Master's Thesis

Stabilizing Off-Policy TD with Feature Normalization

Aditya Bhatt

February 11th, 2019

Albert-Ludwigs-University Freiburg

Department of Computer Science

Computer Vision Group

Writing Period

10. 08. 2018 – 11. 02. 2019

Examiner

Prof. Dr. Thomas Brox

Second Examiner

Prof. Dr. Joschka Boedecker

Advisers

Max Argus, Artemij Amiranashvili

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

In Reinforcement Learning (RL), Temporal-Difference (TD) methods are a family of powerful algorithms that apply *bootstrapping* to efficiently learn various value functions by relating predictions at consecutive time-steps to each other. For TD(0), the most fundamental of these, there exist proofs of convergence when using parametrized function approximators in the on-policy case. TD(0) can also be used for off-policy learning, where data gathered under one behavior policy can be reused to predict the future under a different policy. However, the confluence of approximation, bootstrapping, and off-policy learning — called *the deadly triad* — runs a risk of divergence. In this thesis, we study how feature normalization can affect the convergence properties of TD(0). Following this analysis, we propose *CrossNorm*, a variant of Batch Normalization that accounts for the presence of *two* data distributions in off-policy learning. We show that CrossNorm improves the stability of the learning process in both policy evaluation and policy improvement. In the Deep RL case, CrossNorm is easily incorporated into methods like DDPG and TD3. For the first time, we demonstrate stable and superior training of DDPG without the use of target networks.

Contents

List of Figures

1 Introduction

Modern Reinforcement Learning (RL) has achieved many impressive results (Mnih et al., 2015; Silver et al., 2017), however a major concern is its lack of data efficiency. Indeed, many popular RL algorithms require a large amount of trial and error processes, often on the order of that needed by random search (Mania et al., 2018). Model-based RL methods promise to make the most effective use of available data, but sometimes learning a *good* transition model can be prohibitively difficult. A more practical approach is to avoid learning a model altogether, and use *off-policy* model-free RL methods to estimate how valuable behavior under a hypothetical policy could be, even if the historical data was gathered using a very different policy.

Typically, these methods are used with Temporal Difference (TD) learning, which relates future predictions at consecutive timesteps to each other; known as *bootstrapping*, this is a form of Dynamic Programming that allows for very fast learning from tiny fragments of experience rather than having to process full state-space trajectories.

Off-policy RL methods are very data-efficient learning algorithms, and when combined with Deep Learning they are particularly important in Robotics where trial-and-error learning is expensive (Lillicrap et al., 2015; Gu et al., 2017). Unfortunately, the highly desirable combination of off-policy learning, bootstrapping, and function approximation suffers from the risk of divergence during optimization. These three are appropriately known as *the deadly triad* of RL; a combination of any two without the other is convergent.

To address this problem, a common practice in Deep RL is to use *target networks* (Mnih et al., 2015; Lillicrap et al., 2015) to slow down bootstrapping by maintaining a separate, stale copy of the training parameters; this stabilizes agent training at the cost of learning speed. It is believed that if the dependency on target networks could be eliminated while still preserving stability, agents could learn even faster (Plappert et al., 2018). In this thesis, we discover a simple and convenient manipulation of the features that achieves this goal. The technique, which we name *CrossNorm*, is a variant of Batch Normalization (Ioffe and Szegedy, 2015) that explicitly accounts for the fact that off-policy learning deals with not one but *two* data distributions. Agents trained with CrossNorm no longer require target networks for stable training, and exhibit large speedups in learning.

2 Background

Reinforcement Learning (Sutton and Barto, 2018) studies how an *agent* interacts with an *environment* with the aim of learning to solve problems within it. In the standard RL formulation, the only indicator of how well an agent is performing is a single scalar *reward* determined by its interaction with the environment. The agent must aim to maximize the sum of all rewards it will see during its lifetime.

2.1 Markov Decision Processes

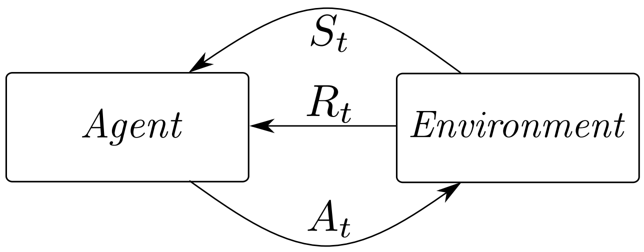


Figure 1: At each timestep t, the agent senses an observation of the environment state S_t, and performs an action A_t that affects the system. For this interaction, it gets an instantaneous reward R_t.

The agent-environment cycle in Figure 1 is formalized using Markov Decision Processes (MDPs). The MDP contains a set of states $\mathcal{S}$ and a set of actions $\mathcal{A}$. All the important quantities at each time step — the state $S_t \in \mathcal{S}$, the action $A_t \in \mathcal{A}$, and the reward $R_t \in \mathbb{R}$ — are random variables. States are sampled by the agent from the environment, and actions are received by the environment from the agent. The behavior of the agent, given by its actions, determines the scalar reward. The transitions between timesteps are assumed to be *markovian*; a system state depends, by construction, only upon the previous state-action pair:

$$p\left(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...\right) \doteq p\left(s_{t+1}|s_t, a_t\right)$$

Any rule or process by which the agent selects actions is called a *policy.* A policy π can either be stochastic — in which case it defines a density over actions as $A_t \sim \pi(a_t|s_t)$ — or it can be deterministic, in which case it is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

Each transition gets a corresponding reward, so the history of the system can be written as a sequence of *experience tuples* (s_t, a_t, r_t, s_{t+1}). As the system evolves over time, the *return* following any timestep t is the total reward gathered until termination:

$$G_t \doteq R_t + R_{t+1} + R_{t+2} + ... + R_T = \sum_{k=0}^{T-t} R_{t+k}$$

where T is some final timestep. Often, T could be very large or even infinite, and the return could grow indefinitely. Therefore, an often-used alternative is the *discounted return*:

$$G_t^\gamma \doteq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

The exponentially decaying weighting of the reward signal forces the sum to converge to a finite value. For convenience, we will drop γ from the return symbol and simply use G_t to mean a discounted return. The return G_t is a random variable that scores a random long-term future trajectory. Thus at each timestep, an optimal agent must behave so as to maximize the *expected return* $\mathbb{E}[G_t]$; that is the goal of RL.

2.1.1 Model-Based RL

If the agent has knowledge of the transition model, it can utilize powerful techniques like model-predictive control to predict the future and determine the best course of action. With learned, differentiable, transition and reward models, a parametrized policy could even be optimized to achieve the best possible return (Nagabandi et al., 2018). Despite the data-efficiency of such approaches (once a model is learned), iterated predictions far into the future could be very wrong because of compounding model errors — resulting in inefficient policies.

2.1.2 Model-Free RL

In model-free RL, which concerns the rest of this thesis, the agent avoids learning a model altogether. Instead, it attempts to directly improve the expected return by learning a better policy from trial-and-error experiences.

2.2 Policies and Value Functions

In this section, we define some fundamental RL concepts that will be extensively used in the rest of the thesis.

Definition 2.2.1. The *value function* $v^\pi : \mathcal{S} \to \mathbb{R}$ is the expected return when starting from state s and following π until termination:

$$v^\pi(s) \doteq \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

Definition 2.2.2. The *action-value function* $q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the expected return when taking action a in state s and then following π until termination:

$$q^\pi(s) \doteq \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right]$$

Value functions evaluate the long-term future behavior of the agent under a given policy. Due to the markov property, we can express them in a recursive form using the *Bellman Expectation Equations*:

$$\begin{aligned} v^\pi(s) &= \mathbb{E}_\pi \left[R_t + \gamma v^\pi(S_{t+1}) | S_t = s \right] \\ q^\pi(s, a) &= \mathbb{E}_\pi \left[R_t + \gamma q^\pi(S_{t+1}, \pi(S_{t+1})) | S_t = s, A_t = a \right] \end{aligned} \tag{1}$$

Once we have scorings on policies, the next step is to compare them.

Definition 2.2.3. For two policies π and π', we define $\pi \geq \pi'$ as holding if and only if $v^{\pi}(s) \geq v^{\pi'}(s)$ for each $s \in \mathcal{S}$.

The partial ordering $\geq$ betwen policies tells us that one policy is *better* than another. It implies the existence of a nonempty set of the best possible policies, with the same corresponding value function.
We refer to any such *optimal policy* as π^*, and the corresponding *optimal value functions* as $v^*(s) \doteq max_{\pi} v^{\pi}(s)$ and $q^*(s,a) \doteq max_{\pi} q^{\pi}(s,a)$. These, too, admit a recursive definition using the *Bellman Optimality Equations*:

$$\begin{aligned} v^*(s) &= \max_a \mathbb{E}_{\pi} \left[r(s,a) + \gamma v^*(S_{t+1}) | S_t = s, A_t = a \right] \\ q^*(s,a) &= \max_{a'} \mathbb{E}_{\pi} \left[r(s,a) + \gamma q^*(S_{t+1}, a') | S_t = s, A_t = a \right] \end{aligned} \tag{2}$$

Knowledge of the function $q^*(s,a)$ over all of $\mathcal{S}$ and $\mathcal{A}$ amounts to having completely solved the RL problem, because then optimal policy is just a greedy maximization of the function at each state:

$$\pi^*(s) = \operatorname*{argmax}_a q^*(s,a)$$

2.3 Dynamic Programming

Using Dynamic Programming, we can use the previously identified recursions to actually estimate value functions from data and find policies. For the following concepts, we assume a tabular representation of the value function, i.e. all value estimates are stored in a large table with an entry for each state (and action, if applicable). To avoid confusion with the true value functions — e.g. $v^{\pi}(s)$ — we use the uppercase notation — e.g. $V^{\pi}(s)$ — to denote the estimated values.

2.3.1 Policy Evaluation

To improve a policy π, the agent must first be able to evaluate it. For this, it must estimate $v^{\pi}(s)$ by gathering data from interactions. The most straightforward way to do this *policy evaluation* for a state s is to simply run π to completion many times, and then average the returns for all those trajectories. This is known as *Monte-Carlo* estimation. However, for some MDPs trajectories may terminate after a very long time, if at all. Waiting for them to finish before updating the valuation can therefore be impractical. A much more efficient way is to use dynamic programming, by turning the Bellman Expectation Equations (Equation (1)) into assignments:

$$\begin{aligned} v^{\pi}(s) &= \mathbb{E}_{\pi}\left[R_t + \gamma v^{\pi}(S_{t+1}) | S_t = s\right] \\ &\downarrow \\ V^{k+1}(s) &\doteq \mathbb{E}_{\pi}\left[R_t + \gamma V^{k}(S_{t+1}) | S_t = s\right] \end{aligned}$$

The above equation looks ahead one step in time, and enforces a consistency condition on the estimated value by updating it to satisfy the bellman equation. Such an operation is called a *backup*, and is succintly described with the *Bellman Backup Operator* T^{π}:

$$V^{k+1} \doteq T^{\pi} V^{k} \tag{3}$$

Here, we have omitted the argument s and equivalently represented the values over the entire state space as the vector $V^k \in \mathbb{R}^{N \times 1}$ where $N = |\mathcal{S}|$. It has been shown in Bertsekas and Tsitsiklis (1996) that T^{π} is a *contraction* in the max-norm $||\cdot||_{\infty}$; for any two value estimates $U(s)$ and $V(s)$:

$$||T^{\pi}U - T^{\pi}V||_{\infty} \leq \gamma ||U - V||_{\infty}$$

The contraction property means that regardless of the valuation V^0 we start with, repeated backups perform fixed-point iteration, causing policy evaluation to converge to the true value function.

$$\lim_{k \to \infty} (T^{\pi})^k V^0 = v^{\pi}$$

2.3.2 Policy Improvement

Having estimated $Q^\pi(s,a)$ using policy evaluation, we can *improve* π. If taking an action at s according to $\pi(s)$ provides the highest-possible return, then π is optimal at s. But if it does not, then π can be modified to become π' so that $\pi'(s) = \text{argmax}_a\, Q^\pi(s,a)$.
Thus, any modification π' that acts greedily w.r.t $Q^\pi(s,a)$ is an improvement over π. *Policy Iteration* is the process of alternating between policy evaluation and policy improvement; it converges to the optimal value function and policy.

