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, the number of which 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. Source code is available at https://github.com/ml-jku/baselines-rudder and demonstration videos at https://goo.gl/EQerZV.

1 Introduction

Assigning the credit for a received reward to actions that were performed is one of the central tasks in reinforcement learning [107]. Long term credit assignment has been identified as one of the largest challenges in reinforcement learning [91]. Current reinforcement learning methods are still significantly slowed down when facing long-delayed rewards [84, 66]. 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 learning methods provide solutions to one or more of these phases. Most prominent are Deep Q-Networks (DQNs) [71, 72], which combine Q-learning with convolutional neural networks for visual reinforcement learning [58]. The success of DQNs is attributed to *experience replay* [64], which stores observed state-reward transitions and then samples from them. Prioritized experience replay [93, 50] advanced the sampling from the replay memory. Different policies perform exploration in parallel for the Ape-X DQN and share a prioritized experience replay memory [50]. DQN was extended to double DQN (DDQN) [113, 114] which helps exploration as the overestimation bias is reduced. Noisy DQNs [21] explore by a stochastic layer in the policy network (see [40, 94]).

*authors contributed equally

Distributional Q-learning [8] profits from noise since means that have high variance are more likely selected. The dueling network architecture [117, 118] separately estimates state values and action advantages, which helps exploration in unknown states. Policy gradient approaches [124] explore via parallel policies, too. A2C has been improved by IMPALA through parallel actors and correction for policy-lags between actors and learners [19]. A3C with asynchronous gradient descent [70] and Ape-X DPG [50] also rely on parallel policies. Proximal policy optimization (PPO) extends A3C by a surrogate objective and a trust region optimization that is realized by clipping or a Kullback-Leibler penalty [96].

Recent approaches aim to solve learning problems caused by delayed rewards. Function approximations of value functions or critics [72, 70] bridge time intervals if states associated with rewards are similar to states that were encountered many steps earlier. For example, assume a function that has 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 beginning of an episode and predict already high reward for states possessing the same feature. Multi-step temporal difference (TD) learning [105, 107] improved both DQNs and policy gradients [39, 70]. AlphaGo and AlphaZero learned to play Go and Chess better than human professionals using Monte Carlo Tree Search (MCTS) [97, 98]. MCTS simulates games from a time point until the end of the game or an evaluation point and therefore captures long delayed rewards. Recently, world models using an evolution strategy were successful [36]. These forward view approaches are not feasible in probabilistic environments with a high branching factor of state transition. Backward view approaches trace back from known goal states [18] or from high reward states [30]. 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) [41, 47] to predict the return of an episode. LSTM was already used in reinforcement learning [95] for advantage learning [4] and learning policies [37, 70, 38]. However, sensitivity analysis by "backpropagation through a model" [75, 87, 88, 5] has major drawbacks: local minima, instabilities, exploding or vanishing gradients in the world model, proper exploration, actions are only regarded by sensitivity but not their contribution (relevance) [40, 94].

Since sensitivity analysis substantially hinders learning, we use contribution analysis for backward analysis like contribution-propagation [60], contribution approach [81], excitation backprop [127], layer-wise relevance propagation (LRP) [3], Taylor decomposition [3, 73], or integrated gradients (IG) [103]. 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 a new MDP with the same optimal policies as for the original MDP. Redistributing the reward is fundamentally different from reward shaping [76, 122], which changes the reward as a function of states but not of actions. Reward shaping and "look-back advice" [123] both keep the original reward, which may still have long delays that cause an exponential slow-down of 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 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 $(q^\pi)^{i+1}(s_t,a_t) = (q^\pi)^i(s_t,a_t) + \alpha \left(\sum_t^T r_{t+1} - (q^\pi)^i(s_t,a_t) \right)$ (constant-α MC). 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 $V^\pi(s,a) = \mathrm{Var}_\pi \left[G_t \mid s,a \right]$ 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 [100, 108]. The variance $V^\pi(s,a)$ consists of three parts: (i) The immediate variance $\mathrm{Var}_r \left[r \mid s,a \right]$ stemming from the probabilistic reward $p(r \mid s,a)$. (ii) The local variance $\gamma^2 \mathrm{Var}_{s',a'} \left[q^\pi(s',a') \mid s,a \right]$ caused by probabilistic state transitions and probabilistic policy. (iii) The expected variance $\gamma^2 \mathrm{E}_{s',a'} \left[V^\pi(s',a') \mid s,a \right]$ of the next Q-values, which is zero for TD since it replaces $q^\pi(s',a')$ by a 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 [56, 6, 106, 99].

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 $q^i(s_{T-i},a_{T-i}) = \alpha^{i+1} r^1_{T+1}$, where r^1_{T+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 bias by a certain amount, TD 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" [82, 83] gives with $\boldsymbol{V}^\pi_{T+1} = \mathbf{0}$, $\boldsymbol{\omega}_{T+1} = \mathbf{0}$, and row-stochastic matrix $\boldsymbol{P}_{t \leftarrow k} = \prod_{\tau=t}^{k-1} \boldsymbol{P}_\tau$:

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

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 in the appendix. 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 (cf. Corollary A1). Consequently, we aim for decreasing the on-site variance $\boldsymbol{\omega}_k$ for large k, in order to reduce the overall 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 grows is in both cases exponentially 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. Thus, $\mathcal{P}$ 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 [85, 86]. These properties are known from state abstraction and aggregation [63] and from bisimulations [28]. 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. For more details see Appendix A1.3.1.

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 the reward at the end of the sequence. 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 A1 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: $g((s, a)_{0:T}) = \tilde{r}_{T+1}$, 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 from the backward analysis. Since $\tilde{\mathcal{P}}$ is an MDP, the reward can be predicted solely from (a_T, s_T). To avoid this Markov property in the input sequence, we replace s by a difference $\Delta(s, s')$ between state s and its successor s'. The difference Δ is assumed to assure statistical independence of $(a_t, \Delta(s_t, s_{t+1}))$ from other components in the sequence $(a, \Delta)_{0:T} = (a_0, \Delta(s_0, s_1), \ldots, a_t, \Delta(s_t, s_{t+1}))$. The function g is decomposed by contribution analysis into a sum of hs: $g((a, \Delta)_{0:T}) = \tilde{r}_{T+1} = \sum_{t=0}^{T} h(a_t, \Delta(s_t, s_{t+1}))$. The actual reward redistribution is $r_{t+1} = \tilde{r}_{T+1} h(a_t, \Delta(s_t, s_{t+1})) / g((a, \Delta)_{0:T})$ 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*. The rewards for $g((a, \Delta)_{0:T}) = \tilde{r}_{T+1}$ are $R_0 = h_0 = h(a_0, \Delta(s_0, s_1)) = \tilde{q}^\pi(s_0, a_0)$ 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 a redistributed reward (I) has the same optimal policies as $\tilde{\mathcal{P}}$ of the delayed reward, and (II) for an optimal return decomposition, 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 A4 in the appendix. In particular, when starting with zero initialized Q-values, 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 [107], 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 an exploration sequence at a certain time in the episode 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. The lessons replay buffer is an episodic memory, which has been used for episodic control [62] and for episodic backward update to efficiently propagate delayed rewards [61]. If RUDDER safe exploration discovers an episode with unexpected delayed reward, it is secured in a lessons replay buffer [64]. Unexpected is indicated by a large prediction error of the LSTM. Episodes with larger error are more often sampled from the lessons replay buffer similar to prioritized experience replay.

(III) LSTM and contribution analysis. LSTM networks [41, 47], are used to predict the return from an input sequence. LSTM solves the vanishing gradient problem [41, 44], which severely impedes credit assignment in recurrent neural networks, i.e. the correct identification of input events that are relevant but far in the past. LSTM backward analysis is done through contribution analysis like layer-wise relevance propagation (LRP) [3], Taylor decomposition [3, 73], or integrated gradients (IG) [103]. These methods identify the contributions of the inputs to the final prediction and, thereby, compute the return decomposition.

The LSTM return decomposition is optimal if LSTM predicts at every time step the expected return. To push LSTM toward optimal return decomposition, we introduce continuous return predictions as auxiliary tasks, where the LSTM has to predict the expected return at every time step. Hyperparameters are when and how often LSTM predicts and how continuous prediction errors are weighted. Strictly monotonic LSTM architecture (see Appendix A4.3.1) can also ensure that LSTM decomposition is optimal.

4 Experiments

We set $\gamma = 1$ for delayed rewards in MDPs with finite time horizon or absorbing states to avoid discounting of the delayed rewards. Discount factors close to 1 have been confirmed to be suited to long delays by meta-gradient reinforcement learning [125].

Grid World: RUDDER is tested on a grid world with delayed reward at the end of an episode. The MDP should illustrate an environment where an agent must run to a bomb to defuse it and afterwards run away as it may still explode. The sooner the agent defuses the bomb, the further away it can run. An alternative strategy is to directly run away, which, however, leads to less return than defusing the bomb. The Grid World task consists of a quadratic 31×31 grid with *bomb* at coordinate $[30, 15]$ and *start* at $[30 - d, 15]$, where d is the delay of the task. The agent can move in four different directions (*up*, *right*, *left*, and *down*), where only moves that keep the agent on the grid are allowed. The episode

finishes after $1.5d$ steps. At the end of the episode the agent receives a reward of 1000 if it has visited *bomb*. At each time step the agent receives an immediate reward of $c \cdot t \cdot h$, where c is a factor that depends on the chosen action, t is the current time step, and h is the Hamming distance to *bomb*. Each move the agent reduces the Hamming distance to *bomb* is penalized by the immediate reward using $c = -0.09$. Each move the agent increases the Hamming distance to *bomb* is rewarded by the immediate reward using $c = 0.1$. Due to this distracting reward the agent is forced to learn the Q-values precisely, since the immediate reward hints at a sub-optimal policy. This is because the learning process has to determine that visiting *bomb* leads to larger Q-values than increasing the distance to *bomb*.

To investigate how the delay affects bias and variance, Q-values are estimated by TD and MC for an ϵ-greedy optimal policy to assure 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 for MC estimators has a smaller number of Q-values with high variance than the original MDP. Figure 1 also shows that RUDDER for TD estimators corrects the bias faster than TD for the original MDP. After these policy evaluation experiments with a fixed policy, we move on to learning an optimal policy. For learning the optimal policy, we compare Q-learning, Monte Carlo (MC), and Monte Carlo Tree Search (MCTS) on the grid world, where *sample updates* are used for Q-learning and MC [107]. 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 exponentially. More information is available in Appendix A5.1.1.

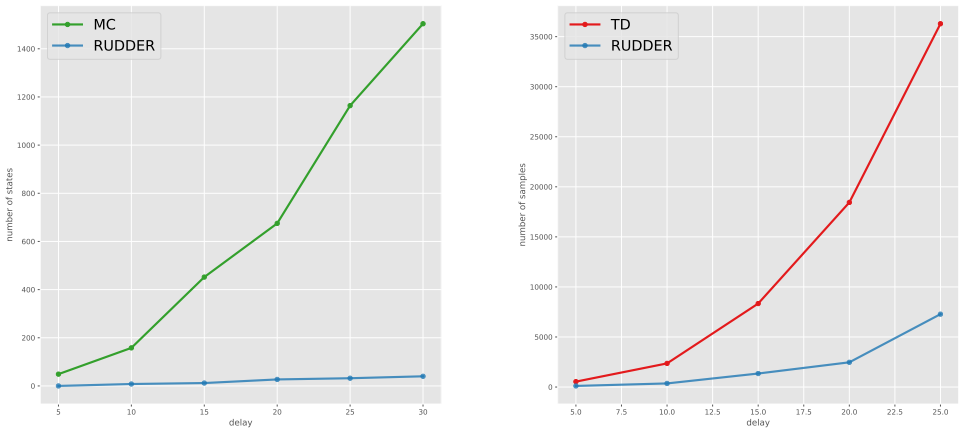


Figure 1: Experimental evaluation of bias and variance of different Q-value estimators on the Grid World. **Left:** The number of states affected by high variance grows exponentially with the delay. **Right:** Number of samples to reduce the bias by 80% of the original error for different delays.

Charge-Discharge environment: We test RUDDER on another task, the Charge-Discharge environment, which has two states: *charged* `C` / *discharged* `D` and two actions *charge* `c` / *discharge* `d`. The deterministic reward is $r(\texttt{D},\texttt{d}) = 1, r(\texttt{C},\texttt{d}) = 10, r(\texttt{D},\texttt{c}) = 0$, and $r(\texttt{C},\texttt{c}) = 0$. The reward $r(\texttt{C},\texttt{d})$ is accumulated for the whole episode and given only at time $T+1$ with $T \in \{3, 5, \ldots, 17\}$, which determines the maximal delay of a reward. The deterministic state transitions are $(\{\texttt{D},\texttt{C}\},\texttt{d}) \to \texttt{D}$ and $(\{\texttt{D},\texttt{C}\},\texttt{c}) \to \texttt{C}$. The optimal policy alternates between charging and discharging to accumulate a reward of 10 every other time step.

For this environment, RUDDER is realized as a monotonic LSTM with layer-wise relevance propagation (LRP) for the backward analysis (see Appendix A5.1.2 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, where Q-learning and MC use sample-updates. The results are

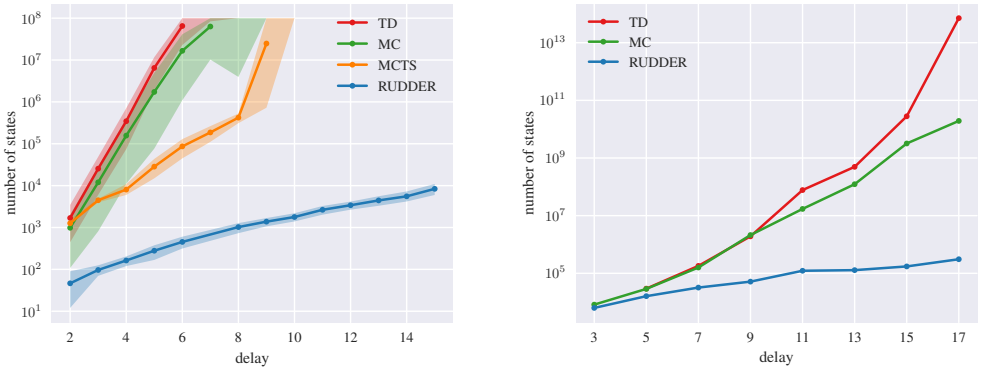


Figure 2: Number of observed states in logarithmic scale required by different methods to learn a policy that achieves 90% of the return of the optimal policy for different delays. We compare TD realized by Q-learning, Monte Carlo (MC), Monte Carlo Tree Search (MCTS), and RUDDER. **Left:** Grid World environment. **Right:** Charge-Discharge environment. The difference between the states required by RUDDER and the other methods grows exponentially with the delay.

shown in Figure 2. Reward redistribution requires to observe an exponentially smaller number of states than Q-learning, MC, and MCTS to learn the optimal policy.

Atari Games: We investigated the Atari games supported by the Arcade Learning Environment [9] and OpenAI Gym [13] for games with delayed reward. Requirements for proper games to demonstrate performance on delayed reward environments 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 each turn, i.e. more than 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 [39], Prioritized DDQN [93], Noisy DQN [21], Dueling DDQN [118], DQN [72], C51 (Distributional DQN) [8], DDQN [113], A3C [70], and Ape-X DQN [50]. 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 Appendix A5.2.

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 [8, 39]. The RUDDER model is chosen based only on its training loss over 12M frames.

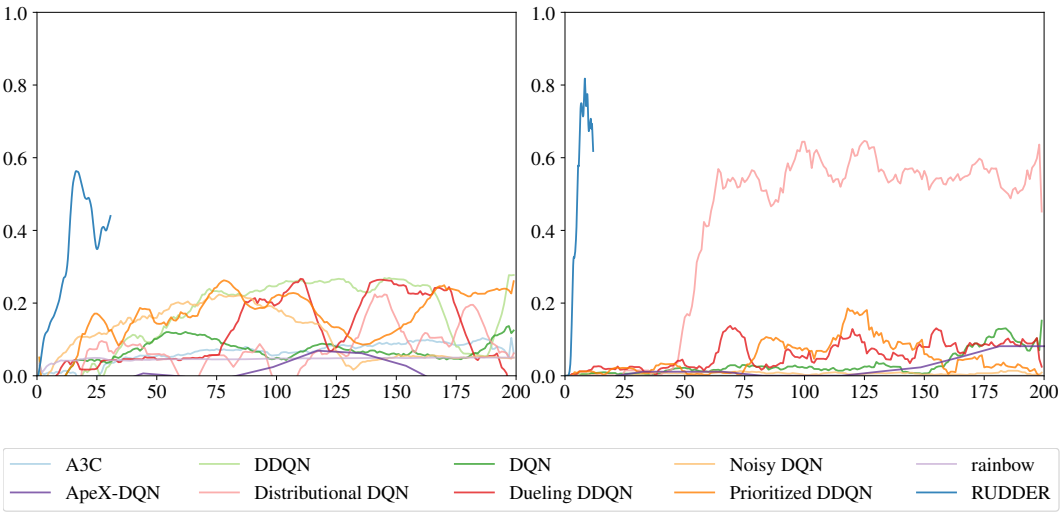


Figure 3: RUDDER learns the delayed reward for the Atari games Bowling and Venture faster than other methods. Normalized human-percentage scores during training for Bowling (left) and for Venture (right), where learning curves are taken from previous publications [39, 50]. RUDDER sets a new state-of-the-art for Bowling.

RUDDER Implementation for Atari Games. We implemented RUDDER for the proximal policy optimization (PPO) algorithm [96]. 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 [107], 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 [103] for the backward analysis of RUDDER. The LSTM prediction g is decomposed by integrated gradients: $g(\boldsymbol{x}) = \sum_{t=0}^{T} (h(a_t, \Delta(s_t, s_{t+1})) + 1/(T+1) g(\mathbf{0}))$. For Atari games, Δ is defined as the 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 is available at `https://github.com/ml-jku/baselines-rudder`.

Evaluation Methodology. Agents were trained for 12M 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 when a life is lost or at a maximum of 108K frames. After training, the best model was selected based on training data and evaluated on 200 games [114]. For comparison across games, the normalized human-percentage scores according to [8] are reported.

Visual Confirmation of Detecting Relevant Events by Reward Redistribution. We visually confirm 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 a 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. Video demonstrations are available at `https://goo.gl/EQerZV`.

5 Discussion and Conclusion

Exploration is most critical, 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 [80, 2].

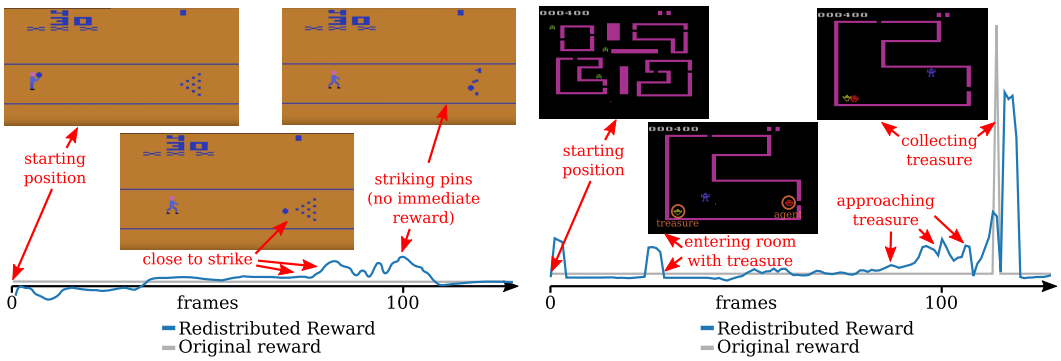


Figure 4: Observed return decomposition by RUDDER in two Atari games with long delayed rewards. **Left:** In the game Bowling, reward is only given after a turn which consist of multiple rolls. RUDDER identifies the actions that guide the ball in the right direction to hit all pins. Once the ball hit the pins, RUDDER detects the delayed reward associated with striking the pins down. In the figure only 100 frames are represented but the whole turn spans more than 200 frames. In the original game, the reward is given only at the end of the turn. **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).

Conclusion. We have shown that for finite Markov decision processes with delayed rewards TD exponentially slowly corrects biases and MC can increase many variances of estimates exponentially, both in the number of delay steps. We have introduced RUDDER, a return decomposition method, which creates a new MDP that keeps 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

References are provided in Appendix A6.

Contents

Appendix

A1 Reinforcement Learning and Credit Assignment

A1.1 Finite Markov Decision Process

We consider a finite Markov decision process (MDP) $\mathcal{P}$, which is 6-tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \pi, \gamma)$:

- $\mathcal{S}$ is a finite set of states; S_t is the random variable for states at time t with values $s, s' \in \mathcal{S}$ and has a discrete probability distribution.
- $\mathcal{A}$ is a finite set of actions (sometimes state-dependent $\mathcal{A}(s)$); A_t is the random variable for actions at time t with values $s, a' \in \mathcal{A}$ and has a discrete probability distribution.
- $\mathcal{R}$ is a finite set of rewards; R_t is the random variable for actions at time t with values $R_{t+1} = r \in \mathcal{A}$ and has a discrete probability distribution.
- $p(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$ are the transition-reward distributions over state-rewards conditioned on state-actions,
- $\pi(A_{t+1} = a' \mid S_{t+1} = s')$ is the policy, which is a distribution over actions given the state,
- $\gamma \in [0, 1]$ is the discount factor.

At time t, the random variables give the states, actions, and reward of the MDP, while low-case letter give possible values. At each time t, the environment is in some state $s_t \in \mathcal{S}$. The agent π takes an action $a_t \in \mathcal{A}$, which causes a transition of the environment to state s_{t+1} and a reward r_{t+1} for the agent. Therefore, the MDP creates a sequence

$$(S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots) \ . \tag{A1}$$

The marginal probabilities are:

$$p(s', r \mid s, a) \ = \ \Pr\left[S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\right] \ , \tag{A2}$$

$$p(r \mid s, a) \ = \ \Pr\left[R_{t+1} = r \mid S_t = s, A_t = a\right] \ = \ \sum_{s'} p(s', r \mid s, a) \ , \tag{A3}$$

$$p(s' \mid s, a) \ = \ \Pr\left[S_{t+1} = s' \mid S_t = s, A_t = a\right] \ = \ \sum_{r} p(s', r \mid s, a) \ . \tag{A4}$$

We used a sum convention: $\sum_{a,b}$ goes over all possible values of a and b, that is, all combinations which fulfill the constraints on a and b. If b is a function of a (fully determined by a), then $\sum_{a,b} = \sum_{a}$.
We denote expectations

- E_π is the expectation where the random variable is an MDP sequence of state, actions, and rewards generated with policy π.
- E_s is the expectation where the random variable is S_t with values $s \in \mathcal{S}$.
- E_a is the expectation where the random variable is A_t with values $a \in \mathcal{A}$.
- E_r is the expectation where the random variable is R_{t+1} with values $r \in \mathcal{R}$.
- $\mathrm{E}_{s',s,a,r,a'}$ is the expectation where the random variables are S_{t+1} with values $s' \in \mathcal{S}$, S_t with values $s \in \mathcal{S}$, A_t with values $a \in \mathcal{A}$, A_{t+1} with values $a' \in \mathcal{A}$, and R_{t+1} with values $r \in \mathcal{R}$. If more or less random variables are used, the notation is consistently adapted.

The return G_t is the accumulated reward starting from $t + 1$:

$$G_t \ = \ \sum_{k=0}^{\infty} \gamma^k \ R_{t+k+1} \ . \tag{A5}$$

The discount factor γ determines how much immediate rewards are favored over more distant rewards. For $\gamma = 0$ the return (the objective) is determined the largest expected immediate reward, while for $\gamma = 1$ the return is determined by the expected sum of future rewards if the sum exists.

State-Value and Action-Value Function. The state-value function $v^\pi(s)$ for policy π and state s is defined as

$$v^\pi(s) \;=\; \mathrm{E}_\pi\left[G_t \mid S_t = s\right] \;=\; \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \; R_{t+k+1} \mid S_t = s\right] . \tag{A6}$$

Starting at $t = 0$:

$$v_0^\pi \;=\; \mathrm{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t \; R_{t+1}\right] \;=\; \mathrm{E}_\pi\left[G_0\right] , \tag{A7}$$

The optimal state-value function v_* and policy π_* are

$$v_*(s) \;=\; \max_\pi v^\pi(s) , \tag{A8}$$

$$\pi_* \;=\; \arg\max_\pi v^\pi(s) \;\text{ for all } s . \tag{A9}$$

The action-value function $q^\pi(s, a)$ for policy π is the expected return when staring from $S_t = s$, taking action $A_t = a$, and following policy π.

$$q^\pi(s,a) \;=\; \mathrm{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \;=\; \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \; R_{t+k+1} \mid S_t = s, A_t = a\right] . \tag{A10}$$

The optimal action-value function q_* and policy π_* are

$$q_*(s,a) \;=\; \max_\pi q^\pi(s,a) , \tag{A11}$$

$$\pi_* \;=\; \arg\max_\pi q^\pi(s,a) \;\text{ for all } (s,a) . \tag{A12}$$

The optimal action-value function q_* can be expressed via the optimal value function v_*:

$$q_*(s,a) \;=\; \mathrm{E}\left[R_{t+1} \;+\; \gamma \, v_*(S_{t+1}) \mid S_t = s, A_t = a\right] . \tag{A13}$$

Vice versa, the optimal state-value function v_* can be expressed via the optimal action-value function q_* using the optimal policy π_*:

$$\begin{aligned} v_*(s) \;&=\; \max_a q^{\pi_*}(s,a) \;=\; \max_a \mathrm{E}_{\pi_*}\left[G_t \mid S_t = s, A_t = a\right] \;= \\ &\max_a \mathrm{E}_{\pi_*}\left[R_{t+1} \;+\; \gamma \, G_{t+1} \mid S_t = s, A_t = a\right] \;= \\ &\max_a \mathrm{E}\left[R_{t+1} \;+\; \gamma \, v_*(S_{t+1}) \mid S_t = s, A_t = a\right] . \end{aligned} \tag{A14}$$

Finite time horizon and no discount. We consider a **finite** time horizon, that is, we consider only episodes of length T but may receive at episode end reward R_{T+1} at time $T + 1$. The finite time horizon MDP creates a sequence

$$(S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_{T-1}, A_{T-1}, R_T, S_T, A_T, R_{T+1}) . \tag{A15}$$

Furthermore we do not discount future rewards, that is, we set $\gamma = 1$. The return G_t from time t to T is the sum of rewards:

$$G_t \;=\; \sum_{k=0}^{T-t} R_{t+k+1} . \tag{A16}$$

The state-value function v for policy π is

$$v^\pi(s) \;=\; \mathrm{E}_\pi\left[G_t \mid S_t = s\right] \;=\; \mathrm{E}_\pi\left[\sum_{k=0}^{T-t} R_{t+k+1} \mid S_t = s\right] \tag{A17}$$

and the action-value function q for policy π is

$$\begin{aligned} q^\pi(s,a) \;&=\; \mathrm{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \;=\; \mathrm{E}_\pi\left[\sum_{k=0}^{T-t} R_{t+k+1} \mid S_t = s, A_t = a\right] \\ &=\; \mathrm{E}_\pi\left[R_{t+1} \;+\; G_{t+1} \mid S_t = s, A_t = a\right] \\ &=\; \sum_{s',r} p(s', r \mid s, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s') \, q^\pi(s', a')\right] . \end{aligned} \tag{A18}$$

From Bellman equation Eq. (A18), we obtain:

$$\sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q^\pi(s',a') \;=\; q^\pi(s,a) \;-\; \sum_r r\, p(r \mid s,a)\;, \tag{A19}$$

$$\mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] \;=\; q^\pi(s,a) \;-\; r(s,a)\;. \tag{A20}$$

The expected return at time $t=0$ for policy π is

$$v_0^\pi \;=\; \mathrm{E}_\pi\left[G_0\right] \;=\; \mathrm{E}_\pi\left[\sum_{t=0}^{T} R_{t+1}\right]\;, \tag{A21}$$

$$\pi^* \;=\; \operatorname*{argmax}_\pi\; v_0^\pi\;.$$

The agent may start in a particular starting state S_0 which is a random variable. Often S_0 has only one value s_0.

