Interference and Generalization in Temporal Difference Learning

Emmanuel Bengio[1 2] **Joelle Pineau**[1] **Doina Precup**[1 3]

Abstract

We study the link between generalization and interference in temporal-difference (TD) learning. Interference is defined as the inner product of two different gradients, representing their alignment; this quantity emerges as being of interest from a variety of observations about neural networks, parameter sharing and the dynamics of learning. We find that TD easily leads to low-interference, under-generalizing parameters, while the effect seems reversed in supervised learning. We hypothesize that the cause can be traced back to the interplay between the dynamics of interference and bootstrapping. This is supported empirically by several observations: the negative relationship between the generalization gap and interference in TD, the negative effect of bootstrapping on interference and the local coherence of targets, and the contrast between the propagation rate of information in TD(0) versus TD(λ) and regression tasks such as Monte-Carlo policy evaluation. We hope that these new findings can guide the future discovery of better bootstrapping methods.

1. Introduction

The interference between two gradient-based processes, objectives J_1, J_2, sharing parameters θ is often characterized in the first order by the inner product of their gradients:

$$\rho_{1,2} = \nabla_\theta J_1^T \nabla_\theta J_2, \tag{1}$$

and can be seen as the *influence*, constructive ($\rho > 0$) or destructive ($\rho < 0$), of applying a gradient update using $\nabla_\theta J_1$ on the value of J_2.

This quantity arises in a variety of ways (for completeness we rederive this quantity and others in appendix A); it is the interference between tasks in multi-task and continual learning (Lopez-Paz & Ranzato, 2017; Schaul et al., 2019), it forms the Neural Tangent Kernel (Jacot et al., 2018), it is the Taylor expansion around θ of a gradient update (Achiam et al., 2019), as well as the Taylor expansion of pointwise loss differences (Liu et al., 2019b; Fort et al., 2019).

Interestingly, and as noted by works cited above, this quantity is intimately related to *generalization*. If the interference between two processes is positive, then updating θ using gradients from one process will positively impact the other. Such processes can take many forms, for example, J_1 being the loss on training data and J_2 the loss on test data, or J_1 and J_2 being the loss on two i.i.d. samples.

The main claim of this work is that in Temporal Difference Learning (TD), interference evolves differently during training than in supervised learning (SL). More specifically, we find that **in TD learning lower interference correlates with a higher generalization gap while the opposite seems to be true in SL**, where low interference correlates with a low generalization gap (the difference between test error and train error) when early stopping.

In supervised learning, there is a wealth of literature suggesting that SGD has a regularization effect (Hardt et al., 2016; Zhang et al., 2016; Keskar et al., 2016, and references therein), pushing the parameters in flat highly-connected (Draxler et al., 2018; Garipov et al., 2018) optimal regions of the loss landscape. It would be unsurprising for such regions of parameters to be on the threshold at the balance of bias and variance (in the traditional sense[1]) and thus have low interference[2] as well as a low generalization gap.

Why is the situation then different with TD? This may be due to a multitude of factors. The evaluation methods of new algorithms in the recent union of neural networks and TD, despite an earlier recognition of the problem (Whiteson et al., 2011), often do not include generalization measures,

[1] Mila, McGill University [2] Work partly done while the author was an intern at Deepmind [3] Deepmind. Correspondence to: Emmanuel Bengio <bengioe@gmail.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

[1] We refer here to the recently studied phase transition between generalizing and overfitting, i.e. when DNNs stop learning general patterns and start fitting dataset noise.

[2] Fort et al. (2019) suggest that *stiffness*, the cosine similarity of gradients, drops but stays positive once a model starts overfitting, it is also known that overfitting networks start having larger weights and thus larger gradients; this should result in the smallest ρ precisely before overfitting happens.

perhaps leading to overfitting in algorithm space as well as solution space. This led to many works showing the brittleness of new TD methods (Machado et al., 2018; Farebrother et al., 2018; Packer et al., 2018; Witty et al., 2018), and works proposing to train on a distribution of environments (Zhang et al., 2018c; Cobbe et al., 2018; Justesen et al., 2018) in order to have proper training and test sets (Zhang et al., 2018a;b).

In TD methods, models also face a different optimization procedure where different components may be at odds with each other, leading to phenomena like the deadly triad (Sutton & Barto, 2018; Achiam et al., 2019) and leakage propagation (Penedones et al., 2018). In its purest version, the tabular bootstrapping of TD expects its targets to be fixed unless the target state is visited for an update; gradient updates create interference in unvisited target states, which breaks this assumption.

With most methods, from value-iteration to policy gradients, parameters are also faced with an inherently non-stationary optimization landscape. In particular for value-based methods, bootstrapping induces an asymmetric flow of information (from newly explored states to known states) which remains largely unexplored in deep learning literature. Such non-stationarity and asymmetry may help explain the success of sparse methods (Sutton, 1996; Liu et al., 2019a) that act more like tabular algorithms (with convergence at the cost of more updates).

Other works also underline the importance of interference. Riemer et al. (2018) show that by simply optimizing for interference accross tasks via a naive meta-learning approach, one can improve RL performance. Interestingly, Nichol et al. (2018) also show how popular meta-learning methods implicitly also maximize interference (and thus constructive updates). Considering that the meta-learning problem is inherently interested in generalization, this also suggests that increasing constructive interference should be beneficial.

Why then do TD methods naturally induce under-generalizing low-interference solutions? We first offer an empirical investigation confirming this behavior. Then, we reinterpret results on popular environments where generalization is not typically measured, showing that models may very well be in memorization-mode and lack temporal coherence. Finally, we attempt to offer some mathematical insights into this phenomenon.

2. Preliminaries

A Markov Decision Process (MDP) (Bellman, 1957; Sutton & Barto, 2018) $\mathcal{M} = \langle S, A, R, P, \gamma \rangle$ consists of a state space S, an action space A, a reward function $R : S \to \mathbb{R}$ and a transition probability distribution $P(s'|s, a)$. RL agents aim to optimize the long-term return,

$$G(S_t) = \sum_{k=t}^{\infty} \gamma^{k-t} R(S_k),$$

in expectation, where $\gamma \in [0, 1)$ is called the discount factor. Policies $\pi(a|s)$ map states to action distributions. Value functions V^π and Q^π map states/states-action pairs to expected returns, and can be expressed recursively:

$$\begin{aligned} V^\pi(S_t) &= \mathbb{E}_\pi[G(S_t)] \\ &= \mathbb{E}_\pi[R(S_t) + \gamma V(S_{t+1}) | A_t \sim \pi(S_t)] \\ Q^\pi(S_t, A_t) &= \mathbb{E}_\pi[R(S_t) + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)] \end{aligned}$$

While V^π could also be learned via regression to observed values of G, these recursive equations give rise to the *Temporal Difference (TD)* update rules for policy evaluation, relying on current estimates of V to *bootstrap*, e.g.:

$$V(S_t) \leftarrow V(S_t) - \alpha(V(S_t) - (R(S_t) + \gamma V(S_{t+1}))), \quad (2)$$

where $\alpha \in [0, 1)$ is the step-size. Bootstrapping leads also to algorithms such as **Q-Learning** (Watkins & Dayan, 1992):

$$\mathcal{L}_{QL} = [Q_\theta(S_t, A_t) - (R_t + \gamma \max_a Q_{\theta'}(S_{t+1}, a))]^2, \quad (3)$$

fitted-Q (Ernst et al., 2005; Riedmiller, 2005), and **TD(λ)**, which trades off between the unbiased target $G(S_t)$ and the biased TD(0) target (biased due to relying on the estimated $V(S_{t+1})$), using a weighted averaging of future targets called a λ-return (Sutton, 1988; Munos et al., 2016):

$$G^\lambda(S_t) = (1 - \lambda) \sum\nolimits_{n=1}^{\infty} \lambda^{n-1} G^n(S_t) \quad (4)$$

$$G^n(S_t) = \gamma^n V(S_{t+n}) + \sum\nolimits_{j=0}^{n-1} \gamma^j R(S_{t+j})$$

$$\mathcal{L}_{TD(\lambda)}(S_t) = (V_\theta(S_t) - G^\lambda(S_t))^2, \quad (5)$$

(note that the return depends implicitly on the trajectory and the actions followed from S_t). When $\lambda = 0$, the loss is simply $(V_\theta(S_t) - (R_t + \gamma V_\theta(S_{t+1})))^2$, leading to the TD(0) algorithm (Sutton, 1988).

Learning how to act can be done solely using a value function, e.g. with the greedy policy $\pi(s) = \text{argmax}_a Q(s, a)$. Alternatively, one can directly parameterize the policy as a conditional distribution over actions, $\pi_\theta(a|s)$. In this case, π can be updated directly with policy gradient (PG) methods, the simplest of which is REINFORCE (Williams, 1992):

$$\nabla_\theta G(S_t) = G(S_t) \nabla_\theta \log \pi(A_t|S_t). \quad (6)$$

2.1. Computing interference quantities

Comparing loss interference in the RL and SL case isn't necessarily indicative of the right trends, due to the fact that in most RL algorithms, the loss landscape itself evolves over time as the policy changes. Instead, we remark that loss interference, $\rho_{1,2} = \nabla_\theta J_1^T \nabla_\theta J_2$, can be decomposed as follows. Let J be a scalar loss, u and v some examples, and f the parameterized function's output:

$$\rho_{u,v} = \frac{\partial J(u)}{\partial f(u)} \frac{\partial f(u)}{\partial \theta}^T \frac{\partial f(v)}{\partial \theta} \frac{\partial J(v)}{\partial f(v)}. \tag{7}$$

While the partial derivative of the loss w.r.t. f may change as the loss changes, we find experimentally that the inner product of gradients of the output of f remains stable[3]. As such, we will also compute this quantity throughout, function interference, as it is more stable and reflects interference at the representational level rather than directly in relation to the loss function:

$$\bar{\rho}_{u,v} = \frac{\partial f(u)}{\partial \theta}^T \frac{\partial f(v)}{\partial \theta}. \tag{8}$$

For functions with more than one output in this work, e.g. a softmax classifier, we consider the output, $f(u)$, to be the max[4], e.g. the confidence of the argmax class.

