Banditry and Quantum Agents

Contents

Abstract

The bandit problem is fundamental in sequential decision-making. In it, an agent must at each step decide between different arms. These are a fixed set of actions, each with unknown reward distributions. The goal is to maximise the cumulative rewards received. Real-world applications include clinical trials, recommender systems, portfolio optimisation and more. This thesis studies the problem and how it can be solved with modern technologies: quantum computing and neural-network-based reinforcement learning (RL). These are compared with classical bandit algorithms.

A primary focus is the quantum UCB (upper confidence bounds) algorithm. It uses a quantum Monte Carlo subroutine to estimate reward means more precisely than is possible classically. This requires rewards received as quantum-mechanical superpositions. Comparisons with classical UCB and the Bayesian method of Thompson sampling were conducted. In addition to new fixed instances, instances drawn from various priors (re Bayesian regret) were used, which is rare in the literature and new for the quantum algorithm. Results indicate that quantum UCB can outperform classical methods, but only for long time horizons and highly correlated arms. Further research is needed to determine which applications permit the use of the quantum algorithm and which are difficult enough to warrant its use.

Common RL, not specifically designed for the bandit problem, is tested on it. It is observed that the algorithms struggle to solve the tested instances. This is presumably due to poor state representations or rewards. While future work could find better setups for these algorithms, bandit-specific algorithms will likely remain the best choice for the problem; with the effort that may be required to match the specific algorithms, the benefit of using a general tool is lost.

To conclude, the thesis finds quantum computing useful for the bandit problem, while general RL appears less advantageous. More studies are needed to determine under which conditions quantum algorithms provide benefits and to test the algorithm on different reward distributions and arm numbers. Future work can consider different bandit variants or attempt to find a better quantum bandit algorithm, building something inherently quantum instead of extending classical algorithms.

Sammendrag

Bandittproblemet er et grunnleggende problem i sekvensiell beslutningstaging. I det må en spiller velge mellom ulike armer i hver runde. Armene er en bestemt mengde valg spilleren kan ta, der hvert valg har en ukjent premiefordeling. Målet er derfra å maksimere samtlige mottatte premier. Faktiske bruksområder inkluderer kliniske studier, anbefalingssystemer, porteføljeoptimering og mer. Her studeres problemet og hvordan det kan løses med moderne teknologier, nemlig kvantedatamaskiner og forsterkende læring (RL) med nevrale nettverk. Disse sammenlignes med klassiske, bandittspesifikke algoritmer.

Kvante-UCB (øvre konfidensintervallgrenser) er en sentral algoritme i oppgaven. I den brukes en kvante-Monte-Carlo-prosedyre for å estimere forventet vinning mer nøyaktig enn det man kan klassisk, hvilket krever premier mottatt som kvantemekaniske superposisjonstilstander. Sammenligninger med klassisk UCB og den bayesiske metoden Thompson-trekking ble gjort. I tillegg til nye fastsatte instanser, er også trukne instanser fra flere priorfordelinger (mht. bayesisk tap) brukt, hvilket er sjeldent i litteraturen og nytt for kvantealgoritmen. Resultatene viser at kvante-UCB kan slå klassiske metoder, men kun for vanskelige instanser. Kvantefordeler krever mange runder og sterkt korrelerte armer. Videre forskning kreves for å finne ut av hvilke anvendelser som tillater bruk av kvantealgoritmen og som er vanskelige nok til å fordre dens bruk.

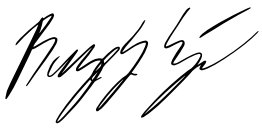
Vanlig RL, som ikke er laget spesifikt for bandittproblemet, testes på det. Det ses at de sliter med å løse de prøvde instansene. Dette skyldes antagelig dårlige tilstandsrepresentasjoner eller premiefunksjoner. Selv om fremtidig arbeid kan finne bedre innstillinger for algoritmene, er nok de bandittspesifikke ennå best egnet; ved å bruke krefter på å anpasse algoritmene, mistes fordelen med å bruke generelle verktøy.

Arbeidet viser at kvantedatamaskiner kan være nyttige for bandittproblemet, mens generell RL tilsynelatende ikke kan det. Mer forskning må til for å finne ut av når kvantealgoritmer er fordelaktige og for å teste algoritmen på andre premiefordelinger og antall armer. Fremtidige studier kan se på andre bandittvarianter eller forsøke å konstruere en bedre kvantebandittalgoritme ved å lage noe helt kvantemekanisk i stedet for å bygge videre på klassiske algoritmer.

Preface

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Note that the theory on machine learning and quantum computing, in addition to parts of the introduction, are based the preparatory specialisation project report [1], carried out in the previous semester, the autumn of 2022. Sections 3.2, 4.1, 4.2, 4.4, 6.1 and 6.2 are reused thence with only minor modifications to the text.



Boye Gravningen Sjo
Trondhjem, Norge
12th of June 2023

Chapter 1

Introduction

In the world of decision-making, the multi-armed bandit problem stands as a persistent and fascinating challenge. At its core, this problem asks: how should choices be optimised when presented with a set of options, each with its own uncertain reward? The metaphor of the bandit refers to the idea that each option is like a slot machine, where it must be decided how much to invest in each one in order to maximise winnings over time. One might imagine that the best strategy would be to invest heavily in the option with the highest expected reward. But what if that option is not immediately clear? Or what if the rewards are highly variable, such that it is difficult to predict which option will ultimately be the most lucrative? How to balance the competing demands of exploration and exploitation? These are the kinds of questions that make the multi-armed bandit problem so intriguing.

Historically, the problem has arisen in a variety of contexts. In the world of clinical trials, researchers may want to test multiple treatments simultaneously in order to find the most effective one. Despite test conclusions being statistically significant, there is always a chance of a false positive, which is how bandit algorithms can provide better long-term results than the classical approach of using a fixed sample size. In the realm of online advertising, companies must decide how to allocate their budget across a range of marketing channels, each with its own level of risk and potential payoff. And in the world of finance, investors must decide how to allocate their resources across different stocks or other assets, each with its own level of volatility and expected return. As computers automate ever more of people's lives, using the bandit problem to model the decision-making process and teach computers how to learn becomes increasingly relevant.

Some of the many real-world settings in which the multi-armed bandit problem is applicable are listed in table 1.1. Despite being a simple problem, its countless variants and applications make it not only a useful tool for a plethora of real-world problems, but also a fascinating theoretical challenge. Netflix uses bandit theory to recommend movies [2], Amazon for its website

Table 1.1: Some applications of the multi-armed bandit problem [9, 2, 3, 4, 5]. The arms are the set of the different actions an agent can take. The reward is the probabilistic outcome of the action, which the agent tries to maximise over time.

Application	Arms	Reward
Medical trials	Drugs	Patient health
Online advertising	Ad placements	No. of clicks
Website design	Layouts &c.	No. of clicks
Recommender systems	Items	No. of clicks
Dynamic pricing	Prices	Profit
Networking	Routes	Ping
Lossy compression	Settings	Quality preserved
Tasking employees	Employees	Response time
Finance	Investments	Profit
Machine learning	Hyperparameters	Performance
Anomaly detection	Thresholds	No. of anomalies

layout [3], Facebook for video compression [4] and Doordash to identify responsive deliverymen [5]. The problem and its variations are still being studied with several results in what follows being recent. Bandits theory has also been applied to other problems, such as the problem of choosing the best hyperparameters or models, where algorithms discussed in this thesis have shown success [6, 7, 8].

Despite the diversity of contexts, the essential challenge remains the same: how to balance the competing demands of exploration and exploitation. On the one hand, one would want to gather as much information as possible about the potential rewards of each option, so that an informed decision can be made. On the other hand, resources should be spent on the most promising option as quickly as possible, so that the returns are maximised. Having to deal with both random rewards and random decisions based on said rewards makes the problem virtually impossible to solve optimally.

> 'The problem is a classic one; it was formulated during the war, and efforts to solve it so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany, as the ultimate instrument of intellectual sabotage' [10].

Luckily, since then, algorithms have been developed that can solve the problem with certain optimality guarantees.

Machine learning and reinforcement learning in particular have in recent years become a popular tools for solving problems of all kinds. Their mathematical formulations and implementations are much more intricate than the bandit problem, but the basic idea is the same: to learn from experience and make decisions that lead to the best possible outcome. Modern methods based on neural networks have reached superhuman performance in many domains, such as Go and modern e-sport video games. These problems are too hard to solve anywhere near optimally, neither for humans nor for computers, so the bar for success is lower than for the bandit problem. Still, one might expect these modern, advanced and general tools to be able to do well on the classic bandit problem.

The field of quantum computing has emerged as a promising new area of research with the potential to solve problems intractable for classical hardware. Their first conception goes back to the early 80s, often being accredited to Richard Feynman's 1982 paper [11], with the goal of simulating difficult quantum mechanical problems. It was however only really with the discovery of Shor's algorithm in 1994 [12] that the potential of quantum computers gained widespread attention. The threat of encrypted data being exposed, not only in the future, but also current data through so-called 'store now, decrypt later' attacks (for which encrypted data is indeed already collected), is a major driving force behind the development of quantum computers and government involvement [13, 14]. As it is assumed that quantum computers are truly able to solve problems intractable for classical computers, the search for what other problems they can solve has been a major focus of research.

How actually to construct a quantum computer is still an open question. There are several types of hardware being developed and researched, such as superconducting circuits used by IBM, Google among others, trapped ions used by IonQ and Honeywell, photonic quantum computers developed by Xanadu and Psi Quantum in addition to many other types. Although some of these have already claimed quantum supremacy, that is they have solved a problem believed to be intractable for even the best classical (super-) computers, supremacy has only been shown in very particular settings with no practical use. Common for all current approaches are difficulties with noise and decoherence. Theoretically, with computers of great enough scale, errors can be corrected, but in the near future, the systematic errors of quantum computers will be a limiting factor and something to be taken into account when designing algorithms.

When it comes to bandit problems and reinforcement learning in general, quantum computing may offer a number of potential benefits. The quantum upper confidence bound algorithm introduced in [15], a main focus of this thesis, is one such example. In it, a quantum Monte Carlo method is used to produce better estimates of the bandit rewards than classical methods can ever do. Thereby it promises to achieve asymptotically better performance than any non-quantum algorithm. While quantum machine learning has received its fair share of attention, quantum reinforcement learning is still a relatively unexplored field. Still, some algorithms have been proposed, and some indications of quantum advantage have been noticed.

This thesis is structured as follows. Chapter 2 introduces the multi-armed bandit problem formally and presents what theoretical guarantees can be made about the performance of algorithms. Several classical algorithms are then presented, including the classical upper confidence bound algorithm, which is the basis for the quantum algorithm, to be discussed later, and Thompson sampling, a Bayesian method that tends to perform best in practice. Thereafter, the more general field of reinforcement learning and its mathematical formulation as Markov decision processes are introduced in chapter 3, along with several state-of-the-art algorithms based on deep learning and artificial neural networks. Next, in chapter 4, quantum computing is explained, including the quantum circuit model and several quantum algorithms, such as Grover's and quantum Monte Carlo. In chapter 5, quantum computing is applied to the bandit problem, leading to the quantum upper confidence bound algorithm, whose theoretically supreme performance is proved. Chapter 6 describes more general methods for quantum reinforcement learning and quantum agents. The quantum upper confidence algorithm is implemented and tested against classical methods on a suite of bandit problems in chapter 7, wherein also general agents of both classical and quantum kinds are tried on the bandit problem. Finally, chapter 8 concludes the thesis and discusses future work.

Chapter 2

Stochastic bandits and classical strategies

The multi-armed bandit (MAB) problem is a classic problem in reinforcement learning and probability theory. It poses a simple yet challenging problem, where a player must sequentially choose between a number of arms (distributions) receiving random rewards, of which the average is unknown, trying to maximise the cumulative reward. This poses a constant struggle between exploration and exploitation, where exploration is the process of trying out new arms, and exploitation is the process of maximising the reward by pulling what seems to be the best arm.

Although the bandit term was not coined before 1952 [16], its study actually dates back to 1933 [17]. The problem of choosing between two treatments for patients was considered: to what degree should the most successful treatment be used versus testing the other to ensure that the truly best is indeed used? What strategy is most likely to maximise the number of patients treated with the best treatment in the long run?

This chapter and thesis in total will focus on the stochastic bandits, where the rewards are drawn independently from a fixed distribution for each arm. The notation will mostly follow the textbook [18], to which the interested reader is referred for more details on the bandit problem, its variants, applications and history.

2.1 Problem definition

In the multi-armed bandit problem, the agent (the player or decision-maker) has knowledge of the set of available actions $\mathcal{A} = \{1, \dots, k\}$, but not the reward distributions $\nu = \{P_a : a \in \mathcal{A}\}$. For each time step $t = 1, \dots, T$, the agent selects an arm $a_t \in \mathcal{A}$ and receives a reward $X_t \sim P_{a_t}$, independent of previous rewards and actions. The time horizon T is usually assumed finite and given, but for many applications, knowledge of it is unreasonable,

motivating algorithms that can work with infinite time horizons, known as being anytime. Nonetheless, it will be assumed greater than k, such that all arms may be pulled at least once, ensuring that empirical means and whatnot are well-defined. The goal is for the agent to maximise its cumulative rewards.

2.1.1 Further assumptions

With no assumptions on the reward distributions, analysis is difficult. It is therefore common to make some assumptions on the reward distributions, defining bandit classes

$$\mathcal{E} = \{\nu = \{P_a : a \in \mathcal{A}\} : P_a \text{ satisfies some property } \forall a \in \mathcal{A}\}. \tag{2.1}$$

Some common bandit classes are listed in table 2.1. Note that some classes are parametric, like Bernoulli and Gaussian bandits, while others are non-parametric, such as the sub-Gaussian and bounded value bandits. Only cases void of any inter-arm structure are considered in this thesis; knowledge of the mean of one arm should never be useful to infer the mean of another.

It is assumed to be only one optimal arm, which is the arm with the highest mean reward. This is rarely a crucial property, but still useful to avoid unnecessary complications and verbosity in the arguments and proofs that follow.

2.1.2 Policies

When interacting with the environment, the agent must select an action at each time step. Hence, a history,

$$\mathcal{D} = \{A_1, X_1, \ldots, A_t, X_t\}, \tag{2.2}$$

is formed, where A_t is the action taken at time t and X_t is the reward received according to the distribution P_{A_t}. The policy $\pi = (\pi_t)_{t=1}^T$ is a sequence of probability distributions over the set of actions $\mathcal{A}$. At each time step t, the agent selects an action $a_t \sim \pi_t \mid \mathcal{D}$. To define an algorithm, a policy

$$\pi_t(a \mid A_1, X_1, \ldots, A_{t-1}, X_{t-1}) = \pi_t(a \mid \mathcal{D}) \tag{2.3}$$

is needed for each time step t, from which samples can be drawn. Some algorithms are deterministic, in which case the policy can be defined as a function of the history $\mathcal{D}$, rather than a probability distribution. Note that any probabilistic policy can be represented as a deterministic policy by defining the action to be the most likely action according to the policy.

Table 2.1: Common bandit classes for the unstructured, stochastic multi-armed bandit problem. The symbol is used for references in the text. The defining properties of the classes are generally the specific distribution of the rewards, but may also be general properties that the distributions must satisfy, giving rise to non-parametric classes.

Class	Symbol	Definition
Bernoulli	$\mathcal{E}^k_{\mathrm{B}}$	$X_a \sim \mathrm{B}(\mu_a)$
Gaussian, unit variance	$\mathcal{E}^k_{\mathrm{N}}(1)$	$X_a \sim \mathrm{N}(\mu_a, 1)$
Gaussian, known variance	$\mathcal{E}^k_{\mathrm{N}}(\sigma^2)$	$X_a \sim \mathrm{N}(\mu_a, \sigma^2)$
Gaussian, unknown variance	$\mathcal{E}^k_{\mathrm{N}}$	$X_a \sim \mathrm{N}(\mu_a, \sigma_a^2)$
Sub-Gaussian	$\mathcal{E}^k_{\mathrm{SG}}(\sigma^2)$	$P(\lvert X_a\rvert \geq \epsilon) \leq 2e^{-\epsilon^2/\sigma^2}$
Bounded value	$\mathcal{E}^k_{[0,1]}$	$X_a \in [0,1]$
Bounded maximum	$\mathcal{E}^k_{(-\infty,b]}$	$X_a \leq b$
Bounded variance	$\mathcal{E}^k_{\mathrm{Var}}(\sigma^2)$	$\mathrm{Var}(X_a) \leq \sigma^2$
Bounded kurtosis	$\mathcal{E}^k_{\mathrm{Kurt}}(\kappa)$	$\mathrm{Kurt}(X_a) \leq \kappa$

2.1.3 Regret

For the analysis of algorithm performance, the regret is most commonly used. Given a bandit instance ν and a policy π, at round T, the regret is defined as

$$R_T(\pi, \nu) = \mathbb{E}_{\pi,\nu}\left[\sum_{t=1}^{T} \mu^* - X_t\right], \tag{2.4}$$

where μ^* is the mean of the optimal arm and the expectation is taken over both the reward distributions and the potentially probabilistic policy. Often the dependences on particular instances and policies are irrelevant or clear from the context and are therefore omitted.

The usage of regret over the sum of rewards provides several advantages. Firstly, it serves as a normalised measure of performance, wherein perfect performance is achieved when the regret is zero. Secondly, it permits the usage of asymptotic notation for the analysis of algorithm performance as a function of the time horizon T. It will be seen that an uninformed policy will generate $O(T)$, while more clever solutions satisfy sublinear regret bounds. Finally, considering expectation rather than the stochastic sum of rewards makes the optimisation problem well-defined without having to introduce any utility measure or other assumptions.

It may in some cases be more convenient to express the regret in terms of the number of times each action has been selected, irrespective of time. Letting T_a be the number of times action a has been selected up to time T, and using the finitude of $\mathcal{A}$ and linearity of expectations, the regret can be rewritten as

$$R_T = \sum_{a \in \mathcal{A}} \Delta_a \mathbb{E}[T_a], \tag{2.5}$$

where $\Delta_a = \mu^* - \mu_a$ is what is known as the suboptimality gap of action a. The dependency on the policy and instance is here omitted for clarity, and it will often be so when the context is clear

2.1.4 Variants

Best-arm identification

An alternative problem is to find the best arm with as few turns as possible. In this version, a δ is given, and the goal is to find the best arm with probability at least $1 - \delta$. The metric here is how the turns needed grows as a function of δ. Unlike regret minimisation, exploitation is less of a concern, but much theory can be transferred from the regret minimisation problem. Though there is no direct benefit from exploitation, as there is in regret optimisation where rewards are collected, it is still desirable mainly to pull good arms, as these will need more consideration to be distinguished from the optimal. Worse arms should new few pulls to be identified as suboptimal.

Bandit generalisations

The multi-armed bandit problem has numerous generalisations, including the non-stationary multi-armed bandit where the underlying reward distributions change over time, presenting a challenging environment for traditional algorithms developed for the standard, stationary multi-armed bandit problem. In this variant, agents must continuously explore and adapt to the changing environment. Other cases give the agent more info, such as letting it know what the rewards for all arms, were they pulled instead, would have been. Another area of study is the contextual multi-armed bandit problem, in which contextual information must be incorporated into the decision-making process for arm selection, adding a layer of complexity to the standard multi-armed bandit problem, particularly useful for recommender systems, where the context is the user and their preferences. Moreover, the adversarial multi-armed bandit problem represents

a significant departure from the standard, stochastic multi-armed bandit problem, where rewards are chosen by an adversary instead of following a stationary distribution. The infinite-armed variants, where the arm space is infinite but constrained by for example linearity, also have practical applications. While beyond the scope of this report, these generalisations of the multi-armed bandit problem represent important areas of study and much of the theory developed for the standard, stochastic multi-armed bandit problem can be extended to these problems as well [19, 18].

2.2 Optimality

In the realm of multi-armed banditry, expressing optimality is fraught with difficulties. Any precise formulation is contingent upon not only the assumptions made, but also the particular instance, namely actual means and distributions overall. Usually, in the literature, no hyper-distributions are placed on the bandit classes $\mathcal{E}$, so no average regret over all instances can be defined. Lower bounds resort then to either determine what a reasonable policy can achieve on a given instance, or to describe its worst performance over all instances in the class. However, by defining some weighted average over all instances, an exact performance measure can be defined, such that optimality is well-defined. This is rarely done in the literature, but is included here to provide a more complete and practical measure of performance.

2.2.1 Instance-dependent lower bound

In order to meaningfully define a lower bound for a given instance, it is imperative to assume a reasonable algorithm. Otherwise, trivial policies, such as always pulling the first arm, could achieve zero regrets, hindering any meaningful comparison. The common assumption is that the algorithm is asymptotically consistent in some class $\mathcal{E}$, which by definition means that for all inferior arms and all $\eta \in (0, 1)$, it holds that[1]

$$\mathbb{E}[T_a] = o(T^\eta), \tag{2.6}$$

for all instances $\nu \in \mathcal{E}$.

For asymptotically consistent and bandit classes with reward distributions parametrised by only one parameter, the Lai-Robbins bound [20] holds. It

[1]The small-o is used for asymptotic upper bounds and is similar to big-O, but with a strict inequality. E.g., $x^2 = o(x^3)$, but $x^2 \neq o(x^2)$.

states that

$$\liminf_{T\to\infty} \frac{\mathbb{E}[T_a]}{\ln T} \geq \frac{1}{D(P_a \parallel P^*)}, \tag{2.7}$$

where P_a is the reward distribution of arm a, P^* that of the optimal distribution and $D(\cdot \parallel \cdot)$ the Kullback-Leibler divergence. The Kullback-Leibler divergence is a measure of the difference between two probability distributions over the same space $\mathcal{X}$, defined as

$$D(P \parallel Q) = \sum_{x\in\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \tag{2.8}$$

for discrete distributions and

$$D(P \parallel Q) = \int_{\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}\, dx \tag{2.9}$$

in the continuous case. For more general bandit classes, given finite means, the Kullback-Leibler in the denominator of eq. (2.10) is instead taken to the distribution in that class which is closest to P_a and with mean equal to the mean of P^*,

$$\liminf_{T\to\infty} \frac{\mathbb{E}[T_a]}{\ln T} \geq \frac{1}{d(P_a, \mu^*, \mathcal{E})}, \tag{2.10}$$

where

$$d(P_a, \mu^*, \mathcal{E}) = \inf_{P\in\nu\in\mathcal{E} \text{ for some } \nu} \{D(P_a \parallel P) : \mathbb{E}[P_a] > \mu^*\}. \tag{2.11}$$

From eq. (2.5), it follows that

$$\liminf_{T\to\infty} \frac{R_T}{\ln T} \geq \sum_{a\in\mathcal{A}} \frac{\Delta_a}{d(P_a, \mu^*, \mathcal{E})}. \tag{2.12}$$

Algorithms satisfying eq. (2.12) with equality are said to be asymptotically optimal.

The Lai-Robbins bound is instance-dependent through its dependence on the Kullback-Leibler divergences. Its dependence on the divergences, which are not in practice known, makes it inapplicable to real-world problems, and as reward distributions approach the optimal distribution, the bound diverges. It is still a useful tool for theoretical analysis and simulated experiments.

2.2.2 Instance-independent lower bound

A more general lower bound is the minimax regret. Given some problem class $\mathcal{E}$, it is defined as

$$\inf_{\pi} \sup_{\nu \in \mathcal{E}} R(\nu, \pi), \tag{2.13}$$

where π is the policy, and ν is the problem instance. The minimax regret is a lower bound on the whole class rather than one particular instance; algorithms may achieve better in some or even most instances, but no algorithm can do better than the minimax regret for all. In [21], it is proven that for all algorithms, given a fixed horizon T and number of arms K, there is at least one problem instance such that[2]

$$R_T = \Omega(\sqrt{KT}). \tag{2.14}$$

Such a bound is independent of the reward distributions, and as such, it is applicable in practice, but it may be overly robust. It can be preferable to sacrifice performance on some instances to gain performance on others. Minimax regret optimality implies a flat risk profile, while in practice, performance may be desired to correlate with instance difficulty. Surprisingly, minimax optimality does not negate instance optimality, and recent algorithms have been shown to achieve both [22, 23].

2.2.3 Bayesian optimality

If a prior distribution were to be placed on the reward distributions, a notion of average or Bayesian regret can be defined. Alternatively, this can be thought of as having some weighing of the particular instances and averaging over them. Following the notation of [18], the Bayesian interpretation is what is here presented.

For some bandit class $\mathcal{E}$, one simply includes the prior distribution of the reward distributions in the expectation taken to define the Bayesian regret,

$$\mathrm{BR}_T(Q, \pi) = \mathbb{E}_{\pi,\nu,Q}\left[\sum_{a \in \mathcal{A}} \Delta_a T_a\right] = \mathbb{E}_{\nu \sim Q}\left[R_T(\nu, \pi)\right], \tag{2.15}$$

where Q is the prior distribution over $\mathcal{E}$. Is it trivially observed that the Bayesian regret is bounded above by the minimax regret. However, it can

[2]The Ω-notation denotes asymptotic lower bounds, effectively the opposite of big-O. E.g., $x^2 + x = \Omega(x^2) = \Omega(x) \neq \Omega(x^3)$.

be proven that there exist priors such that the Bayesian regret is bounded below by some $\Omega(\sqrt{KT})$ [18], such that[3]

$$\sup_{Q} \inf_{\pi} \mathrm{BR}_T(Q, \pi) = \Theta(\sqrt{KT}). \tag{2.16}$$

The Bayesian regret can be a useful tool for designing algorithms, as knowledge inlaid in the prior can be used to guide the algorithm, but it may lead to less robust designs than policies devised by the above discussed methods. Furthermore, calculating the optimal policy is generally intractable for any horizon of size and reward distributions more complicated than Bernoulli [18]. In simpler cases, dynamic programming can be fruitful. With Bernoulli bandits and clever implementations, optimal strategies can indeed be found [24]. For longer horizons, discounting, namely reducing the weight of future rewards, can be used to make the problem tractable or at least easier to approximate well. Still, direct optimisation is restrained to the regimes of small horizons and simple reward distributions, and for more complicated problems, heuristic methods are required.

2.3 Classical solutions

2.3.1 Explore-only

Pure exploration is obviously a suboptimal strategy, but it is a good baseline against which to compare. It can be implemented by selecting an arm uniformly or in order, performing poorly either way. A random arm-selection procedure is described by algorithm 2.1.

Algorithm 2.1: Random arm selection

```
1 Sample a from 𝒜 uniformly
2 return a
```

It is easy to understand that the regret is

$$R_T = T\left(\mu^* - \frac{1}{k}\sum_{i=1}^{k}\mu_i\right), \tag{2.17}$$

[3]The Θ-notation describes both upper and lower asymptotic bounds, effectively the intersection of big-O and Ω (q.v. footnote 2). E.g., $x^2 + x$ is $\Theta(x^2)$, but not $\Theta(x)$ or $\Theta(x^3)$.

Table 2.2: Strategies for the unstructured, stochastic multi-armed bandit problem. The minimax regret is the instance-independent upper bound, while the regret is the dependent. Note that different UCB variants with different regret bounds exist, with UCB1 only achieving the listed bounds for rewards in $[0, 1]$. Also, note that the Thompson sampling properties listed assume Bernoulli rewards and a uniform prior.

Strategy	Minimax regret	Regret	Tuning
Random	$O(T)$	$O(T)$	NA
Greedy	$O(T)$	$O(T)$	NA
Epsilon-greedy	$O(T)$	$O(T)$	Difficult
Epsilon-decay	$O(T)$	$O(\log T)$	Difficult
UCB1	$O(\sqrt{T \log T})$	$O(\log T)$	Barely
Thompson	$O(\sqrt{T})$	$O(\log T)$	Priors

which is necessarily linear in T. This motivates the search for an algorithm with sublinear regret. It should be clear that a linear regret is actually the worst possible.

2.3.2 Greedy

Tending away from pure exploration to exploitation, a greedy algorithm will always select the arm with the highest empirical mean. Here, all arms are pulled an initial $m \geq 1$ times, after which estimated means are used to select the best arm. Then, the arm with the highest empirical mean is selected for all remaining turns. The arm-selection procedure is listed in algorithm 2.2, where $\hat{\mu}_a$ is the estimated mean of arm a.

Algorithm 2.2: Greedy arm selection

```
if t ≤ mk then
    return (t mod k) + 1
else
    return argmax_{a∈A} μ̂_a
```

With greedy selection, the expected regret is clearly still linear in the horizon, as there is a non-zero probability of selecting the wrong arm when taking the greedy action. Still, there is a chance of achieving zero regret and the constant factor is reduced compared to pure exploration selection.

To improve hereupon, it is necessary to occasionally explore other arms, which leads to the epsilon-greedy algorithm.

2.3.3 Epsilon-greedy

The problem with the greedy algorithm is that it may be unlucky and not discover the best arm in the initial exploration phase. To mitigate this, the epsilon-greedy algorithm can be used. In this algorithm, the arm that is presumed to be best is pulled with probability $1 - \epsilon$, while in the other ϵ proportion of the turns, the arm is selected uniformly at random. This ensures convergence to correct exploitation as the horizon increases, and it will generally reduce the regret.

Still, with a constant ϵ, a constant proportion of the turns will be spent exploring, keeping the regret necessarily linear in the horizon. Choosing ϵ is a trade-off between exploration and exploitation and can significantly affect the regret.

Algorithm 2.3: Epsilon-greedy arm selection

```
if t ≤ mk then
    return (t mod k) + 1
else
    Sample u from [0, 1) uniformly
    if u < ε then
        Sample a from 𝒜 uniformly
        return a
    else
        return argmax_{a∈𝒜} μ̂_a
```

Epsilon-decay

To remedy the linear term in the regret, modifications to the epsilon-greedy algorithm have been proposed wherein ϵ is a function of the current time step t. Specifically, in order to achieve sublinear regret, it is necessary to decay ϵ towards zero. Decreasing ϵ over time should make intuitive sense; exploration is more crucial in the early stages of the algorithm, whereas exploitation is more important when the agent has more reliable estimates of the reward means. For example, one successful strategy is to set $\epsilon \propto 1/t$, which has been shown to achieve logarithmic instance-dependent regret [25]. It is worth noting, however, that the optimal decay rate depends on the

specific instance, and achieving logarithmic regret can be challenging in practice [26].

2.3.4 Upper confidence bounds

The upper confidence bound (UCB) algorithm is a family of more sophisticated algorithms based on estimating upper confidence bounds for the means of all arms, and they are a common baseline for more advanced algorithms tackling more complicated bandit variants [27]. The arm whose upper confidence bound is highest is always pulled, a principle known as 'optimism in the face of uncertainty'. This should make sense, as if the wrong arm appears best, it will be pulled more often and the empirical mean will be corrected, while the true best arm with its larger bound will eventually become highest and so pulled. When exploiting the actual best arm, the agent can trust it to be the best, as the confidence bound will remain above those of all the other arms. In addition, by increasing the confidence as the number of pulls increases, getting stuck in suboptimality is avoided.

