HANQ: Hypergradients, Asymmetry, and Normalization for Fast and Stable Deep Q-Learning

Anonymous authors
Paper under double-blind review

Keywords: off-policy reinforcement learning (RL), offline RL, temporal difference learning, bootstrapping, instability, return degradation, value estimation

Summary

In reinforcement learning, deep Q-learning algorithms are often more sample- and compute-efficient than alternatives like the Monte Carlo policy gradient, but tend to suffer from instability that limits their use in practice. Some of this instability can be mitigated through a delayed *target network*, yet this usually slows down convergence. In this work, we explore the possibility of stabilization without sacrificing the speed of convergence. Inspired by self-supervised learning (SSL) and adaptive optimization, we empirically arrive at three modifications to the standard deep Q-network (DQN) — no two of which work well alone in our experiments. These modifications are, in the order of our experiments: 1) an **A**symmetric *predictor* in the neural network, 2) a particular combination of **N**ormalization layers, and 3) **H**ypergradient descent on the learning rate. Aligning with prior work in SSL, **HANQ** (pronounced "*hank*") avoids DQN's target network, uses the same number of hyperparameters as DQN, and yet matches or exceeds DQN's performance in our experiments on three out of four environments.

Contribution(s)

1. We propose to replace the target network in deep Q-network (DQN) with an asymmetric predictor and normalization layers to stabilize training. Empirical results suggest the promise of our approach given appropriate learning rate tuning.
   Context: Asymmetric architectures have been explored in self-supervised learning (Grill et al., 2020; Chen & He, 2021) and reinforcement learning (RL) (Pitis et al., 2020; Guo et al., 2022; Liu et al., 2022; Tang et al., 2023; Wang, 2024; Eysenbach et al., 2024; Amortila et al., 2024; Myers et al., 2025). However, to our knowledge, all prior RL works study auxiliary losses or goal-based RL, and typically keep the target network and increase the total hyperparameters. We study pure end-to-end reward maximization without a target network, without increasing the total hyperparameters.
2. Noting that promise of our first contribution, we use hypergradient descent for that tuning, which achieves stable convergence without compromising the convergence rate. In our experiments, our algorithm (HANQ) matches or outscores DQN in three of four environments.
   Context: Prior works investigate hypergradients for temporal difference learning (Sutton, 2022), but in our experiments using hypergradient descent alone (or asymmetry alone) scores poorly.
3. Our extensive ablations suggest each component of HANQ is important for its high scores.
   Context: None.

HANQ: Hypergradients, Asymmetry, and Normalization for Fast and Stable Deep Q-Learning

Anonymous authors
Paper under double-blind review

Abstract

In reinforcement learning, deep Q-learning algorithms are often more sample- and compute-efficient than alternatives like the Monte Carlo policy gradient, but tend to suffer from instability that limits their use in practice. Some of this instability can be mitigated through a delayed *target network*, yet this usually slows down convergence. In this work, we explore the possibility of stabilization without sacrificing the speed of convergence. Inspired by self-supervised learning (SSL) and adaptive optimization, we empirically arrive at three modifications to the standard deep Q-network (DQN) — no two of which work well alone in our experiments. These modifications are, in the order of our experiments: 1) an **A**symmetric *predictor* in the neural network, 2) a particular combination of **N**ormalization layers, and 3) **H**ypergradient descent on the learning rate. Aligning with prior work in SSL, **HANQ** (pronounced "*hank*") avoids DQN's target network, uses the same number of hyperparameters as DQN, and yet matches or exceeds DQN's performance in our experiments on three out of four environments.

1 Introduction

Temporal difference (TD) algorithms such as Q-learning often improve sample- and compute-efficiency compared to Monte Carlo algorithms. Unfortunately, TD algorithms are more unstable, frequently learning *worse* policies when trained for *longer* (Agarwal et al., 2019; Brandfonbrener et al., 2021; Kumar et al., 2021). The most common stabilization approach requires delaying the update of the *target*, the values they bootstrap from. For example, the standard deep Q-learning algorithm, DQN (Mnih et al., 2015), uses a *target network* (a lagging copy of its main network weights) to slow down target updates. Yet, the target update rates typically used in practice are often not slow enough to fix instability, even though they already slow convergence (Agarwal et al., 2019; Brandfonbrener et al., 2021; Kumar et al., 2021).

Recent works (Gallici et al., 2024; Elsayed et al., 2024; Bjorck et al., 2021) suggest that normalizations can stabilize Q-learning, and TD learning broadly, without delayed targets. However, it is not yet clear if any approach is so fast and stable as to make delayed targets obsolete. In parallel, other recent works (Guo et al., 2020; Kumar et al., 2021) note similarity between TD learning and *self-supervised learning* (SSL), a field that aims to learn representations of data that make downstream tasks more efficient. Some SSL works have found architectural asymmetries and normalizations necessary for good results (Chen & He, 2021; Zhang et al., 2022). Asymmetries have also been found useful in RL (Liu et al., 2022; Wang et al., 2023; Tang et al., 2023; Eysenbach et al., 2024; Khetarpal et al., 2024). However, perhaps due to tuning requirements or the need for additional empirical evidence, none have fully replaced the standard architectures in RL.

In this work, we focus on *offline* RL, as it is easier to test on real-world data and can amplify the instability we study. Compared to online RL, offline RL is particularly valuable when new data is costly. Examples include autonomous vehicle data, medical data, and human expert data. Here, letting suboptimal policies collect training data can cost too much time, money, or even lives. This also means we cannot afford to frequently measure the returns of the policies during training. As a result, algorithms that only temporarily reach high return during training may output a poor policy.

Via asymmetries and normalizations similar to SSL algorithms, we aim for fast, stable, and high return in RL without more hyperparameters than DQN. In preliminary experiments, we find that a particular asymmetry greatly improves DQN's stability when not using a target network. Those experiments also suggest that, when using asymmetry, an adaptive learning rate might yield more returns. We show hypergradient optimization achieves this. Similar to SSL, adding normalizations and more asymmetric elements further increases return. We ablate all three components (hypergradients, asymmetry, and normalization), finding that all three are important for high return.

2 Background

We consider the Markov decision process (MDP) formulation for RL. Let $\mathcal{S}$ be a state space, $\mathcal{A}$ a finite action space, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a reward function, and $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ a transition function. We assume an offline dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$ has already been collected. In the dataset, each tuple (s_i, a_i, r_i, s_i') is a state $s_i \in \mathcal{S}$, an action $a_i \in \mathcal{A}$ taken in that state, the resulting reward $r_i \in \mathbb{R}$ drawn with $\mathbb{E}[r_i \mid s_i, a_i] = R(s_i, a_i)$, and next state $s_i' \in \mathcal{S}$ drawn from $s_i' \sim P(\cdot \mid s_i, a_i)$. We aim to learn a policy $\pi\colon \mathcal{S} \to \mathcal{A}$ to maximize the expected, discounted return (cumulative rewards), $\mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\big]$, from any starting state s_0, where $\gamma \in [0, 1)$ is the discount factor. The Q-function for a policy π is $Q^\pi(s, a) := \mathbb{E}^\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a]$, where $\mathbb{E}^\pi[\cdot]$ is the expectation over trajectories from π. The optimal Q-function is $Q^\star(s, a) := \max_\pi Q^\pi(s, a)$.

Q-learning and Its Instability. A standard algorithm to approximate the optimal value function $Q^\star$ is Q-learning (Watkins, 1989). For the finite-state, finite-action case, Q-learning is guaranteed to converge to $Q^\star$ if the dataset $\mathcal{D}$ is sufficiently exploratory (Watkins & Dayan, 1992). With function approximation, convergence is no longer guaranteed. Let $Q_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a parameterized function. Given a (s, a, r, s') tuple from $\mathcal{D}$[1], consider the following update based on the mean square Bellman error (MSBE):

$$\theta \overset{-}{\leftarrow} \alpha \nabla_\theta \left(Q_\theta(s, a) - r - \gamma \mathsf{sg}\left[\max_{a' \in \mathcal{A}} Q_\theta(s', a') \right] \right)^2 \tag{1}$$

where $f \overset{-}{\leftarrow} g$ means $f \leftarrow f - g$, α is the learning rate, and $\mathsf{sg}[\cdot]$ is the stop-gradient operator. The stop-gradient means any function of θ in $[\cdot]$ will be treated as constant under the gradient operation. Eq. (1) is unstable in general — it can diverge even under the linear function approximation $Q_\theta(s, a) = \phi(s, a)^\top \theta$ for some known feature $\phi(s, a) \in \mathbb{R}^d$. Some counterexamples provably diverge for any α (Baird, 1995; Tsitsiklis & Van Roy, 1996; Sutton & Barto, 2018). Nonlinear function approximators can introduce further instability (Tsitsiklis & Van Roy, 1997; Ollivier, 2018; Brandfonbrener & Bruna, 2019; Gallici et al., 2024).