3. Empirical Setup

For the generalization experiments of Section 4 we loosely follow the setup of Zhang et al. (2018a): we train RL agents in environments where the initial state is induced by a single random seed, allowing us to have proper training and test sets in the form of mutually exclusive seeds. In particular, to allow for closer comparisons between RL and SL, we compare classifiers trained on SVHN (Netzer et al., 2011) and CIFAR10 (Krizhevsky, 2009) to agents that learn to progressively explore a masked image (from those datasets) while attempting to classify it. The random seed in both cases is the index of the example in the training or test set.

More specifically, agents start by observing only the center, an 8×8 window of the current image. At each timestep they can choose from 4 movement actions, moving the observation window by 8 pixels and revealing more of the image, as well as choose from 10 classification actions. The episode ends upon a correct classification or after 20 steps.

We train both RL and SL models with the same architectures, and train RL agents with a Double DQN objective (van Hasselt et al., 2015). We also train REINFORCE (Williams, 1992) agents as a test to entirely remove dependence on value estimation and have a pure Policy Gradient (PG) method.

As much of the existing deep learning literature on generalization focuses on classifiers, but estimating value functions is arguably closer to regression, we include two regression experiments using SARCOS (Vijayakumar & Schaal, 2000) and the California Housing dataset (Pace & Barry, 1997).

Finally, for the interactive environment experiments of Section 5, we investigate some metrics on the popular Atari environment (Bellemare et al., 2013) by training DQN (Mnih et al., 2013) agents, with the stochastic setup recommended by Machado et al. (2018), and by performing policy evaluation with the Q-Learning and TD(λ) objectives. To generate interesting trajectories for policy evaluation, we run an "expert" agent pretrained with Rainbow (Hessel et al., 2018); we denote $\mathcal{D}^*$ a dataset of transitions obtained with this agent, and θ^* the parameters after training that agent.

We measure correlations throughout with Pearson's r:

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \tag{9}$$

which is a measure, between -1 and 1, of the linear correlation between two random variables X, Y. All architectural details and hyperparameter ranges are listed in appendix B. All code is available in the supplementary materials.

4. Empirical observations of interference and generalization

To measure interference in the overparameterized regime and still be able to run many experiments to obtain trends, we instead reduce the number of training samples while also varying capacity (number of hidden units and layers) with smaller-than-state-of-the-art but reasonable architectures.

First, in Fig. 1 for each training set size, we measure the correlation between interference and the generalization gap. We see that, after being given sufficient amounts of data, TD methods tend to have a strong negative correlation, while classification methods tend to have positive correlation.

Regression has similar but less consistent results; SARCOS has a high correlation peak when there starts being enough data, albeit shows no correlation at all when all 44k training examples are given (the generalization gap is then almost 0 for all hyperparameters); on the other hand the California dataset only shows positive correlation when most or all of the dataset is given. The trends for PG SVHN and CIFAR show no strong correlations (we note that $|r| < 0.3$ is normally considered to be a weak correlation; Cohen, 2013) except for PG CIFAR at 100 training seeds, with $r = -.60$.

Second, in Fig. 2, we plot the generalization gap against interference $\bar{\rho}$ for every experiment (normalized for comparison). We then draw the linear regression for each experiment over all training set sizes and capacities. For both classifi-

[3] Although gradients do not always converge to 0, at convergence the parameters themselves tend to *wiggle around* a minima, and as such do not affect the *function* and its derivatives that much.

[4] This avoids computing the (expensive) Jacobian, we also find that this simplification accurately reflects the same trends experimentally.

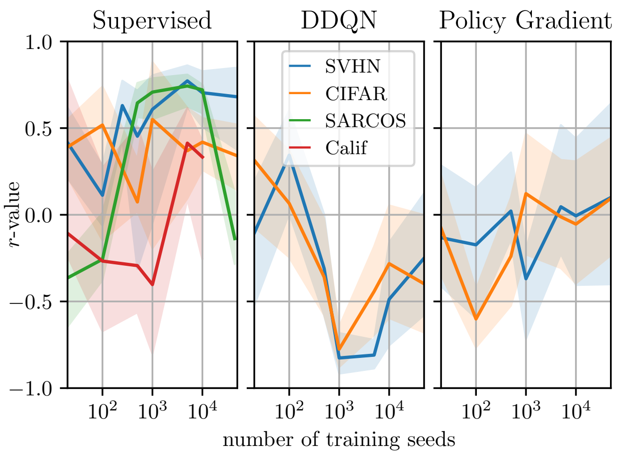

Figure 1. Correlation coefficient r between the (log) function interference $\bar{\rho}$ and the generalization gap, as a function of training set size; shaded regions are bootstrapped 90% confidence intervals. We see different trends for value-based experiments (middle) than for supervised (left) and PG experiments (right). For classification, these methods clearly have different effect on interference even though they roughly approximate the same function.

cation tasks, interference is strongly correlated ($r > 0.9$) with the generalization gap, and also is to a lesser extent for the PG experiments. For all other experiments, regression and value-based, the correlation is instead negative, albeit low enough that a clear trend cannot be extracted. Note that the generalization gap itself is almost entirely driven by the training set size first ($r < -0.91$ for all experiments except PG, where r is slightly higher, see appendix Fig. 9).

The combination of these results tells us that not only does interference evolves differently in TD than in SL, it has some similarities with regression, as well as a different characterization of memorization: **in classification, low-interference solutions tend to generalize, while in TD, low-interference solutions often memorize**. In regression, this seems only true for a fixed quantity of data.

5. Interference in Atari domains

The Arcade Learning Environment (Bellemare et al., 2013), comprised of Atari games, has been a standard Deep RL benchmark in the recent past (Mnih et al., 2013; Bellemare et al., 2017; Kapturowski et al., 2019). We once again revisit this benchmark to provide additional evidence of the memorization-like behaviors of value-based methods on these domains. Understanding the source of these behaviors is important, as presumably algorithms may be able to learn generalizing agents from the same data. Additionally, such low-interference memorization behaviors are not conducive

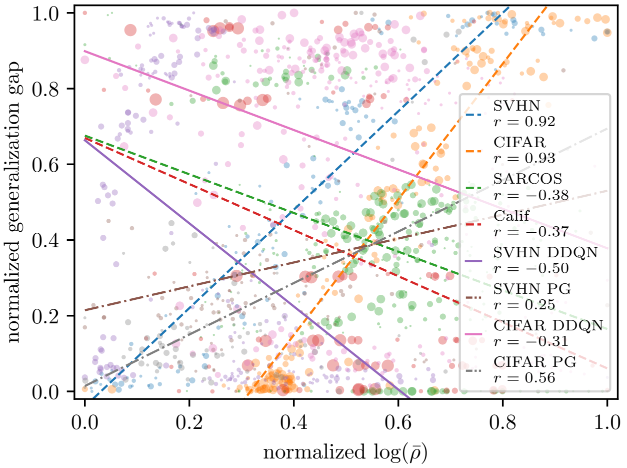

Figure 2. Generalization gap vs interference $\bar{\rho}$ for all runs. Larger circles represent larger capacity models. Here value-based methods seem to be behaving like regression methods.

to sample efficiency, which even in an environment like Atari, could be improved.

Recall that interference is a first order Taylor expansion of the pointwise loss difference, $J_{\theta'} - J_{\theta}$. Evaluating such a loss difference is more convenient to do on a large scale and for many runs, as it does not require computing individual gradients. In this section, we evaluate the expected TD loss difference for several different training objectives, a set of supervised objectives, the Q-Learning objective applied first as policy evaluation (learning from a replay buffer of expert trajectories) and then as a control (learning to play from scratch) objective, and the TD(λ) objective applied on policy evaluation. Experiments are ran on MsPacman, Asterix, and Seaquest for 10 runs each. Results are averaged over these three environments (they have similar magnitudes and variance). Learning rates are kept constant, they affect the magnitude but not the shape of these curves. We use 10M steps in the control setting, and 500k steps otherwise.

We first use the following 3 supervised objectives to train models using $\mathcal{D}^*$ as a dataset and Q_{θ^*} as a *distillation* target:

$$\begin{aligned}\mathcal{L}_{MC}(s,a) &= (Q_\theta(s,a) - G^{(\mathcal{D}^*)}(s))^2\\ \mathcal{L}_{reg}(s,a) &= (Q_\theta(s,a) - Q_{\theta^*}(s,a))^2\\ \mathcal{L}_{TD^*}(s,a,r,s') &= (Q_\theta(s,a) - (r + \gamma \max_{a'} Q_{\theta^*}(s',a')))^2\end{aligned}$$

and measure the difference in pointwise TD loss ($\mathcal{L}_{QL}$) for states *surrounding* the state used for the update (i.e. states with a temporal offset of ±30 in the replay buffer trajectories), shown in Fig. 3.