UCB1

Assuming rewards in $[0, 1]$ and using Hoeffding's inequality, it follows that

$$p = P(\mu_a > \hat{\mu}_a + \mathrm{UCB}_a) \leq \exp\left(-2T_a\mathrm{UCB}_a^2\right), \tag{2.18}$$

where UCB_a is the upper confidence bound for arm a and T_a is the number of times arm a has been pulled. Solving for UCB_a yields

$$\mathrm{UCB}_a = \sqrt{\frac{-\ln p}{2T_a}}. \tag{2.19}$$

Letting $p(t) = t^{-4}$ gives

$$\mathrm{UCB}_a = \sqrt{\frac{2\ln t}{T_a}}, \tag{2.20}$$

which is a common choice for the upper confidence bound, leading to the UCB1-algorithm. In [25], it is shown that UCB1 achieves

$$R_T \leq \left(8 \sum_{a \in \mathcal{A} \setminus a^*} \frac{\log T}{\Delta_a}\right) + \left(1 + \frac{\pi^2}{3}\right) \sum_{a \in \mathcal{A}} \Delta_a, \tag{2.21}$$

which misses instance-dependent optimality only by some constant factor. Nor is UCB1 truly minimax-optimal, as its instance-independent upper bound is $O(\sqrt{kT \log T})$ [26].

Algorithm 2.4: UCB arm selection

```
if t ≤ k then
    return t
else
    return argmax_{a∈A}(μ̂_a + UCB_a)
```

Regardless of the assumptions made and the bandit class, the procedure for UCB strategies follows as in algorithm 2.4. For few arms, the process is easily visualised, as is done in fig. 2.1. There it is made clear how the confidence bound-based strategy naturally exploits the best arm and how the poorer arms are not explored more than needed.

Many variants of the algorithm exist; different assumptions about the distributions change the confidence bounds. While the choice of p is arbitrary, it is less of a nuisance than the choice of ϵ in the epsilon-greedy algorithm, with specific choices of p, such as the UCB1-algorithm, being well-studied and known to perform well. For example, MOSS, a modification of UCB1, has be shown to minimax-optimal for $\mathcal{E}^k_{[0,1]}$ [28]. Further, incorporating estimates of second moments improves performance in some cases [29], while incorporating the whole empirical distributions of observed rewards appears to be the most effective approach [30].

2.3.5 Thompson sampling

Thompson sampling is a Bayesian approach to the multi-armed bandit problem, being the original approach to the problem [17] in 1933, though only in the case of two arms and Bernoulli rewards and without any theoretical guarantees. This method is noteworthy for its ability to incorporate Bayesian modelling concepts into the fundamentally frequentist problem of multi-armed banditry.

The idea is to sample from the posterior distribution of the means of the arms and pull the arm with the highest sample, as described algorithm 2.5. With Bernoulli rewards and uniform priors, which is a special case of the Beta distribution, the posterior distribution is conjugate, namely also a Beta distribution, from which samples are easily drawn. The posterior distribution is updated after each pull, using the observed reward as evidence. By graphing the posterior distribution, such as is done in fig. 2.2, it is possible to see how the algorithm efficiently exploits the best arm while giving enough explorative efforts.

It was first in 2012, 79 years after its introduction, that Thompson

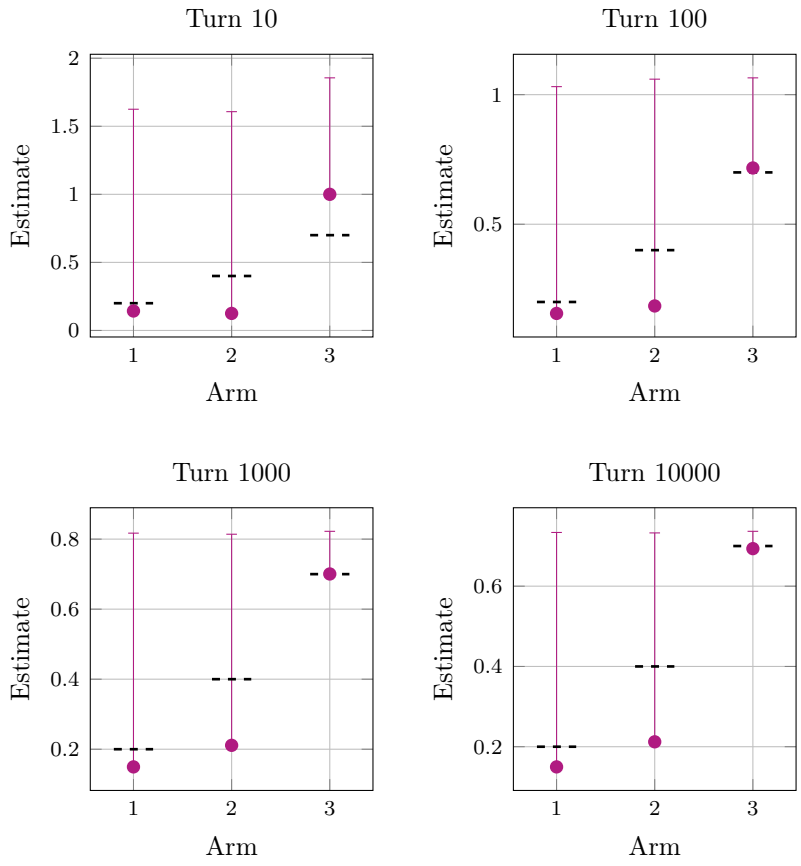


Figure 2.1: Visualisation of the UCB1 algorithm on three Bernoulli arms; estimates and error bars for each arm at different turns. The dots represent the estimates of the means of the arms, while the error bars represent the upper confidence bounds and the dashed lines represent the true means. As the number of turns increases, the estimate of the highest mean converge quickly to the true mean, while the estimates of suboptimal arms remain uncertain.

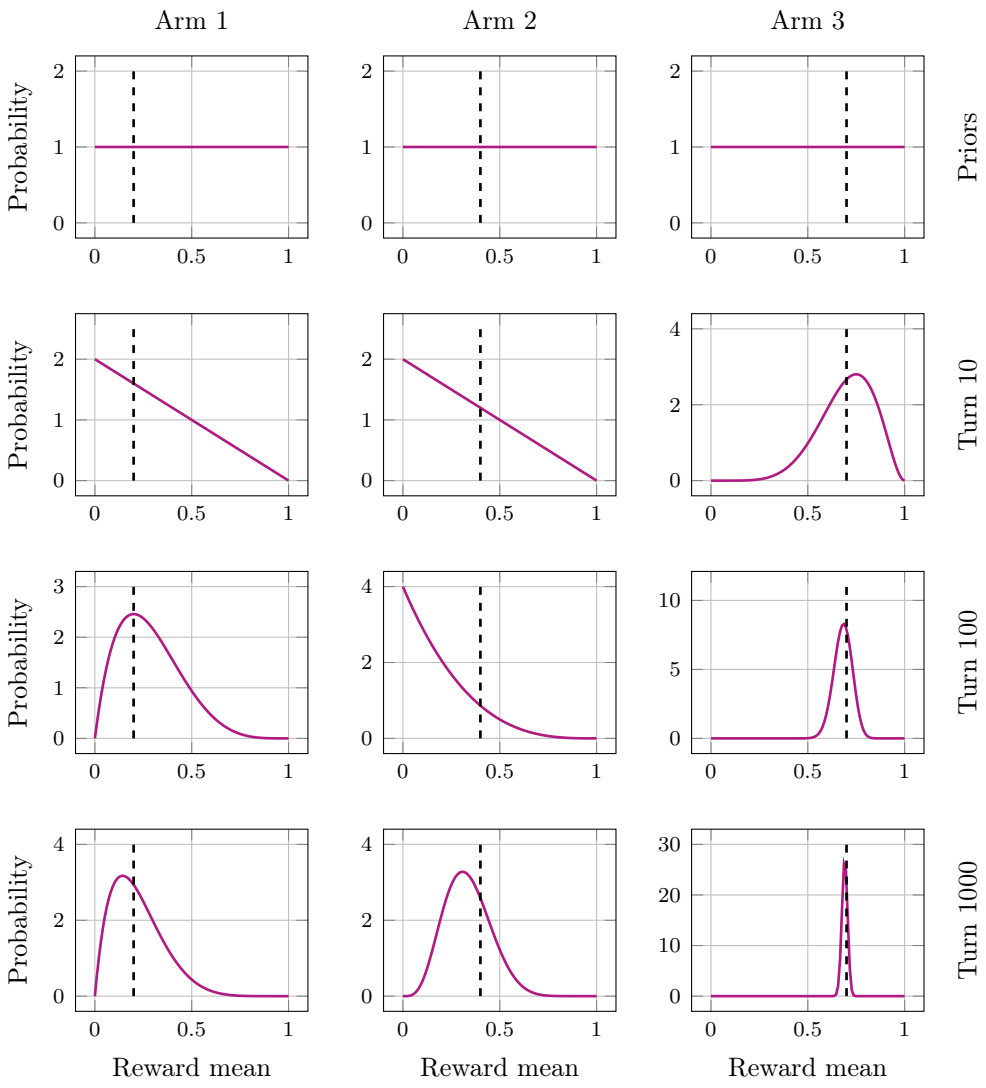


Figure 2.2: Thompson sampling algorithm with Bernoulli rewards and three arms visualised. In the top row, the uniform priors for the reward distribution means are on display. Thereunder, the posteriors are shown at turns 10, 100 and 1000. The dashed vertical lines indicate the true means. As the algorithm progresses, the posterior of the optimal arm quickly spikes and approaches the true mean, while the others remain rather wide, but importantly with almost all their mass less than the optimal arm posterior.

sampling was proven asymptotically optimal for Bernoulli rewards with uniform priors [31] Namely, for every $\epsilon > 0$, there exists a constant C_ϵ such that

$$R_T \leq (1+\epsilon) \sum_{a \in \mathcal{A} \setminus a^*} \Delta_a \frac{\log T - \log \log T}{D(P_a \parallel P^*)} + C_\epsilon, \tag{2.22}$$

where P_a is the reward distribution of arm a and P^* is the reward distribution of the best arm. Meanwhile, the minimax regret can be bounded by either $O(\sqrt{kT \log T})$ [32] or $O(\sqrt{kT \log k})$ [33], neither of which is quite minimax-optimal.

Also for Gaussian rewards, it was proven asymptotically optimal with uniform priors [34]. Notably, the Jeffreys prior was shown to be inadequate in achieving optimal regret, highlighting the importance of the prior selection for the algorithm's performance.

Algorithm 2.5: Thompson sampling arm selection

```
Update posterior for arm a_{t-1} with reward X_{t-1}
for a ∈ A do
    Sample θ_a ~ P(μ_a | D)
return argmax_{a∈A} θ_a
```

One of the key advantages of Thompson sampling is that it can natively incorporate prior knowledge about the arms, whereas doing so with the above methods would require some sort of ad-hoc manipulation of the recorded rewards and arm pull counts. Furthermore, empirical results generally indicate better performance than UCB variants [31]. Still, Thompson sampling is not without its drawbacks. The algorithm can be computationally expensive, as it requires sampling from the posterior distribution for each arm at each time step. Even with conjugate priors, the computational costs of sampling will be higher than the straightforward computations required by UCB and epsilon-greedy algorithms. It is also at its core a probabilistic policy, which for some applications may be undesirable, for example when explainability is a major concern.

2.3.6 The doubling trick

Given an algorithm reliant on knowing the horizon T, it is possible to use the doubling trick to achieve similar regret properties regardless of the horizon, and so turn the algorithm into an anytime algorithm. The doubling trick is a simple idea: simply run the algorithm for T steps first,

then $2T$ steps, $4T$, ad infinitum, possibly with some other geometric factor. It was first introduced in [35], and has since been proven to conserve minimax regrets, but not instance-dependent regrets [36]. Using instead exponential progression of the episode lengths, it is possible to maintain instance-dependent regret bounds instead of minimax optimality [36].

2.3.7 Comparison

As a quick comparison of the algorithm presented above, the regrets for a simple bandit instance are shown in fig. 2.3. Clearly, random selection performs the worst, as expected, while the greedy algorithm performs somewhat better, but still worse than the ϵ-greedy algorithm. All of these also appear to have linear regret curves, which is expected. The UCB algorithm performs better still, while Thompson sampling is without question dominant. Indeed, for at least the instance shown, the performance hierarchy indicated in this chapter is demonstrated.

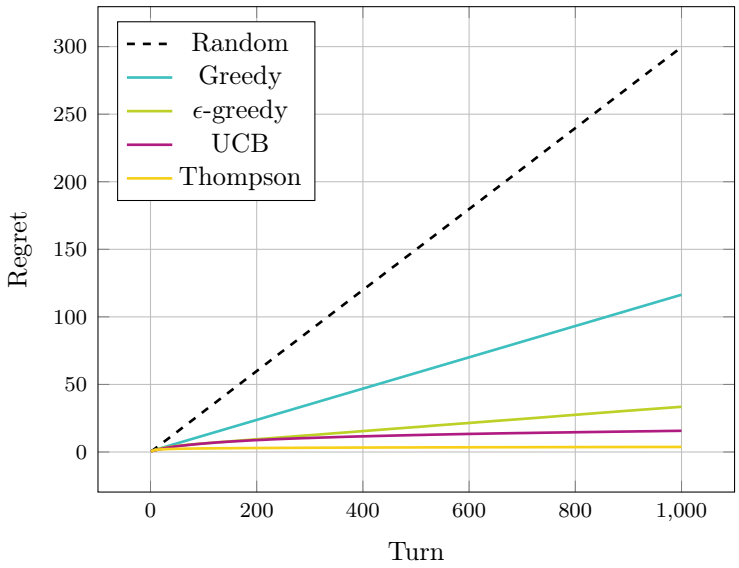


Figure 2.3: Comparison of bandit algorithms on a simple bandit instance of two Bernoulli arms with means 0.2 and 0.8. The algorithms were run for 1000 turns, and the regrets are averaged over 1000 instances.

Chapter 3

Reinforcement learning

Machine learning lies at the intersection of statistics, computer science and optimisation. The central idea is to design an algorithm that uses data to solve a problem, and in so avoid explicitly programming a solution. With ever more data available and with ever more powerful computers, machine learning has become a powerful tool in many fields, solving problems previously thought intractable. Such algorithms or models can be used for a plethora of tasks, which are mainly divided into three main categories:

Supervised learning Given data with corresponding labels, find the relationship and try to assign correct labels to new, unseen data.

Unsupervised learning Given data, find some underlying structure, patterns, properties or relationships, such as clusters or outliers.

Reinforcement learning Given an environment with a set of possible actions, such as a game, explore different strategies and determine one that optimises some reward.

This chapter will focus on the third category, reinforcement learning (RL). While bandits are a useful tool for the study of the interaction of environments, they are limited in their ability to model complex environments wherein actions have long-term consequences. The following will serve mostly as an introduction to the topic of RL, and will not go into the mathematical details of the state-of-the-art algorithms.

Unlike bandit problems which can be considered analytically and solved somewhat optimally, the environments for which RL is used are generally intractable and far too complex to be solved optimally. Exempli gratia, the game of go has approximately 10^{170} possible states, far too complicated to be solved exactly optimally. However, with modern machine learning techniques, particularly neural networks, it is possible to learn good policies and beat the best human players, which was famously demonstrated with AlphaGo in 2016 [37].

In contrast to bandits whose performance is important from the start, reinforcement learning is generally tackled by first training the model on a set of data or a simulated environment before deploying it to the real problem at hand. Consequently, the exploration-exploitation dilemma is not as pressing in reinforcement learning as it is in bandits, but it is still present. While learning strategies, there is indeed a trade-off between optimising the more promising strategies contra attempting something radically different. Epsilon-greedy strategies can be used, though more advanced strategies with adaptive exploration rates appear to be more effective [38].

The first section of this chapter will introduce the Markov decision process (MDPs), which is the mathematical framework for reinforcement learning, and it will be based primarily on [18, 39]. After that, neural networks will be introduced as a tool for solving machine learning problems, and the chapter will conclude with a brief overview of the state-of-the-art algorithms for reinforcement learning.

3.1 Markov decision processes

The mathematical framework for reinforcement learning is the Markov decision process (MDP). It can be considered as a generalisation of multi-armed bandits, where the agent observes a state before choosing an action, from which the reward probabilities — and now also state transition probabilities — depend. More precisely, a Markov decision process is a tuple $M = (\mathcal{S}, \mathcal{A}, P, r)$. The sets $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively, both of which may be infinite. The transition probabilities $P(s'|s, a_t)$ are the probability of transitioning from state s' to state s, given that action a was taken. Next, the reward function $r : \mathcal{S}^2 \times \mathcal{A} \to \mathbb{R}$ is a function that maps each transition and causing action to a real-valued reward.

The game is played similarly to a multi-armed bandit, in that the agent chooses an action at each time step t. However, before each turn, the agent observes the current state $s_t \in \mathcal{S}$. For the first turn, the state is taken from some distribution $P(s_0)$. When the agent commits to an action $a_t \in \mathcal{A}$, the environment transitions to a new state s_{t+1} according to the transition probabilities $P(s_{t+1}|s_t, a_t)$, and the agent receives a reward $r(s_t, a_t)$.

The goal is to find a (potentially probabilistic) policy which maps each state to an action, such that the agent receives the highest possible expected cumulative reward. In particular, one attempts to maximise the expected

future sum of discounted rewards, the return, defined as

$$G_t = \mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma^\tau X_{t+\tau}\right], \tag{3.1}$$

where X_t is the reward received at time t, $\gamma \in (0, 1]$ is the discount factor. The horizon T may be infinite. The discount factor is used to balance the importance of immediate rewards versus future rewards and to ensure convergence regardless of horizon finiteness. In practice, γ is a hyperparameter that is tuned to the problem at hand.

Hence, the optimal policy is given by the policy that maximises the expected return in starting state, namely

$$\pi^* = \operatorname*{argmax}_\pi \mathbb{E}\left[G_0\right], \tag{3.2}$$

where the expectation is taken over the distribution of initial states.

Note that policies will in general depend on the whole history of states, actions and rewards, and not just the current state.

3.1.1 Example: Cart-pole

A popular platform for testing RL algorithms is OpenAI Gym [40], which provides a suite of environments with different difficulty levels and objectives. One of the most well-known environments available in Gym, though first introduced earlier, is the cart-pole problem [41]. In this physically two-dimensional environment, a pole is attached to a cart that can move left or right, initially at some small random angle from the vertical The objective is to keep the pole from falling over for as long as possible by applying appropriate forces to the cart. At each time, the agent must either apply a set constant force to the either right or to the left — it can not idle. The state observed is a vector of four real numbers: the position and velocity of the cart, and the angle and angular velocity of the pole. For each time step where the pole remains upright, the agent receives a constant reward of +1, while the episode ends if the pole angle exceeds 12° or if the cart moves out of frame, id est, too far from the centre. Internally, the environment evolves according to the explicit Euler method, with a time step of 0.02 seconds[4], which, disregarding technicalities with respect to floating point arithmetic, presents an infinite state-space. It is nonetheless far too complicated to be visualised as the toy example in fig. 3.1.

[4]At least in its OpenAI Gym implementation the default settings. Cf. `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`

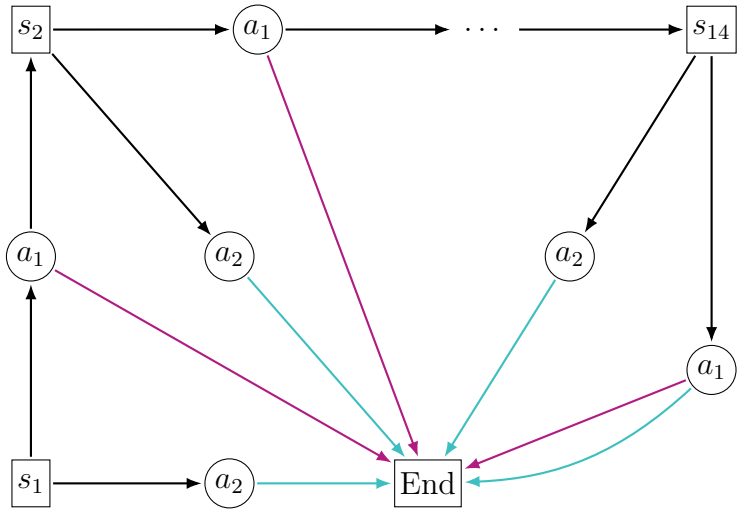


Figure 3.1: A Markov decision process (MDP) graph based on the game-show *Who Wants to be a Millionaire?* The states are how many questions the contestant has answered correctly, stating with one, and the actions are whether to go to the next question, a_1, or to cash out a_2. Cashing out proceeds to the terminal end state with certainty, yielding a reward depending on the number of questions answered correctly. Betting and going to the next question may either result in a loss and the end of the game, with a low reward, or in proceeding to the next question with no immediate reward. The probabilities of answering the questions correctly are not known, so whether continuing or cashing out is the best action is not known a priori. Moreover, with rewards being delayed until cashing out or answering the final question correctly (highlighted in blue), and losing transitions (shown in red) having the same reward of 0 as transitions to new questions, it is not completely trivial to determine the quality of an action.

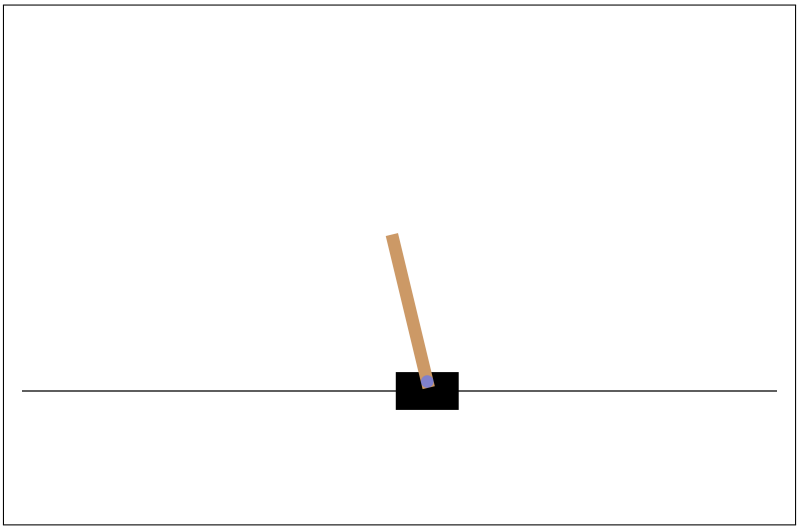


Figure 3.2: The cart-pole environment in OpenAI Gym. An agent must move the cart to the right or the left to combat gravity and keep the pole upright. If the pole falls over or the cart moves too far from the centre, the game is over. The goal is to last as long as possible.

3.1.2 Learning Markov decision processes

One key concept in MDPs is the notion of value functions. A value function is a function that estimates the expected cumulative reward from a given state or state-action pair, under a given policy. Specifically, the state-value function $V^\pi(s)$ estimates the expected cumulative reward from state s, under policy π. It is given by

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right], \tag{3.3}$$

where the expectation is taken over the distribution of future rewards and states, given that the agent is in state s at time t.

Alternatively, one may consider a state-action value function $Q^\pi(s, a)$, which estimates the expected cumulative reward from state s, given that the agent takes action a, under policy π for future steps. It can be expressed as

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right]. \tag{3.4}$$

Naturally, it is desirable to maximise either, leading to the Bellman equations, from which the optimal policies can be derived. This is alas rarely possible exactly; even if the mechanics of the game is known exactly a prior (like in card games or chess), the state spaces will generally be too large to be able to compute the value functions exactly. Instead, one must

resort to approximate methods, such as dynamic programming, Monte Carlo tree-search, and temporal difference learning.

There are three main categories of reinforcement learning algorithms:

Value-based algorithms The state-value function or the action-value function is learnt and then used to determine the optimal policy. Value-based algorithms can be very effective in problems with large state spaces, as they inherently extract the most important features of the state. This also makes them good at generalising to unseen states without requiring too many samples, and they can benefit from data obtained earlier in the training process with an old policy. They can nonetheless be sensitive to the choice of hyperparameters and may struggle with problems with continuous action spaces.

Policy-based algorithms A policy, either a deterministic or stochastic mapping from states to actions, that maximises the expected cumulative reward, is learnt directly. This is done by defining some parametric policy $\pi_\theta(a|s)$, where θ is a vector of parameters, and then optimising the parameters θ given the data collected from the agent's interactions with the environment thus far. Policy-based algorithms are well suited for problems with continuous action spaces, as their parametric mappings will naturally produce continuous actions. Their inherent stochasticity also makes them well suited for exploration, as they will naturally explore different actions, also making them more robust to noise and environmental randomness. They can easily converge to local minima, which may be good enough in some cases, but also make it hard to train them as well as is desired.

Model-based algorithms These algorithms learn a model of the environment, including the transition probabilities and rewards, and then use this model to plan and optimise the agent's behaviour. Model-based algorithms can be very effective in problems where the environment is predictable, and the agent can simulate different action sequences to find the optimal policy. However, these algorithms can be computationally expensive and may require a large amount of data to learn an accurate model.

It is primarily the model-free, latter two methods (or combinations thereof) that have seen the most success in recent years, and these which will be discussed further in section 3.3. But also model-based algorithms have seen some commendable results recently, for example, playing *Minecraft* [42] and being able to find diamonds [43].

3.1.3 Difficulties

The exploration-exploitation dilemma as discussed in chapter 2 is also central to reinforcement learning. To learn an optimal policy, an agent needs to explore the environment to discover new and potentially rewarding actions, while at the same time exploiting the actions that are already known to be rewarding. Balancing exploration and exploitation is a difficult problem, and many RL algorithms use heuristic exploration strategies or rely on random noise to encourage exploration.

Another challenge in RL is the problem of credit assignment. The credit assignment problem refers to the difficulty of assigning credit to the actions that lead to a particular reward. In some cases, the reward may be delayed, making it difficult to determine which actions led to the reward. This problem is especially pronounced in environments with long time horizons, where the actions taken early in the episode may have a significant impact on the final reward.

Designing appropriate rewards is a critical component of reinforcement learning, as they guide the agent's behaviour towards achieving the desired outcome. However, poorly designed rewards can lead to slow or intractable learning, as the agent may not receive sufficient feedback on its actions to adjust its policy. A classic example thereof is pausing the game in *Tetris* as not to lose the game [44].

Specialised rewards that provide more detailed feedback and guidance can improve learning, but they also make it harder to generalise to new situations. For instance, the Atari game *Montezuma's Revenge* [45] is nigh impossible to solve with off-the-shelf methods and without specialising the rewards [46]. While the suite of Atari games is mostly solvable with the same algorithms, the lack of immediate rewards leaves agents clueless as to how to progress. Implementing the necessary guidance can not only be demanding, but it also loses the general applicability that is so central to RL.

RL is fundamentally more challenging than supervised learning because it requires an agent to explore and interact with its environment to learn from experience, as opposed to having access to labelled data. Randomly discovering an optimal strategy can be highly unlikely, especially in high-dimensional state and action spaces, making it necessary to develop specialised algorithms [39]. While humans can employ prior knowledge, such as that keys will open doors, reinforcement learning agents typically have to learn this by chance. In a testing video game, the removal of non-essential visual elements, altering the orientation and employing unconventional control schemes had minimal impact on RL algorithms, whereas it caused a

drastic shift for humans, going from completing the game much faster than the RL algorithms to being unable to complete the game whatsoever [47]. It can be computationally expensive and require significant amounts of data to learn an optimal policy, which is why the idea of algorithms competing against themselves has been so central in achieving super-human performance in board and video games [37]. All this leads to the necessity of deep learning in reinforcement learning. Before those methods can be explained, the general concept of deep learning needs to be discussed, namely the artificial neural network.

3.2 Neural networks

Modern machine learning owes much of its popularity to the success of artificial neural networks, or if the context is clear, just neural networks (NNs). With easier access to larger data sets, more powerful hardware (in particular GPUs or even dedicated TPUs) and the backpropagation algorithm, NNs have become able to solve problems far too complicated for traditional methods.

Though state-of-the-art neural networks can contain billions of parameters [48, 49], training them remains feasible. Modern hardware is of course paramount, but also backpropagation is crucial. Neural networks are trained using gradient methods, and with backpropagation, the gradient can be computed efficiently. By cleverly storing intermediate results, the gradients for all parameters can be computed in a single backward pass through the network.

How such large models avoid overfitting is not entirely clear. Seemingly contradicting the bias-variance trade-off, the double descent phenomenon appears as model complexity increases or as more gradient descent iterations are done. This is a phenomenon where, after some point, the variance no longer increases with model complexity. Instead, it decreases and converges toward some value. This can be seen in fig. 3.3, where convolutional networks[5] were tested with different layer sizes. Unlike statistical methods, once past this complexity threshold, there is little reason, other than the increased computational cost, not to increase the model complexity and thereby the performance. Double descent has been shown to appear for a variety of models [50].

With the great size and complexity, interpretability is sacrificed. The models are deterministic and often black boxes; it is difficult to understand

[5] Q.v. section 3.2.1.

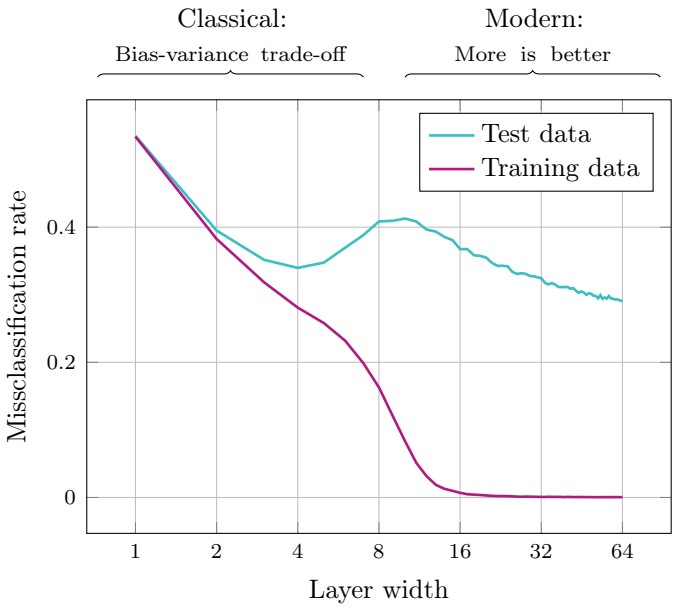


Figure 3.3: Double descent phenomenon. As model complexity increases, the training error decreases steadily, but the test error reaches a minimum and after which the test error starts to increase again. However, past a certain point, the test error decreases again. This defines two regimes: the first classical regime, where the bias-variance trade-off applies, and model complexity must remain low to avoid overfitting, compared to the modern ML regime, where more complexity is always better. The data is from [50], where ResNet18 models were used for image classification with added label noise.