Learning. The **goal** of learning is to find the policy π^* that maximizes the expected future discounted reward (the return) if starting in $t=0$. Thus, the optimal policy π^* is

$$\pi^* \;=\; \operatorname*{argmax}_\pi\; v_0^\pi\;. \tag{A22}$$

We consider two learning approaches for Q-values: Monte Carlo and temporal difference.

Monte Carlo (MC). To estimate $q^\pi(s,a)$, MC computes the arithmetic mean of all observed $(G_t \mid S_t = s, A_t = a)$ in the data. 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

$$(q^\pi)^{i+1}(s_t,a_t) \;=\; (q^\pi)^i(s_t,a_t) \;+\; \alpha\,\left(\sum_t^T r_{t+1} \;-\; (q^\pi)^i(s_t,a_t)\right)\;. \tag{A23}$$

This update is called *constant-α MC* [107].

Temporal difference (TD) methods. TD updates are based on the Bellman equation. If $r(s,a)$ and $\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right]$ have been estimated, the Q-values can be updated according to the Bellman equation:

$$\left(\hat{q}^\pi\right)^{\text{new}}(s,a) \;=\; r(s,a) \;+\; \gamma\,\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right]\;. \tag{A24}$$

The update is applying the Bellman operator with estimates $\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right]$ and $r(s,a)$ to $\hat{q}^\pi$ to obtain $\left(\hat{q}^\pi\right)^{\text{new}}$. The new estimate $\left(\hat{q}^\pi\right)^{\text{new}}$ is closer to the fixed point q^π of the Bellman operator, since the Bellman operator is a contraction (see Section A2.1.3 and Section A2.1.2).
Since the estimates $\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right]$ and $r(s,a)$ are not known, TD methods try to minimize the Bellman residual $B(s,a)$:

$$B(s,a) \;=\; \hat{q}^\pi(s,a) \;-\; r(s,a) \;-\; \gamma\,\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a')\right]\;. \tag{A25}$$

TD methods use an estimate $\hat{B}(s,a)$ of $B(s,a)$ and a learning rate α to make an update

$$\hat{q}(s,a)^{\text{new}} \;\leftarrow\; \hat{q}(s,a) \;+\; \alpha\,\hat{B}(s,a)\;. \tag{A26}$$

For all TD methods $r(s,a)$ is estimated by R_{t+1} and s' by S_{t+1}, while $\hat{q}^\pi(s',a')$ does not change with the current sample, that is, it is fixed for the estimate. However, the sample determines which (s',a') is chosen. The TD methods differ in how they select a'. **SARSA** selects a' by sampling from the policy:

$$\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a')\right] \;\approx\; \hat{q}^\pi(S_{t+1},A_{t+1})$$

and **expected SARSA** averages over selections

$$\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a')\right] \;\approx\; \sum_a \pi(a \mid S_{t+1})\,\hat{q}^\pi(S_{t+1},a).$$

It is possible to estimate $r(s,a)$ separately via an unbiased minimal variance estimator like the arithmetic mean and then perform TD updates with the Bellman error using the estimated $r(s,a)$ [89].

***Q*-learning** [119] is an off-policy TD algorithm which is proved to converge [120, 15], which proofs were later generalized [52, 112]. Q-learning uses

$$\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a')\right] \;\approx\; \max_a \hat{q}(S_{t+1},a)\;. \tag{A27}$$

The learned action-value function q by Q-learning approximates q_* independent of the policy being followed. More precisely, q converges with Q-learning with probability 1 to the optimal q_*. However, the policy still determines which state-action pairs are encountered during learning. The convergence only requires that all action-state pairs are visited and updated infinitely often.

A1.2 Bias-Variance of the Q-Value Estimations

Bias-variance investigation have been done for Q-learning. Grünewälder & Obermayer [35] investigated the bias of temporal difference learning (TD), Monte Carlo estimation (MC), and least-squares temporal difference learning (LSTD). Mannor et al. [67] and O'Donoghue et al. [77] derived bias and variance expressions for updating Q-values,
The true, but unknown, action-value function q^π is the expected future return. We assume to have the data D, which are set of state-action sequences with return, that is a set of episodes with return. Using data D, q^π is estimated by $\hat{q}^\pi = \hat{q}^\pi(D)$, which is an estimate with bias and variance. For bias and variance we have to compute the expectation $\mathrm{E}_D\left[.\right]$ over the data D. The mean squared error (MSE) of an estimator $\hat{q}^\pi(s,a)$ is

$$\mathbf{mse}\,\hat{q}^\pi(s,a) \;=\; \mathrm{E}_D\left[\left(\hat{q}^\pi(s,a) \;-\; q^\pi(s,a)\right)^2\right]\;. \tag{A28}$$

The bias on an estimator $\hat{q}^\pi(s,a)$ is

$$\mathbf{bias}\,\hat{q}^\pi(s,a) \;=\; \mathrm{E}_D\left[\hat{q}^\pi(s,a)\right] \;-\; q^\pi(s,a)\;. \tag{A29}$$

The variance of an estimator $\hat{q}^\pi(s,a)$ is

$$\mathbf{var}\,\hat{q}^\pi(s,a) \;=\; \mathrm{E}_D\left[\left(\hat{q}^\pi(s,a) \;-\; \mathrm{E}_D\left[\hat{q}^\pi(s,a)\right]\right)^2\right]\;. \tag{A30}$$

The bias-variance decomposition of the MSE of an estimator $\hat{q}^\pi(s,a)$ is

$$\mathbf{mse}\,\hat{q}^\pi(s,a) \;=\; \mathbf{var}\,\hat{q}^\pi(s,a) \;+\; \left(\mathbf{bias}\,\hat{q}^\pi(s,a)\right)^2\;. \tag{A31}$$

The bias-variance decomposition of the MSE of an estimator $\hat{q}^\pi$ as a vector is

$$\mathbf{mse}\,\hat{q}^\pi \;=\; \mathrm{E}_D\left[\sum_{s,a}\left(\hat{q}^\pi(s,a) \;-\; q^\pi(s,a)\right)^2\right] \;=\; \mathrm{E}_D\left[\|\hat{q}^\pi \;-\; q^\pi\|^2\right]\;, \tag{A32}$$

$$\mathbf{bias}\,\hat{q}^\pi \;=\; \mathrm{E}_D\left[\hat{q}^\pi\right] \;-\; q^\pi\;, \tag{A33}$$

$$\mathbf{var}\,\hat{q}^\pi \;=\; \mathrm{E}_D\left[\sum_{s,a}\left(\hat{q}^\pi(s,a) \;-\; \mathrm{E}_D\left[\hat{q}^\pi(s,a)\right]\right)^2\right] \;=\; \mathrm{TrVar}_D\left[\hat{q}^\pi\right]\;, \tag{A34}$$

$$\mathbf{mse}\,\hat{q}^\pi \;=\; \mathbf{var}\,\hat{q}^\pi \;+\; \left(\mathbf{bias}\,\hat{q}^\pi\right)^T \mathbf{bias}\,\hat{q}^\pi\;. \tag{A35}$$

A1.2.1 Bias-Variance for MC and TD Estimates of the Expected Return

Monte Carlo (MC) computes the arithmetic mean $\hat{q}^\pi(s,a)$ of the G_t for $(s_t = s, a_t = a)$ over the episodes given by the data.
For **temporal difference (TD)** methods SARSA with learning rate α is

$$\begin{aligned}\left(\hat{q}^\pi\right)^{\mathrm{new}}(s_t,a_t) \;&=\; \hat{q}^\pi(s_t,a_t) \;-\; \alpha\left(\hat{q}^\pi(s_t,a_t) \;-\; R_{t+1} \;-\; \gamma\,\hat{q}^\pi(s_{t+1},a_{t+1})\right)\\ &=\; (1 \;-\; \alpha)\,\hat{q}^\pi(s_t,a_t) \;+\; \alpha\left(R_{t+1} \;+\; \gamma\,\hat{q}^\pi(s_{t+1},a_{t+1})\right)\;.\end{aligned} \tag{A36}$$

Similar updates are used for expected SARSA and Q-learning, where only a_{t+1} is chosen differently. Therefore, for the estimate of $\hat{q}^\pi(s_t,a_t)$, SARSA and Q-learning perform an exponential average of $(R_{t+1} + \gamma\hat{q}^\pi(s_{t+1},a_{t+1}))$. If $\hat{q}^\pi(s_{t+1},a_{t+1})$ is fixed for the updates on some data, then SARSA and Q-learning perform an exponential average of the immediate reward R_{t+1} plus averaging over which $\hat{q}^\pi(s_{t+1},a_{t+1})$ (which (s_{t+1},a_{t+1})) is chosen. In summary, TD methods like SARSA and Q-learning are biased via $\hat{q}^\pi(s_{t+1},a_{t+1})$ and perform an exponential average of the immediate reward R_{t+1} and the next (fixed) $\hat{q}^\pi(s_{t+1},a_{t+1})$.

Bias-Variance for Estimators of the Mean. Both Monte Carlo and TD methods like SARSA or Q-learning estimate $q^\pi(s,a) = \mathrm{E}\left[G_t \mid s,a\right]$, which is the expected future return. The expectations are estimated by either an arithmetic mean with Monte Carlo or an exponential average with TD methods over samples. Therefore we are interested in computing the bias and variance of these estimators of the expectation, in particular we consider the arithmetic mean and the exponential average.
We assume n samples for state-action (s,a); however, the expected number of samples depends on the probability for the number of visits of (s,a) per episode.
Arithmetic mean. For n samples $\{X_1, \ldots, X_n\}$ from a distribution with mean μ and variance σ^2, the variance of the arithmetic mean is

$$\hat{\mu}_n \;=\; \frac{1}{n}\sum_{i=1}^{n} X_i\,, \quad \mathbf{bias}(\hat{\mu}_n) \;=\; 0\,, \quad \mathbf{var}(\hat{\mu}_n) \;=\; \frac{\sigma^2}{n}\,. \tag{A37}$$

The estimation variance of the arithmetic mean is determined by σ^2, the variance of the distribution the samples are drawn from.
Exponential average. For n samples $\{X_1, \ldots, X_n\}$ from a distribution with mean μ and variance σ, the variance of the exponential mean with initial value μ_0 is

$$\hat{\mu}_0 \;=\; \mu_0\,, \quad \hat{\mu}_k \;=\; (1 \,-\, \alpha)\,\hat{\mu}_{k-1} \,+\, \alpha\, X_k\,, \tag{A38}$$

which gives

$$\hat{\mu}_n \;=\; \alpha \sum_{i=1}^{n} (1 \,-\, \alpha)^{n-i}\, X_i \,+\, (1 \,-\, \alpha)^n\, \mu_0\,. \tag{A39}$$

This is an biased estimate since:

$$\begin{aligned}
\mathbf{bias}(\hat{\mu}_n) \;&=\; \mathrm{E}\left[\hat{\mu}_n\right] \,-\, \mu \;=\; \mathrm{E}\left[\alpha \sum_{i=1}^{n} (1 \,-\, \alpha)^{n-i}\, X_i\right] \,+\, (1 \,-\, \alpha)^n\, \mu_0 \,-\, \mu \qquad \text{(A40)} \\
&=\; \alpha \sum_{i=1}^{n} (1 \,-\, \alpha)^{n-i} \mathrm{E}\left[X_i\right] \,+\, (1 \,-\, \alpha)^n\, \mu_0 \,-\, \mu \\
&=\; \mu\, \alpha \sum_{i=0}^{n-1} (1 \,-\, \alpha)^{i} \,+\, (1 \,-\, \alpha)^n\, \mu_0 \,-\, \mu \\
&=\; \mu\, \left(1 \,-\, (1 \,-\, \alpha)^n\right) \,+\, (1 \,-\, \alpha)^n\, \mu_0 \,-\, \mu \;=\; (1 \,-\, \alpha)^n\, (\mu_0 \,-\, \mu)\,.
\end{aligned}$$

Asymptotically ($n \to \infty$) the estimate is unbiased.

The variance is

$$
\begin{aligned}
&\mathbf{var}(\hat{\mu}_n) \;=\; \mathrm{E}\left[\hat{\mu}_n^2\right] \;-\; \mathrm{E}^2\left[\hat{\mu}_n\right] \qquad \text{(A41)}\\
&=\; \mathrm{E}\left[\alpha^2 \sum_{i=1}^{n}\sum_{j=1}^{n}(1\;-\;\alpha)^{n-i}\,X_i\,(1\;-\;\alpha)^{n-j}\,X_j\right]\\
&+\; \mathrm{E}\left[2\,(1\;-\;\alpha)^n\,\mu_0\,\alpha\,\sum_{i=1}^{n}(1\;-\;\alpha)^{n-i}\,X_i\right] \;+\; (1\;-\;\alpha)^{2n}\,\mu_0^2\\
&-\; \left((1\;-\;\alpha)^n\,(\mu_0\;-\;\mu)\;+\;\mu\right)^2\\
&=\; \alpha^2\,\mathrm{E}\left[\sum_{i=1}^{n}(1\;-\;\alpha)^{2(n-i)}\,X_i^2 \;+\; \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(1\;-\;\alpha)^{n-i}\,X_i\,(1\;-\;\alpha)^{n-j}\,X_j\right]\\
&+\; 2\,(1\;-\;\alpha)^n\,\mu_0\,\mu\,\alpha\,\sum_{i=1}^{n}(1\;-\;\alpha)^{n-i} \;+\; (1\;-\;\alpha)^{2n}\,\mu_0^2\\
&-\; \left((1\;-\;\alpha)^n\,\mu_0 \;+\; (1\;-\;(1\;-\;\alpha)^n)\,\mu\right)^2\\
&=\; \alpha^2\,\left(\sum_{i=1}^{n}(1\;-\;\alpha)^{2(n-i)}\,\left(\sigma^2\;+\;\mu^2\right) + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(1\;-\;\alpha)^{n-i}\,(1\;-\;\alpha)^{n-j}\,\mu^2\right)\\
&+\; 2\,(1\;-\;\alpha)^n\,\mu_0\,\mu\,(1\;-\;(1\;-\;\alpha)^n) \;+\; (1\;-\;\alpha)^{2n}\,\mu_0^2\\
&-\; (1\;-\;\alpha)^{2n}\,\mu_0^2 \;-\; 2\,(1\;-\;\alpha)^n\,\mu_0\,(1\;-\;(1\;-\;\alpha)^n)\,\mu \;-\; (1\;-\;(1\;-\;\alpha)^n)^2\,\mu^2\\
&=\; \sigma^2\,\alpha^2\,\sum_{i=0}^{n-1}\left((1\;-\;\alpha)^2\right)^i \;+\; \mu^2\,\alpha^2\,\left(\sum_{i=0}^{n-1}(1\;-\;\alpha)^i\right)^2 \;-\; (1\;-\;(1\;-\;\alpha)^n)^2\,\mu^2\\
&=\; \sigma^2\,\alpha^2\,\frac{1-(1\;-\;\alpha)^{2n}}{1-(1\;-\;\alpha)^2} \;=\; \sigma^2\,\frac{\alpha\,(1\;-\;(1\;-\;\alpha)^{2n})}{2-\alpha}\;.
\end{aligned}
$$

Also the estimation variance of the exponential average is proportional to σ^2, the variance of the distribution the samples are drawn from.
The deviation of random variable X from its mean μ can be analyzed with Chebyshev's inequality. Chebyshev's inequality [12, 110] states that for a random variable X with expected value μ and variance $\tilde{\sigma}^2$ and for any real number $\epsilon > 0$:

$$\Pr\left[|X-\mu| \;\geqslant\; \epsilon\,\tilde{\sigma}\right] \;\leqslant\; \frac{1}{\epsilon^2} \qquad \text{(A42)}$$

or, equivalently,

$$\Pr\left[|X-\mu| \;\geqslant\; \epsilon\right] \;\leqslant\; \frac{\tilde{\sigma}^2}{\epsilon^2}\;. \qquad \text{(A43)}$$

For n samples $\{X_1, \ldots, X_n\}$ from a distribution with expectation μ and variance σ we compute the arithmetic mean $\frac{1}{n}\sum_{i=1}^{n} X_i$. If X is the arithmetic mean, then $\tilde{\sigma}^2 = \sigma^2/n$ and we obtain

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n}X_i \;-\; \mu\right| \;\geqslant\; \epsilon\right] \;\leqslant\; \frac{\sigma^2}{n\,\epsilon^2}\;. \qquad \text{(A44)}$$

Following Grünewälder and Obermayer [35] Bernstein's inequality can be used to describe the deviation of the arithmetic mean (unbiased estimator of μ) from the expectation μ (see Theorem 6 of Gábor Lugosi's lecture notes [65]):

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n}X_i \;-\; \mu\right| \;\geqslant\; \epsilon\right] \;\leqslant\; 2\,\exp\left(-\,\frac{\epsilon^2\,n}{2\,\sigma^2 \;+\; \frac{2\,M\,\epsilon}{3}}\right)\;, \qquad \text{(A45)}$$

where $|X-\mu| < M$.

A1.2.2 Mean and Variance of an MDP Sample of the Return

Since the variance of the estimators of the expectations (arithmetic mean and exponential average) is governed by the variance of the samples, we compute mean and variance of the return estimate $q^\pi(s,a)$. We follow [100, 108, 109] for deriving the mean and variance.
We consider an MDP with finite horizon T, that is, each episode has length T. The finite horizon MDP can be generalized to an MDP with absorbing (terminal) state $s = \mathtt{E}$. We only consider proper policies, that is there exists an integer n such that from any initial state the probability of achieving the terminal state $\mathtt{E}$ after n steps is strictly positive. T is the time to the first visit of the terminal state: $T = \min k \mid s_k = \mathtt{E}$. The return G_0 is:

$$G_0 \;=\; \sum_{k=0}^{T} \gamma^k \; R_{k+1} \; . \tag{A46}$$

The action-value function, the Q-function, is the expected return

$$G_t \;=\; \sum_{k=0}^{T-t} \gamma^k \; R_{t+k+1} \tag{A47}$$

if starting in state $S_t = s$ and action $A_t = a$:

$$q^\pi(s,a) \;=\; \mathrm{E}_\pi \left[G_t \mid s, a \right] \; . \tag{A48}$$

The second moment of the return is:

$$M^\pi(s,a) \;=\; \mathrm{E}_\pi \left[G_t^2 \mid s, a \right] \; . \tag{A49}$$

The variance of the return is:

$$V^\pi(s,a) \;=\; \mathrm{Var}_\pi \left[G_t \mid s, a \right] \;=\; M^\pi(s,a) \;-\; \left(q^\pi(s,a) \right)^2 . \tag{A50}$$

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 A1 gives mean and variance $V^\pi(s,a) = \mathrm{Var}_\pi \left[G_t \mid s, a \right]$ of sampling returns from an MDP.

Theorem A1. *The mean q^π and variance V^π of sampled returns from an MDP are*

$$q^\pi(s,a) = \sum_{s',r} p(s',r \mid s,a) \left(r + \gamma \sum_{a'} \pi(a' \mid s') q^\pi(s',a') \right) = r(s,a) + \gamma \mathrm{E}_{s',a'} \left[q^\pi(s',a') \mid s,a \right],$$
$$V^\pi(s,a) = \mathrm{Var}_r \left[r \mid s,a \right] + \gamma^2 \left(\mathrm{E}_{s',a'} \left[V^\pi(s',a') \mid s,a \right] + \mathrm{Var}_{s',a'} \left[q^\pi(s',a') \mid s,a \right] \right). \tag{A51}$$

Proof. The Bellman equation for Q-values is

$$\begin{aligned} q^\pi(s,a) \;&=\; \sum_{s',r} p(s',r \mid s,a) \left(r \;+\; \gamma \sum_{a'} \pi(a' \mid s') \; q^\pi(s',a') \right) \\ &=\; r(s,a) \;+\; \gamma \, \mathrm{E}_{s',a'} \left[q^\pi(s',a') \mid s,a \right] \; . \end{aligned} \tag{A52}$$

This equation gives the mean if drawing one sample.
We use

$$r(s,a) \;=\; \sum_r r \; p(r \mid s,a) \; , \tag{A53}$$

$$r^2(s,a) \;=\; \sum_r r^2 \; p(r \mid s,a) \; . \tag{A54}$$

For the second moment we obtain [108]

$$
\begin{aligned}
M^\pi(s,a) &= \mathrm{E}_\pi\left[G_t^2 \mid s,a\right] \qquad \text{(A55)}\\
&= \mathrm{E}_\pi\left[\left(\sum_{k=0}^{T-t}\gamma^k R_{t+k+1}\right)^2 \mid s,a\right]\\
&= \mathrm{E}_\pi\left[\left(R_{t+1} + \sum_{k=1}^{T-t}\gamma^k\, R_{t+k+1}\right)^2 \mid s,a\right]\\
&= r^2(s,a) + 2\, r(s,a)\, \mathrm{E}_\pi\left[\sum_{k=1}^{T-t}\gamma^k\, R_{t+k+1} \mid s,a\right]\\
&+ \mathrm{E}_\pi\left[\left(\sum_{k=1}^{T-t}\gamma^k\, R_{t+k+1}\right)^2 \mid s,a\right]\\
&= r^2(s,a) + 2\gamma\, r(s,a) \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q^\pi(s',a')\\
&+ \gamma^2 \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, M^\pi(s',a')\,.\\
&= r^2(s,a) + 2\gamma\, r(s,a)\, \mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] + \gamma^2\, \mathrm{E}_{s',a'}\left[M^\pi(s',a') \mid s,a\right]\,.
\end{aligned}
$$

For the variance we obtain:

$$
\begin{aligned}
V^\pi(s,a) &= M^\pi(s,a) - \left(q^\pi(s,a)\right)^2 \qquad \text{(A56)}\\
&= r^2(s,a) - \left(r(s,a)\right)^2 + \gamma^2\, \mathrm{E}_{s',a'}\left[M^\pi(s',a') \mid s,a\right] - \gamma^2\, \mathrm{E}^2_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\\
&= \mathrm{Var}_r\left[r \mid s,a\right] + \gamma^2 \left(\mathrm{E}_{s',a'}\left[M^\pi(s',a') - \left(q^\pi(s',a')\right)^2 \mid s,a\right]\right.\\
&\left. - \mathrm{E}^2_{s',a'}\left[q^\pi(s',a') \mid s,a\right] + \mathrm{E}_{s',a'}\left[\left(q^\pi(s',a')\right)^2 \mid s,a\right]\right)\\
&= \mathrm{Var}_r\left[r \mid s,a\right] + \gamma^2 \left(\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] + \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right)\,.
\end{aligned}
$$

□

For deterministic reward, that is, $\mathrm{Var}_r\left[r \mid s,a\right] = 0$, the corresponding result is given as Equation (4) in Sobel 1982 [100] and as Proposition 3.1 (c) in Tamar et al. 2012 [108].
For temporal difference (TD) learning, the next Q-values are fixed to $\hat{q}^\pi(s',a')$ when drawing a sample. Therefore TD is biased, that is, both for SARSA and Q-learning are biased. During learning with according updates of Q-values $\hat{q}^\pi(s',a')$ approaches $q^\pi(s',a')$ and the bias is reduced. However, this reduction of the bias is exponentially small in the number of time steps between reward and updated Q-values as we will see later. The reduction of the bias is exponentially small for eligibility traces, too.
The variance recursion Eq. (A51) is meant for drawing a sample and consists of three parts:

- (1) the immediate variance $\mathrm{Var}_r\left[r \mid s,a\right]$ of the immediate reward stemming from the probabilistic reward $p(r \mid s,a)$,
- (2) the local variance $\gamma^2 \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]$ from state transitions $p(s' \mid s,a)$ and new actions $\pi(a' \mid s')$,
- (3) the expected variance $\gamma^2 \mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right]$ of the next Q-values.

For different settings these parts may be zero:

- (1) the immediate variance $\mathrm{Var}_r\left[r \mid s,a\right]$ is zero for deterministic immediate reward,
- (2) the local variance $\gamma^2 \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]$ is zero for (i) deterministic state transitions and deterministic policy and for (ii) $\gamma = 0$ (only immediate reward),
- (3) the expected variance $\gamma^2 \mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right]$ of the next Q-values is zero for (i) temporal difference (TD) learning, since the next Q-values are fixed and set to their current estimates (if just one sample is drawn) and for (ii) $\gamma = 0$ (only immediate reward).

The local variance $\mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]$ is the variance of a linear combination of Q-values weighted by a multinomial distribution $\sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')$. The local variance is

$$\mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] = \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s') \left(q^\pi(s',a')\right)^2 \tag{A57}$$

$$- \left(\sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q^\pi(s',a')\right)^2 .$$

This result is Equation (6) in Sobel 1982 [100]. Sobel derived these formulas also for finite horizons and an analog formula if the reward depends also on the next state, that is, for $p(r \mid s,a,s')$.
Monte Carlo uses the accumulated future rewards for updates, therefore its variance is given by the recursion in Eq. (A51). Temporal difference, however, fixes $q^\pi(s',a')$ to the current estimates $\hat{q}^\pi(s',a')$, which do not change in the current episode. Therefore TD has $\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] = 0$ and only the local variance $\mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]$ is present. For n-step TD, the recursion in Eq. (A51) must be applied $(n-1)$ times then the expected next variances are zero since the future reward is estimated by $\hat{q}^\pi(s',a')$.
Delayed rewards. For TD and delayed reward, information on new data is only captured by the last step of an episode that receives a reward. This reward is used to update the estimates of the Q-values of the last state $\hat{q}(s_T,a_T)$. Subsequently, the reward information is propagated backward one step for each sample via the estimates $\hat{q}$. The drawn samples (state action sequences) determine where information is propagated back. Therefore delayed reward introduces a large bias for TD over a long period of time, since the estimates $\hat{q}(s,a)$ need a long time to reach their true Q-values.
For Monte Carlo with delayed reward the immediate variance $\mathrm{Var}_r\left[r \mid s,a\right] = 0$ except for the last step of the episode. The delayed reward increases the variance of more Q-values according to Eq. (A51) when propagated back than immediate rewards.
Sample Distribution Used by Temporal Difference and Monte Carlo. Monte Carlo (MC) sampling uses the true mean and true variance. The mean which is used by Monte Carlo is

$$q^\pi(s,a) = r(s,a) + \gamma\, \mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] . \tag{A58}$$

The variance which is used by Monte Carlo is

$$V^\pi(s,a) = \mathrm{Var}_r\left[r \mid s,a\right] + \gamma^2 \left(\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] + \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right) . \tag{A59}$$

Temporal difference (TD) methods replace $q^\pi(s',a')$ by $\hat{q}^\pi(s',a')$ which does not depend on the sample that is drawn. The mean which is used by temporal difference is

$$q^\pi(s,a) = r(s,a) + \gamma\, \mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right] . \tag{A60}$$

This mean is biased by

$$\gamma \left(\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right] - \mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right) . \tag{A61}$$

The variance which is used by temporal difference is

$$V^\pi(s,a) = \mathrm{Var}_r\left[r \mid s,a\right] + \gamma^2\, \mathrm{Var}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right] , \tag{A62}$$

since $V^\pi(s',a') = 0$ if $\hat{q}^\pi(s',a')$ is used instead of the future reward from the sample. The variance of TD is smaller than for MC, since variances are not propagated backward.

