UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Department Applied Mathematics
Stochastic Operations Research

Analyzing the convergence of Q-learning through Markov Decision Theory

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Author
Maik Overmars

Supervisors
Prof.dr. R.J. Boucherie
Dr.ir. J. Goseling

Graduation Committee
Prof.dr. R.J. Boucherie
Dr.ir. J. Goseling
Dr. A. Antioniadis

October 4, 2021

Abstract

In this thesis the convergence of Q-learning is analyzed. Q-learning is a reinforcement learning method that has been studied extensively in the past with convergence being shown in multiple cases. Existing convergence proofs rely on multiple assumptions on the algorithm. This is most evident when function approximation is introduced, leading to the deadly triad. The deadly triad is a combination of three properties, including function approximation, which leads to instability and divergence in methods where all three are present. The assumptions needed to still obtain convergence are very strict, which makes them undesirable when applying the algorithm. In this thesis, we provide an alternative method of analyzing Q-learning. Our method analyzes Q-learning by taking into account the properties of the relevant Markov decision process (MDP). This method is applied on MDPs which are akin to a birth-death process. Multiple variations of these MDPs are considered by changing the reward structure. In the most simple case of constant reward exact convergence is shown using our method. For more complex reward structures we show that upper and lower bounds can be found on the expected value of the limiting distribution of Q-learning. Additionally, we show how these results can be extended to a large class of MDPs. Finally, the obtained bounds are examined numerically and the bounds are compared by varying the properties of the MDP. While our method doesn't consider the case of a general MDP, it is a first step in a different approach of tackling the problem of convergence without having to rely on the aforementioned assumptions. As such, this method may be able to circumvent the issues that other methods encounter.

Contents

Chapter 1

Introduction

This thesis concerns Q-learning, a popular method in the field of reinforcement learning. Q-learning and other reinforcement learning method deal with the case of optimizing the reward in an environment where we sequentially make decisions. Many problems can be formulated in such a way, think of board games where we try to win by making smart decisions. Reinforcement learning methods learn through experience. By taking actions in the environment and observing the results, the methods attempt to find out which actions lead to good results. Of course, we want to know if we can guarantee convergence to optimal actions by applying these methods. To analyze convergence, Markov decision theory will be used. This field of mathematics formalizes the sequential decision making problem and provides tools for analysis. We will use this field to analyze the convergence of Q-learning.

The method of Q-learning was introduced by Watkins [23] [22], with convergence being proven by Tsitsiklis [18] and Jaakkola [6]. There have been multiple papers which look at the convergence of Q-learning when the state-action values are approximated by some value function. Melo [11] provides assumptions under which they show convergence for linear function approximation. Xu considers the case of deep Q-learning, where neural networks are used as function approximators. Deep Q-learning in particular has seen a lot of attention in recent years, inspired by Mnih [12]. This paper uses neural networks to learn how to play Atari 2600 games.

While multiple results on the convergence of Q-learning have been shown, they tend to rely on some inconvenient assumptions. In standard Q-learning we need to make assumptions on the step size in the algorithm. When function approximation is introduced Q-learning runs into the deadly triad. This is a combination of three properties that leads to instability and divergence. To still obtain convergence, strong requirements are placed on the behavioural policy. This thesis approaches Q-learning from another direction. We analyze its convergence by taking into account the properties of the Markov decision process. This analysis is done on various types of MDPs and convergence results are shown in multiple cases.

Our thesis is structured into multiple chapters. We will briefly go over each of the chapters and which contributions are made in them. In Chapter 2, we will formulate the problem of finding optimal actions through a Markov decision process and show relevant methods and results from the field of Markov decision theory. In Chapter 3, we will describe reinforcement learning methods and some results on their convergence. We will additionally provide our own variation of an existing proof of the convergence of Q-learning. Chapter 4 deals with function approximation of the value function. This is needed when state and action spaces of our problem become large. Function approximation leads to some problems for proving convergence, which will be examined in this chapter.

Chapters 5 and 6 mostly contain our contributions on the topic of convergence of Q-learning. We will give a short overview of the contributions of this thesis.

- In Chapters 3 and 4 we do an extensive review of multiple reinforcement learning methods and convergence results for these methods from the literature. Thus, these chapters function as a literature review, most notably surrounding Q-learning.
- In Chapter 3 we give a detailed existing proof of the convergence of Q-learning. Our proof is mostly based on earlier proofs. However, earlier proofs by Jaakkola [6] and Melo [10] leave out details and important steps. Another proof by Tsitsiklis [18] uses measure theory, which is out of the scope of this thesis. We provide a precise and detailed proof that gives understanding into Q-learning. This will help to contextualize the results in our thesis.

- In Chapter 5 we apply a new method to analyze the convergence of Q-learning. In our method we analyze which rewards are obtained in the Q-learning algorithm by using the properties of the MDP on which we apply it. We will consider birth-death type MDPs, where we can only take one step to the left or right in every state. First the case of constant rewards is considered. For this case, we show that exact convergence holds and we also provide finite-time bounds.
- Of course, the constant reward case is quite restrictive. As such, we extend our results to a reward structure where we divide the state space in two and subsequently three parts. Each part will have a different reward. For these cases we derive an equation for the random variable to which Q-learning will converge. While we can't calculate this random variable explicitly, we show how its expectation can be bounded from above and below.
- We show how the results can be extended to a more general class of Markov decision processes. Instead of a state-space that consists of two or three parts with a different reward, we will consider the case of any number of parts. For this case we show which equations need to be derived to again achieve upper and lower bounds.
- Finally, in Chapter 6 numerical studies are done to analyze the performance of our method. We calculate the upper and lower bounds that we derived for the cases of a two and three-part state space. We also show how these error bounds change when varying the properties of the MDP.

As we have mentioned above, Chapter 5 provides our main theoretical results on the convergence of Q-learning. Our method takes another approach than results already established in the literature. Through this approach, we can hope to circumvent some of the issues that other methods encounter. While we don't consider a general MDP, our results already encompass a large class of Markov decision processes and can be even be extended further. In Chapter 6 we analyze the performance of our method to show when it is valid to use and if it leads to good results. Finally, Appendix A contains a different approach that was investigated in my research, also intended to analyze the convergence of reinforcement learning methods. This was done by providing bounds on the difference between Markov reward processes. While not used in the final paper, it did lead to some interesting results that were worth an inclusion in the appendix.

Chapter 2

Preliminaries - Markov decision theory

The goal of reinforcement learning methods is to optimize the reward in an environment by sequentially making decisions. This environment will be stated in the form of a Markov decision process. Through the mathematical form of the MDP we will be able to analyze the behaviour of Q-learning. In this chapter we will describe an MDP and the field of Markov decision theory.

2.1 Markov decision process

In each time step, a Markov decision process is in a certain state. Then, the decision maker, in reinforcement learning often called an *agent*, takes an action. This actions leads to some reward and a transition to a new state. This process can repeat indefinitely or stop after a certain number of time steps. Formally, we define a Markov decision process as follows.

Definition 2.1. A Markov decision process is defined as a tuple of $(\mathcal{S}, \mathcal{A}, P, r)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ defines a set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a conditional probability distribution which determines transitions and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function.

We call this a Markov decision process because it satisfies the Markov property. This property says that the dynamics of the environment only depend on the current state and action, and not on all past states and actions. Mathematically we can formulate this as

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1}|s_t, a_t).$$

Here we denote the time step by t such that s_t and a_t are the state and action at time t. The number of time steps in an MDP is called the horizon and will be denoted by T. To determine what actions should be taken in which states, a policy can be used. A policy π is a conditional probability distribution which determines the probability that we take some action depending on the current state. Thus, when the decision maker is in state s he chooses action a with probability $\pi(a|s)$.

2.1.1 Optimization objective

Now we wish to maximize the reward over the horizon T. There are multiple criteria that can be used for this. In the case of a finite horizon, we could just add up the reward in each time step. However, this doesn't work for an infinite horizon. In this paper, we will consider the discounted return objective, since that is the objective used in Q-learning. This objective discounts future reward by some factor $\gamma \in [0, 1)$. Given some policy π, we will then generate an *episode* of states, actions and rewards. The long-term discounted reward for such a policy is then given by

$$V^{\pi}(s) = \mathbb{E}_{a_t \sim \pi, s_t \sim P}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)|s_0 = s]. \tag{2.1}$$

Thus, this total return is the expectation over all episodes of the sum of discounted rewards. By discounting the rewards we ensure that the total return will remain bounded in the case of an infinite horizon. Also, we place more emphasis on current rewards than future ones, which is relevant in many applications. We will

henceforth call $V^\pi(s)$ the state value function for policy π and state s. The goal is to find a policy π^* which maximizes this value function. Mathematically, we obtain $\pi^* = \arg\max_\pi V^\pi$. The optimal value function is defined as $V^* := \max_\pi V^\pi$.

2.2 Markov reward process

If we have some policy π, we know the probabilities of taking each action in some state. Thus, we can calculate the exact probability of transitioning from state s to some other state s' using the probabilities from the policy and transition function. We then obtain Markov reward process (MRP), which is formally defined as follows.

Definition 2.2. A Markov reward process (MRP) is a stochastic process $\{(X_t, r(X_t)) : t = 0, 1, \ldots\}$ with rewards $r(s)$ for each state $s \in \mathcal{S}$ and transition probability P of a Markov chain $\{X_t : t = 0, 1, \ldots\}$. We will denote an MRP by the tuple $(\mathcal{S}, P, r)$.

As mentioned above, we can obtain an MRP from an MDP using a policy. To see this, let π be any policy that assigns probability $\pi(a|s)$ to taking action a in state s. Then we define reward vector r_π and transition probability matrix P_π by

$$r_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) r(s, a),$$
$$P_\pi(s, s') = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s, a).$$

Thus, we have obtained a Markov reward process $(\mathcal{S}, P_\pi, r_\pi)$. The concept of an MRP is useful whenever we want to evaluate a policy for some MDP. This is because an MRP is just a Markov chain with an additional reward function. As a result, we can use Markov chain theory in our analysis of a policy.

2.2.1 Markov chain properties

In this section we will look at properties of Markov chains. These apply to Markov reward processes and will become useful in our analysis. Let $\{X_t, t = 0, 1, \ldots\}$ be some Markov chain with transition matrix P. First of all, we will consider the concept of states that can be reached from some state.

Definition 2.3. A state s' is *accessible* from s if $Pr\{X_t = s'|X_0 = s\} > 0$ for some $t \geq 0$. Two states s and s' are said to *communicate* if both are accessible from the other state.

When some state i communicates with state j and j communicates with k then state i also communicates with state k. This is because states i and k will both be accessible from one another. A group of communicating states in a Markov chain is said to be a *class*. Two classes are always either equal or disjoint, because if some states of either class can communicate with each oter then all states in each class could communicate with one another. If the entire Markov chain belongs to the same class, meaning all states communicate with one another, the Markov chain is said to be *irreducible*. Now we will consider 2 properties: how often we return to a state and the times at which we return to that state.

Definition 2.4. A state s is *recurrent* if $\sum_{t=0}^{\infty} Pr\{X_t = s'|X_0 = s\} = \infty$ and *transient* if $\sum_{t=0}^{\infty} Pr\{X_t = s'|X_0 = s\} < \infty$.

When the expected time to return to a state is finite, that state is said to be *positive recurrent*. There are states which are recurrent, but not positive recurrent, meaning that it takes an infinite amount of time on average to return to a state, while still being visited infinitely often in the limit. In finite state Markov chains, recurrent states are always positive recurrent. Another property of Markov chains is the times at which we return to the same state, leading to the concept of *periodicity*.

Definition 2.5. A state s has *period* d if d is the greatest common divisor of the number of transitions by which s can be reached. Mathematically, this means

$$d = \gcd\{t > 0 : Pr\{X_t = s|X_0 = s\} > 0\}. \tag{2.2}$$

For example, we can have a Markov chain in which we only return to a state at times 2, 4, 6, etc. in which case this state has period 2. A state s is *aperiodic* if it has a period of 1. Periodicity, recurrence and positive recurrence are all class properties. This means that whenever one state in a communicating class has that property, all other states in that class have the same property. Irreducible Markov chains with a finite state space will always be positive recurrent, because not every state can become transient. There will always be

at least one state to which keep returning, so we will also return to each other state infinitely often because they share the property of positive recurrence.

Finally, we say that a state is *ergodic* whenever it is aperiodic and positive recurrent. A Markov chain is ergodic when all its states are irreducible and ergodic. These properties are necessary to show convergence in some of the proofs we will see later. They also influence the existence of a *stationary distribution*, denoted by μ, of the Markov chain.

Theorem 2.1. *Let $\{X_t, t = 0, 1, \ldots\}$ be an ergodic Markov chain with transition probability matrix P. Then, $\lim_{m\to\infty} Pr\{X_t = s'|X_0 = s\}$ exists and is independent of s. Letting $\mu(s') = \lim_{m\to\infty} Pr\{X_t = s'|X_0 = s\}$, then μ satisfies*

$$\mu(s') = \sum_{s\in\mathcal{S}} \mu(s)P(s, s'),$$
$$\sum_{s\in\mathcal{S}} \mu(s) = 1. \tag{2.3}$$

For an ergodic Markov chain, the stationary distribution $\mu(s)$ both equals the limiting probability that the chain is in state s and the long run proportion of time spent in state s.

2.3 Optimality

We have seen that a policy π leads to some value function V^π and induces some Markov reward process. However, we are interested in finding the policy which maximizes our total return given in Equation (2.1). In this section we will give optimality conditions, which will be used to construct algorithms to find optimal policies. We will first consider how to find the value function associated with some policy π. From Markov decision theory we use the Bellman operator $\mathcal{B}^\pi$, which is defined as

$$\mathcal{B}^\pi V^\pi(s) := \mathbb{E}_{a\sim\pi, s'\sim P}[r(s, a) + \gamma V^\pi(s')|s]. \tag{2.4}$$

This gives us the value function for a given policy, but we instead want to find the optimal policy π^*. To do this, we define the Bellman optimality operator which is given by

$$\mathcal{B}V(s) := \max_{a\in A} \big(r(s, a) + \gamma \sum_{s'\in S} P(s'|s, a)V(s')\big). \tag{2.5}$$

Both Bellman operators are contractions, which we will later prove for the case of a state-action value function in Section 3.3. As such, the fixed-point equations $\mathcal{B}^\pi V = V$ and $\mathcal{B}V = V$ have unique solutions V^π and V^* respectively by the Banach fixed-point theorem. From Markov decision theory we know that the value function V^{π^*}, obtained by solving Equation (2.4) for optimal policy π^*, solves the fixed-point equation $\mathcal{B}V = V$. As a result, we have that $V^* = V^{\pi^*}$, motivating our use of the Bellman optimality operator. Thus, we can define a policy based on the unique solution of (2.5) which will be optimal.

In reinforcement learning we often don't use the state value function $V(s)$, but instead the state-action values $Q(s, a)$. These give the value of being in state s and choosing action a. Analogous to Equation (2.1), we can define the long term reward of a policy π, given that we start in state s and taking action a.

$$Q^\pi(s, a) = \mathbb{E}_{a_t\sim\pi, s_t\sim P}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a] \tag{2.6}$$

Again we also have Bellman operators $\mathcal{B}^\pi$ and $\mathcal{B}$, in this case given by:

$$\mathcal{B}^\pi Q(s, a) := r(s, a) + \gamma\mathbb{E}_{s'\sim P, a'\sim\pi}[Q(s', a')], \tag{2.7}$$
$$\mathcal{B}Q(s, a) := r(s, a) + \gamma\mathbb{E}_{s'\sim P}[\max_{a'\in A} Q(s', a')]. \tag{2.8}$$

Again we have that Q^π is the unique solution of $Q = \mathcal{B}^\pi Q$ and Q^* is the unique solution of $Q = \mathcal{B}Q$.

2.4 Dynamic programming

Using the value function defined in the previous section, we wish to find optimal value functions. In this section we will describe two dynamic programming methods, which both obtain a (near-)optimal policy. These methods both assume that we have a finite MDP, which means that the state space $\mathcal{S}$ and action space $\mathcal{A}$ are both finite. The methods we will consider are policy iteration and value iteration. These both try to find the optimal policy π^*, for which they use the Bellman operators of Equations (2.4) and (2.5).

2.4.1 Policy iteration

In policy iteration we try to find the optimal policy by repeating two steps: *policy evaluation* and *policy improvement*. In policy evaluation we want to calculate the value function V^π given some policy π. In policy improvement we try to find a better policy by choosing actions which maximize the reward given the value function V^π. Policy iteration is given in Algorithm 1.

Algorithm 1: Policy iteration for discounted reward

Initialize arbitrary policy π_0 and set $k \leftarrow 0$ and *policy-unequal* $\leftarrow$ true.
while *policy-unequal* **do**
  Evaluate policy π_k by solving the linear system

$$v(s) = \sum_{a \in \mathcal{A}} \pi_k(a|s)\big(r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)v(s')\big), \quad s \in \mathcal{S} \tag{2.9}$$

  to obtain v_k.
  for $s \in \mathcal{S}$ **do**
    Calculate the set of best actions $A_k(s)$ with respect to current value function v_k.

$$A_k(s) \leftarrow \arg\max_{a \in \mathcal{A}} (r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)v_k(s')) \tag{2.10}$$

    Improve policy by setting

$$\pi_{k+1}(a|s) \leftarrow 1$$

    for some $a \in A_k(s)$, where we choose a such that $\pi_k(a|s)$ and $\pi_{k+1}(a|s)$ are equal if possible. Hence, we keep the policy the same.
  end
  if $\pi_{k+1} = \pi_k$ **then**
    policy-unequal $\leftarrow$ false
  end
  Increment k by 1.
end
Return optimal policy π_k.

It can be proven that policy iteration converges to the optimal policy π^*. For this proof we refer to the book by Puterman [13] on Markov decision theory.

2.4.2 Value iteration

A disadvantage of policy iteration is that in each iteration we have to solve for the value function V^π, which might itself be some iterative scheme. Value iteration is obtained by using the Bellman optimality operator of Equation (2.5) in an iterative scheme The goal is to converge to the optimal value function V^*, which can be used to determine an optimal policy. Value iteration is given in Algorithm 2.

Algorithm 2: Value iteration for discounted reward

Initialize arbitrary value function v_0 and set $k \leftarrow 0$. Let $\epsilon > 0$ be some desired accuracy.
while $||v_k - v_{k-1}|| > \epsilon(1-\gamma)/(2\gamma)$ **do**
  for $s \in \mathcal{S}$ **do**

$$v_{k+1}(s) = \max_{a \in A} \big(r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)v_k(s')\big). \tag{2.11}$$

  end
  Increment k by 1.
end
Obtain policy $\pi \approx \pi^*$ with $\pi(a|s) = 1$ where

$$a \in \arg\max_{b \in A} \big(r(s,b) + \gamma \sum_{s' \in S} P(s'|s,b)v_{k+1}(s')\big) \text{for all} s \in \mathcal{S}$$

Value iteration converges to an ϵ-optimal policy, where a policy π_ϵ is ϵ-optimal if $V^{\pi_\epsilon}(s) \geq V^*(s) - \epsilon$. Thus, by making ϵ sufficiently small, we converge to the optimal policy. For a proof of this result we again refer to the book by Puterman [13].

Chapter 3

Reinforcement learning

In the last chapter we saw how to formulate the problem of taking optimal actions in some environment through Markov decision theory. We also looked at methods for finding an optimal policy. However, these methods use the model of the Markov decision process including the transition probabilities and the reward function. Reinforcement learning deals with the case where this model is not known. In this chapter we will give an introduction to some reinforcement learning methods. We will first consider *temporal difference learning*, which is one of the earliest reinforcement learning methods. Temporal difference learning is related to Q-learning, which we will describe afterwards along with another method called SARSA. Finally, we will give an existing convergence proof of Q-learning, adapted from earlier proofs by Melo [10], Jaakkola [6] and Tsitsiklis [18].

3.1 Temporal difference learning

Temporal difference learning is one of the earliest and most common methods in reinforcement learning. It was first introduced by Sutton [17]. The method is influenced by the dynamic programming methods described in the previous chapter, since it also attempts to find a value function. However, temporal difference methods only evaluate policies. It is thus not a *control* method which tries to find optimal policy. As such, temporal difference methods attempt to find the value function V^π for some policy π. Temporal difference learning learns through experience by observing states, actions and rewards following the policy π.

From now on, we will assume that our MDP is finite, with known state and action space. Furthermore, the method learns a state value function $V \in \mathbb{R}^{|\mathcal{S}|}$ stored in look-up table such that each state has a separate value. The initial values are arbitrary, but where we need to know them we will consider them to equal 0. Temporal difference methods will then iteratively update this value function in order to find the value function associated with a fixed policy π.

3.1.1 Monte Carlo methods

Before proceeding to temporal difference methods, we will first consider *Monte Carlo* methods for learning the state value function V^π. Recall that this value function is given by Equation (2.1). To estimate this expectation, Monte Carlo methods generate an *episode* of states, actions and rewards. For an initial state s_0 and policy π, we thus have states and actions $(s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ generated according to the policy π and transition probabilities p. The incurred rewards in this episode are given by $r(s_t, a_t)$. The total discounted return from state s_t at time t then equals

$$G_t = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \ldots \gamma^{T-t} r(s_T, a_T).$$

In principle we can directly use this discounted return to estimate our value function, as this is a realization of the return in an episode. Instead, we usually use a learning rate $0 \leq \alpha \leq 1$ and then look at the difference between G_t and our previous estimate $V(s_t)$ to obtain a new estimate.

$$V(s_t) \leftarrow V(s_t) + \alpha\big[G_t - V(s_t)\big]. \tag{3.1}$$

Notice that when $\alpha = 1$ we simply set our estimation to the actual discounted return. In this update rule G_t is called the *target*, because we want our approximation to move closer to G_t. This update rule is applied to every state s_t in the generated episode using the rewards that follow this state. Episodes can be generated repeatedly until we have reached convergence. For more details on Monte Carlo methods we refer to chapter 5 of [15].

3.1.2 TD(0) and n-step bootstrap

A problem of Monte Carlo methods is that we have have to generate an episode, which is only possible in episodic tasks with a finite horizon T. Even when the horizon is finite, we have to wait until the end of the episode to update our values. Temporal difference methods solve this by using previous estimates of the value function and new incurred rewards. In the most basic method, TD(0), we use $r(s_t, a_t) + \gamma V(s_{t+1})$ as a target instead of the actual return G_t that we used in Monte Carlo methods. We the obtain the following update rule.

$$V(s_t) \leftarrow V(s_t) + \alpha\big[r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)\big] \tag{3.2}$$

For $\alpha = 1$ this update rule is very similar to the policy evaluation step in Equation (2.9). The process of using previous estimates of the value function to obtain a new estimate is called *bootstrapping*. Furthermore, we define $\delta_t = r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$ as the *temporal difference*, sometimes also called the TD(0)-error. TD(0) is a one-step method, since we only use the rewards that were incurred within one time step. The full TD(0) algorithm is given in Algorithm 3.

Another possibility is to take multiple steps. Instead of only looking at the one-step return, we can take any number of steps n and use the following n-step return as a target

$$G_{t:t+n} = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \ldots + \gamma^{n-1} r(s_{t+n-1}, a_{t+n-1}) + \gamma^n V(s_{t+n}). \tag{3.3}$$

For large values of n, this method approaches the aforementioned Monte Carlo methods. This n-step method is also a bootstrapping method, since we still rely on previous estimation of value functions. When n is small, we rely more on the previous estimate of the value function. This has as an advantage that our update is less delayed. However when the previous value estimates are far off the optimal values, we can propagate these errors with the new updates. When n is larger, we rely less on previous value functions, which means there is less of a bias in the updates. However, the updates will be delayed by more time steps and the variance is higher, since the outcome of rewards can differ a lot between episodes.

Algorithm 3: TD(0)

Input: A policy π to be evaluated, a number of iterations t_{iter} and initial state s_0.
Initialize the value function $V(s) = 0$ for all $s \in \mathcal{S}$ and set $t = 0$.
while $t < t_{iter}$ **do**
  Sample action from our policy $a_t \sim \pi(\cdot|s_t)$.
  Observe reward $r(s_t, a_t)$.
  Sample next state $s_{t+1} \sim P(\cdot|s_t, a_t)$.
  Update the value function of state s_t according to Equation (3.2).
  Increment t by 1.
end

3.1.3 TD(λ)

In n-step methods we have to pick a suitable value for n, but this can greatly depend on the domain. Instead, we could also look at an average over n-step returns for different values of n. As a example, if we use the target $0.5G_{t:t+2} + 0.5G_{t:t+4}$ we consider both a two-step and a four-step return. Notice that the weights must sum up to 1. The key idea behind TD(λ) method is to average over all possible n-step returns by cleverly using a parameter λ. This leads to using the λ-return G_t^λ as a target

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{3.4}$$

We have that $\lambda \in [0, 1]$ so the weights sum up to 1, since they form a geometric series. It is clear now that instead of using a single value of n, we take an exponential average over all n-step returns. When $\lambda = 0$ this sum evaluates to $G_{t:t+1} = r(s_t, a_t) + \gamma V(s_{t+1})$ so we obtain the TD(0) method. When λ approaches 1, this sum gets close to the actual return G_t, which we recognize as the Monte Carlo method. By simply varying this value of λ we can thus modify our method how we want.

An apparent problem in this method is the infinite sum, since we will rely on n-step returns $G_{t:t+n}$ with very large values of n. Fortunately, there is another way to perform updates in TD(λ) without having to evaluate G_t^λ. The key to doing this are *eligibility traces*. As Sutton [15] calls it, we have considered the *forward view* up till now. This is because when we update some state s_t at time t, we will look at future states $s_{t+1}, s_{t+2}, \ldots$ to update this state. With eligibility traces we switch to the *backward view*. In the backward view, we instead use the current result of state s_t to update previous states in TD(λ).

The eligibility trace is a vector which keeps track how recently and frequently each previous state was visited. In each time step t this eligibility trace, given by $z_t \in \mathbb{R}^{|\mathcal{S}|}$, is updated according to the state that was visited. We also calculated the temporal difference δ_t. Using the eligibility trace and temporal difference, we then update each state $s \in \mathcal{S}$. The exact workings of TD(λ) are given in Algorithm 4.

Algorithm 4: TD(λ)

Input: A policy π, a number of iterations t_{iter}, initial state s_0 and $\lambda \in [0, 1]$.
Initialize eligibility trace $z_0(s) = 0$ and $V(s) = 0$ for all $s \in \mathcal{S}$ and set $t = 0$.
while $t < t_{iter}$ **do**
Sample action from our policy $a_t \sim \pi(\cdot|s_t)$.
Observe reward $r(s_t, a_t)$.
Sample next state $s_{t+1} \sim P(\cdot|s_t, a_t)$.
Calculate the temporal difference

$$\delta_t = r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t). \tag{3.5}$$

Update the eligibility trace

$$z_{t+1}(s) \leftarrow \gamma\lambda z_t(s) + \mathbb{1}(s_t = s) \tag{3.6}$$

for all states $s \in \mathcal{S}$.
Update the value function

$$V(s) \leftarrow V(s) + \alpha\delta_t z_{t+1}(s) \tag{3.7}$$

for all $s \in \mathcal{S}$.
Increment t by 1.
end

This algorithm updates each state s according to the observed temporal difference δ_t. The eligibility trace then weighs this temporal difference for a state by a factor $(\gamma\lambda)^m$ if this state occurred m time steps in the past. With this algorithm, we can update the value function in each time step, without relying on first generating a sequence of states, actions and rewards. For a more detailed description of TD(λ) we refer to Chapter 12 of Sutton [15]. The convergence of the above described algorithm was proven for general λ by Dayan [5].

The question that remains is why bother using the TD(λ) method over the previously described methods. A first reason is that TD(λ) with the backward view generalizes both TD(0), Monte Carlo and everything in between by varying λ. In this way we can choose small values of λ to bootstrap more or let λ be larger to take future rewards more into account. Bootstrapping leads to more bias, since we use previous estimates which might be wrong. Meanwhile, taking into account more rewards leads to more variance, so by varying λ we also modulate the bias-variance trade-off. Furthermore, the TD(λ) algorithm provides a way to use Monte Carlo methods in non-episodic tasks by simply letting $\lambda = 1$.

Compared to n-step methods, TD(λ) also has some advantages. A clear computational advantage of TD(λ) is that we only need to keep track of one eligibility trace, with which we can update every state using the temporal difference δ_t of the current iteration. In the n-step bootstrap we can also use a backward view instead of the forward view by updating the past n states. However, we can't store all past information in a single eligibility trace and instead, we need to keep track of the past n feature vectors to perform updates. Another possibility is to simply keep track of the past n rewards and update the value function with these, but then all our updates are delayed by n steps. Another advantage is that we average over all n-step returns. The exponential average both allows us to easily update the eligibility trace and gives more weight to recent returns instead of just looking at a single return sequence.

3.2 State-action values and Q-learning

Temporal difference methods attempt to estimate V^π, the value function for a given policy π. This can be viewed as performing the policy evaluation step in policy iteration. To actually find an optimal policy π^*, we need to also improve our policy over time. For this purpose, it is often useful to consider the state action values $Q(s, a)$. Now we transition from state action pair to state action pair instead of from state to state. Given a state-action value function $Q(s, a)$ we can easily find the best action in each state by choosing the one which maximizes the value function. This is not possible for a state value function, were we need knowledge of the model to find optimal actions. As its name suggests, Q-learning uses these state-action values to find optimal actions. We will also describe a method similar to Q-learning called SARSA. In this chapter, we again consider a finite MDP with state-action values initialized in the look-up table $Q_0(s, a)$.

3.2.1 Q-learning

The Q-learning algorithm was first introduced by Watkins [23]. Q-learning is similar to TD(0), but has a few key differences. The use of state-action values was already mentioned. Furthermore, Q-learning uses a *behavioural policy*, denoted by π_b, to generate an episode of states and actions $(s_0, a_0, s_1, a_1, \ldots)$. This behavioural policy determines which state-action pairs are visited in the algorithm. At time t the value function of state-action pair (s_t, a_t) is updated. The update rule is given in Equation (3.8).

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t \big[r(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \big] \tag{3.8}$$

This equation is very similar to Equation (3.2), the update rule of TD(0). The difference lies in the target of the update. In TD(0) we used the value of the next state s_{t+1} of the episode in our target. In Q-learning we update with $\max_{a \in \mathcal{A}} Q_t(s_{t+1}, a)$, which still uses next state s_{t+1} but chooses the action which maximizes the current value function. Another way to look at this is to say that the action with which we update is determined by the *target policy* π_t. This target policy is then defined as

$$\pi_t(a|s) = 1 \quad \text{iff } a = \arg\max_{b \in \mathcal{A}} Q_t(s, b). \tag{3.9}$$

This target policy depends on the current value function and thus changes with time. The Q-learning algorithm is given in Algorithm 5. In this algorithm, we don't generate the episode beforehand and instead sample the states and actions in every time step from transition function P and the behavioural policy π_b respectively. Q-learning is an off-policy method, because the behavioural policy that determines the action which we take differs from the target policy that determines the actions with which we update. In Section 3.2.3 we will describe details and implications of on-policy and off-policy methods. By using the target policy which includes the maximization, Q-learning mimics the Bellman optimality operator from Equation (2.8). As such, Q-learning can be viewed as a sampled version of value iteration, since we also iterate the value function.

Algorithm 5: Q-learning

Input: A behavioural policy π_b, a number of iterations t_{iter} and initial state s_0.
Initialize the value function $Q(s, a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ and set $t = 0$.
while $t < t_{iter}$ **do**
  Sample action a_t from behavioural policy $\pi_b(\cdot|s_t)$.
  Observe reward $r(s_t, a_t)$.
  Sample next state $s_{t+1} \sim P(\cdot|s_t, a_t)$.
  Update the value function of state-action pair (s_t, a_t) according to Equation (3.8).
  Increment t by 1.
end

While Q-learning is directly related to TD(0), which is a special case of TD(λ), it can be extended to also use eligibility traces. The resulting method combines Q-learning and TD(λ) and it is called Q(λ). This method was introduced by Watkins [23]. This would also be an interesting method to look at, but we will focus on Q-learning in this paper.

3.2.2 SARSA

Another reinforcement learning method that uses state-action values is SARSA. This method is very similar to Q-learning as we again generate an episode of states and actions from a behavioural policy. The only difference is that we update with $Q_t(s_{t+1}, a_{t+1})$, the value function of the next state-action pair that is visited by our behavioural policy. Thus, the target policy that chooses the actions in the update is the same from the behavioural policy that determines the actions that we take in the algorithm. Thus, the target policy equals the behavioural policy, making SARSA an on-policy method. The update rule of SARSA is given in the equation below.

