Learning Dynamics and Generalization in Deep Reinforcement Learning

Clare Lyle[1] Mark Rowland[2] Will Dabney[2] Marta Kwiatkowska[1] Yarin Gal[1]

Abstract

Solving a reinforcement learning (RL) problem poses two competing challenges: fitting a potentially discontinuous value function, and generalizing well to new observations. In this paper, we analyze the learning dynamics of temporal difference algorithms to gain novel insight into the tension between these two objectives. We show theoretically that temporal difference learning encourages agents to fit non-smooth components of the value function early in training, and at the same time induces the second-order effect of discouraging generalization. We corroborate these findings in deep RL agents trained on a range of environments, finding that neural networks trained using temporal difference algorithms on dense reward tasks exhibit weaker generalization between states than randomly initialized networks and networks trained with policy gradient methods. Finally, we investigate how post-training policy distillation may avoid this pitfall, and show that this approach improves generalization to novel environments in the ProcGen suite and improves robustness to input perturbations.

1. Introduction

The use of function approximation in reinforcement learning (RL) faces two principal difficulties: existing algorithms are vulnerable to divergence and instability (Baird, 1993), and value estimates that do converge tend to generalize poorly to new observations (Zhang et al., 2018). Crucial to both of these difficulties is the concept of *interference*, the degree to which an update to an agent's predictions at one state influences its predictions at other states. Function approximation schemes with weaker interference, such as those induced by tabular value functions or tile coding schemes, have been shown empirically to produce more stable behaviour and faster convergence in value-based algorithms on a number of classic control domains (Ghiassian et al., 2020). However, such schemes by construction require treating the value functions for different states independently, limiting the function approximator's potential for generalization.

Deep RL algorithms are notoriously prone to overfitting to their training. environment's observations and dynamics (Lewandowski, 2020; Farebrother et al., 2018; Cobbe et al., 2021; Zhang et al., 2018). While many prior works have sought training methodologies to improve generalization (Igl et al., 2019; Raileanu et al., 2021), the source of the relative tendency of deep RL methods to overfit to their training distribution remains under-explored. This work studies a mechanism to explain *why* value-based deep RL agents tend to generalize poorly to new environments and observations. One avenue that we will draw on in particular is the study of agents' *learning dynamics*, which can reveal insights into not just the convergence of algorithms, but also into the trajectory taken by the agent's value function.

In this work, we study how interference evolves in deep RL agents. Our primary contributions will be twofold: first, to provide a rigorous theoretical and empirical analysis of the relationship between generalization, interference, and the dynamics of temporal difference learning; second, to study the effect of distillation, which avoids the pitfalls of temporal difference learning, on generalization to novel environments. Towards this first contribution, we extend the analysis of Lyle et al. (2021) to show that the dynamics of temporal difference learning accelerate convergence along non-smooth components of the value function first, resulting in implicit regularization towards learned representations that generalize weakly between states. Our findings present an explanation for widely-observed vulnerability of value-based deep RL agents to overfit to their training observations (Raileanu and Fergus, 2021; Zhang et al., 2018).

We then evaluate whether these findings hold empirically across a range of popular deep RL benchmarks. We measure interference by constructing a summary statistic which evaluates the extent to which optimization steps computed for one state influence predictions on other states, which we call the *update rank*. We find that value-based agents trained with temporal difference (TD) methods learn representations with weak interference between states, performing

[1]Department of Computer Science, University of Oxford [2]DeepMind. Correspondence to: Clare Lyle <clare.lyle@cs.ox.ac.uk>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

updates similar to those of a lookup table, whereas networks trained with policy-gradient losses learn representations for which an update on one state has a large effect on the policy at other states. Finally, we show that post-training policy distillation is a cheap and simple approach to improve the generalization and robustness of learned policies.

2. Background

We focus on the reinforcement learning problem, which we formalize as a Markov decision process (MDP) (Puterman, 1994). An MDP consists of a tuple $\langle \mathcal{X}, A, R, P, \gamma, \mathcal{X}_0 \rangle$, where $\mathcal{X}$ denotes the set of states, A the set of actions, $R : \mathcal{X} \times A \to \mathbb{R}$ a reward function, $P : \mathcal{X} \times A \to \mathscr{P}(\mathcal{X})$ a possibly stochastic transition probability function, $\gamma \in [0, 1)$ the discount factor, and $\mathcal{X}_0$ the initial state. In reinforcement learning, we seek an optimal policy $\pi^* : \mathcal{X} \to \mathscr{P}(A)$ which maximizes the expected sum of discounted returns from the initial state.

Value-based reinforcement learning seeks to model the action-value function

$$Q^{\pi^*}(x, a) = \mathbb{E}[\sum_{t \geq 0} \gamma^t R_t(x_t, a_t) | x_0 = x, a_0 = a] \quad (1)$$

as a tool to learn an optimal policy. Given a policy π, we leverage a recursive expression of the value function as the fixed point of the policy evaluation Bellman operator, defined as follows

$$(T^\pi Q)(x, a) = \mathbb{E}_{P(x'|x,a), \pi(a'|x')}[R(x, a) + \gamma Q(x', a')]. \quad (2)$$

Temporal difference learning (Sutton, 1988) performs updates based on sampled transitions (x_t, a_t, r_t, x'_t), leveraging a stochastic estimate of the Bellman targets.

To find an optimal policy, we turn to the control setting. We let the policy π_t, used to compute the target, be greedy with respect to the current action-value function Q_t. This results in updates based on the Bellman *optimality* operator

$$(T^* Q_t)(x, a) = \mathbb{E}[R(x, a) + \gamma \max_{a'} [Q^{\pi_t}(x', a')]] \ . \quad (3)$$

In control problems, Q-learning (Watkins, 1989; Watkins and Dayan, 1992) is a widely-used stochastic approximation of the Bellman optimality operator. When a function approximator Q_θ, with parameters θ, is used to approximate Q^π, as in deep reinforcement learning, we perform semi-gradient updates $f(\theta)$ of the following form, where $a^* = \arg\max_a (Q_\theta(x_{t+1}, a))$.

$$f(\theta) = (\nabla_\theta Q_\theta)[r_t + \gamma Q_\theta(x_{t+1}, a^*) - Q_\theta(x_t, a_t)] \quad (4)$$

Policy gradient methods (Sutton et al., 2000) operate directly on a parameterized policy π_θ. We let d^π denote the stationary distribution induced by a policy π over states in the MDP. Policy gradient methods aim to optimize the parameters θ of the policy so as to maximize the objective $J(\pi_\theta) = \mathbb{E}_{x \sim d^{\pi_\theta}} \sum_a \pi_\theta(a|x) Q^{\pi_\theta}(x, a)$. This objective can be maximized by following the gradient

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{x_t, a_t}[\nabla_\theta \log \pi_\theta(a_t|x_t) Q^{\pi_\theta}(x_t, a_t)] \ . \quad (5)$$

Variations on this learning rule include *actor-critic* methods (Konda and Tsitsiklis, 2000), which use a baseline given by a value-based learner to reduce update variance, and trust-region based methods, such as TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017).

Generalization arises in reinforcement learning in the context of solving new reward functions (Dayan, 1993), and in the case of large observation spaces or procedurally-generated environments, where some degree of generalization to new observations is necessary in order to obtain good performance at deployment (Kirk et al., 2021). We will be concerned in this paper with the generalization gap incurred by a policy learned on a set of environments $\mathcal{E}_{X_{\text{train}}}$ on the *test* environment.

$$\mathbb{E}_{\mathcal{E}_{X_{\text{train}}}, \pi}[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)] - \mathbb{E}_{\mathcal{E}_{X_{\text{test}}}, \pi}[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t)] \quad (6)$$

In general, $\mathcal{E}_{X_{\text{test}}}$ will be assumed to share some structure with $\mathcal{E}_{X_{\text{train}}}$. In large observation spaces, $\mathcal{E}_{X_{\text{test}}}$ may be equal to $\mathcal{E}_{X_{\text{train}}}$ with a different initial state distribution, while for multi-environment problems $\mathcal{E}_{X_{\text{train}}}$ may be homomorphic to $\mathcal{E}_{X_{\text{test}}}$ (Zhang et al., 2020). A necessary condition for generalization to new observations is that a parameter update computed on a training state changes the agent's policy on the unseen test states, a phenomenon we will refer to as *interference*. While interference is defined in many ways in the literature, we use it here to refer to the effect of an optimization step performed on the loss of one transition to change the agent's predicted value of other states in the environment, following a usage similar to Bengio et al. (2020).

3. Related work

Generalization in RL. A number of prior works have presented methods to quantify and improve generalization in reinforcement learning (Igl et al., 2019; Raileanu et al., 2021; Hansen and Wang, 2021; Laskin et al., 2020; Yarats et al., 2020; Wang et al., 2020; Cobbe et al., 2020; Kenton et al., 2019). The study of generalization in deep RL has focused principally on overfitting to limited observations (Song et al., 2019), and generalization to novel environments (Farebrother et al., 2018; Cobbe et al., 2019). Work on generalization in deep learning more broadly has shown that neural networks are biased towards 'simple' functions, for varying notions of simplicity (Pérez et al., 2019; Hochreiter and Schmidhuber, 1995; Fort et al., 2020; Izmailov et al., 2018; Farnia et al., 2020). The study of this bias in reinforcement learning tasks (Yang et al., 2022), has demonstrated that the bias of neural networks to smooth functions can harm value approximation accuracy in deep RL, and proposes tuning the scale of learnable Fourier features as one means of ensuring high-frequency components of the value function are captured, an approach also followed by Brellmann et al. (2021). Raileanu and Fergus (2021) further highlight that the process of learning a value function can induce overfitting, improving generalization to novel environments by decoupling value approximation and policy networks in actor critic architectures.

Interference and stability. Off-policy temporal difference learning is not guaranteed to converge in the presence of function approximation (Baird, 1993; Tsitsiklis and Van Roy, 1997), the setting under which deep RL algorithms are most commonly used. The key driver of instability is interference, which has been studied in settings ranging from bandits (Schaul et al., 2019) to deep RL (Fedus et al., 2020; Achiam et al., 2019). A number of approaches which specifically reduce the effect of a gradient update for state s on the target $V(s')$ have been shown to improve the stability and robustness of these methods (Ghiassian et al., 2020; Lo and Ghiassian, 2019). Many prior works have also endeavoured to define and analyze interference in deep RL (Liu et al., 2020b;a; Bengio et al., 2020), and to study its role in the stability of offline algorithms (Kumar et al., 2021). Similarly, some recent methods (Shao et al., 2020; Pohlen et al., 2018) include an explicit penalty which discourages gradient updates from affecting the target values.

4. Learning dynamics and generalization

This section will explore a tension between learning dynamics in neural networks, which tend to 'generalize-then-memorize' (Kalimeris et al., 2019), and the dynamics of temporal difference learning with tabular value functions, discussed in Section 4.1, which tend to pick up information about the value function's global structure only late in training. We go on to study how these learning dynamics may affect the structure of gradient updates in the function approximation setting in Section 4.2.

Eigendecomposition of transition operators. An important concept in our theoretical analysis will be that of the eigendecomposition of the environment transition matrix. We will follow the precedent of prior work in considering diagonalizable transition matrices, for which further discussion can be found in many excellent prior works (Machado et al., 2017; Stachenfeld et al., 2017; Mahadevan, 2005). The relationship between the smoothness of an eigenfunction and its corresponding value has been noted in prior work (Mahadevan, 2005; Mahadevan and Maggioni, 2007). However, previous discussion of this connection has defaulted to an intuitive notion of smoothness without providing an explicit definition. We provide a concrete definition of the smoothness of a function on the state space $\mathcal{X}$ of an MDP $\mathcal{M}$ in order to provide an unambiguous characterization to which we will refer throughout this paper.