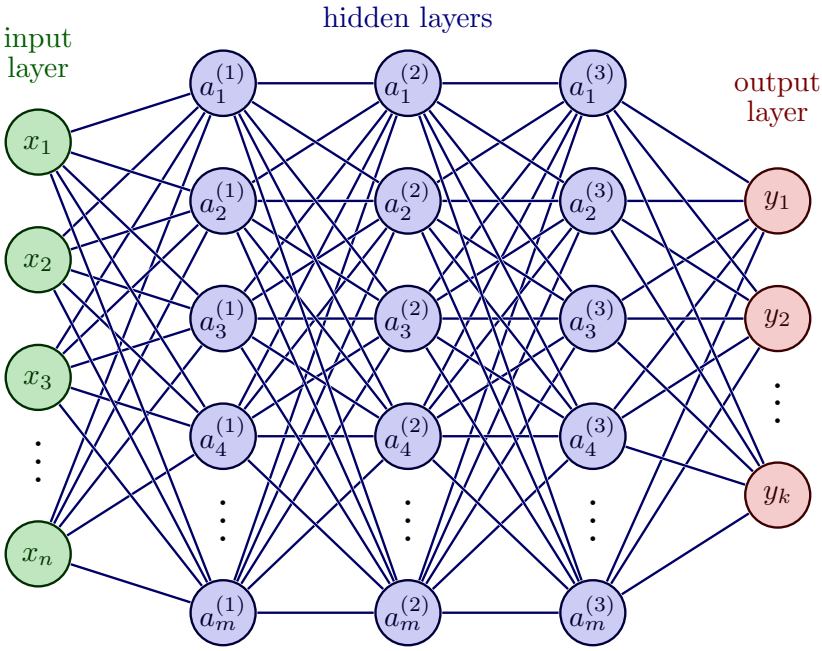


Figure 3.4: Typical structure of a dense feed-forward neural network. Here it has three hidden layers of constant size m, but the number of layers and the size of each layer can be chosen arbitrarily. Being dense means that each node in one layer is connected to all nodes in the next layer, while being feed-forward means that the connections are unidirectional. From [51].

why they make the predictions they do. Luckily, there have been some developments, perhaps most notably the universal approximation theorem. It states that a neural network with a single hidden layer can approximate any continuous function to arbitrary precision, given enough neurons[6]. This gives some credence to the idea that NNs with any kind of structure could be used to approximate intricate functions well.

3.2.1 Architectures

Dense feed-forward neural networks

The basic neural network is a dense feed-forward neural network, with a typical structure shown in fig. 3.4. There can be more or fewer nodes in each layer, and as many or few hidden layers as desired.

[6]In addition to some easily met requirements regarding the activation function.

In the input layer, the data is fed in with each input node corresponding to a feature. Then, in the hidden layers, the data is transformed by a series of linear transformations and non-linear activation functions. These can be expressed as

$$a_i^{(j)} = \sigma\left(b_i^{(j)} + \sum_{n=1}^{m} w_{i,n}^{(j)} a_n^{(j-1)}\right), \tag{3.5}$$

where j is the layer number, i is the node number and w and b are the weights and biases of the network — parameters to be optimised.

The activation function σ is a non-linear function, such as the sigmoid function, the hyperbolic tangent or the rectified linear unit (ReLU). Non-linearity is needed for the network not to collapse into one great linear transformation. Some commonly used activation functions are listed in table 3.1.

The output layer is similar to the hidden layers, though perhaps with different activation functions and fewer nodes. For instance, if the goal is to choose one of k different action, the output layer could have k nodes with an activation function that ensures the sum of the outputs is one. In that way, the output can be interpreted as a policy for choosing an action.

Models being dense mean that each node in one layer depends on all nodes in the previous layer. It is feed-forward, because the data flows in one direction; the perceptrons in a layer depends only on those in the former, and there are no cycles.

Convolutional neural networks

Convolutional neural networks (CNNs) are a special type of neural network that are particularly suited for certain tasks like image processing. A greatly simplified CNN is shown in fig. 3.5.

The basic component of the CNN is the convolutional layer. In it, a kernel or filter is applied to the input data, which often is as simple as a 3×3 matrix. It is applied to only parts of the input, and thus only extracts local properties. Typically, pooling layers complement the convolutions by reducing the dimensionality through some simple non-parametrised operation, such as taking the maximum value in a 2×2 matrix. CNNs generally finish with one or more dense layers, which then operate on a significantly reduced number of features. The reduction of dimensionality is important because it reduces the number of parameters in the model and the risk of overfitting. Furthermore, the convolutional approach forces the model to learn local features. This is very beneficial for tasks where

Table 3.1: Common activation functions in neural networks. Usually, they are applied element-wise to the output of a linear transformation. However, in the case of the softmax function, it depends on the whole layer.

Name	Definition	Typical use case
Identity	$\sigma(x_i) = x_i$	Regression output
Sigmoid	$\sigma(x_i) = 1/(1 + e^{-x_i})$	Hidden layers, binary classification output
Hyperbolic tangent	$\sigma(x_i) = \tanh(x_i)$	Hidden layers, binary classification output
ReLU	$\sigma(x_i) = \begin{cases} x_i, & x \geq 0 \\ 0 & x < 0 \end{cases}$	Hidden layers
Leaky ReLU	$\sigma(x_i) = \begin{cases} x_i, & x \geq 0 \\ 0.01x_i, & x < 0 \end{cases}$	Hidden layers
Softmax	$\sigma(x_i) = e_i^x / \sum_{j=1}^{k} e^{x_j}$	Multi-class classification output

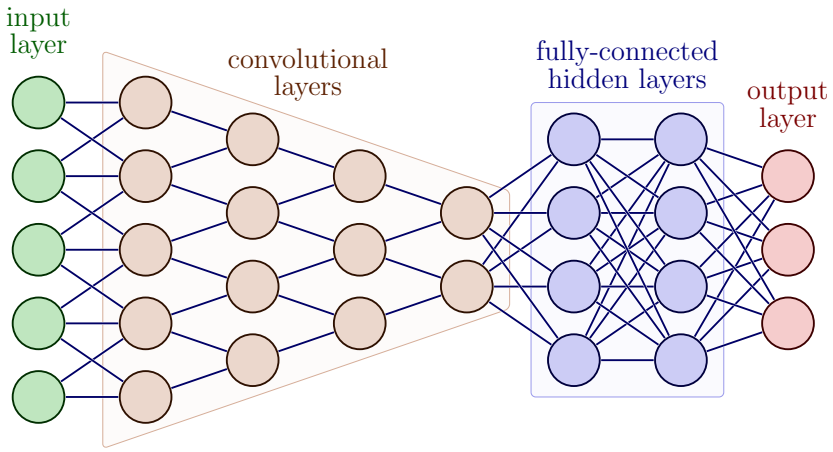


Figure 3.5: The basic structure of a convolutional neural network. Data is input before conventional layers are applied, in which perceptrons are only connected to small regions of the previous layer. After sufficient dimensionality reduction, regular dense layers can be used. From [51].

the inputs are images, as local features, such as edges, are more important than global ones, such as the position of the subject.

Recurrent neural networks

Recurrent neural networks (RNNs) are a special type of neural network wherein cycles are allowed. This enables a temporal quality, which makes them particularly suited for tasks such as time-series prediction, speech recognition and machine translation. RNNs' flexibility permits them to take inputs of varying lengths, which is not natively supported by regular feed-forward networks. They are subsequently well suited for reinforcement learning, where the states and rewards from the environment can be continuously fed into the network as the agent interacts with it. For example, they have shown particularly useful for so-called partially observable Markov decision processes, where the agent does not have access to the full state of the environment [52].

However, the flexibility comes at a cost. As the network is allowed to have cycles, gradients of parameters can compound and either vanish or explode exponentially, which makes training difficult. Therefore, more advanced variants of RNNs are used, such as long short-term memory (LSTM) and gated recurrent units (GRU).

Generative adversarial networks

Generative adversarial networks (GANs) refer to a special kind of design where two different networks are trained by competing against each other. With an unlabelled data set, the goal is to generate data that is indistinguishable from the real data. The first model, the generator, attempts to generate samples from the underlying distribution. On the other hand, the discriminator is tasked with distinguishing between data produced by the generator and the real samples from the data set. Accordingly, discrimination is a supervised problem. Both models are trained simultaneously by first sampling from the generator and the data set, using supervised methods to update the discriminator, and then using the discriminator's predictions to update the generator.

GANs are mainly used for unsupervised learning, having had great success in generating random images. They have also demonstrated success in more abstract tasks, such as translating text prompts to images or predicting what will happen next in a video.

3.3 State-of-the-art algorithms

3.3.1 Value-based methods

One of the most popular modern value-based algorithms is the Deep Q-Network (DQN) algorithm [53]. DQN is an extension of action-value learning that uses a neural network to approximate the Q-function. The neural network takes the current state as input and outputs the Q-values for each action. Storing previous actions and returns, the network is trained using a variant of stochastic gradient descent that minimises the mean squared error between the predicted Q-values and the target Q-values. DQN uses an ϵ-greedy exploration strategy to balance exploration and exploitation during training.

DQN has been shown to be effective in a wide range of reinforcement learning problems, including Atari games and robotics tasks. Receiving only pixel values from the video game screen, the DQN algorithm was able to learn to play Atari games at a level comparable to professional human players [54].

One limitation of DQN is that it can be slow to converge, especially in large state spaces. To address this issue, several extensions to DQN have been proposed, such as the double DQN algorithm [55], prioritised experience replay algorithm [56] and duelling DQN algorithm [57].

3.3.2 Policy gradient methods

Policy gradient methods are a class of Reinforcement Learning algorithms that directly optimise the policy of an agent. These methods are particularly well-suited to problems with continuous or high-dimensional action spaces, where it may be difficult to find the optimal policy using value-based methods.

One popular policy gradient algorithm is the REINFORCE algorithm [58]. The REINFORCE algorithm is a Monte Carlo policy gradient method that estimates the gradient of the expected cumulative reward with respect to the current policy parameters, using this gradient to update the policy. The algorithm is based on the likelihood ratio method, which allows the gradient of the expected returns to be expressed as the product of the reward and the gradient of the log-probability of the actions under the policy.

The REINFORCE algorithm uses this gradient to update the policy parameters in the direction that increases the expected cumulative reward.

Specifically, the update rule for the policy parameters is given by

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \log \pi_\theta(a_k|s_k) G_t, \tag{3.6}$$

where θ_t and θ_{t+1} are the policy parameters at time steps t and $t+1$, α is the learning rate, $\pi_\theta(a_k|s_k)$ is the probability of taking action a_k in state s_k under the policy π_θ and G_t is the expected cumulative reward starting from time step t, found by sampling trajectories from the current policy. This sampling results in high variance, which can make the training process unstable [52].

The Proximal Policy Optimisation (PPO) algorithm [59] has been particularly successful, expanding on the ideas from the Trust Region Policy Optimisation (TRPO) algorithm [60]. It is based on the idea of clipping the policy update, which helps to prevent large policy updates that could destabilise the training process, and was designed to require little hyperparameter tuning. The algorithm uses a surrogate objective function that combines the clipped policy objective and the value function objective, and updates the policy and value function parameters using a combination of stochastic gradient descent and trust region optimisation. It has become a popular baseline for reinforcement learning problems thanks to its performance, ease of implementation and being simple to tune [59]. What is more, PPO has been able to beat the *Dota 2* [61] world champions [62, 63].

3.3.3 Actor-critic methods

Actor-critic methods are types of reinforcement learning algorithms that combine ideas from both value-based and policy-based methods. Proposed in [64], these algorithms maintain both a policy function and a value function, where both are learnt simultaneously during the training process. The value function estimates the expected return from a given state, while the policy function defines the probability distribution over actions given the current state. The policy function is typically represented using a neural network, and the value function can also be represented using a neural network or some other function approximator.

One popular actor-critic algorithm is the Advantage Actor-Critic (A2C) algorithm [65]. The A2C algorithm updates both the policy and value function parameters using stochastic gradient descent. The policy update is based on the policy gradient, while the value function update is based on the temporal difference error.

A major advantage is that actor-critic methods can improve the stability and convergence of the training process by using the value function to

guide the policy updates. This is because the value function provides a baseline estimate of the expected return, which reduces the variance of the policy gradient estimates. Actor-critic methods have been shown to be effective in a wide range of reinforcement learning problems, exempli gratia, among all players of the video game of *Starcraft II* [66], achieving top 0.15% performance [67].

Chapter 4

Quantum computing

The field of quantum computing is split into two main branches: the development of quantum hardware and the study of algorithms that use such hardware. Only the second branch is relevant to this thesis, and even so, only a brief explanation is offered here. For more details, see [68] for a rigorous, complete description or [69] for an introduction focused on programming. Any reader should have a basic understanding of linear algebra, classical computing and computational complexity. Knowledge of quantum mechanics is not assumed, albeit certainly helpful.

4.1 Quantum states

4.1.1 The qubit

The quantum bit, the qubit, is the building block of quantum computing. Like the classical binary digit, it can be either 0 or 1. But being quantum, these are quantum states, $|0\rangle$ and $|1\rangle$[7], and the qubit can be in any superposition of these states. This follows from the first postulate of quantum mechanics[8], which states that an isolated system is entirely described by a normalised vector in a Hilbert space. For the qubit, this is the two-dimensional space where the states $|0\rangle$ and $|1\rangle$ are basis vectors, known as the computational basis states. Thus, the state of a qubit can be expressed as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix}, \tag{4.1}$$

[7]The $|\cdot\rangle$ notation is known as a ket and is used in quantum mechanics to denote a quantum state. It is effectively a column vector. The inner product may be taken with a bra, $\langle\cdot|$, to give a scalar. These inner products are then denoted by $\langle\cdot|\cdot\rangle$. Similarly, outer products are well-defined and denoted by $|\cdot\rangle\langle\cdot|$.

[8]As they are laid out in [68].

where $\alpha, \beta \in \mathbb{C}$. The only requirement is that the state be normalised, $|\alpha|^2 + |\beta|^2 = 1$. Normalisation is required due to the Born rule, as the absolute square of the coefficients is the probability of measuring the qubit in the corresponding basis state.

4.1.2 The Bloch sphere

A useful tool for visualising the state of a qubit is the Bloch sphere. First, it should be noted that states on the form eq. (4.1) are not physically unique, only the relative complex phase matters. There is a global phase which can not be observed, and so it is not physically relevant. Taking that and the normalisation into account, the state of the qubit can be expressed as

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\,, \tag{4.2}$$

where $\theta, \phi \in \mathbb{R}$. Interpreting θ as the polar angle and ϕ as the azimuthal angle, the state of the qubit can be identified with a point on a sphere. See fig. 4.1. The state $|0\rangle$ is typically thought of as the north pole of this sphere and $|1\rangle$ as the south pole.

4.1.3 Mixed states and density operators

It is not only the superpositions of states that are important in quantum computing, but also the mixed states, states that are statistical ensembles of pure states. Pure states are those expressible as a single ket like eq. (4.1), while mixed states arise when the preparation of the system is not perfectly known or when the system interacts with the environment. For the description of mixed states, the formalism of density operators is more useful than the state vector formalism. If there are no classical uncertainties, the state is pure, and the density operator can be expressed a single ket-bra,

$$\rho = |\psi\rangle\langle\psi|\,. \tag{4.3}$$

In a mixed state, however, some classical probabilities p_i are associated with the different pure states $|\psi_i\rangle$, and the state of the system is described by the density operator

$$\rho = \sum_{i=1}^{n} p_i\,|\psi_i\rangle\langle\psi_i| \tag{4.4}$$

where $|\psi_i\rangle$ are the states of the system, and $\langle\psi_i|$ are the corresponding dual vectors. Being probabilities, the p_i must be non-negative and sum

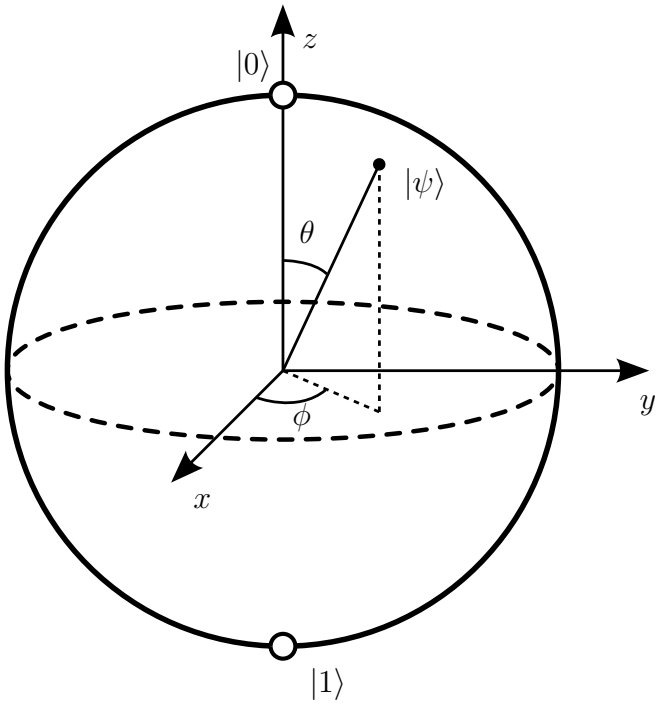


Figure 4.1: The Bloch sphere. On it, the state of a single qubit state is represented by a point. The state $|0\rangle$ is the north pole, and $|1\rangle$ is the south pole. The latitudinal angle θ determines the probability of measuring the qubit in the state $|0\rangle$, while the longitudinal angle ϕ corresponds to the complex phase between the two basis states. From [70].

to one. Given a basis and a finite Hilbert space, the density operator can be expressed as a density matrix[9] where the diagonal elements are the probabilities of measuring the system in the corresponding basis state. Furthermore, it is easily seen that the density operator must be positive semidefinite and Hermitian.

The Pauli matrices,

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{4.5}$$

together with the identity matrix serve as a basis for the real vector space of Hermitian 2×2-matrices. Since the diagonal elements of a density matrix must sum to one, the density matrix for a single qubit can be expressed as

$$\rho = \frac{1}{2}\left(I + x\sigma_x + y\sigma_y + z\sigma_z\right), \tag{4.6}$$

where $x, y, z \in \mathbb{R}$. Being positive semidefinite, the determinant should be non-negative, and thus it can be shown that $x^2 + y^2 + z^2 \leq 1$. This allows density operators to be interpreted as points on the Bloch sphere or indeed within it. Notably, pure states lie on the surface, while mixed states lie within the sphere (or rather, the Bloch ball). A pure quantum superposition of $|0\rangle$ and $|1\rangle$ with equal probabilities would have a complex phase and lie somewhere on the equator, while a statistical mixture with equal classical probabilities of being $|0\rangle$ and $|1\rangle$ would lie in its centre.

4.1.4 Systems of multiple qubits

Although the continuous nature of the qubit is indeed useful, the true power of quantum computers lies in how multiple qubits interact. Having multiple qubits enables entanglement, which is a key feature of quantum computing.

With two qubits, there are four possible states, $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. Each of these four states has its own probability amplitude, and thus its own probability of being measured. A two-qubit system will therefore operate with four complex numbers in the four-dimensional Hilbert space $\mathbb{C}^4$.

Generally, the state of multiple qubits can be expressed using the Kronecker product (a special case of the tensor product) of the individual

[9] Density operators and matrices are often used interchangeably in quantum computing. Due to the finite number of qubits, the Hilbert spaces are always finite-dimensional, and with the canonical basis, there is a canonical way of representing density operators as matrices.

qubits, as in

$$|\psi_1\psi_2\cdots\psi_n\rangle = |\psi_1\rangle\,|\psi_2\rangle\cdots|\psi_n\rangle = |\psi_1\rangle\otimes|\psi_2\rangle\otimes\cdots\otimes|\psi_n\rangle\,. \tag{4.7}$$

What makes this so powerful is that the state of a multi-qubit system has the general form

$$\begin{aligned}|\psi_1\psi_2\cdots\psi_n\rangle &= c_1\,|0\ldots00\rangle + c_2\,|0\ldots01\rangle + \cdots + c_{2^n}\,|1\ldots11\rangle \\ &= (c_1, c_2, \ldots, c_{2^n})^\top \\ &\in \mathbb{C}^{2^n},\end{aligned} \tag{4.8}$$

which means that with n qubits, the system can be in any superposition of the 2^n basis states. Operating on several qubits then, one can do linear algebra in an exponentially large space. This is a key part of the exponential speed-ups possible with quantum computers.

4.2 Quantum operations

4.2.1 Single-qubit gates

To operate on one or more qubits, a unitary operation is applied to the state. This is a computational interpretation of the unitary time evolution resulting from a Hamiltonian acting on the (closed) quantum system, described by the second postulate of quantum mechanics and the Schrödinger equation. As the operations are unitary, a pure state remains pure. These operations are often thought of as gates, paralleling the classical gates in digital logic. Mathematically, with a finite number of qubits, a unitary gate U can be expressed as a matrix acting on the state vector, $|\psi\rangle$, as

$$|\psi'\rangle = U\,|\psi\rangle\,, \tag{4.9}$$

where $|\psi'\rangle$ is the resulting state.

The most basic gates are the Pauli gates, which are applications of the Pauli matrices from eq. (4.5) and are as gates simply denoted as X, Y, and Z. These gates can be seen as half turns around the x-, y- and z-axes, respectively, of the Bloch sphere. The X-gate is also known as the NOT gate, as it mirrors the classical NOT gate by mapping $|0\rangle$ to $|1\rangle$ and vice versa. It is however more general, being also applicable to superposition states.

The Hadamard gate,

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \tag{4.10}$$

is a rotation around the line between the x- and z-axes by $\pi/2$. It is an important gate in quantum computing, as it is used to create superpositions of the computational basis states. Applied on an initial $|0\rangle$ state, it creates the entangled state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Two consecutive applications thereof return the state to the initial state, as can be seen from the matrix squaring to the identity.

The R_X-, R_Y- and R_Z-gates are rotations around the x-, y- and z-axes, respectively, by an arbitrary angle θ:

$$R_X(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix},$$
$$R_Y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix},$$
$$R_Z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}.$$

These parametrised gates will be useful in section 6.1.

4.2.2 Multi-qubit gates

Multi-qubit gates are gates that act non-trivially on more than one qubit. The most used multi-qubit gate is the controlled X-gate, also known as the CNOT. Being controlled means that it only acts on the second qubit if the first qubit is in the state $|1\rangle$. Of course, the first qubit may be in a superposition, and the CNOT this way allows for the creation of entanglement between the two qubits. If the first qubit has probability amplitude α of being in the state $|1\rangle$, the second qubit will have probability amplitude α of being flipped. The CNOT-gate, acting on the leftmost qubit in the tensored two-qubit system can be expressed in matrix form as

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4.11}$$

In theory, any unitary single-qubit operation can be controlled. However, it is often only the CNOT that is used is implemented in the hardware. Another interesting two-qubit gate is the controlled Z-gate, CZ, expressible as the matrix

$$\text{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{4.12}$$

Because it only alters the amplitude of $|11\rangle$, it does not actually matter which qubit is the control and which is the target.

4.2.3 Observables and measurements

For an output to be obtained from a quantum computer, a measurement must be performed. This is typically done at the end of all operations and of all qubits, where each qubit is measured in the computational basis to yield a string of bits.

As described by the third postulate of quantum mechanics, any observable quantity has a corresponding Hermitian operator A, spectrally decomposable as $A = \sum_i \lambda_i P_i$, where λ_i are the (necessarily real) eigenvalues and P_i are the corresponding projectors onto the eigenspaces. When measuring, the probability of obtaining the outcome λ_i is given by

$$p_i = \langle\psi| P_i |\psi\rangle , \tag{4.13}$$

where $|\psi\rangle$ is the state before the measurement. It is one of Nature's great mysteries what exactly a measurement is and even more so how and why it is different from the unitary evolution described by the second postulate. In the quantum computational setting, it can be thought of as taking a random sample with the probabilities as given by the above equation.

Often, the underlying probabilities are what is of interest. Therefore, many measurements will be performed. Usually, these results are averaged to obtain an estimate, but more complicated post-processing methods are also possible. For instance, neural networks have shown useful properties in regard to reducing variance in the estimates, though at the cost of some bias [71].

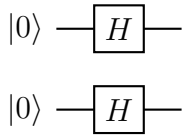
Canonically, the computational Z-basis is used for measurements, and it is usually the only basis for which measurements are physically implemented in a quantum computer. When measuring in the computational basis in which a state is expressed, as eq. (4.8), the probabilities are simply given by the absolute square of the coefficients. To virtually measure another observable, a change of basis is performed. This is achieved by applying a unitary transformation before measurement.

Measurements may be done in the middle of a computation and be used to control gates. If the qubits are entangled, measuring one will affect the measurement probabilities of others. Using such intermediate measurements is a way of introducing non-linearities in the otherwise unitary nature of the unmeasured quantum world.

4.2.4 Quantum circuits

The operations on qubits are often described using quantum circuits, which are a graphical representation of the operations on the qubits, the quantum algorithms. They are read from left to right. It is standard procedure to assume all qubits start in the state $|0\rangle$. Gates are generally written as boxes with the name of the gate inside.

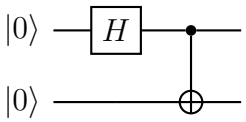
A simple example is the circuit



, (4.14)

which prepares the state $\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. This is a pure state with no entanglement, and so the measurement probabilities of the two qubits are independent. When measured, all four outcomes are equally likely.

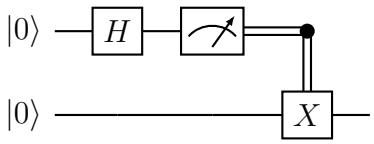
Slightly more interesting is the circuit



(4.15)

in which the first qubit is put into a superposition using the Hadamard gate before a CNOT gate is applied to the second, controlled by the first. This creates the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. The measurement probabilities of the two qubits are now correlated; if the first qubit is measured to be $|1\rangle$, the second will always be $|1\rangle$ and vice versa. The probability of measuring the qubits to be different is nil.

To create a mixed state, an intermediate measurement can be used to control a gate. For instance, the circuit



(4.16)

places the second qubit in the mixed state $\frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$. If it were immediately to be measured, it would have a 50% chance of being $|0\rangle$ and a 50% chance of being $|1\rangle$. The uncertainty is only classical, and it could therefore not be used to create entanglement or for any other quantum spookiness.

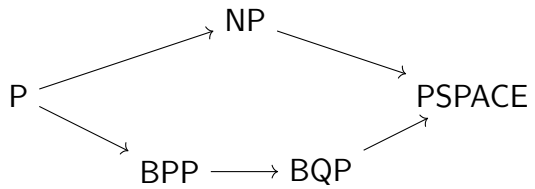
4.3 Quantum algorithms

4.3.1 Computational complexity and quantum supremacy

Quantum computers' allure stems from their ability to solve certain problems more efficiently, exponentially so, than classical computers can. Most (in-) famous of these problems are integer factorisation and discrete logarithms, which could be used to break effectively all public-key cryptography, but there are still several more problems for which quantum computers are believed to offer great advantages. It is proven that the class of problems quantum computers can solve in polynomial time (with high probability), BQP, contains the complexity class P [68] and BPP, the class of problems that can be solved in polynomial time with a classical probabilistic algorithm [68]. This follows from the fact that quantum computers can efficiently run any efficient classical algorithm and reproduce any classical randomness by performing quantum measurements. To produce a sample from a desired probability distribution, a corresponding superposition can be prepared and measured.

The Deutsch-Jozsa algorithm [72], while of no practical use, is proved to be exponentially faster than any deterministic classical algorithm[10] [73]. Since quantum computers can solve problems like integer factorisation and discrete logarithms efficiently, it is believed that BQP is strictly greater than P by containing elements of $\mathsf{NP} \setminus \mathsf{P}$, but as whether $\mathsf{P} = \mathsf{NP}$ is unknown, these problems could turn out actually to be in P. In a similar vein, NP-complete problems are believed to lie outside BQP, such neither BQP nor NP contains the other, but this is again unproven [74].

The complexity hierarchy can be expressed thus:



Other than $\mathsf{P} \stackrel{?}{=} \mathsf{BPP}$, it is widely conjectured that the containments are strict, but this is not proven. As disproving $\mathsf{P} = \mathsf{PSPACE}$ remains elusive,

[10] The provable exponential speed-up does not prove $\mathsf{P} \neq \mathsf{BQP}$, as the algorithm requires an oracle [75]. Oracle separations, a weaker notion than true inequality, have been found not only between P and BQP, but also between BQP and BPP with Simon's algorithm [73].

proving that BQP is strictly greater than P must be even harder. It is therefore hard to say anything definite about quantum supremacy and being able to solve more problems than classical computers can.

The exponential speed-ups that are discovered have not come easily. Although the state spaces are exponentially large, with only a limited set of operations available, states can not be created and manipulated arbitrarily without exponentially many steps. What is more, recapturing all the information in a general state can not be done without exponentially many measurements, negating any exponential gains. The problem must have some structure to be exploited for a speed-up to be possible. Quantum computers have only been shown to solve certain problems exponentially faster than classical computers, and finding the algorithms to do so is not trivial. Shor's algorithm has time complexity $O((\log N)^3)$, while the most efficient known classical algorithm, the general number field sieve, is 'sub-exponential' with a time complexity on the form $\Omega(k^{\frac{1}{3}} \log^{2/3} k)$, where $k = O(2^N)$ [76]. To solve linear systems, there is the HHL algorithm with time complexity $O(\log(N)\kappa^2)$, where κ is the condition number. This is an exponential speed-up over the fastest known classical algorithm[11], which has time complexity $O(N\kappa)$. Still, these are non-trivial algorithms, not yet usable in practice and that were not easily found.

Polynomial speed-ups are more easily found. For example, the Grover algorithm which is used to search for an element in an unsorted list has time complexity $O(\sqrt{N})$ [78]. Classically, this can not be done in less than $O(N)$ time. It can be proven that the Grover algorithm is optimal [79], so for this problem,$>$ an exponential speed-up is impossible. This algorithm and the more general amplitude amplification, on which it builds, solve very general problems and are often used in other algorithms as subroutines to achieve quadratic speed-ups. Being only a quadratic speed-up, it is not as impressive as the exponential speed-ups, and achieving quantum supremacy in that manner would require larger quantum computers than if the speed-up were exponential.

4.3.2 Grover's algorithm

The quantum search algorithm of Grover [78] is a quantum algorithm that finds an element in an unstructured list with high probability. While such a problem necessarily requires $O(N)$ time in a classical setting, needing on

[11] Given that the condition number does not grow exponentially. There are also difficulties in loading the data into the quantum computer and extracting the solution that could negate any exponential speed-up. Cf. [77].

average $N/2$ steps to find the element and in the worst case N, Grover's algorithm finds the element in $O(\sqrt{N})$ steps. This is a quadratic speed-up.

Grover's algorithm is provably optimal; no quantum algorithm can perform such general, unstructured searches any faster [79]. This should not be surprising. If an exponential speed-up were possible, Grover search could be used to find the solution to NP-hard problems fast.