Policy evaluation can be expensive, requiring several sweeps over the entire state space. When coupled with a changing policy, policy iteration could require a lot of computation. It turns out that this is not strictly necessary; we can simply perform a combined single step of evaluation and improvement by turning the Bellman Optimality Equation (Equation (2)) into an assignment. This is called *Value Iteration*, is also a contraction in the max-norm, and converges to the optimal value function as the fixed-point.

2.4 Temporal Difference Learning

In this section, we describe how to evaluate policies with approximate Temporal Difference (TD) learning. From a state space of size $N = |\mathcal{S}|$, the agent sees an observation vector $\boldsymbol{\phi}(s) \in \mathbb{R}^n$ corresponding to each state s. We make use of a linear function approximator [1] parametrized by $\boldsymbol{\theta}$, such that the value estimate is constructed as:

$$V_{\boldsymbol{\theta}}(s) = \boldsymbol{\phi}(s)^\top \boldsymbol{\theta}$$

The return from a state s is recursively related to that from the successor state s' as $v^\pi(s) = R + \gamma v^\pi(s')$. While there are many TD algorithms Sutton et al. (2009), we consider here the simplest, TD(0). Under an imperfect approximation, we will have a

[1]The linear approximator also accounts for the case of tabular value function learning! In fact, if $n = N$ and if the feature vectors are as one-hot encodings of the states, then $\boldsymbol{\theta}$ is exactly a table of values, and $\boldsymbol{\phi}(s)^\top \boldsymbol{\theta}$ corresponds to a table lookup for the state s.

nonzero residual difference between estimates of the two sides:

$$\delta \doteq R + \gamma V_{\boldsymbol{\theta}}(s') - V_{\boldsymbol{\theta}}(s)$$

We can then use δ to slowly correct our estimator, with a small step size η:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\delta$$

This looks very similar to gradient descent learning. In fact, δ is known as the semi-gradient (Sutton and Barto, 2018) of the following loss function:

$$L = \frac{1}{2}\left\{R_t + \gamma V_{\boldsymbol{\theta}}(s') - V_{\boldsymbol{\theta}}(s)\right\}^2$$

2.4.1 On-policy and Off-policy learning

When the dataset of experiences associating s and s' is gathered while following π, then estimating v^{π} is termed *on-policy* learning.

There is another way to learn, termed *off-policy* learning, where we seek to predict v^{π} of the target policy π from samples gathered according to a different *behavior policy* μ. Off-policy learning is much more powerful than on-policy learning because it can make use of all the previous experience collected by an agent. It lets us predict the future under a policy π despite having performed very different policies in the past.

In the tabular case, TD(0) always converges. However, under function approximation, while on-policy TD(0) converges, the off-policy case does not. This combination of bootstrapping, function approximation, and off-policiness is called *the deadly triad.*

2.5 Deterministic Policy Gradient

When there is a very large action space $\mathcal{A}$, choosing an optimal action is impractical as it would require solving an optimization problem at each timestep. In these cases, actor-critic methods are useful because they explicitly represent the policy as a parametrized *actor* function $\pi_{\boldsymbol{\theta}_\pi}(s)$ mapping states to actions, and the action-value function as a *critic* $Q_{\boldsymbol{\theta}_Q}(s,a)$ to evaluate the actor.

In this thesis, we consider *Deterministic Policy Gradients* (Silver et al., 2014), an off-policy actor-critic algorithm which updates the actor parameters $\boldsymbol{\theta}_\pi$ via gradient ascent over the average Q^π prediction of the critic:

$$\nabla_{\boldsymbol{\theta}_\pi} J(\boldsymbol{\theta}_\pi) = \mathbb{E}_\mu \left[\nabla_a Q_{\boldsymbol{\theta}_Q}(s,a)\, |_{a=\pi_{\boldsymbol{\theta}_\pi}(s)} \nabla_{\boldsymbol{\theta}_\pi} \pi_{\boldsymbol{\theta}_\pi}(s) \right]$$

The $\mathbb{E}_\mu$ indicates that the average is taken over (s,a) pairs sampled from data gathered under a historical behavior policy μ. This complex expression simply means that the actor is optimized to increase the predicted return, by backpropagating from the average $Q_{\boldsymbol{\theta}_Q}(s, \pi_{\boldsymbol{\theta}_\pi}(s))$ output through the critic into the actor.

The critic is trained to evaluate the newest actor, by minimizing the following loss with TD(0):

$$L(\boldsymbol{\theta}_Q) = \mathbb{E}_\mu \left[\left\{ Q_{\boldsymbol{\theta}_Q}(s,a) - y \right\}^2 \right]$$
$$y = r + \gamma Q_{\boldsymbol{\theta}_Q}\left(s', \pi_{\boldsymbol{\theta}_\pi}(s')\right)$$

These two steps of policy evaluation and improvement are interleaved one after the other, with just one gradient update performed per step.

When the actor and critic are parametrized with deep neural networks, the method is called *Deep Deterministic Policy Gradients* (DDPG) by Lillicrap et al. (2015).

2.5.1 Target Networks

As DDPG uses off-policy TD, it is prone to divergence with parameters spiraling off to infinity. To avoid the deadly triad, off-policy Deep RL uses a pair of additional *Target Networks*; copies of the actor and critic networks with a different set of parameters $\bar{\boldsymbol{\theta}}_\pi$ and $\bar{\boldsymbol{\theta}}_Q$ which are delayed versions of the "live" network parameters $\boldsymbol{\theta}_\pi$ and $\boldsymbol{\theta}_Q$. These are used to generate the regression targets y as:

$$y = r + \gamma Q_{\bar{\boldsymbol{\theta}}_Q}\left(s', \pi_{\bar{\boldsymbol{\theta}}_\pi}(s')\right)$$

Target Networks can be one of two types:

- **Hard**, which are periodically updated older snapshots of $\boldsymbol{\theta}_\pi$ and $\boldsymbol{\theta}_Q$ as used with DQN (Mnih et al., 2015).

- **Soft**, which are slow moving averages (as proposed in DDPG):

$$\bar{\boldsymbol{\theta}}_Q \leftarrow (1-\tau)\bar{\boldsymbol{\theta}}_Q + \tau\boldsymbol{\theta}_Q$$
$$\bar{\boldsymbol{\theta}}_\pi \leftarrow (1-\tau)\bar{\boldsymbol{\theta}}_\pi + \tau\boldsymbol{\theta}_\pi$$

Soft target networks use basic exponential smoothing (a kind of low-pass filter) applied to the network parameters, with $\tau \ll 1$ being a small *smoothing factor*.

The DDPG authors train agents on various MuJoCo robotics tasks (Todorov et al., 2012), and find that using target networks is crucial for training without divergence. All Deep RL methods with off-policy TD, like NAF (Gu et al., 2016), TD3 (Fujimoto et al., 2018), and Soft Actor-Critic (SAC) (Haarnoja et al., 2018) use target networks;, on-policy methods such as A3C (Mnih et al., 2016) do not. It is unfortunate that we cannot use TD(0)-style updates directly, as target networks artificially slow down credit assignment backwards in time, reducing the speed of learning.

2.5.2 TD3

Algorithms like DDPG that attempt to maximize the *estimated* Q values induce a well-known overestimation bias in the predictions (Hasselt, 2010). This bias can be quite detrimental to learning as it provides misleading gradients to the actor, causing frequent training collapses and suboptimal performance. There exist ways to remedy this by using two Q functions and having them produce the targets for each other, e.g. Double Q Learning (Van Hasselt et al., 2016).

However, using Double Q estimation in DDPG still does not reduce this bias sufficiently. The TD3 algorithm (Fujimoto et al., 2018) manages to resolve this issue by maintaining two differently initialized critics $Q^1_{\boldsymbol{\theta}_1}$ and $Q^2_{\boldsymbol{\theta}_2}$, and using the *minimum* of their estimates as the regression target:

$$y = r + \gamma \min_{i \in \{1,2\}} Q^i_{\bar{\boldsymbol{\theta}}_i} \left(s', \pi_{\bar{\boldsymbol{\theta}}_\pi}(s')\right)$$

Apart from a few orthogonal improvements, this *Clipped Double Q* estimation is the only major difference between TD3 and DDPG.

3 Stability Analysis of TD Learning

In this chapter, we look at the dynamics of TD learning with linear function approximation, study the factors that determines convergence, and finally probe the effects of feature normalization on the stability properties.

We first study the stability of TD(0); the focus of all such discussions in the thesis is on policy evaluation for a policy π, unless otherwise stated. Much of the following analysis is derived from the exposition done in the Emphatic TD paper (Sutton et al., 2016) and the thesis of Mahmood (2017). We assume that the state space is of size $|\mathcal{S}| = N$, and the value function be represented as $V_{\boldsymbol{\theta}}(s) = \boldsymbol{\phi}(s)^\top \boldsymbol{\theta}$, where $\boldsymbol{\phi}(s) \in \mathbb{R}^n$ is a *feature vector* corresponding to the state $s \in \mathcal{S}$. As on-policy learning is a special case of off-policy learning with the behavior policy $\mu = \pi$, we directly study the general case of off-policy TD(0).

3.1 Mean ODE-based formulation

Consider a state transition (S_t, A_t, R_t, S_{t+1}) with $S_t, S_{t+1} \in \mathcal{S}$ and $R_t \in \mathbb{R}$. In the on-policy case, the data for the timestep t is also generated from our *target policy* π, according to $A_t \sim \pi(a|s = S_t)$, and $S_t \sim d_\pi(s)$, where $d_\pi(s)$ is the stationary state distribution that the MDP eventually converges to when following π. Then, at the time t, we have the following TD(0) update:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \eta(R_t + \gamma \boldsymbol{\phi}(S_{t+1})^\top \boldsymbol{\theta}_t - \boldsymbol{\phi}(S_t)^\top \boldsymbol{\theta}_t)\boldsymbol{\phi}(S_t) \tag{4}$$

Here, $\eta > 0$ is the step-size. Equation (4) applies to on-policy learning, but in the off-policy case, the data is gathered by a different *behavior policy* μ, i.e. $S_t \sim d_\mu(s)$. Essentially, we are trying to estimate a quantity under one distribution by sampling from another distribution; this means we must incorporate a *importance sampling ratio*:

$$\rho_t \doteq \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$$

and multiply the on-policy TD(0) update by ρ_t to obtain the off-policy version:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \eta\rho_t(R_t + \gamma\boldsymbol{\phi}(S_{t+1})^\top\boldsymbol{\theta}_t - \boldsymbol{\phi}(S_t)^\top\boldsymbol{\theta}_t)\boldsymbol{\phi}(S_t) \tag{5}$$

The ratio ρ_t is only defined if $\mu(A_t|S_t) > 0$ whenever $\pi(A_t|S_t) > 0$; this is called *coverage* — that μ's support contains that of π — and we assume it holds [1]. Intuitively, it does not make sense to predict the future under π if there is no overlap with what has been attempted by μ.