Definition 4.1. *Given a function $V : \mathcal{X} \to \mathbb{R}$, MDP $\mathcal{M}$, and policy π, define its expected variation $\rho(V)$ as*

$$\rho(V) = \sum_{x \in \mathcal{X}} |V(x) - \mathbb{E}_{P^\pi(x'|x)} V(x')| \,. \tag{7}$$

We say V is smooth *if $\rho(V)$ is small.*

This expression reveals a straightforward relationship between the eigenvalue λ_i associated with a normalized eigenvector v_i and the smoothness of that eigenvector:

$$\sum_{x \in \mathcal{X}} |v_i(x) - \mathbb{E}_{P^\pi(x'|x)} v_i(x')| = \sum_{x \in \mathcal{X}} |(1-\lambda_i) v_i(x)| \tag{8}$$

In other words, the eigenvalue of an eigenvector precisely determines the variation of the eigenvector over the entire state space. If $\lambda = 1$, for example, then the eigenvector must be constant over the MDP, whereas if $\lambda = -1$, then we have $\mathbb{E}_{P^\pi(x'|x)}[V(x')] = -V(x)$ and the expected value fluctuates between extremes when stepping from one state to another. The *variance* over next-state values can in principle be large even for functions of low variation by our definition, though in our empirical evaluations (see e.g. Figure 2) smooth eigenvectors tended to also exhibit little variance. For our analysis of the *expected* updates performed by TD learning, we will find the smoothness of the expected updates to be a more useful quantity than the variance.

4.1. Tabular dynamics

We begin by studying the learning dynamics of tabular value functions. We consider a continuous-time approximation of the dynamics followed by the value function using different update rules. Our analysis will contrast Monte Carlo updates, which regress on the value function, with Bellman updates, which regress on bootstrapped targets and correspond to the expected update performed by TD learning. For simplicity, we will ignore the state visitation distribution; analogous resulst for non-uniform state-visitation distributions are straightforward to derive from our findings. We follow the approach of Lyle et al. (2021) in expressing the dynamics of Monte Carlo (MC) updates as a continuous-time differential equation

$$\partial_t V_t = V^\pi - V_t$$

where $V_t \in \mathbb{R}^{\mathcal{X}}$ is a function on the state space $\mathcal{X}$ of the MDP, resulting in the trajectory

$$V_t = \exp(-t)(V_0 - V^\pi) + V^\pi \,.$$

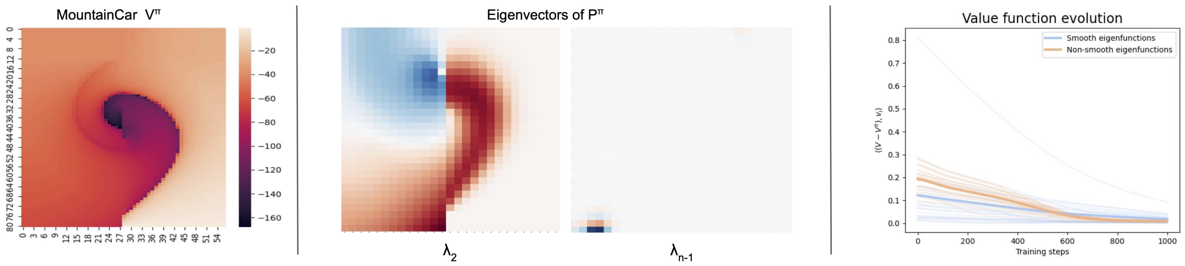

Figure 1. Left: value function of a near-optimal policy on MountainCar. States correspond to velocity (x-axis) and position (y-axis). Middle: eigenvectors associated with this policy computed for a discretization of the MountainCar state space. Right: value approximation error by eigenbasis coefficient along a trajectory generated by tabular TD updates with learning rate $\alpha = 0.1$ on the discretized MountainCar MDP. We compare 25 of the most-smooth eigenfunctions with 25 eigenfunctions corresponding to negative eigenvalues, and normalize the error by the magnitude of the projection of V^π onto the basis spanned by each set of vectors. Each transparent line corresponds to the dot product with a different eigenvector, while the solid lines show the mean over each subspace.

Intuitively, this corresponds to a 'straight line' trajectory where the estimated value function V_t converges to V^π along the shortest path in $\mathbb{R}^{\mathcal{X}}$. In practice, most deep RL algorithms more closely resemble temporal difference updates, which are expressed as

$$\partial_t V_t = -(I - \gamma P^\pi)V_t + R^\pi \tag{9}$$

$$V_t = \exp(-t(I - \gamma P^\pi))(V_0 - V^\pi) + V^\pi \,. \tag{10}$$

Under this decomposition, we can show that a predicted value function trained via TD learning will converge more slowly along smooth eigenvectors of P^π.

Observation 4.1. *Let P^π be real diagonalizable, with eigenvectors $v_1, \dots, v_{|\mathcal{X}|}$ corresponding to eigenvalues $\lambda_1 > \dots \geq \lambda_{|\mathcal{X}|}$, and let V_t be defined as in Equation 10. Write $V_t = \sum_{i=1}^{|\mathcal{X}|} \alpha_i^t v_i$ to express the value function at time t with respect to the eigenbasis $\{v_i\}$. Then the convergence of V_t to the value function $V^\pi = \sum_{i=1}^{|\mathcal{X}|} \alpha_i^\pi v_i$ can be expressed as follows:*

$$\alpha_i^t - \alpha_i^\pi = \exp(-t(1 - \gamma\lambda_i))(\alpha_i^0 - \alpha_i^\pi) \,.$$

The implications of Observation 4.1 on the learned value function depend to some extent on the eigendecomposition of V^π. If V^π is equal to the constant function, then we expect the high-frequency components of V_t to quickly converge to zero. If V^π puts weight on non-smooth eigenvectors, then early values of V_t may assign disproportionately high weights to these components relative to their contribution to V^π. In practice, value functions tend to exhibit a mixture of smooth and discontinuous regions, as can be seen in the illustration of a near-optimal value function in MountainCar in Figure 1. The corresponding expression of V^* with respect to the eigenbasis of P^π consequently places non-zero coefficients on eigenvectors corresponding to negative eigenvalues in order to fit this discontinuity, though its spectrum is dominated by smooth eigenfunctions. The following result highlights that non-smooth components of a predicted value function, while contributing relatively little to the Monte Carlo error, contribute disproportionately to the TD error, providing an incentive to fit these components early in training.

Theorem 4.1. *Let P^π be real diagonalizable with eigenvalues $\lambda_1 > \dots > \lambda_n$ and $(v_k)_{k=1}^n$ the corresponding (normalized) eigenvectors. Then for any value function V, the TD error $\mathrm{TD}(V_t) = \|V_t - T^\pi V_t\|^2$ can be bounded as as*

$$\begin{aligned} \|\mathrm{TD}(V_t)\|^2 &= \|T^\pi V_t - V_t\|^2 && (11)\\ &= \| \sum (1 - \gamma\lambda_i)(\alpha_i^\pi - \alpha_i^t)(v_i)\|^2 && (12)\\ &\leq \sum_{i=1}^n (\alpha_i^\pi - \alpha_i^t)^2 (1 - \gamma\lambda_i)^2 && (13) \end{aligned}$$

with equality when P^π has orthogonal eigenvectors.

Monte Carlo updates, which simply regress on the value function, give equal weight to errors along any component of the basis. These incentives provide some intuition for the different trajectories followed by Monte Carlo and TD updates: in order to minimize the TD loss, the value function must quickly become accurate along non-smooth components of the value function; however, its error due to smooth components such as the bias term of the function will have little effect on the loss and so converges more slowly. We provide an illustrative example of the relationship between the eigenvalue associated with a subspace and the convergence rate of the value function in that subspace in Figure 1.

4.2. Function approximation with kernels

Most function approximation schemes leverage the assumption that states which are close together in observation space are likely to have similar values; i.e. they encode a preference towards smooth (with respect to the observations) functions. This pushes against the tendency of temporal

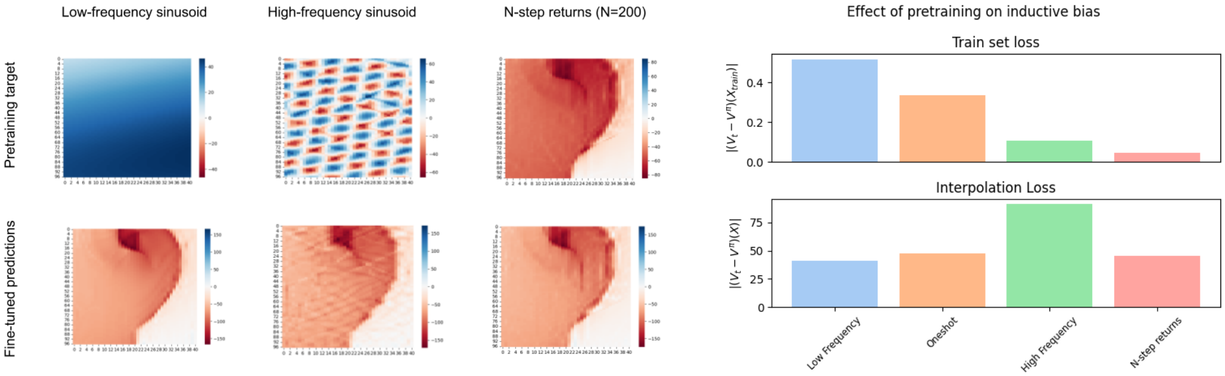

Figure 2. Networks trained to fit high-frequency target functions exhibit pathological interpolation properties when later fine-tuned on a value function. Left: visualization of pre-training targets (top) and final value estimate (bottom) after the pre-trained network is fine-tuned on the value function. Right: loss on the set of training states (top) and a finer-grained set of states which interpolate the training set (bottom) of each fine-tuned network.

difference updates to encourage V_t to fit the components of the value function with large variation first. To investigate this tension, we consider the *kernel gradient descent* regime.

Formally, a kernel is a positive definite symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. In our case, we will define $\mathcal{X}$ to be the state space of an MDP. Letting $\mathbf{x} \subseteq \mathcal{X}$, we denote by $\tilde{K}$ the (symmetric) matrix $K(\mathbf{x}, \mathbf{x})$ with entries $K(\mathbf{x}, \mathbf{x})_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$. Loosely speaking, a kernel encodes the similarity between two states, allowing us to incorporate certain forms of inductive bias into the value function approximation. Importantly, the similarity of two states under K does not inform us about how similar the states' initial values are, but rather how an update to the value function at one state influences the value of the other; in other words, in our setting it is a proxy for the *interference* between two states. Under kernel gradient descent, the trajectory of a function is defined in terms of a kernel K and the function-space gradient of a cost function. We can translate TD semi-gradient updates into the kernel gradient descent regime as follows:

$$\partial_t V_t = \tilde{K}((\gamma P^\pi - I)V_t + R^\pi) \,. \tag{14}$$

It is straightforward then to obtain analogous results as before on the convergence of V_t to V^π based on the eigendecomposition of the matrix $\tilde{K}(\gamma P^\pi - I)$ in cases where this matrix is positive definite, though many notable cases occur where this condition does not hold. This decomposition will not in general have a closed form in terms of the eigendecompositions of $\tilde{K}$ and P^π, but special cases have been studied in the setting of linear regression by Ghosh and Bellemare (2020) and can be related to kernel gradient descent straightforwardly as discussed in Appendix A.2. This setting also describes the dynamics of neural networks in the limit of infinite width (Jacot et al., 2018; Fort et al., 2020; Lee et al., 2020), which follow kernel gradient descent with respect to the neural tangent kernel.

