Deep Residual Reinforcement Learning

Shangtong Zhang, Wendelin Boehmer, Shimon Whiteson
Department of Computer Science, University of Oxford, United Kingdom
shangtong.zhang@cs.ox.ac.uk

ABSTRACT

We revisit *residual algorithms* in both model-free and model-based reinforcement learning settings. We propose the *bidirectional target network* technique to stabilize residual algorithms, yielding a residual version of DDPG that significantly outperforms vanilla DDPG in the DeepMind Control Suite benchmark. Moreover, we find the residual algorithm an effective approach to the *distribution mismatch* problem in model-based planning. Compared with the existing TD(k) method, our residual-based method makes weaker assumptions about the model and yields a greater performance boost.

KEYWORDS

reinforcement learning, residual algorithms

ACM Reference Format:
Shangtong Zhang, Wendelin Boehmer, Shimon Whiteson. 2020. Deep Residual Reinforcement Learning. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Semi-gradient algorithms have recently enjoyed great success in deep reinforcement learning (RL) problems, e.g., DQN [33] achieves human-level control in the Arcade Learning Environment (ALE, [3]). However, such algorithms lack theoretical support. Most semi-gradient algorithms suffer from divergence under nonlinear function approximation or off-policy training [2, 52]. By contrast, *residual gradient* (RG, [2]) algorithms are true stochastic gradient algorithms and enjoy convergence guarantees (to a local minimum) under mild conditions with both nonlinear function approximation and off-policy training. Baird [2] further proposes *residual algorithms* (RA) to unify residual gradients and semi-gradients via mixing them together.

Residual algorithms suffer from the double sampling issue [2]: two independently sampled successor states are required to compute the residual gradients. This requirement can be easily satisfied in model-based RL or in deterministic environments. However, even in these settings, residual algorithms have long been either overlooked or dismissed as impractical. In this paper, we aim to overturn that conventional wisdom with new algorithms built on RA and empirical results showing their efficacy.

Our contributions are threefold. First, we give a thorough overview of existing comparisons between residual gradient algorithms and semi-gradient algorithms.

Second, we showcase the advantages of RA in a model-free RL setting with deterministic environments. While target networks [33] are usually an important component in deep RL algorithms to stabilize training [29, 33], we find a naive combination of target networks and residual algorithms, in general, does not improve performance. Therefore, we propose the *bidirectional target network* technique to stabilize residual algorithms. We show that our residual version of Deep Deterministic Policy Gradients (DDPG, [29]) significantly outperforms vanilla DDPG in the DeepMind Control Suite (DMControl, [51]) and Mujoco benchmarks.

Third, we showcase the advantages of RA in a model-based RL setting, where a learned model generates imaginary transitions to train the value function. In general, model-based methods suffer from a *distribution mismatch* problem [14]. The value function trained on real states does not generalize well to imaginary states generated by a model. To address this issue, Feinberg et al. [14] train the value function on both real and imaginary states via the TD(k) trick. However, TD(k) requires that predictions k steps in the future made by model rollouts will be accurate [14]. In this paper, we show that RA naturally allows the value function to be trained on both real and imaginary states and requires only 1-step rollouts. Our experiments show that RA-based planning boosts performance more than TD(k)-based planning in most cases.

2 BACKGROUND

We consider an MDP [37] consisting of a finite state space $\mathcal{S}$, a finite action space $\mathcal{A}$, a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, a transition kernel $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ and a discount factor $\gamma \in [0, 1)$. With $\pi : \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ denoting a policy, at time t, an agent at a state S_t takes an action A_t according to $\pi(\cdot|S_t)$. The agent then gets a reward R_{t+1} satisfying $\mathbb{E}[R_{t+1}] = r(S_t, A_t)$ and proceeds to a new state S_{t+1} according to $p(\cdot|S_t, A_t)$. We use $G_t \doteq \sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i$ to denote the return from time t, $v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$ to denote the state value function of π, and $q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$ to denote the state-action value function of π. In the rest of this section, we use a bold capital letter to denote a matrix and a bold lowercase letter to denote a column vector. We use $\mathbf{P}_\pi$ to denote the transition matrix induced by a policy π, i.e., $\mathbf{P}_\pi[s, s'] \doteq \sum_a \pi(s, a) p(s'|s, a)$, and use d_π to denote its unique stationary distribution, assuming $\mathbf{P}_\pi$ is ergodic. The reward vector induced by π is $\mathbf{r}_\pi[s] = \sum_a \pi(a|s) r(s, a)$.

The value function v_π is the unique fixed point of the Bellman operator $\mathcal{T}$ [4]. In a matrix form, $\mathcal{T}$ is defined as $\mathcal{T}\mathbf{v} \doteq \mathbf{r}_\pi + \gamma \mathbf{P}_\pi \mathbf{v}$, where $\mathbf{v}$ can be any vector in $\mathbb{R}^N$. Here $N \doteq |\mathcal{S}|$ is the number of states.

Policy Evaluation: We consider the problem of finding v_π for a given policy π and use $\mathbf{v}$, parameterized by $\mathbf{w} \in \mathbb{R}^d$, to denote an estimate of $\mathbf{v}_\pi$, the vector form of v_π. We start with on-policy linear function approximation and use $x : \mathcal{S} \rightarrow \mathbb{R}^d$ to denote a feature function which maps a state to a d-dimensional feature. The feature matrix is then $\mathbf{X} \doteq [\mathbf{x}(s_1), \ldots, \mathbf{x}(s_N)]^{\mathrm{T}} \in \mathbb{R}^{N \times d}$, and the value estimate is $\mathbf{v} \doteq \mathbf{X}\mathbf{w}$.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand. © 2020 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

To approximate $\mathbf{v}_\pi$, one direct goal is to minimize the Mean Squared Value Error:

$$\text{MSVE}(\mathbf{w}) \doteq ||\mathbf{v} - \mathbf{v}_\pi||^2_{d_\pi} \doteq \sum_s d_\pi(s)\big(\mathbf{v}(s) - \mathbf{v}_\pi(s)\big)^2.$$

To minimize MSVE, a Monte Carlo return can be used as a sample for $\mathbf{v}_\pi$ to train $\mathbf{v}$. However, this method suffers from a large variance and usually requires off-line learning [5]. To address those issues, we consider minimizing the Mean Squared Projected Bellman Error (MSPBE) and the Mean Squared Bellman Error (MSBE):

$$\text{MSPBE}(\mathbf{w}) \doteq ||\mathbf{v} - \Pi\mathcal{T}\mathbf{v}||^2_{d_\pi}, \quad \text{MSBE}(\mathbf{w}) \doteq ||\mathbf{v} - \mathcal{T}\mathbf{v}||^2_{d_\pi}.$$

Here Π is a projection operator which maps an arbitrary vector onto the column vector space of $\mathbf{X}$, minimizing a d_π-weighted projection error, i.e., $\Pi\mathbf{v} \doteq \mathbf{X}\bar{\mathbf{w}}$, where $\bar{\mathbf{w}} \doteq \arg\min_{\mathbf{w}} ||\mathbf{v} - \mathbf{X}\mathbf{w}||^2_{d_\pi}$. With linear function approximation and fixed features, Π is linear.

