Studying Optimality Bounds for Reinforcement Learning from a KWIK Perspective

harsh

1 Abstract

One of the key problems in Reinforcement Learning (RL) [6] is the exploration-exploitation trade-off, i.e. for an agent interacting in a unknown environment, how should the agent thrive a balance between using its current knowledge to maximize its reward (exploit) and sacrificing the short-term rewards to increase information about environment in hope of increasing future reward (explore). The main motivation of this study is to explore algorithms in RL setting which can provide theoretical bounds on maintaing this balance. The main difference between RL and supervised setting is that the observations are not-i.i.d., i.e., the behaviour of the environment changes with the actions of the learner and thus analysing these algorithms with the standard PAC style frameworks doesn't feel right. In this paper, we'll explore another framework for studying the RL learning task, known as Knows What It Knows (KWIK) [1].

When evaluating RL algorithms, there are three essential traits to consider: space complexity, computational complexity, and sample complexity. In this paper our focus will mostly on sample complexity (we'll formally define sample complexity in the next section). The general theme of the paper is as follows: in Section 2 we'll define the underlying notations of RL and markov Decision Processes (MDPs), in Section 3, we'll summarize the KWIK framework, in Section 4 we'll study a bound for model-based RL algorithms and in Section 5 we'll go over a bound for model-free setting. Note that this work doesn't contains any original material, therefore the general structure of each section is as follows: a brief summary (of the original work), important highlights and finally the my remark/opinion on that. Finally, in Section 6 we'll conclude with some discussion and future prospects .

2 Introduction to RL and MDP

This section introduces the Markov Decision Process notation used throughout the paper, see Sutton and Barto (1998) [6] for an introduction. An MDP M is a five tuple $< S, A, T, R, \gamma >$, where S is the state space, A is the action space, $T : S \times A \times S \to R$ is a transition function, $R : S \times A \to R$ is a reward function, and $0 \leq \gamma < 1$ is a discount factor on the summed sequence of rewards. We also let S and A denote the number of states and the number of actions, respectively. From state s under action a, the agent receives a random reward r, which has expectation $R(s, a)$, and is transported to state s' with probability $T(s'|s, a)$. A policy is a strategy for choosing actions.

For any policy π, let $V^{\pi}_{M}(s)$ $(Q^{\pi}_{M}(s,a))$ denote the discounted, infinite-horizon value (action-value or Q-value) function for π in M from state s. If T is a positive integer, let $V^{\pi}_{M}(s,T)$ denote the T-step value function of policy π.

$$V^{\pi}_{M}(s) = E[\sum_{j=1}^{\infty} \gamma^{j-1} r_j]$$

$$V^{\pi}_{M}(s,T) = E[\sum_{j=1}^{T} \gamma^{j-1} r_j]$$

where $[r_1, r_2, ...]$ is the reward sequence generated by following policy π from state s. The optimal policy is denoted π^* and has value functions $V^*_M(s)$ and $Q^*(s,a)$.

For a definite horizon MDP, we define the horizon as $H \in \mathcal{N}$, and we define the set of stages as $[H] = \{1, 2, ..., H\}$. An episode is a sequence of H state transitions: $s_1, a_1, r_1, s_2, ..., s_H, a_H, r_H, s_{H+1}$ where s_{H+1} is a terminal state. The greedy policy with respect to Q^*_h is optimal for stage h. The Bellman equation plays a central role to many RL algorithms and is described as: for any $s \in S, a \in A, h \in [H]$,

$$Q^*_h(s,a) = R(s,a) + T(s,a,s') \max Q^*_{h+1}(s',a')$$

As noted in Kakade (2003) [7] We follow the terminology of the machine learning community and define the following terms Assuming $S, A, \epsilon, \delta, and \gamma$ as input to the learner

1. **Sample Complexity:** Sample complexity measures the amount of timesteps for which the algorithm does not behave near optimally or, in other words, the amount of experience it takes to learn to behave well. More formally, for any fixed , Kakade (2003) defines the sample complexity of exploration (sample complexity, for short) of an algorithm $\mathcal{A}$ to be the number of timesteps t such that the policy at time t, $\mathcal{A}_t$, is not-optimal from the current state, s_t at time t (formally $V^{\mathcal{A}_t}(s_t) < V^*(s_t) - \epsilon$).

