Multi-armed Banditry and Quantum Agents

Contents

Abstract

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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Sammendrag

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

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 chapters on machine learning and quantum computing are based on my project report [1], often verbatim.

[illegible]

Boye Gravningen Sjo
Trondhjem, Norge
12th of June 2023

Chapter 1

Introduction

Chapter 2

Multi-armed bandits

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 [2], its study actually dates back to 1933 [3]. 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 maximise the number of patients treated with the best treatment in the long run?

Some of the many real-world setting where the multi-armed bandit problem is applicable are listed in table 2.1. Despite being a simple problem, its countless variants and applications make it not only a useful tool for a plethora of real-world problems. Netflix uses MAB theory to recommend movies [4], Amazon for its website layout [5], Facebook for video compression [6] and Doordash to identify responsive deliverymen [7]. 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 [8], where algorithms discussed in this chapter have shown success [9, 10].

This chapter and its notation will mostly follow the work textbook [11], to which the interested reader is referred for more details on the bandit problem, its variants, applications and history.

Table 2.1: Some applications of the multi-armed bandit problem. Arms correspond to the different actions that an agent can take, initially with unknown outcomes. 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	Number of clicks
Website design	Layouts/fonts &c.	Number of clicks
Recommendation systems	Items	Number of clicks
Dynamic pricing	Prices	Profit
Networking	Routes, settings	Ping
Lossy compression	Compression settings	Quality preserved
Tasking employees	Which employee	Productivity
Finance	Investment options	Profit

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 $\mathcal{A} = \{1, \ldots, k\}$, but not the reward distributions $\nu = \{P_a : a \in \mathcal{A}\}$. For each time step $t = 1, \ldots, T$, 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 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. The goal of the agent is 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 2.2. Note than some

Table 2.2: 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 is generally the specific distribution of the rewards, but may also being general properties that the distributions must satisfy, giving rise to non-parametric classes.

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

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; 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.

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.

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 dependence 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. 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 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 omitted for clarity 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 to mainly pull good arms, as these will need more consideration to be distinguished from the optimal.

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 [12, 11].

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, 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 describing its worst performance over all instances in the class.

2.2.1 Instance-dependent lower bound

In order to meaningfully define a lower bound, it is imperative to assume a reasonable algorithm. Otherwise, trivial policies, such as always pulling the first arm, could achieve zero regret, hindering any meaningful comparison. A useful 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

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

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 [13] holds. It 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 known makes it inapplicable to real-world problems, and as reward distributions approach the optimal distribution, the bound diverges.

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 algorithm 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 in all. In [14], 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

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

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 [15, 16].

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

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

Table 2.3: 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 assumes 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

where Q is the prior distribution over $\mathcal{E}$. Is it trivially observed that the Bayesian regret bounded above by the minimax regret. However, it can be proven that there exists priors such that the Bayesian regret is bounded below by some $\Omega(\sqrt{KT})$, such that $\sup_Q \inf_\pi \mathrm{BR}_T(Q, \pi) = \Theta(\sqrt{KT})$ [11]

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 [11]. In simpler cases, dynamic programming can be fruitful. With Bernoulli bandits and clever implementations, optimal strategies can indeed be found [17]. 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.

2.3 Strategies

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 $\mathcal{A}$ uniformly
2 return $a$
```

It is easy to understand that the regret is

$$R_T = T\left(\mu^* - \frac{1}{k}\sum_{i=1}^{k}\mu_i\right), \tag{2.16}$$

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

```
1 if $t \leq mk$ then
2     return $(t \mod k) + 1$
3 else
4     return $\text{argmax}_{a\in\mathcal{A}}\, \hat{\mu}_a$
```

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 into 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

```
1 if $t \leq mk$ then
2     return $(t \mod k) + 1$
3 else
4     Sample $u$ from $[0, 1)$ uniformly
5     if $u < \epsilon$ then
6         Sample $a$ from $\mathcal{A}$ uniformly
7         return $a$
8     else
9         return $\operatorname{argmax}_{a \in \mathcal{A}} \hat{\mu}_a$
```

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 $\epsilon \sim 1/t$, which has been shown to achieve logarithmic instance-dependent regret [18]. It is worth noting, however, that the optimal decay rate depends on the specific instance, and achieving logarithmic regret can be challenging in practice [19].

2.3.4 Upper confidence bounds

The upper confidence bound (UCB) algorithm is a family of more sophisticated algorithm based on estimating an upper bound for the mean of each arm, and they are commonly used in practice [**NEEDED**]. One always chooses the arm whose upper confidence bound is highest, 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, one has

$$p = P(\mu_a > \hat{\mu_a} + \mathrm{UCB}_a) \leq \exp\left(-2T_a \mathrm{UCB}_a^2\right), \tag{2.17}$$

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 gives

$$\mathrm{UCB}_a = \sqrt{\frac{-\ln p}{2T_a}}. \tag{2.18}$$

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

$$\mathrm{UCB}_a = \sqrt{\frac{2\ln t}{T_a}}, \tag{2.19}$$

which is a common choice for the upper confidence bound, leading to the UCB1-algorithm. In [18], it is shown that UCB1 achieves

$$R_T \leq \left(8 \sum_{a \in \mathcal{A} \setminus a^*} \frac{\log T}{\Delta_a}\right) + \left(1 + \frac{\pi^2}{3}\right) \sum_{a \in \mathcal{A}} \Delta_a, \tag{2.20}$$

which misses instance-dependent optimality only by some constant factor. Nor is UCB1 truly minimax-optimal, as its instance-independent upper bound is $O(\sqrt{kT \log T})$ [19].

Regardless of the assumptions made and the bandit class, the procedure for UCB strageies follows as in algorithm 2.4. For few arms, the process is easily visualised, as is done in line 4. 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 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 $\mathcal{E}^k_{[0,1]}$ [20]. Further, incorporating estimates of second moments improve performance in some cases [21], while incorporating the whole empirical distributions of observed rewards appears be the most effective approach [22].

Algorithm 2.4: UCB arm selection

```
1 if $t \leq k$ then
2     return $t$
3 else
4     return $\mathrm{argmax}_{a \in \mathcal{A}}(\hat{\mu}_a + \mathrm{UCB}_a)$
```

2.3.5 Thompson sampling

Thompson sampling is a Bayesian approach to the multi-armed bandit problem, being the original approach to the problem [3] 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. The posterior distribution is updated after each pull, using the observed reward as evidence. By graphing the posterior distribution, such as is done in section 2.3.5, it is possible to see how the algorithm efficiently exploits the best arm while giving enough explorative efforts.

It was first in 2012 that Thompson sampling was proven asymptotically optimal for Bernoulli rewards with uniform priors [23]. Namely, for every $\epsilon > 0$, there exists a constant C_ϵ such that

$$R_T \leq (1+\epsilon) \sum_{a \in \mathcal{A} \setminus a^*} \Delta_a \frac{\log T - \log\log T}{D(P_a \parallel P^*)} + C_\epsilon, \tag{2.21}$$

where P_a is the reward distribution of arm a and P^* is the reward distribution of the best arm. Meanwhile, the minimax regret is can be bounded by either $O(\sqrt{kT \log T})$ [24] or $O(\sqrt{kT \log k})$ [25], neither of which is quite minimax-optimal.

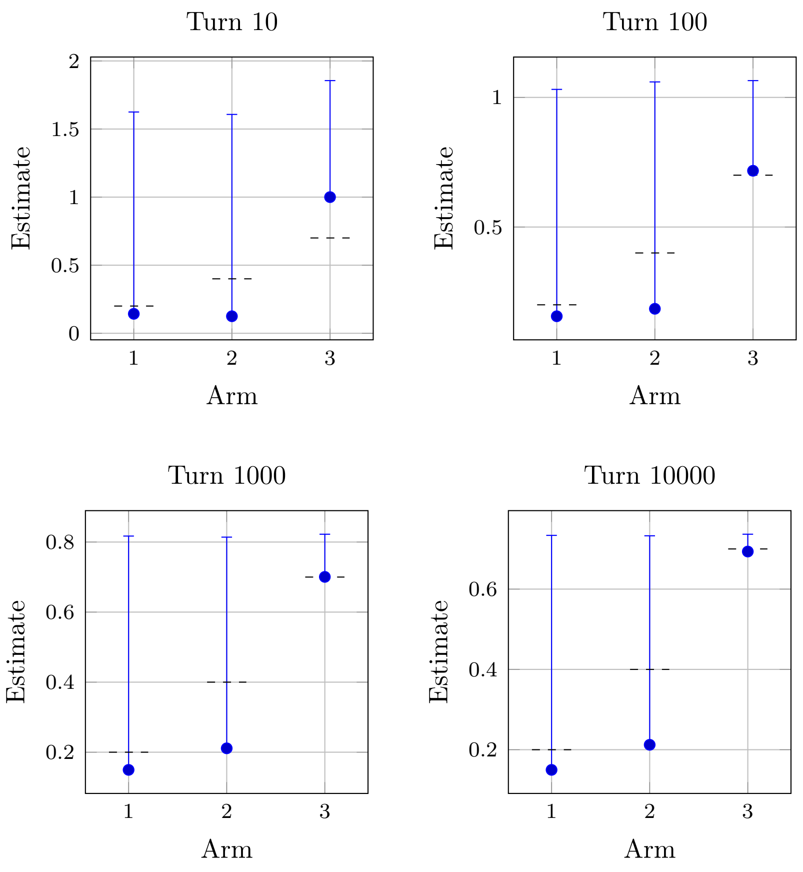


Figure 2.1: Visualisation of the UCB1 algorithm on three Bernoulli arms; estimates and error bars for each arm at different turns. The dots represent the estimates of the means of the arms, while the error bars represent the upper confidence bounds and the dashed lines represent the true means. As the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

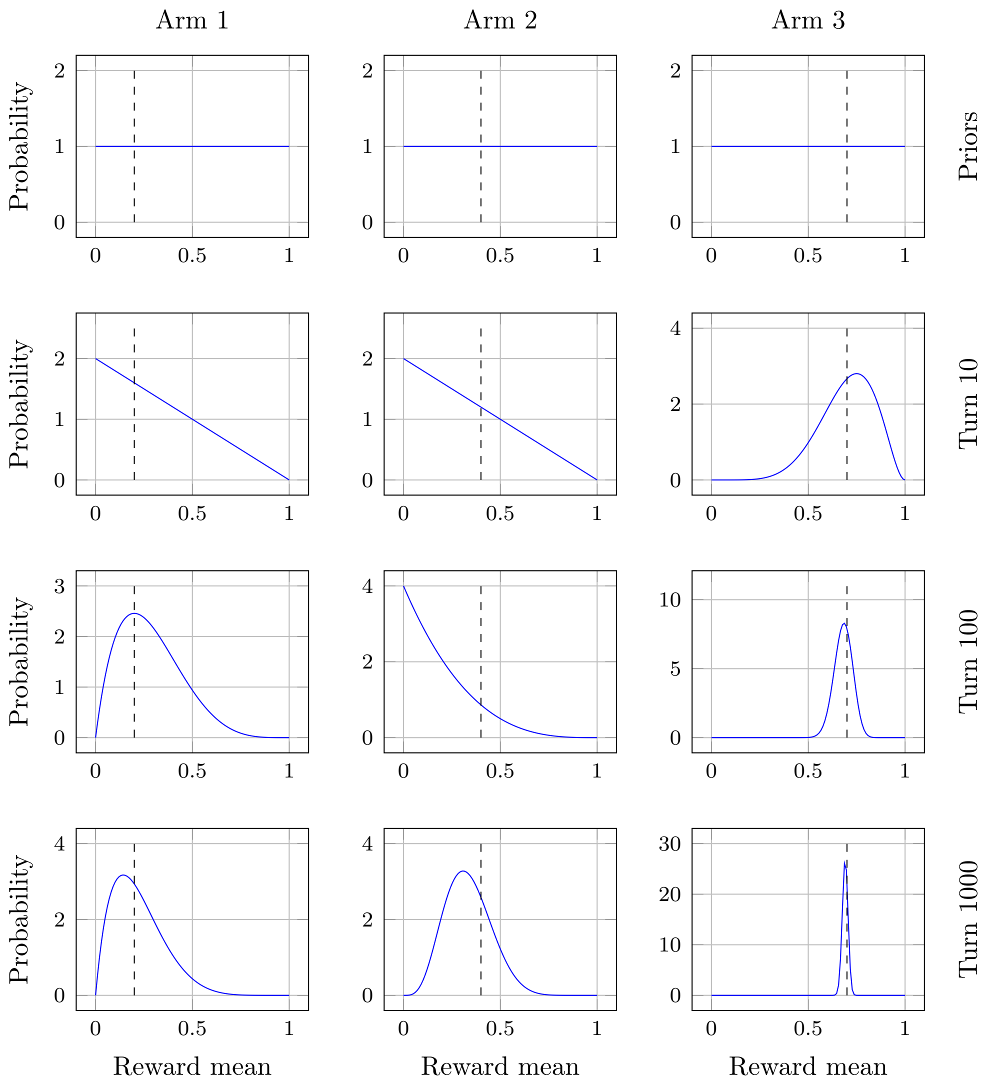


Figure 2.2: Thompson sampling algorithm with Bernoulli rewards and three arms visualised. In the top row, the uniform priors for the reward distribution means are on display. Thereunder, the posteriors are shown at turns 10, 100 and 1000. The dashed vertical lines indicate the true means. As the algorithm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

Also for Gaussian rewards, it was proven asymptotically optimal with uniform priors [26]. 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