There, we see that curves tend to be positive around $x = 0$ (the sample used in the update), especially from indices

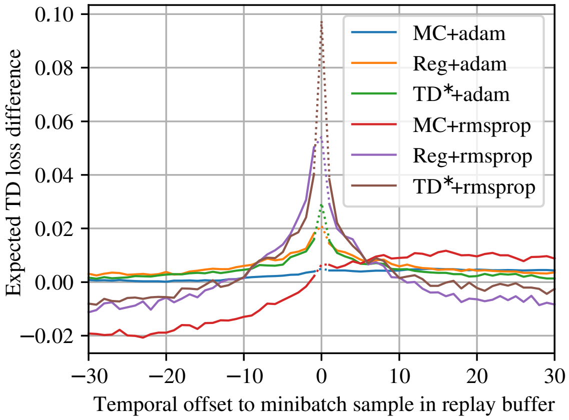

Figure 3. Regression on Atari: loss difference as a function of temporal offset in the replay buffer from the update sample. We use dotted lines at 0 offset to emphasize that the corresponding state was used for the update. The curve around 0 is indicative of the constructive interference of the TD and regression objectives.

-10 to 10, showing that **constructive interference is possible** when learning to approximate Q^* with this data. Since Q_{θ^*} is a good approximation, we expect that $Q_{\theta^*}(s,a) \approx (r + \gamma \max_{a'} Q_{\theta^*}(s',a'))$, so $\mathcal{L}_{reg}$ and $\mathcal{L}_{TD^*}$ have similar targets and we expect them to have similar behaviours. Indeed, their curves mostly overlap.

Next, we again measure the difference in pointwise loss for surrounding states. We train control agents and policy evaluation (or *Batch Q*) agents with the Q-Learning loss:

$$\mathcal{L}_{QL} = [Q_\theta(S_t, A_t) - (R_t + \gamma \max_a Q_\theta(S_{t+1}, a))]^2. \quad (10)$$

We show the results in Fig. 4. Compared to the regressions in Fig. 3, the pointwise difference is more than an order of magnitude smaller, and drops off even faster when going away from $x = 0$. This suggests a low interference, and a low update propagation. For certain optimizers, here RMSProp (Hinton et al., 2012) and SGD, this effect is even slightly negative. We believe this difference may be linked to momentum (note the difference with Adam (Kingma & Ba, 2015) and Momentum-SGD), which might dampen some of the negative effects of TD on interference (further discussed in section 6.1).

Interestingly, while Q-Learning does not have as strong a gain as the regressions from Fig. 3, it has a larger gain than policy evaluation. This may have several causes, and we investigate two.

First, because of the initial random exploratory policy, the DNN initially sees little data variety, and may be able to capture a minimal set of factors of variation; then, upon seeing new states, the extracted features are forced to be mapped onto those factors of variation, improving them, leading to a natural curriculum. By looking at the *singular values* of the last hidden layer's matrix after 100k steps, we do find that there is a *consistently larger spread* in the policy evaluation case than the control case (see appendix D.1), showing that in the control case fewer factors are initially captured. This effect diminishes as training progresses.

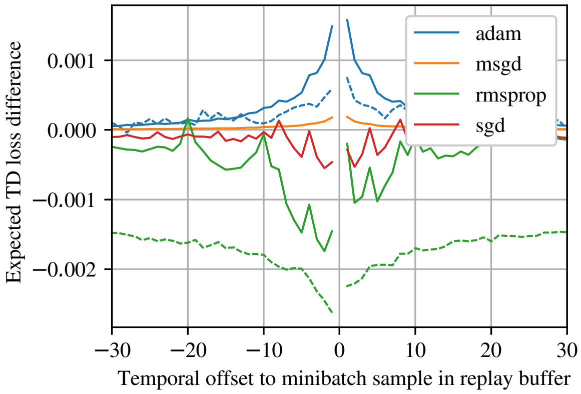

Figure 4. TD Learning on Atari: loss difference as a function of offset in the replay buffer of the update sample. Full lines represent Q-Learning control experiments, while dashed lines represent policy evaluation with a Q-Learning objective. We exclude $x = 0$ for clarity, as it has a high value (see appendix Fig. 11). Compared to regression, the magnitude of the gain is much smaller.

Second, having run for 10M steps, control models could have been trained on more data and thus be forced to generalize better; this turns out **not** to be the case, as measuring the same quantities for only the first 500k steps yields very similar magnitudes. In other words, after a few initial epochs, function interference remains constant (see appendix D.2).

Interestingly, these results are consistent with those of Agarwal et al. (2019), who study off-policy learning. Among many other results, Agarwal et al. (2019) find that off-policy-retraining a DQN model (i.e. Batch Q-Learning) on another DQN agent's lifetime set of trajectories yields much worse performance. This is consistent with our results showing more constructive interference in control than in policy evaluation, and suggests that the order in which data is presented may matter when bootstrapping is used.

5.1. TD(λ) and bootstrapping

A central hypothesis of this work is that bootstrapping causes instability in interference, causing it to become small and causing models to memorize more. Here we perform policy evaluation on $\mathcal{D}^*$ with TD(λ). TD(λ) is by design a way to tradeoff between bias and variance in the target by trading off between few-step bootstrapped targets and long-term bootstrapped targets which approach Monte-Carlo returns. In other words, TD(λ) allows us to diminish reliance on bootstrapping.

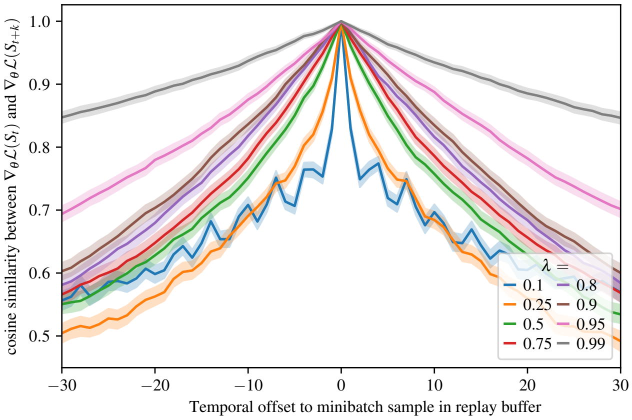

Figure 5. Cosine similarity between gradients at S_t (offset $x = 0$) and the gradients at the neighboring states in the replay buffer (MsPacman). As λ increases, so does the temporal coherence of the gradients.

This tradeoff is especially manifest when measuring the *stiffness* of gradients (cosine similarity) as a function of temporal offset, as shown in Fig. 5. There we see that the closer λ is to 1, the more gradients are similar around an update sample, suggesting that diminishing reliance on bootstrapping reduces the effect of TD inducing low-interference memorizing parameterizations.

Note that this increase in similarity between gradients is also accompanied by an increase in pointwise loss difference (shown in Fig. 6), surpassing that of Q-Learning (Fig. 4) in magnitude. This suggests that TD(λ) offers more coherent targets that allow models to learn faster, for sufficiently high values of λ.

5.2. The high variance of bias in TD(0)

In TD(0), the current target for any state depends on the prediction made at the next state. The difference between that prediction and the true value function makes the target a biased estimator when bootstrapping is in progress and information flows from newly visited states to seen states.

This "bootstrap bias" itself depends on a function approximator which has its own bias-variance tradeoff (in the classical sense). For a high-variance approximator, this bootstrap bias might be inconsistent, making the value function alternate between being underestimated and being overestimated, which is problematic in particular for nearby states[5]. In such a case, a gradient descent procedure cannot truly take advantage of the constructive interference between gradients.

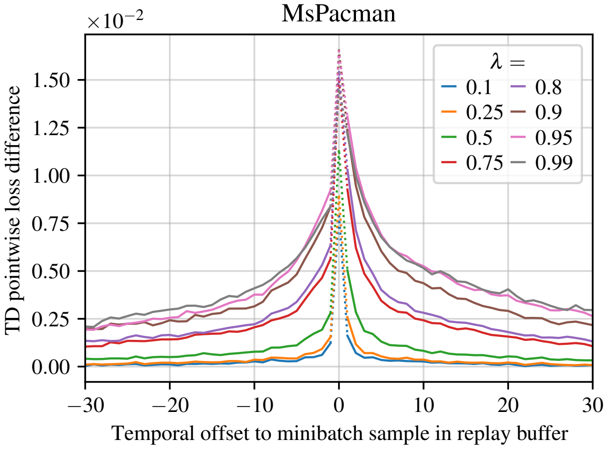

Figure 6. Evolution of TD pointwise loss difference, as a function of λ in TD(λ). Notice the asymmetry around 0.

Indeed, recall that in the case of a regression, interference can be decomposed as:

$$\rho_{x,y} = \frac{\partial J(x)}{\partial f(x)} \frac{\partial f(x)}{\partial \theta}^T \frac{\partial f(y)}{\partial \theta} \frac{\partial J(y)}{\partial f(y)},$$

which for the TD error $\delta_x = V(x) - (r(x) + \gamma V(x'))$ with x' some successor of x, can be rewritten as:

$$\rho_{x,y} = \delta_x \delta_y \nabla_\theta V(x)^T \nabla_\theta V(y).$$

If x and y are nearby states, in some smooth high-dimensional input space (e.g. Atari) they are likely to be close in input space and thus to have a positive function interference $\nabla_\theta V(x)^T \nabla_\theta V(y)$. If the signs of δ_x and δ_y are different, then an update at x will increase the loss at y. As such, we measure the variance of the sign of the TD error along small windows (of length 5 here) in trajectories as a proxy of this local target incoherence.