For brevity, let us use $\boldsymbol{\phi}_t$ to mean $\boldsymbol{\phi}(S_t)$. Then, Equation (5) can be rewritten as:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \eta\left(\underbrace{\rho_t R_t\boldsymbol{\phi}_t}_{\mathbf{b}_t} - \underbrace{\rho_t\boldsymbol{\phi}_t\left(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1}\right)^\top}_{\mathbf{A}_t}\boldsymbol{\theta}_t\right)\\ &= \boldsymbol{\theta}_t + \eta\left(\mathbf{b}_t - \mathbf{A}_t\boldsymbol{\theta}_t\right) \qquad (6)\end{aligned}$$

Equation (6) can be reshaped as the following discrete-time dynamical system:

$$\boldsymbol{\theta}_{t+1} = \left(I - \eta\mathbf{A}_t\right)\boldsymbol{\theta}_t + \eta\mathbf{b}_t \tag{7}$$

Here, $\mathbf{A}_t \in \mathbb{R}^{n\times n}$ and $\mathbf{b}_t \in \mathbb{R}^n$ are data-dependent random variables that vary per timestep. For a practical stability study, we are interested in whether the deterministic *expected update* converges. To study the expected update, it is sufficient to preserve the same structure as in Equation (7), while replacing $\mathbf{A}_t$ and $\mathbf{b}_t$ with their expectations.

[1] Practical behavior policies like ϵ-greedy and centered gaussians already have full coverage of the action space, so this is not a very strong requirement.

We are also interested in the time-independent "steady-state" values of these matrices — the expectation should be taken over the stationary state distribution — induced by the μ-influenced markov chain that generated the data. Therefore, we use:

$$\mathbf{A} = \lim_{t\to\infty} \mathbb{E}_\mu \left[\mathbf{A}_t\right] \quad \text{and} \quad \mathbf{b} = \lim_{t\to\infty} \mathbb{E}_\mu \left[\mathbf{b}_t\right]$$

which finally turns Equation (7) into a discrete-time first-order Ordinary Difference Equation (ODE) [2], where the only time-varying quantity is the vector of parameters $\boldsymbol{\theta}$:

$$\boxed{\boldsymbol{\theta}_{t+1} = \left(\mathbf{I} - \eta\mathbf{A}\right)\boldsymbol{\theta}_t + \eta\mathbf{b}} \tag{8}$$

The fixed-point of this iteration can be found by equating $\boldsymbol{\theta}_{t+1}$ to $\boldsymbol{\theta}_t$, and it is easy to see that the solution of the policy evaluation is $\boldsymbol{\theta}^* = \mathbf{A}^{-1}\mathbf{b}$. The question, however, is if this is a stable attractive fixed-point, so that optimization can reach it.

The convergence of the recurrence (8) is determined entirely by the eigenvalues of the *iteration matrix* $(\mathbf{I} - \eta\mathbf{A})$; to see this observe that the iterates must eventually stop changing, so:

$$\lim_{k\to\infty} (\mathbf{I} - \eta\mathbf{A})^k = \mathbf{0}$$

which can only happen when the spectral radius is less than one. It is demonstrated in **Corollary 3** of the thesis by Mahmood (2017) that this holds whenever η is sufficiently small and $Re(eig_i(\mathbf{A})) > 0, \quad 1 \leq i \leq n$, i.e. the eigenvalues of $\mathbf{A}$ must all have positive real parts[3]. As this matrix is the key to convergence, Sutton et al. (2016) and Mahmood (2017) term $\mathbf{A}$ the *key matrix.*

[2]Such approaches to stability analysis of stochastic approximation algorithms are referred to as "mean ODE" proofs; (Kushner and Yin, 2003; Mahmood, 2017).

[3]Observe that for an i-th eigenvalue λ_i of $\mathbf{A}$, the corresponding i-th eigenvalue of the iteration matrix is $1 - \eta\lambda_i$.

3.2 Structure of the Key Matrix

We now focus on the structure of the key matrix $\mathbf{A}$, to see what could affect its spectrum. Recall the definition of $\mathbf{A}_t$ from Equation (6):

$$\mathbf{A}_t = \rho_t \boldsymbol{\phi}_t \left(\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}\right)^\top$$

Using that, we can express $\mathbf{A}$ as

$$\begin{aligned}
\mathbf{A} &= \lim_{t\to\infty} \mathbb{E}_\mu \left[\mathbf{A}_t\right] \\
&= \lim_{t\to\infty} \mathbb{E}_\mu \left[\rho_t \boldsymbol{\phi}_t \left(\boldsymbol{\phi}_t - \gamma \boldsymbol{\phi}_{t+1}\right)^\top\right] \\
&= \sum_s \sum_a d_\mu(s)\mu(a|s)\rho_k \left[\boldsymbol{\phi}_k \left(\boldsymbol{\phi}_k - \gamma \boldsymbol{\phi}_{k+1}\right)^\top \middle| S_k = s, A_k = a\right] \\
&= \sum_s \sum_a d_\mu(s)\mu(a|s)\frac{\pi(a|s)}{\mu(a|s)} \left[\boldsymbol{\phi}_k \left(\boldsymbol{\phi}_k - \gamma \boldsymbol{\phi}_{k+1}\right)^\top \middle| S_k = s, A_k = a\right] \\
&= \sum_s \sum_a d_\mu(s)\pi(a|s) \left[\boldsymbol{\phi}_k \left(\boldsymbol{\phi}_k - \gamma \boldsymbol{\phi}_{k+1}\right)^\top \middle| S_k = s, A_k = a\right] \\
&= \sum_s d_\mu(s) \left[\boldsymbol{\phi}_k \left(\boldsymbol{\phi}_k - \gamma \sum_a \pi(a|s) \boldsymbol{\phi}_{k+1}\right)^\top \middle| S_k = s, A_k = a\right] \qquad (9)
\end{aligned}$$

Note that ϕ_k depends only on μ and not π, hence it is unaffected by the $\pi(a|s)$ factor which gets marginalized to 1 in the left term. At this point, we can begin transforming Equation (9) into a much more concise matrix form. To do this, we introduce three new matrices; let $\mathbf{P}_\pi \in \mathbb{R}^{N\times N}$ be the *stochastic matrix* of π-influenced state transition probabilities, and let $\mathbf{D}_\mu \in \mathbb{R}^{N\times N}$ be a diagonal matrix with the stationary state occupancy probabilities $d_\mu(s)$ along the diagonal. Lastly, let $\boldsymbol{\Phi} \in \mathbb{R}^{N\times n}$ be a stack of all state feature vectors over the state space:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(s_1)^\top \\ \boldsymbol{\phi}(s_2)^\top \\ \vdots \\ \boldsymbol{\phi}(s_N)^\top \end{bmatrix}$$

Again, following Sutton et al. (2016), with some notational liberties we continue from Equation (9):

$$\begin{aligned}
\mathbf{A} &= \sum_s d_\mu(s)\boldsymbol{\phi}(s)\left(\boldsymbol{\phi}(s) - \gamma\sum_{s'}[\mathbf{P}_\pi]_{ss'}\boldsymbol{\phi}(s')\right)^\top \\
&= \sum_s \boldsymbol{\phi}(s)[\mathbf{D}_\mu]_{ss}\left(\boldsymbol{\phi}(s) - \gamma\sum_{s'}[\mathbf{P}_\pi]_{ss'}\boldsymbol{\phi}(s')\right)^\top \\
&= \sum_s [\boldsymbol{\Phi}]_s^\top[\mathbf{D}_\mu]_{ss}\left(\boldsymbol{\phi}(s) - \gamma\sum_{s'}[\mathbf{P}_\pi]_{ss'}\boldsymbol{\phi}(s')\right)^\top \\
&= \sum_s [\boldsymbol{\Phi}]_s^\top[\mathbf{D}_\mu]_{ss}\left([\boldsymbol{\Phi}]_s^\top - \gamma\sum_{s'}[\mathbf{P}_\pi]_{ss'}[\boldsymbol{\Phi}]_{s'}^\top\right)^\top
\end{aligned}$$

which finally simplifies to

$$\boxed{\mathbf{A} = \boldsymbol{\Phi}^\top\mathbf{D}_\mu(\mathbf{I} - \gamma\mathbf{P}_\pi)\boldsymbol{\Phi}} \tag{10}$$

The features $\boldsymbol{\Phi}$ project the the $N \times N$ matrix $\mathbf{K} = \mathbf{D}_\mu(\mathbf{I} - \gamma\mathbf{P}_\pi)$ into a (typically) lower dimensional $n \times n$ $\mathbf{A}$ matrix. $\mathbf{K}$ is referred to by Mahmood (2017) the *big key matrix.*

3.3 Eigenvalues of the Key Matrix

A much stronger condition than having eigenvalues with positive real parts is positive definiteness[4]. Sutton et al. (2016) use a weaker definition of positive definiteness by dropping the matrix symmetry requirement and simply requiring that a matrix $\mathbf{M}$ is p.d. iff $\mathbf{x}^\top\mathbf{M}\mathbf{x} > 0, \forall\mathbf{x} \in \mathbb{R}^n$. [5]

We first see what happens in the on-policy case. Then, the big key matrix becomes $\mathbf{K} = \mathbf{D}_\pi(\mathbf{I} - \gamma\mathbf{P}_\pi)$.

To prove positive-definiteness of an asymmetric matrix $\mathbf{K}$, it should be sufficient to show that its symmetric form $S = \mathbf{K} + \mathbf{K}^\top$ is p.d., which follows if we can prove

[4]The latter implies the former, but the converse does not hold.

[5]Positive semi-definiteness also avoids divergence, and is often encountered in practice.

that S is diagonally dominant. Sutton et al. (2016) do exactly this, by showing that

- Diagonal entries of K are positive
- Off-diagonal entries of K are negative
- Each row-sum plus corresponding column-sum is greater than zero

The vector of row-sums of $\mathbf{K}$ can be computed by multiplying from the right by a vector of ones $\mathbf{1} \in \mathbb{R}^N$, and keeping in mind that $\mathbf{1}$ is a right-eigenvector of $\mathbf{P}_\pi$:

$$\begin{aligned}
\mathbf{K1} &= \mathbf{D}_\pi(\mathbf{I} - \gamma \mathbf{P}_\pi)\mathbf{1} \\
&= \mathbf{D}_\pi(\mathbf{1} - \gamma \mathbf{P}_\pi \mathbf{1}) \\
&= \mathbf{D}_\pi(\mathbf{1} - \gamma \mathbf{1}) \\
&= (1-\gamma)\mathbf{d}_\pi
\end{aligned}$$

where $\mathbf{d}_\pi$ is the vector representing the diagonal of $\mathbf{D}_\pi$, as $\mathbf{D}_\pi \mathbf{1} = \mathbf{d}_\pi$; all the row-sums are therefore positive. For the column-sums, we multiply from the left:

$$\begin{aligned}
\mathbf{1}^\top \mathbf{K} &= \mathbf{1}^\top \mathbf{D}_\pi(\mathbf{I} - \gamma \mathbf{P}_\pi) \\
&= \mathbf{d}_\pi^\top(\mathbf{I} - \gamma \mathbf{P}_\pi) \\
&= (\mathbf{d}_\pi^\top - \gamma \mathbf{d}_\pi^\top \mathbf{P}_\pi) \\
&= (\mathbf{d}_\pi^\top - \gamma \mathbf{d}_\pi^\top) \qquad (11) \\
&= (1-\gamma)\mathbf{d}_\pi^\top
\end{aligned}$$

where step (11) holds because the vector of stationary probabilities is always a left-eigenvector of the markov transition matrix; again, all column-sums are positive. Thus the on-policy $\mathbf{K}$ is positive definite. It follows that for any vector $\mathbf{x} \neq \mathbf{0} \in \mathbb{R}^N$, we have $\mathbf{x}^\top \mathbf{K} \mathbf{x} > 0$. Let us map any nonzero vector $\mathbf{y} \in \mathbb{R}^n$ into $\mathbb{R}^N$ as $\mathbf{\Phi y} = \mathbf{x}$.

$$\mathbf{x}^\top \mathbf{K} \mathbf{x} > 0 \implies \mathbf{y}^\top \mathbf{\Phi}^\top \mathbf{K \Phi y} > 0 \implies \mathbf{y}^\top \mathbf{A y} > 0$$

which proves that $\mathbf{A}$ is p.d., and therefore on-policy approximate TD(0) is convergent.

