Faster Deep Reinforcement Learning with Slower Online Network

Kavosh Asadi
Amazon Web Services

Rasool Fakoor
Amazon Web Services

Omer Gottesman
Brown University

Taesup Kim
Seoul National University

Michael L. Littman
Brown University

Alexander J. Smola
Amazon Web Services

Abstract

Deep reinforcement learning algorithms often use two networks for value function optimization: an online network, and a target network that tracks the online network with some delay. Using two separate networks enables the agent to hedge against issues that arise when performing bootstrapping. In this paper we endow two popular deep reinforcement learning algorithms, namely DQN and Rainbow, with updates that incentivize the online network to remain in the proximity of the target network. This improves the robustness of deep reinforcement learning in presence of noisy updates. The resultant agents, called DQN Pro and Rainbow Pro, exhibit significant performance improvements over their original counterparts on the Atari benchmark demonstrating the effectiveness of this simple idea in deep reinforcement learning. The code for our paper is available here: Github.com/amazon-research/fast-rl-with-slow-updates.

1 Introduction

An important competency of reinforcement-learning (RL) agents is learning in environments with large state spaces like those found in robotics (Kober et al., 2013), dialog systems (Williams et al., 2017), and games (Tesauro, 1994; Silver et al., 2017). Recent breakthroughs in deep RL have demonstrated that simple approaches such as Q-learning (Watkins & Dayan, 1992) can surpass human-level performance in challenging environments when equipped with deep neural networks for function approximation (Mnih et al., 2015).

Two components of a gradient-based deep RL agent are its objective function and optimization procedure. The optimization procedure takes estimates of the gradient of the objective with respect to network parameters and updates the parameters accordingly. In DQN (Mnih et al., 2015), for example, the objective function is the empirical expectation of the temporal difference (TD) error (Sutton, 1988) on a buffered set of environmental interactions (Lin, 1992), and variants of stochastic gradient descent are employed to best minimize this objective function.

A fundamental difficulty in this context stems from the use of bootstrapping. Here, bootstrapping refers to the dependence of the target of updates on the parameters of the neural network, which is itself continuously updated during training. Employing bootstrapping in RL stands in contrast to supervised-learning techniques and Monte-Carlo RL (Sutton & Barto, 2018), where the target of our gradient updates does not depend on the parameters of the neural network. (Mnih et al., 2015) proposed a simple approach to hedging against issues that arise when using bootstrapping, namely to use a target network in value-function optimization. The target network is updated periodically, and tracks the online network with some delay. While this modification

We now define the Proximal Bellman Operator \( \Pi \) whose fixed point is \( T \), which we can write compactly as:

\[
\Pi \circ T \pi = T^\pi \pi = \arg \min_{\pi'} \| \pi' - T^\pi \pi \|_2^2 + \frac{1}{c} D_f(\pi', \pi) ,
\]

where \( c \in (0, \infty) \). Intuitively, this operator encourages the next iterate to be in the proximity of the previous iterate, while also having a small difference relative to the point recommended by the

2 Background and Notation

RL is the study of the interaction between an environment and an agent that learns to maximize reward through experience. The Markov Decision Process (Puterman, 1994), or MDP, is used to mathematically define the RL problem. An MDP is specified by the tuple \( (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma) \), where \( \mathcal{S} \) is the set of states and \( \mathcal{A} \) is the set of actions. The functions \( \mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \) and \( \mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1] \) denote the reward and transition dynamics of the MDP. Finally, a discounting factor \( \gamma \) is used to formalize the intuition that short-term rewards are more valuable than those received later.

The goal in the RL problem is to learn a policy, a mapping from states to a probability distribution over actions, \( \pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A}) \), that obtains high sums of future discounted rewards. An important concept in RL is the state value function. Formally, it denotes the expected discounted sum of future rewards when committing to a policy \( \pi \) in a state \( s \):

\[
\mathcal{V}^\pi(s) := \mathbb{E}
\left[ \sum_{t=0}^{\infty} \gamma^t R_t \big| S_0 = s, \pi \right].
\]

We define the Bellman Optimality Operator as follows:

\[
\mathcal{T}^\pi \mathcal{V} = \mathcal{V}^\pi = \sum_{a \in \mathcal{A}} \pi(a | s) \mathcal{R}(s, a) + \sum_{s' \in \mathcal{S}} \gamma \mathcal{P}(s, a, s') \mathcal{V}(s'),
\]

which we can write compactly as:\n\[
\Pi \circ T \mathcal{V} := \mathcal{R} + \gamma \mathcal{P} \mathcal{V} ,
\]

and the optimal value of a state, namely: \( \mathcal{V}^* := \max_{\pi} \mathcal{V}^\pi \) as a policy that achieves \( \mathcal{V}^* \) for all states. We define the Bellman Optimality Operator \( \mathcal{T}^* \) as follows:

\[
\mathcal{T}^* \mathcal{V} = \mathcal{V}^* = \sum_{a \in \mathcal{A}} \mathcal{R}(s, a) + \sum_{s' \in \mathcal{S}} \gamma \mathcal{P}(s, a, s') \mathcal{V}(s'),
\]

whose fixed point is \( \mathcal{V}^* \). These operators are at the heart of many planning and RL algorithms including Value Iteration (Bellman, 1957) and Policy Iteration (Howard, 1960).

3 Proximal Bellman Operator

In this section, we introduce a new class of Bellman operators that ensure that the next iterate in planning and RL remain in the vicinity of the previous iterate. To this end, we define the Bregman Divergence generated by a convex function \( f \):

\[
D_f(\pi', \pi) := f(\pi') - f(\pi) - \langle \nabla f(\pi), \pi' - \pi \rangle.
\]

Examples include the \( l_p \) norm generated by \( f(\pi) = \frac{1}{2} \| \pi \|_p^2 \) and the Mahalanobis Distance generated by \( f(\pi) = \frac{1}{2} \langle \pi, Q \pi \rangle \) for a positive semi-definite matrix \( Q \).

