Technische Universität München
Lehrstuhl für Datenverarbeitung

# Accelerated Gradient Algorithms for Robust Temporal Difference Learning

**Dominik Jakob Meyer**

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktor der Naturwissenschaften (Dr. rer. nat.)**

genehmigten Dissertation.

**Vorsitzende(r):** Prof. Dr. Bernhard Wolfrum

**Prüfer der Dissertation:**

1. Prof. Dr.-Ing. Klaus Diepold
2. Prof. Dr.-Ing. Matthias Althoff

Die Dissertation wurde am 09.10.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 28.04.2021 angenommen.

Dominik Jakob Meyer. *Accelerated Gradient Algorithms for Robust Temporal Difference Learning.* Dissertation, Technische Universität München, Munich, Germany, 2021.



Chair for Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de`.

# Abstract

In modern reinforcement learning, as in many modern machine learning algorithms, gradient descent techniques play a central role as the main solution method. One of the first gradient based solution methods for reinforcement learning is the so called gradient descent temporal difference learning. Often it is not possible to use sophisticated feature extraction methods such as deep learning due to computational restrictions. In these cases a linear approximation framework is one of the most widespread methods of making continuous system measurements that are no longer exactly representable, computationally feasible. In such a linear approximation setting, gradient temporal difference learning algorithms were the first linear in computation and storage complexity algorithms that were applicable to reinforcement learning.

If the dimension of features is large compared to the number of samples and the sampling process involves a lot of noise, then the gradient descent temporal difference learning methods tend to degrade in performance. Additionally, stochastic gradient methods can be slow in convergence, especially close to the optimum.

A proper technique to remedy problems with noise and large feature dimensions is regularization. This thesis describes a combination of the original underlying cost function with regularization and derives a solution method that still conserves the linear time and storage complexity. To speed up convergence of the algorithms, this thesis additionally extends the algorithms to Nesterovs method of accelerated gradient descent. In order to be able to perform this extension, it is investigated if the preconditions for the application of the acceleration scheme are met by the cost function.

The resulting algorithms can perform more stable learning in environments with the presence of additive Gaussian noise while still being of linear computational complexity in terms of feature size. On the test domains investigated in this thesis, they converge to the same value of error measure with one quarter of the samples as the original algorithms on average. Also, they are able to converge to sensible results in terms of the problem domain with up to five times as many Gaussian noise features as information bearing features present.

# Contents

# List of Figures

# List of Symbols

| Symbol | Description |
|---|---|
| $\mathcal{S}$ | The input space of states sensed from the environment |
| $V(s) \in \mathbb{R}$ | Value function for state $s \in \mathcal{S}$ |
| $n$ | Number of states |
| $n_a$ | Number of actions in each state |
| $\mu$ | Policy to control the environment |
| $V_\mu \in \mathbb{R}^n$ | Value function induced by policy $\mu$ |
| $V^* \in \mathbb{R}^n$ | Optimal value function |
| $\mathcal{T}, \mathcal{T}_\mu$ | Optimal Bellman operator and Bellman operator associated with policy $\mu$ |
| l | Dimension of the feature vector |
| $\phi(s) = \phi_s$ | Feature transformation applied on state $s$ with $\phi(s) \in \mathbb{R}^l$ |
| $\Phi$ | Feature matrix |
| $\theta \in \mathbb{R}^l$ | Weight vector for the linear value function approximation |
| $\mathcal{V}$ | Feature space |
| $\xi \in \mathbb{R}^n$ | Steady state distribution of the MDP |
| $\Xi \in \mathbb{R}^{n \times n}$ | Matrix with the elements of $\xi$ on the diagonal |
| $\gamma \in [0, 1]$ | Discounting factor |
| $k$ | Algorithm sampling step |
| $t$ | Sampling environment step |
| $\Psi_{\alpha_k \eta}$ | Soft thresholding operator |
| $\alpha$ | learning rate |
| $\beta = \kappa\alpha$ | secondary learning rate depending on $\alpha$ |
| $\kappa > 0$ | parameter to derive secondary weights from $\alpha$ |
| $\eta$ | regularization parameter |
| $\nu$ | acceleration parameter |

# Chapter 1.

# Introduction

Recent developments in machine learning applications are fascinating the world of the capability of intelligent machines powered by learning algorithms and the problems that can nowadays be solved which have been thought of intractable just a few years ago. The ongoing intensive study of self learning systems that can solve complex problems has among other things surely been sparked by the success of the Alpha Go [74] algorithm, which for the first time was capable to defeat the top human players in the ancient Chinese board game of *GO*. This problem space – meaning the amount of possible combination of moves and positions on the game board – was deemed intractable to solve for a long time due to its large size of theoretically $2 \times 10^{170}$, which is more than the number of atoms in the observable universe. By letting an intelligent agent use reinforcement learning and play thousands of games against itself and incorporating prior recorded world class matches, it was possible to tackle the complexity of this domain and make it tractable for processing on modern highly parallel computing infrastructure.

Another landmark of evolution in intelligent systems comprises the work of the company *OpenAI*, whose engineers managed to apply self learning algorithms to a hide-and-seek environment [2]. Here the intelligent agents managed to develop strategies for winning the game by only specifying the rules of the game but not how to solve it. Strategies previously not thought of were discovered and perfected by the intelligent system again while competing with itself.

This is possible due to the recent advances of reinforcement learning algorithms paired together with feature extraction methods based on deep learning.

Although, these recent advances in algorithmic game playing deliver astonishing results, in many applications it is not possible to simulate the environment. Also most of the time

the computational power necessary for a massive parallel implementation that runs for days on thousands of cloud machines, is not available. Often problems can be reduced to work with predefined static or semi-automatic feature extraction and generation and the computational power available at runtime of the algorithm only permits a linear approximation structure. This then allows learning even in computationally restricted – such as embedded industrial setting – environments, where algorithms are much more constrained as to how much computation per algorithm iteration can be used.

## 1.1. Scope of this Work

In this thesis such reinforcement learning algorithms, based on a linear approximation structure with predefined or semi-automatic features, will be covered. The advantage of those algorithms is that they can be applied in restricted and embedded environments.

Let us begin by specifying the reinforcement learning problem that is to be solved by the algorithms covered in this thesis. The basis of it all is a Markov Decision Process. It consists of a state space $\mathcal{S}$, a space of actions that can be chosen in every state, a state transition probability specification that describes how likely the system transits from one state under taking a specific action to any other state and a immediate scalar reward upon transitioning from state to state.

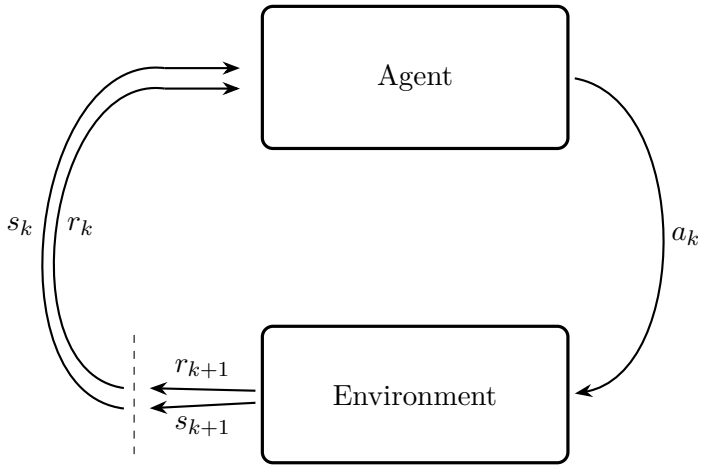



**Figure 1.1.:** Illustration of the agent environment interface (adapted from [77]).

Reinforcement learning now aims to find a suitable policy that selects in every state an action such that the cumulative reward received after each transition will be maximized. Figure 1.1 describes the interaction of the agent, in this case the reinforcement learning agent, with the environment. The agent selects according to its current policy an action and the environment responds to this action by returning the next state and the immediate reward associated with the state transition to the agent. The agent then in turn can use this reward to update its internal representation of the environment and the policy and can select the next action.

This type of behavior can be described as a sequential decision making system. As the interaction with the system goes on, the agent selects what to do in every state. Since one goal is to find the optimal policy, i.e. the policy that does the sequential decision making in such a way that the cumulative received reward will be maximized, we need a tool which measures how good the current realization of the policy is. Such a tool is the so called *value function*. This value function represents for each state how large the expected cumulative future reward would be, when starting in this current state and following the current policy thereafter.

Now we can divide the overall reinforcement learning problem into two stages. The so called *optimal control* problem, where the goal is to find the optimal policy and the *prediction* problem, where the goal is to find the value function for the current policy. In this thesis the prediction problem will be covered. This is useful, since if a value function for some policy is given, an improved policy can be constructed with the help of this value function by greedily selecting those actions that lead to states with a maximal future expected reward, i.e. maximum value function.

How can such a value function be envisioned concretely? If we go back to the example of the game of GO, then one state would be a certain configuration of black and white game pieces on the $19 \times 19$ board. The reward would be a positive quantity for winning and zero for all other moves. For every such board configuration, the value function would tell us how likely it is to win while following the current policy. Since it was already stated that all game configurations cannot be represented since there are simply too many of them, the value function has to be represented by some approximation.

For tackling the game of GO sophisticated representations, such as deep neural networks, are necessary. Since they are successful in approximating almost arbitrary complex functions, they are a popular tool of choice. However, using deep neural networks requires a

large amount of samples and computation to be available during training. In this work, the algorithms will be restricted to linear approximations where the states are each encoded by a set of features. This feature vector should have a dimension smaller than the cardinality of the state space. The value function can then be expressed as the inner product of the feature vector with some weight vector.

Now the prediction problem of finding the value function for some policy is reduced to finding the weight vector for the value function approximation such that it is as close as possible to the true value function. A measure how close the approximation is to the true value function for some weights, is given by the *Mean Squared Projected Bellman Error* (MSPBE).

Minimizing this cost function will yield the best possible approximate solution the the prediction problem which can be achieved in the limits of the approximation framework. This minimization can be done via closed form solution methods or – as will be covered in this thesis – by stochastic gradient descent algorithms. The most popular stochastic gradient descent methods for linear approximated reinforcement learning are the algorithms from the *Gradient Temporal Difference* (GTD) learning family. These algorithms will be investigated and extended in this thesis in the way as will be described in the next section.

## 1.2. Formulation of the Research Problem

As described above, the MSPBE is chosen as a cost function for gradient temporal difference learning. Another choice would be the unprojected form of the Bellman error. The primary motivation why the projection is incorporated, is the stability of the optimization procedure. Nevertheless, other algorithms, such as *Bellman residual minimization* (BRM) or the classical *Temporal Difference* (TD) learning algorithm exist, which aim at minimizing the unprojected or even a different cost function. This gives rise to the question:

- What are the motivations for choosing the projected cost function apart from numerical stability?

When speaking of stability of the optimization procedure, a related issue - the convergence in the presence of noise - comes to mind. To improve algorithm and convergence performance a technique often chosen is to introduce sparsity through regularization to the objective function. Another reason to sparsely regularize the cost function is the selection of

relevant features. This means, that in the feature representation, some components of the feature vector could be useful to solve the task at hand, while other should be discarded. Sparse regularization can act as such an automatic feature selection mechanism. However, existing sparse regularization approaches for reinforcement learning are often based on batch learning algorithms [39] and cannot be used in on-line learning systems or are complicated to implement [45]. The questions that arise from these facts are therefore:

- Can the simple $\ell_1$ sparsity promoting norm be introduced to on-line gradient based reinforcement learning?
- Does this technique help with convergence and noise-robustness of the algorithm?
- Can a sparsity promoting norm be used to achieve light feature selection?
- How does this simple on-line technique compare to the only other regularized algorithm known to the author, namely RO-TD [45]?

In general, for stochastic gradient descent algorithms, the convergence speed can be severely impacted by the way of sampling or noise in the process. Nesterov's accelerated gradient descent provides a method to speed up slow stochastic convergence for convex cost functions. So this seems as a promising fit for a combination and gives rise to the following questions:

- Does the MSPBE fulfill all conditions for the Nesterov accelerated gradient to be applied?
- How is the practical speedup of the new method in some usual benchmark problems?
- Can a combination of accelerated gradient descent and regularization even more improve noise robustness and convergence speed?

Finally, in reinforcement learning another important aspect is to be able to use samples in the learning process that are not generated by the distribution of the current policy $\mu$ but some other sampling distribution. This ability to converge under such conditions is called *off-policy* learning. Additionally, to increase learning performance, not only one step in the environment is used to estimate the return, but several weighted sample steps should be used. The remaining questions that arise are therefore:

- Do the previously introduced algorithms also work in an off-policy sampling setting and how is the performance?
- Can the algorithms be extended to multistep sampling?

## 1.3. Contributions of this Work

In this thesis I introduce a simple, yet effective regularization scheme for stochastic gradient descent temporal difference reinforcement learning. I also motivate the use of the projected Bellman error function as a basis for gradient reinforcement learning algorithms beyond just algorithmic justification but also look at the impact of parameters usually predetermined by the application.

The regularization scheme combines the iterative soft thresholding technique derived with a proximal formulation of the objective function. It enables to have a straightforward derivation of the algorithm, which is simple to implement in just a few additional instructions in each gradient descent step. Therefore, the beneficial computational properties of the stochastic gradient reinforcement learning algorithms are preserved.

The second algorithmic contribution combines the accelerated gradient technique for stochastic gradient descent to improve convergence speed for convex cost functions. As a prerequisite to applying this for gradient reinforcement learning, the necessary conditions for the cost functions have to be ensured. These conditions are convexity and L-Lipschitz continuity, which are derived for the projected Bellman error and proven to hold to varying degree, depending on the chosen feature representation.

Additionally, both types of modifications are combined into an accelerated and regularized variant and extended to be tested in off-policy and multistep sampling settings.

All algorithms are evaluated on several well known benchmark problems in reinforcement learning to demonstrate their claimed performance and to study the specific characteristics empirically which are improved robustness to noise, accelerated convergence and automatic feature selection.

The algorithms of this thesis can perform more stable learning in environments with the presence of additive Gaussian noise while still being of linear computational complexity in terms of feature size. On the benchmark problems they converge to the same value of error measure with one quarter of the samples as the original algorithms on average. Also, they

are able to converge to sensible results in terms of the problem domain with up to five times as many noise features as information bearing features present. This means that systems where computational power is limited and only a linear feature representation is admissible, the same results with four times less samples on average even if real world noisy sensor measurements are used.

## 1.4. Outline of this Thesis

A thorough mathematical formulation of the reinforcement learning problem and the studied algorithms are given in Chapter 2. Also in Chapter 2 the properties of the different possible objective functions are compared. An overview over regularization in gradient descent as well a study of related work are given in Chapter 3. In this chapter the first of the two main algorithms is derived and experiments to study the empirical performance are presented. Chapter 4 proofs convexity and L-Lipschitz continuity of the cost function and introduces the acceleration technique to the gradient reinforcement learning algorithms. In the same manner as in the chapter before, the algorithm is put to an empirical test to show performance in well known benchmark problems. In the last chapter, Chapter 5, both algorithm modifications are extended to the so called off-policy setting, which is an important property for algorithms in reinforcement learning. Also modifications for multistep sampling for even quicker convergence are introduced in this chapter. Chapter 6 concludes this thesis.

# Chapter 2.

# The Reinforcement Learning Setting

In this chapter the basics of reinforcement learning are introduced in finer grained details as in the introductory chapter above. The reason for this is to enable a concise notation of quantities in the chapters thereafter allowing to follow some mathematical rigor in the proofs in the chapters to come. The notation closely follows the notation of Bertsekas [5] with some parts borrowed from the introductory book of Sutton [77].

To begin the mathematical introduction and notations, we start by investigating and defining the Markov Decision Process.

## 2.1. Markov Decision Processes

A Markov Decision Process (MDP) is a 5-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, consisting of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition probability kernel $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, which indicates the probability of transiting from state $s \in \mathcal{S}$ to successor state $s' \in \mathcal{S}$ when taking action $a \in \mathcal{A}$, an immediate reward function $R(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ for the transition from state $s$ to $s'$ taking action $a$, and a discounting factor $\gamma \in [0, 1]$. Often the current state $s$ of the system is denoted as $s_t$ as the system is in that state at timestep $t$ and corresponding $s'$ is denoted as the state following timestep $t$, i.e., $s_{t+1}$ this is done to simplify notation when just any state and the successor should be taken into consideration irregardless in which timestep the system.

The actions in each time step $t$ are selected according to a stationary probabilistic policy $\mu(a|s)$, which gives the probability of taking action $a$ when in state $s$. Note, that it is often easier to write down a deterministic policy $\mu(s)$, which can be obtained from the probabilistic policy by sampling from the action distribution given state $s$. While interacting with the

system, the reward can be recorded and the main goal of RL is to find a so-called *optimal policy* $\mu^*$, which selects actions optimally with regards to maximizing the discounted return

$$R^{\mu}(s) = \sum_{t=0}^{\infty} \gamma^t R(s_t, \mu(s_t), s_{t+1}) = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{2.1}$$

where $r_t := R(s_t, a_t, s_{t+1}) = R(s_t, \mu(s_t), s_{t+1})$ is the immediate reward at timestep $t$ when choosing action $a_t$ in state $s_t$ according to policy $\mu$ starting with $s_0 = s$. This type of discounting is done, as already described in the first chapter, to give more weight to the immediate rewards but also to bound the sum when possibly dealing with an infinite amount of timesteps.

The RL problem can be devised into two types of problems: the first is to find the aforementioned optimal policy, called the *control problem*, the second is the so-called *prediction problem* or *policy evaluation* problem. Here it is the goal to find the associated *value function*

$$\begin{aligned} V &: \mathcal{S} \to \mathbb{R} \\ s &\mapsto V_{\mu}(s) = \mathbb{E}_{\mu}\left[R^{\mu}(s)\right] = \mathbb{E}_{\mu}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right], \end{aligned} \tag{2.2}$$

which indicates the future expected reward starting from state $s$ and following the policy $\mu$ thereafter. The value function associated with the optimal policy $\mu^*$ is denoted by $V^*$.

As the value function can be written in a recursive manner, it satisfies the *Bellman equation*

$$\forall s,\ V_{\mu} = \mathbb{E}_{P,\mu}\left[R(s, \mu(s), s') + \gamma V_{\mu}(s')\right] \tag{2.3}$$

for the value function associated with policy $\mu$ as well as the optimal value function $V^*$. The expectation in the Bellman equation is with respect to the policy as well as the state transition kernel $P$ according to which the successor state $s'$ is determined.

If the state space is finite, we can define $P^{\mu} \in \mathbb{R}^{n \times n}$ with elements $p_{ij} \in [0, 1]$ to be the stochastic state transition matrix, which specifies the transition probabilities from each state to each other when following policy $\mu$ in the environment specified by its own state transition matrix $P$. Additionally, let $R^{\mu} \in \mathbb{R}^n$ be the vector of average rewards when following the policy. Then the value function is represented by the vector $V_{\mu} \in \mathbb{R}^n$, for

which the the *Bellman operator* associated with $\mu$ is defined as

$$\begin{aligned} &\mathcal{T}_\mu : \mathbb{R}^n \to \mathbb{R}^n \\ &\quad \forall s, \quad V_\mu(s) \mapsto (\mathcal{T}_\mu V_\mu)(s) = \mathbb{E}_{P_\mu}\left[R(s, \mu(s), s') + \gamma V_\mu(s')\right]. \end{aligned} \tag{2.4}$$

The value functions associated with a policy unique fixed-points of the Bellman operator associated with this policy, i.e. $V_\mu = \mathcal{T}^*{}_\mu V_\mu$ [4]. Conveniently, knowing this leads to a first scheme of computation to determine the corresponding value function $V_\mu$, associated with a given policy $\mu$. This is called *policy evaluation* and can be written down as

$$V_\mu = \lim_{k \to \infty} \mathcal{T}_\mu^k V, \tag{2.5}$$

where $\mathcal{T}_\mu^k := \mathcal{T}_\mu \mathcal{T}_\mu \dots \mathcal{T}_\mu$ is the $l$-times repeated application of the Bellman operator $\mathcal{T}_\mu$. As $k$ goes to infinity, this will converge to the true value function when initialized with any starting vector $V$ as initial value function.

We can define a *greedy policy* with respect to any value function of any predefined policy $\mu_1$ as

$$\forall s, \ \mu_2(s) \in \arg\max_a \mathbb{E}_P\left[R(s, a, s') + \gamma V_{\mu_1}(s')\right]. \tag{2.6}$$

It is well established, that the performance with respect to the expected reward of policy $\mu_2$ will be better than or equal policy $\mu_1$, i.e. $\forall s, V_{\mu_2}(s) \geq V_{\mu_1}(s)$. Straightforwardly, this fact leads to another computation scheme to determine the optimal policy by alternating two steps, starting with some given policy $\mu_1$: the first one is to calculate the corresponding value function for the policy currently at hand, i.e., $V_{\mu_1}$. The second is to greedily improve the policy using $V_{\mu_2}$ and Equation (2.6) and obtain $\mu_2$. This is called the *policy improvement* step. Then again one goes back to the first step and calculates the value function corresponding the the new policy $\mu_2$. This method is called *policy iteration*.

Repeating the two steps of policy iteration, the final policy and value function it will converge to are the optimal ones due to proven convergence of the policy evaluation and policy improvement step as well as the fact that the number of possible policies is finite.

Another algorithm can be obtained by taking the optimal Bellman operator into consideration. A repeated application of it will lead, given any initialization value function $V$, to the optimal value $V^*$ as in

$$V^* = \lim_{k \to \infty} (\mathcal{T}^*)^k V. \tag{2.7}$$

The desired optimal policy is then given by a greedyfication of $V^*$ using Equation (2.6).

Although, at a first glance, it seems that value iteration is the straightforward way to calculate the optimal policy, when compared with policy iteration, in practice policy iteration is preferred. This is on the one hand due to practical restrictions, where policy evaluation and policy improvement steps can be only implemented approximately. This means, that the policy evaluation step is not conducted until possibly infinite steps, but terminated after a fixed limited amount of applications of the Bellman operator. The approximate greedyfication of the policy then might lead to a new policy, that is only slightly improved, but always better than the old policy. To summarize, it might require less computation steps in total than value iteration due to the improvement of intermediate value functions. Such customized and approximated policy iterations are now successfully implemented for example in robot control [42] or computer GO [74].

## 2.2. Value Function Approximation

For most applications the state space size $n = |\mathcal{S}|$ is large. This fact is often called the *curse of dimensionality*. It is therefore no longer possible to conveniently represent the value function as a vector, having a table of values for each state (this is why the previous approach is also called the *tabular* version). In such situations, it is necessary to employ approximations to represent the value function.

In general every thinkable method that takes states as input and is able to output the approximated scalar value is a possible candidate to be used in value function approximation. The most common approximations include sophisticated neural networks [81, 74] and other techniques such as kernels or Gaussian processes [22]. Often the most used and simplest approximation architecture is a linear framework [5]. For every state, a feature transform is defined, which maps each state $s \in \mathcal{S}$ to a corresponding feature vector $\phi(s) \in \mathbb{R}^l$ as in

$$\begin{aligned} \phi : \mathcal{S} &\to \mathbb{R}^l \\ s &\mapsto \phi(s), \end{aligned} \tag{2.8}$$

where $l$ is the dimension of features. We will sometimes denote $\phi_s = \phi(s)$ for the ease of notation and the feature vector $\phi_s$ is called the *feature vector* of $s$ whereas the components of this vector are called *features*.

An approximation should as closely as possible resemble the original value function, i.e. $V_\theta \approx V_\mu$. Here $V_\theta$ denotes the value function approximation using a linear architecture defined as

$$V_\mu(s) \approx V_\theta(s) := \phi(s)^\top \theta, \tag{2.9}$$

where $\theta \in \mathbb{R}^l$ is the parameter vector of the approximation. The approximated value function then lies in the space which is spanned by the columns of the feature matrix

$$\Phi := (\phi(s_1), \phi(s_2), \ldots, \phi(s_n))^\top \in \mathbb{R}^{n \times l}. \tag{2.10}$$

The feature space then can be defined as

$$\mathcal{V} = \left\{ \Phi\theta | \theta \in \mathbb{R}^l \right\} \tag{2.11}$$

where the columns of the feature matrix is a basis of this space. Most of the time the features are specially crafted vectors that are suitable for the task at hand to be solved. Often also automatically generated or general available feature representations are used, which introduces certain artifacts in the learning system. To cope with this, regularization, as described in Chapter 3 can be used.

In order to find a suitable approximation, an appropriate weight vector $\theta$ has to be found. This can be done by minimizing the squared distance between the ground truth value function and the approximation as in

$$\mathrm{MSE}(\theta) := \left\| V_\theta - V_\mu \right\|_2^2, \tag{2.12}$$

which is called the *mean squared error* (MSE), where $\|\cdot\|_2$ is the commonly known $\ell_2$-norm.

Unfortunately, when trying to solve the MDP, the target of $V_\mu$ is usually not available. As a substitute, we can aim our attention towards the fixed point of the Bellman operator. For the ground truth value function it should be that $V_\mu = \mathcal{T}_\mu V_\mu$ holds, the so called *Bellman error* $V_\mu - \mathcal{T}_\mu V_\mu$ should therefore be zero once we have the solution to the value function. The above MSE then can be substituted by a minimization of the *mean squared Bellman error*

$$\mathrm{MSBE}(\theta) := \left\| V_\theta - \mathcal{T}_\mu V_\theta \right\|_\xi^2, \tag{2.13}$$

where $\|x\|_\xi = \sqrt{\sum_i \xi_i x_i^2}$ is a weighted norm and $\xi \in \mathbb{R}^n$ is the steady state distribution

of the underlying Markov chain. If the Markov chain defined by $P$, is ergodic, irreducible and contains a single recurrent class, this means that all states are reachable from all other states and if a random walk is run in the chain, the probability of visiting each state is greater than zero, then the limiting distribution over states defined as

$$\xi(s) = \lim_{t \to \infty} \mathbb{P}(s_t = s) \tag{2.14}$$

exists. Here $\xi(s)$ is the component of the limiting distribution for state $s$.

As illustrated in Figure 2.1, the result of applying the Bellman operator may be not representable with the chosen features and therefore may lay outside of $\mathcal{V}$. It is therefore advisable to orthogonally project back the result on the feature space. This prevents ambiguities and ensures a unique solution exists. The resulting error function is called the *mean squared projected Bellman error* (MSPBE) and can be defined as

$$\text{MSPBE}(\theta) := \left\| V_\theta - \Pi_\mu \mathcal{T}_\mu V_\theta \right\|_\xi^2, \tag{2.15}$$

where the orthogonal projector is defined as $\Pi_\mu := \Phi(\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi$ with $\Xi = diag(\xi)$ is the matrix with the elements of the state distribution $\xi$ on its diagonal.

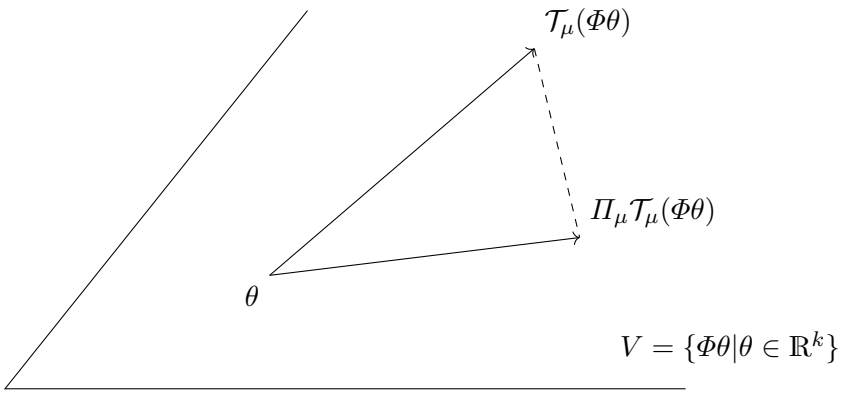



**Figure 2.1.:** Projection (dashed) of the MSBE on the linear space $V$ spanned by features $\Phi$.

In order for the projection to be computable, the steady state distribution $\xi$ over all states has to be existent.

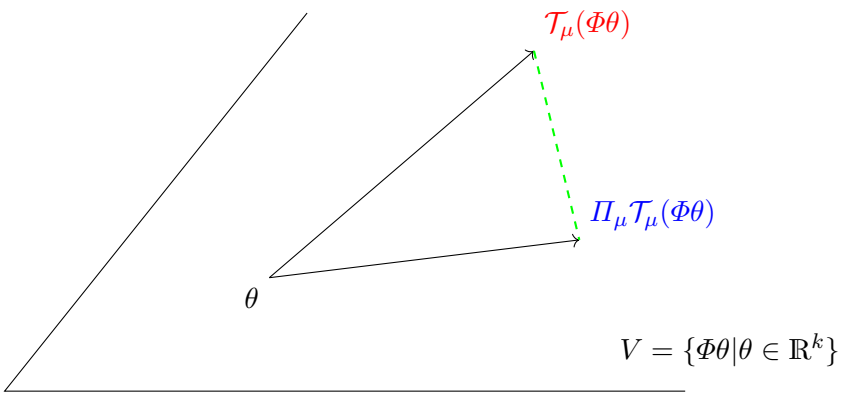



**Figure 2.2.:** The difference (green) in cost function for MSBE (in red) and MSPBE (in blue).

## 2.3. Sample Based Solution Methods

In order to introduce four sample based solution methods, *least squares temporal difference* LSTD learning, *least squares policy evaluation* LSPE, *temporal difference* (TD) learning and *gradient temporal difference* (GTD) learning, to obtain the parameters for the approximation of the value function $V_\theta$ for a given policy $\mu$, we need to reformulate the MSBPE. Therefore, let $\Phi = [\phi_1, \phi_2, \ldots, \phi_n]$ be the feature matrix containing all feature vectors in its rows, $V_\theta = \Phi\theta$ be the approximation of the value function, $\Pi_\mu = \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi$ the orthogonal projection onto the feature space according to the norm $\|\cdot\|_\xi$ weighted by the steady state distribution of the MDP and $\mathcal{T}_\mu$ be the Bellman operator induced by policy $\mu$. Note that it is assumed, that the unique steady state distribution $\xi$ exists due to ergodicity of the underlying Markov chain and that the feature matrix $\Phi$ has rank $l$ (dimension of features).

We then can define the MSBPE in terms of

$$\text{MSPBE}(\theta) := \|\Phi\theta - \Pi_\mu\mathcal{T}_\mu(\Phi\theta)\|_\xi^2, \tag{2.16}$$

which is the corresponding mean squared error formulation for the projected Bellman equation with linear value function approximation as in $\Phi\theta = \Pi_\mu\mathcal{T}_\mu(\Phi\theta)$.

Now it is advised to also reformulate the Bellman operator in terms of matrices. To this end let again $P \in \mathbb{R}^{n\times n}$ be the state transition probability matrix for the underlying problem and let $P^\mu \in \mathbb{R}^{n\times n}$ be the state transition probability matrix incorporating the

selection probabilities for actions according to policy $\mu$. Analogous to this, let $R_\mu \in R^n$ be the vector with components $r_i = \sum_{j=1}^{n} p_{ij} r(s_i, s_j),\ i = 1, \ldots, n$ containing the reward for each state under policy $\mu$ with $p_{ij}$ being the components of matrix $P^\mu$. Then the Bellman operator for policy $\mu$ can be written as

$$\mathcal{T}_\mu V = R_\mu + \gamma P^\mu V,\ V \in \mathbb{R}^n. \tag{2.17}$$

The solution for minimizing the MSPBE is now the solution to

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^l} \|\Phi\theta - (R_\mu + \gamma P^\mu \Phi\theta^*)\|_\xi^2, \tag{2.18}$$

which can be calculated by obtaining the gradient of Equation (2.18) with respect to $\theta$ and setting it to 0 as in

$$\Phi^\top \Xi \left(\Phi\theta^* - (R_\mu + \gamma P^\mu \Phi\theta^*)\right) = 0. \tag{2.19}$$

The straightforward solution to this can be obtained with matrix inversion by first multiplying the right hand side out into individual terms

$$0 = \Phi^\top \Xi \Phi\theta^* - \Phi^\top \Xi R_\mu - \gamma \Phi^\top \Xi P^\mu \Phi\theta^* \tag{2.20}$$

and then isolating $\theta^*$ on the right hand side while moving the middle term to the left hand side

$$\Phi \Xi R_\mu = (\Phi^\top \Xi \Phi - \gamma \Phi^\top \Xi P^\mu \Phi)\theta^*. \tag{2.21}$$

The derivation can be finished by isolating $\theta^*$ on the left hand side and simplifying the rest of the terms as in

$$\theta^* = (\Phi^\top \Xi \Phi - \gamma \Phi^\top \Xi P^\mu \Phi)^{-1}(\Phi \Xi R_\mu) = (\Phi^\top \Xi (I - \gamma P^\mu)\Phi)^{-1}(\Phi \Xi R_\mu) = C^{-1}d, \tag{2.22}$$

where $C = \Phi^\top \Xi (I - \gamma P^\mu)\Phi$ and $d = \Phi \Xi R_\mu$.

The drawback of this formulation is that there has to be computed a matrix inversion of a matrix of dimension $l$ and that the dimension of inner products for calculating $C$ and $d$ is in terms of number of states $n$, which can be huge. Unfortunately, we cannot get rid of the matrix inversion for now, but can reformulate the problem to estimate $C$ and $d$ while sampling and forming a sampling based estimate.

## 2.4. Least Squares Temporal Difference Learning and Policy Evaluation

The main idea to derive the sampling based methods LSTD and LSPE is to approximate the matrices $C$ and $d$ with their sampled counterparts $C_k$ and $d_k$, respectively. The algorithm step is denoted by $k$. Relevant quantities in this calculations can be regarded as expected values with respect to the steady state distribution $\xi$ such as in

$$\begin{aligned}
\Phi^\top \Xi \Phi &= \mathbb{E}_\xi \left[\phi(s)\phi(s)^\top\right] = \sum_{i=1}^{n} \xi_i \phi(s_i)\phi(s_i)^\top, \\
\Phi^\top \Xi P^\mu \Phi &= \mathbb{E}_{\xi,P^\mu} \left[\phi(s)\phi(s')^\top\right] = \sum_{i=1}^{n}\sum_{j=1}^{n} \xi_i p_{ij} \phi(s_i)\phi(s_j)^\top, \\
\Phi^\top \Xi R_\mu &= \mathbb{E}_{\xi,P^\mu} \left[\phi(s) r(s,s')\right] = \sum_{i=1}^{n}\sum_{j=1}^{n} \xi_i p_{ij} \phi(s_i) r(s_i,s_j) = \sum_{i=1}^{n} \xi_i \phi(s_i) r_i.
\end{aligned} \tag{2.23}$$

Assembling together these quantities to form $C = \Phi^\top \Xi \Phi - \gamma \Phi^\top \Xi P^\mu \Phi$ and $d = \Phi \Xi R_\mu$ and at the same time replacing the expected values with a sampled approximation of the expectation using $k$ samples of the form $(\phi(s_i), r(s_i, s_{i+1}), \phi(s_{i+1}))$, we obtain

$$\begin{aligned}
C \approx C_k &= \frac{1}{k+1} \sum_{t=0}^{k} \phi(s_t) \left(\phi(s_t) - \gamma \phi(s_{t+1})\right)^\top \\
d \approx d_k &= \frac{1}{k+1} \sum_{t=0}^{k} \phi(s_t) r(s_t, s_{t+1}).
\end{aligned} \tag{2.24}$$

As the authors present in [90, 89], the estimate of this two quantities can be calculated iteratively as samples are collected with

$$\begin{aligned}
C_k &= (1-\eta_k) C_{k-1} + \eta_k \phi(s_k) \left(\phi(s_k) - \gamma \phi_{s_{k+1}}\right)^\top \\
d_k &= (1-\eta_k) d_{k-1} + \eta_k \phi(s_k) r(s_k, s_{k+1}),
\end{aligned} \tag{2.25}$$

with stepsize $\eta_k = \frac{1}{k+1}$ and initial $C_0 = 0, d_0 = 0$. The authors also give further discussion and analysis on convergence of such an estimation method especially in conjunction with the LSTD algorithm presented in the following.

As presented in Equation (2.22), the matrix $C$ and vector $d$ can now be replaced in every

step with their empirically estimated counterparts $C_k$ and $d_k$. We then obtain the LSTD update step for the linear weights as

$$\theta_k = C_k^{-1} d_k. \tag{2.26}$$

Instead of formulating the problem in terms of finding the linear weights $\theta^*$ with one matrix inversion, the Bellman equation can be used to state an incremental update as in the update of the value iteration algorithm in Equation (2.5) for the optimal Bellman operator. The the optimal weights are then the solution $\theta^*$ of the fixed point equation

$$\Phi\theta = \Pi_\mu \mathcal{T}_\mu(\Phi\theta). \tag{2.27}$$

This hold true since the composed operator $\Pi_\mu \mathcal{T}_\mu$ is a contraction in the weights $\xi$ norm, cf. [5]. Reformulating the projection as a minimization problem then manifests as

$$\begin{aligned} \theta_{k+1} &= \arg\min_{\theta \in \mathbb{R}^l} \|\Phi\theta - \mathcal{T}_\mu(\Phi\theta_k)\|_\xi^2 \\ &= \arg\min_{\theta \in \mathbb{R}^l} \sum_{i=1}^{n} \xi_i \left( \phi(s_i)^\top \theta - \sum_{j=1}^{n} p_{ij}(r(s_i, s_j) + \gamma \phi(s_j)^\top \theta_k \right)^2 . \end{aligned} \tag{2.28}$$

If we now have $k$ samples available, the empirical minimization problem can be written down as

$$\theta_{k+1} = \arg\min_{\theta \in \mathbb{R}^l} \sum_{t=0}^{k} \left( \phi(s_t)^\top \theta - r(s_t, s_{t+1}) - \gamma \phi(s_{t+1})^\top \theta_k \right)^2 . \tag{2.29}$$

By setting the derivative of Equation (2.29) to 0,

$$0 \stackrel{!}{=} \sum_{t=0}^{k} \phi(s_t)^\top \theta \phi(s_t) - r(s_t, s_{t+1}\phi(s_t) - \gamma \phi(s_{t+1})^\top \theta_k \phi(s_t), \tag{2.30}$$

splitting the summation and bring parts of it to the left side of the equation

$$\sum_{t=0}^{k} \phi(s_t)^\top \theta \phi(s_t) = \sum_{t=0}^{k} r(s_t, s_{t+1}) \phi(s_t) + \gamma \phi(s_{t+1})^\top \theta_k \phi(s_t), \tag{2.31}$$

we can rearrange the terms

$$\sum_{t=0}^{k} \phi(s_t)\phi(s_t)^\top \theta = \sum_{t=0}^{k} \phi(s_t)\left(r(s_t, s_{t+1}) + \gamma\phi(s_{t+1})^\top \theta_k\right) \tag{2.32}$$

and obtain the update for the iterative LSPE algorithm as

$$\theta_{k+1} = \left(\sum_{t=0}^{k} \phi(s_t)\phi(s_t)^\top\right)^{-1} \left(\sum_{t=0}^{k} \phi(s_t)(r(s_t, s_{t+1}) + \gamma\phi(s_{t+1})^\top \theta_k\right). \tag{2.33}$$

## 2.5. Stochastic Gradient Descent

To derive the main algorithmic contributions of this thesis, the reinforcement learning problem will be reduced to an optimization problem of the MSPBE cost function. In the previous section one method with its advantages and drawbacks was already illustrated. Other optimization methods that will be used from here onwards are based on gradient descent and stochastic gradient descent. In this section a basic introduction to gradient descent algorithms will be given.

Let us define a cost function as $J(\theta) : \mathbb{R}^l \to \mathbb{R}$, where $\theta \in \mathbb{R}^l$ are the parameters of the function to be optimized. Further, assume that the data on which the cost function evaluates the cost is contained in the sample matrix $\Phi \in \mathbb{R}^{n \times l}$ as rows $\phi_i \in \mathbb{R}^l$.

