agent is its so-called *Episodic Compositional Memory (ECM)*, formally represented as a stochastic network of *clips* (see Figure 2.3). Clips, i.e., the nodes of the stochastic network, represent episodes, i.e., sequences of percepts, rewards and actions, built upon experience of the agent through certain compositional methods applied on more basic clips: *percept-clips*, associated to single percepts perceived by the agent and *action-clips*, associated to single actions it can perform. But for simplicity of visualization, one can consider the simpler variant where the ECM is composed only of percept-clips and action-clips for now. In PS, the deliberation mechanism is based on a random walk process, initiated at the percept-clip associated to the percept received at a certain time-step. There are many ways in which the random walk process can be executed, but in the basic PS, this consists in hopping between clips until an action-clip is hit. The action associated with that action-clip is then executed on the environment. Once it receives a reward for that action, if it is positive, the PS agent can update the weights (or transitions probabilities) of its ECM to increase the probabilities of the path that led to that action.

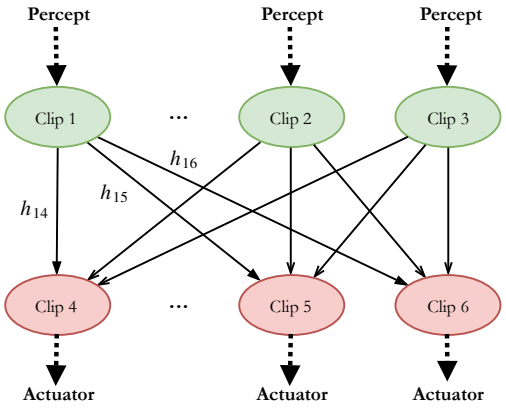

Figure 2.4: **A *two-layered* ECM**

The most basic ECM fitting the PS specification is given in Figure 2.4: it is only composed of one layer of percept-clips linked to another layer of action-clips through weighted connections that we refer to as *h-values* (note that h-values are not specific to this basic PS). These are real-values constrained to be greater than (and initialized at) 1, that, when normalized according to a linear rule $\frac{h_{ij}}{\sum_{j'} h_{ij'}}$ or a Boltzmann rule $\frac{e^{h_{ij}}}{\sum_{j'} e^{h_{ij'}}}$, give rise to the policy of the PS agent.
Learning takes place by updates of these h-values according to the following update rule[3]:

$$\begin{cases} h_{ij}^{(t+1)} = h_{ij}^{(t)} - \gamma(h_{ij}^{(t)} - 1) + \lambda & \text{for the rewarded transition} \\ h_{ij}^{(t+1)} = h_{ij}^{(t)} - \gamma(h_{ij}^{(t)} - 1) & \text{for the other transitions} \end{cases} \tag{2.14}$$

where $\gamma \in [0, 1]$ is an internal *damping* (or forgetfulness) parameter allowing the agent to adapt to changing environments, and λ is the *positive* reward received by the agent for performing its last action. This latter is selected by a one-step random walk on the ECM starting from the percept-clip triggered at the current time-step.
Despite the simplicity of this basic ECM, a bipartite graph with arbitrary real-valued weights is capable of representing any probability distribution, making the *two-layered* PS a model powerful enough to achieve near-optimal results in benchmark tasks (Melnikov, Makmal and Briegel, 2018), doing adaptive Measurment Based Quantum Computation (MBQC) in changing environments (Tiersch et al., 2015) or create new quantum optical experiments (Melnikov, Nautrup, Krenn, Dunjko, Tiersch, Zeilinger and Briegel, 2018).

[3]This is the simplest update rule for PS, and extensions exist

2.3.2 Reflecting Projective Simulation

From this basic model of a PS agent derive several enhancements through additional features, each responding to specific issues. One of these features is called *reflection* and enhances an agent with the ability to retract (or reflect) on the last action it hit during the random walk it initiated for its deliberation. This means that, at each time-step, the agent has the ability to run this random walk multiple times, starting from the same percept of that current time-step. This reflection, i.e., decision to run the random walk again, is governed by so-called *flags* (or *emotions*), generally binary ("good" or "bad"), assigned to each action-clip and indicating for instance if the associated action recently led to a positive reward. This can be seen as a *short-term memory* added to the agent end effectively induces an additional bias toward (or away from) certain percept-action transitions.

Going further with this notion of reflection, Paparo et al. (2014) defined a new agent model called the *Reflecting PS (RPS)*, which restates the random walk on its ECM as an irreducible and aperiodic Markov Chain process (hence with a unique stationary distribution). More formally, at each time-step of its interaction with an environment, an RPS agent has specified an irreducible and aperiodic transition matrix $P_{\boldsymbol{s}}$ that depends on the history of this interaction and which can be pictured as an ECM. The agent then runs an algorithm that will mix this MC until it converges to its stationary distribution (for instance this can be explicitly a random walk running until stationarity or just computations on the transition matrix trying to derive the stationary distribution analytically) and then sample from this stationary distribution. More precisely, the agent samples from the marginal distribution $\boldsymbol{\pi}_{\boldsymbol{s}}^{f}$ over flagged actions encoded in that stationary distribution $\boldsymbol{\pi}_{\boldsymbol{s}}$. Mathematically, the relation between these two distributions is given by $\boldsymbol{\pi}_{\boldsymbol{s}}^{f}(\boldsymbol{a}) \propto \delta(\boldsymbol{a} \in \text{flagged}) \times \boldsymbol{\pi}_{\boldsymbol{s}}(\boldsymbol{a})$, wher δ is the Kronecker-delta function over sets.

2.4 (Restricted) Boltzmann Machines

Boltzmann Machines (BMs) are stochastic *Recurrent Neural Networks*, meaning that their basic constituents are stochastic neurons (or units) which values are set with probabilities depending on their inputs, and that the output of a single unit can impact its own input. They serve as *generative models*: they encode a certain probability distribution over visible variables, associated to a fixed subset of the units of the BM called *visible units*, and can generate samples from that distribution. They also have the particularity of being an *energy-based* model, i.e., a particular

configuration of *visible* and *hidden units* (non-visible units) has an energy, parameterized by symmetric *weights* between units and *biases* of single units, and this energy sets the probability of the configuration. The general layout of a BM is given by Figure 2.5.

We refer the readers to the following introductions to BMs (Kenny, n.d.) and their restricted version (Fischer and Igel, 2012; Hinton, 2012).

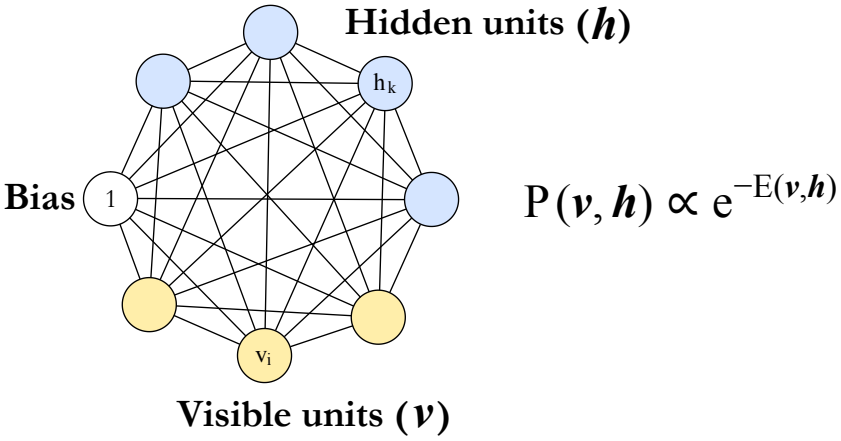

Figure 2.5: **A Boltzmann Machine.** All connections are weighted but weights are not indicated for clarity. The biases of single units can be represented by weighted connections, included in the weights w_{ij} of the BM, to a unit that is always active, i.e., of value always 1

2.4.1 An energy-based model

The energy of a configuration $(\boldsymbol{v}, \boldsymbol{h})$ of visible and hidden units is given by:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i,k} w_{ik} v_i h_k - \sum_{i,i'} w_{ii'} v_i v_{i'} - \sum_{k,k'} w_{kk'} h_k h_{k'} - \sum_i b_i v_i - \sum_k b_k h_k \tag{2.15}$$

where v_i is the binary value of a visible unit, h_k is the binary value of a hidden unit, w_{xy} is the weight between units x and y, and b_x is the bias of unit x.

The associated probability of this configuration is then:

$$P(\boldsymbol{v}, \boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{v}',\boldsymbol{h}'} e^{-E(\boldsymbol{v}',\boldsymbol{h}')}} = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{Z} \tag{2.16}$$

The role of hidden units in a BM is abstractly to encode binary *interpretations*, i.e., a representation in a higher dimensional space, of visible units, and more concretely to add supplementary degrees of correlation between them.

We have then:

$$P(\boldsymbol{v}) = \sum_{\boldsymbol{h}} P(\boldsymbol{v}, \boldsymbol{h}) = \frac{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{v}',\boldsymbol{h}'} e^{-E(\boldsymbol{v}',\boldsymbol{h}')}} \tag{2.17}$$

As a generative model, the use of BMs is a two-step process: i) learn the probability distribution of a training set of binary vectors $\boldsymbol{v}$ (of length the number of visible units $|V|$), i.e., find the weights of the BM (or *train* the BM) such that (2.17) best fits the underlying distribution of the training set, ii) use the BM to generate samples $\boldsymbol{v}$ from the encoded distribution (2.17).

2.4.2 Training a Boltzmann Machine

Suppose we are given a set of binary vectors $\boldsymbol{v}$, called the *training set* $\mathcal{D}$, and we want to encode its underlying probability distribution in a model with pre-specified degrees of freedom, namely of the form (2.16), with an energy function given by (2.15). Then, this objective can be translated into minimizing the so-called *loss function* or *negative log-likelihood* $-\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{v}\in\mathcal{D}}\log\left(P(\boldsymbol{v})\right)$ for this given training set and the parameterized distribution given by the BM. One approach to reach this minimum is by performing *stochastic steepest descent (SSD)* on that loss function.
It has been shown that the derivative of the log probability of a training vector with respect to a weight/bias of the BM has a simple expression:

$$\frac{\partial \log P(\boldsymbol{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\boldsymbol{v}} - \langle s_i s_j \rangle_{model} \tag{2.18}$$

where s refers to the value of either a visible unit, a hidden unit or the bias unit (depending on which weight or bias w_{ij} we consider). $\langle . \rangle_{\boldsymbol{v}}$ refers to the expectation (over all possible assignments of $\boldsymbol{h}$, weighted by their distribution $P(\boldsymbol{h}|\boldsymbol{v})$ derived from (2.16)) of the corresponding product of values when the visible units are set to be equal to the training vector $\boldsymbol{v}$. And $\langle . \rangle_{model}$ refers to the expectation of the corresponding product of values over all possible assignments of $\boldsymbol{v}$ and $\boldsymbol{h}$, weighted by their distribution $P(\boldsymbol{v}, \boldsymbol{h})$ given by (2.16).
This gives a very simple SSD update rule on the weight or bias w_{ij}:

$$\begin{aligned} \Delta w_{ij} &\propto \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{v}\in\mathcal{D}} \frac{\partial \log P(\boldsymbol{v})}{\partial w_{ij}} \\ &= \langle s_i s_j \rangle_{\mathcal{D}} - \langle s_i s_j \rangle_{model} \end{aligned} \tag{2.19}$$

where $\langle . \rangle_{\mathcal{D}}$ refers to the average of all expectations $\langle . \rangle_{\boldsymbol{v}}$ for $\boldsymbol{v}$ in $\mathcal{D}$.
Computing the expectations $\langle . \rangle_{\boldsymbol{v}}$ and $\langle . \rangle_{model}$ is not easy in general, as it involves computing expectations over large sets of values given by probability distributions of variables that have (in general) many correlations. That is why we rely on approximate methods that give us approximations of these expectations. The most commonly used method for BMs is called

Gibbs sampling as it only requires to be able to compute the conditional probability of any unit (visible or hidden) given the value of all the other units in the BM, which is easy to do as we will show below. This allows to construct two Markov Chains that have as stationary distributions respectively $P(\boldsymbol{h}|\boldsymbol{v})$ and $P(\boldsymbol{v},\boldsymbol{h})$, derived from and given by (2.16). Then by mixing these MCs until convergence to their stationary distributions, or what is called *thermal equilibrium* in the BM literature, and collecting a few samples of the visible and hidden units at convergence, we can then compute approximations of the expectations $\langle . \rangle_{\boldsymbol{v}}$ and $\langle . \rangle_{model}$ respectively. The only difference between these two MCs is that the first starts at a configuration $(\boldsymbol{v},\boldsymbol{h})$ where $\boldsymbol{h}$ is initialized randomly and $\boldsymbol{v}$ is set and fixed (or *clamped*, as it is called in the BM literature) during all the updates of the MC to a value from the training set, while for the second MC $(\boldsymbol{v},\boldsymbol{h})$ are both initialized randomly and updated at each step.

All we have left to specify in order to have a complete training procedure is then the conditional probability of any unit given the values of all the others, also called the *activation* of that unit. For that, we first call *energy gap* ΔE_j of a certain visible/hidden unit s_j the difference in the negative energy (2.15) of the BM when that unit is active, i.e., has value 1, and when it is not, i.e., has value 0, while all other units are kept unchanged. We have:

$$\Delta E_j = E(s_j = 0) - E(s_j = 1) = \sum_{i \neq j} w_{ij} v_i + \sum_{k \neq j} w_{kj} h_k + b_j \tag{2.20}$$

i.e., ΔE_j is equal to the input of the unit s_j.
One can compute from (2.16) that, interestingly, the activation of unit s_j is given by:

$$\begin{aligned} P(s_j = 1 | \{s_{j'}\}_{j' \neq j}) &= \sigma(\Delta E_j) \\ &= \frac{1}{1 + e^{-\Delta E_j}} \end{aligned} \tag{2.21}$$

where σ is called the *logistic function*.