We now define the Proximal Bellman Operator \( (\mathcal{T}_{c,f})^n \): 

\[
(\mathcal{T}_{c,f})^n \mathcal{V} := \arg \min_{\mathcal{V}'} \| \mathcal{V}' - (\mathcal{T}^\pi)^n \mathcal{V} \|_2^2 + \frac{1}{c} D_f(\mathcal{V}', \mathcal{V}) ,
\]

where \( c \in (0, \infty) \). Intuitively, this operator encourages the next iterate to be in the proximity of the previous iterate, while also having a small difference relative to the point recommended by the
original Bellman Operator. The parameter $c$ could, therefore, be thought of as a knob that controls the degree of gravitation towards the previous iterate.

Our goal is to understand the behavior of Proximal Bellman Operator when used in conjunction with the Modified Policy Iteration (MPI) algorithm (Puterman 1994, Scherrer et al. 2015). Define $G_v$ as the greedy policy with respect to $v$. At a certain iteration $k$, Proximal Modified Policy Iteration (PMPI) proceeds as follows:

$$\pi_k \leftarrow G_{v_{k-1}}, \quad v_k \leftarrow (T^\pi_{c,f})^n v_{k-1}. \quad (2)$$

The pair of updates above generalize existing algorithms. Notably, with $c \to \infty$ and general $n$ we get MPI, with $c \to \infty$ and $n=1$ the algorithm reduces to Value Iteration, and with $c \to \infty$ and $n=\infty$ we have a reduction to Policy Iteration. For finite $c$, the two extremes of $n$, namely $n=1$ and $n=\infty$, could be thought of as the proximal versions of Value Iteration and Policy Iteration, respectively.

To analyze this approach, it is first natural to ask if each iteration of PMPI could be thought of as a contraction so we can get sound and convergent behavior in planning and learning. For $n > 1$, Scherrer et al. (2015) constructed a contrived MDP demonstrating that one iteration of MPI can unfortunately expand. As PMPI is just a generalization of MPI, the same example from Scherrer et al. (2015) shows that PMPI can expand. In the case of $n=1$, we can rewrite the pair of equations (2) and (3) in a single update as follows: $v_k \leftarrow T^\pi_{c,f} v_{k-1}$. When $c \to \infty$, standard proofs can be employed to show that the operator is a contraction (Littman & Szepesvári 1996). We now show that $T^\pi_{c,f}$ is a contraction for finite values of $c$. See our appendix for proofs.

**Theorem 1.** The Proximal Bellman Optimality Operator $T^\pi_{c,f}$ is a contraction with fixed point $v^\star$.

Therefore, we get convergent behavior when using $T^\pi_{c,f}$ in planning and RL. The addition of the proximal term is fortunately not changing the fixed point, thus not negatively affecting the final solution. This could be thought of as a form of regularization that vanishes in the limit; the algorithm converges to $v^\star$ even without decaying $1/c$.

Going back to the general $n \geq 1$ case, we cannot show contraction, but following previous work (Bertsekas & Tsitsiklis 1996, Scherrer et al. 2015), we study error propagation in PMPI in presence of additive noise where we get a noisy sample of the original Bellman Operator $(T^\pi_k)^n v_{k-1} + \epsilon_k$. The noise can stem from a variety of reasons, such as approximation or estimation error. For simplicity, we restrict the analysis to $D_f(v', v) = \|v' - v\|_2^2$, so we rewrite update (3) as:

$$v_k \leftarrow \arg\min_{v'} \|v' - ((T^\pi_k)^n v_{k-1} + \epsilon_k)\|_2^2 + \frac{1}{c} \|v' - v_{k-1}\|_2^2$$

which can further be simplified to:

$$v_k \leftarrow \frac{(1 - \beta)(T^\pi_k)^n v_{k-1} + \beta v_{k-1} + (1 - \beta)\epsilon_k,}{:= (T^\beta_{c,f})^n v_{k-1}}$$

where $\beta = \frac{1}{1+c}$. This operator is a generalization of the operator proposed by Smirnova & Dohmatob (2020) who focused on the case of $n=1$. To build some intuition, notice that the update is multiplying error $\epsilon_k$ by a term that is smaller than one, thus better hedging against large noise. While the update may slow progress when there is no noise, it is entirely conceivable that for large enough values of $\epsilon_k$, it is better to use non-zero $\beta$ values. In the following theorem we formalize this intuition. Our result leans on the theory provided by Scherrer et al. (2015) and could be thought of as a generalization of their theorem for non-zero $\beta$ values.

**Theorem 2.** Consider the PMPI algorithm specified by:

$$\pi_k \leftarrow G_{v_{k-1}}, \quad v_k \leftarrow (T^\pi_{c,f})^n v_{k-1} + (1 - \beta)\epsilon_k. \quad (4)$$

Define the Bellman residual $b_k := v_k - \pi^{x_{k+1}} v_k$, and error terms $x_k := (I - \gamma P^\pi_k)\epsilon_k$ and $y_k := \gamma P^\pi \epsilon_k$. After $k$ steps:

$$v^\star - v^{\pi_k} = \frac{v^\star - (T^\beta_{c,f})^n v_k + (T^\beta_{c,f})^n v_k - v_{\pi_k}}{d_k}$$
We now endow DQN-style algorithms with proximal updates. Let
\[
\langle s_k \rangle \leq ((1 - \beta)(\gamma P^{\sigma_k})^n + \beta I)(I - \gamma P^{\sigma_k})^{-1}b_{k-1}
\]
\[
s_k \leq ((1 - \beta)(\gamma P^{\sigma_k})^n + \beta I)(I - \gamma P^{\sigma_k})^{-1}b_{k-1}
\]
\[
b_k \leq ((1 - \beta)(\gamma P^{\sigma_k})^n + \beta I)b_{k-1} + (1 - \beta)x_k + \epsilon'_k + 1
\]
The bound provides intuition as to how the proximal Bellman Operator can accelerate convergence in the presence of high noise. For simplicity, we will only analyze the effect of the \( \epsilon' \) noise term, and ignore the \( \epsilon \) term. We first look at the Bellman residual, \( b_k \). Given the Bellman residual in iteration \( k - 1 \), \( b_{k-1} \), the only influence of the noise term \( \epsilon_k \) on \( b_k \) is through the \( (1 - \beta)x_k \), term, and we see that \( b_k \) decreases linearly with larger \( \beta \).

