RUDDER: Return Decomposition for Delayed Rewards

Jose A. Arjona-Medina* **Michael Gillhofer*** **Michael Widrich***
Thomas Unterthiner **Sepp Hochreiter**
LIT AI Lab
Institute for Machine Learning
Johannes Kepler University Linz, Austria

Abstract

We propose a novel reinforcement learning approach for finite Markov decision processes (MDPs) with delayed rewards. In this work, biases of temporal difference (TD) estimates are proved to be corrected only exponentially slowly in the number of delay steps. Furthermore, variances of Monte Carlo (MC) estimates are proved to increase the variance of other estimates, which number can exponentially grow in the number of delay steps. We introduce RUDDER, a return decomposition method, which creates a new MDP with same optimal policies as the original MDP but with redistributed rewards that have largely reduced delays. If the return decomposition is optimal, then the new MDP does not have delayed rewards and TD estimates are unbiased. In this case, the rewards track Q-values so that the future expected reward is always zero. We experimentally confirm our theoretical results on bias and variance of TD and MC estimates. On artificial tasks with different lengths of reward delays, we show that RUDDER is exponentially faster than TD, MC, and MC Tree Search (MCTS). RUDDER outperforms rainbow, A3C, DDQN, Distributional DQN, Dueling DDQN, Noisy DQN, and Prioritized DDQN on the delayed reward Atari game Venture in only a fraction of the learning time. RUDDER considerably improves the state-of-the-art on the delayed reward Atari game Bowling in much less learning time.

1 Introduction

Assigning the credit for a received reward to actions that were performed, is one of the central tasks in reinforcement learning [58]. Long term credit assignment has been identified as one of the largest challenges in reinforcement learning [46]. Current reinforcement learning methods are still slowed down significantly when facing long-delayed rewards [41, 30]. To learn delayed rewards there are three phases to consider: (1) discovering the delayed reward, (2) keeping information about the delayed reward, (3) learning to receive the delayed reward to secure it for the future. Recent successful reinforcement methods provide solutions to one or more of these items. Most prominent are Deep Q-Networks (DQNs) [32, 33], which combine Q-learning with convolutional neural networks for visual reinforcement learning [24]. The success of DQNs is attributed to *experience replay* [29], which stores observed state-reward transitions and then samples from them. Prioritized experience replay [47, 22] advanced the sampling from the reply memory. Different policies perform exploration in parallel for the Ape-X DQN and share a prioritized experience replay memory [22]. IMPALA improves A2C by parallel actors and corrects for policy-lags between actors and learners [10]. DQN was extended to double DQN (DDQN) [60, 61] which helps exploration as the overestimation bias is reduced. Noisy DQNs [11] explore by a stochastic layer in the policy network (see [18, 48]).

*authors contributed equally

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

Distributional Q-learning [6] profits from noise since means that have high variance are more likely selected. The dueling network architecture [62, 63] separately estimates state values and action advantages, which helps exploration in unexperienced states. Policy gradient approaches [66] like A3C with asynchronous gradient descent [31] or Ape-X DPG [22] explore via parallel policies, too. Proximal policy optimization (PPO) extends A3C by a surrogate objective and a trust region optimization realized by clipping or a Kullback-Leibler distance penalty [50].

Recent approaches aim to solve learning problems caused by delayed rewards. Function approximations of value functions or critics [33, 31] bridge time intervals if states associated with rewards are similar to states that were encountered many steps earlier. For example, assume a function that learned to predict a large reward at the end of an episode if a state has a particular feature. The function can generalize this correlation to the begin of an episode and predict already high reward for states possessing the same feature. Multi-step temporal difference (TD) learning [56, 58] improved both DQNs and policy gradients [17, 31]. AlphaGo and AlphaZero learned to play Go and Chess better than human professionals using Monte Carlo Tree Search (MCTS) [51, 52]. MCTS simulates games from a time point until the end of the game or an evaluation point, therefore captures long-delayed rewards. Recently, world models using a evolution strategy were successful [14]. These forward view approaches using world models are not feasible in probabilistic environments with a high state transition branching factor. Backward view approaches trace back from known goal states [9] or from high-reward states [13]. However a step-by-step backward model has to be learned.

We propose learning from a backward view, which is constructed from a forward model. The forward model predicts the return, while the backward analysis identifies states and actions which have caused the return. We apply Long Short-Term Memory (LSTM) [19, 21] to predict the return of an episode. LSTM was already used in reinforcement learning [49] for advantage learning [3] and learning policies [15, 31, 16]. However sensitivity analysis by "backpropagation through a model" [35, 44, 45, 4] has major drawbacks: local minima, instabilities, exploding or vanishing gradients in the world model, proper exploration, contribution (relevance) of actions are not regarded only their sensitivity [18, 48].

Since sensitivity analysis substantially hinders learning, we use contribution analysis for backward analysis like contribution-propagation [25], contribution approach [38], excitation backprop [68], layer-wise relevance propagation (LRP) [2], Taylor decomposition [2, 34], or integrated gradients (IG) [55]. Using contribution analysis, a predicted return can be decomposed into contributions along the state-action sequence. Substituting the prediction by the actual return, we obtain a redistributed reward leading to new MDP with the same optimal policies as for the original MDP. Redistributing the reward is fundamentally different from reward shaping [36, 64], which changes the reward as a function of states but not of actions. Reward redistribution is related to "look-back advice" [65] which, in contrast to reward redistribution, still requires the original MDP for learning. We propose RUDDER, which performs reward redistribution by return decomposition and, therefore, overcomes problems of TD and MC stemming from delayed rewards. RUDDER vastly decreases the variance of MC and largely avoids the exponentially slow bias corrections of TD — for optimal return decomposition TD is even unbiased.