A1.2.3 TD Corrects Bias Exponentially Slowly With Reward Delay

Temporal Difference. We show that TD has an exponential small update for delayed reward which fades exponentially with the number of delay steps. For Q-learning with learning rates $1/i$ at the ith update leads to an arithmetic mean as estimate, which was shown to be exponentially slow [7]. For fixed learning rate The information of the delayed reward has to be propagated back either through the Bellman error or via eligibility traces. We first consider backpropagation of reward information via the Bellman error. For each episode the reward information is propagated back one step at visited state-action pairs via the TD update rule. We denote by q^i the Q-values of episode i and assume that state action pairs (s_t,a_t) are at most visited once. We consider the update of a $q^i(s_t,a_t)$ of a state-action pair (s_t,a_t) that is visited at time t in the ith episode:

$$q^{i+1}(s_t,a_t) = q^i(s_t,a_t) + \alpha\, \delta_t , \tag{A63}$$

$$\delta_t = r_{t+1} + \max_{a'} q^i(s_{t+1},a') - q^i(s_t,a_t) \quad (Q\text{-learning}) \tag{A64}$$

$$\delta_t = r_{t+1} + \sum_{a'} \pi(a' \mid s_{t+1})\, q^i(s_{t+1},a') - q^i(s_t,a_t) \quad (\text{expected SARSA}) . \tag{A65}$$

Temporal Difference with Eligibility Traces. For an episode eligibility traces have been introduced to propagate back reward information and are now standard for TD(λ) [99]. However, the eligibility traces are exponentially decaying when propagated back. The accumulated trace is defined as [99]:

$$e_{t+1}(s,a) \;=\; \begin{cases} \gamma\,\lambda\,e_t(s,a) & \text{for } s \neq s_t \text{ or } a \neq a_t\;, \\ \gamma\,\lambda\,e_t(s,a) \;+\; 1 & \text{for } s = s_t \text{ and } a = a_t\;. \end{cases} \tag{A66}$$

while the replacing trace is defined as [99]:

$$e_{t+1}(s,a) \;=\; \begin{cases} \gamma\,\lambda\,e_t(s,a) & \text{for } s \neq s_t \text{ or } a \neq a_t\;, \\ 1 & \text{for } s = s_t \text{ and } a = a_t\;. \end{cases} \tag{A67}$$

We use the naive $Q(\lambda)$, where eligibility traces are not set to zero. In contrast, Watkins' $Q(\lambda)$ [119] zeros out eligibility traces after non-greedy actions, that is, if not the $\max_a$ is chosen. Therefore the decay is even stronger for Watkin's $Q(\lambda)$. Another eligibility trace method is Peng's $Q(\lambda)$ [79] which also does not zero out eligibility traces.
The next Theorem A2 states that TD has an exponential decay for Q-value updates in an MDP with delayed reward even for eligibility traces. Thus, for delayed reward TD required exponentially many updates to correct the bias, where the number of updates is exponential in the delay steps.

Theorem A2. *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 $q^i(s_{T-i},a_{T-i}) = \alpha^{i+1} r^1_{T+1}$, where r^1_{T+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.*

Proof. If we assume that Q-values are initialized with zero, then $q^0(s_t,a_t) = 0$ for all (s_t,a_t). For delayed reward we have and $r_t = 0$ for $t \leqslant T$. The Q-value $q(s_{T-i},a_{T-i})$ at time $T-i$ can receive an update the first time at episode i. Since all Q-values have been initialized by zero, the update is

$$q^i(s_{T-i},a_{T-i}) \;=\; \alpha^{i+1}\,r^1_{T+1}\;, \tag{A68}$$

where r^1_{T+1} is the reward at $T+1$ for episode 1.
We move on to eligibility traces, where the update for a state s is

$$q_{t+1}(s,a) \;=\; q_t(s,a) \;+\; \alpha\,\delta_t\,e_t(s,a)\;, \tag{A69}$$

$$\delta_t \;=\; r_{t+1} \;+\; \max_{a'} q_t(s_{t+1},a') \;-\; q_t(s_t,a_t)\;. \tag{A70}$$

If states are not revisited, for a visit of state s_t at time t, the eligibility trace at time $t+k$ is:

$$e_{t+k}(s_t,a_t) \;=\; \big(\gamma\,\lambda\big)^k\;. \tag{A71}$$

If all δ_{t+i} are zero except for δ_{t+k}, then the update of $q(s,a)$ is

$$q_{t+k+1}(s,a) \;=\; q_{t+k}(s,a) \;+\; \alpha\,\delta_{t+k}\,e_{t+k}(s,a) \;=\; q_{t+k}(s,a) \;+\; \alpha\,\big(\gamma\,\lambda\big)^k\,\delta_{t+k}\;. \tag{A72}$$

□

A learning rate of $\alpha = 1$ does not work since it would mean to forget all previous learned estimates and there is no averaging over episodes. Since $\alpha < 1$, we observe exponential decay backwards in time for online updates.

A1.2.4 MC Affects the Variance of Exponentially Many Estimates with Delayed Reward

The variance for Monte Carlo is

$$V^\pi(s,a) \;=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \gamma^2\;\left(\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right)\;. \tag{A73}$$

This is a Bellman equation for the variance.
For undiscounted reward $\gamma = 1$ we have

$$V^\pi(s,a) \;=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\;. \tag{A74}$$

If we define the "on-site" variance ω

$$\omega(s,a) \;=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\;, \tag{A75}$$

then we have

$$V^\pi(s,a) \;=\; \omega(s,a) \;+\; \mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] \;. \tag{A76}$$

This is the solution of the general formulation Eq. (A172) of the Bellman operator equation. The Bellman operator is defined for any variance V component-wise as

$$\mathrm{T}^\pi\left[V\right](s,a) \;=\; \omega(s,a) \;+\; \mathrm{E}_{s',a'}\left[V(s',a') \mid s,a\right] \;. \tag{A77}$$

According to the results in Subsection A2.1 for proper policies π an unique fixed point V^π exists:

$$V^\pi \;=\; \mathrm{T}^\pi\left[V^\pi\right] \tag{A78}$$

$$V^\pi \;=\; \lim_{k\to\infty} \left(\mathrm{T}^\pi\right)^k V \;, \tag{A79}$$

where V is any initial variance. In Subsection A2.1 it was shown that the operator T^π is continuous, monotonically increasing (component-wise larger or smaller), and a contraction mapping for a weighted sup-norm.
If we define the operator T^π as depending on the on-site variance ω, that is T^π_ω, then it is monotonically in ω. For $\omega > \tilde{\omega}$ component-wise we have:

$$\begin{aligned} &\mathrm{T}^\pi_\omega\left[q\right](s,a) \;-\; \mathrm{T}^\pi_{\tilde{\omega}}\left[q\right](s,a) \\ &=\; \left(\omega(s,a) \;+\; \mathrm{E}_{s',a'}\left[q(s',a')\right]\right) \;-\; \left(\tilde{\omega}(s,a) \;+\; \mathrm{E}_{s',a'}\left[q(s',a')\right]\right) \\ &=\; \omega(s,a) \;-\; \tilde{\omega}(s,a) \;\geqslant\; 0 \;. \end{aligned} \tag{A80}$$

It follows for the fixed points V^π of T^π_ω and $\widetilde{V}^\pi$ of $\mathrm{T}^\pi_{\tilde{\omega}}$:

$$V^\pi(s,a) \;\geqslant\; \widetilde{V}^\pi(s,a) \;. \tag{A81}$$

Therefore if

$$\begin{aligned} \omega(s,a) \;&=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] \;\geqslant \\ \tilde{\omega}(s,a) \;&=\; \widetilde{\mathrm{Var}}_r\left[r \mid s,a\right] \;+\; \widetilde{\mathrm{Var}}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] \end{aligned} \tag{A82}$$

then

$$V^\pi(s,a) \;\geqslant\; \widetilde{V}^\pi(s,a) \;. \tag{A83}$$

In Stephen Patek's PhD thesis [78] Lemma 5.1 on page 88-89 and proof thereafter state that if $\tilde{\omega}(s,a) = \omega(s,a) - \lambda$ then the solution $\widetilde{V}^\pi$ is continuous and decreasing in λ.
From above inequality follows that

$$\begin{aligned} &V^\pi(s,a) \;-\; \widetilde{V}^\pi(s,a) \;=\; \left(\mathrm{T}^\pi_\omega V^\pi\right)(s,a) \;-\; \left(\mathrm{T}^\pi_{\tilde{\omega}} \widetilde{V}^\pi\right)(s,a) \\ &=\; \omega(s,a) \;-\; \tilde{\omega}(s,a) \;+\; \mathrm{E}_{s',a'}\left[V^\pi(s',a') \;-\; \widetilde{V}^\pi(s',a') \mid s,a\right] \\ &\geqslant\; \omega(s,a) \;-\; \tilde{\omega}(s,a) \;. \end{aligned} \tag{A84}$$

Time-Agnostic States. We defined a Bellman operator as

$$\begin{aligned} \mathrm{T}^\pi\left[\boldsymbol{V}^\pi\right](s,a) \;&=\; \omega(s,a) \;+\; \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s') \, \boldsymbol{V}^\pi(s',a') \\ &=\; \omega(s,a) \;+\; \left(\boldsymbol{V}^\pi\right)^T \boldsymbol{p}(s,a) \;, \end{aligned} \tag{A85}$$

where $\boldsymbol{V}^\pi$ is the vector with value $V^\pi(s',a')$ at position (s',a') and $\boldsymbol{p}(s,a)$ is the vector with value $p(s' \mid s,a)\pi(a' \mid s')$ at position (s',a').
The fixed point equation is known as the *Bellman equation*. In vector and matrix notation the Bellman equation is

$$\mathrm{T}^\pi\left[\boldsymbol{V}^\pi\right] \;=\; \boldsymbol{\omega} \;+\; \boldsymbol{P}\,\boldsymbol{V}^\pi \;. \tag{A86}$$

where $\boldsymbol{P}$ is the row-stochastic matrix with $p(s' \mid s,a)\pi(a' \mid s')$ at position $((s,a),(s',a'))$. We assume that the set of state-actions $\{(s,a)\}$ is equal to the set of next state-actions $\{(s',a')\}$, therefore $\boldsymbol{P}$ is a square row-stochastic matrix. This Bellman operator has the same characteristics as the Bellman operator for the action-value function q^π.

Since $\boldsymbol{P}$ is a row-stochastic matrix, the Perron-Frobenius theorem says that (1) $\boldsymbol{P}$ has as largest eigenvalue 1 for which the eigenvector corresponds to the steady state and (2) the absolute value of each (complex) eigenvalue is smaller equal 1. Only the eigenvector to the eigenvalue 1 has only positive real components.
Equation 7 of Bertsekas and Tsitsiklis, 1991, [10] states

$$\left(\mathrm{T}^{\pi}\right)^{t}\left[\boldsymbol{V}^{\pi}\right] \;=\; \sum_{k=0}^{t-1} \boldsymbol{P}^{k}\,\boldsymbol{\omega} \;+\; \boldsymbol{P}^{t}\,\boldsymbol{V}^{\pi}\;. \tag{A87}$$

Applying the operator T^{π} recursively t times can be written as [10]:

$$\left(\mathrm{T}^{\pi}\right)^{t}\left[\boldsymbol{V}^{\pi}\right] \;=\; \sum_{k=0}^{t-1} \boldsymbol{P}^{k}\,\boldsymbol{\omega} \;+\; \boldsymbol{P}^{t}\,\boldsymbol{V}^{\pi}\;. \tag{A88}$$

In particular for $\boldsymbol{V}^{\pi} = \mathbf{0}$ we obtain

$$\left(\mathrm{T}^{\pi}\right)^{t}\left[\mathbf{0}\right] \;=\; \sum_{k=0}^{t-1} \boldsymbol{P}^{k}\,\boldsymbol{\omega}\;. \tag{A89}$$

For finite horizon MDPs, the values $\boldsymbol{V}^{\pi} = \mathbf{0}$ are correct for time step $T+1$ since there is no reward for $t > T+1$. Therefore the "backward induction algorithm" [82, 83] gives the correct solution:

$$\boldsymbol{V}^{\pi} \;=\; \left(\mathrm{T}^{\pi}\right)^{T}\left[\mathbf{0}\right] \;=\; \sum_{k=0}^{T-1} \boldsymbol{P}^{k}\,\boldsymbol{\omega}\;. \tag{A90}$$

The product of square stochastic matrices is a stochastic matrix, therefore $\boldsymbol{P}^{k}$ is a stochastic matrix. Perron-Frobenius theorem states that the spectral radius $\rho(\boldsymbol{P}^{k})$ of the stochastic matrix $\boldsymbol{P}^{k}$ is one: $\rho(\boldsymbol{P}^{k}) = 1$. Furthermore the largest eigenvalue is 1 and all eigenvalues have absolute value smaller equal one. Therefore, $\boldsymbol{\omega}$ can have large influence on $\boldsymbol{V}^{\pi}$ at every time step.

Time-Aware States. Next we consider time-aware MDPs, therefore transitions occur only from states s_t to s_{t+1}. The transition matrix from states s_t to s_{t+1} is denoted by $\boldsymbol{P}_t$. We assume that $\boldsymbol{P}_t$ are row-stochastic matrices which are rectangular, that is $\boldsymbol{P}_t \in \mathbb{R}^{m \times n}$.

Definition A1. *A row-stochastic matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ has non-negative entries and the entries of each row sum up to one.*

It is known that the product of square stochastic matrices $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is a stochastic matrix. We show in next theorem that this holds also for rectangular matrices.

Lemma A1. *The product $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$ with $\boldsymbol{C} \in \mathbb{R}^{m \times k}$ of a row-stochastic matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and a row-stochastic matrix $\boldsymbol{B} \in \mathbb{R}^{n \times k}$ is row-stochastic.*

Proof. All entries of $\boldsymbol{C}$ are non-negative since they are sums and products of non-negative entries of $\boldsymbol{A}$ and $\boldsymbol{B}$. The row-entries of $\boldsymbol{C}$ sum up to one:

$$\sum_{k} C_{ik} \;=\; \sum_{k} \sum_{j} A_{ij}\,B_{jk} \;=\; \sum_{j} A_{ij} \sum_{k} B_{jk} \;=\; \sum_{j} A_{ij} \;=\; 1\;. \tag{A91}$$

□

We will use the ∞-norm and the 1-norm of a matrix, which are defined based on the ∞-norm $\|\boldsymbol{x}\|_{\infty} = \max_i |x_i|$ and 1-norm $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$ of a vector $\boldsymbol{x}$.

Definition A2. *The ∞-norm of a matrix is the maximum absolute row sum:*

$$\|\boldsymbol{A}\|_{\infty} \;=\; \max_{\|\boldsymbol{x}\|_{\infty}=1} \|\boldsymbol{A}\,\boldsymbol{x}\|_{\infty} \;=\; \max_{i} \sum_{j} |A_{ij}|\;. \tag{A92}$$

The 1-norm of a matrix is the maximum absolute column sum:

$$\|\boldsymbol{A}\|_{1} \;=\; \max_{\|\boldsymbol{x}\|_{1}=1} \|\boldsymbol{A}\,\boldsymbol{x}\|_{1} \;=\; \max_{j} \sum_{i} |A_{ij}|\;. \tag{A93}$$

The statements of next theorem are known as Perron-Frobenius theorem for square stochastic matrices $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, e.g. that the spectral radius ρ is $\rho(\boldsymbol{A}) = 1$. We extend the theorem a ∞-norm equals one property for rectangular stochastic matrices $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.

Lemma A2 (Perron-Frobenius)**.** *If $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is a row-stochastic matrix, then*

$$\|\boldsymbol{A}\|_\infty \; = \; 1 \; , \quad \left\|\boldsymbol{A}^T\right\|_1 \; = \; 1 \; , \; \textit{and for } n = m \quad \rho(\boldsymbol{A}) \; = \; 1 \; . \tag{A94}$$

Proof. $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is a row-stochastic matrix, therefore $A_{ij} = |A_{ij}|$. Furthermore the rows of $\boldsymbol{A}$ sum up to one. Thus, $\|\boldsymbol{A}\|_\infty = 1$. Since the column sums of $\boldsymbol{A}^T$ are the row sums of $\boldsymbol{A}$, it follows that $\left\|\boldsymbol{A}^T\right\|_1 = 1$.

For square stochastic matrices, that is $m = n$, Gelfand's Formula (1941) says that for any matrix norm $\|.\|$, we have for the spectral norm $\rho(\boldsymbol{A})$ of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$:

$$\rho(\boldsymbol{A}) \; = \; \lim_{k \to \infty} \left\|\boldsymbol{A}^k\right\|^{1/k} \; . \tag{A95}$$

Since the product of row-stochastic matrices is a row-stochastic matrix, $\boldsymbol{A}^k$ is a row-stochastic matrix. Consequently $\left\|\boldsymbol{A}^k\right\|_\infty = 1$ and $\left\|\boldsymbol{A}^k\right\|_\infty^{1/k} = 1$. Therefore the spectral norm $\rho(\boldsymbol{A})$ of a row-stochastic matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is

$$\rho(\boldsymbol{A}) \; = \; 1 \; . \tag{A96}$$

The last statement follows from Perron-Frobenius theorem, which says that the spectral radius of $\boldsymbol{P}$ is 1. □

Using random matrix theory, we can guess how much the spectral radius of a rectangular matrix deviates from that of a square matrix. Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be a matrix whose entries are independent copies of some random variable with zero mean, unit variance, and finite fourth moment. The Marchenko-Pastur quarter circular law for rectangular matrices says that for $n = m$ the maximal singular value is $2\sqrt{m}$ [69]. Asymptotically we have for the maximal singular value $s_{\max}(\boldsymbol{A}) \propto \sqrt{m} + \sqrt{n}$ [90]. A bound on largest singular value is [101]:

$$s_{\max}^2(\boldsymbol{A}) \; \leqslant \; (\sqrt{m} \; + \; \sqrt{n})^2 \; + \; O(\sqrt{n} \; \log(n)) \text{ a.s.} \tag{A97}$$

Therefore a rectangular matrix modifies the largest singular value by a factor of $a = 0.5(1 + \sqrt{n/m})$ compared to a $m \times m$ square matrix.

In the case that the states are time aware, transitions are only from states s_t to s_{t+1}. The transition matrix from states s_t to s_{t+1} is denoted by $\boldsymbol{P}_t$.

States affected by the on-site variance $\boldsymbol{\omega}_k$ (reachable states). Typically states in s_t have only few predecessor states in s_{t-1} compared to N_{t-1}, the number of possible states in s_{t-1}. Only for those states in s_{t-1} the transition probability to the state in s_t is larger than zero. That is, each $i \in s_{t+1}$ has only few $j \in s_t$ for which $p_t(i \mid j) > 0$. We now want to know how many states have increased variance due to $\boldsymbol{\omega}_k$, that is how many states are affected by $\boldsymbol{\omega}_k$. In a general setting, we assume random connections.

Let N_t be the number of all states s_t that are reachable after t time steps of an episode. $\bar{N} = 1/k \sum_{t=1}^k N_t$ is the arithmetic mean of the N_t. Let c_t be the average connectivity of a state in s_t to states in s_{t-1}. $\bar{c} = \left(\prod_{t=1}^k c_t\right)^{1/k}$ is the geometric mean of the c_t. Let n_t be the number of states in s_t that are affected by the on-site variance $\boldsymbol{\omega}_k$ at time k for $t \leqslant k$. The number of states affected by $\boldsymbol{\omega}_k$ is $a_k = \sum_{t=0}^k n_t$. We assume that $\boldsymbol{\omega}_k$ has only one component larger than zero, that is, only one state at time $t = k$ is affected: $n_k = 1$. The number of affecting edges from s_t to s_{t-1} is $c_t n_t$. However, states in s_{t-1} may be affected multiple times by different affected states in s_t. Figure A1 shows examples of how affected states affect states in a previous time step. The left panel shows no overlap since affected states in s_{t-1} connect only to one affected state in s_t. The right panel shows some overlap since affected states in s_{t-1} connect to multiple affected states in s_t.

Next theorem says that the on-site variance $\boldsymbol{\omega}_k$ can have large affect on the variance of each previous state-action. Furthermore, for small k the number of affected states growths exponentially, while for large k it growths only linearly after some time $\hat{t}$. Figure A2 shows the function which determines how much a_k growths with k.

Theorem A3. *For $t \leqslant k$, $\boldsymbol{\omega}_k$ 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 the on-site variance $\boldsymbol{\omega}_k$ is*

$$a_k \; = \; \sum_{t=0}^{k} \left(1 \; - \; \left(1 \; - \; \frac{c_t}{N_{t-1}}\right)^{n_t}\right) \; N_{t-1} \; . \tag{A98}$$

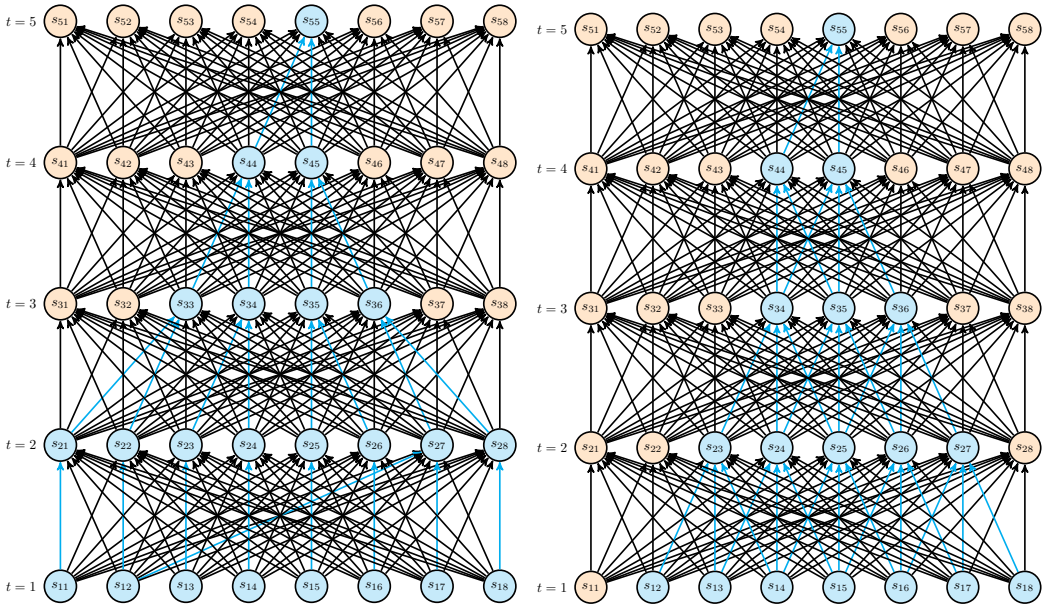


Figure A1: Examples of how affected states (cyan) affect states in a previous time step (indicated by cyan edges) starting with $n_5 = 1$ (one affected state). The left panel shows no overlap since affected states in s_{t-1} connect only to one affected state in s_t. The right panel shows some overlap since affected states in s_{t-1} connect to multiple affected states in s_t.