The analysis of \( s_k \) is slightly more involved but follows similar logic. The bound for \( s_k \) can be decomposed into a term proportional to \( b_{k-1} \) and a term proportional to \( \beta b_{k-1} \), where both are multiplied with positive semi-definite matrices. Since \( b_{k-1} \) itself linearly decreases with \( \beta \), we conclude that larger \( \beta \) decreases the bound quadratically.

The effect of \( \beta \) on the bound for \( d_k \) is more complex. The terms \( \beta y_{k-1} \) and \( \sum_{j=1}^{n-1} (\gamma P^{\sigma_k})^j b_{k-1} \) introduce a linear decrease of the bound on \( d_k \) with \( \beta \), while the term \( \beta(I - \sum_{j=1}^{n-1} (\gamma P^{\sigma_k})^j)b_{k-1} \) introduces a quadratic dependence whose curvature depends on \( I - \sum_{j=1}^{n-1} (\gamma P^{\sigma_k})^j \). This complex dependence on \( \beta \) highlights the trade-off between noise reduction and magnitude of updates. To understand this trade-off better, we examine two extreme cases for the magnitude on the noise. When the noise is very large, we may set \( \beta = 1 \), equivalent to an infinitely strong proximal term. It is easy to see that for \( \beta = 1 \), the values of \( d_k \) and \( s_k \) remain unchanged, which is preferable to the increase they would suffer in the presence of very large noise. On the other extreme, when no noise is present, the \( x_k \) and \( y_k \) terms in Theorem 2 vanish, and the bounds on \( d_k \) and \( s_k \) can be minimized by setting \( \beta = 0 \), i.e. without noise the proximal term should not be used and the original Bellman update performed. Intermediate noise magnitudes thus require a value of \( \beta \) that balances the noise reduction and update size.

4 Deep Q-Network with Proximal Updates

We now endow DQN-style algorithms with proximal updates. Let \( \langle s, a, r, s' \rangle \) denote a buffered tuple of interaction. Define the following objective function:
\[
h(\theta, w) := E_{(s, a, r, s')} \left[ (r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; w))^2 \right].
\]
(6)

Our proximal update is defined as follows:
\[
w_{t+1} = \text{arg min}_w h(w_t, w) + \frac{1}{2\epsilon} \|w - w_t\|^2.
\]
(7)

This algorithm closely resembles the standard proximal-point algorithm (Rockafellar 1976; Parikh & Boyd 2014) with the important caveat that the function \( h \) is now taking two vectors as input. At each iteration, we hold the first input constant while optimizing over the second input.

In the optimization literature, the proximal-point algorithm is well-studied in contexts where an analytical solution to (7) is available. With deep learning no closed-form solution exists, so we approximately solve (7) by taking a fixed number of descent steps using stochastic gradients. Specifically, starting each iteration with \( w = w_t \), we perform multiple \( w \) updates \( w \leftarrow w - \alpha (\nabla h(w_t, w) + \frac{1}{\epsilon} (w - w_t)) \). We end the iteration by setting \( w_{t+1} \leftarrow w \). To make a connection to standard deep RL, the online weights \( w \) could be thought of as the weights we maintain in the interim to solve (7) due to lack of a closed-form solution. Also, what is commonly referred to as the target network could better be thought of as just the previous iterate in the above proximal-point algorithm.

Observe that the update can be written as: \( w \leftarrow (1 - (\alpha / \epsilon)) \cdot w + (\alpha / \epsilon) \cdot w_t - \alpha \nabla h(w_t, w) \).

Notice the intuitively appealing form: we first compute a convex combination of \( w_t \) and \( w \), based on the hyper-parameters \( \alpha \) and \( \epsilon \), then add the gradient term to arrive at the next iterate of \( w \). If \( w_t \) and \( w \) are close, the convex combination is close to \( w \) itself and so this DQN with proximal update
A Q-Pro (DQN Pro) would behave similarly to the original DQN. However, when \( w \) strays too far from \( w_t \), taking the convex combination ensures that \( w \) gravitates towards the previous iterate \( w_t \). The gradient signal from minimizing the squared TD error (6) should then be strong enough to cancel this default gravitation towards \( w_t \). The update includes standard DQN as a special case when \( \tilde{c} \rightarrow \infty \). The pseudo-code for DQN is presented in the Appendix. The difference between DQN and DQN Pro is minimal (shown in gray), and corresponds with a few lines of code in our implementation.

5 Experiments

In this section, we empirically investigate the effectiveness of proximal updates in planning and reinforcement-learning algorithms. We begin by conducting experiments with PMPI in the context of approximate planning, and then move to large-scale RL experiments in Atari.

5.1 PMPI Experiments

We now focus on understanding the empirical impact of adding the proximal term on the performance of approximate PMPI. To this end, we use the pair of update equations:

\[
\pi_k \leftarrow Gv_{k-1} ,
\]

\[
v_k \leftarrow (1 - \beta)(T^{\pi_k})^n v_{k-1} + \epsilon_k + \beta v_{k-1} .
\]

For this experiment, we chose the toy 8 \times 8 Frozen Lake environment from Open AI Gym \cite{Brockman2016}, where the transition and reward model of the environment is available to the planner. Using a small environment allows us to understand the impact of the proximal term in the simplest and most clear setting. Note also that we arranged the experiment so that the policy greedification step \( \pi_{k+1} \forall k \) is error-free, so we can solely focus on the interplay between the proximal term and the error caused by imperfect policy evaluation.