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t \big[r(s_t, a_t) + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \big]. \tag{3.10}$$

We can again use Algorithm 5 to perform SARSA by only changing the update rule. SARSA stands for state-action-reward-state-action. This is because we look at the 5-tuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$. The interpretation of this is as follows: when we are in state s_t, we take an action a_t according to some behavioural policy π_b and observe reward $r(s_t, a_t)$ and state s_{t+1}. In this state, we again take an action a_{t+1} according to our behavioural policy.

For SARSA to actually converge, we have to involve some manner of policy improvement. To do this, we let our behavioural policy π_b be a non-stationary policy, which satisfies the following properties:

1. each action is executed infinitely often in every state that is visited infinitely often, and
2. in the limit, the learning policy is greedy with respect to the Q-value function with probability 1.

Policies that satisfy these properties are labeled as *GLIE*, which stands for "greedy in the limit with infinite exploration". Singh [14] shows that, under some basic assumptions, Q_t converges to Q^* and π_b converges to π^* when applying SARSA with GLIE policies. The proof is an extension of the convergence proof of Q-learning, which we will give in Section 3.3. To given an example of a GLIE policy, let's consider an ϵ-greedy learning policy which chooses in state s a random action with probability $\epsilon_t(s) = c/n_t(s)$ and the greedy action with respect to the current value function Q_t with probability $1 - \epsilon_t(s)$. Here we require that $0 < c < 1$ and $n_t(s)$ is defined as the amount of visits to state s at time t. The paper by Singh shows that this policy is GLIE and thus will converge to the optimal policy when applying SARSA.

3.2.3 On-policy versus off-policy methods

We have seen that Q-learning is off-policy method, while SARSA is an on-policy method. The difference is that on-policy methods use the same policy for exploring the state space and for choosing the action which determines the value function with which we update. In off-policy methods, these actions can be different from each other. The advantage of Q-learning is that by choosing maximizing actions in the update, we can immediately estimate the optimal state-action value. Furthermore, we don't need extra requirements on our behavioural policy, while we require a GLIE policy for SARSA. However, off-policy methods can become unstable and even fail to converge when function approximation is introduced, as we will see in the next chapter. This problem doesn't occur in on-policy methods as SARSA.

3.3 Convergence proof of Q-learning

The convergence of Q-learning has been well studied in the literature. Around the same time both Tsitsiklis [18] and Jaakkola [6] established proofs that show that Q-learning converges to the optimal value function Q^*. Another proof for the convergence of Q-learning is given by Melo [10]. This Melo proof came a bit later and is mostly based on the proof from the Jaakkola paper, but it provides some more details. In this section we will give our own proof of the convergence of Q-learning. Our proof is mostly based on the proof by Melo, but it differs from each of the other three proofs in multiple ways. First of all, our proof doesn't rely on measure theory, which is the case for the proof by Tsitsiklis. Measure theory is more technical than the methods we use in our thesis, so it won't help us in our understanding of Q-learning. On the other hand, the proofs by Jaakkola and Melo lack important details and steps in the proof, making them unclear and a bit incomplete. By formalizing our own proof, we can make the proof explicit and precise in every step, while not having to rely on measure theory. This will help us in the understanding of Q-learning and in the contextualization of our results.

Before we move on the proof of convergence, we will first make a few assumptions on our Markov decision process. First of all, we will assume that our MDP is finite, which means that it has a finite state and action space. This is needed because the Q-values are stored in a look-up table and we want to visit every state-action pair infinitely many times. Furthermore, we let π_b be the behavioural policy that generates a sequence of states, actions and rewards. Given initial state s_0 and probability function P, we thus obtain $(s_0, a_0, s_1, a_1, \ldots)$ as the state-action sequence and $r(s_t, a_t)$ as the reward at time t. We will also require that the Bellman operator of Equation (2.8) is a contraction in the sup-norm, where the sup-norm is defined as $||\cdot||_\infty = \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} |\cdot|$.

Lemma 3.1. *The Bellman operator $\mathcal{B}$ defined in Equation (2.8) is a contraction in the sup-norm with contraction factor γ.*

Proof. Consider any two functions Q_1 and Q_2 that satisfy $Q_1, Q_2 : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$. Then, by Equation (2.8) we have:

$$
\begin{aligned}
||\mathcal{B}Q_1 - \mathcal{B}Q_2||_\infty &= && (3.11)\\
&= \max_{s,a} \big| r(s,a) + \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}[\max_{a'\in A} Q_1(s',a')] - r(s,a) - \gamma \mathbb{E}_{s'\sim P(\cdot|s,a)}[\max_{a'\in A} Q_2(s',a')] \big| && (3.12)\\
&= \max_{s,a} \gamma \big| \mathbb{E}_{s'\sim P(\cdot|s,a)}[\max_{a'\in A} Q_1(s',a')] - \mathbb{E}_{s'\sim P(\cdot|s,a)}[\max_{a'\in A} Q_2(s',a')] \big| && (3.13)\\
&= \max_{s,a} \gamma \big| \sum_{s'\in\mathcal{S}} P(s'|s,a)[\max_{a'\in A} Q_1(s',a') - \max_{a'\in A} Q_2(s',a')] \big| && (3.14)
\end{aligned}
$$

$$\leq \max_{s,a} \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \big| \max_{a' \in A} Q_1(s',a') - \max_{a' \in A} Q_2(s',a') \big| \tag{3.15}$$

$$\leq \max_{s,a} \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{x,b} |Q_1(x,b) - Q_2(x,b)| \tag{3.16}$$

$$= \max_{s,a} \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) ||Q_1 - Q_2||_\infty = \gamma ||Q_1 - Q_2||_\infty \tag{3.17}$$

The first inequality follows due to the triangle inequality. We will briefly explain the second inequality. We need to show that $|\max_{b \in \mathcal{A}} Q_1(y,b) - \max_{b \in \mathcal{A}} Q_2(y,b)| \leq \max_{x,b} |Q_1(x,b) - Q_2(x,b)|$ for all $y \in \mathcal{S}$. Now fix any $y \in \mathcal{S}$ and assume without loss of generality that $\max_{b \in \mathcal{A}} Q_1(y,b) \geq \max_{b \in \mathcal{A}} Q_2(y,b)$. Otherwise we can simply swap Q_1 and Q_2, which will not change the absolute value. Then we have that

$$\begin{aligned}
|\max_{b \in \mathcal{A}} Q_1(y,b) - \max_{b \in \mathcal{A}} Q_2(y,b)| &= \max_{b \in \mathcal{A}} Q_1(y,b) - \max_{b \in \mathcal{A}} Q_2(y,b) \\
&= Q_1(y,b^*) - \max_{b \in \mathcal{A}} Q_2(y,b) \\
&\leq Q_1(y,b^*) - Q_2(y,b^*) \\
&= |Q_1(y,b^*) - Q_2(y,b^*)| \\
&\leq \max_{b \in \mathcal{A}} |Q_1(y,b) - Q_2(y,b)| \\
&\leq \max_{x,b} |Q_1(x,b) - Q_2(x,b)|.
\end{aligned}$$

Here we define $b^* \in \arg\max_b Q_1(y,b)$. This holds for any $y \in \mathcal{S}$ so we have proven that the second inequality holds. As a result, the Bellman operator is a contraction in the sup-norm with contraction factor γ. □

To show convergence to the optimal value function we first need to show that this Q^* exists. The existence of Q^* easily follows from the Banach fixed-point theorem, since the Bellman operator is a contraction and the state and action spaces are finite. Intuitively we have the following for any approximation $q_t = \mathcal{B}q_{t-1}$ and an initial approximation q_0.

$$||q_t - Q^*||_\infty \leq \gamma ||\mathcal{B}q_{t-1} - \mathcal{B}Q^*||_\infty = \gamma ||\mathcal{B}q_{t-1} - Q^*||_\infty \leq \ldots \leq \gamma^t ||q_0 - Q^*||_\infty \tag{3.18}$$

As $\gamma < 1$ we have that $||q_t - Q^*||_\infty$ converges to zero by repeatedly applying the Bellman operator. However, in Q-learning we don't apply the Bellman operator in each iteration and instead use a sampled episode. To show convergence of Q-learning, we need a result from stochastic approximation theory. This result is given in the paper by Jaakkola, Jordan and Singh [6] and we will briefly elaborate on it. A similar result is given in the paper by Tsitsiklis [18]. First, we will need some terminology. Let Δ_t and F_t be stochastic processes where the update rule of Δ_t satisfies

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x). \tag{3.19}$$

Here $\alpha_t(x)$ is the learning rate which can depend on the state x. The random variables Δ_t, F_t and α_t are defined on some probability space $(\Omega, \mathcal{F}, \mathcal{P})$. In the algorithm, we will consider increasing subfields $\{\mathcal{F}_t\}_{t=0}^\infty$ of $\mathcal{F}$ where each $\mathcal{F}_t$ includes the previous subfield $\mathcal{F}_{t-1}$. Intuitively, one can think of these subfields as the history of the algorithm up to time t. In the algorithm we will assume that at time t the variables Δ_t and α_t are $\mathcal{F}_t$-measurable, while F_t is $\mathcal{F}_{t+1}$-measurable. What this means is that at time t we know the current iterate Δ_t and learning rate α_t, while F_t is the new incoming information. The convergence of Δ_t then depends on 3 assumptions on α_t and F_t.

1. For every state x and time t, the learning rate $\alpha_t(x)$ satisfies: $\sum_t \alpha_t(x) = \infty$, $\sum_t \alpha_t(x)^2 < \infty$ and $0 \leq \alpha_t(x) \leq 1$.
2. For every x and t, we have that $||E[F_t(x)|\mathcal{F}_t]||_\infty \leq \gamma ||\Delta_t||_\infty$ with $\gamma \in (0,1)$.
3. For every x and t, F_t satisfies $\text{Var}(F_t(x)|\mathcal{F}_t) \leq C(1 + ||\Delta_t||_\infty^2)$, for $C > 0$.

Theorem 3.2. *Let Assumptions 1, 2 and 3 hold. Then the stochastic process Δ_t given in Equation (3.19). converges to zero with probability 1.*

Proof. The proof of this theorem is given in [6]. □

Theorem 3.3. *Suppose that our MDP given by tuple $(\mathcal{S}, \mathcal{A}, P, r)$ is finite. This means that the spaces $\mathcal{S}$ and $\mathcal{A}$ are finite. Assume that the learning rates satisfy*

$$\sum_t \alpha_t(s,a) = \infty \qquad \sum_t \alpha_t(s,a)^2 < \infty, \tag{3.20}$$

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$*. Assume also that we have* $r(s,a) \geq 0$ *and* $Q_0(s,a) \geq 0$ *for all* s, a*. As a result,* $Q_t(s,a) \geq 0$ *for all* s, a *and* t*. Then, the update rule given by Equation* (3.8) *converges to the optimal Q-function with probability 1.*

Note that the assumption of non-negative rewards and initialization can be satisfied for any MDP by simply shifting the rewards $r(s,a)$ and initial values $Q_0(s,a)$ by some positive value such that they are all positive. This will yield an equivalent MDP and leads to no changes in the algorithm. The other condition in Equation (3.20) allows the learning rates to depend on the state and actions. It requires that each state-action pair (s,a) is visited an infinite number of times, since $0 \leq \alpha_t(s,a) \leq 1$. This also implies that our policy should have $\pi_b(a|s) > 0$ for all pairs (s,a). As a result, we require our policy to explore.

Proof. To prove this theorem, we use Theorem 3.2 to prove convergence of Q_t. We observe that condition 1 of Theorem 3.2 is immediately satisfied by (3.20). To use the theorem, we reformulate our update rule of Equation (3.8) and subtract $Q^*(s,a)$ from both sides.

$$\begin{aligned}
Q_{t+1}(s,a) - Q^*(s,a) &= Q_t(s_t,a_t) + \alpha_t(s_t,a_t)\big[r(s_t,a_t) + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1},a) - Q_t(s_t,a_t)\big] - Q^*(s,a) \\
&= (1 - \alpha_t(s_t,a_t))Q_t(s_t,a_t) + \alpha_t(s_t,a_t)\big[r(s_t,a_t) + \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1},a)\big] - Q^*(s_t,a_t) \\
&= (1 - \alpha_t(s_t,a_t))(Q_t(s_t,a_t) - Q^*(s,a)) + \alpha_t(s_t,a_t)\big[r(s_t,a_t) \\
&+ \gamma \max_{a \in \mathcal{A}} Q_t(s_{t+1},a) - Q^*(s_t,a_t)\big]
\end{aligned}$$

Note that Q^* is the solution of $\mathcal{B}Q = Q$ and is thus fixed. It can simply be seen as some vector in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. Now by letting

$$\Delta_t(s,a) := Q_t(s,a) - Q^*(s,a)$$

and

$$F_t(s,a) := r(s,a) + \gamma \max_{b \in \mathcal{A}} Q_t(s',b) - Q^*(s,a) \tag{3.21}$$

we obtain

$$\Delta_{t+1}(s,a) = (1 - \alpha_t(s,a))(\Delta_t(s,a)) + \alpha_t(s,a)F_t(s,a). \tag{3.22}$$

Now we want to prove that conditions 2 and 3 of theorem 3.2 hold for (3.21). To do this, we will first need to know how the history $\mathcal{F}_t$ is defined for the Q-learning update rule. At time t we know the previous learning rates. These won't be used further in the proof. Furthermore, $\mathcal{F}_t$ contains the current Δ_t and all its previous iterations. Since Δ_t is just a shifted Q_t, they are both known and contain no further randomness. All randomness at time t is concentrated in $F_t(s,a)$, since this quantity isn't part of $\mathcal{F}_t$. Only the previous iterations $F_{t-1}, F_{t-2}, \ldots$ are known.

There are two possible sources of randomness in $F_t(s,a)$. First of all, the reward can be random. While we have implicitly assumed in the Equation (2.8) of the Bellman operator that $r(s,a)$ is deterministic, we still might not know this function r and have not observed the reward yet. The other source of stochasticity lies in the state transition to the next state s'. Keeping these things in mind we get for the conditional expectation of $F_t(s,a)$

$$\begin{aligned}
\mathbb{E}[F_t(s,a)|\mathcal{F}_t] &= \mathbb{E}[r(s,a) + \gamma \max_{b \in \mathcal{A}} Q_t(s',b) - Q^*(s,a)|\mathcal{F}_t] \\
&= \mathbb{E}[r(s,a) + \gamma \max_{b \in \mathcal{A}} Q_t(s',b)|\mathcal{F}_t] - Q^*(s,a) \\
&= \mathcal{B}Q_t(s,a) - Q^*(s,a) = \mathcal{B}Q_t(s,a) - \mathcal{B}Q^*(s,a).
\end{aligned} \tag{3.23}$$

Here we used the definition of the Bellman operator of Equation (2.8) and the fact that $\mathcal{B}Q^*(s,a) = Q^*(s,a)$. Since, the Bellman operator is a contraction, we then obtain that:

$$\begin{aligned}
||\mathbb{E}[F_t(s,a)|\mathcal{F}_t||_\infty &= \max_{s,a} |\mathcal{B}Q_t(s,a) - \mathcal{B}Q^*(s,a)| \\
&= ||\mathcal{B}Q_t - \mathcal{B}Q^*||_\infty \\
&\leq \gamma||Q_t - Q^*||\infty = \gamma||\Delta_t||_\infty.
\end{aligned}$$

This shows that condition 2 holds. For condition 3 we get that the variance, using the first part in (3.23), is given by

$$\begin{aligned}
\mathrm{Var}(F_t(s,a)|\mathcal{F}_t) &= \mathbb{E}\big[(F_t(s,a) - \mathbb{E}[F_t(s,a)|\mathcal{F}_t])^2|\mathcal{F}_t\big] \\
&= \mathbb{E}\big[(r(s,a) + \gamma \max_{b \in \mathcal{A}} Q_t(s',b) - Q^*(s,a) - \mathcal{B}Q_t(s,a) + Q^*(s,a))^2\big|\mathcal{F}_t\big]
\end{aligned}$$

$$
\begin{aligned}
&= \mathbb{E}\big[(r(s,a) + \gamma \max_{b\in\mathcal{A}} Q_t(s',b) - \mathcal{B}Q_t(s,a))^2 \big| \mathcal{F}_t\big] \\
&= \mathbb{E}\big[(r(s,a) + \gamma \max_{b\in\mathcal{A}} Q_t(s',b) - r(s,a) - \gamma \mathbb{E}_{x\sim P}[\max_{b\in\mathcal{A}} Q_t(x,b)])^2 \big| \mathcal{F}_t\big] \\
&= \mathbb{E}\big[(r(s,a) + \gamma \max_{b\in\mathcal{A}} Q_t(s',b) - \mathbb{E}_{x\sim P}[r(s,a) + \gamma \max_{b\in\mathcal{A}} Q_t(x,b)])^2 \big| \mathcal{F}_t\big] \\
&= \mathrm{Var}(r(s,a) + \gamma \max_{b\in\mathcal{A}} Q_t(s',b) | \mathcal{F}_t) \\
&= \mathrm{Var}(r(s,a)|\mathcal{F}_t) + \gamma^2 \mathrm{Var}(\max_{b\in\mathcal{A}} Q_t(s',b)|\mathcal{F}_t) + 2\gamma \mathrm{Cov}(r(s,a), \max_{b\in\mathcal{A}} Q_t(s',b)|\mathcal{F}_t)
\end{aligned}
$$

So we are left with three terms: the variance of our reward, the variance of the maximum of the Q-values at time t over all actions and the covariance between the two. We have assumed bounded rewards, thus we can bound the variance of the reward by some scalar M. The randomness in $\max_{b\in\mathcal{A}} Q_t(s',b)|\mathcal{F}_t$ is entirely due to the next state s', so the covariance will vanish if $r(s,a)$ is independent of this state. Even if the rewards depend on s', Tsitsiklis [18] mentions that a similar argument follows with a slightly different bound. Here we will assume that the rewards are independent of the next state such that this term vanishes. Then we are left with

$$
\begin{aligned}
\mathrm{Var}(F_t(s,a)|\mathcal{F}_t) &= \mathrm{Var}(r(s,a)|\mathcal{F}_t) + \gamma^2 \mathrm{Var}(\max_{b\in\mathcal{A}} Q_t(s',b)|\mathcal{F}_t) + \gamma \mathrm{Cov}(r(s,a), \max_{b\in\mathcal{A}} Q_t(s',b)|\mathcal{F}_t) \\
&\leq M + \gamma^2 \mathrm{Var}(\max_{b\in\mathcal{A}} Q_t(s',b)|\mathcal{F}_t) \\
&= M + \gamma^2 \mathbb{E}[(\max_{b\in\mathcal{A}} Q_t(s',b))^2|\mathcal{F}_t] - \gamma^2 (\mathbb{E}[\max_{b\in\mathcal{A}} Q_t(y,b)|\mathcal{F}_t])^2 \\
&\leq M + \gamma^2 \mathbb{E}[(\max_{b\in\mathcal{A}} Q_t(s',b))^2|\mathcal{F}_t] \\
&\leq M + \gamma^2 (\max_{s'\in\mathcal{S}} \max_{b\in\mathcal{A}} Q_t(s',b))^2 \\
&= M + \gamma^2 ||Q_t||_\infty^2 \\
&= M + \gamma^2 ||\Delta_t + Q^*||_\infty^2 \\
&\leq M + \gamma^2 ||Q^*||_\infty^2 + \gamma^2 ||\Delta_t||_\infty^2 \leq C(1 + ||\Delta_t||_\infty^2)
\end{aligned}
$$

Where we let $C = \max\{M + \gamma^2 ||Q^*||, \gamma^2\}$. Here the first inequality follows from the bounding on the variance and covariance terms described in the previous paragraph. The second inequality trivially follows because $\gamma^2(\mathbb{E}[\max_{b\in\mathcal{A}} Q_t(y,b)|\mathcal{F}_t])^2$ is always positive. The third inequality follows since we can bound $(\max_{b\in\mathcal{A}} Q_t(s',b))^2$ by its largest possible value, which equals $(\max_{s'\in\mathcal{S}} \max_{b\in\mathcal{A}} Q_t(s',b))^2$ because $Q_t(s,a) \geq 0$ for all s, a. Finally we use the triangle inequality to obtain $||\Delta_t||_\infty^2$ which leads to the bound.

As a result, all three properties of theorem 3.2 are proven to hold, such that Δ_t converges to zero with probability 1. Thus, $Q_t(s,a)$ converges to $Q^*(s,a)$ with probability 1. □

Contribution 3.1. In this section we have given our own proof of the convergence of Q-learning, adapted from earlier proofs. Our convergence proof doesn't rely on measure theory, like Tsitsiklis [18], and is more precise than Jaakkola [6] and Melo [10]. With this detailed proof we can better understand the method of Q-learning, which helps us in developing our own results.

Chapter 4

Value function approximation

In the previous chapter, we saw that we could achieve convergence for Q-learning under certain assumptions on the step size. In the standard Q-learning algorithm every state-action value gets updated individually. As such, the state-action values can be viewed as vectors in $\mathbb{R}^{|S||A|}$. In many applications we have to deal with large state and action spaces, meaning it can take long to converge since we require that each state or state-action pair is visited infinitely many times in the limit. Also, if the state or action spaces are continuous, this task becomes impossible. In this chapter, we look into value function approximation, where we use a set of parameters θ to approximate the value function. First, we examine difficulties in showing convergence when function approximation is used via the deadly triad. Then we will show how to modify the problem formulation to deal with the introduction of these parameters. Furthermore, we will look into multiple function approximators for both temporal difference learning and Q-learning. Finally, it is examined what conditions and assumptions are needed to obtain convergence in this cases.

4.1 The deadly triad

One might hope that when function approximation is used, we can obtain convergence in a similar manner as we saw in Section 3.3. Unfortunately, Baird [1] showed that such algorithms can diverge even for general functions with nice properties. One might wonder why we have such poor performance for methods using function approximation. It turns out that it has to do with the combination of three properties that come into play when applying these methods. Sutton [15] identified these properties as the deadly triad.

1. **Function approximation**: In function approximation we approximate our state values or state action values with some function that is parametrized by θ. This is needed in the case that state and action spaces grow large and regular methods that rely on look-up tables become infeasible.
2. **Bootstrapping**: With bootstrapping we use previous estimates of the value functions to update our new estimate. An advantage is that we don't have to rely on a lot of simulated rewards and can instead use information from other estimates. However, when the previous estimates are far off the optimal values, the updated states inherit these errors.
3. **Off-policy learning**: Finally we have that off-policy methods add to the instability. The reason for this is that we might not visit states with which we update our value functions. In SARSA we might still use a bad estimate $Q_t(s_{t+1}, a_{t+1})$ of the next state-action pair (s_{t+1}, a_{t+1}), but we at least visit this state-action pair next since SARSA is an on-policy method. Subsequently, the approximation for this state-action pair is updated. In off-policy methods, we don't guarantee that we visit this state-action pair. As a result, errors of the value function can remain and influence future updates.

These factors combined can form a deadly combination, which leads to instability and divergence. However, we do know that methods that encounter the deadly triad can perform well empircally. For example, Mnih [12] used a neural network as an approximation for the state-action value function to learn how to play Atari games. This paper uses a few extra tricks in the algorithm, such as a replay buffer and a target network. For more details on these we refer to the original paper. A recent paper by van Hasselt [21] investigates the influence of the deadly triad in Deep Q-learning by experimental examination of each of the 3 factors. It focuses on a few ways to modulate the influence of each of these 3 factors. Specifically, it also a analyzes the mentioned tricks used in Mnih [12].

- The influence of function approximation is modified by changing the capacity of the function approximation. This is done by using networks of differing sizes. The hypothesis is that larger, more flexible

networks diverge less easily.

- The influence of bootstrapping is modulated by using multi-step returns, as we have described in section 3.1.2. Furthermore, a target network is used on which we bootstrap. This target network has a similar structure to the main network, but it only updates sporadically using the parameters from the main network. Both these methods are hypothesized to reduce the instability.
- Finally, the amount of off-policy updates can be modulated by using experience replay. From the replay buffer we can elect to prioritize certain samples over others. The authors theorize that using stronger prioritization causes more divergence.

Experimentally it was observed that these approaches lead to more stability and less divergence, except for larger networks. In this last case, it was observed that smaller networks performed better in the examples that were examined.

4.1.1 Baird's Counterexample

To illustrate the influence of the deadly triad, we will describe an example in which function approximation with bootstrapping and off-policy learning leads to divergence of the value function. This counterexample was introduced by Baird [1] and shows that we can have divergence even in very simple Markov decision processes. Consider the seven-state Markov Decision process of figure 4.1. In this MDP each state has two actions, the action with a solid line and the one with a dashed line. The solid action always has state 7 as next state, while the dashed option results in any of the other 6 states, each with equal probability. The approximate state-value function of this MDP is shown inside each state and is determined by the parameter vector $w = (w_1, \dots, w_8)^T$. We will use w instead of θ to follow the notation in the figures. This can be seen as linear function approximation with for example feature vector $\phi(1) = (2, 0, 0, 0, 0, 0, 0, 1)^T$ for state 1 such that $V(s_1) = w^T\phi(s_1) = 2w_1 + w_8$. Finally, the rewards for each transition are 0. As a result, the optimal parameter vector should be the zero vector. We do off-policy updates, meaning that we distinguish between

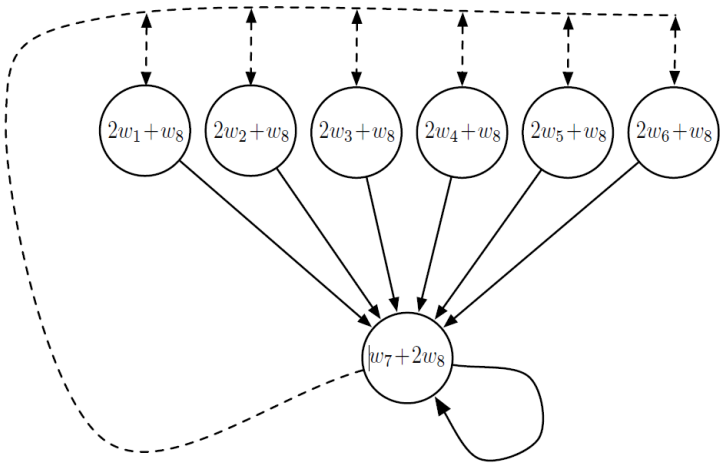


Figure 4.1: Baird's counterexample from the book by Sutton [15]. We have 6 states with solid and dashed actions. The value functions are approximated using linear function approximation. The target policy always chooses the solid actions, while the behavioural policy chooses the solid action with probability 1/7 and the dashed action with probability 6/7.

the behavioural policy and the target policy. The behavioural policy determines the actual transitions and thus which states we visit. In this example, we choose the dashed action with probability 6/7 and the solid action with probability 1/7. The result of this behavioural policy is that the next-state distribution is uniform, so each state is visited equally as often on average. The target policy determines the bootstrapped update target. We let the target policy always choose the solid action, which leads to state 7.

Having described the problem, we want to show how we can have divergence and in which way each element of the deadly triad plays a part. First we need to define how we update the weight vector w in each time step. The update is done using a direct algorithm, which we will describe in the next section. The direct algorithm is very similar to a standard temporal difference update, now only including a gradient with respect to the value function of the current state. For this specific MDP the weigh update is given below.

$$w_{t+1} = w_t + \alpha_t(\gamma V(s_7) - V(s))\nabla_w V(s) = w_t + \alpha_t(\gamma V(s_7) - V(s))\phi(s) \tag{4.1}$$

Here we use that we have zero rewards and that the target policy always chooses state 7. We will initialize the parameter vector as $(1, 1, 1, 1, 1, 1, 10, 1)^T$. As a result we have that initially $V(s_1) = \ldots = V(s_6) = 3$ and $V(s_7) = 12$. Furthermore, we let $\gamma = 0.99$ and $\alpha_t = 0.01$ for all t. Now let's do an update.

If we are in any of the first 6 states we will have a positive difference $\gamma V(s_7) - V(s)$. As a result, w_8 and any of the first 6 parameters are increased. Note that the value of state 7 thus also increases. If we are in state 7, $\gamma V(s_7) - V(s)$ is slightly negative and we decrease the values of w_7 and w_8. However, on average w_8 will keep increasing, since in 6 out of 7 updates we increase the value of w_8, while only decreasing it in 1 out of 7 updates. Because $V(s_7)$ keeps increasing with w_8, $\gamma V(s_7) - V(s)$ will also remain positive for each of the first 6 states. In figure 4.2 taken from the book by Sutton [15] the divergence of the parameters is shown. It also shows that we still have divergence even if we do the average update over all states instead of the simulated updates.

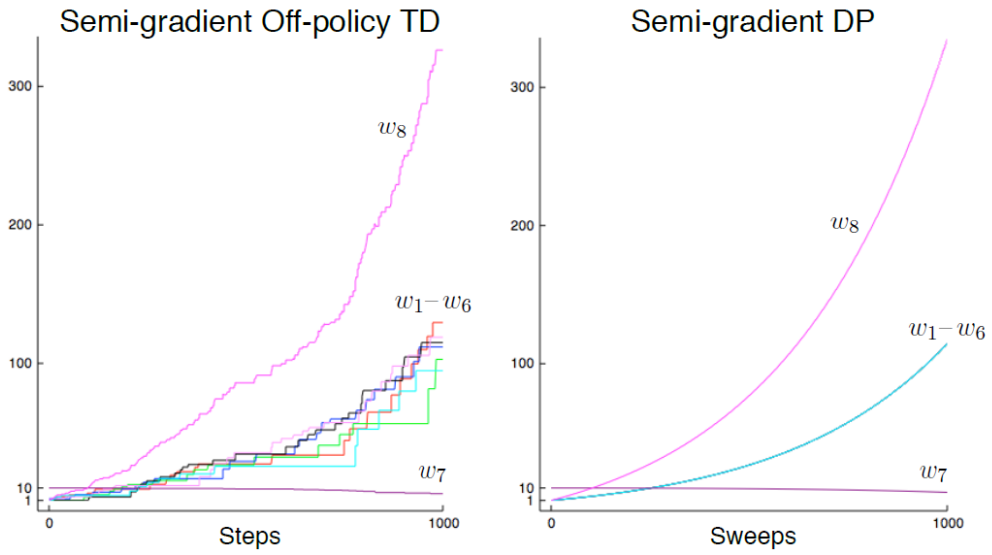


Figure 4.2: Demonstration of instability on Baird's counterexample. Shown are the evolution of the components of the parameter vector w of the two semi-gradient algorithms. The step size was $\alpha = 0.01$, and the initial parameters were $w = (1, 1, 1, 1, 1, 1, 10, 1)^T$.

The influence of function approximation in divergence is very clear from this. Since all the states share parameter w_8, updates to one state also affect all other states. Since the value of w_8 increases in 6 out of 7 updates, $V(s_7)$ keeps increasing as well. The values of states 1 through 6 don't reach the value of state 7, since state 7 depends more on w_8 as the other states and parameters 1 through 6 don't get updated as often as w_8. If we were to remove w_8 we are left with a lookup table, because each state simply has a single parameter. In this case we might see an initial increase of the first 6 parameters, but once the values $V(s_1), \ldots, V(s_6)$ approximately equal $V(s_7)$, the values will decrease due to the discount factor γ. From then, the parameters will converge to the optimal value of zero.

The influence of off-policy learning is also very clear. In each iteration we update with the very wrong value of state 7, while we only visit this state every 7 iterations on average. This leads to temporal differences being positive for updates to the first 6 states, which in turn leads to divergence. If the target policy would equal the behavioural policy, we would also update with the values of states 1 through 6. These values are lower, thus we will often update with negative temporal differences and the parameters will converge to zero.

Finally, bootstrapping also plays a part. If we don't use bootstrapping (or use very little bootstrapping) and instead rely on a long chain of rewards then we would obtain negative temporal differences since all the rewards are zero. So also if we don't use bootstrapping we would have convergence for this example. It is clear that each of these 3 elements contribute to the divergence in certain MDP's, but that we would have convergence if only 2 elements are present. However, these 3 elements are present in many algorithms. To still obtain convergence, we have to be very careful about defining our assumptions and setting up the optimization problem.

4.2 Optimization problem

To properly analyze reinforcement learning methods that use function approximation, we have to properly define the objective which we wish to optimize. In this section, we will show different choices of objective and how they lead to variations in algorithms. This additional structure will also be helpful when to see if we can still converge if all elements of the deadly triad are present.