```
1 Update posterior for arm $a_{t-1}$ with reward $X_{t-1}$
2 for $a \in \mathcal{A}$ do
3     Sample $\theta_a \sim P(\mu_a \mid \mathcal{D})$
4 return $\text{argmax}_{a \in \mathcal{A}} \theta_a$
```

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 pulls. Furthermore, empirical results generally indicate better performance than UCB variants [23]. 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 simple computations required by UCB and epsilon-greedy algorithms.

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 regardless of the horizon, creating an anytime algorithm. The doubling trick is a simple idea: simply first run the algorithm for T steps, then $2T$ steps, $4T$ ad infinitum, possibly with some other geometric factor. It was first introduced in [27], and has since been proven to conserve minimax regrets, but not instance-dependent regrets [28]. Using instead exponential progression, it is possible to maintain instance-dependent regret bounds instead of minimax optimality [28].

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 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, namely neural networks, it is possible to learn good policies and beat the best human players, which was famously demonstrated with AlphaGo in 2016 [29].

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 [30].

The first section of this chapter will introduce the Markov decision process (MDP), which is the mathematical model of reinforcement learning, and it will be based primarily on [11, 31]. 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 expected future

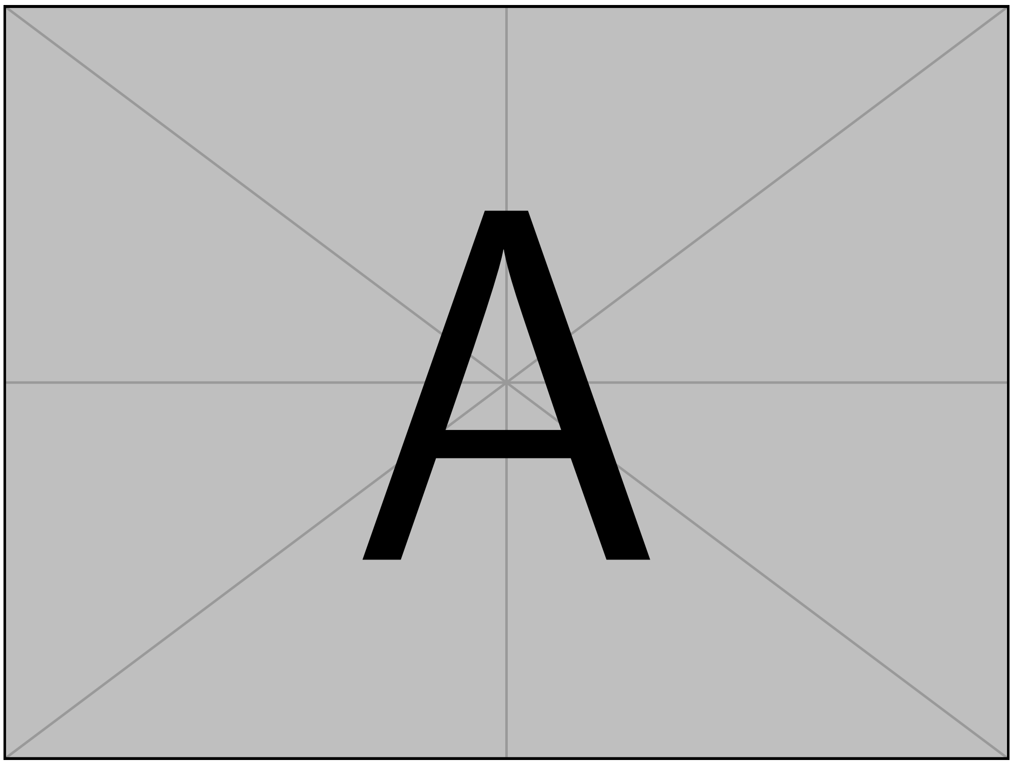


Figure 3.1: A simple Markov decision process.

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, action and rewards, and not just the current state.

3.1.1 Example: Cart-pole

A popular platform for testing RL algorithms is OpenAI Gym [32], 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 [33]. 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[1], 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.

3.1.2 Learning MDPs

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

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

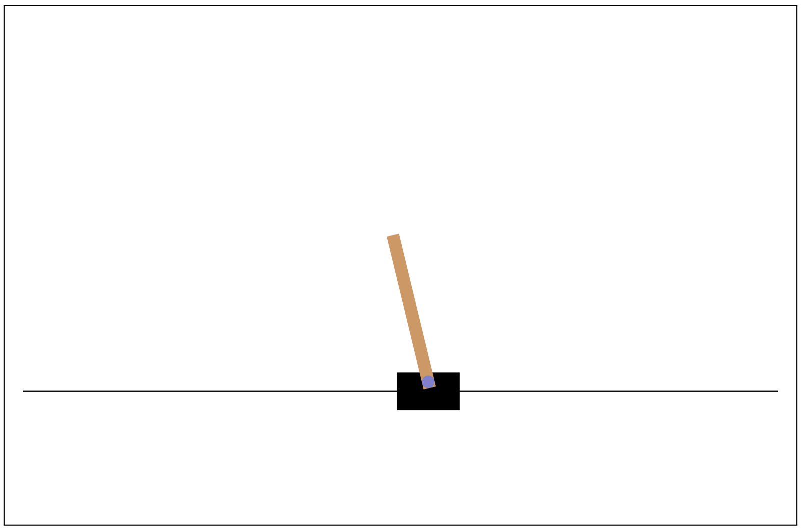


Figure 3.2: The cart-pole environment in OpenAI Gym. An agent must move the cart to the right or the left to combat gravity and keep the pole upright. If the pole falls over or the cart moves too far from the centre, the game is over. The goal is to last as long as possible.

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 and old policy. They can nonetheless be sensitive to the choice of hyperparameters and may struggle in 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 that will be discussed further in section 3.3. But also model-based algorithms have seen some commendable results recently, for example being able to find diamonds in *Minecraft* [34].

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 to not lose the game [35].

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* is nigh impossible to solve with off-the-shelf methods and without specialising the rewards [36]. While the suite of Atari games is mostly solvable with the same algorithms, the lack of immediate rewards leaves agents clueless as how to progress. Implementing the necessary guidance is can not only be demanding, but it also looses 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 [31]. While humans can employ prior knowledge, such as that keys will open doors, reinforcement learning agents typically has 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 [37]. 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 [29]. 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 [38, 39], 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[2] 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 have been shown to appear for a variety of models [40].

With the great size and complexity, interpretability is sacrificed. The models are deterministic and often black boxes; it is difficult to understand 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[3]. 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.

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

[2]Q.v. section 3.2.1.

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

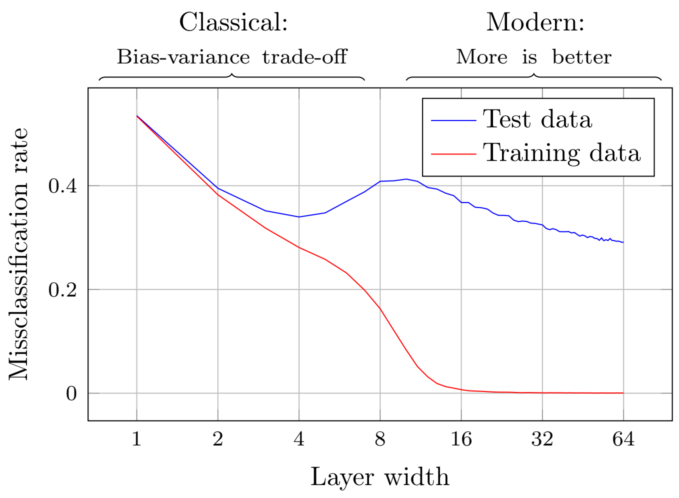


Figure 3.3: Double descent phenomenon. As model complexity increases, the train error decreases steadily, but the test error reaches a minimum and after which the test error starts to increase again. However, past a certain point, the test error decreases again. This defines two regimes: the first classical 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 [40], where ResNet18 models were used for image classification with added label noise.

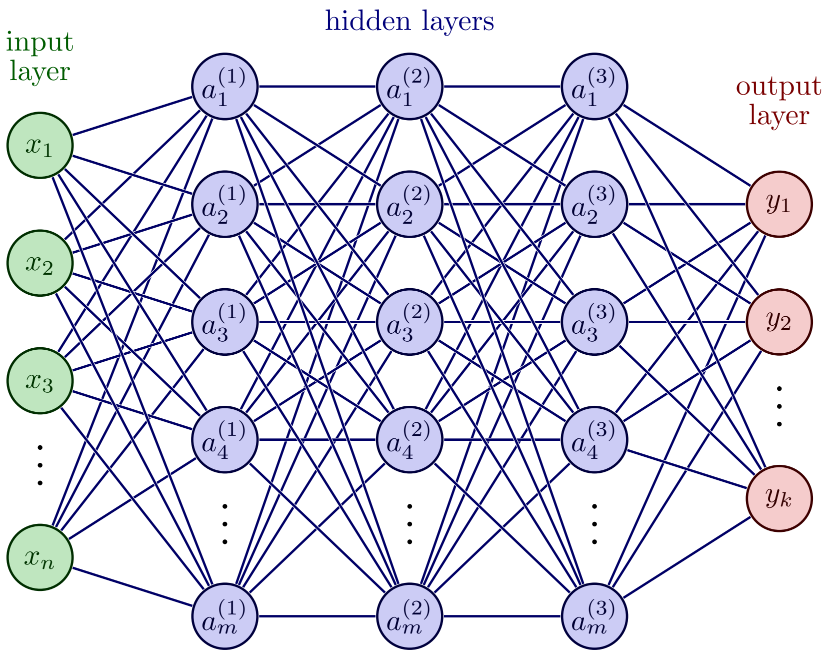


Figure 3.4: Typical structure of a dense feed-forward neural network. Here it has three hidden layers of constant size m, but the number of layers and the size of each layer can be chosen arbitrarily. Being dense means that each node in one layer is connected to all nodes in the next layer, while being feed-forward means that the connections are unidirectional. From [41].

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

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 to 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

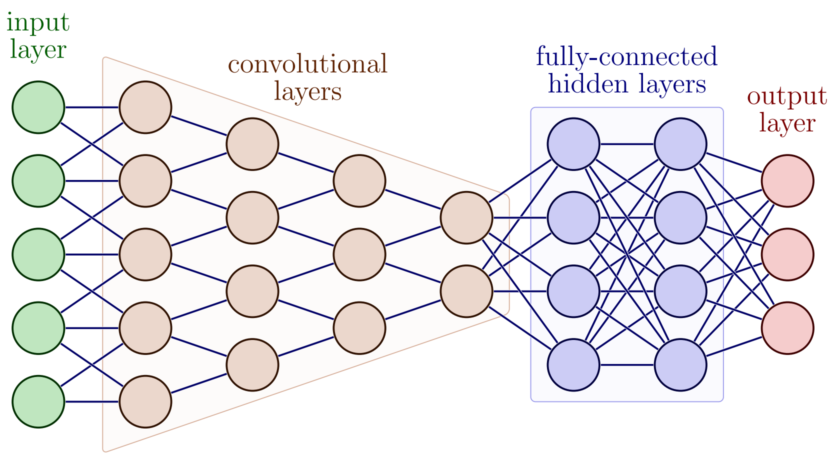


Figure 3.5: The basic structure of a convolutional neural network. Data is input before conventional layers are applied, in which perceptrons are only connected to small regions of the previous layer. After sufficient dimensionality reduction, regular dense layers can be used. From [41].

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 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 length, 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 [42].

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 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, the 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 discriminators 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 [43]. 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 of 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 [44].

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 [45], prioritised experience replay algorithm [46] and duelling DQN algorithm [47].

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 [48]. 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 [42].

The Proximal Policy Optimisation (PPO) algorithm [49] has been particularly successful, expanding on the ideas from the Trust Region Policy Optimisation (TRPO) algorithm [50]. 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 [49]. What is more, PPO has been able to beat the world champions in the video game of *Dota 2* [51, 52].

3.3.3 Actor-critic methods

Actor-critic methods are a type of reinforcement learning algorithms that combine ideas from both value-based and policy-based methods. Proposed in [53], these algorithms maintain both a policy function and a value function that 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 [54]. 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 achieving top 0.15% performance in the video game of *Starcraft II* [55].

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 for this thesis, and even so only a brief explanation is offered here. For more details, see [56] for a rigorous, complete description or [57] 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 $|1\rangle$[4], and the qubit can be in any superposition of these states. This follows from the first postulate of quantum mechanics[5], 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

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

[4]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 $|\cdot\rangle\langle\cdot|$.

[5]As they are laid out in [56].

where $\alpha, \beta \in \mathbb{C}$. The only requirement is that the state is 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 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 $|1\rangle$ 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 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, $\rho = |\psi\rangle\langle\psi|$. 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

$$\rho = \sum_{i=1}^{n} p_i\,|\psi_i\rangle\langle\psi_i| \tag{4.3}$$

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 to one. Given a basis and a finite Hilbert space, the density operator can be expressed as a density matrix[6] where the diagonal elements are

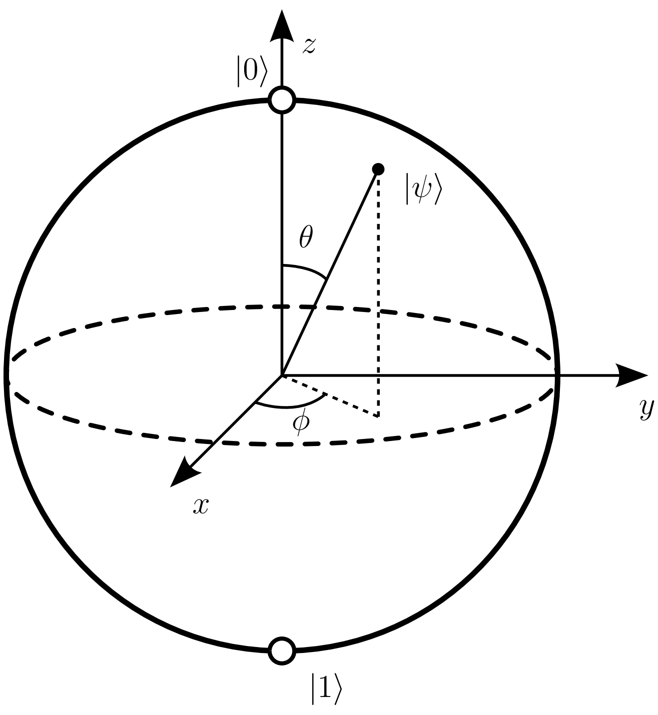


Figure 4.1: The Bloch sphere. On it, the state of a single qubit state is represented by a point. The state $|0\rangle$ is the north pole, and $|1\rangle$ is the south pole. The latitudinal angle θ determines the probability of measuring the qubit in the state $|0\rangle$, while the longitudinal angle ϕ corresponds to the complex phase between the two basis states. From [58].

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,

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{4.4}$$

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

$$\rho = \frac{1}{2}\left(I + x\sigma_x + y\sigma_y + z\sigma_z\right), \tag{4.5}$$

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 lie 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 have their own probability amplitude, and thus their 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 tensor product as

$$|\psi_1\psi_2\cdots\psi_n\rangle = |\psi_1\rangle|\psi_2\rangle\cdots|\psi_n\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle. \tag{4.6}$$

[6] 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.

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

$$\begin{aligned} |\psi_1\psi_2\cdots\psi_n\rangle &= c_1 |0\dots00\rangle + c_2 |0\dots01\rangle + \cdots + c_{2^n} |1\dots11\rangle \\ &= (c_1, c_2, \dots, c_{2^n})^\top \\ &\in \mathbb{C}^{2^n}, \end{aligned} \tag{4.7}$$

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 matrices acting on the state vector, $|\psi\rangle$, as

$$|\psi'\rangle = U |\psi\rangle, \tag{4.8}$$

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

The most basic gates are the Pauli gates, which are applications of the Pauli matrices from eq. (4.4) 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,

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{4.9}$$

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 returns 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:

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

These parametrised gates will be useful in section 4.5.

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

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4.10}$$

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

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

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

$$p_i = \langle\psi| P_i |\psi\rangle \,, \tag{4.12}$$

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 of reducing variance in the estimates, though at the cost of some bias [59].

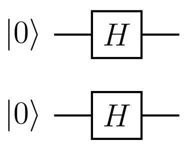
Canonically, the computational Z-basis is used for measurements, and it is usually the only basis for which measurements are physically implemented in a quantum computer. When measuring in the computational basis in which a state is expressed, as eq. (4.7), the probabilities are simply given by the absolute 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.

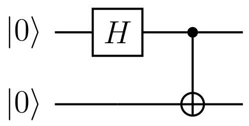
A simple example is the circuit



, (4.13)

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.

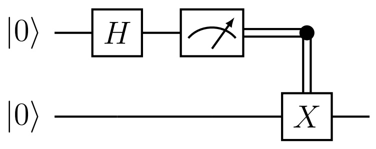
Slightly more interesting is the circuit



(4.14)

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



(4.15)