To address instability, TD algorithms often use a *target network* (Mnih, 2013; Mnih et al., 2015; Lillicrap et al., 2015). They maintain two copies of parameters $\theta, \bar{\theta}$, in each iteration updating

$$\theta \overset{-}{\leftarrow} \alpha \nabla_\theta \left(Q_\theta(s, a) - r - \gamma \max_{a' \in \mathcal{A}} Q_{\bar{\theta}}(s', a') \right)^2, \qquad \bar{\theta} \leftarrow (1 - \beta)\bar{\theta} + \beta\theta, \tag{2}$$

where $Q_{\bar{\theta}}$ is the target network and β is the rate at which it is updated. Eq. (2) stabilizes training by slowing the movement of the target $r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a')$ due to the movement of θ. Notice that Eq. (1) is equivalent to Eq. (2) with $\beta = 1$. Although the target network helps stabilize learning, it also often slows down the overall algorithm. For example, on some environments, replacing the target network with alternative stabilization methods can give algorithms that reach equally high scores in fewer iterations (Gallici et al., 2024).

[1] In fact, a mini-batch of (s, a, r, s') tuples is sampled. We present the version with only one sample (i.e., mini-batch size $= 1$) for ease of exposition. Similar for the rest of the paper.

In fact, even a small β is not always enough to fully stabilize training: on many RL problems, the policy's quality, i.e. the return it can achieve, often drops at some point in training and does not recover. Usually, when this *return degradation* occurs, the training loss also diverges. A common countermeasure is to modify the training loss so that the learned policy stays close to the behavior policy used to collect data. However, even when this pessimism approach succeeds at stabilization, it often reduces the return compared to the peak return temporarily reached in the unstabilized training. A temporary high peak return is impractical in offline RL because, in real-world problems, constantly measuring the returns during training is costly.

Can we achieve stability without slowing down Q-learning? The mentioned counterexamples rely on linear function approximation for divergence. The possibility of improvement with feature learning, where $\phi(s, a)$ may change during training, remains open. We propose a particular combination of methods that, in our experiments, mitigates the downsides of removing the target network.

SSL with Asymmetry and Normalization. As our work aims to leverage the power of feature learning to stabilize Q-learning, we draw inspiration from feature learning schemes outside RL. A class of relevant approaches are **B**ootstrap **Y**our **O**wn **L**atent (BYOL) (Grill et al., 2020) and **s**imple **S**iamese networks (SimSiam) (Chen & He, 2021) for self-supervised learning (SSL), whose original goal is to learn representations for images. Fig. 1 illustrates a simplification of BYOL's architecture. In both methods, an input image is randomly augmented into two views x, x', which are then individually encoded by the encoders f_θ and $f_{\bar{\theta}}$, respectively, yielding $z_\theta = f_\theta(x)$ and $z'_{\bar{\theta}} = f_{\bar{\theta}}(x')$. Then, an asymmetric *predictor* h_ω transforms the first output into $p_{\omega,\theta} = h_\omega(z_\theta)$ and tries to match it to the other output $z'_{\bar{\theta}}$ by minimizing their ℓ_2 distance under ℓ_2-normalization: $\ell(p_{\omega,\theta}, z'_{\bar{\theta}}) = \left\| \frac{p_{\omega,\theta}}{\|p_{\omega,\theta}\|_2} - \frac{z'_{\bar{\theta}}}{\|z'_{\bar{\theta}}\|_2} \right\|^2$. BYOL's update is

$$(\omega, \theta) \overset{-}{\leftarrow} \alpha \nabla_{\omega,\theta} \ell(p_{\omega,\theta}, z'_{\bar{\theta}}), \quad \bar{\theta} \leftarrow (1-\beta)\bar{\theta} + \beta\theta. \tag{3}$$

SimSiam removes the delayed update of $\bar{\theta}$. That is, it shares the parameters in the two branches in Fig. 1 and updates

$$(\omega, \theta) \overset{-}{\leftarrow} \alpha \nabla_{\omega,\theta} \ell(p_{\omega,\theta}, \mathsf{sg}[z'_\theta]). \tag{4}$$

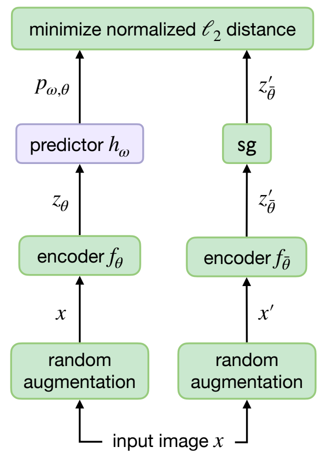


Figure 1: Asymmetrically added neural net weights, collectively called a *predictor*, are often key in SSL. A more symmetric approach would use only the green components, but is unstable. We find preliminary evidence that such asymmetry might be similarly key for RL.

SimSiam is BYOL with $\beta = 1$. Eq. (3) and Eq. (4) are similar to Eq. (2) and Eq. (1), respectively, with $p_{\omega,\theta}$ corresponding to $Q_\theta(s, a)$ and z'_θ corresponding to $r + \gamma \max_{a'} Q_\theta(s', a')$.

As reported by Grill et al. (2020) and Chen & He (2021), BYOL and SimSiam can learn meaningful representation, even though a collapsing solution that encodes everything into the same vector is a clear minimizer of $\ell(p_{\omega,\theta}, z'_{\bar{\theta}})$. To our knowledge, there still lacks a satisfying explanation for why BYOL or SimSiam avoids collapses. In previous attempts (Chen & He, 2021; Zhang et al., 2022; Wen & Li, 2022; Richemond et al., 2023; Tang et al., 2023), the theory either remains to be high-level or makes extra assumptions that are not required by the algorithm.

However, one intriguing observation is that while Eq. (1) generally fails in RL, the similar update Eq. (4) of SimSiam succeeds in SSL. This leads to the question: *Can we make Eq.* (1) *more similar to SimSiam to facilitate its convergence in RL?* As argued in Chen & He (2021); Zhang et al. (2022); Wen & Li (2022), the predictor h_ω that asymmetrizes the two branches is key for preventing collapse. Incorporating this idea into Eq. (1) yields the update

$$(\omega, \theta) \overset{-}{\leftarrow} \alpha \nabla_{\omega,\theta} \left(h_\omega(Q_\theta(s, a)) - r - \gamma \mathsf{sg}\left[\max_{a' \in \mathcal{A}} Q_\theta(s', a') \right] \right)^2$$

where h_ω is the predictor an extra layer for Q_θ. This is the starting point of our algorithm design.

Besides asymmetry from the predictor, another critical element in SimSiam and BYOL is *normalized* ℓ_2 loss, as shown in Fig. 1. Normalizing in the loss is not applicable to Q-learning (Eq. (1) or Eq. (2)) since Q-learning values are scalars, but this suggests that normalization elsewhere could be important.

3 HANQ: Three Components

Now we introduce the three components of HANQ, a modified Q-learning algorithm that aims to achieve both stability and fast convergence. We discuss each component individually in the following subsections, deferring ablations to Section 4.

We compare policies by ***score*** — the empirical return when deployed, normalized for readability. Roughly the least return on an environment setup is 0, and 100 roughly the most (details: Section 11).

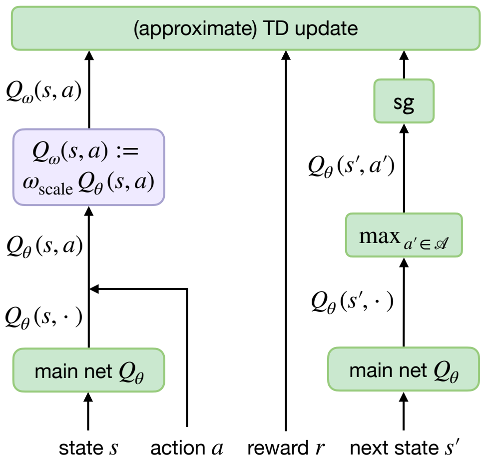


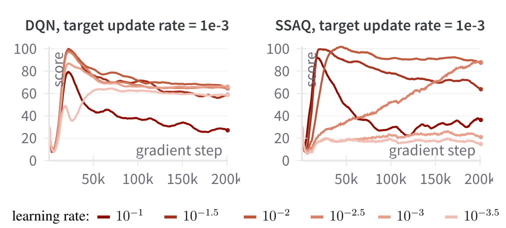
Figure 2: SSAQ modifies DQN by adding a predictor, and by using the main weights θ for $Q(s',a')$ instead of using delayed target net parameters $\bar{\theta}$. SSAQ's predictor is a single, learned weight, ω_{scale}. (A one-unit layer.)