Let K be the dimension of the parameter vector. Then, we want to find the set of parameters $\theta \in \mathbb{R}^K$ for which V_θ most closely approximates V^* (or V^π). The most obvious way to do this is to minimize $||V_\theta - V^*||_2^2$ over θ. The L_2-norm is often used, because calculation of the gradient is straightforward. Often we also take into account the dynamics of the Markov process. Thus, instead of weighting each state equally, we will minimize the error with respect to some state distribution μ. This leads to minimizing the *mean squared value error* (MSVE) over θ.

$$\text{MSVE}(\theta) = E_{s\sim\mu}[(V^*(s) - V_\theta(s))^2] = \sum_{s\in\mathcal{S}} \mu(s)(V^*(s) - V_\theta(s))^2 = (V^* - V_\theta)^T D(V^* - V_\theta) \tag{4.2}$$

Here we define $D = \text{diag}(\mu)$, where μ is often chosen as the steady-state distribution of states. The reason for including this distribution is that when we do online updates, the amount of updates per state will follow some distribution depending on the MDP. Not all states are updated as often so this is reflected in the error. To find the optimal set of parameters θ, we can take the gradient of this expression and perform gradient descent. The gradient of the MSVE is given by

$$\nabla_\theta \text{MSVE}(\theta) = -2(\nabla_\theta V_\theta)^T D(V^* - V_\theta), \tag{4.3}$$

$$\theta_{t+1} = \theta_t + 2\alpha(\nabla_\theta V_\theta)^T D(V^* - V_\theta). \tag{4.4}$$

Note that $\nabla_\theta V_\theta$ is the $|S| \times K$ Jacobi-matrix, since we can view $V_\theta : \mathbb{R}^K \to \mathbb{R}^{|S|}$ as a vector-valued function such that $(\nabla_\theta V_\theta)_{ij} = \frac{\partial}{\partial\theta_j} V_\theta(s_i)$. The factor of 2 is often discard and can simply be incorporated in the learning rate α. Taking the full gradient is often not practical, since it requires knowledge of μ. It also doesn't translate well to the online setting, where we observe transitions between individual states. A better strategy would be to do *stochastic gradient descent* (SGD) to updates the parameters. In SGD we calculate the error for a single state s and update the parameter vector. This leads to the following update:

$$\theta_{t+1} = \theta_t + \alpha(V^*(s) - V_\theta(s))\nabla_\theta V_\theta(s). \tag{4.5}$$

While not obvious, the steady-state distribution μ is still present, since it determines how often we update each state. The problem about optimizing the MSVE is that it requires knowledge of the true value function V^*. Our goal is to find this V^*, so it is not be known beforehand. As a result, we will choose other objectives to minimize. Choosing the objective and optimization method is still an open topic and there is no best way to do it. Multiple objectives and methods have been proposed, each with different degrees of theoretical and experimental convergence.

The most common objective to minimize is the *mean squared Bellman error* (MSBE). In this error, we replace the true value function of the MSVE with a Bellman operator applied on the parametrized value function.

$$\text{MSBE}(\theta) = E_{s\sim\mu}[(\mathcal{B}V_\theta(s) - V_\theta(s))^2] \tag{4.6}$$

If V_θ equals our true value function V^*, then both the MSVE and MSBE will equal zero justifying the use of the Bellman operator. Other operators can be used in place of the Bellman operator. For example, one can consider the expected n-step return or the expected λ-return. We will see more of this later. A disadvantage of the MSBE is that it is not uniquely determined by data. Sutton and Barto [15] showed that two different MDPs inducing the same distribution of transitions and observed rewards can have different Bellman errors for the same parameter vector θ. The Bellman error can only be determined if we were to know the underlying dynamics of the system. This is a problem, as in most applications we only observe the data distribution.

Another objective that is often considered is the *mean squared projected Bellman error* (MSPBE).

$$\text{MSPBE}(\theta) = E_{s\sim\mu}[(\Pi_\mathcal{F}\mathcal{B}V_\theta(s) - V_\theta(s))^2] \tag{4.7}$$

The projection $\Pi_\mathcal{F}$ is done on the function space $\mathcal{F} = \{V_\theta | \theta \in \mathbb{R}^K\}$ of value function approximations. Sutton and Barto [15] showed that the MSPBE is learnable, meaning that it can be uniquely determined by the observed data distribution. It is also often used in the case of non-linear function approximation due to non-convexity of the objective. By smartly choosing the function space, convergence can be achieved in some cases. In a lot of articles convergence is shown in the MSPBE, even though it might not necessarily be used in the update rule. For example, Tsitsiklis and Van Roy [19] apply a direct algorithm for TD(λ) with linear function approximation, but they show convergence in the MSPBE.

4.2.1 Direct algorithms

The simplest extension of temporal difference methods are *direct algorithms*. The method used in Baird's counterexample is in fact a direct algorithm. These methods aren't always stable and can diverge, but they are simple and often work well in practice. To derive direct algorithms, we take the stochastic gradient update of Equation (4.5) and replace the true value function by the Bellman operator applied on V_θ. This leads to the following update rule:

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t(\mathcal{B}V_\theta(s) - V_\theta(s))\nabla_\theta V_\theta(s), \\ &= \theta_t + \alpha_t \mathbb{E}_{s'\sim P, a\sim\pi}[r(s,a) + \gamma V(s') + V(s)|s]\nabla_\theta V_\theta(s). \end{aligned} \tag{4.8}$$

Direct algorithms can be viewed as taking the semi-gradient with respect to the MSBE, where we only take the gradient with respect to V_θ and not with respect to $\mathcal{B}V_\theta$. In practical settings, we of course don't use the expectation in the gradient update and instead use a sampled next state s'. As a result, the gradient update contains temporal difference δ_t. Direct algorithms can easily be extended to λ-returns by using eligibility traces. We can also use direct algorithms when learning state-action values.

4.2.2 Residual algorithms

Residual algorithms were introduced by Baird [1] and they attempt to solve the issue of divergence when function approximation is used. Before considering residual algorithms, we will first describe *residual gradient algorithms*, which were also described by Baird [1]. Residual gradient algorithms take the full gradient with respect to the MSBE of Equation (4.6) instead of just the semi-gradient.

$$\theta_{t+1} = \theta_t + \alpha_t \mathbb{E}_{s'\sim P, a\sim\pi}[r(s,a) + \gamma V(s') - V(s)|s](\mathbb{E}_{s'\sim P}[\gamma\nabla_\theta V(s')|s] - \nabla_\theta V(s)) \tag{4.9}$$

Baird showed that residual gradient algorithms converge to a local minimum of the MSBE for finite MDP's with deterministic transitions. In this case, the expectations vanish in the above equation and we are simply left with the deterministic next state s'. A problem about residual gradient algorithms is their slow experimental convergence. As a result, Baird proposed residual algorithms. These take steps in the direction of a weighted average of the update vectors for the direct and residual gradient algorithms. This weighting is done such that we move as much as possible in the direction taken by the direct algorithm, while still maintaining convergence.

Unfortunately, both residual gradient algorithms and residual algorithms run into another problem, namely the double sample issue. Convergence was shown for deterministic MDP's, since then we don't have to calculate the expectation over the next state. When the next state is random we are left with a product of expectations. We can't simply fill in the next sampled state s' for both expectations, since this would introduce a bias. To get an unbiased estimate of the full gradient, we would have to sample two next states s'_1 and s'_2 from our current state s, one for the temporal difference and one for the gradient over the value function of the next state. Practically, this is inconvenient because in online sampling we only sample one next state.

4.2.3 Gradient TD

A more recent algorithm are *gradient TD* methods, introduced by Sutton et al. [16]. These methods update by applying stochastic gradient descent on the MSPBE. For the derivation of the gradient, we first need to rewrite the MSPBE. The paper by Sutton rewrites in the case of linear function approximation, but it can also be done for any continuously differentiable V_θ as done by Maei [9]. For the derivation we refer to these papers.

$$\text{MSPBE}(\theta) = \mathbb{E}_{s\sim\mu, s'\sim P}[\delta\nabla_\theta V_\theta(s)]^T \mathbb{E}_{s\sim\mu}[\nabla_\theta V_\theta(s)\nabla_\theta V_\theta(s)]^{-1}\mathbb{E}_{s\sim\mu, s'\sim P}[\delta\nabla_\theta V_\theta(s)]$$

Here we again define temporal difference $\delta = r(s,a) + \gamma V_\theta(s') - V_\theta(s)$. It might seem that we are stuck with the double sample issue, since we have a product of two expectations over the next state s'. However, gradient TD methods solve this by approximating $w \approx \mathbb{E}_{s\sim\mu}[\nabla_\theta V_\theta(s)\nabla_\theta V_\theta(s)]^{-1}\mathbb{E}_{s\sim\mu, s'\sim P}[\delta\nabla_\theta V_\theta(s)]$. We then update θ and w separately by calculating the gradient of the above expression. This method is called GTD2. An alternative method is TDC, which uses a slightly different derivation. We again refer to the original papers for this derivation. As these methods calculate the full gradient, they have guaranteed convergence to some local minimum of the MSPBE. Under certain assumptions, global convergence can also be shown. Both residual algorithms and Gradient TD are quite different from temporal difference learning and Q-learning so we will not look further into them for the rest of this paper.

4.3 Linear function approximation

4.3.1 Temporal difference learning

The first type of function approximation we will consider is linear function approximation. Here we estimate the value function by a linear combination of fixed feature functions which are multiplied by parameters. For now, we will consider the case of approximating the value function of Equation (2.1) using a discounted reward for a given policy π. In this case, we will get the following approximation

$$V_\theta(s) = \sum_{k=1}^{K} \theta_k \phi_k(s). \tag{4.10}$$

$$V_\theta = \Phi\theta. \tag{4.11}$$

The second equation gives the vectorized form, where $\theta = (\theta_1, \theta_2, \ldots, \theta_K)^T$ are the parameters. The functions ϕ_k are *feature vectors* of length $|\mathcal{S}|$ in the case of a finite state space, which we will consider in this section. In this case Φ can be seen as an $|\mathcal{S}|$ by K matrix, where the columns are the feature vectors ϕ_k. The update rule is an extension of the rule for general TD(λ) methods and it now uses a parameter update rule. Each time step, we calculate temporal difference δ_t and update parameters θ_t and eligibility trace z_t.

$$\begin{aligned}
\delta_t &= r(s_t, \pi(s_t)) + \gamma\phi(s_{t+1})\theta_t - \phi(s_t)\theta_t \\
\theta_{t+1} &= \theta_t + \beta_t \delta_t z_t \\
z_{t+1} &= \lambda\gamma z_t + \phi(s_{t+1})
\end{aligned}$$

The paper Tsitsiklis and Van Roy [19] gives exact requirements for when temporal difference learning with λ-returns using linear function approximation converges. The exact convergence result and the needed assumptions are given below. It also shows that we can have divergence when some of these requirements are not met, for example in the case of a non-linear approximator. In this section we consider the evaluation of a policy π and the used method is an on-policy method, so the deadly triad is averted.

We will now define the actual objective that we try to optimize when using linear function approximation. Since λ-returns are considered in the paper by Tsitsiklis and Van Roy, we can't use the MSBE with the Bellman operator. Instead we need a TD(λ)-operator, defined as:

$$(\mathcal{B}^{(\lambda)}V)(s) = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\mathbb{E}[G_{t:t+n}|s_t = s]. \tag{4.12}$$

Here $G_{t:t+n}$ is the n-step reward defined in Equation (3.3). We also include a projection in the objective. This projection is now done on the space of linear function approximations. In this projection the authors use a norm that is weighted by μ. The norm and projection are defined as

$$||x||_D = \sqrt{\langle x, x\rangle_D} = \sqrt{x^T D x}, \tag{4.13}$$

$$\Pi_D V = \underset{V' \in \{\Phi\theta | \theta \in \mathbb{R}^K\}}{\arg\min} ||V' - V'||_D. \tag{4.14}$$

Using this projection Π_D and the TD(λ) operator we define our optimization problem as

$$\min_{\theta \in \mathbb{R}^K} \mathbb{E}_{s\sim\mu}[(\Pi_D \mathcal{B}^{(\lambda)} V_\theta(s) - V_\theta(s))^2]. \tag{4.15}$$

Now we will show the main theorem of convergence. This theorem relies on 4 assumptions, which will be given below.

Assumption 1. *The Markov chain induced by our policy π is irreducible and aperiodic and has a stationary distribution μ, which satisfies*

$$\mu = P\mu, \tag{4.16}$$

where $P_{ij} = P(j|i, \pi(i))$ is the transition matrix of the Markov chain. Furthermore, the steady-state variance of the rewards must be finite. Mathematically we require that

$$E_{s\sim\mu}[r^2(s, \pi(s))] < \infty. \tag{4.17}$$

This expectation is taken with respect to the stationary distribution.

Assumption 2. *The matrix Φ has full column rank, which implies that the basis functions ϕ_k are linearly independent. Furthermore, the feature vectors should not grow too fast. Mathematically, they should satisfy*

$$E_{s\sim\mu}[\phi_k^2(s)] < \infty. \tag{4.18}$$

Assumption 3. *The step sizes α_t are positive, non-increasing and chosen prior to the execution of the algorithm. These step sizes must satisfy*

$$\sum_t \alpha_t = \infty \qquad \sum_t \alpha_t^2 < \infty. \tag{4.19}$$

Now we will give the main result from the paper by Tsitsiklis and Van Roy [19].

Theorem 4.1. *If assumptions 1-3 are satisfied, then the following things hold.*

1. *The value function V^π satisfies $||V^\pi||_D < \infty$.*
2. *For $\lambda \in [0,1]$, the TD(λ) algorithm with linear function approximation converges with probability 1.*
3. *The limit of convergence θ^* is the unique solution of the fixed point equation $\Pi\mathcal{B}^\lambda(\Phi\theta^*) = \Phi\theta^*$.*
4. *Furthermore, θ^* satisfies $||\Phi\theta^* - V^\pi||_D \leq \frac{1-\lambda\gamma}{1-\gamma}||\Pi V^\pi - V^\pi||_D$.*

This algorithm shows that we converge to the global minimum of (4.15). It also gives us a bound on error of our approximation with respect to the best possible approximation.

4.3.2 Q-learning with linear function approximation

We can also apply linear function approximation in Q-learning. The deadly triad will apply in this case and can even lead to divergence as we have seen in the counterexample of Section 4.1.1. However, under an assumption on the behavioural policy convergence can still be achieved. Since Q-learning uses state action values, we will apply the following approximation using linear features. We let

$$Q_\theta(s,a) = \sum_{k=1}^{K} \theta_k \phi_k(s,a), \tag{4.20}$$

where the features ϕ_k now are vectors in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ or functions over the space $|\mathcal{S}||\mathcal{A}|$. Of course, we will also need a modified update rule when linear function approximation is introduced. Just as in temporal difference learning we will be using a direct algorithm to update the parameters. We adapt Equation (4.8) of the direct algorithm update rule with the update rule of Q-learning in Equation (3.8). We then obtain the following update for the parameters.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t \nabla_\theta Q_\theta(s,a)[r(s,a) + \gamma \max_b Q_\theta(s',b) - Q_\theta(s,a)] \\ &= \theta_t + \alpha_t \phi(s,a)[r(s,a) + \gamma \max_b \phi(s',b)^T\theta - \phi(s,a)^T\theta]\end{aligned} \tag{4.21}$$

Again we want to show that θ_t converges to a θ^* which is optimal in some sense. The convergence behaviour of Q-learning was analyzed by Melo [11]. We will briefly go over the assumptions and the results from this paper and show their relevance. We will again need Assumptions 1, 2 and 3 from our previous section where we deal with temporal difference learning. However, we will now need an additional assumption on the behavioural policy π. For this assumption, we introduce two matrices Σ_π and $\Sigma_\pi^*(\theta)$. The first matrix is defined by

$$\Sigma_\pi = \mathbb{E}_{\mu,\pi}[\phi(s,a)\phi(s,a)^T]. \tag{4.22}$$

For the second matrix we let $\theta \in \mathbb{R}^K$ and $s \in \mathcal{S}$ be fixed. Then, we let a_s^θ be a maximizing action for s and θ. In particular, it satisfies

$$a_s^\theta \in \arg\max_{a\in\mathcal{A}}\{\phi(s,a)^T\theta\}. \tag{4.23}$$

Then, the second matrix is defined as

$$\Sigma_\pi^*(\theta) = \mathbb{E}_\mu[\phi(s,a_s^\theta)^T\phi(s,a_s^\theta)]. \tag{4.24}$$

The assumption then reads as follows.

Assumption 4. *For any $\theta \in \mathbb{R}^K$, the matrices Σ_π and $\Sigma_\pi^*(\theta)$ satisfy*

$$\Sigma_\pi > \gamma^2 \Sigma_\pi^*(\theta). \tag{4.25}$$

Now we proceed to the main result of the paper by Melo [11].

Theorem 4.2. *If assumptions 1-4 are satisfied, θ_t converges to θ^* w.p. 1. Furthermore, θ^* satisfies*

$$Q_{\theta^*} = \Pi_{\mathcal{F}} \mathcal{B} Q_{\theta^*}. \tag{4.26}$$

In this Equation $\Pi_{\mathcal{F}}$ is the projection onto the space of linear approximators. Most of the assumptions are quite standard an we have seen them in other cases. However, assumption 4 stands out as it is quite different. This assumption basically says that the actions taken by the behavioural policy π are close the current greedy actions. As Melo puts it, we need to have that

$$\max_{a \in \mathcal{A}} \phi^T(s,a)\theta \approx \sum_{a \in \mathcal{A}} \pi(a|s)\phi^T(s,a)\theta. \tag{4.27}$$

This is quite a strong requirement, especially considering that we assumed a stationary behavioural policy. This assumptions also ties in really nicely with the deadly triad. Recall that the deadly triad told us that we have instability when bootstrapping, off-policy learning and function approximation is used. This assumptions controls how 'off-policy' our method can get, since we require the behavioural policy and target policy to be close together. So even though all elements of the deadly triad are present, we still need to control for them to get convergence.

4.3.3 Finite-time bounds in Q-learning

The previous two sections look at convergence in the limit when linear function approximation is used. More recently there have also been papers which look into the finite time behaviour of temporal difference learning and Q-learning. Bhandari [2] provided a finite time analysis for temporal difference learning with linear function approximation, while Chen [4] finds finite time bounds for Q-learning with linear function approximation. We will briefly describe the assumptions and results for the latter paper.

General nonlinear SA

First of all, the paper by Chen looks at the case of general nonlinear stochastic approximation. In particular, it looks at solving the following problem for $\theta^* \in \mathbb{R}^K$:

$$\bar{F}(\theta) := \mathbb{E}_\mu[F(X,\theta)] = 0. \tag{4.28}$$

Here we have that $X \in \mathcal{X}$ is a random vector with distribution μ and $F : \mathcal{X} \times \mathbb{R}^K \to \mathbb{R}^K$ is a general non-linear mapping. The standard stochastic approximation approach to solving this equation is to collect samples X_t of X and iteratively update θ_t, given some initial θ_0, by

$$\theta_{t+1} = \theta_t + \alpha_t F(t) \tag{4.29}$$

with step sizes α_t. Analysis of the convergence of this method is usually done with an ODE argument where we look at the differential equation

$$\dot{\theta}(t) = \bar{F}(\theta). \tag{4.30}$$

The convergence analysis in the article depends on three assumptions. These are that the random samples X_k should follow from an irreducible and aperiodic Markov chain and the function F should be Lipschitz continuous. Finally, $\bar{F}(\theta) = 0$ should have unique solution θ^* and it should satisfy

$$(\theta - \theta^*)^T \bar{F}(\theta) \leq -\kappa ||\theta - \theta^*||_2^2 \tag{4.31}$$

Under these assumptions, Chen proves bounds on $\mathbb{E}[||\theta_t - \theta^*||_2^2]$. The core idea is that the above equation ensures that θ^* is a globally exponentially stable (GES) equilibrium point of the differential equation. To see this, let $V(\theta) = ||\theta - \theta^*||_2^2$ be a candidate Lyapunov function. We then have that

$$\frac{d}{dt} V(\theta(t)) = 2(\theta(t) - \theta^*)^T \dot{\theta}(t) = 2(\theta(t) - \theta^*)^T \bar{F}(\theta) \leq -2\kappa V(\theta(t)). \tag{4.32}$$

Thus we have that $||\theta(t) - \theta^*||_2^2 = V(\theta(t)) \leq V(\theta(0))e^{-2\kappa t} = ||\theta_0 - \theta^*||_2^2 e^{-2\kappa t}$, which decays exponentially fast to 0. Under the assumptions on this nonlinear problem Chen showed finite time bounds on the parameters.

We won't give the exact bounds here, but we will say what factors are important. First of all, the Lipschitz factor and the initial parameter vector θ_0 influence the initial error. The factor κ influences the rate of decay of the bound, as we have already seen in the above argument. Two other factors are really important for the bounds. These are the step sizes and the mixing time of the Markov chain.

For the step sizes we don't use Assumption 3 which is more relevant for convergence in the limit. Instead, Chen places restrictions on α_t dependent on the problem and shows bounds under different choices of step sizes. For instance, bounds are derived when constant step size are used or when they are of the form $\alpha_t = \frac{\alpha}{t+h}$ for some α and h. The mixing time is defined using the total variation between probability distributions.

Definition 4.1. Let ν_1 and ν_2 be two probability distributions defined on $\mathcal{S}$. Then the total variation between ν_1 and ν_2 is defined as

$$||\nu_1(s) - \nu_2(s)||_{\text{TV}} = \frac{1}{2}\sum_{s\in\mathcal{S}} |\nu_1(s) - \nu_2(s)|. \tag{4.33}$$

Definition 4.2. Let the maximal distance between the t-step transition probability vector $P^t(s,\cdot)$ and stationary distribution μ be defined as

$$d_{\max}(t) := \max_{s\in\mathcal{S}} ||P^t(s,\cdot) - \mu(s)||_{\text{TV}}. \tag{4.34}$$

Then, the mixing time $t_{\text{mix}}(\delta)$ with precision δ of a Markov chain is defined as

$$t_{\text{mix}}(\delta) = \min t : d_{\max}(t) \leq \delta. \tag{4.35}$$

Applying nonlinear SA on Q-learning

All of the above analysis is for the general case of nonlinear stochastic approximation. However, we are interested in Q-learning with linear function approximation. To fit this into the nonlinear SA framework, we recall our update rule of Equation (4.21). Note that this is similar to the general update rule of nonlinear SA in Equation (4.29). As such, we now try to solve

$$\mathbb{E}_\mu[\phi(s,a)(r(s,a) + \gamma \max_{a'\in\mathcal{A}} \phi(s',a')^T\theta - \phi(s,a)^T\theta)] = 0 \tag{4.36}$$

to obtain θ^*. This is also a nonlinear problem, even though the approximation architecture is linear. Chen now proposes three assumptions under which we satisfy the assumptions of nonlinear SA. The first two are Assumptions 1 and 2 that we already saw in the previous sections regarding the ergodicity of the Markov chain and the independence and boundedness of the features. The final assumption again regards to the behavioural policy.

Assumption 5. *The problem of Equation* (4.36) *has unique solution* θ^* *and there exists* $\kappa > 0$ *such that the following equation holds for all* $\theta \in \mathbb{R}^K$.

$$\gamma^2\mathbb{E}_\mu[\max_{a\in\mathcal{A}}(\phi(s,a)^T\theta)^2] - \mathbb{E}_{\mu,\pi}[(\phi(s,a)^T\theta)^2] \leq -2\kappa||\theta||_2^2. \tag{4.37}$$

This assumption again restricts how off-policy our behavioural policy can get, in similar vain to Assumption 4. Under these assumptions the problem of Q-learning with linear function approximation fits into the nonlinear SA framework. Under such an extra assumption, finite time bounds on the parameter approximations in Q-learning are obtained in the paper by Chen [4].

4.4 Neural network approximation

Instead of linear function approximation, we can also use neural networks to approximate the value function. Using neural networks has become very popular in recent years, due to the rise in Deep Learning caused by increases in computational power. A popular example that we have already mentioned is the paper by Mnih [12], which applies Deep Q-learning on Atari games. For the basis structure of neural network approximation we look at a feed-forward neural network $f(\theta, x)$.

$$f(\theta,x) = W_L\sigma_L(W_{L-1}\dots\sigma(W_1x + v_1)\dots v_{L-1}) + v_L), \tag{4.38}$$

We have that $\theta = (W_1, \dots, W_L, v_1, \dots, v_L)$ are the parameters with L the number of layers. Furthermore, the layers have a width of m implying that $W_i \in \mathbb{R}^{m \times m}$ for $i = 2, \dots, L-1$. The initial layer has $W_1 \in \mathbb{R}^{m \times d}$ and the final layer has $W_L \in \mathbb{R}^{1 \times m}$. For the input we have $x \in \mathbb{R}^d$ and $\sigma(y) = \max\{y, 0\}$ is the ReLU activation function. Neural networks are characterized by the great number of parameters that are contained in the network. These methods are non-linear due to the activation functions $\sigma(x)$ and thus optimizing them is non-trivial. To update the parameters via gradient descent backpropagation has to be used.

4.4.1 Deep Q-learning

Convergence of Q-learning with neural networks has been analyzed by Xu [24], who also provides finite time bounds. In Deep Q-learning, we estimate the state-value function by a neural network as described in Equation (4.38). Thus, we will approximate the Q-values by $Q_\theta(s, a) = f(\theta, (s, a))$. The objective is to minimize the MSPBE of Equation (4.7), instead using state-action values $Q_\theta(s, a)$ and the Bellman optimality operator defined in Equation (2.8).

$$\min_{\theta \in \Theta} E_\mu[(Q_\theta(s, a) - \Pi_{\mathcal{F}} \mathcal{B} Q_\theta(s, a))^2]. \tag{4.39}$$

Choosing the function space $\mathcal{F}$ is very important when neural network approximation is used. If we simply project on the space of neural network approximators we are still left with a nonconvex problem. In the paper by Xu [24], the function class is chosen to be all local linearizations of $f(\theta, (s, a))$ around some initial point θ_0. Mathematically, this function class is defined as

$$\mathcal{F}_{\Theta,m} = \{f(\theta_0) + \langle \nabla_\theta f(\theta_0), \theta - \theta_0 \rangle | \theta \in \Theta\}. \tag{4.40}$$

Here we abbreviate $f(\theta, (s, a))$ as $f(\theta)$ and $\Theta \subseteq \mathbb{R}^K$ is a constraint set. Using this local linearization, this paper then performs a finite time-analysis to show a rate of convergence of the deep Q-learning algorithm. The algorithm is very similar to standard Q-learning, but now an update rule similar to (4.4) is used. This update rule is defined as

$$\begin{aligned} \delta_t &= r(s, a) + \gamma \max_b Q_\theta(s', b) - Q_\theta(s, a), \\ \theta_{t+1} &\leftarrow \theta_t - \alpha \delta_t \nabla_\theta Q_\theta(s, a). \end{aligned}$$

The authors then show that for any approximate stationary point θ^* of the deep Q-learning algorithm we have that $\hat{f}(\theta^*) = \Pi_{\mathcal{F}_{\Theta,m}} \mathcal{B} \hat{f}(\theta)$, where $\hat{f}(\theta) \in \mathcal{F}_{\Theta,m}$. Thus, we can show convergence of the local linearizations to this fixed point $\hat{f}(\theta^*)$. Then, we connect $\hat{f}(\theta)$ to the actual approximated state-action values $Q_\theta(s, a)$ to show convergence to Q^*.

Following this idea, the paper by Xu [24] proves two theorems. First, convergence is shown of the local linearization $\hat{f}(\theta_t)$ to $\hat{f}(\theta^*)$ with a rate of approximately $O(1/\sqrt{T})$. In Equation (4.41) this convergence result is shown with T being the number of iterations of the algorithm. The constants C_0 and C_1 depend on width of the network m and they vanish as m grows large.

$$\frac{1}{T} \sum_{t=0}^{T} E[(\hat{f}(\theta_t) - \hat{f}(\theta^*))^2] \leq \frac{1}{\sqrt{T}} + C_0 + C_1 \tag{4.41}$$

Using this result, it is then also shown that the function approximation $Q_{\theta_t}(s, a)$ converges to the optimal function $Q^*(s, a)$. The rate of this converges depends on the projection error on our function class and we again have a term of $O(1/\sqrt{T})$.

$$\frac{1}{T} \sum_{t=0}^{T} E[(Q_{\theta_t}(s, a) - Q^*(s, a))^2] \leq \frac{1}{\sqrt{T}} + E[(\Pi_{\mathcal{F}} Q^*(s, a) - Q^*(s, a))^2] + C_0 + C_1 \tag{4.42}$$

The projection error is influenced by the width of the network, since larger networks represent a larger class of functions and thus yield better approximations. To conclude, this paper shows that a converge rate of $O(1/\sqrt{T})$ can be achieved for networks with a large width m. This result relies on multiple assumptions, most notably Assumption 4, which we have already seen earlier. Again we need to control the behavioural policy, since we are dealing with the deadly triad. All these cases show that when function approximation is used, we need extra assumptions to make the convergence proof work.

Chapter 5

Analyzing Q-learning using Markov Decision theory

In Section 3.3 we have seen that Q-learning converges in the general case. In the previous chapter we have seen that convergence can also be achieved when function approximation is used. We saw that these results rely on multiple assumptions. Assumption 1 regards the structure of the Markov Decision process and the behavioural policy. It makes sure that every state and action is visited infinitely often. For convergence this assumptions is always necessary, because $Q(s, a)$ can't converge if we don't visit (s, a). However, Assumptions 3 and 4 or 5 are more restrictive. The first of these was needed for standard Q-learning and says that the step sizes (or learning rate) should decay at a proper rate. Assumptions 4 or 5 were needed when function approximation is used and arose due to the deadly triad. They control the behavioural policy such that it is not far off the current best policy.

In this chapter we take a different approach to analyze the convergence of Q-learning. We will use the properties of the MDP on which we apply Q-learning to study the behaviour of the algorithm. In this way, we can determine which reward terms are added in the Q-learning update and as a result also to what values we will converge. Of course, to use properties of the MDP we need to specify what types of MDPs will be considered. The class of Markov decision processes we will look at are birth-death processes. For these processes, we will consider multiple reward functions. For the most basic case of constant rewards we will prove exact convergence, while for more complex reward structures we will derive upper and lower bounds on the value to which Q-learning will converge. Our method doesn't make additional assumptions on the step size and behavioural policy that we have seen in other results. As a result, we avoid some of the issues that other articles that analyze Q-learning encounter.

5.1 The Markov Decision Process

As we have mentioned above, we will consider birth-death type Markov decision processes. We let our MDP have N states where each state s is adjacent to states $s-1$ and $s+1$, such that we can only move one step to the left or right at a time. The exceptions are state 1 and N: they remain in the same state if they would transition to states 0 and $N+1$ respectively. We let each state in the MDP have the same 2 actions. If we take action 1, we move one state to the right with probability p_1 and move one state to the left with probability $1-p_1$. Similarly, for action 2 we take a step to the right with probability p_2, and a step to the left with probability $1-p_2$. For the rest of this chapter, we will assume w.l.o.g. that $p_1 > p_2$. Only the strictly larger case is considered, since the action doesn't matter if $p_1 = p_2$. The Markov Decision process is displayed in Figure 5.1.

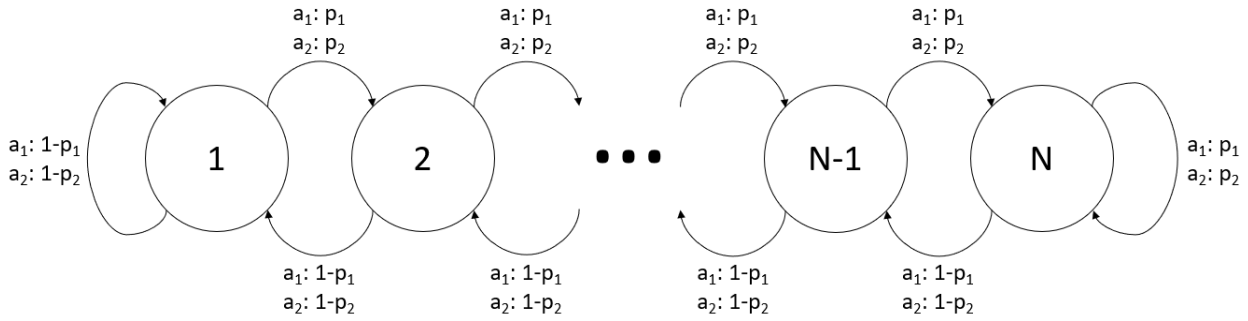


Figure 5.1: The Markov Decision process which will be analyzed in this chapter.