In off-policy TD(0), $\mathbf{K} = \mathbf{D}_\mu(\mathbf{1} - \gamma\mathbf{P}_\pi)$ is *not* guaranteed to be positive definite, because $\mathbf{d}_\mu^\top$ is not necessarily a left-eigenvector of $\mathbf{P}_\pi$, and we cannot use a trick like step 11; column-sums may be of any sign.

However, there is one useful property that holds regardless of on- or off-policy status: the eigenvalues of $\mathbf{K}$ always have positive real parts. This is easy to see by a straightforward application of Gershgorin's Circle Theorem Gershgorin (1931): The diagonal entries are still positive, the off-diagonal row entries are all negative, and the row-sums are positive, meaning that all the gershgorin disks — within which the eigenvalues must be present — lie safely on the right side of imaginary axis.

This helps when we have *tabular features*, in which case $N = n$ and $\mathbf{\Phi} = \mathbf{I}$ so that $\mathbf{A} = \mathbf{K}$, and then $\mathbf{A}$ inherits the well-behaved eigenvalues; thus tabular TD(0) always converges. Despite this property of $\mathbf{K}$, in the approximate case the *projected* matrix $\mathbf{A} = \mathbf{\Phi}^\top\mathbf{K}\mathbf{\Phi}$ can — and often does — have eigenvalues with negative real parts, and TD is then divergent. In the next chapter, we study how this problem could be remedied.

4 Method

We have seen how the convergence of TD(0) is affected by the matrices involved in its ODE. In this chapter, we describe our contributions towards improving its stability, both in the linear setting and with deep networks.

4.1 Stabilizing off-policy TD

Much research has gone into developing several off-policy TD algorithms that can be motivated as gradient descent, which include Gradient Temporal Differences (GTD/GTD-2) and TD with gradient Correction (TDC) Sutton et al. (2009); Bhatnagar et al. (2009). The update rules for these algorithms make use of an extra set of parameters and are not as simple as TD(0) or TD(λ). The newest and most promising alternative is *Emphatic TD* Sutton et al. (2016), which is considerably simpler and is also a semi-gradient algorithm like TD(0), albeit with a state-dependent emphasis weighting. All of these algorithms are proven to be convergent.

To the best of our knowledge, none of those algorithms have been successfully demonstrated in Deep RL so far — which happens to use TD with target networks. In this thesis, our focus is on how TD(0) could be made more stable.

As seen, the instability arises due to the key matrix $\mathbf{A}$ being indefinite (i.e. having eigenvalues with both positive and negative real parts), which means the solution lies at a *saddle point* and a first-order descent method diverges. To make TD(0) converge, we must find a way to ensure that all eigenvalues have positive real parts[1] We will henceforth refer to matrices with such a spectrum as *stable matrices.*

[1]This is a weaker requirement than needing $\mathbf{A}$ to be positive definite.

Recall that the *big* key matrix $\mathbf{K}$ is already a stable matrix. $\mathbf{A}$ is its (generally lower dimesional) *projection* using the features $\mathbf{\Phi}$:

$$\mathbf{A} = \mathbf{\Phi}^\top \mathbf{K} \mathbf{\Phi}$$

The projected version is often unstable. Now, observe that *the features completely determine the nature of the projection.* We already know of one kind of feature encoding ($\mathbf{\Phi} = \mathbf{I}$) that makes $\mathbf{A}$ stable. Could there exist a safe family of features to ensure that the eigenvalues will be well-behaved? Feature selection in TD learning is an open problem, and this possibility motivates our analysis in the rest of this thesis. To quote Sutton and Barto (2018) on off-policy semi-gradient methods:

" *Remember that these methods are guaranteed stable and asymptotically unbiased for the tabular case, which corresponds to a special case of function approximation. So it may still be possible to combine them with feature selection methods in such a way that the combined system could be assured stable.*"

4.2 Data Recentering

When doing linear regression via gradient descent, it is common to normalize or standardize your dataset. In particular, data standardization has numerous benefits. For a dataset with samples $\mathbf{x} \in \mathbb{R}^n$, standardization transforms $\mathbf{x}$ into $\hat{\mathbf{x}}$ whose k-th feature is:

$$\hat{x}_k \doteq \frac{x_k - \mathbb{E}[x_k]}{\sqrt{Var(x_k)}}$$

with the division performed elementwise. Incidentally, this is also what Batch Normalization (Ioffe and Szegedy, 2015) does. We now consider the effects of recentering by the mean features.

In off-policy TD, we are dealing with *not one but two* data distributions, where the "current" features $\boldsymbol{\phi}$ are produced by running the behavior policy μ and the "next" features $\boldsymbol{\phi}'$ are produced according to π. It is much more easy to see what the two datasets are in the case of Q^π estimation. For example in DPG learning Lillicrap et al. (2015), the input to the function approximator $Q^\pi_{\boldsymbol{\theta}_Q}$ is a concatenation of the state and action vectors. During a TD update, we sample an experience (s, a, r, s') and provide *two different* inputs to the estimator in a single step: (s, a) and $(s', \pi_{\boldsymbol{\theta}_\pi}(s'))$, where a was produced by μ. These are distributed differently on account of the differing actions.

Let us consider what happens when we subtract a vector $\mathbf{m} \in \mathbb{R}^n$ from the data, so that $\boldsymbol{\phi} \mapsto \boldsymbol{\phi} - \mathbf{m}$. Recall that in TD(0) our feature vectors are contained in $\boldsymbol{\Phi} \in \mathbb{R}^{N \times n}$. Subtracting $\mathbf{m}$ from the encoding results in a new feature matrix $\hat{\boldsymbol{\Phi}}$:

$$\hat{\boldsymbol{\Phi}} = \boldsymbol{\Phi} - \mathbf{1}\mathbf{m}^\top \tag{12}$$

where $\mathbf{1} \in \mathbb{R}^{N \times 1}$ is a column vector of ones. Now, let us give a meaning to $\mathbf{m}$. We want $\mathbf{m}$ to be the mean feature vector, but since it is unclear as to *which* distribution should be used for computing the feature mean, let us define an α-parametrized mixture of *both*, with α determining the contribution of μ [2]:

$$\begin{aligned} \mathbf{m} &= \mathbb{E}_\mu \left[\alpha\boldsymbol{\phi} + (1-\alpha)\mathbb{E}_\pi \boldsymbol{\phi}'\right] & (13) \\ \implies \mathbf{m}^\top &= \mathbf{1}^\top \mathbf{D}_\mu (\alpha \mathbf{I} + (1-\alpha)\mathbf{P}_\pi)\boldsymbol{\Phi} & (14) \end{aligned}$$

which amounts to sampling a batch of experience tuples and averaging the weighted sum. Note that the $\mathbb{E}_\pi$ is additionally conditioned upon a fixed previous state's $\boldsymbol{\phi} \sim \mu$.

In practice, we will only be able to *estimate* $\mathbf{m}$ by averaging a minibatch, which means it will not be exact, so we are also interested in points that are not exactly on the line connecting the two means $\mathbf{1}^\top \mathbf{D}_\mu \boldsymbol{\Phi}$ and $\mathbf{1}^\top \mathbf{D}_\mu \mathbf{P}_\pi \boldsymbol{\Phi}$. Therefore, let us increase generality and consider a full linear combination of the two:

$$\mathbf{m}^\top = \mathbf{1}^\top \mathbf{D}_\mu (\alpha \mathbf{I} + \beta \mathbf{P}_\pi)\boldsymbol{\Phi} \tag{15}$$

[2] Note that we have intentionally omitted arguments s or (s, a) here because we want to emphasise that this holds both for V^π and Q^π estimation.

Combining Equation (12) and Equation (15), we get our new feature matrix $\hat{\mathbf{\Phi}}$:

$$\begin{aligned}
\hat{\mathbf{\Phi}} &= \mathbf{\Phi} - \mathbf{1}\mathbf{m}^\top \\
&= \mathbf{\Phi} - \mathbf{1}\mathbf{1}^\top \mathbf{D}_\mu(\alpha\mathbf{I} + \beta\mathbf{P}_\pi)\mathbf{\Phi} \\
&= \Big(\mathbf{I} - \mathbf{1}\mathbf{1}^\top \mathbf{D}_\mu(\alpha\mathbf{I} + \beta\mathbf{P}_\pi)\Big)\mathbf{\Phi} \qquad (16)
\end{aligned}$$

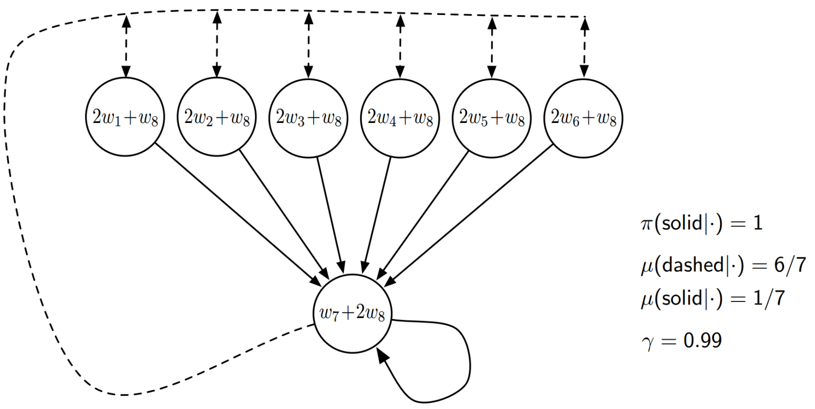
This has a simple structure; it is an $N \times N$ linear operator multiplied to $\mathbf{\Phi}$ from the left. That means our new key matrix can be expressed in terms of the original features with a *new* big key matrix:

$$\begin{aligned}
\mathbf{A}_{\alpha,\beta} &\doteq \hat{\mathbf{\Phi}}^\top \mathbf{K} \hat{\mathbf{\Phi}} \\
&\doteq \mathbf{\Phi}^\top \mathbf{K}_{\alpha,\beta} \mathbf{\Phi} \qquad (17)
\end{aligned}$$

Thus, we can say that recentering features around an expectation over our data amounts to simply changing the big key matrix $\mathbf{K}$ into $\mathbf{K}_{\alpha,\beta}$. Because of the term involving $\mathbf{1}\mathbf{1}^\top$, the new key matrices are low-rank perturbations of the originals; this has an effect of shifting the eigenvalues. A rigorous analysis of the eigenspectrum of $\mathbf{A}_{\alpha,\beta}$ and $\mathbf{K}_{\alpha,\beta}$ is out of scope for this thesis, and we leave it for future work. Instead, we now take a more empirical approach, and study the behavior of different α, β combinations on various problems.