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 stem 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. Encrypted data is already being hoarded [**NEEDED**], and the research and development of quantum computers is already well underway.

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 [56] and BPP, the class of problems that can be solved in polynomial time with a classical probabilistic algorithm [56]. This follows from the fact than quantum computers can efficiently run any efficient classical algorithm and reproduce any classical randomness by performing quantum measurements.

The Deutsch-Jozsa algorithm [60], while of no practical use, is proved to be exponentially faster than any deterministic classical algorithm[7] [**NEEDED**]. 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, but this is not proven [**NEEDED**].

The complexity hierarchy can be expressed thus:

$$\begin{tikzcd}
 & \mathsf{NP} \arrow[rrd, hook] & & \\
\mathsf{P} \arrow[ru, hook] \arrow[rd, hook] & & & \mathsf{PSPACE} \\
 & \mathsf{BPP} \arrow[r, hook] & \mathsf{BQP} \arrow[ru, hook] &
\end{tikzcd}$$

It is widely conjectured that the containments are strict, but this is not proven. As disproving $\mathsf{P} = \mathsf{PSPACE}$ remains elusive, proving that BQP is strictly greater than P must be even harder.

[7] The provable exponential speed-up does not prove $\mathsf{P} \neq \mathsf{BQP}$, as the algorithm requires an oracle [**NEEDED**]. 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 [61].

Exponential speed-ups do not come for free. Although the states spaces are exponentially large, with only a limited set of operations available, states can not be created and manipulated arbitrarily. What is more, recapturing all the information in a general state can not be done without negating any exponential gains. The problem must have some structure to be exploited for a speed-up to be possible. Quantum computers do only solve certain problems more efficiently 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 $k = O(2^N)$ [62]. To solve linear system, 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[8], 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 perhaps more easily found. For example, the Grover algorithm which is used to search for an element in an unsorted list has time complexity $O(\sqrt{N})$ [64]. Classically, this can not be done in less than $O(N)$ time. It can be proven that the Grover algorithm is optimal [65], so for this problem, an exponential speed-up is impossible. This algorithm and the more general amplitude amplification on which it builds solves very general problems and are often used subroutines to achieve quadratic speed-ups in other algorithms. 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 [64] 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 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 searches faster [65]. This should not be surprising. If an

[8]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. C.f. [63].

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. Then, one assumes access to a quantum oracle, $\mathcal{O}_f$ (effectively a black box subroutine) that implements f thus:

$$\mathcal{O}_f \left|x\right\rangle = (-1)^{f(x)} \left|x\right\rangle . \tag{4.16}$$

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. 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 is most likely to be measured, while the amplitudes of the other elements are reduced.

Let the state $|w\rangle$ which be the winner state, a state with amplitude 1 for the desired element and 0 for all others. Then consider the state $|s\rangle$, which is a uniform superposition state, a state with equal amplitudes for all elements. Define the state $|s'\rangle$ by subtracting the projection of $|w\rangle$ onto $|s\rangle$ from $|s\rangle$:

$$|s'\rangle = |s\rangle - \langle w|s\rangle |w\rangle . \tag{4.17}$$

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 $|s\rangle$ serves as a starting point for the algorithm, and is achieved by applying Hadamard gates to all qubits. It is expressible as

$$|s\rangle = \cos(\theta) |s'\rangle + \sin(\theta) |w\rangle , \tag{4.18}$$

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

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

$$D = H^{\otimes n} S_0 (H^{\otimes n})^{-1} = H^{\otimes n} S_0 H^{\otimes n}, \tag{4.19}$$

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 $k \approx \pi/4\sqrt{N/M}$, where M is the number solutions to $f(x) = 1$ [56].

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

$$|\psi\rangle = \sin(\theta)\,|\psi_0\rangle + \cos(\theta)\,|\psi_1\rangle\,, \tag{4.20}$$

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

$$S_\psi = I - 2\,|\psi\rangle\langle\psi| \tag{4.21}$$

$$S_P = I - 2P, \tag{4.22}$$

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

$$G^k\,|\psi\rangle = \sin((2k+1)\theta)\,|\psi_1\rangle + \cos((2k+1)\theta)\,|\psi_0\rangle\,, \tag{4.23}$$

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 [66], quantum Monte Carlo methods [67] and some bandit algorithms to be discussed in what follows.

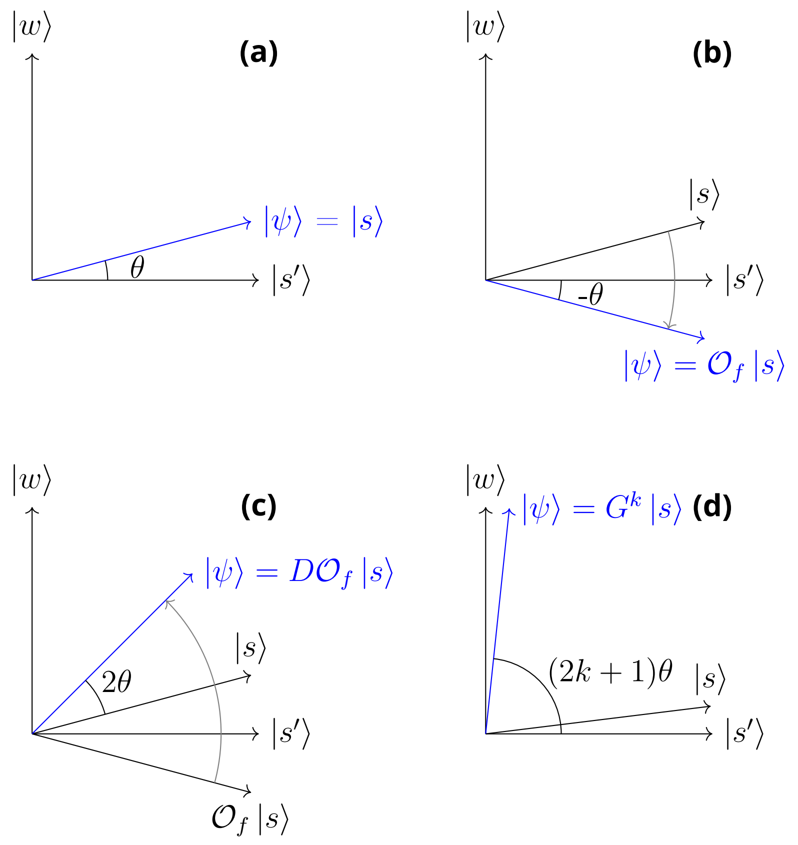


Figure 4.2: Grover's algorithm visualised. (a) The initial uniform superposition state $|s\rangle$ is prepared, which can be seen as a linear combination of $|w\rangle$ and $|s'\rangle$, forming an angle θ to the s'-axis. (b) The oracle $\mathcal{O}_f$ is applied to $|s\rangle$, flipping the sign of 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 $|w\rangle$.

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,

$$\mathcal{A}\left|0\right\rangle = \sqrt{a}\left|\psi_1\right\rangle + \sqrt{1-a}\left|\psi_0\right\rangle, \tag{4.24}$$

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

With its original formulation in [66], it was proven that the amplitude can be estimated with an additive error of

$$\epsilon \leq 2\pi\frac{\sqrt{a(1-a)}}{t} + \frac{\pi^2}{t^2} \tag{4.25}$$

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 $\mathcal{A}$ [67]. Later variants have been proposed to reduce the qubits needed and circuit depths [68, 69, 70], 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 [71, 72, 73]. 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

$$P(|\hat{\mu} - \mu| \geq \epsilon) \leq \frac{\sigma^2}{n\epsilon^2}, \tag{4.26}$$

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 [74].

Generalising amplitude estimation, in [67], 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[9].

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

$$U|x\rangle|0\rangle = |x\rangle\left(\sqrt{1-\phi(x)}\,|0\rangle + \sqrt{\phi(x)}\,|1\rangle\right), \tag{4.27}$$

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

$$|\psi\rangle = U(\mathcal{O} \otimes I)|0\rangle. \tag{4.28}$$

For these bounded random variables in particular, $O(1/\epsilon)$ iterations suffices to achieve an additive error of ϵ, repeating the whole procedure $O(1/\log(\delta))$ times to achieve a certainty $1-\delta$ [67].

4.4 Limitations of NISQ hardware

Quantum hardware have been physically realised and even demonstrated to outperform classical computers in very contrived situations [75, 76, 77], 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 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 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.

[9]The $\tilde{O}$ notation ignores logarithmic factors.

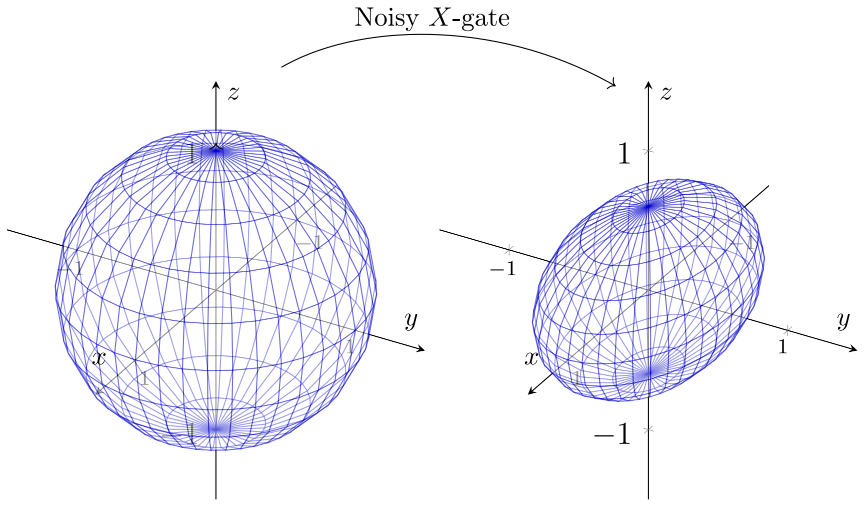


Figure 4.3: Illustration of applying a noisy X-gate on the Bloch sphere. The left plot shows the set of all pure states, the Bloch sphere. On the right, the same set of states is shown after the application of a noisy X-gate with probability $p = 0.2$ of failing. There, the y- and z-axes are contracted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

4.4.1 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 the fails to apply with probability p, the resulting density operator ρ' can be expressed as

$$\rho' = p\rho + (1-p)X\rho X^\dagger, \tag{4.29}$$

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 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.

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 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 amount 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

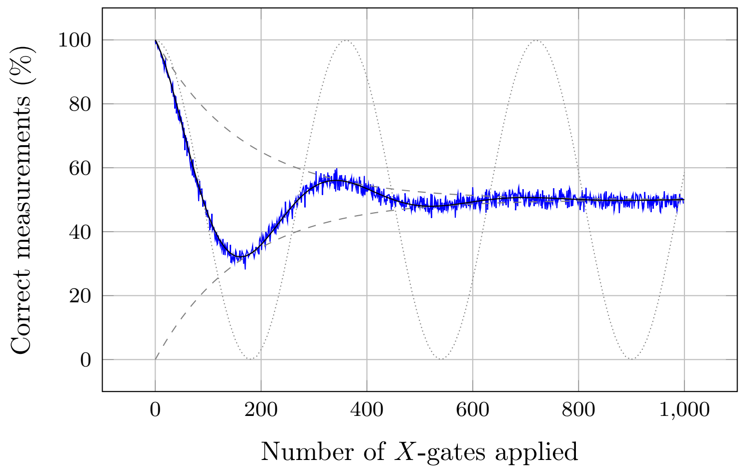


Figure 4.4: Proportion of correct measurements of a qubit after applying several noisy X-gates. The error in the rotation is $1°$, while the probability of failing to apply the gate is $p = 0.3\%$. For each number of gates, the measurement is repeated 1000 times. The coherent error causes a sinusoidal behaviour, in which 180 rotations causes 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.

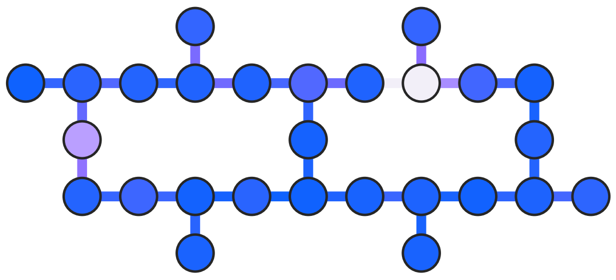


Figure 4.5: Qubit connectivity and error rates in IBM's Montreal 27-qubit Falcon r4 chip [78]. Brighter coloured nodes indicate higher X-error rates, while brighter edges indicate higher CNOT-error rates.

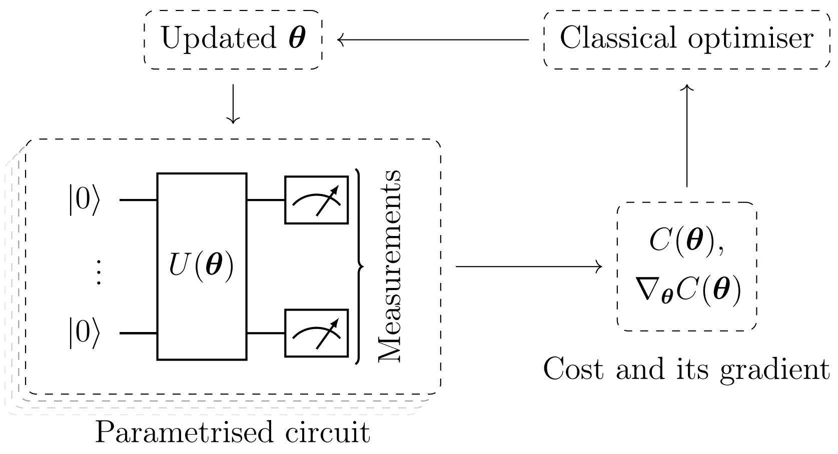


Figure 4.6: The general structure of a variational quantum algorithm.

algorithms. Recent estimates find that millions of noisy qubits would be needed to break RSA encryption [79].

The transpiling 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.

4.5 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. 4.6, the basic idea is to run a parametrised circuit, evaluate a cost 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.

4.5.1 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 [80].

Important to the success of a VQA is the choice of the quantum circuit. The circuit selected is known as the ansatz, and there are a plethora of different ansätze that can be chosen. An example are 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 [80].

4.5.2 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 [81]. 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 tool-box 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 [82]. However, the loss landscape will generally not be convex [83], 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 [84, 85]. Additionally, with noisy quantum hardware, 'conceptually different' but equally problematic noise-induced plateaus can occur [86]. It is therefore important to consider both the hardware and the ansatz when designing a VQA.

4.5.3 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 [80], the noise and decoherence will still be a limiting factor. Error mitigating post-processing can be used to improve the performance of VQAs, but which methods to use, and their effectiveness requires further investigation [87]. 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.

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 lie in sample efficiency rather than computational difficulties, quant-um 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 Oracles as bandit arms

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 [88], 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 $O(n \text{ poly}(\log T))$, which is still substantially better than $\Omega(\sqrt{nT})$ minimax regret for classical bandits.

The algorithm proposed is essentially a UCB-like algorithm, 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 quantum queries, the algorithm must cleverly decide for how many steps to pull each arm in addition to which arm to pull, before running a QMC session.

As listed in algorithm 5.1, the 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 are 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 $|\hat{\mu}_i - \mu_i| \leq \text{UCB}_i$ 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, coming from the big-O notation used to describe QMC convergence. How it should be set for implementation of the algorithm is not described in the paper.

Algorithm 5.1: QUCB1