Proof. The "backward induction algorithm" [82, 83] gives with $\boldsymbol{V}_{T+1}^{\pi} = \mathbf{0}$ and on-site variance $\boldsymbol{\omega}_{T+1} = \mathbf{0}$:

$$\boldsymbol{V}_t^{\pi} \;=\; \sum_{k=t}^{T} \prod_{\tau=t}^{k-1} \boldsymbol{P}_{\tau}\, \boldsymbol{\omega}_k \;, \tag{A99}$$

where we define $\prod_{\tau=t}^{t-1} \boldsymbol{P}_{\tau} = \boldsymbol{I}$ and $[\boldsymbol{\omega}_k]_{(s_k,a_k)} = \omega(s_k, a_k)$.
Since the product of two row-stochastic matrices is a row-stochastic matrix according to Lemma A1, $\boldsymbol{P}_{t\leftarrow k} = \prod_{\tau=t}^{k-1} \boldsymbol{P}_{\tau}$ is a row-stochastic matrix. Since $\|\boldsymbol{P}_{t\leftarrow k}\|_{\infty} = 1$ according to Lemma A2, each on-site variance $\boldsymbol{\omega}_k$ with $t \leqslant k$ can have large effect on $\boldsymbol{V}_t^{\pi}$.
Using the the row-stochastic matrices $\boldsymbol{P}_{t\leftarrow k}$, we can reformulate the variance:

$$\boldsymbol{V}_t^{\pi} \;=\; \sum_{k=t}^{T} \boldsymbol{P}_{t\leftarrow k}\, \boldsymbol{\omega}_k \;, \tag{A100}$$

with $\|\boldsymbol{P}_{t\leftarrow k}\|_{\infty} = 1$. The on-site variance $\boldsymbol{\omega}_k$ at step k increases all variances $\boldsymbol{V}_t^{\pi}$ with $t \leqslant k$.
Next we proof the second part of the theorem, which considers the growth of a_k. To compute a_k we first have to know the n_t. For computing n_{t-1} from n_t, we want to know how many states are affected in s_{t-1} if n_t states are affected in s_t. The answer to this question is the expected coverage when searching a document collection using a set of independent computers [14]. We follow the approach of Cox et al. [14]. The minimal number of affected states in s_{t-1} is c_t, where each of the c_t affected states in s_{t-1} connects to each of the n_t states in s_t (maximal overlap). The maximal number of affected states in s_{t-1} is $c_t n_t$, where each affected state in s_{t-1} connects to only one affected state in s_t (no overlap). We consider a single state in s_t. The probability of a state in s_{t-1} being connected to this single state in s_t is c_t/N_{t-1} and being not connected to this state in s_t is $1 - c_t/N_{t-1}$. The probability of a state in s_{t-1} being not connected to any of the n_t affected states in s_t is

$$\left(1 \;-\; \frac{c_t}{N_{t-1}}\right)^{n_t} . \tag{A101}$$

The probability of a state in s_{t-1} being at least connected to one of the n_t affected states in s_t is

$$1 \;-\; \left(1 \;-\; \frac{c_t}{N_{t-1}}\right)^{n_t} . \tag{A102}$$

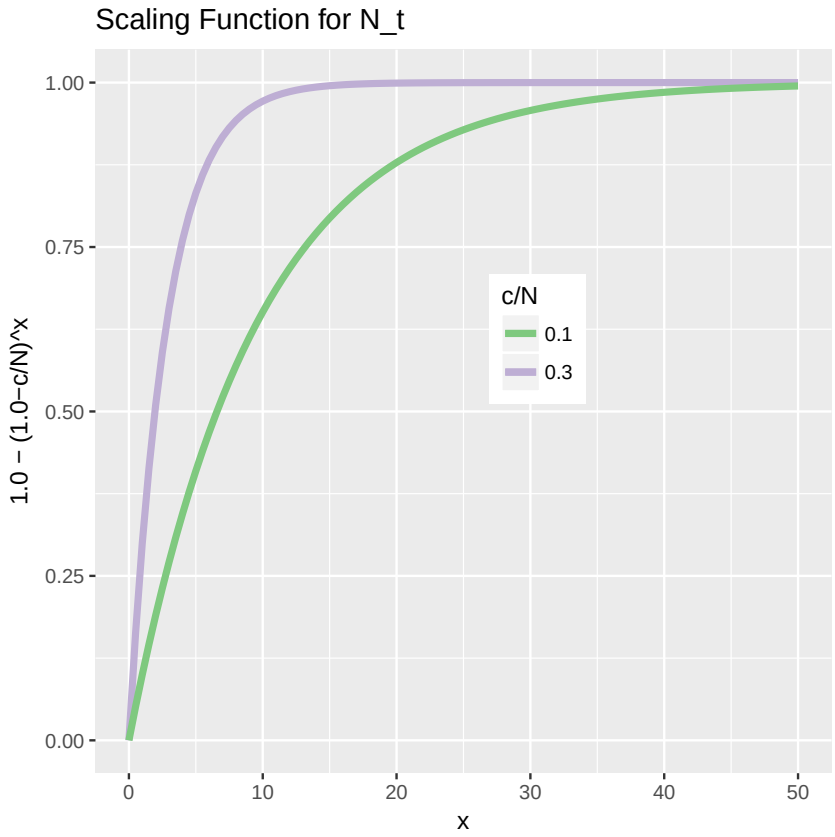


Figure A2: The function $\left(1-\left(1-\frac{c_t}{N_{t-1}}\right)^{n_t}\right)$ which scales N_{t-1} in Theorem A3. This function determines the growth of a_k, which is at the beginning exponentially and then linearly when the function approaches 1.

Thus, expected number of distinct states in s_{t-1} being connected to one of the n_t affected states in s_t is

$$n_{t-1} \;=\; \left(1 \;-\; \left(1 \;-\; \frac{c_t}{N_{t-1}}\right)^{n_t}\right)\; N_{t-1}\,. \tag{A103}$$

The number a_k of affected states by $\boldsymbol{\omega}_k$ is

$$a_k \;=\; \sum_{t=0}^{k} \left(1 \;-\; \left(1 \;-\; \frac{c_t}{N_{t-1}}\right)^{n_t}\right)\; N_{t-1}\,. \tag{A104}$$

□

Corollary A1. *For small k, the number a_k of states affected by the on-site variance $\boldsymbol{\omega}_k$ at step k growths exponentially with k with a factor of $\bar{c}$:*

$$a_k \;>\; \bar{c}^{\,k}\,. \tag{A105}$$

For large k and after some time $t > \hat{t}$, the number a_k of states affected by $\boldsymbol{\omega}_k$ growths linearly with k with a factor of $\bar{N}$:

$$a_k \;\approx\; a_{\hat{t}-1} \;+\; (k-\hat{t}+1)\;\bar{N}\,. \tag{A106}$$

Proof. For small n_t with $\frac{c_t n_t}{N_{t-1}} \ll 1$, we have

$$\left(1 \;-\; \frac{c_t}{N_{t-1}}\right)^{n_t} \;\approx\; 1 \;-\; \frac{c_t\; n_t}{N_{t-1}}\,, \tag{A107}$$

thus

$$n_{t-1} \;\approx\; c_t \, n_t \; . \tag{A108}$$

For large N_{t-1} compared to the number of connection c_t of a single state in s_t to states in s_{t-1}, we have the approximation

$$\left(1 \;-\; \frac{c_t}{N_{t-1}}\right)^{n_t} \;=\; \left(\left(1 \;+\; \frac{-c_t}{N_{t-1}}\right)^{N_{t-1}}\right)^{n_t/N_{t-1}} \;\approx\; \exp(-(c_t \, n_t)/N_{t-1}) \; . \tag{A109}$$

We obtain

$$n_{t-1} \;=\; (1 \;-\; \exp(-(c_t \, n_t)/N_{t-1})) \; N_{t-1} \; . \tag{A110}$$

For small n_t we again have

$$n_{t-1} \;\approx\; c_t \, n_t \; . \tag{A111}$$

Therefore, for $k - t$ small, we have

$$n_t \;\approx\; \prod_{\tau=t}^{k} c_\tau \;\approx\; \bar{c}^{k-t} \; . \tag{A112}$$

Thus, for small k the number a_k of by $\boldsymbol{\omega}_k$ affected states is

$$a_k \;=\; \sum_{t=0}^{k} n_t \;\approx\; \sum_{t=0}^{k} \bar{c}^{k-t} \;=\; \sum_{t=0}^{k} \bar{c}^{t} \;=\; \frac{\bar{c}^{k+1} - 1}{\bar{c} - 1} \;>\; \bar{c}^{k} \; . \tag{A113}$$

Consequently, for small k the number a_k of affected states by $\boldsymbol{\omega}_k$ growths exponentially with k by a factor of $\bar{c}$.
For large k at some time $t > \hat{t}$, n_t has grown so that $c_t n_t > N_{t-1}$ leading to $\exp(-(c_t n_t)/N_{t-1}) \approx 0$, thus

$$n_t \;\approx\; N_t \; . \tag{A114}$$

Therefore

$$a_k \;-\; a_{\hat{t}-1} \;=\; \sum_{t=\hat{t}}^{k} n_t \;\approx\; \sum_{t=\hat{t}}^{k} N_t \;\approx\; (k - \hat{t} + 1) \, \bar{N} \; . \tag{A115}$$

Therefore for large k the number a_k of affected states by $\boldsymbol{\omega}_k$ growths linearly with k by a factor of $\bar{N}$. □

Consequently, we aim for decreasing the on-site variance $\boldsymbol{\omega}_k$ for large k, in order to reduce the variance. In particular we want to avoid delayed reward and provide the reward as soon as possible in each episode. Our goal is to give the reward as early as possible in each episode to reduce the variance of action-values that are affected by late rewards and their associated immediate and local variances.

A1.3 Reward Redistribution

A1.3.1 Return-Equivalent and State Enriched MDPs

Our goal is to compare Markov decision processes (MDPs) without delayed reward to MDPs with delayed reward. Toward this end we consider MDPs $\tilde{\mathcal{P}}$ and $\mathcal{P}$ which differ only in their reward distributions $p(\tilde{r} \mid s, a)$ and $p(r \mid s, a)$ but have the same value of $\tilde{v}_0^\pi = v_0^\pi$ for each policy π.

Definition A3. *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 have the same expected return $\tilde{v}_0^\pi = v_0^\pi$ for each policy π.*

Lemma A3. *Return-equivalent decision processes have the same optimal policies.*

Proof. The optimal policy is defined as maximizing the expected return at time $t = 0$. The expected return is the same for return-equivalent decision processes. Consequently, the optimal policies are the same. □

Return-equivalent MDPs have the same states. For delayed reward the states have to code for the reward; however, for immediate reward the states can be made more compact by removing the reward information. For example states states with memory of delayed reward can be mapped to states without memory. Therefore, we introduce the concept of homomorphic MDPs to compare the MDPs.

Definition A4 (Ravindran and Barto [85, 86])**.** *An MDP homomorphism h from an MDP $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \pi, \gamma)$ to an MDP $\tilde{\mathcal{P}} = (\tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{\mathcal{R}}, \tilde{p}, \tilde{\pi}, \tilde{\gamma})$ is a a tuple of surjections $(f, g_1, g_2, \ldots, g_n)$ (n is number of states), with $h((s,a)) = (f(s), g_s(a))$, where $f : \mathcal{S} \to \tilde{\mathcal{S}}$ and $g_s : \mathcal{A}_s \to \tilde{\mathcal{A}}_{f(s)}$ for $s \in \mathcal{S}$ ($\mathcal{A}_s$ are the admissible actions in state s and $\tilde{\mathcal{A}}_{f(s)}$ are the admissible actions in state $\tilde{s}$). Furthermore, for all $s, s' \in \mathcal{S}, a \in \mathcal{A}_s$:*

$$\tilde{p}(f(s') \mid f(s), g_s(a)) \;=\; \sum_{s'' \in [s']_f} p(s'' \mid s, a) \;, \tag{A116}$$

$$\tilde{p}(\tilde{r} \mid f(s), g_s(a)) \;=\; p(r \mid s, a) \;. \tag{A117}$$

We used $[s]_f = [s']_f$ if and only if $f(s) = f(s')$.

$\tilde{\mathcal{P}}$ is the *homomorphic image* of $\mathcal{P}$ under h.

Lemma A4 (Ravindran and Barto [85])**.** *If $\tilde{\mathcal{P}}$ is a homomorphic image of $\mathcal{P}$, then the optimal Q-values are the same and a policy that is optimal in $\tilde{\mathcal{P}}$ can be transformed to an optimal policy in $\mathcal{P}$ by normalizing for the number of actions a that are mapped to the same action $\tilde{a}$.*

Consequently the original MDP can be solved by solving a homomorphic image.
Similar result have been obtained by Givan et al. using stochastically bisimilar MDPs: "Any stochastic bisimulation used for aggregation preserves the optimal value and action sequence properties as well as the optimal policies of the model" [28]. Theorem 7 and Corollary 9.1 in Givan et al. show the facts of Lemma A4. Li et al. give an overview over state abstraction or state aggregation for Markov decision processes, which covers homomorphic MDPs [63].
A Markov decision process $\tilde{\mathcal{P}}$ 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 define state-enrichment as follows.

Definition A5. *A decision process $\tilde{\mathcal{P}}$ is* state-enriched *compared to a decision process $\mathcal{P}$ if $\mathcal{P}$ is a homomorphic image of $\tilde{\mathcal{P}}$, where $g_{\tilde{s}}$ is the identity and $f(\tilde{s}) = s$ is not bijective.*

In particular state-enrichment does not change the optimal policies or the Q-values.

A1.3.2 Equivalence of Immediate Reward and Delayed Reward MDPs

We assume to have a Markov decision process $\mathcal{P}$ with immediate reward. This process is transformed to a decision process $\tilde{\mathcal{P}}$ with delayed reward, where the reward is given at sequence end. The reward-equivalent process with delayed reward $\tilde{\mathcal{P}}$ is state-enriched which ensures that the transformed process $\tilde{\mathcal{P}}$ is also a Markov decision process.
The transformed delayed state-enriched Markov process has

- reward:

$$\tilde{R}_t \;=\; \begin{cases} 0 \;, & \text{for } t \leqslant T \\ \sum_{k=0}^{T} R_{k+1} \;, & \text{for } t = T+1 \;. \end{cases} \tag{A118}$$

- state:

$$\tilde{s}_t \;=\; (s_t, \rho_t) \;, \tag{A119}$$

$$\rho_t \;=\; \sum_{k=0}^{t-1} r_{k+1} \;, \quad \text{with } R_k = r_k \;. \tag{A120}$$

The random variable R_k is distributed according to $p(r \mid s_k, a_k)$. We assume that the time t is coded in s in order to know when the episode ends and reward is no longer received, otherwise we introduce an additional state variable $\tau = t$ that codes the time.

Proposition A1. *If a Markov decision process $\mathcal{P}$ with immediate reward is transformed by above defined $\tilde{R}_t$ and $\tilde{s}_t$ to a Markov decision process $\tilde{\mathcal{P}}$ with delayed reward, where the reward is given at*

sequence end, then: (I) the optimal policies do not change, and (II) for $\tilde{\pi}(a \mid \tilde{s}) = \pi(a \mid s)$

$$\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;=\; q^{\pi}(s, a) \;+\; \sum_{k=0}^{t-1} r_{k+1} \;, \tag{A121}$$

for $\tilde{S}_t = \tilde{s}$*,* $S_t = s$*, and* $A_t = a$*.*

Proof. For (I) we first perform an state-enrichment of $\mathcal{P}$ by $\tilde{s}_t = (s_t, \rho_t)$ with $\rho_t = \sum_{k=0}^{t-1} r_{k+1}$ for $R_k = r_k$. The we change the reward $\tilde{R}_t$ which is a return-equivalent MDP, where the Markov property is ensured through the enriched state. Therefore the optimal policies do not change.
For (II) we show a proof without Bellman equation and a proof using the Bellman equation.
Equivalence without Bellman equation. We have $\tilde{G}_0 = G_0$. The Markov property ensures that the future reward is independent of the already received reward:

$$\mathrm{E}_{\pi}\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, A_t = a, \rho = \sum_{k=0}^{t-1} r_{k+1}\right] \;=\; \mathrm{E}_{\pi}\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, A_t = a\right] \;. \tag{A122}$$

We assume $\tilde{\pi}(a \mid \tilde{s}) = \pi(a \mid s)$.
We obtain

$$\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;&=\; \mathrm{E}_{\tilde{\pi}}\left[\tilde{G}_0 \mid \tilde{S}_t = \tilde{s}, A_t = a\right] \\
&=\; \mathrm{E}_{\tilde{\pi}}\left[\sum_{k=0}^{T} R_{k+1} \mid S_t = s, \rho = \sum_{k=0}^{t-1} r_{k+1}, A_t = a\right] \\
&=\; \mathrm{E}_{\tilde{\pi}}\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, \rho = \sum_{k=0}^{t-1} r_{k+1}, A_t = a\right] \;+\; \sum_{k=0}^{t-1} r_{k+1} \\
&=\; \mathrm{E}_{\pi}\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, A_t = a\right] \;+\; \sum_{k=0}^{t-1} r_{k+1} \\
&=\; q^{\pi}(s, a) \;+\; \sum_{k=0}^{t-1} r_{k+1} \;.
\end{aligned} \tag{A123}$$

We used $\mathrm{E}_{\tilde{\pi}} = \mathrm{E}_{\pi}$ which is ensured since reward probabilities, transition probabilities, and the probability of choosing an action by the policy correspond to each other in both settings.
Since the optimal policies do not change for reward-equivalent and state-enriched processes, we have

$$\tilde{q}^{*}(\tilde{s}, a) \;=\; q^{*}(s, a) \;+\; \sum_{k=0}^{t-1} r_{k+1} \;. \tag{A124}$$

Equivalence with Bellman equation. With $q^{\pi}(s, a)$ as optimal action-value function for the original Markov decision process, we define a new Markov decision process with action-state function $\tilde{q}^{\tilde{\pi}}$. For $\tilde{S}_t = \tilde{s}$, $S_t = s$, and $A_t = a$ we have

$$\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;:=\; q^{\pi}(s, a) \;+\; \sum_{k=0}^{t-1} r_{k+1} \;, \tag{A125}$$

$$\tilde{\pi}(a \mid \tilde{s}) \;:=\; \pi(a \mid s) \;. \tag{A126}$$

Since $\tilde{s}' = (s', \rho')$, $\rho' = r + \rho$, and $\tilde{r}$ is constant, the values $\tilde{S}_{t+1} = \tilde{s}'$ and $\tilde{R}_{t+1} = \tilde{r}$ can be computed from $R_{t+1} = r$, ρ, and $S_{t+1} = s'$. Therefore we have

$$\tilde{p}(\tilde{s}', \tilde{r} \mid s, \rho, a) \;=\; \tilde{p}(s', \rho', \tilde{r} \mid s, \rho, a) \;=\; p(s', r \mid s, a) \;. \tag{A127}$$

For $t < T$ we have $\tilde{r} = 0$ and $\rho' = r + \rho$, where we set $r = r_{t+1}$:

$$\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;&=\; q^{\pi}(s, a) \;+\; \sum_{k=0}^{t-1} r_{k+1} \\
&= \sum_{s', r} p(s', r \mid s, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \;+\; \sum_{k=0}^{t-1} r_{k+1} \\
&= \sum_{s', \rho'} \tilde{p}(s', \rho', \tilde{r} \mid s, \rho, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \;+\; \sum_{k=0}^{t-1} r_{k+1} \\
&= \sum_{\tilde{s}', \tilde{r}} \tilde{p}(\tilde{s}', \tilde{r} \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \;+\; \sum_{k=0}^{t-1} r_{k+1} \right] \\
&= \sum_{\tilde{s}', \tilde{r}} \tilde{p}(\tilde{s}', \tilde{r} \mid \tilde{s}, a) \left[\tilde{r} \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \;+\; \sum_{k=0}^{t} r_{k+1} \right] \\
&= \sum_{\tilde{s}', \tilde{r}} \tilde{p}(\tilde{s}', \tilde{r} \mid \tilde{s}, a) \left[\tilde{r} \;+\; \sum_{a'} \tilde{\pi}(a' \mid s')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a') \right] .
\end{aligned} \tag{A128}$$

For $t = T$ we have $\tilde{r} = \sum_{k=0}^{T} r_{k+1} = \rho'$ and $q^{\pi}(s', a') = 0$ as well as $\tilde{q}^{\tilde{\pi}}(\tilde{s}', a') = 0$. Both q and $\tilde{q}$ must be zero for $t \geqslant T$ since after time $t = T + 1$ there is no more reward. We obtain for $t = T$ and $r = r_{T+1}$:

$$\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;&=\; q^{\pi}(s, a) \;+\; \sum_{k=0}^{T-1} r_{k+1} \\
&= \sum_{s', r} p(s', r \mid s, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \;+\; \sum_{k=0}^{T-1} r_{k+1} \\
&= \sum_{s', \rho', r} \tilde{p}(s', \rho' \mid s, \rho, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \;+\; \sum_{k=0}^{T-1} r_{k+1} \\
&= \sum_{s', \rho', r} \tilde{p}(s', \rho' \mid s, \rho, a) \left[\sum_{k=0}^{T} r_{k+1} \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \\
&= \sum_{\tilde{s}', \rho'} \tilde{p}(\tilde{s}' \mid \tilde{s}, a) \left[\rho' \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \\
&= \sum_{\tilde{s}', \rho'} \tilde{p}(\tilde{s}' \mid \tilde{s}, a) \; [\rho' \;+\; 0] \\
&= \sum_{\tilde{s}', \tilde{r}} \tilde{p}(\tilde{s}' \mid \tilde{s}, a) \left[\tilde{r} \;+\; \sum_{a'} \tilde{\pi}(a' \mid s')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a') \right] .
\end{aligned} \tag{A129}$$

Since $\tilde{q}^{\tilde{\pi}}(\tilde{s}, a)$ fulfills the Bellman equation, it is the action-value function for $\tilde{\pi}$.

□

A1.3.3 Optimal Reward Redistribution

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 $\mathcal{P}$. $\mathcal{P}$ is return-equivalent to $\tilde{\mathcal{P}}$ where only the reward distributions are different. We have to redistribute the final reward, which is the return, $\tilde{r}_{T+1}$ to previous time steps, therefore we have the decompose the return into a sum of rewards at different time steps.

Return Decomposition. We assume a delayed reward MDP $\tilde{\mathcal{P}}$, where

$$\tilde{R}_t \;=\; \begin{cases} 0\,, & \text{for } t \leqslant T \\ \tilde{R}_{T+1}\,, & \text{for } t = T + 1\,, \end{cases} \tag{A130}$$

where $\tilde{R}_t = 0$ means that the random variable is always zero. We predict the expected accumulated return, that is, the expected reward at the last time step:

$$\tilde{r}(s_T, a_T) \;=\; \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right] \;. \tag{A131}$$

After observing an episode given by the state-action sequence

$$(s_0, a_0, s_1, a_1, \ldots, s_T, a_T) \tag{A132}$$

the prediction should be $\tilde{r}(s_T, a_T)$. Since we assume a Markov environment, $\tilde{r}(s_T, a_T)$ is a function of (s_T, a_T). To allow a reward redistribution, we have to avoid the Markov property in the input sequence. Toward this end, we define a difference $\Delta(s, s')$ between s and s'. We predict $\tilde{r}(s_T, a_T)$ from the sequence

$$\big(a_0, \Delta(s_0, s_1), a_1, \Delta(s_1, s_2), \ldots, a_t, \Delta(s_t, s_{t+1}), \ldots, a_{T-1}\Delta(s_{T-1}, s_T), a_T\big) \;. \tag{A133}$$

We assume that the difference Δ is designed such that it components in a sequence are mutually independent:

$$p\left(a_t, \Delta(s_t, s_{t+1}) \mid a_0, \Delta(s_0, s_1), \ldots, a_{t-1}, \Delta(s_{t-1}, s_t), a_{t+1}, \Delta(s_{t+1}, s_{t+2}), \ldots, a_{T-1}, \Delta(s_{T-1}, s_T), a_T\right) \tag{A134}$$

$$= \; p\left(a_t, \Delta(s_t, s_{t+1})\right) \;.$$

The $(a, \Delta(s, s'))$ are independent from each other and should ideally form a factorial code like the goal of independent component analysis (ICA) [51]. We define the sequence

$$(a, \Delta)_{0:t} \;:=\; \big(a_0, \Delta(s_0, s_1), a_1, \Delta(s_1, s_2), \ldots, a_t, \Delta(s_t, s_{t+1})\big) \;. \tag{A135}$$

The function g maps a sequence to a real value which predicts the reward. We want to decompose g:

$$g\big((a, \Delta)_{0:T}\big) \;=\; \sum_{t=0}^{T} h(a_t, \Delta(s_t, s_{t+1})) \;=\; \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right] \;=\; \tilde{r}(s_T, a_T) \;. \tag{A136}$$

The function g is decomposed into $h(a_t, \Delta(s_t, s_{t+1}))$ which sum up to $\tilde{r}(s_T, a_T)$.
For decomposing g one can use the Taylor decomposition (a linear approximation) of g with respect to the h [3, 73]. For non-linear g, layerwise relevance propagation (LRP) [3, 74] or integrated gradients (IG) [103] can be used to decompose g.

Reward Redistribution. Only if we do not change the accumulated reward via a redistribution, the optimal policies do not change. Therefore we first perform a decomposition:

$$g\big((a, \Delta)_{0:T}\big) \;=\; \sum_{t=0}^{T} h(a_t, \Delta(s_t, s_{t+1})) \;. \tag{A137}$$

The actual reward redistribution for $\tilde{R}_{T+1} = \tilde{r}_{T+1}$ is $R_t = r_t$ given by

$$r_{t+1} \;=\; \frac{h(a_t, \Delta(s_t, s_{t+1}))}{g\big((a, \Delta)_{0:T}\big)} \, \tilde{r}_{T+1} \tag{A138}$$

which gives

$$\sum_{t=0}^{T} \tilde{r}_{t+1} \;=\; \tilde{r}_{T+1} \;=\; \sum_{t=0}^{T} r_{t+1} \;. \tag{A139}$$

Using random variables that is

$$R_{t+1} \;=\; \frac{h(a_t, \Delta(s_t, s_{t+1}))}{g\big((a, \Delta)_{0:T}\big)} \, \tilde{R}_{T+1} \tag{A140}$$

which gives

$$\sum_{t=0}^{T} \tilde{R}_{t+1} \;=\; \tilde{R}_{T+1} \;=\; \sum_{t=0}^{T} R_{t+1} \;. \tag{A141}$$

For the expected reward we obtain

$$r(s_t, a_t) \;=\; \mathrm{E}\left[R_{t+1} \mid s_t, a_t\right] \;=\; \frac{h(a_t, \Delta(s_t, s_{t+1}))}{g\big((a, \Delta)_{0:T}\big)} \, \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right] \tag{A142}$$

$$= \; \frac{h(a_t, \Delta(s_t, s_{t+1}))}{g\big((a, \Delta)_{0:T}\big)} \, \tilde{r}(s_T, a_T) \;.$$

Optimality of the Reward Redistribution. We now consider predictions as approximations to Q-values. Since there is no reward for $t > T+1$, we have

$$\tilde{q}^\pi(s_T, a_T) \;=\; \tilde{r}(s_T, a_T)\;. \tag{A143}$$

The decomposition Eq. (A136) becomes

$$g\big((a,\Delta)_{0:T}\big) \;=\; \sum_{t=0}^{T} h(a_t, \Delta(s_t, s_{t+1})) \;=\; \tilde{q}^\pi(s_T, a_T)\;, \tag{A144}$$

where $\Delta(s_T, s_{T+1}) = \mathbf{0}$.
An optimal prediction must track constantly the expected final return because of independence condition Eq. (A134). For an optimal decomposition we require Eq. (A144) to be correct for partial sums:

$$g\big((a,\Delta)_{0:t}\big) \;=\; \sum_{\tau=0}^{t} h(a_\tau, \Delta(s_\tau, s_{\tau+1})) \;=\; \tilde{q}^\pi(s_t, a_t)\;. \tag{A145}$$

We obtain for $t = 0$

$$g\big((a,\Delta)_{0:0}\big) \;=\; h(a_0, \Delta(s_0, s_1)) \;=\; \tilde{q}^\pi(s_0, a_0) \tag{A146}$$

and for $t > 0$

$$g\big((a,\Delta)_{0:t}\big) \;-\; g\big((a,\Delta)_{0:t-1}\big) \;=\; h(a_t, \Delta(s_t, s_{t+1})) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1})\;. \tag{A147}$$

Therefore for an optimal decomposition we obtain

$$h_0 \;=\; h(a_0, \Delta(s_0, s_1)) \;=\; \tilde{q}^\pi(s_0, a_0) \tag{A148}$$

$$h_t \;=\; h(a_t, \Delta(s_t, s_{t+1})) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \quad \text{for } 0 < t \leqslant T\;. \tag{A149}$$

Reward Redistribution via an Optimal Return Decomposition. We now consider the MDP $\mathcal{P}$ based on the redistributed reward given by an optimal decomposition with $g = g\big((a,\Delta)_{0:T}\big) = \tilde{r}(s_T, a_T)$:

$$r(s_0, a_0) \;=\; \mathrm{E}\left[R_1 \mid s_0, a_0\right] \;=\; \frac{h_0}{\tilde{r}(s_T, a_T)} \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right] \;=\; \frac{h_0}{\tilde{r}(s_T, a_T)} \tilde{r}(s_T, a_T) \;=\; h_0\;, \tag{A150}$$

$$\begin{aligned} r(s_t, a_t) \;&=\; \mathrm{E}\left[R_{t+1} \mid s_t, a_t\right] \;=\; \frac{\mathrm{E}\left[h_t \mid s_t, a_t\right]}{\tilde{r}(s_T, a_T)} \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right] \;=\; \frac{\mathrm{E}\left[h_t \mid s_t, a_t\right]}{\tilde{r}(s_T, a_T)} \tilde{r}(s_T, a_T) \\ &=\; \mathrm{E}\left[h_t \mid s_t, a_t\right] \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \mathrm{E}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right]\;. \end{aligned} \tag{A151}$$

For $p(s_t, a_t \mid s_{t-1}, a_{t-1}) = p(s_t \mid s_{t-1}, a_{t-1}) p(a_t \mid s_t)$ the posterior $\mathrm{E}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right]$ does not depend on a_t and becomes $\mathrm{E}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t\right]$, since $p(a_t \mid s_t)$ cancels in the Bayes formula.
The next theorem gives the major advantage of a redistributed reward by an optimal decomposition. For TD the Q-values are no longer biased and for Monte Carlo the variance of the Q-value estimation is minimal.