4.3 Baird's Counterexample

The MDP we consider is the seven-star Baird's Counterexample (Baird, 1995). It demonstrates a case where off-policy TD diverges, both when doing stochastic updates with single samples, or when performing deterministic expected updates. As shown in Figure 2, there are only two actions: *dashed* and *solid*, which determine the state the agent ends up in. With μ, the probability of being in any one of the seven states is a uniform $\frac{1}{7}$, whereas under π, the agent ends up and remains in the bottom state s_7, having zero visitation probability elsewhere.



$2w_1+w_8$
$2w_2+w_8$
$2w_3+w_8$
$2w_4+w_8$
$2w_5+w_8$
$2w_6+w_8$
w_7+2w_8
$\pi(\text{solid}|\cdot) = 1$
$\mu(\text{dashed}|\cdot) = 6/7$
$\mu(\text{solid}|\cdot) = 1/7$
$\gamma = 0.99$

Figure 2: In Baird's Counterexample, the reward is always zero, and so is the true value function. The solid arrows describe transitions due to the target policy π, whereas the dashed arrows describe the behavior policy μ. The learnable parameters are the weights w_i, and the coefficients of the weights in the various nodes describe the feature vectors in those states. The figure is reproduced from Sutton and Barto (2018).

4.3.1 Convergence with normalization

Despite this being a very simple MDP — with zero rewards everywhere — the weights[3] and value estimations diverge exponentially to infinity, solely on account of off-policy sampling; see Figure 3. Applying feature normalization with $\alpha = \beta = \frac{1}{2}$ to Baird's Counterexample (Figure 4) results in correct convergence

We can also look at other combinations of (α, β). In Figure 5 the pair $(0, 1)$ converges, albeit with some high-frequency oscillations. Figure 6 plots the learning dynamics for a combination $(0.1, 0.1)$ which is not along the $\beta = 1 - \alpha$ line, showing an interesting pattern: it converges *very* slowly, exhibiting low frequency oscillations along the way.

4.3.2 Phase Plots

Having seen that subtracting α, β combinations from the features can make learning stable, we can test which combinations are desirable. To do so, we generate *phase*

[3]The weights vector is initialized as $\mathbf{w} = (1, 1, 1, 1, 1, 1, 10, 1)^\top$

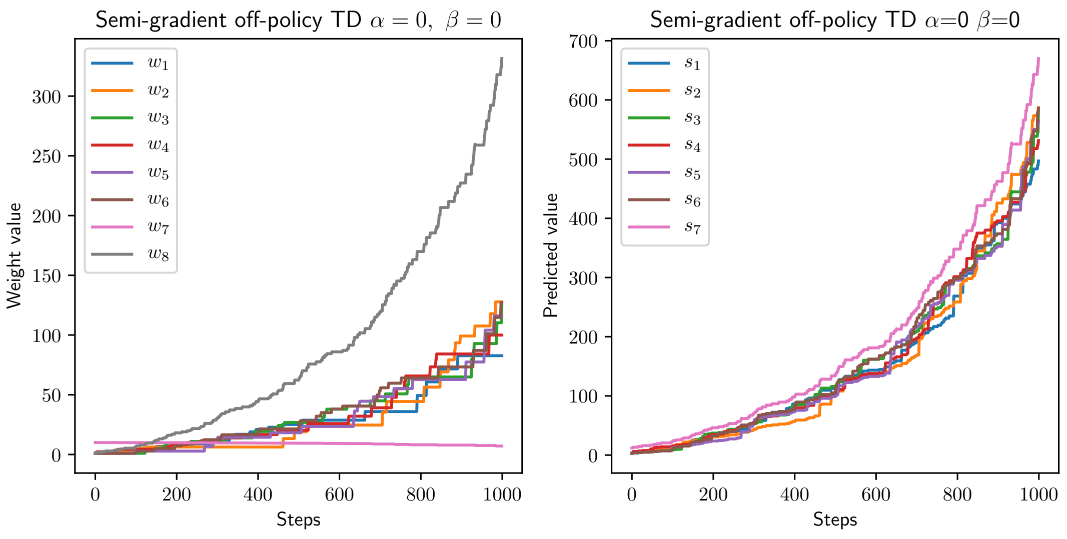


Figure 3: On the left, we show the evolution of weights $\mathbf{w}$ using TD(0) on Baird's Counterexample, corresponding to our $\alpha = 0, \beta = 0$. On the right, we can see evolution of predicted state values. Despite the zero reward, both are diverging to infinity. The plots are made with the accompanying code of Sutton and Barto (2018).

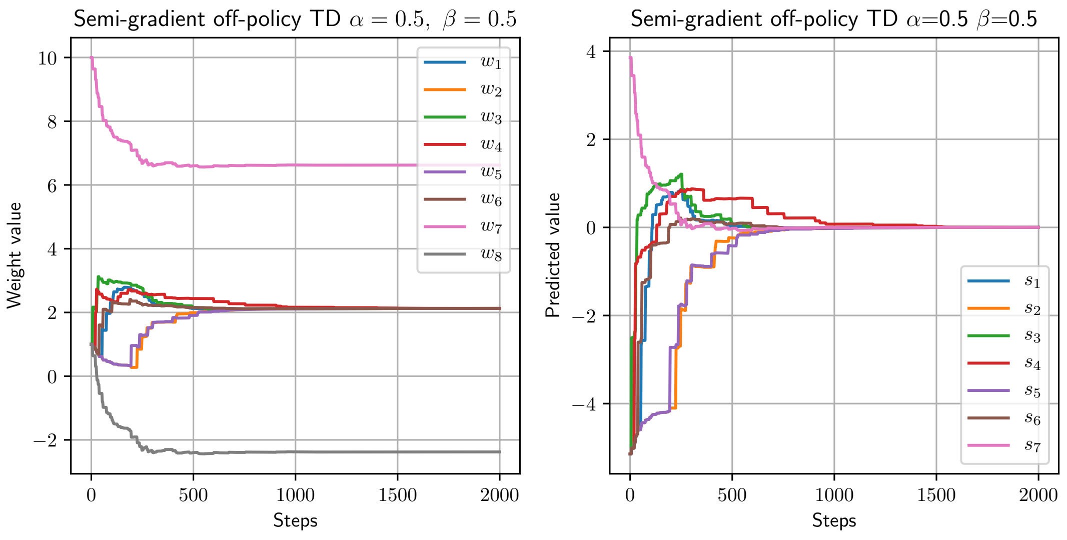


Figure 4: In the left plot, $\alpha = 0.5, \beta = 0.5$ makes the weights $\mathbf{w}$ converge. On the right, the predicted state value function correctly converges to zero everywhere.

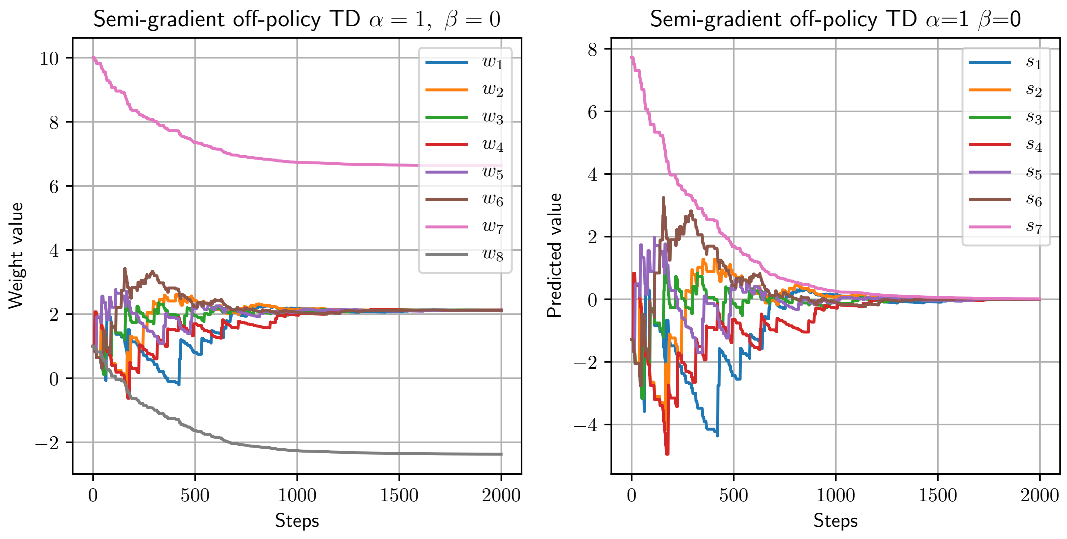


Figure 5: Convergence of Baird's Counterexample with $\alpha = 1, \beta = 0$.

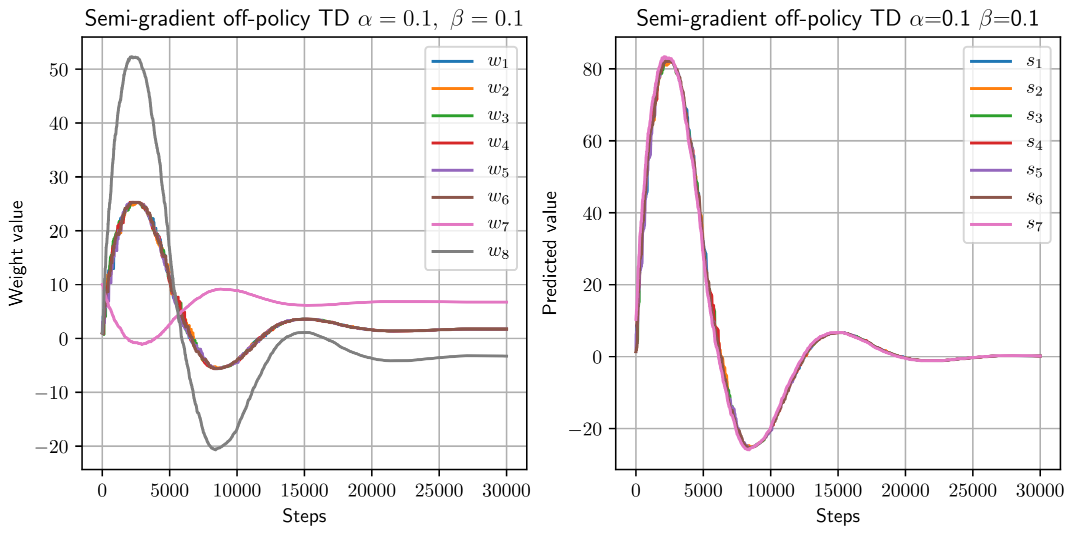


Figure 6: Very slow convergence of Baird's Counterexample with $\alpha = 0.1, \beta = 0.1$.

plots of the value function convergence over the state space, for a grid of α, β values.