Many problems can be solved directly, which means all data and a closed form description is available, as for example for quadratic problems. These solution methods often consist of calculating the inverse of a matrix (as for example in the previous section) which in most of the cases is of computational cost $O(n^3)$. The benefit is that the optimum can be found in one step of computation but at the same time the drawback is that this one step of computation is costly and for a large sample size infeasible to be calculated directly.

An alternative is to calculate the gradient $\nabla_\theta J$ and then use this gradient in a method called *gradient descent* [86] . Hereby, one starts out with an initial guess of the parameters $\theta_0$ and then takes steps in the direction of the negative gradient (for minimization) as in

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta J(\theta_k), \tag{2.34}$$

where $k$ is the iterate over the algorithm steps and $\alpha_k$ is a step size. If $\alpha_k$ is chosen properly

then the iterates $\theta_k$ will converge to a local minimum. If the function $J$ is convex and other conditions are met, then it can be guaranteed for gradient descent to converge to the global optimum $\theta^*$.

A drawback of gradient descent is that the data $\Phi$ has to be available in every step to be able to calculate the full gradient $\nabla_\theta J$. It can be assumed, that the data samples $\phi_i$ are generated by a certain distribution, then we can redefine an *expected cost function* [9] as an expectation over the samples as

$$J(\theta) = \mathbb{E}_\phi \left[ Q(\phi, \theta) \right], \tag{2.35}$$

where $Q(\phi, \theta)$ is the cost function for some specific sample $\phi$ and parameters $\theta$. Unfortunately, this expectation cannot be calculated since the underlying distribution over the samples is unknown. It is therefore necessary to approximate the costs by taking a finite set of independent samples and define the empirical cost as

$$\hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^{n} Q(\phi_i, \theta), \tag{2.36}$$

as well as doing the gradient descent on batches of size $n$ of data as in

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta \hat{J}(\theta_k) = \theta_k - \alpha_k \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta Q(\phi_i, \theta_k). \tag{2.37}$$

Every step in this type of iteration still relies on the computation of the gradient over the whole batch of samples available. This has the drawback for example, that the update has to be delayed until enough data is available for one algorithm update. We can, however, just update the parameters with every random sample as in

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta Q(\phi_k, \theta_k). \tag{2.38}$$

Algorithms of this type are called *stochastic gradient descent* algorithms. Solution methods of this type can be applied in situations, where an update of the solution is necessary as soon as a new sample is available. In general it is necessary that the update directions fulfill the following conditions [9]

$$\mathbb{E}_\phi \left[ \nabla_\theta Q(\phi, \theta) \right] = \nabla_\theta J(\theta). \tag{2.39}$$

This is the reason why in the later derivations in this thesis, the gradients are expressed in terms of expectations in order to obtain a stochastic gradient algorithm.

### 2.5.1. Temporal Difference Learning

The methods for minimizing the Bellman error in this chapter so far all fall in the category of batch methods. This means, that in the process of learning, an amount of samples gets collected and then the parameter vector $\theta$ gets determined such as for example in the LSTD case by a computationally costly $\mathcal{O}(n^3)$ matrix inversion, both in computation time and memory storage. Although the matrix inversion can be accelerated by using incremental versions utilizing the *Sherman-Morrison-Woodbury* update [73] for a rank-1 update of the inverse $C_k^{-1}$. However, as the time complexity reduces to some $\mathcal{O}(kn)$, the memory complexity still remains quadratic [31].

Ideally, an incremental algorithm only retains the weight vector and updates with each sample only requiring linear complexity. The remaining three algorithms to be introduced in this section all fulfill this requirement although each of them has their own advantages and drawbacks.

The probably most widely known an commonly used algorithm in reinforcement learning is the classic *Temporal Difference* (TD) learning, first introduced by Sutton [75, 77]. This method bases on the minimization of the cost function introduced in Equation (2.12) and a heuristic scheme to approximate unknown values by the method of *bootstrapping*.

Let us recall the mean squared error for a linear value function approximation given the ground truth value as

$$\mathrm{MSE}(\theta) := \|V_\theta - V_\mu\|_2^2 = \sum_{i=1}^{n} (V_\theta(s_i) - V_\mu(s_i))^2 = \sum_{i=1}^{n} (\phi(s_i)^\top \theta - V_\mu(s_i))^2. \tag{2.40}$$

Unfortunately, the quantity $V_\mu(s_i)$ is not available and has to be approximated. As it is known that $V_\mu$ is a fixed point of the Bellman operator, we can use this and the current estimate of the value function to approximate it empirically as $\hat{\mathcal{T}}_i : \mathbb{R}^n \to \mathbb{R},\ V \mapsto r_i + \gamma V(s_{i+1})$, cf. [29]. In effect the cost function for TD policy evaluation can be now written

as

$$
\begin{aligned}
J_{\mathrm{TD}}(\theta):\ \mathbb{R}^l &\to \mathbb{R} \\
\theta &\mapsto \mathbb{E}_{\xi,P_\mu}\left[\left(\theta^\top\phi(s_i) - \hat{\mathcal{T}}_i\theta_{k-1}^\top\phi(s_{i+1})\right)^2\right] = \mathbb{E}_{\xi,P_\mu}\left[\left(\theta^\top\phi(s_i) - r - \gamma\theta_{k-1}^\top\phi(s_{i+1})\right)^2\right] \\
&= \|\Phi\theta - R_\mu - \gamma P_\mu\Phi\theta_{k-1}\|_\xi^2,
\end{aligned}
\tag{2.41}
$$

where $\theta_{k-1}$ is the estimate of the value function parameter vector from the previous algorithm iteration $k-1$.

We can now calculate the gradient of the cost function for TD as

$$
\begin{aligned}
\frac{1}{2}\nabla_\theta J_{\mathrm{TD}}(\theta) &= -\Phi^\top\Xi R_\mu + \Phi^\top\Xi\Phi\theta - \gamma\Phi^\top\Xi P_\mu\Phi\theta_{k-1} \\
&= \Phi^\top\Xi\left(\Phi\theta - R_\mu - \gamma P_\mu\Phi\theta_{k-1}\right),
\end{aligned}
\tag{2.42}
$$

where the three summands in the first line of Equation (2.42) can be replaced by their respective expectation formulations as in Equation (2.23) and simplified as in

$$
\begin{aligned}
\frac{1}{2}\nabla_\theta J_{\mathrm{TD}}(\theta) &= -\mathbb{E}_\xi\left[\phi r\right] + \gamma\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi')^\top\right]\theta_{k-1} - \mathbb{E}_{\xi,P_\mu}\left[\phi\phi^\top\right]\theta \\
&= -\mathbb{E}_{\xi,P_\mu}\left[\phi(r + \gamma(\phi')^\top\theta_{k-1}^\top - \phi^\top\theta^\top)\right] \\
&= -\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta_{k-1},\theta)\right],
\end{aligned}
\tag{2.43}
$$

where $\delta_i(\omega,\theta) := r_i + \gamma(\phi')^\top\omega - \phi^\top\theta$ is called the *TD-error* and $\phi' = \phi(s_{i+1})$ is the feature for state $s_{i+1}$ in the sample for state $s_i$ with feature $\phi = \phi(s_i)$.

From Equation (2.43), we can derive the stochastic update rule to obtain the TD update at iteration $k$ for sample $(s_k, r_k, s_{k+1})$ as

$$
\theta_{k+1} = \theta_k + \alpha_k\phi(s_k)(r_k + \gamma\phi(s_{k+1})^\top\theta_{k-1} - \phi(s_k)^\top\theta_k),
\tag{2.44}
$$

where $\alpha_k$ is an appropriate step size that fulfills the Robbins-Monro [70] conditions for stochastic approximation, i.e. $\sum_{k=0}^\infty \alpha_k = \infty$ and $\sum_{k=0}^\infty \alpha_k^2 < \infty$.

Unfortunately, this TD learning algorithm can have stability problems, when used with value function approximations, such as we just derived. In the case of linear value function approximations, TD learning can diverge, when trained with samples generated by a differ-

ent distribution than the policy would generate, cf. [1] for further details and an example application. This is called *off-policy sampling* and will be discussed further in Section 5.2. If TD learning is coupled with a non-linear value function approximation, then it is possibly diverging even under *on-policy* updates, where the samples are generated by the policy $\mu$. An example and further discussion can be found in [84].

An explanation why the TD learning algorithm is unstable under certain conditions is, that it is not a gradient method by strict mathematical definition. Although, the update was derived by taking the gradient of Equation (2.41), the objective function itself already contains a approximation in the application of the empirical one-step Bellman operator. Maei [47] and Barnard [3] have further formal arguments supporting this claim.

To overcome these instability issues under approximation, Baird [1] proposed a true gradient method for temporal difference learning based on the work of Schweitzer and Seidmann [72]. This method and the improved gradient TD algorithms will be presented in the next section.

### 2.5.2. Gradient Temporal Difference Learning

In order to be able to derive a proper gradient based learning algorithm, we start off by taking the MSBE from Equation (2.13) into consideration as an objective function. The cost function for the *residual gradient* (RG) algorithm therefore has to be written as

$$\begin{aligned} J_{\mathrm{RG}}(\theta) : \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \|V_\theta - \mathcal{T}_\mu V_\theta\|_\xi^2 = \|\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2 \\ &= \left(\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta\right)^\top \Xi \left(\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta\right). \end{aligned} \tag{2.45}$$

In order to obtain the RG algorithm, the gradient of $J_{\mathrm{RG}}$ is derived[1] as

$$\begin{aligned} \frac{1}{2}\nabla_\theta J_{\mathrm{RG}}(\theta) &= \left(\Xi\Phi\theta - \Phi R_\mu - \gamma \Xi P_\mu \Phi\theta\right)^\top \left(\Phi - \gamma P_\mu \Phi\right) \\ &= \left(\mathbb{E}_{\xi, P_\mu}\left[\phi^\top\theta - r - \gamma(\phi')^\top\theta\right]\right)^\top \mathbb{E}_{P_\mu}\left[\phi^\top - \gamma(\phi')^\top\right] \\ &= -\mathbb{E}_{\xi, P_\mu}\left[r + \gamma\theta^\top\phi' - \theta^\top\phi\right] \mathbb{E}_{P_\mu}\left[\phi - \gamma\phi'\right] \\ &= -\mathbb{E}_{\xi, P_\mu}\left[\delta(\theta, \theta)\right] \mathbb{E}_{P_\mu}\left[\phi - \gamma\phi'\right]. \end{aligned} \tag{2.46}$$

[1] The full derivation of the RG cost function gradient can be found in Appendix A.1

This formulation is preferred to the TD cost function, because it is based on the gradient of the MSBE cost function without bootstrapped approximations, now is stable with respect to the aforementioned issues in off-policy and non-linear learning [47]. However, since we have the product of two expectations, the stochastic gradient algorithm would require two independent samples (cf. [18] for further details) of the state $s_{k+1}$ transited from state $s_k$. If it is assumed, that these two independent samples are available as $(\phi(s_k), r'_k, \phi(s_{k+1})')$ and $(\phi(s_k), r''_k, \phi(s_{k+1})'')$, then the stochastic gradient formulation at algorithm step $k$ could be written as

$$\theta_{k+1} = \theta_k + \alpha_k \left( r'_k + \gamma(\phi(s_{k+1})')^\top \theta_k - \phi(s_k)^\top \theta_k \right) \left( \phi(s_k) - \gamma \phi(s_{k+1})'' \right). \tag{2.47}$$

If only one sample is used assuming $r'_k = r''_k$ and $\phi(s_{k+1})' = \phi(s_{k+1})''$, then the RG algorithm converges not to the MSBE objective but to the mean squared temporal difference error, cf. [18, 47]. To cope with this problem without having the need to take double samples, which would require a simulator of the environment, Sutton et al. [79, 78, 47] have developed a gradient descent TD learning algorithm (GTD) based on the mean squared projected Bellman error (MSPBE). This algorithm is stable with off-policy sampling and claimed by the authors also to be stable with non-linear value function representations.

To derive the GTD family of algorithms, first let us establish two formulations of error functions that are going to be minimized by the method of stochastic gradient descent. The first formulation of error function is inspired by the above TD algorithm. In the optimum, the expected update of $\mathbb{E}_{\xi,P_\mu}[\phi\delta(\theta,\theta')]$ for the TD algorithm should be 0, where $\theta$ and $\theta'$ are the corresponding local instances of $\theta$ according to each algorithm step. Then an algorithm which minimizes this squared expected update error should be converging to the same solution in the optimum. This form of the error objective is often referred to as the NEU($\theta$) objective and leads to the first version of the *gradient temporal difference* GTD learning algorithm, as derived in [79] and can be written down as

$$\begin{aligned} J_{\mathrm{GTD}}(\theta) : \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \| \mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta')\right] \|^2 = \mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta')\right]^\top \mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta')\right]. \end{aligned} \tag{2.48}$$

The gradient of this objective is derived as follows, where it is assumed that $\theta = \theta'$

$$\begin{aligned}\nabla_\theta(J_{\mathrm{GTD}}) &= 2\,\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right]\mathbb{E}_{\xi,P_\mu}\left[\phi\nabla_\theta\delta(\theta,\theta)\right] \\ &= -2\,\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right]\mathbb{E}_{\xi,P_\mu}\left[\phi\nabla_\theta(\phi-\gamma\phi')^\top\right],\end{aligned} \tag{2.49}$$

and with the same technique, to avoid double sampling, we can write the update rule for the GTD algorithm as

$$\begin{aligned}\theta_{k+1} &= \theta_k + \alpha_k u_k \phi_k(\phi_k - \gamma\phi'_k)^\top \\ u_{k+1} &= u_k + \beta_k(\phi_k\delta(\theta_k,\theta_k) - u_k).\end{aligned} \tag{2.50}$$

Here $u(\theta)$ is again an estimate of the expectation $\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right]$ and $\alpha_k, \beta_k = \kappa\alpha_k$ are appropriate step sizes with the same requirements as for the later GTD2 algorithm. As it is shown in [78] and by experiments at the end of this chapter, that the GTD algorithm converges slower and in general is considered more unstable than its counterparts GTD2 and TDC, this algorithm will not further be considered in the derivations of algorithms in later chapters.

The second formulation of error function is again the already discussed projected Bellman error as for the LSTD and LSPE algorithms. Therefore, the objective function for the second two variants of the gradient TD algorithms is denoted as

$$\begin{aligned}&J_{\mathrm{TDC}}(\theta) : \mathbb{R}^n \to \mathbb{R} \\ &\qquad \theta \mapsto \|V_\theta - \Pi_\mu\mathcal{T}_\mu V_\theta\|_\xi^2 = \|\Phi\theta - \Pi_\mu\mathcal{T}_\mu(\Phi\theta)\|_\xi^2 = \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta))\|_\xi^2,\end{aligned} \tag{2.51}$$

since $\Phi\theta = \Pi_\mu(\Phi\theta)$.

After rewriting the cost function

$$\begin{aligned}J_{\mathrm{TDC}}(\theta) &= \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2 \\ &= (\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)))^\top\Xi(\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta))) \\ &= (\Phi\theta - R_\mu - \gamma P_\mu\Phi\theta)^\top\Xi^\top\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi(\Phi\theta - R_\mu - \gamma P_\mu\Phi\theta),\end{aligned} \tag{2.52}$$

the gradient of $J_{\text{TDC}}$ can be derived[2] as

$$\begin{aligned}\nabla_\theta(J_{\text{TDC}}(\theta)) &= 2\left(\Phi^\top \Xi \Phi - \gamma \Phi^\top P_\mu^\top \Xi \Phi\right)\left(\Phi^\top \Xi \Phi\right)^{-1}\left(\Phi^\top \Xi \Phi\theta - \Phi^\top \Xi R_\mu - \gamma \Phi^\top \Xi P_\mu \Phi\theta\right) \\ &= -2\,\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi - \gamma\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi(r + \gamma(\phi')^\top\theta - \phi^\top\theta)\right].\end{aligned} \tag{2.53}$$

The final form, suitable to derive a stochastic gradient descent scheme, can now be written as

$$\begin{aligned}\frac{1}{2}\nabla_\theta(J_{\text{TDC}}(\theta)) &= -\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi - \gamma\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi(r + \gamma(\phi')^\top\theta - \phi^\top\theta)\right] \\ &= -\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi - \gamma\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right] \qquad (2.54) \\ &= -\left(\mathbb{E}_{\xi,P_\mu}\left[\phi\phi^\top\right] - \gamma\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi')^\top\right]\right)\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right] \\ &= \mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right] + \gamma\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right]. \qquad (2.55)\end{aligned}$$

Unfortunately, this formulation still contains the product of several expected quantities and is therefore still susceptible to the double sampling problem as described above. In order to alleviate this problem, a least mean squares estimation $w(\theta)$ of the last two expected values is introduced, such that

$$w(\theta) \approx \left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right]. \tag{2.56}$$

An update of the auxiliary weights estimation can be derived by first assuming we are at update step $k$ having $w_k$ and moving the inverted matrix on the other side of the equation like

$$\mathbb{E}_\xi\left[\phi\phi^\top\right] w_k = \mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right]. \tag{2.57}$$

Then we can go further with subtracting the left side and combining the expectations as in

$$0 = \mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right] - \mathbb{E}_\xi\left[\phi\phi^\top\right] w_k = \mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta) - \phi\phi^\top w_k\right]. \tag{2.58}$$

[2]The full GTD2/TDC gradient derivation can be found in Appendix A.2

After further simplifications inside the expectation, we have

$$0 = \mathbb{E}_{\xi,P_\mu}\left[\phi(\delta(\theta,\theta) - \phi w_k)\right] \tag{2.59}$$

and a stochastic update of the auxiliary weights as in

$$w_{k+1} = w_k + \beta_k \phi_k(\delta(\theta_k,\theta_k) - \phi_k^\top w_k), \tag{2.60}$$

where $\beta_k = \kappa\alpha_k$ with $\kappa > 0$ is an adequate stepsize.

If we now derive the stochastic gradient formulation starting from Equation (2.54), we obtain the GTD2 algorithm as

$$\begin{aligned} \theta_{k+1} &= \theta_k + \alpha_k \phi_k(\phi_k - \gamma\phi_{k+1})^\top w_k, \\ w_{k+1} &= w_k + \beta_k \phi_k(\delta(\theta_k,\theta_k) - \phi_k^\top w_k), \end{aligned} \tag{2.61}$$

whereas if we start from Equation (2.55), we get the TDC algorithm as

$$\begin{aligned} \theta_{k+1} &= \theta_k + \alpha_k\left(\phi\delta(\theta_k,\theta_k) + \gamma\phi_k(\phi_{k+1})^\top w_k\right) = \theta_k + \alpha_k\phi_k\delta(\theta_k,\theta_k) - \alpha_k\gamma\phi_k(\phi_{k+1})^\top w_k \\ w_{k+1} &= w_k + \beta_k\phi_k(\delta(\theta_k,\theta_k) - \phi_k^\top w_k). \end{aligned} \tag{2.62}$$

The authors of [47, 27] state, that the stepsizes have to satisfy $\sum_{k=0}^{\infty}\alpha_k = \sum_{k=0}^{\infty}\beta_k = \infty$, $\sum_{k=0}^{\infty}\alpha^2 < \infty$, $\sum_{k=0}^{\infty}\beta_k < \infty$ and for GTD2 $\beta_k = \kappa\alpha_k$ with $\kappa > 0$, whereas for TDC $\lim_{k\to\infty}\frac{\alpha_k}{\beta_k} = 0$. This choice of stepsizes ensures that both algorithms are so-called *two-timescale* algorithms. This means that $\theta_k$ gets updated on a slower timescale and $w_k$ acts as if stationary from the view of the $\theta_k$ update. It can be observed, that the first part of the TDC update in Equation (2.62) is the same as for the classical TD algorithm. TDC corrects TD for its shortcomings by the means of a correction term, which is the second part of Equation (2.62). This is the reason why it is called TD learning with gradient Correction.

## 2.6. A closer Look at the Objective Functions

In the previous sections, sample based reinforcement learning algorithms were introduced, that minimize an objective function, based on the Bellman operator and the resulting Bell-

man error. The straightforward definition of an objective function, namely the MSBE, is taking the mean squared error of the Bellman error. The second approach, in order to increase stability and ensure nice algorithm behavior is to introduce a projection onto the span of features $\Phi$. This, of course, is only applicable in the presented way if a linear value function approximation is used.

Which objective function should we now use for the optimization? In order to answer this questions, we can look at several theoretical properties deduced from analyses of the corresponding objective functions. We will first start to take a look on the theoretical bounds of approximation quality using either objective. Second, we will shed more light comparing the classical TD method with gradient based methods based either on the MSBE and the MSPBE.

**Proposition 1.** *If we have $\theta^*$ as the unique solution of the linearly approximated projected Bellman equation $\|\Phi\theta - \Pi_\mu \mathcal{T}_\mu(\Phi\theta)\|_\xi^2$ and $V_\mu$ being the solution of the non-approximated Bellman equation $\|V - \mathcal{T}_\mu V\|_\xi^2$ and the fixed point of the Bellman operator, then it holds that*

$$\|V_\mu - \Phi\theta^*\|_\xi \leq \frac{1}{\sqrt{1-\gamma^2}} \|V_\mu - \Pi_\mu V_\mu\|_\xi, \tag{2.63}$$

*where $\mathcal{T}_\mu$ and $\Pi_\mu \mathcal{T}_\mu$ are contraction mappings with modulus $0 < \gamma < 1$ as in $\|\mathcal{T}_\mu V - \mathcal{T}_\mu V'\|_\xi \leq \gamma \|V - V'\|_\xi$ for some $V, V'$.*

*Proof.* We can use the Pythagorean theorem and the fact that $\Pi_\mu V_\mu$ is a contraction with modulus $\gamma$ and $V_\mu$ is a unique fixed point and derive

$$\begin{aligned} \|V_\mu - \Phi\theta^*\|_\xi^2 &= \|V_\mu - \Pi_\mu V_\mu\|_\xi^2 + \|\|\Pi_\mu V_\mu - \Phi\theta^*\|_\xi^2 \\ &= \|V_\mu - \Pi_\mu V_\mu\|_\xi^2 + \|\Pi_\mu \mathcal{T}_\mu V_\mu - \Pi_\mu \mathcal{T}_\mu(\Phi\theta^*)\|_\xi^2 \\ &\leq \|V_\mu - \Pi_\mu V_\mu\|_\xi^2 + \gamma^2 \|V_\mu - \Phi\theta^*\|_\xi^2. \end{aligned} \tag{2.64}$$

For a full proof refer to [5]. ■

This inequality in Equation (2.63) explains the best error that can be achieved when using linear value function approximation, which is the left side of the inequality. This means that the solution $\theta^*$ is this much away from the ground truth solution $V_\mu$. The right side is the upper bound of this error, which is dominated by the projection error of $V_\mu$ onto the row span of the features span($\Phi$). An illustration can be found in Figure 2.3, where the right

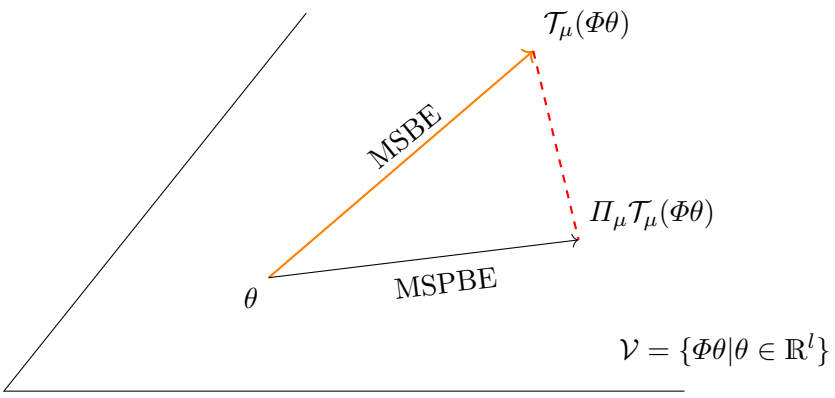



**Figure 2.3.:** MSPBE in the space spanned by the features. Projection error in red, Bellman error in orange.

hand side projection error is indicated in red. Additionally, this is unfortunately not the upper bound on the overall approximation error and the additional prefactor $\frac{1}{\sqrt{1-\gamma^2}}$ pushes this error farther out. This means that the overall approximation error on the left hand side is no longer tightly bounded from above if the discounting factor $\gamma$ is close to 1.

An illustration of this behavior can be seen in Figure 2.4. Here the error bound of the left hand side is indicated in orange, whereas the area in red indicates the possible error bound determined by the right hand side. The error in blue indicates the projection error of the value function. Note that with $\gamma$ approaching 1, this area grows quickly and therefore no definitive statement on the original approximation error can be made. Note that for a real norm on $\xi$, the visualization of the bounds are no longer necessary circular and are drawn here for illustration purpose.

On the other hand, a similar proposition as the above can be derived when working with the Bellman error without projection.

**Proposition 2.** *Let $\zeta^*$ be the unique solution of the linearly approximated Bellman equation without projection $\|\Phi\zeta - \mathcal{T}_\mu(\Phi\zeta)\|_\xi^2$ and $V_\mu$ being the solution of the non-approximated Bellman equation $\|V - \mathcal{T}_\mu V\|_\xi^2$ and the fixed point of the Bellman operator, it holds that*

$$\|V_\mu - \Phi\zeta^*\|_\xi \leq \frac{1+\gamma}{1-\gamma}\|V_\mu - \Pi_\mu V_\mu\|_\xi. \tag{2.65}$$

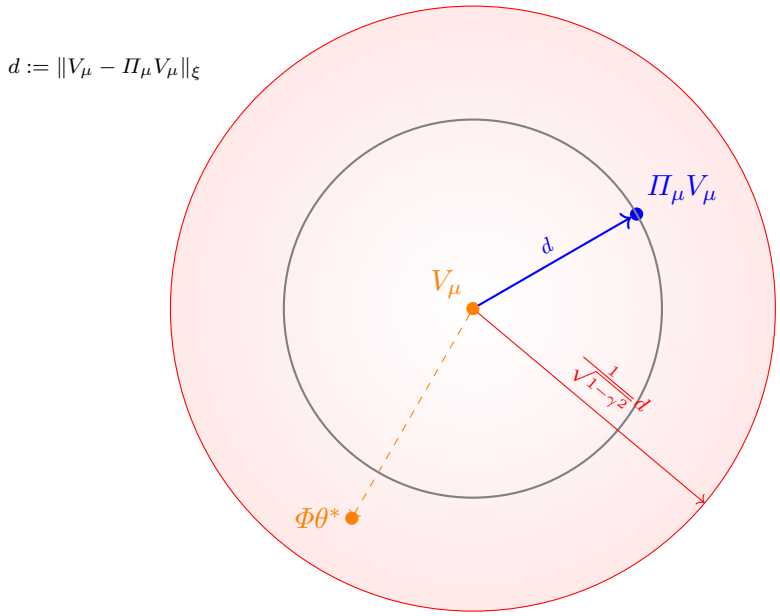



**Figure 2.4.:** Illustration of the error amplification by values of $\gamma$ close to 1.

*Proof.* We assume that the matrix $\Phi$ has rank $l$. This guarantees, that $\zeta^*$ is the unique fipoint to the unprojected Bellman equation $\Phi\zeta = \mathcal{T}_\mu(\Phi\zeta)$. Since the value function approximation space throughout this thesis is restricted to linear only, the space $\mathcal{V} = \{\Phi\theta | \theta \in \mathbb{R}^k\}$ is linear and therefore compact. We can further then establish that

$$\min_{V \in \mathcal{V}} \|V_\mu - V\|_\xi = \|V_\mu - \Pi_\mu V_\mu\|_\xi. \tag{2.66}$$

Then, we can adapt the proof technique from [40] and [59] to derive since $\|P_\mu\|_\xi = 1$

$$\|(I - \gamma P_\mu)^{-1}\|_\xi \leq \sum_{m=0}^{\infty} \gamma^m \|P_\mu\|_\xi^m \leq \frac{1}{1-\gamma}. \tag{2.67}$$

For any value function $V$, we then have by first adding and immediately subtracting $\mathcal{T}_\mu V$ and using the fact that $V_\mu = \mathcal{T}_\mu V_\mu$ and $T_\mu V = R_\mu + \gamma P_\mu V$

$$V_\mu - V = V_\mu - \mathcal{T}_\mu V + \mathcal{T}_\mu V - V = \gamma P_\mu (V_\mu - V) + \mathcal{T}_\mu V - V. \tag{2.68}$$

Then by moving $\gamma P_\mu(V_\mu - V)$ on the left side, we obtain

$$(I - \gamma P_\mu)(V_\mu - V) = \mathcal{T}_\mu V - V, \tag{2.69}$$

and can then substitute $V$ with $\Phi\zeta^*$ as in

$$\|V_\mu - \Phi\zeta^*\|_\xi \leq \|(I - \gamma P_\mu)^{-1}\|_\xi \|\mathcal{T}_\mu \Phi\zeta^* - \Phi\zeta^*\|_\xi. \tag{2.70}$$

Further we have

$$\|\mathcal{T}_\mu \Phi\zeta^* - \Phi\zeta^*\|_\xi = \min_{V\in\mathcal{V}} \|\mathcal{T}_\mu V - V\|_\xi \leq (1+\gamma\|P_\mu\|_\xi) \min_{V\in\mathcal{V}} \|V_\mu - V\|_\xi = (1+\gamma\|P_\mu\|_\xi)\|V_\mu - \Pi_\mu V_\mu\|_\xi. \tag{2.71}$$

and therefore can overall conclude that

$$\|V_\mu - \Phi\zeta^*\|_\xi \leq \|(I - \gamma P_\mu)^{-1}\|_\xi (1 + \gamma\|P_\mu\|_\xi)\|V_\mu - \Pi_\mu V_\mu\|_\xi. \tag{2.72}$$

With the known $\|P_\mu\|_\xi = 1$ and $\|(I - \gamma P_\mu)^{-1}\|_\xi = \frac{1}{1-\gamma}$ the solution can be obtained as

$$\|V_\mu - \Phi\zeta^*\|_\xi \leq \frac{1+\gamma}{1-\gamma}\|V_\mu - \Pi_\mu V_\mu\|_\xi. \tag{2.73}$$

■

If we now put the prefactors of these two bounds into relation and define

$$q(\gamma) := \frac{1+\gamma}{1-\gamma} - \frac{1}{\sqrt{1-\gamma^2}}, \tag{2.74}$$

then we can plot this function with regard to $\gamma$ as seen in Figure 2.5. From this plot it can be clearly concluded, that the prefactor for the MSBE bound grows much faster than the one for the MSPBE when $\gamma$ approaches 1. Therefore, from this standpoint of view, it is beneficial to prefer methods minimizing the MSPBE to ones that minimize the MSBE.

Another argument for using the projected Bellman error as an objective function is that the residual gradient method, which minimizes the unprojected Bellman error attains a solution, that is an upper bound to the methods working on the projected equation. Scherrer [71] conclude this in the following proposition.

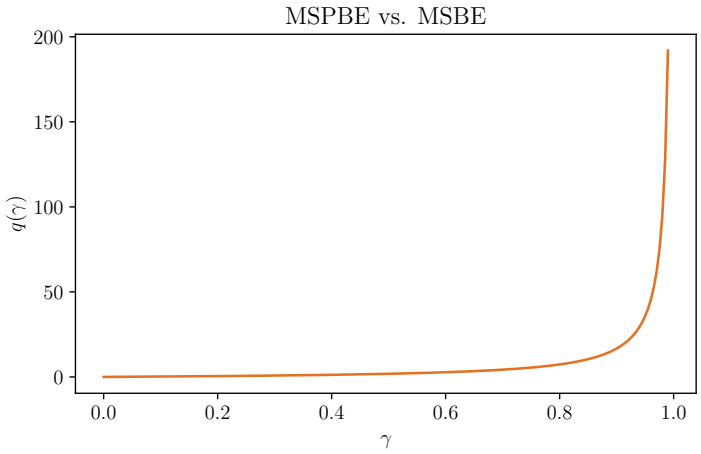



**Figure 2.5.:** Difference of prefactors $q(\gamma)$ for the bounds on MSBE and MSPBE.

**Proposition 3.** *[71] The RG is an upper bound of the TD error, and more precisely:*

$$\forall V \in \mathcal{V},\ E_{RG}(V)^2 = E_{TD}(V)^2 + \|\mathcal{T}_\mu V - \Pi_\mu \mathcal{T}_\mu V\|_\xi^2, \tag{2.75}$$

*where $E_{RG}$ and $E_{TD}$ are the minimal errors attained by the residual gradient and temporal difference algorithms, respectively.*

On the one hand the solution attained with temporal difference methods on the projected objective is better than the gradient solution on the unprojected method, on the other hand if the gradient method tends to zero it will force the temporal difference error towards zero.

In practice temporal difference methods are often used although the convergence is not always guaranteed if sampling does not adhere to the distribution induced by the policy $\mu$ (see Section 5.2 for details). This has two reasons: According to Scherrer [71], the residual gradient method has a higher variance under the influence of the sampling noise, which the temporal difference based methods are less susceptible to and, second, the residual gradient method requires double sampling in order to converge to the minimum of the unprojected objective function.

As the gradient based methods are theoretically more sound and enjoy better convergence guarantees [71, 47, 78, 79], especially if sampling is not done with respect to the distribution determined by the policy, those methods are to be preferred. In order to attain a smaller

minimal error, additionally methods based on the projected Bellman objective, such as the GTD2 and TDC algorithm are overall superior.

# Chapter 3.

# Regularization in Gradient Temporal Difference Learning

In this chapter one of the main contributions, namely regularization for stochastic gradient descent temporal difference learning, will be introduced. After motivating especially in the linear value function approximation case, existing work will be covered and the main algorithm, $\ell_1$-regularized gradient temporal difference learning, will be introduced. Also a comparison to one existing similar approach RO-TD will be investigated and a comparison of the differences of the algorithms of this chapter and RO-TD.

## 3.1. Introduction

As motivated in the previous chapter, the LSTD algorithm [13, 12, 40] due to being a batch algorithm is sample effective but cannot be used in an online setting. Additionally, the algorithm gets computational expensive not only in the number of samples, but also with increasing feature dimension. In a setting, where the features are high dimensional but there are not so many samples available, i.e. $l \gg n$, the LSTD method and methods derived from it tend to overfit [32]. This means that selecting the features that are most relevant in approximating the value function can often not be done a priori and the algorithm starts selecting some set of features that might seem to be relevant to the task, but only appear to be relevant when looking at the data at hand. If more data would have been available the selection would have been a different one.

When coupled with automatic feature generation, it is not always guaranteed that all features will contain useful information with regard to solving the task at hand, even some-

times there is no possibility to define a sensible way of predefined features and one has to resort to generic or automatic feature generation and extraction. Noise and sensory measurement errors contribute to the expressive uncertainty of the feature transformation and the features themselves. If such a noisy feature transformation is coupled with linear value function approximation, the noise naturally degrades learning performance of the value function weights $\theta$ and introduces bias.

One method to overcome these difficulties is to introduce regularization.

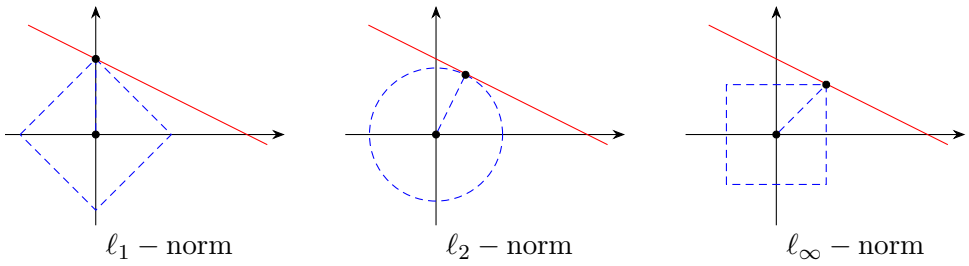



**Figure 3.1.:** Illustration of norms in two dimensional space.

The most common form of regularization for ill-posed problem is the so called $\ell_2$ *regularization* [63], by adding the term $\|\theta\|_2^2$ to the cost function. This additional norm assumes, that the parameter vector should be smooth and helps to solve these ill-posed problems. A huge benefit of this form of regularization is, that it often allows for closed form solutions and the overall cost function is then of the form

$$\hat{J}_2(\theta) = J(\theta) + \eta_1 \|\theta\|_2^2. \tag{3.1}$$

However, in a lot of situations, it is more appropriate to give only weight to a limited set of the features in the linear approximation scheme. The above regularization does not honor this. In those cases, $\ell_1$ regularization is more appropriate and this is why this form is often chosen in signal processing to do noise suppression in the presence of additive gaussian noise.

It forces many entries of the weight vector to be set to zero, i.e. not contributing to the final result. Additionally, it can be shown in e.g. logistic regression, that if the number of training samples compared to the dimension of the parameters is small, then such regularization performs in a superior way. The number of samples required in those cases only grows logarithmically in the number of feature dimensions opposed to the usual linear

growth [63], especially in the presence of irrelevant features, which do not contribute to the result. Therefore, $\ell_1$ regularization provides an implicit way of feature selection and provides for better interpretability of the results obtained [82]. The problem then is formulated as

$$\hat{J}_1(\theta) = J(\theta) + \eta_1 \|\theta\|_1. \tag{3.2}$$

Problems of this form can be usually solved using a so-called *Least Absolute Shrinkage and Selection Operator* (Lasso) [82] method. Another solution method for a stagewise forward selection of which coefficients to use, called *least angle regression* (LARS) [21], tends to obtain the same solution in a more efficient way. The drawback here is, that the solution is not guaranteed to be optimal and can achieve different final values, depending on the initialization used.

In some situations, however, the solution of the Lasso can be sub-optimal and the overall cost function is dominated not by the regularization, but by the original problem cost function [82, 26, 25]. In order to cope with these problems, the *elastic net* formulation was introduced as

$$\hat{J}(\theta) = J(\theta) + \eta_1 \|\theta\|_2^2 + \eta_2 \|\theta\|_1, \tag{3.3}$$

which combines both, $\ell_2$ and $\ell_1$ regularization [91].

A graphical overview of the different regularization behaviors can be seen in Figure 3.1. The leftmost figure describes how an $\ell_2$ norm would look in a two dimensional problem. Note that intuitively the penalty for large coefficients in any dimension is shared as the penalty can be described graphically as a ball (or circle in this case for two dimensions). On the other hand, in the center part of Figure 3.1, the $\ell_1$ norm is depicted. Compared to the $\ell_2$ variant, the $\ell_1$ penalizes especially along the coordinate axes. Intuitively, it would be best for e.g. feature selection and sparsity to have a norm, which is only penalizing the number of nonzero elements, i.e. the $\ell_0$ regularization, but this form is no longer convex. The $\ell_1$ can therefore be seen as a convex relaxation of the $\ell_0$ regularization making it better suited for optimization algorithms. As a comparison, on the rightmost side of the figure, the so-called $\ell_\infty$ (also sometimes referred to as sup-norm) norm is depicted. Here only the largest absolute values play a role in the penalty. This norm is not often used as a regularization variant, but we will encounter it again for looking a theoretical properties of MDPs and when proving the contraction properties of the Bellman operator $\mathcal{T}$.

The two fundamental regularized formulations, $\ell_1$ and $\ell_2$ have been applied to reinforce-

ment learning to improve convergence and provide feature selection, especially in situations, where the parameter vector is high dimensional and the number of samples available is limited. The next section introduces an overview of existing methods for solving regularized reinforcement learning and discusses applicability of each solution method. Additionally, in the section following, the proposed solution method of this work will be introduced and motivated.