2 Bias-Variance for MDP Estimates

We perform a bias-variance analysis for temporal difference (TD) and Monte Carlo (MC) estimators of the action-value function. A finite Markov decision process (MDP) $\mathcal{P}$ is 6-tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \pi, \gamma)$ of finite sets $\mathcal{S}$ of states s (random variable S_t at time t), $\mathcal{A}$ of actions a (random variable A_t), and $\mathcal{R}$ of rewards r (random variable R_{t+1}). Furthermore, $\mathcal{P}$ has transition-reward distributions $p(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$ conditioned on state-actions, a policy given as an action distributions $\pi(A_{t+1} = a' \mid S_{t+1} = s')$ conditioned on states, and a discount factor $\gamma \in [0, 1]$. The marginals are $p(r \mid s, a) = \sum_{s'} p(s', r \mid s, a)$ and $p(s' \mid s, a) = \sum_r p(s', r \mid s, a)$. The expected reward is $r(s, a) = \sum_r r p(r \mid s, a)$. The return G_t is $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. We often consider finite horizon MDPs with sequence length T and $\gamma = 1$ giving $G_t = \sum_{k=0}^{T-t} R_{t+k+1}$. The action-value function $q^\pi(s, a)$ for policy π is $q^\pi(s, a) = \mathrm{E}_\pi [G_t \mid S_t = s, A_t = a]$. Goal of learning is to maximize the expected return at time $t = 0$, that is $v_0^\pi = \mathrm{E}_\pi [G_0]$.

Bias-Variance Analysis for MDP Estimates. MC estimates $q^\pi(s,a)$ by an arithmetic mean of the return, while TD methods like SARSA or Q-learning estimate $q^\pi(s,a)$ by an exponential average of the return. When using Monte Carlo for learning a policy we use an exponential average, too, since the policy steadily changes. The ith update of action-value q at state-action (s_t, a_t) is $(q^\pi)^{i+1}(s_t,a_t) = (q^\pi)^i(s_t,a_t) + \alpha \left(\sum_t^T r_{t+1} - (q^\pi)^i(s_t,a_t)\right)$. Assume n samples $\{X_1,\ldots,X_n\}$ from a distribution with mean μ and variance σ^2. For these samples, we compute bias and variance of the arithmetic mean $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^n X_i$ and the exponential average $\tilde{\mu}_n = \alpha \sum_{i=1}^n (1-\alpha)^{n-i} X_i + (1-\alpha)^n \mu_0$ with μ_0 as initial value and $\alpha \in (0,1)$. We obtain $\mathbf{bias}(\hat{\mu}_n) = 0$ and $\mathbf{var}(\hat{\mu}_n) = \sigma^2/n$ as well as $\mathbf{bias}(\tilde{\mu}_n) = (1-\alpha)^n(\mu_0 - \mu)$ and $\mathbf{var}(\tilde{\mu}_n) = \sigma^2\left(\alpha(1-(1-\alpha)^{2n})\right)/(2-\alpha)$ (see Appendix A1.2.1 for more details). Both variances are proportional to σ^2, which is the variance when sampling a return from the MDP $\mathcal{P}$.

Using $\mathrm{E}_{s',a'}(f(s',a')) = \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s') f(s',a')$, and analog $\mathrm{Var}_{s',a'}$ and Var_r, the next theorem gives mean and variance $V^\pi(s,a) = \mathrm{Var}_\pi[G_t \mid s,a]$ of sampling returns from an MDP.

Theorem 1. *The mean q^π and variance V^π of sampled returns from an MDP are*

$$q^\pi(s,a) = \sum_{s',r} p(s',r \mid s,a)\left(r + \gamma \sum_{a'} \pi(a' \mid s') q^\pi(s',a')\right) = r(s,a) + \gamma \mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right],$$

$$V^\pi(s,a) = \mathrm{Var}_r\left[r \mid s,a\right] + \gamma^2\left(\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] + \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right). \tag{1}$$

The proof is given after Theorem A1 in the appendix. The theorem extends the deterministic reward case [54, 59]. The variance $V^\pi(s,a)$ consists of three parts: (i) The immediate variance $\mathrm{Var}_r[r \mid s,a]$ stemming from the probabilistic reward $p(r \mid s,a)$. (ii) The local variance $\gamma^2 \mathrm{Var}_{s',a'}[q^\pi(s',a') \mid s,a]$ caused by probabilistic state transitions and probabilistic policy. (iii) The expected variance $\gamma^2 \mathrm{E}_{s',a'}[V^\pi(s',a') \mid s,a]$ of the next Q-values, which is zero for TD since it replaces $q^\pi(s',a')$ by fixed $\hat{q}^\pi(s',a')$. Therefore TD has less variance than MC which uses the complete future return. See Appendix A1.2.2 for more details.

Delayed Reward Aggravates Learning. The ith temporal difference update with learning rate α of the action-value $q(s_t,a_t)$ is

$$q^{i+1}(s_t,a_t) \;=\; q^i(s_t,a_t) \;+\; \alpha\;\left(r_{t+1} \;+\; \mathcal{A}_{a'}\left(q^i(s_{t+1},a')\right) \;-\; q^i(s_t,a_t)\right)\;, \tag{2}$$

with $\mathcal{A}_{a'}(.) = \max_{a'}(.)$ (Q-learning), $\mathcal{A}_{a'}(.) = \sum_{a'} \pi(a' \mid s_{t+1})(.)$ (expected SARSA), $\mathcal{A}_{a'}(.)$ sample a' from $\pi(a' \mid s_{t+1})$ (SARSA). The next theorem states that TD has an exponential decay for Q-value updates even for eligibility traces [23, 5, 57, 53].