2. **PAC-MDP:** An algorithm $\mathcal{A}$ is then said to be PAC-MDP (Probably Approximately Correct in Markov Decision Processes) if, for any ϵ and δ, the sample complexity of $\mathcal{A}$ is less than some polynomial in the relevant quantities $(|S|, |A|, \frac{1}{\epsilon}, \frac{1}{\delta})$, with probability at least $(1 - \delta)$.

There are two classes of algorithms in RL:

- **Model-Based:** In this category the agent focuses on learning the MDP dynamics, i.e. we try to model the system, particularly the transition function T and reward function R

- **Model-Free:** In this case we don't make any assumptions on the environment and learn a controller without learning a model.

3 KWIK

Since this the perspective from which we going to analyse the complexities of RL algorithms, we first introduce the notion of KWIK to the reader in this section. It was designed in mind with the idea of distinguishing between instances that have already been learned with sufficient accuracy and those whose outputs are still unknown. The core idea is that the learning algorithm needs to make only accurate predictions, but it can opt out of making a prediction by saying *I dont know* denoted by (λ). However, there must be a (polynomial) bound on the number of times the algorithm can respond λ. Whenever, the learning algorithm responds λ we assume there exists an expert or oracle (or the environment itself) which can give the true label. We call such a learning algorithm KWIK (know what it knows).

3.1 Definition

We denote input set as X and output set Y. The hypothesis class H consists of a set of functions from X to Y : $H \subseteq (X \to Y)$. The target function $h^* \in H$ is the source of training examples and is unknown to the learner. We assume the target function is in the hypothesis class (**realizable**).

The hypothesis class H and accuracy parameters and δ are known to both the learner and environment and the environment selects a target function $h^* \in H$ adversarially. The protocol is:

- The environment selects an input $x \in X$ adversarially and informs the learner.
- The learner predicts an output $\hat{y} \in Y \cup \{\lambda\}$
- If $\hat{y} = \lambda$, it should be accurate: $|\hat{y} - y| \leq \epsilon$, where $y = h^*(x)$. Else, the entire run is considered a failure. The probability of a failed run must be bounded by δ.
- Over a run, the total number of steps on which $\hat{y} = \lambda$ must be bounded by $B(\epsilon, \delta)$, ideally polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$, and parameters defining H.
- If $\hat{y} = \lambda$, the learner makes an observation $z \in Z$ of the output, where $z = y$ in the deterministic case, $z = 1$ with probability y and 0 with probability $1 - y$ in the Bernoulli case,or $z = y + \eta$ for zero-mean random variable η in the additive noise case.

3.2 Comparison to PAC and Mistake Bound

In the PAC model, the learner is provided with the correct outputs for an initial sequence of inputs. After that point, the learner is responsible for producing accurate outputs for all new inputs. Inputs are drawn from a fixed distribution.

In the Mistake Bound (MB) model, the learner is expected to produce an output for every input. Labels are provided to the learner whenever it makes a mistake. Inputs are selected adversarially, so there is no bound on when the last mistake might be made. However, there is guarantee that the total number of mistakes is small (asymptotically speaking ratio of incorrect to correct $\to 0$)

The KWIK model contains the elements of both PAC and MB models. A KWIK algorithm is not allowed to make mistakes (like in PAC) and also the inputs are selected adversarially (like in MB). The major difference is that instead of bounding the mistakes (in MB model), in KWIK we make a bound on number of requests, λs. Also KWIK doesn't make any assumption on independency of the samples it can be used in scenarios where distribution is dependent in complex ways on the KWIK algorithm's behavior.

Remark Note that though on the first glance KWIK seems very ambiguous but in my opinion, the need for KWIK is precisely because of the failure of PAC and MB to address the RL settings, because:

- PAC is based on i.i.d., whereas i.i.d. implies that the learner cannot improve (change) behaviour .
- MB allows for mistakes which means that a high reward can be assumed low-suboptimal

There are many other frameworks which also address this issue, but because the notion of λs can be related to budget, which made me more inclined towards KWIK.

3.3 Examples

In order to familiarize reader to KWIK model we'll walk through a few algorithms which we are going later in Section 4.