We applied 100 iterations of PMPI, then measured the quality of the resultant policy \( \pi := \pi_{100} \) as defined by the distance between its true value and that of the optimal policy, namely \( \| V^* - V^\pi \|_\infty \).

We repeated the experiment with different magnitudes of error, as well as different values of the \( \beta \) parameter.

\[
\text{Figure 1: Performance of approximate PMPI has a U-shaped dependence on the parameter } \beta = \frac{1}{1+c} .
\]

Results are averaged over 30 random seeds with \( n = 1 \) (left) and \( n = 3 \) (right).

From Figure 1, it is clear that the final performance exhibits a U-shape with respect to the parameter \( \beta \). It is also noticeable that the best-performing \( \beta \) is shifting to the right side (larger values) as we increase the magnitude of noise. This trend makes sense, and is consistent with what is predicted by Theorem 2: As the noise level rises, we have more incentive to use larger (but not too large) \( \beta \) values to hedge against it.

5.2 Atari Experiments

In this section, we evaluate the proximal (or Pro) agents relative to their original DQN-style counterparts on the Atari benchmark \cite{Bellemare2013}, and show that endowing the agent with the proximal term can lead into significant improvements in the interim as well as in the final performance. We next investigate the utility of our proposed proximal term through further experiments. Please see the Appendix for a complete description of our experimental pipeline.
5.2.1 Setup

We used 55 Atari games \cite{bellemare2013arcade} to conduct our experimental evaluations. Following \cite{machado2018revisiting} and \cite{castro2018rainbow}, we used sticky actions to inject stochasticity into the otherwise deterministic Atari emulator.

Our training and evaluation protocols and the hyper-parameter settings follow those of the Dopamine baseline \cite{castro2018rainbow}. To report performance, we measured the undiscounted sum of rewards obtained by the learned policy during evaluation. We further report the learning curve for all experiments averaged across 5 random seeds. We reiterate that we used the exact same hyper-parameters for all agents to ensure a sound comparison.

![Figure 2: A minimal hyper-parameter tuning for $\tilde{c}$ in DQN Pro (left) and Rainbow Pro (right).](image)

Our Pro agents have a single additional hyper-parameter $\tilde{c}$. We did a minimal random search on 6 games to tune $\tilde{c}$. Figure 2 visualizes the performance of Pro agents as a function of $\tilde{c}$. In light of this result, we set $\tilde{c} = 0.2$ for DQN Pro and $\tilde{c} = 0.05$ for Rainbow Pro. We used these values of $\tilde{c}$ for all 55 games, and note that we performed no further hyper-parameter tuning at all.

5.2.2 Results

The first question is whether endowing the DQN agent with the proximal term can yield significant improvements over the original DQN. Figure 3 (top) shows a comparison between DQN and DQN Pro in terms of the final performance. In particular, following standard practice \cite{wang2016deep, dabney2018asynchronous, van2019atari}, for each game we compute:

$$\frac{\text{Score}_{\text{DQN Pro}} - \text{Score}_{\text{DQN}}}{\max(\text{Score}_{\text{DQN}}, \text{Score}_{\text{Human}}) - \text{Score}_{\text{Random}}}.$$  

Bars shown in red indicate the games in which we observed better final performance for DQN Pro relative to DQN, and bars in blue indicate the opposite. The height of a bar denotes the magnitude of this improvement for the corresponding benchmark; notice that the y-axis is scaled logarithmically. We took human and random scores from previous work \cite{nair2015deep, dabney2018asynchronous}. It is clear that DQN Pro dramatically improves upon DQN. We defer to the Appendix for full learning curves on all games tested.

Can we fruitfully combine the proximal term with some of the existing algorithmic improvements in DQN? To answer this question, we build on the Rainbow algorithm of \cite{hessel2018rainbow} who successfully combined numerous important algorithmic ideas in the value-based RL literature. We present this result in Figure 3 (bottom). Observe that the overall trend is for Rainbow Pro to yield large performance improvements over Rainbow.

Additionally, we measured the performance of our agents relative to human players. To this end, and again following previous work \cite{wang2016deep, dabney2018asynchronous, van2019atari}, for each agent we compute the human-normalized score:

$$\frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Random}}}{\text{Score}_{\text{Human}} - \text{Score}_{\text{Random}}}.$$  

In Figure 4 (left), we show the median of this score for all agents, which \cite{wang2016deep} and \cite{hessel2018rainbow} argued is a sensible quantity to track. We also show per-game learning curves with standard error in the Appendix.
We make two key observations from this figure. First, the very basic DQN Pro agent is capable of achieving human-level performance (1.0 on the y-axis) after 120 million frames. Second, the Rainbow Pro agent achieves 220 percent human-normalized score after only 120 million frames.

5.2.3 Additional Experiments

Our purpose in endowing the agent with the proximal term was to keep the online network in the vicinity of the target network, so it would be natural to ask if this desirable property can manifest itself in practice when using the proximal term. In Figure 4, we answer this question affirmatively by plotting the magnitude of the update to the target network during synchronization. Notice that we periodically synchronize online and target networks, so the proximity of the online and target network should manifest itself in a low distance between two consecutive target networks. Indeed, the results demonstrate the success of the proximal term in terms of obtaining the desired proximity of online and target networks.