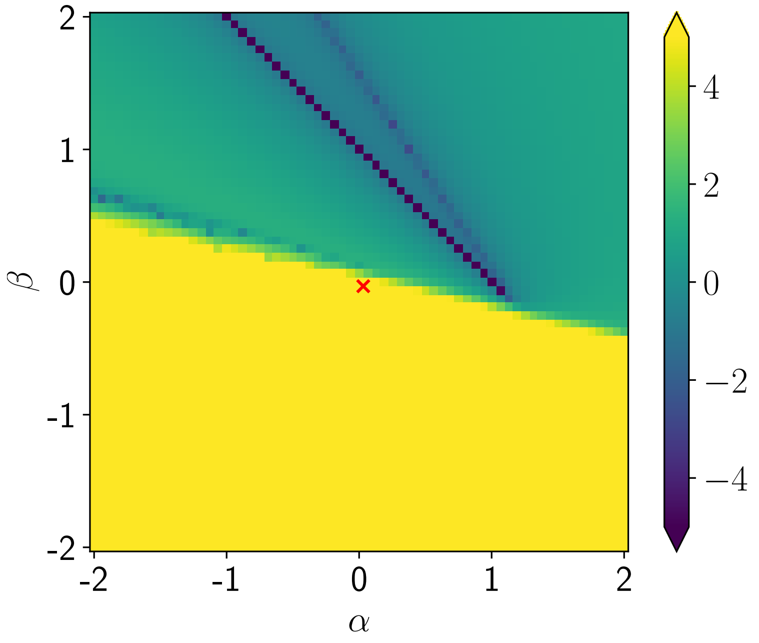


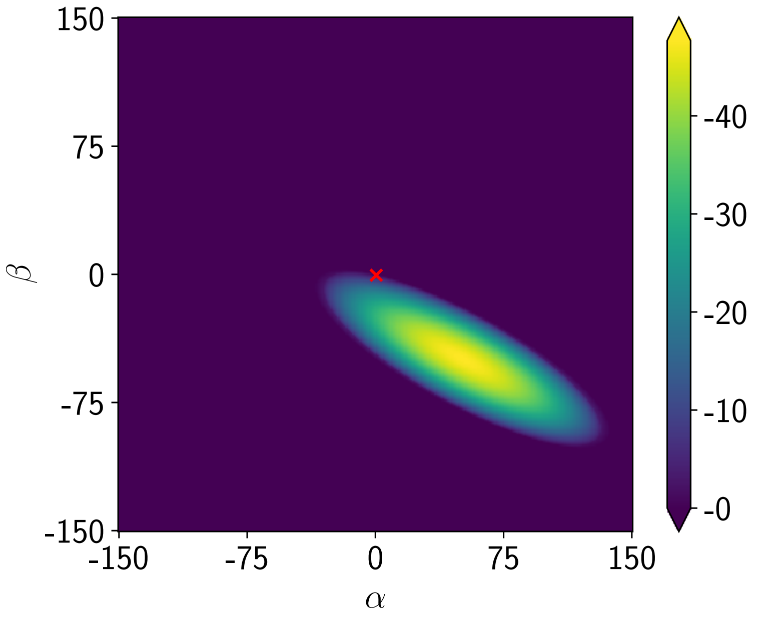
Figure 7: A plot of the value function convergence, marked as $log(|\bar{V}|)$, after running expected TD learning on Baird's Counterexample for different locations on the α, β plane. Each pixel is one run of training for 10^5 iterations, with discount $\gamma = 0.99$ and step size $\eta = 10^{-3}$. The symbol $\times$ marks $\alpha = \beta = 0$. Lower is better.

Some combinations diverge, and some converge to the correct value of zero; we can accommodate the vastly different scales by taking the logarithm of the absolute values of the final predictions. For Baird's Counterexample, Figure 7 depicts the phase plot. By inspecting this plot and the final V values, we make two key observations:

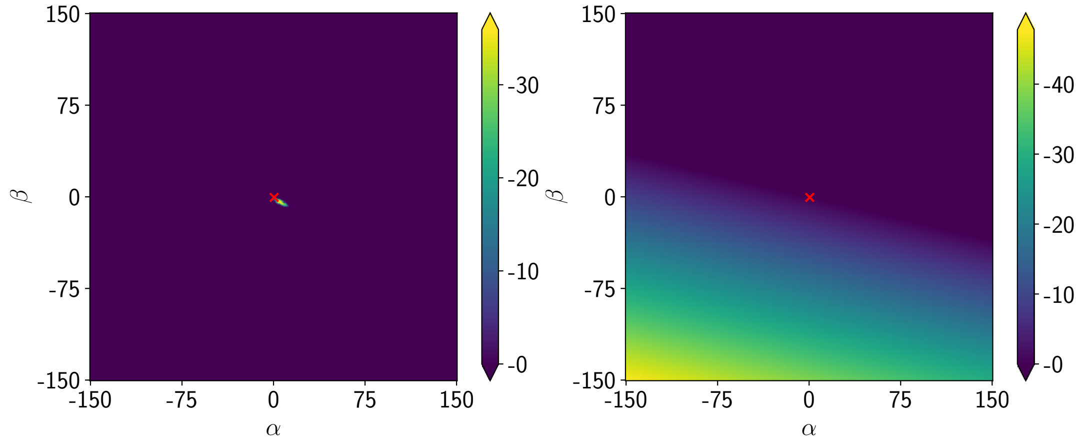
- There exists a "phase" boundary marking a transition from yellow region of divergence to the green region of stability.
- The line $\alpha + \beta = 1$ in the stable zone, converges an order of magnitude faster.

Interestingly, the special line meets the edge of the stability boundary; this point happens to be $\alpha = 1, \beta = 0$.

We can drop the costly optimization and instead sort the eigenvalues of $\mathbf{A}_{\alpha,\beta}$ by their real parts, to directly plot the smallest of them. Figure 8 does this for a much



(a)



(b)

Figure 8: Minimum eigenvalue of $\mathbf{A}_{\alpha,\beta}$ on different locations on the α, β plane. The values have been negated to match the previous color scheme, so that negative eigenvalues are yellowish. (a) With $\gamma = 0.99$. (b) With $\gamma = 0.9$ and $\gamma = 1$ on the left and right respectively.

bigger range of α and β. This reveals an interesting pattern: the unstable zone is shaped like an oval. Changing the discount factor to be smaller ($\gamma = 0.9$) makes the unstable oval shrink, which is intuitively correct because it brings the problem closer to direct linear regression. Upping it to $\gamma = 1$ makes the oval grow to become an entire half-plane, meaning that the dynamics of undiscounted TD(0) are still stable for some feature encodings! Unfortunately this does not tell us much about the line of fast convergence.

4.4 Positive Semi-Definiteness of the Key Matrices

To see where the blue line originates from, recall that the proof for on-policy TD showed that the key matrices $\mathbf{K}$ and $\mathbf{A}$ were positive (semi) definite. Figure 9 shows four different MDPs with 15 states, 8 features, and $\gamma = 1$. $\mathbf{D}_\mu$, $\mathbf{P}_\pi$, and the features are randomly generated. The figure are plots of where $\mathbf{A}_{\alpha,\beta}$ and the corresponding $\mathbf{K}_{\alpha,\beta}$ are p.s.d. As expected, in these examples $\mathbf{K}_{\alpha,\beta}$ always satisfies this desirable property on the special line $\alpha + \beta = 1$ for $\beta > 0$. We can see that whenever $\mathbf{K}_{\alpha,\beta}$ is p.s.d., $\mathbf{A}_{\alpha,\beta}$ has to be so too; for the projected key matrix this appears to hold true in the vicinity of the line as well. This could explain why our feature normalization scheme turns off-policy TD(0) convergent just like the on-policy version.

4.4.1 Rank-reduction property of $\alpha + \beta = 1$ normalization

Having observed this phenomenon in the plots, it would be interesting to see what properties the normalization imparts to the big key matrix.

Using Equation (16) and Equation (17), we can expand the full (and rather messy) expression of the modified key matrix as:

$$\begin{aligned}\mathbf{K}_{\alpha,\beta} &= \mathbf{D}_\mu(\mathbf{I} - \gamma\mathbf{P}_\pi) \\ &\quad - (1-\gamma)\mathbf{D}_\mu\mathbf{1}\mathbf{1}^\top\mathbf{D}_\mu(\alpha\mathbf{I} + \beta\mathbf{P}_\pi) \\ &\quad - (\alpha\mathbf{I} + \beta\mathbf{P}_\pi^\top)\mathbf{D}_\mu\mathbf{1}\mathbf{1}^\top\mathbf{D}_\mu(\mathbf{I} - \gamma\mathbf{P}_\pi) \\ &\quad + (1-\gamma)(\alpha\mathbf{I} + \beta\mathbf{P}_\pi^\top)\mathbf{D}_\mu\mathbf{1}\mathbf{1}^\top\mathbf{D}_\mu(\alpha\mathbf{I} + \beta\mathbf{P}_\pi)\end{aligned} \tag{18}$$

Analogously to the on-policy study, we check the row and column sums of this matrix, which after a few steps of derivation simply to the following:

$$\mathbf{K}_{\alpha,\beta}\mathbf{1} = (1-\gamma)\left(1-(\alpha+\beta)\right)\left(\mathbf{I}-\left(\alpha\mathbf{I}+\beta\mathbf{P}_{\mu}^{\top}\right)\right)\mathbf{D}_{\mu}\mathbf{1} \tag{19}$$

$$\mathbf{1}^{\top}\mathbf{K}_{\alpha,\beta} = \mathbf{1}^{\top}\mathbf{D}_{\mu}\left(1-(\alpha+\beta)\right)\left(\mathbf{I}-\gamma\mathbf{P}_{\pi}-(1-\gamma)\left(\alpha\mathbf{I}+\beta\mathbf{P}_{\pi}\right)\right) \tag{20}$$

Substituting $\alpha+\beta=1$ into the above formulas makes the expressions zero; therefore the modified big key matrix has the surprising property that its rows and columns both sum to 0. This means that $\mathbf{1}$ is a left and right eigenvector with eigenvalue zero, and $\mathbf{K}_{\alpha,\beta}$ is no longer full-rank.

It would appear that α, β normalization has a tendency to knock out the very eigenvalues which cause indefiniteness and unstable $\mathbf{A}$ matrices. While this indeed seems encouraging, it is certainly not the complete picture; we have been unable to establish diagonal dominance, and in fact there exist some counterexample MDPs to our normalization scheme. We have found that with randomly generated $\mathbf{D}_{\mu}$ and permutation matrices $\mathbf{P}$ (e.g. shuffled identity matrices) and smooth relaxations thereof, positive definiteness is sometimes violated even on the special line, and TD(0) still remains unstable there. Curiously, for those same contrived MDPs, ensuring that the number of states N and the number of features n have a large discrepancy (i.e. $N \gg n$) seems to turn those problems convergent!

This has identified interesting phenomena, even without considering the effect of feature scaling. Future work will perhaps find an additional ingredient (or a subset of MDPs) that could make off-policy TD(0) provably converge.