There are various algorithms for minimizing MSPBE and MSBE. Temporal Difference learning (TD, [44]) is commonly used to minimize MSPBE. TD updates $\mathbf{w}$ as

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(R_{t+1} + \gamma\mathbf{v}(S_{t+1}) - \mathbf{v}(S_t)\big)\nabla_{\mathbf{w}}\mathbf{v}(S_t),$$

where α is a step size. Under mild conditions, on-policy linear TD converges to the point where MSPBE is 0 [52]. TD is a *semi-gradient* [46] algorithm in that it ignores the dependency of $\mathbf{v}(S_{t+1})$ on $\mathbf{w}$. There are also *true gradient* algorithms for optimizing MSPBE, e.g., Gradient TD methods [47]. Gradient TD methods compute the gradient of MSPBE directly and also enjoy convergence guarantees.

Baird [2] proposes *residual gradient* algorithms for minimizing MSBE, which updates $\mathbf{w}$ as

$$\begin{aligned}\mathbf{w} \leftarrow \mathbf{w} - \alpha\big(R_{t+1} + \gamma\mathbf{v}(S_{t+1}) - \mathbf{v}(S_t)\big) \\ \cdot\big(\gamma\nabla_{\mathbf{w}}\mathbf{v}(S'_{t+1}) - \nabla_{\mathbf{w}}\mathbf{v}(S_t)\big),\end{aligned} \tag{1}$$

where S'_{t+1} is another sampled successor state for S_t, independent of S_{t+1}. This requirement for two independent samples is known as the *double sampling issue* [2]. If both the transition kernel p and the policy π are deterministic, we can simply use one sample without introducing bias. Otherwise, we may need to have access to the transition kernel p, which is usually not available in model-free RL. Regardless, RG is a true gradient algorithm with convergence guarantees under mild conditions.

We now expand our discussion about policy evaluation into off-policy learning and nonlinear function approximation, where the states $\{S_t\}$ are drawn according to a behavior policy μ instead of the target policy π. True gradient algorithms like Gradient TD methods and RG remain convergent to local minima under off-policy training with any function approximator [2, 31, 47]. However, the empirical success of Gradient TD methods is limited to simple domains due to its large variance [48]. Semi-gradient algorithms are not convergent in general, e.g., the divergence of off-policy linear TD is well-documented [52].

Semi-gradient algorithms are fast but in general not convergent. Residual gradient algorithms are convergent but slow [2]. To take advantage of both, Baird [2] proposes to mix semi-gradients and residual gradients together, yielding the *residual algorithms.* The RA version of TD [2] updates $\mathbf{w}$ as

$$\begin{aligned}\mathbf{w} \leftarrow \mathbf{w} - \alpha\big(R_{t+1} + \gamma\mathbf{v}(S_{t+1}) - \mathbf{v}(S_t)\big) \\ \cdot\big(\gamma\eta\nabla_{\mathbf{w}}\mathbf{v}(S'_{t+1}) - \nabla_{\mathbf{w}}\mathbf{v}(S_t)\big),\end{aligned}$$

where $\eta \in [0, 1]$ controls how the two gradients are mixed. Little empirical study has been conducted for RA.

Control: We now consider the problem of control, where we are interested in finding an optimal policy π^* such that $v_{\pi^*}(s) \geq v_\pi(s)\,\forall(\pi, s)$. We use q_* to denote the state-action value function of π^* and Q to denote an estimate of q_*, parameterized by θ. Q-learning [54] is usually used to train Q and enjoys convergence guarantees in the tabular setting. When Q-learning is combined with neural networks, Deep-Q-Networks (DQN, [33]) update θ as

$$\begin{aligned}\theta \leftarrow \theta + \alpha_1(r_{t+1} + \max_a \bar{Q}(s_{t+1}, a) - Q(s_t, a_t)) \\ \cdot\nabla_\theta Q(s_t, a_t),\end{aligned} \tag{2}$$

where α_1 is a step size, $(s_t, a_t, r_{t+1}, s_{t+1})$ is a transition sampled from a replay buffer [30], and $\bar{Q}$ indicates the estimate is from a target network [33], parameterized by θ^-, which is synchronized with θ periodically.

When the action space is continuous, it is hard to perform the max operation in the DQN update (2). DDPG can be interpreted as a continuous version of DQN, where an actor $\mu : \mathcal{S} \rightarrow \mathcal{A}$, parameterized by ν, is trained to output the greedy action. DDPG updates θ and ν as

$$\begin{aligned}\theta \leftarrow \theta + \alpha_1\big(r_{t+1} \\ + \gamma\bar{Q}(s_{t+1}, \bar{\mu}(s_{t+1})) - Q(s_t, a_t)\big)\nabla_\theta Q(s_t, a_t),\end{aligned} \tag{3}$$

$$\nu \leftarrow \nu + \alpha_2\nabla_a Q(s_t, a)|_{a=\mu(s_t)}\nabla_\nu\mu(s_t), \tag{4}$$

where α_2 is a step size, $\bar{\mu}$ indicates the greedy action is from a target network, parameterized by ν^-.

Both DQN and DDPG are semi-gradient algorithms. There are also true gradient methods for control, e.g., Greedy-GQ [32] and the residual version of Q-learning [2]. As with Gradient TD methods, the empirical success of Greedy-GQ is limited to simple domains due to its large variance [48].

3 COMPARING TD AND RG

In this section, we review existing comparisons between RG and TD. We start by comparing their fixed points, MSBE and MSPBE, in the setting of linear function approximation.

Cons of MSBE:

- Sutton and Barto [46] show that MSBE is not uniquely determined by the observed data. Different MDPs may have the same data distribution due to state aliasing, but the minima of MSBE can still be different. This questions the learnability of MSBE as sampled transitions are all that is available in model-free RL. By contrast, the minima of MSPBE are always the same for MDPs with the same data distribution.
- Empirically, optimizing MSBE can lead to unsatisfying solutions. For example, in the A-presplit example [46], the value of most states can be represented accurately by the function approximator but the MSBE minimizer does not do so, while the MSPBE minimizer does. Furthermore, empirically the MSBE minimizer can be further from the MSVE minimizer than the MSPBE minimizer [11].