## 3.2. Prior Art

Regularization techniques have been applied in reinforcement learning methods before. Generally, those can be divided into two major groups of algorithm types:

1. **Batch**: Batch methods build on the equivalent batch methods for their regularized counterpart in regression and supervised learning. This means that all the data is collected beforehand and then an optimization method is applied to all samples at once. This means that possibly a lot of data samples of state, action and reward tuples have to be stored for later processing. Additionally, most of the batch methods have at least quadratic computational complexity in the size of the feature vectors, which makes them unsuitable for applications in high dimensional scenarios. On the other hand, batch algorithms tend to be sample efficient. This means that with a comparatively low number of samples the methods achieve similar performance as sample-per-sample methods would only reach after a significant amount of additional samples.

2. **Stochastic Gradient**: Stochastic gradient methods update their model parameters after each sample received instantly. This means that the sample tuples do not have to be stored and as such those methods are storage space efficient as only the model parameters have to be held in memory or persisted to disk. Most of the stochastic gradient methods are also algorithmically very simple and therefore it is possible to implement such a method, when working in limited computational environments. A drawback of these methods is that they tend to converge rather slowly in the number of samples and therefore need a significant larger amount of samples to reach similar performance as batch methods. Additionally since the model is updated after

each sample, these methods can be susceptible to noise and high variance samples. Regularization is therefore a useful tool to improve the stability of these methods.

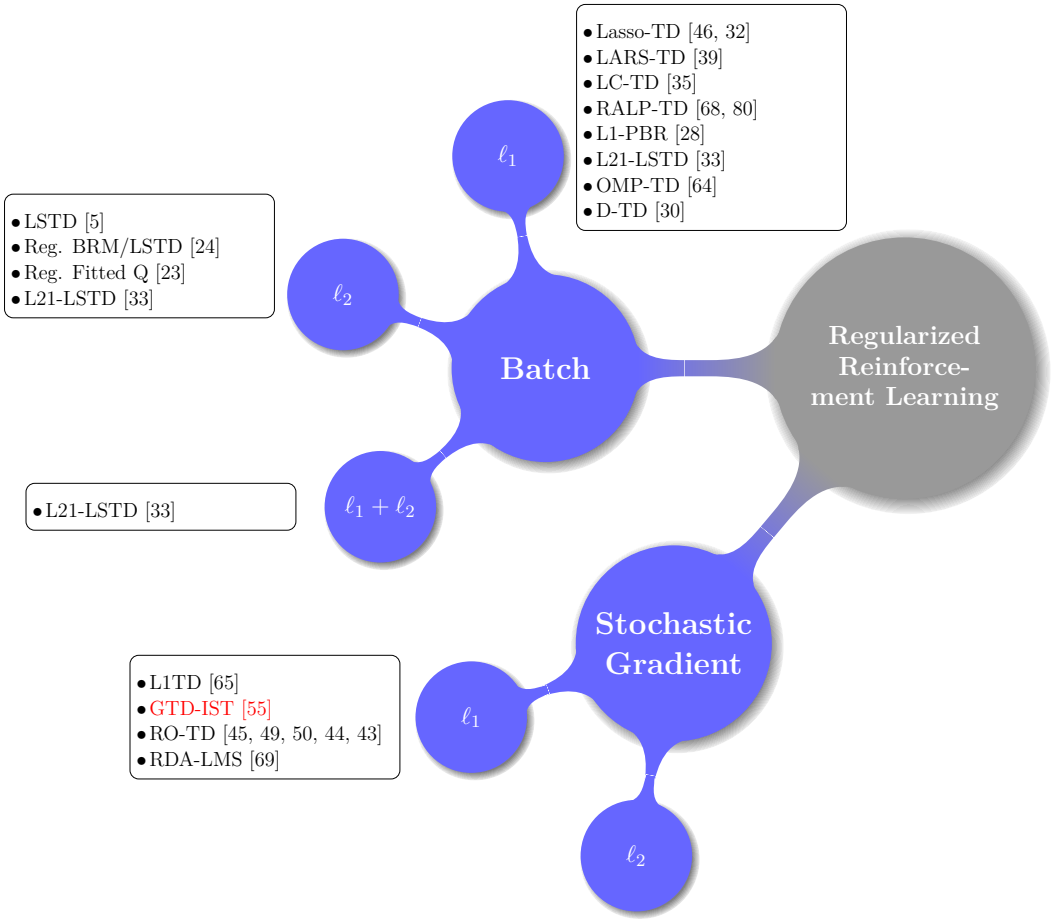



**Figure 3.2.:** Overview of the different types of regularized reinforcement learning.

Due to the high popularity and good performance, and well-behavedness of traditional batch least squares formulation of reinforcement learning algorithms, those were the first to be considered for extension to regularization methods. In the following an extensive list of this development in regularized batch methods will be covered in chronological order of publication to give a sense of the development in the field of this types of algorithms.

As one of the earliest methods, Loth et al. [46] use the so called Lasso formulation from Equation (3.2) to encourage feature selection and sparsity in the weight vector. They employ

the well known regression formulation of the LSTD algorithm (Section 2.4) and extend the loss function with an $\ell_1$ norm, by modifying the projected fixed point operator $\Pi\mathcal{T}$ of the LSTD formulation with a regularized formulation. Then they apply the equi-gradient algorithm on the batch sampled data. In [32], Ghavamzadeh et al. analyze the sample complexity of Lasso-TD in great detail and give performance bounds for various conditions. They can establish that the number of samples required for $\ell_1$ regularized methods only scales in terms of relevant selected features and not with the total amount of features.

Fahramand et al. [24] extend the above idea using methodologies from regression and regularization to introduce an $\ell_2$ regularized cost function. In order to achieve this, they perform regression on the sampled Bellman error for the state action function, which is related to the value function. Additionally, they provide an in depth analysis of the performance of such classification methods as a means of value function approximation in reinforcement learning. The authors apply the same technique to non-control tasks, such as BRM (see Section 2.5.2) and LSTD (see Section 2.4) and add a policy iteration scheme on top. Further, they analyze the sample complexity of those $\ell_2$ regularized algorithms, while pointing out that $\ell_1$ regularized version of those algorithms are worth considering.

Next, Kolter et al. [39], similarly to [46], add an $\ell_1$ penalty term to the projected Bellman error. They apply the least angle regression (LARS) algorithm [21] to solve the augmented cost function by formulating the problem as a set of optimality conditions similar to [38] and derive a method by maintaining an active set of features, that contribute to the solution, i.e. having a weight of non-zero. A drawback of this method is that the projection of the Bellman error is replaced by an $\ell_1$-penalized projection, which does not guarantee correctness any more as it does not correspond to the original Bellman error formulation. Additionally, the matrix of temporal difference errors, stacked together, needs to be a P-matrix in order for the solution to be correct. This assumption could be violated if the samples are obtained via off-policy sampling.

Johns et al. [35] reformulate the problem of the $\ell_1$ regularized projected Bellman error as a linear complementary problem. This enables to use nonspecialized solvers and provides warm starting capabilities, which better capture the iterative nature of the reinforcement learning problem, especially present when the algorithm is used in a policy iteration scheme. Unfortunately, since the formulation is in terms of the Bellman error fixed point, the requirement of the P-matrix still ensues.

Petrik et al. [68, 80] reformulate the minimization of the $\ell_1$ regularized MSPBE in term of

approximate linear programs. Similar to Johns et al. [35] they can use off-the-shelf solvers to tackle the problem.

Geist et al. [28] aim at alleviating the projection problems of LARS-TD [39] by adding the $\ell_1$ penalty in addition to the usual $\ell_2$ penalized projection of the Bellman error. Therefore, in contrast to LARS-TD, which searches for the fixed-point of the Bellman operator, this approach is convex and can therefore be solved by well known solution methods, which do not fail if the feature matrix is not a P-matrix. This advantage is bought, however, with a higher runtime and memory complexity.

The previous $\ell_1$-PBR approach gets extended by Hoffmann et al. [33] to incorporate both types of regularization, $\ell_2$ and $\ell_1$ in one formulation. Additionally, they introduce a data standardization step in order to have equal feature scaling and remove possible bias in feature selection.

Finally, Geist et al [30] apply the Dantzig selector (DS) [15] algorithm to the projected Bellman residual formulation. Since the Dantzig selector is a method for statistical learning in situations, especially where the number of features is larger than the number of samples, it is an adequate method to apply here. A benefit is that the reformulated problem can be efficiently solved by an interior point linear programming solver. It is to note, that the Dantzig selector penalizes the Bellman error not with the $\ell_2$ norm, but with the $\ell_\infty$ norm (sup- or max-norm), which measures the largest absolute value.

Due to the high computational complexity for the optimization of regularized batch methods, the interest in regularized stochastic gradient reinforcement learning methods, was sparked around the year 2012.

As one of the first to investigate stochastic gradient methods, Painter-Wakefield et al. [64], apply the orthogonal matching pursuit (OMP) [53] to feature selection in RL. Here features are selected greedily one after another depending on the decrease of the error residual. The work is related to [67], where new features are generated based on their decrease of the Bellman error. Here, however, the features are selected from a fixed, predefined set. It builds on the previous connections of matching pursuit algorithms such as [36, 51, 52] and it has to be mentioned, that this method indeed cannot be fully classified as a stochastic algorithm. In [65], the authors additionally derive an $\ell_1$ regularized version of the original temporal difference learning algorithm. The detailed derivation and discussion will be covered in Section 2.5.2.

The other stochastic gradient type of regularized reinforcement learning algorithm known

to the author, RO-TD [43], is closely related to the algorithm in this work, which will be described in the next sections. Therefore, an in-detail comparison with RO-TD will be covered in a separate Section 3.4.2.

## 3.3. Motivation of the proposed Approach

As was established in Section 3.1, the introduction of a regularization scheme for reinforcement is useful. As we have seen in that section, we have – given no other information about the problem is available – the reasonable choice between the $\ell_2$ and $\ell_1$ regularization or a combination of both.

If the main problem lies therein that the underlying problem might be under determined, then in a lot of cases the application of $\ell_2$ regularization is advised. On the one hand, in most of the cases it makes the problem again uniquely solvable and on the other hand in a lot of cases, there exists an analytical solution method. A good example for this behavior are again linear regression problems. A drawback is that this form makes no distinction between the different components of the data and weights all contributions equally. It can be beneficial for some problems where there exist especially noisy features, to only use a subset of the features of the provided data. Additionally, in the presence of noise, this has a big impact on algorithm performance when only coupled with $\ell_2$ regularization.

In order to alleviate those problems, $\ell_1$ regularization can be used. Similar to the $\ell_2$ variant, it helps making the overall problem again solvable for the case of underdetermined problems. This can happen, if the dimension of features is large and on the other hand the number of training samples is small compared to that.

To illustrate this further, one can regard the theory of Bellman error fixed points: If the feature matrix has full rank, this means that the dimension of the features is equal or smaller than the total number of unique states, then the overall problem of the MSBPE is strictly convex (as will also be discussed in Chapter 4). If now the feature transformation generates high dimensional feature vectors, which can easily happen for example when tile coding [76] or kernel features are used, then the full rank condition is no longer necessarily met. This means that the MSBPE is no longer strictly convex but only convex. A solution can then be found, but other equally beneficial in terms of the cost function exist. Then regularization helps to make the solution again unique, be it $\ell_2$ or $\ell_1$. The difference in

both regularization techniques now is in terms of noise resistance as will be elaborated in the next paragraph.

Another benefit of $\ell_1$ regularization is, that in the presence of noise, e.g. additive gaussian noise, the overall problem performance is better preserved. Imagine a problem, where some of the feature dimensions contain the useful information leading to a solution of the underlying reinforcement learning task, but some features contain irrelevant information that do not contribute to the task. Then this type of regularization discards these features and sets them to zero. The further iterations of the optimization algorithms then do no longer take these noisy feature dimensions into considerations leading to a better performance in accuracy. Additionally, also computational performance can be increased. This is due to the fact, that if the specialized implementation knows, that a feature dimension, whose weights are set to zero is not considered in the optimization process, then these can be left out of the overall calculation, whereas for $\ell_2$ regularization those still have to be considered as further samples could again contain nonzero information. In addition to a faster computational implementation, also memory requirements can be impacted positively by using specialized data structures, that honor the fact, that some components of the data can be discarded [41, 65].

Another interpretation of the $\ell_1$ regularization can be done in terms of feature selection. As was established before, the regularization promotes to have as many components of the weight vector set to zero. Those zero components effectively cause the feature dimensions corresponding to them to be discarded. This is useful if some of the features contain noise and therefore are not contributing to the solution of the problem or if some features are irrelevant to the solution and can be left out. The behavior of feature selection is especially useful in situations, where not specially crafted features for the underlying problem at hand can be specified and one has to rely on generic feature mappings or generic feature extraction mechanisms. In most of the generic feature mappings for reinforcement learning, such as *tile coding* [76], the number of features generated is large and possibly only a subset of them is useful in encoding the state of the physical system. Then feature selection can get rid of a lot of noncontributing features (by setting the respective weights to zero).

In this thesis $\ell_1$ regularization is applied to the cost functions of stochastic gradient temporal difference learning algorithms and subsequently optimization algorithms will be derived. Now, often it is assumed, that in online stochastic algorithms the data available is plentyful and therefore the situation, where batch algorithms profit when there are less

samples of data available than the feature dimension size, it is no longer valid. This might be true for the convergence and stability of the algorithm, but leaves out the fact, that sometimes even for batch data one would like to use a stochastic gradient algorithm because of computational framework conditions. The argument, that is even more convincing is that, even in the online setting the problem can be noisy and therefore slow to converge and instable. If this situation is given, $\ell_1$ regularization can therefore still be helpful.

## 3.4. Derivation and Algorithm

We have three regularized formulations of cost functions. These can now be applied to the cost functions introduced in reinforcement learning. As in this thesis the focus lies on $\ell_1$ regularization, only this variant will be further regarded.

Three basic cost functions exist, which we will closer inspect and add regularization in order to derive an online stochastic gradient descent optimization algorithm. The first cost function is the basis for the temporal difference (TD) learning algorithm, as introduced in Section 2.5.1

$$\begin{aligned} J_{\mathrm{TD}}(\theta):\ \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \|\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta_{k-1}\|_\xi^2. \end{aligned} \tag{3.4}$$

The second one is the projected mean squared Bellman error, which is the basis for the LSTD/LSPE and GTD2/TDC algorithms

$$\begin{aligned} J_{\mathrm{RG}}(\theta) : \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \|V_\theta - \mathcal{T}_\mu V_\theta\|_\xi^2 = \|\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2, \end{aligned} \tag{3.5}$$

and the NEU objective function of the GTD algorithm

$$\begin{aligned} J_{\mathrm{GTD}}(\theta) : \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \|\mathbb{E}_{\xi,P_\mu}\left[\delta(\theta,\theta')\phi\right]\|^2. \end{aligned} \tag{3.6}$$

Let us now in the same fashion as for the introduction of the $\ell_1$ regularization in Equation (3.2), formulate modified objective functions, where the respective modified variant is

indicated by $\hat{J}$ as

$$\begin{aligned} \hat{J}_{\mathrm{TD}}(\theta): \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \|\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta_{k-1}\|_\xi^2 + \eta\|\theta\|_1, \end{aligned} \tag{3.7}$$

for the temporal difference learning algorithm,

$$\begin{aligned} \hat{J}_{\mathrm{RG}}(\theta) : \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \|V_\theta - \mathcal{T}_\mu V_\theta\|_\xi^2 + \eta\|\theta\|_1 = \\ &\quad \|\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2 + \eta\|\theta\|_1, \end{aligned} \tag{3.8}$$

for the residual gradient algorithm family and

$$\begin{aligned} \hat{J}_{\mathrm{GTD}}(\theta) : \mathbb{R}^n &\to \mathbb{R} \\ \theta &\mapsto \|\mathbb{E}_{\xi,P_\mu}\left[\delta(\theta,\theta')\phi\right]\|^2 + \eta\|\theta\|_1, \end{aligned} \tag{3.9}$$

and

$$\begin{aligned} \hat{J}_{\mathrm{TDC}}(\theta) \ : \mathbb{R}^n &\to \mathbb{R} \\ \theta &\to \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta))\|_\xi^2 + \eta\|\theta\|_1 \end{aligned} \tag{3.10}$$

for the stochastic gradient temporal difference learning class of algorithms.

To minimize the regularized cost functions, we will derive a gradient descent algorithm for the exemplary problem of Equation (3.2) and afterwards apply it to the three objective function in Equations (3.7), (3.8) and (3.9). Since the method will be based on the method of subgradient descent [6], the definition of the subdifferential is stated as

**Definition 1** (Subdifferential)**.** *For a convex function $f : \mathbb{R}^N \to \mathbb{R}$, the subdifferential of $f$ at point $w$ is defined as*

$$\partial f(w) = \left\{ u \in \mathbb{R}^N \,|\, \forall y \in \mathbb{R}^N : (y-w)^\top u + f(w) \leq f(y) \right\}. \tag{3.11}$$

In order to minimize a problem like in Equation (3.2), one takes iterative steps in the subgradient directions by iteratively applying

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta J(\theta_k) - \alpha_k \partial\|\theta_k\|_1, \tag{3.12}$$

where $k$ is the iteration step and $\alpha_k$ is an appropriate step size. The problem now lies therein, that unfortunately the $\ell_1$ norm is not differentiable in the origin. Although most of the times, the evaluation happens in parts of the definition space, where differentiation is possible, the solution $\theta^*$ for the regularized problems mostly lies in points that are not differentiable, cf. [20].

In order to deal with such problems, that can be split into a differential subgradient part ($J$ in our case) and one non-differentiable part ($\|\cdot\|_1$), a two step algorithm, called *Forward-Backward Splitting* (FOBOS) [20] can be used. Here the minimization procedure is alternatively taking unconstrained subgradient steps for the first part and analytical minimization steps for the regularization part.

To derive the analytical minimization step, we need to define the concept of the proximity operator as for example in [56, 16] or originally in [57, 58].

**Definition 2** (Proximal Operator)**.** *Let $\psi$ be a real-valued convex function in $\mathbb{R}^N$, the the proximity operator of $\psi$ for $x \in \mathbb{R}^N$ is defined as*

$$\text{prox}_{\psi}(x) = \arg\min_u \left\{ \frac{1}{2}\|u - x\|_2^2 + \psi(u) \right\}. \tag{3.13}$$

The proximal operator is a replacement of the projection in projected subgradient methods, in order to get the result in the regularized domain. The advantage is now for certain functions $\psi$, a closed form solution can be analytically derived. Since our interest lies in the case, where $\psi = \eta\|\cdot\|_1$ is the $\ell_1$ regularizer, the derivation will be conducted with respect to these settings. The proximal operator for $\ell_1$ is denoted by

$$\text{prox}_{\eta\|\cdot\|}(x) = \arg\min_u \left\{ \frac{1}{2}\|u - x\|_2^2 + \eta\|x\|_1 \right\}. \tag{3.14}$$

The subdifferential then is derived and set to zero to obtain the closed form solution as

$$u = \text{prox}_{\eta\|\cdot\|}(x) \tag{3.15}$$

which is equivalent to

$$0 \in \partial \left( \frac{1}{2}\|u - x\|_2^2 + \eta\|x\|_1 \right). \tag{3.16}$$

As the first term is differentiable, we have

$$0 \in \eta \partial \|x\|_1 + u - x \tag{3.17}$$

and by moving $x - u$ over, the solution satisfies

$$x - u \in \eta \partial \|x\|_1. \tag{3.18}$$

With the element-wise subdifferential of the $\ell_1$ norm

$$\partial \|x_i\|_1 \in \begin{cases} [1] & \text{if } x_i > 0 \\ [-1] & \text{if } x_i < 0 \\ [-1, 1] & \text{if } x_i = 0 \end{cases}, \tag{3.19}$$

we get as the solution the soft-thresholding operator [19]

$$[\Psi_\nu(x)]_i = \begin{cases} x_i - \nu & \text{if } x_i > \nu \\ 0 & \text{if } -\nu \leq x_i \leq \nu \\ x_i + \nu & \text{if } x_i < \nu \end{cases}, \tag{3.20}$$

with $[\cdot]_i$ denoting the $i$-th vector element.

This can be rewritten as

$$\Psi_\nu(x) := \operatorname{sgn}(x) \odot \max\{|x| - \nu, 0\} \tag{3.21}$$

where sgn is the elementwise sign of the vector elements and $\odot$ is the elementwise multiplication of two vectors.

Since we now have derived the closed-form solution to the non-differentiable part of the problem, we can put together the full FOBOS algorithm as

$$\begin{aligned} \theta_{k+\frac{1}{2}} &= \theta_k - \alpha_k \partial_\theta J(\theta_k) \\ \theta_{k+1} &= \arg\min_\theta \left\{ \frac{1}{2} \|\theta - \theta_{k+\frac{1}{2}}\|_2^2 + \alpha_{k+\frac{1}{2}} \eta \|\theta\|_1 \right\}. \end{aligned} \tag{3.22}$$

Here, $\alpha_k$ is an appropriate step size and $\alpha_{k+\frac{1}{2}}$ is an interim stepsize, which can be set to

$\alpha_{k+\frac{1}{2}} = \alpha_k$ in the stochastic optimization setting and to $\alpha_{k+\frac{1}{2}} = \alpha_{k+1}$ in the batch setting [20]. The final notation for the stochastic optimization of equation (3.2) then is written down as

$$\begin{aligned} \theta_{k+\frac{1}{2}} &= \theta_k - \alpha_k \nabla_\theta J(\theta_k) \\ \theta_{k+1} &= \Psi_{\alpha_k \eta}(\theta_{k+\frac{1}{2}}) = \operatorname{sgn}(\theta_{k+\frac{1}{2}}) \odot \max\left\{\theta_{k+\frac{1}{2}} - \eta, 0\right\} \\ \Rightarrow \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k - \alpha_k \nabla_\theta J(\theta_k)\right), \end{aligned} \tag{3.23}$$

since the first part is differentiable and the gradient of $J$ is estimated with a stochastic sampling process.

### 3.4.1. Regularized Gradient Temporal Difference Learning

By taking the three regularized objective functions (3.7), (3.8) and (3.9), their respective gradients as well as the optimization scheme just derived, we can write down three regularized gradient algorithms for reinforcement learning.

The first is the well known temporal difference learning update, which gives us the algorithm update

$$\theta_{k+1} = \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \Phi^\top \Xi \left(\Phi\theta_k - R_\mu - \gamma P_\mu \Phi \theta_{k-1}\right)\right), \tag{3.24}$$

or in a stochastic gradient type formulation as

$$\begin{aligned} \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k(\phi_k(\phi_{k+1}^\top - \phi_k^\top)\theta_k + \phi_k r_k)\right) \\ &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \phi_k(r_k + \phi_{k+1}^\top \theta_k - \phi_k^\top \theta_k)\right). \end{aligned} \tag{3.25}$$

This version of a regularized TD learning algorithm is called L1TD and was studied in [65]. It enjoys proven convergence guarantees and is able to select the relevant features for a task even in the presence of noise. A drawback of this method is that regularized algorithms, derived from the TD formulations inherit all drawbacks of the original TD algorithms. These include possible instability when combined with nonlinear feature extractions, biased estimations of the value function and non-convergence when the samples are not collected on-policy, i.e. generated by the policy.

As discussed in the previous chapter, one possibility to remedy this situation is to derive the gradient descent formulation without relying on bootstrapping, i.e. basing the gradient

calculation on a potentially noisy estimation of the current value. The first derived algorithm in this direction is the Bellman residual minimization, which can be applied in the similar fashion to the iterative soft-thresholding formulation as in

$$\theta_{k+1} = \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k (r_k + \gamma \phi_{k+1}^\top \theta_k - \phi_k^\top \theta_k)(\phi_k - \gamma \phi_{k+1})\right). \tag{3.26}$$

This algorithm would then be called a regularized version of the residual gradient (RG) algorithm. Unfortunately, if independent double sampling is not a feasible option, this algorithm will still not converge to the desired task objective and also inherit the drawbacks of the original algorithm version, just as the regularized TD method.

In order to remedy the drawbacks of the two previously introduced regularized formulations, the formulation of the gradient temporal difference learning algorithms can be combined with the iterative soft-thresholding operator technique in order to obtain a method, that is both, more robust towards noise and outliers as well as reliably convergent to the minimum of the mean squared projected Bellman error.

Following the derivation in [55], the three update rules of GTD, GTD2 and TDC can be composed with the soft-thresholding operator in order to obtain three new methods for $\ell_1$-regularized online gradient TD learning. The first algorithm will be GTD-IST and the update is written down as

$$\begin{aligned} \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k u_k \phi_k (\phi_k - \gamma \phi_k')^\top\right), \qquad \text{(GTD-IST)} \\ u_{k+1} &= u_k + \beta_k (\phi_k \delta(\theta_k, \theta_k) - u_k), \end{aligned} \tag{3.27}$$

again with $u_k$ being the auxiliary weights, $\alpha_k$ and $\beta_k$ appropriate step sizes and $\eta$ a parameter controlling the influence of the $\ell_1$ penalty. The two other gradient TD methods GTD2-IST and TDC-IST then again are written down as

$$\begin{aligned} \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \phi_k (\phi_k - \gamma \phi_{k+1})^\top w_k\right) \qquad \text{(GTD2-IST)} \\ \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \delta(\theta_k, \theta_k) - \alpha_k \gamma \phi_k (\phi_{k+1})^\top w_k\right) \qquad \text{(TDC-IST)} \\ w_{k+1} &= w_k + \beta_k \phi_k (\delta(\theta_k, \theta_k) - \phi_k^\top w_k). \end{aligned} \tag{3.28}$$

### 3.4.2. Analytical Comparison to RO-TD

Liu et al. [45] derive a similar online regularized TD algorithm, but solve using a mirror-descent method. Their resulting algorithm is called *Regularized Off-policy convergent TD-learning* (RO-TD) and is re-stated and analyzed in several publications [49, 50, 44, 43]. This algorithm can be considered as the closest to the algorithm presented in this chapter.

Their derivation closely follows the steps for obtaining the TDC algorithm variants, but combines the two timescale update in a combined matrix vector update formulation. From the original derivation, we get the first equality condition for the auxiliary weights $w$ from Equation (2.55) as

$$\mathbb{E}_{\xi}\left[\phi\phi^{\top}\right] w = \mathbb{E}_{\xi,P_{\mu}}\left[\phi\delta(\theta,\theta)\right], \tag{3.29}$$

which can be expanded into the matrix formulation, as

$$\left(\Phi^{\top}\Xi\Phi\right) w = \left(\Phi^{\top}\Xi\Phi\theta - \Phi^{\top}\Xi R_{\mu} - \gamma\Phi^{\top}\Xi P_{\mu}\Phi\theta\right), \tag{3.30}$$

and the two vector $w$ and $\theta$ be extracted via

$$-\left(\Phi^{\top}\Xi\Phi\right) w + \left(\Phi^{\top}\Xi\Phi - \gamma\Phi^{\top}\Xi P_{\mu}\Phi\right)\theta = \Phi^{\top}\Xi R_{\mu}. \tag{3.31}$$

By setting the gradient $\frac{1}{2}\nabla_{\theta}(J_{\mathrm{TDC}}(\theta)) = 0$ to zero with substituting $w$, we can obtain another equality condition as in

$$\gamma\mathbb{E}_{\xi,P_{\mu}}\left[\phi(\phi')^{\top}\right] w = -\mathbb{E}_{\xi,P_{\mu}}\left[\phi\delta(\theta,\theta)\right], \tag{3.32}$$

again substitute the expectations with the matrix form

$$\left(\gamma\Phi^{\top}\Xi P_{\mu}\Phi\right) w = \left(\gamma\Phi^{\top}\Xi P_{\mu}\Phi\theta - \Phi^{\top}\Xi\Phi\theta + \Phi^{\top}\Xi R_{\mu}\right), \tag{3.33}$$

and extract $w$ and $\theta$ as in

$$\left(\gamma\Phi^{\top}\Xi P_{\mu}\Phi\right) w + \left(\Phi^{\top}\Xi\Phi - \gamma\Phi^{\top}\Xi P_{\mu}\Phi\right)\theta = \Phi^{\top}\Xi R_{\mu}. \tag{3.34}$$

These two equations (3.29) and (3.32) can then be put together in a combined matrix vector

equation system as follows

$$\begin{bmatrix} -\eta\Phi^\top \Xi\Phi & \eta\Phi^\top \Xi\Phi - \gamma\Phi^\top \Xi P_\mu \Phi \\ \gamma\Phi^\top \Xi P_\mu \Phi & \Phi^\top \Xi\Phi - \gamma\Phi^\top \Xi P_\mu \Phi \end{bmatrix} \begin{bmatrix} w \\ \theta \end{bmatrix} = \begin{bmatrix} \eta\Phi^\top \Xi R_\mu \\ \Phi^\top \Xi R_\mu \end{bmatrix} \Leftrightarrow Ax = b. \tag{3.35}$$

The corresponding empirical estimations of the quantities $A = \mathbb{E}[A_k]$, $b = \mathbb{E}[b_k]$ and $x = [w, \theta]^\top$ can be denoted as

$$A_k := \begin{bmatrix} -\eta\phi_k\phi_k^\top & \eta\phi_k(\phi_k - \gamma\phi_{k+1})^\top \\ \gamma\phi_k(\phi_{k+1})^\top & \phi_k(\phi_k - \gamma\phi_{k+1})^\top \end{bmatrix},\ b_k := \begin{bmatrix} \eta\phi_k r_k \\ \phi_k r_k. \end{bmatrix} \tag{3.36}$$

In short, the objective function optimized by the RO-TD algorithm including the sparse $\ell_1$ regularization on the weights $\theta$ then can be conveniently written as

$$J_{\text{RO-TD}}(x) := \|Ax - b\|_2 + \eta\|x\|_1. \tag{3.37}$$

By following the techniques stated in [61] and [37], Liu et al. [45] reformulate the objective function as a saddle-point problem as in

$$J_{\text{RO-TD}}(x) = \|Ax - b\|_2 + \eta\|x\|_1 = \max_{\|y\|_n \leq 1} y^\top (Ax - b) + \eta\|x\|_1, \tag{3.38}$$

where $n = 2$ is the conjugate number $\frac{1}{m} + \frac{1}{n} = 1$ corresponding to the original problem fit of $m = 2$ ($\ell_2$-norm) and $y = [y_1, y_2]^\top$ is a vector of the same structure as $x$. This reformulated objective function can now be solved by an iterative procedure by applying

$$\begin{aligned} x_{t+\frac{1}{2}} &= x_t - \alpha_t A_t^\top y_t \quad , \quad y_{t+\frac{1}{2}} = y_t + \alpha_t (A_t x_t - b_t) \\ x_{t+1} &= \Psi_{\alpha_k \eta}\left(x_{t+\frac{1}{2}}\right) \quad , \quad y_{t+1} = \Pi_n\left(y_{t+\frac{1}{2}}\right), \end{aligned} \tag{3.39}$$

where $\Pi_n(x) := \min\left(1, \frac{1}{\|x\|_n}\right) x$ is the projection of x on the $\ell_n$ unit-ball, $\Psi$ is the already well known thresholding operator, $x_0 := 0, y_0 := 0$ and $\alpha_t$ an appropriate step size. To have optimal computational cost, the update steps involving the matrix $A$ and vector $b$ are decomposed into separate updates for the vector components of $y_k = [y_{1,k}, y_{2,k}]^\top$ and

$x_k = [w_k, \theta_k]$ and can be written down as

$$A_k^\top y_k = \begin{bmatrix} -\eta\phi_k\phi_k^\top y_{1,k} - \gamma\phi_k\phi_{k+1}^\top y_{2,k} \\ \eta\phi_k\phi_k^\top y_{1,k} - \gamma\eta\phi_k\phi_{k+1}^\top y_{1,k} + \phi_k\phi_k^\top y_{2,k} - \gamma\phi_k\phi_{k+1}^\top y_{2,k} \end{bmatrix}, \tag{3.40}$$

and

$$A_k x_k - b_k = \begin{bmatrix} -\eta\phi_k\phi_k^\top w_k - \eta\phi_k\delta(\theta_k, \theta_k) \\ \gamma\phi_k\phi_{k+1}^\top w_k - \phi_k\delta(\theta_k, \theta_k). \end{bmatrix} \tag{3.41}$$

Liu et al. [45] argue against the objective function in Equation (3.10), where the sparse penalty is simple added to the MSPBE cost function because the derivation of the gradient in their opinion does not admit an analytical derivation and is difficult to compute. However, in the derivations of the subgradient and the proximal formulation of the simple sparse augmented cost function Equation (3.10), this argument was shown to be not valid and a more straightforward derivation and solution method is indeed possible. It is worth noticing, that the formulation of Liu et al. [45] admits straightforward application of standard methods to analyze the sample complexity of the resulting algorithms.

Mahadevan et al. [48], extend the RO-TD algorithm to variable basis adaptation. Here, the regularization not only selects from a fixed basis $\Phi$, but a nonlinearly parametrized basis $\Phi(\alpha)$ is assumed, whose weights $\alpha$ are adjusted in a two-timescale scheme.

In parallel, Qin et al. [69], first derive a batch version of $\ell_1$ regularized RL which can be solved by a method of alternating directions [88] applied in basis pursuit problems for compressive sensing. Those methods also employ a shrinkage operator in the gradient descent method. Then they derive an online version of the algorithm, called regularized dual averaging (RDA) [87], which in the same fashion as the previous algorithms imposes a proximal penalty and solves using the soft thresholding stochastic gradient method. Note, that this method similarly as GTD/2/TDC-IST, solves for the Bellman errors which use a $\xi$ weighted norm, which ensure the proper convergence and fixed point properties. RO-TD [45] on the other hand solves for a square weighted Bellman error ($\|\cdot\|_2$ instead of $\|\cdot\|_\xi$) which is only valid if the feature basis is orthonormal and the steady state distribution of the underlying MDP is the identity, i.e. $\Xi = I$. This could be an explanation for the observations by White and White [85], where the mirror-prox versions of gradient TD learning (of which RO-TD is one example) performed poorly compared to the original versions of GTD/2 and TDC.

## 3.5. Experiments

In order to evaluate the empirical behavior and performance of the derived algorithms they were tested on several well established standard experiments in reinforcement learning. All those experiments are simulations to highlight the behavior of the algorithms with regard to specific difficulties. In the following, two different simulated test environments and their specifics will be presented. Some of these environments will also be used in the next chapters about accelerated algorithms, multistep methods and off-policy variants of those algorithms. Additional, specialized experiment variants to highlight the specific properties of the respective algorithms will be introduced in the corresponding chapters.

#### Random Markov Decision Process

The first evaluation to test a new RL algorithm is to check the ability for finding an adequate set of weights in a random Markov decision process. Here the convergence speed can be tested in environments, to which the solution is well known and a squared error to the ground truth solution $\theta^*$ can be calculated in every iteration step of the learning procedure. Of course some algorithms behave differently and scale with the number of states and the feature dimension. Therefore, two different random MPDs were generated. The first one is referred to as the *small random MDP* and it consists of $n = 30$ states, each with $n_a = 4$ actions to choose from. The feature size is $l = 10$ randomly generated features individually for each state. This means that for each state a specific random vector $\phi(s)$ is generated by sampling each dimension from a normal distribution, which is unique and stays the same for every state throughout the experiment. The reward for each action in every state is also generated randomly from a normal distribution as well as the transition probabilities $P$ from each state to every other state taking a specific action. These rewards and transition probabilities are stationary in the experiment. The second random MDP is the *big random MDP* and is generated in the same fashion as the small one with the difference that there exist $n = 400$ states, $l = 200$ feature size and $n_a = 10$ actions in every state. The discount factor for both MDPs is arbitrarily set to constant $\gamma = 0.95$. The policy for evaluating the value function on the random MDPs was chosen to be uniform randomly selecting between the possible actions in each state and the starting state distribution of this environment is chosen uniformly random over all states.

### Inverted Pendulum

The random test environment consist of discrete state and actions. To test the performance of the algorithms on continuous problems, two variations of inverted pendulums as introduced in [18] are used in this work. The first environment is a linear cart pole balancing task. Here, a movable cart is attached to a linear rail on which it can move left or right, refer to Figure 3.3. The action are then the horizontal forces on the car to the left and the right. The state feature vector consists of the position of the cart $x$, the velocity $\dot{x}$, the angle of the pole with respect to the upright position $\upsilon$ and the angular velocity $\dot{\upsilon}$, i.e. $s = (x, \dot{x}, \upsilon, \dot{\upsilon})^\top$. The

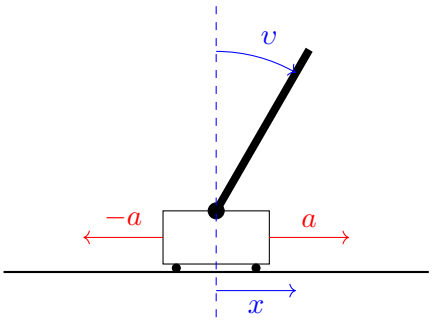



**Figure 3.3.:** Linear cart pole balancing task.

pendulum is initialized in the upright position and controlled via a linear stochastic policy $\pi(a|s) = \mathcal{N}(a|s^\top \theta_p, \sigma^2)$ with a small amount of random exploration controlled by $\sigma$. The introduced noise can be controlled by increasing the noise coefficient added to the car acceleration component, that is used to determine the next state in the simulation. Using as a feature all components of the state vector and their squared products additional to a constant bias term $\phi_p(s) = (1, s_{(1)}, s_{(1)}s_{(2)}, s^2_{(1)}, s_{(3)}, s_{(1)}s_{(4)}, s_{(2)}, \ldots, s^2_{(4)})^\top \in \mathbb{R}^{11}$, the linearized problem can be solved exactly by dynamic programming and the optimal policy can be determined. This feature set is referred to as *perfect features*, whereas an *approximate feature* (or imperfect) set can be defined by only using the squared state components (due to symmetry of the problem) and one additional bias feature $\phi_a(s) = (1, s^2_{(1)}, s^2_{(2)}, s^2_{(3)}, s^2_{(4)})^\top \in \mathbb{R}^5$. The discounting factor is set to $\gamma = 0.95$ and the reward is defined as

$$r(s, a) = r(x, \dot{x}, \upsilon, \dot{\upsilon}, a) := -100\upsilon^2 - x^2 - \frac{1}{10}a^2. \tag{3.42}$$

This can be interpreted as heavily penalizing deviations from the upright position, penalizing deviations from the middle position as well as lightly penalizing applying big forces to the left or right.

## 3.6. Experiment Methodology

As the main focus of this chapter is to introduce new algorithms with respect to noise tolerance and feature selection, All the above experiments were modified to allow for some varying amount of noise to be introduced. These variants of the experiments will then be reused in later chapters, where the noise will be combined with on- and off-policy variants to show the performance of the combined algorithms.

#### Simulation of Noise

As the main goal of the soft thresholded algorithms is to make gradient temporal difference learning more robust to noise and outliers. All experiments are modified in similar ways: First the feature vector gets concatenated with a second vector $\phi_\sigma \sim \mathcal{N}(0, \Sigma^2)$ only consisting of Gaussian distributed noise components. This second vector is of length $l_\sigma$ resulting in a overall feature dimension of $l + l_\sigma$. To study the influence of the ratio between meaningful original features (up to component $l$) and noisy features, the amount of noisy components will be varied and indicated in each experiment. Additionally, the magnitude of the noise will be varied by adjusting $\Sigma$, which will be a diagonal matrix containing for each noise feature component the magnitude of the noise. To test the influence of normalization, in every experiment it will be indicated if the overall resulting feature vector $(\phi(s), \phi_\sigma)^\top$ is normalized to overall length 1 or not.