Theorem A4. *The MDP $\mathcal{P}$ based on a redistributed reward (I) has the same optimal policies as $\tilde{\mathcal{P}}$ of the delayed reward, and (II) for an optimal return decomposition, the Q-values are given by*

$$q^\pi(s_t, a_t) \;=\; r(s_t, a_t) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \mathrm{E}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right]\;. \tag{A152}$$

Proof. We observe using Eq. (A141):

$$v_0^\pi \;=\; \mathrm{E}_\pi\left[\sum_{t=0}^{T} R_{t+1} \mid s_0\right] \;=\; \mathrm{E}_\pi\left[\tilde{R}_{T+1} \mid s_0\right] \;=\; \mathrm{E}_\pi\left[\sum_{t=0}^{T} \tilde{R}_{t+1} \mid s_0\right] \;=\; \tilde{v}_0^\pi\;. \tag{A153}$$

Consequently, $\tilde{v}_0^\pi = v_0^\pi$ for each policy π. Thus, the two decision processes $\tilde{\mathcal{P}}$ and $\mathcal{P}$ are return-equivalent according to Definition A3. According to Lemma A3 $\tilde{\mathcal{P}}$ and $\mathcal{P}$ have the same optimal policies.

We derive an important property of the redistributed reward given by an optimal decomposition. We have for $t > 0$

$$\begin{aligned}
&\mathrm{E}_\pi\left[\sum_{\tau=t}^{T} R_{\tau+1} \mid s_{t-1}, a_{t-1}\right] \;=\; \mathrm{E}_\pi\left[\sum_{\tau=t}^{T} \frac{h_t}{\tilde{r}(s_T, a_T)} \tilde{R}_{T+1} \mid s_{t-1}, a_{t-1}\right] \qquad \text{(A154)}\\
&= \mathrm{E}_\pi\left[\sum_{\tau=t}^{T} \frac{h_t}{\tilde{r}(s_T, a_T)} \tilde{r}(s_T, a_T) \mid s_{t-1}, a_{t-1}\right] \;=\; \mathrm{E}_\pi\left[\sum_{\tau=t}^{T} h_t \mid s_{t-1}, a_{t-1}\right]\\
&= \mathrm{E}_\pi\left[\sum_{\tau=t}^{T} (\tilde{q}^\pi(s_\tau, a_\tau) \;-\; \tilde{q}^\pi(s_{\tau-1}, a_{\tau-1})) \mid s_{t-1}, a_{t-1}\right]\\
&= \mathrm{E}_\pi\left[\tilde{q}^\pi(s_T, a_T) \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_{t-1}, a_{t-1}\right]\\
&= \mathrm{E}_\pi\left[\tilde{q}^\pi(s_T, a_T) \mid s_{t-1}, a_{t-1}\right] \;-\; \mathrm{E}_\pi\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_{t-1}, a_{t-1}\right]\\
&= \mathrm{E}_\pi\left[\mathrm{E}_\pi\left[\tilde{R}_{T+1}\right] \mid s_{t-1}, a_{t-1}\right] \;-\; \mathrm{E}_\pi\left[\mathrm{E}_\pi\left[\sum_{\tau=t-1}^{T} \tilde{R}_{\tau+1}\right] \mid s_{t-1}, a_{t-1}\right]\\
&= \mathrm{E}_\pi\left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1}\right] \;-\; \mathrm{E}_\pi\left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1}\right]\\
&= 0\,.
\end{aligned}$$

In last equation the term $\mathrm{E}_\pi\left[\tilde{q}^\pi(s_T, a_T) \mid s_{t-1}, a_{t-1}\right]$ and the term $\mathrm{E}_\pi\left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1}\right]$ means the expectation over all episodes starting in (s_{t-1}, a_{t-1}) and ending in some (s_T, a_T). We used that $\tilde{R}_{t+1} = 0$ for $t < T$.
For the Q-values of the MDP $\mathcal{P}$ with redistributed reward, we therefore have

$$\begin{aligned}
&\mathrm{E}_{s_t, a_t}\left[q^\pi(s_t, a_t) \mid s_{t-1}, a_{t-1}\right] \;=\; \mathrm{E}_{s_t, a_t}\left[\mathrm{E}_\pi\left[\sum_{\tau=t}^{T} R_{\tau+1} \mid s_t, a_t\right] \mid s_{t-1}, a_{t-1}\right] \qquad \text{(A155)}\\
&= \mathrm{E}_\pi\left[\sum_{\tau=t}^{T} R_{\tau+1} \mid s_{t-1}, a_{t-1}\right] \;=\; 0\,.
\end{aligned}$$

The Bellman equation Eq. (A19) gives

$$q^\pi(s, a) \;-\; r(s, a) \;=\; \mathrm{E}_{s', a'}\left[q^\pi(s', a') \mid s, a\right] \;=\; 0\,, \qquad \text{(A156)}$$

which is

$$q^\pi(s, a) \;=\; r(s, a)\,. \qquad \text{(A157)}$$

□

The local variance is now centered to zero:

$$\mathrm{Var}_{s', a'}\left[q^\pi(s', a') \mid s, a\right] \;=\; \mathrm{E}_{s', a'}\left[\left(q^\pi(s', a')\right)^2 \mid s, a\right]\,. \qquad \text{(A158)}$$

We introduced variance in the reward with mean $\mathrm{E}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right]$ that is given by the posterior $p(s_{t-1}, a_{t-1} \mid s_t, a_t)$. This mean is

$$\mathrm{E}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right] \;=\; \sum_{s_{t-1}, a_{t-1}} p(s_{t-1}, a_{t-1} \mid s_t, a_t)\; \tilde{q}^\pi(s_{t-1}, a_{t-1})\,, \qquad \text{(A159)}$$

$$\begin{aligned}
p(s_{t-1}, a_{t-1} \mid s_t, a_t) \;&=\; \frac{p(s_t, a_t \mid s_{t-1}, a_{t-1})\; p(s_{t-1}, a_{t-1})}{p(s_t, a_t)} \qquad \text{(A160)}\\
&=\; \frac{p(s_t \mid s_{t-1}, a_{t-1})\; p(s_{t-1}, a_{t-1})}{p(s_t)}\,,
\end{aligned}$$

where we used $p(s_t, a_t \mid s_{t-1}, a_{t-1}) = \pi(a_t \mid s_t) p(s_t \mid s_{t-1}, a_{t-1})$ and $p(s_t, a_t) = \pi(a_t \mid s_t) p(s_t)$.
For policy gradients we have:

$$\begin{aligned}
&\mathrm{E}_\pi\left[\nabla_\theta \log \pi(a \mid s; \boldsymbol{\theta})\; q^\pi(s, a)\right] \;=\; \mathrm{E}_\pi\left[\nabla_\theta \log \pi(a \mid s; \boldsymbol{\theta})\; r(s, a)\right] \qquad \text{(A161)}\\
&= \mathrm{E}_\pi\left[\nabla_\theta \log \pi(a \mid s; \boldsymbol{\theta})\; R \mid s, a\right]\,.
\end{aligned}$$

With baseline this gives

$$\begin{aligned} &\mathrm{E}_\pi \left[\nabla_\theta \log \pi(a \mid s; \boldsymbol{\theta}) \; (q^\pi(s,a) \; - \; \mathrm{E}_a \left[q^\pi(s,a) \right]) \right] \\ &= \; \mathrm{E}_\pi \left[\nabla_\theta \log \pi(a \mid s; \boldsymbol{\theta}) \; (R \; - \; \mathrm{E}_{r,a} \left[R \mid s, a \right]) \mid s, a \right] \; . \end{aligned} \tag{A162}$$

For policy gradients with eligibility traces using $\lambda \in [0,1]$ for G_t^λ [107], we have the expected updates

$$\mathrm{E}_\pi \left[\nabla_\theta \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \sum_{\tau=0}^{T-t} \lambda^\tau q^\pi(s_{t+\tau}, a_{t+\tau}) \right] \; = \; \mathrm{E}_\pi \left[\nabla_\theta \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \sum_{\tau=0}^{T-t} \lambda^\tau r(s_{t+\tau}, a_{t+\tau}) \right] \; . \tag{A163}$$

For $z_t = \sum_{\tau=0}^{T-t} \lambda^\tau r_{t+\tau}$ we obtain the recursion

$$z_t \; = \; r_t + \lambda z_{t+1} \tag{A164}$$

$$z_{T+1} \; = \; 0 \; . \tag{A165}$$

The proximal policy optimization (PPO) algorithm [96] uses an δ-error of

$$\delta_t \; = \; R_{t+1} \; + \; v^\pi(s_{t+1}) \; - \; v^\pi(s_t) \tag{A166}$$

which gives

$$\begin{aligned} \mathrm{E}_{s_{t+1}, r_{t+1}} \left[\delta_t \right] \; &= \; r(s_t, a_t) \; + \; \mathrm{E}_{s_{t+1}} \left[v^\pi(s_{t+1}) \right] \; - \; v^\pi(s_t) \\ &= \; r(s_t, a_t) \; + \; \mathrm{E}_{s_{t+1}, a_{t+1}} \left[q^\pi(s_{t+1}, a_{t+1}) \mid s_t, a_t \right] \; - \; v^\pi(s_t) \\ &= \; r(s_t, a_t) \; - \; \mathrm{E}_{a_t} \left[r(s_t, a_t) \right] \; . \end{aligned} \tag{A167}$$

A2 Related Reinforcement Topics

A2.1 Properties of the Bellman Operator for a Policy

At each time t the environment is in some state $s = s_t \in \mathcal{S}$. The agent π takes an action $a = a_t \in \mathcal{A}$, which causes a transition of the environment to state $s' = s_{t+1} \in \mathcal{S}$ and a reward $r = r_{t+1} \in \mathcal{R}$ for the agent with probability $p(s', r \mid s, a)$.
The Bellman operator maps a action-value function $q = q(s,a)$ to another action-value function. We do not require that q are Q-values and that r is the actual reward. We define the Bellman operator T^π for policy π as:

$$\mathrm{T}^\pi \left[q \right] (s,a) \; = \; \sum_{s', r} p(s', r \mid s, a) \; \left[r \; + \; \sum_{a'} \pi(a' \mid s') \; q(s', a') \right] \; . \tag{A168}$$

We often rewrite the operator as

$$\mathrm{T}^\pi \left[q \right] (s,a) \; = \; r(s,a) \; + \; \mathrm{E}_{s', a'} \left[q(s', a') \right] \; , \tag{A169}$$

where

$$r(s,a) \; = \; \sum_r r \; p(r \mid s, a) \; , \tag{A170}$$

$$\mathrm{E}_{s', a'} \left[q(s', a') \right] \; = \; \sum_{s'} p(s' \mid s, a) \; \sum_{a'} \pi(a' \mid s') \; q(s', a') \; . \tag{A171}$$

We did not explicitly express the dependency on the policy π and the state-action pair (s,a) in the expectation $\mathrm{E}_{s', a'}$, more precise would be to write $\mathrm{E}^\pi_{s', a'} \left[. \mid s, a \right]$.
More general we have

$$\mathrm{T}^\pi \left[q \right] (s,a) \; = \; g(s,a) \; + \; \mathrm{E}_{s', a'} \left[q(s', a') \right] \; . \tag{A172}$$

In the following we show properties for this general formulation.

A2.1.1 Monotonically Increasing and Continuous

We assume the general formulation Eq. (A172) of the Bellman operator. Proposition 2.1 on pages 22-23 in Bertsekas and Tsitsiklis, 1996, [11] shows that the fixed point q^π the Bellman operator exists and that for every q:

$$q^\pi \ = \ \mathrm{T}^\pi \left[q^\pi\right] \tag{A173}$$

$$q^\pi \ = \ \lim_{k \to \infty} \left(\mathrm{T}^\pi\right)^k q \ . \tag{A174}$$

The fixed point equation

$$q^\pi \ = \ \mathrm{T}^\pi \left[q^\pi\right] \tag{A175}$$

is called *Bellman equation* or *Poisson equation*. For the Poisson equation see Equation 33 to Equation 37 for the undiscounted case and Equation 34 and Equation 43 for the discounted case in Alexander Veretennikov, 2016, [116]. This form of the Poisson equation describes the Dirichlet boundary value problem.
The Poisson equation is

$$q^\pi(s,a) \ + \ \bar{g} \ = \ g(s,a) \ + \ \mathrm{E}_{s',a'} \left[q(s',a') \mid s,a\right] \ , \tag{A176}$$

where $\bar{g}$ is the long term average reward or the expected value of the reward for the stationary distribution:

$$\bar{g} \ = \ \lim_{T \to \infty} \frac{1}{T+1} \sum_{t=0}^{T} g(s_t, a_t) \ . \tag{A177}$$

We assume $\bar{g} = 0$ since after some time the agent does no longer receive reward for MDPs with finite time horizon or MDPs with absorbing states that have zero reward.
T^π is *monotonically increasing* in its arguments [11]. For q_1 and q_2 with $q_1 \geqslant q_2$ component-wise, we have

$$\begin{aligned} &\mathrm{T}^\pi \left[q_1\right](s,a) \ - \ \mathrm{T}^\pi \left[q_2\right](s,a) \\ &= \ \left(g(s,a) \ + \ \mathrm{E}_{s',a'} \left[q_1(s',a')\right]\right) \ - \ \left(g(s,a) \ + \ \mathrm{E}_{s',a'} \left[q_2(s',a')\right]\right) \\ &= \ \mathrm{E}_{s',a'} \left[q_1(s',a') \ - \ q_2(s',a')\right] \ \geqslant \ 0 \ , \end{aligned} \tag{A178}$$

where "$\geqslant$" is component-wise. The last inequality follows from the fact that $q_1 \geqslant q_2$ component-wise. We define the norm $\|.\|_\infty$ which gives the maximal difference of the Q-values:

$$\|q_1 \ - \ q_2\|_\infty \ = \ \max_{s,a} |q_1(s,a) \ - \ q_2(s,a)| \ . \tag{A179}$$

T is a *non-expansion mapping* for q_1 and q_2:

$$\|\mathrm{T}^\pi \left[q_1\right] \ - \ \mathrm{T}^\pi \left[q_2\right]\|_\infty \ = \ \max_{s,a} |\mathrm{T}[q_1](s,a) \ - \ \mathrm{T}[q_2](s,a)| \tag{A180}$$

$$\begin{aligned} &= \ \max_{s,a} \left| \left[g(s,a) \ + \ \sum_{s'} p(s' \mid s,a) \ \sum_{a'} \pi(a' \mid s') \ q_1(s',a') \right] \ - \right. \\ &\quad \left. \left[g(s,a) \ + \ \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s') \ q_2(s',a') \right] \right| \\ &= \ \max_{s,a} \left| \sum_{s'} p(s' \mid s,a) \ \sum_{a'} \pi(a' \mid s') \ \left(q_1(s',a') \ - \ q_2(s',a')\right) \right| \\ &\leqslant \ \max_{s,a} \sum_{s'} p(s' \mid s,a) \ \sum_{a'} \pi(a' \mid s') \ |q_1(s',a') \ - \ q_2(s',a')| \\ &\leqslant \ \max_{s',a'} |q_1(s',a') \ - \ q_2(s',a')| \ = \ \|q_1 \ - \ q_2\|_\infty \ . \end{aligned}$$

The first inequality is valid since the absolute value is moved into the sum. The second inequality is valid since the expectation depending on (s,a) is replaced by a maximum that does not depend on (s,a).
Consequently, the operator T^π is continuous.

A2.1.2 Contraction for Undiscounted Finite Horizon

For states that are time aware, we can define another norm with $0 < \gamma < 1$ which allows for a contraction mapping:

$$\|q_1 - q_2\|_{\infty,t} = \max_{t=0}^{T} \gamma^{T-t+1} \max_{s_t,a} |q_1(s_t,a) - q_2(s_t,a)| \ . \tag{A181}$$

T^π is a *contraction mapping* for q_1 and q_2 [11]:

$$\begin{aligned}
&\|\mathrm{T}^\pi[q_1] - \mathrm{T}^\pi[q_2]\|_{\infty,t} = \max_{t=0}^{T} \gamma^{T-t+1} \max_{s_t,a} |\mathrm{T}[q_1](s_t,a) - \mathrm{T}[q_2](s_t,a)| \qquad \text{(A182)}\\
&= \max_{t=0}^{T} \gamma^{T-t+1} \max_{s_t,a} \left| \left[g(s_t,a) + \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, q_1(s_{t+1},a') \right] - \right.\\
&\quad \left. \left[g(s_t,a) + \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, q_2(s_{t+1},a') \right] \right|\\
&= \max_{t=0}^{T} \gamma^{T-t+1} \max_{s_t,a} \left| \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, [q_1(s_{t+1},a') - q_2(s_{t+1},a')] \right|\\
&\leqslant \max_{t=0}^{T} \gamma^{T-t+1} \max_{s_t,a} \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, |q_1(s_{t+1},a') - q_2(s_{t+1},a')|\\
&\leqslant \max_{t=0}^{T} \gamma^{T-t+1} \max_{s_{t+1},a'} |q_1(s_{t+1},a') - q_2(s_{t+1},a')|\\
&\leqslant \max_{t=0}^{T} \gamma \, \gamma^{T-(t+1)+1} \max_{s_{t+1},a'} |q_1(s_{t+1},a') - q_2(s_{t+1},a')|\\
&= \gamma \max_{t=1}^{T+1} \gamma^{T-t+1} \max_{s_t,a'} |q_1(s_t,a') - q_2(s_t,a')|\\
&= \gamma \max_{t=0}^{T} \gamma^{T-t+1} \max_{s_t,a'} |q_1(s_t,a') - q_2(s_t,a')|\\
&= \gamma \, \|q_1 - q_2\|_{\infty,t} \ .
\end{aligned}$$

The equality in the last but one line stems from the fact that all Q-values at $t = T+1$ are zero and all Q-values at $t = 1$ have the same constant value.
Furthermore for q with correct values have values equal to zero for additionally introduced states at $t = T+1$ since for $t > T+1$ all rewards are zero. We have

$$q^\pi = \mathrm{T}^T[q] \ . \tag{A183}$$

q is correct for additionally introduced states at time $t = T+1$ since they are zero. Then in the next iteration Q-vales of states at time $t = T$ are correct. After iteration i, Q-vales of states at time $t = T - i + 1$ are correct. This iteration is called the "backward induction algorithm" [82, 83]. If we perform this iteration for a policy π instead of the optimal policy, then this procedure is called "policy evaluation algorithm" [82, 83].

A2.1.3 Contraction for Undiscounted Infinite Horizon With Absorbing States

A stationary policy is *proper* if there exists an integer n such that from any initial state x the probability of achieving the terminal state after n steps is strictly positive.
If all terminal states are absorbing and cost/reward free and if all stationary policies are proper the Bellman operator is a contraction mapping with respect to a weighted sup-norm.
That the Bellman operator is a contraction mapping with respect to a weighted sup-norm has been proved in Tseng, 1990, in Lemma 3 with equation (13) and text thereafter [111]. Also Proposition 1 in Bertsekas and Tsitsiklis, 1991, [10], Theorems 3 and 4(b) & 4(c) in Tsitsiklis, 1994, [112], and Proposition 2.2 on pages 23-24 in Bertsekas and Tsitsiklis, 1996, [11] have proved the same fact.

A2.1.4 Fixed Point of Contraction is Continuous wrt Parameters

q^π and V^π are continuous with respect to π, that is $\pi(a' \mid s')$, with respect to the reward distribution $p(r \mid s,a)$ and with respect to the transition probabilities $p(s' \mid s,a)$

A complete metric space or a Cauchy space is a space where every Cauchy sequence of points has a limit in the space, that is, every Cauchy sequence converges in the space. The Euclidean space $\mathbb{R}^n$ with the usual distance metric is complete. Lemma 2.5 in Jachymski, 1996, is [53]:

Theorem A5 (Jachymski: complete metric space)**.** *Let (X, d) be a complete metric space, and let (P, d_P) be a metric space. Let $F : P \times X \to X$ be continuous in the first variable and contractive in the second variable with the same Lipschitz constant $\alpha < 1$. For $\boldsymbol{p} \in P$, let $\boldsymbol{x}^*(\boldsymbol{p})$ be the unique fixed point of the map $\boldsymbol{x} \to F(\boldsymbol{p}, \boldsymbol{x})$. Then the mapping $\boldsymbol{x}^*$ is continuous.*

This theorem is Theorem 2.3 in Frigon, 2007, [22]. Corollary 4.2 in Feinstein, 2016, generalized the theorem to set valued operators, that is, these operators may have more than one fixed point [20] (see also [55]). All mappings $F(p, .)$ must have the same Lipschitz constant $\alpha < 1$.
A locally compact space is a space where every point has a compact neighborhood. $\mathbb{R}^n$ is locally compact as a consequence of the Heine-Borel theorem. Proposition 3.2 in Jachymski, 1996, is [53]:

Theorem A6 (Jachymski: locally compact complete metric space)**.** *Let (X, d) be a locally compact complete metric space, and let (P, d_P) be a metric space. Let $F : P \times X \to X$ be continuous in the first variable and contractive in the second variable with not necessarily the same Lipschitz constant. For $\boldsymbol{p} \in P$, let $\boldsymbol{x}^*(\boldsymbol{p})$ be the unique fixed point of the map $\boldsymbol{x} \to F(\boldsymbol{p}, \boldsymbol{x})$. Then the mapping $\boldsymbol{x}^*$ is continuous.*

This theorem is Theorem 2.5 in Frigon, 2007, [22] and Theorem 2 in Kwiecinski, 1992, [59]. The mappings $F(p, .)$ can have different Lipschitz constants.

A2.1.5 t-fold Composition of the Operator

We defined the Bellman operator as

$$\begin{aligned} \mathrm{T}^\pi [q](s,a) &= g(s,a) + \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q(s',a') \\ &= g(s,a) + \boldsymbol{q}^T \boldsymbol{p}(s,a)\,, \end{aligned} \tag{A184}$$

where $\boldsymbol{q}$ is the vector with value $q(s', a')$ at position (s', a') and $\boldsymbol{p}(s, a)$ is the vector with value $p(s' \mid s, a)\pi(a' \mid s')$ at position (s', a').
In vector notation we obtain the *Bellman equation* or *Poisson equation*. For the Poisson equation see Equation 33 to Equation 37 for the undiscounted case and Equation 34 and Equation 43 for the discounted case in Alexander Veretennikov, 2016, [116]. This form of the Poisson equation describes the Dirichlet boundary value problem. The *Bellman equation* or *Poisson equation* is

$$\mathrm{T}^\pi [\boldsymbol{q}] = \boldsymbol{g} + \boldsymbol{P}\,\boldsymbol{q}\,, \tag{A185}$$

where $\boldsymbol{P}$ is the row-stochastic matrix with $p(s' \mid s, a)\pi(a' \mid s')$ at position $((s, a), (s', a'))$.
The Poisson equation is

$$\boldsymbol{q}^\pi + \bar{g}\mathbf{1} = \boldsymbol{g} + \boldsymbol{P}\,\boldsymbol{q}\,, \tag{A186}$$

where $\mathbf{1}$ is the vector of ones and $\bar{\boldsymbol{g}}$ is the long term average reward or the expected value of the reward for the stationary distribution:

$$\bar{g} = \lim_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{T} g(s_t, a_t)\,. \tag{A187}$$

We assume $\bar{g} = 0$ since after some time the agent does no longer receive reward for MDPs with finite time horizon or MDPs with absorbing states that have zero reward.
Since $\boldsymbol{P}$ is a row-stochastic matrix, the Perron-Frobenius theorem says that (1) $\boldsymbol{P}$ has as largest eigenvalue 1 for which the eigenvector corresponds to the steady state and (2) the absolute value of each (complex) eigenvalue is smaller equal 1. Only the eigenvector to the eigenvalue 1 has only positive real components.
Equation 7 of Bertsekas and Tsitsiklis, 1991, [10] states

$$\left(\mathrm{T}^\pi\right)^t [\boldsymbol{q}] = \sum_{k=0}^{t-1} \boldsymbol{P}^k\,\boldsymbol{g} + \boldsymbol{P}^t\,\boldsymbol{q}\,. \tag{A188}$$

If $\boldsymbol{p}$ is the stationary distribution vector for $\boldsymbol{P}$, that is,

$$\lim_{k\to\infty} \boldsymbol{P}^k = \mathbf{1}\,\boldsymbol{p}^T \tag{A189}$$

$$\lim_{k\to\infty} \boldsymbol{p}_0^T \boldsymbol{P}^k = \boldsymbol{p}^T \tag{A190}$$

then

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \boldsymbol{P}^i \;=\; \mathbf{1}\, \boldsymbol{p}^T \tag{A191}$$

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \boldsymbol{p}_0^T \boldsymbol{P}^i \;=\; \boldsymbol{p}^T \,. \tag{A192}$$

A2.2 Q-value Transformations: Shaping Reward, Baseline, and Normalization

The Bellman equation for the action-value function q^π is

$$q^\pi(s,a) \;=\; \sum_{s',r} p(s', r \mid s, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^\pi(s', a') \right] . \tag{A193}$$

The expected return at time $t = 0$ is:

$$v_0 \;=\; \sum_{s_0} p(s_0)\, v(s_0) \,. \tag{A194}$$

As introduced for the REINFORCE algorithm, we can subtract a baseline v_0 from the return. We subtract the baseline v_0 from the last reward. Therefore, for the new reward $\bar{R}$ we have $\bar{R}_t = R_t$ for $t \leqslant T$, $\bar{R}_{T+1} = R_{T+1} - v_0$. Consequently $\bar{q}(s_t, a_t) = q(s_t, a_t) - v_0$ for $t \leqslant T$.
The TD update rules are:

$$q(s_t, a_t) \;\longleftarrow\; q(s_t, a_t) \;+\; \alpha \left(r_t \;+\; \sum_a \pi(a \mid s_{t+1})\, q(s_{t+1}, a) \;-\; q(s_t, a_t) \right) . \tag{A195}$$

The δ-errors are

$$\begin{aligned} & R_{t+1} \;+\; \sum_a \pi(a \mid s_{t+1})\, q(s_{t+1}, a) \;-\; q(s_t, a_t) \\ &= R_{t+1} \;+\; \sum_a \pi(a \mid s_{t+1}) \left(q(s_{t+1}, a) - v_0 \right) \;-\; \left(q(s_t, a_t) - v_0 \right) \\ &= \bar{R}_{t+1} \;+\; \sum_a \pi(a \mid s_{t+1})\, \bar{q}(s_{t+1}, a) \;-\; \bar{q}(s_t, a_t) \end{aligned} \tag{A196}$$

and for the last step

$$\begin{aligned} & R_{T+1} \;-\; q(s_T, a_T) \;=\; (R_{T+1} \;-\; v_0) \;-\; (q(s_T, a_T) \;-\; v_0) \\ &= \bar{R}_{T+1} \;-\; \bar{q}(s_T, a_T) \,. \end{aligned} \tag{A197}$$

If we set

$$\bar{q}(s_t, a_t) \;=\; \{ q(s_t, a_t) \;-\; v_0 \,, \quad \text{for } t \leqslant T \,. \tag{A198}$$

$$\bar{R}_t \;=\; \begin{cases} R_t \,, & \text{for } t \leqslant T \\ R_{T+1} \;-\; v_0 \,, & \text{for } t = T+1 \,, \end{cases} \tag{A199}$$

then the δ-errors and the updates remain the same for q and $\bar{q}$. We are equally far away from the optimal solution in both cases.
Removing the offset v_0 at the end by $\bar{R}_{T+1} = R_{T+1} - v_0$, can also be derived via reward shaping; however, the offset has to be added at the beginning: $\bar{R}_1 = R_1 + v_0$. Reward shaping requires for the shaping reward F and a potential function Φ [76, 122]:

$$F(s_t, a_t, s_{t+1}) \;=\; \Phi(s_{t+1}) - \Phi(s_t) \,. \tag{A200}$$

For introducing a reward of c at time $t = k$ and removing it from time $t = m < k$ we set:

$$\Phi(s_t) \;=\; \begin{cases} 0 \,, & \text{for } t \leqslant m \,, \\ -c \,, & \text{for } m+1 \leqslant t \leqslant k \,, \\ 0 \,, & \text{for } t > k \,, \end{cases} \tag{A201}$$

then the shaping reward is

$$F\big(s_t, a_t, s_{t+1}\big) \;=\; \begin{cases} 0\,, & \text{for } t < m\,, \\ -c\,, & \text{for } t = m\,, \\ 0\,, & \text{for } m+1 \leqslant t < k\,, \\ c\,, & \text{for } t = k\,, \\ 0\,, & \text{for } t > k\,. \end{cases} \tag{A202}$$

For $k = T$, $m = 0$, and $c = -v_0$ we obtain above situation but with $\bar{R}_1 = R_1 + v_0$ and $\bar{R}_{T+1} = R_{T+1} - v_0$, that is, v_0 removed at the end and added at the beginning. All Q-values except $q(s_0, a_0)$ are decreased by v_0. In the general case, all Q-values $q(s_t, a_t)$ with $m + 1 \leqslant t \leqslant k$ are increased by c.