We observe this at play in Fig. 7, which shows interference and rewards as a function of sign variance for a DQN agent trained on MsPacman. As predicted, parameterizations with a large $\bar{\rho}$ and a large sign variance perform much worse. We note that this effect can be lessened by using a much smaller

[5] Consider these two sequences of predictions of V: $[1, 2, 3, 4, 5, 6]$ and $[1, 2, 1, 2, 1, 2]$. Suppose no rewards, $\gamma = 1$, and a function interference ($\bar{\rho}$) close to 1 for illustration, both these sequences have the same average TD(0) error, 1, yet the second sequence will cause any TD(0) update at one of the states to only correctly update half of the values.

learning rate than is normal, but this comes at the cost of having to perform more updates for a similar performance (in fact, presumably because of reduced instability, performance is slightly better, but only towards the end of training; runs with a normal α plateau halfway through training).

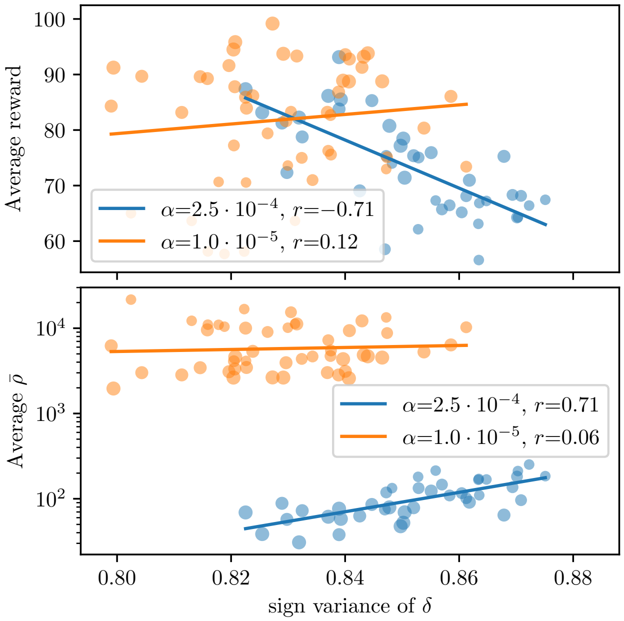

Figure 7. Top, average reward after training as a function of the sign variance for different learning rates (α) and number of hidden units (size of markers). We can see that by using a much smaller learning rate than normal, the biasing effect of TD is lessened, at the cost of many more updates. Bottom, average function interference $\bar{\rho}$ after training. We see that, as predicted, parameterizations with large $\bar{\rho}$ and a large sign variance perform much worse (note that the x-axis of both plots are aligned, allowing for an easy reward/interference comparison).

Interestingly, parameterizations with large $\bar{\rho}$ *generally do* have a large sign variance ($r = 0.71$) in the experiment of Fig. 7. Indeed, we believe that the evolution of interference in TD methods may be linked to sign variance, the two compounding together, and may explain this trend.

These results are consistent with the improvements obtained by Thodoroff et al. (2018), who force a temporal smoothing of the value function through convex combinations of sequences of values which likely reduces sign variance. These results are also consistent with Anschel et al. (2017) and Agarwal et al. (2019) who obtain improvements by training ensembles of value functions. Such ensembles should partly reduce the sign variance of δ, as bias due to initialization should average to a small value, making targets more temporally consistent.

Finally, note that in regression, this problem may eventually go away as parameters converge. Instead, in TD(0), especially when making use of a frozen target, this problem simply compounds with time and with every update. In what follows we consider this problem analytically.

6. Understanding the evolution of interference

Here we attempt to provide some insights into how interference evolves differently in classification, regression, and TD learning. For detailed derivations see appendix A.

Recall that the interference ρ can be obtained by the negative of the derivative of the loss $J(A)$ after some update using B w.r.t. the learning rate α, i.e.

$$\begin{aligned} \theta' &= \theta - \alpha \nabla_\theta J(B) \qquad (11) \\ \rho_{AB} = -\partial J_{\theta'}(A)/\partial\alpha &= \nabla_{\theta'} J(A) \cdot \nabla_\theta J(B) \qquad (12) \\ &\approx \nabla_\theta J(A) \cdot \nabla_\theta J(B). \qquad (13) \end{aligned}$$

The last step being a simplification as $\theta \approx \theta'$.

To try to understand how this quantity evolves, we can simply take the derivative of ρ (and $\bar{\rho}$) w.r.t. α but evaluated at θ', that is, $\rho'_{AB} = \partial(\nabla_{\theta'} J(A) \cdot \nabla_{\theta'} J(B))/\partial\alpha$. In the general case, we obtain (assuming $\theta \approx \theta'$, we omit the θ subscript and subscript A and B for brevity):

$$\begin{aligned} \rho'_{AB} &= -(\nabla J_B^T H_A + \nabla J_A^T H_B)\nabla J_B \qquad (14) \\ \bar{\rho}'_{AB} &= -(\nabla f_B^T \bar{H}_A + \nabla f_A^T \bar{H}_B)\nabla J_B \qquad (15) \end{aligned}$$

where $H_A = \nabla^2_\theta J(A;\theta)$, $\bar{H}_A = \nabla^2_\theta f(A;\theta)$ are Hessians.

Interpreting this quantity is non-trivial, but consider $\nabla f_A^T \bar{H}_B \nabla J_B$; parameters which make f_A change, which have high curvature at B (e.g. parameters that are not stuck in a saddle point or a minima at B), and which change the loss at B will contribute to change ρ. Understanding the sign of this change requires a few more assumptions.

Because neural networks are somewhat smooth (they are Lipschitz continuous, although their Lipschitz constant might be very large, see Scaman & Virmaux (2018)), it is likely for examples that are close in input space *and* target space to have enough gradient and curvature similarities to increase their corresponding interference, while examples that are not similar would decrease their interference. Such an interpretation is compatible with our results, as well as those of Fort et al. (2019) who find that *stiffness* (cosine similarity of gradients) is mostly positive only for examples that are in the same class.

Indeed, notice that for a given softmax prediction σ, for A and B of different classes y_A, y_B, the sign of the partial derivative at $\sigma_{y_A}(A)$ will be the opposite of that of $\sigma_{y_A}(B)$. Since gradients are multiplicative in neural networks, this will flip the sign of all corresponding gradients related to σ_{y_A}, causing a mismatch with curvature, and a decrease in interference. Thus the distribution of targets and the loss has a large role to play in aligning gradients, possibly just as much as the input space structure.

We can also measure ρ' to get an idea of its distribution. For a randomly initialized neural network, assuming a normally distributed input and loss, we find that it does not appear to be 0 mean. While the median is close to 0, but consistently negative, the distribution seems heavy-tailed with a slightly larger negative tail, making the negative mean further away from 0 than the median. In what follows we decompose ρ' to get some additional insights.

In the case of regression, $J_A = 1/2(f_A - y_A)^2$, $\delta_A = f_A - y_A$, we get that:

$$\begin{aligned}\rho'_{reg;AB} &= -\bar\rho^2_{AB}\delta^2_B - 2\delta_A\delta_B\bar\rho_{AB}\bar\rho_{BB} \\ &\quad - \delta_A\delta^2_B\nabla f_B(\bar H_A\nabla f_B + \bar H_B\nabla f_A) \qquad (16)\end{aligned}$$

Another interesting quantity is the evolution of ρ when J is a TD loss if we assume that the bootstrap target also changes after a weight update. With the $\theta \approx \theta'$ simplification, $\delta_A = V_A - (r + \gamma V_{A'})$ the TD error at A, A' some successor of A, we get:

$$\begin{aligned}\rho'_{TD;AB} &= -\delta^2_B\bar\rho_{AB}(\bar\rho_{AB} - \gamma\bar\rho_{A'B}) \\ &\quad - \delta_A\delta_B\bar\rho_{AB}(\bar\rho_{BB} - \gamma\bar\rho_{B'B}) \\ &\quad - \delta_A\delta^2_B\nabla f_B(\bar H_A\nabla f_B + \bar H_B\nabla f_A) \qquad (17)\end{aligned}$$

Again considering the smoothness of neural networks, if A and B are similar, but happen to have opposite δ signs, their interference will decrease. Such a scenario is likely for high-capacity high-variance function approximators, and is possibly **compounded by the evolving loss landscape**. As the loss changes–both prediction and target depend on a changing θ–it evolves imperfectly, and there are bound to be many pairs of nearby states where only one of the δs flips signs, causing gradient misalignments. This would be consistent with our finding that higher-capacity neural networks have a smaller interference in TD experiments (see appendix Fig. 10) while the reverse is observed in classification.

We now separately measure the three additive terms of ρ'_{reg} and ρ'_{TD}, which we refer to as $\rho' = -r_1 - r_2 - r_3$, in the same order in which they appear in (16) and (17).

We measure these terms in four scenarios, using a MsPacman expert replay buffer. We regress to $Q_{\theta*}$ (measuring ρ'_{reg}), and run policy evaluation with three different targets (measuring ρ'_{TD}). In DQN, the target $Q_{\bar\theta}$ is a frozen network updated every 10k iterations; in DDQN the target is updated with an exponential moving average rule, $\bar\theta = (1-\tau)\bar\theta + \tau\theta$, with $\tau = 0.01$; in QL the target is the model itself Q_θ as assumed in (17). This is shown in Fig. 8. We see that in regression r_1 and r_2 are positive much more often than they are negative, while in TD methods, the positive samples tend to dominate but the proportion of negative samples is much larger, especially for r_2, which contains a $\delta_A\delta_B$ product. We see that r_3 tends to have a smaller magnitude than other terms for TD methods, and is negative on average. These results again suggest that TD methods do not have a stable evolution of interference.