While using the proximal term leads to significant improvements, one may still wonder if the advantage of DQN Pro over DQN is merely stemming from a poorly-chosen period hyper-parameter in the original DQN, as opposed to a truly more stable optimization in DQN Pro. To refute this hypothesis, we ran DQN with various settings of the period hyper-parameter \{2000, 4000, 8000, 12000\}. This set included the default value of the hyper-parameter (8000) from the original paper (Mnih et al., 2015), but also covered a wider set of settings.

Additionally, we tried an alternative update strategy for the target network, referred to as Polyak averaging, which was popularized in the context of continuous-action RL (Lillicrap et al., 2015): \(\theta \leftarrow \tau w + (1 - \tau) \theta\). For this update strategy, too, we tried different settings of the \(\tau\) hyper-parameter, namely \{0.05, 0.005, 0.0005\}, which includes the value 0.005 used in numerous papers (Lillicrap et al., 2015; Fujimoto et al., 2018; Asadi et al., 2021).

Figure 5 presents a comparison between DQN Pro and DQN with periodic and Polyak target updates for various hyper-parameter settings of period and \(\tau\). It is clear that DQN Pro is consistently
outperforming the two alternatives regardless of the specific values of \textit{period} and \(\tau\), thus clearly demonstrating that the improvement is stemming from a more stable optimization procedure leading to a better interplay between the two networks.

Finally, an alternative approach to ensuring lower distance between the online and the target network is to anneal the step size based on the number of updates performed on the online network since the last online-target synchronization. In this case we performed this experiment in 4 games where we knew proximal updates provide improvements based on our DQN Pro versus DQN result in Figure\[3\]. In this case we linearly decreased the step size from the original DQN learning rate \(\alpha\) to \(\alpha' \ll \alpha\) where we tuned \(\alpha'\) using random search. Annealing indeed improves DQN, but DQN Pro outperforms the improved version of DQN. Our intuition is that Pro agents only perform small updates when the target network is far from the online network, but naively decaying the learning rate can harm progress when the two networks are in vicinity of each other.

6 Discussion

In our experience using proximal updates in the parameter space were far superior than proximal updates in the value space. We believe this is because the parameter-space definition can enforce the proximity globally, while in the value space one can only hope to obtain proximity locally and on a batch of samples. One may hope to use natural gradients to enforce value-space proximity in a more principled way, but doing so usually requires significantly more computational resources.
Figure 6: Learning-rate decay does not bridge the gap between DQN and DQN Pro.

Lerner (2018). This is in contrast to our proximal updates which add negligible computational cost in the simple form of taking a dimension-wise weighted average of two weight vectors.

In addition, for a smooth (Lipschitz) Q function, performing parameter-space regularization guarantees function-space regularization. Concretely: \( \forall s, \forall a \ |Q(s, a; \theta) - Q(s, a; \theta')| \leq L||\theta - \theta'|| \), where \( L \) is the Lipschitz constant of \( Q \). Moreover, deep networks are Lipschitz (Neyshabur et al., 2015; Asadi et al., 2018), because they are constructed using compositions of Lipschitz functions (such as ReLU, convolutions, etc) and that composition of Lipschitz functions is Lipschitz. So performing value-space updates may be an overkill. Lipschitz property of deep networks has successfully been leveraged in other contexts, such as in generative adversarial training Arjovsky et al. (2017).

A key selling point of our result is simplicity, because simple results are easy to understand, implement, and reproduce. We obtained significant performance improvements by adding just a few lines of codes to the publicly available implementations of DQN and Rainbow Castro et al. (2018).

7 Related Work

The introduction of proximal operators could be traced back to the seminal work of Moreau (1962, 1965), Martinet (1970) and Rockafellar (1976), and the use of the proximal operators has since expanded into many areas of science such as signal processing (Combettes & Pesquet, 2009), statistics and machine learning (Beck & Teboulle, 2009; Polson et al., 2015; Reddi et al., 2015), and convex optimization (Parikh & Boyd, 2014; Bertsekas, 2011b,a).

In the context of RL, Mahadevan et al. (2014) introduced a proximal theory for deriving convergent off-policy algorithms with linear function approximation. One intriguing characteristic of their work is that they perform updates in primal-dual space, a property that was leveraged in sample complexity analysis (Liu et al., 2020) for the proximal counterparts of the gradient temporal-difference algorithm (Sutton et al., 2008). Proximal operators also have appeared in the deep RL literature. For instance, Fakoor et al. (2020b) used proximal operators for meta learning, and Maggipinto et al. (2020) improved TD3 (Fujimoto et al., 2018) by employing a stochastic proximal-point interpretation.

The effect of the proximal term in our work is reminiscent of the use of trust regions in policy-gradient algorithms (Schulman et al., 2015, 2017; Wang et al., 2019; Fakoor et al., 2020a; Tomar et al., 2021). However, three factors differentiate our work: we define the proximal term using the value function, not the policy, we enforce the proximal term in the parameter space, as opposed to the function space, and we use the target network as the previous iterate in our proximal definition.

8 Conclusion and Future work

We showed a clear advantage for using proximal terms to perform slower but more effective updates in approximate planning and reinforcement learning. Our results demonstrated that proximal updates lead to more robustness with respect to noise. Several improvements to proximal methods exist, such as the acceleration algorithm (Nesterov, 1983; Li & Lin, 2015), as well as using other proximal terms (Combettes & Pesquet, 2009), which we leave for future work.

9 Acknowledgment

We thank Lihong Li, Pratik Chaudhari, and Shoham Sabach for their valuable insights in different stages of this work.
References


Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence o (1/k^2). In Doklady an USSR, 1983.