Pros of MSBE:

- Williams and Baird [56] show MSBE can be used to bound MSVE (up to a constant). By contrast, at a point where MSPBE is minimized, MSVE can be arbitrarily large [5].
- MSBE is an upper bound of MSPBE [38], indicating that optimizing MSBE implicitly optimizes MSPBE.

We now compare RG and TD.

Cons of RG:

- Due to the double sampling issue, it is usually hard to apply RG in the stochastic model-free setting [2], while TD is compatible with both deterministic and stochastic environments.
- RG is usually slower than TD. Empirically, this is observed by Baird [2], van Hasselt [53], Gordon [19] and Gordon [20]. Intuitively, in the RG update (1), a state S_t and its successor S'_{t+1} are often similar under function approximation. As a result, the two gradients $\nabla_{\mathbf{w}} \mathbf{v}(S_t)$ and $\nabla_{\mathbf{w}} \mathbf{v}(S'_{t+1})$ tend to be similar and cancel each other, slowing down the learning. Theoretically, Schoknecht and Merke [40] prove TD converges faster than RG in a tabular setting.
- Lagoudakis and Parr [27] argue that TD usually provides a better solution than RG, even though the value function is not as well approximated. The TD solution "preserves the shape of the value function to some extent rather than trying to fit the absolute values". Thus "the improved policy from the corresponding approximate value function is closer to the improved policy from the exact value function" [27, 28, 43].

Pros of RG:

- RG is a true gradient algorithm and enjoys convergence guarantees in most settings under mild conditions. By contrast, the divergence of TD with off-policy learning or nonlinear function approximation is well documented [52]. Empirically, Munos [34] and Li [28] show that RG is more stable than TD.
- Schoknecht and Merke [40] observe that RG converges faster than TD in the four-room domain [49] with linear function approximation. Scherrer [38] shows empirically that the TD solution is usually slightly better than RG but in some cases fails dramatically.

Others:

- Li [28] proves that TD makes more accurate predictions (i.e., the predicted state value is close to the true state value), while RG yields smaller temporal differences (i.e., the value predictions for a state and its successor are more consistent). This is also explained in Sutton and Barto [46].

To summarize, previous insights about RG and TD are mixed. There is little empirical study for RG in deep RL problems, much less RA. It is not clear whether and how we can take advantage of RA in model-free and model-based RL to solve deep RL problems.

4 RESIDUAL ALGORITHMS IN MODEL-FREE RL

In this section, we investigate how to combine RA and DDPG. In particular, we consider (almost) deterministic environments (e.g., DMControl) to avoid the double sampling issue.

In semi-gradient algorithms, value propagation goes backwards in time. The value of a state depends on the value of its successor through bootstrapping, and a target network is used to stabilize this bootstrapping. RA allows value propagation both forwards and backwards. The value of a state depends on the value of both its successor and predecessor. Therefore, we need to stabilize the bootstrapping in both directions. To this end, we propose the *bidirectional target network* technique. Employing this in DDPG yields Bi-Res-DDPG, which updates the critic parameters θ as:

$$\begin{aligned}\theta \leftarrow \theta &- \alpha_1\big(r_{t+1} + \gamma\bar{Q}(s_{t+1}, \bar{\mu}(s_{t+1})) - Q(s_t, a_t)\big)\\ &\times\big(-\nabla_\theta Q(s_t, a_t)\big)\\ &- \alpha_1\big(r_{t+1} + \gamma Q(s_{t+1}, \mu(s_{t+1})) - \bar{Q}(s_t, a_t)\big)\\ &\times \eta\gamma\nabla_\theta Q(s_{t+1}, \mu(s_{t+1})),\end{aligned}$$

where $\bar{Q}, \bar{\mu}$ are target networks and $\eta \in [0, 1]$ controls how the two gradients are mixed. The actor update remains unchanged.

We compared Bi-Res-DDPG to DDPG in 28 DMControl tasks and 5 Mujoco tasks. Our DDPG implementation uses the same architecture and hyperparameters as Lillicrap et al. [29], which are inherited by Bi-Res-DDPG (and all other DDPG variants in this paper). For Bi-Res-DDPG, we tune η over $\{0, 0.05, 0.1, 0.2, 0.4, 0.8, 1\}$ on `walker-stand` and use $\eta = 0.05$ across all tasks. We perform 20 deterministic evaluation episodes every 10^4 training steps and plot the averaged evaluation episode returns. All curves are averaged over 5 independent runs and are available in the appendix. In the main text, we report the improvement of AUC (area under the curve) of the evaluation curves in Figure 2. AUC serves as a proxy for learning speed (e.g., see Example 8.2 in Sutton and Barto [46]). Bi-Res-DDPG achieves a 20% (41%) AUC improvement over the original DDPG in terms of the median (mean). Our DDPG baseline reaches the same performance level as the DDPG baseline in Fujimoto et al. [16] and Buckman et al. [6] in Mujoco tasks.

To further investigate the relationship between the target network and RA, we study several variants of DDPG. We define a shorthand $g_t \doteq \eta\gamma\nabla_\theta Q(s_{t+1}, \mu(s_{t+1})) - \nabla_\theta Q(s_t, a_t)$ and the update rule for θ is $\theta \leftarrow \theta - \alpha_1(r_{t+1} + \Delta)g_t$, where Δ is different for different variants. We use "T" and "O" to denote the target network and the online network respectively. We have:

$$\text{Res-DDPG:}\Delta \doteq \gamma Q(s_{t+1}, \mu(s_{t+1})) - Q(s_t, a_t), \quad (5)$$

$$\text{TO-Res-DDPG:}\Delta \doteq \gamma \bar{Q}(s_{t+1}, \bar{\mu}(s_{t+1})) - Q(s_t, a_t), \quad (6)$$

$$\text{OT-Res-DDPG:}\Delta \doteq \gamma Q(s_{t+1}, \mu(s_{t+1})) - \bar{Q}(s_t, a_t), \quad (7)$$

$$\text{TT-Res-DDPG:}\Delta \doteq \gamma \bar{Q}(s_{t+1}, \bar{\mu}(s_{t+1})) - \bar{Q}(s_t, a_t). \quad (8)$$

Res-DDPG is a direct combination of RA and DDPG without a target network. TO-Res-DDPG simply adds a residual gradient term to the original DDPG. OT-Res-DDPG stabilizes the bootstrapping for the forward value propagation. TT-Res-DDPG stabilizes bootstrapping in both directions but destroys the connection between prediction and error. By contrast, Bi-Res-DDPG stabilizes bootstrapping in both directions and maintains the connection between prediction and error.