```
  Input: Set of arms $\mathcal{A}$, $\mathcal{O}_i$ as in eq. (5.1), $T$ horizon, $0 < \delta \ll 1$
1 for $a \in \mathcal{A}$ do
2     $\text{UCB}_a \leftarrow 1$
3     $N_a \leftarrow (C_1/\text{UCB}_a)\log(1/\delta)$
4     Estimate $\mu_a$ using QMC with $N_a$ samples
5 while Total number of queries to the oracles is less than $T$ do
6     $a \leftarrow \text{argmax}_a(\hat{\mu}_a + \text{UCB}_a)$
7     $\text{UCB}_a \leftarrow \text{UCB}_a/2$
8     $N_a \leftarrow (C_1/\text{UCB}_a)\log(1/\delta)$
9     Update estimate of $\mu_a$ using QMC with $N_a$ samples
```

5.1.1 Proof of logarithmic regret

Recall that 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 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.

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. With 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

$$\sum_{a \in \mathcal{A}} K_a \leq k \log_2 \left(\frac{T}{k C_1 \log \frac{1}{\delta}} \right) - k, \tag{5.6}$$

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

$$N_{\mathrm{QMC}} \leq k \log_2 \left(\frac{T}{k C_1 \log \frac{1}{\delta}} \right). \tag{5.7}$$

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

$$\begin{aligned} 1 - \left(\bigcup_{i=1}^{N_{\mathrm{QMC}}} P(\text{Query } i \text{ fail}) \right) &\leq 1 - \sum_{i=1}^{N_{\mathrm{QMC}}} P(\text{Query } i \text{ fail}) \\ &\leq 1 - \sum_{i=1}^{N_{\mathrm{QMC}}} \delta \\ &= 1 - N_{\mathrm{QMC}} \delta \\ &\leq k \log_2 \left(\frac{T}{k C_1 \log \frac{1}{\delta}} \right) \delta. \end{aligned} \tag{5.8}$$

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

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

as desired.

5.2 Implementing QUCB1

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.

5.2.1 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.

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

Algorithm 5.2: QUCB 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
```