In particular, the transition probabilities for this MDP are

$$P(s+1|s,a) = \begin{cases} p_1 & \text{if } s = 1,\dots,N-1, \quad a = a_1 \\ p_2 & \text{if } s = 1,\dots,N-1, \quad a = a_2 \end{cases}$$

$$P(s-1|s,a) = \begin{cases} 1-p_1 & \text{if } s = 2,\dots,N, \quad a = a_1 \\ 1-p_2 & \text{if } s = 2,\dots,N, \quad a = a_2 \end{cases}$$

$$P(s|s,a) = \begin{cases} p_1 & \text{if } s = N, \quad a = a_1 \\ p_2 & \text{if } s = N, \quad a = a_2 \\ 1-p_1 & \text{if } s = 1, \quad a = a_1 \\ 1-p_2 & \text{if } s = 1, \quad a = a_2 \end{cases}$$

For all other cases we have that $P(s'|s,a) = 0$. The reward function will be varied throughout the sections in this chapter. For convenience, we will again state the Q-learning update rule from Equation (3.8) below.

$$Q_{t+1}(s,a) = \begin{cases} (1-\alpha)Q_t(s,a) + \alpha[r(s,a) + \gamma \max_{a' \in \{a_1,a_2\}} Q_t(s_{t+1},a')] & \text{if } (s,a) = (s_t,a_t), \\ Q_t(s,a) & \text{if } (s,a) \neq (s_t,a_t). \end{cases} \tag{5.1}$$

Here we again have that α is our learning rate and γ is the discount factor. Note that we have slightly modified our notation to clarify that we only update a state-action pair when it actually gets visited. Any behavioural policy applied on this MDP induces an ergodic Markov Reward process as long as p_1 and p_2 don't both equal 0 or 1. This is always satisfied for our MDP, since we assume that $p_1 > p_2$ and as a result, the induced MRP will satisfy Assumption 1. Furthermore, every state-action pair will be visited infinitely many times in the limit if the behavioural policy assigns a positive probability to both actions. Mathematically, we require the behavioural policy to satisfy $\pi_b(a|s) > 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. This requirement is always needed, since if we don't visit a state-action pair (s,a), its value $Q(s,a)$ can never converge. Thus, we will always assume that we use a behavioural policy that satisfies this requirement.

5.2 Constant rewards

We will start our analysis with the most basic case, in which we have a constant reward on the entire state space. In particular, we let $r(s,a) = r$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. It is very easy to determine what the optimal value function Q^* should equal. For any policy π, we get that

$$Q^\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t r = \frac{r}{1-\gamma}$$

by using the definition of $Q^\pi(s,a)$ in Equation (2.6). As a result it is also clear that $Q^* = \frac{r}{1-\gamma}$. Now we want to examine what happens if we apply Q-learning on our MDP with a constant reward of r. We let $Q_0(s,a) = 0$ for all (s,a) be the initial values and choose a constant step size $\alpha \in (0,1]$. We want to analyze how far we have converged for any time finite time t and what happens if t goes to infinity. If we can obtain an exact formula for $Q_t(s,a)$, we could examine both of these cases by letting $t \to \infty$. Let $t > 0$ be some time and (s,a) be any state-action pair. Then, if (s,a) has at least been updated once at time t, we have

$$Q_t(s,a) = (1-\alpha)Q_{t_-1}(s,a) + \alpha[r + \gamma \max_{a' \in \{a_1,a_2\}} Q_{t_-1}(s',a')]. \tag{5.2}$$

We see that we add a reward term αr and also two previous state-action values. Here we let t_{-1} indicate the time of the last update to (s,a). If $Q_{t_{-1}}(s,a)$ and $Q_{t_{-1}}(s',a')$ have also seen an update at time t_{-1}, we can further expand $Q_t(s,a)$ by including these updates. This will lead to additional reward terms.

$$\begin{aligned} Q_t(s,a) &= (1-\alpha)\big((1-\alpha)Q_{t_{-2}}(s,a) + \alpha[r + \gamma \max_{a' \in \{a_1,a_2\}} Q_{t_{-2}}(s',a')]\big) \\ &\quad + \alpha r + \alpha\gamma\big((1-\alpha)Q_{t_{-2}}(s',a') + \alpha[r + \gamma \max_{a'' \in \{a_1,a_2\}} Q_{t_{-2}}(s'',a'')]\big) \end{aligned}$$

Here we see that $Q_t(s,a)$ consists of reward terms, which are weighted by powers of α, $(1-\alpha)$ and γ. Furthermore, we remain with state-action values such as $Q_{t_{-2}}(s,a)$. These can again be expanded until we hit time 0 and run out of updates. Thus, $Q_t(s,a)$ has weighted reward terms and the amount of them depends on the time t and which updates have been performed. Unfortunately, it is not easy to work out when and how many times each state-action pair has been visited (and thus updated) by the behavioural policy. This was already apparent in the previous equation, as we couldn't exactly index and clarify the states, actions and update times. These issues arise because Q-learning is an *asynchronous method*, which means that a state-action pair is only updated when it is visited by the behavioural policy. This is also clear from our update rule in Equation (5.1). To help us, we will first look at a simpler case in the next section.

5.2.1 Synchronous Q-learning

As we have seen, working out $Q_t(s,a)$ isn't very easy due to update rule of Q-learning. In this section, we will consider an alternative for Q-learning in which we update every state-action pair in every time step. This will make Q-learning a *synchronous* method. For this case, it is clear how to work out $Q_t(s,a)$ and we can also show exact convergence. This is shown in the following result.

Lemma 5.1. *Let our MDP be given in Figure* (5.1) *and assume we have constant reward $r \in \mathbb{R}$. On this MDP we apply the the update rule of Equation* (5.1) *with the added change of updating every state-action pair in every time step t. Then, we obtain the following two results:*

1. *For any time t, we obtain the following equation for $Q_t(s,a)$.*

$$Q_t(s,a) = \sum_{m=0}^{t-1} \sum_{l=0}^{m} \binom{m}{l} (\alpha\gamma)^l (1-\alpha)^{m-l} \alpha r. \tag{5.3}$$

2. *$Q_t(s,a)$ converges to the optimal value function $Q^*(s,a) = \frac{r}{1-\gamma}$ as $t \to \infty$.*

Proof. For the first result, we can simply work out $Q_t(s,a)$ with the added knowledge that every state-action pair has been updated exactly t times. Thus we also now how many and which reward terms appear in $Q_t(s,a)$. This leads to the following equation.

$$\begin{aligned} Q_t(s,a) &= (1-\alpha)Q_{t-1}(s,a) + \alpha[r + \gamma \max_{a' \in \{a_1,a_2\}} Q_{t-1}(s',a')] \\ &= (1-\alpha)\big((1-\alpha)Q_{t-2}(s,a) + \alpha[r + \gamma \max_{a' \in \{a_1,a_2\}} Q_{t-1}(s',a')]\big) \\ &\quad + \alpha\big[r + \gamma[(1-\alpha)Q_{t-2}(s',a') + \alpha[r + \gamma \max_{a' \in \{a_1,a_2\}} Q_{t-2}(s'',a')]\big] \\ &= \dots \\ &= \sum_{m=0}^{t-1} \sum_{l=0}^{m} \binom{m}{l} (\alpha\gamma)^l (1-\alpha)^{m-l} \alpha r. \end{aligned} \tag{5.4}$$

Here we have assumed that $Q_0(s,a) = 0$ for all (s,a) such that we have already obtained the first result.

For the second result, we exploit the structure of this equation. We see that each reward term is weighted by a powers of $(1-\alpha)$ and $\alpha\gamma$, while also being weighted by the binomial coefficients of these powers. To prove convergence to $Q^*(s,a)$, we need to analyze what happens to Equation (5.4) as $t \to \infty$. We will also change the order of summation to obtain

$$\sum_{l=0}^{\infty} \alpha r (\alpha\gamma)^l \sum_{m=l}^{\infty} \binom{m}{l} (1-\alpha)^{m-l} = \sum_{l=0}^{\infty} \alpha r (\alpha\gamma)^l \sum_{n=0}^{\infty} \binom{n+l}{l} (1-\alpha)^n. \tag{5.5}$$

Now if we fix any value of $l = 0, 1, 2, \ldots$ we want to prove that

$$\alpha r(\alpha\gamma)^l \sum_{n=0}^{\infty} \binom{n+l}{l} (1-\alpha)^n = \gamma^l r, \tag{5.6}$$

because then we get for the total sum of rewards

$$\sum_{l=0}^{\infty}\sum_{n=0}^{\infty} \binom{n+l}{l} (1-\alpha)^n (\alpha\gamma)^l \alpha r = \sum_{l=0}^{\infty} \gamma^l r = \frac{r}{1-\gamma}. \tag{5.7}$$

As a result, we would have that $Q_t(s,a) \to Q^*(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ as $t \to \infty$. To satisfy Equation (5.6) we need the following equality to hold:

$$\sum_{n=0}^{\infty} \binom{n+l}{l} (1-\alpha)^n = \frac{1}{\alpha^{l+1}}. \tag{5.8}$$

That this holds can be shown by taking derivatives of the geometric series with powers of $(1-\alpha)$. In particular we take the l-th derivative of both the infinite sum and $\frac{1}{(1-(1-\alpha))}$, which is what the geometric series equals.

$$\frac{d^l}{d(1-\alpha)^l} \sum_{n=0}^{\infty} (1-\alpha)^n = \sum_{n=0}^{\infty} (n+1)(n+2)\cdots(n+l)(1-\alpha)^n = \sum_{l=0}^{\infty} \frac{(n+l)!}{n!}(1-\alpha)^l, \quad \text{while also} \tag{5.9}$$

$$\frac{d^l}{d(1-\alpha)^l} \frac{1}{(1-(1-\alpha))} = \frac{l(l-1)\cdots 1}{(1-(1-\alpha))^{l+1}} = \frac{l!}{\alpha^{l+1}} \tag{5.10}$$

Both right hand sides will equal each other, because they both are derivatives of the geometric series. If we divide both the top and bottom right hand sides by $l!$ we obtain Equation (5.8). As a result, Equations (5.6) and (5.7) will also hold, proving that Q_t converges to $Q^*(s,a) = \frac{r}{1-\gamma}$ as $t \to \infty$. □

5.2.2 'Uniformization' of the update rule

In the previous section we have proven convergence in the synchronous case. In principle, the same result should hold in the asynchronous case as the amount of updates goes to infinity, because then the same reward terms will appear such that we have convergence. The problem is that for any time t it is hard to determine how many updates have occurred to which state-action pair and when. While we can perhaps look at the limiting case, we won't be able to give bounds for a finite value of t. As a result, we want to introduce some regularity to the problem, such that working out these expressions is easier. We achieve this by analyzing a slightly modified update rule, inspired by uniformization in continuous-time Markov chains (CTMC).

The technique of uniformization allows us to approximate a continuous-time Markov chain (CTMC) by a discrete-time Markov chain (DTMC). Let's say we transition out of state s of some CTMC at a rate of λ_s. We will define $\lambda := \max_s \lambda_s$. Then, we introduce a modified process with transition rate λ for each state. However, the transitions are now accepted with probability $\frac{\lambda_s}{\lambda}$ and rejected with probability $1 - \frac{\lambda_s}{\lambda}$. Accepting means we do make a transition according to the original CTMC, while rejecting means we stay in the same state. This can be described by a DTMC where we update every state in every time step, but accept or reject the updates with the probabilities mentioned above.

We can apply similar techniques to our process. While we don't have a continuous time Markov chain, we have different rates of updates to each state s. In particular, the time between updates to state s is determined by the recurrence time. The Markov chain induced by our behavioural policy is ergodic, so the recurrence times exist and are finite. Furthermore, they equal the inverse of the stationary distribution $\mu \in \mathbb{R}^{|\mathcal{S}|}$ of the Markov chain. A definition of μ can be found in Section 2.2.1. However, we are not interested in the recurrence times of a state, but the recurrence times of a state-action pair. We define

$$\nu(s,a) = \pi_b(a|s)\mu(s) \tag{5.11}$$

as the stationary distribution of the state action pair (s,a) under our behavioural policy π_b. This is defined as the fraction of time that we spend in state-action pair (s,a) on average in the long run. Since, we have defined $\pi_b(a|s) > 0$, the stationary distribution will also be nonzero fore very state-action pair. The recurrence time of state-action pair (s,a) then equals $1/\nu(s,a)$. Now we will apply a similar trick as is done in uniformization. In each time step, we update every state-action pair. However, we accept an update by the rate of visits to this state-action pair. That is, we accept an update with probability $\frac{1}{1/\nu(s,a)} = \nu(s,a)$ and

reject with probability $1-\nu(s,a)$. Note that we do not have to scale these by a maximum rate, because the stationary distribution already forms a probability distribution. We then finally have the following update rule at each time t

$$Q_{t+1}(s,a) = \begin{cases} (1-\alpha)Q_t(s,a) + \alpha[r + \gamma \max_{a' \in \{a_1,a_2\}} Q_t(s',a')] & \text{with probability } \nu(s,a) \\ Q_t(s,a) & \text{with probability } 1-\nu(s,a) \end{cases}. \tag{5.12}$$

Here s' is sampled from transition function $P(s'|s,a)$. This is of course a different update rule than the one in Equation (5.12). However, the reason for introducing this update rule is that we simplify the way updates are done, such that the analysis becomes easier. So this update rule will mainly have to help us in proving our results. In Section 5.2.4 we will show how this 'uniformized' update rule differs from the standard Q-learning update rule.

5.2.3 Convergence for the 'uniformized' update rule

Now let's apply this new update rule. In the synchronous case we saw that by updating every time step we make sure that the eventual weighted sum of rewards converges to the optimal value function. The key here was that we update all state-action pairs in every time step. By using our modified update rule, we can still update in each time step but accept or reject the updates. Since we know the probability of accepting the update, we can work out how many updates have been applied to each state-action pair.

What we will do is calculate the time needed to update each state-action pair at least once. If it would take T time steps to do this, then we know that at time T each state-action value has seen at least one update. Thus, when a new Q-learning update is performed to (s,a), we guarantee that its value function now has at least two updates. Generalizing this, for any m we at least have

$$Q_{(m+1)T}(s,a) = (1-\alpha)Q_{mT}(s,a) + \alpha[r + \gamma \max_{a' \in \{a_1,a_2\}} Q_{mT}(s',a')] \tag{5.13}$$

for every state-action pair (s,a). As $m \to \infty$ we guarantee infinitely many updates and reward terms, for which we have seen that convergence holds from the synchronous case. Additionally, more updates could have happened in between mT and $(m+1)T$ to (s,a), but we won't take them into account here. Unfortunately, we can't simply calculate a number of time steps T in which we guarantee that each state-action pair gets updated. This is because accepting an update is a Bernoulli trial and thus random. However, we can calculate the distribution of time steps needed to update every state-action pair, because of the structure we added to our update rule.

In every time step we perform a Bernoulli trial with probability $\nu(s,a)$ to update (s,a). Thus, the amount of time steps it takes to update (s,a) follows a geometric distribution with parameter $\nu(s,a)$. We will denote this by random variable by $G(s,a) \sim \text{Geo}(\nu(s,a))$. To update every state-action pair at least once, we need to consider the maximum over these geometric variables. Define $G := \max\{G(s,a) | (s,a) \in \mathcal{S} \times \mathcal{A}\}$. After T time steps, the probability that we have updated every state-action pair at least once will be denoted by p_T and equals

$$\begin{aligned} p_T &= P(G \leq T) = P(\max\{G(s,a) | (s,a) \in \mathcal{S} \times \mathcal{A}\} \leq T) \\ &= P(G(s_1,a_1) \leq T, G(s_1,a_2) \leq T, \ldots, G(s_N,a_2) \leq T) \\ &= P(G(s_1,a_1) \leq T) \cdot P(G(s_1,a_2) \leq T) \cdots P(G(s_N,a_2) \leq T) \\ &= (1-(1-\nu(s_1,a_1))^T) \cdot (1-(1-\nu(s_1,a_2))^T) \cdots (1-(1-\nu(s_N,a_2))^T). \end{aligned} \tag{5.14}$$

Thus, for any amount of time steps T we can calculate the probability that every state-action pair will be updated in that time frame. Thus, the probability that we update every state-action pair at least once in T time steps is again a Bernoulli trial, now with parameter p_T. At every time t, we have thus had at least $\left\lfloor \frac{t}{T} \right\rfloor$ of such Bernoulli trials. The number of successes will be a binomial random variable, which we denote by N_t and is defined as:

$$N_t \sim \text{Binom}(\left\lfloor \frac{t}{T} \right\rfloor, p_T). \tag{5.15}$$

This gives the number of times that we have updated every state-action pair at least once in a time frame of T steps. On average, we expect at least

$$n_t := \left\lfloor p_T \left\lfloor \frac{t}{T} \right\rfloor \right\rfloor \leq \mathbb{E}[N_t] \tag{5.16}$$

successes. Using these definitions, we move onto our result for this section.

Theorem 5.2. *Let our MDP be given as described in Figure 5.1 and assume we have constant reward $r \in \mathbb{R}$. Furthermore, let $T \in \mathbb{N}$ be any positive number of time steps. Then, the following two results hold for the update rule given in Equation* (5.12) *when applied to the MDP.*

1. *For any time t and state-action pair (s, a) the following bound holds.*

$$\mathbb{E}|Q_t(s,a) - Q^*(s,a)| \leq \Big| \sum_{m=0}^{n_t-1} \sum_{l=0}^{m} \binom{m}{l} (\alpha\gamma)^l (1-\alpha)^{m-l} \alpha r - \frac{r}{1-\gamma} \Big| \tag{5.17}$$

   Here n_t and p_T are defined as in Equations (5.16) *and* (5.14) *respectively.*

2. *The state-action value $Q_t(s, a)$ converges to $Q^*(s, a)$ with probability 1 as $t \to \infty$.*

Proof. For this proof, we will assume w.l.o.g. that $r \geq 0$. Then, we know that $Q_t(s, a)$ must be a sum of only positive reward terms, since $\alpha \in (0, 1]$ and $\gamma \in (0, 1)$. Since t is finite, the amount of reward terms must also be finite. From Equation (5.7) we see that the infinite sum of reward terms will converge to $Q^*(s, a)$. Because we only have finitely many positive reward terms, it must hold that $Q^*(s, a) \geq Q_t(s, a)$. As a direct result it is obvious that

$$\mathbb{E}|Q_t(s,a) - Q^*(s,a)| = \mathbb{E}[Q^*(s,a) - Q_t(s,a)] = Q^*(s,a) - \mathbb{E}[Q_t(s,a)] = \frac{r}{1-\gamma} - \mathbb{E}[Q_t(s,a)]. \tag{5.18}$$

At time t we have updated every state-action on average at least $n_t \geq \mathbb{E}[N_t]$ times. Thus, for $\mathbb{E}[Q_t(s, a)]$ we can use the result of Equation (5.4) for the synchronous reward by replacing t with n_t. However, this is a lower bound on the number of updates and as a result we will have more reward terms on average. Thus, the following holds

$$\mathbb{E}[Q_t(s,a)] \geq \sum_{m=0}^{n_t-1} \sum_{l=0}^{m} \binom{m}{l} (\alpha\gamma)^l (1-\alpha)^{m-l} \alpha r. \tag{5.19}$$

Combining this with Equation (5.18) proves the first part of the Theorem.

The second part of this proof simply requires that $N_t \to \infty$ as $t \to \infty$. This is always true, because N_t is a binomial random variable with a positive probability of success. As a result, $Q_t(s, a)$ will converge to $Q^*(s, a)$ with probability 1. □

Contribution 5.1. In this section we have shown that exact convergence holds for Q-learning when constant rewards are used. We have also provided finite-time bounds for this case. It is thus clear that our method of analysis provides logical results for cases where convergence should be obvious. The only change between standard Q-learning is that we look at a slightly modified update rule. In the next section we will show how our update rule is different, but that it will still lead to similar asymptotic behaviour w.r.t. standard Q-learning.

5.2.4 Relating the 'uniformized' and standard update rule

The results of Theorem 5.2 where obtained by analyzing the update rule of Equation (5.12). The reason for using this modified update rule was to add structure to the updating process to help with our analysis. First we make analyzing Q_t for any time t easier by not having to rely on the sequence of state-action pairs that were visited. Next to that, the behaviour of Q_t becomes more clear as $t \to \infty$.

However, our initial goal was to analyze Q-learning, and not some modified process. Thus, we have to show how these update rules are related and what our results for the 'uniformized' update rule imply for standard Q-learning. The updates rules differ in exactly one way: when each state-action pair is updated. In Q-learning the behavioural policy generates an episode of state-action pairs $s_0, a_0, s_1, a_2, \ldots$. Then, at every time t we update state-action pair (s_t, a_t) from this episode. In our modified update rule we update each state-action pair (s, a) with probability $\nu(s, a)$, the stationary distribution of state-action pairs.

In our analysis of Theorem (5.12) we considered how often we have updated each state-action pair. Thus, to compare the 'uniformized' and standard update rules we need to analyze the amount of times the amount of times the behavioural policy, corresponding to the standard Q-learning update rule, has visited every state-action pair. For example, if $t = 1$ then standard Q-learning will have updated (s_0, a_0), while for the 'uniformized' rule we could have updated every state-action pair, but also just one of them or none at all.

Let's now do this analysis for a general value of t. For the 'uniformized' update rule of Equation (5.12) the amount of updates to state-action pair (s, a) follows a $\text{Binom}(t, \nu(s, a))$ distribution. The probability that

we update (s, a) in time step t will always be the same and equals $\nu(s, a)$. For the standard update rule the story is different. At time t, we want to know the probability that the behavioural policy is in state-action pair (s, a), given that we started in (s_0, a_0). We define

$$\mathbf{P}_{\pi_b}(s', a'|s, a) := \pi_b(a'|s')P(s'|s, a) \tag{5.20}$$

as the transition matrix between state-action pairs under the behavioural policy. We have that $\mathbf{P}_{\pi_b} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}||\mathcal{A}|}$ and the stationary distribution ν satisfies $\nu = \mathbf{P}_{\pi_b}\nu$. Additionally, we define $\mathbf{v}_0 \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ as the initial probability distribution of state-action pairs. In particular, $\mathbf{v}_0$ is zero for every entry except that $\mathbf{v}_0(s_0, a_0) = 1$. The probability that we update state-action pair (s, a) at time t is given by element (s, a) of $\mathbf{v}_0'\mathbf{P}_{\pi_b}^t$.

To compare the update rules, we will define

$$d(t) := \max_{(s,a)\in\mathcal{S}\times\mathcal{A}} |(\mathbf{v}_0'\mathbf{P}_{\pi_b}^t)(s, a) - \nu(s, a)| \tag{5.21}$$

as the maximum difference between the probabilities of updating a state-action pair in the 'uniformized' or standard process. We recognize this as a measure for how mixed our Markov chain is at time t. Our behavioural policy π_b applied on the MDP of Figure 5.1 induces an ergodic Markov chain states. Because $\pi_b(a|s) > 0$ for all (s, a), the transition matrix $\mathbf{P}_{\pi_b}$ of the Markov chain between state-action pairs will also be ergodic. From the theory on Markov chains, we thus know that $\mathbf{v}_0'\mathbf{P}_{\pi_b}^t$ will converge to the stationary distribution ν. Furthermore, using theory on mixing times we know that $d(t)$ will approach 0 at an exponential rate. For more details on mixing times, we refer to chapter 5 of the book by Levin and Peres [8].

To conclude, as we increase t the probability of updating state-action pair (s, a) approach stationary distribution $\nu(s, a)$. As a result, the amount of updates to (s, a) at time t will approach $t\nu(s, a)$ for both the 'uniformized' and standard Q-learning update rules when t grows larger. For any fixed value of t, the difference in the number of updates depends on $d(t)$, which determines the mixing factor of our updating process.

Contribution 5.2. In this section we have shown how the 'uniformized' update rule of Equation (5.12) we use in the analysis of Q-learning compares to the standard update rule of Equation (5.1). We have seen that the difference in the probability of updating a state-action pair at time t depends on $d(t)$, which measures the mixing factor of our Markov chain. In the limiting case where $t \to \infty$, we have seen that both update rules update every state-action pair as often on average. As a result, the asymptotic behaviour of the update rules will be similar, validating our use of the 'uniformized' Q-learning update rule.

5.2.5 Non-constant rewards

We have seen that we can obtain exact convergence for Q-learning when the rewards are constant. We are interested if similar results apply when the rewards are not the same for every state and action, so we let $r(s, a)$ be any reward function. We assume that the initial values satisfy $Q_0(s, a) = Q^*(s, a)$ for all states and actions. Thus, we are already at the optimal values. If we now apply a Q-learning update, we obtain

$$Q^{\text{new}}(s, a) = (1 - \alpha)Q^*(s, a) + \alpha[r(s, a) + \gamma \max\{Q^*(s', a_1), Q^*(s', a_2)\}]. \tag{5.22}$$

Of course, we want $Q^{\text{new}}(s, a)$ to still equal $Q^*(s, a)$ such that we remain with an optimal value. Thus, we need that

$$Q^*(s, a) = r(s, a) + \gamma \max\{Q^*(s', a_1), Q^*(s', a_2)\}.$$

However, from the Bellman equation and the Bellman optimality operator (2.8) we know

$$Q^*(s, a) = r(s, a) + \gamma\mathbb{E}_{s'}[\max\{Q^*(s', a_1), Q^*(s', a_2)\}] \neq r(s, a) + \gamma \max\{Q^*(s', a_1), Q^*(s', a_2)\}$$

in the general case. The clear difference is the expectation, which is sampled in the Q-learning update. When the optimal state-action values differ, which is almost always the case for non-constant rewards, this expectation won't equal its sampled version. As a result $Q^{\text{new}}(s, a) \neq Q^*(s, a)$. This implies that we can not converge for non-constant rewards, since even if we do converge, we still move away from the optimal values in a new iteration. This is also the reason why the decaying step sizes are need in the Q-learning proof.

5.3 Two-part Markov process

From the example in the previous section it is apparent that we will generally not converge exactly when non-constant rewards are used. However, we can still examine the convergence properties for such cases,

for example by considering bounds. We will again use the MDP described in Figure 5.1, but now we will change our reward structure. The state space will be divided into two parts: $S_1 = \{1, \dots, N^*\}$ and $S_2 = \{N^* + 1, \dots, N\}$. We set the reward to r_1 on the first part and we will have a reward of r_2 on the second part. In particular, we let our reward equal

$$r(s,a) = \begin{cases} r_1 & \text{if } s \in S_1, \\ r_2 & \text{if } s \in S_2. \end{cases} \tag{5.23}$$

We let $N^* \in \{2, \dots, N-1\}$ be any state outside of the edge states. As such, N should at least equal 3. The reason for these requirements on N^* and N will become apparent later. Furthermore, we will assume that $r_1 > r_2$. This means that if we start at the left side of the chain, we see a lot of updates with reward r_1, while at the right side of the chain we have mostly updates with r_2. If we now apply the update rule of Equation (5.12), we obtain for any t and (s,a)

$$Q_t(s,a) \approx \sum_{m=0}^{N_t-1} \sum_{l=0}^{m} \binom{m}{l} (\alpha\gamma)^l (1-\alpha)^{m-l} \alpha r(s^{(l)}, a^{(l)}). \tag{5.24}$$

The reason that we don't use an exact inequality here is because the amount of updates (and thus reward terms) is underestimated by N_t, because N_t only counts how many times we have seen each state-action pair being updated at least once in a frame of T updates. Since the rewards can now equal either r_1 or r_2, we can't simply use this sum as a lower bound like was done in the constant reward case. As a result, we will from now on only look at the limiting case where $t \to \infty$, because we know that we will have an infinite amount of updates and reward terms. We use the following definition to analyze what happens to the Q-learning algorithm as t goes to infinity.

Definition 5.1. We define the random variable $Q'(s,a)$ as the limiting distribution of $Q_t(s,a)$, where $Q_t(s,a)$ is given by the update rule in Equation (5.12). In particular we define

$$Q'(s,a) := \lim_{t \to \infty} Q_t(s,a). \tag{5.25}$$

We know that for every value of t, $Q_t(s,a)$ consists of a finite sum of rewards which either equal r_1 or r_2. Thus, the limiting distribution $Q'(s,a)$ will be bounded by the case where we only have rewards r_1 and the case where we only have rewards r_2.

$$\frac{r_1}{1-\gamma} \geq Q'(s,a) \geq \frac{r_2}{1-\gamma}. \tag{5.26}$$

The distribution of $Q'(s,a)$ depends on the structure of the Markov chain and the value of α.

5.3.1 Total sum of rewards

We will analyze the convergence of Q-learning by analyzing this limiting distribution $Q'(s,a)$. This can be done by working out which reward term are present using the properties of the Markov decision process. We will consider the case where $\alpha = 1$. This can be seen as restrictive, but it makes the analysis clearer. Also, we will later look at the expectation $\mathbb{E}[Q'(s,a)]$, where this learning rate likely wouldn't be of influence.

To work out $Q'(s,a)$ we need to know the properties of $Q_t(s,a)$. We know that it consists of a sum of rewards, but we don't know which rewards they are. To help us work out $Q_t(s,a)$, let's look at a single step of the Q-learning update rule. Assume that we perform an update to (s,a) at time t, where we let $s \in S_1$. For this update we get

$$Q_{t+1}(s,a) = r_1 + \gamma \max_{a'} Q_t(s', a'). \tag{5.27}$$

To work out what $Q_{t+1}(s,a)$ will equal, we need to know what $Q_t(s',a')$ equals. Unless (s',a') hasn't been updated yet, it will be the result of a previous update. For this update we know that if $s' \in S_1$ it will have added a reward of r_1 and if $s' \in S_2$ it will have added a reward of r_2. From this update, we again add the value of another state-action pair, let's say (s'', a''). If $s'' \in S_1$ this would have meant a reward r_1, otherwise we have $s'' \in S_2$ and a reward r_2. This repeats until we have run out of updates. Thus, the reward terms in $Q_{t+1}(s,a)$ are determined by the properties of the MDP.

This is best seen in an example. Consider $N = 4$ and $N^* = 2$, such that $r(1,a) = r(2,a) = r_1$ and $r(3,a) = r(4,a) = r_2$. Let's start in the state-action pair $(1, a_1)$ and look at the reward terms that we see in

this chain of updates. An example path of updates

$$\begin{array}{lccccccccccccc} \text{State-action pair:} & (1,a_1) & \to & (1,a^*_{t_5}) & \to & (2,a^*_{t_4}) & \to & (3,a^*_{t_3}) & \to & (4,a^*_{t_2}) & \to & (3,a^*_{t_1}) & \to & (2,a^*_{t_0}) \\ \text{Reward:} & r_1 & & r_1 & & r_1 & & r_2 & & r_2 & & r_2 & & r_1 \\ \text{Update time:} & t_6 & \leftarrow & t_5 & \leftarrow & t_4 & \leftarrow & t_3 & \leftarrow & t_2 & \leftarrow & t_1 & \leftarrow & t_0 \end{array} \tag{5.28}$$

The times of updates in indexed by $t_0, \ldots, t_6$. In the Q-learning update rule (Equation (5.12)) we update with the value of the maximizing action. Hence, the action in this chain of updates are defined as

$$a^*_{t_i} = \arg\max_{a'} Q_{t_i}(s, a')$$

With this we mean that at time t_i the maximizing action in state s equals $a^*_{t_i}$. Now let's examine the reward sum following in the example in Equation (5.28). Whenever the update happened in section S_1 of the state space, the reward equals r_1 and vice versa for S_2 and r_2. Because updates always happen with a neighbouring state, we can only switch the reward when moving between N^* and $N^* + 1$. So, if we know how much time we spend left and right of N^* among this chain of updates, we know which reward terms are added to $Q_t(s, a)$. As a result, we will be also be able to work out $Q'(s, a)$.

5.3.2 Excursions

In the previous section we saw that $Q'(s, a)$ will consist of an infinite sum of rewards and that we can classify which rewards are present in this sum. This is because the state-action pairs in the updates of Q-learning will follow some path through the state space. Since our MDP is a birth-death type process, every update will be done using a state-action pair of a neighbouring state. As such, we can define what paths through the state space we can take, and thus what rewards we obtain. From Example (5.28) we have seen that along this path through the state space, the reward will only swap between r_1 and r_2 when we move between states N^* and $N^* + 1$. Thus, we can calculate how long the trips left and right of N^* will take. We call call such trips *excursions*. On an excursion left of N^* we only see reward r_1, while on an excursion right of N^* we only see r_2. An excursion left of N^* starts at $N^* - 1$ and ends when we finally return to N^*. Vice versa, excursions right of N^* start in N^+1 and end in N^*. We can calculate the length of these excursions using hitting times.