For Grover's algorithm to work, assume there is a function f that maps the index of an element to 1 if it is the one desired and 0 otherwise. (Assume for now that there is only one such element.) Then, one assumes access to a quantum oracle, $\mathcal{O}_f$ (effectively a black box subroutine) that implements f thus:

$$\mathcal{O}_f \left|x\right\rangle = (-1)^{f(x)} \left|x\right\rangle . \tag{4.17}$$

A single application of this oracle is not enough to find the desired element, as the square of the amplitude of the desired element remains unchanged. The measurement probabilities are not immediately affected. Central to Grover's algorithm is the idea of amplifying the amplitude of the desired element. This is done by applying a sequence of operations that is repeated until the amplitude of the desired element is large enough for it to be most likely to be measured, while the other elements have their amplitudes reduced.

Let the state $\left|w\right\rangle$ be the winner state, a state with amplitude 1 for the desired element and 0 for all others. Then consider the state $\left|s\right\rangle$, which is a uniform superposition state, a state with equal amplitudes for all elements, trivially constructed by applying a Hadamard gate to each qubit in the canonical starting state $\left|0\right\rangle$. Define the state $\left|s'\right\rangle$ by subtracting the projection of $\left|w\right\rangle$ onto $\left|s\right\rangle$ from $\left|s\right\rangle$:

$$\left|s'\right\rangle = \left|s\right\rangle - \left\langle w|s\right\rangle \left|w\right\rangle . \tag{4.18}$$

These two orthogonal states form a basis of a two-dimensional subspace of the greater Hilbert space. This permits a perspicuous visualisation of the algorithm, as in fig. 4.2. The uniform superposition state $\left|s\right\rangle$ serves as the starting point for the algorithm, and is expressible as

$$\left|s\right\rangle = \cos(\theta) \left|s'\right\rangle + \sin(\theta) \left|w\right\rangle , \tag{4.19}$$

where $\theta = \arcsin \left\langle s|w\right\rangle = \arcsin(1/\sqrt{N})$.

Applying the oracle on $\left|s\right\rangle$ leaves its $\left|s'\right\rangle$ component unchanged, but flips the sign of the $\left|w\right\rangle$ component. This results in the state $\left|\psi\right\rangle = \cos(-\theta) \left|s'\right\rangle + \sin(-\theta) \left|s'\right\rangle$, which can be seen as reflection of $\left|s\right\rangle$ in the $\left|s'\right\rangle$ direction.

Next, the state $|\psi\rangle$ is reflected about the initial $|s\rangle$ state, resulting in the state $|\psi'\rangle = \cos(3\theta)\,|s'\rangle + \sin(3\theta)\,|w\rangle$. Reflection thus is achieved by the diffusion operator

$$D = H^{\otimes n} S_0 (H^{\otimes n})^{-1} = H^{\otimes n} S_0 H^{\otimes n}, \tag{4.20}$$

where $S_0 = 2\,|0\rangle\langle 0| - I$ is the reflection operator about the $|0\rangle$ state, that is an operator that flips the sign of all but the $|0\rangle$ component.

The product of the oracle and the diffusion operator defines the Grover operator, which is simply applied until the amplitude of the $|w\rangle$ is sufficiently amplified. After k iterations, the state is $|\psi_k\rangle = \cos((2k+1)\theta)\,|s'\rangle + \sin((2k+1)\theta)\,|w\rangle$. Measuring the correct state has probability $\sin^2((2k+1)\theta)$. Therefore, $k \approx \pi/4\theta$ iterations should be completed. Assuming large N, for a short list would not warrant the use of Grover's algorithm, $\theta = \arcsin(1/\sqrt{N}) \approx 1/\sqrt{N}$, and so $k \approx \pi\sqrt{N}/4$.

For lists with more than a single desired element, a similar reasoning will lead to the same algorithm, but instead with $k \approx \pi/4 \cdot \sqrt{N/M}$, where M is the number of solutions to $f(x) = 1$ [68].

4.3.3 Amplitude amplification

Amplitude amplification can be considered a generalisation of Grover's algorithm. Instead of a single oracle, let the partitioning of the state space be given by a Hermitian projector P, whose image will be the space of states to amplify. Then, for some initial state $|\psi\rangle$, it is decomposed into the orthogonal components

$$|\psi\rangle = \sin(\theta)\,|\psi_0\rangle + \cos(\theta)\,|\psi_1\rangle\,, \tag{4.21}$$

where $|\psi_1\rangle = P\,|\psi\rangle$ and $|\psi_0\rangle = |\psi\rangle - |\psi_1\rangle$, effectively the projections onto the image and kernel of P. Clearly, the angle θ is given by $\arcsin(|P\,|\psi\rangle\,|)$.

The Grover operator is now given by $G = -S_\psi S_P$ where

$$S_\psi = I - 2\,|\psi\rangle\,\langle\psi| \tag{4.22}$$
$$S_P = I - 2P, \tag{4.23}$$

such that S_ψ being analogue to the diffusion operator D and S_P being the oracle operator.

Following the same reasoning as in the previous section, the state after k iterations is given by

$$G^k\,|\psi\rangle = \sin((2k+1)\theta)\,|\psi_1\rangle + \cos((2k+1)\theta)\,|\psi_0\rangle\,, \tag{4.24}$$

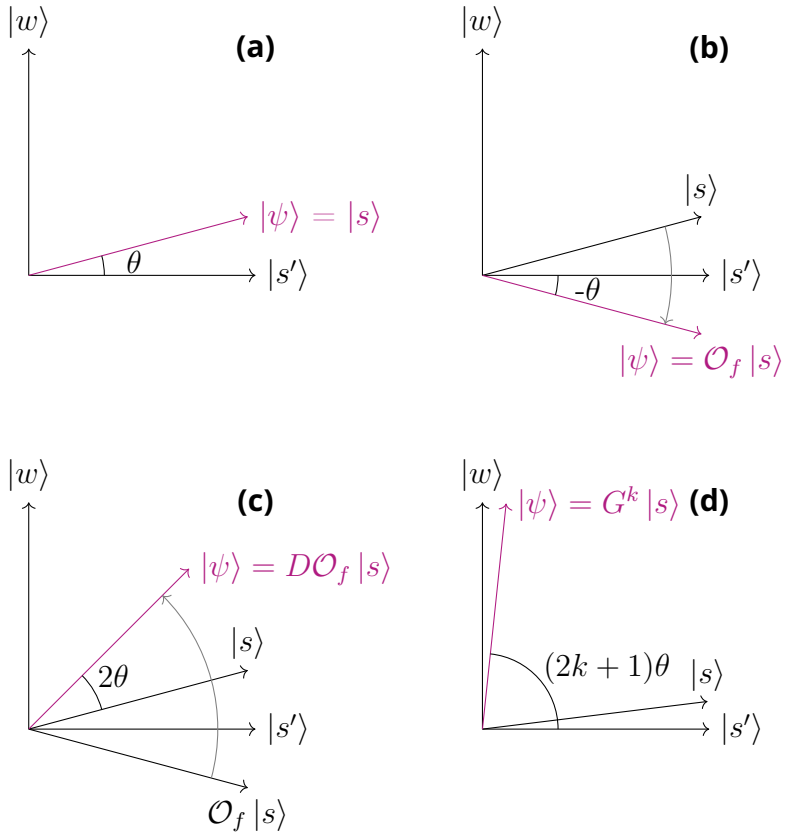


Figure 4.2: Grover's algorithm visualised. (a) The initial uniform superposition state $|s\rangle$ is prepared, which can be seen as a linear combination of $|w\rangle$ and $|s'\rangle$, forming an angle θ to the s'-axis. (b) The oracle $\mathcal{O}_f$ is applied to $|s\rangle$, flipping the sign of its $|w\rangle$ component, inverting the angle θ. (c) The diffusion operator D is applied, reflecting the state about the initial state and towards the goal, resulting in a state with an angle 3θ to the w-axis. (d) After repeating the previous two steps a k times, the angle is $2k + 1\theta$, and if k is chosen wisely, this means that the system is in a state close to the desired state $|w\rangle$, such that measuring the system will likely result in $|w\rangle$.

meaning that $k \approx \frac{\pi}{4\theta}$ will result in a state close to $|\psi_1\rangle$.

Amplitude amplification can be used to speed up Grover search by using an informed prior rather than a uniform prior. Furthermore, it is useful as a subroutine in other algorithms, such as finding the number of 'good' states for a Grover search [80], quantum Monte Carlo methods [81] and some bandit algorithms to be discussed in what follows.

4.3.4 Amplitude estimation

As with amplitude amplification, amplitude estimation considers states that are decomposed into a superposition of two states, and, as the name suggests, estimates the amplitude of one of the states. Given a state, or more generally the algorithm, $\mathcal{A}$ with which it is generated,

$$\mathcal{A}\,|0\rangle = \sqrt{a}\,|\psi_1\rangle + \sqrt{1-a}\,|\psi_0\rangle\,, \tag{4.25}$$

where $|\psi_1\rangle$ is a state of interest and $|\psi_0\rangle$ its orthogonal complement, the goal is to estimate its amplitude $a = |\langle\psi_1|\psi_1\rangle|^2$.

With its original formulation in [80], it was proven that the amplitude can be estimated with an additive error of

$$\epsilon \leq 2\pi\frac{\sqrt{a(1-a)}}{t} + \frac{\pi^2}{t^2} \tag{4.26}$$

with probability at least $8/\pi^2 (\approx 81.06\%)$ using t calls to the algorithm $\mathcal{A}$. The probability can be increased to $1-\delta$, requiring $O(\log(1/\delta))$ calls to $\mathcal{A}$ [81]. Later variants have been proposed to reduce the qubits needed and circuit depths [82, 83, 84], making it more feasible for NISQ devices, while still achieving similar asymptotic error bounds.

4.3.5 Quantum Monte Carlo

Monte Carlo methods have been a powerful tool to analyse the behaviour of quantum mechanical systems, where probabilistic methods are natural to describe probabilistic physics [85, 86, 87]. On the other hand, more in line with the scope of this report, recent advances in quantum computing have opened up a new avenue for the intersection of quantum mechanics and Monte Carlo methods.

In the problem of estimating the mean of a random variable without assumptions of its distribution or properties, the additive error can be bounded by Chebyshev's inequality as

$$P(|\hat{\mu} - \mu| \geq \epsilon) \leq \frac{\sigma^2}{n\epsilon^2}, \tag{4.27}$$

where $\hat{\mu}$ is the sample mean, μ is the true mean, σ is the standard deviation and n is the number of samples. Consequently, there is a need of quadratically many samples to achieve a given error, which for example means that estimating the mean of a random variable with a standard deviation of 1 with four decimals' accuracy and a certainty of 99.9% would require 10^9 samples. Moreover, this is provably optimal asymptotically [88].

Generalising amplitude estimation, in [81], a 'near-quadratic' speed-up of Monte Carlo methods was achieved by using amplitude estimation to estimate the mean of a random variable encoded by quantum algorithms. Given an algorithm $\mathcal{O}$ whose measurement outputs are assigned real values such that $v(\mathcal{A})$ is a random variable with mean μ and variance σ^2, it is proved that approximating μ with an additive error of ϵ can be achieved with only $\tilde{O}(\sigma/\epsilon)$ calls to $\mathcal{A}$ and its inverse, which is a near-quadratic speed-up over the classical case[12].

The simpler version on which the general builds, $v(\mathcal{A})$ is assumed to lie in the interval $[0, 1]$. Thus, the value can be encoded in a single qubit by through a unitary

$$U|x\rangle|0\rangle = |x\rangle\left(\sqrt{1-\phi(x)}\,|0\rangle + \sqrt{\phi(x)}\,|1\rangle\right), \tag{4.28}$$

where $\phi(x)$ is the output of the algorithm were x to be measured. Thence, the amplitude of the last qubit is simply estimated using amplitude estimation, as described in section 4.3.4, using an appropriate number of iterations and repetitions. The initial state for the amplitude estimation is set to

$$|\psi\rangle = U(\mathcal{O} \otimes I)|0\rangle. \tag{4.29}$$

For these bounded random variables in particular, $O(1/\epsilon)$ iterations suffice to achieve an additive error of ϵ, repeating the whole procedure $O(1/\log(\delta))$ times to achieve a certainty $1-\delta$ [81].

4.4 Limitations of near-term quantum hardware

Quantum hardware has been physically realised and even demonstrated to outperform classical computers in very contrived situations [89, 90, 91], but the hardware is still limited. The hardware is limited in the number of qubits, the connectivity between the qubits, and the noise and decoherence

[12]The $\tilde{O}$ notation is the same as regular big-O notation, but with (poly-) logarithmic factors omitted, e.g., $n^2\log(n)^2 = \tilde{O}(n^2)$.

of the qubits. It is believed that quantum hardware will continue to improve and eventually perform demanding algorithms like Shor's algorithm on numbers too great for classical computers to factorise. Once enough qubits are available and error rates are low enough, error correction schemes can be implemented. Then, the hardware will be able to perform these provably faster algorithms. Still, the era dubbed NISQ (noisy intermediate-scale quantum) is the first step, and to make use of the hardware, the inherent noise and its consequences must be understood. Algorithms must take the following limitations into consideration.

4.4.1 Quantum channels and decoherent noise

The unitary gates from section 4.2 are an idealisation, and in reality, the gates are not perfect. A more realistic description of the gates is given by quantum channels, which can take noise into account. In an unmeasured, ideal quantum system, the state evolution is veritably unitary, but as the system in practice does interact with the environment, this can be modelled as a quantum channel in which classical probabilistic noise is introduced.

Take the X-gate as an example. If it fails to apply with probability p, the resulting density operator ρ' can be expressed as

$$\rho' = p\rho + (1-p)X\rho X^\dagger, \tag{4.30}$$

where ρ is the initial density. Such a channel has eigenvalues p and $1-2p$, where I and X share the former and Y and Z the latter. Consequently, states with Y- or Z-components will have mixing introduced. Geometrically, this can be interpreted as a contraction of states on the Bloch sphere towards the centre, illustrated in fig. 4.3.

Any physical gate will suffer some such noise, and so the tendency will be for states to degenerate towards the mixed centre of the Bloch sphere (or its higher-dimensional analogue). Furthermore, measurements, letting a qubit idle when operating on others and even the preparation of the initial state will suffer from decoherence.

This noise is very hard to avoid, as controlling the qubits necessarily requires some interaction with the environment. Additionally, to operate on multiple qubits, a connection between them is required, which too will introduce noise. Nature tends to work against large-scale quantum systems, made evident by the absence of quantum effects in regular life.

Due to the multiplicative effect of noise, the overall degeneracy will increase exponentially with the number of operations. This means that there will be limits to the number of operations that can be performed before the system becomes unusable, assuming no error correction is done.

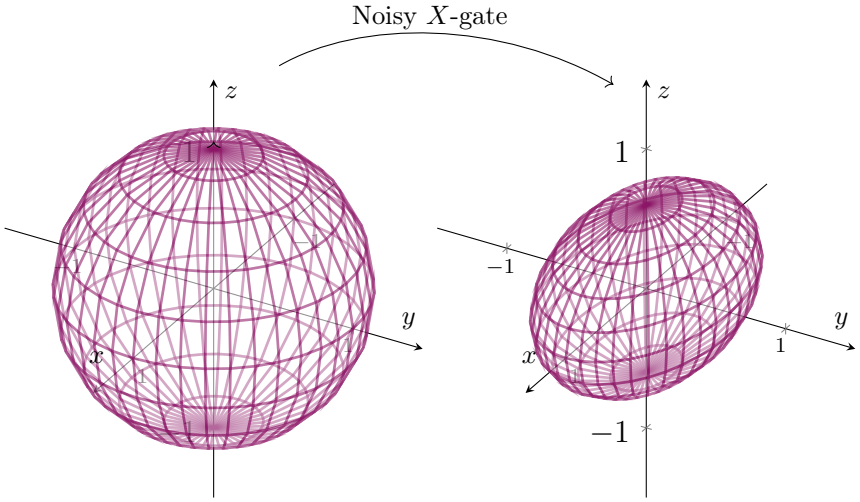


Figure 4.3: Illustration of applying a noisy X-gate on the Bloch sphere. The left plot shows the set of all pure states, the Bloch sphere. On the right, the same set of states is shown after the application of a noisy X-gate with probability $p = 0.2$ of failing. There, the y- and z-axes are contracted towards the centre by a factor $1 - 2p = 0.6$, introducing mixing.

4.4.2 Coherent noise

There may also be systematic errors in the unitary gates applied. This kind of noise, known as coherent noise does not need quantum channels to be modelled, but may rather be modelled simply by slightly different unitary gates. Here too, the X-gate can serve as an example. As gates are unitary evolutions resulting from a Hamiltonian acting on the system, any deviance in the energies of the Hamiltonian or length of the time interval will cause a different evolution. If it is not calibrated correctly, the effective operation will be a rotation that slightly deviates from the intended half-turn around the Bloch sphere. When a gate like the X-gate is applied many times, even small errors will add up, which could cause an overall rotation of the state by a significant angle.

Figure 4.4 shows the effects of both coherent and decoherent noise on the measurement of a qubit after applying several noisy X-gates. Clearly, information is quickly lost as the quantum state ends up different from what is expected and as the system becomes more and more mixed, losing quantum properties and instead obeying classical probabilities. Even though current hardware suffers much less noise than the figure shows, NISQ algorithms must nonetheless be shallow, meaning that the number of

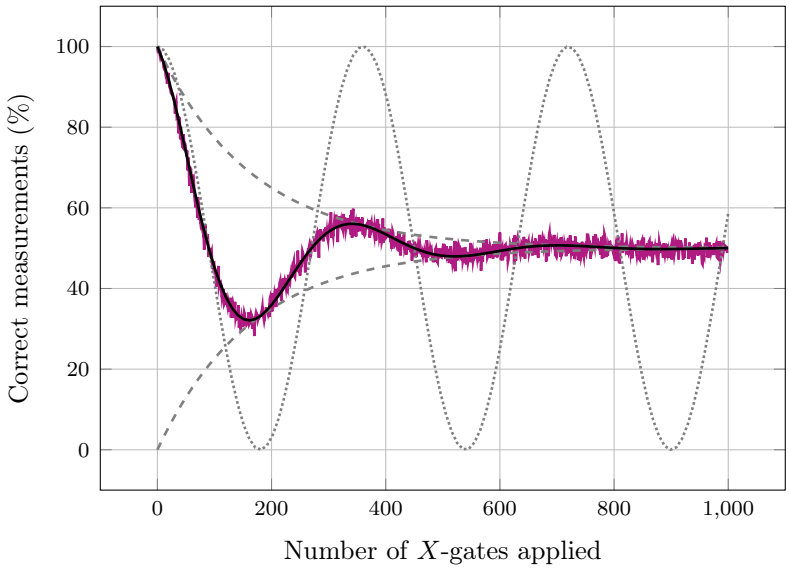


Figure 4.4: Proportion of correct measurements of a qubit after applying several noisy X-gates. The error in the rotation is 1°, while the probability of failing to apply the gate is $p = 0.3\%$. For each number of gates, the measurement is repeated 1000 times. The coherent error causes a sinusoidal behaviour, in which 180 rotations cause the overall rotation to be completely opposite. The decoherent error causes an exponential dampening, as the state becomes more and more mixed. The wiggly pattern stems from the shot noise, simply classical randomness from the probabilistic measurement with a finite number of samples.

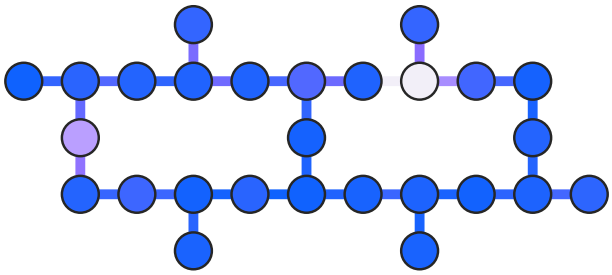


Figure 4.5: Qubit connectivity and error rates in IBM's Montreal 27-qubit Falcon r4 chip [92]. Brighter coloured nodes indicate higher X-error rates, while brighter edges indicate higher CNOT-error rates.

gates applied before measurement must be low. There are more sources of noise than the two included in the figure, and having to deal with multiple qubits certainly does not help.

4.4.3 Qubit counts and connectivity

Another limiting factor is the number of qubits. Current hardware has around 10 to 100, which though still may be enough to express states too large for classical computers, is not enough to perform the most demanding algorithms. Recent estimates find that millions of noisy qubits would be needed to break RSA encryption [93].

The transpilation of the circuit to what the hardware can actually do is also an important factor. Consider for example IBM's Falcon r4 chip, which has 27 qubits and a connectivity graph as shown in fig. 4.5. Qubits are generally only linked to two other qubits. Moreover, only a basis set of gates is available, which in this case are the CNOT-, R_Z-, X- and $\sqrt{X}$-gates. All this means that qubits may have to be swapped, often many times, and that multiple gates may be needed where only was intended. Circuit depths are subsequently increased, which in turn exacerbates the effects of noise.

Chapter 5

Quantum bandits

Several formulations of the multi-armed bandit problem have been made for a quantum computing setting. As the central issue in bandit problems lies in sample efficiency rather than computational difficulties, quantum computers offer little advantage assuming classical bandits. However, by granting some sort of superposition queries, means can be estimated more efficiently, and so regrets may be reduced, or best arms found more quickly.

Querying in superposition may at first appear to remove any real-world relevance; administering medications to patients can certainly not be done in superposition. However, with the training for reinforcement learning primarily being done in simulation, it is conceivable that the theory of quantum bandits may be applied to the learning of agents that are trained on quantum hardware and subsequently deployed to the real world.

5.1 Quantum upper confidence bounds

A way to quantise the bandit problem is to assign to each arm a quantum oracle. For each arm $a \in \mathcal{A}$, a bandit oracle can be defined as

$$\mathcal{O}_a : |0\rangle \otimes |0\rangle \mapsto \sum_{\omega \in \Omega} \left(\sqrt{P_a(\omega)} \, |\omega\rangle \otimes |X_a(\omega)\rangle \right), \tag{5.1}$$

where ω is some sample space on which X_a is a random variable with probability measure P_a. Applying the oracle to the state $|0\rangle$ produces a superposition of all possible outcomes of the random variable X_a, such that measuring the second register will produce a sample from X_a. In this way, this quantum version of the bandit problem can be reduced to the classical case, but by maintaining the superposition, quantum advantages can be gained.

As with classical arms, the agent decides for each step in the bandit problem which oracle to invoke, trying to minimise the cumulative regret,

where the means here are defined as

$$\begin{aligned}\mu_a &= \sum_{\omega\in\Omega} P_a(\omega)X_a(\omega) \\ &= \langle 00|\, \mathcal{O}_a^\dagger(I\otimes Z)\mathcal{O}_a)\,|00\rangle\end{aligned} \tag{5.2}$$

In [15], an algorithm for bounded-value arms achieving $O(n\log T)$ regret was proposed, n being the number of arms. For bounded variances, the regret is $O(n\ \mathrm{poly}(\log T))$, which is still substantially better than $\Omega(\sqrt{nT})$ minimax regret for classical bandits. Only the bounded-value variant of the algorithm is described here for its simplicity and easier simulation. What is here denoted simply as QUCB is in [15] called QUCB_1, while the variant for bounded variances is denoted by QUCB_2.

The algorithm proposed is essentially a UCB-like algorithm[13], where QMC (as described in section 4.3.5) is used to estimate means more efficiently. Because QMC estimates are only produced after a set number of consecutive quantum queries, the algorithm must cleverly decide for how many steps to pull each arm, in addition to which arm to pull, before commencing a QMC session.

As listed in algorithm 5.1, the QUCB algorithm first runs a preliminary phase where the means are estimated using QMC with a fixed number of samples, after which it iteratively pulls the arms with the highest confidence bounds, after which the confidence bound is halved and the number of QMC samples to be used for that arm is doubled. A confidence parameter δ is used to determine the number of QMC samples to use, satisfying $|\hat{\mu}_i - \mu_i| \leq \mathrm{UCB}_i$ with probability $1-\delta$. The constant $C_1 > 1$ is only described existentially to give an upper bound to the number of QMC queries needed, and is a result of the big-O notation used to describe QMC convergence. How it should be set for implementation of the algorithm is not described in [15].

5.1.1 Proof of logarithmic regret

As presented in section 4.3.5, for variables in $Y \in [0,1]$ producible by quantum algorithms, QMC can estimate $\mathbb{E}[Y]$ with error $|\hat{\mu}-\mu| \leq \epsilon$ with probability $1-\delta$ using $\frac{C_1}{\epsilon}\log\frac{1}{\delta}$ queries or less to the oracle or its adjoint, for some universal constant $C_1 > 1$. That means that for each arm a,

$$|\hat{\mu}_a - \mu_a| \leq \mathrm{UCB}_a \quad \text{with probability} \quad 1-\delta, \tag{5.3}$$

for each QMC session in the algorithm.

[13] Q.v. section 2.3.4.

Algorithm 5.1: QUCB

Input: Set of arms $\mathcal{A}$, $\mathcal{O}_i$ as in eq. (5.1), T horizon, $0 < \delta \ll 1$

for $a \in \mathcal{A}$ **do**
  $\mathrm{UCB}_a \leftarrow 1$
  $N_a \leftarrow (C_1/\mathrm{UCB}_a)\log(1/\delta)$
  Estimate μ_a using QMC with N_a samples
while Total number of queries to the oracles is less than T **do**
  $a \leftarrow \mathrm{argmax}_a(\hat{\mu}_a + \mathrm{UCB}_a)$
  $\mathrm{UCB}_a \leftarrow \mathrm{UCB}_a/2$
  $N_a \leftarrow (C_1/\mathrm{UCB}_a)\log(1/\delta)$
  Update estimate of μ_a using QMC with N_a samples

Let S_a be the set of stages where arm a is pulled in the while-block of the algorithm, and let its cardinality be $K_a = |S_a|$. Then, each arm is pulled $(2^{K_a+1}-1)C_1 \log\frac{1}{\delta}$ in the second phase. In the preliminary phase, the total number of queries is

$$C_1 \log\frac{1}{\delta} \sum_{a\in\mathcal{A}} 2^{K_a+1} = T. \tag{5.4}$$

Further, using Jensen's inequality,

$$\sum_{a\in\mathcal{A}} 2^{K_a+1} \geq k 2^{1/k \sum_{a\in\mathcal{A}}(K_a+1)}, \tag{5.5}$$

it follows that

$$\sum_{a\in\mathcal{A}} K_a \leq k \log_2\left(\frac{T}{kC_1 \log\frac{1}{\delta}}\right) - k, \tag{5.6}$$

which considering the first k rounds, gives that the total number of QMC sessions is

$$N_{\mathrm{QMC}} \leq k \log_2\left(\frac{T}{kC_1 \log\frac{1}{\delta}}\right). \tag{5.7}$$

The probability of all QMC queries satisfying the error bound of eq. (5.3)

is

$$
\begin{aligned}
1 - \left(\bigcup_{i=1}^{N_{\mathrm{QMC}}} P(\text{Query } i \text{ fail}) \right) &\leq 1 - \sum_{i=1}^{N_{\mathrm{QMC}}} P(\text{Query } i \text{ fail}) \\
&\leq 1 - \sum_{i=1}^{N_{\mathrm{QMC}}} \delta \\
&= 1 - N_{\mathrm{QMC}}\delta \\
&\leq k \log_2 \left(\frac{T}{kC_1 \log \frac{1}{\delta}} \right) \delta.
\end{aligned} \tag{5.8}
$$

Denote this event by E. For the arm a chosen in the arguments of the maxima at line 6 of the algorithm,

$$
\hat{\mu}_a + \mathrm{UCB}_a \geq \hat{\mu}^* + \mathrm{UCB}^*, \tag{5.9}
$$

and given E, it generally holds that

$$
\hat{\mu} \leq \mu + \mathrm{UCB} \quad \text{and} \quad \mu \leq \hat{\mu} + \mathrm{UCB}. \tag{5.10}
$$

Hence, the suboptimality gap of arm a is bounded by

$$
\begin{aligned}
\Delta_a &:= \mu^* - \mu_a \\
&\leq (\hat{\mu}^* + \mathrm{UCB}^*) - (\hat{\mu}_a - \mathrm{UCB}_a) \\
&\leq (\hat{\mu}_a + \mathrm{UCB}_a) - (\hat{\mu}_a - \mathrm{UCB}_a) \\
&= 2\mathrm{UCB}_a.
\end{aligned} \tag{5.11}
$$

Notably, this holds regardless of what stage of the algorithm the arm is pulled in, so the last, most precise estimate of μ_a is used, wherein $\mathrm{UCB}_a = 2^{1-K_a}$. Consequently, the regret contribution of arm a is bounded by

$$
\begin{aligned}
T_a\Delta_a &\leq 2T_a\mathrm{UCB}_a. \\
&= 2(2^{K_a+1}C_1 \log \frac{1}{\delta})2^{1-K_a} \\
&= 2^3 C_1 \log \frac{1}{\delta}.
\end{aligned} \tag{5.12}
$$

Given E, it is hence clear that

$$
\sum_{a \in \mathcal{A}} T_a\Delta_a \leq 8(k-1)C_1 \log \frac{1}{\delta}. \tag{5.13}
$$

In the case where E does not occur, the regret contribution can be bounded by

$$
\sum_{a \in \mathcal{A}} T_a\Delta_a \leq T \max_{a \in \mathcal{A}} \Delta_a \leq T, \tag{5.14}
$$

as the arms are assumed to be Bernoulli.

The regret can hence be bounded by

$$\begin{aligned}
R_T &= \mathbb{E}\left[\sum_{a\in\mathcal{A}} T_a\Delta_a\right] \\
&= \mathbb{E}\left[\mathbb{E}\left[\sum_{a\in\mathcal{A}} T_a\Delta_a \;\middle|\; E\right]\right] \\
&= P(E)\mathbb{E}\left[\sum_{a\in\mathcal{A}} T_a\Delta_a \;\middle|\; E\right] + P(\neg E)\mathbb{E}\left[\sum_{a\in\mathcal{A}} T_a\Delta_a \;\middle|\; \neg E\right] \\
&\leq (1-P(\neg E))\left(8(k-1)C_1\log\frac{1}{\delta}\right) + P(\neg E)T \\
&\leq 8(k-1)C_1\log\frac{1}{\delta} + P(\neg E)T. \\
&\leq 8(k-1)C_1\log\frac{1}{\delta} + k\delta\log_2\left(\frac{T}{kC_1\log\frac{1}{\delta}}\right)T.
\end{aligned} \tag{5.15}$$

With this, setting $\delta = 1/T$, it is obtained that

$$\begin{aligned}
R_T &\leq 8(k-1)C_1\log\frac{1}{\delta} + k\log_2\left(\frac{T}{kC_1\log\frac{1}{\delta}}\right) \\
&\leq 8(k-1)C_1\log T + k\log_2\left(\frac{T}{kC_1\log T}\right) \\
&\leq 8(k-1)C_1\log_2 T + k\log_2 T \\
&= (8(k-1)C_1 + k)\log_2 T \\
&= O(k\log T).
\end{aligned} \tag{5.16}$$

Quod erat demonstrandum.