5.2.2 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).

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, which will be used to update the QUCB estimate. The number of oracle calls is also automatically tracked by the `AmplitudeEstimation` class, which is iterated by 1 to account for the initialisation of the oracle before amplitude estimation is performed. In total, the QMC subroutine is implemented as in algorithm 5.3.

Algorithm 5.3: QMC for a Bernoulli arm oracle

```
Input: Oracle $\mathcal{O}$, number of iterations $N$, confidence $\delta$
Output: Estimate of $p$ with confidence $1-\delta$, number of oracle calls
1 Initialise EstimationProblem with oracle $\mathcal{O}$ and objective qubit 0
2 Number of shots $N_S \leftarrow \log(1/\delta)$
3 Number of qubits $N_Q \leftarrow \lceil \log_2 N \rceil$
4 Initialise AmplitudeEstimation with number of shots $N_S$ and
    number of qubits $N_Q$
5 Estimate $p$ with AmplitudeEstimation on the EstimationProblem
    object
6 $\hat{p} \leftarrow$ MLE result of AmplitudeEstimation
7 $T \leftarrow$ number of iterations used by AmplitudeEstimation
8 return $\hat{p}, T+1$
```

5.2.3 QUCB1

Oracles and QMC being available, the QUCB1 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.

5.3 Best arm identification

In [89], 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, but it may still serve as a useful baseline for comparison, with the more complicated algorithms discussed in the following sections seeable as extensions hereof.