A more interesting case occurs when we assume some states in the environment are not updated during training.

Theorem 4.2. *Let K be a kernel and π a fixed policy in an MDP with finite state space X. Let $X_{\text{train}} \subset \mathcal{X}$ be a subset of states in the support of π, $X_{\text{test}} = \mathcal{X} \setminus X_{\text{train}}$, and let V_t be a value trajectory obtained by applying kernel semi-gradient updates on the set X_{train} to some initial value function $V_0(X_{\text{train}})$ with kernel K. Let K_{all} be defined as*

$$K_{\text{all}} = K(X_{\text{train}}, X_{\text{train}}) \oplus K(X_{\text{test}}, X_{\text{train}}) \,. \tag{15}$$

Then the trajectory of V_t on the entire state space X will be as follows,

$$\partial_t V_t(X) = (K_{\text{all}})[(T^\pi V_t - V_t)(X_{\text{train}})] \,. \tag{16}$$

A full derivation is provided in Appendix A.2. These dynamics diverge notably from the standard kernel gradient descent regime in that changes to predictions on the test set can now influence the dynamics of V_t on the training set. A large $K(X_{\text{test}}, X_{\text{train}})$ implies that updates to the training set carry great influence over predictions on the test set, but at the cost of increasing asymmetry in K_{all} when viewed as an operator on $\mathbb{R}^{\mathcal{X}}$. In Appendix C.2 we illustrate how this asymmetry can harm stability in the case of a simple radial basis function kernel when the test states are used as bootstrap targets. Combining insights from 4.2 and Observation 4.1, we arrive at an intriguing conclusion: in the case of smooth kernels, the components of the value function most suitable to approximation via the kernel K are precisely those which appear in the value estimate of the training set only later in the trajectory. As a result, the kernel does not receive the necessary information to generalize accurately to new observations. This observation runs contrary to the standard kernel regression regime, where one argument in support of early stopping is that kernel gradient descent methods converge along smooth components fastest (Jacot et al., 2018). At the same time it is

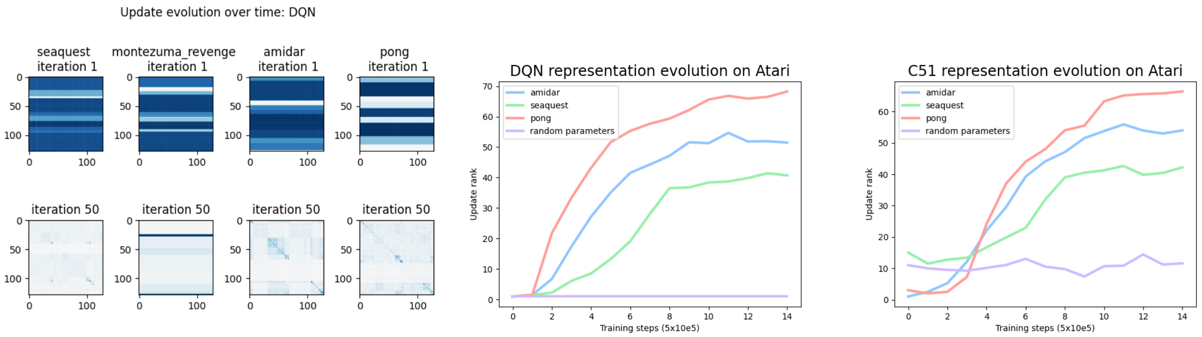

Figure 3. Agents trained on games from Atari. The networks initially exhibit low update rank, but after 50 iterations (5M frames of experience), the updates rank increases significantly. This is tracked in the bottom plots over the course of approximately 7M frames. Random parameters refers to the update rank obtained by a randomly initialized neural network.

an obvious effect of bootstrapping, which requires that the agent update its predictions several times in order to propagate information about the value function through the entire input space. This effect is illustrated in Figure 10 in Appendix C.2.

4.3. Non-linear function approximation

The linearized dynamics followed in the neural tangent regime fail to take into account the evolution of the network *features*. We now turn our attention toward the effect of temporal difference updates on the gradient structure of a function approximator by considering the second-order effects of TD semi-gradient updates under finite step sizes. We consider the system

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta V(\theta_t) \cdot [(\gamma P^\pi - I)V(\theta_t) + R^\pi] \quad (17)$$

which can be viewed as an Euler discretization of the dynamics described in Equation 10 in the presence of function approximation. We will use the notation $f(\theta_t)$ to refer to the semi-gradient update on parameters θ_t inducing value function V_{θ_t}, and write $\mathrm{TD}(\theta) = \frac{1}{2}\|V_\theta - \Box T^\pi V_\theta\|^2$, where the $\Box$ denotes a stop-gradient. This results in the following gradient flow:

$$\partial_t \theta_t = \nabla_\theta V(\theta_t) \cdot [(\gamma P^\pi - I)V(\theta_t) + R^\pi] \, . \quad (18)$$

Using the continuous-time system (18) to approximate the discrete-time system (17) will gradually accumulate increasing errors, proportional to $(\alpha n)^2$, as it does not take into account the effect of the discrete step size on higher-order gradients of V_θ. We apply a similar analysis to that of Barrett and Dherin (2021) and Smith et al. (2020) to understand the effect of the discrete learning dynamics on the gradient structure of V_θ itself. We let

$$f_1(\theta) = \frac{1}{2}\nabla_\theta \|\nabla_\theta \mathrm{TD}(\theta)\|^2 + \gamma(\nabla_\theta^\top V P^\pi \nabla_\theta V) f(\theta) \quad (19)$$

to obtain a second-order correction describing the effect of gradient descent on the gradient structure of the learned representation.

Observation 4.2 (Second-order dynamics)**.** *Let θ_t be defined by the discrete-time system (17) with step size α. Let $f_1(\theta)$ be defined as in (19). Let $\tilde{\theta}_t$ denote the trajectory obtained by following the dynamics:*

$$\partial_t \tilde{\theta}_t = f(\tilde{\theta}_t) + \frac{\alpha}{2} f_1(\tilde{\theta}_t) \, . \quad (20)$$

Then we have $\theta_n \approx \tilde{\theta}_{n\alpha} + O((n\alpha)^3)$, where $\tilde{\theta}_{n\alpha}$ denotes the value of $\tilde{\theta}_t$ at time $t = n\alpha$.

The form of f_1 constructed in Equation 19 consists of two terms: a semi-gradient norm penalty term with respect to the instantaneous TD error, and a gradient dot product term which discourages negative interference between nearby states. Since early in training the TD error will tend to be relatively discontinuous and less structured than the true value function (see e.g. Figure 2), the gradient norm penalty will have the effect of discouraging interference between states. Figure 3 illustrates how fitting highly variable targets early in training can discourage a neural network from smoothly generalizing between states. We observe a similar phenomenon in deep RL agents in Figure 3, where networks trained in dense-reward games (whose early TD targets will exhibit greater variation) exhibit weaker interference after training than they did at initialization.

In combination, the findings of this section suggest that the dynamics of temporal difference learning work to discourage interference between states in deep RL by fitting high-frequency components of the value function early in training while also encouraging flatness in parameter space. While this may result in more stable learning, as highlighted in Theorem 4.2, it has the double-edged effect of reducing the degree to which the network can generalize to novel observations. The following section will leverage these results to gain insight into deep RL agents trained in rich-observation environments.

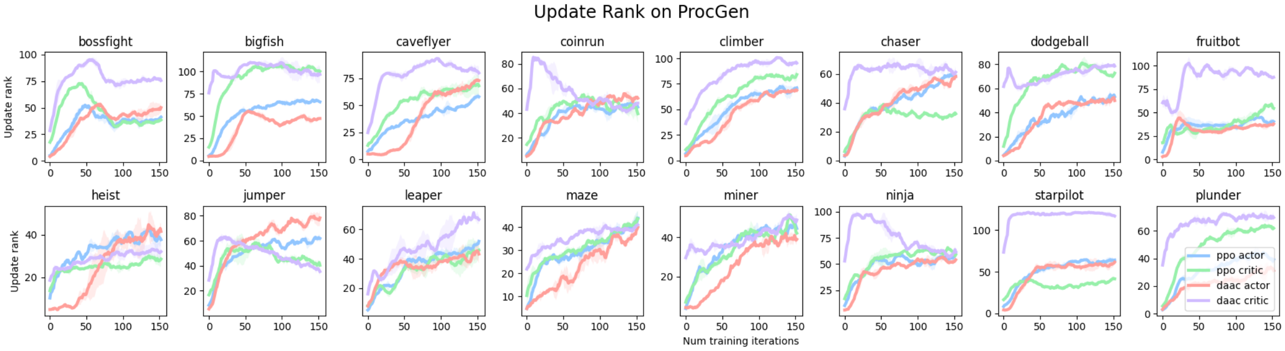

Figure 4. Update dimension of actor-critic methods in ProcGen. Shading indicates minimum and maximum values over 4 seeds. We observe that the update dimension of the separate critic architecture in DAAC (the lilac line) consistently has the highest update rank early in training, while the actors have the lowest rank in the early training stages and only surpass the DAAC critic later in training.

5. Generalization and interference in deep RL

We now explore how TD learning dynamics influence the representations and learned update structure of deep RL agents. We begin by presenting a quantitative approach to measure the degree to which interference is occurring between states in the agent's visitation distribution. Armed with this metric, we evaluate the following hypotheses. First, that deep neural networks trained with TD updates will exhibit weaker interference between states as training progresses compared to their value at initialization (**H1**). Second, we conjecture that networks trained with TD learning will exhibit weaker interference than those trained with policy gradient objectives (**H2**).

5.1. Representation evolution in value-based RL

We begin by developing intuition into how the representations learned by deep RL agents evolve over time. Given a set of transitions $\tau_1, \dots, \tau_n$ of the form $\tau_i = (x_i, a_i, r_i, x_i')$ and a value function V with parameters θ, we let θ_i denote the network parameters after performing an optimization step with respect to the transition τ_i. We then construct a matrix A entry-wise as follows:

$$A_{i,j} = V_{\theta_j}(x_i) - V_\theta(x_i)\,. \tag{21}$$

See Figure 3 for an illustration. The properties of this matrix will depend on the optimizer used to perform updates, leading to notable differences from the neural tangent kernel regime studied elsewhere (Yang et al., 2022) in the case of non-linear function approximators trained with adaptive optimizers. At one extreme, the update matrix A for a tabular value function will have non-zero entries only along the diagonal and the matrix will have full rank. At the other, if the value function were represented by a single parameter $\theta \in \mathbb{R}$, then every row will be identical and the matrix will have rank one. Thus, the rank of this matrix can be interpreted as a proxy for whether an agent tends to *generalize* updates between states (low rank), or whether it *memorizes* the value of each state-action pair independently from other states (high rank). In our evaluations, we use an approximate version of the rank that discards negligible components of the matrix based on the singular value decomposition, described in more detail in Appendix B. We will refer to this quantity as the *update rank*. An alternative approach outlined by Daneshmand et al. (2021) involves computing the Frobenius norm of the difference between the matrix A and the identity, however this may overestimate interference in optimizers which use momentum due to constant terms in the update matrix.