#### Hyperparameter Optimization and Performance Criterion

In order to allow for a fair algorithm comparison, for each environment and setting within this environment (e.g. number of added noisy features), an extensive hyperparameter search was conducted. This search was conducted as a gridsearch for all applicable parameters. This means that any combination of a subselection of parameter values was run in every experiment for 20 independent runs and the average of the performance criterion was

recorded. The best parameter set is then determined by selecting those with the best average performance criterion. The parameter ranges considered were

- learning rate $\alpha \in [0.001, 0.004, 0.007, 0.01, 0.04, 0.07, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1.0, 4.0]$,
- the factor to derive the secondary weights,
  $\kappa \in [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16]$,
- the regularization parameter,
  $\eta \in [0.0, 0.001, 0.006, 0.01, 0.04, 0.07, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.5, 2.0]$ and
- the acceleration parameter $\nu \in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$

To determine the algorithm performance it is most straightforward to directly select the mean squared Bellman error as criterion as this both indicates fit in terms of the original cost functions for each of the algorithms in this thesis as well as the performance of the resulting value function in terms of the underlying problem. This follows from modeling the problem to be solved as a Markov decision process and the direct correspondence of the weighted Bellman error with this problem modeling.

In unknown problem domains, it is usually not possible to exactly calculate the Bellman error since the steady state distribution $\Xi$ and the projection $\Pi_\mu$ onto the features $\Phi$ are not known since the state space can be too big or otherwise no approximation through features would have been used in the first place. To be able to calculate an exact error, fortunately the problem domains are well behaved in terms of modeling and can be solved via dynamic programming algorithms (e.g. policy evaluation) and the ground truth value function $V^*$ is therefore easily determinable. The steady state distribution can be approximated with extensive sampling as this only has to be done once for every domain considered since the policies over which they are evaluated stay the same throughout the whole thesis.

## 3.7. Results

In the following sections, the results of studying the modified algorithms are presented. First the convergence speed in terms of the number of samples to reach a certain value of error measure are studied. Afterwards, the influence of the problem domain size and therefore the size of feature vector is looked upon. Since the algorithms aim to improve the convergence

performance in the presence of noise, the influence of additional additive Gaussian noise features is investigated, followed by a look at the parameter sensitivity. The section closes with an experimental comparison to RO-TD and the algorithm performance in continuous experiment domains.

### 3.7.1. Convergence Speed

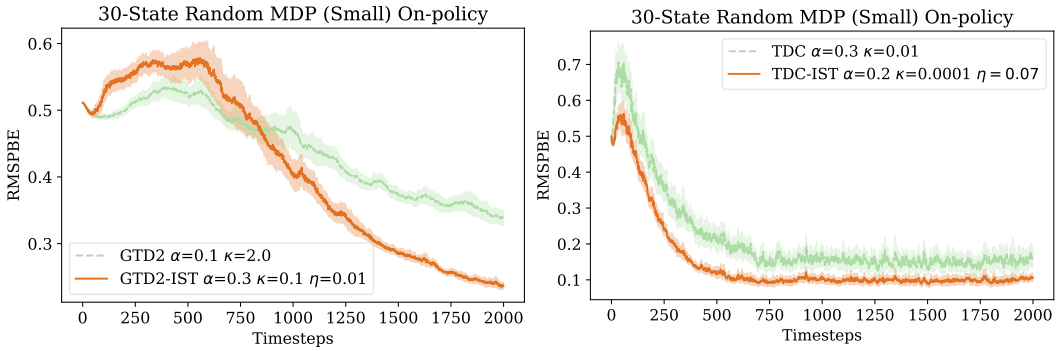
To investigate basic convergence properties the first set of experiments will be conducted on a small random MDP as described in Section 3.5. This is beneficial due to the well-behavedness of this designed experiment. It constitutes an ideal environment which fulfills all the theoretical prerequisites for convergence which is not always given in more complex or real-world applications.

Both algorithms, GTD2-IST and TDC-IST, show similar convergence behavior, especially in the case of the TDC variants in Figure 3.4b, where convergence to similar final error values can be observed. For GTD2 this is similar if the experiment is run for more than 2000 timesteps. On the x-axis, the sampling timesteps are denoted, while on the y-axis the Bellman error can be seen. For this and all future plots of this thesis, the mean across multiple independent runs is depicted with a bold colored line, while the variance across the runs is represented by the shaded area around. In general the soft-thresholding not only seems to help with noise robustness and feature selection, but also with convergence speed in terms of samples used to reach a certain value in error measure. In the case of the random MDP, this is somewhat expected as the sampling due to the policy and also in the feature generation inherently introduce some noise in the overall sampling process.

#### Convergence Problems of GTD

As can be seen in Figure 3.5, the GTD variant of the algorithms has problems in finding the correct parameter set for convergence. Even with an exhaustive gridsearch it was not possible to find an optimal parameter set with satisfying convergence. Previous papers (e.g. [17]) give a parameter set and plots in which this algorithm variant seems to consistently decrease in error measure. If the same experiment with those parameters is ran for 4000 instead of 2000 episodes, a similar behavior as in Figure 3.5a can be observed. This is just masked by the fact that the experiments were terminated after 2000 episodes.

Looking at the experiment on the big random MDP in Figure 3.5b, slightly different

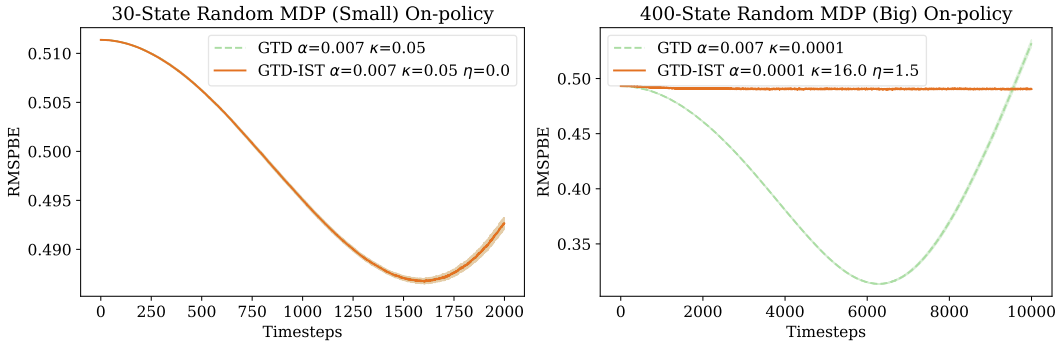



**(a)** GTD2 **(b)** TDC

**Figure 3.4.:** Comparison of algorithm modification for the different GTD variants.

behavior can be observed. At first both, the original and modified version of the algorithm seem to consistently decrease in error, but after a certain amount of timesteps the error measure increases again and for the thresholded version stabilize around a much higher value than for the other algorithm variants (GTD2 and TDC). The original version of the algorithm, even after letting the experiment run for a long time, still seems to diverge. Due to computational limits, it could not be explored, whether unmodfied GTD exposes a similar oscillatory behavior as GTD-IST.

In the following, due to these undeterministic behavior, all variants of the GTD formulation of the algorithms have not been considered further.



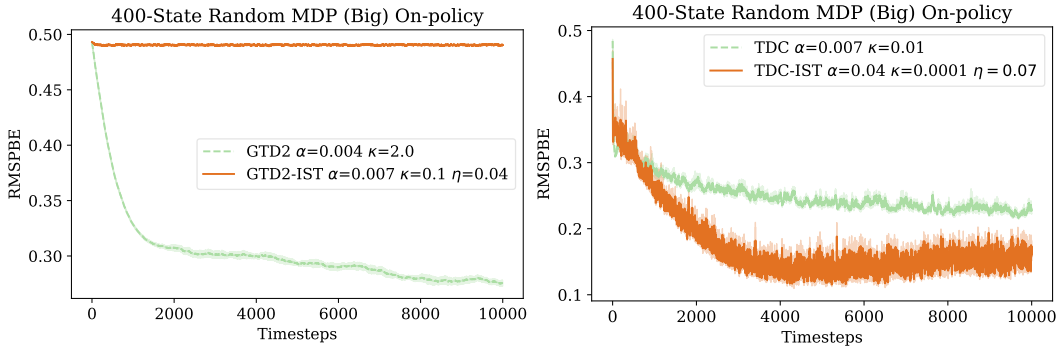

**(a)** Small random MDP **(b)** Big random MDP

**Figure 3.5.:** Comparison of convergence problems of the GTD algorithms on random MDP variants.

### 3.7.2. Size of Domain

The next experiment is to compare the soft-thresholded algorithms on a larger version of the randomly generated MDP (with 400 states). The increase in difficulty here is twofold: first, from a computational view, domain is of course more challenging as the state vectors and the sampled feature vectors are larger. Second, this domain also inherently introduces more sampling noise and difficulty in approximation with a linear structure. The experiments are therefore run over 10000 timesteps instead of 2000 as for the small MDP.

The modified as well as the original GTD2 and TDC algorithm variants can both deal suitably well with the increased complexity of the larger experiment domain, as can be seen in Figure 3.6. Overall – as expected – the convergence of both algorithms is a bit slower and the soft-thresholded variants can keep their convergence speed advantage. Other algorithms, however, seem to have problems to cope with the increased features size and noise, as can be seen in the comparison to RO-TD in Section 3.4.2.




**(a)** GTD2 **(b)** TDC

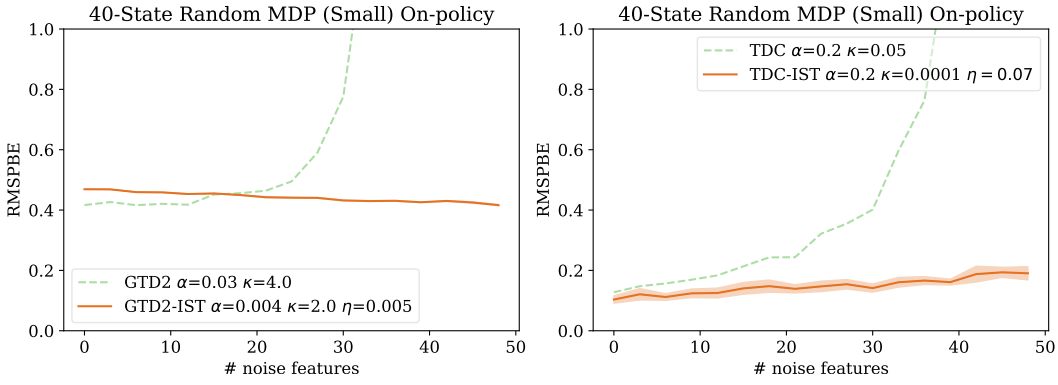
**Figure 3.6.:** Comparison of modified algorithms for the different GTD variants on large Random MDP.

### 3.7.3. Noise Sensitivity

One main goal of the thresholded algorithm modifications is to make the learning process more robust to noise. This behavior is evaluated by letting the algorithms run on an environment with a fixed amount of noisy features added (10 for the small random MDP, 200 for the big random MDP). Then the best parameters were found by the usual gridsearch

procedure. Afterwards, while keeping those parameters fixed, the amount of noise features is increased step by step and the final error is evaluated at the end of each experiment. This is done for 20 independent runs.

As visible in Figure 3.7, the behavior on the small random MDP is as expected: The unmodified algorithm versions degrade in convergence performance as more and more noise features are added to the environment. The thresholded algorithms are able to decrease the weight on the noise feature and are able to select those features meaningful for the correct solution of the environment. On the x-axis of both plots, the amount of noise features are denoted and on the y-axis the averaged error measure at the end of each experiment.




**(a)** GTD2 **(b)** TDC

**Figure 3.7.:** Noise sensitivity for varying amount of noise features

The GTD2 variants of the algorithms in Figure 3.7a even seem to be able to decrease the error measure by selecting the correct features up to the performance of the non-thresholded variant of the algorithm with no noise added. The TDC-IST variant of the algorithm in Figure 3.7b significantly outperforms the original version by allowing just a small increase in error measure whereas the original version seems to diverge for a great amount of noisy features added. Keep in mind that the amount of noisy to non-noisy features is highly skewed where $l = 10$ original non-noisy features stand against $l_\sigma = 50$ noisy features on the rightmost of both graphs.

### 3.7.4. Sparsity

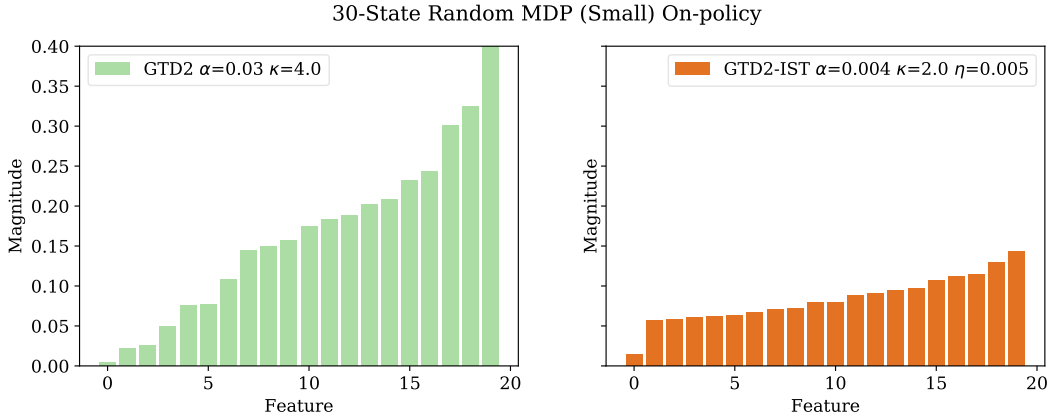
Building on the observations in the previous sections, that the thresholded algorithms can cope with an increasing amount of noise well, the next thing to study now is the structure of the solution vector. Intuitively, one would expect a correlation of a sparser solution vector with a better algorithm performance in noisy environments. This is investigated in Figure 3.8. Here on the x-axis the individual feature weights of the solution vector $\theta$ are listed sorted by their magnitude. On the y-axis the magnitude of the individual feature components is denoted.

Upon studying the figures, one cannot conclude that the above hypothesis is correct. Rather the solution tends to stay similar with respect to sparsity of the weight vector. Overall the magnitude of the individual weight components seem to shrink in general with thresholding, which is a natural tendency of the algorithm and is to be expected. This might already be enough in attenuating the noise features in order to improve the overall solution performance. Especially, when looking at the largest weight, the GTD2-IST algorithm manages to bring down its magnitude significantly as it is off-scale in the non-thresholded setting.

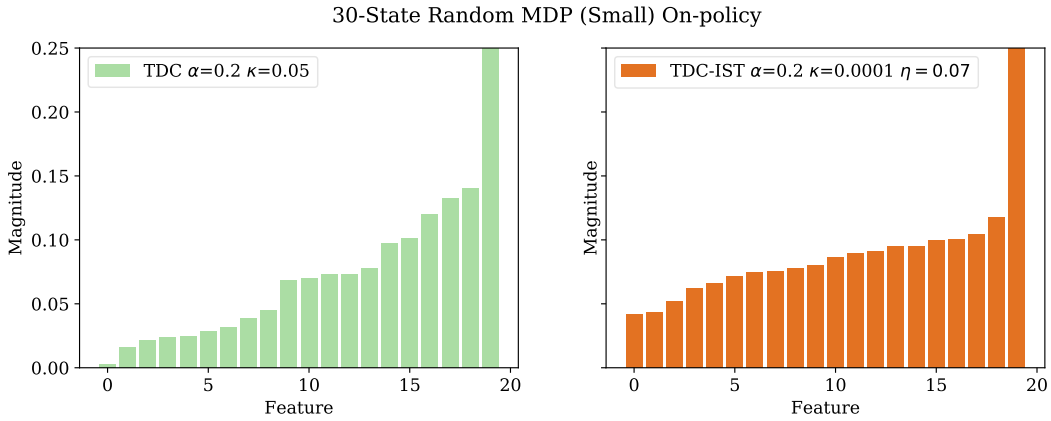
To further investigate this, a second similar experiment is conducted, but with 40 instead of ten noise features. The result can be seen in Figure 3.9. For the GTD2 algorithm, the thresholding amplifies the relative difference between noisy and non-noisy features, as can be seen in Figure 3.9a, where a increase in the sorted weight magnitudes can be observed. For the TDC algorithm in this setting, the thresholding is especially efficient in bringing the magnitudes down. In the left half of Figure 3.9b, the unmodified TDC algorithm assigns large weights across almost all feature vector components, while the regularization brings the feature magnitudes down to a reasonable level. Be aware that the y-axes limits of those two sub-figures are in very different ranges. The relative magnitude of the feature weights compared to the maximum feature weight magnitude are – like for GTD – emphasized and compared to information-bearing features, the noise features only receive small weights.

### 3.7.5. Parameter Sensitivity

Since the modified variants of the GTD algorithms have one additional parameter, the tuning of such poses some additional difficulty in optimally selecting the best working point in a system that uses this type of algorithms. However, even with only the two original




**(a)** GTD2



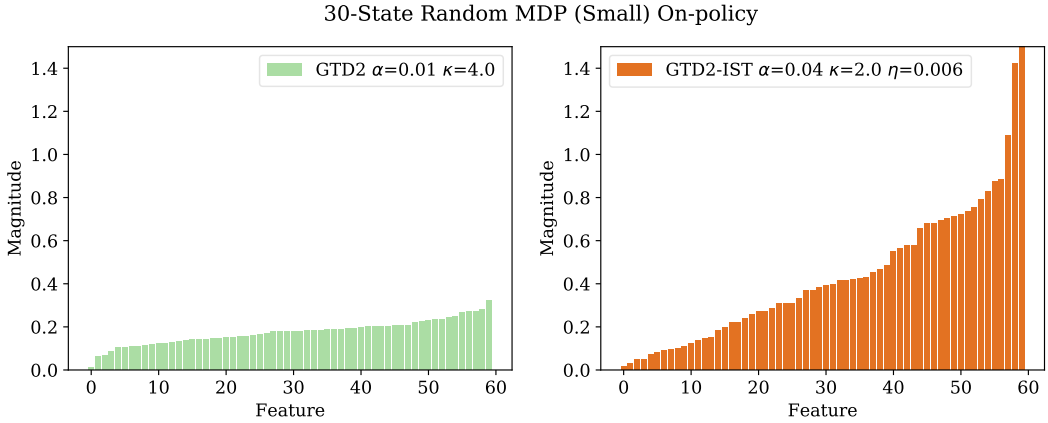

**(b)** TDC

**Figure 3.8.:** Sparsity of weight vector after convergence 10 noise features (absolute values)

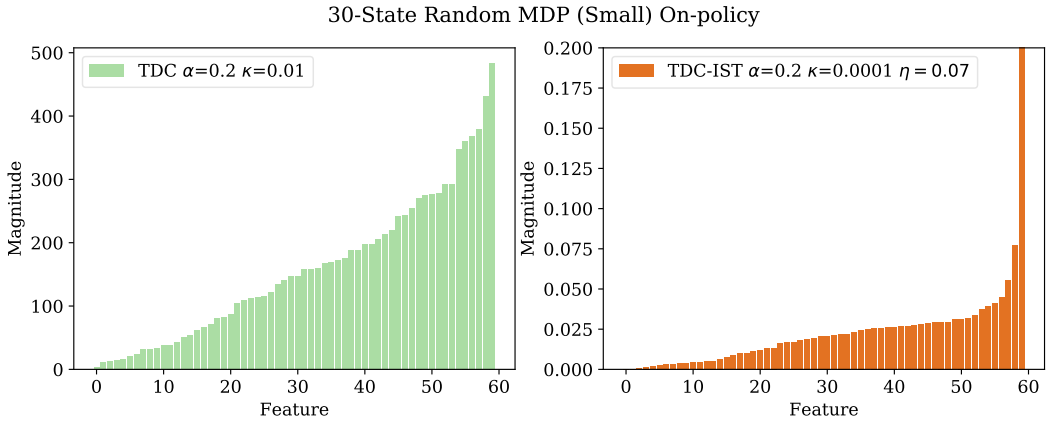
parameters, stepsize $\alpha$ and derivation factor for the secondary stepsize $\kappa$, the criterion on how to select those is not clear and there is also no general rule on how to select them given by the original authors. It seems therefore useful to study the behavior of the unmodified and modified algorithms for different sets of parameters.

The first such experiment is done to investigate the error after 2000 samples, where the algorithms are considered to be converged. This is done by selecting different $\alpha$'s and $\kappa$'s while keeping the $\eta$ parameter fixed to the optimal value found by gridsearch in order to be able to compare all algorithm versions.

In Figure 3.10 the results can be studied. Here in the left column, the original and in




**(a)** GTD2

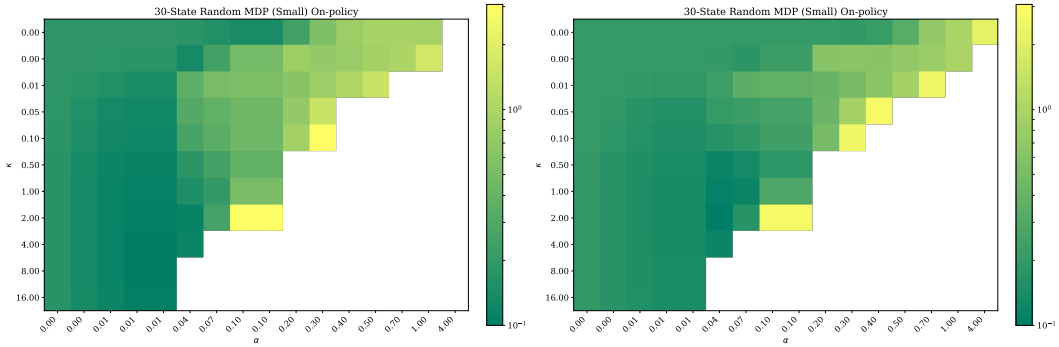



**(b)** TDC

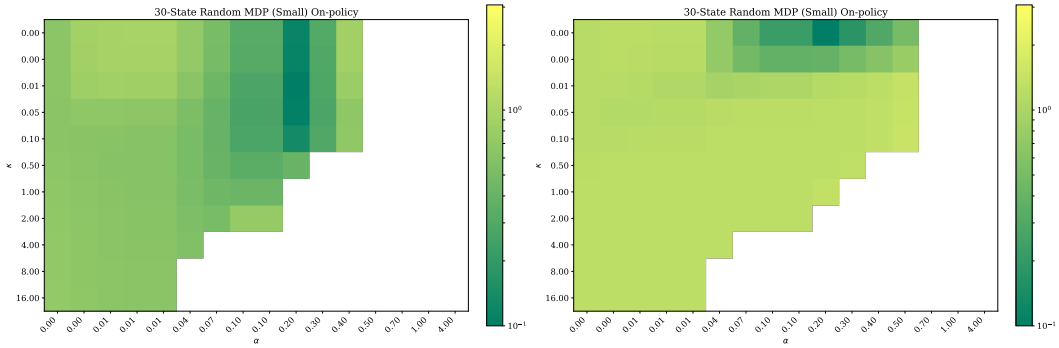
**Figure 3.9.:** Sparsity of weight vector after convergence 40 noise features (absolute values)

the right column the modified versions of the algorithms are plotted. Each plot has the different parameter values for $\kappa$ on the y-axis and $\alpha$ on the x-axis. The color of the fields depict the RMSPBE value after 2000 samples at the end of the experiment averaged over 20 independent runs, just like done in the gridsearch experiment. A darker green value here depicts a lower final error, while a lighter yellow color means a bigger final error. If a field is white, this means that the algorithm has not converged, i.e. the final error was above some threshold.

In general when comparing the graphs on each row, it can be observed that the algorithm




**(a)** GTD2 **(b)** GTD2-IST



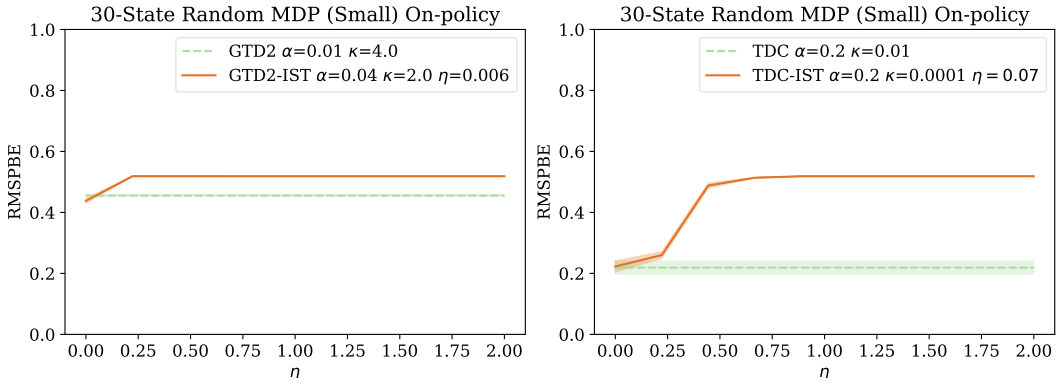

**(c)** TDC **(d)** TDC-IST

**Figure 3.10.:** Parameter sensitivity on the noisy small random MDP with respect to $\alpha$ and $\kappa$ while fixing the optimal $\eta$ for the environment.

variants do converge in roughly the same regions for both parameters. The unmodified versions seem to have a slightly greater region of parameters to select from where almost equally optimal final error values are obtained (observe the dark green regions). The thresholded algorithm variants seem to converge on a narrower region of parameter range. This can be seen especially for TDC-IST which seems to have only one square of parameter combination, where the darkest green is obtained. However, the best error values within those narrow regions are lower than the original algorithm versions, as already seen above in the investigation of the convergence speed and final error attained.

Another important parameter to check for sensitivity is the $\eta$ regularization weight. As it controls how much weight is given to the $\ell_1$ regularization part of the cost function, it is expected in domains with a high amount of noise to be necessary to be set higher than

in domains with a low amount. For the small random MDP with ten original features and ten noisy features, this however does not seem the be the case. Even if the amount of noisy features is increased to 20, the curves stay more or less in the same form as in Figure 3.11. In Figure 3.11a, it can be seen that starting from a value of $\eta = 0.25$, an increase in value for this parameter does not influence the final attained error much. This is also reflected in the optimal value found by gridsearch of $\eta = 0.006$ for this environment. A similar behavior is observed for the TDC algorithm, that also manages to perform optimally within the parameter search range at $\eta = 0.07$.




**(a)** GTD2 **(b)** TDC

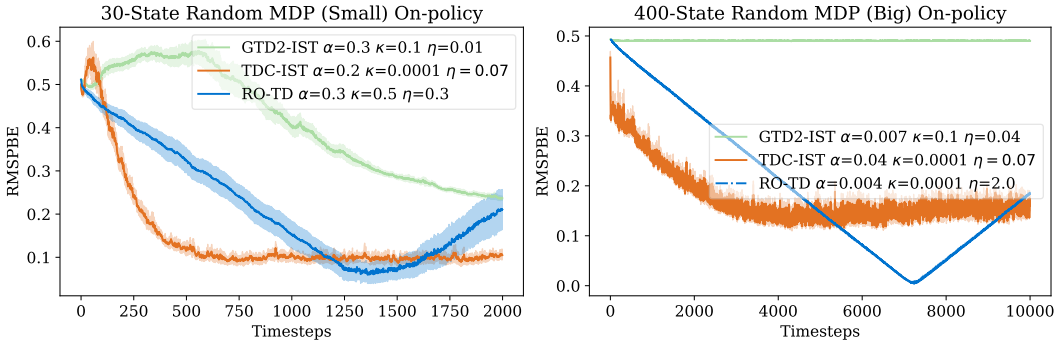
**Figure 3.11.:** Parameter sensitivity regarding $\eta$ on the noisy small random MDP.

In general this allows to conclude that some small value of $\eta$ is appropriate for most applications that have some noise. Too high values for this parameter can degrade the performance a bit, but tend to plateau off.

### 3.7.6. Comparison to RO-TD

If we draw the attention on Figure 3.12, RO-TD performance is degraded in the random MDP environment and within the searched parameter range, no set of parameters was found, where the convergence was satisfactory. This seems to indicate, that with increased difficulty of approximation and noise, RO-TD still seems to degrade although it was initially designed to be more robust to noise because it incorporates the $\ell_1$-sparsity. Another reason for the error increasing after a certain amount of iterations is also the use of a fixed step size throughout the whole experiment. This can be seen from the error measure going

down up to a certain point and the increasing again. Especially the modified TDC variant of the algorithm seems to perform well in this type of comparison. The error goes down significantly faster than for the other two algorithms and then stays on its minimum value.

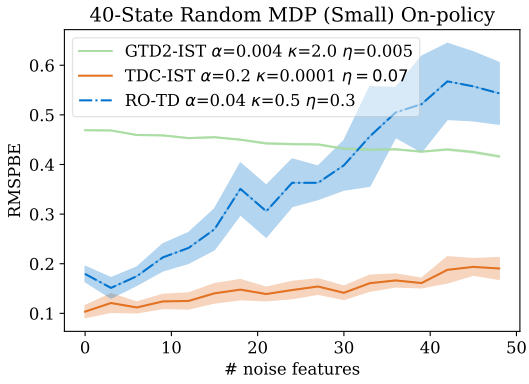



**(a)** Small MDP **(b)** Big MDP

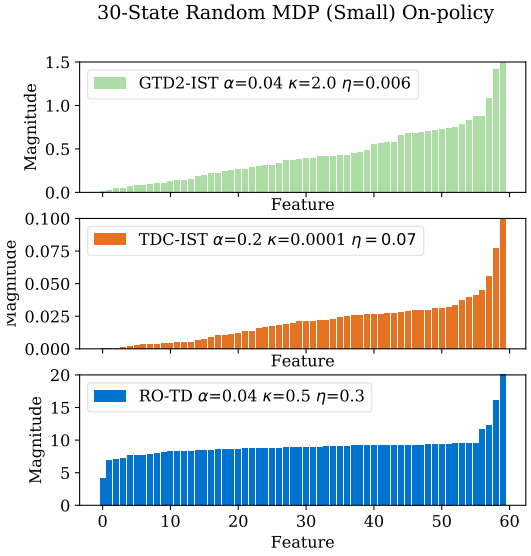
**Figure 3.12.:** Comparison of soft-thresholded GTD variants with RO-TD

When studying the robustness to an increasing amount of noisy features added to the experiment as in Figure 3.13a, the observation is that for TDC the behavior is exactly as expected: for an increasing amount of noise the final error at the end of the experiment tends to increase as well as the variance of the results. For GTD2 the error even seems to decrease and for RO-TD from a certain amount of noise features onwards the performance of the algorithm degrades significantly reflected in an increased average final error and greatly increased variance among different algorithm runs.

A likely explanation for this degradation can be seen in Figure 3.13b, where the magnitude of the individual components of the solution vector $\theta$ are plotted, sorted by increasing magnitude. As can be nicely observed, GTD2-IST and TDC-IST manage to put weight on meaningful features and therefore selecting those, while features only contributing noise are weighted with zero. RO-TD distributes the weight evenly across all the features and therefore also is susceptible to this noise. This explains the degradation as more and more noisy features are added.




**(a)** Noise sensitivity of RO-TD for a varying amount of noise features




**(b)** Sparsity of solution weights with 40 noisy features present

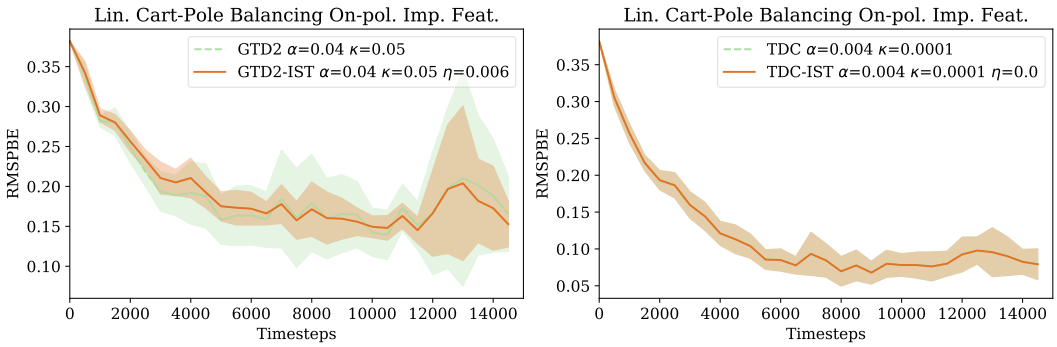
**Figure 3.13.:** Noise sensitivity and sparsity of RO-TD

### 3.7.7. Continuous Domain Performance

Up to now all experimental domains that were tested were discrete. This means that the simulation the algorithm runs against has a finite number of states and all those problems could in theory also be solved by simple dynamic programming approaches. In continuous domains, as the linearized cart pole balance task, this is no longer possible. The algorithm has to go through a feature representation in order to make the problem tractable. Also some sort of value function approximation has to be used.

The first set of experiments is conducted on the cart pole environment without any noise in order to ensure that convergence works in principle. The approximate (or imperfect) feature representation was used in this case, because of the lower dimensionality the gridsearch for optimal parameters and the experiment itself was much less computational intensive. The results are shown in Figure 3.14. It can be seen that both algorithms converge in this environment. Also, if we take a closer look at the parameters chosen, it can be seen that the optimal parameters determined by gridsearch are exactly the same for the original and

the modified versions of the algorithms. Additionaly, the soft thresholding parameter $\eta$ was either chosen small (GTD2) or to be zero (TDC). This makes sense, as this version of the environment is supposed to not contain any extra noise apart from the usual sampling process. For GTD2, it nevertheless was helpful to introduce a little bit of regularization even in the non-noisy case. This improved the variance as can be seen in Figure 3.14a, where the variance across the 20 independent runs is reduced (observe the smaller orange to green area).



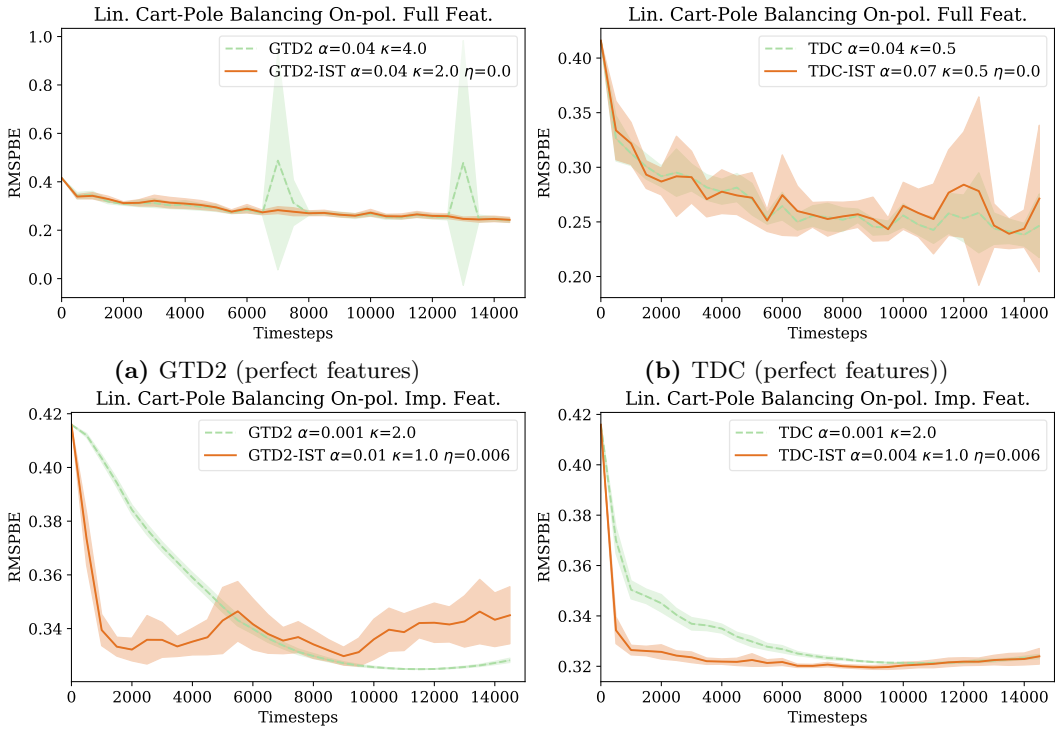

**(a)** GTD2 **(b)** TDC

**Figure 3.14.:** Comparison of performance in the continuous domain (no extra noise).

The advantage of introducing regularization in the continuous domain experiments can be seen when noise is introduced. Figure 3.15 shows the results for GTD2 and TDC for both the perfect and approximate feature setting. While in the perfect feature setting the advantage is not visible apart for the reduction of sampling artifacts in the case for the GTD2 algorithm in Figure 3.15a, the advantage of regularization can be observed in the second row of Figure 3.15. When using approximate features, the dimension is reduced by half and therefore less samples are necessary to achieve good improvement in performance. This shows that the thresholded algorithms converge much faster at around 1000 samples, whereas the original algorithms need up to 10000 samples to achieve similar results.

## 3.8. Summary

In this chapter a regularized variant of stochastic gradient temporal difference reinforcement learning was introduced and compared to other existing works in the literature. Special




**(a)** GTD2 (perfect features)

**(b)** TDC (perfect features))

**(c)** GTD2 (approximate features)

**(d)** TDC (approximate features)

**Figure 3.15.:** Comparison of performance in the continuous domain (with extra noise).

care was given to the detailed understanding and comparison to the RO-TD algorithm the most similar formulation of regularization for stochastic gradient reinforcement learning algorithms.

The GTD2 and TDC variants of the algorithms show a good convergence behavior on the benchmark settings without noise and performance depending on the size of domain is as expected. GTD, however, shows unstable convergence and is therefore left out of all further investigations. As laid down in the motivation, the regularized algorithm variants are robust to noise which can be clearly seen when the number of noisy influences is increased over the course of an experiment. The modified algorithms still converge in terms of error measure even when as many as five times noisy features as information bearing features are present. Also the thresholding manages to keep the value function weights small and even drive some towards zero.

A very sharp feature selection cannot be observed, but some attenuation of features, that only contribute noise. This, however, comes at some cost of increased parameter sensitivity for $\beta$ and $\alpha$ and could increase the difficulty when tuning the algorithms for a specific application. The regularization parameter $\eta$ in contrast seems to be uncritical when selecting some sensible values close to 0.

A dedicated comparison to RO-TD shows, comparable algorithm performances in terms of convergence speed as well as final error reached with a slight superiority, when the number of noisy features is extremely increased. Continuous domain convergence speed can be increased and the final error reached can be lowered when introducing regularization, especially for certain types of features.

# Chapter 4.

# Accelerated Algorithms

In this chapter, the second main contribution of this work will be covered, i.e. the acceleration of the stochastic gradient descent learning algorithms. For this, the method of Nesterov accelerated gradient descent is applied to the gradient based learning methods. In order to be able to do this, some preliminary properties of the cost function, the MSPBE have to be investigated and proven to hold in our setting. After that two variants of the accelerated algorithms, one standard and one regularized version will be derived. Most of the results in this chapter have been previously published in [54].

## 4.1. Motivation

The stochastic gradient descent algorithms, introduced in the previous chapters have the guaranteed property to converge to the global optimum. Also another beneficial property is that they are applicable in general settings, where for example not all samples can be held in memory or where only computation is available that is linear in terms of feature size. Inevitably, they inherit weaknesses of stochastic gradient descent algorithms. One such foremost is possible slow convergence speed. Although stochastic gradient algorithms can be as fast as batch gradient algorithms, the convergence can be significantly slowed down due to unfavorable sampling and severely be impacted by noise. As Bottou et al. [11, 10] indicate, this slowdown of convergence is especially prevalent towards the getting closer to the optimal point. Although the convergence speed can be in terms of $O(\frac{1}{k})$, with $k$ being the iteration steps, this can only be guaranteed if a prescaling on the gradient components is employed [11]. This prescaling requires to approximate the Hessian of the cost function, which in itself can be impossible for large feature dimensions.

Historically, one approach to tackle slow convergence rates in stochastic gradient methods is the seminal work done by Nesterov [60]. The so called *accelerated gradient descent* method allows for convergence rates of order $O(\frac{1}{k^2})$ guaranteed for (strongly) convex functions.

## 4.2. Analysis of the Projected Bellman Error

As already stated in the introduction of this chapter, before deriving the actual algorithm, some preliminary notations and results describing the properties of the MSPBE function need to be introduced. In the next subsection, four definitions of convexity and Lipschitz continuity are introduced to be used in the section thereafter in proving the Lipschitz continuity of the MSPBE functions' gradient.