Theorem 2. *For initialization $q^0(s_t,a_t) = 0$ and delayed reward with $r_t = 0$ for $t \leqslant T$, $q(s_{T-i},a_{T-i})$ receives its first update not earlier than at episode i via $q^i(s_{T-i},a_{T-i}) = \alpha^{i+1} r_{T+1}^1$, where r_{T+1}^1 is the reward of episode 1. Eligibility traces with $\lambda \in [0,1)$ lead to an exponential decay of $(\gamma\lambda)^k$ when the reward is propagated k steps back.*

The proof is given after Theorem A2 in the appendix. To correct the TD bias by a certain amount requires exponentially many updates with the number of delay steps.

For Monte Carlo the variance of a single delayed reward can increase the variance of action-values of all previously visited state-actions. We define the "on-site" variance ω

$$\omega(s,a) \;=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\;, \tag{3}$$

$\boldsymbol{V}^\pi$ is the vector with value $V^\pi(s',a')$ at position (s',a') and $\boldsymbol{P}_t$ the transition matrix from states s_t to s_{t+1} with entries $p(s_{t+1} \mid s_t,a_t)\pi(a_{t+1} \mid s_{t+1})$ at position $((s_t,a_t),(s_{t+1},a_{t+1}))$. For finite time horizon, the "backward induction algorithm" [39, 40] gives with $\boldsymbol{V}_{T+1}^\pi = \boldsymbol{0}$ and $\boldsymbol{\omega}_{T+1} = \boldsymbol{0}$ and row-stochastic matrix, $\boldsymbol{P}_{t\leftarrow k} = \prod_{\tau=t}^{k-1} \boldsymbol{P}_\tau$:

$$\boldsymbol{V}_t^\pi \;=\; \sum_{k=t}^{T} \prod_{\tau=t}^{k-1} \boldsymbol{P}_\tau\, \boldsymbol{\omega}_k \;=\; \sum_{k=t}^{T} \boldsymbol{P}_{t\leftarrow k}\, \boldsymbol{\omega}_k\;, \tag{4}$$

where we define $\prod_{\tau=t}^{t-1} \boldsymbol{P}_\tau = \boldsymbol{I}$ and $[\boldsymbol{\omega}_k]_{(s_k,a_k)} = \omega(s_k,a_k)$. We are interested in the number of action-values which variances are affected through the increase of the variance of a single delayed reward. Let N_t be the number of all states s_t that are reachable after t time steps of an episode. Let c_t be the random average connectivity of a state in s_t to states in s_{t-1}. Let n_t be number of states in s_t that are affected by $\boldsymbol{\omega}_k$ for $t \leqslant k$ with $n_k = 1$ (only one action-value with delayed reward at time $t = k$). Next theorem says that the on-site variance $\boldsymbol{\omega}_k$ can have large effects on the variance of action-values of all previously visited state-actions, which number can grow exponentially.

Theorem 3. *For $t \leqslant k$, on-site variance $\boldsymbol{\omega}_k$ at step k contributes to $\boldsymbol{V}_t^\pi$ by the term $\boldsymbol{P}_{t \leftarrow k} \boldsymbol{\omega}_k$, where $\|\boldsymbol{P}_{t \leftarrow k}\|_\infty = 1$. The number a_k of states affected by $\boldsymbol{\omega}_k$ is $a_k = \sum_{t=0}^{k} \left(1 - \left(1 - \frac{c_t}{N_{t-1}}\right)^{n_t}\right) N_{t-1}$.*

The proof can be found after Theorem A3. For small k, the number a_k of states affected by on-site variance $\boldsymbol{\omega}_k$ at step k growths exponentially with k. For large k and after some time $t > \hat{t}$, the number a_k of states affected by $\boldsymbol{\omega}_k$ growths linearly. (See Corollary A1 in the appendix). Consequently, we aim for decreasing the on-site variance $\boldsymbol{\omega}_k$ for large k, in order to reduce the variance. In summary, delayed rewards lead to exponentially slow corrections of biases of temporal difference (TD) and can increase exponentially many variances of Monte Carlo (MC) action-value estimates, where the exponentially grows is in both cases in the number of delay steps.

3 Return Decomposition and Reward Redistribution

A Markov decision process (MDP) $\tilde{\mathcal{P}}$ is *state-enriched* compared to a MDP $\mathcal{P}$ if $\tilde{\mathcal{P}}$ has the same states, actions, transition probabilities, and reward probabilities as $\mathcal{P}$ but with additional information in their states. We observe that $\mathcal{P}$ is a homomorphic image of $\tilde{\mathcal{P}}$ with the same actions. Therefore each optimal policy $\tilde{\pi}^*$ of $\tilde{\mathcal{P}}$ has an equivalent optimal policy π^* of $\mathcal{P}$, and vice versa, with the same optimal return [42, 43]. These properties are known from state abstraction and aggregation [28] and from bisimulation [12]. For more details see Appendix A1.3.1. Two Markov decision processes $\tilde{\mathcal{P}}$ and $\mathcal{P}$ are *return-equivalent* if they differ only in $p(\tilde{r} \mid s, a)$ and $p(r \mid s, a)$ but for each policy π they have the same expected return at $t = 0$: $\tilde{v}_0^\pi = v_0^\pi$. Return-equivalent decision processes have the same optimal policies.