2.4.3 Restricted Boltzmann Machine

As explained in the last section, *inference* on a BM, i.e., computing *exactly* properties about the probability distribution it encodes, may it be expectation values of its units or just a sample from that distribution, is hard in general. It is actually equivalent to computing the *partition function* Z appearing in the BM probability distribution (2.16), i.e., summing over an exponentially large

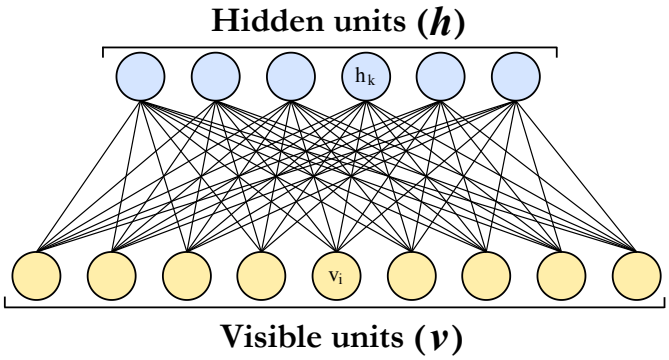

Figure 2.6: **A Restricted Boltzmann Machine.** There are no connections between two visible or two hidden units.

number of terms in the number of units in the BM. That is why we commonly rely on Gibbs sampling to get approximations of these properties. Now, intuitively we can see from Eq. (2.20) and (2.21) that if we restrict the connections in our BM such that there are no connections between two hidden or two visible units, then the probabilities $P(\boldsymbol{h}|\boldsymbol{v})$ and $P(\boldsymbol{v}|\boldsymbol{h})$ can factorize to respectively $\prod_k P(h_k|\boldsymbol{v})$ and $\prod_i P(v_i|\boldsymbol{h})$. This means that given visible (hidden) units, one can sample all hidden (visible) units in parallel because the factorization is equivalent to their pairwise conditional independence given the visible (hidden) units. This implies that we can efficiently parallelize the Gibbs sampling process: computing $\langle . \rangle_{\boldsymbol{v}}$ can now be done in one step, and computing $\langle . \rangle_{\boldsymbol{model}}$ can be done by an MC where the transitions are alternative block updates of all the hidden units or all the visible units at the same time.

This simpler model (referred to as a sparser model in ML terminology), depicted in Figure 2.6, is called a *Restricted Boltzmann Machine (RBM)* and has the following expression for its energy:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i,k} w_{ik} v_i h_k - \sum_i b_i v_i - \sum_k b_k h_k \tag{2.22}$$

to be compared to Eq. (2.15).

Unfortunately, it is shown that the *approximate inference*, done for instance with Gibbs sampling, of RBMs is still hard. More precisely, Long and Servedio (2010) prove that, if P≠NP, then returning a probability value that approximates (2.17) up to an exponentially large multiplicative factor for all $\boldsymbol{v}$ cannot be done in polynomial time in the number of units of the RBM. Gibbs sampling on an RBM, for instance, is shown in Appendix A to have an exponentially small convergence rate in the number of units of the RBM. This is solved in classical literature by so-called *Contrastive Divergence (CD)*, which acts by restricting the number of steps of the MC

produced by the Gibbs sampler that are actually performed, but leads to large approximation errors in the produced distributions (Fischer and Igel, 2011).
This clearly motivates the use of Quantum Algorithms to try speeding-up the inference process without having to trade-off the approximation accuracy.

2.5 Quantum Amplitude Amplification with Szegedy Operators

Having presented classical Markov Chains in section 2.1, we now look into their quantization, meaning that we describe some of the tools that can be used to mix classical MCs, i.e., produce their stationary distribution, in the quantum domain and compare their time complexity with classical ones. These tools essentially consist, first, in a quantum walk operator (constructed out the quantum counterparts of the transition matrix and its reversal) that we can use to construct a reflector around an encoding of the stationary distribution of the associated MC. Add to that Amplitude Amplification (Brassard et al., 2002), a generalization of Grover's search algorithm (Grover, 1996), using this reflector to prepare a coherent encoding of either the stationary distribution, or of a tail distribution on a subset of the elements in the state space of the MC, i.e., a resampling of the stationary distribution with support only on that subset of elements.

2.5.1 Szegedy quantum walk operator

Given a transition matrix $P = (p_{ij})_{N\times N}$, the associated Szegedy walk operator $W(P)$ (Szegedy, 2004; Magniez et al., 2011) is constructed as follows. We first define the associated diffusion operator U_P, a unitary acting on two quantum registers containing the states of the MC, as $U_P \left|c_i\right\rangle_1 \left|0\right\rangle_2 = \left|c_i\right\rangle_1 \sum_{j=1}^{N} \sqrt{p_{ij}} \left|c_j\right\rangle_2$, i.e., one application of U_P on a coherent encoding of a distribution $\boldsymbol{v}$ over the states of the MC in the first quantum register produces a coherent encoding of $P\boldsymbol{v}$ in the second register. Similarly, one can define the *time-reversed* diffusion operator V_P out of the time reversal of P, but for simplicity we only consider time-reversible MCs (always assumed from this point on), such that $V_P = \textsc{Swap}\ U_P\ \textsc{Swap}$ and the Szegedy Operator is the unitary:

$$W(P) := V_P(Z_1 \otimes \mathbb{I}_2)V_P^\dagger\ U_P(\mathbb{I}_1 \otimes Z_2)U_P^\dagger \tag{2.23}$$

where SWAP interchanges the first and second registers and $Z_k = \mathbb{I}_k - 2\,|0\rangle\langle 0|_k$.
The interesting properties of the Szegedy Operator are, first, that it acts non-trivially only on the subspace $A+B$, where $A = \text{span}_{c_i}\{U_P\,|c_i\rangle_1\,|0\rangle_2\}$ and $B = \text{span}_{c_i}\{V_P\,|0\rangle_1\,|c_i\rangle_2\}$, and because $W(P)$ can be rewritten as $\text{Ref}(A)\text{Ref}(B)$, when P is moreover aperiodic and irreducible we have that the intersection $A \cap B$ only contains the state $|\boldsymbol{\pi'}\rangle = U_P\,|\boldsymbol{\pi}, 0\rangle = V_P\,|0, \boldsymbol{\pi}\rangle$. When we restrict our attention to the $A + B$ subspace, $|\boldsymbol{\pi'}\rangle$ is the only $+1$-eigenstate of $W(P)$, while all the other eigenvalues of $W(P)$ are given by $\{e^{\pm 2i\theta_l}\}_{l\in\{1,\,...,\,N-1\}}$, where $\{\cos(\theta_l)\}_{l\in\{0,\,...,\,N-1\}}$ are the eigenvalues of P. Then, by noting $\delta = \min_l\{1 - |\cos(\theta_l)|\}$ the spectral gap of P, $\Delta = \min_l\{\theta_l\}$ the phase gap of $W(P)$ and that $|\cos(\theta_l)| \geq 1 - \frac{\theta_l^2}{2}$, we have: $\Delta \geq \sqrt{2\delta}$.

2.5.2 Reflector around $|\boldsymbol{\pi}\rangle$

From the Szegedy Operator, we can design an *Approximate Reflection Operator* (or *Reflector*) around the coherent encoding of the stationary distribution $|\boldsymbol{\pi}\rangle$, which is an approximation of $\text{Ref}(\boldsymbol{\pi}) = \mathbb{I} - 2\,|\boldsymbol{\pi}\rangle\langle\boldsymbol{\pi}|$. For that, we use a modified version of Kitaev's Phase Estimation (PE) algorithm (Kitaev, 1995): using $\mathcal{O}(\sqrt{\delta^{-1}}\log\epsilon^{-1})$ applications of $W(P)$ we can approximately perform the transformation

$$|\Psi\rangle\,|0\rangle = \sum_l \Psi_l\,|w_l\rangle\,|0\rangle \quad \longmapsto \quad \Psi_0\,|\boldsymbol{\pi}\rangle\,|1\rangle + \sum_{l\neq 0}\Psi_l\,|w_l\rangle\,|0\rangle$$

where ϵ is the trace distance between the resulting state and the desired output state, and $|w_l\rangle$ are the eigenvectors of $W(P)$.
Having this transformed state, we can perform a controlled ϕ-rotation on the auxiliary flag qubit, which gives an approximation of:

$$\text{Ref}_\phi(\boldsymbol{\pi}) = e^{i\phi}\,|\boldsymbol{\pi}\rangle\langle\boldsymbol{\pi}| + (\mathbb{I} - |\boldsymbol{\pi}\rangle\langle\boldsymbol{\pi}|) \tag{2.24}$$

equal to $\text{Ref}(\boldsymbol{\pi})$ for $\phi = \pi$.

2.5.3 Quantum Amplitude Amplification to/from $|\boldsymbol{\pi}\rangle$

Grover (1996) presented in 1996 an algorithm to search for marked elements in an unordered database of N elements when having oracular access only to a unitary performing the reflection $\text{Ref}(\boldsymbol{\pi})$ around the uniform superposition over marked elements $|\boldsymbol{\pi}\rangle = \frac{1}{\mathcal{N}}\sum_{i\ \text{marked}}|x_i\rangle$ ($\mathcal{N}$ is a

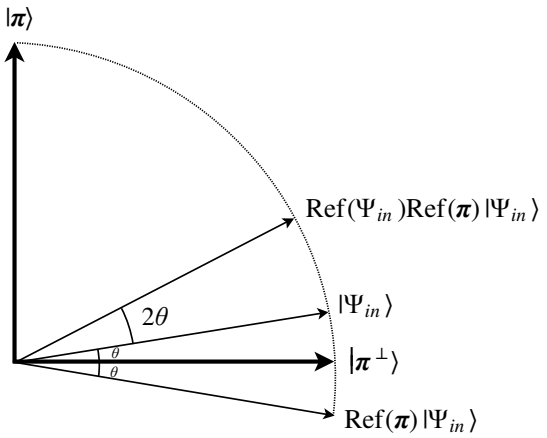

Figure 2.7: **Quantum Amplitude Amplification**

normalization factor). The algorithm starts from the uniform superposition over all elements in the database $|\boldsymbol{u}\rangle$ and works by sequentially applying a constructed reflection $\mathrm{Ref}(\boldsymbol{u})$ around $|\boldsymbol{u}\rangle$ and $\mathrm{Ref}(\boldsymbol{\pi})$. The query complexity, i.e., in number of oracular calls to $\mathrm{Ref}(\boldsymbol{\pi})$, of the algorithm is of the order $\tilde{\mathcal{O}}(\sqrt{N}\sqrt{\epsilon})$, assuming we know the proportion ϵ of marked elements in the database. This algorithm was later generalized by Brassard et al. (2002) to allow the transformation of an arbitrary initial quantum state $|\boldsymbol{\Psi}_{in}\rangle$ to a final quantum state $|\boldsymbol{\pi}\rangle$, having the ability to construct the reflectors $\mathrm{Ref}(\boldsymbol{\pi})$ and $\mathrm{Ref}(\boldsymbol{\Psi}_{in})$. The query complexity of this algorithm is of the order $\tilde{\mathcal{O}}(\sqrt{\gamma^{-1}})$ calls to $\mathrm{Ref}(\boldsymbol{\pi})$ and $\mathrm{Ref}(\boldsymbol{\Psi}_{in})$, assuming we know the overlap $\gamma = |\langle\boldsymbol{\Psi}_{in}|\,\boldsymbol{\pi}\rangle|^2$ between $|\boldsymbol{\Psi}_{in}\rangle$ and $|\boldsymbol{\pi}\rangle$. A representation of these elements is given in Figure 2.7.

The problem is that we commonly do not know in advance the overlap γ between our initial and final states, which prevents the output state of the algorithm to get arbitrarily close to the target final state. That is why Grover (2005) introduced the so-called *fixed-point* version of QAA, later improved to an optimal number of queries by Yoder et al. (2014), that instead assumes the ability to construct the *partial* reflectors $\mathrm{Ref}_\phi(\boldsymbol{\pi})$ and $\mathrm{Ref}_\phi(\boldsymbol{\Psi}_{in})$ to perform the transformation $|\boldsymbol{\Psi}_{in}\rangle \to |\boldsymbol{\pi}\rangle$ with $\tilde{\mathcal{O}}(\sqrt{\gamma'^{-1}})$ calls to $\mathrm{Ref}_\phi(\boldsymbol{\pi})$ and $\mathrm{Ref}_\phi(\boldsymbol{\Psi}_{in})$, where γ' is now a lower bound on the overlap $|\langle\boldsymbol{\Psi}_{in}|\,\boldsymbol{\pi}\rangle|^2$.

Knowing how to construct $\mathrm{Ref}_\phi(\boldsymbol{\pi})$ for the coherent encoding of the stationary distribution of our MC P of interest, we now can i) either prepare $|\boldsymbol{\pi}\rangle$ and, if we know how to construct $\mathrm{Ref}_\phi(\boldsymbol{\Psi}_{out})$, transform it to a final state $|\boldsymbol{\Psi}_{out}\rangle$ in a time complexity of the order $\tilde{\mathcal{O}}(\sqrt{\epsilon^{-1}\delta^{-1}})$, where $\epsilon = |\langle\boldsymbol{\Psi}_{in}|\,\boldsymbol{\Psi}_{out}\rangle|^2$ and δ is the spectral gap of P, or ii) prepare a chosen initial state $|\boldsymbol{\Psi}_{in}\rangle$ and transform it to $|\boldsymbol{\pi}\rangle$ in time complexity $\tilde{\mathcal{O}}(\sqrt{\gamma^{-1}\delta^{-1}})$, where $\gamma = |\langle\boldsymbol{\Psi}_{in}|\,\boldsymbol{\pi}\rangle|^2$.