Figure 1 compares these variants on `walker-stand`. The main points to note are: (1) Both Bi-Res-DDPG($\eta = 0$) and TO-Res-DDPG($\eta = 0$) are the same as vanilla DDPG. The curves are similar, verifying the stability of our implementation. (2) Res-DDPG($\eta = 0$) corresponds to vanilla DDPG without a target network, which performs poorly. This confirms that a target network is important

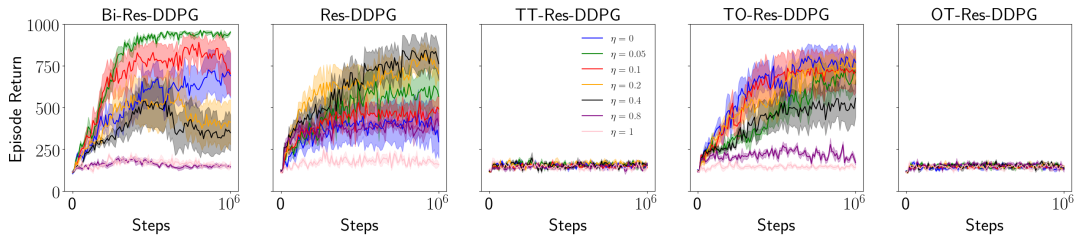

Figure 1: Performance of Bi-Res-DDPG variants on `walker-stand`, focusing on the role of target networks.

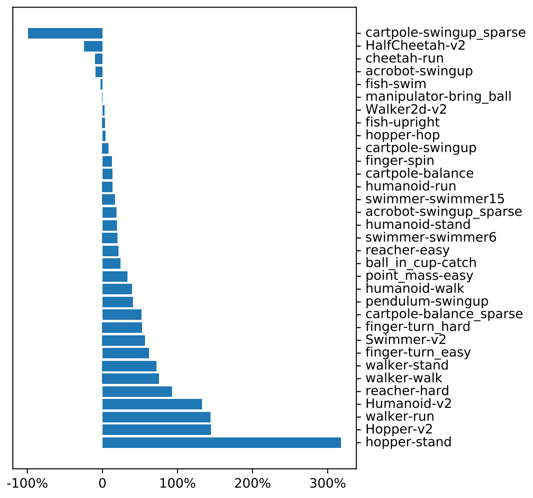

Figure 2: AUC improvements of Bi-Res-DDPG over DDPG on 28 DMControl tasks and 5 Mujoco tasks, computed as $\frac{\text{AUC}_{\text{Bi-Res-DDPG}} - \text{AUC}_{\text{DDPG}}}{\text{AUC}_{\text{DDPG}}}$.

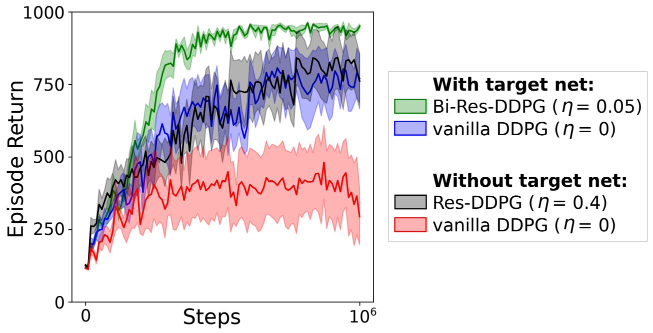

Figure 3: A selection of the best parameters η from Figure 1. Note that residual updates stabilize performance as much as the introduction of target networks.

for stabilizing training and mitigating divergence when a nonlinear function approximator is used [29, 33]. (3) Increasing η improves Res-DDPG's performance. This complies with the argument from Baird [2] that residual gradients help semi-gradients converge. All variants fail with a large η (e.g., 0.8 or 1). This complies with the argument from Baird [2] that pure residual gradients are slow. (4) TO-Res-DDPG($\eta = 0$) (i.e., vanilla DDPG) is similar to Res-DDPG($\eta = 0.4$), indicating a naive combination of RA and DDPG without a target network is ineffective. (5) For TO-Res-DDPG, $\eta = 0$ achieves the best performance, indicating adding a residual gradient term to DDPG directly is ineffective. To summarize, these variants confirm the necessity of the bidirectional target network. To better understand the role of residual updates, we summarize the results of Figure 1 in Figure 3. Res-DDPG does not have a target network and outperforms DDPG without a target network. Res-DDPG also increases the stability. Bi-Res-DDPG has target networks and also outperforms DDPG with a target network, as well as increases the stability. This comparison confirms the importance of residual updates.

We also evaluated a Bi-Res version of DQN in three ALE environments (BeamRider, Seaquest, Breakout). The performance was similar to the original DQN. One of the many differences between DMControl and ALE is that rewards in ALE are much more sparse. This might indicate that the forward value propagation in RA is less likely to yield a performance boost with sparse rewards.

We do not expect residual updates to improve the performance of all semi-gradient baselines. However, our results do show that the residual update together with the bidirectional target network is beneficial in many tasks. Despite the popularity of semi-gradient methods, we do believe residual algorithms deserve more study. The combination of residual updates and other semi-gradient algorithms, e.g., TD3 [16], is a possibility for future work. We also do not address the double sampling issue in stochastic environments. This is indeed a restriction, but we would like to emphasize that most available benchmarks with continuous actions have deterministic transitions, which indicates that this class of problems is of practical concern.