We assume to have an MDP $\mathcal{P}$ with immediate reward which is transformed to a state-enriched MDP $\tilde{\mathcal{P}}$ with delayed reward, where the return is given as reward at sequence end. The transformed delayed state-enriched MDP has reward $\tilde{r}_t = 0$, $t \leqslant T$, and $\tilde{r}_{T+1} = \sum_{k=0}^{T} R_{k+1}$. The states are enriched by ρ which records the accumulated already received rewards, therefore $\tilde{s}_t = (s_t, \rho_t)$, where $\rho_t = \sum_{k=0}^{t-1} r_{k+1}$. We show in Proposition **??** that $\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) = q^\pi(s, a) + \sum_{k=0}^{t-1} r_{k+1}$ for $\tilde{\pi}(a \mid \tilde{s}) = \pi(a \mid s)$. Thus, each immediate reward MDP can be transformed into a delayed reward MDP without changing the optimal policies.

Next we consider the opposite direction, where the delayed reward MDP $\tilde{\mathcal{P}}$ is given and we want to find an immediate reward MDP $\mathcal{P}$. $\mathcal{P}$ should be return-equivalent to $\tilde{\mathcal{P}}$ and differ from $\tilde{\mathcal{P}}$ only by its reward distributions. We have to redistribute the final reward, which is the return, $\tilde{r}_{T+1}$ to previous time steps, therefore we have to decompose the return into a sum of rewards at different time steps. To allow for a return decomposition, we predict the return $\tilde{r}_{T+1}$ by a function g using the state-action sequence: $g\big((s, a)_{0:T}\big) = \tilde{r}_{T+1}$, where $(s, a)_{0:T}$ is the state-action sequence from $t = 0$ to $t = T$. In a next step we decompose g into a sum: $g((s, a)_{0:T}) = \sum_{t=0}^{T} h(a_t, s_t)$, where h is the prediction contribution. Since $\tilde{\mathcal{P}}$ is an MDP, the reward can be predicted from (a_T, s_T) since s_T contains information about the already accumulated reward. Therefore we use a difference $\Delta(s, s')$ between state s and its successor s' instead of s to avoid the Markov property in the input sequence. The difference Δ is assumed to make $(a_t, \Delta(s_t, s_{t+1}))$ statistically independent from each other in the sequence $(a, \Delta)_{0:T} = \big(a_0, \Delta(s_0, s_1), \ldots, a_t, \Delta(s_t, s_{t+1})\big)$. The function g is decomposed by contribution analysis into a sum of h by $g\big((a, \Delta)_{0:T}\big) = \tilde{r}_{T+1} = \sum_{t=0}^{T} h(a_t, \Delta(s_t, s_{t+1}))$. The actual reward redistribution is $r_{t+1} = \tilde{r}_{T+1} h(a_t, \Delta(s_t, s_{t+1})) / g\big((a, \Delta)_{0:T}\big)$ to ensure $\sum_{t=0}^{T} \tilde{r}_{t+1} = \tilde{r}_{T+1} = \sum_{t=0}^{T} r_{t+1}$.

If for partial sums $\sum_{\tau=0}^{t} h(a_\tau, \Delta(s_\tau, s_{\tau+1})) = \tilde{q}^\pi(s_t, a_t)$ holds, then the *return decomposition is optimal*. We have for $g\big((a, \Delta)_{0:T}\big) = \tilde{r}_{T+1}$ rewards $R_0 = h_0 = h(a_0, \Delta(s_0, s_1)) = \tilde{q}^\pi(s_0, a_0)$ and $R_t = h_t = h(a_t, \Delta(s_t, s_{t+1})) = \tilde{q}^\pi(s_t, a_t) - \tilde{q}^\pi(s_{t-1}, a_{t-1})$. The term $\tilde{q}^\pi(s_{t-1}, a_{t-1})$ introduces variance in R_t.

Theorem 4. *The MDP $\mathcal{P}$ based on the redistributed reward given by an optimal return decomposition (I) has the same optimal policies as $\tilde{\mathcal{P}}$ of the delayed reward, and (II) the Q-values are given by* $q^{\pi}(s_t, a_t) = r(s_t, a_t) = \tilde{q}^{\pi}(s_t, a_t) - \mathrm{E}\left[\tilde{q}^{\pi}(s_{t-1}, a_{t-1}) \mid s_t, a_t\right]$.

The proof can be found after Theorem 4 in the appendix. In particular, when starting with zero initialized Q-values, then TD learning of $\mathcal{P}$ is not biased at the beginning. For policy gradients with eligibility traces using $\lambda \in [0, 1]$ for G_t^{λ} [58], we have the expected updates $\mathrm{E}_{\pi}\left[\nabla_{\theta} \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \sum_{\tau=0}^{T-t} \lambda^{\tau} q^{\pi}(s_{t+\tau}, a_{t+\tau})\right] = \mathrm{E}_{\pi}\left[\nabla_{\theta} \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \sum_{\tau=0}^{T-t} \lambda^{\tau} r(s_{t+\tau}, a_{t+\tau})\right]$, where $r(s_t, a_t)$ is replaced during learning by a sample from R_t which is the redistributed reward for an episode.

RUDDER: Return Decomposition using LSTM. We introduce RUDDER "RetUrn Decomposition for DElayed Rewards", which performs return decomposition using a Long Short-Term Memory (LSTM) network for redistributing the original reward. RUDDER consists of (I) a safe exploration strategy, (II) a lessons replay buffer, and, most importantly, (III) an LSTM with contribution analysis for return decomposition. **(I) Safe exploration.** Exploration strategies should assure that LSTM receives training data with delayed rewards. Toward this end we introduce a new exploration strategy which initiates at a certain time in the episode an exploration sequence to discover delayed rewards. To avoid an early stop of the exploration sequence, we perform a safe exploration which avoids actions associated with low Q-values. Low Q-values hint at states with zero future reward where the agent gets stuck. Exploration parameters are starting time, length, and the action selection strategy with safety constraints. **(II) Lessons replay buffer.** If safe exploration discovers an episode with unexpected delayed reward, it is secured in a lessons replay buffer [29]. Episodic memory has been used for episodic control [27] and for episodic backward update to efficiently propagate delayed rewards [26]. Unexpected is indicated by a large prediction error of LSTM. Sampling from lessons replay buffer is done similar to prioritized experience replay. Episodes with larger error are more often sampled. **(III) LSTM and contribution analysis.** LSTM networks [19, 21]), are used to predict the return from a input sequence. LSTM solves the vanishing gradient problem [19, 20], which severely impedes credit assignment in recurrent neural networks, i.e. the correct identification of relevant but delayed input events. LSTM backward analysis is done through contribution analysis like layer-wise relevance propagation (LRP) [2], Taylor decomposition [2, 34], or integrated gradients (IG) [55]. These methods identify the contributions of the inputs to the final prediction, therefore supply the return decomposition.