3.1 Component 1: Asymmetry

Motivated by the success of SimSiam and BYOL, we start by adding a simple predictor to the standard DQN. Specifically, we only add a single, learned scaling parameter ω to the output of the main Q-network, Q_θ. We also reuse the main Q-network for both terms in the loss, rather than using a target network $Q_{\bar{\theta}}$ for the target term. This results in the following update (cf. Eq. (2)):

$$(\omega,\theta) \leftharpoonup \alpha\nabla_{\omega,\theta}\left(\omega Q_\theta(s,a) - r - \gamma \text{sg}\left[\max_{a'\in\mathcal{A}} Q_\theta(s',a')\right]\right)^2. \qquad \textbf{(SSAQ)}$$

We call this algorithm SSAQ (Single-Scaler Asymmetric-Q), show its architecture in Fig. 2, and its pseudocode in Section 7. We compare DQN and SSAQ on an offline RL benchmark where DQN is known to have return degradation. The benchmark is a discrete-action version of the classic control problem *Pendulum* (Brockman et al., 2016; Xiao et al., 2022; Snyder et al., 2023). Following prior work (Xiao et al., 2022; Snyder et al., 2023), we collect an offline dataset of 1000 samples of state-action pairs, using a uniformly sampled initial state, taking uniformly random actions.



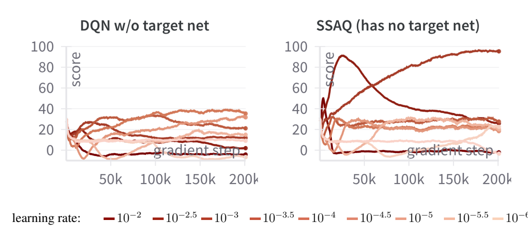


Figure 3a: Every figure uses 30 seeds. Scores (episodic return normalized for readability) on Pendulum. *Left*: DQN's score degrades over gradient steps. *Right*: SSAQ with a target net. SSAQ stabilizes scores at smaller learning rates, yet slows convergence.

Figure 3b: The same as Fig. 3a, but without target nets. (From here on, we test our new algorithms only without target nets.) *Left*: DQN scores poorly. *Right*: SSAQ can score highly sometimes, but remains sensitive to the learning rate.

Fig. 3a shows the scores of DQN and SSAQ when *keeping* the target network. For both algorithms, we use an intermediate target update rate of $\beta = 10^{-3}$ here, because we find it gives DQN the highest gradient-step-averaged score (which we discuss later). Empirically, compared to DQN, SSAQ changes the effect of tuning the learning rate. With SSAQ, a large learning rate like 10^{-1} converges quickly, but also diverges quickly. A smaller learning rate like 10^{-2} converges slowly compared to 10^{-1}, but gains stability. In contrast, DQN remains unstable over all learning rates. This makes us conjecture that the asymmetric element ω takes a role similar to delaying the target update. This view has been shared by SimSiam (Chen & He, 2020). In preliminary experiments (not shown), placing the predictor on the $Q(s', a')$ loss path scored no better than on the $Q(s, a)$ loss path. This may align with Zhang et al. (2022). As a result, we test only the latter placement.

Could a predictor avoid the need for a target network? Fig. 3b gives some evidence. When neither algorithm uses a target network, SSAQ's peak score is over double DQN's. Removing a target network in general would not only avoid the need to tune the delay hyperparameter, but might also avoid delay to the overall optimization. SimSiam's success in SSL, without a target network, provides additional evidence that this might be possible in RL as well.

Relation to adaptive discount factors. Scaling $Q(s, a)$ relates to modifying the discount factor (see Section 8), and using a smaller discount factor is a regularization (Jiang et al., 2015). Thus, SSAQ relates to adaptive regularization. Later, we make ω state-dependent, which relates to state-dependent discount factors (Rathnam et al., 2024). Given these connections between asymmetric architectures and discount factors, we test discount tuning in Section 9.3.

3.2 Component 2: Normalization

Next, given the importance of normalizations in SSL, we try feature normalizations: ℓ_2-normalization and Layer Normalization (LayerNorm). Unfortunately, neither help SSAQ in our experiments (Fig. 4a). See Section 10 for details, including some of the many supporting prior works in SSL and RL.

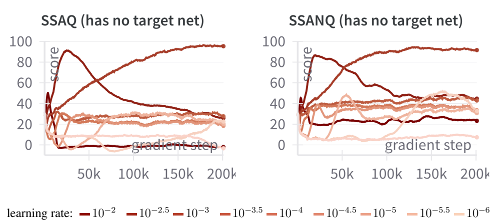


Figure 4a: SSAQ (duplicated for reference) vs. SSAQ with LayerNorm (which we call SSANQ). LayerNorm has barely any noticeable effects here.

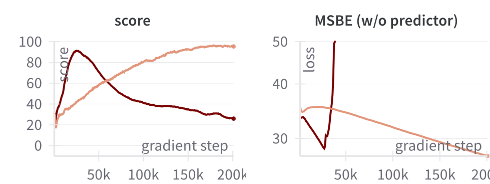


Figure 4b: SSAQ score (duplicated for reference) and MSBE, for only the two best learning rates, $\boxed{10^{-2.5}}$ and $\boxed{10^{-3}}$. MSBE anticorrelates with score, across learning rates and across gradient steps.

3.3 Component 3: Hypergradients

Compared to plain DQN, SSAQ enables *either* faster or stabler convergence on Pendulum (Fig. 3a, Fig. 3b). That tradeoff of speed vs stability for SSAQ is greatly controlled by the learning rate, whereas DQN's learning rate does not appear to control that tradeoff much. Further, the MSBE (i.e., the value of $(Q_\theta(s, a) - r - \gamma \max_{a'} Q_\theta(s', a'))^2$ on training data) anticorrelates with the score across learning rates and across gradient steps (Fig. 4b). It is tempting to try to automatically adjust SSAQ's learning rate during training, to get *both* speed and stability. We attempt this with hypergradient optimization, which optimizes hyperparameters using gradient descent.

The hypergradient tunes the learning rate as

$$\alpha_{i+1} \leftarrow \alpha_i - \kappa \frac{\partial \widetilde{\mathcal{L}}(\theta_i)}{\partial \alpha_i},$$

where α_i is the learning rate used in iteration i, κ is the hyperlearning rate, θ_i is the neural net weights in iteration i (for simplicity, here we use θ_i for *all* weights in the network, i.e., both the θ and ω

described in previous sections), and $\widetilde{\mathcal{L}}$ is the hyperoptimization loss function, which is not necessarily the same as the main loss function. In Section 12, we consider two forms of $\widetilde{\mathcal{L}}$, deriving two ways to tune the learning rates. They are

$$\text{(i)}\ \ \alpha_{i+1} \leftarrow \alpha_i - \kappa\,(\mathrm{SG}_i \cdot -\mathrm{SG}_{i-1}) \qquad \text{and} \qquad \text{(ii)}\ \ \alpha_{i+1} \leftarrow \alpha_i - \kappa\,(\mathrm{RG}_i \cdot -\mathrm{SG}_{i-1})\,,$$

where $\mathrm{RG}_i = \frac{\partial \mathcal{L}_{\mathrm{RG}}(\theta_i)}{\partial \theta_i}$ and $\mathrm{SG}_i = \frac{\partial \mathcal{L}_{\mathrm{SG}}(\theta_i)}{\partial \theta_i}$, with $\mathcal{L}_{\mathrm{RG}}(\theta) \triangleq (Q_\theta(s,a) - r - \gamma \max_{a'} Q_\theta(s',a'))^2$ and $\mathcal{L}_{\mathrm{SG}}(\theta) \triangleq (Q_\theta(s,a) - r - \gamma\mathsf{sg}[\max_{a'} Q_\theta(s',a')])^2$.

Due to high scores in preliminary experiments (Section 9.2), we use (ii). We test only deterministic environments for simplicity, avoiding double sampling bias (Baird, 1995). After we add hyperoptimization to SSAQ, we call it HSSAQ. HSSAQ takes one hypergradient step (to update the learning rate) after every standard gradient step (to update the Q-network weights, including the metapredictor and predictor).

Might hypergradient optimization give SSAQ both fast and stable scores? HSSAQ indeed automatically decreases the learning rate, adding some stability without sacrificing the early returns of larger learning rates (Fig. 5). However, HSSAQ tends not to reach scores quite as high or as quickly as DQN can (before DQN diverges).

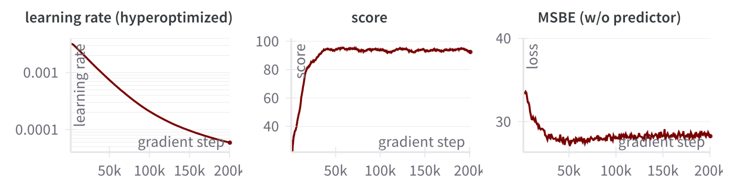


Figure 5: HSSAQ with its best initial learning rate, $\boxed{10^{-2.5}}$ (and hyperlearning rate $\kappa = 10^{-4}$). HSSAQ's hypergradient can stabilize the score and MSBE greatly, but HSSAQ does not quite reach DQN's peak score (~100, Fig. 3a). For consistency with earlier plots, we omit error bars.