Definition 5.2. The hitting time, also known as first passage time, from a state s to another state s' is a random variable which gives the number of time steps needed to reach s' when starting in s. We will denote such hitting times by the letter H.

By using this concept of excursions, we want to establish a formula for $Q'(s, a)$. We do so by defining random variables that define the length of these excursions. Additionally, we need to know the probability that we do a left or right excursion. Finally, the starting state s generally doesn't equal N^*, so we need to determine the reward earned in the initial few updates along our update path until we reach N^*. After this, the update path follows excursions left and right of N^*. These will all follow a similar structure, and will be indexed by $i = 1, 2, \ldots$. We have defined the random variables that we need in Table 5.1.

Variable	Definition
H^1_i	The length of the i-th if it is a left excursion. This variable will be a hitting time from state $N^* - 1$ to N^*.
H^2_i	The length of the i-th if it is a right excursion. This variable will be a hitting time from state $N^* + 1$ to N^*.
D_i	A Bernoulli random variable that equals 1 if excursion i is a left excursion, meaning we move from N^* to $N^* - 1$ at the start of excursion i. $D_i = 0$ if we do a right excursion.
$H^1_0(s)$	The hitting time from initial state s to N^*, given that $s \leq N^*$. This gives the amount of time steps until we first reach N^*.
$H^2_0(s)$	The hitting time from initial state s to N^*, given that $s > N^*$. This gives the amount of time steps until we first reach N^*.

Table 5.1: The random variables that we will need to derive an equation for $Q'(s, a)$.

Before deriving an equation for $Q'(s,a)$, we define the following auxiliary random variables.

$$R_0(s) = \mathbb{1}(s \in S_1)\frac{r_1}{1-\gamma}(1-\gamma^{H_0^1(s)+1}) + \mathbb{1}(s \in S_2)\frac{r_2}{1-\gamma}(1-\gamma^{H_0^2(s)+1}), \tag{5.29}$$

$$L_0(s) = \mathbb{1}(s \in S_1)H_0^1(s) + \mathbb{1}(s \in S_2)H_0^2(s), \tag{5.30}$$

$$R_i = r_1 + \frac{\gamma}{1-\gamma}(D_i r_1(1-\gamma^{H_i^l+1}) + (1-D_i)r_2(1-\gamma^{H_i^r+1})), \quad \text{and} \tag{5.31}$$

$$L_i = 1 + D_i H_i^1 + (1-D_i)H_i^2. \tag{5.32}$$

Here $\mathbb{1}(\cdot)$ denotes the indicator function. $R_0(s)$ gives the initial reward before reaching N^* from s, $L_0(s)$ gives the amount of time steps until reaching N^* from s, R_i gives the reward in excursion i and L_i gives the length of excursion i. We then obtain the following result.

Proposition 5.3. *Let our MDP be given in Figure 5.1 with reward function as defined in Equation* (5.23)*. We apply the Q-learning update rule from Equation* (5.12) *on this MDP. We obtain the following result for* $Q'(s,a)$

$$Q'(s,a) = R_0(s) + \sum_{i=1}^{\infty} \gamma^{L_0(s)+\sum_{j=1}^{i-1} L_j} R_i \tag{5.33}$$

Proof. To obtain a formula for $Q'(s,a)$, we will first work out the reward we earn in the initial time until N^*, the first excursion and the second excursion. This is given in the equation below.

$$\begin{aligned}
Q'(s,a) = {} & \mathbb{1}(s \in S_1) r_1 \sum_{n=0}^{H_0^1(s)} \gamma^n + \mathbb{1}(s \in S_2) r_2 \sum_{n=0}^{H_0^2(s)} \gamma^n \\
& + r_1 + D_1 r_1 \sum_{n=1}^{H_1^1} \gamma^{L_0(s)+n} + (1-D_1) r_2 \sum_{n=1}^{H_1^2} \gamma^{L_0(s)+n} \\
& + r_1 + D_2 r_1 \sum_{n=1}^{H_2^1} \gamma^{L_0(s)+D_1 H_1^1 + (1-D_1)H_1^2 + 1 + n} + (1-D_2) r_2 \sum_{n=1}^{H_2^2} \gamma^{L_0(s)+D_1 H_1^1 + (1-D_1)H_1^2 + 1 + n} \\
& + \dots
\end{aligned} \tag{5.34}$$

The first line contains the initial reward until we reach N^*. If $s \leq N^*$ we have $H_0^1(s)$ steps with a reward of r_1. On the other hand, if $s > N^*$ we have $H_0^2(s)$ steps of reward r_2. Take also into account that each further reward term needs to be discounted by an additional factor γ.

After the initial reward, the first excursion starts. Since the excursion starts in $N^* \in S_1$, we always first obtain a reward of r_1. If $D_1 = 1$, this excursion will have H_1^1 steps of reward r_1 and if $D_1 = 0$ we have H_1^2 steps of reward r_2. Furthermore, we have to discount this excursion by the amount of time until we reach N^*, which equals $L_0(s)$. The third line gives the reward in the second excursion. This follows basically the same structure from excursion 1, but now we also discount by a factor of $\gamma^{D_1 H_1^1 + (1-D_1)H_1^2 + 1} = \gamma^{L_1}$.

These excursions will repeat indefinitely, with each excursion being discounted more than the last. To include infinitely many excursion, we will rewrite Equation (5.34) by observing that each individual reward sum is just a finite geometric sum. For these we know that $\sum_{n=0}^{k} r\gamma^n = r\frac{1-\gamma^{k+1}}{1-\gamma}$. Thus we can Equation (5.34) to

$$\begin{aligned}
Q'(s,a) = {} & \mathbb{1}(s \in S_1)\frac{1}{1-\gamma} r_1(1-\gamma^{H_0^1(s)+1}) + \mathbb{1}(s \in S_2)\frac{1}{1-\gamma} r_2(1-\gamma^{H_0^2(s)+1}) \\
& + \sum_{i=1}^{\infty} \gamma^{L_0(s)+\sum_{j=1}^{i-1} L_j} \big(r_1 + \frac{\gamma}{1-\gamma}(D_i r_1(1-\gamma^{H_i^l+1}) + (1-D_i)r_2(1-\gamma^{H_i^r+1}))\big) \\
= {} & R_0(s) + \sum_{i=1}^{\infty} \gamma^{L_0(s)+\sum_{j=1}^{i-1} L_j} R_i.
\end{aligned} \tag{5.35}$$

Thus, we have derived an equation for $Q'(s,a)$. □

Contribution 5.3. Proposition 5.3 gives us an explicit formula for $Q'(s,a)$. Thus, we can determine the value that Q-learning will converge to in terms of the random variables in Table 5.1, which in turn depend on the properties of our MDP.

5.3.3 Expectation of $Q'(s,a)$

In Proposition 5.3 we have obtained an explicit expression for $Q'(s,a)$ in terms of multiple random variables, most notably hitting time variables. Of course, the actual distribution of $Q'(s,a)$ will be quite complex. However, we can work out the expectation of $Q'(s,a)$, which is done in the following Lemma.

Lemma 5.4. *Let $Q'(s,a)$ be given by Equation (5.33) of Proposition 5.3. Then, we get that the expectation of $Q'(s,a)$ will equal*

$$\mathbb{E}[Q'(s,a)] = \mathbb{E}[R_0(s)] + \mathbb{E}[\gamma^{L_0(s)}]\frac{\mathbb{E}[R_i]}{1-\mathbb{E}[\gamma^{L_i}]}. \tag{5.36}$$

The expectations for the elements in this equation are given in Equations (5.38), (5.39), (5.41) and (5.42) in the proof.

Proof. We first take the expectation of Equation (5.33), leading to

$$\begin{aligned}\mathbb{E}[Q'(s,a)] &= \mathbb{E}[R_0(s)] + \sum_{i=1}^{\infty}\mathbb{E}[\gamma^{L_0(s)+\sum_{j=1}^{i-1}L_j}R_i],\\ &= \mathbb{E}[R_0(s)] + \sum_{i=1}^{\infty}\mathbb{E}[\gamma^{L_0(s)+\sum_{j=1}^{i-1}L_j}]\mathbb{E}[R_i].\end{aligned} \tag{5.37}$$

We are able to separate the expectations of $\gamma^{L_0(s)+\sum_{j=1}^{i-1}L_j}$ and R_i, since every excursion both starts and ends in N^*, so the excursions are independent from each other. Thus, every variable defined per excursion $(D_i,\, H_i^1,\, H_i^2)$ is independent between excursions. Thus, we can work out each expectation separately. For the initial reward $R_0(s)$ we have

$$\begin{aligned}\mathbb{E}[R_0(s)] &= \mathbb{E}[\mathbb{1}(s\in S_1)\frac{r_1}{1-\gamma}(1-\gamma^{H_0^1(s)+1}) + \mathbb{1}(s\in S_2)\frac{r_2}{1-\gamma}(1-\gamma^{H_0^2(s)+1})],\\ &= \mathbb{1}(s\in S_1)\frac{r_1}{1-\gamma}(1-\mathbb{E}[\gamma^{H_0^1(s)+1}]) + \mathbb{1}(s\in S_2)\frac{r_2}{1-\gamma}(1-\mathbb{E}[\gamma^{H_0^2(s)+1}]).\end{aligned} \tag{5.38}$$

For the reward per excursion R_i we obtain

$$\begin{aligned}\mathbb{E}[R_i] &= \mathbb{E}\Big[r_1 + \frac{\gamma}{1-\gamma}\big(r_1D_i(1-\gamma^{H_i^1}) + r_2(1-D_i)(1-\gamma^{H_i^2})\big)\Big],\\ &= r_1 + \frac{\gamma}{1-\gamma}\big(r_1\mathbb{E}[D_i(1-\gamma^{H_i^1})] + r_2\mathbb{E}[(1-D_i)(1-\gamma^{H_i^2})]\big),\\ &= r_1 + \frac{\gamma}{1-\gamma}\big(r_1\mathbb{E}[D_i](1-\mathbb{E}[\gamma^{H_i^1}])) + r_2(1-\mathbb{E}[D_i])(1-\mathbb{E}[\gamma^{H_i^2}])\big).\end{aligned} \tag{5.39}$$

Finally, we need to calculate the discount factor of each excursion.

$$\mathbb{E}[\gamma^{L_0(s)+\sum_{j=1}^{i-1}L_j}] = \mathbb{E}[\gamma^{L_0(s)}\prod_{j=1}^{i-1}\gamma^{L_j}] = \mathbb{E}[\gamma^{L_0(s)}]\prod_{j=1}^{i-1}\mathbb{E}[\gamma^{L_j}] \tag{5.40}$$

Here we again used that the excursions are independent to separate the expectation. For the separate parts we have

$$\mathbb{E}[\gamma^{L_0(s)}] = \mathbb{E}[\gamma^{H_0^1(s)}]^{\mathbb{1}(s\in S_1)}\mathbb{E}[\gamma^{H_0^2(s)}]^{\mathbb{1}(s\in S_2)}, \tag{5.41}$$

$$\mathbb{E}[\gamma^{L_j}] = \mathbb{E}[\gamma^{1+D_jH_j^1+(1-D_j)H_j^2}] = \gamma P(D_j=1)\mathbb{E}[\gamma^{H_j^1}] + P(D_j=0)\mathbb{E}[\gamma^{H_j^2}]. \tag{5.42}$$

The variables H_i^1, H_i^2 and D_i that are each i.i.d. over the excursions. As such, $\mathbb{E}[R_1] = \mathbb{E}[R_2] = \ldots = \mathbb{E}[R_i] = \ldots$ and the same holds for for $L_1, L_2, \ldots$. Furthermore, we have that $0 \leq \gamma^{L_i} \leq 1$. Thus, the sum in Equation (5.37) forms a geometric series such that we obtain

$$\begin{aligned}\mathbb{E}[Q'(s,a)] &= \mathbb{E}[R_0(s)] + \mathbb{E}[\gamma^{L_0(s)}]\sum_{i=1}^{\infty}(\mathbb{E}[\gamma^{L_i}])^{i-1}\mathbb{E}[R_i],\\ &= \mathbb{E}[R_0(s)] + \mathbb{E}[\gamma^{L_0(s)}]\frac{\mathbb{E}[R_i]}{1-\mathbb{E}[\gamma^{L_i}]}.\end{aligned} \tag{5.43}$$

□

5.3.4 Bounds on $\mathbb{E}[Q'(s,a)]$

We can work out the expectation of Equation (5.36) in Lemma 5.4, provided that we are able to calculate the expectations of $\gamma^{H_i^1}$, $\gamma^{H_i^2}$, $\gamma^{H_0^1(s)}$ and $\gamma^{H_0^2(s)}$. Furthermore, we need to know the parameter of the Bernoulli variable D_i. Unfortunately, we don't know the exact distributions of these random variables. This is because these depend on the actions chosen in the maximization in our update rule in Equation (5.12). These actions determine which state-action pairs we visit in the path through the state space from which we count up the rewards. In the ideal case we would always pick the optimal action, but this is not something we can guarantee. Instead, the actions with which we update are determined by the target policy. Recall that this target policy simply chooses the action that maximizes the current value function in the visited state. This target policy can change over time and also it depends on the value functions.

Thus, we can not exactly calculate $\mathbb{E}[Q'(s,a)]$. What we can do is bound this quantity. For any policy we are able to determine the distributions of the hitting times. This means we are also able to evaluate Equation (5.36) by instead evaluating the random variables for the set of actions determined by the policy. To obtain bounds, we simply need to know which policies lead to upper and lower bounds. We define the random variables for our upper and lower bounds as follows.

Definition 5.3. The random variables $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ are defined to equal the r.h.s. of Equation (5.33). However, we will evaluate the random variables $H_0^1(s)$, $H_0^2(s)$, H_i^1, H_i^2 and D_i of this Equation for the best and worst policy for the cases of $Q^{(U)}(s,a)$ and $Q^{(U)}(s,a)$ respectively.

Remark 5.1. Since $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ are defined similarly to $Q'(s,a)$, the results from Lemma 5.4 will also apply to these random variables. Thus, we are able to evaluate their expectations to obtain lower and upper bounds on $\mathbb{E}[Q'(s,a)]$. In particular, we obtain

$$\mathbb{E}[Q^{(L)}(s,a)] \leq \mathbb{E}[Q'(s,a)] \leq \mathbb{E}[Q^{(U)}(s,a)]. \tag{5.44}$$

This follows directly from the definition above, since we have defined the random variables to take the worst and best policies.

For our MDP, we have assumed that $p_1 > p_2$ and $r_1 > r_2$. As a result, we want to move to the left of the state space as much as possible. This is achieved by picking action a_2, since this has a higher probability of transitioning one state to the left. Thus, for $Q^{(L)}(s,a)$ we will always pick action a_1, while for $Q^{(U)}(s,a)$ we will always pick a_2.

Contribution 5.4. In the previous section we derived an explicit formula for the expectation of $Q'(s,a)$ when applied on our MDP with the reward function of Equation (5.23). Thus we have shown we can determine the value that Q-learning will converge to on average using our method of analysis. However, we couldn't calculate $\mathbb{E}[Q'(s,a)]$ exactly, so we introduced $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ to determine lower and upper bounds instead. The convergence results of our algorithm don't require extra assumptions on the Q-learning algorithm, with the small exception that we used a slightly modified update rule. However, we have shown in Section 5.2.4 that our 'uniformized' update rule has similar asymptotic behaviour w.r.t. the standard Q-learning algorithm.

5.3.5 Hitting time distribution

To evaluate $\mathbb{E}[Q^L(s,a)]$ and $\mathbb{E}[Q^U(s,a)]$, we need to know the distributions of the hitting time variables. We denote by $H_{p_1}^1$ the hitting time from N^*-1 to N^* when we always choose action a_1. The hitting time from N^*-1 to N^* under action a_2 is denoted by $H_{p_2}^1$. The variables $H_{p_1}^2$ and $H_{p_2}^2$ are similarly defined as the hitting times from N^*+1 to N. Then, since $p_1 \geq p_2$, we have

$$E[H_{p_1}^1] \leq E[H^1] \leq E[H_{p_2}^1], \tag{5.45}$$

$$E[H_{p_2}^2] \leq E[H^2] \leq E[H_{p_1}^2]. \tag{5.46}$$

Here we ignored excursion index i for H_i^1 and H_i^2. However, we don't need to evaluate the expectation of the hitting times, but the expectation of the hitting times to the power γ. In other words, we want to evaluate the probability generating function (PGF) of the hitting time variables. The derivation for this PGF will be similar for $H_{p_1}^1$, $H_{p_2}^1$, $H_{p_1}^2$ and $H_{p_2}^2$. We will consider $H_{p_1}^1$, the hitting time from N^*-1 to N^* under action a_1. We will define $P_1 \in \mathbb{R}^{(N^*-1)\times(N^*-1)}$ as the transition matrix between the states left of N^*. These are transient, with N^* being the absorbing state. The vector $P_{1,0} \in \mathbb{R}^{N^*-1)}$ contains the absorption

probabilities of these states. For example, if $N^* = 5$ we obtain

$$P_1 = \begin{bmatrix} 1-p_1 & p_1 & 0 & 0 \\ 1-p_1 & 0 & p_1 & 0 \\ 0 & 1-p_1 & 0 & p_1 \\ 0 & 0 & 1-p_1 & 0 \end{bmatrix}, \tag{5.47}$$

and

$$P_{1,0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ p_1 \end{bmatrix}. \tag{5.48}$$

Furthermore, it holds for $P_1 e + P_{1,0} = e$, where e is the ones vector of appropriate length.

Now we will calculate the distribution of the hitting times. At time t, $(P_1^t)_{ij}$ gives the probability of being in state j after t time steps when starting in state i. Since we consider a transient Markov chain, the sum $\sum_j (P_1^t)_{ij}$ gives the probability of not having absorbed until time step t. Using ones vector e, we get that $P_1^t e$ gives the vector of probabilities of not absorbing. By left multiplying by $\vec{s}_0$ we obtain the cumulative probability of not absorbing by time t, given a starting state. Thus, the probability to absorb by time t is given by

$$F_1(t) = 1 - \vec{s}_0 P_1^t e. \tag{5.49}$$

To absorb in time step t, we must not have absorbed until time step $t-1$, but also transition to the absorbing state in time step t. This probability is described in the following equation

$$f_1(t) = \vec{s}_0 P_1^{t-1} P_{1,0}. \tag{5.50}$$

To evaluate the PGF of $H^1_{p_1}$, we will use the probability mass function from this lemma. We obtain that

$$\begin{aligned} \mathbb{E}[\gamma^{H^1_{p_1}}] &= \sum_{t=1}^{\infty} f_1(t)\gamma^t \\ &= \sum_{t=1}^{\infty} \vec{s}_0 P_1^{t-1} P_{1,0} \gamma^t \\ &= \gamma \vec{s}_0 \big(\sum_{t=1}^{\infty} \gamma^{t-1} P_1^{t-1} \big) P_{1,0} \\ &= \gamma \vec{s}_0 \big(\sum_{t=0}^{\infty} (\gamma P_1)^t \big) P_{1,0} \\ &= \gamma \vec{s}_0 (I - \gamma P_1)^{-1} P_{1,0}. \end{aligned} \tag{5.51}$$

By evaluating these for each of the hitting times we can calculate $\mathbb{E}[Q^{(L)}(s,a)]$ and $\mathbb{E}[Q^{(U)}(s,a)]$.

5.4 Three-part Markov proces

In the previous sections we have obtained closed-form expressions for an expected upper bound and an expected lower bound on $\mathbb{E}[Q'(s,a)]$, where $Q'(s,a)$ is of course the limiting distribution of the Q-learning algorithm applied on our MDP. These were obtained for a fairly simple MDP, namely one with only 2 different rewards on different parts of the state space. We would of course like to extend our results to more complex cases, to obtain a more general result. In this section we will look at a three-part Markov Decision Process. This means that we split up our process into three parts, each with a different reward. While this may still seem very specific, it is actually quite a big step from the two-part case. This is because we now have multiple states at which we switch rewards. Once we are able to obtain bounds for a three-part Markov chain, we should in principle be able to extend this result with any amount of parts.

For this Markov process, we will divide our state space in three parts: $S_1 := 1 \ldots, N_1$, $S_2 := N_1 + 1 \ldots, N_2$ and $S_3 := N_2 + 1 \ldots, N$. Each of these parts has a different reward. In particular we have

$$r(s,a) = \begin{cases} r_1 \text{ if } & s \in S_1 \\ r_2 \text{ if } & s \in S_2 \\ r_3 \text{ if } & s \in S_3 \end{cases}. \tag{5.52}$$

This reward structure means that we switch rewards in N_1 and N_2. To bound $\mathbb{E}[Q'(s,a)]$ we will again calculate the discount factors and rewards per excursion. However, in this case we will start each excursion in the state N_1 and then calculate the earned reward and amount of time steps taken until we return to N_1. This process will of course repeat indefinitely, since we have a positive recurrent Markov chain (given that $0 < p_2 \leq p_1 < 1$). Every excursion will again be independent, since we both start and end each excursion in N_1. The added difficulty is that now if we move to the right of N_1 we can reach N_2, leading to rewards of either r_2 or r_3. To find the total return, we will need to calculate the following things.

1. Determine initial reward and number of time steps until we reach the state N_1 from our initial state s.
2. Calculate the reward and length of each excursion. Again we use that the excursion are independent and follow a similar structure.
3. In each excursion, we can visit state N_2. We can calculate the reward we earn from state N_2 by considering the amount of times we return to N_2 before reaching N_1. After we eventually do reach N_1, the excursion will end. Subsequent visits to N_2 in a single excursion are again independent and they will follow a similar structure.

5.4.1 Random variables

Now that we have defined our MDP and the approach we take to finding the total return, we need to define the relevant random variables. These will mostly be similar than in the two-part case, but now there will be more since our MDP is more complex. We add one new type of random variable, namely Bernoulli variables that determine the probability of transitioning from N_1 to N_2 or vice versa. We will again use the i to index the variables per excursion. However, now we will also use the index l to denote the l-th visit to N_2 in a single excursion. We will only include the index l when it is relevant, i.e. for variables that relate to state N_2. In Table 5.2 the random variables are given.

Variable	**Definition**
H_i^1	This variable denotes the hitting time from state $N_1 - 1$ to N_1 for excursion i. In other words, it gives the excursion length if we move to the left of N_1 in excursion i.
$H_i^{2,l}$	This denotes the time to hit either N_1 or N_2 from state $N_1 + 1$.
$H_{i,l}^{2,r}$	Gives the time to hit N_1 or N_2 when starting in $N_2 - 1$, given that we are in excursion i and visit l to N_2.
$H_{i,l}^3$	Gives the first passage time to return to N_2 after moving right to state $N_2 + 1$ in visit l to N_2 in excursion i.
D_{i,N_1}	Bernoulli variable that equals 1 if we move to the right in state N_1 at the start of excursion i. $D_{i,N_1} = 0$ corresponds to moving left.
D_{i,l,N_2}	Similarly defined as D_{i,N_1}, but now for moving left or right from state N_2 in visit l to N_2 during excursion i.
F_{i,N_1}	Bernoulli variable that determines if we transition to N_2 after moving right from N_1. $F_{i,N_1} = 1$ if we return to N_1, while it equals 0 if we reach N_2.
F_{i,l,N_2}	Determines if we reach N_1 after moving to the left of N_2 in excursion i after l visits to N_2. F_{i,l,N_2} equals 1 if we return to N_2 and 0 if we reach N_1.
$H_0^1(s)$	Gives the number of time steps to reach N_1 when $s \in S_1$.
$H_0^2(s)$	Gives the number of time steps to reach either N_1 or N_2 when $s \in S_2$.
$H_0^3(s)$	Gives the number of time steps to reach N_2 when $s \in S_3$.
$F_0(s)$	Determines if we reach N_1 or N_2 first when starting in state s in the case that $s \in S_2$. $F_0(s) = 0$ if we reach N_1 first and $F_0(s) = 1$ if we reach N_2 first.

Table 5.2: The random variables that we will need to derive an equation for $Q'(s,a)$ for a three-part MDP.

Each of these variables is i.i.d. between the different excursions and visits. As such, we only need to calculate the amount of time steps and rewards for a single excursion or visit to N_2. We will use these to derive an equation for $Q'(s,a)$. We will subsequently work out its expectation $\mathbb{E}[Q'(s,a)]$. Just like in the case of of a two-part MDP, we will again not be able to calculate this expectation exactly. The random variables $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ will again be defined as in Definition 5.3. While we will perform the derivation for $Q'(s,a)$, the obtained equations will also hold for $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ by changing which actions are chosen. As such, we will calculate lower and upper bounds.

5.4.2 Total return

Using these random variables defined in Table 5.2 we can again derive a formula for $Q'(s,a)$. This formula is given in the following proposition.

Proposition 5.5. *Let our MDP be given as in Figure 5.1 with reward function defined as in Equation* (5.52)*. We apply the update rule of Equation* (5.12) *on this MDP. Then, we obtain the following formula for* $Q'(s,a)$*.*

$$Q'(s,a) = R_0(s) + \sum_{i=1}^{\infty} \gamma^{L_0(s)+\sum_{j=1}^{i-1} L_j} R_i. \tag{5.53}$$

The variables in this equation will be given in Equations (5.59)*,* (5.60)*,* (5.57) *and* (5.54) *in this proof respectively.*

Proof. First, we will calculate the reward R_i per excursion. When we leave N_1 at the start of excursion i, three things can happen.

1. We move to the left of N_1 in the case that $D_{i,N_1} = 0$. As a result, we will be in section S_1 and we will eventually return to N_1 after H_i^1 time steps. This leads to the start of the next excursion.
2. We move to the right of N_1, meaning that $D_{i,N_1} = 1$, but we return to N_1 ($F_{i,N_1} = 1$) after $H_i^{2,l}$ time steps. This also leads to the start of the next excursion.
3. We move to the right of N_1 and we reach N_2 ($F_{i,N_1} = 0$) after $H_i^{2,l}$ time steps. Now, we have to calculate the reward and time before we return to N_1 from N_2.

The reward R_i is given in the equation below by taking these 3 cases in mind. We additionally have the term r_1 in front, because N_1 has this reward.

$$R_i = r_1 + (1 - D_{i,N_1})\frac{1-\gamma^{H_i^1}}{1-\gamma}\gamma r_1 + D_{i,N_1}F_{i,N_1}\frac{1-\gamma^{H_i^{2,l}}}{1-\gamma}\gamma r_2 + D_{i,N_1}(1-F_{i,N_1})\big(\frac{1-\gamma^{H_i^{2,l}}}{1-\gamma}\gamma r_2 + \gamma^{H_i^{2,l}} R_{i,1}^*\big) \tag{5.54}$$

The final part $R_{i,1}^*$ gives the total reward between reaching state N_2 and returning to N_1. We index it with $l = 1$ to make it clear that it is the first visit to N_2. In every visit to N_2, we again distinguish three cases.

1. We move to the left of N_2 and reach N_1 ($F_{i,l,N_2} = 0$). In this case the excursion will end so we don't need to add additional rewards.
2. We move to the left of N_2 but return to N_2 ($F_{i,l,N_2} = 1$). In this case we visit N_2 an additional time and we again need to add the reward and time until we return to N_1.
3. We move to the right of N_2, guaranteeing another visit to N_2. Again we need to add the reward until we return to N_1 from N_2.

For excursion i and visit l we obtain

$$\begin{aligned} R_{i,l}^* = {} & r_2 + (1 - D_{i,l,N_2})(1 - F_{i,l,N_2})\frac{1-\gamma^{H_{i,l}^{2,r}}}{1-\gamma}\gamma r_2 \\ & + (1 - D_{i,l,N_2})F_{i,N_2}\big(\frac{1-\gamma^{H_{i,l}^{2,r}}}{1-\gamma}\gamma r_2 + \gamma^{H_{i,l}^{2,r}} R_{i,l+1}^*\big) + D_{i,l,N_2}\big(\frac{1-\gamma^{H_{i,l}^{3}}}{1-\gamma}\gamma r_3 + \gamma^{H_{i,l}^{3}} R_{i,l+1}^*\big). \end{aligned} \tag{5.55}$$

Because each of these variables is i.i.d. between different excursions and visits to N_2, we have that the variables $R_{i,l}^*$ are also i.i.d. This will be helpful when we work out the expectations of these quantities.

We have obtained the reward per excursion R_i, now we will work out the discount factor per excursion. To do this, we need to calculate the length of the initial time before reaching N_1, given by $L_0(s)$, and the length of an excursion L_i. The amount of time steps before we start excursion i is then given by

$$L_0(s) + \sum_{j=1}^{i-1} L_j \tag{5.56}$$

We will deal with $L_0(s)$ a bit later, but we will calculate L_j here. Since each excursion follows the same structure, the L_j variables will be i.i.d. For L_j we again look at the 3 cases as we did for the reward per excursion. We then get

$$L_j = 1 + (1 - D_{j,N_1})H_j^1 + D_{j,N_1}F_{j,N_1}H_j^{2,l} + D_{j,N_1}(1-F_{j,N_1})(H_j^{2,l} + L_{j,1}^*). \tag{5.57}$$

The factor $L^*_{j,1}$ now gives the amount of time steps until we return to N_1 from N_2. We can again calculate this quantity separately.

$$L^*_{j,l} = 1 + (1 - D_{j,l,N_2})(1 - F_{j,l,N_2})H^{2,r}_{j,l} + (1 - D_{j,l,N_2})F_{j,l,N_2}(H^{2,r}_{j,l} + L^*_{j,l+1}) + D_{j,l,N_2}(H^3_{j,l} + L^*_{j,l+1}) \tag{5.58}$$

Finally, we of course have the initial reward $R_0(s)$, and also the initial time until we hit N_1 $L_0(s)$. From initial state s we can either reach N_1 first, starting the first excursion, or we can reach N_2 first. In this case, we can simply add the reward and time to reach N_1 from N_2 as defined in Equations (5.55) and (5.58). We will use the variable $F_0(s)$ to determine if we either reach N_1 or N_2 if $s \in S_2$. For the initial reward we obtain

$$\begin{aligned} R_0(s) = {} & \mathbb{1}(s \in S_1)\frac{1 - \gamma^{H^1_0(s)}}{1 - \gamma} r_1 + \mathbb{1}(s \in S_2)\frac{1 - \gamma^{H^2_0(s)}}{1 - \gamma} r_2 + \mathbb{1}(s \in S_3)\frac{1 - \gamma^{H^3_0(s)}}{1 - \gamma} r_3 \\ & + R^*_{0,1}\big(\mathbb{1}(s \in S_2)F_0(s)\gamma^{H^2_0(s)} + \mathbb{1}(s \in S_3)\gamma^{H^3_0(s)}\big) \end{aligned} \tag{5.59}$$

The initial time to hit N_1 is given by

$$L_0(s) = \mathbb{1}(s \in \{1, \dots, N_1\})H^1_0(s) + \mathbb{1}(s \in \{N_1 + 1, \dots, N_2\})H^2_0(s) + \mathbb{1}(s \in \{N_2 + 1, \dots, N\})H^3_0(s) \tag{5.60}$$

$$+ L^*_{0,1}\big(F_0(s)\mathbb{1}(s \in \{N_1 + 1, \dots, N_2\}) + \mathbb{1}(s \in \{N_2 + 1, \dots, N\})\}\big). \tag{5.61}$$

The relevant variables R_i, L_i, $R_0(s)$ and $L_0(s)$ from Equations (5.54), (5.57), (5.59) and (5.60) are finally combined to give us our result.

$$Q'(s, a) = R_0(s) + \sum_{i=1}^{\infty} \gamma^{L_0(s) + \sum_{j=1}^{i-1} L_j} R_i \tag{5.62}$$

□

5.4.3 Expectation of $Q'(s, a)$

Now that we have again obtained a formula for $Q'(s, a)$, we can work out its expectation. This expectation will be given in the following lemma.

Lemma 5.6. *Let $Q'(s, a)$ be given by Equation (5.53) of Proposition 5.5. Then, we get that the expectation of $Q'(s, a)$ will equal*

$$\mathbb{E}[Q'(s, a)] = \mathbb{E}[R_0(s)] + \mathbb{E}[\gamma^{L_0(s)}]\frac{\mathbb{E}[R_i]}{1 - \mathbb{E}[\gamma^{L_i}]}. \tag{5.63}$$

The expectations for the elements in this equation are given in Equations (5.65), (5.67), (5.69) and (5.70) in the proof of this lemma.