Algorithm 1 (memorization) The memorization algorithm can learn any hypothesis class with input space X with a KWIK bound of $|X|$. This algorithm can be used when the input space X is finite and observations are noise free.

The algorithm simply keeps a mapping $\hat{h}$ initialized to $\hat{h}(x) = \lambda$ for all $x \in X$. When the environment chooses an input x, the algorithm reports $\hat{h}(x)$. If $\hat{h}(x) = \lambda$, the environment will provide a label y and the algorithm will assign $\hat{h}(x) := y$. It will only report λ once for each input, so the KWIK bound is $|X|$.

Algorithm 2 (enumeration) When the hypothesis class H is finite, the enumeration algorithm can learn with a KWIK bound of $|H| - 1$.

The algorithm keeps track of $\hat{H}$,the version space, and initially $\hat{H} = H$. Each time the environment provides input $x \in X$, the algorithm computes $\hat{L} = \{h(x)|h \in H\}$. We have three scenarios:

- If $|\hat{L}| = 0$, the version space has been exhausted and the target hypothesis candidates are present, this implies that the hypothesis is not in the hypothesis class ($h^* \in H$).
- If $|\hat{L}| = 1$, it means that all hypotheses left in H agree on the output for this input, and therefore the algorithm returns $\hat{y} \in \hat{L}$.
- If $|\hat{L}| > 1$, two hypotheses in the version space conflict. Therefore the algorithm returns λ and receives the true label y. Since there was a conflict, this means there must be some $h \in H$ such that $h(x) \neq y$. Therefore, the new version space must be smaller than the previous (as we eliminate the h which were incorrect). Before the next input is received, the version space is updated and if $|H| = 1$ at any point then we'll only have $|\hat{L}| = 1$ and algorithm will no longer return λ. Therefore, $|H| - 1$ is the maximum number of λs the algorithm can return.

Algorithm 3 (Coin-learning) We have a biased coin whose unknown probability of heads is p. But in this case, observations are noisy. Instead of observing p, we see either 1 (with probability

p) or 0 (with probability $1-p$). Each time the algorithm says λ, it gets an independent trial that it can use to compute $\hat{p} = \frac{1}{T}\sum_{t=1}^{T} z_t$, where z_t is the tth observation in T trials. The number of trials needed before we are $1-\delta$ certain our estimate is within can be computed using a Hoeffding bound:

$$T \geq \frac{1}{2\epsilon^2}\ln\frac{2}{\delta} = O\left(\frac{1}{\epsilon^2}\ln\frac{1}{\delta}\right)$$

Thus, we can predict the probability that a biased coin will come up heads given Bernoulli observations with a KWIK bound of $B(\epsilon,\delta) = O\left(\frac{1}{\epsilon^2}\ln\frac{1}{\delta}\right)$

Algorithm 4 (Union) Let $F : X \to Y$ be the set of functions mapping input set X to output set Y . Let $H_1, ..., H_k$ be a set of KWIK learnable hypothesis classes with bounds of $B_1(\epsilon,\delta), ..., B_k(\epsilon,\delta)$ where $H_i \subseteq F$ for all $1 \leq i \leq k$ (all the hypothesis classes share the same input and output sets). The union algorithm can learn the joint hypothesis class $H = \bigcup_i H_i$ with a KWIK bound of $B(\epsilon,\delta) = (1-k) + \sum_i B_i(\epsilon,\delta)$.

The main idea behind this is to observe that the union algorithm can be considered as a high-level abstraction of the enumeration algorithm (Algorithm 2). Like in enumeration, we maintains a set of active algorithms $\hat{A}$ one for each hypothesis class. Given an input x, the union algorithm queries each algorithm $i \in \hat{A}$ to obtain a prediction $\hat{y_i}$. Using the previous notation, $\hat{L} = \left\{\hat{y_i} | i \in \hat{A}\right\}$

If $\lambda \in L$, the correct output y. Any algorithm i for which $\hat{y} = \lambda$ is then sent the correct output y to allow it to learn.