3 Details of question & motivation

We saw in the previous chapter, section 2.3, that PS is a very interesting model for RL, due to various reasons. These include its simplicity yet efficiency in some benchmark tasks (Melnikov, Makmal and Briegel, 2018), its interpretability, an important aspect when used to learn new quantum experiments for instance (Melnikov, Nautrup, Krenn, Dunjko, Tiersch, Zeilinger and Briegel, 2018), and, most importantly in the context of Quantum Machine Learning, its quantization (Paparo et al., 2014). This last aspect describes the ability to design a quantum algorithm to run the agent's deliberation process that provides a certain speed-up compared to the classical algorithm. In this chapter, we will motivate possible improvements on that quantization for it to support generalization properties.

3.1 Generalization in Projective Simulation

One limiting aspect of the basic two-layered PS presented in Section 2.3.1 is its lack of generalization. This feature describes the ability of an agent to relate its previous experience to new unencountered situations or put in another way, its ability to abstract from received percepts into their important features and subsequently use these to deliberate on an action to perform. In fact, when no mechanism allows to abstract received percepts, an RL (in particular PS) agent has to learn a good policy independently for each percept it receives, thus rendering learning hard, i.e., it requires many interaction steps, for environments with a large percept space. For that reason, Melnikov et al. (2017) introduced a first mechanism that could exploit the ability of the ECM of a PS agent to grow dynamically during its interaction with the environment in order to enable such an abstraction feature. This mechanism namely constructs new intermediary clips between percept-clips and action-clips which purpose is to gather many percept-clips together and allow them to share the same weights with the action-clips. These intermediary

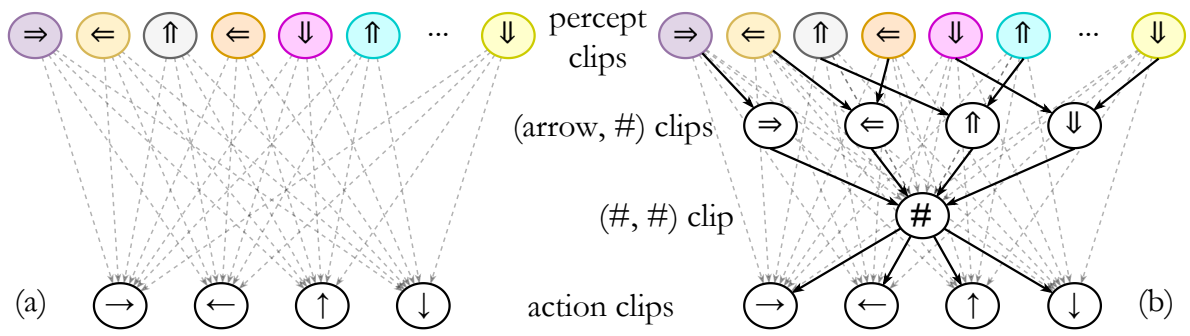

Figure 3.1: **Generalization in PS.** The basic two-layered (a) and the generalized (b) PS networks for agents that receive percepts of the form (arrow, color) and never perceive the same color twice. For clarity, only one-level edges to/from wildcard clips are solid.

clips are called *wildcard-clips* and their construction relies on an essential property of the percept space: its *factorization*. This property indicates that a percept is, by definition or after projections on several subspaces of the per cept space, separable (or factorable) in many components, each of them supposedly describing a different aspect of the percept. This property is intrinsically dependent on the environment in question and the assumptions made on it. For some environments, factorization is natural, as for the case of a robot interacting with the world through multiple sensors: its percepts are separable between the inputs of each of its sensors. For other environments, this assumption relies on some domain-specific knowledge pre-specified to the agent, for instance in the form of pre-trained NNs that extract separable features out of the raw percepts, later fed to the agent. This is the idea behind Deep RL applications such as (Mnih et al., 2015) that use the first layers of Convolutional NNs to extract features from the raw screen of an Atari game, further processed in the remaining layers of the NN to approximate the Q-function of an RL agent.

To create wildcard-clips, an agent compares at each time-step of its interaction with the environment its received percept with previously visited percept-clips and already created wildcard-clips. It then comes up with new relevant wildcard-clips specified by wildcard values for a subset of their features and fixed values for the complementary subset. For instance if the received percept is compared to an old percept and shares $1 \leq l \leq |C|$ features with it (where $|C|$ is the total number of features), a wildcard-clip with wildcard values for the l different features will be added to the l-th layer of the ECM. These wildcard-clips have as parent clips, i.e., clips connected to them in a directed way, all the percept-clips and existing wildcard-clips that share

the same values for the subset of fixed features. They also have as child clips, i.e., clips they are connected to in a directed way, all the action-clips and other wildcard-clips that are further abstractions of them, meaning that they have wildcard values for at least one of their fixed features. This allows gathering together percept-clips according to common features considered to be relevant for the deliberation process, where the relevance comes from the fact that the wildcard clip was actually created, and from the weights in the ECM that favor some wildcard-clips. A comparison of the ECMs built by a basic two-layered PS agent and a PS agent with generalization in a specific task is given in Figure 3.1.

This generalization feature is shown to be a necessity to learn in certain extreme (from the tabular perspective) environments. Namely, these include an environment that would never send twice the same percept to the agent but nevertheless where all the percepts share common features that are sufficient to learn an optimal policy through appropriate generalization. Even though this may seem at first as an unrealistic environment, it actually illustrates the asymptotic scenario that an agent may encounter when dealing with an environment having a large state space. Take for instance the case of a robot having as only input the image fed by a digital camera: it has high probability of never perceiving the *exact* same image twice, but by extracting the relevant features out of its input, it can deal with similar inputs in the same way.

This generalization mechanism in PS is actually very appealing because it is a mechanism that puts into practice the ability of an ECM to grow dynamically in a functional way, hence representing an instance of *composition*. Nevertheless, one may want to implement a more general generalization method that would encompass such a mechanism of favoring some features of the percepts for the deliberation process, while avoiding to add up to an exponential (in number of features $|C|$) number of clips to the ECM and an even bigger number of transition weights to be trained. Such a mechanism would, for instance, rely on a model that would parameterize the transition probabilities in the ECM, such that one could update one parameter of the model and modify all the probabilities of the ECM to guide them in the desired direction. The number of parameters of such a model would just need not to be too large with respect to the size of the percept space.

3.2 Enhancing the quantization

As was described in the section 2.5, Szegedy walk operators can be used to perform Quantum Amplitude Amplification for two different purposes: either prepare a coherent encoding of the stationary distribution of a MC starting from a specified initial state, or if we already have access to a coherent encoding of that stationary distribution (for instance in an oracular way, or if it is easy to prepare) starting from it we can prepare a coherent encoding of the tail-distribution that has support over a subspace of the state space of the MC only. The complexity of the first algorithm scales as $\tilde{\mathcal{O}}(\sqrt{\gamma^{-1}\delta^{-1}})$ (compared to $\tilde{\mathcal{O}}(\delta^{-1})$ classically), where γ is the overlap between the coherent encoding of the stationary distribution and the initial state, while the complexity of the second algorithm scales as $\tilde{\mathcal{O}}(\sqrt{\varepsilon^{-1}\delta^{-1}})$ (compared to $O(\varepsilon^{-1}\delta^{-1})$ classically), where ε is the weight of the target tail-distribution in the stationary distribution. In the case of the *two-layered* PS, the MC behind the ECM actually mixes in one step, which gives a trivial spectral gap $\delta = 1$, meaning that we have an easy-to-prepare stationary distribution (making the first algorithm irrelevant) and that only the second algorithm is of potential interest. One can then use this second algorithm for the deliberation process of a *Reflecting* PS (or RPS) agent, presented in section 2.3.2, constructed out of two-layered ECMs. Indeed, by adding the notion of *flagged* actions to the ECM, the goal of the deliberation process is now to sample from the tail-distribution of the ECM that has support only on these flagged actions. This is the idea behind (Paparo et al., 2014).

Unfortunately, it becomes unclear how to map the generalization feature of Melnikov et al. (2017), introduced in the last section, into the RPS model, as the MC behind the resulting ECM doesn't mix in one step anymore ($\delta < 1$ in general, so we cannot prepare the stationary distribution easily) but more importantly, because the MC is not time-reversible in general (and we don't have an applicable transformation to make it time-reversible), rendering the quantum RPS algorithm inapplicable.

One can then be interested in finding out new designs for RPS agents that by definition come with generalization features and non-trivially mixing time-reversible MCs for their deliberation process, such that they could benefit from a quantum speed-up through the use of the second algorithm.

4 Literature Review & Previous Work

Now that we have given the general notions that are dealt with in this report and motivated our question, we present the previous work from which we derive our results and describe different approaches to that question found in the literature.

4.1 RBM-based Reinforcement Learning

Sallans and Hinton (2004) introduced a new model combining Function Approximation and Action Selection, i.e., the deliberation mechanism of an agent, in RL. This model encodes the policy of an agent in an RBM, parameterized by the Q-function of the agent. This Q-function is itself "approximated" by the so-called *free energy* of the RBM, which by property has tractable derivatives with respect to the weights of the RBM, hence allowing to derive weight update formulas based on the Q-learning update rule of the Q-function (Formula (2.13)). Here we provide the details of this model.

4.1.1 Adapting an RBM for Function Approximation in RL

The main idea in (Sallans and Hinton, 2004) was to use an RBM to encode a joint probability distribution over states and actions by assigning to each of them complementary subsets of the visible units. One can then derive a policy, i.e., a conditional probability over actions given a state, from this joint distribution by looking at it when the states units are fixed to represent a certain state. To understand this more abstractly note that any stochastic function, including a policy, can be encoded in a bipartite distribution, and since the RBM is a *universal approximator*, i.e., it can approximate any distribution to any precision given sufficiently many units (Le Roux and Bengio, 2008), thus, it can then approximate any policy.

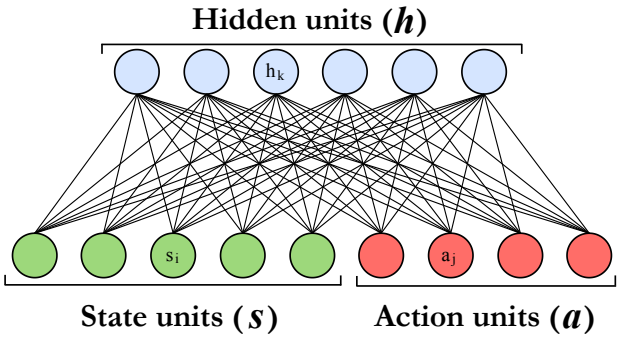

Figure 4.1: **An RBM to do RL.** The visible units are split between state and action units.

The expression of the energy of a state-action-hidden configuration first becomes:

$$E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h}) = -\sum_{i,\ k} w_{ik}s_ih_k - \sum_{j,k} w_{jk}a_jh_k - \sum_i b_is_i - \sum_j b_ja_j - \sum_k b_kh_k \tag{4.1}$$

and the joint probability distribution of that configuration:

$$P(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h}) = \frac{e^{-E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h})/T}}{\sum_{\boldsymbol{s}',\boldsymbol{a}',\boldsymbol{h}'} e^{-E(\boldsymbol{s}',\boldsymbol{a}',\boldsymbol{h}')/T}} \tag{4.2}$$

T is an additional temperature hyperparameter ($T = 1$ cancels the temperature).

Because we are seeking for the marginal probability distribution over ($\boldsymbol{s}$,$\boldsymbol{a}$), we marginalize $\boldsymbol{h}$:

$$P(\boldsymbol{s},\boldsymbol{a}) = \frac{e^{-F(\boldsymbol{s},\boldsymbol{a})/T}}{\sum_{\boldsymbol{s}',\boldsymbol{a}'} e^{-F(\boldsymbol{s}',\boldsymbol{a}')/T}},\ e^{-F(\boldsymbol{s},\boldsymbol{a})/T} = \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h})/T} \tag{4.3}$$

where $F(\boldsymbol{s},\boldsymbol{a})$ is called the *free energy* of the RBM.

Starting from the conditional distribution of hidden units given state and action units:

$$P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a}) = \frac{e^{-E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h})/T}}{\sum_{\boldsymbol{h}'} e^{-E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h}')/T}} \tag{4.4}$$

we can derive an expression of the free energy:

$$\begin{aligned}
F(\boldsymbol{s},\boldsymbol{a}) &= -T \times \log\left(\sum_{\boldsymbol{h'}} e^{-E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h'})}\right) \\
&= -T \times \log\left(\frac{e^{-E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h})}}{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})}\right) \quad \forall \boldsymbol{h} \\
&= E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h}) + T \times \log\left(P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})\right) \quad \forall \boldsymbol{h} \\
&= \sum_{\boldsymbol{h}} P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})\left[E(\boldsymbol{s},\boldsymbol{a},\boldsymbol{h}) + T \times \log\left(P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})\right)\right] \\
&= E(\boldsymbol{s},\boldsymbol{a},\langle h\rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})}) - T \times S_{entropy}(\langle h\rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})})
\end{aligned} \tag{4.5}$$

where $\langle h\rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})} = \left(\langle h_k\rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})}\right)_k$ and $\langle h_k\rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})}$ is the expectation of the value of the hidden

unit h_k under the probability distribution $P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})$ given by Formula (4.4).

Especially, we have $\langle h_k \rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})} = \sigma\left(\sum_i w_{ik}s_i + \sum_j w_{jk}a_j + b_k\right)$, where σ is the logistic function presented in Eq. (2.21).

We can then express the conditional probability distribution over actions given a state:

$$P(\boldsymbol{a}|\boldsymbol{s}) = \frac{e^{-F(\boldsymbol{s},\boldsymbol{a})/T}}{\sum_{\boldsymbol{a}'} e^{-F(\boldsymbol{s},\boldsymbol{a}')/T}} = \frac{e^{-F(\boldsymbol{s},\boldsymbol{a})/T}}{Z_{\boldsymbol{s}}} \tag{4.6}$$

where $Z_{\boldsymbol{s}}$ is called the *partition function* of this Gibbs distribution.