5 RESIDUAL ALGORITHMS IN MODEL-BASED RL

In model-based RL, the double sampling issue can be easily addressed by querying the learned model (either deterministic or stochastic). Given the empirical success of deterministic models and their robustness in complex tasks [6, 14, 26], we consider deterministic models in this paper. Dyna [45] is a commonly used model-based RL framework that trains a value function with imaginary transitions from a learned model. In this paper, we consider the combination of Dyna and DDPG. For each planning step, we

Algorithm 1: Dyna-DDPG

```
Input: ;
$Q$ : a critic parameterized by $\theta$ ;
$\mu$ : an actor parameterized by $\nu$ ;
$P$ : planning steps ;
$\epsilon$ : a noise process ;
$f$ : a critic update procedure ;
;
Initialize target networks $\theta^- \leftarrow \theta, \nu^- \leftarrow \nu$ ;
Initialize a replay buffer $\mathcal{B}$, a model $\mathcal{M}$ ;
Get an initial state $S_0$ and set $t \leftarrow 0$ ;
while true do
    $A_t \leftarrow \mu(S_t)$ ;
    Execute $A_t$ and get $R_{t+1}, S_{t+1}$ ;
    Store $(S_t, A_t, R_{t+1}, S_{t+1})$ into $\mathcal{B}$ ;
    Fit $\mathcal{M}$ with data in $\mathcal{B}$ ;
    Sample a batch of transitions from $\mathcal{B}$ ;
    for $(s, a, r, s')$ in batch do
        Update $\theta, \nu$ with $(s, a, r, s')$, (3), (4) ;
        // Planning
        for $i \leftarrow 1, \dots, P$ do
            $\hat{a} \leftarrow a + \epsilon$ ;
            $\hat{r}, \hat{s}' \leftarrow \mathcal{M}(s, \hat{a})$ ;
            Update $\theta$ with $(s, \hat{a}, \hat{r}, \hat{s}')$ and $f$ ;
        end
    end
    $t \leftarrow t + 1$ ;
    Update $\theta^-, \nu^-$ according to $\theta, \nu$
end
```

sample a transition (s, a, r, s') from a replay buffer and add some noise ϵ to the action a, yielding a new action $\hat{a}$. We then query a learned model with $(s, \hat{a})$ and get $(\hat{r}, \hat{s}')$. This imaginary transition is then used to train the Q-function. The pseudocode of this Dyna-DDPG is provided in Algorithm 1. We aim to investigate different strategies for updating Q during planning (i.e., the selection of f in Algorithm 1).

One naive choice is to use the semi-gradient critic update (3). However, this suffers from the *distribution mismatch* problem [14]. When we apply (3) in an imaginary transition $(s, \hat{a}, \hat{r}, \hat{s}')$, we need the Q-value on $\hat{s}'$ for bootstrapping. The Q-function is trained to make an accurate prediction on the state distribution of s, which is usually different from the state distribution of $\hat{s}'$. This distribution mismatch results from both an imperfect model and the different sampling strategies for a and $\hat{a}$. It yields an inaccurate prediction for $Q(\hat{s}', \mu(\hat{s}'))$, leading to poor performance [14]. The TD(k) trick [14] is one attempt to address this issue. With a real transition $(s_{-1}, a_{-1}, r_0, s_0)$ sampled from a replay buffer, a model is unrolled for k steps following $\bar{\mu}$, yielding a trajectory $(s_{-1}, a_{-1}, r_0, s_0, a_0, r_1, s_1, \dots, r_k, s_k)$. TD($k$) then updates θ to minimize

$$\frac{1}{k+1} \sum_{t=-1}^{k-1} \Big(Q(s_t, a_t) - \big(\sum_{i=t+1}^{k} \gamma^{i-t-1} r_i + \gamma^{k-t} \bar{Q}(s_k, \bar{\mu}(s_k)) \big) \Big)^2. \quad (9)$$

With this update, Q is trained on distributions of almost all the states $(s_{-1}, \dots, s_{k-1})$, which Feinberg et al. [14] show helps performance. However, TD(k) still does not train Q on the last imaginary state s_k, which is used for bootstrapping. On the one hand, the influence of the bootstrapping error from s_k decreases as the trajectory gets longer thanks to discounting. On the other hand, even small state prediction errors typically compound as trajectories get longer, yielding a large prediction error of the state s_k itself. This contradiction is deeply embedded in TD(k). Consequently, TD(k) must assume the model is accurate for k-step unrolling, which is usually hard to satisfy in practice.

In this paper, we seek to mitigate this distribution mismatch issue through RA. For an imaginary transition $(s, \hat{a}, \hat{r}, \hat{s}')$, RA naturally allows the Q-function to be trained on both s and $\hat{s}'$, without requiring further unrolling like TD(k). The use of RA in model-based planning is inspired by the theoretical results from Li [28], who proves that TD makes better predictions than RG. On a real transition, this accelerates backward value propagation by providing better bootstrapping. However, on an imaginary transition from a model, the value function is never trained on the imaginary successor state. It is questionable whether we should trust the value prediction on an imaginary state as much as a real state. We, therefore, propose to use RA on imaginary transitions, which encourages the Q-function to be consistent with the model as showed by Li [28].

We now evaluate RA in model-based planning experimentally. We compare the performance of Dyna-DDPG(f = Eq.(3)) (referred to as Dyna-DDPG), Dyna-DDPG(f = Eq.(5)) (referred to as Res-Dyna-DDPG), and DDPG+TD(k) (referred to as MVE-DDPG following, [14]). We consider five Mujoco tasks used by Buckman et al. [6], which is a superset of tasks used by Feinberg et al. [14]. In Feinberg et al. [14], the unrolling steps of MVE-DDPG are different for different tasks, which serve as domain knowledge. For a fair comparison, Buckman et al. [6] set $k = 3$ for all tasks in their baseline MVE-DDPG. In our empirical study, we followed this convention. We use a slightly different TD(k) loss to improve stability of MVE-DDPG, which is explained in detail in the appendix.

To separate planning from model learning, we first consider planning with an oracle model. In this section, we restrict our empirical study on Mujoco tasks as we do not have direct access to the oracle models in DMControl tasks. We tune hyperparameters for Dyna-DDPG and Res-Dyna-DDPG on Walker and set $\eta = 0.2$ for all tasks. Other details are provided in the appendix. The results are reported in Figure 4. Curves are averaged over 8 independent runs and shadowed regions indicate standard errors. Both Dyna-DDPG and MVE-DDPG with an oracle model improve performance in 2 of 5 games, while Res-Dyna-DDPG improves performance in 4 out of 5 games. These results suggest that RA is a more effective approach to exploit a model for planning. In HalfCheetah, both

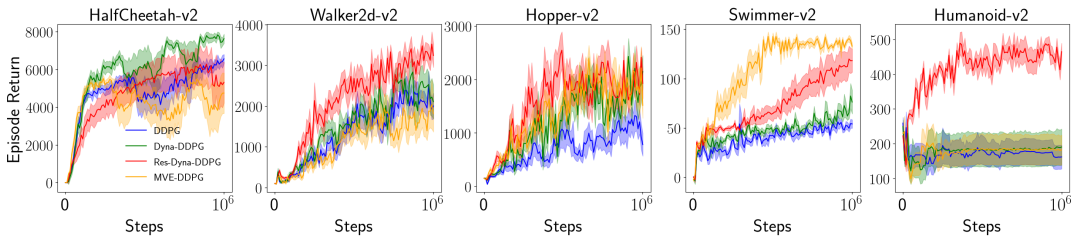

Figure 4: Evaluation performance for different model-based DDPG with an oracle model.