If $\hat{L} > 1$, then there is disagreement among the underlying algorithms and the union algorithm reports λ (since there is conflict, this means atleast one of the algorithms has $y \neq h^*(x)$ and thus it needs to know which in order to remove that)

On each input for which the union algorithm reports λ, either one of the subalgorithms reported λ (at most $\sum_i B_i(\epsilon,\delta)$ times) or two algorithms disagreed and at least one was removed from $\hat{A}$ (at most $k-1$ times). Hence we have the bound $B(\epsilon,\delta) = (1-k) + \sum_i B_i(\epsilon,\delta)$

4 Model Based RL

In this section we'll analyze the bounds for Model based methods in RL, where an algorithm learns a mapping from the size n^2m input space of state–action–state combinations to probabilities via Bernoulli observations.

Let $X_1, ..., X_k$ be a set of disjoint input spaces ($X_i \bigcap X_j = \phi$ if $i = j$) and Y be output space. Let $H_1, ..., H_k$ be a set of KWIK learnable hypothesis classes with bounds of $B_1(\epsilon,\delta), ..., B_k(\epsilon,\delta)$ where $Hi \in (X_i \to Y)$. Then $\exists$ algorithm that can learn the hypothesis class $H \in (X_1 \bigcup ... \bigcup X_k \to Y)$ with bound of $B(\epsilon,\delta) = \sum_i B_i(\epsilon, \frac{\delta}{k})$.

We run the learning algorithm for each subclass H_i. When it receives an input $x \in X_i$, it queries the learning algorithm for class H_i and returns its response, which is accurate, by request. To achieve $1-\delta$ certainty, it insists on $1-\frac{\delta}{k}$ certainty from each of the subalgorithms. By the union algorithm (Algorithm 4), the overall failure probability must be less than the sum of the failure probabilities for the subalgorithms.

Extending this algorithm to model-based settings (n^2m input space), we can apply this algorithm over the set of individual probabilities learned in the coin-learning algorithm (Algorithm 3). The resulting KWIK bound $B(\epsilon, \delta) = O\left(\frac{n^2m}{\epsilon^2} \ln \frac{nm}{\delta}\right)$

Remark Note that this is the same bound found precisely by Kearns and Singh(2002) [3]. What I personally admire about this approach in comparison to [3] is the way how various KWIK learners can be combined to provide learning guarantees for more complex hypothesis classes. There exists work for a similar extension for factored-MDPs that consists four-level decomposition of our example algorithms.

Examples of PAC-MDP algorithms for finite-state, finite-action MDPs include E3 [3], Rmax [4] and delayed Q-learning [5]

5 Model Free RL

In this section I'll basically go through the work by Li and Littman (2010) [2], in which the authors tackle the problem of model free RL by reducing it to online regression as per the KWIK learning model (we'll define the online regression problem shortly). They also show that if the KWIK online regression problem can be solved efficiently, then the sample complexity of exploration of their algorithm is polynomial.

Remark The main reason for choosing this work is because their approach algorithm explicitly addresses the question of balancing exploration and exploitation, and then they analyse its sample complexity. Also in their work they have proof-of-concept examples.

5.1 KWIK Online Regression

Let $\mathcal{F}$ be a class of real-valued functions such that $f : \mathcal{R}^d \to [-1, 1]$ for each $f \in \mathcal{F}$. At timestep $t = 1, 2, 3, ...$, a KWIK online regression learner acts according to the following protocol:

- It receives an input vector $x \in \mathcal{R}^d$. (we use $||x||$ to denote the 2-norm)
- It provides an output $\hat{y}_t \in [-1, 1] \bigcup \{\lambda\}$, where λ is the same as per our previous notation, i.e. learner is unsure and requests actual label.
- If $\hat{y}_t = \lambda$, the agent observes the (possibly noisy) ground truth $y_t \in [-1, 1]$ with which it can improve its future predictions.

Also the non-i.i.d. assumption still is valid here.

Remark Though the authors only consider the linear regression in their case, I believe that since it's essentially an regression problem, therefore we can easily capture the non-linearity using the kernels methods.