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

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

Algorithm 5.4: QUCB1 simulation with a set of Bernoulli arms

```
Input: Set of arm probabilities $\{p_1, \ldots, p_k\}$,
number of iterations $T$, confidence $\delta$
Output: Regret at each iteration
1  Initialise $t \leftarrow 0$
2  for $a \in \{1, \ldots, k\}$ do
3      Initialise oracle $\mathcal{O}_a$
4      Initialise $\mathrm{UCB}_a \leftarrow 1$
5      Initialise $N_a \leftarrow (C_1/\mathrm{UCB}_a)\log(1/\delta)$
6      Estimate $\hat{\mu}_a$ with algorithm 5.3
7      Update $t$ with the number of oracle calls
8      Log regret
9  while $t < T$ do
10     $a \leftarrow \mathrm{argmax}_a(\hat{\mu}_a + \mathrm{UCB}_a)$
11     $\mathrm{UCB}_a \leftarrow \mathrm{UCB}_a/2$
12     $N_a \leftarrow (C_1/\mathrm{UCB}_a)\log(1/\delta)$
13     Update estimate $\hat{\mu}_a$ with new estimate from algorithm 5.3
14     Update $t$ with the number of oracle calls
15     Log regret
16 Discard any regret from excess turns
17 return regrets $R_1, \ldots, R_T$
```

where a is the arm to be queried, 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. 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 bandits, regret minimisation is no longer 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 [86] propose a more sophisticated algorithm, improving the results of [89] 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

There are four fundamental ways to combine machine learning and quantum computing [90]. One can differentiate between the type of the 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 [91].

CQ 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 [59]. Alternatively, advanced machine learning models like neural networks can be employed to describe quantum states more efficiently.

QC 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 [92] or fast quantum procedures for linear algebra, e.g., HHL to invert matrices in linear regression [93]. However, whether NISQ-era quantum computers can provide any benefits for machine learning is still an open question. To this end, the study of VQAs as machine learning models have garnered much attention [94], 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 [95]. 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 not of immediate interest. 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, the encoding is such a big part of CQ that results thence are not necessarily applicable to QQ.

6.1 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.

6.1.1 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

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	$b(N)$	$O(b(N))$	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)

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.

6.1.2 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, that 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 severly, 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 [90]. In general, for classical data, circuits must be linearly deep in the size of the data and ergo exponentially deep in the amount of qubits, which makes it beyond the reach of NISQ hardware.

6.1.3 Angle encoding

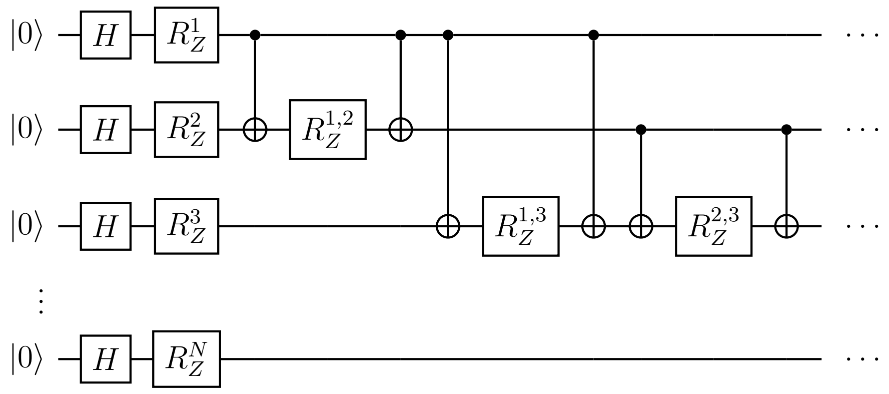
A third option is to encode data into the angles of rotations. Here, the potentially continuous components of the data are mapped to rotations of the qubits. For the rotations to be meaningful angles and not loop around, the data need to be normalised. An N-dimensional data point $\boldsymbol{x}$ is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

6.1.4 Second order angle encoding

Conjectured to be hard to simulate classically, a second-order angle encoding is proposed in [96]. 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.

6.1.5 Repeats

The expressive power of models heavily rely 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 is studied in [97], where it is shown that simply repeating simple encoding blocks allows for learning of more frequencies and thus more complicated functions. Asymptotically, such repeats lets a quantum model learn arbitrary functions.

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 follow classical feed-forward networks. Figure 6.1 shows the general circuit layout. In the first step (or layer), data is encoded into the qubits, typically using a method discussed in section 6.1. 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 [98], 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 6.2.2. 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.

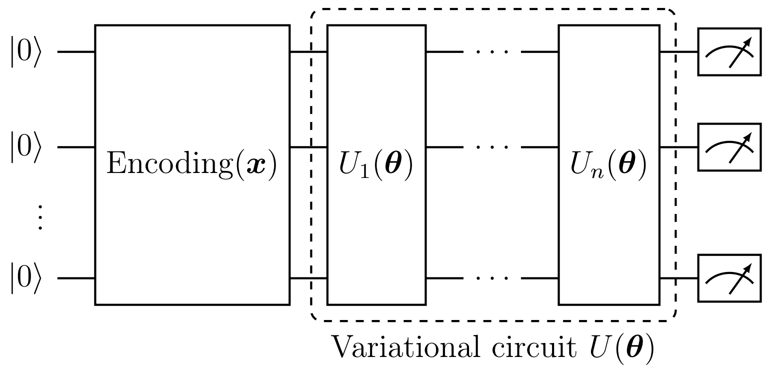

$|0\rangle$
$|0\rangle$
$|0\rangle$
Encoding($\boldsymbol{x}$)
$U_1(\boldsymbol{\theta})$
$U_n(\boldsymbol{\theta})$
Variational circuit $U(\boldsymbol{\theta})$

Figure 6.1: 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.

6.2.1 Challenges

While the power of classical neural network 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 6.1) 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 [99], 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 [100]. Another way is so-called repeat-until-success schemes [101], 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 [102]. 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 4.5, and it is clear that NISQ-era QNNs can not be expected to scale as well as classical NNs do.

6.2.2 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[10].

Quantum convolutional neural networks

Originally introduced in [100], 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, 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 [100] and that they inherit their classical counterparts' ability to

[10]Q.v. section 3.2.

classify images [103]. Also, QCNNs have desirable properties with regard to avoiding barren plateaus [104], 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 [105]. 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 [106].

Hybrid quantum-classical neural networks

Another option is to include a quantum layer or node is 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 [107], 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, they can take over more and more of the hybrid models.

Quanvolutional neural networks, proposed in [108], 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 [109], 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. There is therefore not straightforward to translate the classical recurrent neural network (RNN) to a variational quantum circuit in the manner of fig. 6.1 and the above discussed architectures. Still, there attempts have been made to define a QRNN, such as in [110] 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 [111], 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, we will discuss how quantum neural networks can be used as agents in reinforcement learning algorithms. 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 [112]. 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 the interacting with the environment is quantum also.

In [113], 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 [114], 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 [115] 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, [116] show that the learning of an agent can be accelerated. Somewhat similarly, in [117], hybrid agents are shown to learn quadratically faster than purely classical agents.

Chapter 7

Simulations

While the upper bound shown in section 5.1 implies QUCB1 advantage, it does not necessarily mean it is supreme in all cases or even on average. It is therefore of much interest to test it on a variety of problems and compare its average performance to other algorithms — not only with the classical UCB1 to which it is compared in the original paper, but also with the more performant Thompson sampling algorithm.

All algorithms were implemented in Python 3 [118] with quantum computing support provided by IBM's Qiskit library [119]. The experiments were run on an external computation server with an Intel Xeon E5-2690 v4 CPU and 768 GB of RAM, permitting 28 concurrent simulations. For fixed bandit instances, 100 simulations were run for each algorithm.

To limit the scope and computational resources required, only Bernoulli rewards were considered. As noted in [88], setting the confidence parameter to a higher value than the theoretically desirable $1/T$ leads to better performance. It was thus set to 0.01 for all experiments. Moreover, the constant C_1 which was 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 [88].

7.1 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 [88]. The results are shown in fig. 7.1, agreeing with the original paper; QUCB greatly outperforms UCB. However, Thompson sampling which was not considered in [88], is not that far behind.

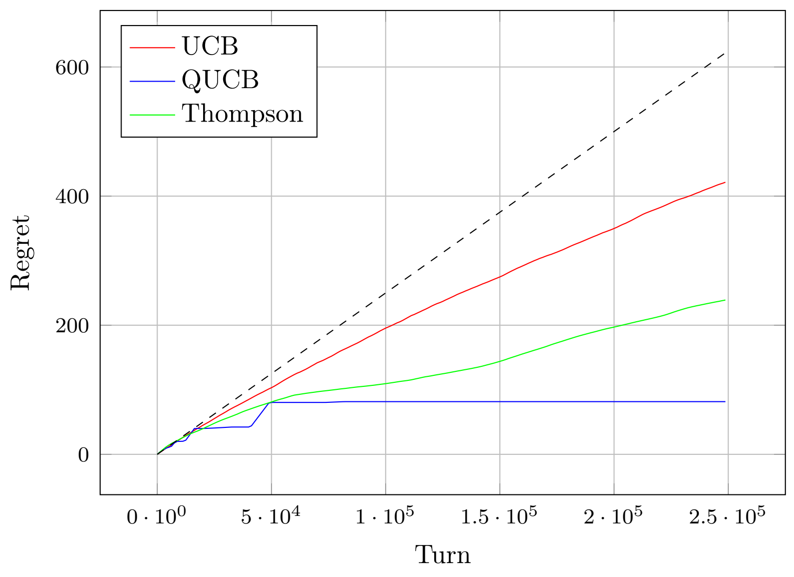


Figure 7.1: QUCB1 regret for two arms, with mean 0.5 and 0.505.