3.4 Revisiting Component 1: Additional Asymmetric Elements

Inspired by the benefits of larger predictors in SSL (Zhang et al., 2022), we test the same for RL. To avoid adding hyperparameters, we avoid auxiliary losses such as self-predicting latent representations (Gelada et al., 2019). The simplest effective approach we found for adding more parameters is to add a *metapredictor*. The metapredictor outputs the ω_{scale} parameter of the predictor, which is then used as before in the TD update (like SSAQ), with end-to-end training as usual.

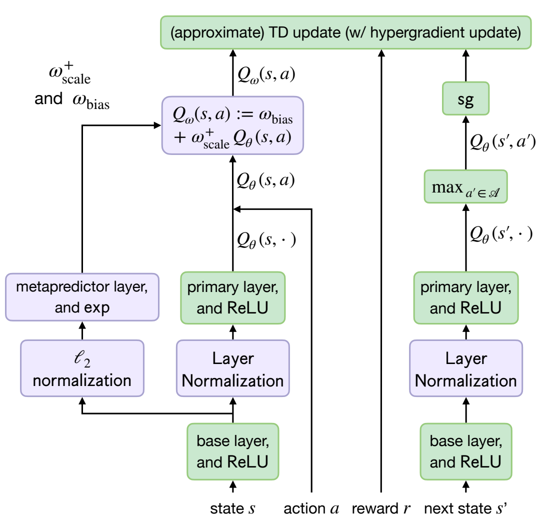


Figure 6a: HANQ modifies DQN by: adding a metapredictor, which outputs the predictor; adding normalizations; and adapting the learning rate for the TD update, using hypergradients.

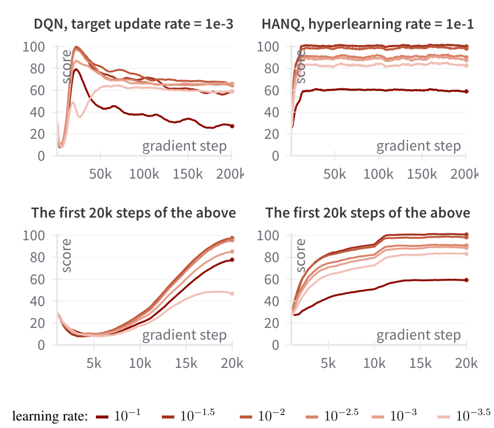


Figure 6b: *Top*: Like Fig. 3a, but DQN vs. HANQ instead of DQN vs. SSAQ. HANQ's scores are more stable than DQN over gradient steps. *Bottom*: Zoomed in to the first 20k gradient steps, showing HANQ also learns more quickly than DQN. *Overall*: Note that HANQ has *the same number of hyperparameters* as DQN.

We combine **H**ypergradient optimization with this metapredictor **A**symmetry, along with the **N**ormalization layers discussed above, and call the combined $\boldsymbol{Q}$-learning algorithm **HANQ**. HANQ reaches high scores both faster and more stably than DQN on our Pendulum problem. Fig. 6a shows HANQ's architecture, and Fig. 6b compares the scores of DQN and HANQ.

4 Further Experiments

Unless marked otherwise, we use 30 seeds, tune baselines extensively (Section 11), and show only the best score per algorithm–configuration–environment combination. That is, each table cell gives only the best score of e.g. the 7 learning rates we usually tune over, combined with tuning over e.g. DQN's β, unless stated otherwise. Recall, *scores* are the return normalized for readability. We define *score* in Section 11. Unlike our graphs, scores in all tables are averaged over all gradient steps within each training run. This measures both training speed and stability.

Table 1: Confidence intervals (CIs) overlapping the CI of the top mean are highlighted (Patterson et al., 2023). Scores averaged over gradient steps. Recall, β is the target update rate (DQN only), and κ is the hyperlearning rate (HANQ only). "**DQN-LN**" and "**DQN-ℓ_2N**" are DQN with LayerNorm or ℓ_2-normalization.

	Best $\beta \in \{10^0, 10^{-1}, \dots, 10^{-4}\}$			$\beta = 10^0$			$\kappa = 10^{-1}$
	DQN	**DQN-LN**	**DQN-ℓ_2N**	**DQN**	**DQN-LN**	**DQN-ℓ_2N**	**HANQ**
Pendulum	72.4	83.0	86.1	30.0	40.1	31.9	100.1
(95% CI)	(61.0, 82.5)	(77.6, 87.9)	(80.9, 91.0)	(22.1, 37.0)	(27.0, 53.6)	(20.1, 43.0)	(98.0, 102.2)
Acrobot	91.9	90.6	89.6	67.9	90.0	80.0	90.1
(95% CI)	(90.3, 93.5)	(89.0, 92.0)	(87.3, 91.7)	(61.7, 73.2)	(87.9, 92.0)	(77.1, 82.4)	(88.0, 92.0)
CartPole	55.6	49.5	50.9	41.1	47.7	36.5	24.0
(95% CI)	(51.3, 60.1)	(47.8, 51.2)	(43.7, 58.7)	(39.1, 43.6)	(45.3, 49.9)	(33.0, 40.3)	(19.1, 30.2)

HANQ vs Standard Algorithms: Classic Control. In our experiments, on two of the three total classic control environments we test, HANQ matches or outscores DQN (Table 1). On the third environment, CartPole, HANQ and all other asymmetries score poorly in our experiments. They often learn large predictor parameters (not shown). Their ω_{scale} values reach, e.g., 1.5, which may relate to discount factors (Section 8) that are too small. Due to those consistently low scores, we exclude CartPole from the remaining ablations, leaving the issue for future work.

PQN. Gallici et al. (2024) propose parallel Q-network, which avoids target networks by LayerNorm and ℓ_2-regularization. The best configuration we test (Section 11) scores 43.3 (CI $30.2, 57.2$), far below HANQ's 100.1 (CI $98.0, 102.2$) and comparable to using no regularization (DQN-LN with $\beta = 10^0$ in Table 1).

Table 2: Ablating HANQ's predictor does not improve scores.

	Pendulum	(95% CI)	Acrobot	(95% CI)
HANQ	100.1	(98.0, 102.2)	90.1	(88.0, 92.0)
w/o ω^+_{scale}	84.3	(78.2, 90.3)	65.7	(50.7, 79.2)
w/o ω_{bias}	94.8	(88.4, 99.5)	90.5	(88.7, 91.9)
w/o metapred.	93.6	(88.9, 97.7)	90.7	(88.0, 93.1)
w/o any pred.	43.5	(28.6, 56.7)	91.7	(90.2, 93.2)
symmetrized	39.2	(25.1, 54.0)	93.6	(92.2, 94.7)

HANQ's Predictor. Our experiments suggest two changes to SSAQ's predictor, both of which we use in HANQ (as shown in Fig. 6a). First, HANQ forces the predictor's scaler parameter to be positive by optimizing $\omega_{\text{scale}} \in \mathbb{R}$, and using $\omega^+_{\text{scale}} := \exp(\omega_{\text{scale}})$ in the predictor. Second, HANQ also learns a bias parameter ω_{bias} for the predictor. Table 2 shows the scores of HANQ again, compared with excluding either of those two changes ("**w/o ω_{scale}**" and "**w/o ω_{bias}**"), learning those predictor parameters directly instead of a metapredictor (“**w/o metapred.**”), excluding all asymmetry (“**w/o any pred.**”), or using the metapredictor for both $Q(s,a)$ and $Q(s',a')$.

Ablating Normalizations. Table 3 suggests that HANQ's particular normalizations are important for its high scores. Removing either the metapredictor's ℓ_2-normalization (“**w/o ℓ_2N**”) or the main network's LayerNorm (“**w/o LN**”) scores less than half as high. Table 4 similarly suggests that changing the types of either of those normalizations might give worse algorithms. Here, we avoid hypergradient tuning for simplicity. We observed similar results in further experiments comparing these configurations (for example, when using hypergradient tuning; not shown).

Table 3: Removing normalizations lowers scores.

	Pendulum	**(95% CI)**	**Acrobot**	**(95% CI)**
HANQ	100.1	(98.0, 102.2)	90.1	(88.0, 92.0)
w/o ℓ_2N	40.8	(27.7, 52.9)	87.9	(81.3, 92.5)
w/o LN	34.9	(32.1, 37.7)	25.0	(15.4, 34.4)

Table 4: Ablating more normalizations, without hypergradient tuning. In the column names, the first item is the normalization type in the metapredictor, and the second item the type in the main network. For example, the first column (“**ℓ_2N LN**”) is HANQ. “__” indicates no normalization.