The LSTM return decomposition is optimal if LSTM predicts at every time step the expected final return. To push LSTM toward optimal return decomposition, we introduce continuous return predictions as auxiliary tasks, where the LSTM has to predict the final return during the sequence. Hyperparameters are when and how often LSTM predicts and how continuous prediction errors are weighted. Strictly monotonic LSTM architecture (see AppendixA4.3.1) can also ensure that LSTM decomposition is optimal.

4 Experiments

We use $\gamma = 1$ for delayed rewards in MDPs with finite time horizon or absorbing states which has been confirmed to be suited to long delays by meta-gradient reinforcement learning [67].

Grid World: RUDDER is tested on a grid world with delayed reward at the end of an episode. The MDP is a 7×7 grid with 3 special locations (*start*, *key* and *door*), and 4 actions (*up*, *down*, *left* and *right*). An episode begins with a random *start* and ends at the fixed *door* or after 25 time steps. *key* defines the minimal delay. If the agent visits *door* at time t, then the reward is $-t \cdot 0.1$ and increased by 10 if the agent has visited *key*, otherwise the reward is 0. To investigate how the delay affects bias and variance, Q-values are estimated by TD and MC for a random policy which assures that all states are visited. After computing the true Q-values by backward induction, we compare the bias, variance and mean squared error (MSE) of the estimators for the MDP with delayed reward and the new MDP obtained by RUDDER with optimal reward redistribution. Figure 1 shows that RUDDER has smaller number of Q-values with high variance than the original MDP, when learning MC estimators. It also shows that RUDDER corrects the bias faster than TD estimators in the original MDP. So far we kept the policy constant and focused on learning the action-value function. Next, we compare Q-learning, Monte Carlo (MC), and Monte Carlo Tree Search (MCTS) at learning a policy for the grid world. Figure 2 shows the number of episodes required by different methods to learn a policy that achieves 90% of the return of the optimal policy for different delays. Optimal reward redistribution speeds up learning a policy exponentially. More information is available in Appendix A5.1.1.

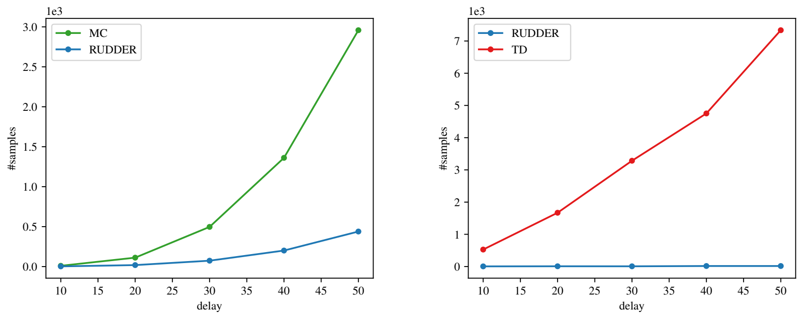

Figure 1: Experimental evaluation of MSE, bias, and variance of different Q-value estimators on the Grid World. Left: Variance of the MC estimator grows exponentially with the delay. Shown are the number of samples needed to go below a threshold. Right: Bias correction in TD is exponentially small with the delay. Shown are the number of samples needed to half the initial error for the initial state estimator in TD.

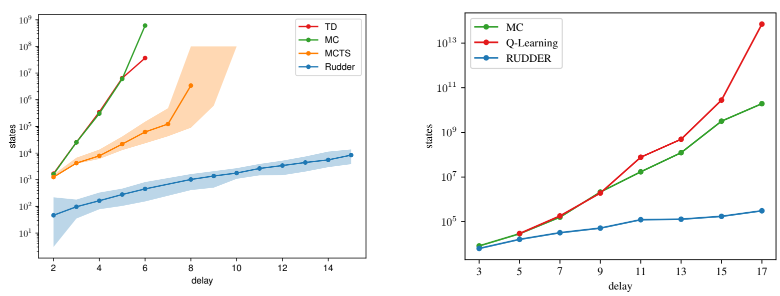

Figure 2: Number of observed states required by different methods to learn a policy that achieves 90% of the return of the optimal policy for different delays. We compare Q-learning, Monte Carlo (MC), and Monte Carlo Tree Search (MCTS). Left: Grid World environment. Right: Charge-Discharge environment. Reward redistribution requires an exponentially smaller number of states than the original methods.