This conditional probability distribution over actions given a state is used to encode a Boltzmann policy for our agent (see Formula (2.12)), and as this policy is parameterized by the Q-function in "classic" methods of RL, we choose to consider the negative free energy of the RBM as a Q-function parameterizing the encoded conditional probability distribution:

$$Q(\boldsymbol{s},\boldsymbol{a}) \leftarrow -F(\boldsymbol{s},\boldsymbol{a}) \tag{4.7}$$

The advantage of using the free energy of an RBM as a Q-function instead of a table listing its values is that this function is defined on the entire state-action space as opposed to a sparsely filled table. Moreover, it is parameterized by a small number of parameters, i.e., the weights and biases of the RBM, relatively to the size of its input space. This means it can be efficiently computed and, as we will see in Section (4.1.3), easily updated to best fit the target values we want to assign to it.

4.1.2 Sampling from the encoded policy (Policy Improvement)

Computing the partition function Z_s is hard for large action spaces as, given a state, we must evaluate the free energy of all state-action pairs having that state to get it. For this reason we rely on an approximate sampling algorithm called a *sampler* to get samples from our encoded policy, i.e., the conditional probability distribution over actions given a state $P(\boldsymbol{a}|\boldsymbol{s})$, without having to compute the partition function $Z_{\boldsymbol{s}}$.

One example of such sampler is the *Gibbs sampling* algorithm (Geman and Geman, 1984) (an instance of the *Metropolis-Hasting* algorithm), which relies on the ability to compute efficiently a vector proportional (up to the partition function) to a target *multivariate* probability distribution of a random variable $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ with multiple components X_i, or more operationally

on the ability to compute efficiently the conditional distribution of each component X_i given the value of the others. In our case the random variable $\boldsymbol{X}$ comprises the action units and hidden units of the RBM $\boldsymbol{X} = ((a_1, \ldots, a_{|A|}), (h_1, \ldots, h_{|H|}))$, and indeed these two properties are satisfied.

Algorithm 1 Block Gibbs Sampling for Action Selection on an RBM

Input: A state $\boldsymbol{s}$, the conditional probability distributions given by Formulas (5.6) and (5.7)

Parameter: The number of iterations N needed to reach thermal equilibrium

Output: An action $\boldsymbol{a}$ approximately sampled from the conditional probability distribution (4.6)

1: Initialize $\boldsymbol{X}^{(0)} = ((a_1, \ldots, a_{|A|}), (h_0, \ldots, h_{|H|}))$ (the action and hidden units) randomly
2: Compute $\boldsymbol{X}^{(i+1)}$ from $\boldsymbol{X}^{(i)}$ (N times, until thermal equilibrium):
3: **if** i is odd **then**
4: Sample hidden units according to their conditional probability given action units and state units in parallel
5: **else**
6: Sample action units according to their conditional probability given hidden units and state units in parallel
7: **end if**
8: **return** The last sample of action units

This is actually a *Markov Chain Monte Carlo* process having as stationary distribution the conditional joint distribution over actions and hidden values given a state encoded in the RBM. It has a non-trivial mixing time, shown to be exponential in the number of units of the RBM and a measure of the amplitude of its weights in Appendix A. In practice, one additionally relies on some known methods such as *simulated annealing*, i.e., a schedule of gradually decreasing temperature T is imposed during the MCMC process, but note that these are only heuristic methods without quantifiable speed-up.

4.1.3 Updating the RBM (Policy Evaluation)

Let us now look at how we can translate an update formula on a Q-function given by a value-based RL method into an update on the parameters, i.e., weights and biases, of the RBM. To illustrate this, we take the same example as in (Sallans and Hinton, 2004), namely the Q-learning

update rule (see Eq. (2.13)).

Once we ran our sampler on the RBM for a given state clamped on its state units, executed the resulting action on the environment and got a reward back, we have to update the policy that is encoded in the RBM. Because it is parameterized by the Free-energy of the RBM and that we consider this latter as a Q-function, we need to relate known update rules of the Q-function to updates of the free energy of the RBM.

The general method used to update the free energy of the RBM is *gradient descent* on an error (or difference) between our target free energy (or Q-function) and our current one. For that purpose, we first specify that our free energy $F(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta})$ (and subsequently our Q-function $Q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta})$) is itself parameterized by the weights and biases of the RBM, gathered in a vector called $\boldsymbol{\theta} = (w_{xy}, \ldots, b_x, \ldots)$. We are doing gradient descent on these parameters.

We then re-express the Q-learning update rule (see Eq. (2.13)) as a *Temporal Difference (TD) error*, i.e., difference between a target estimate of the return of a given state-action pair and its current estimated value, defined for a visited state-action pair at time-step t by:

$$\mathcal{E}_{TD}^{\boldsymbol{\theta}}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) = [r^{(t+1)} + \gamma_{env} \max_{\boldsymbol{a}} Q(\boldsymbol{s}^{(t+1)}, \boldsymbol{a}; \boldsymbol{\theta})] - Q(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta}) \tag{4.8}$$

and it is equal to zero for all other state-action pairs.

This TD-error $\mathcal{E}_{TD}^{\boldsymbol{\theta}}$ can be visualized as a big table gathering the differences between our target Q-function and the current Q-function for all state-action pairs. We are trying to minimize its squared L_2-norm $\left(\|\mathcal{E}_{TD}^{\boldsymbol{\theta}}\|_2\right)^2$. For that we can adopt a common method in ML and optimization literature, that is *gradient descent*. This consists in applying small modifications in the parameters of an objective function, here $\left(\|\mathcal{E}_{TD}^{\boldsymbol{\theta}}\|_2\right)^2$, in the direction that minimizes it, i.e., the direction where the gradient is negative. For that we perform steps proportional to the negative of the gradient at the current point.

The gradient descent update rule, i.e., the change in the parameters $\boldsymbol{\theta}$, is then:

$$\begin{aligned}
\Delta\boldsymbol{\theta} &= -\frac{\alpha}{2}\nabla_{\boldsymbol{\theta}} \left(\left(\|\mathcal{E}_{TD}^{\boldsymbol{\theta}}\|_2 \right)^2 \right) \\
&= -\frac{\alpha}{2}\nabla_{\boldsymbol{\theta}} \left(\left(\sqrt{\sum_{\boldsymbol{s}', \boldsymbol{a}'} {\mathcal{E}_{TD}^{\boldsymbol{\theta}}}^2} \right)^2 \right) \\
&= -\frac{\alpha}{2}\nabla_{\boldsymbol{\theta}} \left(\mathcal{E}_{TD}^{\boldsymbol{\theta}}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)})^2 \right) \\
&= -\alpha \mathcal{E}_{TD}^{\boldsymbol{\theta}}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) \nabla_{\boldsymbol{\theta}} \mathcal{E}_{TD}^{\boldsymbol{\theta}}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)})
\end{aligned}$$

$$
\begin{aligned}
&= -\alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) \times \ \nabla_{\boldsymbol{\theta}} \left(\gamma_{env} \max_{\boldsymbol{a}} Q(\boldsymbol{s}^{(t+1)}, \boldsymbol{a}; \boldsymbol{\theta}) - Q(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta}) \right) \\
&\approx \alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) \nabla_{\boldsymbol{\theta}} \left(Q(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta}) \right) \text{[1]} \\
&\approx -\alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) \nabla_{\boldsymbol{\theta}} \left(F(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta}) \right)
\end{aligned}
$$

with α a learning rate in $(0, 1)$, a hyperparameter effectively setting the magnitude of the update.

All we need to do now is relate the derivatives of the free energy with respect to the weights and the biases of the RBM (forming the gradient) to the values of the units in the RBM when a sample of action units was drawn. Conveniently, the derivatives of the free energy of an RBM are tractable and have simple expressions:

$$
\begin{cases}
\frac{\partial F(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta})}{\partial w_{ik}} = -s_i \langle h_k \rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})} \\
\frac{\partial F(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta})}{\partial w_{jk}} = -a_j \langle h_k \rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})} \\
\frac{\partial F(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta})}{\partial b_i} = -s_i \\
\frac{\partial F(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta})}{\partial b_j} = -a_j \\
\frac{\partial F(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}; \boldsymbol{\theta})}{\partial b_k} = -\langle h_k \rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})}
\end{cases}
\tag{4.9}
$$

This finally gives the following update rules:

$$
\begin{cases}
\Delta w_{ik} = \alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) s_i \langle h_k \rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})} \\
\Delta w_{jk} = \alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) a_j \langle h_k \rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})} \\
\Delta b_i = \alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) s_i \\
\Delta b_j = \alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) a_j \\
\Delta b_k = \alpha \mathcal{E}^{\boldsymbol{\theta}}_{TD}(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) \langle h_k \rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})}
\end{cases}
\tag{4.10}
$$

4.2 "Quantum" BM-based Reinforcement Learning

A first extension of Sallans and Hinton (2004) RBM-based RL model to the quantum domain was introduced by Crawford et al. (2016) and later tested on an actual D-Wave quantum annealer by Levit et al. (2017). The authors first extend the model to *Deep* architectures of *Boltzmann Machines* (or *DBM*s), namely the concatenation of many RBMs in multiple layers (see Figure 4.2), justified by the fact that, in order to represent a desired policy with good accuracy, RBMs

[1] The approximation comes from considering $Q(\boldsymbol{s}^{(t+1)}, \boldsymbol{a}; \boldsymbol{\theta})$ constant with respect to the parameters $\boldsymbol{\theta}$

may require an exponential number of hidden units with respect to their number of visible units (Le Roux and Bengio, 2008). Moreover, they note that a DBM layout is close to the architecture of D-Wave quantum annealers with respect to the proximity and couplings their qubits (Harris et al., 2010), making them a good candidate to embed *Quantum Boltzmann Machines* (*QBM*s). These latter are a generalization of BMs to the quantum domain first introduced by Amin et al. (2016), where the energy (2.15) of the model is substituted by the quantum Hamiltonian of a transverse-field Ising spin model, i.e., of the form:

$$H = -\sum_{i,j} w_{ij}\sigma_i^z\sigma_j^z - \sum_i b_i\sigma_i^z - \Gamma\sum_i \sigma_i^x \tag{4.11}$$

where σ_i^z are σ_i^x are respectively the Pauli z and x-matrices acting on qubit i, that acts as the unit i of the BM and $\Gamma \geq 0$ is the strength of the transverse field. Note that this transverse field is the contribution to the Hamiltonian that allows quantum effects, especially *quantum tunneling*, and when decreased gradually effectively gives *quantum annealing* (Shin et al., 2014).

When we consider DBMs instead of RBMs, we loose on essential property for the approach presented in the last section, namely that the free energy (4.5) of a DBM (and especially $S_{entropy}(\langle h\rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})})$) along with its derivatives with respects to the weights are not easily computable anymore. The quantum annealer then comes in handy when we want to approximate the free energy of a certain $(\boldsymbol{s},\boldsymbol{a})$ configuration, as the Hamiltonian (4.11), with fixed values for state and action units corresponding to the configuration of interest, has the energies appearing in $P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})$ (Eq. (4.4)) as eigenvalues. If one samples many times from this Hamiltonian, one then gets many samples from $P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})$ and can hence also evaluate estimates of the expectations $\langle h_k\rangle_{P(\boldsymbol{h}|\boldsymbol{s},\boldsymbol{a})}$ and an approximation of $F(\boldsymbol{s},\boldsymbol{a})$ using Eq. (4.5). This also allows to estimate the derivatives of the free energy of the DBM and devise updates rules similar to Eq. (4.10).

The quantum advantage that this approach promises is in the approximate evaluation of $F(\boldsymbol{s},\boldsymbol{a})$ and values that depend on it, for DBMs. As we are still dealing with parameterized policies, this helps in one aspect of generalization, namely when the state space is large, but doesn't help anymore in the case of additionally large action spaces, as this method, as it is presented, would need to evaluate the free energy an exponential number of times with respect to the number of action units to compute (4.6). Of course, then one could think of i) associating a method such as Gibbs sampling, presented in Alg. 1, to these fast (approximate) evaluations of $F(\boldsymbol{s},\boldsymbol{a})$ in order to sample from (4.6) without explicitly computing its partition function. Or else, ii) one

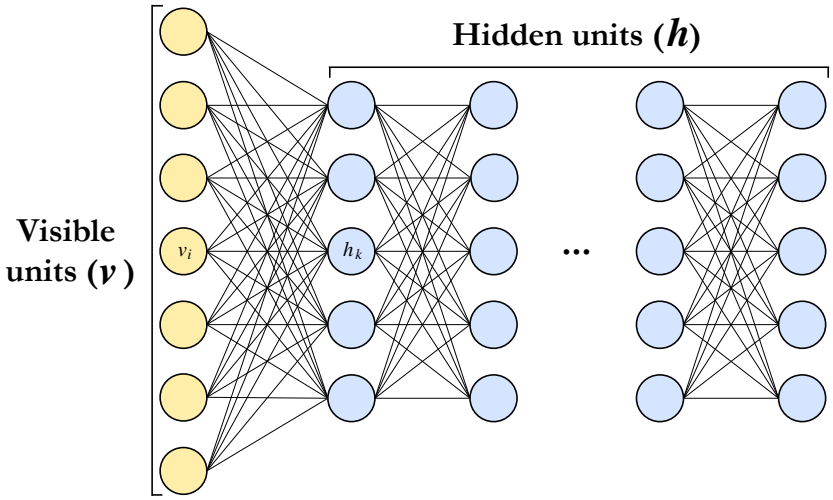

Figure 4.2: **A Deep Boltzmann Machine.** There are multiple hidden layers but no intralayer connections.

could sample from the Hamiltonian (4.11) when only state units are constrained, thus effectively sampling from (4.6) without explicit evaluation. The problems with these two ideas are: for i) the Gibbs sampling process has still an exponential mixing time with respect to the number of units of the DBM, ii) quantum annealing is not guaranteed to find the ground state of the Hamiltonian (4.11), especially when we want to sample from (4.6) at very low temperature[2]. This hence effectively turns our Q-learning update rule (Eq. (2.13)) into the so-called *SARSA* rule:

$$Q(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) \leftarrow (1-\alpha) Q(\boldsymbol{s}^{(t)}, \boldsymbol{a}^{(t)}) + \alpha[r^{(t+1)} + \gamma_{env} Q(\boldsymbol{s}^{(t+1)}, \boldsymbol{a}^{(t+1)})] \tag{4.12}$$

shown in certain instances to fail to converge to an optimal policy, for a Boltzmann policy under lower bounded temperature (Asadi and Littman, 2016).