	ℓ_2N LN	**LN ℓ_2N**	**ℓ_2N ℓ_2N**	**LN LN**	**__ ℓ_2N**	**__ LN**	**ℓ_2N __**	**LN __**	**__ __**
Pendulum	79.2	18.9	8.2	14.0	45.9	28.4	24.5	11.6	9.2
(95% CI)	(73.5, 84.1)	(15.4, 22.4)	(4.9, 11.2)	(11.2, 16.9)	(40.5, 51.1)	(24.3, 32.3)	(21.7, 27.5)	(9.1, 14.1)	(4.7, 14.6)

4.1 Atari Seaquest

To test a more complex, higher-dimensional problem, we use Atari (Bellemare et al., 2013) Seaquest. For each random seed, we collect an offline dataset of 100k state-action pairs using a uniformly random-action policy, then train for 1M gradient steps. For computational simplicity, we test only one target update rate for DQN, and only one hyperlearning rate for HANQ. Preliminary experiments (not shown) suggested $\beta = 10^{-5}$ (for DQN) and $\kappa = 0$ (for HANQ). We also compare against DQN-LN without a target net, as in PQN (Gallici et al., 2024). Since we are not using hyperlearning, the only difference between DQN-LN and HANQ here is HANQ's metapredictor and predictor. For DQN-LN, we test the normalization before or after the ReLU, and show only the best here (after the ReLU). Table 5 suggests that, compared to DQN-LN, asymmetry may be beneficial even for complex environments. Those scores also provide additional preliminary evidence for HANQ matching or exceeding DQN.

Table 5: Preliminary, in that this uses only 10 seeds. “**w/o metapred.**” is HANQ without a metapredictor — i.e., HANQ with a standard predictor like SSAQ has. No hypergradient learning.

	Seaquest	**(95% CI)**
DQN-LN	76.1	(74.7, 77.8)
HANQ	91.7	(83.0, 100.8)
w/o metapred.	87.4	(80.8, 93.3)
DQN	90.4	(86.2, 93.9)

5 Conclusions

Our results add to the evidence that more asymmetry might be key for faster and stabler optimization for deep Q-learning. One future direction is to more closely compare such asymmetries with adaptive discount factors. Another direction may be: instead of treating ω_{scale} as a parameter, treat it as a hyperparameter tuned via hyperoptimization. Treating it as a parameter risks modifying the original MSBE loss too much, effectively changing the discount factor. Rather, by hyperoptimizing ω_{scale} for the original MSBE loss, we can preserve alignment with the original MSBE loss. Granted, to our knowledge, it is not yet clear when precisely either optimization approach would be theoretically sound, especially for the state-dependent ω_{scale} case (like our metapredictor).

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An Optimistic Perspective on Offline Reinforcement Learning. *arXiv*, July 2019. DOI: 10.48550/arXiv.1907.04543.

Philip Amortila, Dylan J. Foster, Nan Jiang, Akshay Krishnamurthy, and Zakaria Mhammedi. Reinforcement Learning under Latent Dynamics: Toward Statistical and Algorithmic Modularity. *arXiv*, October 2024. DOI: 10.48550/arXiv.2410.17904.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv*, July 2016. DOI: 10.48550/arXiv.1607.06450.

Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, 1995. URL https://api.semanticscholar.org/CorpusID:621595.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient Online Reinforcement Learning with Offline Data. *arXiv*, February 2023. DOI: 10.48550/arXiv.2302.02948.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Johan Bjorck, Carla P. Gomes, and Kilian Q. Weinberger. Is High Variance Unavoidable in RL? A Case Study in Continuous Control. *arXiv*, October 2021. DOI: 10.48550/arXiv.2110.11222.

David Brandfonbrener and Joan Bruna. Geometric Insights into the Convergence of Nonlinear TD Learning. *arXiv*, May 2019. DOI: 10.48550/arXiv.1905.12185.

David Brandfonbrener, William F. Whitney, Rajesh Ranganath, and Joan Bruna. Offline RL Without Off-Policy Evaluation. *arXiv*, June 2021. DOI: 10.48550/arXiv.2106.08909.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv*, June 2016. DOI: 10.48550/arXiv.1606.01540.

Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient Descent: The Ultimate Optimizer. *arXiv*, September 2019. DOI: 10.48550/arXiv.1909.13371.

Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. *arXiv*, November 2020. DOI: 10.48550/arXiv.2011.10566.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15750–15758, 2021.

Mohamed Elsayed, Gautham Vasan, and A. Rupam Mahmood. Streaming Deep Reinforcement Learning Finally Works. *arXiv*, October 2024. DOI: 10.48550/arXiv.2410.14606.

Benjamin Eysenbach, Vivek Myers, Ruslan Salakhutdinov, and Sergey Levine. Inference via Interpolation: Contrastive Representations Provably Enable Planning and Inference. *arXiv*, March 2024. DOI: 10.48550/arXiv.2403.04082.

Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies. *arXiv*, December 2015. DOI: 10.48550/arXiv.1512.02011.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. *arXiv preprint arXiv:2004.07219*, 2020.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying Deep Temporal Difference Learning. *arXiv*, July 2024. DOI: 10.48550/arXiv.2407.04811.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning Continuous Latent Space Models for Representation Learning. *arXiv*, June 2019. DOI: 10.48550/arXiv.1906.02736.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap Latent-Predictive Representations for Multitask Reinforcement Learning. *arXiv*, April 2020. DOI: 10.48550/arXiv.2004.14646.

Zhaohan Daniel Guo, Shantanu Thakoor, Miruna Pîslar, Bernardo Avila Pires, Florent Altché, Corentin Tallec, Alaa Saade, Daniele Calandriello, Jean-Bastien Grill, Yunhao Tang, Michal Valko, Rémi Munos, Mohammad Gheshlaghi Azar, and Bilal Piot. BYOL-Explore: Exploration by Bootstrapped Prediction. *arXiv*, June 2022. DOI: 10.48550/arXiv.2206.08332.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout Q-Functions for Doubly Efficient Reinforcement Learning. *arXiv*, October 2021. DOI: 10.48550/arXiv.2110.02034.

Shengyi Huang, Rousslan Fernand JulienDossa Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João GM Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *The Journal of Machine Learning Research*, 23(1):12585–12602, 2022.

Marcel Hussing, Claas Voelcker, Igor Gilitschenski, Amir-massoud Farahmand, and Eric Eaton. Dissecting Deep RL with High Update Ratios: Combatting Value Divergence. *arXiv*, March 2024. DOI: 10.48550/arXiv.2403.05996.

Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pp. 1181–1189, 2015.

Khimya Khetarpal, Zhaohan Daniel Guo, Bernardo Avila Pires, Yunhao Tang, Clare Lyle, Mark Rowland, Nicolas Heess, Diana Borsa, Arthur Guez, and Will Dabney. A Unifying Framework for Action-Conditional Self-Predictive Reinforcement Learning. *arXiv*, June 2024. DOI: 10.48550/arXiv.2406.02035.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv*, December 2014. DOI: 10.48550/arXiv.1412.6980.

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. DR3: Value-Based Deep Reinforcement Learning Requires Explicit Regularization. *arXiv*, December 2021. DOI: 10.48550/arXiv.2112.04716.

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline Q-Learning on Diverse Multi-Task Data Both Scales And Generalizes. *arXiv*, November 2022. DOI: 10.48550/arXiv.2211.15144.

Hojoon Lee, Hanseul Cho, Hyunseung Kim, Daehoon Gwak, Joonkee Kim, Jaegul Choo, Se-Young Yun, and Chulhee Yun. PLASTIC: Improving Input and Label Plasticity for Sample Efficient Reinforcement Learning. *arXiv*, June 2023. DOI: 10.48550/arXiv.2306.10711.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R. Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. SimBa: Simplicity Bias for Scaling Up Parameters in Deep Reinforcement Learning. *arXiv*, October 2024. DOI: 10.48550/arXiv.2410.09754.

Qiyang Li, Aviral Kumar, Ilya Kostrikov, and Sergey Levine. Efficient Deep Reinforcement Learning Requires Regulating Overfitting. *arXiv*, April 2023. DOI: 10.48550/arXiv.2304.10466.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv*, Sep 2015. DOI: 10.48550/arXiv.1509.02971.

Bo Liu, Yihao Feng, Qiang Liu, and Peter Stone. Metric Residual Networks for Sample Efficient Goal-Conditioned Reinforcement Learning. *arXiv*, August 2022. DOI: 10.48550/arXiv.2208.08133.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. *arXiv*, March 2023. DOI: 10.48550/arXiv.2303.01486.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning. *arXiv*, July 2024a. DOI: 10.48550/arXiv.2407.01800.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, Hado van Hasselt, Razvan Pascanu, James Martens, and Will Dabney. Disentangling the Causes of Plasticity Loss in Neural Networks. *arXiv*, February 2024b. DOI: 10.48550/arXiv.2402.18762.