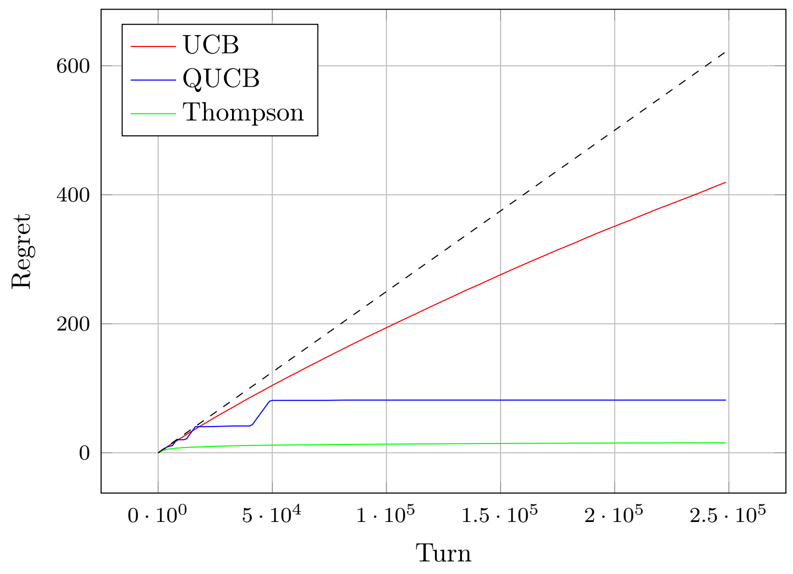
7.1.1 Low and high probabilities

Next, mean more extreme mean values of 0.01 and 0.005 with a reward gap of 0.005 equal to the above, are tested. Figure 7.2a contains the results. Here, QUCB beats UCB thoroughly still, both algorithms achieving similar regrets as in fig. 7.1, but now Thompson sampling is clearly superior.

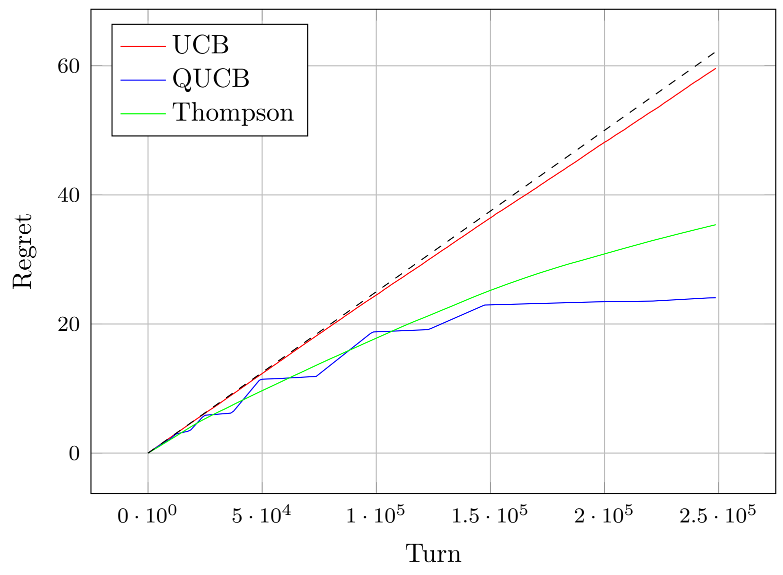
At these extreme values, it seems the gap must shrink for QUCB to be able to outperform Thompson sampling. This is shown in fig. 7.2b, where the gap is reduced to 0.0005, Even so, the QUCB advantage is not as pronounced as in fig. 7.1.

7.1.2 Four arms

As a final test, the number of arms is increased to four. The results are plotted in fig. 7.3. 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. In general, QUCB provides no advantages or noteworthy changes from UCB with respect to the number of arms, so its behaviour should be expected to be similar as UCB when the number of arms is increased.



(a) Two arms, means 0.01, 0.005.



(b) Regret for two arms, means 0.99, 0.9905.

Figure 7.2: QUCB1 regret for especially high and low means.

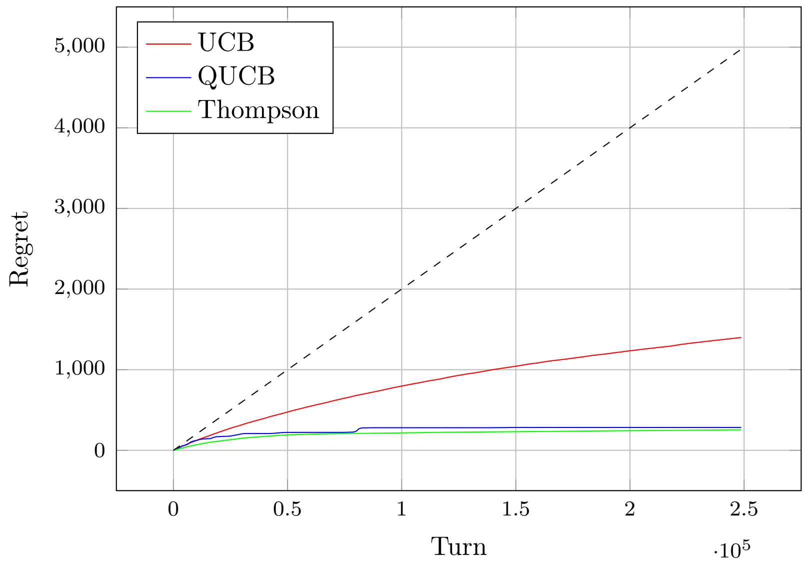


Figure 7.3: Regret for four arms, with mean 0.5, 0.51, 0.52 and 0.53.

7.2 Bayesian regret

7.2.1 Uniform prior

As described in section 2.2.3, the Bayesian regret is the average regret over some set prior. What priors to choose is a topic for discussion, which will not be covered deeply here. Instead, 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. 7.4. All three were run on 294 instances sampled from the prior[11], and the results are averaged over these instances.

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 only after some time.

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. Nonetheless, after the rather wasteful inaugural period, QUCB does indeed produce a very flat regret curse; when looking at the log-log-plot in fig. 7.4, it may seem like the QUCB regret will be lower than even Thompson sampling for some great and admittedly unrealistic number of turns.

7.2.2 Challenging prior

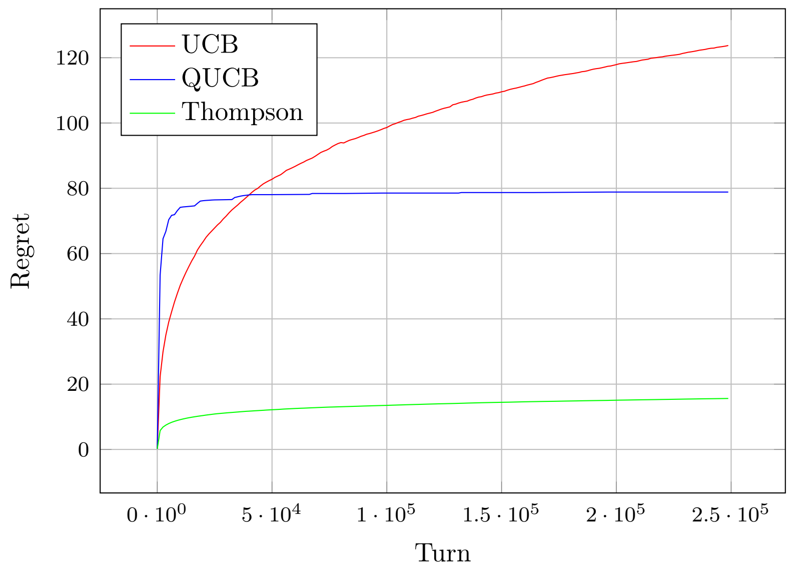
One might argue that the uniform prior is too easy, and that the quantum advantage should be more apparent in a more challenging prior. Firstly, for real-world Bernoulli bandits, there are reasons to believe that the means would lie closer to the endpoints 0 and 1 than in the middle. Secondly, for the problem to be interesting and warrant the use of clever quantum algorithms or even just any bandit theory, the means should be close to each other. Thus, the following prior was chosen:

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

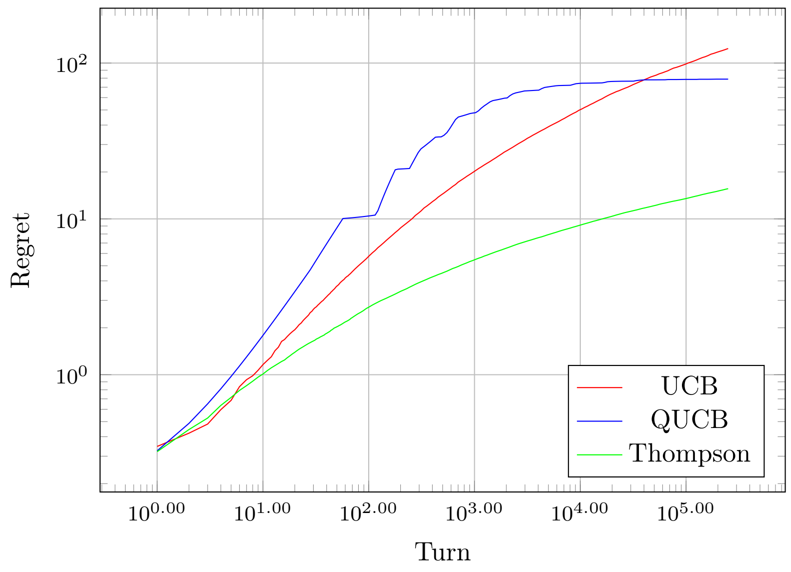
where $\mathrm{logit}(\mu) = \log(\mu/(1-\mu))$.

Running a successful 1000 simulations with this prior, the regrets visualised in fig. 7.5a were obtained. At the final 250,000th turn, the QUCB and

[11]During the 295th run, the program crashed due issues relating to disk usage permissions.



(a) Linear scale.



(b) Log-log scale.

Figure 7.4: Bayesian regret for two arms, independent and uniform priors. While Thompson sampling performs best, the log-log plot may indicate that QUCB could be better with a very large number of turns. Thompson sampling performs best, but it can appear as QUCB would be best at unreasonably great time horizons.