Q-value normalization: We apply reward shaping [76, 122] for normalization of the Q-values. The potential $\Phi(s)$ defines the shaping reward $F(s_t, a_t, s_{t+1}) = \Phi(s_{t+1}) - \Phi(s_t)$. The optimal policies do not change and the Q-values become

$$q^{\text{new}}(s_t, a_t) \;=\; q(s_t, a_t) \;-\; \Phi(s_t)\,. \tag{A203}$$

We change the Q-values for all $1 \leqslant t \leqslant T$ but not for $t = 0$ and $t = T + 1$, the first and the last Q-values are not normalized. All the shaped reward is added/subtracted to/from the initial and the last reward.

- The maximal Q-values are zero and the non-optimal Q-values are negative for all $1 \leqslant t \leqslant T$:

$$\Phi(s_t) \;=\; \max_a q(s_t, a)\,. \tag{A204}$$

- The minimal Q-values are zero and all others Q-values are positive for all $1 \leqslant t \leqslant T - 1$:

$$\Phi(s_t) \;=\; \min_a q(s_t, a)\,. \tag{A205}$$

A2.3 Alternative Definition of State Enrichment

Next we define state-enriched processes $\tilde{\mathcal{P}}$ compared to $\mathcal{P}$. The state $\tilde{s}$ of $\tilde{\mathcal{P}}$ is enriched with a deterministic information compared to a state s of $\mathcal{P}$. The enriched information in $\tilde{s}'$ can be computed from the state-action pair $(\tilde{s}, a)$ and the reward r. Enrichments may be the accumulated reward, count of the time step, a count how often a certain action has been performed, a count how often a certain state has been visited, etc. Givan et al. have already shown that state-enriched Markov decision processes preserves the optimal action-value and action sequence properties as well as the optimal policies of the model [28]. Theorem 7 and Corollary 9.1 in Givan et al. show these properties [28] proved by bisimulations (stochastically bisimilar MDPs). A homomorphism between MDPs maps a MDP to another one with corresponding reward and transitions probabilities. Ravindran and Barto have shown that solving the original MDP can be done by solving a homomorphic image [86]. Therefore Ravindran and Barto have also shown that state-enriched Markov decision processes preserves the optimal action-value and action sequence properties. Li et al. give an overview over state abstraction or state aggregation for Markov decision processes, which covers state-enriched Markov decision processes [63].

Definition A6. *A decision process $\tilde{\mathcal{P}}$ is* state-enriched *compared to a decision process $\mathcal{P}$ if following conditions hold. If $\tilde{s}$ is the state of $\tilde{\mathcal{P}}$, then there exists a function $f : \tilde{s} \to s$ with $f(\tilde{s}) = s$, where s is the state of $\mathcal{P}$. There exists a function $g : \tilde{s} \to \mathcal{R}$ where $g(\tilde{s})$ gives the additional information of state $\tilde{s}$ compared to $f(\tilde{s})$. There exists a function ν with $\nu(f(\tilde{s}), g(\tilde{s})) = \tilde{s}$, that is, the state $\tilde{s}$ can be constructed from the original state and the additional information. There exists a function H with $h(\tilde{s}') = H(r, \tilde{s}, a)$, where $\tilde{s}'$ is the next state and r the reward. H ensures that $h(\tilde{s}')$ of next state $\tilde{s}'$ can be computed from reward r, actual state $\tilde{s}$, and the actual action a. Consequently, $\tilde{s}'$ can be computed from $(r, \tilde{s}, a)$. For all $\tilde{s}$ and $\tilde{s}'$ following holds:*

$$\tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \;=\; p(f(\tilde{s}'), r \mid f(\tilde{s}), a)\,, \tag{A206}$$

$$\tilde{p}_0(\tilde{s}_0) \;=\; p_0(f(\tilde{s}_0))\,, \tag{A207}$$

where $\tilde{p}_0$ and p_0 are the probabilities of the initial states of $\tilde{\mathcal{P}}$ and $\mathcal{P}$, respectively.

If the reward is deterministic, then $\tilde{p}(\tilde{s}', r \mid \tilde{s}, a) = \tilde{p}(\tilde{s}' \mid \tilde{s}, a)$ and $\tilde{p}_0(\tilde{s}_0, r) = \tilde{p}_0(\tilde{s}_0)$.

The following lemma has already been shown in Givan et al. by stochastically bisimilar Markov decision processes: "Any stochastic bisimulation used for aggregation preserves the optimal value and action sequence properties as well as the optimal policies of the model" [28]. Theorem 7 and Corollary 9.1 in Givan et al. show these properties. Ravindran and Barto have shown that solving the original MDP can be done by solving a homomorphic image, therefore state-enriched Markov decision processes preserves the optimal action-value and action sequence properties [86]. Li et al. give an overview over state abstraction or state aggregation for Markov decision processes, which covers state-enriched Markov decision processes [63].
We proof the following theorem, even if it has been proved several times as mention above.

Theorem A7. *If decision process $\tilde{\mathcal{P}}$ is state-enriched compared to $\mathcal{P}$, then for each optimal policy $\tilde{\pi}^*$ of $\tilde{\mathcal{P}}$ there exists an equivalent optimal policy π^* of $\mathcal{P}$, and vice versa, with $\tilde{\pi}^*(\tilde{s}) = \pi^*(f(\tilde{s}))$. The optimal return is the same for $\tilde{\mathcal{P}}$ and $\mathcal{P}$.*

Proof. We proof by induction that $\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) = q^{\pi}(f(\tilde{s}), a)$ if $\tilde{\pi}(\tilde{s}) = \pi(f(\tilde{s}))$.
Basis: the end of the sequence. For $t \geqslant T$ we have $\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) = q^{\pi}(f(\tilde{s}), a) = 0$, since no policy receives reward for $t \geqslant T$.
Inductive step ($t \to t-1$): Assume $\tilde{q}^{\tilde{\pi}}(\tilde{s}', a') = q^{\pi}(f(\tilde{s}'), a')$ for the next state $\tilde{s}'$ and next action a'.

$$
\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;&=\; \mathrm{E}_{\tilde{\pi}}\left[\tilde{G}_t \mid \tilde{s}_t = \tilde{s}, A_t = a\right] \;=\; \sum_{\tilde{s}', r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), g(\tilde{s}'), r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \qquad \text{(A208)} \\
&= \sum_{f(\tilde{s}'), G(r, \tilde{s}, a), r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), r} p(f(\tilde{s}'), r \mid f(\tilde{s}), a) \left[r \;+\; \sum_{a'} \pi(a' \mid f(\tilde{s}'))\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), r} p(f(\tilde{s}'), r \mid f(\tilde{s}), a) \left[r \;+\; \sum_{a'} \pi(a' \mid f(\tilde{s}'))\, q^{\pi}(f(\tilde{s}'), a')\right] \\
&= q^{\pi}(f(\tilde{s}), a)\,.
\end{aligned}
$$

For induction step $1 \to 0$ we use $\tilde{p}_0(\tilde{s}_0, r) = p_0(f(\tilde{s}_0), r)$ instead of $\tilde{p}(\tilde{s}', r \mid \tilde{s}, a) = p(f(\tilde{s}'), r \mid f(\tilde{s}), a)$.
It follows that $\tilde{q}^*(\tilde{s}, a) = q^*(f(\tilde{s}), a)$, and therefore

$$
\tilde{\pi}^*(\tilde{s}) \;=\; \operatorname*{argmax}_a \;\tilde{q}^*(\tilde{s}, a) \;=\; \operatorname*{argmax}_a \;q^*(f(\tilde{s}), a) \;=\; \pi^*(f(\tilde{s}))\,. \qquad \text{(A209)}
$$

Using Bellman's optimality equation would give the same result, where in above equation both $\sum_{a'} \pi(a' \mid f(\tilde{s}'))$ and $\max_{a'}$ and $\sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')$ are replaced by $\max_{a'}$. □

Theorem A8. *If Markov decision process $\tilde{\mathcal{P}}$ is state-enriched compared to the MDP $\mathcal{P}$, then for each optimal policy $\tilde{\pi}^*$ of $\tilde{\mathcal{P}}$ there exists an equivalent optimal policy π^* of $\mathcal{P}$, and vice versa, with $\tilde{\pi}^*(f(s)) = \pi^*(s)$. The optimal return is the same for $\tilde{\mathcal{P}}$ and $\mathcal{P}$.*

Proof. The MDP $\tilde{\mathcal{P}}$ is a homomorphic image of $\mathcal{P}$. For state-enrichment, the mapping g is bijective, therefore the optimal policies in $\tilde{\mathcal{P}}$ and $\mathcal{P}$ are equal according to Lemma A4. The optimal return is also equal since it does not change via state-enrichment. □

A2.4 Variance of the Weighted Sum of a Multinomial Distribution

State transitions are multinomial distributions and the future expected reward is a weighted sum of multinomial distributions. Therefore we are interested in the variance of the weighted sum of a multinomial distribution. Since we have

$$\mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] = \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q^\pi(s',a')\;, \tag{A210}$$

the variance of $\mathrm{E}_{s',a'}\left[q^\pi(s',a')\right]$ is determined by the variance of the multinomial distribution $p(s' \mid s,a)$. In the following we derive the variance of the estimation of a a linear combination of variables of a multinomial distribution like $\sum_{s'} p(s' \mid s,a) f(s')$.

A multinomial distribution with parameters $(p_1,\ldots,p_N)$ as event probabilities satisfying $\sum_{i=1}^N p_i = 1$ and support $x_i \in \{0,\ldots,n\}$, $i \in \{1,\ldots,N\}$ for n trials, that is $\sum x_i = n$, has

$$\text{pdf} \qquad \frac{n!}{x_1!\cdots x_k!}\, p_1^{x_1}\cdots p_k^{x_k}\;, \tag{A211}$$

$$\text{mean} \qquad \mathrm{E}[X_i] \;=\; n\, p_i\;, \tag{A212}$$

$$\text{variance} \qquad \mathrm{Var}[X_i] \;=\; n\, p_i\,(1-p_i)\;, \tag{A213}$$

$$\text{covariance} \qquad \mathrm{Cov}[X_i,X_j] \;=\; -\,n\, p_i\, p_j\;,\;\; (i \neq j)\;, \tag{A214}$$

where X_i is the random variable and x_i the actual count.

A linear combination of random variables has variance

$$\begin{aligned}\mathrm{Var}\left[\sum_{i=1}^N a_i\, X_i\right] \;&=\; \sum_{i,j=1}^N a_i\, a_j\, \mathrm{Cov}\left[X_i,X_j\right] \\ &= \sum_{i=1}^N a_i^2\, \mathrm{Var}\left[X_i\right] \;+\; \sum_{i\neq j} a_i\, a_j\, \mathrm{Cov}\left[X_i,X_j\right]\;.\end{aligned} \tag{A215}$$

The variance of estimating the mean X of independent random variables $(X_1,\ldots,X_n)$ that have all the variance σ^2 is:

$$\begin{aligned}\mathrm{Var}\left[X\right] \;&=\; \mathrm{Var}\left[\frac{1}{n}\sum_{i=1}^n X_i\right] \\ &=\; \frac{1}{n^2}\sum_{i=1}^n \mathrm{Var}\left[X_i\right] \;=\; \frac{1}{n^2}\sum_{i=1}^n \sigma^2 \;=\; \frac{\sigma^2}{n}\;.\end{aligned} \tag{A216}$$

For estimating the mean $\bar{y}$ over n samples of a linear combination of variables of a multinomial distribution $y = \sum_{i=1}^N a_i X_i$, where each y has n_y trials, we obtain:

$$\begin{aligned}\mathrm{Var}\left[\bar{y}\right] \;&=\; \frac{\sigma_y^2}{n} \;=\; \frac{1}{n}\left(\sum_{i=1}^N a_i^2\, n_y\, p_i\,(1-p_i) \;-\; \sum_{i\neq j} a_i\, a_j\, n_y\, p_i\, p_j\right) \\ &=\; \frac{n_y}{n}\left(\sum_{i=1}^N a_i^2\, p_i\,(1-p_i) \;-\; \sum_{i\neq j} a_i\, a_j\, p_i\, p_j\right) \\ &=\; \frac{n_y}{n}\left(\sum_{i=1}^N a_i^2\, p_i \;-\; \sum_{(i,j)=(1,1)}^{(N,N)} a_i\, a_j\, p_i\, p_j\right) \\ &=\; \frac{n_y}{n}\left(\sum_{i=1}^N a_i^2\, p_i \;-\; \left(\sum_{i=1}^N a_i\, p_i\right)^2\right)\;.\end{aligned} \tag{A217}$$

A3 Backward Analysis Through Contribution Analysis

For attributing the prediction of a deep network to its input features several methods have been proposed. We consider Layer-Wise Relevance Propagation (LRP), Input Zeroing, and integrated gradients (IG).

A3.1 Layer-Wise Relevance Propagation (LRP)

We first introduce Layer-Wise Relevance Propagation (LRP) in order use it for credit assignment over time. LRP guarantees credit assignment, that is, the redistribution of delayed reward or the return over time.

A3.1.1 Review of LRP

Layer-Wise Relevance Propagation (LRP) [3] has been introduced to interpret machine learning models. LRP is an extension of the contribution-propagation algorithm [60] based on the contribution approach [81]. Recently "excitation backprop" was proposed [127], which is like LPR but uses only positive weights and shifts the activation function to have non-negative values. Both algorithms assign a relevance or importance value to each node of a neural network which describes how much it contributed to generating the network output. The relevance or importance is recursively propagated back: A neuron was important to the network output if it was important to its parents, and its parents were important to the network output. LRP moves through a neural network like backpropagation: it starts at the output, redistributes the relevance scores of one layer to the previous layer until the input layer is reached. The redistribution procedure satisfies a local relevance conservation principle. All relevance values a node obtains from its parents will redistributed to its children. This is analog to Kirchhoff's first law for the conservation of electric charge or the continuity equation in physics for the transport of some quantity.
LRP has been used for deep neural networks (DNN) [74] and for recurrent neural networks like LSTM [1].
We consider a neural network with activation x_i for neuron i. The weight from neuron l to neuron i is denoted by w_{il}. The activation function is g and net_i is the netinput to neuron i with bias b_i. We have following forward propagation rules:

$$\mathrm{net}_i \;=\; \sum_l w_{il}\, x_l \;, \tag{A218}$$

$$x_i \;=\; f_i(\mathrm{net}_i) \;=\; g(\mathrm{net}_i + b_i) \;. \tag{A219}$$

Let R_i be the relevance for neuron i and $R_{i\leftarrow k}$ the share of relevance R_k that flows from neuron k in the higher layer to neuron i in the lower layer. The parameter z_{ik} is a weighting for the share of R_k of neuron k that flows to neuron i. We define $R_{i\leftarrow k}$ as

$$R_{i\leftarrow k} \;=\; \frac{z_{ik}}{\sum_l z_{lk}}\, R_k \;. \tag{A220}$$

The relative contributions z_{ik} were in previous work defined as [3, 74, 1]:

$$z_{ik} \;=\; w_{ik}\, x_k \;. \tag{A221}$$

Here z_{ik} is the contribution of x_k to the netinput value net_i. If neuron k would be removed from the network, then z_{ik} is the difference to the original net_i.
The relevance R_i of neuron i is the sum of relevances it obtains from its parents k from a layer above:

$$R_i \;=\; \sum_k R_{i\leftarrow k} \;. \tag{A222}$$

Furthermore a unit k passes on all its relevance R_k to its children, which are units i of the layer below:

$$R_k \;=\; \sum_i R_{i\leftarrow k} \;. \tag{A223}$$

It follows the *conservation of relevance*. The sum of relevances R_k of units k in a layer is equal to the sum of relevances R_i of units i of a layer below:

$$\sum_k R_k \;=\; \sum_k \sum_i R_{i\leftarrow k} \;=\; \sum_i \sum_k R_{i\leftarrow k} \;=\; \sum_i R_i \;. \tag{A224}$$

The scalar output $g(\boldsymbol{x})$ of a neural network with input $\boldsymbol{x} = (x_1, \ldots, x_d)$ is considered as relevance R which is decomposed into contributions R_i of the inputs x_i:

$$\sum_i R_i \;=\; R \;=\; g(\boldsymbol{x}) \;. \tag{A225}$$

The decomposition is valid for recurrent neural networks, where the relevance at the output is distributed across the sequence elements of the input sequence.

A3.1.2 New Variants of LRP

An alternative definition of z_{ik} is

$$z_{ik} \;=\; w_{ik}\,(x_k \;-\; \bar{x}_k)\;, \tag{A226}$$

where $\bar{x}_k$ is the mean of x_k across samples. Therefore $(x_k \;-\; \bar{x}_k)$ is the contribution of the actual sample to the variance of x_k. This in turn is related to the information carried by x_k. Here z_{ik} is the contribution of x_k to the variance of net_i. However, we can have negative values of $(x_k \;-\; \bar{x}_k)$ which may lead to negative contributions even if the weights are positive.
Another alternative definition of z_{ik} is

$$z_{ik} \;=\; f_i(\mathrm{net}_i) \;-\; f_i(\mathrm{net}_i \;-\; w_{ik}\,x_k)\;. \tag{A227}$$

Here z_{ik} is the contribution of x_k to the activation value $x_i = f_i(\mathrm{net}_i)$. If neuron k would be removed from the network, then z_{ik} is the difference to the original x_i. If f_i is strict monotone increasing and $x_k > 0$, then positive weights w_{ik} lead to positive values and negative weights w_{ik} to negative values.
Preferred Solution:
A definition of z_{ik} is

$$z_{ik} \;=\; w_{ik}\,(x_k \;-\; x_{\min})\;, \tag{A228}$$

where $x_{\min}$ is the minimum of x_k either across samples (mini-batch) or across time steps. The difference $(x_k - x_{\min})$ is always positive. Using this definition, activation functions with negative values are possible like for excitation backprop [127]. The minimal value is considered as default off-set, which can be included into the bias.

A3.1.3 LRP for Products

Here we define relevance propagation for products of two units. We assume that $z = x_1 x_2$ with $x_1 > 0$ and $x_2 > 0$. We view x_1 and x_2 as units of a layer below the layer in which z is located. Consequently, R_z has to be divided between x_1 and x_2, which gives the conservation rule

$$R_z \;=\; R_{x_1 \leftarrow z} \;+\; R_{x_2 \leftarrow z}\;. \tag{A229}$$

Alternative 1:

$$R_{x_1 \leftarrow z} \;=\; 0.5\,R_z \tag{A230}$$

$$R_{x_2 \leftarrow z} \;=\; 0.5\,R_z\;. \tag{A231}$$

The relevance is equally distributed.
Preferred Solution:
Alternative 2: The contributions according to the deep Taylor decomposition around (a, a) are

$$\left.\frac{\partial z}{\partial x_1}\right|_{(a,a)} (x_1 - a) \;=\; (x_1 - a)\,a\;, \tag{A232}$$

$$\left.\frac{\partial z}{\partial x_2}\right|_{(a,a)} (x_2 - a) \;=\; a\,(x_2 - a)\;. \tag{A233}$$

We compute the relative contributions:

$$\frac{(x_1 - a)\,a}{(x_1 - a)\,a \;+\; a\,(x_2 - a)} \;=\; \frac{x_1 - a}{(x_1 \;+\; x_2 \;-\; 2\,a)}\;, \tag{A234}$$

$$\frac{(x_2 - a)\,a}{(x_1 - a)\,a \;+\; a\,(x_2 - a)} \;=\; \frac{x_2 - a}{(x_1 \;+\; x_2 \;-\; 2\,a)}\;. \tag{A235}$$

For $\lim_{a \to 0}$ we obtain $x_1/(x_1 + x_2)$ and $x_2/(x_1 + x_2)$ as contributions.
We use this idea but scale x_1 and x_2 to the range $[0, 1]$:

$$R_{x_1 \leftarrow z} \;=\; \frac{\frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}}{\frac{x_1 - x_{\min}}{x_{\max} - x_{\min}} \;+\; \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}}\,R_z \tag{A236}$$

$$R_{x_2 \leftarrow z} \;=\; \frac{\frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}}{\frac{x_1 - x_{\min}}{x_{\max} - x_{\min}} \;+\; \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}}\,R_z\;. \tag{A237}$$

The relevance is distributed according to how close the maximal value is achieved and how far away it is from the minimal value.
Alternative 3:

$$R_{x_1 \leftarrow z} \;=\; \frac{\ln\left(1 - \frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}\right)}{\ln\left(1 - \frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}\right) \;+\; \ln\left(1 - \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}\right)} \, R_z \tag{A238}$$

$$R_{x_2 \leftarrow z} \;=\; \frac{\ln\left(1 - \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}\right)}{\ln\left(1 - \frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}\right) \;+\; \ln\left(1 - \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}\right)} \, R_z \; . \tag{A239}$$

All $\ln$-values are negative, therefore the fraction in front of R_z is positive. $x_1 = x_{\min}$ leads to a zero relevance for x_1. The ratio of the relevance for x_1 increases to 1 when x_1 approaches $x_{\max}$. The relevance is distributed according to how close the maximal value is achieved. We assume that the maximal value is a saturating value, therefore we use $\ln$, the natural logarithm.

A3.2 Input Zeroing

Input Zeroing sets a particular input x_i to zero and then computes the network output. If the original input was $\boldsymbol{x} = (x_1, \ldots, x_d)$ and the input with $x_i = 0$ is $\tilde{\boldsymbol{x}}_i = (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_d)$, then we compute $\Delta x_i = F(\boldsymbol{x}) \;-\; F(\tilde{\boldsymbol{x}}_i)$ to obtain the contribution of x_i. We obtain for the difference of $F(\boldsymbol{x})$ to the baseline of average zeroing $\frac{1}{d} \sum_{i=1}^{d} F(\tilde{\boldsymbol{x}}_i)$:

$$F(\boldsymbol{x}) \;-\; \frac{1}{d} \sum_{i=1}^{d} F(\tilde{\boldsymbol{x}}_i) \;=\; \frac{1}{d} \sum_{i=1}^{d} \Delta x_i \; . \tag{A240}$$

Problem is that $F(\tilde{\boldsymbol{x}}_i)$ have to be computed d-times, that is, for each input component zeroed out.