The quantum speed-up we are seeking in the remaining of this work applies on the mixing time of MC methods, such as the one given by Gibbs sampling, to sample from (4.6). Because we are not dealing with DBMs, the problem of the hardness to compute $F(\boldsymbol{s}, \boldsymbol{a})$ doesn't apply.

[2]Note that evaluating $\max_{\boldsymbol{a}} Q(\boldsymbol{s}^{(t+1)}, \boldsymbol{a})$ for the Q-learning rule (see Eq. (2.13)) is equivalent to sampling from the distribution (4.6) in the limit $T \to 0$, i.e., at very low temperature.

5 Realizing MCMC-based RL Methods in RPS

In Section 2.3.2, we presented the model of RPS and shown how it is used to define a PS agent that runs MC processes for its deliberation. Paparo et al. (2014) have shown that it can thus possibly benefit from a quadratic quantum speed-up for that deliberation process. But so far, RPS has only been applied to the basic *two-layered* PS, a restriction of PS unfortunately lacking generalization. A solution to the problem of generalization was proposed by Melnikov et al. (2017), taking advantage of the ability of the ECM network to grow dynamically during its interaction with the environment. But as of today, there are no published methods to fit this generalized PS into RPS while still benefiting from the former quadratic quantum speed-up.
On the other hand, we presented in Section 4.1 a model based on RBMs allowing to do Function Approximation of a Q-function that is also based on an MCMC process for its deliberation. By contrast, the underlying MC of this model can be made time-reversible by construction and has non-trivial mixing times, thus rendering a potential embedding in RPS of high interest with respect to the possible quadratic quantum speed-up we would gain. In this chapter, we then start by establishing a direct connection between RPS and MCMC-based RL methods inspired by the work of Sallans and Hinton (2004), that we later illustrate by embedding into RPS the RBM-based RL model.

5.1 General formalization of Function Approximation in RPS

What we want to achieve in this section is to describe a general procedure to design RPS agents with the ability to do Function Approximation. To specify an RPS agent, one needs to define an ECM, i.e., its clips and its transition probabilities, an update rule that applies on these transition probabilities and allows to learn from interaction, and finally a deliberation mechanism that returns samples from the policy encoded in the ECM. An additional requirement that we have

here is for the RPS agent to benefit from the generalization features of Function Approximation methods for RL, namely that the updates of the ECM should act on the network more globally than tabular methods and thus allow the agent to act non-randomly in unencountered situations.

5.1.1 Energy-based policies

Because we want for the updates of our RPS agent's ECM to make it learn, we decide to derive them from the update rules of known RL methods. We then start our design by considering *energy-based*[1] policies, that is to say policies that are computed out of a real-valued function defined on the whole state-action space that we call an energy function. This can be for instance a Q-function (Eq. (2.10)) or the h-values behind a two-layered PS (Eq. (2.14)). We call this energy function f. The only constraints that we put on f are that we have update rules that allow learning from the interaction with the environment. If we represent f in a table form where the states index the rows and the actions index the columns, then deriving our policy, i.e., a conditional distribution over actions given a state, corresponds to normalizing the columns of the table following an appropriate transformation. Examples of such transformations were given by formulas (2.11) and (2.12).

5.1.2 Specification of a family of policies

Once we have chosen the energy function f for our policies, we need to specify a parameterized generative model, e.g., a Boltzmann Machine, that encodes our energy-based policies. This generative model has the benefit of being specified by a set of parameters that, when updated, act globally on the encoded policy, hence allowing generalization features. This is where the Function Approximation comes in as the generative model implicitly approximates the function f by encoding probability distributions specified by that function. Gibbs/Boltzmann distributions are an interesting choice of probability distribution for energy-based policies because we know a class of generative models that can encode this type of distributions and are specified by energy functions. These generative models are called *Markov Random Fields (MRF)* and they are represented in the form of undirected graphs, where the nodes of a graph correspond

[1] These are called *parameterized policies* in Section 2.2 and in the literature, but to avoid confusion with the parameterized functions associated with our generative models, we intentionally give them another name here.

to variables and weighted connections between nodes account for dependencies between these variables. The advantage of such a graphical representation is the ability to easily specify which variables factorize and can be considered conditionally independent, thus implying parallelization capabilities during the inference process, as we saw for the RBM in Section 2.4.3.
Having defined our parameterized generative model, its possible configurations, i.e., the assignments of its composing random variables, then specify the clips of our RPS agent's ECM.

5.1.3 Deliberation with a Markov Chain-based method

The generative models that we use only *encode* probability distributions, meaning that deriving these from the model is in general nontrivial and requires an additional process to perform *inference*. And because, similarly to what was explained in Section 2.4.3, for most models of interest (especially MRFs) exact inference is intractable (or hard), we resort to *approximate* schemes (or *approximate inference*). This can take the form of a Markov Chain Monte Carlo, meaning that the additional process will generate an MC which stationary distribution is the encoded probability distribution of our model. Literature about such MCMC algorithms called *samplers*, namely the Metropolis-Hasting algorithm, the Gibbs sampler or the Slice sampler to give a few, is very broad. Interestingly, these (in general) are constructed to satisfy the convergence properties of MCs given in Section 2.1, namely aperiodicity, irreducibility (for the existence of the stationary distribution) and time-reversibility (to bound the convergence rate). Such a MCMC algorithm is the missing piece that serves to define the transition probabilities in our RPS agent's ECM and it is what moreover constitutes our RPS agent's deliberation process.

5.1.4 Devising an update rule

Now, the last step to fully specify our RPS agent is to translate the update rules associated with our chosen energy function f to updates on the parameters of the generative model. The common approach taken with parameterized generative models is to update through *gradient descent*, defined in Section 2.4.2, the parameters of the model in the direction that minimizes a loss function defined out of its encoded probability distribution. Fortunately, when considering energy-based policies, this loss can be derived from the update rule of the energy function f. All we need to do is to reexpress it into a Temporal-Difference error as in Eq. (4.8) to be minimized.

5.2 Embedding the RBM-RL model in RPS

Here we illustrate our specification of RPS agents with Function Approximation abilities by fitting the RBM-based model presented in Section 4.1 into it. Our goal is to show how restating such a model in the RPS framework ultimately achieves the same learning method but enhances the model with additional degrees of freedom given by RPS, namely the ability to tweak some of its transition probabilities and to freely select how its deliberation process should be performed.

Model	**RBM-based RL**	**RPS**
Elements	Configurations of the RBM	Clips of the ECM
	Weights of the RBM + Sampler	Transition probabilities of the ECM
Update rule	Gradient descent on the weights of the RBM	Global implicit update
Deliberation process	Gibbs sampler	Random walk

Table 5.1: Mapping of RBM-based RL components to the RPS framework

5.2.1 Energy-based policies

In the RBM-based model, policies are parameterized by the Q-function according to a Gibbs/ Boltzmann distribution:

$$\boldsymbol{\pi}(\boldsymbol{s}) \sim \frac{e^{Q(\boldsymbol{s},\boldsymbol{a})/T}}{\sum_{a'} e^{Q(\boldsymbol{s},\boldsymbol{a'})/T}} \tag{5.1}$$

where T is the temperature parameter.

5.2.2 Specification of a family of policies

The model we use to encode these distributions over actions given states is an RBM depicted by Figure 4.1. The model hence relies on additional hidden variables (they constitute the hidden layer of the RBM) and parameterizes the Q-function of the agent with the free energy of the RBM. How fine this distribution can be is determined by the number of hidden units of the

RBM, as the number of parameters of an RBM with $|V|$ visible units and $|H|$ hidden units is $\mathcal{O}(|V| \times |H|)$.

5.2.3 Deliberation with a Markov Chain-based method

How RBM-RL specifies an MC to deliberate on an action is through the use of a sampler. In order to run the sampler, we also need to specify the parameters of the RBM, namely its weights and biases. Having all this information, we can devise the ECM of our RPS agent. We illustrate our case with the Gibbs sampler, described by Alg. 1.
We define our clip space to be:

- State/Percept-clips $\boxed{\boldsymbol{s}_n}$ for each state in the state space $\mathcal{S}$
- For each state $\boldsymbol{s}_n$ of the state space $\mathcal{S}$:
  - Intermediary construction clips $\boxed{(\boldsymbol{a}_m, \boldsymbol{h}_l)_{\boldsymbol{s}_n}}$ for each (action, hidden) pair in the space $\mathcal{A} \times \mathcal{H}$
  - Wildcard-action clips of the form $\boxed{(\boldsymbol{a}_m, \#)_{\boldsymbol{s}_n}}$ for each action in the action space $\mathcal{A}$, where the wildcard values refer to an abstraction of the hidden values, gathering all of them together
  - Wildcard-hidden clips of the form $\boxed{(\#, \boldsymbol{h}_l)_{\boldsymbol{s}_n}}$ for each hidden value in the hidden space $\mathcal{H}$, where the wildcard values refer to an abstraction of the actions, gathering all of them together

We consider intermediary construction clips because they intuitively describe the states of the MC behind the Gibbs sampler: at a given iteration of the Gibbs sampling process, we have one configuration of action and hidden units $(\boldsymbol{a}, \boldsymbol{h})$ while the values of state units are fixed during the whole run. We then consider these wildcard-clips because as we can see in Alg. 1, in our Gibbs sampling procedure, the conditional probabilities of actions (hidden values) given hidden values (actions) are independent of their previous value. This allows to design a more compact ECM, i.e., with less clips and less transitions, as we can we gather all hidden values (actions) together in wildcard-clips. We can now discard the intermediary construction clips.

Because at each time-step t of its interaction with the environment, our RPS agent is in *one*

particular state $\boldsymbol{s}_n$ and that we do not allow direct transitions to other states, we can then impose on its ECM to be separable in state-specific sub-graphs on which we restrict our attention. This translates into dividing the transition matrix $P^{(t)}$ behind the ECM in state-specific transition matrices:

$$P^{(t)} = \begin{pmatrix} P^{(t)}_{\boldsymbol{s}_1} & 0 & 0 & 0 \\ 0 & P^{(t)}_{\boldsymbol{s}_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & P^{(t)}_{\boldsymbol{s}_{2|S|}} \end{pmatrix} \tag{5.2}$$

where each state-specific transition matrix $P^{(t)}_{\boldsymbol{s}_n}$ describes the transition probabilities of our sub-graph of interest, depicted in Figure 5.1.

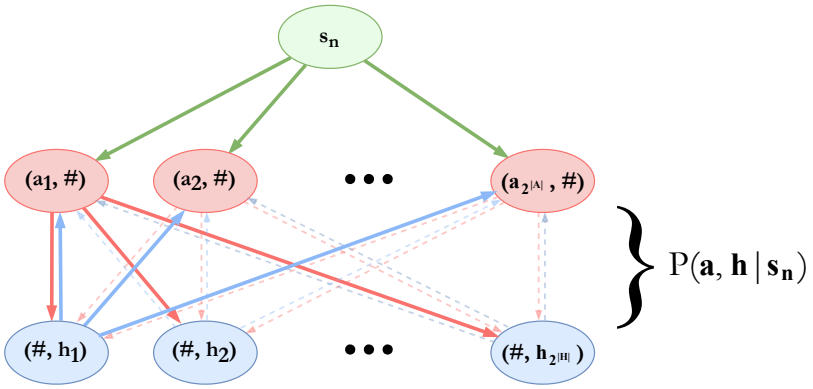

Figure 5.1: **Percept-specific sub-graph of our constructed ECM**

The random walk on the sub-graph associated to $P^{(t)}_{\boldsymbol{s}_n}$ starts at the percept-clip $\boxed{\boldsymbol{s}_n}$, then hops with equal probability on any of the wildcard-action clips $\boxed{(\boldsymbol{a}_m, \#)}$. From there on, the walk stays in the bipartite sub-graph with a layer of wildcard-action clips and a layer of wildcard-hidden clips, hopping back and forth between the two layers.