Proof. First of all, we simply take the expectation of Equation (5.53)

$$\mathbb{E}[Q'(s, a)] = \mathbb{E}[R_0(s)] + \sum_{i=1}^{\infty} \mathbb{E}[\gamma^{L_0(s) + \sum_{j=1}^{i-1} L_j}]\mathbb{E}[R_i] \tag{5.64}$$

We will work out each expectation separately, starting with the reward per excursion.

$$\begin{aligned} \mathbb{E}[R_i] &= \mathbb{E}\Big[r_1 + (1 - D_{i,N_1})\frac{1 - \gamma^{H^1_i}}{1 - \gamma}\gamma r_1 + D_{i,N_1}F_{i,N_1}\frac{1 - \gamma^{H^{2,l}_i}}{1 - \gamma}\gamma r_2 + D_{i,N_1}(1 - F_{i,N_1})\big(\frac{1 - \gamma^{H^{2,l}_i}}{1 - \gamma}\gamma r_2 + \gamma^{H^{2,l}_i} R^*_{i,1}\big)\Big] \\ &= r_1 + (1 - \mathbb{E}[D_{i,N_1}])\frac{1 - \mathbb{E}[\gamma^{H^1_i}]}{1 - \gamma}\gamma r_1 + \mathbb{E}[D_{i,N_1}]\frac{1 - \mathbb{E}[\gamma^{H^{2,l}_i}]}{1 - \gamma}\gamma r_2 + \mathbb{E}[D_{i,N_1}](\mathbb{E}[(1 - F_{i,N_1})\gamma^{H^{2,l}_i}])\mathbb{E}[R^*_{i,1}]. \end{aligned} \tag{5.65}$$

To evaluate this expression, we need to calculate the expectation of $\mathbb{E}[R^*_{i,l}]$. We obtain

$$\begin{aligned}\mathbb{E}[R^*_{i,l}] &= \mathbb{E}\Big[r_2 + (1-D_{i,l,N_2})(1-F_{i,l,N_2})\frac{1-\gamma^{H^{2,r}_{i,l}}}{1-\gamma}\gamma r_2 \\ &+ (1-D_{i,l,N_2})F_{i,l,N_2}\big(\frac{1-\gamma^{H^{2,r}_{i,l}}}{1-\gamma}\gamma r_2 + \gamma^{H^{2,l}_{i,l}}R^*_{i,l+1}\big) + D_{i,l,N_2}\frac{1-\gamma^{H^3_{i,l}}}{1-\gamma}\gamma r_3 + \gamma^{H^3_{i,l}}R^*_{i,l+1}\Big] \\ &= r_2 + (1-\mathbb{E}[D_{i,l,N_2}])\frac{1-\mathbb{E}[\gamma^{H^{2,r}_{i,l}}]}{1-\gamma}r_2 \\ &+ (1-\mathbb{E}[D_{i,l,N_2}])E[F_{i,l,N_2}\gamma^{H^{2,r}_{i,l}}]\mathbb{E}[R^*_{i,l+1}] + \mathbb{E}[D_{i,l,N_2}](\frac{1-\mathbb{E}[\gamma^{H^3_{i,l}}]}{1-\gamma}r_3 + \mathbb{E}[\gamma^{H^3_{i,l}}]\mathbb{E}[R^*_{i,l+1}]) \qquad (5.66)\end{aligned}$$

We know that the variables $R^*_{i,l}$ are i.i.d., so we can solve this equation because $\mathbb{E}[R^*_{i,l}] = \mathbb{E}[R^*_{i,l+1}]$ and substitute it into $\mathbb{E}[R_i]$. Now let's move on to the discount factor per excursion. We need to evaluate

$$\mathbb{E}[\gamma^{L_0(s)+\sum_{j=1}^{i-1}L_j}] = \mathbb{E}[\gamma^{L_0}]\prod_{j=1}^{i-1}\mathbb{E}[\gamma^{L_j}].$$

Again it is used that the excursions are independent from each other. We will have to evaluate $\mathbb{E}[\gamma^{L_j}]$, which gives us the expected discount per excursion.

$$\begin{aligned}\mathbb{E}[\gamma^{L_j}] &= \mathbb{E}[\gamma^{1+(1-D_{j,N_1})H^1_j + D_{j,N_1}F_{j,N_1}H^{2,l}_j + D_{j,N_1}(1-F_{j,N_1})(H^{2,l}_j+L^*_j)}] \\ &= \gamma\mathbb{E}[\gamma^{(1-D_{j,N_1})H^1_j + D_{j,N_1}H^{2,l}_j + D_{j,N_1}(1-F_{j,N_1})L^*_j}] \\ &= \gamma\big(P(D_{j,N_1}=0)\mathbb{E}[\gamma^{H^1_j}] + P(D_{j,N_1}=1)E[\gamma^{H^{2,l}_j+(1-F_{j,N_1})L^*_j}]\big) \\ &= \gamma\big(P(D_{j,N_1}=0)\mathbb{E}[\gamma^{H^1_j}] + P(D_{j,N_1}=1)\cdot \\ &\big(P(F_{j,N_1}=0)\mathbb{E}[\gamma^{H^{2,l}_j}|F_{j,N_1}=0]\mathbb{E}[\gamma^{L^*_j}] + P(F_{j,N_1}=1)\mathbb{E}[\gamma^{H^{2,l}_j}|F_{j,N_1}=1]\big)\big). \qquad (5.67)\end{aligned}$$

We now need to evaluate $\mathbb{E}[\gamma^{L^*_{j,l}}]$.

$$\begin{aligned}\mathbb{E}[\gamma^{L^*_{j,l}}] &= \mathbb{E}[\gamma^{1+(1-D_{j,l,N_2})(1-F_{j,l,N_2})H^{2,r}_{j,l}+(1-D_{j,l,N_2})F_{j,l,N_2}(H^{2,r}_{j,l}+L*_{j,l+1})+D_{j,l,N_2}(H^3_{j,l}+L^*_{j,l+1})}] \\ &= \gamma P(D_{j,l,N_2}=0)(\mathbb{E}[\gamma^{H^{2,r}_{j,l}+F_{j,l,N_2}L^*_{j,l+1}}) + P(D_{j,l,N_2}=1)\mathbb{E}[\gamma^{H^3_{j,l}}]\mathbb{E}[\gamma^{L^*_{j,l+1}}] \\ &= \gamma P(D_{j,l,N_2}=0)(P(F_{j,l,N_2}=0)\mathbb{E}[\gamma^{H^{2,r}_{j,l}}|F_{j,l,N_2}=0] + P(F_{j,l,N_2}=1)\mathbb{E}[\gamma^{H^{2,r}_{j,l}}|F_{j,l,N_2}=1]\mathbb{E}[\gamma^{L^*_{j,l+1}}]) \\ &+ P(D_{j,l,N_2}=1)\mathbb{E}[\gamma^{H^3_{j,l}}]\mathbb{E}[\gamma^{L^*_{j,l+1}}] \qquad (5.68)\end{aligned}$$

And again we can solve for $\mathbb{E}[\gamma^{L^*_{j,l}}]$ using the fact that the variables are independent. Finally we have to find the expected initial reward $\mathbb{E}[R_0(s)]$ and the initial discount $\mathbb{E}[\gamma^{L_0}]$. The derivation will follow similarly, and we will only give the final equations.

$$\begin{aligned}\mathbb{E}[R_0(s)] &= \mathbb{1}(s\in S_1)\frac{1-\mathbb{E}[\gamma^{H^1_0(s)}]}{1-\gamma}r_1 + \mathbb{1}(s\in S_2)\frac{1-\mathbb{E}[\gamma^{H^2_0(s)}]}{1-\gamma}r_2 + \mathbb{1}(s\in S_3)\frac{1-\mathbb{E}[\gamma^{H^3_0(s)}]}{1-\gamma}r_3 \\ &+ \mathbb{E}[R^*_{0,1}](\mathbb{1}(s\in S_2)\mathbb{E}[F_0(s)]\mathbb{E}[\gamma^{H^2_0(s)}] + \mathbb{1}(s\in S_3)\mathbb{E}[\gamma^{H^3_0(s)}]) \qquad (5.69)\end{aligned}$$

For the initial discount we obtain

$$\begin{aligned}\mathbb{E}[\gamma^{L_0(s)}] &= \mathbb{E}[\gamma^{H^1_0(s)}]^{\mathbb{1}(s\in S_1)}\cdot\mathbb{E}[\gamma^{H^2_0(s)}]^{\mathbb{1}(s\in S_2)}\cdot\mathbb{E}[\gamma^{H^3_0(s)}]^{\mathbb{1}(s\in S_3)} \\ &\cdot(P(F_0(s)=0) + P(F_0(s)=1)\mathbb{E}[\gamma^{L^*_{0,1}}])^{\mathbb{1}(s\in S_2)}\cdot\mathbb{E}[\gamma^{L^*_{0,1}}]^{\mathbb{1}(s\in S_3)}. \qquad (5.70)\end{aligned}$$

We can put the powers outside of the expectation because they are simply indicator functions which always equal 0 or 1. For both cases the expectation will be the same with the power inside or out. Thus, we have obtained explicit formulas for $\mathbb{E}[R_i]$, $\mathbb{E}[L_i]$, $\mathbb{E}[R_0(s)]$ and $\mathbb{E}[L_0(s)]$ in Equations (5.65), (5.67), (5.69) and (5.70) respectively. Since these expectations are independent of excursion, we again obtain a geometric series similar to the case of the two-part MDP. As such, when we plug each of these equations in we obtain

$$\mathbb{E}[Q'(s,a)] = \mathbb{E}[R_0(s)] + \mathbb{E}[\gamma^{L_0(s)}]\mathbb{E}[R_i]\frac{1}{1-\mathbb{E}[\gamma^{L_i}]}, \qquad (5.71)$$

proving our result. □

Contribution 5.5. In the previous sections we have shown that we can also derive $\mathbb{E}[Q'(s,a)]$ for an MDP with different rewards on three parts of the state space. Thus, we have shown that our method is able to provide convergence results on Q-learning for an even larger class of methods.

5.4.4 Evaluating bounds and random variables

Of course, we again can't evaluate the expectation in Equation (5.71), so we will again obtain bounds. To obtain $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ we need to know which actions lead to a lower bound and which actions lead to an upper bound. Thus, we in fact need the worst and the best policy on our MDP. Unfortunately, this isn't as easy to determine, because we have multiple cases for r_1, r_2 and r_3.

1. First of all, the cases with $r_1 \geq r_2 \geq r_3$ and $r_3 \geq r_2 \geq r_1$ and will follow very similarly to the case of a two-part Markov Decision process. For the bounds we either always pick action a_1 or a_2 to move to the left or right side of the state-space as much as possible.
2. In the cases that $r_2 \geq r_3 \geq r_1$ and $r_2 \geq r_1 \geq r_1$ we always want to move to section S_2. In this case, we always pick action a_1 to move to the right if we are in S_1 and we always pick a_2 if we are in S_3. Within S_2 it is harder to know which action is best, since we want to stay in this section. There will be some state for which we pick a_1 if we are to the left of it, and a_2 if we are to the right of it. The lower bound will follow in a similar manner, only by picking actions that lead out of S_2
3. The final cases where $r_3 \geq r_1 \geq r_2$ and $r_1 \geq r_3 \geq r_2$ are more complex. We of course want to move to sections S_3 and S_1 respectively. So when $s \in S_3$ for the first case, we always want to move to the right. However in sections S_1 and S_2 we might want to move to the left in the case that r_2 is much lower than r_1. This is because we may discount the rewards a lot even if we reach S_3, so it may be beneficial to stay in S_1 to not have to move to S_2. There will be a state $s \in S_1 \cup S_2$ to the left of which we always pick a_2 to move left and to the right of which we always pick a_1. On the other hand, we choose the actions in the lower bound to stay in S_2 as long as possible, since it has the worst reward.

We have described how to obtain the policies that give the variables $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$. Now, we need to be able to evaluate them. First of all, we again need to evaluate the PGF of our hitting time variables. This can be done using Equation (5.51) in Section 5.3.5. Only for the hitting time variables for section S_2, such as $H_i^{2,l}$, do we need to be more careful because we have two absorbing states. To deal with this, we still use Equation (5.51) but now by allowing absorption from multiple states in the vector $P_{2,0}$. The Bernoulli variables D_{i,N_1} and D_{i,l,N_2} for the moving right or left in N_1 and N_2 are also easy to evaluate, their expectation just equals p_1 or p_2 under any policy.

However, we have included additional variables in the derivation of $Q'(s,a)$. First of all, F_{i,N_1} and F_{i,l,N_2} are Bernoulli variables which determine whether we absorb in state N_1 or N_2 from section S_2. These will thus give us absorption probabilities when we take their expectation. Calculating these is essentially the gamblers ruin problem, where we need to determine the probability of ruin or winning the game from any starting state. This problem has been well studied in the literature and we instead refer to Chapter 5 of the book by Kemeny, Snell and Knapp [7] for the calculation of these absorption probabilities.

Furthermore, these variables introduce another new quantity to evaluate, since the probability of absorption in either N_1 or N_2 is dependant on the hitting time variables. In particular, we have to evaluate expectations such as $\mathbb{E}[\gamma^{H_i^{2,l}}|F_{i,N_1}]$, which is the expected number of time steps raised to the power γ, given that we either go to N_1 and N_2. To evaluate this expectation, we need to find the distribution $P(H_i^{2,l} = h|F_{i,N_1})$. From Bayes' theorem we simply obtain that

$$P(H_i^{2,l} = h|F_{i,N_1}) = \frac{P(H_i^{2,l} = h, F_{i,N_1})}{P(F_{i,N_1})}. \tag{5.72}$$

We have shown how to calculate the denominator, since this is just the absorption probability. The joint probability is simply the probability that we absorb in either N_1 or N_2 in time step h. But this is something we have already calculated, namely in Equation (5.50). To calculate the probability that we absorb in N_1 in time step h, we simply calculate $f_2(h)$. The only difference is that the absorption vector $P_{2,0}$ will only include the probability of absorbing in N_1. As such, we can calculate the expectation of each of the terms that arose in the previous section.

5.5 Extension to general case

While the case of a Markov Decision process with three parts doesn't seem much more special than one with two parts, it does give us the tools to extend to a process with any number of parts. In this section

we will show how to do this extension. We won't give a full derivation like in the previous two cases, but we will show what equations need to be solved to obtain it. For some notation, we will let the state space of our MDP consist out of K parts. On part $k \in 1, \ldots, K$ we will have the reward r_k. The states at which we switch reward from r_k to r_{k+1} are denoted as N_k, with final state $N := N_K$. Just like in the case with a three-part MDP, we will look at excursions that start in N_1 and end in N_1. We will again calculate the reward and number of time steps per excursion.

We will obtain equations for the reward and length, which we denote by R_1 and L_1 respectively. To obtain the equations for the extension to the case of K parts, we again need to define our random variables. Each of these random variables will be defined for each of the switch states $1 \leq k \leq K-1$, expect for R_k and L_k as will be explained

Variable	Definition
R_k	The total reward earned before we return to N_1 when starting in state N_k. We require $2 \leq k \leq K-1$, since we have already defined R_1 as the reward per excursion.
R_k^*	The reward when starting in state N_k before we reach either N_{k-1}, N_k or N_{k+1} (if possible).
L_k	The amount of time steps until we return to state N_1 when starting in N_k. Also only defined for $2 \leq k \leq K-1$.
F_k^l	Bernoulli variable which equals 1 if we transition to N_{k-1} from N_k.
F_k^r	Bernoulli variable which equals 1 if we transition to N_{k+1} from N_k.
L_k^l	The amount of time steps when we transition from state N_k to state N_{k-1}.
L_k^r	The amount of time steps when we transition from state N_k to state N_{k+1}.
L_k^0	The amount of time steps until we return to N_k after starting in N_k.

Table 5.3: The random variables that we will need to derive an equation for $Q'(s,a)$ for an MDP where the state space consists of K parts.

Note that we have ignored the index per excursion in each of these random variables, since we again have that each excursion will follow a similar structure. In an excursion, we will make a path through the states N_k before we finally return to N_1 and start another excursion. We use the random variables of Table 5.3 to define the following system of equations to obtain R_1, the reward per excursion.

$$
\begin{aligned}
R_1 &= R_1^* + F_1^r \gamma^{L_1^r} R_2 \\
R_2 &= R_2^* + F_2^r \gamma^{L_2^r} R_3 + (1 - F_2^l - F_2^r)\gamma^{L_2^0} R_2 \\
R_k &= R_k^* + F_k^l \gamma^{L_k^l} R_{k-1} + F_k^r \gamma^{L_k^r} R_{k+1} + (1 - F_k^l - F_k^r)\gamma^{L_k^0} R_k \quad \text{for } k = 3, \ldots, K-2 \\
R_{K-1} &= R_{K-1}^* + F_{K-1}^l \gamma^{L_{K-1}^l} R_{K-2} + (1 - F_{K-1}^l)\gamma^{L_{K-1}^0} R_{K-1}
\end{aligned}
$$

We can solve this system recursively starting at R_{K-1}. We solve R_{K-2} using R_{K-1}, then we solve R_{K-3} using R_{K-2}, etc. In this system of equations we note a few things. First, at the end nodes N_1 and N_{K-1} we of course don't have variables for moving to the left and right respectively. Also note that for R_1 and R_2 we ignore the case where we move to N_1, because then the excursion ends.

We have a similar system of equations for the amount of time steps per excursion, which we need to calculate discount factors. Now we will denote by L_1 the number of time steps in an excursion. We get the following system.

$$
\begin{aligned}
L_1 &= F_1^r(L_1^r + L_2) + (1 - F_1^r)L_1^0 \\
L_2 &= F_2^l L_2^l + F_2^r(L_2^r + L^3) + (1 - F_2^l - F_2^r)(L_2^0 + L_2) \\
L_k &= F_k^l(L_k^l + L_{k-1}) + F_k^r(L_k^r + L^{k+1}) + (1 - F_k^l - F_k^r)(L_k^0 + L_k) \quad \text{for } k = 3, \ldots, K-2 \\
L_{K-1} &= F_{K-1}^l(L_{K_1}^l + L_{K-2}) + (1 - F_{K-1}^l)(L_{K-1}^0 + L_{K-1})
\end{aligned}
$$

This can again be solved for L_1. So now we have obtained the reward per excursion and the time steps per excursion. We only need the initial reward and amount of time steps before reaching N_1 now. We won't give equations for these cases, but we will mention the procedure to obtain them. We have two cases for s, either s is one of our $K-1$ nodes, or s is in one of the K sections.

1. If s is at node N_k with $k \neq 1$, the initial reward is given by R_k and the initial time steps are given by L_k. If $k = 1$, then the initial reward an time equals 0.
2. If s is not at a node, we need to calculate the reward and time steps until a node N_k is reached. If $k \neq 1$, we again add R_k to the initial reward and L_k to the initial time steps. Otherwise we are done since we reached N_1 and the first excursion will start.

Using these methods we can again obtain an expression for $Q'(s,a)$ and calculate its expectation. Subsequently, upper and lower bounds can again be obtained.

Remark 5.2. For a Markov decision process with K parts we require that each section between two switch states contains at least 1 state. In particular, we require that $N_{k+1} \geq N_k + 1$ for $k = 1, \ldots, K-1$ and $N_1 \geq 2$. In our method we calculate the time before returning to N_k or transitioning to another state N_{k-1} or N_{k+1} using hitting times. We can only calculate these if each section outside of the switch states contains at least 1 state. This requirement also holds for the cases we already looked at involving a two and three-part state space. For instance, we require at least one state to the left and right N^* for the two-part case.

Contribution 5.6. In our final contribution of this chapter we show that our method of analyzing Q-learning can be extended to a large class of birth-type Markov decision processes. This shows that our method isn't just restricted to the cases where we have two or three sections in our state space with different rewards. We did use those cases to derive the systems of equations introduced in this section, which must be solved to obtain $Q'(s,a)$. Thus, our method is able to provide convergence results for a large class of birth-death type processes.

Chapter 6

Numerical studies

In the last chapter we analyzed the convergence of Q-learning for multiple classes of Markov decision processes. For the constant reward case we proved exact convergence, while for the case with K different rewards we only set up the Equations. For the case of a two and three-part MDP we obtained bounds on $\mathbb{E}[Q'(s,a)]$. These upper and lower bounds will be the expectations of $Q^{(U)}(s,a)$ and $Q^{(L)}(s,a)$ respectively. In this chapter, we will numerically evaluate these bounds. To evaluate these bounds, we will look at two error measures: the absolute error and relative error. These are defined below.

$$\text{absolute error}(p_1,p_2) = \left|\mathbb{E}[Q^{(U)}(s,a)] - \mathbb{E}[Q^{(L)}(s,a)]\right| \tag{6.1}$$

$$\text{relative error}(p_1,p_2) = \left|\frac{\mathbb{E}[Q^{(U)}(s,a)] - \mathbb{E}[Q^{(L)}(s,a)]}{\mathbb{E}[Q^{(U)}(s,a)]}\right|. \tag{6.2}$$

The goal of the numerical studies is to first evaluate these bounds, but also to see how they change when we vary the properties of our MDP.

6.1 Two-part Markov chain

First we will consider MDPs where the state-space consists of two parts, the first part with reward r_1 and the second part with reward r_2. We had already assumed that $r_1 > r_2$ in the last chapter. We don't have to look into the other case due to symmetry of the MDP. We will examine the bounds by varying the properties of our MDP.

1. We will first only vary the transition probabilities p_1 and p_2, while keeping the rest of the MDP the same.
2. Secondly, we will examine how the error bounds change as we vary both the amount of states N, switch state N^* and starting state s.
3. Finally, the influence of the reward terms is investigated by changing the reward per section and the discount factor γ.

We start with an MDP with $N = 10$ states and $N^* = 5$. Furthermore, the rewards are given by $r_1 = 2$ and $r_2 = 1$ with discount factor $\gamma = 0.9$. Finally, we choose our starting state to be $s = 3$. The absolute and relative error will then be functions of the transition probabilities p_1 and p_2. These are functions of two variables, so we will make surface plots for them. In Figures 6.1 and 6.2 the absolute and relative error are shown respectively, where we range the values of p_1 and p_2 between 0 to 1. We discard the values where $p_1 < p_2$, since we only consider the case of $p_1 \geq p_2$.

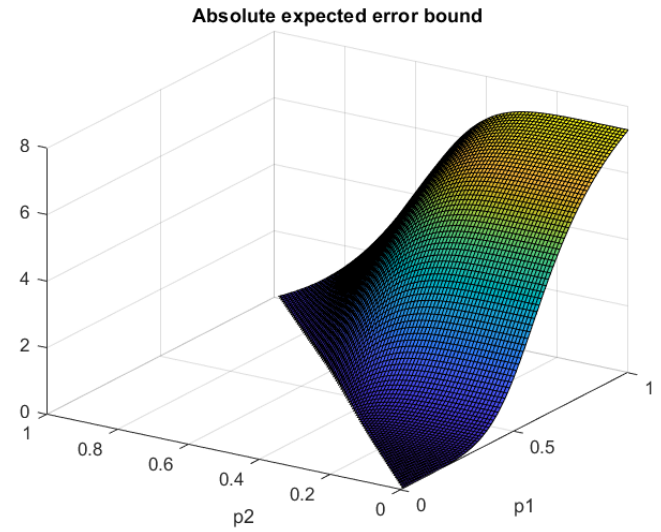


Figure 6.1: Absolute error for varying values of p_1 and p_2.

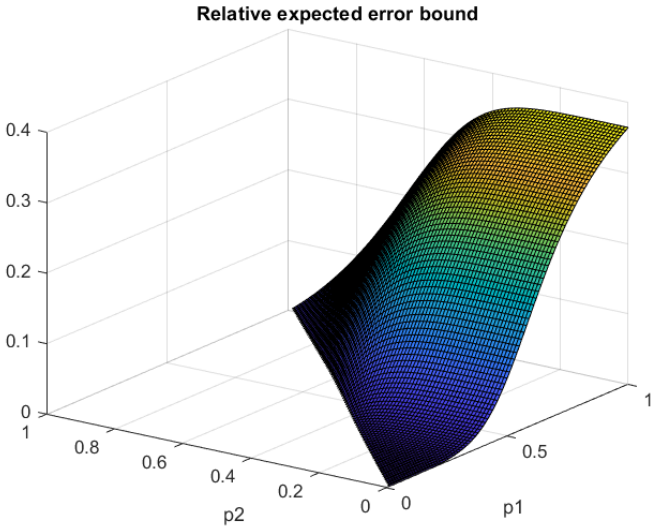


Figure 6.2: Relative error for varying values of p_1 and p_2.

The shape of both these surface plots is very similar, so we will just consider the relative error. It is apparent that the error becomes larger when the difference between p_1 and p_2 is large. It also seems that this increase is that fastest near the middle of the plot, where p_1 and p_2 are both near values of 0.5. To help us interpret this plot, we will make additional 2D graphs. These graphs will then be a single line of this surface plot. We will look at the case where we keep the difference between p_1 and p_2 constant. This allows us to see near what values the error is the largest. This is displayed in Figure 6.3.

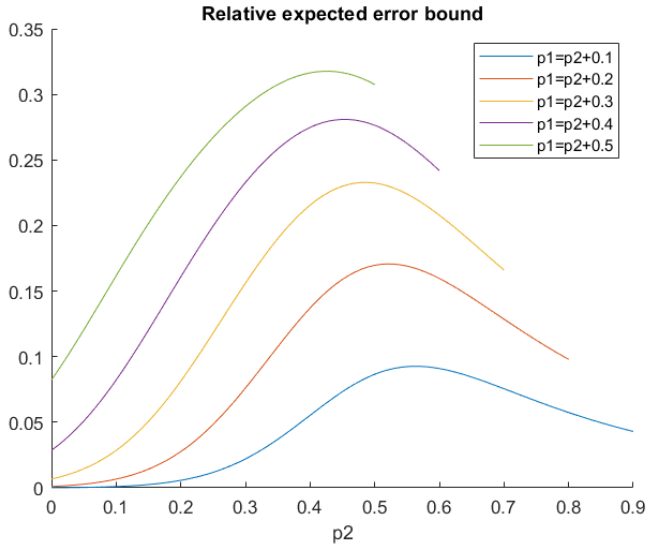


Figure 6.3: Relative error along multiple lines of a constant difference between p_1 and p_2.

The different graphs correspond to varying amount of differences between p_1 and p_2. For instance, the blue graph considers a difference of 0.1. These graphs correspond to diagonal lines in our surface plot, where each line gets closer and closer to the corner where $p_1 = 1$ and $p_2 = 0$. In this figure it is clear the error increases up to values of p_2 between 0.4 and 0.6, where we see a peak for each of the graphs. From this it is clear that our bounds are widest when p_1 and p_2 are near these middle values.

To explain this rapid change near values of 0.5, we can examine either $\mathbb{E}[Q^{(L)}(s,a)]$ or $\mathbb{E}[Q^{(U)}(s,a)]$. It doesn't matter which we pick, since they both follow from the same formula only using different transition probabilities. In Figure 6.4 we have plotted $\mathbb{E}[Q^{(L)}(s,a)]$.

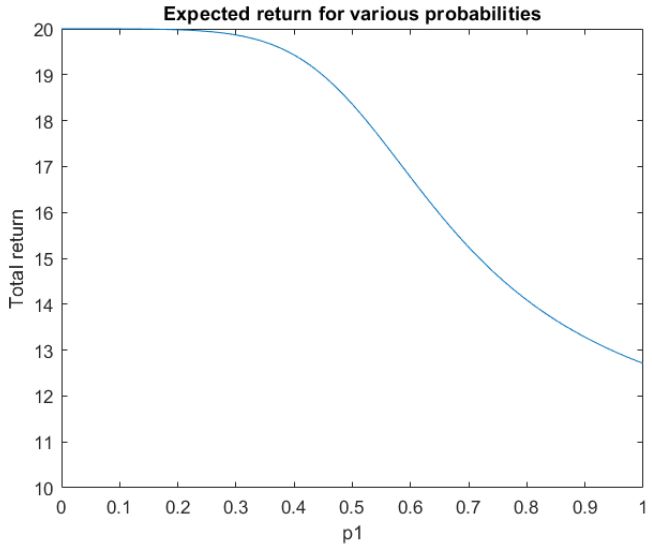


Figure 6.4: Total return $\mathbb{E}[Q^{(L)}(s,a)]$ for various values of p_1.

The total return in this figure decreases from 20 to just below 13 when p_1 goes from 0 to 1. The graph is the steepest near values of of 0.5 or 0.6 for p_1. Thus, when $p_1 > 0.5$ and $p_2 < 0.5$ we can expect a larger bound, which is what we have observed in Figure (6.3). The reason for this large decrease near the middle of the graph has to do with the hitting time variables. For our MDP, these hitting times quickly change between large and small values when the transition probability goes under or above 0.5. As a result, $\mathbb{E}[Q^{(L)}(s,a)]$ and $\mathbb{E}[Q^{(U)}(s,a)]$ also quickly change near these values.

Next we will vary the amount of state N, switch state N^* and starting state s in our MDP. We will examine 4 MDPs of varying sizes.

1. MDP 1 has $N = 10$ states, $N^* = 5$ and $s = 3$.
2. MDP 2 has $N = 20$ states, $N^* = 10$ and $s = 3$.
3. MDP 3 has $N = 50$ states, $N^* = 25$ and $s = 13$.
4. MDP 4 has $N = 100$ states, $N^* = 50$ and $s = 25$.

Note that we choose N^* to be in the middle of the state space and the starting state to be about a quarter into the state space. We could make surface plots of the relative error for each of these MDP's, but then it will be hard to compare. Instead, we will consider a constant difference of $p1 - p2 = 0.2$, corresponding to a diagonal line in the surface plot. In Figure 6.5 we have plotted the resulting relative error for each MDP.

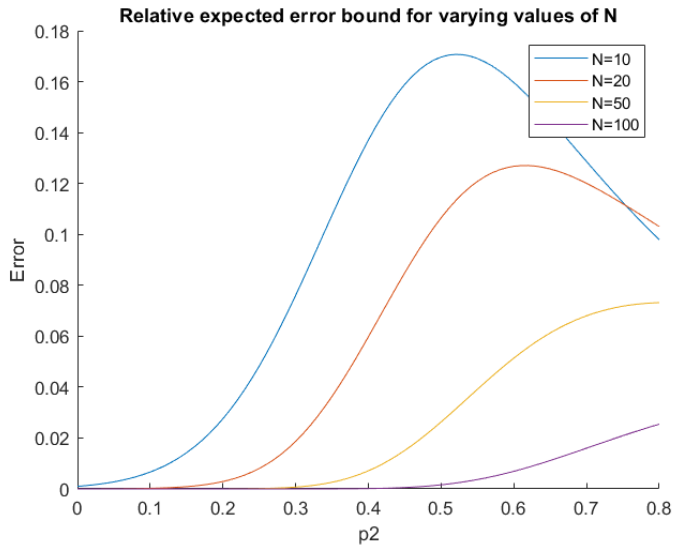


Figure 6.5: Relative error for MDPs of sizes 10, 20, 50 and 100 states where we vary p_1 and p_2 requiring $p_1 = p_2 + 0.2$.

We see that for a larger state space, the overall relative error tends to go down. This is logical, since we need more time steps to reach the right part of our state space. More surprisingly, it can be noticed that the largest relative error seems to happen for larger values of p_1 when the state space becomes larger. An explanation could be that since more time is needed to reach the right side of the state space, we only see differences when we reach this side fast, i.e. when p_1 and p_2 are close to 1.

Finally, we will change the reward and discount factor in our MDP. We will keep $r_2 = 1$, but change the value of r_1 to look at different ratios of reward. We will again choose the MDP to have $N = 10$ states, with switch state $N^* = 5$ and starting state $s = 3$. For the changing rewards we will keep the discount factor at $\gamma = 0.9$. Finally, we again look at a constant difference between p_1 and p_2 of 0.2, such that we don't have to make a surface plot.