5.1.2 Implementing quantum upper confidence bounds

To translate algorithm 5.1 into something that can be either run on a quantum computer or simulated on a classical computer, the arm-oracles need to be made available and QMC needs to be implemented, building on amplitude estimation[14].

[14] Qq.v. sections 4.3.4 and 4.3.5

Arm-oracles

The oracles of eq. (5.1) can be implemented as quantum gates by first constructing their matrix representation. With Bernoulli variables and considering the simplest case of $\Omega = \{0, 1\}$ and associating only $0 \in \Omega$ with the reward, the first row of the matrix representation of the oracle is

$$\sqrt{p}\,|0\rangle\,|1\rangle + \sqrt{1-p}\,|1\rangle\,|0\rangle = (0, 0, \sqrt{p}, \sqrt{1-p}). \tag{5.17}$$

They are, however, not well-defined unitary operators, leaving several rows open for free choice. These can be easily filled by using Gram-Schmidt orthogonalisation on the remaining rows. More explicit solutions are of course possible, but these become more intricate with different reward distributions and Ω.

As a result, the oracle initialisation can be implemented as is listed in algorithm 5.2, which is easily done in Qiskit.

Algorithm 5.2: Oracle initialisation for a Bernoulli arm

Input: Arm probability p
Output: Quantum circuit
1 Initialise first row $(0, 0, \sqrt{p}, \sqrt{1-p})$
2 Complete matrix using Gram-Schmidt orthogonalisation
3 **return** Quantum circuit implementing matrix

QMC

Thankfully, amplitude estimation is a well-studied problem in quantum computing, and there are several implementations available. For example, in the Qiskit library, the `AmplitudeEstimation` class can be used to perform amplitude estimation on a quantum circuit with a given number of qubits. It assumes a `EstimationProblem` object is initialised with the quantum algorithm to be estimated and the objective qubit(s) specified.

What is more, with binary rewards encoded in the final qubit, QMC can be implemented without the extra qubit that is described in section 4.3.5. Instead, amplitude estimation can be performed directly on the oracles as they are implemented in algorithm 5.2. Additionally, since the number of iterations is described by QUCB, it is fed directly into the amplitude estimation simulation. The MLE estimate is automagically calculated by the `AmplitudeEstimation` class[15], which will be used to update the QUCB estimate. The number of oracle calls is also automatically tracked by the

`AmplitudeEstimation` class, which is incremented by 1 to account for the initialisation of the oracle before amplitude estimation is performed. In total, the QMC subroutine is implemented as described by algorithm 5.3.

Algorithm 5.3: QMC for a Bernoulli arm oracle

Input: Oracle $\mathcal{O}$, number of iterations N, confidence δ
Output: Estimate of p with confidence $1-\delta$, number of oracle calls

Initialise `EstimationProblem` with oracle $\mathcal{O}$ and objective qubit 0
Number of shots $N_S \leftarrow \log(1/\delta)$
Number of qubits $N_Q \leftarrow \lceil \log_2 N \rceil$
Initialise `AmplitudeEstimation` with number of shots N_S and number of qubits N_Q
Estimate p with `AmplitudeEstimation` on the `EstimationProblem` object
$\hat{p} \leftarrow$ MLE result of `AmplitudeEstimation`
$T \leftarrow$ number of iterations used by `AmplitudeEstimation`
return $\hat{p}, T+1$

QUCB simulation

With oracles and QMC being available, the QUCB algorithm can be implemented easily. First, a set of oracles is initialised for each arm, and the algorithm is run for a given number of iterations. The number of oracle calls is tracked, and the algorithm is stopped when the number of oracle calls exceeds the number of iterations. Regret contributions are tracked and output after the simulation. The simulation code thus appears as in algorithm 5.1.

5.2 Other quantum bandit advances

Heavy-tailed bandits

The more difficult problem of reward distributions with heavy tails is considered in [94]. Using a similar upper confidence bound approach as QUCB, their algorithm shows quantum advantage for bandits where the

[15] Canonical amplitude estimation only returns probabilities for two to the power of the number of qubits many states. From those probabilities, the MLE estimate for the mean is calculated as the mean most likely to have produced the observed distribution.

Algorithm 5.4: QUCB simulation with a set of Bernoulli arms

Input: Set of arm probabilities $\{p_1, \ldots, p_k\}$,
number of iterations T, confidence δ
Output: Regret at each iteration

Initialise $t \leftarrow 0$
for $a \in \{1, \ldots, k\}$ **do**
    Initialise oracle $\mathcal{O}_a$
    Initialise $\text{UCB}_a \leftarrow 1$
    Initialise $N_a \leftarrow (C_1/\text{UCB}_a)\log(1/\delta)$
    Estimate $\hat{\mu}_a$ with algorithm 5.3
    Update t with the number of oracle calls
    Log regret
while $t < T$ **do**
    $a \leftarrow \text{argmax}_a(\hat{\mu}_a + \text{UCB}_a\ )$
    $\text{UCB}_a \leftarrow \text{UCB}_a/2$
    $N_a \leftarrow (C_1/\text{UCB}_a)\log(1/\delta)$
    Update estimate $\hat{\mu}_a$ with new estimate from algorithm 5.3
    Update t with the number of oracle calls
    Log regret
Discard any regret from excess turns
return regrets $R_1, \ldots, R_T$

assumption of bounded $(1+\epsilon)$-moments for some $0 < \epsilon \leq 1$ is made, a weaker assumption than the bounded value or bounded variance assumptions made in [15]. Specifically, a polynomially improved regret is achieved, and the improvement is verified with experimental simulations.

Quantum contextual bandits

Variational methods are used in [95] to study the problem of quantum contextual bandits. There it is noted how quantum computers naturally encode continuous variables, and how this may be used to improve the performance of bandit algorithms. No quantum benefits are shown, but the potential is made clear.

In [96], contextual linear bandits with quantum data were studied. A 'quantum context' is observed, from which a recommender system is designed. Similarly to the above QUCB algorithm, the rewards are assumed to come from quantum measurements.

Learning quantum states

The bandit framework is in [97] used to study the problem of learning quantum states efficiently. There, the set of arms are some observables and the bandits are some fixed quantum state. Thence a copy of the state is prepared and measured with the observable corresponding to the arm pulled, from which the measurement is interpreted as a reward. Several classical algorithms are considered for this problem, including UCB, and classical bounds are similarly proved for the regret. Because the quantum state is measured at each step, something like QMC is not possible, so quantum speed-ups are not possible.

Best arm identification

In [98], an algorithm based on amplitude amplification is proposed and is shown to find the optimal arm with quadratically fewer queries than the best classical algorithm for classical bandits in the case of Bernoulli rewards. There is albeit a significant drawback: the probability of the correct arm being suggested can not be set arbitrarily high, but is instead given by the ratio of the best arm's mean to the sum of the means of all arms. This greatly limits the usefulness of the algorithm.

They assume access to an oracle $\mathcal{O}_e$ that encodes the reward probabilities

of the arms,

$$\mathcal{O}_e : |a\rangle \otimes |0\rangle \mapsto |a\rangle \otimes \sum_{\omega \in \Omega} \sqrt{P_a(\omega)}\, |Y(\omega)\rangle\,, \tag{5.18}$$

where a is an arm, ω some state in the sample space Ω, $P_a(\omega)$ the probability measure thereon, from which the random variable $Y(\omega)$ is drawn, some internal state. Remark that with the first qubit register being placed in superpositions, all arms can effectively be queried simultaneously. For a given arm a and the internal state $|y\rangle$, the reward is determined by some function $f(a, y) \to \{0, 1\}$, accessed through the phase oracle $\mathcal{O}_f$,

$$\mathcal{O}_f : |a\rangle \otimes |y\rangle \mapsto (-1)^{f(x,y)}\, |a\rangle \otimes |y\rangle\,. \tag{5.19}$$

For such oracle bandits where all arms are in a way pulled simultaneously, regret minimisation is no longer really a valid objective, as all arms are in a sense pulled simultaneously. Instead, the problem is to find a strategy that maximises the probability of finding the optimal arm with as few applications of $\mathcal{O}_e$ as possible.

The authors of [99] propose a more sophisticated algorithm, improving the results of [98] by allowing the probability of finding the optimal arm to be set arbitrarily high. Theirs is also quadratic speed-up over the best classical algorithm, but is more complicated and requires a quantum computer with more qubits.

Chapter 6

Quantum reinforcement learning

Four fundamental ways of combining machine learning and quantum computing are commonly described in the literature [100]. One can differentiate between the type of data being processed and the way of processing these data, giving the following table:

		Computing device	
		Classical	*Quantum*
Data	*Classical*	CC	CQ
	Quantum	QC	QQ

CC Classical data being processed on classical computers is classical machine learning. Though not explicitly linked to quantum computing, there are some ways in which quantum computing influences classical machine learning, such as the quantum-inspired application of tensor networks [101].

QC Using classical machine learning for quantum data includes improving quantum computers' general performance. For example, with machine learning algorithms, the variance of the measurements can be reduced [71]. Alternatively, advanced machine learning models like neural networks can be employed to describe quantum states more efficiently.

CQ How to use quantum computers to process classical data is the main topic of this thesis and is what will be meant when quantum machine learning (QML) is mentioned. QML concerns itself with how to

improve classical machine learning, be it in terms of being easier to train, requiring less data or delivering better predictions. Quantum algorithms are most often advertised with speed-ups contra classical algorithms, often exponentially so as with Shor's algorithm. In the fault-tolerant setting, speed-ups can be achieved using support-vector machines with discrete logarithms [102] or fast quantum procedures for linear algebra, such as using HHL to invert matrices in linear regression [103]. However, whether NISQ-era quantum computers can provide any benefits for machine learning is still an open question. To this end, the study of variational circuits as machine learning models has garnered much attention [104], which is the topic to be discussed in what follows.

QQ Lastly, using quantum computing for handling quantum data is another way to combine quantum computing and machine learning. Here, quantum data can have two meanings. Either it can be data from quantum measurements, or it can be data that is already encoded in quantum states [105]. For example, in the second sense, quantum machine learning could be used on the quantum state produced by a quantum chemical simulation. Inputting quantum states natively is not trivial, and daisy-chaining the data generation and data processing could lead to a deep circuit. QQ is therefore less of immediate interest than QC is. There is obviously a large overlap with CQ as the data is quantum once encoded into the quantum computer, but as will be made clear, encoding is such a big part of CQ that results thence are not necessarily applicable to QQ.

6.1 Variational quantum algorithms

Variational quantum algorithms (VQAs) are envisioned as the most likely candidate for quantum advantage to be achieved in the NISQ-era. Summarised in fig. 6.1, the central idea is to run a parametrised circuit, evaluate a cost function and optimise it classically. Running the same circuit many times with different parameters and inputs in a classical-quantum-hybrid fashion, rather than a complete quantum implementation, means that the quantum operations can be shallow enough for the noise and decoherence to be manageable, while still potentially offering a speed-up over classical algorithms.

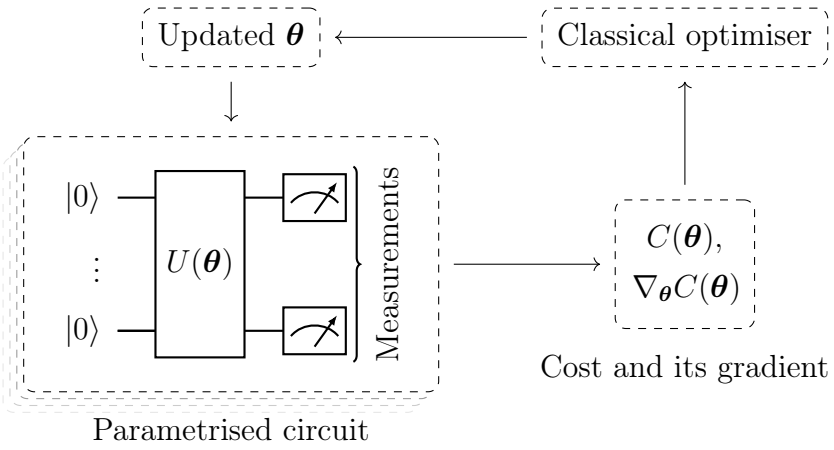


Figure 6.1: The general structure of a variational quantum algorithm.

6.1.1 Design

Generally, VQAs start with defining a cost function, depending on some input data (states) and the parametrised circuit, to be minimised with respect to the parameters of the quantum circuit. For example, the cost function for the variational quantum eigensolver (VQE) is the expectation value of some Hamiltonian, the energy of a quantum mechanical system such as different molecules. The cost function should be meaningful in the sense that the minimum coincides with the optimal solution to the problem, and that lower values generally imply better solutions. Additionally, the cost function should be complicated enough to warrant quantum computation by not being easily calculated on classical hardware, while still having few enough parameters to be efficiently optimised [106].

Important to the success of a VQA is the choice of the quantum circuit. The circuit selected is known as the ansatz, and there is a plethora of different ansätze that can be chosen. An example is hardware-efficient ansätze, which, as the name implies, is a general term used for ansätze designed in accordance with the hardware properties of a given quantum computer, taking, for instance the physical gates available into account and minimising the circuit depth. Other ansätze may be problem-specific, like the quantum alternating operator ansatz used for the quantum approximate optimisation algorithm (QAOA) (both sharing the QAOA acronym, confusingly), while others are more general and can be used to solve a variety of problems. It is believed that the development of better ansätze for specific applications will be important for the success of VQAs in the future [106].

6.1.2 Optimisation, gradients and barren plateaus

The optimisation of the cost function is often done with gradient descent methods. To evaluate the gradient of the quantum circuit with respect to the parameters, the very convenient parameter-shift rule can be used [107]. Though appearing almost as a finite difference scheme, relying on evaluating the circuit with shifted parameters, it is indeed an exact formula. Not having to rely on finite differences is a major advantage, as the effects of small changes in parameters would quickly be drowned out in noise. Furthermore, it may be used recursively to evaluate higher order derivatives, which allows the usage of more advanced optimisation methods like the Newton method that requires the Hessian, though this requires more circuit evaluations.

Once the gradient is known, the parameters can be updated with gradient descent methods. This leverages the well-developed toolbox of classical optimisation methods. These can be adapted to the quantum setting, for example by adjusting the number of circuit evaluations rather than the step-sizes [108]. However, the loss landscape will generally not be convex [109], so convergence is not necessarily guaranteed.

A major obstacle with VQAs is that the gradients of the cost function can be exponentially small, a phenomenon known as barren plateaus. Barren plateaus prohibit the optimisation from converging, as evaluating the gradient would need exponentially many function calls. The ansatz and parameter initialisation seem to be the main culprits for the barren plateaus [110, 111]. Additionally, with noisy quantum hardware, 'conceptually different' but equally problematic noise-induced plateaus can occur [99]. It is therefore important to consider both the hardware and the ansatz when designing a VQA.

6.1.3 Applications and outlook

VQAs can be used to solve a variety of problems. A typical example is finding the ground state of a Hamiltonian for a molecule with VQE. Such problems are exponential in the particle count, and thus intractable on classical hardware for larger molecules, while the problem of evaluating the Hamiltonian on quantum hardware is typically polynomial. This is useful in chemistry, nuclear physics, condensed matter physics, material sciences and more. VQAs are also well suited for general mathematical problems and combinatorial optimisation, with another common example being QAOA for the max-cut problem.

Yet, despite the potential of VQAs, there are still many challenges to overcome. Despite having some inherent resilience to noise [106], noise and

decoherence will still be limiting factors. Error mitigating post-processing can be used to improve the performance of VQAs, but which methods to use, and their effectiveness requires further investigation [112]. The optimisation of the cost function is also a major challenge, as the gradients can vanish, so the choices of ansätze and the initialisation strategies are important.

6.2 Quantum neural networks

Quantum neural networks (QNNs) are simply an abstraction of parametrised quantum circuits with some sort of data encoding. As classical artificial neural networks have made classical machine learning into a powerful tool, QNNs are envisioned as a quantum counterpart, inheriting some classical theory, nomenclature and perhaps unfounded hype. The main goal of QNNs is to do what classical NNs do, but with some quantum advantage, be it in terms of generalisability, training required or something else.

The structure of most quantum neural networks follows classical feed-forward networks. Figure 6.2 shows the general circuit layout. In the first step (or layer), data is encoded into the qubits, typically using a method discussed in section 6.2.1. Next, the data is passed through a sequence of parametrised quantum gates which often can be interpreted as belonging to layers. Lastly, an output is produced by measuring the qubits, potentially with some post-processing. Thence, a cost function is calculated, and the parameters are updated.

In [113], it was shown that QNNs can have higher expressibility and be easier to train than comparable classical NNs. What is more, several particular architectures inspired by classical NNs have evinced advantages over their classical forerunners, as will be seen in section 6.2.3. Withal, any intrinsic quantum advantage is still to be proven, and there are challenges that must be overcome for QNNs to be useful in practice.

6.2.1 Data encoding

In order for quantum computers to use classical data, it must first be encoded in a way that is compatible with the quantum hardware. How this is done has major implications on both the computational performance and the model expressibility. While naïve techniques like basis encoding are possible and easy to understand, more complex procedures are often needed to achieve good performance. The four methods that will be discussed in this section are summarised in table 6.1.

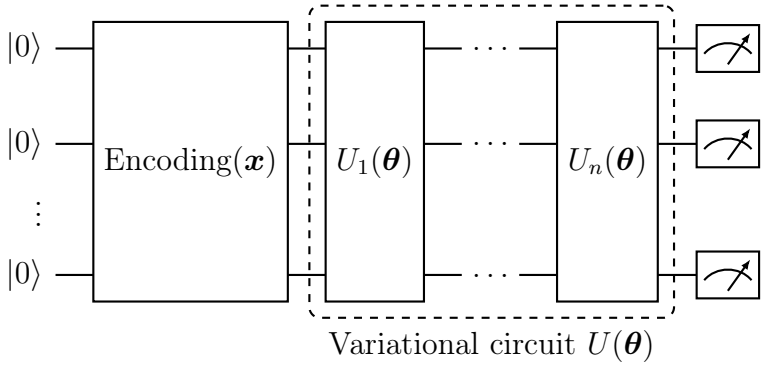


Figure 6.2: General structure of quantum neural networks. First, some data $\boldsymbol{x}$ is encoded into a state $|\psi(\boldsymbol{x})\rangle$ using some encoding strategy. Then, the state is transformed by a parametrised quantum circuit $U(\theta)$. This variational circuit needs to be decomposed into a sequence of gates $U_1, \ldots, U_n$, making the QNN structure more akin to the layered classical neural networks. These gates or layers do not need to use all qubits, but can be restricted to a subset, mimicking the classical concept of differently sized hidden layers. Finally, measurements are made and used to calculate the model output.

Table 6.1: Properties of different data encodings. Given an N-dimensional data set of M data points, the qubits needed is a lower bound for qubits required to represent the data, and circuit depth is the number of gates needed for the encoding algorithm. For basis encoding, $b(N) \geq N$ is the number of bits needed to represent an N-dimensional data point, for instance by using floating point representations of continuous data.

Encoding strategy	Qubits needed	Circuit depth	Hard to simulate classically
Basis encoding	$b(N)$	$O(N)$	No
Amplitude encoding	$\lceil \log_2 N \rceil$	$O(N)$	Yes
Angle encoding	N	$O(N)$	No
Second order angle encoding	N	$O(N^2)$	Yes? (Conjectured)

Basis encoding

The perhaps simplest way to encode data is to use the computational basis states of the qubits. This is done in the same way that classical computers use binary numbers. For example, some data x can be expressed as a bit-string $x = \{x_1, x_2, \ldots, x_n\}$, where each x_i is either 0 or 1, where any continuous variables are encoded as floating point numbers. For multidimensional data, the bit-strings are simply concatenated. If, for instance, the data point 010101 is to be encoded in a quantum computer, it is simply mapped to the computational basis state $|010101\rangle$. This allows for multiple data points to be encoded in parallel as

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left|\boldsymbol{x}^{(m)}\right\rangle, \tag{6.1}$$

where $\mathcal{D}$ is the data set, M the total number of data points and $\boldsymbol{x}^{(m)}$ the m-th binarised data point. This is a simple encoding and has some significant disadvantages. There must be at least as many qubits as there are bits in the binarised data. For N bits, there are 2^N possible states, but at most M are used, which means that the embedding will be sparse. This means that the computational resources required to encode the data will in some sense wasted, and that the quantum computer will not be able to exploit the full power of the quantum hardware. To utilise the entire Hilbert space, amplitude encoding is better suited.

Amplitude encoding

A more efficient way to encode data is to use amplitude encoding, exploiting the exponentially large Hilbert space of quantum computers. This is done by mapping the bits in the bit-string not to individual qubits, but to individual amplitudes in the exponentially large Hilbert space. Mathematically, for some N-dimensional data point $\boldsymbol{x}$, this reads

$$|\psi(\boldsymbol{x})\rangle = \sum_{i=1}^{N} x_i \, |i\rangle, \tag{6.2}$$

where x_i is the ith component of the data point and $|i\rangle$ is the ith computational basis state. This has the advantage of being able to encode any numeric type natively, and perhaps more importantly, only needing logarithmically many qubits. For N-dimensional data points, only $\lceil \log_2 N \rceil$ qubits are needed. This is a significant improvement over the basis encoding, which requires N qubits (or more if integers and floats are to be binarised).

Amplitude encoding can easily be extended to cover the entire data set. This is done by concatenating the data points, after which the data set $\mathcal{D}$ with M data points can be encoded as

$$|\mathcal{D}\rangle = \sum_{m=1}^{M} \sum_{i=1}^{N} x_i^{(m)} |i\rangle |m\rangle , \tag{6.3}$$

where $x_i^{(m)}$ is the ith component of the mth data point. For such encodings, only $\lceil \log_2(NM) \rceil$ qubits are needed.

There are two main drawbacks of amplitude encoding. First, the data must be normalised, which can be done without loss of information by requiring an additional bit to encode the normalisation constant. Also, some padding may be needed if the dimension of the data is not a power of two. Secondly and more severely, there are significant practical difficulties with preparing such states. Any state of the form

$$|\psi\rangle = \sum_i a_i |i\rangle \tag{6.4}$$

must be efficiently and correctly prepared, which is not trivial. Unless some very specific assumptions are made, this is not possible with polynomially many gates (as a function of the number of qubits), which limits the potential for exponential speed-ups [100]. In general, for classical data, circuits must be linearly deep in the size of the data and ergo exponentially deep in the number of qubits, which makes it beyond the reach of NISQ hardware.

Angle encoding

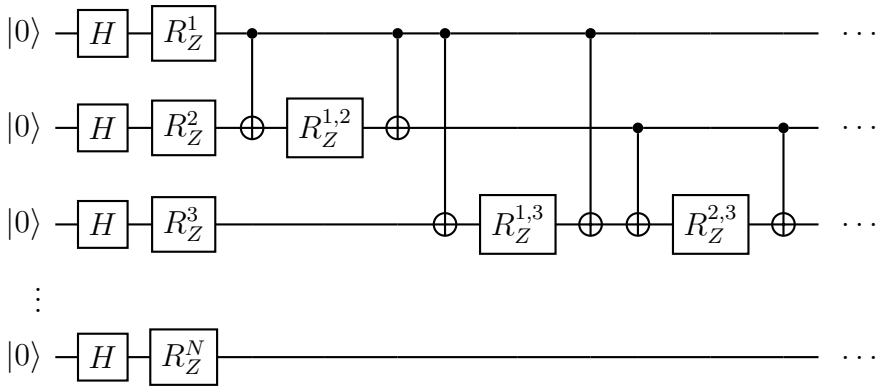
A third option is to encode data into the angles of rotations. Here, the potentially continuous components of the data are mapped to rotations of the qubits. For the rotations to be meaningful angles and not loop around, the data need to be normalised. An N-dimensional data point $\boldsymbol{x}$ is then encoded as

$$|\psi(\boldsymbol{x})\rangle = \bigotimes_{i=1}^{N} R_\sigma(x) |0\rangle , \tag{6.5}$$

where σ can be chosen to be either X, Y or Z. For Z-rotations, a Hadamard gate is prepended for the rotation to have an effect. N qubits are still required, but with native support for continuous variables, angle encoding can require fewer qubits than basis encoding. A constant number of gates are needed to prepare the state, which is a significant advantage over amplitude encoding. Still, being a product state, it offers no inherent quantum advantage.

Second-order angle encoding

Conjectured to be hard to simulate classically, a second-order angle encoding is proposed in [114]. First, angles are encoded as above, but thereafter the qubits are entangled and rotated further based on second-order terms. In circuit notation, such an encoding with Z-rotations reads



, (6.6)

where $R_Z^i = R_Z(x_i)$ and $R_Z^{i,j} = R_Z((\pi - x_i)(\pi - x_j))$ and with the entanglements and second-order rotations being applied pairwise for all N qubits. This increases the circuit depth to order N^2, and full connectivity is needed. Nonetheless, the increased circuit depth could be compensated for by the extra entanglement and subsequent expressibility, compared to the first-order encoding. Were it indeed classically hard to simulate, it could provide quantum advantage.

Repeats

The expressive power of models heavily relies on the encoding strategy. For instance, a single qubit rotation only allows the model to learn sine functions, where the frequency is determined by the scaling of the data. Generally, quantum models will learn periodic functions, and thus Fourier analysis is a useful tool. The implications of this are studied in [115], where it is shown that simply repeating simple encoding blocks allows for the learning of more frequencies and thus more complicated functions. Asymptotically, such repeats let a quantum model learn arbitrary functions.

6.2.2 Challenges

While the power of classical neural networks relies on the non-linear activation functions, the unitary operations in quantum computing are inherently

linear. However, depending on how the data is encoded, the linear transformations in the Hilbert space may not be linear in the input space. With basis encoding, for example, mapping $|b\rangle |0\rangle$ to $|b\rangle |\sigma(b)\rangle$ is doable, where b is a bit-string and σ some function. Amplitude encoding, on the other hand, has its input necessarily transformed linearly, and is for this reason (in addition to those mentioned in section 6.2.1) less suitable for QNNs. Such linear transformations in an embedding space can indeed be useful, both in the classical and quantum settings, but they may not be sufficient to make QNNs as powerful as classical NNs. It may make more sense to consider those models as kernel methods instead of neural networks, for which some quantum advantage has been indicated [116], but is beyond the scope of this report.

If intermediate measurements are used, non-linearities can be introduced in the quantum circuit. One way of doing this includes controlling gates with measurement results, as in [117]. Another way is so-called repeat-until-success schemes [118], where the circuit is run until the measurement of one qubit is what is desired before the remaining state can be used for further computation. Mid-circuit measurements can be used to define non-linear quantum neurons [119]. There, qubits are grouped together to represent one neuron, and intermediate measurements are used to approximate activation functions with piecewise constant functions.

Though there are methods like the parameter-shift rule that make it possible to find gradients and train the network using classical methods, none are as efficient as classical backpropagation for classical neural networks. This is because these methods require separate evaluations of the circuit, including numerous shots, for each parameter, whereas backpropagation is easily done after a forward pass through the network. Add to this the problems of noise and vanishing gradients, discussed in sections 4.4 and 6.1, and it is clear that NISQ-era QNNs can not be expected to scale as well as classical NNs do.

6.2.3 Architectures

Many architectures for QNNs have been proposed, commonly inspired by those for classical NNs. Still being in its infancy, it is not clear which (if any) will prove useful. Below follow some promising architectures, with a brief description of how they work compared to classical neural networks[16].

[16] Q.v. section 3.2.

Quantum convolutional neural networks

Originally introduced in [117], quantum convolutional neural networks (QCNNs) take inspiration from classical convolutional neural networks in that a sequence of convolutional and pooling layers are used to extract features and reduce the dimension before an output is made. In the quantum convolutional layers, neighbouring qubits are entangled by some parametrised gates, after which pooling layers reduce the active qubit count (usually by half). By basing the pooling on measurements, and controlling a gate based on a neighbouring qubit measurement, non-linearities are introduced. Because of the constant reduction of layer sizes in (Q)CNNs, the total parameter count can be reduced to only logarithmic order of the network depth, making them easier to train than dense networks of similar input size. After several iterations of convolution and pooling, gates can be employed on the remaining qubits, analogous to a finishing fully connected layer in classical CNNs, before the final measurement and output.

QCNNs have been shown to be able to classify topological phases of matter [117] and that they inherit their classical counterparts' ability to classify images [120]. Also, QCNNs have desirable properties with regard to avoiding barren plateaus [121], which could prove essential in training for problems of interesting size.

Quantum generative adversarial networks

Quantum generative models have been shown potentially to have an exponential advantage over their classical counterparts [122]. Due to the inherent probabilistic nature of quantum machines, it should not be surprising that they could learn difficult distributions more naturally than classical computers do. Moreover, leveraging a classical model as the adversary ensures that the quantum model can be of reasonable scale. Real quantum hardware has been used to generate (admittedly low-resolution) images of handwritten images [123].

Hybrid quantum-classical neural networks

Another option is to include a quantum layer or node in some larger pipeline or even non-linear graph structure. As parametrised quantum circuits are differentiable in their parameters, they can be handled using the chain rule when backpropagating a hybrid model. In [124], several such models are described and tested. The authors note that for the NISQ-era, limiting quantum components of models to very particular tasks to which they are

especially suited should be beneficial. As quantum hardware develops, it can take over more and more of the hybrid models.

Quanvolutional neural networks, proposed in [125], are a hybrid model in which the convolutional layers of a classical CNN are replaced by quantum layers. As the quantum part is restrained to a single layer and a small convolutional kernel, the design can be implemented with small quantum circuits with little requirements for error mitigation, still being able to process high-dimensional data, thereby making it a good candidate for NISQ-era hardware.

More recently, in [126], using a hybrid model for multi-class classification on real-world data sets using a CNN-inspired structure was explored. In the model used, the quantum part was placed in the middle, after classical convolutions and pooling and before a classical fully connected layer. There, it was shown that the hybrid model could outperform a classical CNN of similar parameter size.

Quantum recurrent neural networks

The quantum state is collapsed upon measurement, losing information in the process. It is therefore not straightforward to translate the classical recurrent neural network (RNN) to a variational quantum circuit in the manner of fig. 6.2 and the above discussed architectures. Still, attempts have been made to define a QRNN, such as in [127] where parametrised quantum neurons are defined and used to construct layers. These layers have two sets of qubits. The first set is not measured, and the state persists to the next layer. In contrast, the second set consists of ancillary qubits, which are either initiated in the state $|0\rangle$ or used to input data. The ancillary qubits are measured after each layer and interpreted as the output of the layer. It is shown to achieve 95% accuracy on classifying 0 and 1 in the MNIST data set with only 12 qubits.