Volodymyr Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level Control through Deep Reinforcement Learning. *nature*, 518(7540):529–533, 2015.

Vivek Myers, Catherine Ji, and Benjamin Eysenbach. Horizon Generalization in Reinforcement Learning. *arXiv*, January 2025. DOI: 10.48550/arXiv.2501.02709.

Michal Nauman, Michał Bortkiewicz, Piotr Miłoś, Tomasz Trzciński, Mateusz Ostaszewski, and Marek Cygan. Overestimation, Overfitting, and Plasticity in Actor-Critic: the Bitter Lesson of Reinforcement Learning. *arXiv*, March 2024. DOI: 10.48550/arXiv.2403.00514.

Yann Ollivier. Approximate Temporal Difference Learning is a Gradient Descent for Reversible Policies. *arXiv*, May 2018. DOI: 10.48550/arXiv.1805.00869.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical Design in Reinforcement Learning. *arXiv*, April 2023. DOI: 10.48550/arXiv.2304.01315.

Silviu Pitis, Harris Chan, Kiarash Jamali, and Jimmy Ba. An Inductive Bias for Distances: Neural Nets that Respect the Triangle Inequality. *arXiv*, February 2020. DOI: 10.48550/arXiv.2002.05825.

Sarah Rathnam, Sonali Parbhoo, Siddharth Swaroop, Weiwei Pan, Susan A Murphy, and Finale Doshi-Velez. Rethinking discount regularization: New interpretations, unintended consequences, and solutions for regularization in reinforcement learning. *Journal of Machine Learning Research*, 25(255):1–48, 2024.

Pierre Harvey Richemond, Allison Tam, Yunhao Tang, Florian Strub, Bilal Piot, and Felix Hill. The edge of orthogonality: A simple view of what makes byol tick. In *International Conference on Machine Learning*, pp. 29063–29081. PMLR, 2023.

Laura Smith, Ilya Kostrikov, and Sergey Levine. A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning. *arXiv*, August 2022. DOI: 10.48550/arXiv.2208.07860.

Braham Snyder, Amy Zhang, and Yuke Zhu. Target Rate Optimization: Avoiding Iterative Error Exploitation. NeurIPS Foundation Models for Decision Making Workshop, 2023. URL https://openreview.net/forum?id=yD9JAKItJE.

Richard S. Sutton. A History of Meta-gradient: Gradient Methods for Meta-learning. *arXiv*, February 2022. DOI: 10.48550/arXiv.2202.09701.

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

Yunhao Tang, Zhaohan Daniel Guo, Pierre Harvey Richemond, Bernardo Avila Pires, Yash Chandak, Rémi Munos, Mark Rowland, Mohammad Gheshlaghi Azar, Charline Le Lan, Clare Lyle, et al. Understanding self-predictive learning for reinforcement learning. In *International Conference on Machine Learning*, pp. 33632–33656. PMLR, 2023.

John N Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1):59–94, 1996.

John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5), 1997.

Gautham Vasan, Mohamed Elsayed, Alireza Azimi, Jiamin He, Fahim Shariar, Colin Bellinger, Martha White, and A. Rupam Mahmood. Deep Policy Gradient Methods Without Batch Updates, Target Networks, or Replay Buffers. *arXiv*, November 2024. DOI: 10.48550/arXiv.2411.15370.

Che Wang, Yanqiu Wu, Quan Vuong, and Keith Ross. Striving for Simplicity and Performance in Off-Policy DRL: Output Normalization and Non-Uniform Sampling. *arXiv*, October 2019. DOI: 10.48550/arXiv.1910.02208.

Tongzhou Wang. *Intelligent Agents via Representation Learning*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, September 2024. URL https://www.tongzhouwang.info/phd_thesis_Wang_Tongzhou_MIT.pdf. Submitted on August 9, 2024.

Tongzhou Wang, Antonio Torralba, Phillip Isola, and Amy Zhang. Optimal Goal-Reaching Reinforcement Learning via Quasimetric Learning. *arXiv*, April 2023. DOI: 10.48550/arXiv.2304.01203.

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

Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989. URL http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

Zixin Wen and Yuanzhi Li. The mechanism of prediction head in non-contrastive self-supervised learning. *Advances in Neural Information Processing Systems*, 35:24794–24809, 2022.

Chenjun Xiao, Bo Dai, Jincheng Mei, Oscar A. Ramirez, Ramki Gummadi, Chris Harris, and Dale Schuurmans. Understanding and Leveraging Overparameterization in Recursive Value Estimation. *OpenReview*, March 2022. URL https://openreview.net/forum?id=shbAgEsk3qM.

Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-Gradient Reinforcement Learning. *arXiv*, May 2018. DOI: 10.48550/arXiv.1805.09801.

Chaoning Zhang, Kang Zhang, Chang-Dong Yoo, and In-So Kweon. How does simsiam avoid collapse without negative samples? towards a unified understanding of progress in ssl. In *The International Conference on Learning Representations, ICLR 2022*. The International Conference on Learning Representations (ICLR), 2022.

Chongyi Zheng, Benjamin Eysenbach, Homer Walke, Patrick Yin, Kuan Fang, Ruslan Salakhutdinov, and Sergey Levine. Stabilizing Contrastive RL: Techniques for Robotic Goal Reaching from Offline Data. *arXiv*, June 2023. DOI: 10.48550/arXiv.2306.03346.

Supplementary Materials

The following content was not necessarily subject to peer review.

6 Plotting the Predictor Parameter

Recall that HSSAQ is SSAQ (Single-Scaler Asymmetric-Q) with hypergradient learning (details in Section 3.3). Fig. 7 shows how HSSAQ's learned predictor parameter, ω_{scale}, changes over the course of training. At initialization, $\omega_{\text{scale}} = 1$ (which is hard to see due to plotting artifacts). It immediately rises high, past 1.3, then slowly drops back down again, close to 1.2. Interpreting ω_{scale} as loosely similar to the inverse of the discount factor γ (Section 8), HSSAQ's learning here resembles the finding that increasing the discount factor from a smaller value to a larger value over the course of training can improve scores (François-Lavet et al., 2015). In this analogy, HSSAQ automatically dropped to the smaller discount factor on its own, then automatically increased the discount factor again over the course of further training.

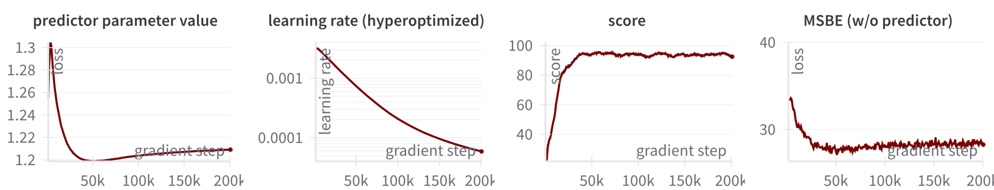


Figure 7: The same as Fig. 5, but with a plot of the predictor parameter value ω_{scale} (leftmost plot) as well. The predictor parameter immediately increases at the start of training, then slowly returns to a smaller value. Original caption, from Fig. 5: HSSAQ with its best initial learning rate, $\boxed{10^{-2.5}}$ (and hyperlearning rate $\kappa = 10^{-4}$)..

7 Algorithm Pseudocode

Algorithm 1 SSAQ. (Changes from DQN (Mnih et al., 2015) with a soft target update (Lillicrap et al., 2015) are in cyan blue.)

Parameters: target update rate τ, learning rate α.
Input: Offline dataset $\mathcal{D}$ of tuples $\{(s, a, r, s')\}$.
Randomly initialize θ for the main network Q_θ. ▷ Default SSAQ uses no target net
Initialize $\omega_{\text{scale}} = 1$.
for each algorithm step **do**
  Sample a batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.
  Compute the main Q-values and scale the result $Q_{\omega,\theta}(s, a) := \omega_{\text{scale}} Q_\theta(s, a)$.
  Compute the targets $y := r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s', a')$. ▷ Default SSAQ uses no target net
  Take a gradient descent step

$$\theta \leftarrow \theta - \alpha \nabla_{\omega,\theta} \left(\frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} \left(Q_{\omega,\theta}(s, a) - \text{sg}[y] \right)^2 \right)$$

  where sg$[y]$ is a stop-gradient.
end for

Algorithm 2 HANQ. (Changes from DQN (Mnih et al., 2015) with a soft target update (Lillicrap et al., 2015) are in cyan blue.)