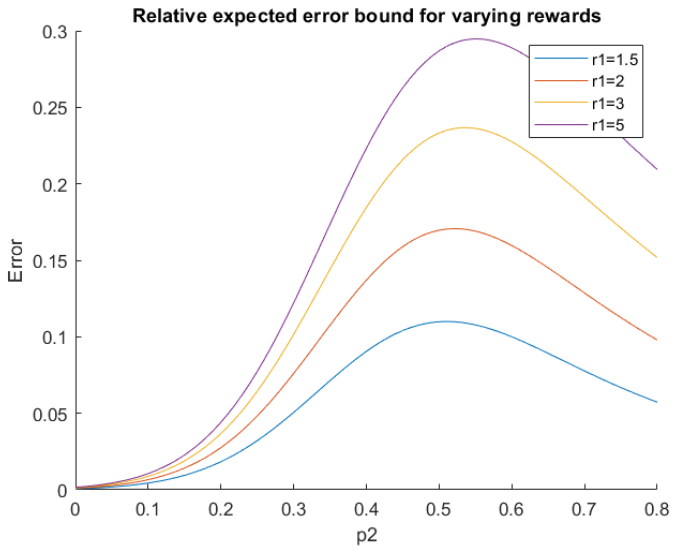


Figure 6.6: Relative error for MDPs with varying values for r_1, while keeping $r_2 = 1$. Furthermore, we look at a constant difference of $p_1 = p_2 + 0.2$ between the transition probabilities.

In Figure 6.6 it is clear that for a larger ratio of rewards, we also obtain a larger error. It is interesting to notice that the shape of the graphs are almost exactly the same, only being scaled up as we increase r_1. The reason for this is that r_1 and r_2 are constant factors in the derivation of $\mathbb{E}[Q'(s, a)]$ and don't influence the probability distributions. The results of this figure make a lot of intuitive sense, since if the difference between the obtainable rewards is larger, we also expect that the worst and best policies are farther apart, meaning wider bounds. In our final figure we will vary the discount factors, while keeping the rest of the MDP the same. For the rewards we will again set $r_1 = 2$ and $r_2 = 1$.

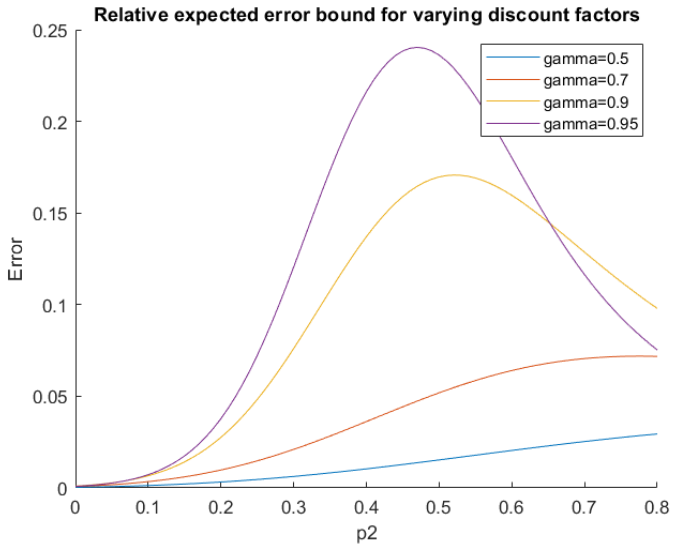


Figure 6.7: Relative error for MDPs with varying discount factors. We again consider a constant difference of $p_1 = p_2 + 0.2$ between the transition probabilities.

Figure 6.7 leads to more interesting results than the case of varying rewards. In this figure we see that for a smaller discount factor, we generally also have a smaller error. This is logical, since $Q^{(L)}(s, a)$ and $Q^{(U)}(s, a)$ both first earn reward r_1, so they will be closer together if we weigh future rewards are weighted less.

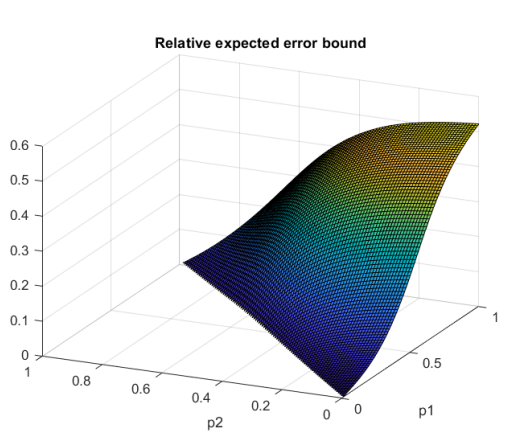
As the discount factor increases, the largest error occurs for different values of the transition probability. For $\gamma = 0.5$ the largest error is made right at the end for $p_2 = 0.8$ and $p_1 = 1$, while for $\gamma = 0.95$ the largest error occurs around $p_2 = 0.5$. We can explain why this happens as follows. If it takes long to reach N^* from our starting state s, meaning that the transition probabilities are smaller, then the rewards after reaching N^* will be discounted a lot and won't make a large difference. Only when we reach N^* fast can

we see a larger difference between $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$, since then the rewards after N^* will be taken into account. For this to happen, we want large transition probabilities. When the discount factor is larger, the rewards after we reach N^* will be discounted less. What will then play a bigger role is how the hitting times change with regards to the transition probabilities. We have already seen in Figure 6.4 that this occurs when the transition probabilities are near 0.5. Thus, for larger discount values we also see larger errors near $p_2 = 0.5$.

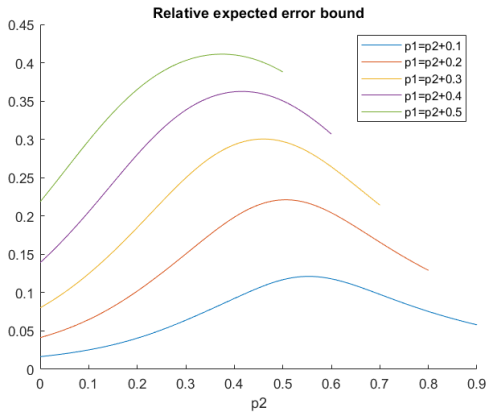
6.2 Three-part Markov chain

Now we move on to the case of a three-part MDP. In Section 5.4.4 we have seen that in this case there are more variations of MDPs to consider due to the added complexity of our reward function. In particular, we distinguished three cases in that section. For each of them we will examine the relative error, while varying the transition probabilities p_1 and p_2, for which we recall that $p_1 \geq p_2$. The other properties of our MDP will be set values. In particular, we let $N = 15$, $N_1 = 5$, $N_2 = 10$ and have starting state $s = 7$. Finally, we again use discount factor $\gamma = 0.9$.

For the first case we consider $r_1 \geq r_2 \geq r_3$. We could also look at $r_3 \geq r_2 \geq r_1$, but due to symmetry this will lead to similar results. For this case we set $r_1 = 3$, $r_2 = 2$ and $r_3 = 1$. In Figure 6.8a a surface plot of the relative error is given, while Figure 6.8b contains graphs of the relative error for a constant difference between p_1 and p_2.



(a) Surface plot of the relative error.

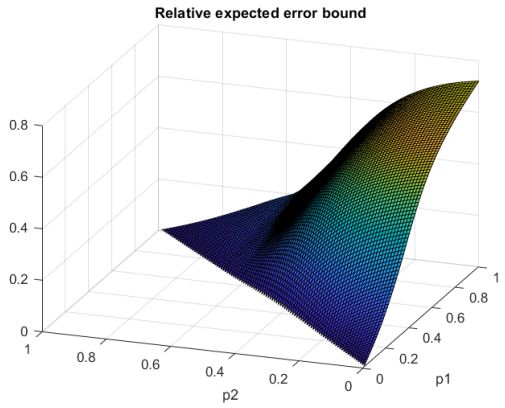


(b) Graphs of the relative error along lines of constant difference between p_1 and p_2.

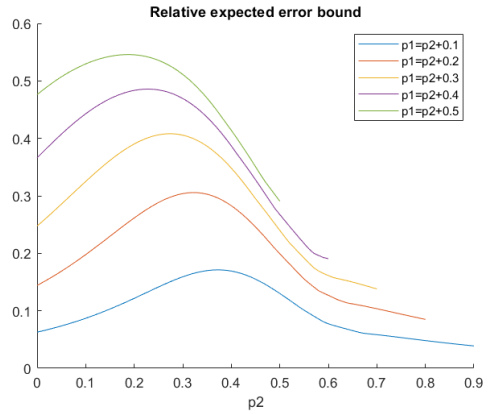
Figure 6.8: Relative error for the MDP where we require for the rewards that $r_1 \geq r_2 \geq r_3$.

In the left figure we see that the error increases as p_1 gets near to 1 and p_2 approaches 0. When looking at a constant difference between p_1 and p_2 in the right figure, we see that the error is the largest when the transition probabilities are near 0.5. These figures are very similar to the case of a two-part MDP we looked at in the previous section. In essence, the MDP we look at in these figures is very similar, we only include an extra reward step. Thus, these figures are as expected and similar arguments apply as to why the error is the largest near transition probabilities of 0.5.

For our second MDP, we define the rewards as follows: $r_1 = 1$, $r_2 = 3$ and $r_3 = 2$. Keeping the rest of the MDP the same, we obtain the following figures for the relative error.



(a) Surface plot of the relative error.

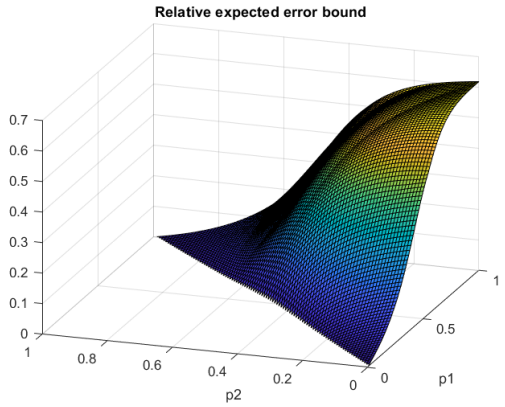


(b) Graphs of the relative error along lines of constant difference between p_1 and p_2.

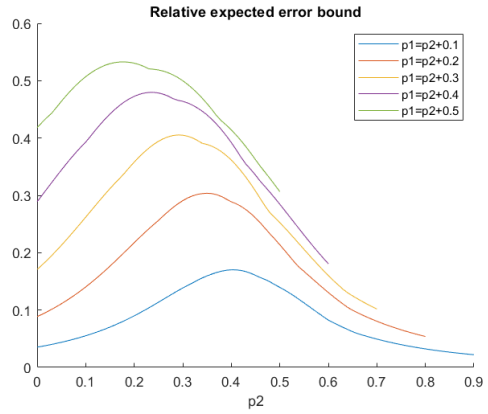
Figure 6.9: Relative error for the MDP where we require for the rewards that $r_2 \geq r_3 \geq r_1$.

In Figures 6.9a and 6.9b we have some clear differences with respect to the previous MDP. The error is larger for smaller transition probabilities, but decreases quicker for larger values. This makes sense if we look at our reward function. When p_1 and p_2 are small we spent more time in the left time of the state space, i.e. sections S_1 and S_2. When p_1 and p_2 are larger, we spent more time in sections S_2 and S_3. The difference in rewards r_1 and r_2 is larger than the difference between r_2 and r_3. Thus, for smaller transition probabilities we see a large difference in the reward earned. Thus, the difference in total return between $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ will be larger, leading to a larger error.

For our final MDP we let $r_1 = 3$, $r_2 = 1$ and $r_3 = 1$. The relative error for this MDP will be given in Figures 6.10a and 6.10b.



(a) Surface plot of the relative error.



(b) Graphs of the relative error along lines of constant difference between p_1 and p_2.

Figure 6.10: Relative error for the MDP where we require for the rewards that $r_1 \geq r_3 \geq r_2$.

These figures paint a very similar picture to the previous case. Again, the error is larger for smaller transition probabilities. This will again be caused by the fact that the difference $r_1 - r_2 = 2$ is larger than the difference $r_3 - r_2 = 1$. Thus, when we spent more time in the left side of the state space, we will have a larger error.

Of course, we would again like to examine what happens when we vary the properties of our MDP. However, we now have a lot more cases to deal with and we won't be able to examine each of them. From the cases that we looked at, we saw that the first MDP with $r_1 \geq r_2 \geq r_3$ was very similar to the two-part MDP. As such, it won't be that interesting to consider. The other two cases lead to very similar error functions, so we won't consider both of them. Thus, from now on we will analyze the MDP with reward $r_1 = 1$, $r_2 = 3$ and $r_3 = 2$.

For this MDP, we will vary the other properties in two ways. First we modify the amount of states N. We will always place the states N_1 and N_2 to be one third and two thirds into the state space respectively. After

this, we will also vary the starting state s to be in sections S_1, S_2 and S_3 respectively. We won't consider different values of the reward and discount factor. In the last section we already saw that these will lead to straightforward increases or decreases in the error. First let's consider MDPs with varying values of N. We will analyze the following three MDPs.

1. MDP 1 has 15 states, with $N_1 = 5$, $N_2 = 10$ and starting state $s = 7$.
2. MDP 2 has 30 states, with $N_1 = 10$, $N_2 = 20$ and starting state $s = 15$.
3. MDP 3 has 45 states, with $N_1 = 15$, $N_2 = 30$ and starting state $s = 22$.

The other properties of our MDP will be held the same. Furthermore, we won't consider the entire surface plot, but instead look at a constant difference of $p_1 = p_2 + 0.2$ of the transition probabilities. Then, the relative error for these MDPs is given in Figure 6.11.

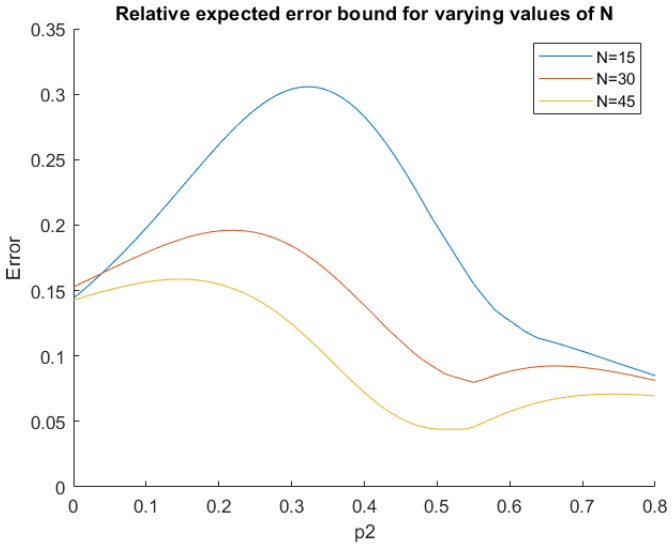


Figure 6.11: Relative error for MDPs with a varying amount of states N. We keep the difference in transition probabilities constant such that $p_1 = p_2 + 0.2$.

For larger MDPs, we see that the relative error goes down in general. This is logical, because we spend more time in the middle section before we reach N_1 or N_2. Thus, for both $Q^{(L)}(s, a)$ and $Q^{(U)}(s, a)$ we first earn a lot of rewards r_2. The more interesting thing to note in this figure is the decrease in error for p_2 near 0.5 for the cases of 30 and 45 states. When the transition probability is near 0.5, we stay in the middle section S_2 for a long time, because our starting state is also there. Thus, we won't reach either sections S_1 or S_3 for a long time, meaning that future rewards r_1 and r_3 will be discounted a lot when we eventually do reach them. Both $Q^{(L)}(s, a)$ and $Q^{(U)}(s, a)$ will then mostly consist of rewards r_2, so the error is smaller. When the transition probability increases further, we will move faster to section S_3, meaning we earn less reward leading to larger errors.

Finally, we will vary the starting state s. We will use MDP 1 from the previous case, but now we will consider starting states 3, 7 and 12. These will each be in the middle of our three sections S_1, S_2 and S_3. In Figure 6.12 the relative error is given for these three cases.

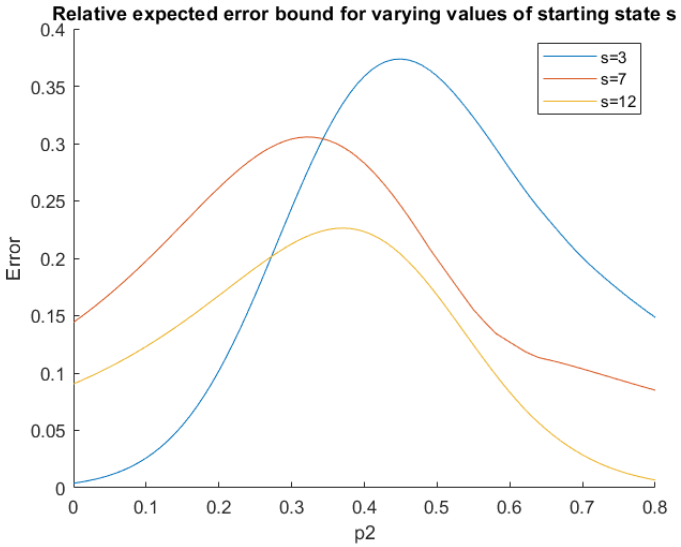


Figure 6.12: Relative error for MDPs with a varying starting state s. We keep the difference in transition probabilities constant such that $p_1 = p_2 + 0.2$.

In this figure we see that the error is larger when we start in in the left side of the state space. This is again caused by the fact that we have a larger difference between r_1 and r_2. Thus, we see a larger difference between $Q^{(L)}(s, a)$ and $Q^{(U)}(s, a)$, because the former will contains more rewards $r_1 = 1$, while the latter contains more rewards $r_2 = 3$. Interestingly, we also see in Figure 6.12 that the relative error for $s = 3$ is smaller for small transition probabilities. This is likely caused by the fact that if the transition probability is small, both $Q^{(L)}(s, a)$ and $Q^{(U)}(s, a)$ will spend most time in S_1 with reward r_1 before even reaching the better rewards the higher rewards of r_2 and r_2, so the difference between them will also be small.

Chapter 7

Conclusion and discussion

7.1 Conclusions

In this thesis, we have analyzed the convergence of Q-learning. First, we performed a review of convergence results from the literature in Chapters 3 and 4. In Chapter 3 we also we also gave an existing proof of convergence to understand what assumptions are needed for convergence and how such a proof works. In Chapter 4 we saw that due to function approximation, additional assumptions are needed to obtain convergence. Due to the restrictions in these results, we introduced another approach to analyze the convergence of Q-learning in Chapter 5. We examined certain classes of Markov decision process and we used their properties to obtain convergence results. We obtained exact convergence for constant rewards using our method, while for more complex rewards we obtained bounds on the expectation of the limiting distribution of Q-learning. Finally, we showed how to extend these results to a larger class of MDPs. Compared to results from the literature, we place no extra requirements on the algorithm. Thus, our method of analysis can circumvent some of the issues that we saw in Chapters 3 and 4 for earlier results on the convergence of Q-learning. To evaluate the results from our method, we also examined the bounds for the case of two and three different rewards numerically. We showed how the upper and lower bounds depend on the properties of the MDP, with a larger difference between p_1 and p_2 leading to a larger error. We also showed how the amount of states, the starting state, the values for our reward and the discount factor influence these bounds.

7.2 Discussion and Future work

In this thesis, we restricted our analysis to certain cases so our results can be extended in multiple ways. First of all, we looked at certain classes of MDPs. While we showed the extension to a quite large class of processes in Section 5.5, there are still many MDPs that aren't considered by our method. The first way to extend our analysis would be to consider more of such classes, such that our results apply to more general cases.

Furthermore, we could improve the bounds that we obtained in this thesis. We found an exact formula for $\mathbb{E}[Q'(s,a)]$, but we weren't able to evaluate it due to the way in which actions are chosen in the target of the update rule in Q-learning. As such, we looked at variables $Q^{(L)}(s,a)$ and $Q^{(U)}(s,a)$ which used fixed policies to obtain lower and upper bounds. For future results it may be helpful to analyze alternative ways to evaluate $\mathbb{E}[Q'(s,a)]$, without having to bound it from above and below. Then we can more accurately say what Q-learning will actually converge to.

Finally, certain assumptions were made in the derivation of our results. We first used a 'uniformized' update rule, but we have showed that this update rule has similar asymptotic behaviour w.r.t. the standard Q-learning update rule. For the case of two and three-part MDPs, we also assumed that Q-learning used a step size of $\alpha = 1$ to help us in our analysis. Since we calculate the expectation $\mathbb{E}[Q'(s,a)]$, the value of this learning rate α will likely not matter, because we take the average anyway and its original purpose was to provide smoothing. However, it likely does have an influence on the full distribution of $Q'(s,a)$. If we would for example need to calculate higher moments $Q'(s,a)$, we would want a formula for a general value of α. So using a general learning rate would be another possible extension of these results.

Bibliography

[1] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[2] Jalaj Bhandari, Daniel Russo, and Raghav Singal. A finite time analysis of temporal difference learning with linear function approximation. In *Conference On Learning Theory*, pages 1691–1692. PMLR, 2018.

[3] Richard J Boucherie and Nico M Van Dijk. *Queueing networks: a fundamental approach*, volume 154. Springer Science & Business Media, 2010.

[4] Zaiwei Chen, Sheng Zhang, Thinh T Doan, Siva Theja Maguluri, and John-Paul Clarke. Finite-time analysis of q-learning with linear function approximation. *arXiv preprint arXiv:1905.11425*, 2019.

[5] Peter Dayan. The convergence of td (λ) for general λ. *Machine learning*, 8(3-4):341–362, 1992.

[6] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201, 1994.

[7] John G Kemeny, J Laurie Snell, and Anthony W Knapp. *Denumerable Markov chains: with a chapter of Markov random fields by David Griffeath*, volume 40. Springer Science & Business Media, 2012.

[8] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[9] Hamid Reza Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *NIPS*, pages 1204–1212, 2009.

[10] Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

[11] Francisco S Melo, Sean P Meyn, and M Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, pages 664–671, 2008.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[13] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[14] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000, 2009.

[17] Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[18] John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

[19] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[20] Nico M Van Dijk and Martin L Puterman. Perturbation theory for markov reward processes with applications to queueing systems. *Advances in applied probability*, pages 79–98, 1988.

[21] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[22] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[23] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[24] Pan Xu and Quanquan Gu. A finite-time analysis of q-learning with neural network function approximation. In *International Conference on Machine Learning*, pages 10555–10565. PMLR, 2020.

Appendix A

Markov Reward Process approach

In this Appendix we will describe another method we have analyzed in our research to obtain error bounds in reinforcement learning. This method is based on chapter 9 from the book by Boucherie and Van Dijk [3], where error bounds are established between performance measures of Markov reward processes. Recall that a policy of a Markov decision process induces a Markov reward process, so this method can for instance be used to compare policies.

A.1 Main results

For the main results we will look at two MRPs defined as tuples $(\mathcal{S}, P_1, r_1)$ and $(\mathcal{S}, P_2, r_2)$. For now we will assume ergodic Markov chains such that we have unique stationary distributions μ_1 and μ_2. In their book, Boucherie and Van Dijk [3] provide results that apply to the average reward criterion. However, these results can be extended to the discounted reward criterion, so we will show both. For the average reward criterion, we let $V^1_{(T)}(s)$ be defined as the total reward obtained in the first MRP until time T when starting from state s. Similarly, $V^2_{(T)}(s)$ defines the total reward of the second process. Subsequently, ρ^1 and ρ^2 define the gains of both MRPs. The following result compares the gains of both MRP'.

Theorem A.1. *Suppose that for all $s \in \mathcal{S}$ and $T \geq 0$ the following holds:*

$$r_2(s) - r_1(s) + \sum_{s'} [P_2(s, s') - P_1(s, s')][V^1_{(T)}(s') - V^1_{(T)}(s)] \geq 0. \tag{A.1}$$

Then,

$$\rho^2 \geq \rho^1. \tag{A.2}$$

For the discounted reward criterion, a similar result hold. We will define V^1_γ and V^2_γ as the discounted rewards corresponding to the MRP's respectively. Then, we can compare these as follows.

Theorem A.2. *Suppose that for all $s \in \mathcal{S}$ the following holds:*

$$r_2(s) - r_1(s) + \gamma \sum_{s'} [P_2(s, s') - P_1(s, s')][V^1_\gamma(s') - V^1_\gamma(s)] \geq 0. \tag{A.3}$$

Then,

$$V^2_\gamma(s) \geq V^1_\gamma(s) \tag{A.4}$$

for all $s \in \mathcal{S}$

Both of the above results can be used to show that one Markov reward process outperforms the other. This can for example be used to show that one policy performs better than another policy. To show such a thing, we need to satisfy Equation (A.1) or (A.3) for the average reward criterion or discounted reward criterion respectively. Another way to compare two MRP's is by examining how close their value functions are. In this way we can bound the difference between the gain or value function. This leads to the following two results.

Theorem A.3. *Suppose that for some function $\epsilon(\cdot)$ defined on $\mathcal{S}$ and for all $s \in \mathcal{S}$ and $T \geq 0$ the following holds:*

$$\left| r_2(s) - r_1(s) + \sum_{s'} [P_2(s, s') - P_1(s, s')][V^1_{(T)}(s') - V^1_{(T)}(s)] \right| \leq \epsilon(s). \tag{A.5}$$

Then,

$$|\rho^2 - \rho^1| \leq \sum_s \mu_2(s)\epsilon(s) = \mu_2^T \epsilon. \tag{A.6}$$

Again we can consider the discounted criterion, leading to the following result. Such a bound on the discounted reward criterion was already applied in an earlier paper by Van Dijk and Puterman [20].

Theorem A.4. *Suppose that for some function $\epsilon(\cdot)$ defined on $\mathcal{S}$ and for all $s \in \mathcal{S}$:*

$$\left| r_2(s) - r_1(s) + \gamma \sum_{s'} [P_2(s,s') - P_1(s,s')][V_\gamma^1(s') - V_\gamma^1(s)] \right| \leq \epsilon(s). \tag{A.7}$$

Then the following hold:

$$\max_{s \in \mathcal{S}} |V_\gamma^2(s) - V_\gamma^1(s)| \leq \max_{s \in \mathcal{S}} \epsilon(s)/(1-\gamma) \tag{A.8}$$

and

$$|\mu_2^T V_\gamma^2 - \mu_2^T V_\gamma^1| \leq \sum_s \mu_2(s)\epsilon(s) = \mu_2^T \epsilon/(1-\gamma). \tag{A.9}$$

For the proofs of the comparison and error bound result for the average reward result we refer to Chapter 9 of [3]. The proofs for results with the discounted rewards will be omitted for now.

A.2 Difference terms

To use the error bound theorems we need to be able to satisfy Equations (A.5) and (A.7). These depend on the difference terms $V_{(T)}^1(s') - V_{(T)}^1(s)$ and $V_\gamma^1(s') - V_\gamma^1(s)$ for the average reward and discounted reward respectively. The paper by Van Dijk and Puterman [20] places bounds on these terms using the mean hitting times. The mean hitting time $H_{ss'}$ denotes the expected number of time steps for the process to reach state s' when starting in state s.

Theorem A.5. *For all $s, s' \in \mathcal{S}$,*

$$|V_{(T)}^1(s') - V_{(T)}^1(s)| \leq 2M \min\{H_{ss'}, H_{s's}\}, \text{ and} \tag{A.10}$$

$$|V_\gamma^1(s') - V_\gamma^1(s)| \leq M \min\{H_{ss'}, H_{s's}\}. \tag{A.11}$$

A.2.1 Example of applying the error bound method

In this section we will give an example of how to apply the error bound result in comparison with the bounding of the hitting times. We consider a Markov decision process with 4 states, numbered 1 through 4, and two actions 'left' (L) and 'right' (R) per state. These actions succeed with probability 0.9 and take the process one state to the left or right respectively. With probability 0.1 the opposite happens. The reward vector equals $(0, 1, 1, 0)$ and only depends on the state and not on the action taken. As a result, the optimal policy would be to stay in states 2 and 3 as much as possible so RRLL is the optimal policy. In Figure A.1 this MDP is displayed. Now we want to apply our actual error bound result. For this example we will look at the case of discounted rewards. To compare two policies π_1 and π_2, we first need to satisfy Equation (A.7). The rewards per state are the same for all policies, since they are independent of action taken. As a result, the difference in rewards will vanish. Furthermore, we only need to calculate difference terms for neighbouring states, since $P(s'|s,a) = 0$ for $a = L, R$ if s and s' aren't next to one another.

If we now let $\pi_1 = \pi^* =$ RRLL, we can compare any other policy π_2 to this optimal policy. To do this we need to find the difference terms for policy π_1. To apply Theorem A.5 we see that the rewards are bounded by 1 such that $M = 1$. Thus, we only need to calculate the hitting times $H_{ss'}$ for neighbouring states. These hitting times depend on transition probabilities of the Markov Reward Process induced by π_1. These transition probabilities for policy RRLL are:

$$\begin{aligned} P_1(s_1, s_1) &= 0.1, \quad P_1(s_1, s_2) = 0.9, \\ P_1(s_2, s_1) &= 0.1, \quad P_1(s_2, s_3) = 0.9, \\ P_1(s_3, s_2) &= 0.9, \quad P_1(s_3, s_4) = 0.1, \\ P_1(s_4, s_3) &= 0.9, \quad P_1(s_4, s_4) = 0.1. \end{aligned}$$

All other transition probabilities are 0. The 6 hitting times we need to calculate are $H_{s_1 s_2}$, $H_{s_2 s_1}$, $H_{s_2 s_3}$, $H_{s_3 s_2}$, $H_{s_3 s_4}$ and $H_{s_4 s_3}$. Due to symmetry we know that $H_{s_1 s_2} = H_{s_4 s_3}$ and $H_{s_2 s_3} = H_{s_3 s_2}$. Calculating

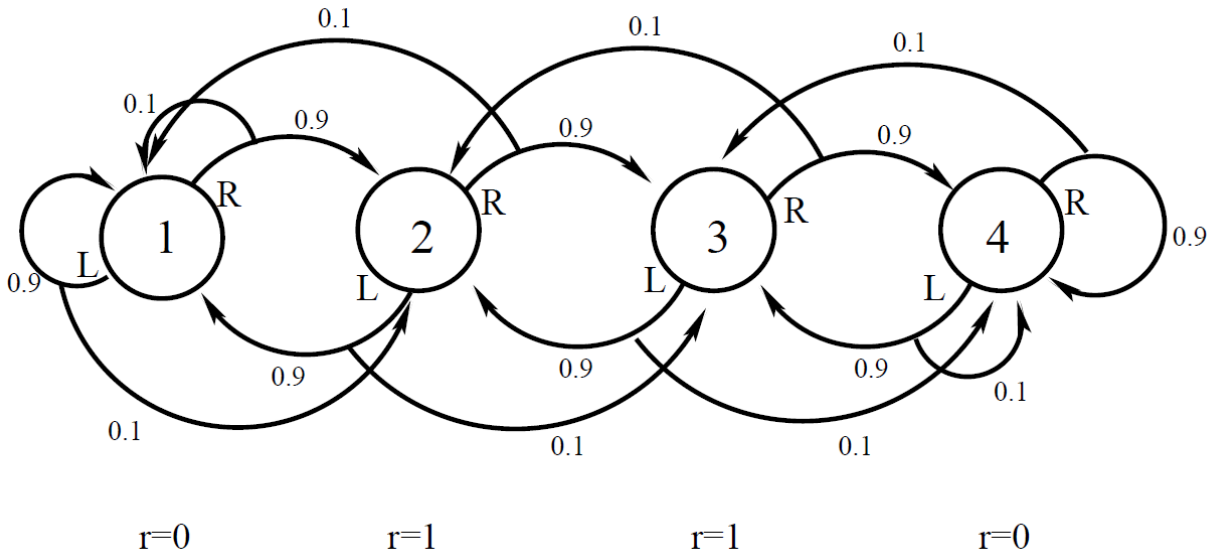


Figure A.1: Example of a simple MDP with 4 states and 2 actions per state.