### 4.2.1. Preliminaries

**Definition 3** (Convexity)**.** *In general, convexity refers to sets and functions. So for any two points $x \in \mathcal{X}$ and $y \in \mathcal{X}$ in a set $\mathcal{X} \in \mathbb{R}^l$, we have that*

$$\forall \alpha \in [0,1], \quad \alpha x + (1-\alpha)y \in \mathcal{X}. \tag{4.1}$$

*Intuitively, this means, that for a linear combination of those two points out of the set, the combination also lies in the set [86, 9]. Similar for a convex function $f : \mathcal{X} \to \mathcal{Y}$, we have*

$$\forall \alpha \in [0,1], \quad f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y). \tag{4.2}$$

*An example for a convex function is the quadratic function $f(x) = x^\top H x$, where $H$ has to be a symmetric positive semidefinite matrix.*

**Definition 4** ($\mu$-strong Convexity)**.** *For a function $f(x) : \mathcal{X} \to \mathcal{Y}$ and any $x \in \mathcal{X}$ and $y \in \mathcal{X}$, as well as $g(x) \in \partial f(x)$ being the subgradient, then $f$ is $\mu$-strongly convex if*

$$f(y) \geq f(x) + (g(x))^\top (y - x) + \frac{\mu}{2}\|y - x\|^2 \tag{4.3}$$

*for any norm $\|\cdot\|^2$ [62].*

*If the function is twice differentiable, then an alternative formulation for strong convexity*

*for $f$ is given if*

$$\nabla_x^2 f(x) = \mathcal{H} \succeq \mu I, \tag{4.4}$$

*where $\mathcal{H}$ is the Hessian, $I$ is the identity matrix and $\succeq$ denotes positive semi-definiteness. In other words, $\mathcal{H} - \mu I$ has to be positive semi-definite or equivalently, the minimum eigenvalue of $\mathcal{H}$ has to be greater or equal to $\mu$.*

**Definition 5** (Lipschitz Continuity)**.** *A function $f(x) : \mathcal{X} \to \mathcal{Y}$ is L-Lipschitz continuous if for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ there exists a real constant $L > 0$ and it holds that*

$$\|f(x) - f(y)\| \leq L\|x - y\|. \tag{4.5}$$

An equivalent formulation of L-Lipschitz continuity can be obtained by looking at the gradient. A function is L-Lipschitz continuous, if and only if the first derivative is bounded, i.e. $L = \sup |\nabla_x f(x)|$. Additionally, the gradient of a function can also be characterized in terms of Lipschitz continuity.

**Definition 6** (Lipschitz Gradient)**.** *For any $x \in \mathcal{X}$ and $y \in \mathcal{X}$ and for the gradient $\nabla_x f(x)$ of a differentiable function $f(x) : \mathcal{X} \to \mathcal{Y}$, the gradient is L-Lipschitz continuous if and only if*

$$f(y) \leq f(x) + (\nabla_x f(x))^\top (y - x) + \frac{L}{2}\|x - y\|^2, \tag{4.6}$$

*cf. [62].*

### 4.2.2. Properties of the Projected Bellman Error Function

In order to derive an accelerated Nesterov method for gradient temporal difference learning, we have to ensure that the cost function $J(\theta)$ we would like to optimize is convex. Additionally, we have to derive the Lipschitz constant $L$ enabling us to apply the accelerated gradient learning scheme from [34]. As it was empirically shown (cf. [17]), that the GTD2 and TDC variants of the gradient TD learning methods enjoy better convergence speeds than GTD, we will focus on accelerating those two methods in the following.

Recall, that we have the cost function for the MSPBE as

$$J_{\text{TDC}}(\theta) = \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2, \tag{4.7}$$

which can be rewritten in terms of the non-weighted norm[1] as in

$$\begin{aligned} J_{\mathrm{TDC}}(\theta) &= \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2 \\ &= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top \Pi_\mu^\top \Xi \Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\ &= \|\hat{\Pi}_\mu\sqrt{\Xi}(\Phi\theta - \mathcal{T}_\mu(\Phi\theta))\|_2^2. \end{aligned} \tag{4.8}$$

The projection for the re-weighted MSPBE is then

$$\hat{\Pi}_\mu = \sqrt{\Xi}\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\sqrt{\Xi}, \tag{4.9}$$

which is an orthogonal projector on the column span of $\sqrt{\Xi}\Phi$.

From these expressions, we can compute the Hessian $\mathcal{H}_J(\theta)$ of $J_{\mathrm{TDC}}(\theta)$ directly starting with the derivative as

$$\frac{1}{2}\nabla_\theta J_{\mathrm{TDC}}(\theta) = \left(\Phi^\top\Xi\Phi - \gamma\Phi^\top P_\mu^\top\Xi\Phi\right)\left(\Phi^\top\Xi\Phi\right)^{-1}\left(\Phi^\top\Xi\Phi\theta - \Phi^\top\Xi R_\mu - \gamma\Phi^\top\Xi P_\mu\Phi\theta\right) \tag{4.10}$$

and continuing to derive the Hessian as

$$\begin{aligned} \frac{1}{2}\nabla_\theta^2 J_{\mathrm{TDC}}(\theta) &= \left(\Phi^\top\Xi\Phi - \gamma\Phi^\top P_\mu^\top\Xi\Phi\right)\left(\Phi^\top\Xi\Phi\right)^{-1}\left(\Phi^\top\Xi\Phi - \gamma\Phi^\top\Xi P_\mu\Phi\right) \\ &= \Phi^\top\left(I - \gamma P_\mu\right)^\top\sqrt{\Xi}\hat{\Pi}_\mu\sqrt{\Xi}\left(I - \gamma P_\mu\right)\Phi \\ &= \Phi^\top\left(I - \gamma P_\mu\right)^\top\Pi_\mu^\top\Xi\Pi_\mu\left(I - \gamma P_\mu\right)\Phi \\ &= A^\top A, \end{aligned} \tag{4.11}$$

with $A := \sqrt{\Xi}\Pi_\mu\left(I - \gamma P_\mu\right)\Phi$.

**Lemma 1** (Strong Convexity of MSPBE)**.** *Assume that the feature matrix $\Phi \in \mathbb{R}^{n\times l}$ is full rank and the Markov chain defined by $P \in \mathbb{R}^{n\times n}$ is irreducible and aperiodic. Furthermore, there exists a unique limiting distribution $\xi \in \mathbb{R}^n$, which satisfies $P\xi = \xi$ with $\xi_i > 0, \forall i \in 1\ldots n$, then the MSPBE, defined in Equation* (4.8) *is strongly convex.*

*Proof.* As stated in Definition 4, for a function to be strongly convex, the Hessian of this function has to be positive semi-definite, which in terms means that we have to check, that

$$\nabla_\theta^2 J_{\mathrm{TDC}}(\theta) = \mathcal{H}_J(\theta) \succeq \mu I \tag{4.12}$$

[1]The full derivation can be found in Appendix A.3

holds. Also for a matrix to be positive definite equates to showing that for any $x$ it holds that

$$x^\top \mathcal{H}_J(\theta) x \geq 0. \tag{4.13}$$

We can therefore further decompose the Hessian as

$$x^\top \mathcal{H}_J(\theta) x = x^\top A^\top A x = \|Ax\|_2^2 \geq 0, \tag{4.14}$$

as per Equation (4.10) and the definition of the $\ell_2$ norm. It holds, that the norm is always non-negative and therefore, we can conclude that the Hessian of the MSPBE is positive definite and the MSPBE itself therefore strongly convex.

Further, since the transition matrix $P$ defines an aperiodic and irreducible Markov chain, it follows that the matrix $(I - \gamma P)$ is full rank, cf. [84, 83]. The computation of the Hessian is a product of full rank matrices, since the assumption that $\Phi$ is full rank and the result is in the dimension of the features $\mathbb{R}^l$, which is the smallest dimension involved. It holds therefore, that the Hessian itself is full rank and has no eigenvalue, that is zero. ■

**Lemma 2.** *For a transition probability matrix of a Markov chain, defined by $P \in \mathbb{R}^{n \times n}$ with limiting unique distribution $\xi$, which satisfies $P\xi = \xi$ with $\xi_i > 0, \forall i \in 1 \dots n$ and any vector $x \in \mathbb{R}^n$, we have*

$$\|Px\|_\xi \leq \|x\|_\xi. \tag{4.15}$$

*Proof.* We can closely follow the proof in [5, 7, 83] and write

$$\begin{aligned} \|Px\|_\xi^2 &= \sum_{i=1}^n \xi_i \left( \sum_{j=1}^n p_{ij} x_j \right)^2 \\ &\leq \sum_{i=1}^n \xi_i \sum_{j=1}^n p_{ij} x_j^2 = \sum_{j=1}^n \sum_{i=1}^n \xi_i p_{ij} x_j^2 = \sum_{j=1}^n \xi_j x_j^2 = \|x\|_\xi^2. \end{aligned} \tag{4.16}$$

The first inequality follows from the convexity of the quadratic function, coupled with Jensen's inequality, and the last step is a result of the definition of the limiting distribution as $P\xi = \xi$, or component-wise $\sum_{i=1}^n \xi_i p_{ij} = \xi_j$. ■

**Lemma 3** ($\mu$-strong Convexity of MSPBE)**.** *Let the feature matrix $\Phi \in \mathbb{R}^{n \times l}$ be full rank and the Markov chain defined by $P \in \mathbb{R}^{n \times n}$ be irreducible and aperiodic. Furthermore, there*

*exists a unique limiting distribution* $\xi \in \mathbb{R}^n$*, which satisfies* $P\xi = \xi$ *with* $\xi_i > 0, \forall i \in 1 \dots n$*. Define* $C = \Phi^\top \Xi \Phi$*, where* $\Xi = \operatorname{diag}(\xi)$ *is the matrix with the elements* $\xi_i$ *on the diagonal and let* $\lambda_{\min}$ *be the smallest eigenvalue of* $C$*. Then the MSPBE, defined in Equation* (4.8) *is* $\mu$*-strong convex with*

$$\mu \geq (1-\gamma)^2 \lambda_{\min} \tag{4.17}$$

*Proof.* We have

$$\begin{aligned} x^\top \mathcal{H}_J(\theta) x &= x^\top \Phi^\top (I - \gamma P_\mu)^\top \Pi_\mu^\top \Xi \Pi_\mu (I - \gamma P_\mu) \Phi x \\ &= x^\top A^\top A x \\ &= \|\sqrt{\Xi} \Pi_\mu (I - \gamma P_\mu) \Phi x\|_2^2 \end{aligned}$$

and by using the change of norm, we come to

$$= \|\Pi_\mu (I - \gamma P_\mu) \Phi x\|_\xi^2.$$

Then by using the definition of $\Pi_\mu$, we can go further as in

$$\begin{aligned} &= \|\Phi(\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi (I - \gamma P_\mu) \Phi x\|_\xi^2 \\ &= \|\Phi(\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi \Phi x - \gamma \Phi (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi P_\mu \Phi x\|_\xi^2 \\ &= \|\Phi x - \gamma \Pi_\mu P_\mu \Phi x\|_\xi^2, \end{aligned}$$

and use the reverse triangle equality to obtain

$$\geq \left(\|\Phi x\|_\xi - \gamma \|\Pi_\mu P_\mu \Phi x\|_\xi\right)^2.$$

We now have $\|\Pi_\mu x\|_\xi \leq \|x\|_\xi$ because $\Pi_\mu$ is an orthogonal projector and continue in

$$\geq \left(\|\Phi x\|_\xi - \gamma \|P_\mu \Phi x\|_\xi\right)^2,$$

with $\|P_\mu x\|_\xi \leq \|x\|_\xi$ due to Lemma 2 to conclude that

$$\begin{aligned} &\geq \left(\|\Phi x\|_\xi - \gamma \|\Phi x\|_\xi\right)^2 \\ &= (1-\gamma)^2 \|\Phi x\|_\xi^2. \end{aligned} \tag{4.18}$$

Additionally, we know, that the matrix $C = \Phi^\top \Xi \Phi$ is symmetric positive definite and therefore there exists an orthonormal basis $(v^{(j)})_{j \in 1,\dots,n}$ of $\mathbb{R}^n$ composed of eigenvectors of $C$, with respective eigenvalues $\lambda_j$, such that we can represent any vector $x$ in terms of this new basis as in

$$x = \sum_{j=1}^{n} x_j v^{(j)}, \tag{4.19}$$

and in the following derive

$$\begin{aligned}\|\Phi x\|_\xi &= x^\top \Phi^\top \Xi \Phi x \\ &= \left(\sum_{j=1}^{n} x_j v^{(j)}\right) \Phi^\top \Xi \Phi \left(\sum_{j=1}^{n} x_j v^{(j)}\right).\end{aligned}$$

Then, because $C$ has an orthogonal eigenvector basis we can simplify

$$\begin{aligned}&= \left(\sum_{j=1}^{n} x_j v^{(j)}\right)^\top \left(\sum_{j=1}^{n} \lambda_j x_j v^{(j)}\right) \\ &= \sum_{j=1}^{n} \lambda_j x_j^2 \geq \lambda_{\min} \sum_{j=1}^{n} x_j \\ &= \lambda_{\min} \|x\|_2^2.\end{aligned} \tag{4.20}$$

With this result and the result from Equation (4.18), we can therefore conclude, that $\mathcal{H}_J(\theta) \succeq \mu I$ with

$$\mu \geq (1-\gamma)^2 \lambda_{\min} \tag{4.21}$$

and the MSPBE is $\mu$-strong convex. ■

**Remark 1.** *Although the MSBPE $J_{TDC}(\theta)$ is $\mu$-strongly convex, the smallest eigenvalue of the Hessian $\mathcal{H}_J(\theta)$ is unbounded from below for an arbitrary feature matrix $\Phi$.*

*This means, that for an unfortunate selection or construction of the feature matrix, the convergence results that depend on the $\mu$-strong convexity, can be arbitrarily bad. By chance or adversarial construction the convergence of the accelerated algorithms can therefore be hindered in an arbitrary way. For most of the times, the feature matrix should be constructed in a sensible way and should be well behaved which will result in nice convergence properties.*

**Lemma 4** (L-Lipschitz MSPBE Gradient)**.** *Let the feature matrix $\Phi \in \mathbb{R}^{n \times l}$ be full rank and the Markov chain defined by $P \in \mathbb{R}^{n \times n}$ be irreducible and aperiodic. Furthermore, there exists a unique limiting distribution $\xi \in \mathbb{R}^n$, which satisfies $P\xi = \xi$ with $\xi_i > 0, \forall i \in 1 \dots n$. Define $C = \Phi^\top \Xi \Phi$, where $\Xi = \operatorname{diag}(\xi)$ is the matrix with the elements $\xi_i$ on the diagonal and let $\lambda_{\max}$ be the maximal eigenvalue of $C$. Then the gradient of the MSPBE, defined in*

*Equation* (4.8) *is L-Lipschitz with*

$$L = (1+\gamma)^2 \max_j \|\phi_j\|_2^2, \tag{4.22}$$

*or equivalently*

$$L = (1+\gamma)^2 \lambda_{\max}. \tag{4.23}$$

*Proof.* The MSBPE function $J_{\mathrm{TDC}}(\theta)$ is L-Lipschitz if the inequality

$$x^\top \mathcal{H}_J(\theta) x \leq L\|x\|_2^2 \tag{4.24}$$

holds for all $x \in \mathbb{R}^n$ since $J_{\mathrm{TDC}}(\theta)$ is strongly convex. We additionally do have just as in the first four lines of Equation (4.18)

$$\begin{aligned}
x^\top \mathcal{H}_J(\theta) x &= \|\Pi_\mu (I - \gamma P_\mu)\Phi x\|_\xi^2 \\
&\text{and because } \Pi_\mu \text{ is an orthogonal projector it holds, that} \\
&\leq \|(I - \gamma P_\mu)\Phi x\|_\xi^2 \\
&= \|\Phi x - \gamma P_\mu \Phi x\|_\xi^2 \\
&\leq \|\Phi x + \gamma P_\mu \Phi x\|_\xi^2. \\
&\text{By using the triangle inequality we get} \\
&\leq \left(\|\Phi x\|_\xi + \gamma \|P_\mu \Phi x\|_\xi\right)^2 \\
&\text{and due to Lemma 2 we can continue as} \\
&\leq \left(\|\Phi x\|_\xi + \gamma \|\Phi x\|_\xi\right)^2 \\
&= (1+\gamma)^2 \|\Phi x\|_\xi^2.
\end{aligned} \tag{4.25}$$

Now since we have $x^\top \mathcal{H}_J(\theta) x \leq (1+\gamma)^2 \|\Phi x\|_\xi^2$, let us further look at $\|\Phi x\|_\xi^2$ for which we

have

$$\begin{aligned} \|\Phi x\|_\xi^2 &= \sum_{j=1}^{n} \xi_j (\phi_j^\top x)^2 \\ \text{with the Cauchy-Schwarz inequality further as} \\ &\leq \sum_{j=1}^{n} \xi_j \|\phi_j\|_2^2 \|x\|_2^2 \leq \sum_{j=1}^{n} \xi_j (\max_i \|\phi_i\|_2^2) \|x\|_2^2 \\ &= (\max_i \|\phi_i\|_2^2) \|x\|_2^2. \end{aligned} \tag{4.26}$$

Overall, we can therefore conclude that

$$L \leq (1+\gamma)^2 \max_j \|\phi_j\|_2^2. \tag{4.27}$$

Alternatively, we have the matrix $C = \Phi^\top \Xi \Phi$ as a symmetric positive definite matrix and therefore there exists an orthonormal basis $(v^{(j)})_{j \in 1,\dots,n}$ of $\mathbb{R}^n$ of eigenvectors of $C$ with respective eigenvalues $\lambda_i$ and $\lambda_{\max}$ the biggest eigenvalue. We can now express $x$ as $x = \sum_{j=1}^n x_j v^{(j)}$ and rewrite

$$\begin{aligned} \|\Phi x\|_\xi &= x_j^\top \Phi^\top \Xi \Phi x \\ &= \left( \sum_{j=1}^{n} x_j v^{(j)} \right)^\top \Phi^\top \Xi \Phi \left( \sum_{j=1}^{n} x_j v^{(j)} \right) \\ &= \left( \sum_{j=1}^{n} x_j v^{(j)} \right)^\top \left( \sum_{j=1}^{n} \lambda_j x_j v^{(j)} \right) \\ &= \sum_{j=1}^{n} \lambda_j x_j^2 \leq \lambda_{\max} \sum_{j=1}^{n} x_j^2 \\ &= \lambda_{\max} \|x\|_2^2 \end{aligned} \tag{4.28}$$

and conclude analogous to the above that

$$L \leq (1+\gamma)^2 \lambda_{\max}. \tag{4.29}$$

■

## 4.3. Algorithm Derivation

If we have the problem of optimizing a smooth convex cost function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient descent method, described in Section 2.5 is the tool of choice. It guarantees convergence to the global optimum for convex functions and can be adapted to be applicable in a stochastic sampling online setting. Although these benefits exist, the standard gradient descent methods have only a convergence rate of $O(\frac{1}{k})$. This means the error reduces proportional to $\frac{1}{k}$, where $k$ are the algorithm iteration steps.

In the pioneering work of Nesterov [60] a method of gradient descent is derived, which works for convex function with L-Lipschitz first order derivatives. This so-called Nesterov's accelerated gradient descent attains a convergence rate of $O(\frac{1}{k^2})$. Hu et al. [34] adapted Nesterov's work to a stochastic online learning setting, called *Stochastic Accelerated GradieEnt* (SAGE). Here, they find the minimum of the expectation of a smooth convex function $f_\theta(x)$, parametrized by $\theta$ as

$$\min_\theta \mathbb{E}_\mathcal{X} \left[ f_\theta(x) \right], \tag{4.30}$$

where $\mathcal{X}$ is the distribution of input samples. Correspondingly, this problem can be approximated in terms of finite samples as the sample average, cf. [34], as

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} f_\theta(x_i), \tag{4.31}$$

for a sample set of size $n$. An accelerated stochastic gradient descent algorithm solving this problem is described in Algorithm 4.1. The illustration, how the acceleration works is illustrated in Figure 4.1 and can be described as follows. In its core, the method does the same as the ordinary gradient descent, but at the same time maintains a second set of weights, that get updated in an alternating fashion $x_k$ and $y_k$. The first step in every iteration is to do a step in the gradient direction, but then linearly extrapolate with the set of secondary weights from the previous iterations. It therefore *overshoots* the gradient step in a direction governed by the previous estimates. The larger the parameter controlling this behavior is chosen, the more past gradient updates are taken into consideration.

As pointed out in Remark 1, the smallest eigenvalue of the Hessian of the MSPBE is not bounded from below, but for a given feature matrix $\Phi$ the fact, that there exists a fixed strong convexity constant, guarantees convergence as stated in [34]. In order to practically

**Algorithm 4.1** Nesterov's Accelerated Stochastic Gradient Descent Algorithm.

**Input:** The Lipschitz constant $L$ of the function $f_\theta(x)$.

**Initialize:** Arbitrary initial guess $x_0$, initial $y_0 = x_0, \lambda_0 = 0$ and $k = 0$.

**repeat**

$\lambda_{k+1} = \frac{1+\sqrt{1+4\lambda_k^2}}{2}$

$\gamma_k = \frac{1-\lambda_k}{\lambda_{k+1}}$

$y_{k+1} = x_k + \frac{1}{L}\nabla_\theta f_\theta(x_k)$

$x_{k+1} = (1-\gamma_k)y_{k+1} + \gamma_k y_k$

**until** converged

**Output:** $y_{k+1}$.

implement an accelerated algorithm, the concrete eigenvalue calculation of $\Phi$ is not possible and therefore omitted and an algorithm is derived, which does not depend on the assumption of strong convexity, similar to [34]. The resulting algorithm for the TDC gradient update can be found in Algorithm 4.2, the algorithm for the similarly derived GTD2 update can be found in Algorithm 4.3.

It can be observed, that the stochastic gradient update of the GTD2 and TDC algorithms are incorporated in the stochastic version of the SAGE, however an intricate reshuffling of terms is necessary. First the sequence for parameter $L_k$ is updated and the current estimate for the weight vector $\theta_k$ is calculated. This step is necessary in order to be able to get the gradient estimate $g_k$ in the next step. As the last two steps, the two auxiliary parameter sequences $y_k$ and $z_k$ can be calculated according to the original algorithm formulation.

A remaining step for completeness is to mention the initialization of the additional parameters. The gradient steps size sequences $\alpha_k$ and $\beta_k$, for the auxiliary sequence $w_k$, are chosen as in the original algorithm formulation, cf. [79]. The initial $L_0 > 0$ has to be chosen to be a value greater than the Lipschitz constant $L$, bounded in Equation (4.22) and equation (4.23). The parameter $\nu \in [0, 1]$ can be interpreted as a momentum parameter. A value close to 0 gives more weight to recent updates. Specifically, when $\nu$ is set to 0, the accelerated method resembles the standard gradient descent and a higher value will give more importance to past sample update directions.

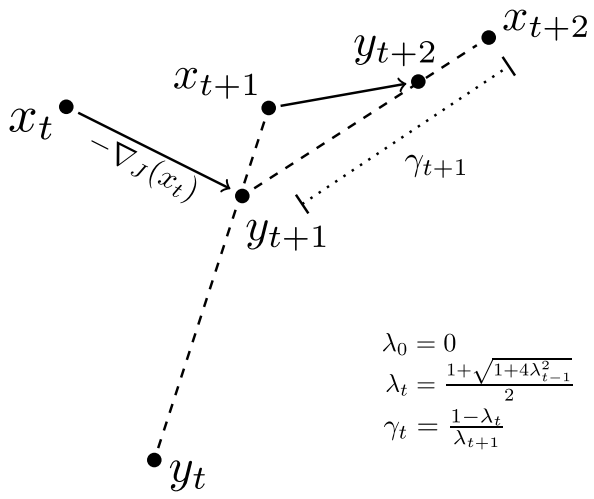



**Figure 4.1.:** Illustration of the Nesterov accelerated gradient method. Adapted from S. Bubeck [14].

## 4.4. Related Work

Pan et al. [66] derived a different approach to accelerating temporal difference learning methods. In their work, they first derive a second order gradient descent method for minimizing the MSPBE. Their gradient steps taken in every iteration of the algorithm are

$$\begin{aligned}\theta_{k+1} &= \theta_k - \frac{\alpha_k}{2}\mathcal{H}_J(\theta_k)^{-1}\nabla_\theta J_{\mathrm{TDC}}(\theta_k) \\ &= \theta_k + \alpha_k B^{-1} c,\end{aligned} \tag{4.32}$$

with $\alpha_k$ being an appropriate step size and $B = \Phi^\top \Xi \Phi - \gamma \Phi^\top P_\mu^\top \Xi \Phi$ and $C = (\Phi^\top \Xi \Phi \theta_k - \Phi^\top \Xi R_\mu - \gamma \Phi^\top \Xi P_\mu \Phi \theta)$. For a stochastic gradient descent algorithm the quantity $c$ can be approximated with samples of the TD-error (as seen in the chapters above) as it holds true that $c = \mathbb{E}_{\mu,\xi}[\delta(\theta_k, \theta_k)]$. The matrix $B$ could be stochastically approximated as the algorithm goes along sampling, but still requires quadratic computation in every step because of the inversion. Pan et al. [66] choose to approximate the inverse $B^{-1}$ by a low-rank approximation, for which they utilize a truncated singular value decomposition (SVD), which admits a faster update, but is still in the order of $O(ld)$, when $l$ is the dimension of the feature vectors and $d$ the dimension of the chosen truncation. For a truncation dimension

**Algorithm 4.2** Accelerated TDC Algorithm

**Input:** Parameters $L_0 > L$ and $\nu \in [0, 1]$ for the optimization, sequence of parameters $\beta_k$ for the computation of the gradient in TDC, sequence of state transitions and rewards $(\phi_k, r_k, \phi'_k)$.

**Initialize:** Given an arbitrary guess $y_0 \in \mathbb{R}^l$, set $z_0 = y_0$, $w_0 = 0$, and $k = 1$.

**repeat**

Update $L_k = (\nu\sqrt{k-1} + 1)L_0$

Compute $\theta_k = (1 - \nu)y_{k-1} + \nu z_{k-1}$

Draw a sample $(\phi_k, r_k, \phi'_k)$

Compute a stochastic estimate of the TDC gradient:

$\delta(\theta_k, \theta_k) = r_k + \gamma\phi_k'^{\top}\theta_k - \phi_k^{\top}\theta_k$

$g_k = -\phi_k\delta(\theta_k, \theta_k) + \gamma\phi_k(\phi'_k)^{\top}w_k)$

$w_{k+1} = w_k + \beta_k\phi_k(\delta(\theta_k, \theta_k) - \phi_k^{\top}w_k)$

Compute the SAGE update:

$y_k = \theta_k - \frac{1}{L_k}g_k$

$z_k = z_{k-1} - \frac{\nu}{L_k}g_k$

Set $k = k + 1$

**until** converged

**Output:** weight vector $\theta = y_k$.

of $d = 0$ the method reduces to standard TD(0) learning, whereas for the extreme $d = l$ the method resembles the well known LSTD algorithm. This algorithm is therefore here not being considered due to the clear non-linear computational complexity.

## 4.5. Combined Regularization and Acceleration

In Chapter 3 a feature selection algorithm using a regularized formulation of the objective functions was derived. Due to the fact that this algorithms is also a stochastic gradient descent, it in terms suffers from the same drawbacks, including slower convergence in terms of samples. As this regularized formulation is also convex and L-Lipschitz, a modified version of the Nesterov's accelerated gradient can be applied in order to speed up convergence for the regularized formulation.

Let us recall the formulation of the regularized problem, its objective function properties

**Algorithm 4.3** Accelerated GTD2 Algorithm

**Input:** Parameters $L_0 > L$ and $\nu \in [0, 1]$ for the optimization, sequence of parameters $\beta_k$ for the computation of the gradient in GTD2, sequence of state transitions and rewards $(\phi_k, r_k, \phi'_k)$.

**Initialize:** Given an arbitrary guess $y_0 \in \mathbb{R}^l$, set $z_0 = y_0$, $w_0 = 0$, and $k = 1$.

**repeat**

Update $L_k = (\nu\sqrt{k-1} + 1)L_0$

Compute $\theta_k = (1 - \nu)y_{k-1} + \nu z_{k-1}$

Draw a sample $(\phi_k, r_k, \phi'_k)$

Compute a stochastic estimate of the GTD2 gradient:

$\delta(\theta_k, \theta_k) = r_k + \gamma\phi_k'^{\top}\theta_k - \phi_k^{\top}\theta_k$

$g_k = -\phi_k(\phi_k - \gamma\phi'_k)^{\top}w_k$

$w_{k+1} = w_k + \beta_k\phi_k(\delta(\theta_k, \theta_k) - \phi_k^{\top}w_k)$

Compute the SAGE update:

$y_k = \theta_k - \frac{1}{L_k}g_k$

$z_k = z_{k-1} - \frac{\nu}{L_k}g_k$

Set $k = k + 1$

**until** converged

**Output:** weight vector $\theta = y_k$.

and derive a solution algorithm. The regularized MSPBE is written as

$$\begin{aligned}\hat{J}_{\mathrm{TDC}}(\theta) \; &: \mathbb{R}^n \to \mathbb{R} \\ &\theta \to \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta))\|_\xi^2 + \eta\|\theta\|_1 \\ &= J_{\mathrm{TDC}}(\theta) + \eta\|\theta\|_1.\end{aligned} \tag{4.33}$$

The following two necessary properties can now be derived and afterwards it is possible to apply a stochastic variant of Nesterov's accelerated gradient descent.

**Lemma 5** (Convexity of Regularized MSPBE)**.** *Assume that the feature matrix $\Phi \in \mathbb{R}^{n\times l}$ is full rank and the Markov chain defined by $P \in \mathbb{R}^{n\times n}$ is irreducible and aperiodic. Furthermore, there exists a unique limiting distribution $\xi \in \mathbb{R}^n$, which satisfies $P\xi = \xi$ with $\xi_i > 0, \forall i \in 1 \dots n$, then the $\ell_1$-regularized MSPBE, defined in Equation* (4.33) *is strongly convex.*

*Proof.* We have, that for two convex functions $f(x)$ and $g(x)$, that the sum $h(x) = f(x) + g(x)$ is also convex. Additionally, observe, that in Lemma 1, the unregularized MSPBE is strongly convex. We know, that the $\ell_1$ norm is convex as it is the special case of the $\ell_p$ pseudo norm for $p = 1$, which is known to be a true norm for $p \geq 1$ due to the *Minkowski inqeuality* and true norms are convex.

To conclude, the cost function $\hat{J}_{\mathrm{TDC}}(\theta) = J_{\mathrm{TDC}}(\theta) + \eta\|\theta\|_1$ is a sum of two convex functions convex in $\theta$ and therefore overall also convex in $\theta$. ■

**Lemma 6** ($L$-Lipschitz Gradient of the Regularized MSPBE)**.** *Let the feature matrix $\Phi \in \mathbb{R}^{n\times l}$ be full rank and the Markov chain defined by $P \in \mathbb{R}^{n\times n}$ be irreducible and aperiodic. Furthermore, there exists a unique limiting distribution $\xi \in \mathbb{R}^n$, which satisfies $P\xi = \xi$ with $\xi_i > 0, \forall i \in 1 \dots n$. Then the gradient of the MSPBE, defined in equation* (4.33) *is L-Lipschitz with*

$$L = (1+\gamma)^2 \max_j \|\theta_j\|_2^2 + |\eta|. \tag{4.34}$$

*Proof.* Observe, that the $\ell_1$ norm is L-Lipschitz continuous with $L = 1$. Recall, that a function $f(x) : \mathcal{X} \to \mathcal{Y}$ is L-Lipschitz if $\|f(x) - f(y)\| \leq L\|x - y\|$, $\forall x \in \mathcal{X}, y \in \mathcal{X}$. Due to the reverse triangle inequality, we can write for the $\ell_1$ norm

$$||x| - |y|| \leq L|x - y|, \tag{4.35}$$

with $L = 1$.

Let $f(x) : \mathcal{X} \to \mathcal{Y}$ be a function that is $L_f$-Lipschitz and $g(x) : \mathcal{X} \to \mathcal{Y}$ be $L_g$-Lipschitz, then it holds for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$

$$\begin{aligned} |(f+g)(x) - (f+g)(y)| &= |(f(x) - f(y)) + (g(x) - g(y))| \\ &\leq |f(x) - f(y)| + |g(x) - g(y)| \\ &\leq L_f|x - y| + L_g|x - y| \\ &= (L_f + L_g)|x - y|, \end{aligned} \tag{4.36}$$

where the first inequality is due to the triangle inequality and the second due to the definition

of Lipschitz continuity. Additionally, for a constant $c$ we have

$$\begin{aligned} |(cf(x)) - (cf(y))| &= |c(f(x) - f(y))| = |c||f(x) - f(y)| \\ &\leq |c|L_f|x - y|. \end{aligned} \tag{4.37}$$

Taking these results and the fact that the cost function $\hat{J}_{\mathrm{TDC}}(\theta) = J_{\mathrm{TDC}}(\theta) + \eta\|\theta\|_1$ is a linear combination of two L-Lipschitz continuous functions, we can conclude, that $\hat{J}_{\mathrm{TDC}}(\theta)$ is L-Lipschitz with

$$L \leq (1+\gamma)^2 \max_j \|\theta_j\|_2^2 + |\eta| \tag{4.38}$$

■

To derive an accelerated version of the regularized gradient descent algorithm, we have to revisit the work in [34] and review the general algorithm description. Assume, that the cost function is given by $f_\theta(x) + \eta\|\theta\|_1$, then the general algorithm description for the stochastic SAGE can be seen in Algorithm 4.4. We recognize a parallel construction to

**Algorithm 4.4** SAGE-based Online Learning

**Input:** Parameters $L_0 > L$ and $\nu \in [0, 1]$ for the optimization,
**Initialize:** Given an arbitrary guess $y_0 \in \mathbb{R}^l$, set $z_0 = y_0$, and $k = 1$.
**repeat**
  Update $L_k = (\nu\sqrt{k-1} + 1)L_0$
  Compute $\theta_k = (1-\nu)y_{k-1} + \nu z_{k-1}$
  Compute $y_k = \arg\min_\theta \left\{ \langle \nabla_\theta f(\theta_k), \theta - \theta_k \rangle + \frac{L_k}{2}\|\theta - \theta_k\|_2^2 + \|\theta\|_1 \right\}$
  Update $z_k = z_{k-1} - \frac{\nu}{L_k}(L_k(\theta_k - y_k))$.
**until** converged
**Output:** weight vector $\theta = y_k$.

the regularized FOBOS algorithm from the previous chapter. Step 3 in Algorithm 4.4 resembles a similar formulation as the soft-thresholding formulation and solution. And it is the formulation of the proximal sub-gradient optimization. Indeed, the second step of selecting the intermediate point $\theta_k$ as the reference center for the proximal step is an interpolation between the previous iteration's results, whereas the last step then updates these interpolation points $z_k$. It is therefore advised to apply here the derivations from the previous chapter. Let us recall, that the soft thresholded update for the proximal

subgradient step of the regularized algorithms was written as

$$\begin{aligned} \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k u_k \phi_k (\phi_k - \gamma \phi_k')^\top\right), \qquad \text{(GTD-IST)} \\ u_{k+1} &= u_k + \beta_k(\phi_k \delta(\theta_k, \theta_k) - u_k), \end{aligned} \tag{4.39}$$

for the GTD-IST algorithm, and the improved versions for GTD2-IST and TDC-IST as

$$\begin{aligned} \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \phi_k (\phi_k - \gamma \phi_{k+1})^\top w_k\right) \qquad \text{(GTD2-IST)} \\ \theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \delta(\theta_k, \theta_k) - \alpha_k \gamma \phi_k (\phi_{k+1})^\top w_k\right) \qquad \text{(TDC-IST)} \\ w_{k+1} &= w_k + \beta_k \phi_k (\delta(\theta_k, \theta_k) - \phi_k^\top w_k). \end{aligned} \tag{4.40}$$

By now taking the step size to $\alpha_k = \frac{1}{L_k}$ in the SAGE based formulation, we can write down the three accelerated regularized algorithm versions as seen in Algorithm 4.5. Note, that in

**Algorithm 4.5** Accelerated Regularized GTD

**Input:** Parameters $L_0 > L$ and $\nu \in [0, 1]$ for the optimization, $\eta$ to control the regularization, $\kappa > 0$ to derive the secondary weights $\beta_k$,

**Initialize:** Given an arbitrary guess $y_0 \in \mathbb{R}^l$, set $z_0 = y_0$, $u_0 = 0$ and $w_0 = 0$ respectively and $k = 1$.

**repeat**

Update $L_k = (\nu\sqrt{k-1} + 1)L_0$

Compute $\theta_k = (1 - \nu)y_{k-1} + \nu z_{k-1}$

Draw a sample $(\phi_k, r_k, \phi_k')$

Set step-sizes $\alpha_k = \frac{1}{L_k}$ and $\beta_k = \kappa \alpha_k$

Compute a stochastic update of the respective gradient:

$y_k = \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k u_k \phi_k (\phi_k - \gamma_k \phi_k')^\top\right), \quad$ (GTDa-IST)

$u_k = u_k + \beta_k(\phi_k \delta(\theta_k, \theta_k) - u_k)$

$y_k = \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \phi_k (\phi_k - \gamma \phi_{k+1})^\top w_k\right) \quad$ (GTD2a-IST)

$y_k = \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k \delta(\theta_k, \theta_k) - \alpha_k \gamma \phi_k (\phi_{k+1})^\top w_k\right) \quad$ (TDCa-IST)

$w_{k+1} = w_k + \beta_k \phi_k (\delta(\theta_k, \theta_k) - \phi_k^\top w_k)$

SAGE Update $z_k = z_{k-1} - \frac{\nu}{L_k}(L_k(\theta_k - y_k))$.

**until** converged

**Output:** weight vector $\theta = y_k$.

this version, the step sizes for the secondary set of weights are set explicitly to $\beta_k = \kappa\alpha_k$, with $\kappa > 0$ to have a quasi-stationary update of the secondary weights from the viewpoint of the update rate of the primary weights, for details cf. [78].

## 4.6. Experiments and Results

As the aspects of the algorithms of this chapter, accelerated regularized GTD2 and TDC, that are to be studied are more or less similar to those in the previous chapter, the same set of experiments – the random MDP and the linear cart pole balance task – are used.