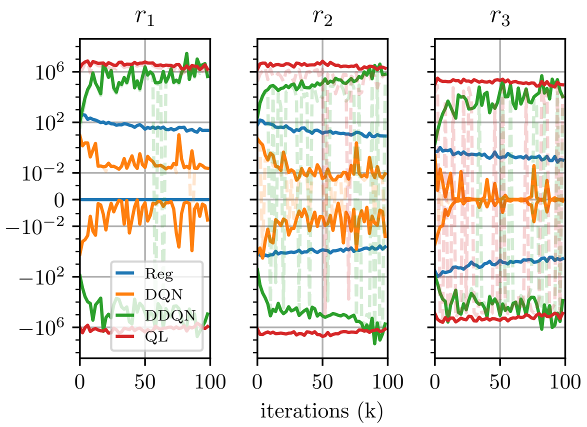

Figure 8. r_1, r_2, r_3 for ρ'_{reg} (Reg) and ρ'_{TD} (DQN, DDQN, QL) measured early in training. The transparent dashed lines are the mean r_i, averaged over 1024 (32 × 32) sample pairs, averaged over 3 runs. The full lines above and below 0 are the average of the positive and negative samples of r_i respectively. These lines show the relative magnitudes of each part: in general, positive samples dominate for r_1, r_2 varies a lot between positive and negative for TD, while r_3 is mostly negative with some variance for TD.

6.1. Interference and momentum

Momentum SGD has the following updates, $\beta \in [0, 1)$:

$$\mu_t = (1-\beta)\nabla_\theta J_B + \beta\mu_{t-1} \qquad (18)$$
$$\theta' = \theta - \alpha(\beta\mu_{t-1} + (1-\beta)\nabla_\theta J_B) \qquad (19)$$

yielding the following quantities:

$$\rho_{\mu;AB} = (1-\beta)\nabla_{\theta'} J_A \cdot \nabla_\theta J_B + \beta\nabla_{\theta'} J_A \cdot \mu_{t-1} \qquad (20)$$
$$\rho'_{\mu;AB} = -(1-\beta)\rho'_{AB} - \beta\nabla J_B H_A \mu_{t-1} \qquad (21)$$

Note that the first term of $\rho'_{\mu;AB}$ is simply eq. (14) times $1-\beta$. The second term is more interesting, and presumably larger as β is usually close to 1. It indicates that for interference to change, the curvature at A and the gradient at B need to be aligned with μ, the moving average of gradients. As such, the evolution of interference may be driven more by the random (due to the stochasticity of SGD) alignment of the gradients with μ, which should be stable over time since μ changes slowly, than by the (high-variance) alignment of curvature at A and gradient at B. As such, momentum should lower the variance of ρ' and dampen the evolution of interference when it is high-variance, possibly including dampening the negative effects of interference in TD.

7. Discussion

RL is generally considered a harder problem than supervised learning due to the non-i.i.d. nature of the data. Hence, the fact that TD-style methods require more samples than supervised learning when used with neural networks is not necessarily surprising. However, with the same data and the same final targets (the "true" value function), it is not clear why TD updates lead to parameters that generalize worse than in supervised learning. Indeed, our results show that the interference of a converged model evolves differently as a function of data and capacity in TD than in supervised learning.

Our results also show that Q-Learning generalizes poorly, leading to DNNs that memorize the training data (not unlike table lookup). Our results also suggest that TD(λ), although not widely used in recent DRL, improves generalization. Additionally, we find differences between Adam and RMSProp that we initially did not anticipate. Very little work has been done to understand and improve the coupling between optimizers and TD, and our results indicate that this would be an important future work direction.

While a full description of the mechanisms that cause TD methods to have such problems remains elusive, we find that understanding the evolution of gradient interference reveals intriguing differences between the supervised and temporal difference objectives, and hint at the importance of stable targets in bootstrapping.

Finally, our work suggests that the RL community should pay special attention to the current research on generalization in DNNs, as naively approaching the TD bootstrapping mechanism as a supervised learning problem does not seem to leverage the full generalization potential of DNNs.

Acknowledgements

The first author is grateful for funding from the Fonds de recherche du Québec – Nature et technologies (FRQNT), as well great discussions from people at Deepmind and Mila. We are in particular grateful for feedback to Joshua Romoff, Harsh Satija, Veronica Chelu, Pierre Thodoroff, and Yoshua Bengio.

References

Achiam, J., Knight, E., and Abbeel, P. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

Agarwal, R., Schuurmans, D., and Norouzi, M. Striving for simplicity in off-policy deep reinforcement learning. *arXiv preprint arXiv:1907.04543*, 2019.

Anschel, O., Baram, N., and Shimkin, N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 176–185. JMLR. org, 2017.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning, 2017.

Bellman, R. A markovian decision process. *Journal of mathematics and mechanics*, pp. 679–684, 1957.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

Cohen, J. *Statistical power analysis for the behavioral sciences*. Routledge, 2013.

Dangel, F., Kunstner, F., and Hennig, P. Back{pack}: Packing more into backprop. In *International Conference on Learning Representations*, 2020.

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. A. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*, 2018.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Farebrother, J., Machado, M. C., and Bowling, M. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

Fort, S., Nowak, P. K., and Narayanan, S. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., and Wilson, A. G. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, 2018.

Hardt, M., Recht, B., and Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, pp. 1225–1234, 2016.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *CSC321*, 2012.

Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. Are deep policy gradient algorithms truly policy gradient algorithms? *arXiv preprint arXiv:1811.02553*, 2018.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Justesen, N., Torrado, R. R., Bontrager, P., Khalifa, A., Togelius, J., and Risi, S. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*, 2018.

Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J., and Munos, R. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kingma, D. and Ba, J. Adam: a method for stochastic optimization (2014). *arXiv preprint arXiv:1412.6980*, 15, 2015.

Krizhevsky, A. Learning multiple layers of features from tiny images. 2009.

Liu, V., Kumaraswamy, R., Le, L., and White, M. The utility of sparse representations for control in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4384–4391, 2019a.

Liu, V., Yao, H., and White, M. Toward understanding catastrophic interference in value-based reinforcement learning. 2019b. URL https://optrl2019.github.io/accepted_papers.html.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. 2013.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Morcos, A. S., Barrett, D. G., Rabinowitz, N. C., and Botvinick, M. On the importance of single directions for generalization. *arXiv preprint arXiv:1803.06959*, 2018.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. 2011.

Nichol, A., Achiam, J., and Schulman, J. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Oymak, S., Fabian, Z., Li, M., and Soltanolkotabi, M. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*, 2019.

Pace, R. K. and Barry, R. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291 – 297, 1997. ISSN 0167-7152. doi: https://doi.org/10.1016/S0167-7152(96)00140-X.

Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., and Song, D. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Penedones, H., Vincent, D., Maennel, H., Gelly, S., Mann, T., and Barreto, A. Temporal difference learning with neural networks-study of the leakage propagation problem. *arXiv preprint arXiv:1807.03064*, 2018.

Pineau, J. The machine learning reproducibility checklist v1.2. 2019.

Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pp. 6076–6085, 2017.

Riedmiller, M. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pp. 317–328. Springer, 2005.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., and Tesauro, G. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

Scaman, K. and Virmaux, A. Lipschitz regularity of deep neural networks: analysis and efficient estimation, 2018.

Schaul, T., Borsa, D., Modayil, J., and Pascanu, R. Ray interference: a source of plateaus in deep reinforcement learning. *arXiv preprint arXiv:1904.11455*, 2019.

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

Sutton, R. S. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pp. 1038–1044, 1996.

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

Thodoroff, P., Durand, A., Pineau, J., and Precup, D. Temporal regularization for markov decision process. In *Advances in Neural Information Processing Systems*, pp. 1779–1789, 2018.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning, 2015.

Vijayakumar, S. and Schaal, S. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, Vol. 1, 05 2000.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Whiteson, S., Tanner, B., Taylor, M. E., and Stone, P. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 120–127. IEEE, 2011.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696.

Witty, S., Lee, J. K., Tosch, E., Atrey, A., Littman, M., and Jensen, D. Measuring and characterizing generalization in deep reinforcement learning. *arXiv preprint arXiv:1812.02868*, 2018.

Zhang, A., Ballas, N., and Pineau, J. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018a.

Zhang, A., Wu, Y., and Pineau, J. Natural environment benchmarks for reinforcement learning, 2018b.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018c.

A. Derivations

A.1. Interference

Consider an objective-based SGD update from $\nabla_\theta J$ using sample B (here A and B can be understood as samples, but in general they can be tasks, or even entire data distributions):

$$\theta' = \theta - \alpha \nabla_\theta J(B)$$

The effect of this update on the objective elsewhere, here at sample A, can be understood as the derivative of the loss elsewhere with respect to the learning rate, yielding the well-known gradient interference quantity ρ:

$$\begin{aligned}\rho_{AB} = \frac{\partial J_{\theta'}(A)}{\partial \alpha} &= -\frac{\partial J_\theta(A)}{\partial \theta'}\frac{\partial \theta'}{\partial \alpha} &(22)\\ &= -\nabla_{\theta'} J_{\theta'}(A)^T \nabla_\theta J_\theta(B) &(23)\\ &\approx -\nabla_\theta J_\theta(A)^T \nabla_\theta J_\theta(B) &(24)\end{aligned}$$

This quantity can also be obtained from the Taylor expansion of the loss difference at A after an update at B:

$$\begin{aligned}J_{\theta'(B)}(A) - J_\theta(A) &\approx J_\theta(A) - J_\theta(A) + \nabla_\theta J(A)^T(\theta' - \theta) + O(||\theta' - \theta||^2) &(25)\\ &\approx -\alpha \nabla_\theta J_\theta(A)^T \nabla_\theta J_\theta(B) &(26)\end{aligned}$$

For what follows we use the following notation for brevity: we subscript $f(A)$ as f_A, when writing $\nabla_{\theta'} J_A$ we imply that $J_A = J(A;\theta')$, when writing a gradient ∇ or Hessian H a lack of θ subscript implies ∇_θ or H_θ rather than θ'.