We evaluate **H1** by measuring the update rank of deep RL agents trained on popular benchmarks. We train a standard deep Q-network (DQN) architecture on environments from the Atari 2600 suite, and save checkpoints every 10 million frames. We begin by visualizing the evolution of agents' update matrices over the course of training in Figure 3. RL agents trained in dense-reward environments tend to develop update matrices which resemble those of tabular value functions. Those trained in the absence of reward, i.e. those for which the target value function has no high-frequency components, maintain low-rank update matrices through training as our theory would predict. We find that similar results hold for a range of update rules, including distributional updates performed in the C51 algorithm (Bellemare et al., 2017). We include further evaluations in Appendix D.

5.2. Actor-critic methods

Policy gradient methods present an opportunity to avoid the pathologies discussed previously in temporal difference targets while still preserving other properties of the RL problem. While these methods tend to exhibit other pathologies, in particular suffering from high variance, there is no reason to expect that this variance will discourage interference in the same way as in TD updates. We investigate **H2** using two different algorithms on the ProcGen suite: PPO (Schulman et al., 2017), which uses a shared

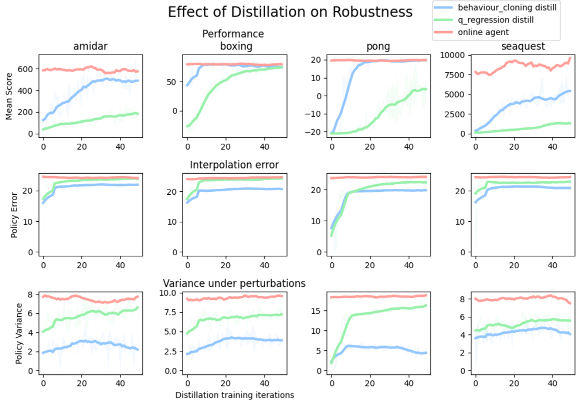

Figure 5. Performance and robustness to perturbations of different distillation approaches in games from the Atari suite. Post-training distillation results in policies that are more consistent under perturbations and under interpolation between observations. Axes indicate the ℓ_1 norm between the policy on the original input batch and on the perturbed input batch.

representation network for both the actor and critic, and DAAC (Raileanu and Fergus, 2021), where there are no shared parameters between the actor and the critic. This setup allows us to study both the effect of the TD loss on a network's update dimension, and long-term effect of TD gradients on the representation. We run our evaluations in the ProcGen environment (Cobbe et al., 2019), which consists of 16 games with procedurally generated levels. While the underlying mechanics of each game remain constant across the different levels, the layout of the environment may vary. The agent is given access to a limited subset of the levels during training, in this case 10, and then evaluated on the full distribution.

We evaluate the update dimension of the actor and critic networks of each method in Figure 4. The critic network in DAAC, which receives only TD gradients, exhibits markedly higher update rank in all environments in at least the early stages of training, and often throughout the entire trajectory, than the other networks which have access to the actor gradients. Our results suggest that the actor gradients in the PPO architecture exhibit a strong regularizing effect on the representation, leading to lower update rank for the critic than would be obtained by an independent critic architecture, but in a manner that is highly variable between environments, highlighting the complexity of representation learning in deep RL.

6. Post-training distillation and generalization

The previous sections have shown that TD learning dynamics discourage interference, and that while this may have a beneficial effect on stability during training, it can reduce the ability of the network to generalize to new observations. This bias towards memorization arises when, during the network's crucial early development stage, it is trained to fit target functions that do not capture the global structure of the value function. One simple solution to this problem is to train a freshly initialized network on the final value function obtained by TD learning. If the learned value function was able to pick up on the global structure of the value function, then the freshly initialized network will be able to benefit from incorporating this structure into its predictions more systematically than the teacher. Such approaches have seen success in prior work (Igl et al., 2019; Nikishin et al., 2022); this section presents a deeper study of a mechanism driving this success.

6.1. Value distillation

We first consider value distillation as a means of eliminating the counterproductive bias towards memorization induced by early TD targets. We leverage a data collection policy from a pre-trained teacher network q_t, and perform distillation of a freshly initialized network q_s on this data. We follow a similar procedure to that of Ostrovski et al. (2021) to perform distillation of the function q_s on data collected sampled from the teacher's replay buffer $\mathcal{B}_T$, leveraging their insight that distillation on *all* action values, rather than only the value of the action taken by the teacher agent, yields significantly higher performance. We additionally study the effect of behaviour cloning with entropy regularization, obtaining the objectives

$$\ell_{\mathrm{VD}}(q_{\mathrm{S}}, q_{\mathrm{T}}) = \mathbb{E}_{s\sim\mathcal{B}_{\mathrm{T}}}\left[\sum_{a\in\mathcal{A}} (q_{\mathrm{S}}(a) - q_{\mathrm{T}}(a))^2\right] \quad (22)$$

$$\ell_{\mathrm{BC}}(\theta) = \mathbb{E}_{s,a\sim\mathcal{B}_{\mathrm{T}}}[\log \pi_\theta(s,a) + \lambda H(\pi_\theta(s))] \quad (23)$$

where $H(\cdot)$ denotes the entropy of the policy. We set $\lambda = 1e-2$ in our evaluations. We show results for value distillation (22), which regresses on the outputs of the frozen Q-network, and behaviour cloning (23), which predicts the action taken by the frozen Q-network. We track three quantities: the performance of the learned policy, the robustness of the learned policy to perturbations, and the consistency of the learned policy when interpolating between observations. The performance is measured by following an ϵ-greedy policy in the training environment, with $\epsilon = 0.01$. The robustness to perturbations is measured by tracking whether the network takes the same action under a Gaussian perturbation to its input as in the unperturbed observation. Finally, we investigate the network's interpolation behaviour by evaluating whether, given a convex combination of observations o_1 and o_2, the network takes the same action under the combination as it does in either of the original observations.

Figure 5 shows that the distilled networks are more robust to perturbations and are more consistent under interpolations between observations. We observe that the behaviour cloning method matches or nearly matches the

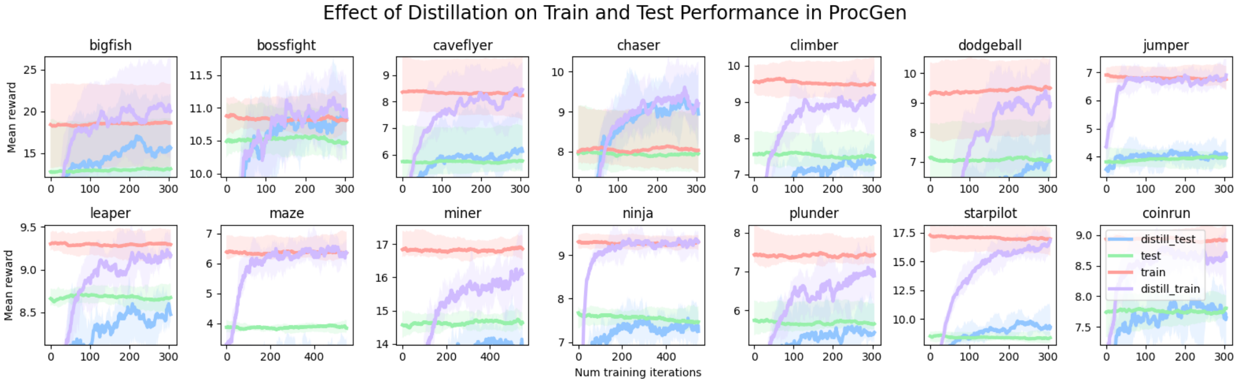

Figure 6. Effect of policy distillation on generalization in environments from the Procgen suite. We plot the pretrained networks train environment and test environment performance, along with the performance of the distilled agent on the test environments. We see significant improvement on test environments in bigfish, caveflyer, chaser, climber, and bossfight.

performance of the pretrained agent in three of the four environments, while also obtaining the best robustness. Both behaviour cloning and value distillation improve upon the teacher network that was trained online. We conclude that while value distillation can mitigate some of the effect of TD methods on interference, policy-based methods exhibit better robustness properties. This finding motivates the next section, where we will dig deeper into policy distillation.

6.2. Policy distillation

Thus far, we have studied generalization between states seen during training. Given the success of policy distillation to improve robustness, we now explore whether distillation may also improve generalization to *novel* environments. We return to the ProcGen benchmark, with the hypothesis that post-training distillation of PPO agents should produce policies which improve on the ability of the final trained actor to generalize to new levels, provided that the levels generated for the test set are sufficiently similar to the training distribution that such generalization is feasible. We reuse the PPO agents from Figure 4 as teacher networks. We train a freshly initialized network (the student) to minimize the KL divergence with the teacher's policy π_T on transitions collected by the teacher and stored in a buffer $\mathcal{B}_T$, yielding the following objective:

$$\mathbb{E}_{s\sim\mathcal{B}_T}[D_{KL}(\pi_S(s)||\pi_T(s)) + \lambda H(\pi_S)] \ . \quad (24)$$

We then evaluate the distilled agent's performance on the test environments. Results are shown in Figure 6. We find that post-training distillation consistently meets or improves upon the generalization *gap* obtained by the original network, in many environments significantly improving on the final test set performance of the PPO agent. We attribute this improvement to the absence of TD learning gradients in the distillation process and the stationarity of the distillation targets, avoiding the pitfalls of non-stationarity highlighted by Igl et al. (2021). It is likely that the raw performance obtained by the student could be improved using lessons from the policy distillation literature (Czarnecki et al., 2019; Rusu et al., 2016; Teh et al., 2017).

7. Conclusion

Our analysis has shown that temporal difference learning targets converge along non-smooth components of the value function first, resulting in a bias towards memorization when deep neural networks are employed as function approximators in value-based RL. In the context of prior work demonstrating that weaker generalization can improve the stability and convergence of RL algorithms, this phenomenon may be beneficial to an agent's stability, but comes at the cost of observational overfitting. We further show that post-training distillation improves generalization and robustness, mitigating some of the tendency of value-based RL objectives to encourage overfitting. Our insights may prove useful in a range of future directions, such as using different architectures during training and distillation, leveraging larger neural network function approximators to minimize harmful interference, and modifying the update rule used in TD learning to adaptively promote or inhibit interference between inputs. Further, the role of the optimiser is fundamental to the phenomena studied in this paper, and RL-specific optimization approaches may benefit from our findings.

Acknowledgements

MK received funding from the ERC under the European Union's Horizon 2020 research and innovation programme (FUN2MODEL, grant agreement No. 834115). CL is funded by the Open Philanthropy AI Fellowship. Thanks to Georg Ostrovski for provided valuable feedback on an early draft of this paper, along with Maximilian Igl and Charline Le Lan for helpful discussions.

References

Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep Q-learning. *arXiv*, 2019.

L Baird. Advantage updating. *Technical Report*, 1993.

David Barrett and Benoit Dherin. Implicit gradient regularization. In *International Conference on Learning Representations*, 2021.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *Proceedings of the International Conference on Machine Learning*, 2020.

David Brellmann, Goran Frehse, and David Filliat. Fourier features in reinforcement learning with neural networks. *arXiv*, 2021.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2019.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2020.

Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *Proceedings of the International Conference on Machine Learning*, 2021.

Wojciech M Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2019.

Hadi Daneshmand, Amir Joudaki, and Francis Bach. Batch normalization orthogonalizes representations in deep random networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in DQN. *arXiv*, 2018.

Farzan Farnia, Jesse M Zhang, and N Tse David. A Fourier-based approach to generalization and optimization in deep learning. *IEEE Journal on Selected Areas in Information Theory*, 1(1):145–156, 2020.