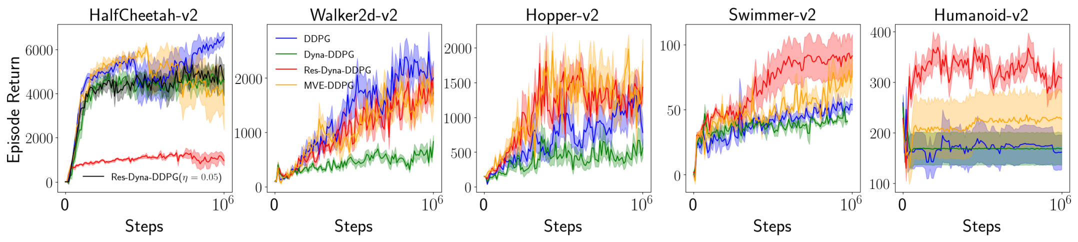

Figure 5: Evaluation performance for different model-based DDPG with a learned model.

MVE-DDPG and Res-Dyna-DDPG fail to outperform Dyna-DDPG. This could suggest that the distribution mismatch problem is not significant in this task. Furthermore, MVE-DDPG exhibits instability in HalfCheetah, which is also observed by Buckman et al. [6].

We now consider planning with a learned model. We use the same model parameterization and model training protocol as Feinberg et al. [14]. We set $\eta = 0.2$ for all tasks. The results are reported in Figure 5. In Swimmer and Humanoid, Res-Dyna-DDPG significantly outperforms all other methods, where Humanoid is usually considered to be the hardest task among all Mujoco tasks. In Walker and Hopper, Res-Dyna-DDPG reaches similar performance as MVE-DDPG. In HalfCheetah, Res-Dyna-DDPG ($\eta = 0.2$) fails dramatically. We further test other values for η and find $\eta = 0.05$ produces reasonable performance, as shown by the extra black curve. This indicates that η can serve as domain knowledge, reflecting our confidence in a learned model. A possibility for future work is to use model uncertainty estimation from a model ensemble to determine η automatically, similar to what Buckman et al. [6] propose for the unrolling steps in TD(k), which significantly improves performance over MVE-DDPG.

In this section, we consider the vanilla residual update (5) without the bidirectional target network. Our preliminary experiments show that introducing the bidirectional target network during planning does not further boost performance. The main purpose of a target network is to stabilize bootstrapping (value propagation). Due to the distribution mismatch problem on imaginary transitions, however, it may be more important for the value function to be consistent with the model than simply propagating the value in either direction. This may reduce the importance of the bidirectional target network.

6 RELATED WORK

There are other studies on Bellman residual methods. Geist et al. [18] show that for policy-based methods, maximizing the average reward is better than minimizing the Bellman residual. Schoknecht and Merke [39] show RG converges with a problem-dependent constant learning rate when combined with certain function approximators. Dabney and Thomas [9] extend RG with natural gradients. However, this paper appears to be the first to contrast residual gradients and semi-gradients in deep RL problems and demonstrate the efficacy of RA with new algorithms. Dai et al. [10] attack the double sampling issue via dual embedding, which translates the minimization of MSBE into a minimax problem. For this translation to hold, the maximization step has to be conducted over a function class which is rich enough to contain the true maximizer. This condition, however, does not necessarily hold when linear function approximation is considered. It can be easily verified that with linear function approximation, dual embedding indeed translates MSBE into MSPBE. Besides MSPBE, other losses have also been proposed to avoid the double sampling issue in minimizing MSBE, for example, Feng et al. [15] propose a kernel loss based on the Bellman equation, Antos et al. [1] add a penalty term to MSBE.

Dyna-style planning in RL has been widely used. Gu et al. [21] learn a local linear model for planning. Kurutach et al. [26] learn a model ensemble to avoid overfitting to an imperfect model, which is also achieved by meta-learning [8]. Kalweit and Boedecker [25] use a value function ensemble to decide when to use a model. Besides Dyna-style planning, learned models are also used for a lookahead tree-search to improve value estimation at decision time [36, 41, 50]. This tree-search is also used as an effective inductive bias in value function parameterization [13, 42, 57]. Trajectories from a learned model are also used as extra inputs for value functions [55], which reduces the negative influence of the model prediction error. In this paper, we focus on the simplest Dyna-style planning and leave the

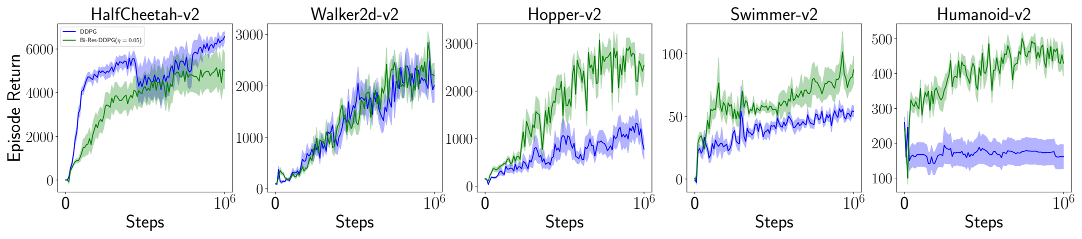

Figure 6: Evaluation curves of DDPG and Bi-Res-DDPG($\eta = 0.05$) on 5 Mujoco tasks. Curves are averaged over 5 independent runs and shaded regions indicate standard errors.

combination of RA and more advanced planning techniques for future work.

Besides RL, learned models are also used in other control methods, e.g., model predictive control (MPC, [17]). Nagabandi et al. [35] learn deterministic models via neural networks for MPC. Chua et al. [7] conduct a thorough comparison between deterministic models and stochastic models and use particle filters when unrolling a model. Besides modeling the observation transition, Ha and Schmidhuber [22] and Hafner et al. [23] propose to model the abstract state transition and use MPC on the abstract state space. In this paper, we focus on the simplest deterministic model and leave the combination of RA and more advanced models for future work.