A.2. Second order quantities

The derivative of ρ w.r.t. α, or second order derivative of $J_{\theta'}$ w.r.t. α is:

$$\begin{aligned}\frac{\partial^2 J_{\theta'}(A)}{\partial \alpha^2} &= -\frac{\partial}{\partial \alpha}\nabla_{\theta'} J_A^T \nabla_\theta J_B &(27)\\ &= -(\frac{\partial(\nabla_{\theta'} J(A))}{\partial \theta'}\frac{\partial \theta'}{\partial \alpha})^T \nabla_\theta J_B &(28)\\ &= -(-\nabla^2_{\theta'} J_A \nabla_\theta J_B)^T \nabla_\theta J_B &(29)\\ &\approx \nabla J_B^T H_A \nabla J_B &(30)\end{aligned}$$

assuming $\theta \approx \theta'$ in the last step, and where $H_A = \nabla^2_\theta J_A$ is the Hessian. Again the only approximation here is $\theta \approx \theta'$

While this quantity is interesting, it is in a sense missing a part: what happens to the interference itself after an update? At **both** A and B at θ'?

$$\begin{aligned}\rho'_{AB} &= \frac{\partial}{\partial \alpha}\nabla_{\theta'} J_A^T \nabla_{\theta'} J_B &(31)\\ &= (\frac{\partial(\nabla_{\theta'} J_A)}{\partial \theta'}\frac{\partial \theta'}{\partial \alpha})^T \nabla_\theta J_B + \nabla_{\theta'} J_A^T(\frac{\partial(\nabla_{\theta'} J_B)}{\partial \theta'}\frac{\partial \theta'}{\partial \alpha}) &(32)\\ &= (-\nabla^2_{\theta'} J_A \nabla_\theta J_B)^T \nabla_\theta J_B + \nabla_{\theta'} J_A^T(-\nabla^2_{\theta'} J_B \nabla_\theta J_B) &(33)\\ &\approx -\nabla J_B^T H_A \nabla J_B - \nabla J_A^T H_B \nabla J_B &(34)\end{aligned}$$

Following Nichol et al. (2018) we can rewrite this as:

$$\begin{aligned}&= -\,(\nabla J_B^T H_A + \nabla J_A^T H_B)\nabla J_B &(35)\\ &= -\,(\nabla_\theta(\nabla J_B^T \nabla J_A))\nabla J_B &(36)\end{aligned}$$

This last form is easy to compute with an automatic differentiation software and does not require explicitly computing the hessian. We also verify empirically that this quantity holds with commonly used small step-sizes.

The derivative of function interference can also be written similarly:

$$\begin{aligned}
\bar{\rho}'_{AB} &= \frac{\partial}{\partial\alpha}\nabla_{\theta'} f_A^T \nabla_{\theta'} f_B && (37)\\
&= (\frac{\partial(\nabla_{\theta'} f_A)}{\partial\theta'}\frac{\partial\theta'}{\partial\alpha})^T \nabla_{\theta'} f_B + \nabla_{\theta'} f_A^T (\frac{\partial(\nabla_{\theta'} f_B)}{\partial\theta'}\frac{\partial\theta'}{\partial\alpha}) && (38)\\
&= (-\nabla^2_{\theta'} f_A \nabla_\theta J(B))^T \nabla_{\theta'} f_B + \nabla_{\theta'} f_A^T(-\nabla^2_{\theta'} f_B \nabla_\theta J(B)) && (39)\\
&\approx -\nabla J_B^T \bar{H}_A \nabla f_B - \nabla f_A^T \bar{H}_B \nabla J_B && (40)\\
&= -(\nabla f_B^T \bar{H}_A + \nabla f_A^T \bar{H}_B)\nabla J_B && (41)
\end{aligned}$$

where by $\bar{H}$ we denote the Hessian of the function f itself rather than of its loss.

Note that for the parameterized function f_θ

$$\nabla_\theta J = \frac{\partial J}{\partial f}\frac{\partial f}{\partial \theta}$$

Let's write $\frac{\partial J}{\partial f} = \delta$. For any regression-like objective $(f-y)^2/2$, $\delta = (f-y)$. δ's sign will be positive if f needs to decrease, and negative if f needs to increase.

Let's rewrite the interference as:

$$\nabla_\theta J_\theta(A)^T \nabla_\theta J_\theta(B) = \delta_A \delta_B \nabla_\theta f_\theta(A)^T \nabla_\theta f_\theta(B)$$

Then notice that ρ' can be decomposed as follows. Let $g_{AB} = \nabla_\theta f_\theta(A)^T \nabla_\theta f_\theta(B)$, $g'_{AB} = \nabla_{\theta'} f_{\theta'}(A)^T \nabla_{\theta'} f_{\theta'}(B)$:

$$\begin{aligned}
\rho'_{reg;AB} =& \frac{\partial}{\partial\alpha}\delta_A\delta_B \nabla_{\theta'} f_A^T \nabla_{\theta'} f_B && (42)\\
=& \frac{\partial \delta_A}{\partial\alpha}\delta_B g'_{AB} + \frac{\partial\delta_B}{\partial\alpha}\delta_A g'_{AB} && (43)\\
&+ \delta_A\delta_B(\frac{\partial}{\partial\alpha}\nabla_{\theta'} f_A)^T \nabla_{\theta'} f_B + \delta_A\delta_B(\frac{\partial}{\partial\alpha}\nabla_{\theta'} f_B)^T \nabla_{\theta'} f_A && (44)\\
=& -\nabla_{\theta'} f_A^T \nabla_\theta J_B \delta_B g'_{AB} - \nabla_{\theta'} f_B^T \nabla_\theta J_B \delta_A g'_{AB} && (45)\\
&+ \delta_A\delta_B(-\bar{H}_{\theta';A}\nabla_\theta J_B)^T \nabla_{\theta'} f_B + \delta_A\delta_B(-\bar{H}_{\theta';B}\nabla_\theta J_B)^T \nabla_{\theta'} f_A && (46)\\
&\text{if we assume } \theta \approx \theta',\ g \approx g' \text{ we can simplify} && (47)\\
\approx& -g\delta_B\delta_B g - 2\delta_B g \delta_A g_{BB} - \delta_A\delta_B\delta_B\nabla_\theta f_B \bar{H}_A \nabla_\theta f_B - \delta_A\delta_B\delta_B \nabla_\theta f_B \bar{H}_B \nabla_\theta f_A && (48)\\
=& -g_{AB}^2\delta_B^2 - 2\delta_A\delta_B g_{AB} g_{BB} - \delta_A \delta_B^2 \nabla_\theta f_B(\bar{H}_A \nabla f_B + \bar{H}_B \nabla_\theta f_A) && (49)
\end{aligned}$$

We can also compute ρ' for TD(0) assuming that the target is not frozen and is influenced by an update to θ. Again we want $\partial/\partial\alpha[g'_{AB}]$ for an update at B, interference at A, assuming that B' is a successor state of B used for the TD update, and A' a successor of A in δ_A:

$$\begin{aligned}
\theta' &= \theta - \alpha\delta_B\nabla_\theta f_B && (50)\\
&= \theta - \alpha(f_B - (r + \gamma f_{B'})\nabla_\theta f_B && (51)
\end{aligned}$$

Also note that:

$$\begin{aligned}
\frac{\partial\delta_A}{\partial\alpha} =& \left(\frac{\partial f_A}{\partial\theta'}\frac{\partial\theta'}{\partial\alpha} - \gamma\frac{\partial f_{A'}}{\partial\theta'}\frac{\partial\theta'}{\partial\alpha}\right) && (52)\\
=& -\delta_B(\nabla_{\theta'} {f_A}^T \nabla_\theta f_B - \gamma \nabla_{\theta'} {f_{A'}}^T \nabla_\theta f_B) && (53)\\
& && (54)
\end{aligned}$$

Let $g_{AB} = \nabla_\theta f_A^T \nabla_\theta f_B$, $g'_{AB} = \nabla_{\theta'} f_A^T \nabla_{\theta'} f_B$ and $g^{\setminus}_{AB} = \nabla_{\theta'} f_A^T \nabla_\theta f_B$:

$$\begin{aligned}
\rho_{TD;AB} &= \frac{\partial}{\partial\alpha}\left[\delta_A\delta_B\nabla_{\theta'} f_A^T \nabla_{\theta'} f_B\right] && (55)\\
&= \frac{\partial}{\partial\alpha}\delta_A\delta_B\nabla_{\theta'} f_A^T \nabla_{\theta'} f_B && (56)\\
&\quad + \delta_A\frac{\partial}{\partial\alpha}\delta_B\nabla_{\theta'} f_A^T \nabla_{\theta'} f_B && (57)\\
&\quad + \delta_A\delta_B\frac{\partial}{\partial\alpha}\nabla_{\theta'} f_A^T \nabla_{\theta'} f_B && (58)\\
&\quad + \delta_A\delta_B\nabla_{\theta'} f_A^T \frac{\partial}{\partial\alpha}\nabla_{\theta'} f_B && (59)\\
&= -\delta_B(g^{\setminus}_{AB} - \gamma g^{\setminus}_{A'B})\delta_B g'_{AB} && (60)\\
&\quad - \delta_B(g^{\setminus}_{BB} - \gamma g^{\setminus}_{B'B})\delta_A g'_{AB} && (61)\\
&\quad + \delta_A\delta_B(-\bar{H}_{\theta';A}\nabla_\theta J_B)^T\nabla_{\theta'} f_B + \delta_A\delta_B(-\bar{H}_{\theta';B}\nabla_\theta J_B)^T\nabla_{\theta'} f_A && (62)
\end{aligned}$$

which again if we assume $\theta' \approx \theta, g_{AB} \approx g'_{AB} \approx g^{\setminus}_{AB}$, we can simplify to:

$$\begin{aligned}
\rho'_{TD;AB} \quad = \quad & -\delta_B^2 g_{AB}(g_{AB} - \gamma g_{A'B}) - \delta_A\delta_B g_{AB}(g_{BB} - \gamma g_{B'B})\\
& -\delta_A\delta_B^2\nabla_\theta f_B(\bar{H}_A\nabla_\theta f_B + \bar{H}_B\nabla_\theta f_A)
\end{aligned}$$

B. Architectures, hyperparameter ranges, and other experimental details

We use the PyTorch library (Paszke et al., 2019) for all experiments. To efficiently compute gradients for a large quantity of examples at a time we use the backpack library (Dangel et al., 2020).