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our contributions are primarily abstract with no immediate societal application. That said, there are very important concerns regarding ethical and responsible applications of reinforcement learning, which requires us to think carefully to ensure we can hedge against reinforcement learning running amok in the real world. In particular, we understand maximizing a wrong objective, maximizing an objective with bad or unintended consequences, and safety guarantees or lack thereof, to be the primary examples of the kind of concerns that should be carefully thought about before deploying reinforcement learning.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] In the Appendix

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We are planning to release our code with the camera-ready version of our paper. We believe we have provided a thorough explanation of our ideas, and so in light of the simplicity of our reinforcement algorithm, reproducing our results should be easy even before the camera-ready version of our paper becomes available.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In the Appendix
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In the Appendix

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [not applicable as no asset was used]
   (b) Did you mention the license of the assets? [not applicable]
   (c) Did you include any new assets either in the supplemental material or as a URL? [not applicable]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [not applicable]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [not applicable]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [not applicable as we did not perform crowdsourcing]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [not applicable]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [not applicable]
10 Appendix

10.1 Pseudo-code for DQN Pro

Below, we present the pseudo-code for DQN Pro. Notice that the difference between DQN and DQN Pro is minimal (highlighted in gray).

Algorithm 1 DQN with Proximal Iteration (DQN Pro)

1: Initialize $\theta$, $N$, period, replay buffer $D$, $\alpha$, and $\tilde{\alpha}$
2: $s \leftarrow$ env.reset(), $w \leftarrow \theta$, numUpdates $\leftarrow 0$
3: repeat
4: $a \sim \epsilon$-greedy($Q(s, \cdot; w)$)
5: $s', r \leftarrow$ env.step($s, a$)
6: add $(s, a, r, s')$ to $D$
7: if $s'$ is terminal then
8: $s \leftarrow$ env.reset()
9: end if
10: for $n$ in $\{1, \ldots, N\}$ do
11: sample $B = \{(s, a, r, s')\}$, compute $\nabla_w h(w)$
12: $w \leftarrow (1 - (\alpha/\tilde{\alpha}))w + (\alpha/\tilde{\alpha})\theta - \alpha \nabla_w h(w)$
13: numUpdates $\leftarrow$ numUpdates + 1
14: if numUpdates $\%$ period $= 0$ then
15: $\theta \leftarrow w$
16: end if
17: end for
18: until convergence

10.2 Implementation Details

Table 1 and 2 show hyper-parameters, computing infrastructure, and libraries used for the experiments in this paper for all games tested. Our training and evaluation protocols and the hyper-parameter settings closely follow those of the Dopamine baseline. To report performance results, we measured the undiscounted sum of rewards obtained by the learned policy during evaluation.
<table>
<thead>
<tr>
<th>DQN hyper-parameters (shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replay buffer size</td>
</tr>
<tr>
<td>Target update period</td>
</tr>
<tr>
<td>Max steps per episode</td>
</tr>
<tr>
<td>Evaluation frequency</td>
</tr>
<tr>
<td>Batch size</td>
</tr>
<tr>
<td>Update period</td>
</tr>
<tr>
<td>Number of frame skip</td>
</tr>
<tr>
<td>Number of episodes to evaluate</td>
</tr>
<tr>
<td>Update horizon</td>
</tr>
<tr>
<td>$\epsilon$-greedy (training time)</td>
</tr>
<tr>
<td>$\epsilon$-greedy (evaluation time)</td>
</tr>
<tr>
<td>$\epsilon$-greedy decay period</td>
</tr>
<tr>
<td>Burn-in period / Min replay size</td>
</tr>
<tr>
<td>Learning rate</td>
</tr>
<tr>
<td>Discount factor ($\gamma$)</td>
</tr>
<tr>
<td>Total number of iterations</td>
</tr>
<tr>
<td>Sticky actions</td>
</tr>
<tr>
<td>Optimizer</td>
</tr>
<tr>
<td>Network architecture</td>
</tr>
<tr>
<td>Random seeds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rainbow hyper-parameters (shared)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
</tr>
<tr>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DQN Pro and Rainbow Pro hyper-parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{c}$ (DQN Pro)</td>
</tr>
<tr>
<td>$\tilde{c}$ (Rainbow Pro)</td>
</tr>
</tbody>
</table>

Table 1: Hyper-parameters used for all methods for all 55 games of Atari-2600 benchmarks. All results reported in our paper are averages over repeated runs initialized with each of the random seeds listed above and run for the listed number of episodes.

<table>
<thead>
<tr>
<th>Computing Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Type</td>
</tr>
<tr>
<td>GPU Family</td>
</tr>
<tr>
<td>CPU Family</td>
</tr>
<tr>
<td>CUDA Version</td>
</tr>
<tr>
<td>NVIDIA-Driver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Library Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
</tr>
<tr>
<td>Numpy</td>
</tr>
<tr>
<td>Gym</td>
</tr>
<tr>
<td>Pytorch</td>
</tr>
</tbody>
</table>

Table 2: Computing infrastructure and software libraries used in all experiments in this paper.
10.3 Proofs

Theorem 1. The Proximal Bellman Optimality Operator $T_{c,f}^*$ is a contraction with fixed point $v^*$.

We make two assumptions:

1. $f$ is smooth, or more specifically that its gradient is 1-Lipschitz: $||\nabla f(v_1) - \nabla f(v_2)|| \leq ||v_1 - v_2|| \forall v_1, v_2$.
2. the value of the parameter $c$ is large, in particular $c > \frac{2}{1-\gamma}$.