This walk on the bipartite sub-graph is governed by the following probabilities:

$$P^{(t)}_{\boldsymbol{s}_n} = \begin{pmatrix} 0 & P^{(t)}(\boldsymbol{a} \mid \boldsymbol{h}, \boldsymbol{s}_n) \\ P^{(t)}(\boldsymbol{h} \mid \boldsymbol{a}, \boldsymbol{s}_n) & 0 \end{pmatrix} \tag{5.3}$$

with

$$P^{(t)}(\boldsymbol{h} \mid \boldsymbol{a}, \boldsymbol{s}_n) = \begin{pmatrix} p^{(t)}(\boldsymbol{h}_1 \mid \boldsymbol{a}_1, \boldsymbol{s}_n) & \cdots & p^{(t)}(\boldsymbol{h}_1 \mid \boldsymbol{a}_{2^{|A|}}, \boldsymbol{s}_n) \\ \vdots & \ddots & \vdots \\ p^{(t)}(\boldsymbol{h}_{2^{|H|}} \mid \boldsymbol{a}_1, \boldsymbol{s}_n) & \cdots & p^{(t)}(\boldsymbol{h}_{2^{|H|}} \mid \boldsymbol{a}_{2^{|A|}}, \boldsymbol{s}_n) \end{pmatrix} \tag{5.4}$$

and

$$P^{(t)}(\boldsymbol{a} \mid \boldsymbol{h}, \boldsymbol{s}_n) = \begin{pmatrix} p^{(t)}(\boldsymbol{a}_1 \mid \boldsymbol{h}_1, \boldsymbol{s}_n) & \cdots & p^{(t)}(\boldsymbol{a}_1 \mid \boldsymbol{h}_{2^{|H|}}, \boldsymbol{s}_n) \\ \vdots & \ddots & \vdots \\ p^{(t)}(\boldsymbol{a}_{2^{|A|}} \mid \boldsymbol{h}_1, \boldsymbol{s}_n) & \cdots & p^{(t)}(\boldsymbol{a}_{2^{|A|}} \mid \boldsymbol{h}_{2^{|H|}}, \boldsymbol{s}_n) \end{pmatrix} \tag{5.5}$$

We may now have a closer look at the wildcard-action-specific and wildcard-hidden-specific sub-graphs:

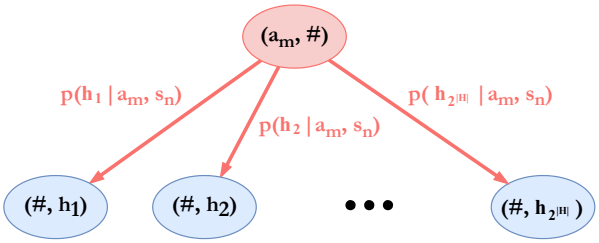

Figure 5.2: **Wildcard-action-specific sub-graph of our constructed ECM**

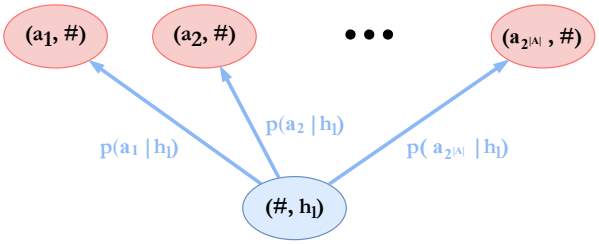

Figure 5.3: **Wildcard-hidden-specific sub-graph of our constructed ECM**

and the conditional probabilities of certain binary configurations of hidden (action) units given

an action (hidden value):

$$\begin{aligned} p(\boldsymbol{h}_l \mid \boldsymbol{s}_n, \boldsymbol{a}_m) &= \prod_k p(h_k \mid \boldsymbol{s}_n, \boldsymbol{a}_m) \\ &= \prod_{k \text{ active}} p(h_k = 1 \mid \boldsymbol{s}_n, \boldsymbol{a}_m) \times \prod_{k' \text{ inactive}} p(h_{k'} = 0 \mid \boldsymbol{s}_n, \boldsymbol{a}_m) \\ &= \prod_{k \text{ active}} \sigma\left(\left(\sum_i w_{ik} s_i + \sum_j w_{jk} a_j + b_k\right)/T\right) \times \\ &\quad \prod_{k' \text{ inactive}} 1 - \sigma\left(\left(\sum_i w_{ik'} s_i + \sum_j w_{jk'} a_j + b_{k'}\right)/T\right) \end{aligned} \tag{5.6}$$

$$\begin{aligned} p(\boldsymbol{a}_m \mid \boldsymbol{s}_n, \boldsymbol{h}_l) = \; p(\boldsymbol{a}_m \mid \boldsymbol{h}_l) &= \prod_j p(a_j \mid \boldsymbol{h}_l) \\ &= \prod_{j \text{ active}} p(a_j = 1 \mid \boldsymbol{h}_l) \times \prod_{j' \text{ inactive}} p(a_{j'} = 0 \mid \boldsymbol{h}_l) \\ &= \prod_{j \text{ active}} \sigma\left(\left(\sum_k w_{jk} h_k + b_j\right)/T\right) \times \\ &\quad \prod_{j' \text{ inactive}} 1 - \sigma\left(\left(\sum_k w_{j'k} h_k + b_{j'}\right)/T\right) \end{aligned} \tag{5.7}$$

where we divide the units of a certain configuration between those that are active, i.e., equal to 1, and those that are not, i.e., equal to 0. The conditional probability of a single unit given all the others is given by Eq. (2.21). The additional simplifications we have here come from the restricted connectivity of an RBM, that allows having additional conditional independence between units.

Let us now show that the MC specified by the transition matrix $P^{(t)}_{\boldsymbol{s}_n}$ converges to the policy encoded by the RBM model.

We consider here the percept-specific bipartite graphs $P^{(t)}_{\boldsymbol{s}_n}$ defined by Eq. (5.3) to (5.7). We start by noticing that all the transitions in that sub-graph have non-zero probability because of the strict positivity of the logistic function used in Eq. (5.6) and (5.7), and that any wildcard-action clip can transit to any wildcard-hidden clip. These are sufficient conditions for irreducibility.

A bipartite graph is clearly not aperiodic but instead has a period of 2 (all its clips have period 2). We can recover aperiodicity simply by considering ${P^{(t)}_{\boldsymbol{s}_n}}^2$ as the transition matrix of our MC instead of $P^{(t)}_{\boldsymbol{s}_n}$ (each step of the new MC process is equivalent to two steps of the previous one). Having irreducibility and aperiodicity, we know from Thm. 2.1.2 that the MC has a stationary

distribution and converges to it.

One can additionally show that the probability distribution given by Eq. (4.6), that we will denote $\pi_{\boldsymbol{s}_n}$, satisfies the detailed balance equation:

$$\pi_{\boldsymbol{s}_n}(\boldsymbol{a}_m) \times \sum_l p(\boldsymbol{a}_{m'}|\boldsymbol{h}_l)p(\boldsymbol{h}_l|\boldsymbol{a}_m, \boldsymbol{s}_n) = \sum_l p(\boldsymbol{a}_m|\boldsymbol{h}_l)p(\boldsymbol{h}_l|\boldsymbol{a}_{m'}, \boldsymbol{s}_n) \times \pi_{\boldsymbol{s}_n}(\boldsymbol{a}_{m'}) \tag{5.8}$$

by noting that from Eq. (5.6) and (5.6), $\sum_l p(\boldsymbol{a}_{m'}|\boldsymbol{h}_l)p(\boldsymbol{h}_l|\boldsymbol{a}_m, \boldsymbol{s}_n) = \frac{e^{-F(\boldsymbol{s}_n, \boldsymbol{a}_{m'})/T}}{e^{-F(\boldsymbol{s}_n, \boldsymbol{a}_m)/T} + e^{-F(\boldsymbol{s}_n, \boldsymbol{a}_{m'})/T}}$ for any $\boldsymbol{a}_m$ and $\boldsymbol{a}_{m'}$ depicted in Figure 5.1. We will not detail the calculation here.
The detailed balance equation gives us by definition time-reversibility of the MC and shows that $\pi_{\boldsymbol{s}_n}$ is its stationary distribution.

5.2.4 Devising an update rule

Once the RPS agent has run its algorithm to mix the MC associated with its ECM, sampled an action from its obtained policy and performed that action on the environment, it can then use the update rules given by Eq. (4.10) to update the weights of the RBM in its memory. This update will modify all the transition probabilities in the ECM, and we particularly emphasize that not only the weights of the percept-specific sub-graph are impacted but every single transition in the entire ECM because they all depend on the weights of the RBM. This is essentially the form of the Policy Evaluation in our model.

Finally, such an RBM-based RPS agent essentially leads to the same learning method as the RBM-based method it was inspired from. We just have re-expressed the MC behind MCMC process that it ran to generate and sample from its policy (in a way, its Policy Improvement) in terms of an ECM. Any algorithm that would mix the intrinsic MC corresponding to that ECM and sample from the obtained (approximation of the) stationary distribution would then yield to the same action selection process.

6 Quantization Methods

We presented in the last chapter a procedure allowing to realize MCMC-based RL methods, such as RBM-based RL (Section 4.1), in RPS. Being inspired by the quadratic quantum speed-up in deliberation time of an RPS agent achieved by Paparo et al. (2014), we now want to look at possible quantization methods, i.e., quantum algorithms, that could allow us to benefit from a certain computational quantum advantage for these newly defined RPS agents. Namely, one possible goal is to prepare and sample from, i.e., measure in the computational basis, the quantum state $|\boldsymbol{\pi}\rangle = \sum_{m=1}^{2^{|A|}} \sqrt{\pi_{\boldsymbol{s}_n}(\boldsymbol{a}_m)}\,|\boldsymbol{a}_m\rangle$, i.e., a coherent encoding of the policy $\pi_{\boldsymbol{s}_n}(\boldsymbol{a})$ of our RPS agent for a given state/percept $\boldsymbol{s}_n$. This policy is conveniently encoded in the percept-specific sub-graph of the agent's ECM as the stationary distribution of its associated MC $P_{\boldsymbol{s}_n}$.

6.1 Naïve approach with Szegedy walk operators

In Section 2.5, we described a tool to perform quantum walks. This tool is the Szegedy walk operator $W(P)$, a unitary built out of the transition matrix P of an MC of interest and that has the property of having a phase gap Δ quadratically larger than the spectral gap δ of P. But because the direct application of $W(P)$ doesn't help to prepare $|\boldsymbol{\pi}\rangle$, we resort to the use of a reflector around $|\boldsymbol{\pi}\rangle$, $\mathrm{Ref}(\boldsymbol{\pi})$, built out of it. This reflector allows us to do Quantum Amplitude Amplification, QAA for short, presented in Section 2.5.3, to amplitude amplify an initial quantum state $|\boldsymbol{\Psi}_{in}\rangle$ to $|\boldsymbol{\pi}\rangle$. The time complexity of this transformation is of the order $\tilde{\mathcal{O}}(\sqrt{\gamma^{-1}\delta^{-1}})$ (compared to $\tilde{\mathcal{O}}(\delta^{-1})$ classically), where $\gamma = |\langle\boldsymbol{\Psi}_{in}|\,\boldsymbol{\pi}\rangle|^2$ is the overlap between the states $|\boldsymbol{\Psi}_{in}\rangle$ and $|\boldsymbol{\pi}\rangle$. The naïve choice for $|\boldsymbol{\Psi}_{in}\rangle$ is to take $|\boldsymbol{\Psi}_{in}\rangle = |u\rangle = \frac{1}{\sqrt{2^{|A|}}}\sum_{m=1}^{2^{|A|}} |a_m\rangle$, i.e., the uniform superposition of all actions. But, noting $g(\boldsymbol{\pi}) = \sum_{m=1}^{2^{|A|}} \sqrt{\pi_{\boldsymbol{s}}(\boldsymbol{a_m})}$, we have that

$1 \leq g(\boldsymbol{\pi}) \leq \sqrt{2^{|A|}}$, and hence:

$$\frac{1}{\sqrt{2^{|A|}}} \quad \leq \quad |\langle \boldsymbol{\Psi}_{in} | \boldsymbol{\pi} \rangle| = \frac{g(\boldsymbol{\pi})}{\sqrt{2^{|A|}}} \quad \leq \quad 1 \tag{6.1}$$

which means that the complexity of the transformation $|\boldsymbol{\Psi}_{in}\rangle \rightarrow |\boldsymbol{\pi}\rangle$ is $\tilde{\mathcal{O}}(\sqrt{2^{|A|}}\sqrt{\delta^{-1}})$ in the worst case. Comparing this complexity with $\tilde{\mathcal{O}}(\delta^{-1})$, that we have classically, we would need $\delta < 2^{-|A|}$ to gain a speed-up, which would not even be very significant if δ is close to $2^{-|A|}$. This is unfortunately the case for our RBM-based model, as is shown in Appendix A.

6.2 Quantum preparation of thermal Gibbs states

Conveniently, the problem of deliberating on a Gibbs/Boltzmann policy (Eq. 2.12) given access to a model that encodes it, for instance an RBM or the ECM for RBM-based RPS agents, is equivalent to another physical problem greatly studied in literature during the past 20 years (at least), namely the preparation of *thermal Gibbs states* $\rho(\beta) = \frac{e^{-\beta H}}{Z(\beta)}$ of a Hamiltonian H, where $\beta = 1/T$ is the *inverse temperature* parameter and $Z(\beta) = \mathrm{Tr}(e^{-\beta H})$ is the *partition function*. This problem for *classical* Hamiltonians was first introduced by Lidar and Biham (1997) and extended to *quantum* Hamiltonian (that include classical ones) by Terhal and DiVincenzo (2000), that proposed heuristic methods but without deriving rigorous time complexity bounds. We give a comparison of currently known methods to prepare a thermal Gibbs state in Table 6.1.

In our case of interest, we are dealing with a classical Hamiltonian, namely the Hamiltonian having as eigenstates the possible actions of our RPS agent and as associated eigenvalues the free energy of its RBM (Eq. (4.5)) for those actions and a given state. We are interested in speeding-up a MC-based process, so the methods by Somma et al. (2008) and Wocjan and Abeyesinghe (2008) seem to be the most suited for our case. What is appealing in those methods is the polynomial dependency of their time complexity in the number of qubits n of the system (equal to the number of action units $|A|$ here), compared to the exponential dependency we had with our naïve approach in Section 6.1. These methods rely on *Quantum Simulated Annealing (SQA)*, that is, they derive from an initial MC P multiple MCs P_k having as stationary distribution $\boldsymbol{\pi}_k$, a Gibbs state of gradually decreasing temperature T_k, and slowly evolve an initial state that is easily preparable (the uniform superposition of all actions, corresponding to a Gibbs state at infinite temperature $|\boldsymbol{\pi}_0\rangle$) to the Gibbs state at the desired final temperature $|\boldsymbol{\pi}\rangle$.