In [128], a similar design for QRRNs is used to learn temporal patterns in data, such as cosine waves or spin quantum dynamics.

6.3 Quantum agents

As quantum neural networks are shown to be usable function approximators, they can be used in the same way classical neural networks are used in classical deep reinforcement learning algorithms. In this section, how quantum neural networks can be used as agents in reinforcement learning algorithms is discussed. For this, several designs have been proposed, though

the field of quantum reinforcement learning is still much less developed than quantum supervised learning is.

Like with QML, the two fields of reinforcement learning and quantum machine learning can be combined in several ways. The entire environment or Markov decision process can be assumed to be quantum, such as is done in [129]. There, approaches based on dynamic programming are used to solve the problem. Nonetheless, continuing the theme of solving classical problems with quantum methods, this thesis will focus mainly on the use of quantum neural networks as agents in classical reinforcement learning algorithms. Still, as will be seen, quantum agents provide the greatest benefits when also the environment exhibits quantum behaviour.

In [130], quantum neural networks are used to approximate Q-functions, which are used in reinforcement learning. Moreover, by including experience replay and a target network, the model proposed is effectively a true translation of the classical DQN algorithm to the quantum setting, a QDQN. The authors note that the model is more efficient than classical DQN in terms of memory usage and parameter counts.

Quantum Q-learning is also studied in [131], where a QNN-based Q-learning is tested in several standard benchmark environments, including the cart-pole environment. The model achieves results comparable to those of a DQN. As is the case with classical reinforcement learning, it appears that hyperparameter tuning and model architectures matter more than the pure parameter count for the performance of the model.

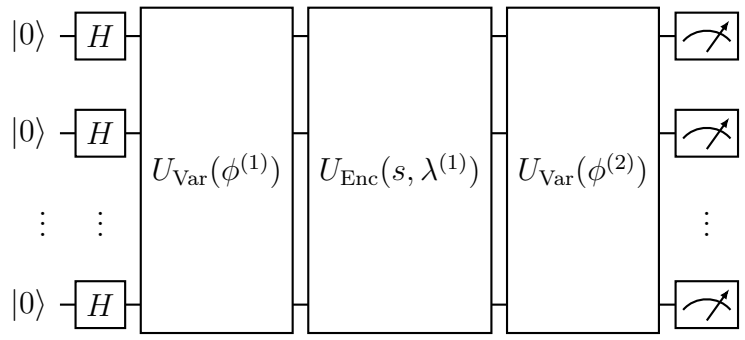
The authors of [132] propose to use a QNN to approximate the policy function in reinforcement learning and train it with the REINFORCE algorithm. Their model is shown to solve several basic benchmark problems, including the cart-pole environment, with performance comparable to that of classical neural networks. Furthermore, by designing quantum environments particularly suited for the quantum model, it is shown to outperform classical neural networks.

By letting the agent communicate over a quantum channel, [133] shows that the learning of an agent can be accelerated. Somewhat similarly, in [134], hybrid agents are shown to learn quadratically faster than purely classical agents.

6.3.1 Implementing a quantum agent

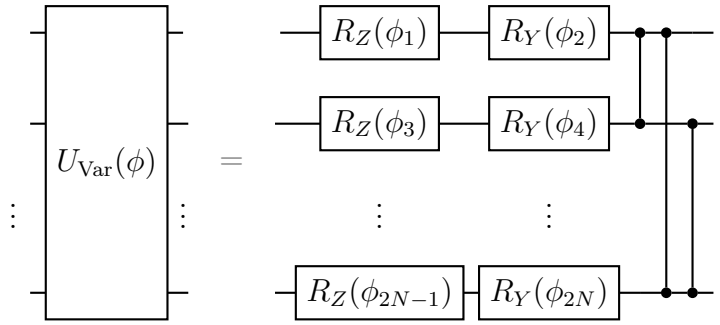
The quantum model as proposed in [135] consists of alternating variational and encoding blocks with as many qubits as there are state features. It is of linear depth with respect to the number of state features, and it is therefore tractable to implement and simulate classically. Interestingly, the

model does not begin with data encoding, but with a variational block. Overall, the network appears as



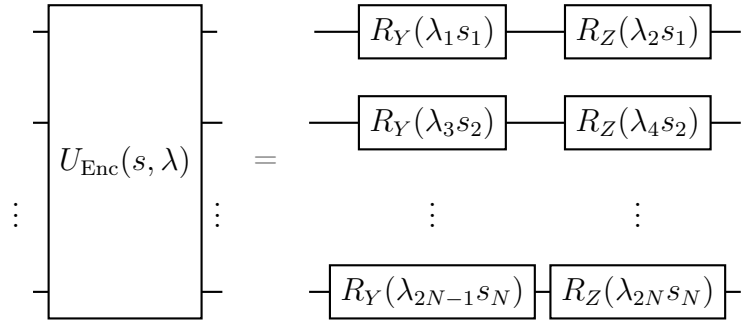
(6.7)

where the variational blocks take in parameters ϕ while the encoding block depends on both its parameters λ and the input state $|s\rangle$. For more expressibility, further repetitions of the variational and encoding blocks may be appended pre-measurement. The variational blocks are given by



(6.8)

where the rightmost CZ-gates are applied pairwise on all qubits. The encoding blocks appear as



(6.9)

such that the λ-parameters simply scale the input state features. Note that these blocks do not produce any intra-qubit actions and so no entanglement; for that the variational blocks are used.

To define a policy hence, a set of measurements, $\{O_a : a \in \mathcal{A}\}$ is used whereby each element corresponds to an action a. So,

$$\pi(a|s) = \frac{\exp\left(\langle O_a \rangle\right)}{\sum_{a' \in \mathcal{A}} \exp\left(\langle O_{a'} \rangle\right)}, \tag{6.10}$$

where $\langle O_a \rangle$ is the expectation value of the observable O_a. For all experiments, the measurement was taken to be the parity when measuring all qubits in the Z-basis with a trainable weight w, id est $O_a = \pm w \prod_{i=1}^{N} Z_i$.

All this is easily done using the PennyLane library [136] in Python. In it, the circuits as described above are implemented procedurally as functions, which in Python are first-class objects, such that they are easily composable. The circuits are then compiled to the desired backend, which in this case was its built-in `default.qubit` simulator. What is more, the library provides seamless integration with PyTorch [137], such that gradients can be computed easily with automatic differentiation and optimised using standard machine learning tools and optimisers.

The quantum policy was trained using the REINFORCE algorithm with the Adam optimiser, as listed in algorithm 6.1. For the Adam optimiser, learning rates of 0.01 was used for the variational parameters ϕ and 0.1 for the encoding parameter λ and observable weight w.

Algorithm 6.1: REINFORCE algorithm for a quantum policy.

Input: Initial policy π, batch size N,
while Some stopping criterion is yet to be met **do**
  for $i = 1, 2, \ldots,$ Batch size **do**
    while Episode is not finished **do**
      Sample action $a \sim \pi(a|s)$
      Perform action a
      Observe reward r
      Observe next state s'
      $s \leftarrow s'$
    Compute expected return $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$
    Store history $(s_0, a_0, G_0), (s_1, a_1, G_1), \ldots, (s_{T-1}, a_{T-1}, G_{T-1})$
  Compute loss $L = \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T-1} \log \pi(a_t^{(i)}|s_t^{(i)}) G_t^{(i)}$
  Compute gradient $\nabla_{(\phi,\lambda,w)} L$ using automatic differentiation
  Update (ϕ, λ, w) using the Adam optimiser

Chapter 7

Results

This chapter presents empirical evaluations of the quantum upper bound confidence bound algorithm's performance as it is described in chapter 5 and its original proposal paper [15]. Despite the algorithm's theoretical guarantees, promising asymptomatically lower regret bounds than any classical algorithm can achieve, how it performs in practice has not been extensively studied. In [15], simulations were done to show its supreme performance versus the classical UCB algorithm for certain fixed bandit instances. Here, the algorithm is compared also with the overall better Thompson sampling algorithm, and by considering more bandit instances and Bayesian regrets, it is shown that the algorithm is superior to the classical algorithms for only a certain set of problems.

Furthermore, in section 7.2, modern reinforcement learning algorithms are tested on the bandit problem, including a novel quantum neural network policy agent presented in [135]. Applying such general algorithms to the bandit problem is not a common approach, but it is interesting to see how they perform and compare to the more problem-specific algorithms.

All algorithms were implemented in Python 3 [138] with quantum computing support provided by IBM's Qiskit library (v0.41) [139] for the QUCB algorithm and PennyLane (v0.29) [136] for the quantum neural network policy agent. The experiments were executed on an external computation server with an Intel Xeon E5-2690 v4 CPU and 768 GB of RAM, permitting 28 concurrent simulations. Approximately 25,000 CPU hours were used. The source code for the simulations presented here is available on GitHub[17].

[17] Cf. `https://www.github.com/boyesjo/tma4900/`.

7.1 Algorithms for the bandit problem

By producing better predictions of the rewards of the arms, the QUCB algorithm promises to outperform classical algorithms in the multi-armed bandit problem. While the upper bound shown in section 5.1 implies QUCB advantage, it does not necessarily mean it is supreme in all cases or even on average. It is therefore interesting to test it on a variety of problems and compare its average performance to other algorithms — not only with the classical UCB to which it is compared in the original paper, but also with the more performant Thompson sampling algorithm[18]. The UCB algorithm and Thompson algorithms were implemented in a straight-forward manner per the descriptions in sections 2.3.4 and 2.3.5, relying primarily on the NumPy (v1.23) [140] and SciPy (v1.10) libraries [141], while the (simulation of the) QUCB algorithm was implemented as in explained in section 5.1.2, with the Qiskit library (v0.41) [139] for quantum computing support and simulations of quantum algorithms.

Remark that it is specifically the UCB1 of [25] that is implemented and that the digit is here omitted from the notation for brevity. Similarly, what here is simply called QUCB is in the original paper called QUCB_1. The Thompson sampling algorithm that is implemented follows the original paper [17], with its uniform prior and conjugate Beta posterior distributions.

For fixed bandit instances, 100 simulations were run for each algorithm, while for Bayesian regrets 1000 parallels were run. Simulating quantum algorithms is computationally expensive, in general exponentially hard[19]. To emulate the logic of a quantum computer, the classical libraries for simulated quantum computing effectively perform the linear algebra operations as described in chapter 4 in the exponentially large vector spaces, perhaps with some clever optimisations when possible.

Due to this computational cost, much higher numbers of simulations were not feasible for the QUCB algorithm and the time horizons considered on the hardware dedicated to the project. As such, there is undoubtedly some Monte Carlo error in the results, but it is not expected to be significant. The differences between the performances are in many cases large enough to draw statistically significant conclusions, and the results are consistent with the original paper [15].

To limit the scope and computational resources required, only Bernoulli rewards were considered, and in every case except one, only two arms were used. How the algorithm handles more arms may be of some interest, but

[18] Q.v. chapter 2.

[19] The exponential complexity of simulating quantum mechanics is after all what lead to the conception of quantum computing in the first place (q.v. chapter 1).

its principal advantage is its better dependence on the time horizon T, and it handles multiple arms the same was as the classical UCB algorithm, so there is no reason to expect any regret improvements as the number of arms increases.

The QUCB algorithm that was implemented could in principle handle any rewards distributions with bounded values, but doing so would increase the qubit counts and in turn the computational cost of the quantum Monte Carlo estimates of the reward means Likewise, the QUCB_2 algorithm for bounded variance rewards could be tested on even more problems, but it is too would demand yet more computational resources to test on long enough horizons to observe quantum advantage.

Across all bandit simulations, a time horizon of $T = 250,000$ was used. This is indeed a large number, but still merely a quarter of what was tested in [15]. The quantum algorithm presents no immediate advantage for smaller time horizons, so a time horizon of substance must be chosen to see its effects in the simulations. Specifically, due to its exponentially long quantum Monte Carlo periods, as presented in section 5.1, the horizon must be large enough for the greater accuracy of the QMC estimates to outweigh the cost of the long periods.

As noted in [15], setting the confidence parameter to a higher value than the theoretically desirable $1/T$ leads to better performance. It was for this reason set to 0.01 across all experiments in this chapter. Moreover, the constant C_1, which is only defined existentially, was arbitrarily set to 2 across all experiments, as it was not clear how to choose it in a principled manner, and this value seemed able to reproduce the results of [15].

7.1.1 Fixed arms

First, the algorithms are tested on fixed bandit instances with two arms. The mean of the first arm is set to 0.5 and the mean of the second arm is set to 0.505, an instance also tested in [15]. The results are shown in fig. 7.1. QUCB greatly outperforms UCB. However, Thompson sampling which was not considered in [15], is not that far behind.

It is worth acknowledging that the algorithms do not outperform the random, uninformed baseline by substantial amounts. Especially UCB, which performs worst of the three proper methods, does accumulate about two-thirds of the baseline regret. This is indicative of the difficulty of the instance, wherein the arm means are closely set.

This instance was chosen as it was tested in [15], and by replicating the results thence, it is easier to trust that the implementation of the quantum upper confidence bound algorithm is correct. Due to the computational cost, a slightly reduced time horizon of only $T = 250,000$ was used, instead of the original paper's $T = 1,000,000$, but this is enough to see the same trends as were reported in the original paper, and particularly notice the quantum advantage of QUCB when compared to its classical counterpart in UCB.

Note the jagged and periodically completely flat behaviour of the QUCB algorithm regret. The regret is recorded at every turn, so this is not due to poor plotting. Instead, this is caused by the long quantum Monte Carlo periods in which its quantum advantage is gained; it must repeatedly pull the same arm for the QMC estimates to be produced, and these periods are exponentially long in the time horizon T. When the algorithm pulls the optimal arm, the change in regret is indisputably zero. Additionally, these periods get longer as the algorithm progresses, as the QMC estimates become more accurate. The exponential lengthening of the QMC periods assures that such non-smooth and piecewise linear behaviour is to be expected. Because of these long periods, there are not as many possible trajectories as with the classical algorithms, so the jaggedness persists despite averaging over many simulations. Due to this exponential lengthening, similar behaviour should appear in all experiments with fixed instances regardless of the horizon. Such behaviour is not seen in the classical algorithms, as they can pull any arm at any time, so there are many more possible trajectories, and the regret curves are much smoother.

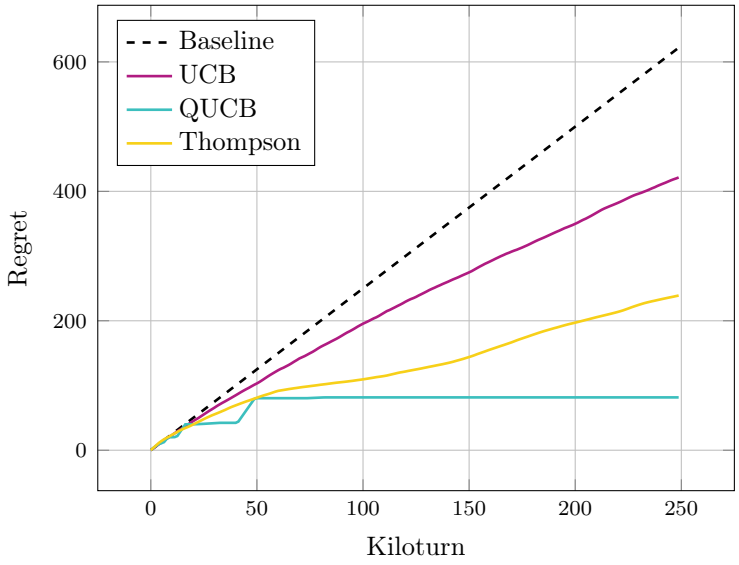


Figure 7.1: Regrets for two Bernoulli arms, with means 0.5 and 0.505.

Low and high probabilities

Next, more extreme mean values of 0.01 and 0.005 with a reward gap of 0.005 equal to the above, are tested. In the original paper, only cases of 0.5 and a nearby value for the second arm were considered, but it is interesting to see how the quantum algorithm performs on more extreme values, and to what degree it is the difference of means that matters or some other function of the means.

Figure 7.2 contains the results for these simulations. There, it can be seen that QUCB beats UCB thoroughly still, as both algorithms achieve similar regrets as in fig. 7.1. But now Thompson sampling is clearly superior. UCB seems barely able to learn in the horizon considered, and QUCB took a substantial regret hit in the beginning, before its long flat QMC period and pulling of the optimal arm allowed it to beat UCB.

At these extreme values, it seems the gap must shrink for QUCB to be able to outperform Thompson sampling. This is shown in fig. 7.3, where the gap is reduced to 0.0005, Even so, the QUCB advantage is not as pronounced as in fig. 7.1. Here, UCB is very close to the random, uninformed baseline, while the two other algorithms are able to learn. Still, their regret curves are also here rising in the time horizon considered. It is clear that this is a very difficult instance, and perhaps a greater time horizon could have shown the quantum advantage of QUCB more clearly.

In fig. 7.3, the UCB algorithm is only marginally better than the random baseline, and it is likely the case that its regret is not significantly different from the baseline at this horizon, considering the limited amount of simulations performed here of only 100. Evidently, this is such a hard instance that the UCB algorithm is unable to learn in the horizon considered.

It appears as for the upper confidence bound algorithms, solely the difference between arm rewards means determines the performance. For the Thompson sampling algorithm, however, the absolute values of the means are also important, such that maybe some other scale like the difference of logit or probit means could be more relevant. This may be due to its inlaid knowledge of Bernoulli rewards, while the UCB and QUCB algorithm tested here only assumes bounded rewards, and they must thus be more conservative.

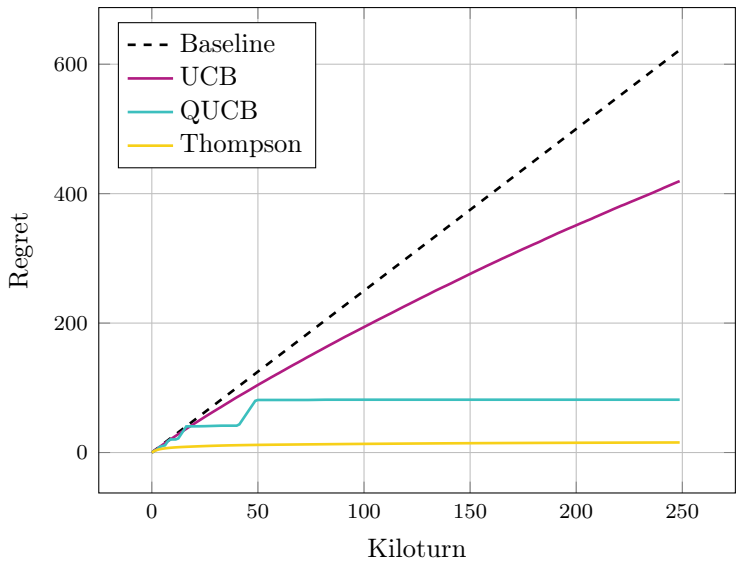


Figure 7.2: Regrets for Bernoulli arms with means 0.01 and 0.005.

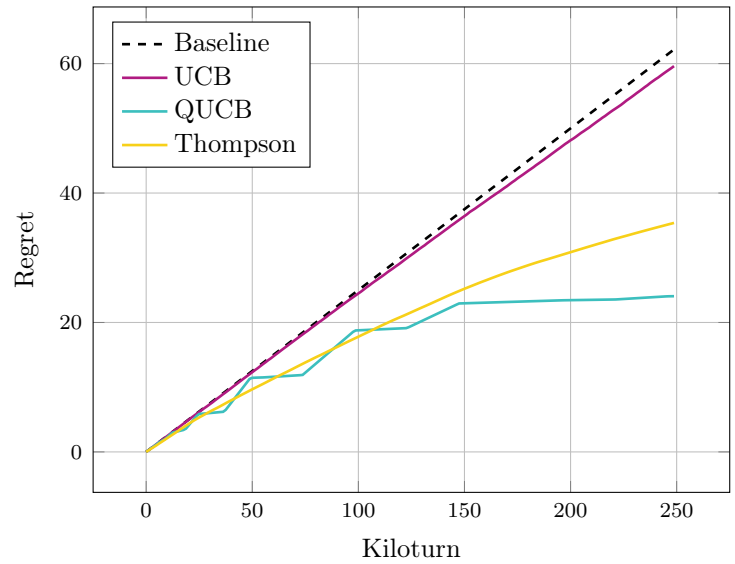


Figure 7.3: Regrets for two Bernoulli arms with means 0.99 and 0.9905.

Four arms

As a final test, the number of arms is increased to four. The results are plotted in fig. 7.4. Only cases with two arms were tested in the original paper, so this is a new test, further validating the correctness of the algorithm and its implementation. The results are similar to the two-arm case, with QUCB outperforming UCB. At these particular reward means, Thompson sampling performs very similarly to QUCB.

With the extra arms, the qubit count is increased. Therefore, the computation price of simulation and testing such cases is higher. For this reason, no further increases in arm counts were investigated and no deeper studies in the case of four arms were performed. In general, QUCB provides no advantages or noteworthy changes from UCB with respect to the number of arms. While the dependence of the regret on the number of arms can be of major concern for many applications, this is not within the scope of this thesis. Anyway, there is little reason to expect QUCB to show any different behaviour than UCB in this regard, as it handles the arms in the same way as classical UCB does. It may be that its better predictions could lead to better performance in some cases. There is no a priori reason to expect this, however, and in the bound described in section 5.1.1, the regret is only bounded linearly in the number of arms, which is worse than the square root dependence of the regret on the number of arms for UCB. This may be a small price to ask for the logarithmic horizon dependency, and a square root bound for both is likely achievable, but this is not within the scope of this thesis. Overall, the arm count is not a major concern for the QUCB algorithm, and it is not investigated further here.

In the figure, it is clear that all three algorithms significantly surpass the random baseline. It is then perhaps the case that the instance chosen for this experiment had too easily identifiable optimal arms, and that more interesting behaviour and clear quantum supremacy could be observed for a more difficult instance.

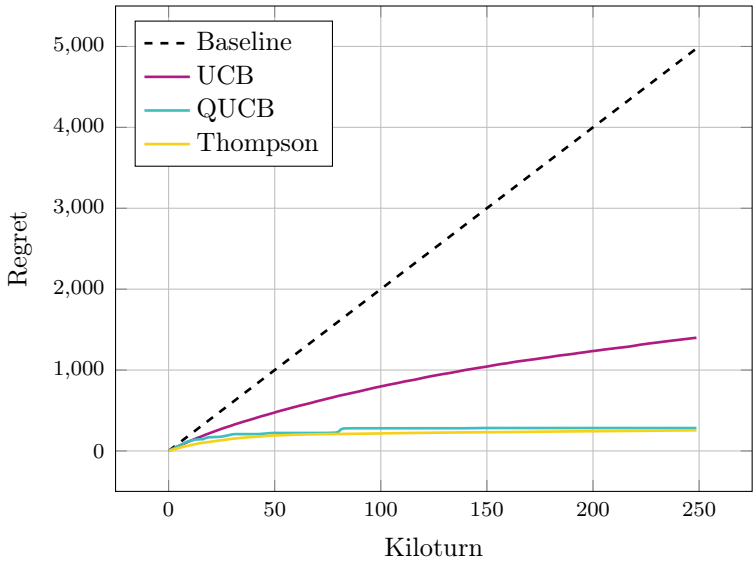


Figure 7.4: Regrets for four Bernoulli arms with means 0.5, 0.51, 0.52 and 0.53.

7.1.2 Bayesian regret

As described in section 2.2.3, the Bayesian regret is the average regret over some prior. What priors to choose is a topic for discussion, which will not be covered deeply in this section nor thesis in general. It may not be the most relevant measure for which to optimise policies. In any case, it provides a measure of robustness, measuring the performance over a range of possible instances.

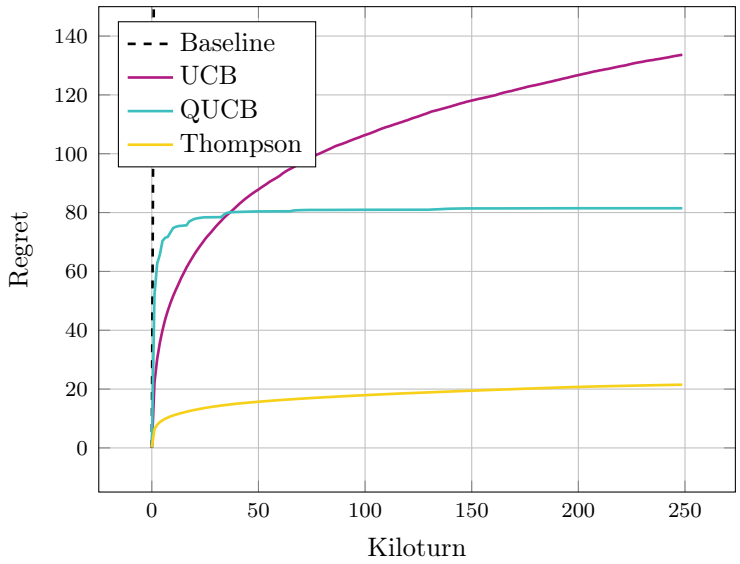
The use of Bayesian regrets instead of the regret for some fixed bandit instances is rarely seen in the literature. No such experiments were done in [15], so what follows are new results. The same three algorithms as in the previous section are tested, but now on a different set of bandit instances drawn from different prior distributions for the arms. With Bernoulli rewards that are determined by their mean and just two arms, these priors are distributions over $[0, 1]^2$.

Uniform prior

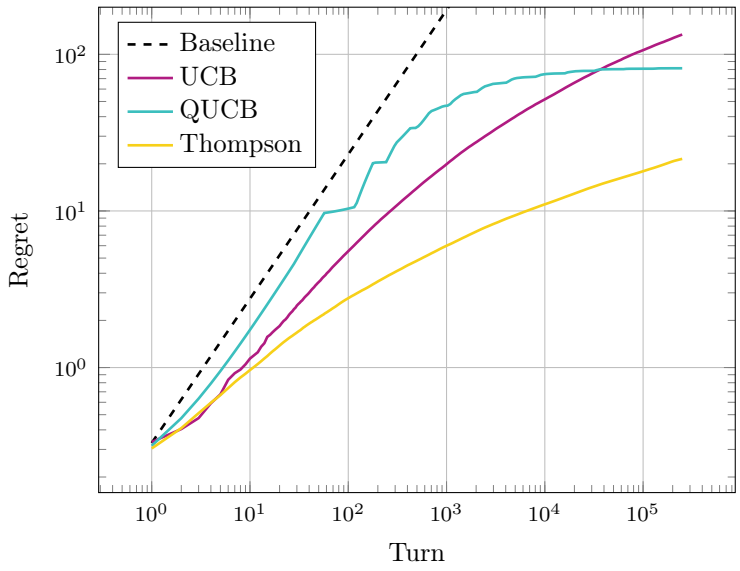
First, the simple case of two arms, whose means are independently drawn from a uniform distribution on the interval $[0, 1]$ will be considered. The Bayesian regret for this case is on display in fig. 7.5. All three were run on 1,000 instances sampled from the prior. Having randomness also in the bandit instances motivated the use of a larger number of simulations than in the previous experiments. The results shown are the averages over these instances. The distribution of the final regrets is considered last in this subsection.

It is obvious that the Thompson sampling algorithm performs best. QUCB does outperform UCB, but not by as much as the previously considered cases and not in the first $\approx 30,000$ turns.

The lack of any quantum advantage is probably due to this prior generally yielding 'easy' problems, where the optimal arm can be quickly identified. In such cases, the initial overhead of the quantum algorithm appears not to be worth the effort. QUCB with its current QMC implementation relies on a large number of initial samples to produce it supreme estimates, but with the uniform prior and only two arms, the expected difference of means is a third, so the optimal arm can often be identified after only a few turns with high certainty. Nonetheless, after the rather wasteful inaugural period, QUCB does indeed produce a very flat regret curve; inspecting the log-log-plot in fig. 7.5, it may seem like the QUCB regret curve would lie below even Thompson sampling at some great and admittedly unrealistic number of turns, perhaps something around the order of 10^9.



(a) Linear scale.



(b) Log-log scale.

Figure 7.5: Bayesian regret for two arms with independent and uniform priors.

Challenging prior

One might argue that the uniform prior is too easy, and that the quantum advantage should be more apparent in a more challenging prior. Firstly, for real-world Bernoulli bandits, there may be reasons to believe that the means would lie closer to the endpoints 0 and 1 than in the middle. Secondly, for the problem to be interesting and warrant the use of clever quantum algorithms or even any bandit theory whatsoever, the means should be assumed close to each other. With means that lie far apart, the optimal arm can be identified with high certainty after only a few turns. Such problems are certainly also interesting, but it is not for these that the quantum algorithm is designed and for which it can be expected to provide benefits.

Thus, to consider a more challenging prior, the following was chosen:

$$\begin{aligned} \mu_1 &\sim \mathrm{B}(0.5, 0.5), \\ \mathrm{logit}(\mu_2) &\sim \mathrm{N}(\mathrm{logit}(\mu_1), 0.1), \end{aligned} \tag{7.1}$$

where the logit function is defined as

$$\mathrm{logit}(\mu) = \log(\mu/(1-\mu)). \tag{7.2}$$

This prior is displayed in fig. 7.6. The difference of the means is there plotted rather than the second mean, as the joint density of both arms would simply appear non-informatively as a thin diagonal line.

The logit is used to ensure that the means are in the interval $[0, 1]$. It is particularly well-suited for interpreting the means of Bernoulli bandits due to its ability to capture meaningful differences between probabilities. For example, on the logit scale, the distance between 0.001 and 0.101 is much greater than that between 0.4 and 0.5. Its symmetry around zero and coverage of the entire real number line allow for a comprehensive representation of the reward means.

Running a successful 1,000 simulations with this prior, the regrets visualised in fig. 7.7 were obtained. At the ultimate 250,000th turn, the QUCB and Thompson sampling algorithms have virtually equal regrets. QUCB lags behind Thompson sampling in the beginning, but its regret curve appears flatter towards the end, and it is likely that it would beat Thompson sampling in the long run. The UCB algorithm, on the other hand, is as before clearly outperformed by both QUCB and Thompson sampling.