Proof. Both terms are convex and differentiable, therefore by setting the gradient to zero, we have:

\[
T_{c,f}^* v = T^* v + \frac{1}{c} (\nabla f(v) - \nabla f(T_{c,f}^* v)),
\]

We can then show:

\[
||T_{c,f}^* v_1 - T_{c,f}^* v_2|| = ||T^* v_1 + \frac{1}{c} (\nabla f(v_1) - \nabla f(T_{c,f}^* v_1)) - T^* v_2 - \frac{1}{c} (\nabla f(v_2) - \nabla f(T_{c,f}^* v_2))||
\]

\[
\leq ||T^* v_1 - T^* v_2|| + \frac{1}{c} ||\nabla f(v_1) - \nabla f(T_{c,f}^* v_1)|| + \frac{1}{c} ||\nabla f(v_2) - \nabla f(T_{c,f}^* v_2)||
\]

(first assumption)

\[
\leq ||T^* v_1 - T^* v_2|| + \frac{1}{c} ||\nabla f(v_1) - \nabla f(v_2)|| + \frac{1}{c} ||T_{c,f}^* v_1 - T_{c,f}^* v_2||
\]

This implies:

\[
\frac{c-1}{c} ||T_{c,f}^* v_1 - T_{c,f}^* v_2|| \leq ||T^* v_1 - T^* v_2|| + \frac{1}{c} ||\nabla f(v_1) - \nabla f(v_2)||
\]

(first assumption)

\[
\leq ||T^* v_1 - T^* v_2|| + \frac{1}{c} ||v_1 - v_2||
\]

\[
\leq \frac{\gamma c + 1}{c} ||v_1 - v_2||
\]

Therefore,

\[
||T_{c,f}^* v_1 - T_{c,f}^* v_2|| \leq \frac{\gamma c + 1}{c-1} ||v_1 - v_2||,
\]

Allowing us to conclude that $T_{c,f}^*$ is a contraction (second assumption).

Further, to show that $v^*$ is indeed the fixed point of $T_{c,f}^*$, notice from the original formulation:

\[
T_{c,f}^* v := \arg \min_{v'} ||v' - T^* v||^2 + \frac{1}{c} D_f(v', v),
\]

that, at point $v^*$ setting $v' = v^*$ jointly minimizes the first term, because $v^* = T^* v^*$ due to fixed-point definition, and it also minimizes the second term because $D_f(v^*, v^*) = 0$ and that Bregman divergence is non-negative. Therefore, $T_{c,f}^* v^* = v^*$; $v^*$ is the fixed-point of $T_{c,f}^* v$. Since, $T_{c,f}^*$ is a contraction, this fixed point is unique. 

Theorem 2. Consider the PMPI algorithm specified by:

\[
\pi_k \leftarrow G_{\epsilon_k} v_{k-1}, \quad (4)
\]

\[
v_k \leftarrow (T_{\beta}^n)^n v_{k-1} + (1 - \beta) \epsilon_k. \quad (5)
\]

Define the Bellman residual $b_k := v_k - T_{\pi}^n v_k$, and error terms $x_k := (I - \gamma P_{\pi}) \epsilon_k$ and $y_k := \gamma P_{\pi}^* \epsilon_k$. After $k$ steps:

\[
v^* - v^{\pi_k} = \underbrace{(T_{\beta}^n)^n v_k}_d_k + \underbrace{(T_{\beta}^n)^n v_k - v_{\pi_k}}_{s_k}
\]

16
where \( d_k \leq \gamma P^\pi d_{k-1} - ((1 - \beta)y_{k-1} + \beta b_{k-1}) + (1 - \beta) \sum_{j=1}^{n-1} (\gamma P^{\pi_k})^j b_{k-1} + \epsilon_k \)

- \( s_k \leq ((1 - \beta)(\gamma P^{\pi_k})^n + \beta I)(I - \gamma P^{\pi_k})^{-1} b_{k-1} \)
- \( b_k \leq ((1 - \beta)(\gamma P^{\pi_k})^n + \beta I)b_{k-1} + (1 - \beta)x_k + \epsilon_k'_{k+1} \)

We make two assumptions:

1. we assume \( \epsilon \) error in policy evaluation step, as already stated in equation (4).
2. we assume \( \epsilon' \) error in policy greedification step \( \pi_k \leftarrow G_{\epsilon'}y_{k-1}\forall k \). This means \( \forall \pi T^\pi v_k - T^{\pi + \epsilon_{k+1}}v_k \leq \epsilon_{k+1} \). Note that this assumption is orthogonal to the thesis of our paper, but we kept it for generality.

**Proof.**

**Step 0:** bound the Bellman residual: \( b_k := v_k - T^{\pi_{k+1}}v_k \).

\[
b_k = v_k - T^{\pi_{k+1}}v_k \\
\leq v_k - T^{\pi_{k+1}}v_k + \epsilon'_{k+1}
\]

\(= v_k - (1 - \beta)\epsilon_k - T^{\pi_k}v_k + (1 - \beta)\gamma P^{\pi_k}\epsilon_k + (1 - \beta)\gamma P^{\pi_k}\epsilon_k + \epsilon'_{k+1}
\]

\(= v_k - (1 - \beta)\epsilon_k - T^{\pi_k}(v_k - (1 - \beta)\epsilon_k) + (1 - \beta)\gamma P^{\pi_k}\epsilon_k + \epsilon'_{k+1}
\]

\(= b_k - (1 - \beta)\gamma P^{\pi_k}(v_k - (1 - \beta)\epsilon_k) + (1 - \beta)\epsilon_k'_{k+1}
\]

allowing us to conclude:

\( b_k = ((1 - \beta)(\gamma P^{\pi_k})^n + \beta I)b_{k-1} + (1 - \beta)x_k + \epsilon_k'_{k+1} \).