Method	Hamiltonian	MC-based	Walk	QAA	Input	Output	Time Complexity
Lidar and Biham (1997)	Classical	No	X	No			Unclear
Terhal and DiVincenzo (2000)	Quantum	Yes	Classical	No	Any ρ	$\rho(\beta)$	Unclear
Somma et al. (2008)	Classical	Yes	Quantum	No	$\|+\rangle^{\otimes n}$	CETS[a]	$\tilde{\mathcal{O}}\left(\frac{n^3}{\sqrt{\delta}}\left(\frac{\|\|H\|\|}{\gamma}\right)^2\right)$[b]
Wocjan and Abeyesinghe (2008)	Classical	Yes	Quantum	Yes	$\|+\rangle^{\otimes n}$	CETS	$\tilde{\mathcal{O}}\left(\frac{n}{\sqrt{\delta}}\frac{\|\|H\|\|}{\gamma}\right)$
Poulin and Wocjan (2009)	Quantum	No	X	Yes	$\|\boldsymbol{\pi}_0\rangle$ [c]	CETS	$\tilde{\mathcal{O}}\left(\sqrt{\frac{2^n}{Z(\beta)}}\right)$
Chowdhury and Somma (2016)	Quantum	No	X	Yes	$\|\boldsymbol{\pi}_0\rangle$	CETS	$\tilde{\mathcal{O}}\left(\sqrt{\frac{2^n\beta}{Z(\beta)}}\right)$
Temme et al. (2011)	Quantum	Yes	Classical	No	Any ρ	$\rho(\beta)$	Unclear
Yung and Aspuru-Guzik (2012)	Quantum	Yes	Quantum	No	$\|\alpha_0\rangle$ [d]	CETS	$\tilde{\mathcal{O}}\left(\frac{\beta^2\langle H^2\rangle_0}{\sqrt{\delta}\epsilon}\right)$ [ef]
Ozols et al. (2013)	Quantum	Yes	Classical	Yes	Any ρ	$\rho(\beta)$	Unclear[g]

Table 6.1: A comparison of various quantum algorithms for Gibbs state preparation, inspired by (Yung and Aspuru-Guzik, 2012)

[a] One can alternatively prepare the coherent version of the thermal density matrix $\rho(\beta)$, the *Coherent Encoding of the Thermal State* $|\boldsymbol{\pi}\rangle = \sum_i \sqrt{\frac{e^{-\beta E_i}}{Z}}|\varphi_i\rangle$, with $H|\varphi\rangle = E_i|\varphi\rangle$

[b] Where $||H|| = \max_i |E_i|$ is the largest eigenvalue modulus of the Hamiltonian H and γ is the spectral gap of H, i.e., the difference between its two smallest eigenvalues

[c] Where $|\boldsymbol{\pi}_0\rangle = \frac{1}{\sqrt{2^n}}\sum_i^{2^n}|\varphi_i\rangle|\phi_i\rangle$ is any maximally entangled state with $|\varphi_i\rangle$ the eigenstates of the Hamiltonian H

[d] Where $|\alpha_0\rangle$ is a uniform superposition of the eigenstates of the Hamiltonian H, entangled one-to-one with the corresponding eigenstates of its time-reversal H^*

[e] Where $\langle H^2\rangle_0$ is an upper-bound on the energy fluctuation of the Hamiltonian H an ϵ the accuracy of the resulting state

[f] Provides a quadratic speed-up in the spectral gap δ of the underlying MC in (Temme et al., 2011)

[g] Provides a quadratic speed-up in a subroutine, namely the rejection sampling steps, of (Temme et al., 2011) through QAA

This is commonly called a *Zeno evolution*. In order to do that, both methods rely on Szegedy operators constructed out of the intermediary MCs. The one of Somma et al. (2008) uses the Szegedy operator $W(P_k)$ to construct a projective measurement on $|\boldsymbol{\pi}_k\rangle$, then the state is iteratively projected, with high probability, from $|\boldsymbol{\pi}_{k-1}\rangle$ to $|\boldsymbol{\pi}_k\rangle$. While the method of Wocjan and Abeyesinghe (2008) relies on reflectors around $|\boldsymbol{\pi}_k\rangle$ that we showed how to construct in Section 2.5.2, and iterative QAA from $|\boldsymbol{\pi}_{k-1}\rangle$ to $|\boldsymbol{\pi}_k\rangle$. The key difference between SQA and the naïve method presented in Section 6.1 is that the overlaps between $|\boldsymbol{\pi}_{k-1}\rangle$ and $|\boldsymbol{\pi}_k\rangle$ are made large by construction with allows to avoid an exponentially small overlap that would take over the total time complexity. Also what we get from the comparison of these two new methods is that the use of QAA instead of projective measurements allows to gain polynomial speed-ups in many parameters appearing in the time complexity of the QSA method.

6.3 In-context modification of the RBM

The advantage of the general Szegedy approach to prepare $|\boldsymbol{\pi}\rangle$ presented in Section 2.5 is that this latter doesn't necessarily need to be a Gibbs state for it to be applicable. For instance, this approach is also compatible with RBM-based RPS agents that have for MC a convex combination of the MC generated by Gibbs sampling on the RBM and another MC, as long as this keeps the convergence properties of the associated MCs, i.e., irreducibility, aperiodicity and time-reversibility. Note that the stationary distribution of the resulting MC won't be a convex combination of the parent ones in general, making the effect of this transformation non-trivial.

A possible transformation could introduce in every percept-specific sub-graph a new *fiducial* wildcard-action clip $\boxed{(\boldsymbol{a}_0, \#)}$ not present in the ECM depicted by Figure 5.1, and enforced through additional updates of the transitions of the ECM a high (but bounded, e.g., $\frac{1}{2} + \epsilon$) probability of selection in the stationary distribution $\boldsymbol{\pi}'$ of the resulting MC. This *in-context* transformation would result in a bounded large overlap between the states $|\boldsymbol{a}_0\rangle$ and $|\boldsymbol{\pi}'\rangle$, so a runtime complexity of the QAA (Section 2.5.3) from $|\boldsymbol{a}_0\rangle$ to $|\boldsymbol{\pi}'\rangle$ of the order $\tilde{\mathcal{O}}(\sqrt{\delta'^{-1}})$, where δ' is the spectral gap of the resulting MC. For an overlap close to $\frac{1}{2}$, resampling from the resulting $|\boldsymbol{\pi}'\rangle$ on average 2 times will be approximately the same as sampling from the desired distribution $\boldsymbol{\pi}$ coming from the RBM. More generally, this is done through so-called *rejection sampling*.
Note however that the resulting spectral gap will be different from the spectral gap of the MC

generated from the RBM, which makes it hard to quantify the speed-up of this method, unless if we know that our transformations induce $\delta' \geq \delta$.

6.4 Preparation from the Mean-Field approximation of the (R)BM

In this last section, we would like to briefly mention one method that doesn't rely on Markov Chains to prepare a coherent encoding of the policy of our RPS agents, but that still belongs to the Monte Carlo family. This method is inspired by (Wiebe et al., 2014), and essentially prepares the so-called *Mean-Field (MF) approximation* of the underlying distribution $P(\boldsymbol{v}, \boldsymbol{h})$ encoded in a BM (Eq. (2.16)). This variational approach finds the factorized distribution $Q(\boldsymbol{v}, \boldsymbol{h})$ that has minimal Kullback-Leibler divergence $KL(Q||P)$ with $P(\boldsymbol{v}, \boldsymbol{h})$. This is an approximation of $P(\boldsymbol{v}, \boldsymbol{h})$ in the sense that there exists a constant κ such that:

$$P(\boldsymbol{v}, \boldsymbol{h}) \leq \kappa Q(\boldsymbol{v}, \boldsymbol{h}) \quad \forall (\boldsymbol{v}, \boldsymbol{h}) \tag{6.2}$$

The fact that this distribution is factorized means that it is efficiently computable. We can then efficiently prepare the state $|\boldsymbol{\pi}_Q\rangle = \sum_{\boldsymbol{v},\boldsymbol{h}} \sqrt{Q(\boldsymbol{v}, \boldsymbol{h})}\, |v\rangle\, |h\rangle$ and use *quantum rejection sampling* (Ozols et al., 2013) to approximately resample this state to $|\boldsymbol{\pi}_P\rangle = \sum_{\boldsymbol{v},\boldsymbol{h}} \sqrt{P(\boldsymbol{v}, \boldsymbol{h})}\, |v\rangle\, |h\rangle$ with probability of success $\geq \frac{1}{\sqrt{\kappa}}$. Comparatively to Gibbs sampling, finding this MF distribution is computationally efficient as it requires to sample over only a polynomial number of configurations of the BM with respect to its number of units in order to evaluate it. Assuming that we know an upper bound κ' on κ, this gives a global time complexity of the algorithm of the order $\tilde{\mathcal{O}}(E\sqrt{\kappa'})$, where E is the number of edges in the BM, i.e., polynomial in its number of units.

Note that the advantage of this method is that it is compatible with deep architectures of BMs that allow to learn probability distributions with more complex correlations between visible units.

7 Conclusions and Prospects

We have presented a direct connection between MCMC-based methods for generalization in Reinforcement Learning and the model of Reflecting PS. This allowed us to define a new class of RPS agents constructed out of energy-based generative models such as RBMs, that we may now refer to in this last case as RBM-RPS agents. These RBM-RPS agents benefit from update rules that act globally on the transition probabilities of their ECMs, which allow them to act non-randomly in new unencountered situations, that is to say, generalize their learned behavior. Not only are they able to deal with environments having large state spaces, they are also able to deliberate on large numbers of actions as a result of the MC-based process they run for their deliberation. But because this process may be computationally intensive classically if one wants to avoid large approximation errors, we also presented quantization methods that allow to achieve a close-to-quadratic speed-up if one considers the RBM-RPS ECM without any additional transformations. Alternatively, when we consider in-context modifications of the agent's ECM, we may obtain a more promising speed-up. The restricted duration of this internship unfortunately didn't allow to fully explore this last option, which is why we leave the exact nature of the relevant in-context modifications as an open question for future work.

The specification of this new class of RPS agents opens the door to new ways of combining tabular RL methods with Function Approximation methods, e.g., RBM-RL, in a framework which allows advantageous quantization. The specification is general enough to consider, for instance, more complex energy functions allowing more levels of correlation between states and actions for our new RPS agents as long as we can keep tractable update rules. Also, one could additionally consider using the update rules of the h-values between precept and action clips of PS to update the energy function of these agents. This would hence allow to enjoy other features of PS agents such as *glow*, an extension to the basic update rule for bootstrapping delayed rewards to the h-values of previously experienced state-action pairs.

Bibliography

Adcock, J., Allen, E., Day, M., Frick, S., Hinchliff, J., Johnson, M., Morley-Short, S., Pallister, S., Price, A. and Stanisic, S. (2015), 'Advances in quantum machine learning', *arXiv preprint arXiv:1512.02900* .

Aldous, D. J. (1982), 'Some inequalities for reversible markov chains', *Journal of the London Mathematical Society* **2**(3), 564–576.

Amin, M. H., Andriyash, E., Rolfe, J., Kulchytskyy, B. and Melko, R. (2016), 'Quantum boltzmann machine', *arXiv preprint arXiv:1601.02036* .

Asadi, K. and Littman, M. L. (2016), 'An alternative softmax operator for reinforcement learning', *arXiv preprint arXiv:1612.05628* .

Bertoluzzo, F. and Corazza, M. (2014), Reinforcement learning for automated financial trading: Basics and applications, *in* 'Recent Advances of Neural Network Models and Applications', Springer, pp. 197–213.

Bertsekas, D. P. and Tsitsiklis, J. N. (1995), Neuro-dynamic programming: an overview, *in* 'Proceedings of the 34th IEEE Conference on Decision and Control', Vol. 1, IEEE Publ. Piscataway, NJ, pp. 560–564.

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N. and Lloyd, S. (2017), 'Quantum machine learning', *Nature* **549**(7671), 195.

Bishop, C. M. (2016), *Pattern Recognition and Machine Learning*, Springer-Verlag New York.

Brassard, G., Mosca, M. and Tapp, A. (2002), 'Quantum amplitude amplification and estimation'.

Brémaud, P. (2013), *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, Vol. 31, Springer Science & Business Media.

Briegel, H. J. and De las Cuevas, G. (2012), 'Projective simulation for artificial intelligence', *Scientific reports* **2**, 400.

Chowdhury, A. N. and Somma, R. D. (2016), 'Quantum algorithms for gibbs sampling and hitting-time estimation', *arXiv preprint arXiv:1603.02940* .

Ciliberto, C., Herbster, M., Ialongo, A. D., Pontil, M., Rocchetto, A., Severini, S. and Wossnig, L. (2018), 'Quantum machine learning: a classical perspective', *Proc. R. Soc. A* **474**(2209), 20170551.

Crawford, D., Levit, A., Ghadermarzy, N., Oberoi, J. S. and Ronagh, P. (2016), 'Reinforcement learning using quantum boltzmann machines', *arXiv preprint arXiv:1612.05695* .

Crites, R. H. and Barto, A. G. (1998), 'Elevator group control using multiple reinforcement learning agents', *Machine learning* **33**(2-3), 235–262.

Dunjko, V., Taylor, J. M. and Briegel, H. J. (2016), 'Quantum-enhanced machine learning', *Physical review letters* **117**(13), 130501.

Fischer, A. and Igel, C. (2011), 'Bounding the bias of contrastive divergence learning', *Neural Computation* **23**(3), 664–673.

Fischer, A. and Igel, C. (2012), An introduction to restricted boltzmann machines, *in* 'Iberoamerican Congress on Pattern Recognition', Springer, pp. 14–36.

Geman, S. and Geman, D. (1984), 'Stochastic relaxation, gibbs distributions, and the bayesian restoration of images', *IEEE Transactions on pattern analysis and machine intelligence* (6), 721–741.

Grover, L. K. (1996), A fast quantum mechanical algorithm for database search, *in* 'Proceedings of the twenty-eighth annual ACM symposium on Theory of computing', ACM, pp. 212–219.

Grover, L. K. (2005), 'Fixed-point quantum search', *Physical Review Letters* **95**(15), 150501.

Harris, R., Johnson, M., Lanting, T., Berkley, A., Johansson, J., Bunyk, P., Tolkacheva, E.,

Ladizinsky, E., Ladizinsky, N., Oh, T. et al. (2010), 'Experimental investigation of an eight-qubit unit cell in a superconducting optimization processor', *Physical Review B* **82**(2), 024511.

Harrow, A. W., Hassidim, A. and Lloyd, S. (2009), 'Quantum algorithm for linear systems of equations', *Physical review letters* **103**(15), 150502.

Hinton, G. E. (2012), A practical guide to training restricted boltzmann machines, *in* 'Neural networks: Tricks of the trade', Springer, pp. 599–619.

Katehakis, M. N. and Veinott Jr, A. F. (1987), 'The multi-armed bandit problem: decomposition and computation', *Mathematics of Operations Research* **12**(2), 262–268.