$H_{s_1 s_2}$ is easy, since with probability 0.9 it takes 1 step to reach s_2 from s_1 and with probability 0.1 we stay in s_1. So we need to solve $H_{s_1 s_2} = 1 + 0.1 H_{s_1 s_2}$ such that $H_{s_1 s_2} = 10/9$. Intuitively we also see that $H_{s_2 s_1} > H_{s_1 s_2}$ because we only have a probability of 0.1 to reach s_1 in 1 step and we can also wander off to states 3 and 4. For our last hitting time we have

$$\begin{aligned} H_{s_2 s_3} &= 1 + 0.1 H_{s_1 s_3} = 1 + 0.1(H_{s_1 s_2} + H_{s_2 s_3}) = 1 + 1/9 + 0.1 H_{s_2 s_3}, \\ 9/10 H_{s_2 s_3} &= 10/9, \\ H_{s_2 s_3} &= 100/81. \end{aligned}$$

Thus we can bound each of the difference terms:

$$\begin{aligned} |V_\gamma^1(s_1) - V_\gamma^1(s_2)| = |V_\gamma^1(s_3) - V_\gamma^1(s_4)| \leq 10/9. \\ |V_\gamma^1(s_2) - V_\gamma^1(s_3)| \leq 100/81 \end{aligned}$$

In this example, we will consider $\pi_2 =$ RRRR as policy to compare to. In states 1 and 2 the transition probabilities are the same in π_1 and π_2, while in states 3 and 4 the absolute difference in the transition probabilities equals 0.8. This leads to the following error bound per state:

$$\begin{aligned} &\left| r_2(s) - r_1(s) + \sum_{s'} [P_2(s,s') - P_1(s,s')][V_\gamma^1(s') - V_\gamma^1(s)] \right| \\ &= \left| \sum_{s'} [P_2(s,s') - P_1(s,s')][V_\gamma^1(s') - V_\gamma^1(s)] \right| \\ &\leq \begin{cases} 0 & \text{if } s = s_1, s_2 \\ 0.8 * 100/81 + 0.8 * 10/9 = 152/81 & \text{if } s = s_3 \\ 0.8 * 10/9 = 8/9 & \text{if } s = s_4. \end{cases} \end{aligned}$$

As a result, we have $\max_s |V_\gamma^2(s) - V_\gamma^1(s)| \leq \frac{152}{81(1-\gamma)}$.

To see if this is a good bound, we want to find the actual value functions V_γ^1 and V_γ^2. Finding the true solutions is straightforward, we just need to calculate the transition probability matrices and reward vectors. We have already seen that the reward vector doesn't depend on policy. We have that:

$$P_{\pi_1} = \begin{bmatrix} 0.1 & 0.9 & 0 & 0 \\ 0.1 & 0 & 0.9 & 0 \\ 0 & 0.9 & 0 & 0.1 \\ 0 & 0 & 0.9 & 0.1 \end{bmatrix}, P_{\pi_1} = \begin{bmatrix} 0.1 & 0.9 & 0 & 0 \\ 0.1 & 0 & 0.9 & 0 \\ 0 & 0.1 & 0 & 0.9 \\ 0 & 0 & 0.1 & 0.9 \end{bmatrix}, r = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{A.12}$$

We then have that $V_\gamma^1 = (I - \gamma P_{\pi_1})^{-1} r = [8.10, 9.10, 9.10, 8.10]^T$ and $V_\gamma^2 = (I - \gamma P_{\pi_2})^{-1} r = [2.57, 2.89, 2.04, 0.97]^T$ where we set $\gamma = 0.9$. Thus, the maximum absolute error equals

$$\max_s |V_\gamma^2(s) - V_\gamma^1(s)| \approx |0.97 - 8.10| \approx 7.13, \tag{A.13}$$

while our error bound gives $\max_s |V_\gamma^2(s) - V_\gamma^1(s)| \leq \frac{152}{81*0.1} \approx 18.77$.

A.3 Policy iteration and the Markov reward approach

Our motivation behind the introduction of the Markov reward approach is to use the results in Section (A.1) and apply them in reinforcement learning. One can think of finding a bound on the error on the approximated value function. We will distinguish between learning methods that do just evaluation of a single policy and control methods that try to learn the optimal value function. The reason for this is that in methods like temporal differnce methods, since we only evaluate a singe policy π and as such we only have a single Markov reward process. Instead, methods like Q-learning and SARSA have changing policies, so the Markov reward approach applies better here. These methods can be thought of as doing *Generalized policy iteration* (GPI), as described by Sutton [15], since we simultaneously update state-action values to match the current policy (evaluation) and improve the policy with respect to the current value function (improvement).

Before we consider these learning methods, we will first describe why standard policy iteration works. The key property of policy iteration is that in each iteration, the new policy is better than the previous one. This is evident from the algorithm, because we specifically choose the new policy to be better. In the following result by Puterman [13] we show that the policy iteration algorithm in Algorithm 1 indeed converges. We will modify notation to let V^t be the value function in iteration t, such that we match the notation in this chapter.

Theorem A.6. *Let V^t and V^{t+1} be successive values generated by policy iteration, described in Algorithm 1. Then it holds that $V^{t+1} \geq V^t$ componentwise.*

Proof. Corresponding to current value function V_t we have the current policy π_t. Let π_{t+1} be the policy obtained by applying the improvement step of the algorithm. In other words we have:

$$\pi_{t+1}(s) \in \arg\max_{a \in \mathcal{A}} r(s,a) + \gamma \sum_{s \in \mathcal{S}} P(s'|s,a) V^t(s').$$

Then because of this maximization we have

$$r_{\pi_{t+1}} + \gamma P_{\pi_{t+1}} V^t \geq r_{\pi_t} + \gamma P_{\pi_t} V^t = V^t, \tag{A.14}$$

where r_π and P_π are the reward vector and transition probability matrix respectively corresponding to a policy π.

$$r_{\pi_{t+1}} \geq (I - \gamma P_{\pi_{t+1}}) V^t \tag{A.15}$$

$$V^{t+1} = (I - \gamma P_{\pi_{t+1}})^{-1} r_{\pi_{t+1}} \geq V^t. \tag{A.16}$$

Here we used that if $u \geq$ v, then $(I - \gamma P)^{-1} u \geq (I - \gamma P)^{-1} v$ for any stochastic matrix P. Thus, we preserve the inequality. This fact is proven in Puterman [13]. □

Close inspection of Equation (A.14) in this proof reveals a deeper result about policy iteration with regards to the Markov reward approach. If we rearrange this equation, we obtain that

$$r_{\pi_{t+1}} - r_{\pi_t} + \gamma [P_{\pi_{t+1}} - P_{\pi_t}] V^t \geq 0. \tag{A.17}$$

This exactly satisfies Equation (A.3) of the comparison result for discounted rewards. The only difference is that here we have only $V^t(s')$ instead of the bias term $V^t(s') - V^t(s)$. However, the inclusion of $V^t(s)$ isn't necessary and the inclusion can mainly help for approximation of the term. To see that the comparison result and policy iteration match closely, we will give the proof of Theorem A.2 below.

Proof. For the discounted value functions V_γ^1 and V_γ^2 corresponding to each of our MRP's, we have for all $s \in \mathcal{S}$:

$$\begin{aligned} V_\gamma^2(s) - V_\gamma^1(s) &= r_2(s) - r_1(s) + \gamma \sum_{s' \in \mathcal{S}} P_2(s,s') V_\gamma^2(s') - \gamma \sum_{s' \in \mathcal{S}} P_1(s,s') V_\gamma^1(s') \\ &= r_2(s) - r_1(s) + \gamma \sum_{s' \in \mathcal{S}} [P_2(s,s') - P_1(s,s')] V_\gamma^1(s') + \gamma \sum_{s' \in \mathcal{S}} P_2(s,s') [V_\gamma^2(s') - V_\gamma^1(s')]. \end{aligned}$$

For the second term we get

$$\begin{aligned}
\sum_{s'\in\mathcal{S}}[P_2(s,s')-P_1(s,s')]V_\gamma^1(s') &= \sum_{s'\neq s}[P_2(s,s')-P_1(s,s')]V_\gamma^1(s') + [P_2(s,s)-P_1(s,s)]V_\gamma^1(s) \\
&= \sum_{s'\neq s}[P_2(s,s')-P_1(s,s')]V_\gamma^1(s') + [(1-\sum_{s'\neq s}P_2(s,s'))-(1-\sum_{s'\neq s}P_1(s,s'))]V_\gamma^1(s) \\
&= \sum_{s'\neq s}[P_2(s,s')-P_1(s,s')]V_\gamma^1(s') - \sum_{s'\neq s}[P_2(s,s')-P_1(s,s')]V_\gamma^1(s) \\
&= \sum_{s'\neq s}[P_2(s,s')-P_1(s,s')][V_\gamma^1(s')-V_\gamma^1(s)] \\
&= \sum_{s'\in\mathcal{S}}[P_2(s,s')-P_1(s,s')][V_\gamma^1(s')-V_\gamma^1(s)]
\end{aligned}$$

Here we used in the second inequality that $\sum_{s'\in\mathcal{S}} P_1(s,s') = 1$ such that $P_1(s,s) = 1 - \sum_{s'\neq s} P_1(s,s')$ and similarly for P_2. Using this in the earlier equation and applying Equation (A.3) we obtain

$$\begin{aligned}
V_\gamma^2(s) - V_\gamma^1(s) &\geq \gamma \sum_{s'\in\mathcal{S}} P_2(s,s')[V_\gamma^2(s') - V_\gamma^1(s')] \\
V_\gamma^2(s) - V_\gamma^1(s) - \gamma \sum_{s'\in\mathcal{S}} P_2(s,s')[V_\gamma^2(s') - V_\gamma^1(s')] &\geq 0.
\end{aligned}$$

Now if we transform this to vector form we get $(I - \gamma P_2)[V_\gamma^2 - V_\gamma^1] \geq 0$, so we obtain our result that $V_\gamma^2 - V_\gamma^1 \geq 0$. This last result follows because multiplying by $(I-\gamma P_2)^{-1}$ preserves the inequality, just as we saw in the proof of policy iteration. □

This confirms that the comparison result for the Markov reward approach is a generalization of policy iteration. In policy iteration, we specifically choose the new policy to satisfy Equation (A.3), but any other policy or Markov reward process that satisfies it will lead to the same result.

A.4 State-action values

A problem when applying either the comparison or error bound result in control methods like Q-learning or SARSA is that these use state-action values, instead of the state values that we saw in these results. A natural question is if we can extend these results to the case of state-action values. We will now consider a Markov reward process in which we transition from state-action pair (s,a) to state-action pair (s',a') instead of from state s to state s'. This should lead to similar result, as we still have a value function $Q(s,a)$ defined on each new 'state' (s,a). The reward $r(s,a)$ is also still well-defined and transition probabilities now equal $p(s',a'|s,a) = P(s'|s,a)\pi(a'|s')$. First we will recall the Bellman Equation (2.4) for the state-action values:

$$\begin{aligned}
Q^\pi(s,a) &= r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} \pi(a'|s')Q^\pi(s',a') \quad \text{or in vector form,} \\
Q^\pi &= r + \gamma P_\pi Q^\pi. \qquad \text{(A.18)}
\end{aligned}$$

The matrix $P_\pi \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}||\mathcal{A}|}$ contains the transition probabilities of transitioning from state-action pair to state-action pair.

$$P_\pi((s,a),(s',a')) = p(s',a'|s,a)$$

This matrix can be separated such that $P_\pi = P\vec{\pi}$, where $P \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}|}$ contains the probability of transitioning to state s' after taking action a in state s. The matrix $\vec{\pi} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}||\mathcal{A}|}$ describes policy π. For these matrices we have

$$\begin{aligned}
P((s,a),s') &= P(s'|s,a) \quad \text{and,} \\
\vec{\pi}(s',(s',a')) &= \pi(a'|s').
\end{aligned}$$

Policy iteration can be done similarly as before, now using this Bellman equation. In the policy evaluations step we let π_t be the current policy. Then, we can solve Equation (A.18) for this policy to obtain Q^{π_t}. The policy improvement step is done by maximizing over this value function such that

$$\pi(a|s) = 1 \quad \text{iff} \quad a \in \arg\max_{b\in\mathcal{A}} Q^{\pi_t}(s,a). \qquad \text{(A.19)}$$

This policy iteration also converges, as is shown in the following theorem.

Theorem A.7. *Let Q^{π_t} and $Q^{\pi_{t+1}}$ be successive values generated by the policy iteration algorithm using state-action values. Then it holds that $Q^{\pi_{t+1}} \geq Q^{\pi_t}$ componentwise.*

Proof. The proof is very similar to the one of policy iteration with state values. We again note that

$$r + \gamma P_{\pi_{t+1}} Q^{\pi_t} \geq r + \gamma P_{\pi_t} Q^{\pi_t} = Q^{\pi_t}.$$

Furthermore, $P_{\pi_{t+1}}$ is still a stochastic matrix such that $(I - \gamma P_{\pi_{t+1}})^{-1} r = Q^{\pi_{t+1}} \geq Q^{\pi_t}$. □

Let's move onto our main goal of proving comparison and error bound results for state-action values. The extension of policy iteration to state-action values was also done with the hope that the comparison result again gives a generalization. Fortunately, this result also works for state-action values if we restrict ourselves to MRPs that are induced by policies. We will consider the policies π_1 and π_2 with associated state-action values Q^{π_1} and Q^{π_2}.

Theorem A.8. *Suppose that for all $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$:*

$$\sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a \in \mathcal{A}} [\pi_2(a'|s') - \pi_1(a'|s')] Q^{\pi_1}(s', a') \geq 0. \tag{A.20}$$

Then,

$$Q^{\pi_2}(s,a) \geq Q^{\pi_1}(s,a) \tag{A.21}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

Proof. For any $s \in \mathcal{S}$ and $a \in \mathcal{A}$ we have

$$\begin{aligned}
&Q^{\pi_2}(s,a) - Q^{\pi_1}(s,a) = \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi_2(a'|s') Q^{\pi_2}(s', a') - r(s,a) - \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi_1(a'|s') Q^{\pi_1}(s', a') \\
&= \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} [\pi_2(a'|s') - \pi_1(a'|s')] Q^{\pi_1}(s', a') + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi_2(a'|s') [Q^{\pi_2}(s', a') - Q^{\pi_1}(s', a')] \\
&\geq \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi_2(a'|s') [Q^{\pi_2}(s', a') - Q^{\pi_1}(s', a')].
\end{aligned}$$

The last inequality follows from the condition in Equation (A.20). In vector form we obtain

$$Q^{\pi_2} - Q^{\pi_1} \geq \gamma P_{\pi_2} [Q^{\pi_2} - Q^{\pi_1}] \tag{A.22}$$

$$(I - \gamma P_{\pi_2}) [Q^{\pi_2} - Q^{\pi_1}] \geq 0. \tag{A.23}$$

Since P_{π_2} is a stochastic matrix it again applies that $Q^{\pi_2} - Q^{\pi_1} \geq 0$ componentwise. □

Again we have that this comparison result is a generalization of policy iteration in the sense that the new policy chosen in policy iteration will always satisfy Equation (A.20).

A.4.1 Error bound result for state-action values

Just like we have extended the comparison result to state-action values, we can also extend the error bound result to them. To do this, we again consider policies π_1 and π_2. We will require that the Markov reward processes $(\mathcal{S}, P_{\pi_1}, r_{\pi_1})$ and $(\mathcal{S}, P_{\pi_2}, r_{\pi_2})$ on the standard state space will be ergodic, such that the steady-state distributions μ_1 and μ_2 exist. Now we consider the Markov reward processes induced by π_1 and π_2 that transitions from state-action pair to state-action pair. This Markov reward process over the state-action space will have steady-state distribution ν_1 and ν_2, which are defined as

$$\nu(s,a) = \mu(s)\pi(a|s). \tag{A.24}$$

Thus, these steady-state distributions determine the long-term probability of being in state s and taking action a. Now we move onto our result.

Theorem A.9. *Suppose that for some $\epsilon \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ and for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ the following holds:*

$$\left| \sum_{s'} p(s'|s,a) \sum_{a'} [\pi_2(a'|s') - \pi_1(a'|s')] Q^{\pi_1}(s', a') \right| \leq \epsilon(s,a). \tag{A.25}$$

Then the following hold:

$$\max_{s\in\mathcal{S},a\in\mathcal{A}} |Q^{\pi_2}(s,a) - Q^{\pi_2}(s,a)| \leq \frac{\gamma \max_{s\in\mathcal{S},a\in\mathcal{A}} \epsilon(s,a)}{1-\gamma} \tag{A.26}$$

and

$$|\nu_2^T Q^{\pi_2} - \nu_2^T Q^{\pi_1}| \leq \sum_{s,a} \nu_2(s,a)\epsilon(s,a) = \frac{\gamma \nu_2^T \epsilon}{1-\gamma}. \tag{A.27}$$

Proof. First of all, we will look at the maximum absolute difference between Q^{π_1} and Q^{π_2}. For this we will use Equation ... of the proof for the comparison result in Theorem A.8.

$$\begin{aligned}
&\max_{s,a} |Q^{\pi_2}(s,a) - Q^{\pi_2}(s,a)| \\
&\leq \max_{s,a} \Big| \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} [\pi_2(a'|s') - \pi_1(a'|s')] Q^{\pi_1}(s',a') \\
&+ \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} \pi_2(a'|s') [Q^{\pi_2}(s',a') - Q^{\pi_1}(s',a')] \Big| \\
&\leq \gamma \max_{s,a} \epsilon(s,a) + \gamma \max_{s,a} \Big| \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} \pi_2(a'|s') [Q^{\pi_2}(s',a') - Q^{\pi_1}(s',a')] \Big| \\
&\leq \gamma \max_{s,a} \epsilon(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} \pi_2(a'|s') \max_{s^*,a^*} |Q^{\pi_2}(s^*,a^*) - Q^{\pi_1}(s^*,a^*)| \\
&= \gamma \max_{s,a} \epsilon(s,a) + \gamma \max_{s,a} |Q^{\pi_2}(s,a) - Q^{\pi_1}(s,a)|.
\end{aligned}$$

The second inequality follows from our condition in Equation (A.25) and the third inequality follows from the fact that each of the differences $|Q^{\pi_2}(s,a) - Q^{\pi_1}(s,a)|$ is bounded by the maximum over s and a. Subsequently, the sums over transition probabilities and policies will sum up to 1. Finally, we then get

$$\max_{s,a} |Q^{\pi_2}(s,a) - Q^{\pi_2}(s,a)| \leq \frac{\gamma \max_{s,a} \epsilon(s,a)}{1-\gamma} \tag{A.28}$$

proving the first result. For the second result, we have that

$$\begin{aligned}
&|\nu_2^T Q^{\pi_2} - \nu_2^T Q^{\pi_1}| \\
&\leq \Big| \gamma \sum_{s,a} \nu_2(s,a) \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} [\pi_2(a'|s') - \pi_1(a'|s')] Q^{\pi_1}(s',a') \\
&+ \gamma \sum_{s,a} \nu_2(s,a) \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} \pi_2(a'|s') [Q^{\pi_2}(s',a') - Q^{\pi_1}(s',a')] \Big| \\
&\leq \gamma \sum_{s,a} \nu_2(s,a)\epsilon(s,a) + \gamma \Big| \sum_{s,a} \nu_2(s,a) \sum_{s'\in\mathcal{S}} p(s'|s,a) \sum_{a'\in\mathcal{A}} \pi_2(a'|s') [Q^{\pi_2}(s',a') - Q^{\pi_1}(s',a')] \Big|.
\end{aligned}$$

We can expand the steady-state distribution over the state-action pairs to $\nu_2(s,a) = \mu_2(s)\pi_2(a|s)$. Here the steady-state distribution over the states satisfies $\mu_2 P_2 = \mu_2$. Here P_2 are the transition probabilities of s to s' in the state MRP induced by π_2. These transition probabilities equal $P_2(s,s') = \sum_a \pi(a|s) p(s'|s,a)$ Thus, we have that

$$\sum_{s,a} \nu(s,a) \sum_{s'} p(s'|s,a) = \sum_s \mu_2(s) \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) = \sum_s \sum_{s'} \mu_2(s) P_2(s,s') = \sum_{s'} \mu_2(s') \tag{A.29}$$

Finally, filling this in into Equation ... we obtain

$$\begin{aligned}
&\nu_2^T |Q^{\pi_2} - \nu_2^T Q^{\pi_1}| \\
&\leq \gamma \sum_{s,a} \nu_2(s,a)\epsilon(s,a) + \gamma \Big| \sum_{s'\in\mathcal{S}} \mu_2(s') \sum_{a'\in\mathcal{A}} \pi_2(a'|s') [Q^{\pi_2}(s',a') - Q^{\pi_1}(s',a')] \Big| \\
&\leq \gamma \sum_{s,a} \nu_2(s,a)\epsilon(s,a) + \gamma \Big| \sum_{s',a'} \nu(s',a') [Q^{\pi_2}(s',a') - Q^{\pi_1}(s',a')] \Big| \\
&= \gamma \nu_2^T \epsilon + \gamma |\nu^T Q^{\pi_2} - \nu^T Q^{\pi_1}]|.
\end{aligned}$$

We can rearrange this equation to obtain the final result of $|\nu_2^T Q^{\pi_2} - \nu_2^T Q^{\pi_1}| \leq \frac{\gamma \nu_2^T \epsilon}{1-\gamma}$ □

These results with state-action values give multiple advantage over their state value counterparts.

- First of all, many reinforcement learning methods use state-action values, so it obviously makes more sense to use these over state value functions. State-action values are also likely easier to relate to the approximations $Q_t(s,a)$ which follow from Q-learning and SARSA for example.
- Secondly, notice that Equations (A.20) and (A.25) are a lot easier to satisfy than Equations (A.3) and (A.7). The reward term is completely gone, because the policy only influences the next state as we have already chosen action a in state-action pair (s,a). Thus, both MRPs have the same reward and this term vanishes. Also, we can separate the transition structure from the policy. From the state action pair (s,a) the transition probabilities to the next state s' are fixed, and the policy then determines the next action a'. This has mayor advantages, as we generally know the policy but not the transition probabilities.
- Finally, notice that we omitted the bias terms in Equations (A.20) and (A.25). Instead, we just consider $Q^{\pi_1}(s',a')$. However, we can still include these bias terms like we did for the state values and we even have more freedom in choosing them. To see this, let (s,a) be any state-action pair. Then we have

$$\begin{aligned}
&\sum_{a'\in\mathcal{A}}[\pi_2(a'|s')-\pi_1(a'|s')][Q^{\pi_1}(s',a')-Q^{\pi_1}(s,a)]\\
&=\sum_{a'\in\mathcal{A}}[\pi_2(a'|s')-\pi_1(a'|s')][Q^{\pi_1}(s',a')]-\sum_{a'\in\mathcal{A}}[\pi_2(a'|s')-\pi_1(a'|s')][Q^{\pi_1}(s,a)]\\
&=\sum_{a'\in\mathcal{A}}[\pi_2(a'|s')-\pi_1(a'|s')][Q^{\pi_1}(s',a')]-Q^{\pi_1}(s,a)\sum_{a'\in\mathcal{A}}[\pi_2(a'|s')-\pi_1(a'|s')]\\
&=\sum_{a'\in\mathcal{A}}[\pi_2(a'|s')-\pi_1(a'|s')][Q^{\pi_1}(s',a')].
\end{aligned}$$

  Here we used that the policies are probability distributions and thus sum up to 1. The advantage of including these bias terms is that they may be easier to bound, which is especially useful in the error bound result. Compared to the state values we also have more freedom in choosing these bias terms. For example, we can bound Equation (A.25) as follows:

$$\begin{aligned}
&\Big|\sum_{s'}p(s'|s,a)\sum_{a'}[\pi_2(a'|s')-\pi_1(a'|s')]Q^{\pi_1}(s',a')\Big|\\
&\leq\sum_{s'}p(s'|s,a)\sum_{a'}\big|\pi_2(a'|s')-\pi_1(a'|s')\big|\big|Q^{\pi_1}(s',a')-Q^{\pi_1}(s',a)\big|.
\end{aligned}$$

  This comes down to placing a bound on the difference in value function when taking one action over another. However, we can also choose to let $\big|Q^{\pi_1}(s',a')-Q^{\pi_1}(s,a)\big|$ be our bound, thus also taking a different state. Bounds can perhaps be found in a similar manner as in Theorem A.5 in Section A.2.

A.5 Q-learning

In this section we will try to apply the comparison and error bound result for state-action values in Q-learning. Recall that the update rule of Q-learning from Equation (3.8) is given by

$$Q_{t+1}(s,a)\leftarrow Q_t(s,a)+\alpha_t[r(s,a)+\gamma\max_{a'}Q_t(s',a')-Q_t(s,a)] \tag{A.30}$$

for some transition (s,a,s'). The Markov reward approach results depend on the process of state-action pairs induced by some policy π. Thus we need policies if we want to apply these methods. In Q-learning, multiple policies are present, namely the policy with which we explore, policies with respect to current approximation and the optimal policy. Below we elaborate on these policies.

- First of all, the *behavioural policy* determines the actions we actually take in the process and thus what states, actions and rewards we observe. We saw in the proof in Section that the only requirement for converge of Q-learning is that we visit every state action pair infinitely often. So this behaviour policy can be some stationary random policy, but it can also depend on the current state-action value function and be $\epsilon-$greedy for example.
- Secondly, we can consider the policy corresponding to the current approximation $Q_t(s,a)$. In this case, we let

$$\pi_t(a|s)=1\quad\text{iff}\quad\arg\max_{a\in\mathcal{A}}Q_t(s,a). \tag{A.31}$$

Since we update the state-action values in each time step, we also keep updating this policy. Thus we can look at what happens in one iteration.

- The final policy is the optimal policy π^*. We know that in Q-learning we converge to the optimal value function $Q^*(s,a)$ with associated policy π^*. We can compare our current policy to this optimal policy using the error bound method.

Now we wish to compare policies with the error bound method, because different policies induce different MRPs. First of all, we can look if consecutive policies are better. Let π_t be the policy corresponding Q_t and π_{t+1} the one corresponding to Q_{t+1}, where Q_{t+1} is obtained from the one-step transition using update rule (3.8). Then, we can compare these policies by considering the Markov reward processes over the state-action pairs induced by these policies. We can think of this as if we were to continue using these policies until we converge to the corresponding value functions

One idea is to use the comparison result to show that $Q^{\pi_{t+1}} \geq Q^{\pi_t}$. To do that we need to show for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$

$$\sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} [\pi_{t+1}(a'|s') - \pi_t(a'|s')] Q^{\pi_t}(s',a') \geq 0. \tag{A.32}$$

This seems like a hard condition to verify, but we can make an observation about policies π_t and π_{t+1}. At time step t we only update Q_t in the state-action pair (s_t, a_t) that we observe. So Q_{t+1} is the same as Q_t except in this state action pair. As a result, π_{t+1} is the same as π_t in all states except possibly s_t. Thus, for any state action pair (s,a) the above condition reduces to

$$p(s_t|s,a) \sum_{a' \in \mathcal{A}} [\pi_{t+1}(a'|s_t) - \pi_t(a'|s_t)] Q^{\pi_t}(s_t,a') \geq 0, \tag{A.33}$$

from which we can obviously discard the transition probability $p(s_t,s,a)$. What we are left with is something that doesn't depend on the current state-action pair (s,a), so we only have to satisfy one condition. Furthermore, the policies that we consider choose the action which maximize the current value function. As a result, these policies are deterministic and assign a probability of 1 to the maximizing action. If we let a_{old} be the maximizing action of π_t in state s_t and a_{new} be the maximizing action of π_{t+1} in state s_t, we need to show that

$$Q^{\pi_t}(s_t, a_{new}) \geq Q^{\pi_t}(s_t, a_{old}). \tag{A.34}$$

This again reminds us of policy iteration. We choose an action which gives a higher value with respect to some current value function approximation. In the context of Q-learning, this has to be done for current state s_t. But how do we verify that our new action yields a higher value? What we have are the approximation Q_t, the observed transition (s_t, a_t, s_{t+1}) and the incurred reward $r(s_t, a_t)$. Somehow we need to use this information to show that this inequality holds.

We can also try to apply the error bound result. This can again be done for consecutive policies π_t and π_{t+1}. With similar arguments as before we need to bound for all state-action pairs (s,a)

$$\begin{aligned}
&\Big| \sum_{s'} p(s'|s,a) \sum_{a'} [\pi_{t+1}(a'|s') - \pi_t(a'|s')] Q^{\pi_t}(s',a') \Big| \\
&= \Big| p(s_t|s,a) \sum_{a'} [\pi_{t+1}(a'|s_t) - \pi_t(a'|s_t)] Q^{\pi_t}(s_t,a') \Big| \\
&= p(s_t|s,a) \Big| Q^{\pi_t}(s_t, a_{new}) - Q^{\pi_t}(s_t, a_{old}) \Big|.
\end{aligned}$$

The only dependence on (s,a) is now the transition probability, which can of course be bounded by 1 such that we obtain a general bound. Now we need to say something about the absolute difference between the values of the best actions of the current and next policy.

Let's shift gears for a moment and look at other ways to apply these results. Our initial goal was to find a bounds of our approximate process, for example between some approximation Q_t and the optimal value Q^*. However, we can only compare the state-action values corresponding to different policies using our results. While Q_t induces some policy π_t, the associated value function Q^{π_t} will be different. As a result, we find a bound between Q^{π_t} and Q^{π^*} and not between Q_t and Q^{π^*}. Thus, we need some way to connect the approximate process to the true one.

One idea is to connect the bias terms of the true process to those of the approximate process. Letting Q_t

and Q_{t+1} be the approximations of our Q-learning process, we know for the bias terms

$$Q_{t+1}(s',a') - Q_{t+1}(s,a) = \tag{A.35}$$

$$\begin{cases} Q_t(s',a') - Q_t(s,a) & \text{if } (s,a) \neq (s_t,a_t) \text{ and } (s,a) \neq (s_t,a_t) \\ Q_t(s',a') - Q_t(s,a) - \alpha[r(s,a) + \gamma \max_b Q_t(s_{next},b) - Q_t(s,a)] & \text{if } (s,a) = (s_t,a_t) \\ Q_t(s',a') - Q_t(s,a) + \alpha[r(s',a') + \gamma \max_b Q_t(s_{next},b) - Q_t(s',a')] & \text{if } (s',a') = (s_t,a_t) \end{cases} \tag{A.36}$$

where we require $(s,a) \neq (s',a')$.

A.5.1 Example using Q-learning

We still have some difficulty to connect the approximate process of Q-learning to the Markov reward approach which relies on the true value functions corresponding to some policy. To gain more insight, we again work out examples to see how the comparison and error bound results apply. The example we will look at is the same one as in Section A.2.1. First we will try to see if we can apply the comparison result. We can try to show that in Q-learning $Q^{\pi_{t+1}} \geq Q^{\pi_t}$. This would prove that the policies related to our value function approximation improves. We saw that this inequality holds if $Q^{\pi_t}(s_t, a_{new}) \geq Q^{\pi_t}(s_t, a_{old})$, where $a_{old} = \arg\max_b Q_t(s_t,b)$ and $a_{new} = \arg\max_b Q_{t+1}(s_t,b)$. These are the actions chosen in state s_t by our policies π_t and π_{t+1}.

Unfortunately, this doesn't hold for Q-learning and consecutive policies can become worse. We can even show this in our example MDP in Figure A.1. First of all, the behavioural policy doesn't matter for this counterexample, as under any policy there is at least a 10% chance of going to either the left or right (or stay in the same state for states 1 and 4). Then, we initialize our state-action values as follows:

$$Q_0(s,a) = \begin{cases} 0 & \text{if } s = 1,2 \text{ and } a = L \text{ or } (s,a) = (3,R) \\ 1 & \text{if } s = 1,2 \text{ and } a = R \text{ or } (s,a) = (3,L) \\ 5 & \text{if } (s,a) = (4,R) \\ 6 & \text{if } (s,a) = (4,L). \end{cases} \tag{A.37}$$

Thus, our initial policy by choosing the maximizing action in each state equals $\pi_0 = RRLL$. From before we know that this is our optimal policy, as this policy always tries to be in states 2 and 3 which give a positive reward. Now suppose that $s_0 = 3$ and $a_0 = R$ and we observe $s_1 = 4$. We saw that this tuple can happen under any behaviour policy. We let our learning rate be $\alpha_0 = 1$ at time 0 and the discount rate be $\gamma = 1/2$. Then we get the following update for $(s_0,a_0) = (3,R)$

$$\begin{aligned} Q_1(3,R) &\leftarrow Q_0(3,R) + 1 * [r(3,R) + \gamma \max_a Q_0(4,a) - Q_0(3,R)] \\ &= r(3,R) + \gamma \max_a Q_0(4,a) = 1 + 3/4 * Q_0(4,L) = 1 + 1/2 * 6 = 4. \end{aligned}$$

All other state-action pairs will stay the same. We see that now $Q_1(3,R) = 4 > 1 = Q_1(3,L)$ and our policy updates to $\pi_1 = RRRL$. So we have obtained a worse policy after doing an update of Q-learning. This counterexample shows that Q-learning doesn't necessarily lead to improvement in every step, even though it does converge in the limit. If we actually want to apply the comparison result, we would probably have to show that we improve on average. This was also done in the proof of Q-learning in Section 3.3, where the F_t, and thus Δ_t, gets smaller on average.