7 CONCLUSIONS

In this paper, we give a thorough review of existing comparisons between RG and TD. We propose the bidirectional target network technique to stabilize bootstrapping in both directions in RA, yielding a significant performance boost. We also demonstrate that RA is a more effective approach to the distribution mismatch problem in model-based planning than the existing TD(k) method. Our empirical study showed the efficacy of RA in deep RL problems, which has long been underestimated by the community. A possibility for future work is to study RA in model-free RL with stochastic environments, where the double sampling issue cannot be trivially resolved.

ACKNOWLEDGMENTS

SZ is generously funded by the Engineering and Physical Sciences Research Council (EPSRC). This project has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement number 637713). The experiments were made possible by a generous equipment grant from NVIDIA.

A EXPERIMENT DETAILS

All our implementations and the corresponding Docker environment are made publicly available.[1] Open AI Gym and DMControl are available at https://gym.openai.com/ and https://github.com/deepmind/dm_control.

Our DDPG implementation uses the same parameterization and hyperparameters as Lillicrap et al. [29], which are inherited by all the variants of DDPG in this paper without further turning. We do not use batch normalization.

For the model-based experiments, we tune extra hyperparameters in Walker with an oracle model for both Dyna-DDPG and Res-Dyna-DDPG. The planning steps P is tuned over $\{1, 2, 4\}$. The noise process ϵ is Gaussian noise $\mathcal{N}(0, \sigma^2)$, with σ tuned over $\{0.05, 0.1, 0.2\}$. The mix coefficient η in RA is tuned over $\{0, 0.05, 0.1, 0.2, 0.4, 0.8, 1\}$. In all our experiments (with both an oracle model and a learned model), we set $P = 1, \sigma = 0.1, \eta = 0.2$.

For MVE-DDPG, we find the original TD(k) loss (9) yields significant instability. To improve stability, we made two modifications. First, for a trajectory $(s_{-1}, a_{-1}, r_0, s_0, a_0, r_1, s_1, \dots, r_k, s_k)$, instead of minimizing the loss (9), we minimize

$$\Big(Q(s_{-1}, a_{-1}) - \big(r_0 + \gamma \bar{Q}(s_0, a_0)\big)\Big)^2 + \frac{1}{k}\sum_{t=0}^{k-1}\Big(Q(s_t, a_t) - \big(\sum_{i=t+1}^{k} \gamma^{i-t-1} r_i + \gamma^{k-t}\bar{Q}(s_k, \bar{\mu}(s_k))\big)\Big)^2.$$

This new loss is different from (9) mainly in that it uses the real transition $(s_{-1}, a_{-1}, r_0, s_0)$ more. We find this significantly improves stability. Second, we replace the mean squared loss with a Huber loss [24], which has been reported to improve stability [12]. Our MVE-DDPG implementation significantly outperforms the MVE-DDPG baselines in Buckman et al. [6] in Hopper and Walker while maintains a similar performance in remaining tasks. The MVE-DDPG in Feinberg et al. [14] has task-dependent learning rates. By contrast, we do not tune hyperparameters task by task for any compared algorithm.

We conducted our experiments on an Nvidia DGX-1 with PyTorch.

B OTHER EXPERIMENTAL RESULTS

The evaluation curves of DDPG and Bi-Res-DDPG($\eta = 0.05$) on 5 Mujoco tasks and 28 DMControl tasks are reported in Figure 6 and Figure 7 respectively.

[1]https://github.com/ShangtongZhang/DeepRL

REFERENCES

[1] András Antos, Csaba Szepesvári, and Rémi Munos. 2008. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning* (2008).
[2] Leemon Baird. 1995. Residual algorithms: Reinforcement learning with function approximation. *Machine Learning* (1995).
[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* (2013).
[4] Richard E. Bellman. 1957. *Dynamic programming.* Princeton University Press.