5.2 REKWIRE (REinforcement learning based on KWIk online REgression)

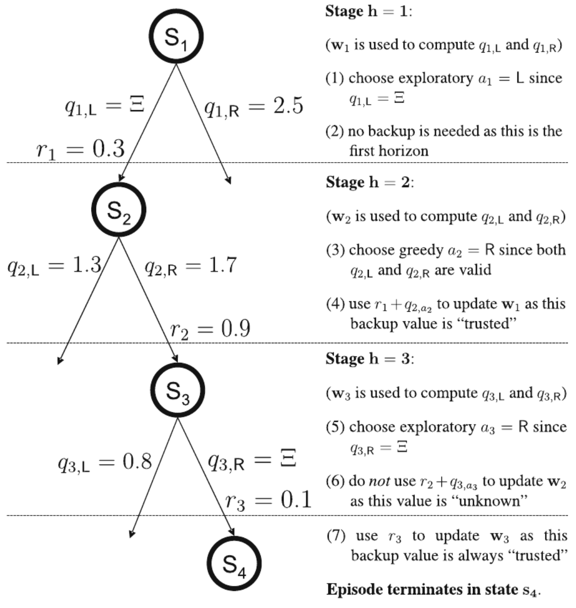
REKWIRE is the algorithm proposed by original authors of the work for H-horizon reinforcement learning in which value function approximations are used. In this section I'll try to briefly to summarize the gist of the algorithm, and would encourage the reader to look into original work to actually go through the algorithm. [MAYBE IN APPENDIX]

The authors of original work assume a set of d features are predefined: $\phi : S \times A \rightarrow [-1, 1]^d$. A Q-function can then be represented as a function $f \in \mathcal{F}$ over this feature vector: for each $h \in [H] : \hat{Q}_h(s, a) = f(\phi(s, a))$. In particular, the set of linear value functions is defined by:

$$\mathcal{F} = \{f | f(x) = w^T x, x \in \mathcal{R}^d\}$$

Basically, the algorithm learns the optimal state-value functions Q^* by treating them as H related KWIK online regression problems. It runs H copies of the base algorithm $\mathcal{A}_0$ to KWIK-learn the optimal value function Q^* for all stages h. The authors use the Bellman equation (like in every other model-free RL algorithm) to recursively define the Q^*_h function at stage h. Q^*_h is defined as the sum of immediate reward at stage h and the expected optimal Q-value of the next states (when a greedy policy is being followed). Therefore, the algorithm improves its value-function estimates by performing Bellman-backup-style updates.

The central idea behind the author's algorithm is that they do an efficient reduction by using a previous Q-value (which they call as backup value, since it is recursive of bellman-update) only when is *known.* A backup value is *known* only when the prediction made by $\mathcal{A}_0$ is valid.



S1
$q_{1,L} = \Xi$
$q_{1,R} = 2.5$
$r_1 = 0.3$
S2
$q_{2,L} = 1.3$
$q_{2,R} = 1.7$
$r_2 = 0.9$
S3
$q_{3,L} = 0.8$
$q_{3,R} = \Xi$
$r_3 = 0.1$
S4
Stage h = 1:
($\mathbf{w}_1$ is used to compute $q_{1,L}$ and $q_{1,R}$)
(1) choose exploratory a_1 = L since $q_{1,L} = \Xi$
(2) no backup is needed as this is the first horizon
Stage h = 2:
($\mathbf{w}_2$ is used to compute $q_{2,L}$ and $q_{2,R}$)
(3) choose greedy a_2 = R since both $q_{2,L}$ and $q_{2,R}$ are valid
(4) use $r_1 + q_{2,a_2}$ to update $\mathbf{w}_1$ as this backup value is "trusted"
Stage h = 3:
($\mathbf{w}_3$ is used to compute $q_{3,L}$ and $q_{3,R}$)
(5) choose exploratory a_3 = R since $q_{3,R} = \Xi$
(6) do *not* use $r_2 + q_{3,a_3}$ to update $\mathbf{w}_2$ as this value is "unknown"
(7) use r_3 to update $\mathbf{w}_3$ as this backup value is always "trusted"
Episode terminates in state s_4.