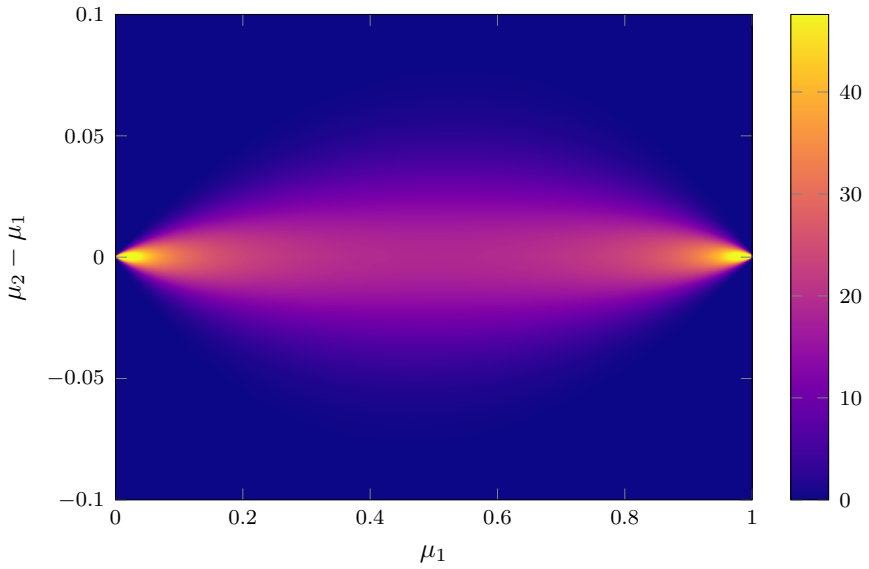


Figure 7.6: Challenging prior used in for Bayesian regret experiment, as defined in eq. (7.1). Note that the density diverges at $(\mu_1, \mu_2) = (0, 0)$ and $(1, 1)$. Values above the colour bar maximum are therefore clipped.

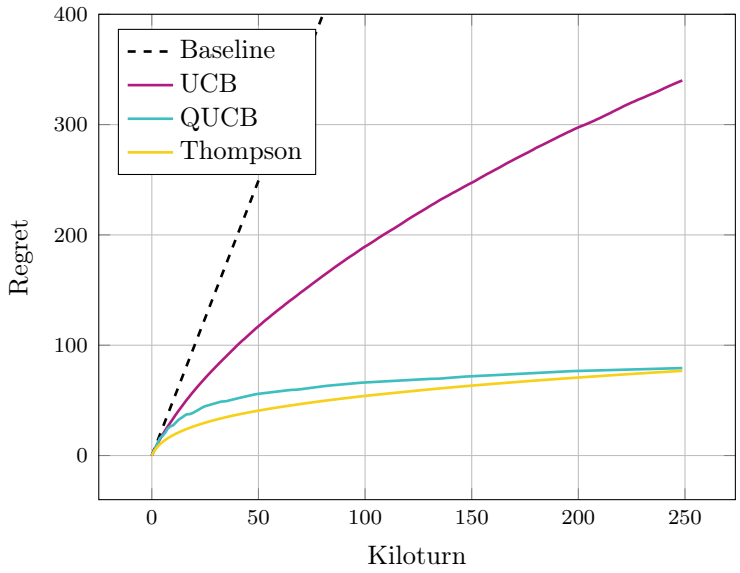


Figure 7.7: Bayesian regret for two Bernoulli arms with a challenging prior as in eq. (7.1).

More challenging prior

As the above prior proved to be still too easy to demonstrate quantum advantage, a third was chosen, catering even more to QUCB's strengths discovered in section 7.1.1. Based on the relative performances of QUCB and Thompson sampling in fig. 7.1 and fig. 7.2, the quantum algorithm seems to perform relatively better when the arm rewards means are centred around 1/2. Furthermore, as should have been made clear throughout this thesis, the quantum algorithm is designed to be most useful when the arms are close to each other. Accordingly, the even more challenging prior was constructed similarly to the above one in eq. (7.1) for the same reasons and for simplicity, but with different parameters to achieve the desired effect. The resulting prior is

$$\begin{aligned} \mu_1 &\sim \mathrm{B}(2,2), \\ \operatorname{logit}(\mu_2) &\sim \mathrm{N}(\operatorname{logit}(\mu_1), 0.02), \end{aligned} \tag{7.3}$$

as is rendered in fig. 7.8. There it can be visually confirmed that the means are not only more likely to be close to 1/2, but closer to each other as well — at least when μ_1 is close to 1/2.

This simulation was also conducted with 1,000 parallels from which the average regrets at each turn is graphed in fig. 7.9. There, it is seen that QUCB reclaims superiority, while Thompson sampling still clearly outperforms UCB. It is clear that this prior produces so challenging instances, that the advantage over pure exploration is not as apparent as in the previous priors, but more akin to the first considered fixed instances in section 7.1.1, where the QUCB advantage was initially observed[20].

Unlike the uniform prior in fig. 7.5, where the baseline is completely outclassed by the more sophisticated algorithms already in the first few turns, and the challenging prior in fig. 7.7, where the baseline is still clearly subjugated by the three more advanced algorithms, the baseline here in fig. 7.9 is now in the same order of magnitude as the other algorithms. This indicates that this prior is indeed more challenging than the previous ones, placing its difficulty more in line with the one considered in fig. 7.1.

[20] Viz. figs. 7.1 and 7.3.

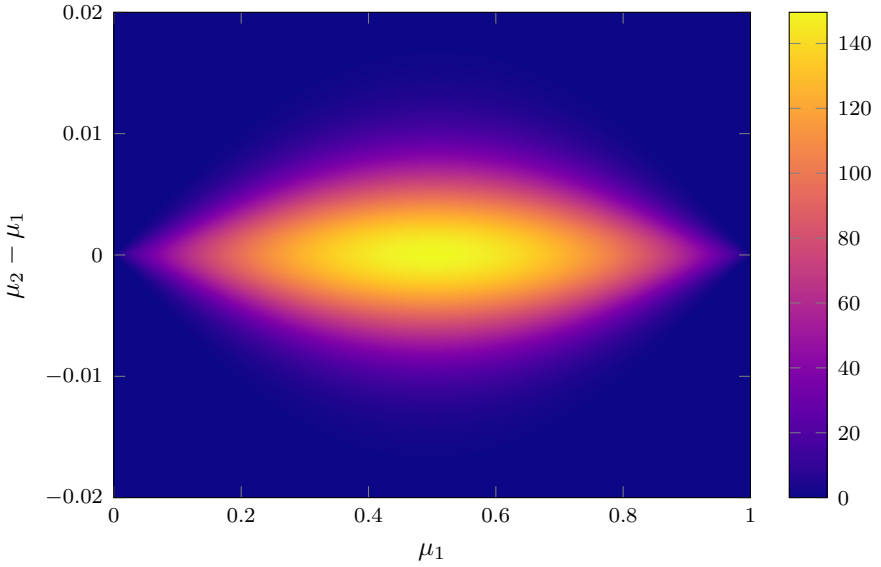


Figure 7.8: More challenging prior used in for Bayesian regret experiment, as defined in eq. (7.3).

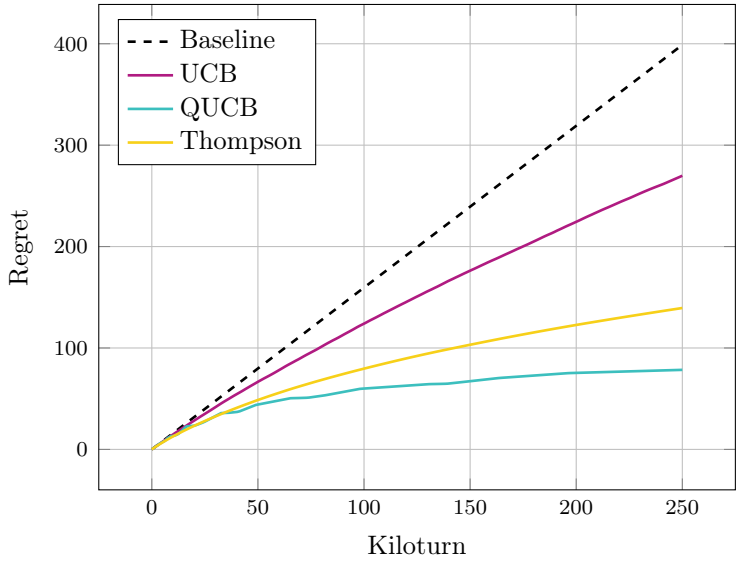


Figure 7.9: Bayesian regret for two Bernoulli arms with a more challenging prior as in eq. (7.3).

Regret distributions

With more simulations computed, it is possible to garner more information about the performance of the algorithms. In fig. 7.10, the regrets at the final turn for the three algorithms are shown for the three priors that were tested. The distributions are clearly wide, but with 1,000 samples, the conclusions drawn in the previous sections are nevertheless significant.

For the uniform prior, Thompson sampling is unquestionably supreme, while for the prior of eq. (7.1), QUCB and Thompson achieve a similar average regret, but with QUCB's distribution being more concentrated. It seems that even the classical UCB algorithm has a better mode than QUCB, but with its heavy tail, it is still outperformed by QUCB on average. Across all three histograms, it appears as QUCB is more consistent in its performance, being less dependent on the random instances that are generated, while Thompson sampling, though better at many instances, suffers from outliers with very high regret. QUCB seemingly produces equal regrets independent of the prior, while the two classical algorithms perform worse on the more challenging priors. This could perhaps be explained with its logarithmic instance-independent regret bound as shown in section 5.1.1. Classically, such bounds are at best proportional to the square root of the horizon. With its early and expensive exploration period, the QUCB algorithm is able to identify the best arm within the horizons considered here, seemingly independent of the difficulty of the instance.

The classical UCB algorithm appears to have even more spread in its regret distribution than Thompson sampling, and simply worse performance overall. This is to be expected, as it does not achieve the same guarantees as Thompson sampling, as discussed in chapter 2.

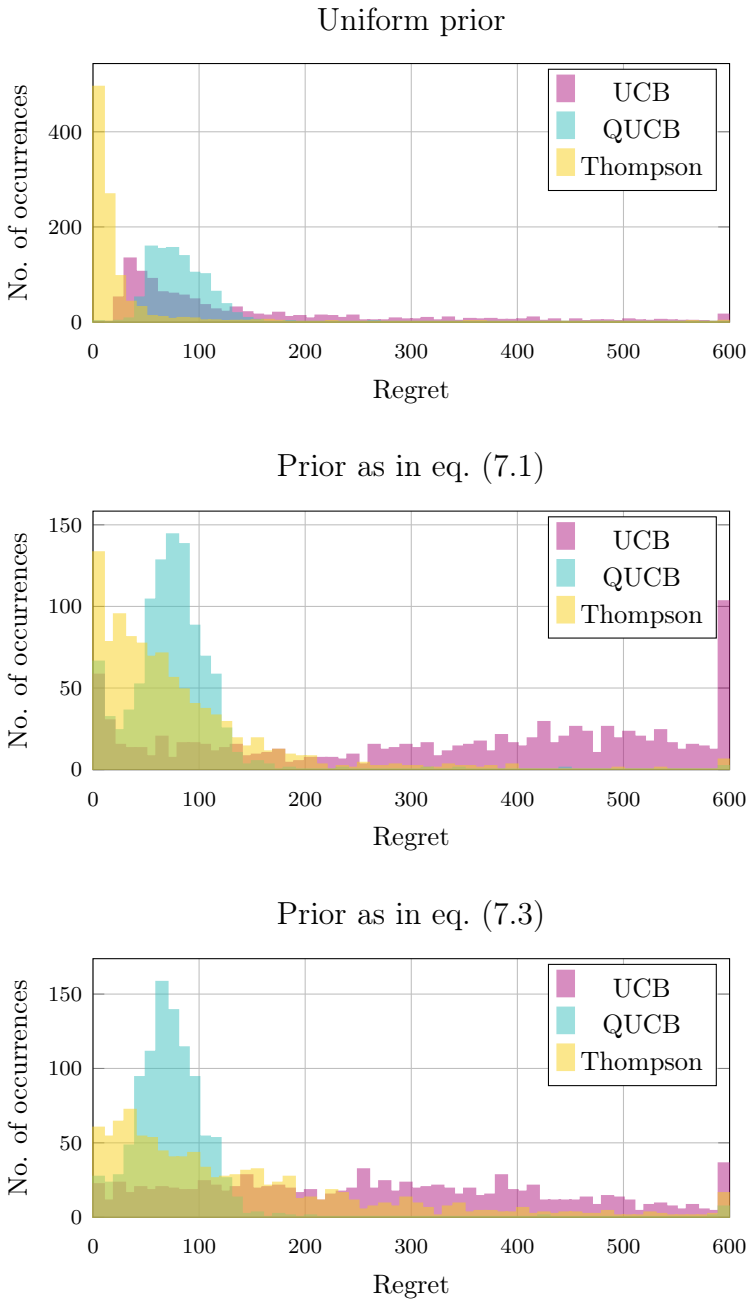


Figure 7.10: Histograms of final turn regret for the three priors considered. Note that the rightmost bins are open-ended; they include all values greater than the highest regret value indicated. This was done to keep the interesting parts of the histograms at a reasonable scale.

7.2 Reinforcement learning algorithms

In addition to the above considered algorithms specifically made for the bandit problem, knowledge may also be gained from considering how more general-purpose reinforcement learning algorithms perform. Here, three state-of-the-art algorithms are tested on a general reinforcement learning problem and then the bandit problem. These algorithms are the deep Q-network (DQN), a value-based algorithm, the advantage actor-critic (A2C), a mixed policy and value-based algorithm, and the proximal policy optimisation (PPO), a policy-based algorithm, all described in further detail in section 3.3 They are all implemented in the Stable Baselines 3 (v1.7) [142] library, which is a high-level reinforcement learning library built on top of PyTorch (v2.0) [137] and OpenAI's Gym (v0.21) [40] library, intended to make it easy to apply reinforcement learning algorithms to new problems without having to implement them from scratch, worry about the details of the algorithms or having to tune their hyperparameters. Moreover, the general-purpose quantum reinforcement algorithm of [132] was also implemented and tested. See section 6.3.1 for more details on the PennyLane (v0.29) [136] implementation of the quantum policy-based algorithm and its optimisation.

7.2.1 Cart-pole

The first problem considered is the cart-pole problem. This was introduced in section 3.1.1. All algorithms were trained for 2,000 episodes, repeated ten times, and the results are rendered in fig. 7.11. Note that most of the algorithms display very inconsistent performance, at least before completely solving the problem, so a rolling mean was used. This inconsistency is likely caused by both the random initialisation, which the agents do not immediately learn and handle, and by the stochastic policies.

From the figure, it is easily seen that the proximal policy optimisation algorithm is the only algorithm that consistently solves the problem. DQN appears completely futile, while both A2C and the QNN-based method of [132] certainly do learn and perform better than their initial random policies, but they do not achieve the same level of performance as PPO.

7.2.2 Bandits

Finally, the same agents as above are tested on the bandit problem. Note that translating the bandit problem to a reinforcement learning problem is not at all trivial. The modern algorithms rely on predicting an action given

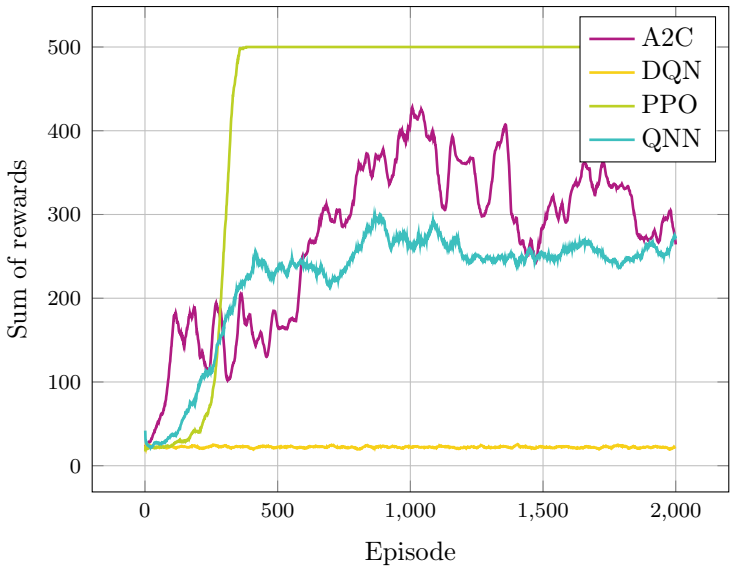


Figure 7.11: Accumulated rewards during cart-pole training. Each algorithm was trained 10 times, and the average is shown. Thereupon a rolling mean of 20 episodes was used to smooth the still highly noisy data. In the cart-pole environment, rewards are given at each time step, so the sum of rewards equals the number of frames the pole was balanced.

a state, and the bandit problem is in principle stateless. It was therefore decided to encode the empirical means of arms as the state. In addition, to provide all the information, also the number of times each arm has been pulled is encoded, and the current time step is also included. The number of times the arms were pulled was divided by the current time step to normalise the values, and the current time step was transformed by the hyperbolic tangent to normalise it to the range $[0, 1]$, ensuring that the state is always in the range $[0, 1]^{2k+1}$ This helps the stability of the classical neural network-based algorithms and is crucial for encoding the data in the quantum neural network. The state was consequently a vector of length $2k + 1$, k being the number of arms, given by

$$S_t = \left(\bar{X}_1, \bar{X}_2, \ldots, \frac{T_1}{T}, \frac{T_2}{T}, \ldots, \tanh(T) \right). \tag{7.4}$$

The same instance as in fig. 7.1 was used. It was chosen due to being difficult enough to be interesting, but not as computationally expensive as something with more arms or a Bayesian regret which warrants more parallel simulations. Each agent trained on the problem 100 times, from which the average results are shown in fig. 7.12. It appears as if only the A2C algorithm is able to learn something in the time horizon considered.

Why these algorithms are not able to learn is not entirely clear, and could warrant a much deeper investigation than is covered here. It is possible that the state is not a good representation of the problem, but it is also possible that the algorithms are simply not suited for the problem. They can not be expected to perform as well as the problem-specific algorithms of UCB, QUCB, Thompson sampling et cetera, but it is surprising that they do not perform better than random. A relevant factor is that these algorithms are not designed for optimal sample efficiency, as is inherent to the bandit algorithms, but rather for optimal final performance. It may be the case that for further training, the algorithms could display better performance and sublinear regret. Nonetheless, it is clear that the problem-specific algorithms are indeed superior to the modern general-purpose algorithms, and that the problem-specific algorithms are not simply a special case of the general-purpose algorithms. For the general RL agents to be on par with the problem-specific agents, design and tuning of both the state representation and reward function would be required, requiring effort which defeats the purpose of the agent's generality.

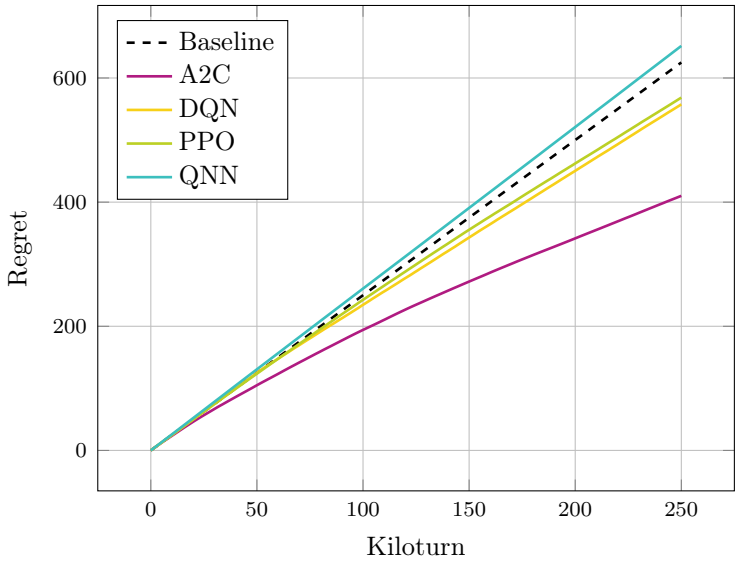


Figure 7.12: Reinforcement learning algorithms regrets for two Bernoulli arms with means 0.5 and 0.505. Each algorithm was 'trained' on the bandit problem from which the average regrets are shown. There two arms were the same as in fig. 7.1.

7.3 Discussion

In the above, the potential superiority of the quantum upper confidence bound algorithm was explored in solving the stochastic multi-armed bandit problem. Specifically, the QUCB algorithm was tested against its classical ancestor, the UCB algorithm, and the oftentimes best classical algorithm, namely Thompson sampling. These tests included static bandit instances where the means were fixed, and dynamic bandit instances the means were drawn from a prior distribution, from which Bayesian regrets were studied. Additionally, state-of-the-art reinforcement learning algorithms were tested on the bandit problem, where also a quantum policy agent was included.

From the experiments, it is evident that the QUCB algorithm indeed has the potential to outperform classical algorithms in the bandit problem. As seen in figs. 7.1 and 7.3, the algorithm does not only outperform the classical UCB algorithm, but for these particular instances, also the better Thompson sampling algorithm. However, as seen in figs. 7.2 and 7.5, settings wherein quantum advantage is not achieved also arise. While still performing better than classical UCB, the algorithm is outperformed by Thompson sampling for these 'easier' scenarios.

Most visibly in fig. 7.5, the QUCB algorithm can be very inefficient in the early turns of a bandit problem, after which it may be too late for its asymptotic advantages to be realised. With easy instances, such as the uniform prior in fig. 7.5, arm means are customarily relatively far apart. Accordingly, arms can be eliminated early with a high probability of the elimination being correct. The QUCB algorithm, however, does not take proper advantage thereof, and instead proceeds with its conservative strategy of producing quantum supreme estimates.

Figure 7.10 indicate that the QUCB regret distribution is more concentrated around the mean and is less affected by the actual bandit instance, but the regret curves in figs. 7.5, 7.7 and 7.9 show that it does positively matter which bandit instance is at hand. It is likely the asymptomatic regret properties of the QUCB algorithm, namely the logarithmic upper bound, that enables its regret to flatten out as the number of rounds increases, achieving on occasion better final regrets at the time horizons that were attempted. For truly great time horizons, it may be the case that the QUCB algorithm would always outperform the classical algorithms.

As briefly covered in the opening of this chapter, what can be considered the hyperparameter of the QUCB algorithm, that is its confidence level δ and the coefficient C_1, were not tuned or investigated in these experiments. The confidence level δ determines at which probability the upper bound, internally used by the algorithm to decide which arm to pull, is guaran-

teed to hold, and should according to the logarithmic bound presented in section 5.1.1 be set according to the inverse time horizon $1/T$. It was seen in the original paper and some preliminary testing for this thesis that fixing this value to a much higher $\delta = 0.01$ yields good results, at least for the instance in fig. 7.1. This may be a good value for all instances, but its certainty remains obscure, and it is as such something which could warrant deeper investigation. In a similar manner, the coefficient C_1 was arbitrarily set to 2. It is not clear whether this is a good value, or if it should be set differently for different instances. Overall, there may be better ways to tune the algorithm and extract lower regrets Still, changing these hyperparameters would not change the fundamental properties of the algorithm, particularly its flatter regret curves with an initial, potentially costly, learning period, and its tendencies that were observed here should likely remain.

In contrast to the QUCB algorithm, reinforcement learning algorithms did not seem to be well suited for solving the bandit problem. They do offer great power and flexibility, but only for problems correctly designed for them. The bandit problem, even with some help in explicit observations of the means, do not seem to work well with these algorithms. A quantum neural network policy could in principle inherit some QUCB advantages, but this would require a different, specialised data inputting scheme to allow direct input of the quantum states representing rewards. For it to be possible, another architecture would be needed, such as a quantum recurrent neural network. As it stands, the simpler, but specialised bandit algorithms appear to be the best choice for the bandit problem.

Chapter 8

Final remarks

8.1 Conclusions

This thesis has explored the multi-armed bandit problem and the application of quantum computing to solve it. Moreover, the application of modern reinforcement learning has also been explored, with the aim of solving the bandit problem using general reinforcement learning algorithms of both classical and quantum nature. The bandit problem is a widely applicable and useful problem that has been studied for many decades. As discussed in the introduction, it is indeed useful to real-world problems, and with computers automating yet more tasks, ensuring these are well-informed is pertinent. Despite its simplicity, the bandit problem remains intractable to solve optimally in most cases. In fact, defining precisely what an 'optimal' policy is for the bandit problem is a non-trivial task, and three main definitions are discussed in this thesis, namely instance-optimality, minimax-optimality and Bayesian optimality with respect to some prior distribution. Still, classical algorithms that achieve optimal performance with respect to either of these, though often only asymptotically, do exist. Perhaps most notable of these is the Thompson sampling algorithm, which is asymptotically instance-optimal and nearly minimax-optimal, and which tends to perform well in practice. Research in stochastic bandits has been active for decades, but is still very much ongoing.

Quantum computing has the potential to solve certain problems that are intractable to solve classically. Based on quantum Monte Carlo methods to estimate the mean of random variables with better precision than is classically permitted, the quantum upper bound algorithm proposed in [15] that is shown to achieve logarithmic minimax regret, something which is provably impossible classically, has been explained, implemented and studied. Simulating quantum computing is exponentially hard, but something that can be done with powerful classical hardware without any approximations.

The simulations done for this project show that the quantum upper confidence bound algorithm is only supreme on what can be considered 'hard' instances of the bandit problem, that is those in which reward means are close and for which many samples will be needed to thoroughly distinguish between them. For easy instances, where expected rewards are far apart, the algorithm is, in essence, wasting turns on sampling and producing unnecessarily accurate estimates of the reward means. It is subsequently significantly outperformed by the classical Thompson sampling algorithm. So far, the quantum algorithm had not been compared against Thompson sampling, such that highlighting the cases in which QUCB performs relatively poorly is here a novel contribution.

The algorithm's difficulty-dependent performance should not be entirely unexpected, as the algorithm relies on exponentially many consecutive samples to achieve its superior predictive performance, and in the easy instances, arms must be eliminated much earlier to achieve the lowest reasonable regret. The provably improved regret bound is no guarantee of improved performance in practice, being both a bound rather than a guarantee and asymptotic in nature. The algorithm is not a panacea for the bandit problem.

There were some arbitrary choices made in the implementation of the algorithm, primarily in the choices made for the QUCB hyperparameters. Different choices may be able to improve the algorithm's performance, but its overall behaviour is unlikely to change. Still, it would be interesting to study the effects of different choices of hyperparameters.

While the quantum upper bound algorithm may be useful for certain problems, it can not be applied to any bandit problem. Its quantum Monte Carlo subroutine relies on the rewards being observed as quantum mechanical superpositions corresponding to the reward distributions. For most, if not all, of the applications discussed in this thesis, this can not be achieved. '(...) human participants, unlike computer programs, cannot be queried in superposition.' [99]. This does not mean that the QUCB algorithm is purely an academic curiosity, however. Were quantum computing and quantum communication to advance to an interesting scale, quantum bandit algorithms could be used to enhance quantum computing-based algorithms in the same way classical bandit algorithms are used classically.

Modern reinforcement learning has shown the ability to solve some problems better than humans can. With neural networks and increasingly powerful computers, state-of-the-art methods can beat even the best humans at board games such as Go and video games like *Starcraft II* and *Dota 2*. Combining quantum computing with reinforcement learning has the goal

of elevating the performance of RL algorithms to a new level, and recent results have indeed shown some signs of variational quantum algorithms having certain advantages over classical algorithms. This is particularly true for problems with inherent quantum properties.

Attempts to solve the bandit problem with both classical reinforcement learning and quantum reinforcement learning left lacklustre results. It may be that these algorithms require proper tuning and design of both reward functions and state representations, as is often the case with RL problems. Nonetheless, it shows that bandit-specific algorithms are much better suited than general-purpose ones, despite the power provided by neural networks, quantum neural networks and great parameter counts.

In conclusion, this thesis has shown that the multi-armed bandit problem is a challenging and important problem that has been studied for many years. The application of quantum computing and reinforcement learning to bandit problems has shown promise, but more research is needed to determine their effectiveness in practice. The development of purpose-designed algorithms is crucial to achieving optimal performance in real-world applications.

8.2 Outlook

The pursuit of optimal policies for the multi-armed bandit problem remains a perennial subject of research. The algorithms explicated in this thesis, though applicable and theoretically sound, represent merely a fraction of the voluminous literature devoted to bandits. The fundamental, but basic stochastic bandit case is but the tip of the iceberg, as there exist a plethora of generalisations and variations, many of which provide even more useful tools to be applied in practice. Among these are contextual bandits, adversarial bandits, combinatorial bandits, linear bandits and many more. How the methods presented here may be extended to these cases is an interesting question.

By introducing superposition as a permissible mode of querying the arms, it is obvious that advantages will be gained. However, it is clear that there exist lower bounds which place limits on the performance attainable through these quantum bandit formulations. It may thus be an interesting pursuit to investigate the nature of these quantum lower bounds, both for minimax- and instance-optimality.

Furthermore, being limited to arms that provide superpositional rewards, it is a daunting task to apply quantum algorithms. As such, it would be of great interest to find any explicit scenarios wherein these algorithms could be used and provide real benefits. Could some quantum reinforcement

learning algorithms be made more efficient with QUCB?

The quantum bandit model considered in this thesis, where arms are pulled by applying corresponding oracles, is merely one of many plausible quantum bandit schematics. Several others have been proposed thus far, as touched upon in section 5.2, some of which may be more amenable to practical applications. These have necessarily other properties, and it would be interesting to study what quantised bandits be the most useful in practice.

The impressive strides taken in the field of reinforcement learning bear witness to the potential efficacy of applying quantum computing to this discipline. Nevertheless, it is plagued by the difficulties inherent in quantum machine learning more generally, whereby empirical demonstrations of real benefits remain elusive and intricate practical problems are often too complicated for exact, analytical considerations. Still, it is undoubtedly a promising field of research.

In future work, one may investigate whether the classical and quantum reinforcement learning algorithms considered here are truly incapable of producing a satisfactory solution to the bandit problem, or whether they simply necessitate more design and fine-tuning, which time constraints precluded here. It would also be interesting to investigate the use of other quantum reinforcement learning algorithms, such as a quantum value-based algorithm or some actor-critic algorithm where either or both are quantised. Overall, the study and comparison of these quantum reinforcement learning algorithms will be a fruitful area of research.

References

[1] Boye Gravningen Sjo. 'Quantum Neural Networks'. Industrial Mathematics, Specialisation Project (TMA4500) Report. Norwegian University of Science and Technology, 2022. URL: https://github.com/boyesjo/tma4500/.

[2] Jaya Kawale and Elliot Chow. 'A Multi-Armed Bandit Framework for Recommendations at Netflix'. Data Council. 2018. URL: https://www.datacouncil.ai/talks/a-multi-armed-bandit-framework-for-recommendations-at-netflix.

[3] Daniel N. Hill et al. 'An Efficient Bandit Algorithm for Realtime Multivariate Optimization'. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '17: The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Halifax NS Canada: ACM, 2017, pp. 1813–1821. ISBN: 978-1-4503-4887-4. DOI: 10.1145/3097983.3098184.

[4] Sam Daulton. 'Facebook Talk at the Netflix ML Platform Meetup (Part 3)'. Machine Learning Platform (Los Gatos). 2019. URL: https://youtu.be/A-JJvYaBPUU.

[5] Arjun Sharma. *Using a Multi-Armed Bandit with Thompson Sampling to Identify Responsive Dashers*. DoorDash Engineering Blog. 2022. URL: https://doordash.engineering/2022/03/15/using-a-multi-armed-bandit-with-thompson-sampling-to-identify-responsive-dashers/.