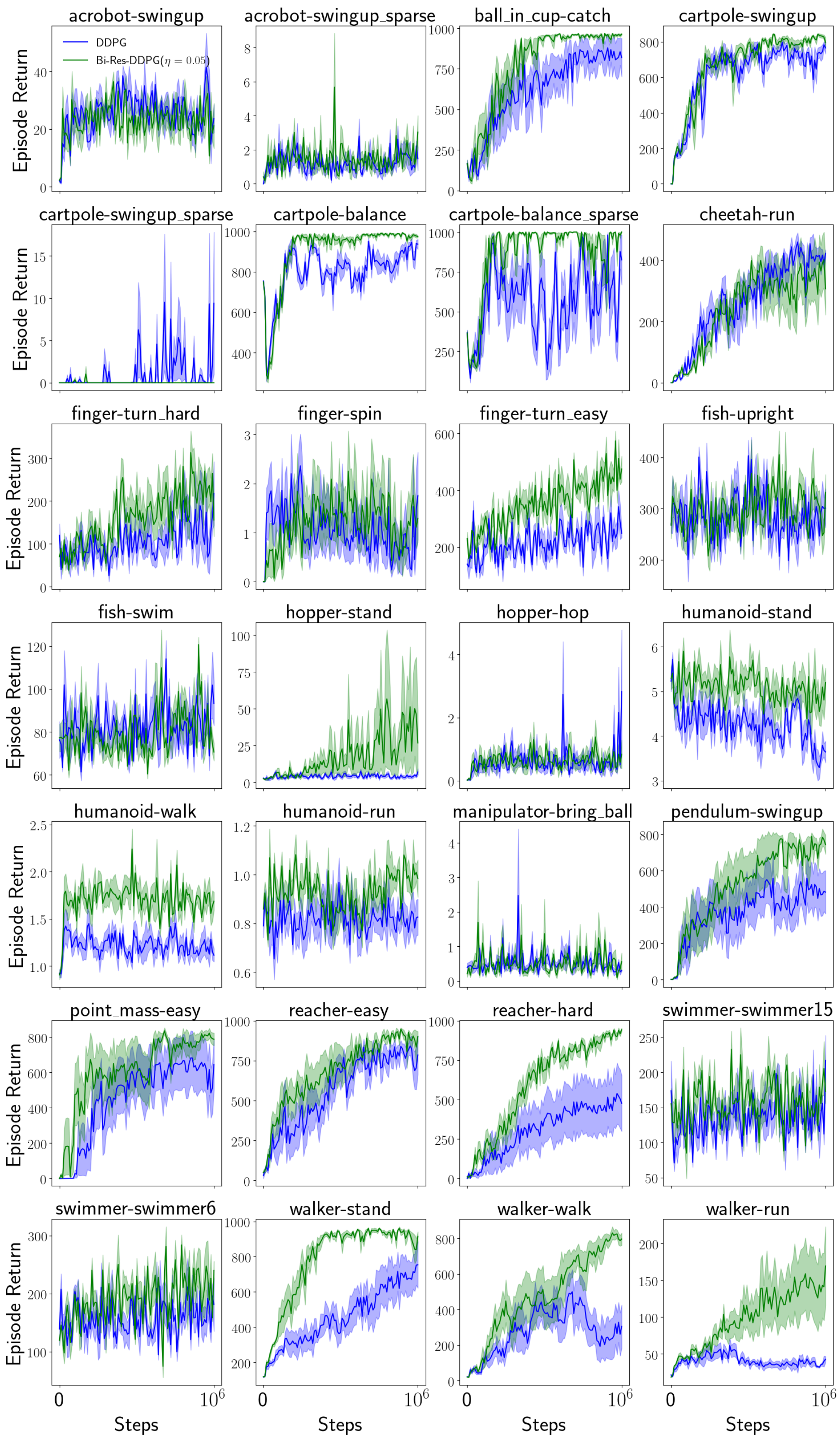

Figure 7: Evaluation curves of DDPG and Bi-Res-DDPG(η = 0.05) on 28 DMControl tasks. Curves are averaged over 5 independent runs and shaded regions indicate standard errors.

[5] Dimitri P Bertsekas and John N Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific Belmont, MA.

[6] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. 2018. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*.

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*.

[8] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. 2018. Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809.05214* (2018).

[9] William Dabney and Philip Thomas. 2014. Natural temporal difference learning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*.

[10] Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. 2017. SBEED: Convergent reinforcement learning with nonlinear function approximation. *arXiv preprint arXiv:1712.10285* (2017).

[11] Christoph Dann, Gerhard Neumann, and Jan Peters. 2014. Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research* (2014).

[12] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. OpenAI Baselines. https://github.com/openai/baselines. (2017).

[13] Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. 2018. TreeQN and ATreeC: Differentiable tree-structured models for deep reinforcement learning. *arXiv preprint arXiv:1710.11417* (2018).

[14] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. 2018. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101* (2018).

[15] Yihao Feng, Lihong Li, and Qiang Liu. 2019. A Kernel Loss for Solving the Bellman Equation. *arXiv preprint arXiv:1905.10506* (2019).

[16] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* (2018).

[17] Carlos E Garcia, David M Prett, and Manfred Morari. 1989. Model predictive control: theory and practice—a survey. *Automatica* (1989).

[18] Matthieu Geist, Bilal Piot, and Olivier Pietquin. 2017. Is the Bellman residual a bad proxy?. In *Advances in Neural Information Processing Systems*.

[19] Geoffrey J Gordon. 1995. Stable function approximation in dynamic programming. *Machine Learning* (1995).

[20] Geoffrey J Gordon. 1999. *Approximate solutions to Markov decision processes*. Ph.D. Dissertation. Carnegie Mellon University.

[21] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. 2016. Continuous deep q-learning with model-based acceleration. In *Proceedings of the 33rd International Conference on Machine Learning*.

[22] David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122* (2018).

[23] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2018. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551* (2018).

[24] Peter J Huber et al. 1964. Robust estimation of a location parameter. *The Annals of Mathematical Statistics* (1964).

[25] Gabriel Kalweit and Joschka Boedecker. 2017. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Proceedings of the 2017 Conference on Robot Learning*.

[26] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. 2018. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592* (2018).

[27] Michail G Lagoudakis and Ronald Parr. 2003. Least-squares policy iteration. *Journal of Machine Learning Research* (2003).

[28] Lihong Li. 2008. A worst-case comparison between temporal difference and residual gradient with linear function approximation. In *Proceedings of the 25th International Conference on Machine Learning*.

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

[30] Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* (1992).

[31] Hamid Reza Maei. 2011. *Gradient temporal-difference learning algorithms*. Ph.D. Dissertation. University of Alberta.

[32] Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S Sutton. 2010. Toward off-policy learning control with function approximation.. In *Proceedings of the 27th International Conference on Machine Learning*.

[33] 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. 2015. Human-level control through deep reinforcement learning. *Nature* (2015).

[34] Rémi Munos. 2003. Error bounds for approximate policy iteration. In *Proceedings of the 20th International Conference on Machine Learning*.

[35] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *Proceedings of the 2018 International Conference on Robotics and Automation*.

[36] Junhyuk Oh, Satinder Singh, and Honglak Lee. 2017. Value prediction network. In *Advances in Neural Information Processing Systems*.

[37] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[38] Bruno Scherrer. 2010. Should one compute the temporal difference fix point or minimize the bellman residual? the unified oblique projection view, In Proceedings of the 27nd International Conference on Machine Learning. *arXiv preprint arXiv:1011.4362*.

[39] Ralf Schoknecht and Artur Merke. 2003. Convergent combinations of reinforcement learning with linear function approximation. In *Advances in Neural Information Pprocessing Systems*.

[40] Ralf Schoknecht and Artur Merke. 2003. TD (0) converges provably faster than the residual gradient algorithm. In *Proceedings of the 20th International Conference on Machine Learning*.

[41] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. 2017. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning*.

[42] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. 2018. Universal planning networks. *arXiv preprint arXiv:1804.00645* (2018).

[43] Wen Sun and J Andrew Bagnell. 2015. Online Bellman Residual algorithms with predictive error guarantees. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*.

[44] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* (1988).

[45] Richard S Sutton. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on Machine Learning*.

[46] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction (2nd Edition)*. MIT press.

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

[48] Richard S Sutton, A Rupam Mahmood, and Martha White. 2016. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research* (2016).

[49] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* (1999).

[50] Erik Talvitie. 2017. Self-correcting models for model-based reinforcement learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

[51] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. 2018. DeepMind control suite. *arXiv preprint arXiv:1801.00690* (2018).

[52] John N Tsitsiklis and Benjamin Van Roy. 1997. Analysis of temporal-diffference learning with function approximation. In *Advances in Neural Information Pprocessing Systems*.

[53] Hado Philip van Hasselt. 2011. *Insights in reinforcement learning*. Ph.D. Dissertation. Utrecht University.

[54] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* (1992).

[55] Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. 2017. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203* (2017).

[56] Ronald J Williams and Leemon C Baird. 1993. *Tight performance bounds on greedy policies based on imperfect value functions*. Technical Report. Citeseer.

[57] Shangtong Zhang, Hao Chen, and Hengshuai Yao. 2019. ACE: An Actor Ensemble Algorithm for Continuous Control with Tree Search. *Proceedings of the 33rd AAAI Conference on Artificial Intelligence* (2019).