Thompson sampling algorithms have virtually equal regrets. QUCB lags behind 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.

7.2.3 More challenging prior

As the above prior proved to still too easy to demonstrate quantum advantage, a third was chosen, catering even more to QUCB's strengths discovered in section 7.1, id est having close means and means centred around one half. The prior was constructed similarly to the one in eq. (7.1), but with different parameters, namely:

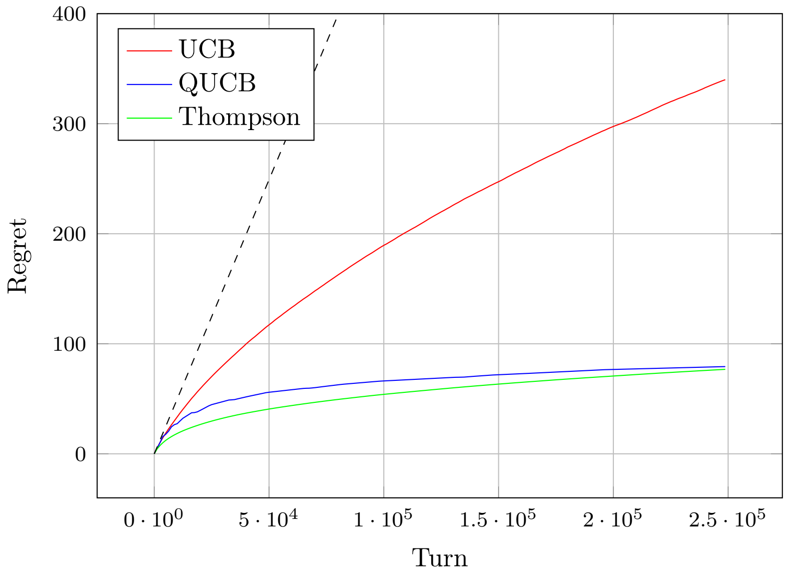
$$\begin{aligned} \mu_1 &\sim \mathrm{B}(2,2), \\ \mathrm{logit}(\mu_2) &\sim \mathrm{N}(\mathrm{logit}(\mu_1), 0.02), \end{aligned} \tag{7.2}$$

This simulation was run for 966 parallels before the computation server got tired. As is seen in fig. 7.5b, QUCB is reclaims superiority, while Thompson sampling still clearly outperforms UCB.

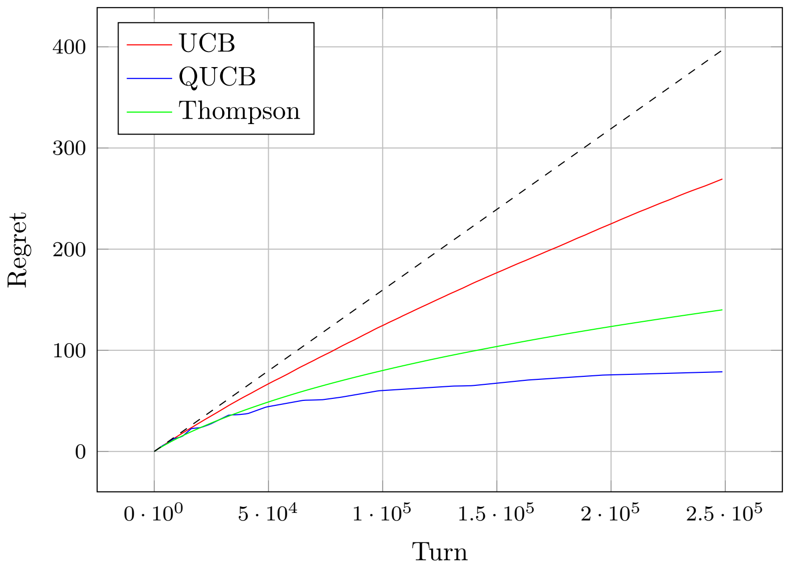
7.3 Reinforcement learning

7.3.1 Cart-pole

7.3.2 Bandits



(a) Prior as in eq. (7.1).



(b) Prior as in eq. (7.2).

Figure 7.5: Bayesian regret for two arms, challenging priors, The lower figure has the means closer to the 1/2 and even higher correlation between the means, which is more favourable to QUCB.

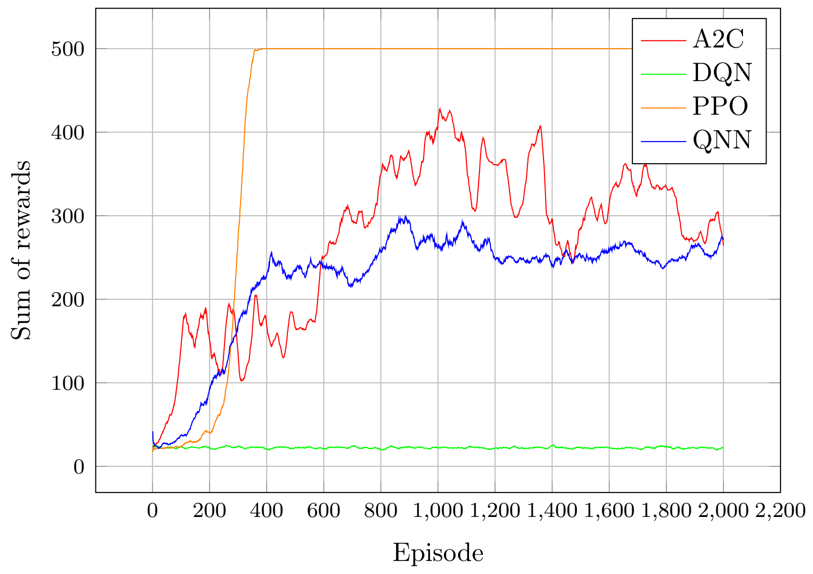


Figure 7.6: Cart-pole training. Each algorithm was trained 10 times. A rolling mean of 20 episodes was used to smooth the highly noisy data.

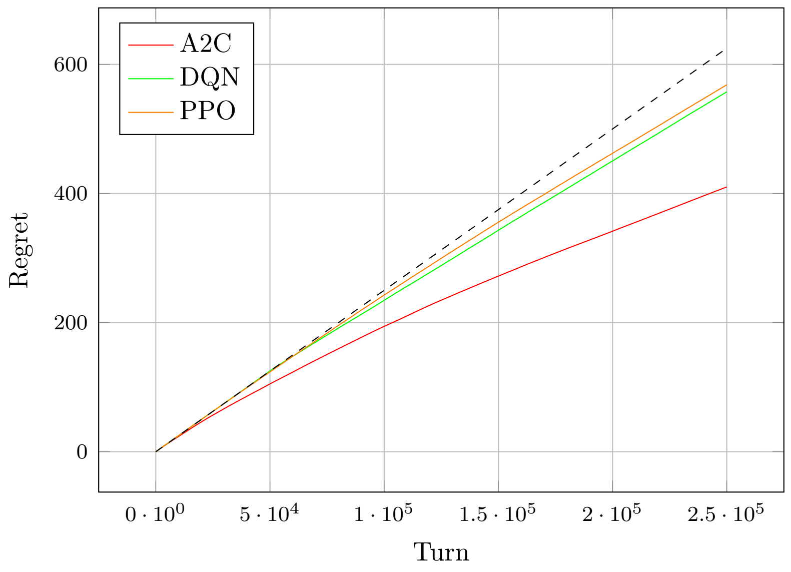


Figure 7.7: Regret for standard reinforcement learning algorithms.

Chapter 8

Final remarks

8.1 Conclusions

8.2 Outlook

References

[1] Boye Gravningen Sjo. ‘Quantum Neural Networks’. TMA4500 Project Report. Norwegian University of Science and Technology, 2022. URL: `https://github.com/boyesjo/tma4500/`.

[2] 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`.

[3] 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`.

[4] 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`.

[5] 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`.

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

[7] 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/`.

[8] 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.

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

[10] Djallel Bouneffouf et al. 'Context Attentive Bandits: Contextual Bandit with Restricted Context'. In: 22nd Aug. 2017. DOI: 10.24963/ijcai.2017/203.

[11] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms.* 1st ed. Cambridge University Press, 2020. ISBN: 978-1-108-57140-1 978-1-108-48682-8. DOI: 10.1017/9781108571401.

[12] 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.

[13] 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.

[14] 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.

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

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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: 978-3-642-34105-2 978-3-642-34106-9. DOI: 10.1007/978-3-642-34106-9_18.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] Michel Tokic and Günther Palm. 'Value-Difference Based Exploration: Adaptive Control between Epsilon-Greedy and Softmax'. In: *Lecture Notes in Computer Science* (4th Oct. 2011). DOI: 10.1007/978-3-642-24455-1_33.

[31] 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.

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

[33] 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.

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

[35] 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.

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

[37] 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].

[38] 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: arXiv:2201.11990. preprint.

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

[40] 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.

[41] 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/.

[42] 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.

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

[44] 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.

[45] 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.

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

[47] 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.

[48] 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.

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

[50] 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.

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

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

[53] 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.

[54] 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.

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

[56] 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.

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

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

[59] 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.

[60] 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 (8th Dec. 1992), pp. 553–558. ISSN: 0962-8444, 2053-9177. DOI: 10.1098/rspa.1992.0167.

[61] 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.

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

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

[64] 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.

[65] 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.

[66] 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: 978-0-8218-2140-4 978-0-8218-7895-8. DOI: 10.1090/conm/305/05215.

[67] 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.

[68] 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].

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

[70] 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].

[71] 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.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] 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.

[79] 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.

[80] 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.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

[95] 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].

[96] 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].

[97] 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.

[98] 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.

[99] 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.

[100] 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.

[101] 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].

[102] 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.

[103] 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.

[104] 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.

[105] 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.

[106] 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.

[107] 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.

[108] 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.

[109] 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.

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

[111] 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.

[112] 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.

[113] 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.

[114] 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].

[115] 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.

[116] 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.

[117] 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.

[118] 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.

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

Figures

Tables

Algorithms