Kenny, P. (n.d.), 'Notes on boltzmann machines'.

Kitaev, A. Y. (1995), 'Quantum measurements and the abelian stabilizer problem', *arXiv preprint quant-ph/9511026* .

Le Roux, N. and Bengio, Y. (2008), 'Representational power of restricted boltzmann machines and deep belief networks', *Neural computation* **20**(6), 1631–1649.

Levit, A., Crawford, D., Ghadermarzy, N., Oberoi, J. S., Zahedinejad, E. and Ronagh, P. (2017), 'Free energy-based reinforcement learning using a quantum processor', *arXiv preprint arXiv:1706.00074* .

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J. and Jurafsky, D. (2016), 'Deep reinforcement learning for dialogue generation', *arXiv preprint arXiv:1606.01541* .

Li, Y. (2017), 'Deep reinforcement learning: An overview', *arXiv preprint arXiv:1701.07274* .

Lidar, D. A. and Biham, O. (1997), 'Simulating ising spin glasses on a quantum computer', *Physical Review E* **56**(3), 3661.

Lin, L.-J. (1993), Reinforcement learning for robots using neural networks, Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Long, P. M. and Servedio, R. (2010), Restricted boltzmann machines are hard to approximately evaluate or simulate, *in* 'Proceedings of the 27th International Conference on Machine Learning (ICML-10)', pp. 703–710.

Magniez, F., Nayak, A., Roland, J. and Santha, M. (2011), 'Search via quantum walk', *SIAM Journal on Computing* **40**(1), 142–164.

Melnikov, A. A., Makmal, A., Dunjko, V. and Briegel, H. J. (2017), 'Projective simulation with generalization', *Scientific reports* **7**(1), 14430.

Melnikov, A., Makmal, A. and Briegel, H. J. (2018), 'Benchmarking projective simulation in navigation problems', *arXiv preprint arXiv:1804.08607* .

Melnikov, A., Nautrup, H. P., Krenn, M., Dunjko, V., Tiersch, M., Zeilinger, A. and Briegel, H. J. (2018), 'Active learning machine learns to create new quantum experiments', *Proceedings of the National Academy of Sciences* p. 201714936.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015), 'Human-level control through deep reinforcement learning', *Nature* **518**(7540), 529.

Montenegro, R., Tetali, P. et al. (2006), 'Mathematical aspects of mixing times in markov chains', *Foundations and Trends in Theoretical Computer Science* **1**(3), 237–354.

Ozols, M., Roetteler, M. and Roland, J. (2013), 'Quantum rejection sampling', *ACM Transactions on Computation Theory (TOCT)* **5**(3), 11.

Paparo, G. D., Dunjko, V., Makmal, A., Martin-Delgado, M. A. and Briegel, H. J. (2014), 'Quantum speedup for active learning agents', *Physical Review X* **4**(3), 031002.

Poulin, D. and Wocjan, P. (2009), 'Sampling from the thermal quantum gibbs state and evaluating partition functions with a quantum computer', *Physical review letters* **103**(22), 220502.

Sallans, B. and Hinton, G. E. (2004), 'Reinforcement learning with factored states and actions', *Journal of Machine Learning Research* **5**(Aug), 1063–1088.

Schuld, M., Sinayskiy, I. and Petruccione, F. (2015), 'An introduction to quantum machine learning', *Contemporary Physics* **56**(2), 172–185.

Shani, G., Heckerman, D. and Brafman, R. I. (2005), 'An mdp-based recommender system', *Journal of Machine Learning Research* **6**(Sep), 1265–1295.

Shin, S. W., Smith, G., Smolin, J. A. and Vazirani, U. (2014), 'How" quantum" is the d-wave machine?', *arXiv preprint arXiv:1401.7087* .

Shor, P. W. (1999), 'Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer', *SIAM review* **41**(2), 303–332.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. (2017), 'Mastering the game of go without human knowledge', *Nature* **550**(7676), 354.

Somma, R., Boixo, S., Barnum, H. and Knill, E. (2008), 'Quantum simulations of classical annealing processes', *Physical review letters* **101**(13), 130504.

Sutton, R. S. (1988), 'Learning to predict by the methods of temporal differences', *Machine learning* **3**(1), 9–44.

Sutton, R. S., Barto, A. G. et al. (1998), *Reinforcement learning: An introduction*.

Szegedy, M. (2004), Quantum speed-up of markov chain based algorithms, *in* 'Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on', IEEE, pp. 32–41.

Temme, K., Osborne, T. J., Vollbrecht, K. G., Poulin, D. and Verstraete, F. (2011), 'Quantum metropolis sampling', *Nature* **471**(7336), 87.

Terhal, B. M. and DiVincenzo, D. P. (2000), 'Problem of equilibration and the computation of correlation functions on a quantum computer', *Physical Review A* **61**(2), 022301.

Tesauro, G. (1995), 'Temporal difference learning and td-gammon', *Communications of the ACM* **38**(3), 58–68.

Tiersch, M., Ganahl, E. and Briegel, H. J. (2015), 'Adaptive quantum computation in changing environments using projective simulation', *Scientific reports* **5**, 12874.

Wiebe, N., Kapoor, A. and Svore, K. M. (2014), 'Quantum deep learning', *arXiv preprint arXiv:1412.3489* .

Wocjan, P. and Abeyesinghe, A. (2008), 'Speedup via quantum sampling', *Physical Review A* **78**(4), 042336.

Yoder, T. J., Low, G. H. and Chuang, I. L. (2014), 'Fixed-point quantum search with an optimal number of queries', *Physical review letters* **113**(21), 210501.

Yung, M.-H. and Aspuru-Guzik, A. (2012), 'A quantum–quantum metropolis algorithm', *Proceedings of the National Academy of Sciences* **109**(3), 754–759.

Appendix A:
On the convergence rate of Gibbs sampling on an RBM

The result given in this appendix is just a restatement of a previous result given by Fischer and Igel (2011) and based on a proof on the convergence rate of Periodic Gibbs sampling given by Brémaud (2013). The aim is just to better restate that proof for the case of Gibbs sampling on an RBM and better justify the result given by Fischer and Igel (2011).

We know that the mixing time (the minimal number of mixing steps it takes to get ϵ close to the stationary distribution) of an irreducible, aperiodic and reversible finite space-state MC P with spectral decomposition $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_r > -1$ and stationary distribution $\boldsymbol{\pi}$ satisfies:

$$t_\epsilon^{mix} \leq \left\lceil \frac{1}{\delta} \log_2 \frac{1}{\epsilon \pi^*} \right\rceil \quad \forall \epsilon > 0 \tag{7.1}$$

where $\delta = 1 - \max_{i>0}|\lambda_i|$ is called the *spectral gap* of P and $\pi^* = \min_i(\boldsymbol{\pi}(c_i))$.

Let's call $\rho = \max_{i>0}|\lambda_i| = \max\{\lambda_1, |\lambda_r|\}$ the Second Largest Eigenvalue Modulus (or SLEM) of the transition matrix. It is clear that upper-bounding the SLEM is equivalent to lower-bounding the spectral gap of the MC, i.e., upper-bounding its mixing time. For that reason, when it is hard to look into the eigenstructure of the transition matrix, we look at a known upper-bound of the SLEM such as Dobrushin's Egodic coefficient δ', typically used to study non-homogeneous MCs (when the transition probabilities of the MC depend on time) but that is also well defined for homogeneous MCs:

$$\begin{aligned} \delta'(P) &:= \sup_{i,j} d_V(p_{.i}, p_{.j}) = \frac{1}{2} \sup_{i,j} \sum_k |p_{ki} - p_{kj}| \\ &= 1 - \inf_{i,j} \sum_k p_{ki} \wedge p_{kj} = 1 - \inf_{i,j} \sum_k \min\{p_{ki}, p_{kj}\} \end{aligned} \tag{7.2}$$

Then if N is the size of the state space,

$$\delta'(P) \leq 1 - N\left(\inf_{i,j} p_{ij}\right) \tag{7.3}$$

Because we are dealing with an (R)BM, we perform *Periodic* Gibbs sampling, meaning that we alternate between updating all the units of each layer (either hidden or visible) periodically. We then end up with a period 2 transition matrix, and because we aim to use Dobrushin's coefficient, we look at the associated aperiodic transition matrix (here, the square of the original matrix, with support only on the visible units). Moreover we know that in an RBM there are no intra-layer weights, which allows us to sample all the units of a same layer in parallel (we do *Block* Gibbs sampling). This translates into:

$$p_{ij} = \sum_h p(h|v^i)p(v^j|h) = \sum_h \left[\prod_{k=1}^{m} p(h_k|v) \prod_{k'=1}^{n} p(v_{k'}|h)\right] \tag{7.4}$$

where p_{ij} refers to the (*double speed*) transition probability between configurations v_i and v_j of the visible units.

Then we know that the probability distribution of a binary stochastic unit follows a Bernoulli distribution of probability:

$$\sigma\left(\sum_l w_{lk}s_l + b_k\right) = \frac{1}{1 + e^{-\left(\sum_l w_{lk}s_l + b_k\right)}} = \frac{e^{\left(\sum_l w_{lk}s_l + b_k\right)}}{1 + e^{\left(\sum_l w_{lk}s_l + b_k\right)}} \tag{7.5}$$

i.e., a sigmoid σ applied on its input (s_l are the values of the units connected to it).

We know focus on $p(h_k|v)$. Let's define:

$$\Delta_k^{(v)} = \max\left\{\sum_{l=1}^{n} w_{lk}v_l + b_k, -\sum_{l=1}^{n} w_{lk}v_l - b_k\right\} \tag{7.6}$$

Then the probabilities of the hidden unit k to be respectively active and inactive are:

$$\left\{\frac{e^{\left(\sum_l w_{lk}v_l + b_k\right)}}{1 + e^{\left(\sum_l w_{lk}v_l + b_k\right)}}, \frac{e^{-\left(\sum_l w_{lk}v_l + b_k\right)}}{1 + e^{-\left(\sum_l w_{lk}v_l + b_k\right)}}\right\} = \left\{\frac{e^{\left(\sum_l w_{lk}v_l + b_k\right)}}{1 + e^{\left(\sum_l w_{lk}v_l + b_k\right)}}, \frac{e^{-\left(\sum_l w_{lk}v_l + b_k\right) + \Delta_k^{(v)}}}{e^{\left(\Delta_k^{(v)}\right)} + e^{-\left(\sum_l w_{lk}v_l + b_k\right) + \Delta_k^{(v)}}}\right\} \tag{7.7}$$

Now suppose that $\Delta_k^{(v)} = \sum_l w_{lk}v_l + b_k$ (otherwise the symmetry of the probabilities gives the same result), then these probabilities become:

$$\left\{\frac{e^{\left(\Delta_k^{(v)}\right)}}{1 + e^{\left(\Delta_k^{(v)}\right)}}, \frac{1}{e^{\left(\Delta_k^{(v)}\right)} + 1}\right\} = \left\{\frac{e^{\left(\Delta_k^{(v)}\right)}}{1 + e^{\left(\Delta_k^{(v)}\right)}}, \frac{e^{-\left(\Delta_k^{(v)}\right)}}{1 + e^{-\left(\Delta_k^{(v)}\right)}}\right\} \tag{7.8}$$

Because it is the maximum (7.6), $\Delta_k^{(v)} \geq 0$, so $e^{\left(\Delta_k^{(v)}\right)} \geq 1$ and $e^{-\left(\Delta_k^{(v)}\right)} \leq 1$.
This gives us a lower bound on both probabilities:

$$p(h_k|v) \geq \frac{e^{-\left(\Delta_k^{(v)}\right)}}{2} \tag{7.9}$$

We want to get rid of the indexation by v of $\Delta_k^{(v)}$ and look at an upper bound for all v. Because v corresponds to all binary strings of length n, this gives us:

$$\Delta_k^{(v)} \leq \max\left\{\left|\sum_{l=1}^{n} I_{\{w_{lk}>0\}} w_{lk} + b_k\right|, \left|\sum_{l=1}^{n} I_{\{w_{lk}<0\}} w_{lk} + b_k\right|\right\} = \Delta_k \tag{7.10}$$

Then:

$$p(h_k|v) \geq \frac{e^{-(\Delta_k)}}{2} \tag{7.11}$$

A similar analysis could be done for $p(v_{k'}|h)$, that gives us:

$$p(v_{k'}|h) \geq \frac{e^{-(\Delta_{k'})}}{2} \tag{7.12}$$

where

$$\Delta_{k'} = \max\left\{\left|\sum_{l=1}^{m} I_{\{w_{lk'}>0\}} w_{lk'} + b_{k'}\right|, \left|\sum_{l=1}^{m} I_{\{w_{lk'}<0\}} w_{lk'} + b_{k'}\right|\right\} \tag{7.13}$$

Finally, we upper bound all values given in (7.10) and (7.14) by:

$$\Delta = \max\left\{\max_{k\in\{1,\ldots,m\}} \Delta_k, \max_{k'\in\{1,\ldots,n\}} \Delta_{k'}\right\} \tag{7.14}$$

and use (7.4) to bound p_{ij}:

$$\begin{aligned} p_{ij} &\geq \sum_h \left[\prod_{k=1}^{m} \frac{e^{-(\Delta)}}{2} \prod_{k'=1}^{n} \frac{e^{-(\Delta)}}{2}\right] \\ &= \sum_h \left[\frac{e^{-(m\Delta)}}{2^m} \frac{e^{-(n\Delta)}}{2^n}\right] \\ &= 2^m \left[\frac{e^{-(m+n)\Delta}}{2^{m+n}}\right] \\ &= \frac{e^{-(m+n)\Delta}}{2^n} \end{aligned} \tag{7.15}$$

so that, using (7.3):

$$\delta'(P) \leq 1 - 2^n \min_{i,j} p_{ij} \leq 1 - e^{-(m+n)\Delta} \tag{7.16}$$

A lower bound of the spectral gap δ is then $e^{-(m+n)\Delta}$.