William Fedus, Dibya Ghosh, John D Martin, Marc G Bellemare, Yoshua Bengio, and Hugo Larochelle. On catastrophic interference in Atari 2600 games. *arXiv*, 2020.

Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In *Advances in Neural Information Processing Systems*, 2020.

Sina Ghiassian, Banafsheh Rafiee, Yat Long Lo, and Adam White. Improving performance in reinforcement learning by breaking generalization in neural networks. *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 2020.

Dibya Ghosh and Marc G Bellemare. Representations for stable off-policy reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2020.

Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in Neural Information Processing Systems*, 1995.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, 2019.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Böhmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. *Proceedings of the International Conference on Learning Representations*, 2021.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. SGD on neural networks learns functions of increasing complexity. In *Advances in Neural Information Processing Systems*, 2019.

Zachary Kenton, Angelos Filos, Owain Evans, and Yarin Gal. Generalizing from a few environments in safety-critical reinforcement learning. *arXiv*, 2019.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv*, 2021.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 2000.

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. DR3: Value-based deep reinforcement learning requires explicit regularization. *arXiv*, 2021.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Jaehoon Lee, Samuel Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. In *Advances in Neural Information Processing Systems*, 2020.

Alex Lewandowski. Generalization across space and time in reinforcement learning. *NeurIPS Pre-registration Workshop*, 2020.

Vincent Liu, Adam White, Hengshuai Yao, and Martha White. Towards a practical measure of interference for reinforcement learning. *arXiv*, 2020a.

Vincent Liu, Adam M White, Hengshuai Yao, and Martha White. Measuring and mitigating interference in reinforcement learning. *arXiv*, 2020b.

Yat Long Lo and Sina Ghiassian. Overcoming catastrophic interference in online reinforcement learning with dynamic self-organizing maps. *arXiv*, 2019.

Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. On the effect of auxiliary tasks on representation dynamics. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2021.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

Wesley J Maddox, Gregory Benton, and Andrew Gordon Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*, 2020.

Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 553–560, 2005.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(10), 2007.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. *arXiv preprint arXiv:2205.07802*, 2022.

Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. The difficulty of passive learning in deep reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

Guillermo Valle Pérez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. In *Proceedings of the International Conference on Learning Representations*, 2019.

Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Večerík, et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. *ICML*, 2021.

Roberta Raileanu, Maxwell Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in reinforcement learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

Andrei A Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *ICLR*, 2016.

Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *arXiv*, 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017.

Lin Shao, Yifan You, Mengyuan Yan, Qingyun Sun, and Jeannette Bohg. GRAC: self-guided and self-regularized actor-critic. *CoRR*, 2020.

Samuel L Smith, Benoit Dherin, David Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2020.

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*, 2019.

Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643–1653, 2017.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Tamara Tošić and Pascal Frossard. Graph-based regularization for spherical signal interpolation. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 878–881. IEEE, 2010.

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

Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. In *NeurIPS*, 2020.

Christopher J C H Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

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

Ge Yang, Anurag Ajay, and Pulkit Agrawal. Overcoming the spectral bias of neural value approximation. In *International Conference on Learning Representations*, 2022.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2020.

Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block MDPs. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11214–11224, 2020.

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv*, 2018.

A. Proofs

A.1. Characterizing smoothness in MDPs

Throughout the text, many references are made to 'smooth' functions, without giving a strict definition. While this is useful to convey a rough idea of the types of functions we are interested in, we provide a more rigorous discussion in this section. First, we distinguish between smoothness with respect to a notion of distance in the observation space, for example ℓ_2 distance between vectors, and distance with respect to the MDP's transition dynamics, which measures how easily the agent can reach one state from another. In most settings of interest, the two definitions will largely agree, motivating our use of the generic term smoothness in our discussion of neural network function approximation. In these cases, the inductive bias of the neural network towards smooth functions over the observation space corresponds to an inductive bias twoards functions that are smooth with respect to the MDP transition dynamics. However, this may not always be the case. For example, when walking through a door that leads from one level to another in a video game; though the last frame of the old level and the first frame of the new one may be visually distinct, they will have low distance in the MDP.

The notions of smoothness we refer to in Section 4.1 relates to the variation of the value function between adjacent states in time. This definition resembles graph total variation (Tošić and Frossard, 2010), which characterizes the degree to which a node's value differs from the average of its neighbours. In our case, we treat the transition matrix P^π as a weighted directed graph, and will be interested in the quantity $|V(x) - \mathbb{E}_{P^\pi(x'|x)}[V(x')]|$. We note trivially that if V is an eigenvector of P^π with eigenvalue λ, then

$$\sum_x |V(x) - \mathbb{E}_{P^\pi(x'|x)} V(x')| = \sum_x |(1-\lambda)V(x)| \tag{25}$$

In other words, the eigenvalue of an eigenvector precisely determines the variation of the eigenvector over the entire state space. If $\lambda = 1$, for example, then the eigenvector must be constant in expectation over the MDP, whereas if $\lambda = -1$, then we have $\mathbb{E}_{P^\pi(x'|x)}[V(x')] = -V(x)$ and the value fluctuates between extremes when stepping from one state to another. We obtain an analogous result if, rather than taking the max over states, we take a weighted average or a sum.

A.2. Proofs of main results

Observation 4.1. *Let P^π be real diagonalizable, with eigenvectors $v_1, \dots, v_{|\mathcal{X}|}$ corresponding to eigenvalues $\lambda_1 > \dots \geq \lambda_{|\mathcal{X}|}$, and let V_t be defined as in Equation 10. Write $V_t = \sum_{i=1}^{|\mathcal{X}|} \alpha_i^t v_i$ to express the value function at time t with respect to the eigenbasis $\{v_i\}$. Then the convergence of V_t to the value function $V^\pi = \sum_{i=1}^{|\mathcal{X}|} \alpha_i^\pi v_i$ can be expressed as follows:*

$$\alpha_i^t - \alpha_i^\pi = \exp(-t(1-\gamma\lambda_i))(\alpha_i^0 - \alpha_i^\pi)\,.$$

Proof. Recall we assume the following dynamical system

$$\partial_t V_t = -(I - \gamma P^\pi)V_t + R$$

Inducing the trajectory

$$V_t = \exp(-t(I - \gamma P^\pi))(V_0 - V^\pi) + V^\pi$$

As we assume P^π is diagonalizable, this implies that $(I - \gamma P^\pi)$ is also diagonalizable. Let $u_1, \dots, u_n$ denote the right eigenvectors of P^π with corresponding eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$. Let $V_0 = \sum \alpha_i^0 u_i$.

$$\begin{aligned} V_t &= \sum \alpha_i^t u_i \\ &= \exp(-t(I - \gamma P^\pi))(\sum \alpha_i^0 - \alpha_i^\pi u_i) + \sum \alpha_i^\pi u_i \\ &= \sum \exp(-t(1-\gamma\lambda_i))\bigg(\sum(\alpha_i^0 - \alpha_i^\pi)u_i + \sum \alpha_i^\pi u_i\bigg) \end{aligned}$$

Now, we consider the value of $V_t - V^\pi$ along each coordinate. Note that we have not assumed an orthogonal eigenbasis, thus cannot speak directly to the norm of the projection of this difference onto the eigenspace corresponding to each eigenvector λ_k. However, treating the eigendecomposition as a basis, we can discuss how the coordinates α_i^t of the value function V_t converge with respect to this basis.

$$\begin{aligned}|V_t - V^\pi|[i] = |\alpha_i^t - \alpha_i^\pi| &= |\exp(-t(1-\gamma\lambda_i))(\alpha_i^0 - \alpha_i^\pi) + \alpha_i^\pi - \alpha_i^\pi| \\ &= |\exp(-t(1-\gamma\lambda_i))(\alpha_i^0 - \alpha_i^\pi)| = \exp(-t(1-\gamma\lambda_i))|(\alpha_i^0 - \alpha_i^\pi)|\end{aligned}$$

We conclude by noting that for large values of λ_i, the exponential term $\exp(-t(1-\gamma\lambda_i))$ will decay more slowly as a function of t than for smaller values of λ_i. Thus, these coordinates (which correspond to non-smooth functions over the state space) will converge fastest. When the eigenvectors form an orthogonal basis, as is the case for symmetric P^π, we can go further and observe that this convergence will apply to the norm of the projection of the value function into the corresponding eigenspace. Thus for symmetric P^π, we obtain the following stronger convergence result, where U_k denotes the eigenspace corresponding to the eigenvalue λ_k.

$$\|\Pi_{U_k}(V_t - V^\pi)\| = \exp(-t(1-\gamma\lambda_k))\|\Pi_{U_k}(V_0 - V^\pi)\| \tag{26}$$

□

Theorem 4.1. *Let P^π be real diagonalizable with eigenvalues $\lambda_1 > \cdots > \lambda_n$ and $(v_k)_{k=1}^n$ the corresponding (normalized) eigenvectors. Then for any value function V, the TD error $\mathrm{TD}(V_t) = \|V_t - T^\pi V_t\|^2$ can be bounded as as*

$$\|\mathrm{TD}(V_t)\|^2 = \|T^\pi V_t - V_t\|^2 \tag{11}$$

$$= \|\sum(1-\gamma\lambda_i)(\alpha_i^\pi - \alpha_i^t)(v_i)\|^2 \tag{12}$$

$$\leq \sum_{i=1}^n (\alpha_i^\pi - \alpha_i^t)^2(1-\gamma\lambda_i)^2 \tag{13}$$

with equality when P^π has orthogonal eigenvectors.

Let $V_0 = \sum \alpha_i v_i$. Then, letting V_t be defined as in Equation 10.

$$\mathrm{TD}(V_t) \leq \sum_{i=1}^n \exp(-2t(1-\gamma\lambda_i))(\alpha_i^\pi - \alpha_i^0)^2(1-\gamma\lambda_i)^2\ . \tag{27}$$

Proof. By our assumption on the diagonalizability of P^π, we can leverage the previous result on the coordinates of V_t.

$$V_t - V^\pi = \sum \exp(-t(1-\gamma\lambda_i))\left(\sum(\alpha_i^0 - \alpha_i^\pi)u_i\right)$$

We then bound the TD error as follows.

$$\begin{aligned}\|V_t - \gamma P^\pi V_t - R\|^2 &= \|V_t - \gamma P^\pi V_t + \gamma P^\pi V^\pi - \gamma P^\pi V^\pi - R\| \\ &= \|V_t - \gamma P^\pi V^\pi - R - \gamma P^\pi(V_t - V^\pi)\|\end{aligned}$$

Since $V^\pi = R + \gamma P^\pi V^\pi$, we obtain the following.

$$\begin{aligned}&= \|(I - \gamma P^\pi)(V_t - V^\pi)\|^2 \\ &= \|\sum(1-\gamma\lambda_k)(\alpha_i^t - \alpha_i^\pi)u_i\|^2 \\ &\leq \sum(\alpha_i^\pi - \alpha_i^t)^2(1-\gamma\lambda_i)^2\end{aligned}$$

The remainder follows a straightforward substitution. □

Observation 4.2 (Second-order dynamics)**.** *Let θ_t be defined by the discrete-time system (17) with step size α. Let $f_1(\theta)$ be defined as in (19). Let $\tilde{\theta}_t$ denote the trajectory obtained by following the dynamics:*

$$\partial_t\tilde{\theta}_t = f(\tilde{\theta}_t) + \frac{\alpha}{2}f_1(\tilde{\theta}_t)\ . \tag{20}$$

Then we have $\theta_n \approx \tilde{\theta}_{n\alpha} + O((n\alpha)^3)$, where $\tilde{\theta}_{n\alpha}$ denotes the value of $\tilde{\theta}_t$ at time $t = n\alpha$.