Charge-Discharge environment: We test RUDDER on another task, the Charge-Discharge environment, which has two states: *discharged* `D` / *charged* `C` and two actions *discharge* `d` / *charge* `c`. The deterministic reward is $r(\mathtt{D},\mathtt{d}) = 1, r(\mathtt{C},\mathtt{d}) = 10, r(\mathtt{D},\mathtt{c}) = 0$, and $r(\mathtt{D},\mathtt{c}) = 0$. The reward $r(\mathtt{C},\mathtt{d})$ is accumulated for the whole episode and given only at time $T \in \{3,\ldots,13\}$, which determines the maximal delay of a reward. The deterministic state transitions are $(\{\mathtt{D},\mathtt{C}\},\mathtt{d}) \rightarrow \mathtt{D}$ and $(\{\mathtt{D},\mathtt{C}\},\mathtt{c}) \rightarrow \mathtt{C}$. The optimal policy alternates between charging and discharging to accumulate a reward of 10 every other time step. RUDDER is based on a monotonic LSTM with layer-wise relevance propagation (LRP) for the backwards analysis (see Appendix **??** for more details). The reward redistribution provided by RUDDER served to learn a policy by Q-learning. We compare RUDDER with Q-learning, MC and MCTS. The results are shown in Figure 2. Reward redistribution requires an exponentially smaller number of states than Q-learning, MC and MCTS to learn the optimal policy.

Atari Games Bowling and Venture: We investigated the Atari games supported by the Arcade Learning Environment [7] and OpenAI Gym [8] for games with delayed reward. Requirements for proper games to demonstrate performance on delayed reward are: (I) large delay between an action and the resulting reward, (II) no distractions due to other rewards or changing characteristics of the environment, (III) no skills to be learned to receive the delayed reward. The requirements were met by Bowling and Venture. In Bowling the only reward of the game is given at the end of the episode, 200 frames after the first relevant action. In Venture the first reward has a minimum delay of 120 frames from the first relevant action. Figure 3 shows that RUDDER learns faster than rainbow [17], Prioritized DDQN [47], Noisy DQN [11], Dueling DDQN [63], DQN [33], C51 (Distributional DQN) [6], DDQN [60], A3C [31], and Ape-X DQN [22]. RUDDER sets a new state-of-the-art score in Bowling after 12M environment frames. Thus, RUDDER outperforms its competitors in only 10% of their training time, as shown in Table 1. For more details see For more details see Appendix A5.2.

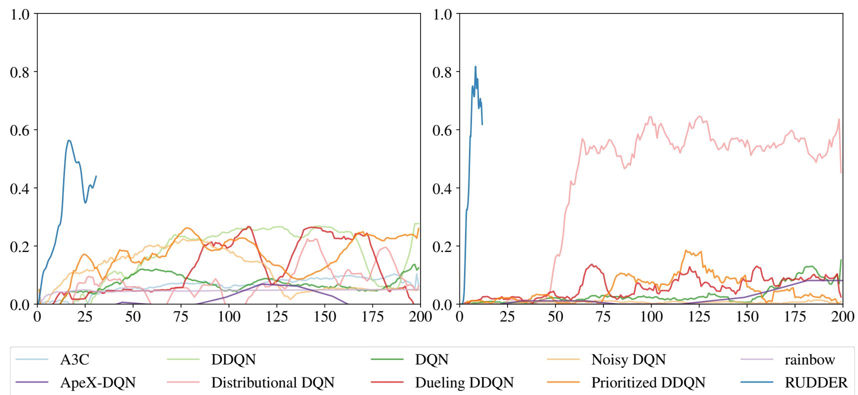

Figure 3: RUDDER learns the delayed reward for the Atari games Bowling and Venture faster than other methods. Normalized human-percentage scores during training for Bowling (left) and for Venture (right), where learning curves are taken from previous publications [17, 22]. RUDDER sets a new state-of-the-art for Bowling.

Algorithm	Frames	Bowling		Venture	
		%	raw	%	raw
RUDDER	12M	**62.10**	**108.55**	96.55	1,147
rainbow	200M	5.01	30	0.46	5.5
Prioritized DDQN	200M	28.71	62.6	72.67	863
Noisy DQN	200M	39.39	77.3	0	0
Dueling DDQN	200M	30.81	65.5	41.85	497
DQN	200M	19.84	50.4	13.73	163
Distributional DQN	200M	37.06	74.1	93.22	1,107
DDQN	200M	32.7	68.1	8.25	98
Ape-X DQN	22,800M	-17.6	4	**152.67**	**1,813**
Random	–	0	23.1	0	0
Human	–	100	160.7	100	1,187

Table 1: Results of RUDDER and other methods when learning the Atari games Bowling and Venture. Normalized human-percentage and raw scores over 200 testing-games with no-op starting condition: A3C scores are not reported, as not available for no-op starting condition. Scores for other methods were taken from previous publications [6, 17]. The RUDDER model is chosen based only on its training loss over 12M frames.

RUDDER Implementation for Bowling and Venture. We implemented RUDDER for the proximal policy optimization (PPO) algorithm [50]. For policy gradients the expected updates are $\mathrm{E}_{\pi}\left[\nabla_{\theta} \log \pi(a \mid s; \boldsymbol{\theta}) q^{\pi}(s,a)\right]$, where $q^{\pi}(s,a)$ is replaced during learning by the return G_t or its expectation. RUDDER policy gradients replace $q^{\pi}(s,a)$ by the redistributed reward $r(s,a)$ assuming an optimal return decomposition. With eligibility traces using $\lambda \in [0,1]$ for G_t^{λ} [58], we have the rewards $\rho_t = r_t + \lambda \rho_{t+1}$ with $\rho_{T+1} = 0$ and the expected updates $\mathrm{E}_{\pi}\left[\nabla_{\theta} \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \rho_t\right]$. We use integrated gradients [55] for the backward analysis of RUDDER. The LSTM prediction g is decomposed by the integrated gradient IG via the equation $g(\boldsymbol{x}) = \mathrm{IG}(g, \boldsymbol{x}, \boldsymbol{0}) + g(\boldsymbol{0}) = \sum_{t=0}^{T} \big(h(a_t, \Delta(s_t, s_{t+1})) + (1/(T+1)) g(\boldsymbol{0})\big)$. For Atari games, Δ is defined as pixel-wise difference of two consecutive frames. To make static objects visible, we augment the input with the current frame. For more implementation details see Appendix A5.2. Source code will be made available.