A3.3 Integrated Gradients

Integrated gradients is a recently introduced method [103]. Integrated gradients decomposes the difference $F(\boldsymbol{x}) - F(\tilde{\boldsymbol{x}})$ between the network output $F(\boldsymbol{x})$ and a baseline $F(\tilde{\boldsymbol{x}})$.

$$\begin{aligned} F(\boldsymbol{x}) \;-\; F(\tilde{\boldsymbol{x}}) \;&=\; \sum_{i=1}^{d} (x_i \;-\; \tilde{x}_i) \int_{t=0}^{1} \frac{\partial F}{\partial x_i}(\tilde{\boldsymbol{x}} + t(\boldsymbol{x} - \tilde{\boldsymbol{x}}))\, \mathrm{d}t \\ &\approx\; \sum_{i=1}^{d} (x_i \;-\; \tilde{x}_i) \, \frac{1}{m} \sum_{k=1}^{m} \frac{\partial F}{\partial x_i}(\tilde{\boldsymbol{x}} + (k/m)(\boldsymbol{x} - \tilde{\boldsymbol{x}})) \; . \end{aligned} \tag{A241}$$

In contrast to previous approach we have F and its derivative to evaluate only m-times, where $m < d$. The equality can be seen if we define $\boldsymbol{h} = \boldsymbol{x} - \tilde{\boldsymbol{x}}$ and

$$\begin{cases} g : [0, 1] \to \mathbb{R} \\ g(t) = F(\boldsymbol{x} + t\boldsymbol{h}) \end{cases} \tag{A242}$$

then we have

$$F(\boldsymbol{x} + \boldsymbol{h}) - F(\boldsymbol{x}) \;=\; g(1) - g(0) \;=\; \int_0^1 g'(t)\, \mathrm{d}t \tag{A243}$$

$$= \int_0^1 \left(\sum_{i=1}^{d} \frac{\partial F}{\partial x_i}(\boldsymbol{x} + t\boldsymbol{h})\, h_i \right) \mathrm{d}t \;=\; \sum_{i=1}^{d} \left(\int_0^1 \frac{\partial F}{\partial x_i}(\boldsymbol{x} + t\boldsymbol{h})\, \mathrm{d}t \right) \, h_i \; . \tag{A244}$$

For the final reward decomposition, we obtain

$$F(\boldsymbol{x}) \;=\; \sum_{i=1}^{d} \left((x_i \;-\; \tilde{x}_i) \, \frac{1}{m} \sum_{k=1}^{m} \frac{\partial F}{\partial x_i}(\tilde{\boldsymbol{x}} + (k/m)(\boldsymbol{x} - \tilde{\boldsymbol{x}})) \;+\; \frac{1}{d} \, F(\tilde{\boldsymbol{x}}) \right) \; . \tag{A245}$$

A4 Long Short-Term Memory (LSTM)

A4.1 LSTM Introduction

Recently, *Long Short-Term Memory* (LSTM; 41, 46, 47) networks have emerged as the best-performing technique in speech and language processing. LSTM networks have been overwhelming successful in different speech and language applications, including handwriting recognition [31], generation of writings [32], language modeling and identification [29, 126], automatic language translation [104], speech recognition [92, 23] analysis of audio data [68], analysis, annotation, and description of video data [17, 115, 102]. LSTM has facilitated recent benchmark records in TIMIT phoneme recognition (Google), optical character recognition, text-to-speech synthesis (Microsoft), language identification (Google), large vocabulary speech recognition (Google), English-to-French translation (Google), audio onset detection, social signal classification, image caption generation (Google), video-to-text description, end-to-end speech recognition (Baidu), and semantic representations. In the proceedings of the flagship conference *ICASSP 2015* (40th IEEE International Conference on Acoustics, Speech and Signal Processing, Brisbane, Australia, April 19–24, 2015), 13 papers had "LSTM" in their title, yet many more contributions described computational approaches that make use of LSTM.
The key idea of LSTM is the use of memory cells that allow for constant error flow during training. Thereby, LSTM avoids the *vanishing gradient problem*, that is, the phenomenon that training errors are decaying when they are back-propagated through time [41, 44]. The vanishing gradient problem severely impedes *credit assignment* in recurrent neural networks, i.e. the correct identification of relevant events whose effects are not immediate, but observed with possibly long delays. LSTM, by its constant error flow, avoids vanishing gradients and, hence, allows for *uniform credit assignment*, i.e. all input signals obtain a similar error signal. Other recurrent neural networks are not able to assign the same credit to all input signals, therefore they are very limited concerning the solutions they will find. Uniform credit assignment enabled LSTM networks to excel in speech and language tasks: if a sentence is analyzed, then the first word can be as important as the last word. Via uniform credit assignment, LSTM networks regard all words of a sentence equally. Uniform credit assignment enables to consider all input information at each phase of learning, no matter where it is located in the input sequence. Therefore, uniform credit assignment reveals many more solutions to the learning algorithm which would otherwise remain hidden.

A4.2 LSTM in a Nutshell

The central processing and storage unit for LSTM recurrent networks is the *memory cell*. As already mentioned, it avoids vanishing gradients and allows for uniform credit assignment. The most commonly used LSTM memory cell architecture in the literature [33, 95] contains forget gates [25, 26] and peephole connections [24]. In our previous work [49, 45], we found that peephole connections are only useful for modeling time series, but not for language, meta-learning, or biological sequences. That peephole connections can be removed without performance decrease, was recently confirmed in a large assessment, where different LSTM architectures have been tested [34]. While LSTM networks are highly successful in various applications, the central memory cell architecture was not modified since 2000 [95]. A memory cell architecture without peepholes is depicted in Figure A3.
In our definition of a LSTM network, all units of one kind are pooled to a vector: $\boldsymbol{z}$ is the vector of cell input activations, $\boldsymbol{i}$ is the vector of input gate activations, $\boldsymbol{f}$ is the vector of forget gate activations, $\boldsymbol{c}$ is the vector of memory cell states, $\boldsymbol{o}$ is the vector of output gate activations, and $\boldsymbol{y}$ is the vector of cell output activations. We assume to have an input sequence, where the input vector at time t is $\boldsymbol{x}^t$. The matrices $\boldsymbol{W_z}$, $\boldsymbol{W_i}$, $\boldsymbol{W_f}$, and $\boldsymbol{W_o}$ correspond to the weights of the connections between inputs and cell input, input gate, forget gate, and output gate, respectively. The vectors $\boldsymbol{b_z}$, $\boldsymbol{b_i}$, $\boldsymbol{b_f}$, and $\boldsymbol{b_o}$ are the bias vectors of cell input, input gate, forget gate, and output gate, respectively. The activation functions are g for the cell input, h for the cell state, and σ for the gates, where these functions are evaluated in a component-wise manner if they are applied to vectors. Typically, either the sigmoid $\frac{1}{1+\exp(-x)}$ or $\tanh$ are used as activation functions. $\odot$ denotes the point-wise multiplication of two

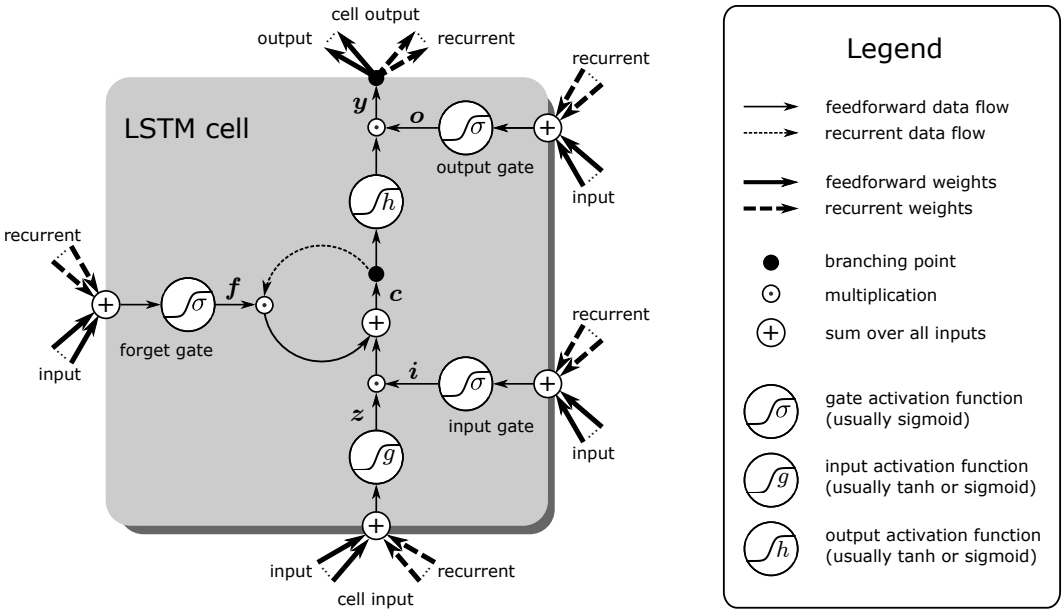


Figure A3: LSTM memory cell without peepholes. $\boldsymbol{z}$ is the vector of cell input activations, $\boldsymbol{i}$ is the vector of input gate activations, $\boldsymbol{f}$ is the vector of forget gate activations, $\boldsymbol{c}$ is the vector of memory cell states, $\boldsymbol{o}$ is the vector of output gate activations, and $\boldsymbol{y}$ is the vector of cell output activations. The activation functions are g for the cell input, h for the cell state, and σ for the gates. Data flow is either "feed-forward" without delay or "recurrent" with a one-step delay. "Input" connections are from the external input to the LSTM network, while "recurrent" connections take inputs from other memory cells and hidden units of the LSTM network with a delay of one time step.

vectors. Without peepholes, the LSTM memory cell forward pass rules are (see Figure A3):

$$\boldsymbol{z}^t = g\left(\boldsymbol{W_z}\, \boldsymbol{x}^t + \boldsymbol{b_z}\right) \qquad \text{cell input} \qquad \text{(A246)}$$

$$\boldsymbol{i}^t = \sigma\left(\boldsymbol{W_i}\, \boldsymbol{x}^t + \boldsymbol{b_i}\right) \qquad \text{input gate} \qquad \text{(A247)}$$

$$\boldsymbol{f}^t = \sigma\left(\boldsymbol{W_f}\, \boldsymbol{x}^t + \boldsymbol{b_f}\right) \qquad \text{forget gate} \qquad \text{(A248)}$$

$$\boldsymbol{c}^t = \boldsymbol{i}^t \odot \boldsymbol{z}^t + \boldsymbol{f}^t \odot \boldsymbol{c}^{t-1} \qquad \text{cell state} \qquad \text{(A249)}$$

$$\boldsymbol{o}^t = \sigma\left(\boldsymbol{W_o}\, \boldsymbol{x}^t + \boldsymbol{b_o}\right) \qquad \text{output gate} \qquad \text{(A250)}$$

$$\boldsymbol{y}^t = \boldsymbol{o}^t \odot h\left(\boldsymbol{c}^t\right) \qquad \text{cell output} \qquad \text{(A251)}$$

A4.3 Long-Term Dependencies vs. Uniform Credit Assignment

The LSTM network has been proposed with the aim to learn *long-term dependencies* in sequences which span over long intervals [47, 48, 42, 43]. However, besides extracting long-term dependencies, LSTM memory cells have another, even more important, advantage in sequence learning: as already described in the early 1990s, LSTM memory cells allow for *uniform credit assignment*, that is, the propagation of errors back to inputs without scaling them [41]. For uniform credit assignment of current LSTM architectures, the forget gate $\boldsymbol{f}$ must be one or close to one. A memory cell without an input gate $\boldsymbol{i}$ just sums up all the squashed inputs it receives during scanning the input sequence. Thus, such a memory cell is equivalent to a unit that sees all sequence elements at the same time, as has been shown via the "Ersatzschaltbild" [41]. If an output error occurs only at the end of the sequence, such a memory cell, via backpropagation, supplies the same delta error at the cell input unit $\boldsymbol{z}$ at every time step. Thus, all inputs obtain the same credit for producing the correct output and are treated on an equal level and, consequently, the incoming weights to a memory cell are adjusted by using the same delta error at the input unit $\boldsymbol{z}$.
In contrast to LSTM memory cells, standard recurrent networks scale the delta error and assign different credit to different inputs. The more recent the input, the more credit it obtains. The first inputs of the sequence are hidden from the final states of the recurrent network. In many learning tasks, however, important information is distributed over the entire length of the sequence and can

even occur at the very beginning. For example, in language- and text-related tasks, the first words are often important for the meaning of a sentence. If the credit assignment is not uniform along the input sequence, then learning is very limited. Learning would start by trying to improve the prediction solely by using the most recent inputs. Therefore, the solutions that can be found are restricted to those that can be constructed if the last inputs are considered first. Thus, only those solutions are found that are accessible by gradient descent from regions in the parameter space that only use the most recent input information. In general, these limitations lead to sub-optimal solutions, since learning gets trapped in local optima. Typically, these local optima correspond to solutions which efficiently exploit the most recent information in the input sequence, while information way back in the past is neglected.

A4.3.1 Special LSTM Architectures for Backward Analysis

LRP has already been used for LSTM in order to identify important terms in sentiment analysis [1]. Positive as well as negative terms in text could be identified.
For backward analysis applied to LSTM we make following assumptions:

- **(A1)** $\boldsymbol{f}^t = 1$ for all t. That is the forget gate is always 1 and nothing is forgotten. We assume uniform credit assignment, which is ensured by the forget gate set to one.
- **(A2)** $g > 0$, that is, g is positive. For example we can use a sigmoid $\sigma(x) = a_g \frac{1}{1+\exp(-x)}$: $g(x) = a_g \sigma(x)$, with $a_g = 2, 3, 4$. Methods like LRP have problems with negative contributions which cancel with positive contributions [74]. With a positive g all contributions are positive. The cell input $\boldsymbol{z}$ (the function g) has a negative bias, that is, $\boldsymbol{b_z} < 0$. This is important to avoid the drift effect. The drift effect is that the memory content only gets positive contributions which lead to an increase of $\boldsymbol{c}$ over time. Typical values are $\boldsymbol{b_z} = -1, -2, -3, -4, -5$.
- **(A3)** We want to ensure that $h(0) = 0$. If the memory content is zero than nothing is transferred to the next layer. Therefore we set $h = a_h \tanh$ with $a_h = 1, 2, 4$.
- **(A4)** The cell input gate $\boldsymbol{z}$ is only connected to the input but not to other memory cells. $\boldsymbol{W_z}$ has only connections to the input. This ensures that LRP assigns relevance $\boldsymbol{z}$ to the input and $\boldsymbol{z}$ is not disturbed by redistributing relevance to the network.
- **(A5)** The input gate $\boldsymbol{i}$ is not connected to the input, that is, $\boldsymbol{W_i}$ has only connections to other memory cells. This ensures that LRP assigns relevance only via $\boldsymbol{z}$ to the input.
- **(A6)** The output gate $\boldsymbol{o}$ is not connected to the input, that is, $\boldsymbol{W_o}$ has only connections to other memory cells. This ensures that LRP assigns relevance only via $\boldsymbol{z}$ to the input.
- **(A7)** The input gate $\boldsymbol{i}$ has a negative bias, that is, $\boldsymbol{b_i} < 0$. Like with the cell input the negative bias avoids the drift effect. Typical values are $\boldsymbol{b_i} = -1, -2, -3, -4$.
- **(A8)** The output gate $\boldsymbol{o}$ may also have a negative bias, that is, $\boldsymbol{b_o} < 0$. This allows to bring in different memory cells at different time points. It is related to resource allocation.
- **(A9)** The memory cell content is initialized with zero at time $t = 0$, that is, $\boldsymbol{c}^0 = 0$.
- **(A10)** The relevance for gates is zero according to previous work [1]. Thus, for relevance propagation the gate activations are treated as constants.

The resulting LSTM forward pass rules for LRP are:

$$\boldsymbol{z}^t = a_g \, \sigma\left(\boldsymbol{W_z} \, \boldsymbol{x}^t + \boldsymbol{b_z}\right) \qquad \text{cell input} \qquad \text{(A252)}$$

$$\boldsymbol{i}^t = \sigma\left(\boldsymbol{W_i} \, \boldsymbol{x}^t + \boldsymbol{b_i}\right) \qquad \text{input gate} \qquad \text{(A253)}$$

$$\boldsymbol{c}^t = \boldsymbol{i}^t \odot \boldsymbol{z}^t + \boldsymbol{c}^{t-1} \qquad \text{cell state} \qquad \text{(A254)}$$

$$\boldsymbol{o}^t = \sigma\left(\boldsymbol{W_o} \, \boldsymbol{x}^t + \boldsymbol{b_o}\right) \qquad \text{output gate} \qquad \text{(A255)}$$

$$\boldsymbol{y}^t = \boldsymbol{o}^t \odot a_h \, \tanh\left(\boldsymbol{c}^t\right) \qquad \text{cell output} \qquad \text{(A256)}$$

See Figure A4 which depicts these forward pass rules for LRP. However, gates may be used while no relevance is given to them which may lead to inconsistencies. See Figure A5 for a LSTM memory cell that is nondecreasing and complies to being Markov since current cell inputs are not modified by output or forget gates activated by subsequent input information. See Figure A6 for a LSTM memory cell without gates which perfectly distributes the relevance across the input.

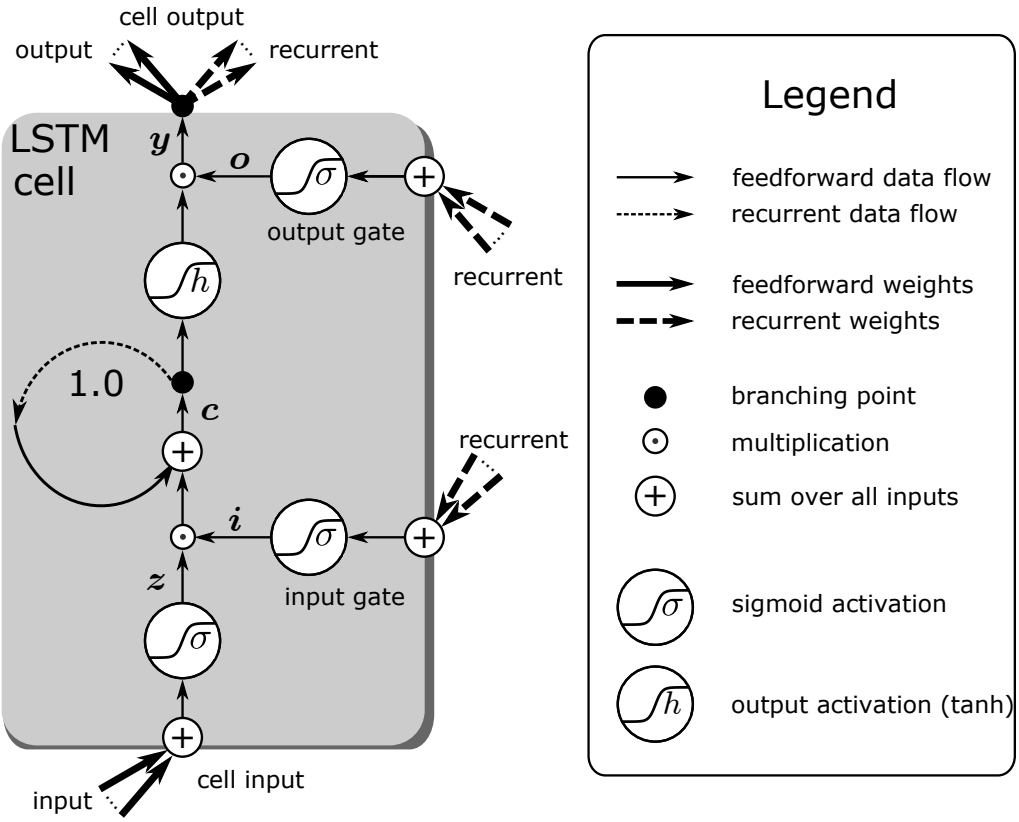


Figure A4: LSTM memory cell used for Layer-Wise Relevance Propagation (LRP). $\boldsymbol{z}$ is the vector of cell input activations, $\boldsymbol{i}$ is the vector of input gate activations, $\boldsymbol{c}$ is the vector of memory cell states, $\boldsymbol{o}$ is the vector of output gate activations, and $\boldsymbol{y}$ is the vector of cell output activations. The activation functions are the sigmoid $\sigma(x) = a_g \frac{1}{1+\exp(-x)}$ and the cell state activation $h(x) = a_h \tanh(x)$. Data flow is either "feed-forward" without delay or "recurrent" with a one-step delay. External input reaches the LSTM network only via the cell input $\boldsymbol{z}$. All gates only receive recurrent input, that is, from other memory cells.

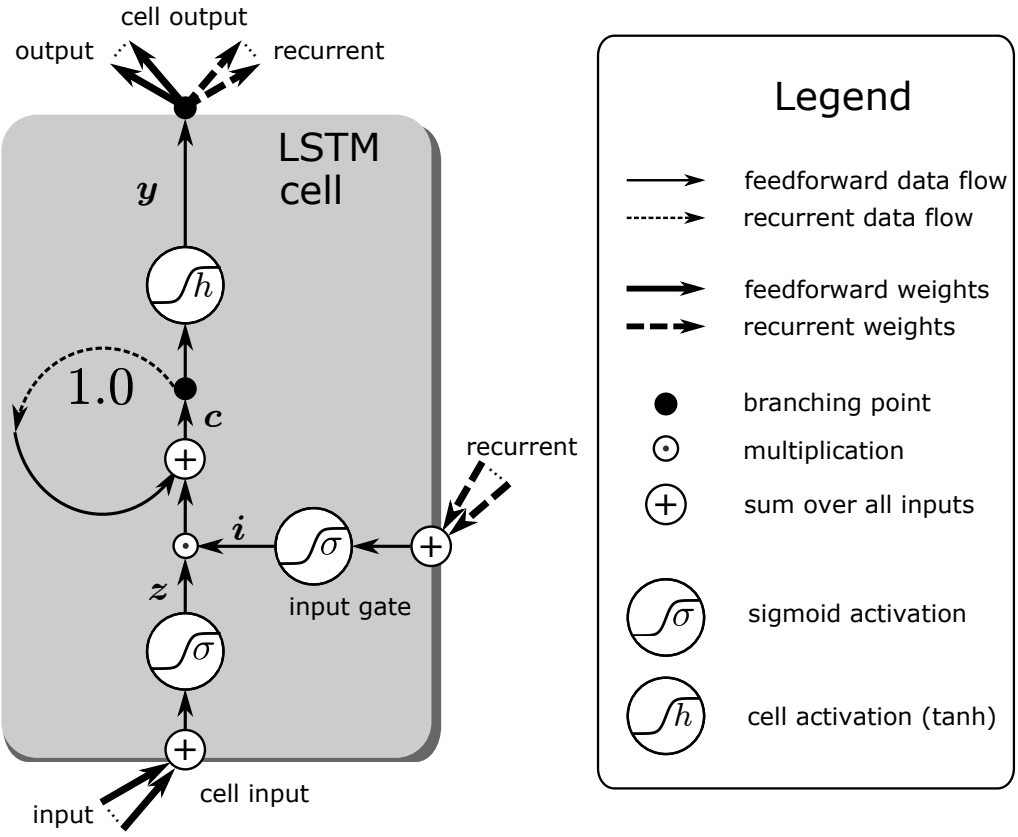


Figure A5: LSTM memory cell used for Layer-Wise Relevance Propagation (LRP). The memory cell is nondecreasing and guarantees the Markov property. Forget gates and output gates can modify all cell inputs at times after they have been observed, which violates the Markov property. $\boldsymbol{z}$ is the vector of cell input activations, $\boldsymbol{i}$ is the vector of input gate activations, $\boldsymbol{c}$ is the vector of memory cell states, and $\boldsymbol{y}$ is the vector of cell output activations. The activation functions are the sigmoid $\sigma(x) = a_g \frac{1}{1+\exp(-x)}$ and the cell state activation $h(x) = a_h \tanh(x)$. Data flow is either "feed-forward" without delay or "recurrent" with a one-step delay. External input reaches the LSTM network only via the cell input $\boldsymbol{z}$. The input gate only receive recurrent input, that is, from other memory cells.

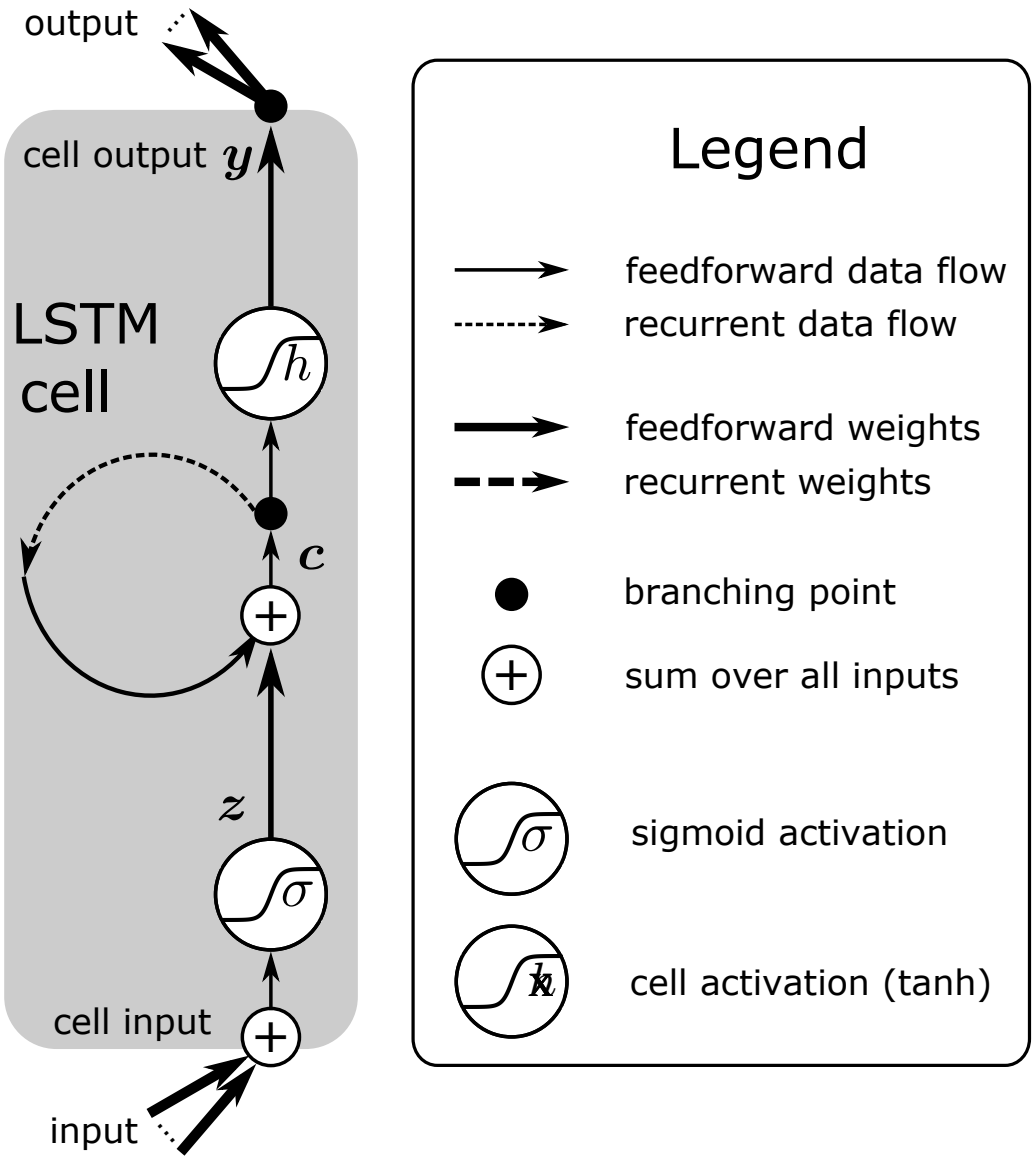


Figure A6: LSTM memory cell without gates used for Layer-Wise Relevance Propagation (LRP). $\boldsymbol{z}$ is the vector of cell input activations, $\boldsymbol{c}$ is the vector of memory cell states, and $\boldsymbol{y}$ is the vector of cell output activations. The activation functions are the sigmoid $\sigma(x) = a_g \frac{1}{1+\exp(-x)}$ and the cell state activation $h(x) = a_h \tanh(x)$. External input is stored in the memory cell via the input $\boldsymbol{z}$.