Proof. While our prior analysis has considered the continuous time system $\tilde{\theta}_t$, this does not perfectly approximate the discrete system θ_t. When a fixed step size is used, the first-order continuous-time approximation accrues error roughly proportional to αt. We then follow the procedure of Barrett and Dherin (2021) to reduce the order of this error term, applying a Taylor expansion to the evolution of $\tilde{\theta}_t$ with respect to time. We will use the notation $\tilde{\theta}(t)$ to denote the explicit dependence of $\tilde{\theta}$ as a function of time.

$$\tilde{\theta}(\alpha t) = \tilde{\theta}(0) + \sum \frac{(\alpha t)^n}{n!}\theta^{(n)}(0) \tag{28}$$

$$= \tilde{\theta}(0) + \alpha t f(\tilde{\theta}(0)) + \frac{(\alpha t)^2}{2}\nabla_\theta f \cdot f(\tilde{\theta}(0)) + O(\alpha^3) \tag{29}$$

$$= \tilde{\theta}(0) + \alpha t f(\tilde{\theta}(0)) + \frac{(\alpha t)^2}{2} f_1(\tilde{\theta}(0)) + O(\alpha^3) \tag{30}$$

Relating this back to the discrete system θ_t

$$\theta_1 = \theta_0 + \alpha f(\theta_0) = \tilde{\theta}(0) + \alpha f(\tilde{\theta}(0)) \tag{31}$$

$$\theta_1 = \tilde{\theta}(\alpha 1) - \frac{\alpha^2}{2} f_1(\tilde{\theta}(0)) - O(\alpha^3) \tag{32}$$

Thus, the system $\partial_t \check{\theta}_t = f(\check{\theta}_t) + \frac{\alpha^2}{2} f_1(\check{\theta}_t)$ satisfies

$$\theta_1 = \check{\theta}(\alpha) + O(\alpha^3) \tag{33}$$

We are therefore interested in obtaining the form of f_1 inducing the above approximation $\check{\theta}_t$. We begin by observing that $\nabla_\theta \|V_\theta - \Box T^\pi V_\theta\|^2 = (V_\theta - TV_\theta) \cdot \nabla_\theta V_\theta = f(\theta)$. Importantly, while the function f can be expressed as the gradient of a function in which the target $T^\pi V_\theta$ is fixed, the target in the dynamical system will still depend on the parameter θ and so will also evolve over time. This means that the change in the target $T^\pi V_\theta$ must still be accounted for in our computation of $f_1(\theta) = \nabla_\theta f(\theta) \cdot f(\theta)$ – in particular, $\nabla_\theta f(\theta)$ does *not* equal the second derivative of the fixed-target TD error $\|V_\theta - \Box T^\pi V_\theta\|^2$, but rather the gradient of f treated simply as a function of θ.

$$\theta = \theta_0 + \alpha n f(\theta_0) + (\alpha n)^2/2 \nabla_\theta f(\theta_0) \cdot f(\theta_0) + O((\alpha n)^3) \tag{34}$$

$$= \theta_0 + \alpha n f(\theta_0) + \frac{(\alpha n)^2}{2} f_1(\theta_0) + O((n\alpha)^3) \tag{35}$$

We then express $f_1(\theta)$ as follows.

$$f_1(\theta_0) = \nabla_\theta[f(\theta_0)] \cdot [f(\theta_0)] \tag{36}$$

$$= [\nabla_\theta^2 V_\theta \cdot ((\gamma P^\pi - I)V_\theta + r) + \nabla_\theta V_\theta \cdot ((\gamma P^\pi - I)\nabla_\theta V_\theta)][f(\theta)] \tag{37}$$

$$= [\nabla_\theta^2 V_\theta \cdot ((\gamma P^\pi - I)V_\theta + r) - \nabla_\theta V_\theta \cdot \nabla_\theta V_\theta][f(\theta)] + \gamma[\nabla_\theta V_\theta P^\pi \nabla_\theta V_\theta][f(\theta)] \tag{38}$$

Noting that the left hand side term is equal to the gradient of the gradient norm penalty for the stop-gradient version of the TD regression problem, we simplify as follows:

$$= -\frac{1}{2}\nabla_\theta \left\| \nabla_\theta \frac{1}{2}\|V_\theta - \Box T^\pi V_\theta\|^2 \right\|^2 + \gamma[\nabla_\theta V_\theta \cdot P^\pi \cdot \nabla_\theta V_\theta][f(\theta)] \tag{39}$$

We note that, unlike in the stochastic gradient descent setting, f_1 does not correspond to a gradient of any function. Instead, it corresponds to the second-order correction we would get for a frozen target, which corresponds to a gradient norm penalty, plus a term that measures the alignment of the gradients between each state and its expected successor. Intuitively, both of these terms minimize the 'variance' in the loss induced by noisy, discrete gradient steps, but the right-hand-side term incorporates the effect of the target's evolution on the TD error. The flatter loss surfaces induced by the gradient norm penalty will naturally lead to greater robustness to parameter perturbations. The gradient alignment term reflects the observation previously that non-smooth functions contribute the most to the TD error, and so encourages the first-order gradient effects on successive states to move in a similar direction.

We note that this final observation seems to be at odds with the tendency for TD learning to encourage more tabular updates. Why would a second-order correction term which promotes flat minima and gradient alignment result in tabular updates? To answer this, we point to the tendency of TD targets to converge along. the non-smooth components of the value function first. We are therefore faced with finding a flat region of parameter space to fit a discontinuous function. A representation which succeeds at this will benefit from minimizing interference between states, as the gradients for one transition will be on average uncorrelated with even nearby other states. The gradient alignment penalty suggests that, while the implicit regularization will prefer flat minima, smooth interference patterns which move other states in a similar direction to the current state will be penalized less than non-smooth directions. □

Corollary A.1. *The second-order dynamics push features towards precisely the worst direction w.r.t. stability. I.p. looking at the set of positive definite representations introduced by Ghosh and Bellemare (2020) we see*

$$\{v : v^\top P^\pi v < \gamma^{-1} \|v\|_\Xi\} \tag{40}$$

whereas the optimal gradients for the second order term implicitly solve the following optimization problem

$$\min \mathbb{E}_{x \sim \eta(x)}[g(x)^\top g(x) - \gamma g(x)^\top (P^\pi g)(x)] \tag{41}$$

Theorem 4.2. *Let K be a kernel and π a fixed policy in an MDP with finite state space X. Let $X_{\text{train}} \subset \mathcal{X}$ be a subset of states in the support of π, $X_{\text{test}} = \mathcal{X} \setminus X_{\text{train}}$, and let V_t be a value trajectory obtained by applying kernel semi-gradient updates on the set X_{train} to some initial value function $V_0(X_{\text{train}})$ with kernel K. Let K_{all} be defined as*

$$K_{\text{all}} = K(X_{\text{train}}, X_{\text{train}}) \oplus K(X_{\text{test}}, X_{\text{train}})\,. \tag{15}$$

Then the trajectory of V_t on the entire state space X will be as follows,

$$\partial_t V_t(X) = (K_{\text{all}})[(T^\pi V_t - V_t)(X_{\text{train}})]\,. \tag{16}$$

Proof. We leverage the dynamics $\partial_t V_t = K(X,X)\nabla_\theta V_\theta \cdot ((\gamma P^\pi - I) + r)$ and follow the derivation of Section 5 of Jacot et al. (2018). □

We can develop intuitions for the kernel gradient descent setting by considering the special case of linear function approximation, where $K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$ for some feature map ϕ. For the moment, we will define Φ to be a matrix consisting of features for every state in the state space X (i.e. we update all states in the mdp at once). We then obtain

$$\partial_t \mathbf{w}_t = \alpha \Phi^\top (R^\pi + \gamma P^\pi \Phi \mathbf{w}_t - \Phi \mathbf{w}_t)\,. \tag{42}$$

We can express the evolution of the value function constructed by multiplication of Φ and w as follows.

$$\begin{aligned}
\partial_t V_t &= (\partial_w V_t)^\top \partial_t w_t = \Phi \partial_t w_t & (43)\\
&= -\Phi(\Phi^\top (I - \gamma P^\pi)\Phi)w & (44)\\
&= -\Phi\Phi^\top (I - \gamma P^\pi) V_t & (45)\\
&= -K(I - \gamma P^\pi) V_t & (46)
\end{aligned}$$

We further consider the dynamics of the value function on inputs outside of the set of states on which the Bellman updates are computed as follows.

$$\begin{aligned}
\partial_t V_t(x_{\text{test}}) &= (\partial_w V_t(x_{\text{test}}))^\top \partial_t w_t & (47)\\
&= -\phi(x_{\text{test}})^\top \Phi^\top (I - \gamma P^\pi) V_t & (48)\\
&= -K(x_{\text{test}}, X_{\text{train}}) K(X_{\text{train}}, X_{\text{train}})^{-1} \partial_t V_t & (49)
\end{aligned}$$

We now lift the assumption that all states are updated. In this more general kernel gradient descent setting, we let K be a kernel as before, with $\tilde{K} = K(X_{\text{train}}, X_{\text{train}})$ and $\kappa_{x_{\text{test}}} = K(x_{\text{test}}, X_{\text{train}})$. We then obtain the following dynamics

$$\partial_t V_t(x_{\text{test}}) = \kappa_{x_{\text{test}}} \tilde{K}^{-1} \partial_t V_t(X_{\text{train}}) \tag{50}$$

In particular, this results in the following trajectory.

$$V_t(x_{\text{test}}) = V_0(x_{\text{test}}) + \kappa_{x_{\text{test}}} \tilde{K}^{-1}[V_t(X_{\text{train}}) - V_0(X_{\text{train}})] \tag{51}$$

An interesting case study occurs when we consider, e.g., off-policy evaluation where the bootstrap targets used in the TD updates may not have been visited by the agent during training. This will be the case in many offline RL problems, where the action that would be selected by the policy we seek to evaluate was not taken by the behaviour policy, and so the agent leverages bootstrap targets which are not updated directly as part of the training process, but rather only indirectly via the influence of updates to other states. In such cases, we will decompose the state space as $X = X_{\text{train}} \oplus X_{\text{test}}$. The dynamics we get in this case look quite different from standard kernel regression, as the dynamics of the training states will depend on the predictions on the 'test' states. To condense notation, we will use $T^\pi V_t$ to refer to an application of the Bellman operator $V_t \mapsto \gamma P^\pi V_t + R^\pi$.

$$\partial_t V_t(X_{\text{train}}) = \Phi_{\text{train}} \Phi_{\text{train}}^\top ((T^\pi V_t)(X_{\text{train}}) - V_t(X_{\text{train}})) \tag{52}$$

$$\partial_t V_t(X_{\text{test}}) = \Phi_{\text{test}} \Phi_{\text{train}}^\top ((T^\pi V_t)(X_{\text{train}}) - V_t(X_{\text{train}})) \tag{53}$$

We note that $(T^\pi V_t)(X_{\text{train}})$ depends on both $V(X_{\text{train}})$ and $V(X_{\text{test}})$ due to the application of the Bellman operator T^π. We thus end up with the following joint system.

$$\partial_t V_t(X_{\text{train}} \oplus X_{\text{test}}) = \Phi_{\text{test}} \Phi_{\text{train}}^\top ((T^\pi V_t)(X_{\text{train}}) - V_t(X_{\text{train}})) \oplus \Phi_{\text{train}} \Phi_{\text{train}}^\top ((T^\pi V_t)(X_{\text{train}}) - V_t(X_{\text{train}})) \tag{54}$$

$$\partial_t V_t(X_{\text{train}} \oplus X_{\text{test}}) = (\Phi_{\text{test}} \oplus \Phi_{\text{train}}) \Phi_{\text{train}}^\top ((T^\pi V_t)(X_{\text{train}}) - V_t(X_{\text{train}})) \tag{55}$$

Using a non-standard notation of $K_1 \oplus K_2 := X \mapsto K_1(X) \oplus K_2(X)$, we can then rewrite the above in terms of the dot product kernel $K(x, x')$ as follows.

$$\partial_t V_t(X_{\text{all}}) = (\tilde{K} \oplus \kappa_{x_{\text{test}}})[(T^\pi V_t - V_t)(X_{\text{train}})] \tag{56}$$

We emphasize that while this at first looks as though the dynamics are independent of the value $V_t(X_{\text{test}})$, this is an artefact of the Bellman operator notation $(T^\pi V_t)(X_t)$, which hides the dependence of the Bellman targets $T^\pi V_t$ on X_{test}. In particular, we can write $(T^\pi V_t)(X_t) = \Pi_{X_{\text{train}}}[\gamma P^\pi V_t(X_{\text{train}} \oplus X_{\text{test}}) + R^\pi]$, which makes this dependence more explicit but is less succinct.