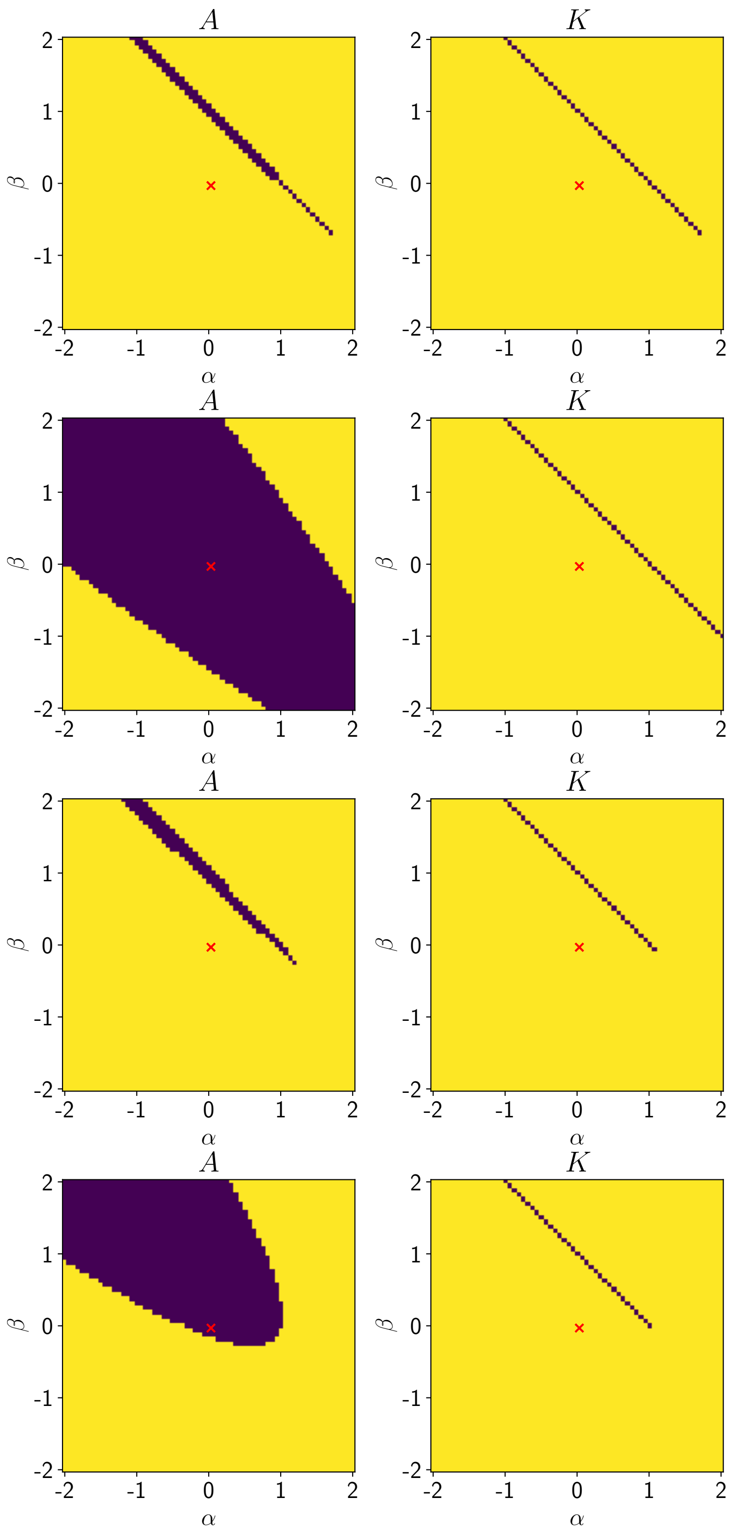


Figure 9: Positive Semi-Definiteness plots for $\mathbf{A}_{\alpha,\beta}$ and $\mathbf{K}_{\alpha,\beta}$, on four randomly generated MDPs. The special blue line corresponds to both matrices being p.s.d, unlike anywhere else in the plane.

5 CrossNorm for Deep Reinforcement Learning

In the Chapter 4 we saw that the $\alpha + \beta = 1$ normalization scheme often tends to make off-policy semi-gradient TD(0) convergent. We now turn our attention to Deep Reinforcement Learning and larger-scale problems. In this chapter we will extend the normalization scheme to deep neural networks and compare the performance with existing approaches.

The focus of our experimentation will be on DDPG (Lillicrap et al., 2015) and its state-of-the-art derivative TD3 (Fujimoto et al., 2018), both of which are actor-critic methods involving a policy evaluation step.

5.1 From BatchNorm to CrossNorm

In DDPG, the the action-value function $Q^{\pi}_{\boldsymbol{\theta}_Q}(s, a)$ is represented as a deep neural network that consumes a concatenation of the state and action vectors as input, and emits a single scalar value as its prediction. The hidden layers within it have highly nonlinear activations.

It is very difficult to do an analysis of how such neural networks can be better conditioned; nevertheless it is common practice to treat them as cascades of data features by independently applying standardization to each layer with Batch Normalization Ioffe and Szegedy (2015). Our normalization scheme can also be performed by carefully applying BatchNorm to every layer, including the inputs. Note that recentering can artificially reduce the model expressivity, so BatchNorm also includes learnable scale and shift vectors that are applied right after the standardization.

A common mistake in using BatchNorm with DDPG is that batches of (s, a) and $(s', \pi_{\boldsymbol{\theta}_\pi}(s'))$ — which are sampled from two different distributions — are processed in succession while keeping BatchNorm in *train mode*; this is wrong because then very different batch statistics will be used to re-center the activations in each forward pass. The "correct" way to use BatchNorm would be to set the critic to *test mode* when performing the forward pass for $(s', \pi_{\boldsymbol{\theta}_\pi}(s'))$, ensuring that both passes will re-center by the same moments of (s, a); this corresponds to $\alpha = 1, \beta = 0$.

We have seen from the phase plots that the BatchNorm setting of $\alpha = 1, \beta = 0$ often lies at the very edge of the stability boundary, and it is possible that due to sampling noise or policy updates the mean subtraction might cross over into the unstable region. For this reason, we propose a safer trick; the two batches could be concatenated as $\tilde{s} = [s, s']$ and $\tilde{a} = [a, \pi_{\theta_\pi}]$, then processed as a *single forward pass* of the batch $(\tilde{s}, \tilde{a})$ through the critic. As we are normalizating equally across the two distributions, we choose to call this $\alpha = \beta = \frac{1}{2}$ combination *Cross Normalization*. CrossNorm is also simpler to use, as it does not require switching between the two modes during TD learning.

5.2 Experiments

CrossNorm is a modular improvement that can be immediately incorporated into existing Deep RL algorithms that learn Q value estimators. We apply it to DDPG and TD3 with the aim of making them more stable. For evaluation we compare our improvement against their baselines on four MuJoCo (Todorov et al., 2012) continuous control tasks provided by OpenAI Gym (Brockman et al., 2016) as shown in Figure 12. We use the code provided by Fujimoto et al. (2018) which contains the authors' TD3 implementation and a highly performant version of DDPG.

5.2.1 Fixed-Buffer Experiments

To motivate using CrossNorm here, we first test if the stabilizing properties of normalization can hold in more complex MDPs with continuous state and action spaces. We take transitions from a large Walker2d experience buffer — gathered by a different agent — and estimate approximate $\mathbf{A}_{\alpha,\beta}$ matrices by sampling a large batch of size 20,000. The feature vectors are extracted from the last layer of a randomly initialized 3-layer neural network with LeakyReLU Maas (2013) activations. The

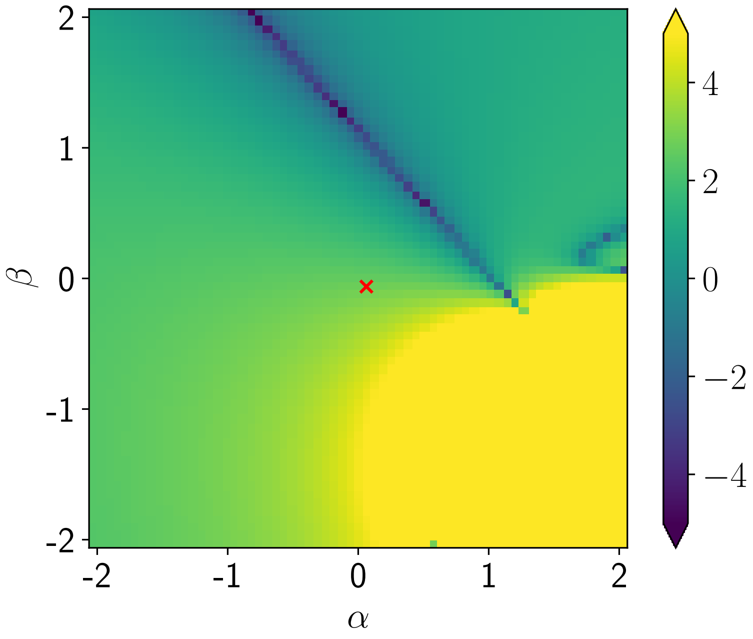


Figure 10: α, β phase plot from experience gathered by a Walker2d-v2 agent. The phenomenon with the ellipse and the fast-converging line persists in more complex MDPs. The "convergence" is established by setting the rewards to zero and then performing expected TD(0) updates on the last layer; the previous layers are frozen.

resulting phase plot Figure 10 exhibits the very properties that we seek, albeit for the case of training just the last layer.

Next, we take the aforementioned experience memory of 10^6 transitions and perform *policy improvement* by training a full DDPG agent from scratch to solve Walker2d. The baseline DDPG agent without target networks quickly diverges to infinity. But the incorporation of CrossNorm indeed stabilizes off-policy TD even with complex

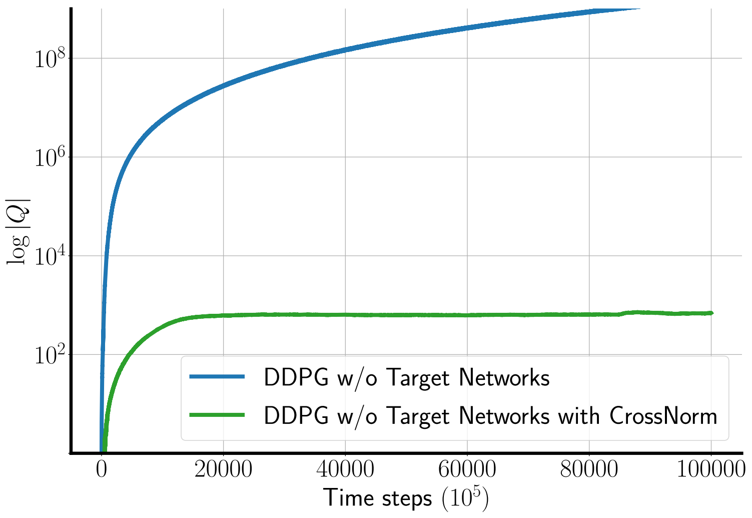
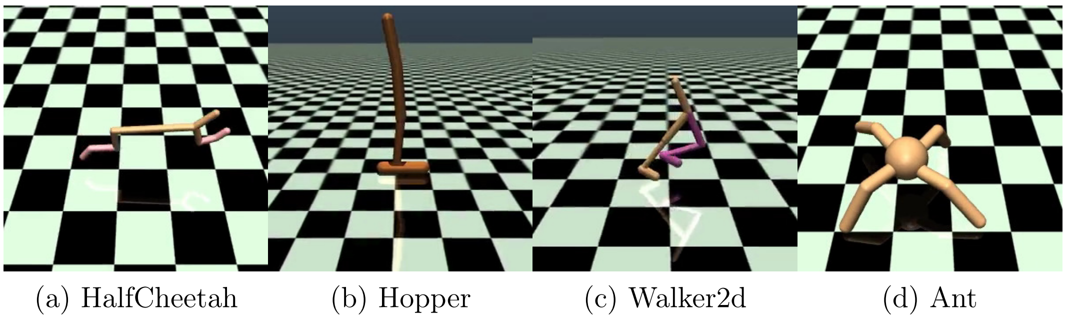


Figure 11: DDPG agents training on a fixed experience memory with 10^6 samples. Due to vastly different scales, the plots are the log of the averaged Q value predictions for a batch size of 1024. DDPG diverges without target networks; CrossNorm makes it converge.