Parameters: target update rate τ, learning rate α, hyperlearning rate κ.
Input: Offline dataset $\mathcal{D}$ of tuples $\{(s, a, r, s')\}$.
Randomly initialize θ for the main network Q_θ and the metapredictor ω_θ. ▷ Branching (Fig. 6a)
for each algorithm step **do**
  Sample a batch $\mathcal{B} = \{(s, a, r, s')\}$ from $\mathcal{D}$.
  Compute the main Q-values $Q_\theta(s, a)$ and predictor parameters $\omega_\theta(s, a) = \{\omega_{\text{scale}}, \omega_{\text{bias}}\}$.
  Compute the forced positive scaler $\omega^+_{\text{scale}} := \exp(\omega_{\text{scale}})$)
  Scale and bias to get $Q_{\omega,\theta}(s, a) := \omega^+_{\text{scale}} Q_\theta(s, a) + \omega_{\text{bias}}$.
  Compute the targets $y := r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s', a')$. ▷ Discard or do not compute $\omega_\theta(s', a')$
  Take a gradient descent step

$$\theta \leftarrow \theta - \alpha \nabla_{\omega,\theta} \left(\frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} \left(Q_{\omega,\theta}(s, a) - \text{sg}[y]\right)^2 \right)$$

  where $\text{sg}[y]$ is a stop-gradient.
  Compute the residual hypergradient and update the learning rate

$$\alpha \leftarrow \alpha - \kappa \frac{\partial}{\partial \alpha} \left(\frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} \left(Q_{\omega,\theta}(s, a) - (r + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s', a')) \right)^2 \right)$$

end for

8 Connection Between Scaling and Discount Factor

We argue that the adding the scaling factor in SSAQ is effectively changing the discount factor. To see this, note that with the scaling factor ω, the loss minimization procedure essentially tries to find a fixed-point solution for $\omega Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[\max_{a'} Q(s', a')]$, which can be found to be $Q(s, a) = \mathbb{E}\left[\omega^{-1} R(s, a) + \omega^{-2} \gamma R(s_1, a_1) + \cdots\right] = \omega^{-1} \mathbb{E}[\sum_{t=0}^{\infty} (\gamma/\omega)^t R(s_t, a_t)]$ provided its existence, where $(s_0 = s, a_0 = a, s_1, a_1, \cdots)$ is a trajectory that follows the policy $\pi(s) = \arg\max_a Q(s, a)$. Clearly, the effective discount factor is γ/ω.

This perhaps suggests additional connections with meta-gradient RL algorithms that learn discount factors (Xu et al., 2018).

9 Supplementary Experiments

9.1 Number of Parameters

Even three-layer neural nets for DQN with normalization do not let it outscore HANQ in our experiments on Pendulum (Table 6). This provides further evidence, even beyond the "**symmetrized**" results in Table 2, that HANQ's high score is not due to the small number of additional parameters in HANQ's metapredictor. However, ideally we would test additional, symmetric DQN architectures with more parameters, especially wider architectures. In any case, note that three-layer DQN-LN in this setting has *over 10 times as many parameters* as two-layer HANQ, because the input and output dimensions for Pendulum are small.

9.2 Hypergradients

Table 6: Even with a three-layer network (***10× as many parameters*** as HANQ) and normalization, DQN does not exceed HANQ's score of 100.1 on Pendulum (Table 1; though their CIs do overlap).

	No target net ($\beta = 10^0$)			Best $\beta \in \{10^0, 10^{-1}, \ldots, 10^{-4}\}$		
	DQN-ℓ_2N	**DQN-LN**	**DQN**	**DQN-ℓ_2N**	**DQN-LN**	**DQN**
Pendulum	57.3	59.2	17.3	96.6	97.7	84.7
(95% CI)	(45.4, 68.5)	(48.1, 70.3)	(5.4, 29.8)	(93.5, 99.1)	(95.3, 99.8)	(77.4, 90.5)

Semi-Gradient vs Full-Gradient. Section 3.3 describes two possible objectives for hypergradient optimization of TD learning algorithms: the semi-gradient $\mathcal{L}_{\text{SG}}$ and the full-gradient $\mathcal{L}_{\text{RG}}$. See Section 12 for the derivation. Table 7 compares their scores.

Hypergradient DQN. Table 8 suggests that adding hypergradient learning (optimizing the learning rate during training, as we do with HANQ) does not enable DQN to match HANQ's scores on Pendulum. This aligns with, e.g., Fig. 3a, Fig. 3b, and Fig. 5 to suggest that architectural asymmetry as in SSAQ and HANQ can enable hypergradient learning to be more helpful in some cases.

Table 7: The residual gradient $\mathcal{L}_{\text{RG}}$ (HANQ's default) and semi-gradient $\mathcal{L}_{\text{SG}}$ hyperobjectives work equally well on these two problems.

	Pendulum	**(95% CI)**	**Acrobot**	**(95% CI)**
$\mathcal{L}_{\text{RG}}$	100.1	(98.0, 102.2)	90.1	(88.0, 92.0)
$\mathcal{L}_{\text{SG}}$	98.7	(95.9, 101.2)	87.8	(85.8, 89.6)

Table 8: Hypergradient learning might not enable DQN to match HANQ's scores on Pendulum. We use $\mathcal{L}_{\text{RG}}$ here, like HANQ's default. Recall, κ is the hypergradient learning rate. This table tunes $\beta \in \{10^0, 10^{-1}, \ldots, 10^{-3}\}$.

	$\kappa = 10^{-1}$	$\kappa = 10^{-2}$	$\kappa = 10^{-3}$	$\kappa = 10^{-4}$	$\kappa = 10^{-5}$	$\kappa = 0$
Pendulum	53.3	63.1	65.5	69.7	70.9	72.4
(95% CI)	(39.8, 66.6)	(49.6, 75.9)	(51.2, 80.7)	(54.8, 83.2)	(56.1, 84.4)	(61.0, 82.5)

9.3 Controlling For a Tuned Discount Factor

Given the connection between a predictor and the discount factor (Section 8), we test manually tuning DQN's discount factor, with and without normalizations, and with and without a target net (in combination with 7 learning rates for every configuration, as usual). Despite tuning over 5 new discount factors for DQN, for a total of three tuned hyperparameters compared to HANQ's two tuned hyperparameters, no mean score reaches HANQ's CIs. However, some configurations give CIs that overlap with HANQ's CI on this problem. We show all these scores and CIs in Table 9.

10 Details on Normalizations

ℓ_2-normalization is empirically important in many SSL algorithms (Grill et al., 2020; Chen & He, 2021) and RL algorithms (Wang et al., 2019; Bjorck et al., 2021; Kumar et al., 2022; Hussing et al., 2024; Vasan et al., 2024). ℓ_2-normalization operates independently on each data point x, over the feature axis, projecting the features to the unit hypersphere (of the same dimension as the input): $f(x) := x/||x||_2$. Similar to ℓ_2-normalization, LayerNorm (Ba et al., 2016) has extensive support in RL (Hiraoka et al., 2021; Smith et al., 2022; Ball et al., 2023; Lee et al., 2023; Lyle et al., 2023; Lee et al., 2024; Lyle et al., 2024a; Nauman et al., 2024; Vasan et al., 2024; Elsayed et al., 2024; Zheng et al., 2023; Li et al., 2023; Lyle et al., 2024b; Gallici et al., 2024). Also like ℓ_2-normalization, LayerNorm operates independently on each data point x over the feature axis, without changing the

Table 9: Manually tuning the discount factor does not enable DQN to beat HANQ's scores on Pendulum. Recall, γ is the discount factor (which for HANQ we always leave at its upstream default of 0.99), and κ is the hyperlearning rate (HANQ only). "**DQN-LN**" and "**DQN-ℓ_2N**" are again DQN with LayerNorm or ℓ_2-normalization. For convenience, we include HANQ's scores from Table 1 here as well. We highlight every cell in this table whose CI overlaps the CI of the top mean score on this problem (Pendulum).

	$\beta = 10^0$			$\beta = 10^{-3}$		
	DQN	**DQN-LN**	**DQN-ℓ_2N**	**DQN**	**DQN-LN**	**DQN-ℓ_2N**
$\gamma = \mathbf{0.95}$	70.0	76.2	65.9	84.4	92.0	95.8
(95% CI)	(55.8, 83.7)	(63.8, 87.3)	(52.4, 77.4)	(72.3, 94.2)	(85.7, 96.4)	(91.0, 99.7)
$\gamma = \mathbf{0.9}$	91.6	89.6	87.4	92.3	93.6	96.1
(95% CI)	(82.2, 99.5)	(78.9, 98.3)	(75.3, 96.4)	(84.2, 98.4)	(84.6, 99.7)	(89.8, 100.5)
$\gamma = \mathbf{0.85}$	95.3	90.2	91.9	96.5	95.1	95.2
(95% CI)	(90.4, 99.0)	(80.1, 98.2)	(86.3, 96.7)	(94.6, 97.9)	(89.9, 98.5)	(92.1, 97.8)
$\gamma = \mathbf{0.8}$	83.4	88.0	87.7	81.5	85.1	87.0
(95% CI)	(79.7, 86.5)	(84.5, 91.5)	(84.4, 90.6)	(78.9, 83.8)	(80.7, 88.7)	(83.2, 90.1)
$\gamma = \mathbf{0.75}$	54.8	59.5	57.8	47.7	53.2	58.8
(95% CI)	(46.6, 61.1)	(55.3, 62.8)	(53.3, 62.3)	(45.2, 50.1)	(51.2, 55.0)	(55.6, 62.4)

dimensionality. In particular, for each data point x, LayerNorm maps it to $f(x) := \frac{x-\bar{x}}{\sqrt{s^2+\epsilon}} \odot \gamma_{\text{LN}} + \beta_{\text{LN}}$, where $\bar{x}$ and s^2 are the mean and variance across the features, respectively, ϵ avoids division by zero, and γ_{LN} and β_{LN} are per-feature learnable parameters. Similar to those prior works, we add the normalizations before the final weights of both the $Q_\theta(s, a)$ and $Q_\theta(s', a')$ paths.