B. Experiment details

B.1. Estimation of Update Rank

To estimate the update rank of an agent, we sample k transitions from the agent's replay buffer and compute the matrix $A(\theta)$ as described in Section 5. We use the agent's current optimizer state and its current parameters in this computation. We then take the singular value decomposition of A to obtain k singular values $S = \{\sigma_1, \dots, \sigma_k\}$. We then threshold using the numerical approach taken in prior works (Maddox et al., 2020), and compute the size of the set $S_\epsilon = \{\sigma \in S : \sigma > \epsilon \max(S)\}$. This allows us to ignore directions of near-zero variation in the update matrix. In practice, we use $\epsilon = 0.1$.

Because the Q-functions learned by value-based deep RL agents are vector- rather than scalar-valued functions of state, and our estimator depends on an 2-dimensional update matrix, we must make a choice on how to represent the change in the state-value function. We considered taking the maximum over actions, the mean over actions, selecting a fixed action index, and selecting the action taken in the transition on which the update was computed, and found that both choices produced similar trends. In all evaluations in this paper, Q-functions are reduced using the max operator. We apply the same approach for distributional agents by taking the expectation over the distribution associated with each state-action pair.

To evaluate the policy-based agents, whose outputs correspond to distributions over actions, we compute the norm of the difference in the output probability distributions for each state in lieu of taking the difference of output values. I.e., the entry $A_{i,j} = \|p_\theta(x_j) - p_{\theta_i}(x_j)\|$, where the discrete probability distribution p_θ is taken as a vector.

B.2. ProcGen

The ProcGen benchmark consists of sixteen procedurally generated environments. Each environment consists of a set of randomly generated levels, of which a fixed subset are used for training and a disjoint subset are used for evaluation. Levels

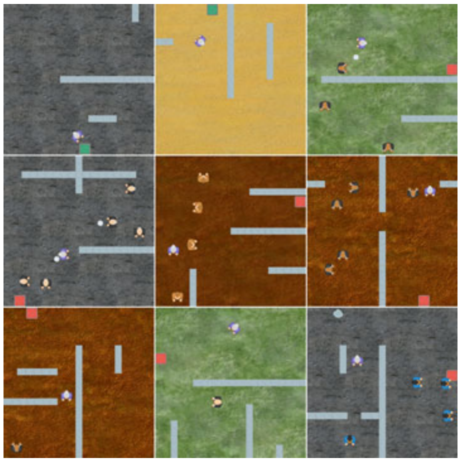

Figure 7. Example levels from the dodgeball environment.

differ superficially in their observations and initial sprite layouts but retain the same underlying structure, as can be seen in Figure 7. The observation space is a box space with the RGB pixels the agent sees in a numpy array of shape (64, 64, 3).

Our PPO and DAAC agents use the same hyperparameters and implementation as is provided by Raileanu and Fergus (2021). Our behaviour cloning objective minimizes the KL divergence between the distillation agent the pretrained agent's policies, with an entropy bonus equal to that used to train the original PPO agent.

B.3. Atari

We additionally perform evaluations on environments from the Atari benchmarks. Due to computational constraints, we consider only a subset of the entire benchmark. We obtain a mixture of easy games, such as pong and boxing, and more challenging games like seaquest, where we measure difficulty by the time it takes for the agent to meet human performance. For some experiments, we used the sparse-reward environment Montezuma's Revenge.

In our distillation experiments, we train the original agent for 50M frames using ϵ-greedy exploration with $\epsilon = 0.1$, and train the distillation agents for a number of updates equivalent to 10M frames of data collected online. We base our implementation off of the open-source implementations in Ostrovski et al. (2021).

For our behaviour cloning objective, we use the same architecture as is used for DQN, but feed the final layer of actions into a softmax to obtain a probability distribution over actions, which we denote as $P_\theta(a|x)$. Given a state-action pair taken by the target agent, we implement the following behaviour cloning loss for distillation

$$\ell(\theta, x_i, a_i) = -\log P_\theta(a_i|x_i) - 0.1 H(P_\theta(\cdot|x_i)) \tag{57}$$

where H denotes the entropy of a distribution. We use a replay capacity of 1e6 and allow the pre-trained agent to collect additional data during distillation to further increase the training set size of the distilled agents.

C. Additional numerical evaluations

We provide additional numerical evaluations to provide additional insight into the theoretical results of Section 4.

C.1. Fourier analysis

We begin by studying the Fourier decomposition of value and reward functions in popular Atari domains by treating the value function as a function of *time* rather than as a function of *observations*. In this sense, the Fourier decomposition is measuring the continuity of the value function with respect to time and so is a closer approximation of the notion of smoothness we focus on in Section 4.1. We show our evaluations in Figure 8.

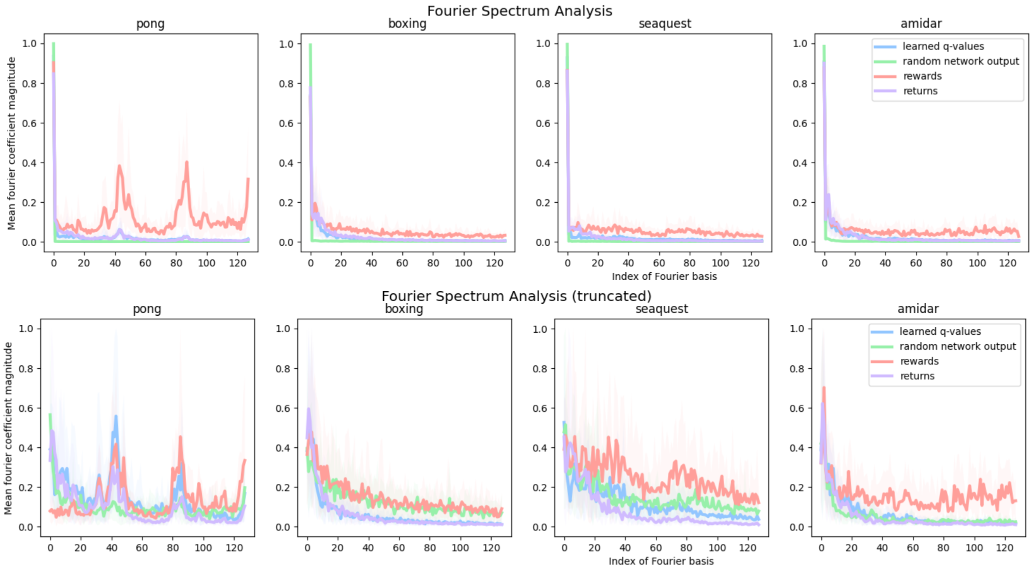

Figure 8. Fourier decomposition of Atari value functions when viewed as a function of time. We sample k consecutive states from the replay buffer and compute the predicted value on each state (fixing an arbitrary action) to get a function $V : \{1, \dots, k\} \to \mathbb{R}$. We then compute the Fourier decomposition of this function. The top row shows indices $k = 0 \dots 50$, while the bottom row omits the $k = 0$ index (the constant function) to better illustrate the rate of decay of the spectrum of each function.

C.2. Kernel gradient descent

We include an illustration of the kernel gradient descent dynamics described in Section 4.2 in Figure 9. We run our evaluations using a radial basis function (RBF) kernel of varying lengthscale, with shorter lengthscales corresponding to weaker generalization between states. While the shorter lengthscale corresponds to more stable learning dynamics and better fitting of the value function on the training set, it also induces greater value approximation error on the test states. In contrast, the longer lengthscales result in better generalization to novel test states under Monte Carlo dynamics, but result in divergence for large values of γ.

Additionally, as promised in Section 4.2, we illustrate the role of smooth eigenfunctions in generalization in Figure 10. To produce this figure, we randomly generate an unweighted graph and then construct an MDP whose dynamics correspond to a random walk on this graph. We consider the generalization error of a kernel regression process where the kernel K_S is of the form $K_S(x, y) = \sum_{i \in S} v_{\lambda_i}(x) v_{\lambda_i}(y)$ for some $S \subseteq \text{spec}(P^\pi)$. In the right-hand-side plot of Figure 10, we set $S = \{1, \dots, 20\}$, so that our analysis concentrates on smooth eigenfunctions. We then consider the generalization error of this smooth kernel when we only regress on a subset of the state space selected uniformly at random[1]. We study the effect of varying the size of this set, i.e. the fraction of states in the training set, in Figure 10, in order to quantify the degree to which additional information about the value function translates to improved generalization. We consider three regression problems: regression on V^π, regression on the projection of V^π onto the span of $T = \{v_1, \dots, v_{20}\}$, and $B = \{v_{n-19}, \dots, v_n\}$. Unsurprisingly, we see that the smooth kernel is able to improve its generalization performance as the size of the training set increases when it is set to regress V^π or $\Pi_T V^\pi = V^\pi_T$. However, when the kernel regresses only on the projection of V^π onto the non-smooth eigenvectors, we don't see a benefit of adding additional training points: because there is no information about the smooth components of the function in the targets, adding additional data points will not help to improve regression accuracy. The left hand side of the figure shows similarly that fitting local information in the form of n-step returns for small n also does not provide the kernel with sufficient information for it to be able to extrapolate and improve its generalization error as the size of the training set increases.

[1] Because the MDP-generating process is invariant to permutations of the state indices, we sample the indices $\{1, \dots, \lfloor |\mathcal{X}| \times \text{trainingfraction} \rfloor\}$, and average over randomly generated MDPs.