MDPs, neural networks, and policy improvement — which entails an ever-changing $\mathbf{P}_\pi$ — and its Q value estimates soon converge to a stable value.

5.2.2 Concurrent Training



(a) HalfCheetah (b) Hopper (c) Walker2d (d) Ant

Figure 12: MuJoCo simulation environments.

Lastly, we do standard concurrent training wherein the agent gathers samples from the environment whilst learning. We compare the baseline DDPG algorithm against the CrossNorm variant, with and without target networks (Figure 13). No changes are made other than a new set of hyperparameters; while the baseline uses the Adam optimizer (Kingma and Ba, 2014) and a learning rate of 10^{-3}, we use RMSprop (Tieleman and Hinton, 2012) with a learning rate of 10^{-4}. The default batch size of 100 is used.

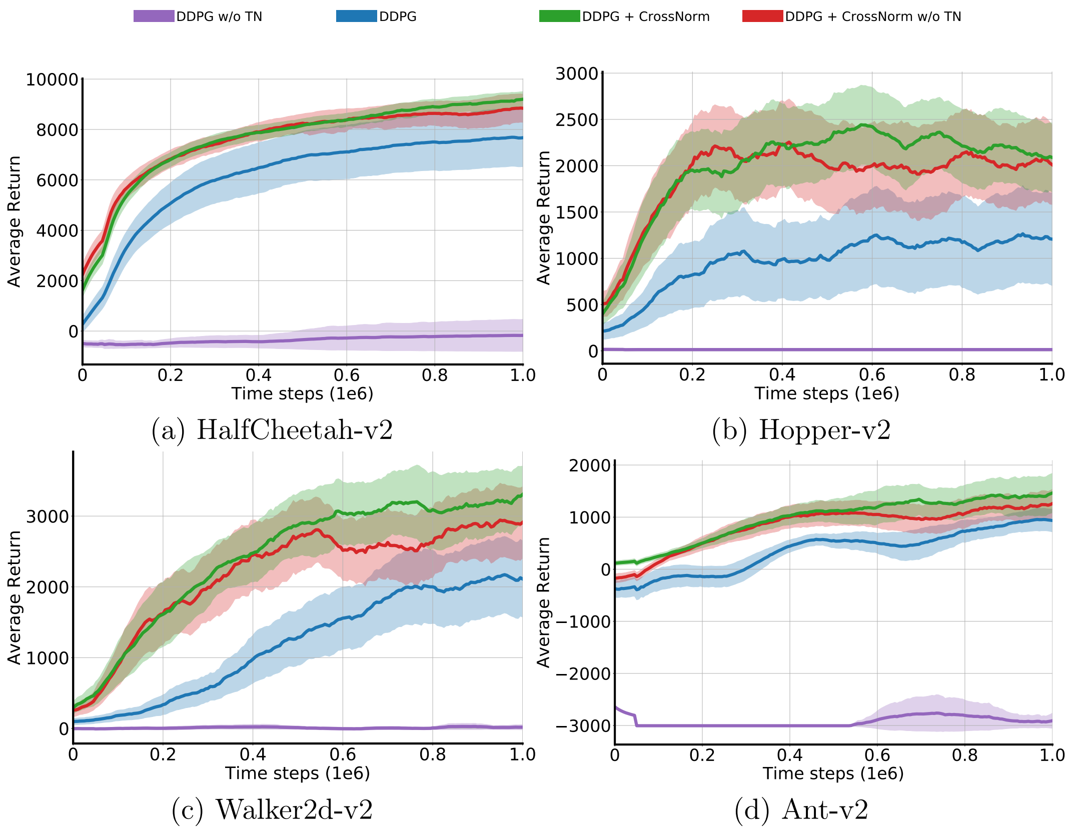
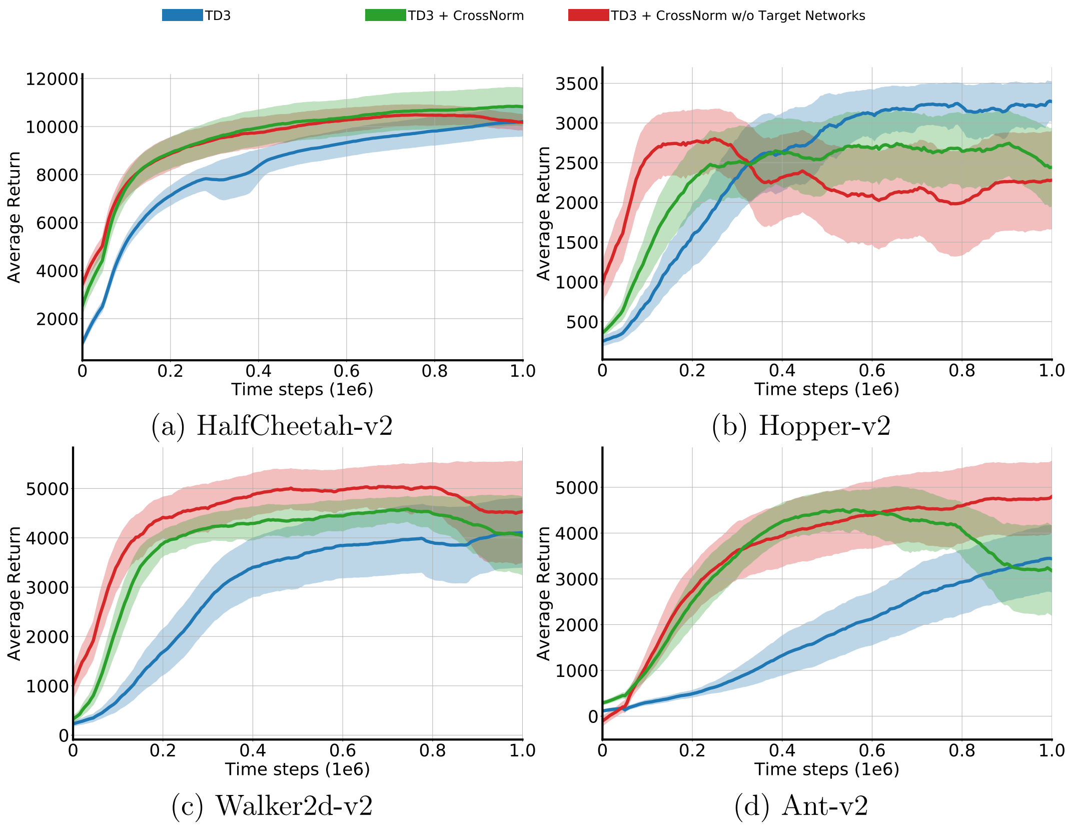


Figure 13: Comparisons for DDPG with CrossNorm. The curves are averages and halved standard deviations over 10 runs with unique random seeds.

In the TD3 evaluations (Figure 14), we increase the batch size to 256 for all the algorithms and use LeakyReLU activations (Maas, 2013) for the CrossNorm variants, and again use RMSprop instead of Adam. The same default learning rate of 10^{-3} is used in all runs.

In both sets of experiments, we find that the CrossNorm versions perform far better than the DDPG baseline. With CrossNorm, we are able to train DDPG and TD3 agents without relying on target networks. There is a particularly visible lead early on in the TD3 comparisons due to the faster credit assignment enabling faster policy improvement; the sample efficiency is very high.

With TD3 and CrossNorm, we notice a certain drop in performance later on in the training. From inspections of Q value predictions we do not observe any divergence; the policy suddenly degrades and the Q predictions correctly track the performance of the policy. As this issue is not seen with DDPG and CrossNorm, we suspect that this is a consequence of the special dual critic configuration that favors underestimation bias.



(a) HalfCheetah-v2

(b) Hopper-v2

(c) Walker2d-v2

(d) Ant-v2

Figure 14: Comparisons for TD3 with CrossNorm. The curves are averages and halved standard deviations over 10 runs with unique random seeds.

6 Conclusion

6.1 Summary

Reinforcement Learning has a famous problem called *the deadly triad* which states that a combination of function approximation, bootstrapping, and off-policy learning risks divergence for iterative learning algorithms like TD(0). While attempts to remedy the problem by developing alternative TD algorithms exist, TD(0) remains desirable for its simplicity and ease of use with deep neural networks.

This thesis studies the dynamics of approximate TD learning and the conditions under which it can diverge. A major cause is the difference in data distributions caused by off-policy learning. We propose a normalization method that takes into account these differences, called *CrossNorm*. Empirically, the results show that this prevents divergence on Baird's Counterexample and many synthetic MDPs.

This method was applied to two continuous control Deep Reinforcement Learning algorithms — DDPG and TD3 — and it improved the learning of complex control policies in standard MuJoCo tasks. The incorporation of CrossNorm allowed the training of agents without requiring target networks. To the best of our knowledge, this is the first time that an off-policy Deep RL algorithm like DDPG has been stably trained without target networks; and with better performance.

6.2 Future Work

In this thesis we considered normalization in the very narrow sense of mean subtraction. This yielded promising results in the linear TD(0) case, however it stabilizes many but not all MDPs. We derived an interesting property of the modified big key matrix, but were still unable to establish diagonal dominance.

It is possible that we are missing an extra ingredient in our analysis. Future lines of research can include identifying classes of MDPs which cannot be stabilized with mean subtraction, or the effect of scaling features by their standard deviations.

We have seen that CrossNorm in Deep RL gives good results; it works well over a wide range of MDPs as policy improvement changes the $\mathbf{P}_\pi$ matrix all the time. This suggests that perhaps we could simply get a better behaved function class by incorporating *learnable* feature shifts in addition to the mean recentering.

If any of these ideas completes the puzzle, a convergence proof for off-policy approximate TD(0) might be within reach.

In the case of TD3 with CrossNorm, it would be worthwhile to pinpoint a cause for the observed collapse in performance in the latter stages of training; as it occurs both with and without target networks, it appears to not be due to divergence. A possibility is that this might be an issue with clipped dual critics, which when paired with saturating policies can cause an underestimation bias, possibly providing a misleading training signal to the actor.

Acknowledgments

Firstly, I would like to thank Prof. Thomas Brox for giving me total freedom to choose the topic for my research; and for listening to each idea I proposed before settling on this one.

Max and Artemij, thank you for the very engaging discussions we've had together over the past year or so, and for writing a paper on this topic with me. The whiteboard sessions in pursuit of the ill-fated proof will not be forgotten. And neither will the matplotlib magic.

I would also thank Thorsten for setting up the spare office laptop for me when mine wouldn't start anymore, days before the deadline.

Thanks to all the students in the LMB pool, for all the chit-chat and career advice. And thanks to all my friends in Freiburg who made my stay here colorful.

I thank my parents for always being supportive, despite the long distance, and pushing me to keep moving when I was stuck.

Lastly, and most of all, I want to thank my wife Aakriti; without whose infinite patience, care, and prodding I would not have finished this.

Bibliography

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529 EP –, Feb 2015.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al.

Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

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

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1st edition, 1996. ISBN 1886529108.

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

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. URL `http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12`.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 1582–1591. JMLR.org, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. 2016.

Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):2603–2631, 2016.

Ashique Mahmood. *Incremental Off-policy Reinforcement Learning Algorithms*. PhD thesis, University of Alberta, 2017.

Harold Kushner and G George Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

Semyon Aranovich Gershgorin. Uber die abgrenzung der eigenwerte einer matrix. 1931.

Shalabh Bhatnagar, Doina Precup, David Silver, Richard S Sutton, Hamid R Maei, and Csaba Szepesvári. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, pages 1204–1212, 2009.

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

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.