Note that we run natural images experiments first to get a more accurate comparison of the generalization gap between RL and SL (Section 4). We then run Atari experiments to analyse information propagation, TD(λ), and the local coherence of targets (Section 5), because Atari agents (1) have long term decision making which highlights the issues of using TD for long term reward predictions (which is TD's purpose) and (2) are a standard benchmark.

B.1. Figure 1, 2, 8 and 9

In order to generate these figures we train classifiers, regression models, DDQN agents and REINFORCE agents.

Models trained on SVHN and CIFAR10, either for SL, DDQN, or REINFORCE, use a convolutional architecture. Let n_h be the number of hiddens and n_L the number of extra layers. The layers are:

- Convolution, 3 in, n_h out, filter size 5, stride 2
- Convolution, n_h in, $2n_h$ out, filter size 3
- Convolution, $2n_h$ in, $4n_h$ out, filter size 3
- n_L layers of Convolution, $4n_h$ in, $4n_h$ out, filter size 3, padding 1
- Linear, $4n_h \times 10 \times 10$ in, $4n_h$ out
- Linear, $4n_h$ in, n_o out.

All layers except the last use a Leaky ReLU (Maas et al., 2013) activation with slope 0.01 (note that we ran a few experiments with ReLU and tanh activations out of curiosity, except for the slightly worse training performance the interference dynamics remained fairly similar). For classifiers n_o is 10, the number of classes. For agents n_o is 10+4, since there are 10 classes and 4 movement actions.

Models trained on the California Housing dataset have 4 fully-connected layers: 8 inputs, 3 Leaky ReLU hidden layers with n_h hiddens, and a linear output layer with a single output.

Models trained on the SARCOS dataset have 2+n_L fully-connected layers: 21 inputs, 1+n_L Leaky ReLU hidden layers with n_h hiddens, and a linear output layer with 8 outputs.

Let n_T be the number of training seeds. We use the following hyperparameter settings:

- SVHN, $n_h \in \{8, 16, 32\}$, $n_L \in \{0, 1, 2, 3\}$, $n_T \in \{20, 100, 250, 500, 1000, 5000, 10000, 50000\}$
- CIFAR10, $n_h \in \{16, 32, 64\}$, $n_L \in \{0, 1, 2, 3\}$, $n_T \in \{20, 100, 250, 500, 1000, 5000, 10000, 50000\}$
- SARCOS, $n_h \in \{16, 32, 64, 128, 256\}$, $n_L \in \{0, 1, 2, 3\}$, $n_T \in \{20, 100, 250, 500, 1000, 5000, 10000, 44484\}$
- California Housing, $n_h \in \{16, 32, 64, 128\}$, $n_T \in \{20, 100, 250, 500, 1000, 5000, 10000\}$

For SVHN and CIFAR10, we use the same architecture and hyperparameter ranges for classification, DDQN and REINFORCE experiments. Each hyperparameter setting is run with 3 or more seeds. The seeds affect the initial parameters, the sampling of minibatches, and the sampling of ϵ-greedy actions.

Note that while we run REINFORCE on SVHN and CIFAR, we do not spend a lot of time analyzing its results, due the relatively low relevance of PG methods to the current work. Indeed, the goal was only to highlight the difference in trends between TD and PG, which do indicate that the two have different behaviours. Policy gradient methods do sometimes rely on the TD mechanism (e.g. in Actor-Critic), but they use different update mechanisms and deserve their own independent analysis, see for example Ilyas et al. (2018).

For optimizers, we use the standard settings of PyTorch:

- Adam, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$
- RMSProp, $\alpha = 0.99$, $\epsilon = 10^{-8}$
- Momentum SGD, $\beta = 0.9$ (with Nesterov momentum off)

B.2. Figure 3, 4, 10, 12, and 13

Figure 3 is obtained by training models for 500k steps with a standard DQN architecture (Mnih et al., 2013): 3 convolutional layers with kernels of shape $4 \times 32 \times 8 \times 8$, $32 \times 64 \times 4 \times 4$, and $64 \times 64 \times 3 \times 3$ and with stride 4, 2, and 1 respectively, followed by two fully-connected layers of shape 9216×512 and $512 \times |\mathcal{A}|$, $\mathcal{A}$ being the legal action set for a given game. All activation are leaky ReLUs except for the last layer which is linear (as it outputs value functions). Experiments are run on MsPacman, Asterix and Seaquest for 10 runs each. A learning rate of 10^{-4} is used, with L2 weight regularization of 10^{-4}. We use $\gamma = 0.99$, a minibatch size of 32, an ϵ of 5% to generate $\mathcal{D}^*$, and a buffer size of 500k. The random seeds affect the generation of $\mathcal{D}^*$, the weight initialization, the minibatch sampling, and the choice of actions in ϵ-greedy rollouts.

As per the previous figure, for Figure 4 we run experiments with a standard DQN architecture, train our policy evaluation models for 500k and our control models for 10M steps. When boostrapping to a frozen network, the frozen network is updated every 10k updates.

Figures 10, 12, and 13 also use results from these experiments.

B.3. Figure 5, 11, and 13

The experiments of Figure 5 are run for 500k steps, as previously described, on MsPacman. λ-targets are computed with the forward view, using the frozen network to compute the target values – this allows us to cheaply recompute all λ-targets once every 10k steps when we update the frozen network. Each setting is run with 5 random seeds.

Figures 11 and 13 also use results from these experiments.

B.4. Figure 6

Figure 6 reuses the results of Figure 4's policy evaluation experiments run with Adam.

B.5. Figure 7

Figure 7 uses the same experiment setup as in the Atari regression experiments on MsPacman, as well as policy evaluation experiments on MsPacman as previously described, all the while measuring individual terms of ρ'_{reg} and $\rho'_T D$. Experiments are only run for the first 100k steps. Minibatches of size 32 are used.

C. Reproducibility checklist

We follow the Machine Learning reproducibility checklist (Pineau, 2019), and refer to corresponding sections in the text when relevant.

For all models and algorithms presented, check if you include:

- **A clear description of the mathematical setting, algorithm, and/or model.** We use unmodified algorithms, described in the technical background, and only analyse their behaviour. The measures we propose are straightforward to implement and only require minimal changes. For more details see section B.
- **An analysis of the complexity (time, space, sample size) of any algorithm.** The measures we propose only add a constant instrumentation overhead.
- **A link to a downloadable source code, with specification of all dependencies, including external libraries.** All code is included in supplementary materials, dependencies are documented within.

For any theoretical claim, check if you include:

- **A statement of the result.** See section A.
- **A clear explanation of any assumptions.** idem.
- **A complete proof of the claim.** idem.

For all figures and tables that present empirical results, check if you include:

- **A complete description of the data collection process, including sample size.** We collect data by running standard implementations of common algorithms with repeated runs.
- **A link to a downloadable version of the dataset or simulation environment.** Included in the code available in supplementary materials.
- **An explanation of any data that were excluded, description of any pre-processing step.** We generally chose hyperparameters that best represent state-of-the-art usage, then if necessary that best represent our findings. In most cases only minor learning rate adjutments were necessary, although they would not significantly change most plots.
- **An explanation of how samples were allocated for training / validation / testing.** We use standard train/valid/test splits as per the literature of each dataset.
- **The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.** See section B.
- **The exact number of evaluation runs.** idem.
- **A description of how experiments were run.** idem.
- **A clear definition of the specific measure or statistics used to report results.** See section 2.1.
- **Clearly defined error bars.** Figures with error bars compute a bootstrapped 90% or 95% confidence interval of the mean. We only use 90% for Figure 9 because of the many outliers.
- **A description of results with central tendency(e.g. mean) & variation(e.g. stddev).** idem.
- **A description of the computing infrastructure used** Almost all experiments were run on P100 and V100 GPUs, otherwise they were run on Intel i7 processors.