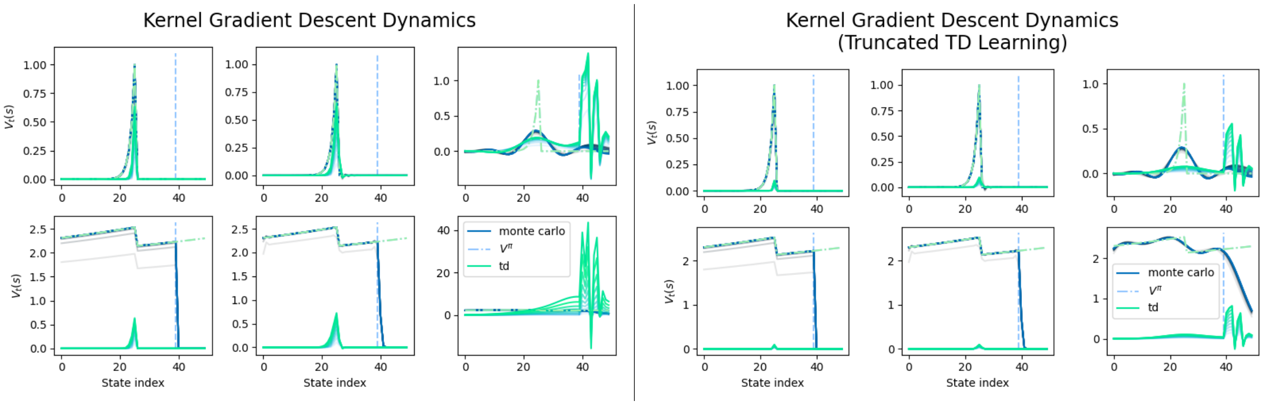

Figure 9. Numerical evaluations of kernel gradient descent with an RBF kernel. The MDP in question is a ”circle MDP” whose states are integers $n \in \{1, \ldots, 50\}$. We assume the agent is ‘trained’ on states 1 to 40, and doesn’t perform value function updates on the final ten states, use the policy which always takes the agent from state n to $n + 1 \mod 50$, and set a single reward at state 25. Each row corresponds to a different value of the discount factor γ: the top corresponds to $\gamma = 0.5$, and the bottom to $\gamma = 0.99$. Each column corresponds to the lengthscale which parameterizes the kernel, going left to right: 0.01, 1.0, and 100. The left hand side and right hand side are distinguished by the number of update steps which the TD dynamics are evaluated for. The LHS runs TD for only 20 steps, while the RHS runs it for 100 steps. MC updates are run for 1500 steps on both figures. We see that for $\gamma = 0.99$, the larger-lengthscale kernel predictions diverge under TD dynamics, though not Monte Carlo. The Monte Carlo dynamics further nicely illustrate the trade-off between generalizing out of the training set and ability to fit the discontinuities of the value function on the training set. The larger lengthscale has lower MSE from the value function on the test set, but fails to fit the discontinuity of the value function at the reward state. Meanwhile, the smaller lengthscales easily fit the value function on the training set but predict zero for all over states.

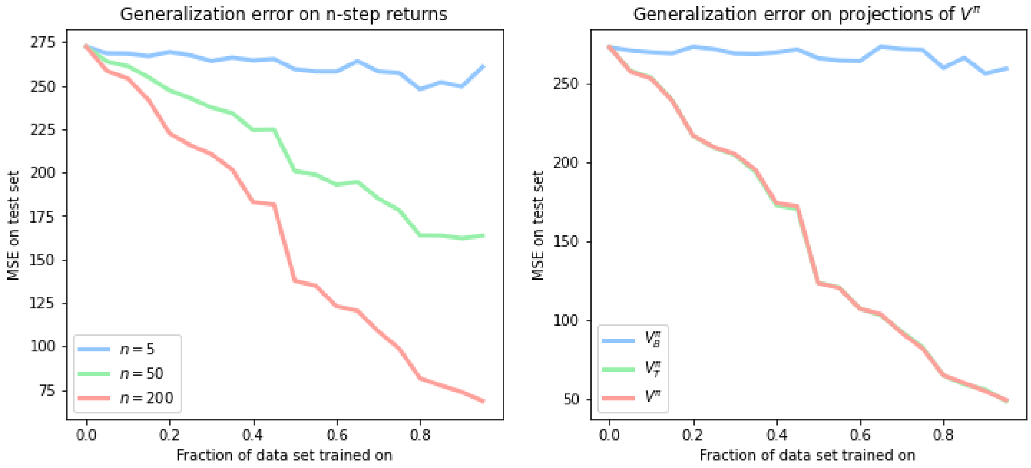

Figure 10. Generalization of predicted function under kernel regression using n-step return targets evaluated on a random subset of states (left), and projecting value function onto top or bottom eigenvectors of P^π (right). We see a similar trend where for larger n (corresponding to smoother targets), the kernel regression method generalizes better with increasing dataset sizes. For smaller n and for the projection of V^π onto non-smooth eigenvectors, adding additional data points doesn’t improve generalization performance.

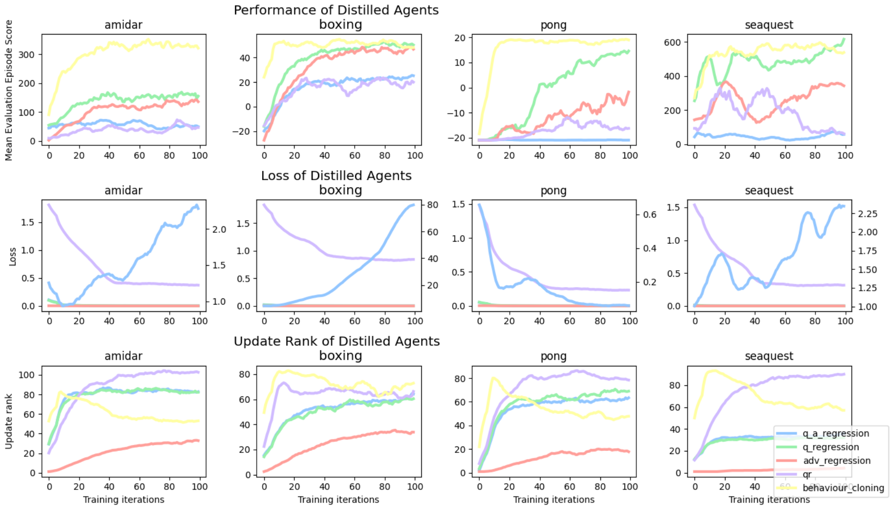

Figure 11. Results from post-training distillation on a variety of objectives. We note that advantage regression tends to exhibit the lowest update rank, with the qr agent tending to exhibit the highest update rank and the q-regression objectives falling somewhere in between. Because the behaviour cloning objective minimizes a cross-entropy loss rather than a regression loss, further investigation is required to understand how the trajectory of its update dimension differs from those of the regression objectives.

D. Additional empirical results

D.1. Additional value distillation results

We consider three different types of regression to the outputs of the pre-trained network, along with two more traditional bootstrapping methods for offline RL. `Q-regression` regresses the outputs of the distilled network to those of the pre-trained network for every action. `qa-regression` only does q-value regression on the action taken by the pre-trained agent. `adv-regression` regresses on the advantage function (computed as the q-value minus the mean over all actions) given by the pre-trained agent; `qr` does quantile regression q-learning on the offline data; `double-q` performs a standard double q-learning update on the offline data.

We find that all of these methods obtain an initial update rank significantly below that of the pre-trained network when they begin training, which increases over time. Regression to the advantages obtains a significantly lower update rank than any other method, suggesting that the advantage function may be much smoother than the action-value function. With respect to performance on the original environment, we see that the methods which use all action values at every update obtain significantly higher performance than those which only update a single action at a time. This improvement in performance isn't mediated by an auxiliary task effect or an increase in the network's ability to distinguish states: the advantage regression network attains low update rank but high performance, while the qr-regression task provides a great deal of information to the representation but is not competitive with the q-regression network.

D.2. More detailed update trajectories

We include a more detailed view of the update matrices obtained by DQN and C51 agents during the first 7 million frames of training, roughly 5% of the training budget, in Figure 12. We see that even early in training, the DQN and C51 agents both exhibit significant overfitting behaviour. Note that states are sampled uniformly at random from the replay buffer, and then assigned an index based on the output of a clustering algorithm to improve readability of the figures.

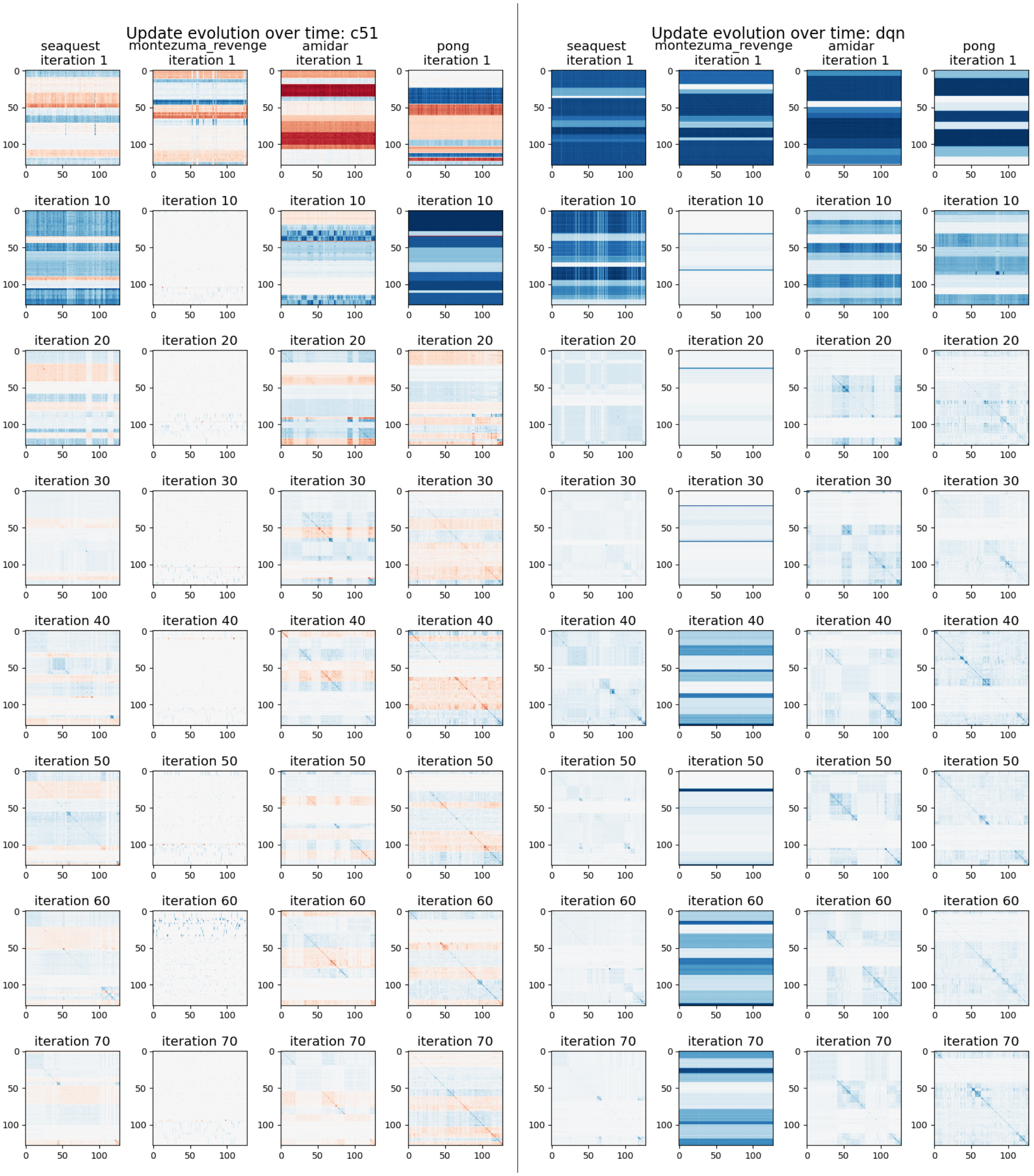

Figure 12. Update matrices for distributional and DQN agents on four games from the Atari suite, chosen to represent a range of reward densities and difficulties. Each iteration corresponds to 1e5 training frames.