[6] Matteo Gagliolo and Jürgen Schmidhuber. 'Algorithm Selection as a Bandit Problem with Unbounded Losses'. In: *Learning and Intelligent Optimization*. Ed. by Christian Blum and Roberto Battiti. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 82–96. ISBN: 978-3-642-13800-3. DOI: 10.1007/978-3-642-13800-3_7.

[7] Jialei Wang et al. 'Online Feature Selection and Its Applications'. In: *IEEE Transactions on Knowledge and Data Engineering* 26.3 (2014), pp. 698–710. ISSN: 1558-2191. DOI: 10.1109/TKDE.2013.32.

[8] Djallel Bouneffouf et al. ‘Context Attentive Bandits: Contextual Bandit with Restricted Context’. In: 2017. DOI: 10.24963/ijcai.2017/203.

[9] Djallel Bouneffouf, Irina Rish and Charu Aggarwal. ‘Survey on Applications of Multi-Armed and Contextual Bandits’. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. 2020 IEEE Congress on Evolutionary Computation (CEC). Glasgow, United Kingdom: IEEE, 2020, pp. 1–8. ISBN: 978-1-72816-929-3. DOI: 10.1109/CEC48606.2020.9185782.

[10] Peter Whittle. ‘Discussion of Dr Gittins’ Paper’. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 41.2 (1979), pp. 164–177. ISSN: 2517-6161. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1979.tb01069.x.

[11] Richard P. Feynman. ‘Simulating Physics with Computers’. In: *International Journal of Theoretical Physics* 21.6-7 (1982), pp. 467–488. ISSN: 0020-7748, 1572-9575. DOI: 10.1007/BF02650179.

[12] Peter W. Shor. ‘Algorithms for Quantum Computation: Discrete Logarithms and Factoring’. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. Santa Fe, New Mexico, United States of America: IEEE, 1994, pp. 124–134. ISBN: 978-0-8186-6580-6. DOI: 10.1109/SFCS.1994.365700.

[13] Gorjan Alagic et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST Internal or Interagency Report (NISTIR) 8413. National Institute of Standards and Technology, 2022. DOI: 10.6028/NIST.IR.8413-upd1.

[14] Cybersecurity & Infrastructure Security Agenecy. *Prepare for a New Cryptographic Standard to Protect Against Future Quantum-Based Threats*. Prepare for a New Cryptographic Standard to Protect Against Future Quantum-Based Threats. 2022. URL: https://www.cisa.gov/news-events/alerts/2022/07/05/prepare-new-cryptographic-standard-protect-against-future-quantum-based-threats.

[15] Zongqi Wan et al. ‘Quantum Multi-Armed Bandits and Stochastic Linear Bandits Enjoy Logarithmic Regrets’. Version 1. In: (2022). DOI: 10.48550/ARXIV.2205.14988.

[16] Herbert Robbins. 'Some Aspects of the Sequential Design of Experiments'. In: *Bulletin of the American Mathematical Society* 58.5 (1952), pp. 527–535. ISSN: 0273-0979, 1088-9485. DOI: 10.1090/S0002-9904-1952-09620-8.

[17] William R. Thompson. 'On the Likelihood That One Unknown Probability Exceeds Another in View of Evidence of Two Samples'. In: *Biometrika* 25.3-4 (1933), pp. 285–294. ISSN: 0006-3444. DOI: 10.1093/biomet/25.3-4.285.

[18] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. 1st ed. Cambridge University Press, 2020. ISBN: 978-1-108-57140-1. DOI: 10.1017/9781108571401.

[19] Aleksandrs Slivkins. 'Introduction to Multi-Armed Bandits'. In: *Foundations and Trends® in Machine Learning* 12.1-2 (2019), pp. 1–286. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000068.

[20] T.L Lai and Herbert Robbins. 'Asymptotically Efficient Adaptive Allocation Rules'. In: *Advances in Applied Mathematics* 6.1 (1985), pp. 4–22. ISSN: 01968858. DOI: 10.1016/0196-8858(85)90002-8.

[21] Peter Auer et al. 'The Nonstochastic Multiarmed Bandit Problem'. In: *SIAM Journal on Computing* 32.1 (2002), pp. 48–77. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/S0097539701398375.

[22] Pierre Ménard and Aurélien Garivier. 'A Minimax and Asymptotically Optimal Algorithm for Stochastic Bandits'. Version 2. In: (2017). DOI: 10.48550/ARXIV.1702.07211.

[23] Tianyuan Jin et al. 'MOTS: Minimax Optimal Thompson Sampling'. Version 3. In: (2020). DOI: 10.48550/ARXIV.2003.01803.

[24] Sebastian Pilarski, Slawomir Pilarski and Dániel Varró. 'Optimal Policy for Bernoulli Bandits: Computation and Algorithm Gauge'. In: *IEEE Transactions on Artificial Intelligence* 2.1 (2021), pp. 2–17. ISSN: 2691-4581. DOI: 10.1109/TAI.2021.3074122.

[25] Peter Auer, Nicolò Cesa-Bianchi and Paul Fischer. 'Finite-Time Analysis of the Multiarmed Bandit Problem'. In: *Machine Learning* 47.2 (2002), pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352.

[26] Sébastien Bubeck. 'Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems'. In: *Foundations and Trends® in Machine Learning* 5.1 (2012), pp. 1–122. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000024.

[27] Lihong Li et al. 'A Contextual-Bandit Approach to Personalized News Article Recommendation'. In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10: The 19th International World Wide Web Conference. Raleigh North Carolina USA: ACM, 2010, pp. 661–670. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772758.

[28] Jean-Yves Audibert and Sëbastien Bubeck. 'Minimax Policies for Adversarial and Stochastic Bandits'. In: *Colt* 7 (2009), pp. 217–226.

[29] Jean-Yves Audibert, Rémi Munos and Csaba Szepesvári. 'Exploration–Exploitation Tradeoff Using Variance Estimates in Multi-Armed Bandits'. In: *Theoretical Computer Science* 410.19 (2009), pp. 1876–1902. ISSN: 03043975. DOI: 10.1016/j.tcs.2009.01.016.

[30] Odalric-Ambrym Maillard, Rémi Munos and Gilles Stoltz. 'A Finite-Time Analysis of Multi-armed Bandits Problems with Kullback-Leibler Divergences'. Version 1. In: (2011). DOI: 10.48550/ARXIV.1105.5820.

[31] Emilie Kaufmann, Nathaniel Korda and Rémi Munos. 'Thompson Sampling: An Asymptotically Optimal Finite-Time Analysis'. In: *Algorithmic Learning Theory*. Ed. by Nader H. Bshouty et al. Red. by David Hutchison et al. Vol. 7568. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 199–213. ISBN: 978-3-642-34105-2. DOI: 10.1007/978-3-642-34106-9_18.

[32] Shipra Agrawal and Navin Goyal. 'Further Optimal Regret Bounds for Thompson Sampling'. In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. PMLR, 2013, pp. 99–107. URL: https://proceedings.mlr.press/v31/agrawal13a.html.

[33] Shipra Agrawal and Navin Goyal. 'Near-Optimal Regret Bounds for Thompson Sampling'. In: *Journal of the ACM* 64.5 (2017), 30:1–30:24. ISSN: 0004-5411. DOI: 10.1145/3088510.

[34] Junya Honda and Akimichi Takemura. 'Optimality of Thompson Sampling for Gaussian Bandits Depends on Priors'. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Vol. 33. Reykjavik, Iceland}: PMLR, 2014, pp. 375–383. arXiv: 1311.1894 [math, stat]. URL: http://proceedings.mlr.press/v33/honda14.pdf.

[35] Peter Auer et al. 'Gambling in a rigged casino: the adversarial multi-armed bandit problem'. In: *Annual Symposium on Foundations of Computer Science - Proceedings*. Proceedings of the 1995 IEEE 36th Annual Symposium on Foundations of Computer Science. 1995, pp. 322–331. URL: https://collaborate.princeton.edu/en/publications/gambling-in-a-rigged-casino-the-adversarial-multi-armed-bandit-pr.

[36] Lilian Besson and Emilie Kaufmann. 'What Doubling Tricks Can and Can't Do for Multi-Armed Bandits'. Version 1. In: (2018). DOI: 10.48550/ARXIV.1803.06971.

[37] David Silver et al. 'Mastering the Game of Go with Deep Neural Networks and Tree Search'. In: *Nature* 529.7587 (7587 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961.

[38] Michel Tokic and Günther Palm. 'Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax'. In: *Lecture Notes in Computer Science* (2011). DOI: 10.1007/978-3-642-24455-1_33.

[39] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, 2nd Ed.* Reinforcement Learning: An Introduction, 2nd Ed. Cambridge, MA, US: The MIT Press, 2018, pp. xxii, 526. xxii, 526. ISBN: 978-0-262-03924-6.

[40] Greg Brockman et al. 'OpenAI Gym'. Version 1. In: (2016). DOI: 10.48550/ARXIV.1606.01540.

[41] Andrew G. Barto, Richard S. Sutton and Charles W. Anderson. 'Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems'. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5 (1983), pp. 834–846. ISSN: 2168-2909. DOI: 10.1109/TSMC.1983.6313077.

[42] Mojang Studios. *Minecraft*. [Windows PC]. Sweden: Mojang Studios, 2011.

[43] Danijar Hafner et al. *Mastering Diverse Domains through World Models*. Version 1. 2023. arXiv: 2301.04104 [cs, stat]. URL: http://arxiv.org/abs/2301.04104. preprint.

[44] Tom Murphy. *The First Level of Super Mario Bros. Is Easy with Lexicographic Orderings and Time Travel*. 2013. URL: http://tom7.org/mario/mario.pdf. preprint.

[45] Utopia Software. *Montezuma's Revenge*. [Atari 2600]. United States of America: Parker Brothers, 1984.

[46] Tim Salimans and Richard Chen. *Learning Montezuma's Revenge from a Single Demonstration*. 2018. DOI: 10.48550/arXiv.1812.03381. arXiv: 1812.03381 [cs, stat]. preprint.

[47] Rachit Dubey et al. 'Investigating Human Priors for Playing Video Games'. In: *ICML 2018*. Thirty-Fifth International Conference on Machine Learning. Stockholm, 2018. DOI: 10.48550/arXiv.1802.10217. arXiv: 1802.10217 [cs].

[48] Shaden Smith et al. *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model*. 2022. DOI: 10.48550/arXiv.2201.11990. arXiv: 2201.11990 [cs]. preprint.

[49] OpenAI. *GPT-4 Technical Report*. 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs]. preprint.

[50] Preetum Nakkiran et al. 'Deep Double Descent: Where Bigger Models and More Data Hurt'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (2021), p. 124003. ISSN: 1742-5468. DOI: 10.1088/1742-5468/ac3a74.

[51] Izaak Neutelings. *Neural networks*. Licensed under a Creative Commons Attribution-ShareAlike 4.0 International License with © Copyright 2021 – TikZ.net. 2021. URL: https://tikz.net/neural_networks/.

[52] Kai Arulkumaran et al. 'Deep Reinforcement Learning: A Brief Survey'. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. ISSN: 1558-0792. DOI: 10.1109/MSP.2017.2743240.

[53] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. DOI: 10.48550/arXiv.1312.5602. arXiv: 1312.5602 [cs]. preprint.

[54] Volodymyr Mnih et al. 'Human-Level Control through Deep Reinforcement Learning'. In: *Nature* 518.7540 (7540 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236.

[55] Hado van Hasselt, Arthur Guez and David Silver. 'Deep Reinforcement Learning with Double Q-Learning'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (1 2016). ISSN: 2374-3468. DOI: 10.1609/aaai.v30i1.10295.

[56] Tom Schaul et al. 'Prioritized Experience Replay'. In: (2015).

[57] Ziyu Wang et al. 'Dueling Network Architectures for Deep Reinforcement Learning'. In: *Proceedings of The 33rd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, 2016, pp. 1995–2003. URL: https://proceedings.mlr.press/v48/wangf16.html.

[58] Ronald J. Williams. 'Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning'. In: *Machine Learning* 8.3 (1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696.

[59] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs]. URL: http://arxiv.org/abs/1707.06347. preprint.

[60] John Schulman et al. 'Trust Region Policy Optimization'. In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, 2015, pp. 1889–1897. URL: https://proceedings.mlr.press/v37/schulman15.html.

[61] Valve. *Dota 2*. [Windows PC]. United States of America: Valve, 2013.

[62] Greg Brockman et al. *OpenAI Five*. OpenAI Blog. 2018. URL: https://openai.com/research/openai-five.

[63] OpenAI. *OpenAI Five Defeats Dota 2 World Champions*. OpenAI Blog. 2019. URL: https://openai.com/research/openai-five-defeats-dota-2-world-champions.

[64] Vijay Konda and John Tsitsiklis. 'Actor-Critic Algorithms'. In: *Advances in Neural Information Processing Systems*. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

[65] Volodymyr Mnih et al. 'Asynchronous Methods for Deep Reinforcement Learning'. In: *Proceedings of The 33rd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, 2016, pp. 1928–1937. URL: https://proceedings.mlr.press/v48/mniha16.html.

[66] Blizzard Entertainment. *StarCraft II*. [Windows PC]. United States of America: Blizzard Entertainment, 2010.

[67] Oriol Vinyals et al. 'Grandmaster Level in StarCraft II Using Multi-Agent Reinforcement Learning'. In: *Nature* 575.7782 (7782 2019), pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z.

[68] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 1st ed. Cambridge University Press, 2012. ISBN: 978-0-511-97666-7. DOI: 10.1017/CBO9780511976667.

[69] Amira Abbas et al. *Learn Quantum Computation Using Qiskit*. 2020. URL: https://qiskit.org/textbook/.

[70] Smite Meister. *Bloch sphere*. 2009. URL: https://upload.wikimedia.org/wikipedia/commons/6/6b/Bloch_sphere.svg.

[71] Giacomo Torlai et al. 'Precise Measurement of Quantum Observables with Neural-Network Estimators'. In: *Physical Review Research* 2.2 (2020), p. 022060. ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.2.022060.

[72] David Deutsch and Richard Jozsa. 'Rapid Solution of Problems by Quantum Computation'. In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439.1907 (1992), pp. 553–558. ISSN: 0962-8444, 2053-9177. DOI: 10.1098/rspa.1992.0167.

[73] D.R. Simon. 'On the Power of Quantum Computation'. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 35th Annual Symposium on Foundations of Computer Science. Santa Fe, NM, USA: IEEE Comput. Soc. Press, 1994, pp. 116–123. ISBN: 978-0-8186-6580-6. DOI: 10.1109/SFCS.1994.365701.

[74] Scott Aaronson. 'BQP and the Polynomial Hierarchy'. In: *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*. STOC '10. New York, NY, USA: Association for Computing Machinery, 5th June 2010, pp. 141–150. ISBN: 978-1-4503-0050-6. DOI: 10.1145/1806689.1806711.

[75] Niklas Johansson and Jan-Åke Larsson. 'Efficient Classical Simulation of the Deutsch–Jozsa and Simon's Algorithms'. In: *Quantum Information Processing* 16.9 (2017), p. 233. ISSN: 1570-0755, 1573-1332. DOI: 10.1007/s11128-017-1679-7.

[76] Danial Dervovic et al. 'Quantum Linear Systems Algorithms: A Primer'. 2018. DOI: 10.48550/ARXIV.1802.08227. arXiv: 1802.08227 [quant-ph].

[77] Scott Aaronson. 'Read the Fine Print'. In: *Nature Physics* 11.4 (2015), pp. 291–293. ISSN: 1745-2473, 1745-2481. DOI: 10.1038/nphys3272.

[78] Lov K. Grover. 'A Fast Quantum Mechanical Algorithm for Database Search'. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. Philadelphia, Pennsylvania, United States of America: ACM Press, 1996, pp. 212–219. ISBN: 978-0-89791-785-8. DOI: 10.1145/237814.237866.

[79] Christof Zalka. 'Grover's Quantum Searching Algorithm Is Optimal'. In: *Physical Review A* 60.4 (1999), pp. 2746–2751. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.60.2746.

[80] Gilles Brassard et al. 'Quantum Amplitude Amplification and Estimation'. In: *Contemporary Mathematics*. Ed. by Samuel J. Lomonaco and Howard E. Brandt. Vol. 305. Providence, Rhode Island: American Mathematical Society, 2002, pp. 53–74. ISBN: 978-0-8218-2140-4. DOI: 10.1090/conm/305/05215.

[81] Ashley Montanaro. 'Quantum Speedup of Monte Carlo Methods'. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471.2181 (2015), p. 20150301. ISSN: 1364-5021, 1471-2946. DOI: 10.1098/rspa.2015.0301.

[82] Yohichi Suzuki et al. 'Amplitude Estimation without Phase Estimation'. In: *Quantum Information Processing* 19.2 (2020), p. 75. ISSN: 1570-0755, 1573-1332. DOI: 10.1007/s11128-019-2565-2. arXiv: 1904.10246 [quant-ph].

[83] Kouhei Nakaji. 'Faster Amplitude Estimation'. In: *Quantum Information and Computation* 20 (2020), pp. 1109–1123. DOI: 10.26421/QIC20.13-14-2.

[84] Dmitry Grinko et al. 'Iterative Quantum Amplitude Estimation'. In: *npj Quantum Information* 7.1 (2021), p. 52. ISSN: 2056-6387. DOI: 10.1038/s41534-021-00379-1. arXiv: 1912.05559 [quant-ph].

[85] David Ceperley and Berni Alder. 'Quantum Monte Carlo'. In: *Science* 231.4738 (1986), pp. 555–560. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.231.4738.555.

[86] Brian M. Austin, Dmitry Yu. Zubarev and William A. Lester. 'Quantum Monte Carlo and Related Approaches'. In: *Chemical Reviews* 112.1 (2012), pp. 263–288. ISSN: 0009-2665, 1520-6890. DOI: 10.1021/cr2001564.

[87] J. E. Gubernatis, N. Kawashima and P. Werner. *Quantum Monte Carlo Methods: Algorithms for Lattice Models*. Cambridge: Cambridge University Press, 2016. 488 pp. ISBN: 978-1-107-00642-3.

[88] Paul Dagum et al. 'An Optimal Algorithm for Monte Carlo Estimation'. In: *SIAM Journal on Computing* 29.5 (2000), pp. 1484–1496. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/S0097539797315306.

[89] Frank Arute et al. 'Quantum Supremacy Using a Programmable Superconducting Processor'. In: *Nature* 574.7779 (7779 2019), pp. 505–510. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5.

[90] Han-Sen Zhong et al. 'Quantum Computational Advantage Using Photons'. In: *Science* 370.6523 (2020), pp. 1460–1463. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.abe8770.

[91] Lars S. Madsen et al. 'Quantum Computational Advantage with a Programmable Photonic Processor'. In: *Nature* 606.7912 (7912 2022), pp. 75–81. ISSN: 1476-4687. DOI: 10.1038/s41586-022-04725-x.

[92] IBM Quantum Compute Resources. *IBM Montreal: Falcon r4 (1.11.26)*. 2022. URL: https://quantum-computing.ibm.com/services/resources?tab=systems&system=ibmq_montreal&order=processorType%20DESC.

[93] Craig Gidney and Martin Ekerå. 'How to Factor 2048 Bit RSA Integers in 8 Hours Using 20 Million Noisy Qubits'. In: *Quantum* 5 (2021), p. 433. ISSN: 2521-327X. DOI: 10.22331/q-2021-04-15-433.

[94] Yulian Wu et al. 'Quantum Heavy-tailed Bandits'. Version 1. In: (2023). DOI: 10.48550/ARXIV.2301.09680.

[95] Wei Hu and James Hu. 'Training a Quantum Neural Network to Solve the Contextual Multi-Armed Bandit Problem'. In: *Natural Science* 11.01 (2019), pp. 17–27. ISSN: 2150-4091, 2150-4105. DOI: 10.4236/ns.2019.111003.

[96] Shrigyan Brahmachari, Josep Lumbreras and Marco Tomamichel. 'Quantum Contextual Bandits and Recommender Systems for Quantum Data'. Version 1. In: (2023). DOI: 10.48550/ARXIV.2301.13524.

[97] Josep Lumbreras, Erkka Haapasalo and Marco Tomamichel. 'Multi-Armed Quantum Bandits: Exploration versus Exploitation When Learning Properties of Quantum States'. In: *Quantum* 6 (2022), p. 749. ISSN: 2521-327X. DOI: 10.22331/q-2022-06-29-749.

[98] Balthazar Casalé et al. 'Quantum Bandits'. In: *Quantum Machine Intelligence* 2.1 (2020), p. 11. ISSN: 2524-4906, 2524-4914. DOI: 10.1007/s42484-020-00024-8.

[99] Daochen Wang et al. 'Quantum Exploration Algorithms for Multi-Armed Bandits'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.11 (11 2021), pp. 10102–10110. ISSN: 2374-3468. DOI: 10.1609/aaai.v35i11.17212.

[100] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-96423-2. DOI: 10.1007/978-3-319-96424-9.

[101] Timo Felser et al. 'Quantum-Inspired Machine Learning on High-Energy Physics Data'. In: *npj Quantum Information* 7.1 (2021), p. 111. ISSN: 2056-6387. DOI: 10.1038/s41534-021-00443-w.

[102] Yunchao Liu, Srinivasan Arunachalam and Kristan Temme. 'A Rigorous and Robust Quantum Speed-up in Supervised Machine Learning'. In: *Nature Physics* 17.9 (2021), pp. 1013–1017. ISSN: 1745-2473, 1745-2481. DOI: 10.1038/s41567-021-01287-z.

[103] Nathan Wiebe, Daniel Braun and Seth Lloyd. 'Quantum Algorithm for Data Fitting'. In: *Physical Review Letters* 109.5 (2012), p. 050505. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.109.050505.

[104] Marcello Benedetti et al. 'Parameterized Quantum Circuits as Machine Learning Models'. In: *Quantum Science and Technology* 4.4 (2019), p. 043001. ISSN: 2058-9565. DOI: 10.1088/2058-9565/ab4eb5.

[105] Maria Schuld, Ryan Sweke and Johannes Jakob Meyer. 'The Effect of Data Encoding on the Expressive Power of Variational Quantum Machine Learning Models'. In: *Physical Review A* 103.3 (2021), p. 032430. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.103.032430. arXiv: 2008.08605 [quant-ph, stat].

[106] Marco Cerezo et al. 'Variational Quantum Algorithms'. In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00348-9.

[107] Maria Schuld et al. 'Evaluating Analytic Gradients on Quantum Hardware'. In: *Physical Review A* 99.3 (2019), p. 032331. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.99.032331.

[108] Ryan Sweke et al. 'Stochastic Gradient Descent for Hybrid Quantum-Classical Optimization'. In: *Quantum* 4 (2020), p. 314. ISSN: 2521-327X. DOI: 10.22331/q-2020-08-31-314. arXiv: 1910.01155.

[109] Patrick Huembeli and Alexandre Dauphin. 'Characterizing the Loss Landscape of Variational Quantum Circuits'. In: *Quantum Science and Technology* 6.2 (2021), p. 025011. ISSN: 2058-9565. DOI: 10.1088/2058-9565/abdbc9.

[110] Jarrod R. McClean et al. 'Barren Plateaus in Quantum Neural Network Training Landscapes'. In: *Nature Communications* 9.1 (2018), p. 4812. ISSN: 2041-1723. DOI: 10.1038/s41467-018-07090-4. arXiv: 1803.11173.

[111] Marco Cerezo et al. 'Cost Function Dependent Barren Plateaus in Shallow Parametrized Quantum Circuits'. In: *Nature Communications* 12.1 (1 2021), p. 1791. ISSN: 2041-1723. DOI: 10.1038/s41467-021-21728-w.

[112] Suguru Endo et al. 'Hybrid Quantum-Classical Algorithms and Quantum Error Mitigation'. In: *Journal of the Physical Society of Japan* 90.3 (2021), p. 032001. ISSN: 0031-9015, 1347-4073. DOI: 10.7566/JPSJ.90.032001.

[113] Amira Abbas et al. 'The Power of Quantum Neural Networks'. In: *Nature Computational Science* 1.6 (2021), pp. 403–409. ISSN: 2662-8457. DOI: 10.1038/s43588-021-00084-1.

[114] Vojtech Havlicek et al. 'Supervised Learning with Quantum Enhanced Feature Spaces'. In: *Nature* 567.7747 (2019), pp. 209–212. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-019-0980-2. arXiv: 1804.11326 [quant-ph, stat].

[115] Maria Schuld and Francesco Petruccione. *Machine Learning with Quantum Computers*. Quantum Science and Technology. Cham: Springer International Publishing, 2021. ISBN: 978-3-030-83097-7. DOI: 10.1007/978-3-030-83098-4.

[116] Maria Schuld and Nathan Killoran. 'Quantum Machine Learning in Feature Hilbert Spaces'. In: *Physical Review Letters* 122.4 (2019), p. 040504. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.122.040504.

[117] Iris Cong, Soonwon Choi and Mikhail D. Lukin. 'Quantum Convolutional Neural Networks'. In: *Nature Physics* 15.12 (2019), pp. 1273–1278. ISSN: 1745-2473, 1745-2481. DOI: 10.1038/s41567-019-0648-8.

[118] Yudong Cao, Gian Giacomo Guerreschi and Alán Aspuru-Guzik. 'Quantum Neuron: An Elementary Building Block for Machine Learning on Quantum Computers'. 2017. DOI: 10.48550/ARXIV.1711.11240. arXiv: 1711.11240 [quant-ph].

[119] Shilu Yan, Hongsheng Qi and Wei Cui. 'Nonlinear Quantum Neuron: A Fundamental Building Block for Quantum Neural Networks'. In: *Physical Review A* 102.5 (2020), p. 052421. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.102.052421.

[120] Seunghyeok Oh, Jaeho Choi and Joongheon Kim. 'A Tutorial on Quantum Convolutional Neural Networks (QCNN)'. In: *International Conference on Information and Communication Technology Convergence*. Jeju Island, Republic of Korea: IEEE, 2020, pp. 236–239. ISBN: 978-1-72816-758-9. DOI: 10.1109/ICTC49870.2020.9289439.

[121] Arthur Pesah et al. 'Absence of Barren Plateaus in Quantum Convolutional Neural Networks'. In: *Physical Review X* 11.4 (2021), p. 041011. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.11.041011.

[122] Xun Gao, Z.-Y. Zhang and Luming Duan. 'A Quantum Machine Learning Algorithm Based on Generative Models'. In: *Science Advances* 4.12 (2018), eaat9004. ISSN: 2375-2548. DOI: 10.1126/sciadv.aat9004.

[123] He-Liang Huang et al. 'Experimental Quantum Generative Adversarial Networks for Image Generation'. In: *Physical Review Applied* 16.2 (2021), p. 024051. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.16.024051.

[124] Nathan Killoran et al. 'Continuous-Variable Quantum Neural Networks'. In: *Physical Review Research* 1.3 (2019), p. 033063. ISSN: 2643-1564. DOI: 10.1103/PhysRevResearch.1.033063.

[125] Maxwell Henderson et al. 'Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits'. In: *Quantum Machine Intelligence* 2.1 (2020), p. 2. ISSN: 2524-4906, 2524-4914. DOI: 10.1007/s42484-020-00012-y.

[126] Yi Zeng et al. 'A Multi-Classification Hybrid Quantum Neural Network Using an All-Qubit Multi-Observable Measurement Strategy'. In: *Entropy* 24.3 (2022), p. 394. ISSN: 1099-4300. DOI: 10.3390/e24030394.

[127] Johannes Bausch. 'Recurrent Quantum Neural Networks'. In: *Advances in neural information processing systems* 33 (2020), pp. 1368–1379.

[128] Yuto Takaki et al. 'Learning Temporal Data with a Variational Quantum Recurrent Neural Network'. In: *Physical Review A* 103.5 (2021), p. 052414. DOI: 10.1103/PhysRevA.103.052414.

[129] Ming-Sheng Ying, Yuan Feng and Sheng-Gang Ying. 'Optimal Policies for Quantum Markov Decision Processes'. In: *International Journal of Automation and Computing* 18.3 (2021), pp. 410–421. ISSN: 1751-8520. DOI: 10.1007/s11633-021-1278-z.

[130] Samuel Yen-Chi Chen et al. 'Variational Quantum Circuits for Deep Reinforcement Learning'. In: *IEEE Access* 8 (2020), pp. 141007–141024. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3010470.

[131] Andrea Skolik, Sofiene Jerbi and Vedran Dunjko. 'Quantum Agents in the Gym: A Variational Quantum Algorithm for Deep Q-learning'. In: *Quantum* 6 (2022), p. 720. ISSN: 2521-327X. DOI: 10.22331/q-2022-05-24-720. arXiv: 2103.15084 [quant-ph].

[132] Sofiene Jerbi et al. 'Quantum Enhancements for Deep Reinforcement Learning in Large Spaces'. In: *PRX Quantum* 2.1 (2021), p. 010328. DOI: 10.1103/PRXQuantum.2.010328.

[133] V. Saggio et al. 'Experimental Quantum Speed-up in Reinforcement Learning Agents'. In: *Nature* 591.7849 (2021), pp. 229–233. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-021-03242-7.

[134] Arne Hamann and Sabine Wölk. 'Performance Analysis of a Hybrid Agent for Quantum-Accessible Reinforcement Learning'. In: *New Journal of Physics* 24.3 (2022), p. 033044. ISSN: 1367-2630. DOI: 10.1088/1367-2630/ac5b56.

[135] Sofiene Jerbi et al. *Parametrized Quantum Policies for Reinforcement Learning*. 2021. arXiv: 2103.05577 [quant-ph, stat]. URL: http://arxiv.org/abs/2103.05577. preprint.

[136] Ville Bergholm et al. *PennyLane: Automatic Differentiation of Hybrid Quantum-Classical Computations*. 2022. DOI: 10.48550/arXiv.1811.04968. arXiv: 1811.04968 [physics, physics:quant-ph]. preprint.

[137] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[138] Guido van Rossum and Fred L. Drake. *The Python Language Reference*. Release 3.0.1 [Repr.] Python Documentation Manual / Guido van Rossum; Fred L. Drake [Ed.] Pt. 2. Hampton, NH: Python Software Foundation, 2010. 109 pp. ISBN: 978-1-4414-1269-0.

[139] Matthew Treinish et al. *Qiskit: An Open-source Framework for Quantum Computing*. Version 0.39.2. Zenodo, 2022. DOI: 10.5281/ZENODO.2573505.

[140] Charles R. Harris et al. 'Array Programming with NumPy'. In: *Nature* 585.7825 (17th Sept. 2020), pp. 357–362. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-020-2649-2.

[141] Pauli Virtanen et al. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17.3 (2nd Mar. 2020), pp. 261–272. ISSN: 1548-7091, 1548-7105. DOI: 10.1038/s41592-019-0686-2.

[142] Antonin Raffin et al. 'Stable-Baselines3: Reliable Reinforcement Learning Implementations'. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

Figures

Tables

Algorithms