### Convergence Speed

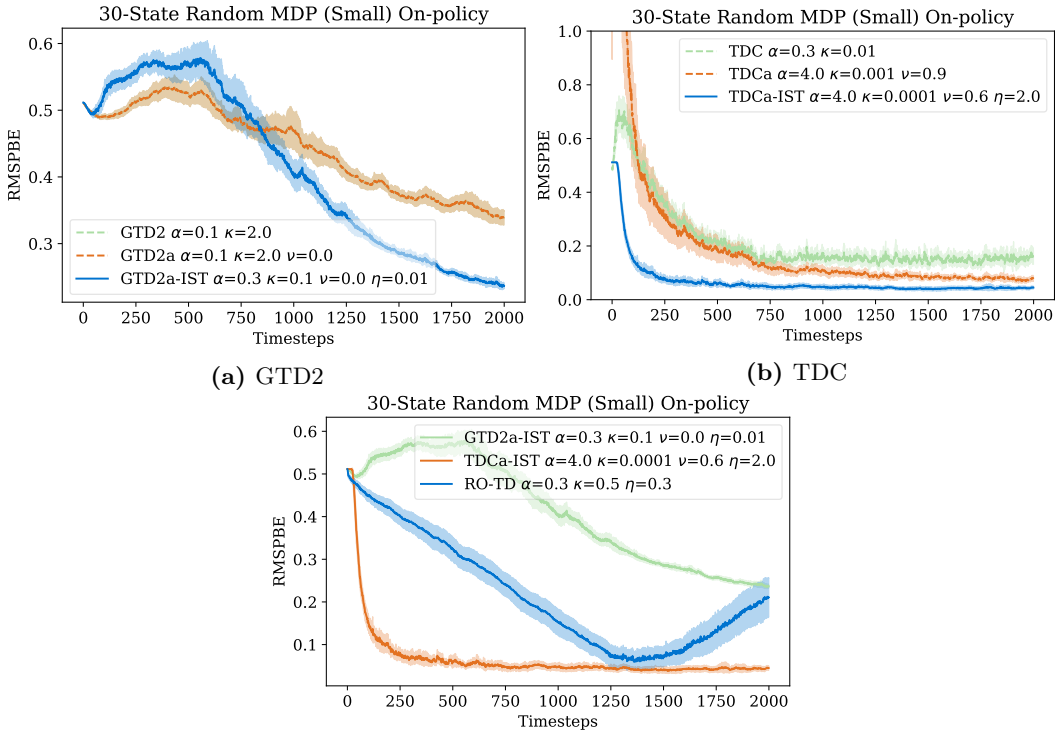
To get a basic overview on the convergence properties, all algorithms were run on the small random MDP and the results can be seen in Figure 4.2. Here in Figures 4.2a and 4.2b it can be seen, that for GTD2 and TDC by just applying acceleration a marginal advantage is gained. Especially for TDC the attained final error is a bit lower than for the non-accelerated version. But when combined with regularization, the gain in convergence speed and final attained error is more pronounced. TDC converges with less samples to the lowest final error and GTD2 converges to a lower error after about half the samples in the experiment. When looking at the performance of RO-TD in Figure 4.2c the performance of this algorithm can be put in between GTD2 and TDC. It is to note, that in this cases RO-TD seems to struggle with the chosen parameter setting in the search space, as the error tends to increase again after a certain number of samples.

### Size of Domain

When increasing the size of the test domain to the 400 state random MDP, the performance exhibits similar behavior. The TDC algorithm converges faster than the unregularized and non-accelerated counterparts with less samples and manages to attain a lower final error, which can be seen in Figure 4.3.

### Noise Sensitivity and Sparsity

As with the thresholded algorithms a study of robustness against additive noise in the environment, features is useful in gaining an understanding of the accelerated version of the




**(a)** GTD2 **(b)** TDC

**(c)** RO-TD

**Figure 4.2.:** Comparison of convergence for GTD2, TDC and RO-TD

gradient TD algorithms. To study this, the small random MDP environment is run with an increasing amount of noisy features added to the original features of the environment.

In Figure 4.4 on the x-axis the number of additional noise features are plotted and on the y-axis the final error of the algorithms reached after convergence. It can be seen that the original GTD2 algorithm struggles to converge starting from approximately 25 added additional noise features and the error continues to go up. This is the expected behavior as no additional robustness against noise is implemented in this algorithm. The accelerated GTD2a algorithm seems to handle the added noise in a better way. Observe, that the chosen parameter set is exactly the same as for the original GTD2 algorithm and the acceleration constant was found to be optimal at $\nu = 0$ by parameter grid search. The better handling of noise in this setting can be explained by the adaptive update step size based on the Lipschitz constant in the modified algorithm. This seems to dampen gradient steps in a

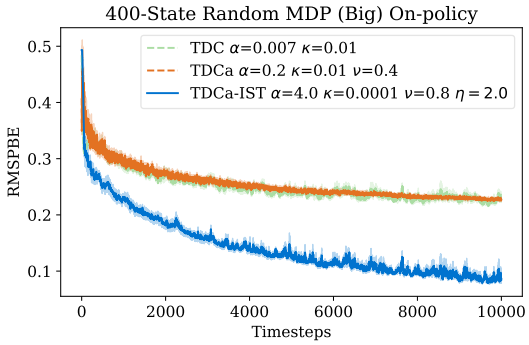



**Figure 4.3.:** Convergence speed of TDCa on the large random MDP (400 states)

*wrong* direction due to noise. The same can be observed for the accelerated and thresholded algorithm variant GTD2a-IST. Here acceleration with $\nu = 0.07$ helps with faster convergence (which unfortunately cannot be seen in this plot because only the final error values are drawn) and the thresholding adds robustness against noise.

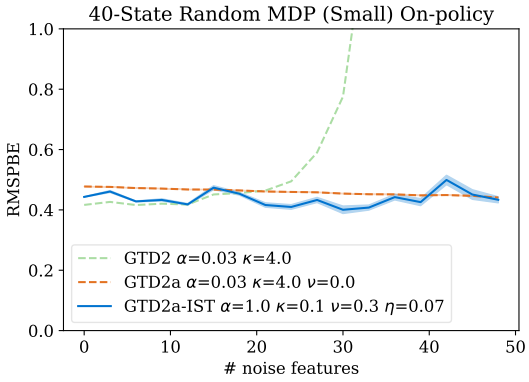
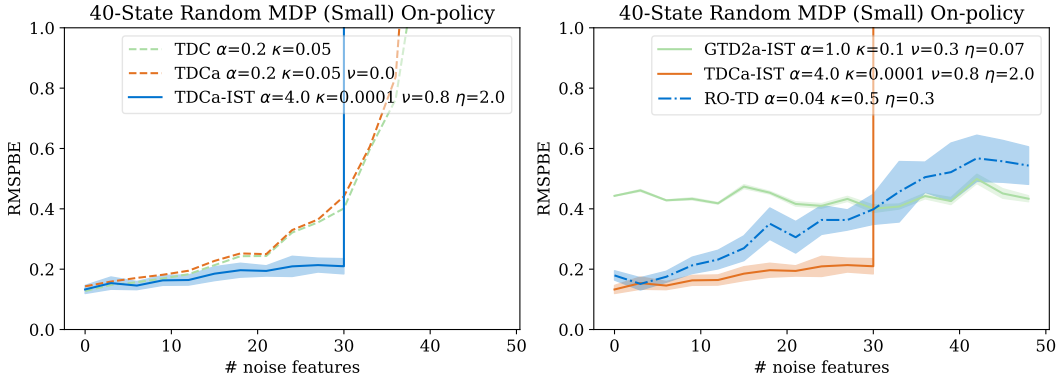



**Figure 4.4.:** Noise sensitivity of accelerated GTD2

In Figure 4.5a the same setup can be inspected for the TDC algorithm family. Again for the accelerated version the same parameter set as for the original TDC algorithm version was found to be optimal. In contrast to the GTD2 algorithm the Nesterov update does not seem to help mitigate the negative influence of noise in this case, but only the thresholding in the TDCa-IST algorithm helps to increase convergence speed up to 30 noisy additional features. After that algorithm convergence behavior drastically breaks down and the final

error obtained is far worse than for the original or accelerated only version. Also note that this drastic breakdown of performance seems to stem from the large $\nu = 2$ acceleration parameter, which achieves quick convergence, but as soon as the regularization breaks down after more than 30 noisy features, the large acceleration also drastically amplifies the error.




**(a)** TDC **(b)** RO-TD

**Figure 4.5.:** Comparison to RO-TD and instability of TDCa-IST

Figure 4.5b compares the accelerated and thresholded gradient TD algorithms to their closest contender RO-TD. While the performance of the GTD family of algorithms is as described before, RO-TD exhibits typically expected behavior in the presence of noise. With an increasing amount of noisy features, the algorithm degrades gracefully. The final error attained goes up as the variance in results also increases with more noise.

Similar to Section 3.7.4, to further investigate regularization and acceleration, we now direct our attention to the sparsity of the solution vector. In Figure 4.6 on the x-axis, the elements of the vector $\theta$ and on the y-axis the magnitude of the solution vector components are plotted.

In general a lower final error corresponds to a sparser solution, as can be seen in the fact, that the solution vector for TDC and RO-TD is sparser than the one for GTD2. The behavior seems to be similar as with the non-accelerated algorithms. TDC and RO-TD possess an inherent advantage in selecting the relevant features and suppressing the noisy feature components. In general it can be concluded that the sparsity of solution is increased not by a great amount but thresholding does help with overall algorithm performance.

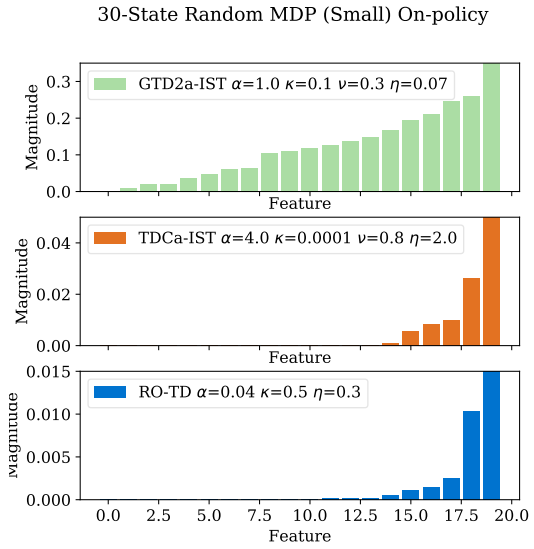
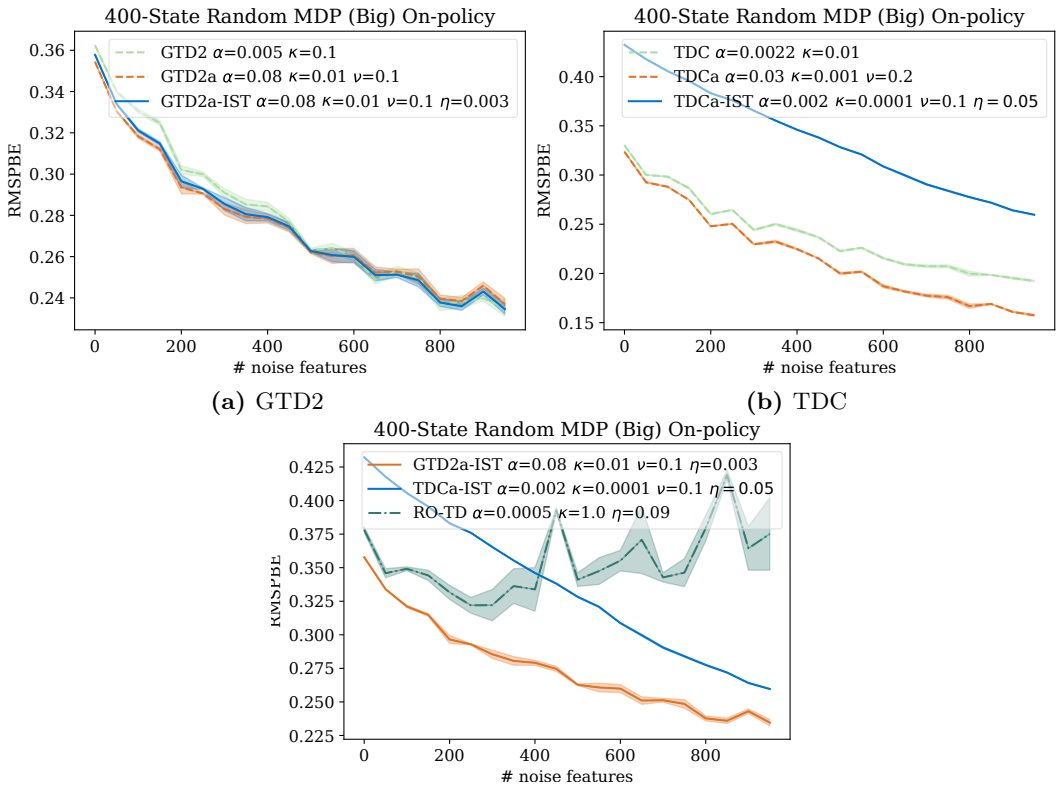



**Figure 4.6.:** Sparsity of solution for accelerated algorithms

### Influence of Problem Size

The next question to be asked is how the size of test domain influence the algorithm performance. When looking at Figure 4.7, we can see the number of sample steps on the x-axis and the Bellman error on the y-axis for the different algorithms. For GTD2 in Figure 4.7a, neither acceleration nor thresholding seems to have a great impact on the overall algorithm performance, for TDC in Figure 4.7b there is a difference between the non-thresholded and thresholded algorithm convergence speed. However, acceleration counteracts the beneficial influence of thresholding again as larger gradient steps are taken. Overall, in the 400 state random MDP the inherent noise of the environment is so large that the additional noisy features do not seem to have any great impact on the algorithm performance.

For RO-TD in Figure 4.7c, the algorithm seems to struggle with convergence in this environment. This is due to the fact, that the large random MDP already contains a large amount of noise in its basic setting and therefore non-thresholded algorithms struggle to converge to a stable value and variance of the results is greatly increased.




**(a)** GTD2 **(b)** TDC

**(c)** RO-TD

**Figure 4.7.:** Noise sensitivity on the big random MDP domain

### Parameter Sensitivity

In addition to the previous algorithms, the accelerated and thresholded algorithms have two more parameters, acceleration $\nu$ and regularization $\eta$ on top of the usual gradient step size parameters $\alpha$ and $\kappa$. These extra parameters have to be tuned for optimal learning results. First, we will have a look at the regularization parameter $\eta$ and afterwards study the interplay of regularization $\eta$ versus acceleration $\nu$.

In Figure 4.8, on the x-axis different values for the parameter $\eta$ are plotted and on the y-axis the resulting Bellman error value after convergence. With increasing regularization parameter value, the noise features get attenuated and the algorithm performance increases. Such a typical behavior can be seen for the TDCa-IST algorithm, where with small regular-

ization parameter, the variance of results and error is high and both decrease with increasing parameter value until the effect levels off depending on the amount of noise in the environment. For RO-TD the effect is even more pronounced, where for values of $\eta$ too small, the algorithm diverges. The GTD2a-IST algorithm seems to have little sensitivity on the chosen regularization parameter value. Note that the final error of the GTD2 algorithms was in general higher than for the other algorithms, similar for small as for moderate values of $\eta$.

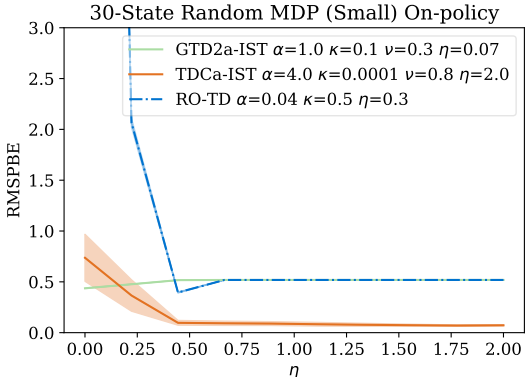
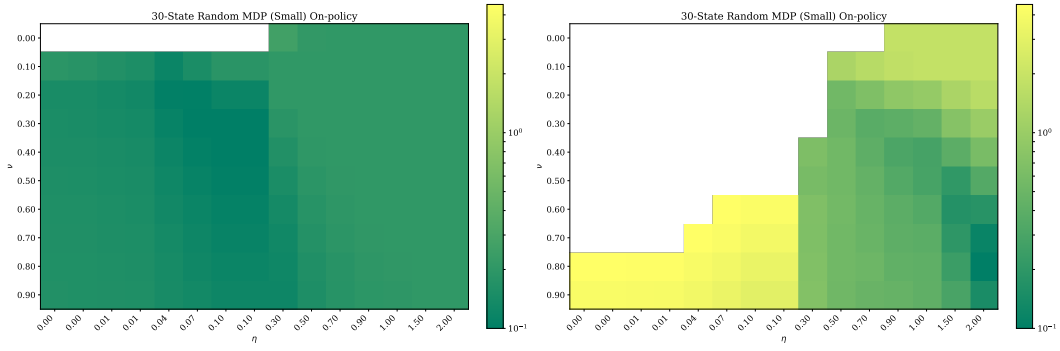



**Figure 4.8.:** Parameter sensitivity with respect to regularization parameter $\eta$

Similar as in Section 3.7.5, we conclude similarly that setting the parameter $\eta$ to some small value but not too small in the presence of noise is safe and again larger parameter values do not help in speeding up algorithm convergence as the effect of regularization seems to plateau off depending on the amount of noise.

The last aspect to study with regard to parameter settings is to look at the interplay of $\eta$ and $\nu$. In Figure 4.9 on the x-axis the parameter values for $\eta$ and on the y-axis the parameter values for $\nu$ are plotted. Each box depicts via color coding the final Bellman error attained for a parameter combination, where yellow shades correspond to a larger value and darker green to a lower value of final error. White areas mean that the algorithm did not converge with the chosen parameter combination.

For GTD2, it can be concluded that convergence is possible for a wider range of parameter combinations than for TDC. It is worth noticing that even shades with light green still mean, that the final error is quite large (around ten), where only dark green areas mean that the algorithm actually found a suitable solution. For TDC the area of convergence is drastically shrunk. Only parameter combinations around $\eta = 2$ and $\nu = 0.8$ yield satisfactory




**(a)** GTD2 **(b)** TDC

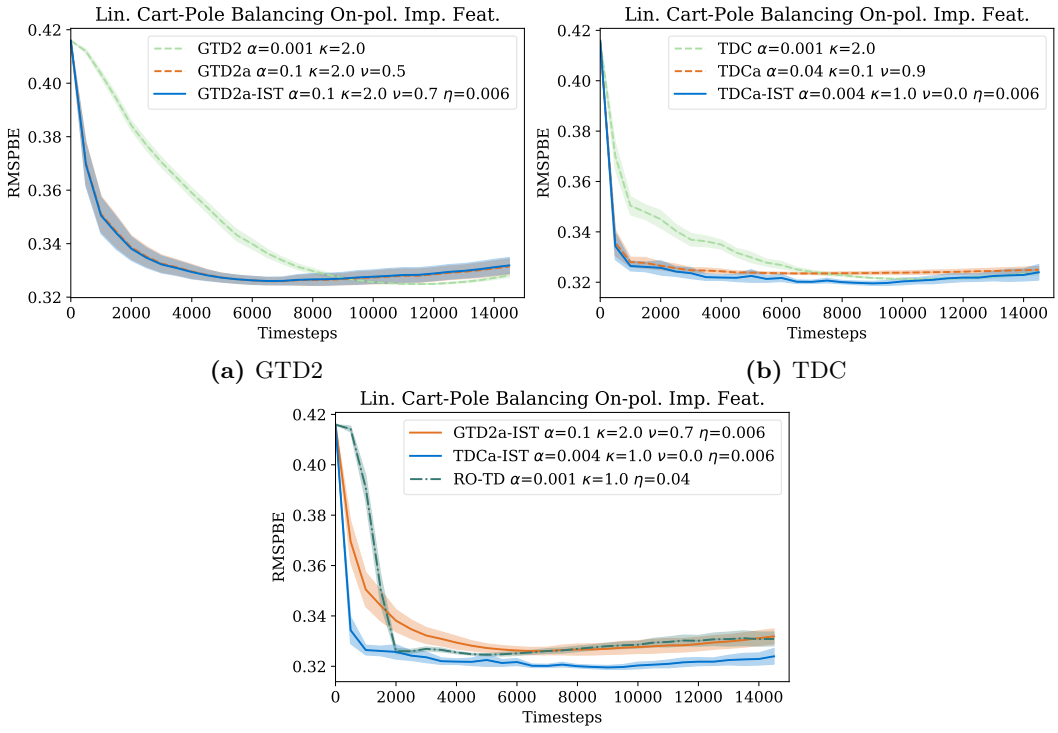
**Figure 4.9.:** Parameter sensitivity on the noisy small random MDP with respect to $\eta$ and $\nu$ while fixing the optimal $\alpha$ and $\kappa$ for the environment.

results and the performance quickly degrades when deviated from the optimal area. This means that the TDC algorithm is able to attain better final errors for optimal parameter combinations, while being susceptible for deviations from this optimal area and therefore might be harder to tune in real life applications.

### Continuous Domain Performance

Finally, let us focus towards the continuous domain performance, which closest models the performance in applications, where no ground truth representation of the state space is possible any more. The results in Figure 4.10, where number of samples is plotted on the x-axis and Bellman error value on the y-axis, show, that acceleration helps in this environment. In Figures 4.10a and 4.10b it can be seen, that compared with the unmodified algorithms, the accelerated version converge with much less samples in the presence of noise without having regularization activated. For GTD2, the additional regularization seems not to help much in terms of final low error attained, whereas for TDC the combination of acceleration and regularization brings a slight advantage.

Compared to RO-TD in Figure 4.10c, both algorithms GTD2 and TDC have similar performance, where TDC achieves a low error more consistently along more samples in the environment. RO-TD seems to struggle with the additional noise introduced, when more samples are drawn from the environment after having converged to the lowest cost function value.




**(a)** GTD2 **(b)** TDC

**(c)** RO-TD

**Figure 4.10.:** Algorithm performance on the continuous linear cart pole balancing domain

## 4.7. Summary

In this chapter accelerated and regularized versions of the stochastic gradient descent TD algorithms were derived. As a prerequisite the $\mu$-convexity and the L-Lipschitz continuity of the underlying cost function were analyzed.

The modified algorithms show a beneficial behavior in convergence speed in the random MDP domain settings. Even without regularization, the accelerated versions of the algorithms seem to converge up to four times faster in terms of samples needed on average since the stepsize is controlled by a predefined schedule. Like the non-accelerated versions of the GTD2 and TDC algorithm, the regularization parameter can be set to some small value in order to achieve good performance. Care has to be taken when selecting the parameter set for the TDC algorithm variant. Slight deviations from the optimal setting seem to de-

grade the performance significantly both in convergence speed as well as in final error value reached.

Compared with RO-TD both algorithms again show well behavedness and a clear advantage when the amount of noise is large in an environment.

# Chapter 5.

# Off-Policy and Multistep Algorithm Variants

As discussed in Section 2.6, TD methods have weaknesses to converge in certain application scenarios. Those weakness to converge can be gotten rid of if so-called *multistep* version of these algorithms are used. This means that not only one application of the Bellman operator is used, but a combination of multiple applications of the operator. Additionally, the convergence speed increases and the minimum error attained improves, this is often referred to as a decrease in *bias*. On the other hand the methods susceptibility to sampling noise is increased, this is referred to as an increased *variance*. The balance between those two is usually called the *bias-variance-tradeoff*.

Additionally, in many real world applications of reinforcement learning, it can be necessary to follow a specially crafted and *safe* policy[1], while trying to learn and improve a new policy. The samples gathered in the system then no longer fit the new and improved policy to be learned. This means that we want to learn with samples from a different policy and hence the name of this mode of learning is called *off-policy*. Unmodified TD learning often diverges in such a scenario and therefore special considerations in reformulating the learning problem and the cost function have to be taken. This variants of algorithms are discussed in Section 5.2.

[1] This means that the policy is known to well-behave and for example in an industrial robotic setting the machine is not going to harm bystanders or crash self driving cars.

## 5.1. Methods with a Multistep Lookahead

To derive regularized and accelerated stochastic gradient temporal difference learning algorithms in their respective multistep versions, let us first define a geometrically averaged Bellman operator as

$$\mathcal{T}_\mu^{(\lambda)} = (1-\lambda)\sum_{l=0}^{\infty}\lambda^l \mathcal{T}_\mu^{l+1}, \tag{5.1}$$

with parameter $\lambda \in (0,1)$ controlling the weighting of the averaging over the various length of applications of the Bellman operator and $\mathcal{T}^l$ is the $l$ times application of the operator. We then have the operator and the Bellman equation as

$$\mathcal{T}_\mu^{(\lambda)} V = R_\mu^{(\lambda)} + \gamma P_\mu^{(\lambda)} V, \tag{5.2}$$

where

$$P_\mu^{(\lambda)} = (1-\lambda)\sum_{l=0}^{\infty}\gamma^l\lambda^l P_\mu^{l+1}, \qquad R_\mu^{(\lambda)} = \sum_{l=0}^{\infty}\gamma^l\lambda^l P_\mu^l R_\mu = (I-\gamma\lambda P_\mu)^{-1}R_\mu. \tag{5.3}$$

To observe the claimed speedup in convergence, we can look at the following proposition

**Proposition 4.** *Similar to Proposition 1, we have for the mapping $\mathcal{T}_\mu^{(\lambda)}$ and the composition with the projection $\Pi_\mu \mathcal{T}_\mu^{(\lambda)}$ both as contraction mappings with the modulus $\gamma_\lambda = \frac{\gamma(1-\lambda)}{1-\gamma\lambda}$ with respect to the $\xi$ weighted norm and*

$$\|V_\mu - \Phi\theta^*\|_\xi \leq \frac{1}{\sqrt{1-\gamma_\lambda^2}}\|V_\mu - \Pi_\mu V_\mu\|_\xi, \tag{5.4}$$

*with $\theta^*$ being the solution to the fixed point equation for $\Pi_\mu \mathcal{T}_\mu^{(\lambda)}$.*

*Proof.* For the full proof refer to [5]. ■

This proposition implies that the contraction modulus goes to 0 when $\lambda$ goes towards 1, hence a faster convergence. Additionally, the error bound above gets better in the same way as $\lambda$ goes towards 1. This is, as indicated above, called the reduction of the *bias*. At the same time, since $\mathcal{T}^{(\lambda)}$ is an average of multiple applications of the Bellman operator, noise from the sampling process has a more pronounced impact for bigger values of $\lambda$. Therefore, the *variance* of the respective method is increased. In practice, there can be no definitive

recommendation be given on how to set this parameter, but the optimal setting has to be found out by experimentation.

### 5.1.1. Multistep Least Squares Temporal Difference Learning and Policy Evaluation

Using the above definition of the geometrically averaged Bellman operator for multistep methods, we can analogous to Section 2.4, derive the two algorithms LSTD and LSPE in a multistep variant.

The previous derivation does rely on the stochastic estimate of the matrix $C_k$ and the vector $d_k$. Similar Section 2.4 two variants can be derived as $C^{(\lambda)} = \Phi^\top \Xi (I - \gamma P_\mu^{(\lambda)})\Phi)$ and $d^{(\lambda)} = \Phi \Xi R_\mu^{(\lambda)}$. An extended multistep estimation variant of those two quantities of Equation (2.24) is then given by

$$\begin{aligned} C_k^{(\lambda)} &= \frac{1}{k+1} \sum_{t=0}^{k} \phi(s_t) \sum_{m=t}^{k} \gamma^{m-t} \lambda^{m-t} \left(\phi(s_m) - \gamma \phi(s_{m+1})\right)^\top, \\ d_k^{(\lambda)} &= \frac{1}{k+1} \sum_{t=0}^{k} \phi(s_t) \sum_{m=t}^{k} \gamma^{m-t} \lambda^{m-t} r(s_m, s_{m+1}). \end{aligned} \tag{5.5}$$

Often, an intermediate vector, called *eligibility trace* [5, 77] is introduced to ease stating update formulas. This eligibility vector is given by

$$z_t = \sum_{m=0}^{t} (\gamma\lambda)^{t-m} \phi(s_m), \tag{5.6}$$

and the sample by sample update is then defined by

$$\begin{aligned} z_k &= \gamma\lambda s_{k-1} + \phi(s_k), \\ C_k^{(\lambda)} &= (1-\eta_k) C_{k-1}^{(\lambda)} + \eta_k z_k (\phi(s_k) - \gamma\phi(s_{k+1}))^\top, \\ d_k^{(\lambda)} &= (1-\eta_k) d_{k-1}^{(\lambda)} + \eta_k z_k r(s_k, s_{k+1}), \end{aligned} \tag{5.7}$$

where $\eta_k = \frac{1}{k+1}$ is the step size and $C_0^{(\lambda)} = 0, d_0^{(\lambda)} = 0$.

As we now have for the fixed point solution of the multistep Bellman equation $C^{(\lambda)}\theta^{*(\lambda)} = d^{(\lambda)}$, the solution which leads to the LSTD($\lambda$) algorithm for the step $k$ at which $k$ samples

are available is

$$\theta_k^{(\lambda)} = \left(C_k^{(\lambda)}\right)^{-1} d_k^{(\lambda)}, \tag{5.8}$$

where $\theta_k^{(\lambda)}$ is the LSTD solution at step $k$.

If we express the projection, before formulated as $\Pi_\mu$ in the form of a least squares minimization in the linear value function approximation parameters, we arrive at

$$\theta_{k+1} = \arg\min_{\theta \in \mathbb{R}^l} \|\Phi\theta - \mathcal{T}_\mu^{(\lambda)}(\Phi\theta_k)\|_\xi^2. \tag{5.9}$$

Then we can follow the derivation in [5], Chapter 6, and rewrite this minimization problem in the terms of single samples in the sampled setting. Now assume that at algorithm iteration $k$, there are $k$ samples available, we than can write an intermediate goal $\tilde{\theta}_{k+1}$ as

$$\tilde{\theta}_{k+1} = \arg\min_{\theta \in \mathbb{R}^l} \sum_{t=0}^{k} \left[\phi(s_i)^\top \theta - \phi(s_i)^\top \theta_k - \sum_{m=i}^{k} \left((\gamma\lambda)^{m-i}\delta_{k,m}\right)\right]^2, \tag{5.10}$$

where $\delta_{k,m} = r(s_m, s_{m+1}) + \gamma\phi(s_{m+1})^\top\theta_k - \phi(s_m)^\top\theta_k$ is the TD-error for sample $m$ at algorithm iteration $k$. By setting the gradient of Equation (5.10) to zero as in

$$0 \stackrel{!}{=} \sum_{i=0}^{k} \left[(\phi(s_i)^\top\theta)\phi(s_i) - (\phi(s_i)^\top\theta_k)\phi(s_i) - \left(\sum_{m=i}^{k}(\gamma\lambda)^{m-i}\delta_{k,m}\right)\phi(s_i)\right], \tag{5.11}$$

bringing the first term to the left side as

$$\sum_{i=0}^{k}(\phi(s_i)^\top\theta)\phi(s_i) = \sum_{i=0}^{k}\left[(\phi(s_i)^\top\theta_k)\phi(s_i) + \left(\sum_{m=i}^{k}(\gamma\lambda)^{m-i}\delta_{k,m}\right)\phi(s_i)\right], \tag{5.12}$$

and extracting $\theta$ in each respective summand

$$\sum_{i=0}^{k}(\phi(s_i)\phi(s_i)^\top\theta) = \sum_{i=0}^{k}\left[\phi(s_i)(\phi(s_i)^\top)\theta_k + \phi(s_i)\left(\sum_{m=i}^{k}(\gamma\lambda)^{m-i}\delta_{k,m}\right)\right], \tag{5.13}$$

one can get the solution of LSPE by matrix inversion as

$$\tilde{\theta}_{k+1} = \left(\sum_{i=0}^{k}\phi(s_i)\phi(s_i)^\top\right)^{-1}\left(\sum_{i=0}^{k}\phi(s_i)\left[\phi(s_i)^\top\theta_k + \sum_{m=i}^{k}(\gamma\lambda)^{m-i}\delta_{k,m}\right]\right). \tag{5.14}$$

If this method is then used within a policy iteration scheme to solve the evaluation part, the estimate given by Equation (5.14) can contain some noise due to the sampling process of this batch. The stability of the LSPE method can therefore be increased by taking the average over the last and the new information of data as $\theta_{k+1} = (1 - \eta_k k)\theta_k + \eta_k \tilde{\theta}_{k+1}$ with some positive stepsize $\eta_k \in (0, 1]$.

### 5.1.2. Multistep Temporal Difference Learning

Taking the multistep Bellman equation in the same form as Equation (2.41) but with the geometrically averaged Bellman operator from Equation (5.1) and then follow similar derivations, we obtain a multstep version of the TD algorithm as

$$\begin{aligned} z_k &= \gamma\lambda s_{k-1} + \phi(s_k), \\ \theta_k &= \theta_k + \alpha_k z_k \delta(\theta_{k-1}, \theta_k), \end{aligned} \tag{5.15}$$

where again $\alpha_k$ is a sequence of adequate step sizes and $\delta(\theta_{k-1}, \theta_k)$ is the TD-error. Usually, in the literature the quantity $z_k$ is referred to as the *eligibility trace* [77]. In this reference these methods are mostly presented to be favorable when the reward is delayed for several time steps and the eligibility trace can then be seen as accumulating the past knowledge going back with the sampling path through the MDP. In this cases the multistep versions can be faster and more efficient in learning the value function.

### 5.1.3. Multistep Gradient Temporal Difference Learning

Just as the previous algorithms, gradient TD learning, can be augmented using the geometrically averaged Bellman error of Equation (5.1). The resulting objective function of the projected mean squared Bellman error is then written as

$$\begin{aligned} J^{(\lambda)}_{\mathrm{TDC}}(\theta) &= \|\Phi\theta - \Pi_\mu \mathcal{T}^{(\lambda)}_\mu(\Phi\theta)\|^2_\xi = \|\Pi_\mu(\Phi\theta - \mathcal{T}^{(\lambda)}_\mu(\Phi\theta))\|^2_\xi \\ &= (\Phi\theta R^{(\lambda)}_\mu - \gamma P^{(\lambda)}_\mu \Phi\theta)^\top \Xi\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi(\Phi\theta R^{(\lambda)}_\mu - \gamma P^{(\lambda)}_\mu\Phi\theta) \end{aligned} \tag{5.16}$$

and the gradient is derived as

$$\frac{1}{2}\nabla_\theta(J^{(\lambda)}_{\mathrm{TDC}}(\theta)) = (\Phi^\top\Xi\Phi - \gamma\Phi^\top P^{(\lambda)}_\mu \Xi\Phi\theta)^\top(\Phi\Xi\Phi)^{-1}(\Phi^\top\Xi\Phi\theta - \Phi^\top\Xi R^{(\lambda)}_\mu - \gamma\Phi^\top\Xi P^{(\lambda)}_\mu\Phi\theta). \tag{5.17}$$

As was shown in Section 5.1.1, the two quantities $C_k^{(\lambda)}$ and $d_k^{(\lambda)}$ can be used to derive sampling based estimation of the two matrices for the geometrically averaged case. And with the collection of more and more samples when $k \to \infty$, $C_k^{(\lambda)} \to C^{(\lambda)} = \Phi^\top \Xi \Phi - \gamma \Phi^\top \Xi P^{(\lambda)} \Phi$ converges to the desired quantity. We now have an equivalent formulation in expected sampled value for the first part of the gradient equation, cf. [5]. The same procedure applies to $d_k^{(\lambda)} \to d^{(\lambda)} = \Phi^\top \Xi R_\mu^{(\lambda)}$, as $k \to \infty$. And we can again use the eligibility trace $z$ to rewrite the gradient in terms of expectations as

$$\begin{aligned}\frac{1}{2}\nabla_\theta(J_{\mathrm{TDC}}^{(\lambda)}(\theta)) &= -\mathbb{E}_{\xi,P_\mu}\left[z(\phi-\gamma\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[z\delta(\theta,\theta)\right] \\ &= \mathbb{E}_{\xi,P_\mu}\left[\phi\phi^\top - \gamma(1-\lambda)z(\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[z\delta(\theta,\theta)\right] \\ &= \mathbb{E}_{\xi,P_\mu}\left[z\delta(\theta,\theta)\right] - \mathbb{E}_{\xi,P_\mu}\left[\gamma(1-\lambda)z(\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[z\delta(\theta,\theta)\right],\end{aligned} \tag{5.18}$$

with $\delta(\theta,\theta)$ being the TD-error.

By again approximating $w(\theta) \approx \left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[z\delta(\theta,\theta)\right]$, we obtain the GTD2($\lambda$) algorithm as

$$\theta_{k+1} = \theta_k + \alpha_k z_k(\phi_k - \gamma\phi_{k+1})^\top w_k \tag{5.19}$$

and the TDC($\lambda$) algorithm as

$$\theta_{k+1} = \theta_k + \alpha_k\left(z_k\delta(\theta_k,\theta_k) - \gamma(1-\lambda)z_k(\phi_{k+1})^\top w_k\right) \tag{5.20}$$

with

$$w_{k+1} = w_k + \beta_k\left(z_k\delta(\theta_k,\theta_k) - \phi\phi^\top w_k\right) \tag{5.21}$$

and

$$z_{k+1} = \gamma\lambda z_k + \phi_k. \tag{5.22}$$

## 5.2. Off-Policy Algorithms

Up to now all algorithms working on samples are relying on these samples to be generated by a distribution induced by the current policy $\mu$. This is called *on policy* sampling as samples and norms for theoretical results are formulated with respect to $\xi$, the steady state

distribution of the underlying Markov chain, which itself does again rely on the policy $\mu$. Therefore, the samples where $s'$ indicates the successor state of $s$, are distributed as $(s, a, r, s') \sim P_\mu$, with $P_\mu$ being the combined distribution of the MDP and the policy $\mu$.

In some cases, it might now be beneficial to use samples generated by a different distribution than the one induced by the policy for which we would like to estimate the value function. This could be in cases, where some sampling policy is available but the value should be estimated with respect to some other optimal policy which cannot be used online as it could select actions that are unsafe for the system under study. Another reason is to enhance the so called *exploration*. This means, that parts of the state space get sampled by a modified policy that are not sampled as much by the policy currently available and therefore would increase accuracy in those regions. Most reinforcement learning literature refers to such policies as *behavior policies* $\mu_b$, whereas the policy for which the value should be estimated is called the *target policy* $\mu$.

Mathematically, we can express this change of sampling, where we use samples $(s, a, r, s') \sim P_{\mu_b}$ but estimate for $\mu$ as a different probability transition matrix $P_{\mu_b}$. It can be written as a weighted average of the original transition probability matrix $P_\mu$ and some additional square transition probability matrix $Q$ as

$$P_{\mu_b} = (1 - B)P_\mu + BQ, \tag{5.23}$$

where $B$ is a diagonal matrix with diagonal components $\beta_i$. The model therefore is sampled according to $P_\mu$ and occasionally, determined by the probabilities $\beta_i$, according to $Q$. $Q$ does not necessarily represent a existing transition probability matrix, but is introduced to enable the mathematical analysis of the off-policy sampled MSPBE. Later a concrete tool to relate the two policies $\mu$ and $\mu_b$ will be introduced. They will be related to each other by the means of so called *importance weighting* factors $\rho$. If the minimization of the respective objective functions would be now done by the above unmodified algorithms, the outcome would be the solution to the modified linear approximated Bellman equation

$$\Phi\theta = \Pi_{\mu_b}\mathcal{T}_{\mu_b}(\Phi\theta) = \Pi_{\mu_b}(R_{\mu_b} + \gamma P_{\mu_b}(\Phi\theta)). \tag{5.24}$$

Here $\Pi_{\mu_b}$ is a projection with respect to the modified norm $\|\cdot\|_{\xi_b}$, where $\xi_b$ is the steady state distribution of the Markov chain with respect to $P_{\mu_b}$. Therefore, the solution to

Equation (5.24) would be the parameters corresponding to the behavior policy $\mu_b$, but the real target we are after are the corresponding parameters vector for policy $\pi$, i.e. the solution to the Bellman equation

$$\Phi\theta = \Pi_{\mu_b}\mathcal{T}_{\mu}(\Phi\theta). \tag{5.25}$$

Note two particularities about this modified off-policy Bellman equation: first the projection is with respect to the modified steady state distribution $\xi_b$ as for a Markov chain $\|\Pi_{\mu}P_{\mu}\|_{\xi_b}$ is no longer necessarily a contraction since the steady state distributions $\xi_b$ and $\xi$ do not match and algorithms might diverge [71, 5, 8]. Secondly, this modified Bellman equation is with respect to the Bellman operator associated with target policy $\mu$ as this is the policy that we aim to compute the value function for.

A method to cope with all these problems and solve with respect to the modified off-policy Bellman equation (5.25), is to use *importance sampling*. In a nutshell, importance sampling is to express expected values with respect to different distributions, in our case to express these expected values for the target policy in terms of the distribution of the behavior policy.