**Step 1:** bound the distance to the optimal value: \( d_{k+1} := v^* - (T^{\pi_{k+1}})^n v_k \).

\[
d_{k+1} = v^* - (T^{\pi_{k+1}})^n v_k \\
\leq \gamma P^{\pi^*}(v^* - v_k) + \epsilon'_{k+1} + g_{k+1}
\]

\(= \gamma P^{\pi^*}(v^* - v_k) + (1 - \beta)\gamma P^{\pi^*}\epsilon_k - (1 - \beta)\gamma P^{\pi^*}\epsilon_k + \epsilon'_{k+1} + g_{k+1}
\]

\(= \gamma P^{\pi^*}(v^* - (v_k - (1 - \beta)\epsilon_k)) - (1 - \beta)\gamma P^{\pi^*}\epsilon_k + \epsilon'_{k+1} + g_{k+1}
\]

\(= \gamma P^{\pi^*}d_k - (1 - \beta)y_k + \epsilon'_{k+1} + g_{k+1}
\]
Additionally we can bound $g_{k+1}$ as follows:

\[
g_{k+1} = T^{\pi_{k+1}} v_k - (T_{\beta}^{\pi_{k+1}})^n v_k \\
= (1 - \beta)(T^{\pi_{k+1}} v_k - (T_{\beta}^{\pi_{k+1}})^n v_k) + \beta(T^{\pi_{k+1}} v_k - v_k) \\
= (1 - \beta) \sum_{j=1}^{n-1} (\gamma P_{\pi_{k+1}})^j b_k + \beta(-b_k)
\]

Allowing us to conclude that:

\[
d_{k+1} \leq \gamma P^{\pi_k} d_k - ((1 - \beta)y_k + \beta b_k) + (1 - \beta) \sum_{j=1}^{n-1} (\gamma P_{\pi_{k+1}})^j b_k + \epsilon_{k+1}
\]

Step 2: bound the distance between the approximate value and the value of the policy: $s_k := (T_{\beta}^{\pi_k})^n v_{k-1} - v^{\pi_k}$.

\[
s_k = (T_{\beta}^{\pi_k})^n v_{k-1} - v^{\pi_k} \\
= (T_{\beta}^{\pi_k})^n v_{k-1} - (T^{\pi_k})^\infty v_{k-1} \\
= (1 - \beta)(T^{\pi_k})^n v_{k-1} + \beta v_{k-1} - (1 - \beta)(T^{\pi_k})^\infty v_{k-1} - \beta(T^{\pi_k})^\infty v_{k-1} \\
= (1 - \beta)((T^{\pi_k})^n v_{k-1} - (T^{\pi_k})^\infty v_{k-1}) + \beta(v_{k-1} - (T^{\pi_k})^\infty v_{k-1}) \\
= (1 - \beta)(\gamma P_{\pi_k})^n v_{k-1} - (T^{\pi_k})^\infty v_{k-1} + \beta v_{k-1} - (T^{\pi_k})^\infty v_{k-1} \\
= \left((1 - \beta)(\gamma P_{\pi_k})^n + \beta I\right)(v_{k-1} - (T^{\pi_k})^\infty v_{k-1}) \\
= \left((1 - \beta)(\gamma P_{\pi_k})^n + \beta I\right)((I - \gamma P^{\pi_k})^{-1} v_{k-1} - T^{\pi_k} v_{k-1})
\]

Allowing us to conclude that:

\[
s_k = (1 - \beta)(\gamma P^{\pi_k})^n + \beta I((I - \gamma P^{\pi_k})^{-1} b_{k-1})
\]
11 Learning curves

We present full learning curves of DQN, DQN Pro, Rainbow, and Rainbow Pro for the 55 Atari games. All results are averaged over 5 independent seeds.
Figure 8: Comparison between DQN Pro (red) and DQN (blue) over 55 Atari games (Part II).
Figure 9: Comparison between Rainbow Pro (yellow) and Rainbow (green) over 55 Atari games (Part I).
Figure 10: Comparison between Rainbow Pro (yellow) and Rainbow (green) over 55 Atari games (Part II).
Figure 11: A study on games with the strongest (top) and weakest performance for Rainbow Pro with the original $\tilde{c} = 0.05$. Using a slightly less powerful proximal term (corresponding to larger $\tilde{c} = 0.1$) enables us to recover the downside (bottom) while still providing benefits on games that are more conducive to using the proximal updates (top).

12 Motivating Example for Adaptive $\tilde{c}$

In this section we specifically look at 4 domains in which Rainbow Pro did significantly better than the original Rainbow, as well 4 domains where Rainbow Pro is underperforming Rainbow. Note, again, that it is uncommon for Rainbow Pro with the original $\tilde{c}$ to underperform, but here we have a deeper dive into these cases for a better understanding.

From Figure 11 we observe that by using a slightly larger value of $\tilde{c}$, which slightly decreases the incentive for online-target proximity, we can recover from the downside, while still maintaining superior performance on games that are conducive to proximal updates. This suggest that, while using a fixed $\tilde{c}$ value is enough to obtain significant performance improvement, adaptively choosing $\tilde{c}$ would provide us with even more reliable improvements when performing proximal updates. In this context, a promising idea would be to hinge on the variance of our gradient updates when setting $\tilde{c}$. We leave this promising direction for future work.
13 Proximal Updates in the Value Space

Our primary contribution was to show the usefulness of performing proximal updates in the parameter space. That said, we also implemented a version of proximal updates that operated in the value space. More specifically, in this case we updated the parameters of the online network as follows:

\[
    w \leftarrow E_{(s,a,r,s')} \left[ (r + \gamma \max_{a'} \hat{Q}(s', a'; \theta) - \hat{Q}(s, a; w))^2 \right] + \frac{1}{C} E_{(s,a)} \left[ (\hat{Q}(s, a; w) - \hat{Q}(s, a; \theta))^2 \right].
\]

We conducted numerous experiments using variants of this idea (such as using separate replay buffer for each term, performing the update for all actions in buffered states, etc) but we generally found the value-space updates to be ineffective. As mentioned in the main paper, we believe this is because the parameter-space definition can enforce the proximity globally, while in the value space one can only hope to obtain proximity locally and on a batch of samples. To perform global updates we may need to compute the natural gradient, which typically requires matrix invasion [Knight & Lerner (2018)], and thus adding significant computational burden to the original algorithm. In comparison, the parameter-space version is effective, simple to implement, and capable of enforcing proximity globally due to the Lipschitz property of neural networks.