Figure 1: Algorithm in 3-horizon MDP where only two actions are allowed in every state, [L,R] [2]

The important theorems proved by the authors are:

Theorem 1 Let $\mathcal{A}_0(h)$ (be run with parameters: ϵ_h and δ_h) in REKWIRE, then:

- The number of λ outputted in stage $h \in [H]$ is at most $\sum_{l=h}^{H} \zeta_0(d, \frac{1}{\epsilon_l}, \frac{1}{\delta_l})$, where ζ_0 denotes the sample complexity of $\mathcal{A}_0$
- The total number of λ outputted during the whole run of REKWIRE in all stages is at most $\sum_{h=1}^{H} \Big(h.\zeta_0(d, \frac{1}{\epsilon_l}, \frac{1}{\delta_l})\Big)$

Theorem 2 Given any $\epsilon, \delta > 0$, if we run $\mathcal{A}_0(h)$ with $\epsilon_h = O(\frac{\epsilon}{H^3})$ and $\delta_h = \frac{\delta}{2H}$ in REKWIRE, then the policy used by the agent is optimal except in $o\left(\frac{H^3}{\epsilon}\left(\zeta_0(d, \frac{H^3}{\epsilon}, \frac{H}{\delta}) + \log\frac{1}{\delta}\right)\right)$ episodes, with probability at least $(1 - \delta)$.

We are not going to dwell into proofs of these theorems (as that'll make the scope of this paper too large and thus the reader is encouraged to read the proofs from actual work), but basically, this theorem gives a polynomial upper bound on the number of samples needed to learn a near-optimal policy.

Remark Authors claims that the sample complexity of exploration of their algorithm is polynomial, but they make too many assumptions, some of which are not trivial, particularly:

- The most underlying assumption is on that the KWIK online regression problem can be solved efficiently (i.e. polynomial sample-complexity) but that in itself is a very hard problem [1]. This becomes even more challenging in the cases where "realizability" condition is not valid (the true target function doesn't exist in the hypothesis space)
- Since it is online system there also exists a possibility that Q_h^* may change with time, and thus the back-ups may not be valid anymore. The authors haven't this issue at all in their work.

Remark A particular interesting way to use idea from this is will be apply to apply similar concept in the traditional RL algorithms such as Value Iteration, but with an added sense of budget on the exploration within the computation of value iteration itself.

6 Final Remarks

The KWIK provides us with an algorithm where the learner is aware of its prediction accuracy, which is the authors claim is fundamental to the sample complexity of exploration analysis. However, there still exists a few issues which I believe also need to be addressed:

- Though this allows the learner to make a decision on when to explore or exploit, but there is still an issue with regard to how an learner's confidence in its predictions should change w.r.t time. Say, after a particular time and transition, the optimal action for a particular state changes, and since KWIK doesn't allows for any mistakes it will regard that as a fail. Maybe we need some relaxations (similar to slack variables in SVMs) to address this issue.
- Also for real-world data, how can KWIK be adapted to apply in an unrealizable setting. As for real-world problems the target hypothesis can be chosen from outside the hypothesis class H.
- Another potential problem is that in some RL problems there is not always an oracle or expert present which can respond to the label request, λ
- Also it is known that KWIK is a harder-to-learning model than the PAC and the MB models. Aside from the above the original authors have cited many open-problems [1] which still haven't been addressed.

However KWIK still provides an interesting perspective and I believe though it may not be a perfect framework for RL , but other machine-learning problems could benefit from this perspective, particularly:

- In particular, the Imitation Learning and Learning from Demonstrations learners, I believe can be easily analyzed with this perspective (since there is always an oracle and environment is rarely affected from external changes)
- It would be also interesting to apply this approach to Bandit problems and its variants.
- Active-learning and Anomaly detection, also have an uncertainty factor, where the learners need to have confidence in prediction vs exploitation and require some reasoning as to discriminate if the current input is predictable from previous inputs.

References

[1] Li, L., Littman, M.L., Walsh, T.J.: Knows what it knows: A framework for self-aware learning. In: Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08), pp. 568–575 (2008)

[2] Reducing reinforcement learning to KWIK online regression, Lihong Li · Michael L. Littman (2010)

[3] Kearns, M.J., Singh, S.P.: Near-optimal reinforcement learning in polynomial time. Mach. Learn. 49, 209–232 (2002)

[4] Brafman, R.I., Tennenholtz, M.: R-max—a general polynomial time algorithm for near-optimal reinforcement learning. J. Mach. Learn. Res. 3, 213–231 (2002)

[5] Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: Proceedings of the Twenty-Third International Conference on Machine Learning, pp. 881–888 (2006)

[6] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)

[7] Kakade, S. M. (2003). On the sample complexity of reinforcement learning. Doctoral dissertation, Gatsby Computational Neuroscience Unit, University College London.