If an expectation of a function $f : \mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S} \to \mathbb{R}$ over the the samples $(s, a, r, s')$ has to be calculated, then it can be written and with an empirical estimation of this expectation as

$$\mathbb{E}_{(s,a,r,s')\sim P_{\mu}}\left[f(s,a,r,s')\right] = \sum_{(s,a,r,s')} \xi(s,a,r,s')f(s,a,r,s') \approx \frac{1}{k}\sum_{t=0}^{k} f(s_t,a_t,r_t,s_{t+1}), \tag{5.26}$$

where $\xi$ is again the steady state distribution corresponding to the target policy $\mu$. In order to express the expectation in terms of the behavior policy, we can rewrite the above terms as

$$\begin{aligned}
\sum_{(s,a,r,s')} \xi(s,a,r,s')f(s,a,r,s') &= \sum_{(s,a,r,s')} \xi(s,a,r,s')\frac{\xi_b(s,a,r,s')}{\xi_b(s,a,r,s')}f(s,a,r,s') \\
&= \sum_{(s,a,r,s')} \xi_b(s,a,r,s')\frac{\xi(s,a,r,s')}{\xi_b(s,a,r,s')}f(s,a,r,s') \\
&= \sum_{(s,a,r,s')} \xi_b(s,a,r,s')\rho f(s,a,r,s') \\
&= \mathbb{E}_{(s,a,r,s')\sim P_{\mu_b}}\left[\rho f(s,a,r,s')\right],
\end{aligned} \tag{5.27}$$

where $\rho = \frac{\xi(s,a,r,s')}{\xi_b(s,a,r,s')}$. In practice, for the applications in this work, $\rho$ only depends on the quotient of the two policy probabilities as the rest of the model is considered unchanged. Therefore, $\rho = \frac{\mu(a|s)}{\mu_b(a|s)}$.

If we re-derive the update formulas for the most popular reinforcement learning algorithm, namely TD, and the algorithms used in this work, the GTD family of algorithms, then we obtain update formulas, that contain $\rho_k = \frac{\mu(a_k|s_k)}{\mu_b(a_k|s_k)}$. Obviously, the behavior policy should not contain any zeroes, which in turn means that the Markov chain associated with $P_{\mu_b}$ induced by the behavior or sampling policy has to be ergodic.

The modified update equations for the TD($\lambda$) algorithm are now given as

$$\begin{aligned} z_{k+1} &= \rho_k(\lambda\gamma z_k + \phi(s_k)), \\ \theta_{k+1} &= \theta_k + \alpha_k(r_k + \gamma\phi(s_{k+1})^\top\theta_{k-1} - \phi(s_k)^\top\theta_k)z_{k+1} = \theta_k + \alpha_k\delta_k(\theta_{k-1},\theta_k)z_{k+1}. \end{aligned} \tag{5.28}$$

Whereas the update for off-policy GTD2($\lambda$) and off-policy TDC($\lambda$) are rewritten as

$$\begin{aligned} \theta_{k+1} &= \theta_k + \alpha_k\left(\phi(s_k)\phi(s_k)^\top - \gamma\rho_k(1-\lambda)(\phi(s_{k+1})^\top z_k)\right)w_k \qquad \text{(GTD2}(\lambda)\text{)} \\ \theta_{k+1} &= \theta_k + \alpha_k\delta(\theta_k,\theta_k)z_k - \alpha_k\gamma\rho_k(1-\lambda)(\phi(s_{k+1})^\top z_k)w_k \qquad \text{(TDC}(\lambda)\text{)} \end{aligned} \tag{5.29}$$

with the update of the secondary weights as

$$w_{k+1} = w_k + \beta_k\left(\delta(\theta_k,\theta_k)z_k - \phi(s_k)(\phi(s_k)^\top w_k)\right). \tag{5.30}$$

## 5.3. Regularized and Accelerated Algorithm Variants

In this section, first the regularized off-policy algorithm variants shall be derived. After that acceleration shall be applied to this algorithm to complete the study.

As established in Section 3.4.1, imposing regularization with soft-thresholding on a gradient algorithm is as easy as interwebbing the stochastic gradient update step with the soft-thresholding operator $\Psi_{\alpha_k\eta}$ in the gradient update step of the objective function weights.

The update equations for GTD2 and TDC then can be written down as

$$\begin{aligned}
\theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k\left(\phi(s_k)\phi(s_k)^\top - \gamma\rho_k(1-\lambda)(\phi(s_{k+1})^\top z_k)\right) w_k\right) && \text{(GTD2}(\lambda)\text{-IST)}\\
\theta_{k+1} &= \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k\delta(\theta_k,\theta_k) z_k - \alpha_k\gamma\rho_k(1-\lambda)(\phi(s_{k+1})^\top z_k) w_k\right) && \text{(TDC}(\lambda)\text{-IST)}\\
w_{k+1} &= w_k + \beta_k\left(\delta(\theta_k,\theta_k) z_k - \phi(s_k)(\phi(s_k)^\top w_k)\right),
\end{aligned} \tag{5.31}$$

where $w_k$ are the auxiliary weights and $\eta$ is controlling the influence of the $\ell_1$ regularization penalty.

In order to obtain the Nesterov accelerated and regularized algorithm variants, we can follow a similar route. Take the above regularized equations and carefully rearrange the SAGE gradient update steps with the known update equation of accelerated TDC and GTD2. We then obtain the algorithms in Algorithm 5.1.

This step is valid as we will see in the next section, where the properties of the multistep cost functions will be studied. Also the selection of the initial parameter $L_0$ can be done following the Equation (5.39) derived from the upper bound on the norm of the feature vectors for each state.

## 5.4. Properties of the Cost Function

For the algorithms in the previous sections to be applicable, especially Nesterov's accelerated stochastic gradient method, the cost function has to fulfill the same conditions as in the previous chapters. Those are to ensure that the cost function is convex and posseses a L-Lipschitz gradient.

Let us recall that the objective function with a multistep Bellman operator and a projection $\Pi_{\mu_b}$ according to the behavior policy (as introduced in Equation (5.25)) can be denoted as

$$\begin{aligned}
\bar{J}^{(\lambda)}(\theta) &= \|\Pi_{\mu_t}(\Phi\theta - \mathcal{T}_\mu^{(\lambda)}(\Phi\theta))\|_{\xi_t}^2\\
&= (\Phi\theta - \mathcal{T}_\mu^{(\lambda)}(\Phi\theta))^\top \Pi_{\mu_t}^\top \Xi_t \Pi_{\mu_t}(\Phi\theta - \mathcal{T}_\mu^{(\lambda)}(\Phi\theta))\\
&= \|\hat{\Pi}_\mu \sqrt{\Xi_t}(\Phi\theta - \mathcal{T}_\mu^{(\lambda)}(\Phi\theta))\|_2^2\\
&= \|\hat{\Pi}_\mu \sqrt{\varrho\Xi_b}(\Phi\theta - \mathcal{T}_\mu^{(\lambda)}(\Phi\theta))\|_2^2,
\end{aligned} \tag{5.32}$$

**Algorithm 5.1** Accelerated Regularized off-policy GTD($\lambda$)

**Input:** Parameters $L_0 > L$ and $\nu \in [0, 1]$ for the optimization, $\eta$ to control the regularization, $\kappa > 0$ to derive the secondary weights $\beta_k$, $0 \leq \lambda \leq 1$ to control the multistep behavior, discount factor $\gamma$

**Initialize:** Given an arbitrary guess $y_0 \in \mathbb{R}^l$, set $z_0 = y_0$, $u_0 = 0$, $e_0 = 0$ and $w_0 = 0$ respectively and $k = 1$.

**repeat**

Update $L_k = (\nu\sqrt{k-1} + 1)L_0$

Compute $\theta_k = (1 - \nu)y_{k-1} + \nu z_{k-1}$

Draw a sample $(\phi_k, r_k, \phi_{k+1})$

Set step-sizes $\alpha_k = \frac{1}{L_k}$ and $\beta_k = \kappa\alpha_k$

Calculate off-policy weight $\rho_k = \frac{\mu(a_k|s_k)}{\mu_b a_k|s_k}$

Update e-trace $e_k = \rho_k(\lambda\gamma e_{k-1} + \phi_k)$

Compute a stochastic update of the respective gradient:

$$y_k = \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k\left(\phi_k\phi_k^\top - \gamma\rho_k(1-\lambda)(\phi_{k+1}^\top e_k)\right)w_k\right) \quad \text{(GTD2a}(\lambda)\text{-IST)}$$

$$y_k = \Psi_{\alpha_k \eta}\left(\theta_k + \alpha_k\delta(\theta_k, \theta_k)e_k - \alpha_k\gamma\rho_k(1-\lambda)(\phi_{k+1})^\top e_k)w_k\right) \quad \text{(TDCa}(\lambda)\text{-IST)}$$

$$w_{k+1} = w_k + \beta_k\left(\delta(\theta_k, \theta_k)e_k - \phi_k(\phi_k^\top w_k)\right)$$

SAGE Update $z_k = z_{k-1} - \frac{\nu}{L_k}(L_k(\theta_k - y_k))$.

**until** converged

**Output:** weight vector $\theta = y_k$.

where $\xi$ (and corresponding $\Xi$) are the steady state distribution probabilities according to the target policy $\mu$ and $\xi_b$ (and $\Xi_b$) correspond to the behavior policy $\mu_b$ and transition probabilities defined in Equation (5.23). Now with off-policy sampling, we can evaluate the equations according to the target policy, but have to keep in mind, that the steady state distributions $\xi_t$ and $\Xi_t$ are actually obtained by sampling according to the behavior policy and then are multiplied by a diagonal matrix containing the off-policy importance weighting factors $\rho$ and $\varrho = \text{diag}(\rho)$ (a diagonal matrix containing the elements of $\rho$).

With that in mind, we can write the steady state distributions as $\xi_t = \xi = \varrho\xi_b$ and $\Xi_t = \Xi = \varrho\Xi_b$. Now $\hat{\Pi}_\mu = \sqrt{\varrho\Xi_b}\Phi(\Phi^\top\varrho\Xi_b\Phi)^{-1}\Phi^\top\sqrt{\varrho\Xi_b}$ is an orthogonal projection on the column span of $\sqrt{\varrho\Xi_b}\Phi$.

In analogy to Chapter 4, we can derive a similar Hessian $\bar{\mathcal{H}}_J(\theta)$ of the off-policy multistep

cost function $\bar{J}^{(\lambda)}(\theta)$ as

$$\begin{aligned}
\frac{1}{2}\nabla_\theta \bar{J}^{(\lambda)}(\theta) &= \left(\Phi^\top \varrho \Xi_b \Phi - \gamma \Phi^\top P_\mu^{(\lambda)\top} \varrho \Xi_b \Phi\right) \left(\Phi^\top \varrho \Xi_b \Phi\right)^{-1} \\
&\qquad \left(\Phi^\top \varrho \Xi_b \Phi\theta - \Phi^\top \varrho \Xi_b R_\mu^{(\lambda)} - \gamma \Phi^\top \varrho \Xi_b P_\mu^{(\lambda)} \Phi\theta\right) \\
\frac{1}{2}\nabla_\theta^2 \bar{J}^{(\lambda)}(\theta) &= \left(\Phi^\top \varrho \Xi_b \Phi - \gamma \Phi^\top P_\mu^{(\lambda)\top} \varrho \Xi_b \Phi\right) \left(\Phi^\top \varrho \Xi_b \Phi\right)^{-1} \left(\Phi^\top \varrho \Xi_b \Phi - \gamma \Phi^\top \varrho \Xi_b P_\mu^{(\lambda)} \Phi\right) \\
&= \Phi^\top \left(I - \gamma P_\mu^{(\lambda)}\right)^\top \sqrt{\varrho \Xi_b} \hat{\Pi}_\mu \sqrt{\varrho \Xi_b} \left(I - \gamma P_\mu^{(\lambda)}\right) \Phi \\
&= \Phi^\top \left(I - \gamma P_\mu^{(\lambda)}\right)^\top \Pi_\mu^\top \varrho \Xi_b \Pi_\mu \left(I - \gamma P_\mu^{(\lambda)}\right) \Phi \\
&= \bar{A}^\top \bar{A},
\end{aligned} \tag{5.33}$$

with $\bar{A} := \sqrt{\varrho \Xi_b} \Pi_\mu (I - \gamma P_\mu^{(\lambda)}) \Phi$.

**Lemma 7** (Strong convexity of Multistep Off-Policy MSPBE)**.** *Assume that the feature matrix $\Phi \in \mathbb{R}^{n \times l}$ is full rank and the Markov chains defined by $P_{\mu_b} \in \mathbb{R}^{n \times n}$ and $P_\mu \in \mathbb{R}^{n \times n}$ are both irreducible and aperiodic. Furthermore, there exists a unique limiting distribution $\xi_b \in \mathbb{R}^n$, which satisfies $P_{\mu_b} \xi_b = \xi_b$ with $\xi_{b,i} > 0, \forall i \in 1 \dots n$ and some positive definite diagonal re-weighting matrix $\varrho$, then the MSPBE $\bar{J}(\theta)$ is strongly convex.*

*Proof.* The proof follows exactly the same structure as the proof for Lemma 1, but with some slightly modified ingredients. What lies to show for the proof to hold is that the quantity $(I - \gamma P_\mu^{(\lambda)})$ is full rank: From the definition of $P_\mu^{(\lambda)} = (1 - \lambda) \sum_{l=0}^\infty \gamma^l \lambda^l P_\mu^{l+1}$, we can see that this is a product of matrices, that are full rank, since we assumed that also the Markov chain defined by following the target policy is irreducible an aperiodic, and therefore the product is also full rank. From there the argument of the referenced Lemma 1 can be followed. ■

**Remark 2.** *A similar inequality as for Lemma 2 can be derived for the averaged Bellman operator models, defined by $P^{(\lambda)} \in \mathbb{R}^{n \times n}$ as in*

$$\|P^{(\lambda)} x\|_{\xi_t} \leq (1 - \lambda) \sum_{l=0}^\infty \gamma^l \lambda^l \|P^{l+1} x\|_{\xi_t} \leq (1 - \lambda) \sum_{l=0}^\infty \gamma^l \lambda^l \|x\|_{\xi_t} = \frac{1 - \lambda}{1 - \gamma\lambda} \|x\|_{\xi_t}, \tag{5.34}$$

*where $x \in \mathbb{R}^n$, cf. [5].*

**Lemma 8** ($\mu$-strong Convexity of Multistep Off-Policy MSPBE)**.** *Let the feature matrix $\Phi \in \mathbb{R}^{n\times l}$ be full rank and the Markov chain defined by $P_{\mu_b} \in \mathbb{R}^{n\times n}$ be irreducible and aperiodic. Furthermore, there exists a unique limiting distribution $\xi_b \in \mathbb{R}^n$, which satisfies $P_{\mu_b}\xi_b = \xi_b$ with $\xi_{b,i} > 0, \forall i \in 1 \dots n$. Define $C_b = \Phi^\top \varrho\Xi_b\Phi$, where $\varrho\Xi_b = diag(\rho\xi_b)$ is the matrix with the elements $\rho_i\xi_{b,i}$ on the diagonal and let $\lambda_{\min,b}$ be the minimal eigenvalue of $C_b$. then the MSPBE $\bar{J}(\theta)$ is $\mu$-strong convex with*

$$\mu > \left(1 - \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \lambda_{\min,b}. \tag{5.35}$$

*Proof.* The proof will again follow closely along the proof for Lemma 3. We have

$$\begin{aligned} x^\top \bar{\mathcal{H}_J}(\theta)x &= x^\top \bar{A}^\top \bar{A}x \\ \text{(derivation analogous to proof of Lemma 3)} \\ &\geq \left(\|\Phi x\|_{\xi_t} - \gamma\|P_\mu^{(\lambda)}\Phi x\|_{\xi_t}\right))^2 \\ &\geq \left(\|\Phi x\|_{\xi_t} - \gamma\frac{1-\lambda}{1-\gamma\lambda}\|\Phi x\|_{\xi_t}\right)^2 \\ &= \left(1 - \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \|\Phi x\|_{\xi_t}^2. \end{aligned} \tag{5.36}$$

Additionally, we have – as in the referenced proof – for $C_b = \Phi^\top \varrho\Xi_b\Phi$

$$\|\Phi x\|_{\xi_t} = x^\top \Phi^\top \varrho\Xi_b\Phi x \geq \lambda_{\min,b}\|x\|_2^2. \tag{5.37}$$

and can therefore conclude that $\bar{\mathcal{H}_J}(\theta) \succeq \mu I$ with

$$\mu \geq \left(1 - \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \lambda_{\min,b}. \tag{5.38}$$

■

Although we could also show that this cost function is $\mu$-strongly convex, the same precaution holds as for the non-multistep version in that the smallest eigenvalue is still unbounded from below for an arbitrary feature matrix $\Phi$.

**Lemma 9** (L-Lipschitz Multistep Off-Policy MSPBE Gradient)**.** *Let the feature matrix $\Phi \in \mathbb{R}^{n\times l}$ be full rank and the Markov chain defined by $P_{\mu_b} \in \mathbb{R}^{n\times n}$ be irreducible and*

*aperiodic. Furthermore, there exists a unique limiting distribution* $\xi_b \in \mathbb{R}^n$*, which satisfies* $P_{\mu_b}\xi_b = \xi_b$ *with* $\xi_{b,i} > 0, \forall i \in 1 \dots n$*. Define* $C_b = \Phi^\top \varrho\Xi_b\Phi$*, where* $\varrho\Xi_b = diag(\rho\xi_b)$ *is the matrix with the elements* $\rho_i\xi_{b,i}$ *on the diagonal and let* $\lambda_{max,b}$ *be the largest eigenvalue of* $C_b$*. Then the gradient of the MSPBE* $\bar{J}(\theta)$ *is L-Lipschitz with*

$$L \leq \left(1 + \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \max_j \|\phi\|_2^2, \tag{5.39}$$

*or equivalently*

$$L \leq \left(1 + \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \lambda_{max}. \tag{5.40}$$

*Proof.* The proof closely follows the proof for Lemma 4. We state that the cost function $\bar{J}(\theta)$ is L-Lipschitz if the inequality

$$x^\top \bar{\mathcal{H}}_J(\theta)x \leq L\|x\|_2^2 \tag{5.41}$$

holds for all $x \in \mathbb{R}^n$ since $\bar{J}(\theta)$ is strongly convex. As in the referenced proof, we can write

$$\begin{aligned} x^\top \bar{\mathcal{H}}_J(\theta)x &= x^\top \bar{A}^\top \bar{A}x \\ &\leq \left(\|\Phi x\|_{\xi_t} + \gamma\|P_\mu^{(\lambda)}\Phi x\|_{\xi_t}\right)^2 \\ &\leq \left(\|\Phi x\|_{\xi_t} + \gamma\frac{1-\lambda}{1-\gamma\lambda}\|\Phi x\|_{\xi_t}\right)^2 \\ &= \left(1 + \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \|\Phi x\|_{\xi_t}^2. \end{aligned} \tag{5.42}$$

With the same argument as from the proof of Lemma 4, we can now conclude that

$$L \leq \left(1 + \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \max_j \|\phi\|_2^2, \tag{5.43}$$

or alternatively in terms of the greatest eigenvalue of $C_b$

$$L \leq \left(1 + \gamma\frac{1-\lambda}{1-\gamma\lambda}\right)^2 \lambda_{max}. \tag{5.44}$$

■

For the combined accelerated and regularized algorithms, the same arguments as in Sec-

tion 4.5 apply. We again have a combination of two L-Lipschitz continuous functions and therefore Nesterov's method is again applicable.

## 5.5. Experiments and Results

In addition to the previously introduced experiment environments, in this chapter, two new environments will be presented to highlight the specific behavior of the algorithms with respect to multistep learning and off-policy sampling. The first is the so called *Random Walk Chain*, which highlights the use of a multistep Bellman operator and the second one is the *Baird Star*, which is designed to test off-policy methods.

### Random Walk Chain

The *random walk chain* [77] environment consists of $n = 7$ nodes aligned in a one-dimensional chain arrangement. The agent can choose to go *left* or *right* in each state, except the outermost two states. Upon transitioning the agent receives a reward of $r_k = 0$, except for the rightmost state, where the agent receives a reward of $r_7 = 1$, see Figure 5.1. The leftmost and rightmost states are terminal states. This means, when the agent transitions in one of those two states, the episode ends and the environment is restarted. The starting state for the agent is always the center state. Usually, the discounting factor is chosen to be $\gamma = 0.95$ if not indicated otherwise and the policy for transitioning through this environment is chosen to be random. In each state the agent chooses with equal probability $p = 0.5$ to either go left or right as indicated in Figure 5.1. This environment is especially

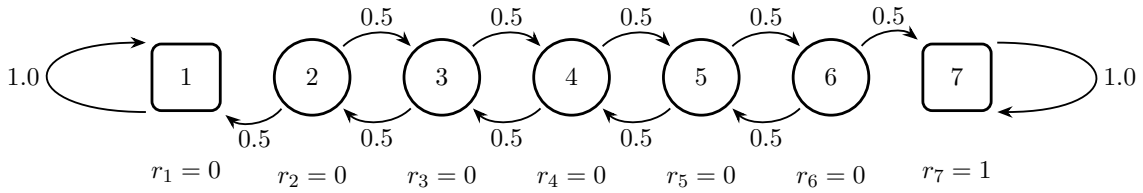



**Figure 5.1.:** Illustration of the 7-state random walk environment.

well suited to study the behavior of the different algorithms with respect to favorable and unfavorable feature generation. The most straightforward way to generate the features for this environment is to use indicator vectors. This is done by defining the feature dimension

to be the same as the number of states in the chain $l = n$ and then placing a singular 1 in position in the vector according to the state numbering. For example if the environment is in state 2, the second element of the vector is set to 1 and 0 for all others. These are very favorable features as exact value function representation is possible and the features are linearly independent and therefore the feature matrix $\Phi$ has full rank. This way of feature generation is most of the times referred to as *tabular features* as all the states get enumerated and the weight vector acts as a table of values for each individual states. The second way to derive features for this environment are the so-called *inverted features*, where the coding is done in the same way as for the tabular features with the difference of a 0 in the position of the state and 1s for all other positions. The features are then normalized to be of length 1. This type of feature generation aims at demonstrating the resilience of the algorithms towards inappropriate generalization. This means that it might be possible that some value of a particular state could be shared among several states in an inappropriate way since the information is encoded in the ones of the feature vector. A third way to generate features is to use so called *dependent features*. Here, the feature dimension is smaller than the number of states $k < n$ and the value function can only be approximated, cf. [78]. The states are then encoded in a way similar to binary numbers. Afterwards, the feature vectors are again normalized to length 1. As an example, a chain with $n = 5$ states has the feature vectors $\phi_1 = (1, 0, 0)^\top$, $\phi_2 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^\top$, $\phi_3 = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})^\top$, $\phi_4 = (0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^\top$ and $\phi_5 = (0, 0, 1)^\top$.

It is typical for this environment to perform well for a selected value of $\lambda$ depending on the number of states in the chain. This fact can be explained by looking at the structure: As the agent can only move to the left or the right and only in the rightmost state, the positive reward is given, by supplying a kind of lookahead (more or less equivalent to choosing a $\lambda > 0$) the agent can better decide if it is beneficial to choose the left or right action. Depending on the length of the chain, a certain number of lookahead steps, starting from the center, is necessary to come to the correct decision. If a number too small is chosen, the learning will converge slower, because more sampling has to be performed, if the number is too high, again more sampling has to be performed as the reward might be discounted in an ill posed fashion.

### Baird Star

The *Baird star* [1] environment was designed to demonstrate convergence issues with temporal difference based algorithms. Although the environment is based on linear indicator vectors, standard TD learning will diverge in this example. It consists of $n = 7$ states and $n_a = 2$ actions in every state. The first action will bring the agent to the state $s_1$, also referred to the *center state*, which is *absorbing*, i.e. the second action cannot be chosen and the first action will always bring the agent back to the center. The second action will bring the agent uniformly randomly into any of the other 6 states. No matter what action was chosen, the reward will be always $r_s = 0$. The discounting factor is set to $\gamma = 0.99$. All states are represented by a tabular feature. Additionally, the feature vector contains a constant bias feature, which is set to 1 for every state, except for the center state, for which the bias is set to 2. Associated with this environment is a specific set of two policies. The

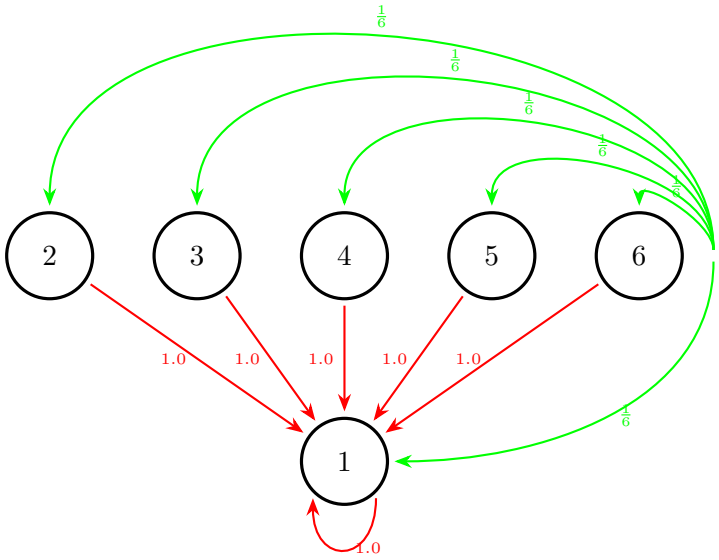



**Figure 5.2.:** Baird star, adapted from [1].

first policy, the *behavior policy*, will choose with probability $\frac{1}{7}$ the first action, which will bring the agent to the center, and with probability $\frac{6}{7}$ the second action. The value function is then to be calculated with respect to the *target policy*, which will choose with probability 1 the second action and never the first action.

### On- and Off-Policy Sampling

In general every environment (except the Baird star, which is off-policy by definition), can be converted in an off-policy learning problem. This is done by sampling with a distribution different from the control policy. The random MDP can be converted to an off-policy problem by randomly generating a second policy and use this second policy for sampling. The random walk and Boyan chain example are converted to off-policy by uniformly sampling the states, which is different to the behavior policy. Finally, the inverted pendulum can be sampled off-policy by using the same policy for which the value function is to be determined, but sample with a behavior policy with a much more increased variance parameter $\sigma$.

### Ergodicity of Studied Domains

A MDP is considered ergodic if all states can eventually be reached from all other states given enough time. In other words, the total state transition probability given any action, that will be chosen eventually, of getting from one state to any other is strictly positive. In a formal definition, we have the steady state distribution vector $\xi \in [0,1]^n$ with $n = |\mathcal{S}|$ being the number of states and every entry $\xi_i$ giving the probability of ending up in state $i$ after following the state transition probabilities given in $P_\mu$. Since $P_\mu$ is a Markov matrix, the state transition probabilities can be obtained by calculating

$$\xi = \operatorname{diag}\left(\lim_{k=0}^{\infty} P_\mu^k\right), \tag{5.45}$$

where diag takes the diagonal entries. As mentioned in the section before, the state transition probability matrix has to take the current policy $\mu$ into account and therefore it is often easier to directly sample the environment following the policy a sufficient amount of time in order to get a good enough approximation of the steady state distribution.

**Definition 7** (Ergodicity)**.** *The MDP characterized by $P_\mu$ is ergodic iff $\forall i,\ \xi_i > 0$.*

If we now direct our attention on which of the experimental domains are ergodic, it is clear, that the randomly generated MDPs are ergodic by construction and the inverted pendulum is able to be moved to any cart and pendulum position. Baird star and random walk environments are not ergodic simply due to the fact, that they have so called *absorbing states*. This means that the only outgoing edges from such a state loop back to itself.

Therefore, no other state can be reached starting in such a state. For the state transition probabilities this means that all the mass will be concentrated in that state because at some point for sure the Markov process will end up reaching that state and not being able to escape it. On the other hand this means that at least one other state has a corresponding 0 in $\xi$.

A non ergodic environment means that the MSPBE cost function is no longer a proper $\xi$-weighted $\ell_2$-norm, but only a semi-norm. This can impact the convergence properties of gradient algorithms in a negative way.

One way to remedy this situation is to allow some small probability for every state to reach all other states in the modeling phase. Another way that is often followed and applied in the experiments for the random walk and Baird star environment, is to declare the environment as episodic and terminate the sampling as soon as one of the absorbing states is reached. The subsequent sampling is then restarted in some other non-absorbing state. This is equivalent for the steady state distribution of having unique transition probabilities from the absorbing state to the set of starting states.

## 5.6. Results

In this section the results of experiments shall be presented to highlight the algorithms performances with respect to two aspects. The first is the multistep behavior of the algorithm, which directly reflects in the dependence of the chosen $\lambda$ parameter value. This behavior will be highlighted using the random walk environment. The second is the performance when off-policy sampling is done during learning. To illustrate this, first the results for the discrete and episodic Baird star, a classical environment for demonstrating off-policy performance, are presented. Afterwards, off-policy results in a continuous domain are investigated.

### 5.6.1. Random Walk Environment

The first experiment to investigate is the random walk domain while sampling on-policy to look at the behavior of the algorithms with respect to multistep sampling. As noted above, every domain should have an optimal value of $\lambda$ for which algorithm performance is optimal. This value unfortunately cannot be determined in a deterministic manner, but has to be evaluated experimentally, depending on the type of domain the algorithm is applied to.

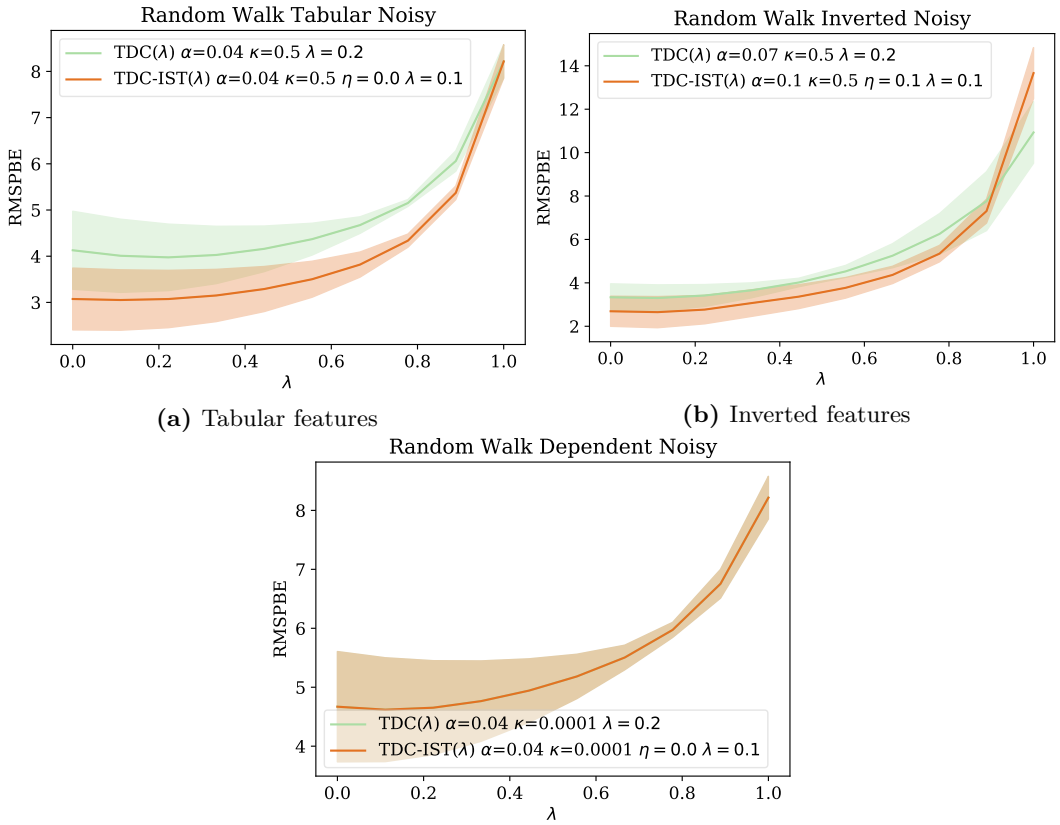
When taking a look at Figure 5.3, the different performance depending on the selected types of features can be seen. On the y-axis, the square root of Bellman error is depicted, on the y-axis the different parameter values for $\lambda$ are denoted. In Figure 5.3a, the ideal feature representation for this domain is depicted. Here the original version of the TDC algorithm and the regularized version both perform well as the true value function can be perfectly represented with this type of features. The thresholding modification helps in obtaining a lower error as this environmental setup contains additional noisy features that degrade the performance of non-regularized algorithms. Similar can be observed in Figure 5.3b the setting with inverted features, where the generalization power of the algorithm, i.e. the ability to prevent putting weight on features that do not belong to the state, is investigated. Also here thresholding helps minimizing weights on noisy features.

In the setting with dependent feature in Figure 5.3c, where the value function no longer can be perfectly approximated, both algorithm performances seem to degrade in a similar way. The variance of results over 20 independent runs increases and the influence of thresholding to prevent noisy features is smaller than the disturbance introduced by the non-perfect representation of the ground truth value function.

The second interesting property of the algorithms to be studied in this experiment is the value of $\lambda$. As said in the description of the experiment above, the random walk chain has a distinct optimal value for $\lambda$. This can also be observed here. For the original, unmodified algorithms the optimal value for lambda lies around $\lambda = 0.2$, whereas for the thresholded versions the optimal value seems to be $\lambda = 0.1$. This difference stems from the complex interplay of the many parameters the algorithms have. Since the thresholded algorithms have their optimal performance for a different set of parameters also for $\alpha$ and $\kappa$, the optimal value for $\lambda$ also changes. In general, unfortunately, no definitive recipe can be given for choosing the optimal $\lambda$ but this has to be determined individually depending on the algorithm, environment and other parameters.

### 5.6.2. Off-Policy Results

In this subsection the off-policy performance of the algorithms is investigated. The purpose of this is twofold: First to ensure that the algorithms do converge in off-policy settings at all, as for example the well known TD algorithm fails to do so and second, to investigate if the regularization is still useful in this sampling setting.




**(a)** Tabular features

**(b)** Inverted features

**(c)** Dependent features

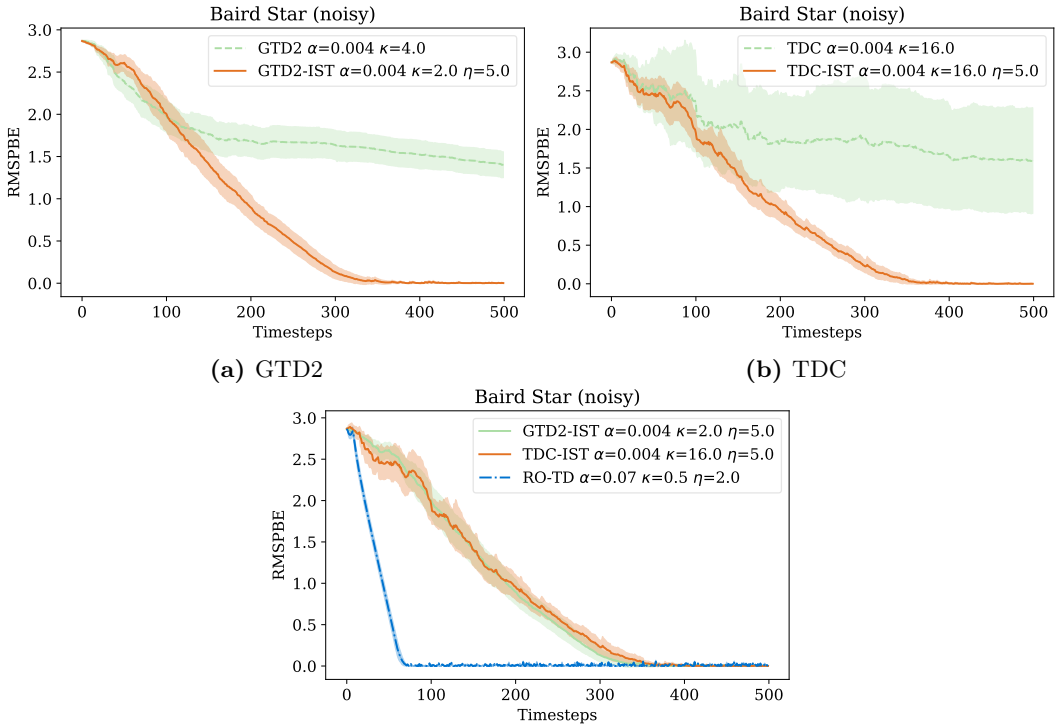
**Figure 5.3.:** Comparison of parameter sensitivity with respect to $\lambda$ for TDC and TDC-IST

### Baird Star

A classical example to demonstrate the off-policy behavior of reinforcement learning algorithms is the Baird star. This is also the reason this environment was chosen to demonstrate the basic convergence properties of the regularized gradient TD algorithms.

In Figure 5.4 the convergence performance of the thresholded algorithms can be observed. Each subfigure depicts the Bellman error on the y-axis and the number of sampling steps on the x-axis. In Figures 5.4a and 5.4b, the performance of the GTD2 and TDC algorithm variants can be observed. A clear advantage of regularization is present, since the iterative soft thresholded algorithm modifications obtain a much lower error within the same

number of iterations. Also when looking at the variance across the independent runs (the shaded areas in the curves), it can be seen that this is greatly reduced and the original algorithm versions struggle to cope with the additional introduced noise features. This effect if especially visible for TDC.




**(a)** GTD2 **(b)** TDC

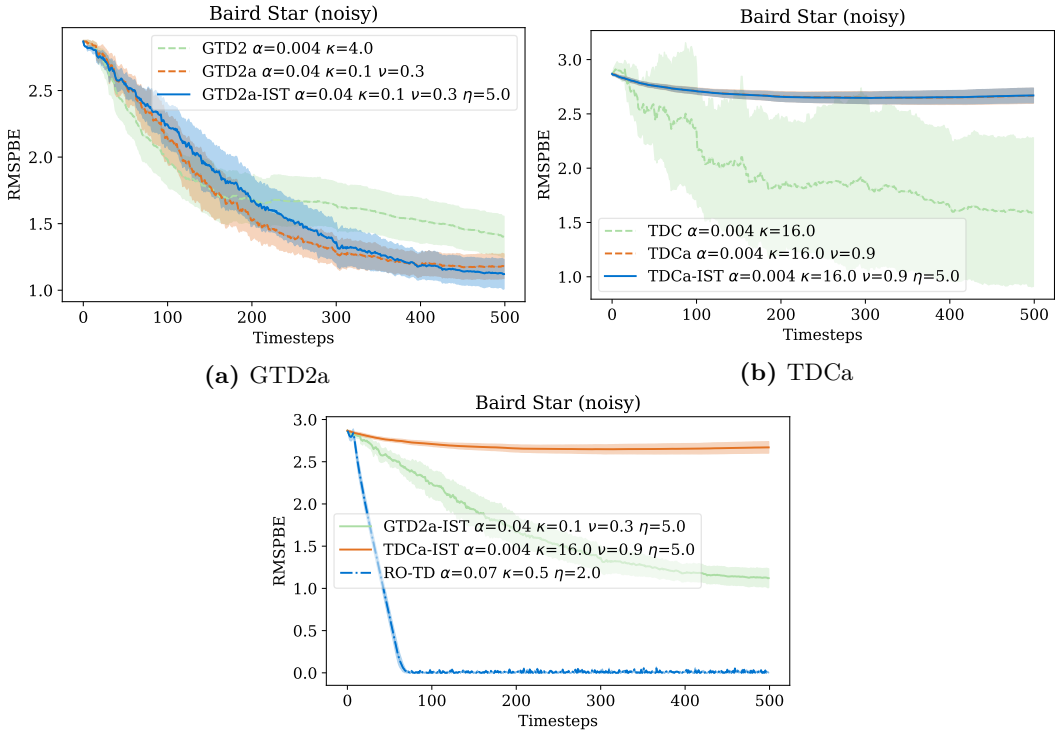
**(c)** Comparison to RO-TD

**Figure 5.4.:** Off-policy performance on the Baird star example.

It is observed in Figure 5.4c, the RO-TD algorithm performs especially well in this environment. It reaches a low error in approximately $\frac{1}{3}$ of samples compared to the thresholded algorithm variants.