Evaluation Methodology. Agents were trained for 12M environment frames with *no-op starting condition*, i.e. a random number of up to 30 no-operation actions at the start of a game. Training episodes are terminated by loss of life or at 108K frames. After training, the best model was selected based on training data and evaluated on 200 games with no-op starting condition and a maximum length of 108K frames, following [61]. For comparison across games, the normalized human-percentage scores according to [6] are reported.

Visual Confirmation of the Learning Boost of Reward Redistribution. We visually confirmed a meaningful and helpful redistribution of reward in both Bowling and Venture during training. As illustrated in Figure 4, RUDDER is capable of redistributing a reward to key events in game, drastically shortening the delay of the reward and quickly steering the agent toward good policies. Furthermore, it enriches sequences that were sparse in reward with a dense reward signal.

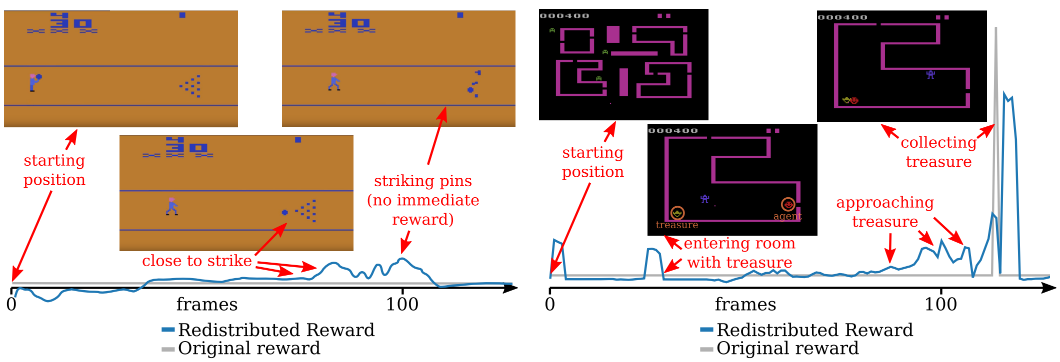

Figure 4: Observed return decomposition by RUDDER in two Atari games with long delayed rewards. **Left:** In the game Bowling reward is only given after three strikes have been performed. RUDDER identifies the actions that guide the ball in the right direction to hit all pins. Once the ball hit the pins, RUDDER detects the delayed reward associated with striking the pins down. In the figure only 100 frames are represented but the whole episode spans 200 frames. In the original game, the reward is given only at the end of the episode. **Right:** In the game Venture reward is only obtained after picking the treasure. RUDDER guides the agent (red) towards the treasure (golden) via reward redistribution. Reward is redistributed to entering a room with treasure. Furthermore, the redistributed reward gradually increases as the agent approaches the treasure. For illustration purposes, the green curve shows the return redistribution before applying lambda. The environment only gives reward at the event of collecting treasure (blue curve).

5 Discussion and Conclusion

Exploration is the most critical part of RUDDER, since discovering delayed rewards is the first step to exploit them.

Human expert episodes are an alternative to exploration and can serve to fill the lessons replay buffer. Learning can be sped up considerably when LSTM identifies human key actions. Return decomposition will reward human key actions even for episodes with low return since other actions that thwart high returns receive negative reward. Using human demonstrations in reinforcement learning led to a huge improvement on some Atari games like Montezuma's Revenge [37, 1].

Conclusion. We have shown that for finite Markov decision processes with delayed rewards TD exponentially slowly corrects biases and MC can increase exponentially many variances of estimates, both in the number of delay steps. We have introduced RUDDER, a return decomposition method, which creates a new MDP that keps the optimal policies but its redistributed rewards do not have delays. In the optimal case TD for the new MDP is unbiased. On two artificial tasks we demonstrated that RUDDER is exponentially faster than TD, MC, and MC Tree Search (MCTS). For the Atari game Venture with delayed reward RUDDER outperforms all methods except Ape-X DQN in much less learning time. For the Atari game Bowling with delayed reward RUDDER improves the state-of-the-art and outperforms PPO, Rainbow, and APE-X with less learning time.

Acknowledgments

This work was supported by NVIDIA Corporation, Bayer AG with Research Agreement 09/2017, Merck KGaA, Zalando SE with Research Agreement 01/2016, Audi.JKU Deep Learning Center, Audi Electronic Venture GmbH, Janssen Pharmaceutica, IWT research grant IWT150865 (Exaptation), LIT grant LSTM4Drive, and FWF grant P 28660-N31.

References

[1] Y. Aytar, T. Pfaff, D. Budden, T. Le Paine, Z. Wang, and N. de Freitas. Playing hard exploration games by watching YouTube. *ArXiv*, 2018.

[2] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7):e0130140, 2015.

[3] B. Bakker. Reinforcement learning with long short-term memory. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1475–1482. MIT Press, 2002.

[4] B. Bakker. Reinforcement learning by backpropagation through an lstm model/critic. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 127–134, 2007.

[5] A. G. Barto and R. S. Sutton. Landmark learning: An illustration of associative search. *Biological Cybernetics*, 42(1):1–8, 1981.

[6] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research (ICML)*, pages 449–458. PMLR, 2017.

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

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *ArXiv*, 2016.

[9] A. D. Edwards, L. Downs, and J. C. Davidson. Forward-backward reinforcement learning. *ArXiv*, 2018.

[10] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, 2018. ArXiv: 1802.01561.

[11] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration. *ArXiv*, 2018. Sixth International Conference on Learning Representations (ICLR).