D. Extra figures

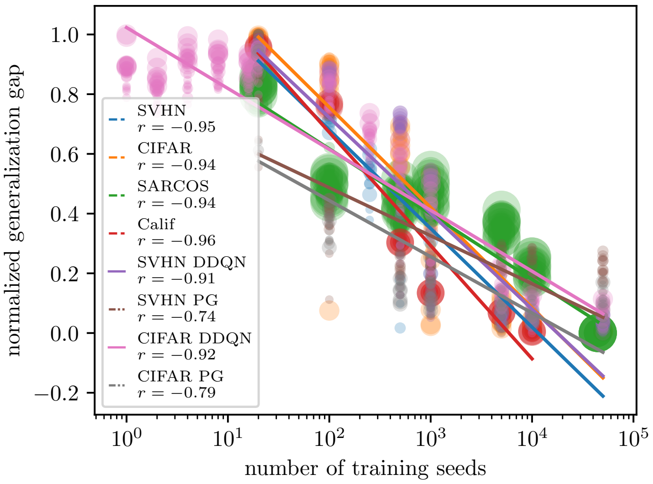

Figure 9. Generalization gap vs number of training seeds. The size of each circle (which represents a single experiment) is proportional to the number of hidden units.

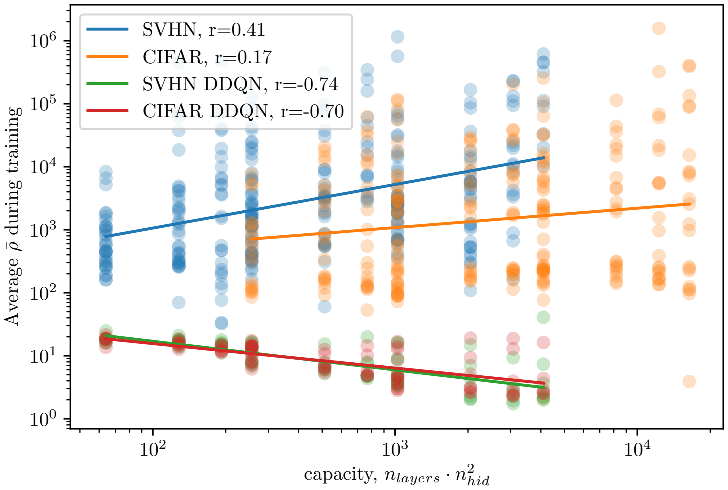

Figure 10. Average function interference during training as a function of capacity. TD methods and classifiers have very different trends.

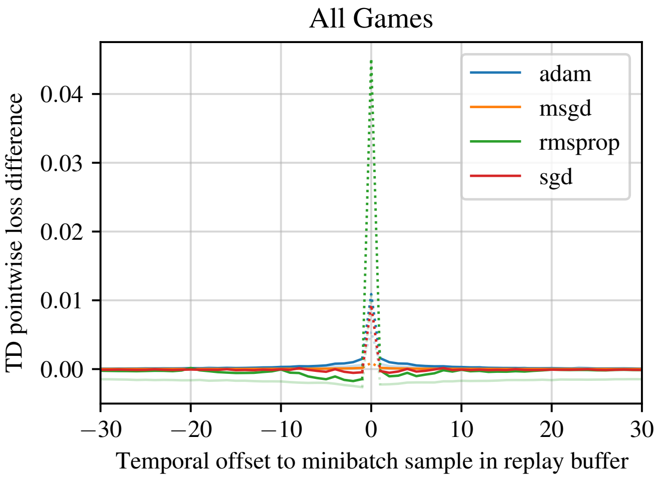

Figure 11. Reproduction of Figure 4 including $x = 0$. RMSprop has a surprisingly large expected gain at $x = 0$, but a negative gain around $x = 0$, suggesting that RMSprop enables memorization more than Adam.

D.1. Singular values, control vs policy evaluation

Figure 12 shows the spread of singular values after 100k minibatch updates on MsPacman for the Q-Learning objective and Adam/RMSProp. The difference between the control case and policy evaluation supports our hypothesis that policy evaluation initially captures more factors of variation. It remains unclear if the effect of the control case initially having fewer captured factors of variation leads to a form of feature curriculum.

Figure 13 shows the spread of singular values after 500k minibatch updates for TD(λ). Interestingly, larger λ values yield larger singular values and a wider distribution. Presumably, TD(λ) having a less biased objective allows the parameters to capture all the factors of variation faster rather than to rely on bootstrapping to gradually learn them.

Note that current literature suggests that having fewer large singular values is a sign of generalization *in classifiers*, see in particular Oymak et al. (2019), as well as Morcos et al. (2018) and Raghu et al. (2017). It is not clear whether this holds for regression, nor in our case for value functions. Interestingly all runs, even for TD(λ), have a dramatic cutoff in singular values after about the 200th SV, suggesting that there may be in this order of magnitude many underlying factors in MsPacman, and that by changing the objective and the data distribution, a DNN may be able to capture them faster or slower.

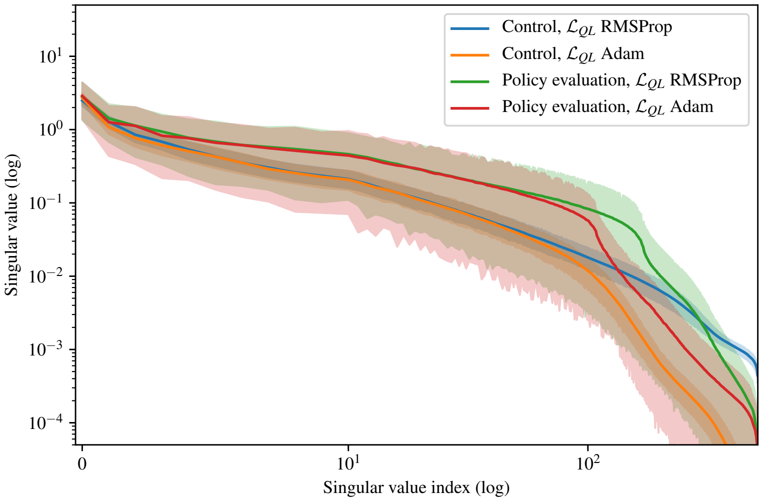

Figure 12. Spread of singular values after 100k iterations. Despite having seen the same amount of data, the control experiments generally have seen fewer unique states, which may explain the observed difference. Shaded regions show bootstrapped 95% confidence intervals.

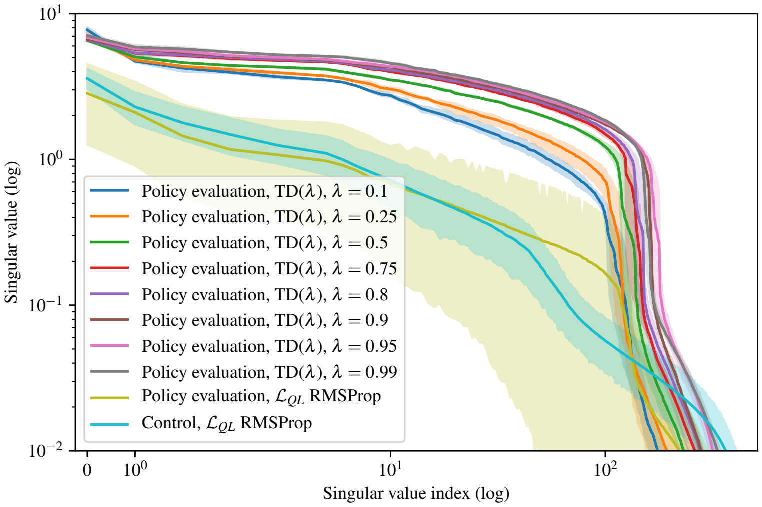

Figure 13. Spread of singular values after 500k iterations. Shaded regions show bootstrapped 95% confidence intervals.

D.2. Evolution of TD gain with training

Figure 14 shows the evolution of TD pointwise loss difference during training; in relation to previous figures like Figure 4, the y axis is now Fig. 4's x axis – the temporal offset to the update sample in the replay buffer, the y axis is now training time, and the color is now Fig. 4's y axis – the magnitude of the TD gain.

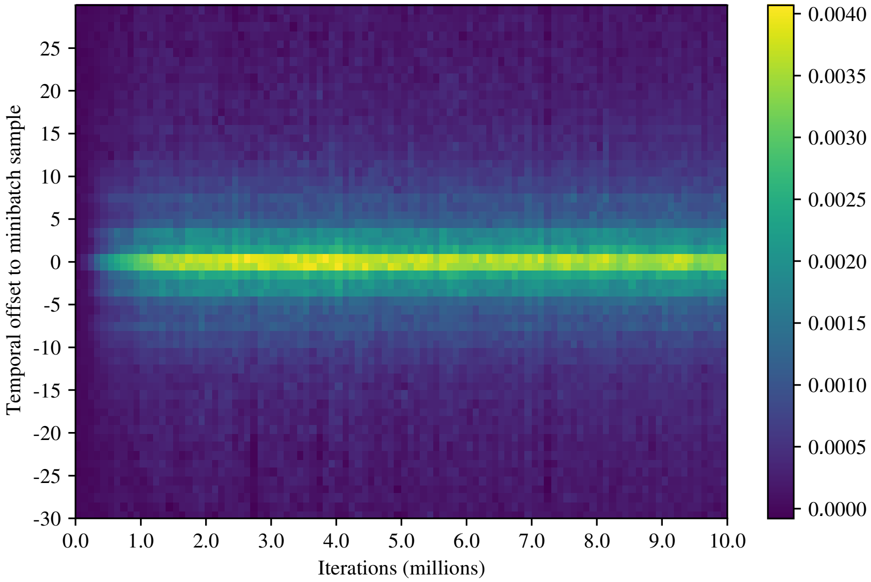

Figure 14. Evolution of TD pointwise loss difference, during training. Control experiment with Adam, MsPacman, averaged over 10 runs. Note that index 0 is excluded as its magnitude would be too large and dim all other values.