The situation for the accelerated modifications of the gradient TD algorithms presents itself a little bit different. When observing in Figure 5.5a, the accelerated version and the accelerated and soft-thresholded version of the GTD2 algorithm still present some advantage over the original version of the algorithm, although not as pronounced as with the non-accelerated but thresholded algorithm version. On the other hand for the TDC algorithm,

Figure 5.5b, acceleration in the presence of noise for the Baird star example seems to degrade algorithm performance significantly in terms of minimal error attained. On the other hand, the variance across different independent runs is reduced by a relatively large amount.




**(a)** GTD2a **(b)** TDCa

**(c)** Comparison to RO-TD

**Figure 5.5.:** Off-policy performance of accelerated algorithms on the Baird star example.

This behavior can be explained with two arguments: First, the TDC algorithm was found to be working the best with the parameter set containing the acceleration parameter $\nu = 0.9$ this is comparatively large when looking at the other experiments. While a large acceleration parameter can speed up gradient performance, it also increases instability. Therefore, the tradeoff done here in the gridsearch – only a finite set of parameters could be tested – was to select a larger acceleration parameter while keeping the other parameters, such as step size $\alpha$ smaller to still have stable convergence. In the presence of noise, like it is the case here, then this instability of a large acceleration parameter gets amplified at the cost of having to choose smaller step size parameters. The second argument to always keep in mind, when

interpreting the results is, that the optimal parameter set, where the accelerated algorithm might have outperformed the unmodified algorithm version might have not been in the parameter space, that was tested during the gridsearch.

Again, also for the accelerated algorithms, RO-TD, as seen in Figure 5.5c outperforms the gradient TD algorithms.

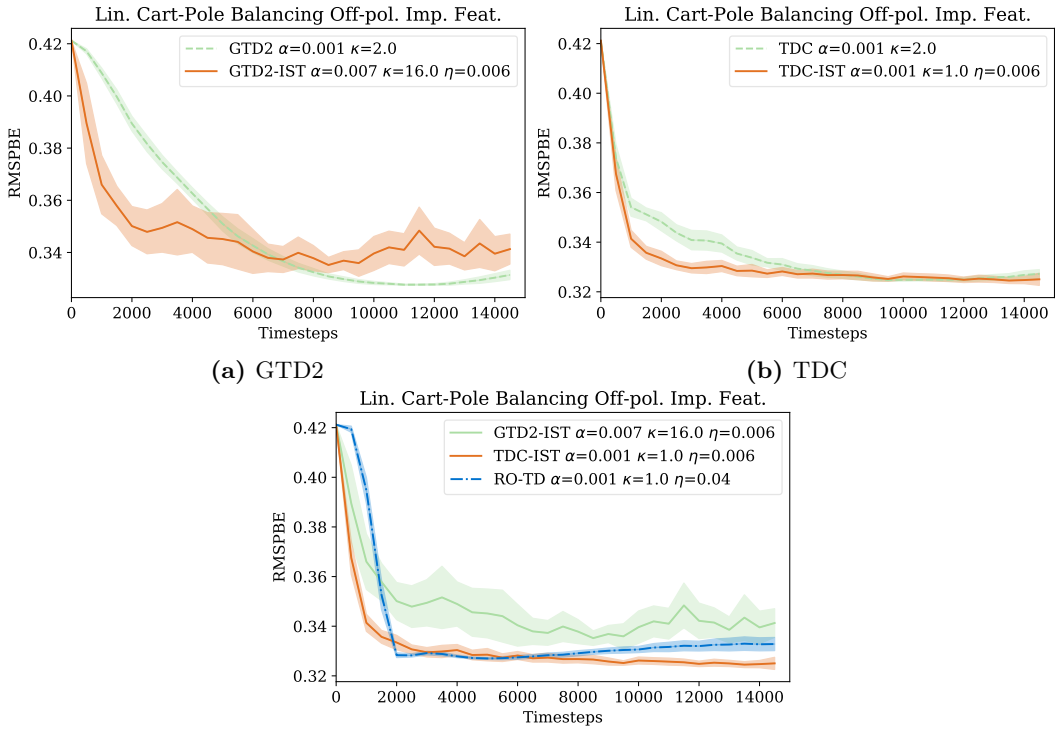
### Continuous Domain

As in the previous chapters, it is important that the algorithms not only perform well on discrete state environments, but also on continuous state environments, where the state space has to be approximated. Here an exact representation of the ground truth value function is no longer possible and the algorithms are challenged to adapt to generalization problems. This means, that weights could contribute to more than one environment state and a change in weights not only affects performance in a small confined area of the state space, but could change the overall behavior.

In Figure 5.6 the performance of the thresholded algorithms is depicted. On the x-axis again the number of samples and on the y-axis the Bellman error is plotted. For GTD2 in Figure 5.6a, it can be observed that the increase in error is accelerated at some cost of increased variance across the independent runs and a slightly higher final error attained. This can be explained by the parameters chosen by gridsearch. Faster convergence and a lower average final error was traded for an increase in variance of the results across runs.

For TDC in Figure 5.6b the situation presents itself more favorable for the regularized algorithm. Not only the convergence is quicker in terms of samples needed, but also the variance across runs is decreased. Similar can be said for RO-TD in Figure 5.6c, where the algorithm converges almost as fast as TDC-IST with a lower variance and reaches a low final average error.

Results for the accelerated versions of the algorithm can be seen in Figure 5.7. Results compare to results for the off-policy Baird star example. GTD2 presents to be working quite well for this environment and acceleration as well as thresholding increase algorithm performance. TDC in Figure 5.7b can gain convergence speed with a small acceleration parameter $\nu$ and relatively similar parameters for $\alpha$ and $\kappa$, while finding optimal parameters for the accelerated and regularized version TDCa-IST poses a difficulty. The hypothesis is




**(a)** GTD2 **(b)** TDC

**(c)** Comparison to RO-TD

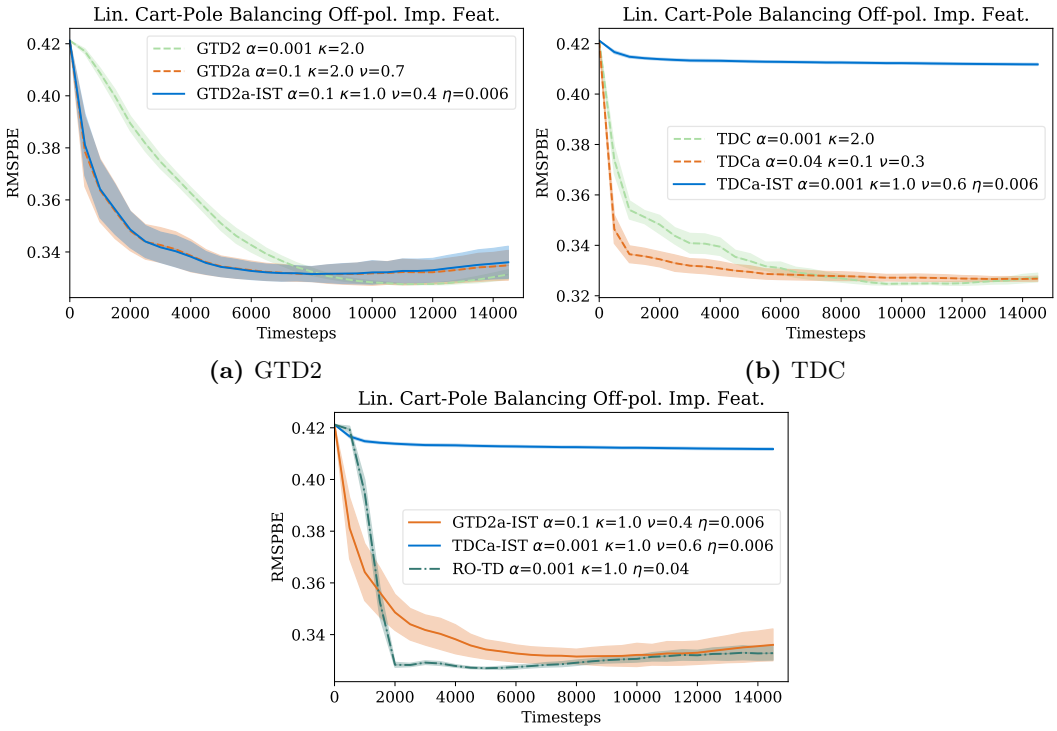
**Figure 5.6.:** Off-policy performance on the continuous linear cart pole balance task using the soft-thresholded algorithms.

again, that this is due to the limited number of parameters that could be tested during the gridsearch due to computational limitations.

The same situation presents itself for RO-TD in Figure 5.7c where RO-TD outperforms all other algorithms as it has already also in the Baird star environment. This suggests, that off-policy performance of RO-TD is especially favorable.

## 5.7. Summary

In this section extensions of the gradient TD algorithms to off-policy and multistep versions have been presented. It was established that the cost function of the off-policy and multistep




**(a)** GTD2 **(b)** TDC

**(c)** Comparison to RO-TD

**Figure 5.7.:** Off-policy performance on the continuous linear cart pole balance task using the accelerated and thresholded algorithms.

formulation of the Bellman error has the same theoretical guarantees that allows to apply Nesterov's acceleration scheme.

Experiments were conducted to empirically show the convergence performance of the modified algorithms as well as to compare to the closest existing algorithm formulation, RO-TD. It is seen, that all gradient TD algorithm also work in off-policy and multistep settings and expose the expected behavior in terms of the $\lambda$ parameter. Difficulties in finding optimal parameters for the modified TDC formulation of the algorithm have been surfaced and it was established, that in off-policy settings the RO-TD formulation of optimization is especially well suited and exposes stable and quick convergence behavior.

# Chapter 6.

# Conclusion

In this thesis, I introduced two new algorithm extensions to gradient temporal difference learning. The first is a simple, yet effective regularization scheme, while the latter combines the accelerated gradient optimization with the original and the regularized stochastic gradient algorithm.

Since the cost function variant usually is chosen because of algorithmic properties, I added an additional justification for projection when working with linear approximated temporal difference learning. It is the benefit of better upper bounds in error, when working with discounting factors close to one which can happen if the environment cannot be fully controlled.

Hereafter, I derive the regularized gradient temporal difference algorithm, which has the benefit of being very easy to implement and at the same time still maintains the linear computational complexity in terms of the feature size. It performs very well in reaching low final error measures in noisy environments when the unmodified algorithms fail to converge at all. Also it helps in speeding up the convergence up to a factor of four in terms of samples needed. The additional regularization parameter can be chosen to a small value without too much additional parameter tuning effort. When interpreting the $\ell_1$-regularization as a feature selection scheme, the algorithms can achieve a attenuation of features that only contribute noise to the learning. However, this selection is not as hard as could be expected in the original sense and only a soft attenuation is reached.

Since the stochastic gradient descent algorithms, as this is one instance of such, suffer from slower convergence in general, in the next part of the thesis I combine the accelerated gradient descent technique with the linear temporal difference learning.

As a prerequisite for the application of the accelerated gradient algorithm, the strong

convexity and L-Lipschitz continuity of the underlying cost function had to be checked in order to safely apply the algorithm. I prove the strong convexity and give an upper bound on the Lipschitz factor derived from the norm of the features, which is also required as a parameter to the optimization procedure. Then, additionally, I combine the accelerated algorithm with the previous regularization scheme.

The accelerated algorithms can improve the convergence speed in terms of samples needed in certain test scenarios, but also can fail to achieve acceleration in others. Here the noise robustness and the acceleration play against each other in noisy settings. Additionally, in some settings, it can be challenging to determine good parameters for fast and proper convergence. The viable range of parameters for the step size is smaller and more sensitive to deviation from the optimum.

In the last part of the thesis, I extend all algorithms to an off-policy sampling setting, which is crucial to have in real-world applications. The algorithms behave in a similar way as with on-policy sampling and failure to converge is no issue here. Finally, I extend the algorithms to multistep sampling, which can utilize multiple discounted samples for one update. In a lot of scenarios this helps with algorithm convergence and is therefore also a required feature.

Finally, I evaluated all algorithms and their specific behavior in specially tailored extensive simulation environments, discrete and continuous, and report the results in each.

Overall, in this thesis I present modified stochastic gradient temporal difference learning algorithms, that are still linear in computational complexity with respect to the dimension of features, that can be used when a significant amount of additive Gaussian sampling noise is present. The algorithms allow learning in computationally restricted environments, such as embedded systems, under real world conditions where non-ideal sensor measurements are used. The algorithms allow to reach the same results as the unmodified variants with four times less samples on average and still enable learning when up to five times the number of noisy measurement features than information bearing features are present.

# Bibliography


1. L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th international conference on machine learning*, pp. 30–37. ACM, 1995.

2. B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, M. Bob, and I. Mordatch. Emergent tool use from multi-agent autocurricula. In *arXiv preprint: 1909.07528*, 2019.

3. E. Barnard. Temporal-difference methods and markov models. In *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2), pp. 357–365, 1993.

4. R. Bellman. A markovian decision process. In *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

5. D.P. Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena scientific Belmont, MA, 1995.

6. D.P. Bertsekas. *Nonlinear Programming*. Athena scientific Belmont, MA, 1999.

7. D.P. Bertsekas, V.S. Borkar, and A. Nedich. Improved temporal difference methods with linear function approximation. In *Handbook of Learning and Approximate Dynamic Programming*, pp. 233–259. John Wiley & Sons, Ltd, 2012.

8. D.P. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena scientific Belmont, MA, 1996.

9. L. Bottou. Online algorithms and stochastic approximations. In D. Saad (ed.), *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

10. L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, pp. 161–168. 2008.

11. L. Bottou and Y.L. Cun. Large scale online learning. In *Advances in neural information processing systems*, pp. 217–224. 2004.

12. J.A. Boyan. Technical update: Least-squares temporal difference learning. In *Machine Learning*, 49(2), pp. 233–246, 2002.

13. S.J. Bradke and A. Barto. Linear least-squares algorithms for temporal difference learning. In *Machine Learning*, 22, pp. 33–57, 1996.

14. S. Bubeck. Convex optimization: Algorithms and complexity. In *Foundations and Trends in Machine Learning*, 8(3-4), pp. 231–357, 2015.

15. E. Candes and T. Tao. The dantzig selector: Statistical estimation when p is much larger than n. In *The Annals of Statistics*, pp. 2313–2351, 2007.

16. P.L. Combettes and V.R. Wajs. Signal recovery by proximal forward-backward splitting. In *Multiscale Modeling & Simulation*, 4(4), pp. 1168–1200, 2005.

17. C. Dann. *Algorithms for Fast Gradient Temporal Difference Learning.* Technical Report, TU Darmstadt, 2012.

18. C. Dann, G. Neumann, and J. Peters. Policy evaluation with temporal differences: A survey and comparison. In *Journal of Machine Learning Research*, 15, pp. 809–883, 2014.

19. I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. In *Communications on pure and applied mathematics*, 57, pp. 1416–1457, 2004.

20. J. Duchi and Y. Singer. Efficient online and batch learning using forward backward splitting. In *Journal of Machine Learning Research*, 10(99), pp. 2899–2934, 2009.

21. B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. In *The Annals of statistics*, 32(2), pp. 407–499, 2004.

22. Y. Engel, S. Mannor, and R. Meir. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *International Conference on Machine Learning*, pp. 154–161. 2003.

23. A.M. Farahmand, M. Ghavamzadeh, S. Mannor, and C. Szepesvári. Regularized policy iteration. In *Advances in Neural Information Processing Systems*, pp. 441–448. 2009.

24. A.M. Farahmand, M. Ghavamzadeh, C. Szepesvári, and S. Mannor. Regularized fitted q-iteration for planning in continuous-space markovian decision problems. In *American Control Conference*, pp. 725–730. 2009.

25. L.E. Frank and J.H. Friedman. A statistical view of some chemometrics regression tools. In *Technometrics*, 35(2), pp. 109–135, 1993.

26. W.J. Fu. Penalized regressions: the bridge versus the lasso. In *Journal of computational and graphical statistics*, 7(3), pp. 397–416, 1998.

27. M. Geist and O. Pietquin. Algorithmic survey of parametric value function approximation. In *Transactions on Neural Networks and Learning Systems*, 24(6), pp. 845–867, 2013.

28. M. Geist and B. Scherrer. l1-penalized projected bellman residual. In *European Workshop on Reinforcement Learning*, pp. 89–101. 2011.

29. M. Geist and B. Scherrer. Off-policy learning with eligibility traces: A survey. In *The Journal of Machine Learning Research*, 15(1), pp. 289–333, 2014.

30. M. Geist, B. Scherrer, A. Lazaric, and M. Ghavamzadeh. A dantzig selector approach to temporal difference learning. In *International Conference on Machine Learning*, pp. 347–354. 2012.

31. A. Geramifard, M. Bowling, and R.S. Sutton. Incremental least-squares temporal difference learning. In *National Conference on Artificial Intelligence*, volume 21, p. 356. 2006.

32. M. Ghavamzadeh, A. Lazaric, R. Munos, and M. Hoffman. Finite-sample analysis of lasso-td. In *International Conference on Machine Learning*, pp. 1177–1184. 2011.

33. M.W. Hoffman, A. Lazaric, M. Ghavamzadeh, and R. Munos. Regularized least squares temporal difference learning with nested l2 and l1 penalization. In *European Workshop on Reinforcement Learning*, pp. 102–114. Springer, 2011.

34. C. Hu, J.T. Kwok, and W. Pan. Accelerated methods for stochastic optimization and online learning. In *Advances in Neural Information Processing Systems*, pp. 781–789. 2009.

35. J. Johns, C. Painter-Wakefield, and R. Parr. Linear complementarity for regularized policy evaluation and improvement. In *Advances in Neural Information Processing Systems*, pp. 1009–1017. 2010.

36. J.T. Johns. *Basis construction and utilization for markov decision processes using graphs*. Ph.D. thesis, University of Massachusetts Amherst, 2010.

37. A. Juditsky and A. Nemirovski. First order methods for nonsmooth convex large-scale optimization, ii: Utilizing problem's structure. In *Optimization for Machine Learning*, pp. 121–148. MIT Press. Cambridge, Massachusetts., 2011.

38. S.J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale $\ell_1$-regularized least squares. In *Journal of selected Topics in Signal Processing*, 1(4), pp. 606–617, 2007.

39. J.Z. Kolter and A.Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning*, pp. 521–528. 2009.

40. M.G. Lagoudakis and R. Parr. Least-squares policy iteration. In *Journal of Machine Learning Research*, 4(Dec), pp. 1107–1149, 2003.

41. J. Langford, L. Li, and T. Zhang. Sparse online learning via truncated gradient. In *Journal of Machine Learning Research*, 10(Mar), pp. 777–801, 2009.

42. S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. In *The Journal of Machine Learning Research*, 17(1), pp. 1334–1373, 2016.

43. B. Liu. *Algorithms for First-order Sparse Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst, 2016.

44. B. Liu, J. Liu, M. Ghavamzadeh, S. Mahadevan, and M. Petrik. Finite-sample analysis of proximal gradient td algorithms. In *Conference on Uncertainty in Artificial Intelligence*, pp. 504–513. 2015.

45. B. Liu, S. Mahadevan, and J. Liu. Regularized off-policy td-learning. In *Advances in Neural Information Processing Systems*, pp. 836–844. 2012.

46. M. Loth, M. Davy, and P. Preux. Sparse temporal difference learning using lasso. In *International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 352–359. 2007.

47. H.R. Maei. *Gradient temporal-difference learning algorithms*. Ph.D. thesis, University of Alberta, 2011.

48. S. Mahadevan, S. Giguere, and N. Jacek. Basis adaptation for sparse nonlinear reinforcement learning. In *Conference on Artificial Intelligence*. 2013.

49. S. Mahadevan and B. Liu. Sparse q-learning with mirror descent. In *Conference on Uncertainty in Artificial Intelligence*, pp. 564–573. 2012.

50. S. Mahadevan, B. Liu, P. Thomas, W. Dabney, S. Giguere, N. Jacek, I. Gemp, and J. Liu. Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. In *arXiv preprint arXiv:1405.6757*, 2014.

51. S. Mahadevan and M. Maggioni. Value function approximation with diffusion wavelets and laplacian eigenfunctions. In *Advances in Neural Information Processing Systems*, pp. 843–850. 2006.

52. S. Mahadevan and M. Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. In *Journal of Machine Learning Research*, 8(Oct), pp. 2169–2231, 2007.

53. S.G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. In *Transactions on Signal Processing*, 41(12), pp. 3397–3415, 1993.

54. D. Meyer, R. Degenne, A. Omrane, and H. Shen. Accelerated gradient temporal difference learning algorithms. In *Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 1–8. 2014.

55. D. Meyer, H. Shen, and K. Diepold. $\ell_1$-regularized gradient temporal difference learning algorithms. In *European Workshop on Reinforcement Learning*. 2012.

56. C.A. Micchelli, L. Shen, and Y. Xu. Proximity algorithms for image models: denoising. In *Inverse Problems*, 27(4), 2011.

57. J.J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. In *CR Acad. Sci. Paris Ser. A Math.*, 255, pp. 2897–2899, 1962.

58. J.J. Moreau. Proximité et dualité dans un espace hilbertien. In *Bull. Soc. Math. France*, 93(2), pp. 273–299, 1965.

59. R. Munos. Error bounds for approximate policy iteration. In *International Conference on Machine learning*, volume 3, pp. 560–567. 2003.

60. Y.E. Nesterov. A method of solving a convex programming problem with convergence rate $o(\frac{1}{k^2})$. In *Soviet Mathematics Doklady*, 27(2), pp. 372–376, 1983.

61. Y. Nesterov. Gradient methods for minimizing composite objective function. In *Mathematical Programming*, pp. 1436–4646, 2007.

62. Y. Nesterov. *Introductory lectures on convex optimization: A basic course.* Springer Science, New York, 2013.

63. A.Y. Ng. Feature selection, $l_1$ vs. $l_2$ regularization, and rotational invariance. In *International conference on Machine learning*, p. 78. 2004.

64. C. Painter-Wakefield and R. Parr. Greedy algorithms for sparse reinforcement learning. In *arXiv preprint:1206.6485*, 2012.

65. C. Painter-Wakefield and R. Parr. *L1 Regularized Linear Temporal Difference Learning.* Technical report, Duke University, Durham, NC, 2012.

66. Y. Pan, A.M. White, and M. White. Accelerated gradient temporal difference learning. In *Conference on Artificial Intelligence*, pp. 2464–2470. 2017.

67. R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *International Conference on Machine learning*, pp. 737–744. 2007.

68. M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for markov decision processes. In *International Conference on Machine Learning*, pp. 871–878. 2010.

69. Z. Qin, W. Li, and F. Janoos. Sparse reinforcement learning via convex optimization. In *International Conference on Machine Learning*, pp. 424–432. 2014.

70. H. Robbins and S. Monro. A stochastic approximation method. In *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

71. B. Scherrer. Should one compute the temporal difference fix point or minimize the bellman residual? the unified oblique projection view. In *International Conference on Machine Learning*, pp. 959–966. 2010.

72. P.J. Schweitzer and A. Seidmann. Generalized polynomial approximations in markovian decision processes. In *Journal of Mathematical Analysis and Applications*, 110(2), pp. 568–582, 1985.

73. J. Sherman and W.J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. In *The Annals of Mathematical Statistics*, 21(1), pp. 124–127, 03 1950.

74. D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. In *Nature*, pp. 484–489, 2016.

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

76. R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, pp. 1038–1044. 1996.

77. R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, 1998.

78. R.S. Sutton, H.R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning*, pp. 993–1000. 2009.

79. R.S. Sutton, H.R. Maei, and C. Szepesvári. A convergent $o(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pp. 1609–1616. 2009.

80. G. Taylor. *Feature Selection for Value Function Approximation.* Ph.D. thesis, Duke University, 2011.

81. G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. In *Neural Computation*, 6(2), pp. 215–219, 1994.

82. R. Tibshirani. Regression shrinkage and selection via the lasso. In *Journal of the Royal Statistical Society. Series B*, pp. 267–288, 1996.

83. J.N. Tsitsiklis and B.V. Roy. An analysis of temporal-difference learning with function approximation. In *Transactions on Automatic Control*, 42(5), pp. 674–690, 1997.

84. J.N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1075–1081. 1997.

85. A. White and M. White. Investigating practical linear temporal difference learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pp. 494–502. 2016.

86. S. Wright and J. Nocedal. *Numerical optimization.* Springer Science, New York, 2006.

87. L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. In *Journal of Machine Learning Research*, 11(Oct), pp. 2543–2596, 2010.

88. J. Yang and Y. Zhang. Alternating direction algorithms for $\ell_1$-problems in compressive sensing. In *Journal on Scientific Computing*, 33(1), pp. 250–278, 2011.

89. H. Yu. Convergence of least squares temporal difference methods under general conditions. In *International Conference on Machine Learning*, pp. 1207–1214. 2010.

90. H. Yu. Least squares temporal difference methods: An analysis under general conditions. In *Journal on Control and Optimization*, 50(6), pp. 3310–3343, 2012.

91. H. Zou and T. Hastie. Regularization and variable selection via the elastic net. In *Journal of the Royal Statistical Society. Series B*, 67(2), pp. 301–320, 2005.

# Appendix A.

# Extended Derivations

In this Appendix, the extended derivation of gradients and properties of the MSPBE are presented that were too detailed to be included in the main text.

## A.1. Full RG Gradient Derivation

The objective function of RG is defined in equation (2.45) and can be further rewritten as

$$\begin{aligned}
J_{\mathrm{RG}} &= \|V_\theta - \mathcal{T}_\mu V_\theta\|_\xi^2 = \|\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2 \\
&= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top \Xi (\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\
&= (\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta)^\top \Xi (\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta) \\
&= \left(\theta^\top \Phi^\top \Xi - R_\mu^\top \Xi - \gamma \theta^\top \Phi^\top P_\mu^\top \Xi\right) (\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta) \\
&= \theta^\top \Phi^\top \Xi \Phi\theta - \theta^\top \Phi^\top \Xi R_\mu - \gamma \theta^\top \Phi^\top \Xi P_\mu \Phi\theta \\
&\quad - R_\mu^\top \Xi \Phi\theta + R_\mu^\top \Xi R_\mu + \gamma R_\mu^\top \Xi P_\mu \Phi\theta \\
&\quad - \gamma \theta^\top \Phi^\top P_\mu^\top \Xi \Phi\theta + \gamma \theta^\top \Phi^\top P_\mu^\top \Xi R_\mu + \gamma^2 \theta^\top \Phi^\top P_\mu^\top \Xi P_\mu \Phi\theta.
\end{aligned} \tag{A.1}$$

Deriving the gradient with respect to $\theta$ is now possible as in

$$\begin{aligned}
\nabla_\theta J_{\mathrm{RG}}(\theta) &= 2\Phi^\top \Xi \Phi\theta - \Phi^\top \Xi R_\mu - 2\gamma \Phi^\top \Xi P_\mu \Phi\theta - \Phi^\top \Xi R_\mu \\
&\quad + \gamma \Phi^\top P_\mu^\top \Xi R_\mu - 2\gamma \Phi^\top P_\mu^\top \Xi \Phi\theta + \gamma \Phi^\top P_\mu^\top \Xi R_\mu + 2\gamma^2 \Phi^\top P_\mu^\top \Xi P_\mu \Phi\theta \\
&= 2\left(\Phi^\top \Xi \Phi\theta - \Phi^\top \Xi R_\mu - \gamma \Phi^\top \Xi P_\mu \Phi\theta + \gamma \Phi^\top P_\mu^\top \Xi R_\mu + \gamma \Phi^\top P_\mu^\top \Xi \gamma P_\mu \Phi\theta\right) \\
&= 2\left(\Phi^\top - \gamma \Phi^\top P_\mu^\top\right) (\Xi \Phi\theta - \Xi R_\mu - \gamma \Xi P_\mu \Phi\theta).
\end{aligned} \tag{A.2}$$

Using the equivalent expectation formulation for these terms as presented in equation (2.23) and observing that

$$\Phi - \gamma P_\mu \Phi = \mathbb{E}_{P_\mu}\left[\phi^\top - \gamma(\phi')^\top\right] = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}(\phi(s_i)^\top - \gamma\phi(s_j)^\top), \tag{A.3}$$

and likewise

$$\Phi^\top - \gamma\Phi^\top P_\mu^\top = \mathbb{E}_{P_\mu}\left[\phi - \gamma\phi'\right] = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}(\phi(s_i) - \gamma\phi(s_j)), \tag{A.4}$$

we can continue the derivation of the gradient as

$$\begin{aligned}
\nabla_\theta J_{\mathrm{RG}}(\theta) &= 2\mathbb{E}_{P_\mu}\left[\phi - \gamma(\phi')\right]\left(\mathbb{E}_\xi\left[\phi^\top\right]\theta - \mathbb{E}_\xi\left[r\right] - \gamma\mathbb{E}_{\xi,P_\mu}\left[(\phi')^\top\right]\theta\right) \\
&= 2\mathbb{E}_{P_\mu}\left[\phi - \gamma(\phi')\right]\left(\mathbb{E}_{\xi,P_\mu}\left[\phi^\top\theta - r - \gamma(\phi')^\top\theta\right]\right) \\
&= -2\mathbb{E}_{P_\mu}\left[\phi - \gamma(\phi')\right]\left(\mathbb{E}_{\xi,P_\mu}\left[r + \gamma(\phi')^\top\theta - \phi^\top\theta\right]\right)
\end{aligned} \tag{A.5}$$

and obtain the final gradient for RG as

$$\begin{aligned}
\frac{1}{2}\nabla_\theta J_{\mathrm{RG}}(\theta) &= -\mathbb{E}_{P_\mu}\left[\phi - \gamma\phi'\right]\mathbb{E}_{\xi,P_\mu}\left[r + \gamma(\phi')^\top\theta - \phi^\top\theta\right] \\
&= -\mathbb{E}_{P_\mu}\left[\phi - \gamma\phi'\right]\mathbb{E}_{\xi,P_\mu}\left[\delta(\theta,\theta)\right].
\end{aligned} \tag{A.6}$$

## A.2. Full GTD Gradient Derivation

As the objective function for the GTD2 and TDC algorithm, we have the mean squared projected Bellman error as defined in equation (2.51). For deriving the gradient, we first have to establish in concordance with [47] the following quantities as

$$\begin{aligned}
\Pi_\mu^\top \Xi \Pi_\mu &= (\Phi(\Phi^\top\Xi\Phi)^{-1}\Xi)^\top\Xi(\Phi(\Phi^\top\Phi)^{-1}\Phi^\top\Xi \\
&= ((\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi)^\top\Phi^\top\Xi\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi \\
&= \Xi^\top\Phi(\Phi^\top\Xi\Phi)^{-1}(\Phi^\top\Xi\Phi)(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi \\
&= \Xi^\top\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi.
\end{aligned} \tag{A.7}$$

The as a first step the objective function $J_{\text{TDC}}$ can be rewritten as

$$\begin{aligned}
J_{\text{TDC}}(\theta) &= \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2 \\
&= (\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top \Xi(\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\
&= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta)^\top \Pi_\mu^\top \Xi \Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta) \\
&= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\
&= (\Phi\theta - R_\mu - \gamma P_\mu \Phi\theta)^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi(\Phi\theta - R_\mu - \gamma P_\mu\Phi\theta) \\
&= (\theta^\top \Phi^\top \Xi^\top \Phi - R_\mu^\top \Xi^\top \Phi - \gamma\theta^\top \Phi^\top P_\mu^\top \Xi^\top \Phi)(\Phi^\top \Xi\Phi)^{-1} \\
&\quad (\Phi^\top \Xi\Phi\theta - \Phi^\top \Xi R_\mu - \gamma\Phi^\top \Xi P_\mu \Phi\theta) \\
&= (\theta^\top \Phi^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1} - R_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1} - \gamma\theta^\top \Phi^\top P_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}) \\
&\quad (\Phi^\top \Xi\Phi\theta - \Phi^\top \Xi R_\mu - \gamma\Phi^\top \Xi P_\mu \Phi\theta) \\
&= \theta^\top \Phi^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi\Phi\theta - R_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi\Phi\theta \\
&\quad - \gamma\theta^\top \Phi^\top P_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi\Phi\theta - \theta^\top \Phi^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi R_\mu \\
&\quad + R_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi R_\mu + \gamma\theta^\top \Phi^\top P_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi R_\mu \\
&\quad - \gamma\theta^\top \Phi^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi P_\mu\Phi\theta + \gamma R_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi P_\mu\Phi\theta \\
&\quad + \gamma^2\theta^\top \Phi^\top P_\mu^\top \Xi^\top \Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi P_\mu\Phi\theta
\end{aligned} \tag{A.8}$$

Now deriving the gradient of this form can be done as follows, noting that

$$\mathbb{E}_\xi\left[\phi\phi^\top\right]^\top = \left(\Phi^\top \Xi\Phi\right)^\top = \Phi^\top \Xi\Phi = \mathbb{E}_\xi\left[\phi\phi^\top\right], \tag{A.9}$$

because $\Xi$ is a diagonal matrix

$$\begin{aligned}
\nabla_\theta(J_{\text{TDC}}(\theta)) &= 2\Big(\Phi^\top \Xi\Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi\Phi\theta - \Phi^\top \Xi\Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi R_\mu \\
&\quad - \gamma\Phi^\top P_\mu^\top \Xi\Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi\Phi\theta + \gamma\Phi^\top P_\mu^\top \Xi\Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi R_\mu \\
&\quad -\gamma\Phi^\top \Xi\Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi P_\mu\Phi\theta + \gamma^2\Phi^\top P_\mu^\top \Xi\Phi(\Phi^\top \Xi\Phi)^{-1}\Phi^\top \Xi P_\mu\Phi\theta\Big) \\
&= 2\left(\Phi^\top \Xi\Phi - \gamma\Phi^\top P_\mu^\top \Xi\Phi\right)\left(\Phi^\top \Xi\Phi\right)^{-1}\left(\Phi^\top \Xi\Phi\theta - \Phi^\top \Xi R_\mu - \gamma\Phi^\top \Xi P_\mu\Phi\theta\right)
\end{aligned} \tag{A.10}$$

Using the established formulations for the expectations of the matrix quantities in equation (2.23), we can reformulate the gradient in terms of expectations as

$$\begin{aligned}\nabla_\theta(J_{\mathrm{TDC}}(\theta)) &= \left(\mathbb{E}_\xi\left[\phi\phi^\top\right] - \gamma\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi')^\top\right]\right)\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1} \\ &\qquad \left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\theta - \mathbb{E}_{\xi,P_\mu}\left[\phi r\right] - \gamma\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi')^\top\right]\theta\right) \\ &= 2\mathbb{E}_{\xi,P_\mu}\left[\phi\phi^\top - \gamma\phi(\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi\phi^\top\theta - \phi r - \gamma\phi(\phi')^\top\theta\right] \\ &= -2\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi^\top - \gamma(\phi')^\top)\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi(r + \gamma(\phi')^\top\theta - \phi^\top\theta)\right].\end{aligned} \tag{A.11}$$

and hence the gradient simplifies to

$$\begin{aligned}\frac{1}{2}\nabla_\theta(J_{\mathrm{TDC}}(\theta)) &= -\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi - \gamma\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi(r + \gamma(\phi')^\top\theta - \phi^\top\theta)\right] \\ &= -\mathbb{E}_{\xi,P_\mu}\left[\phi(\phi - \gamma\phi')^\top\right]\left(\mathbb{E}_\xi\left[\phi\phi^\top\right]\right)^{-1}\mathbb{E}_{\xi,P_\mu}\left[\phi\delta(\theta,\theta)\right].\end{aligned} \tag{A.12}$$

## A.3. Derivations for Properties of the MSPBE Function

In order to show the convexity and bounds on the Lipschitz constant $L$ for the MSPBE, we need to re-weight the cost function in terms of the standard $\ell_2$ norm.

$$\begin{aligned}J_{\mathrm{TDC}}(\theta) &= \|\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)\|_\xi^2 \\ &= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta)^\top\Pi_\mu^\top\Xi\Pi_\mu(\Phi\theta - \mathcal{T}_\mu(\Phi\theta) \\ &= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top\Xi\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\ &= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top\sqrt{\Xi}\sqrt{\Xi}\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\sqrt{\Xi}\sqrt{\Xi}\Phi(\Phi^\top\Xi\Phi)^{-1} \\ &\qquad \Phi^\top\sqrt{\Xi}\sqrt{\Xi}(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\ &= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top\sqrt{\Xi}^\top(\sqrt{\Xi}\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\sqrt{\Xi})^\top \\ &\qquad (\sqrt{\Xi}\Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\sqrt{\Xi})\sqrt{\Xi}(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\ &= (\Phi\theta - \mathcal{T}_\mu(\Phi\theta))^\top\sqrt{\Xi}^\top\hat{\Pi}_\mu^\top\hat{\Pi}_\mu\sqrt{\Xi}(\Phi\theta - \mathcal{T}_\mu(\Phi\theta)) \\ &= \|\hat{\Pi}_\mu\sqrt{\Xi}(\Phi\theta - \mathcal{T}_\mu(\Phi\theta))\|_2^2,\end{aligned} \tag{A.13}$$

with $\hat{\Pi}_\mu = \sqrt{\Xi}\Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top\sqrt{\Xi}$.

To derive further properties of the MSBPE function, the Hessian $\mathcal{H}_J(\theta)$ is needed and the derivation is written as

$$
\begin{aligned}
\frac{1}{2}\nabla_\theta J_{\mathrm{TDC}}(\theta) &= \left(\Phi^\top \Xi \Phi - \gamma \Phi^\top P_\mu^\top \Xi \Phi\right)\left(\Phi^\top \Xi \Phi\right)^{-1}\left(\Phi^\top \Xi \Phi\theta - \Phi^\top \Xi R_\mu - \gamma \Phi^\top \Xi P_\mu \Phi\theta\right) \\
&= \Phi^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi \Phi\theta - \Phi^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi R_\mu \\
&\quad - \gamma \Phi^\top P_\mu^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi \Phi\theta + \gamma \Phi^\top P_\mu^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi R_\mu \\
&\quad - \gamma \Phi^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi P_\mu \Phi\theta + \gamma^2 \Phi^\top P_\mu^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi P_\mu \Phi\theta, \\
\frac{1}{2}\nabla_\theta^2 J_{\mathrm{TDC}}(\theta) &= \Phi^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi \Phi - \gamma \Phi^\top P_\mu^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi \Phi \\
&\quad - \gamma \Phi^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi P_\mu \Phi + \gamma^2 \Phi^\top P_\mu^\top \Xi \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi P_\mu \Phi \\
&= \left(\Phi^\top \Xi \Phi - \gamma \Phi^\top P_\mu^\top \Xi \Phi\right)\left(\Phi^\top \Xi \Phi\right)^{-1}\left(\Phi^\top \Xi \Phi - \gamma \Phi^\top \Xi P_\mu \Phi\right) \\
&= \left(\Phi - \gamma P_\mu \Phi\right)^\top \Xi \Phi \left(\Phi^\top \Xi \Phi\right)^{-1}\Phi^\top \Xi \left(\Phi - \gamma P_\mu \Phi\right) \\
&= \Phi^\top \left(I - \gamma P_\mu\right)^\top \sqrt{\Xi}\hat{\Pi}_\mu \sqrt{\Xi}\left(I - \gamma P_\mu\right)\Phi \\
&= \Phi^\top \left(I - \gamma P_\mu\right)^\top \Pi_\mu^\top \Xi \Pi_\mu \left(I - \gamma P_\mu\right)\Phi \\
&= A^\top A,
\end{aligned}
\tag{A.14}
$$

with $A := \sqrt{\Xi}\Pi_\mu\left(I - \gamma P_\mu\right)\Phi$.