[12] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1):163–223, 2003.

[13] A. Goyal, P. Brakel, W. Fedus, T. Lillicrap, S. Levine, H. Larochelle, and Y. Bengio. Recall traces: Backtracking models for efficient reinforcement learning. *ArXiv*, 2018.

[14] D. Ha and J. Schmidhuber. World models. *ArXiv*, 2018.

[15] M. J. Hausknecht and P. Stone. Deep recurrent Q-Learning for partially observable MDPs. *ArXiv*, 2015.

[16] N. Heess, G. Wayne, Y. Tassa, T. P. Lillicrap, M. A. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *ArXiv*, 2016.

[17] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *ArXiv*, 2017.

[18] S. Hochreiter. Implementierung und Anwendung eines 'neuronalen' Echtzeit-Lernalgorithmus für reaktive Umgebungen. Practical work, Supervisor: J. Schmidhuber, Institut für Informatik, Technische Universität München, 1990.

[19] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Technische Universität München, 1991.

[20] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In J. F. Kolen and S. C. Kremer, editors, *A Field Guide to Dynamical Recurrent Networks*. IEEE Press, 2001.

[21] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[22] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver. Distributed prioritized experience replay. *ArXiv*, 2016. Sixth International Conference on Learning Representations (ICLR).

[23] A. H. Klopf. Brain function and adaptive systems - a heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, L. G. Hanscom Field, Bedford, MA, 1972.

[24] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1061–1068, 2013.

[25] W. Landecker, M. D. Thomure, L. M. A. Bettencourt, M. Mitchell, G. T. Kenyon, and S. P. Brumby. Interpreting individual classifications of hierarchical networks. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 32–38, 2013.

[26] S. Y. Lee, S. Choi, and S.-Y. Chung. Sample-efficient deep reinforcement learning via episodic backward update. *ArXiv*, 2018.

[27] M. Lengyel and P. Dayan. Hippocampal contributions to control: The third way. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 889–896. Curran Associates, Inc., 2008.

[28] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for MDPs. In *Ninth International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2006.

[29] L. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, 1993.

[30] J. Luoma, S. Ruutu, A. W. King, and H. Tikkanen. Time delays, competitive interdependence, and firm performance. *Strategic Management Journal*, 38(3):506–525, 2017.

[31] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937. PMLR, 2016.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing Atari with deep reinforcement learning. *ArXiv*, 2013.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, , and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[34] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognition*, 65:211 – 222, 2017.

[35] P. W. Munro. A dual back-propagation scheme for scalar reinforcement learning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, pages 165–176, 1987.

[36] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML'99)*, pages 278–287, 1999.

[37] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. van Hasselt, J. Quan, M. Večerík, M. Hessel, R. Munos, and O. Pietquin. Observe and look further: Achieving consistent performance on Atari. *ArXiv*, 2018.

[38] B. Poulin, R. Eisner, D. Szafron, P. Lu, R. Greiner, D. S. Wishart, A. Fyshe, B. Pearcy, C. MacDonell, and J. Anvik. Visual explanation of evidence in additive classifiers. In *Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence (IAAI)*, volume 2, pages 1822–1829, 2006.

[39] M. L. Puterman. Markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, chapter 8, pages 331–434. Elsevier, 1990.

[40] M. L. Puterman. *Markov Decision Processes*. John Wiley & Sons, Inc., 2005.

[41] H. Rahmandad, N. Repenning, and J. Sterman. Effects of feedback delay on learning. *System Dynamics Review*, 25(4):309–338, 2009.

[42] B. Ravindran and A. G. Barto. Symmetries and model minimization in Markov decision processes. Technical report, University of Massachusetts, Amherst, MA, USA, 2001.

[43] B. Ravindran and A. G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1011–1016, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[44] A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Trinity Hall and Cambridge University Engineering Department, 1989.

[45] T. Robinson and F. Fallside. Dynamic reinforcement driven error propagation networks with application to game playing. In *Proceedings of the 11th Conference of the Cognitive Science Society, Ann Arbor*, pages 836–843, 1989.

[46] H. Sahni. Reinforcement learning never worked, and 'deep' only helped a bit. `himanshusahni.github.io/2018/02/23/reinforcement-learning-never-worked.html`, 2018.

[47] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *ArXiv*, 2015.

[48] J. Schmidhuber. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90 (revised), Institut für Informatik, Technische Universität München, 1990. Experiments by Sepp Hochreiter.

[49] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[50] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, 2018.

[51] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[52] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, 2017.

[53] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[54] M. J. Sobel. The variance of discounted Markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.

[55] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. *ArXiv*, 2017.

[56] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[57] R. S. Sutton and A. G. Barto. Towards a modern theory of adaptive networks: expectation and prediction. *Psychol. Rev.*, 88(2):135–170, 1981.

[58] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2017. Draft from November 2017.

[59] A. Tamar, D. DiCastro, and S. Mannor. Policy gradients with variance related risk criteria. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*, 2012.

[60] H. van Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[61] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100. AAAI Press, 2016.

[62] Z. Wang, N. de Freitas, and M. Lanctot. Dueling network architectures for deep reinforcement learning. *ArXiv*, 2015.

[63] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003. PMLR, 2016.

[64] E. Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.

[65] E. Wiewiora, G. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML'03)*, pages 792–799, 2003.

[66] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[67] Z. Xu, H. van Hasselt, and D. Silver. Meta-gradient reinforcement learning. *ArXiv*, 2018.

[68] J. Zhang, Z. L. Lin, J. Brandt, X. Shen, and S. Sclaroff. Top-down neural attention by excitation backprop. In *Proceedings of the 14th European Conference on Computer Vision (ECCV)*, pages 543–559, 2016. part IV.