11 Experiment Details

For classic control, every algorithm is tuned over learning rates in $\{10^{-1}, 10^{-1.5}, \ldots, 10^{-4}\}$, extended in either direction if the algorithm's best learning rate is found to be the minimum or maximum of that set. Algorithms that use a target net (DQN and QS-DQN) are combinatorially tuned over target update rates in $\{10^0, 10^{-1}, \ldots, 10^{-5}\}$ unless stated otherwise. HANQ's hypergradient step size is tuned in that latter set as well. For PQN, we test with LayerNorm both before or after the ReLU (rectified linear unit), and ℓ_2-regularizations in $\{10^{-1}, 10^{-2}, \ldots, 10^{-5}, 10^{-6}, 10^{-8}, 10^{-10}\}$.

Every hyperparameter combination was compared using 30 random seeds unless otherwise stated, and we show only the best combination per algorithm–problem combination.

For Atari, we use 10 random seeds for every hyperparameter combination, and tune learning rates in $\{10^{-2.5}, 10^{-3}, 10^{-3.5}, \ldots, 10^{-8}\}$. For computational simplicity, we use only a target update rate $\beta = 10^{-5}$ (for DQN) and a hyperlearning rate of $\kappa = 0$ (for HANQ), which were suggested by preliminary experiments (not shown).

We use Adam (Kingma & Ba, 2014) for all algorithms. When using hyperoptimization, we hyperoptimize Adam via Adam (Chandra et al., 2019).

For classic control, we use two-layer neural networks (two sets of weights, one hidden layer) unless otherwise stated. (HANQ and SSAQ use additional parameters for one term of the loss function, i.e. for $Q(s, a)$. However, to reiterate, our results in e.g. Section 9.1 and Section 4 suggest that adding more parameters in more standard ways does not increase speed and stability as much.) We also use, unless otherwise stated: a hidden width of 128; a batch size of 128; 200k gradient steps for training, unless otherwise stated; and a discount factor of $\gamma = 0.99$ (we ignore the theoretical issues in combining discount factors with function approximation (Sutton & Barto, 2018)).

For Atari, we use: the architecture from Huang et al. (2022), based on Mnih et al. (2015) (DQN-LN uses LayerNorm before the final weights, as in e.g. PQN (Gallici et al., 2024), and HANQ's metapredictor again takes as input the features before they enter that LayerNorm, as in the classic control version of HANQ); a batch size of 128; 1M gradient steps for training; and a discount factor of $\gamma = 0.99$ unless otherwise stated.

Normalizations placement. In preliminary experiments (not shown), placing normalizations before vs. after the ReLU typically made little difference, so unless otherwise noted we use only the latter (after the ReLU) for all algorithms.

Environments. The environments we use are: `CartPole-v1` and `Acrobot-v1` from classic control (Brockman et al., 2016); a discrete-action version of `Pendulum-v1` (Brockman et al., 2016; Xiao et al., 2022; Snyder et al., 2023) with action space $\{-2, 0, -2\}$; `SeaquestNoFrameskip-v4` with the manual 4-frame stacking, resizing, and grayscale transformations of Huang et al. (2022), and with `repeat_action_probability=0.0` for determinism.

Offline dataset construction. We construct our offline datasets like prior work (Xiao et al., 2022; Snyder et al., 2023). For Pendulum, we collect an offline dataset of 1000 samples of state-action pairs, using uniformly sampled initial states, taking uniformly random actions. For CartPole and Acrobot, we instead use the standard initial states, and collect 10000 samples. For Atari, we again use the standard initial states, and collect 100k samples.

11.1 Scores to Returns Conversion

For readability, we give normalized "scores" throughout our paper instead of episodic return. Similar to Fu et al. (2020), we define

$$\texttt{score} := 100 \left(\frac{\texttt{return} - \texttt{low\_return}}{\texttt{high\_return} - \texttt{low\_return}} \right)$$

where `low_return` is unrigorously defined as the typical episodic return of a policy that takes random actions, and `high_return` is unrigorously defined as the typical episodic return of an expert policy. For Seaquest, we picked 300 as the `high_return`, which was roughly the peak return of our earliest experiments (note that our training dataset is 100k random actions, so this return is far lower than, e.g., human expert scores). Table 10 shows those return values.

Table 10: Episodic returns for our definition of "score."

	Pendulum	**Acrobot**	**CartPole**	**Seaquest**
`low_return`	-1500	-500	0	0
`high_return`	-200	-100	500	300

12 Derivation of Hypergradients

We follow the derivation in Chandra et al. (2019). At the beginning of step i, let θ_i be the weights of the neural network (for simplicity, here we use θ_i to represent *all* parameters in the network, which include the θ and ω described in previous sections), α_i be the main learning rate, $\mathcal{L}(\theta_i)$ be the main loss function, and $\widetilde{\mathcal{L}}(\theta_i)$ be the hyperoptimization loss function (which could be the same as or different from the main loss function). Let κ be the hypergradient learning rate (hyperlearning rate). To hyperoptimize SGD, hypergradient descent updates the main learning rate and the weights in the following manner:

$$\alpha_{i+1} = \alpha_i - \kappa \frac{\partial \widetilde{\mathcal{L}}(\theta_i)}{\partial \alpha_i}, \tag{5}$$

$$\theta_{i+1} = \theta_i - \alpha_{i+1} \frac{\partial \mathcal{L}(\theta_i)}{\partial \theta_i}. \tag{6}$$

The hypergradient in Eq. (5) can be calculated as the following:

$$\frac{\partial \widetilde{\mathcal{L}}(\theta_i)}{\partial \alpha_i} = \frac{\partial \widetilde{\mathcal{L}}(\theta_i)}{\partial \theta_i} \cdot \frac{\partial \theta_i}{\partial \alpha_i} = \frac{\partial \widetilde{\mathcal{L}}(\theta_i)}{\partial \theta_i} \cdot \left(- \frac{\partial \mathcal{L}(\theta_{i-1})}{\partial \theta_{i-1}} \right) \tag{7}$$

where the second equality is due to Eq. (6).

When $\widetilde{\mathcal{L}} = \mathcal{L}$, the hypergradient is the negative dot product of the two most recent gradients. Intuitively, if the two most recent gradients have a large dot product, it is sensible to increase the learning rate.

For the main loss $\mathcal{L}$, we use the semi-gradient loss as in standard DQN: $\mathcal{L}_{\text{SG}}(\theta) \triangleq (Q_\theta(s,a) - r - \gamma \mathsf{sg}[\max_{a'} Q_\theta(s', a')])^2$. For hyperoptimization loss $\widetilde{\mathcal{L}}$, we tested two options: (i) the semi-gradient loss $\mathcal{L}_{\text{SG}}$ same as the main optimizer, and (ii) the full-gradient loss $\mathcal{L}_{\text{RG}}$ used in residual gradient (Baird, 1995), defined as $\mathcal{L}_{\text{RG}}(\theta) \triangleq (Q_\theta(s,a) - r - \gamma \max_{a'} Q_\theta(s', a'))^2$.

Denote $\text{SG}_i = \frac{\partial \mathcal{L}_{\text{SG}}(\theta_i)}{\partial \theta_i}$ and $\text{RG}_i = \frac{\partial \mathcal{L}_{\text{RG}}(\theta_i)}{\partial \theta_i}$. By Eq. (7), option (i) $\mathcal{L} = \mathcal{L}_{\text{SG}}, \widetilde{\mathcal{L}} = \mathcal{L}_{\text{SG}}$ leads to the hypergradient

$$\frac{\partial \widetilde{\mathcal{L}}(\theta_i)}{\partial \alpha_i} = \text{SG}_i \cdot -\text{SG}_{i-1}, \tag{8}$$

and option (ii) $\mathcal{L} = \mathcal{L}_{\text{SG}}$, $\widetilde{\mathcal{L}} = \mathcal{L}_{\text{RG}}$ leads to

$$\frac{\partial \widetilde{\mathcal{L}}(\theta_i)}{\partial \alpha_i} = \text{RG}_i \cdot -\text{SG}_{i-1}. \tag{9}$$