A4.3.2 LSTM Relevance and Contribution Propagation

We consider one memory cell:

$$c^t \;=\; i^t \; z^t \;+\; c^{t-1} \; . \tag{A257}$$

Here we treat i^t like a weight for z^t and c^{t-1} has weight 1.
Both relevance and contribution propagation leads to

$$R_{c^t \leftarrow y^t} \;=\; R_{y^t} \tag{A258}$$
$$R_{c^t} \;=\; R_{c^t \leftarrow c^{t+1}} \;+\; R_{c^t \leftarrow y^t} \tag{A259}$$
$$R_{c^{t-1} \leftarrow c^t} \;=\; \frac{c^{t-1}}{c^t} \; R_{c^t} \tag{A260}$$
$$R_{z^t \leftarrow c^t} \;=\; \frac{i^t \; z^t}{c^t} \; R_{c^t} \; . \tag{A261}$$

Since we predict only at the last step $t = T$, we have $R_{y^t} = 0$ for $t < T$. For $t = T$ we obtain $R_{c^T} = R_{y^T}$, since $R_{c^T \leftarrow c^{T+1}} = 0$.
We obtain for $t = 1 \ldots T$:

$$R_{c^T} \;=\; R_{y^T} \tag{A262}$$
$$R_{c^{t-1}} \;=\; \frac{c^{t-1}}{c^t} \; R_{c^t} \tag{A263}$$

which gives

$$R_{c^t} \;=\; R_{y^T} \prod_{\tau=t+1}^{T} \frac{c^{\tau-1}}{c^\tau} \;=\; \frac{c^t}{c^T} \; R_{y^T} \tag{A264}$$

and consequently as $c^0 = 0$ we obtain

$$R_{c^0} \;=\; 0 \, , \tag{A265}$$
$$R_{z^t} \;=\; \frac{i^t \; z^t}{c^T} \; R_{y^T} \; . \tag{A266}$$

Since we assume $c^0 = 0$, we have

$$c^T \;=\; \sum_{t=1}^{T} i^t \; z^t \tag{A267}$$

and therefore

$$R_{z^t} \;=\; \frac{i^t \; z^t}{\sum_{\tau=1}^{T} i^\tau \; z^\tau} \; R_{y^T} \; . \tag{A268}$$

Therefore the relevance R_{y^T} is distributed across the inputs z^t for $t = 1 \ldots T-1$, where input z^t obtains relevance R_{z^t}.

A5 Experiments

A5.1 Artificial examples

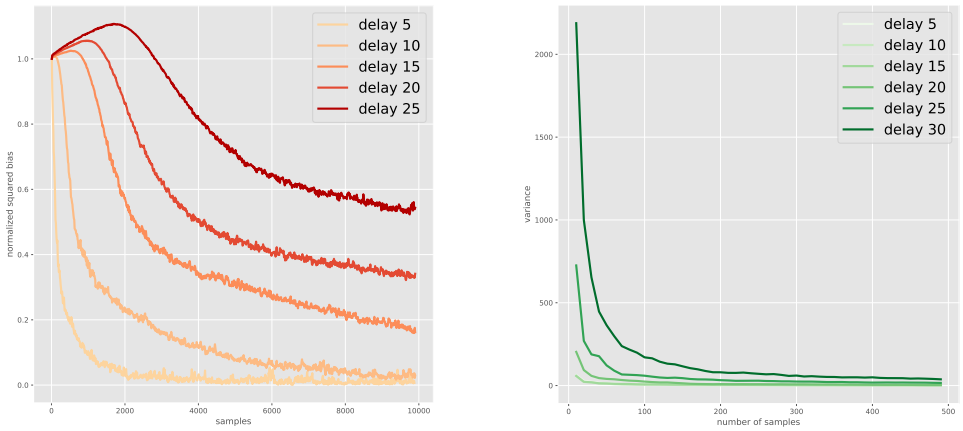
In this section we describe the experiments with artificial problems that support our theoretical analysis and show that RUDDER is exponentially faster than temporal difference and Monte Carlo methods.

A5.1.1 Grid World

The Grid World should illustrate an environment where an agent must run to a bomb and defuse it. Subsequently it must run away in case the bomb still explodes. The shorter the path to the bomb was, the further away the agent can run. An alternative strategy is to directly run away, which, however, leads to less return than defusing the bomb. The Grid World task consists of a quadratic 31×31 grid with *bomb* at coordinate $(30, 15)$ and *start* at $(30 - d, 15)$, where d is the delay of the task. The agent can move in four different directions (*up*, *right*, *left*, and *down*). Only moves that keep the agent on the grid are allowed, e.g. if the agent is at x-position 1, a move to the left is not possible. The episode finishes after $1.5d$ steps. At the end of the episode the agent receives a reward of 1000 if it has visited *bomb*. At each time step the agent receives an immediate reward of $c \cdot t \cdot h$, where c is a factor that depends on the chosen action, t is the current time step, and h is the Hamming distance to *bomb*. Each move the agent reduces the Hamming distance to *bomb* is penalized by the immediate reward using $c = -0.09$. Each move the agent increases the Hamming distance to *bomb* is rewarded by the immediate reward using $c = 0.1$. Due to this distracting reward the agent is forced to learn the Q-values precisely, since the immediate reward hints at a sub-optimal policy. This is because the learning process has to determine that visiting *bomb* leads to larger Q-values than increasing the distance to *bomb* for a particular fixed policy.
For non-deterministic reward, the agent receives the delayed reward for having visited *bomb* with probability $p(r_{T+1} = 100 \mid s_T, a_T)$. For non-deterministic transitions, the probability of transiting to next state s' is $p(s' \mid s, a)$. For the deterministic environment these probabilities were either 1 or zero.

Policy evaluation: learning action-value estimator for a fixed policy. First, the theoretical statements on bias and variance of estimating the action-values by TD in Theorem A2 and by MC in Theorem A3 are verified for a fixed policy. Secondly, we consider the bias and variance of TD and MC estimators of the transformed MDP with optimal return decomposition according to Theorem A4. The new MDP with the optimal return decomposition has advantages over the original MDP both for TD and MC. For TD the new MDP corrects the bias faster and for MC it has fewer number of action-values with high variance. Consequently, estimators for the new MDP learn exponentially faster than the same estimators in the original MDP.
Since the bias-variance analysis is defined for a particular number of samples drawn from a fixed distribution, we need to fix the policy for sampling. We use an ϵ-greedy version of the optimal policy, where ϵ is chosen that on average in 10% of the episodes the agent visits *bomb*. For the analysis, the delay ranges from 5 to 30 in steps of 5. The true Q-table for each delay is computed by backward induction and we use 10 different action-value estimators for computing bias and variance.
For the TD update rule we use the exponential average that is sample-updates, with initial value $q^0(s, a) = 0$. We only monitor the mean and the variance for action-value estimators at the first time step, since we are interested in the time required for correcting the bias, 10 different estimators were run for 10,000 episodes. Figure A7a shows the bias correction for different delays, normalized by the first error.
For the MC update rule we used the arithmetic mean for policy evaluation (later we will use constant-α MC for learning the optimal policy). For each delay, a test set of state-actions for each delay is generated by drawing 5,000 episodes with the ϵ-greedy optimal policy. For each action-value estimator the mean and the variance is monitored every 10 visits. If every action-value has 500 updates (visits), learning is stopped. Bias and variance are computed based on 10 different action-value estimators. As expected from Appendix A1.2.1, in Figure A7b the variance decreases by $1/n$, where n is the number of samples. Figure 1 shows that the number of state-actions with a variance larger than a threshold increases exponentially with the delay. This confirms the statements of Theorem A3.

Learning the optimal policy For finding the optimal policy for the Grid World task, we applied Monte Carlo Tree Search (MCTS), Q-learning, and Monte Carlo (MC). We trained a greedy policy until it reaches 90% of the return of the optimal policy. We used *sample updates* for Q-learning and MC [107]. For MCTS the greedy policy uses 0 for the exploration constant in UCB1 [57] . The



(a) (b)

Figure A7: Experimental evaluation of bias and variance of different Q-value estimators on the Grid World. Left: Normalized bias reduction for different delays. Right: Average variance reduction for the 10th highest values.

greedy policy was evaluated in 100 episodes intervals. For MCTS the learning time was measured by the number of model evaluations, while for MC and Q-learning we used environment steps. Model evaluations and environment steps should correspond to each other, since either the environment or its model is executed. The MCTS selection step begins in the start state, which is the root of the game tree that is traversed using UCB1 [57] as the tree policy. If a tree-node is visited the first time, it is expanded with an initial value obtained by 100 simulated trajectories that start at this node. These simulations use a uniform random policy whose average Return is calculated. The backpropagation step uses the maxMCTS(1) update rule [54]. The tree policies exploration constant is $\sqrt{2}$. Q-learning and MC use a learning rate of 0.3 and an ϵ-greedy policy with $\epsilon = 0.3$. For RUDDER the optimal Return Decomposition as stated in Appendix A1.3.3 was used. For each *delay* and each method, 300 runs with different seeds were performed to obtain statistically relevant results.

Estimation of the average learning time. The performance of different methods is measured by the average learning time in model evaluations and environment steps. We stopped training at 100 million environment steps. Some runs were not finished and the average learning time must be estimated by fitting a distribution using right censored data [27] .We found that for delays where all runs were finished the learning time follows a Log-normal distribution. Therefore we fitted a Log-normal distribution on the right censored data. We used maximum likelihood estimation to find the distributions parameters.

A5.1.2 Charge-Discharge

The Charge-Discharge task depicted in Figure A8 consists of two basic states *discharged* `D` / *charged* `C` and two actions *discharge* `d` / *charge* `c`. The deterministic reward is $r(\texttt{D},\texttt{d}) = 1, r(\texttt{C},\texttt{d}) = 10, r(\texttt{D},\texttt{c}) = 0$, and $r(\texttt{D},\texttt{c}) = 0$. The reward 10 for discharging a charged state is only given at the end of an episode at time $T+1$ for $T \in \{3,\ldots,13\}$, which determines the maximal delay of a reward. The deterministic state transitions are $(\{\texttt{D},\texttt{C}\},\texttt{d}) \to \texttt{D}$ and $(\{\texttt{D},\texttt{C}\},\texttt{c}) \to \texttt{C}$. To ensure Markov properties, we included the currently accumulated delayed reward and the current time point t in the state which contains the grid coordinates.
The optimal policy for the Charge-Discharge task alternates between charging and discharging to accumulate a delayed reward of 10 every other time step. However, the delayed accumulated reward will only be given at sequence end T. The smaller immediate reward of 1 distracts the agent from the larger delayed reward. The distraction forces the agent to learn the value function well enough to distinguish between the contribution of the immediate and the delayed reward to the final return.
For this task, the RUDDER backward analysis is based on monotonic LSTMs as described in Section A4.3.1 and on layer-wise relevance propagation (LRP) as described in Section A3.1. The

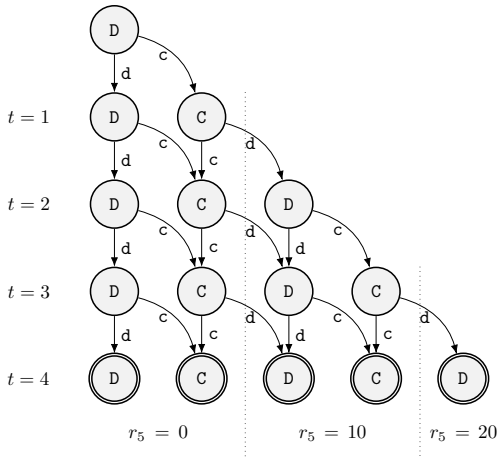


Figure A8: The Charge-Discharge task with two basic states discharged `D` and charged `C`. In each state the actions discharge `d` leading to the discharged state `D` and charge `c` leading to charged state `C` are possible. Action `d` in the discharged state `D` leads to a small immediate reward of 1 and in charged state `C` to a delayed reward of 10. After sequence end $T = 4$, the accumulated delayed reward $r_{T+1} = r_5$ is given.

reward redistribution provided by RUDDER uses an LSTM which consists of 5 memory cells and was trained with Adam and a learning rate of 0.01. The reward redistribution was used to learn an optimal policy by Q-learning and by MC with a learning rate of 0.1 and an exploration rate of 0.1. Again, we used *sample updates* for Q-learning and MC [107]. The learning was stopped either if the agent achieved 90% of the reward of the optimal policy or after a maximum number of 10 million episodes. For each T and each method, 100 runs with different seeds were performed to obtain statistically relevant results. For runs which did not finish within 100m environment steps we estimated parameters like described in A5.1.2.

A5.2 RUDDER Implementation for Atari Games

In this section we describe the implementation of RUDDER for Atari games. The implementation is largely based on the *OpenAI baselines* package [16] for the RL components and our *TeLL* package [121] for the LSTM reward redistribution model. Standard input processing, such as skipping or stacking frames, is performed by the baseline package, if not specified otherwise. Source code is available at `https://github.com/ml-jku/baselines-rudder`.

A5.2.1 Architecture

A2C architecture. The design of the policy and the value network relies on the *ppo2* implementation [16], which is depicted in Figure A9 and summarized in Table A1. Contrary to the original implementation of PPO [96], the policy and the value function network share parameters in 3 convolution layers with ReLU activation functions, followed by a fully connected layer with ReLU activation functions and 2 output units for policy and value function prediction, respectively. Striding is applied in the convolutional layers to downscale the number of features. Network input are 4 stacked Atari game frames [72]. The vision features trained through the LSTM serve as additional input to the fully connected layer; however, no gradient from the policy or value function is propagated on these connections.

Reward redistribution model. Core of the reward redistribution LSTM model is a fully connected memory cell layer containing 64 memory cells with sigmoid gate activations, tanh input nonlinearities, and linear output functions, as illustrated in Figure A9 and summarized in Table A1. Trainable input features to the memory cell layer are 3 convolutional layers, 2 of which process single frames or delta frame (pixel-wise differences between consecutive frames) before being combined via the third convolutional layer. Two-by-two max-pooling is applied to reduce the number of features, additionally to striding in the 2 lower convolutional layers. Max-pooling was favored in this setup, as stacked frames appeared more robust to heavy striding than single or delta frames. Additionally, the memory cell layer receives the vision features of the A2C network (which are not adjusted by LSTM), the current action, and the approximate in-game time. After the memory cell layer, the

LSTM has 4 output nodes for the prediction of the return G_0 and the auxiliary tasks as described in Section A5.2.2.

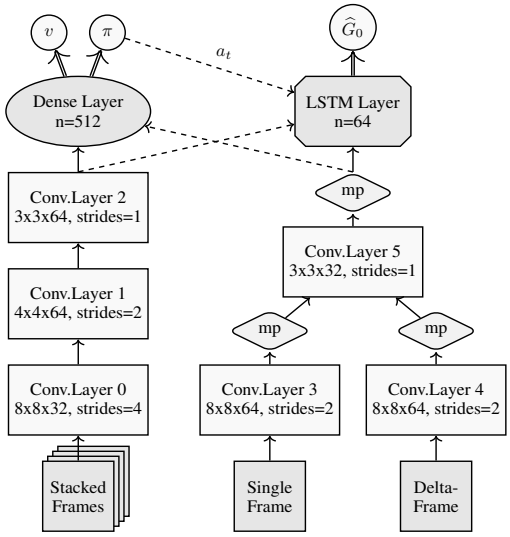


Figure A9: RUDDER models for Atari games as described in Section A5.2.1. Left: The *ppo2* implementation [16]. Right: LSTM reward redistribution model. Each network has access to the other vision features (dashed lines) but no gradient is propagated between the networks. The LSTM memory cell layer receives the current action and an approximate in-game-time as additional input.

Layer	Specifications		Layer	Specifications	
Conv.Layer 0	features	32	Conv.Layer 4	features	64
	kernelsize	8x8		kernelsize	8x8
	striding	4x4		striding	2x2
	act	ReLU		act	ReLU
	initialization	orthogonal, `gain`=$\sqrt{2}$		initialization	orthogonal, `gain`=0.1
Conv.Layer 1	features	64	Conv.Layer 5	features	32
	kernelsize	4x4		kernelsize	3x3
	striding	2x2		striding	1x1
	act	ReLU		act	ReLU
	initialization	orthogonal, `gain`=$\sqrt{2}$		initialization	orthogonal, `gain`=0.1
Conv.Layer 2	features	64	LSTM Layer	cells	64
	kernelsize	3x3		gate act.	sigmoid
	striding	1x1		ci act.	tanh
	act	ReLU		output act.	linear
	initialization	orthogonal, `gain`=$\sqrt{2}$		bias ig	trunc.norm., `mean`= -5
Dense Layer	features	512		bias og	trunc.norm., `mean`= -5
	act	ReLU		bias ci	trunc.norm., `mean`= 0
	initialization	orthogonal, `gain`=$\sqrt{2}$		bias fg	trunc.norm., `mean`= 12
Conv.Layer 3	features	64		fwd.w. ig	trunc.norm., `scale`= 0.1
	kernelsize	8x8		fwd.w. og	trunc.norm., `scale`= 0.1
	striding	2x2		fwd.w. ci	trunc.norm., `scale`= 0.0001
	act	ReLU		fwd.w. fg	trunc.norm., `scale`= 0.1
	initialization	orthogonal, `gain`=0.1		rec.w. ig	trunc.norm., `scale`= 0.001
				rec.w. og	trunc.norm., `scale`= 0.001
				rec.w. ci	trunc.norm., `scale`= 0.001
				rec.w. fg	trunc.norm., `scale`= 0.001

Table A1: RUDDER models for Atari games. Specifications of layers shown in Figure A9. Truncated normal initialization has `mean`= 0, `stddev`= 1 for all weights and biases, except for the LSTM biases (gates and cell-input) where it has `stddev`= 0.1 and is multiplied by a factor `scale`.

A5.2.2 Game Processing and Update/Target Design

The original *ppo2* implementation [16] trains a set of agents in parallel. These agents play individual environments but share all weights, i.e. they are distinguished by random effects in the environment

or by exploration. The value function and policy network is trained on a batch of transitions sampled from the environment. However, RUDDER requires a whole episode to redistribute the reward. To this end, episode sequences are stored in a buffer until each actor has completed at least one episode. Subsequently, these completed episodes are enhanced with redistributed reward. The episodes are forwarded to the original training procedure for training the policy/value function network. The reward redistribution LSTM model is updated on each episode with a static learning rate of 10^{-4} and l_1 weight decay with a factor of 10^{-7}.
For the A2C module, the default learning rate is $2.5 \cdot 10^{-4}$ [16]. The learning rate is decayed over 200M training frames by a factor of $1.0 - (u-1)/n$, where u is the number of the update and n is the number of maximal updates. Games are divided into episodes, in which the loss of an agent's life is viewed as starting a new episode without resetting the environment [16].

Target and loss design. The policy/value function network contains a policy loss, a value function loss, and an entropy term [16]. The loss of the reward redistribution LSTM model is composed of the main task to predict the return G_0, i.e. the squared prediction loss of $\widehat{G}_0$ at the last time step of the episode, as well as 3 auxiliary tasks. The auxiliary tasks are introduced for two reasons. First they push the LSTM model toward performing an optimal reward redistribution. Secondly they overcome problems of Markov properties in the environment which hinder storing important events since they are coded in later states. At every time step t, these auxiliary tasks are (i) the prediction of the action-value function q_t, (ii) the prediction of the reward in the next 10 frames $r_{t,10}$, and (iii) the so far accumulated reward $r_{0,t}$ at time step t. At sequence end T and for reward redistribution only the latter task is used. The squared losses of the auxiliary tasks are averaged over an episode (divided by $3T$). The final loss L for training the LSTM model is

$$
\begin{aligned}
r_{t,10} &= \sum_{i=t}^{t+10} r_i \,, \quad r_{0,t} = \sum_{i=0}^{t} r_i \,, \quad r_{t+1,T-t} = \sum_{i=t}^{T} r_{t+1} \,, \quad G_0 = \sum_{i=0}^{T+1} r_i \,, \\
L &= \left(G_0 - \widehat{G_0}\right)^2 \\
&\quad + \frac{1}{3\,T} \left(\sum_{t=0}^{T} \left(r_{t+1,T-t} - \widehat{r_{t+1,T-t}}\right)^2 + \sum_{t=0}^{T} \left(r_{t,10} - \widehat{r_{t,10}}\right)^2 + \sum_{t=0}^{T} \left(r_{0,t} - \widehat{r_{0,t}}\right)^2 \right) .
\end{aligned}
$$

We used hats to indicate predictions where all prediction are made at time t, except G_0 which is predicted at sequence end. r_0 serves to redistribute reward to the very beginning of the episode, e.g. the average reward of the current policy. The action-value function is $q_t = \mathrm{E}[r_{t+1,T-t}]$ and $r_{0,t}$ is the Q-value difference between immediate and delayed reward according to Proposition A1. The occasionally very high raw reward from the environment is divided by a constant factor of 100 for numerical stability.

Sequence junking. Due to memory restrictions and increase in computational complexity, sequences longer than 500 time steps were cropped into junks of 500 time steps and processed junk-wise. This junking does not affect the training targets of the reward redistribution model. Junks are overlapping with their adjacent junks by one half each to allow reward redistribution across junk boarders. The integrated gradients signals (see next paragraph) of the first halves of the junks are concatenated and used as redistributed reward for the whole episode. This procedure of overlapping junks and transferring redistributed reward reduces the effects of removing artifacts of integrated gradients at the end of episodes as described in next paragraph.

Reward redistribution. Integrated gradients [103] with a step-size of 500 is used for backward analysis of the LSTM prediction. Integrated gradients extract the contributions of the individual time steps to the prediction of the return at the last time step. Occasional strong artifacts due to numerical instabilities via strong non-linearities were observed at the last time steps of long episodes. These artifacts were removed by setting the integrated gradients signal for the last 10 timesteps of an episode to 0.
Furthermore, the quality of the integrated gradients and the prediction error of the LSTM were used to estimate the reliability of the reward redistribution. Reward redistribution is only used if both the prediction and the backward analysis were judged to be reliable for the current episode. The relative squared prediction error E is used to asses the reliability of the reward redistribution. E is computed

as follows using the current average of the return $\bar{G}_0$, the target G_0, and prediction y:

$$E = \frac{(G_0 - y)^2}{G_0^2 + \epsilon^2}, \qquad \epsilon^2 = \sqrt{\max\left\{\left|\bar{G}_0\right|, 10^{-5}\right\}}. \tag{A269}$$

The contribution of the redistributed reward is linearly down-scaled or omitted if the relative squared prediction error $E > 0.2$. The quality of integrated gradients is determined by the difference $F(\boldsymbol{x}) - F(\mathbf{0})$ and its approximation by the integrated gradients according to Eq. (A241). We omit the integrated gradients signal if the sum of integrated gradients is not within an interval from 80% to 120% error of the above difference.

A5.2.3 Exploration

Safe exploration to increase the likelihood to observe a delayed rewards is an important feature of RUDDER. For Atari games we use two sources of exploration which are described in the following.

Annealed *ppo2* exploration. The *ppo2* exploration in the baselines package [16] is annealed. Samples from a Gumble distribution are added to the output activations of the policy network before actions are chosen by argmax over the activations. The Gumble noise is downscaled by η w.r.t. the average sequence length, while ensuring that $1 \geqslant \eta = 0.01 \cdot \bar{l} \geqslant 10^{-6}$ with $\bar{l}$ denoting the average sequence length. Consequently, longer episodes of games are encouraged.

Safe exploration strategy. Safe exploration is realized by normalizing the output of the policy network and randomly picking one of the actions that is above a threshold θ given as ratio of the largest value. Safe exploration is activated once per sequence at a random sequence position for a random duration between 0 and the average game length $\bar{l}$. Thereby we encourage long on-policy sequences without dying interrupted by temporary safe off-policy trajectories resulting in high variance for certain parts of the game. θ is linearly increased from an agent with the strongest exploration $\theta = \theta_{\min}$ to an on-policy agent with $\theta = 1$.

A5.2.4 Lessons Replay Buffer

The lessons replay buffer is realized as simple priority-based buffer containing up to 64 episodes. New episodes are added to the buffer if (i) the buffer is not filled or if (ii) the loss of the LSTM model for the new sample exceeds the smallest loss in the buffer, in which case the new sample replaces the sample with the smallest loss. The lessons buffer is sampled via a softmax function of the losses of episodes in the buffer. After each update on an observed new episode, an update on an episode from the lessons buffer is performed. A2C learns from the lessons buffer, too. During training, after each episode sampled from the environment, one episode is sampled from the lessons buffer.

A6 References

[1] L. Arras, G. Montavon, K.-R. Müller, and W. Samek. Explaining recurrent neural network predictions in sentiment analysis. *CoRR*, abs/1706.07206, 2017.

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

[3] 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.

[4] 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.

[5] 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.

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

[7] F. Beleznay, T. Grobler, and C. Szepesvári. Comparing value-function estimation algorithms in undiscounted problems. Technical Report TR-99-02, Mindmaker Ltd., 1999.

[8] 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.

[9] 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.

[10] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3), 1991.

[11] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1996.

[12] I.-J. Bienaymé. Considérations àl'appui de la découverte de laplace. *Comptes Rendus de l'Académie des Sciences*, 37:309–324, 1853.

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

[14] I. J. Cox, R. Fu, and L. K. Hansen. Probably approximately correct search. In *Advances in Information Retrieval Theory*, pages 2–16. Springer, Berlin, Heidelberg, 2009.

[15] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341, 1992.

[16] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Openai baselines. https://github.com/openai/baselines, 2017.

[17] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *ArXiv*, 2014.

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

[19] 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.

[20] Z. Feinstein. Continuity properties and sensitivity analysis of parameterized fixed points and approximate fixed points. Technical report, Operations Research and Financial Engineering Laboratory, Washington University in St. Louis, 2016. preprint.

[21] 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).

[22] M. Frigon. Fixed point and continuation results for contractions in metric and Gauge spaces. *Banach Center Publications*, 77(1):89–114, 2007.

[23] J. T. Geiger, Z. Zhang, F. Weninger, B. Schuller, and G. Rigoll. Robust speech recognition using long short-term memory recurrent neural networks for hybrid acoustic modelling. In *Proc. 15th Annual Conf. of the Int. Speech Communication Association (INTERSPEECH 2014)*, pages 631–635, Singapore, September 2014.

[24] F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN 2000)*, volume 3, pages 189–194. IEEE, 2000.

[25] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN '99)*, pages 850–855, Edinburgh, Scotland, 1999.

[26] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Comput.*, 12(10):2451–2471, 2000.

[27] Irène Gijbels. Censored data. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(2):178–188, 2010.

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

[29] J. Gonzalez-Dominguez, I. Lopez-Moreno, H. Sak, J. Gonzalez-Rodriguez, and P. Moreno. Automatic language identification using long short-term memory recurrent neural networks. In *Proc. 15th Annual Conf. of the Int. Speech Communication Association (INTERSPEECH 2014)*, pages 2155–2159, Singapore, September 2014.

[30] 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.

[31] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(5):855–868, 2009.

[32] A. Graves, A.-R. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2013)*, pages 6645–6649, Vancouver, BC, 2013.

[33] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[34] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *ArXiv*, 2015.

[35] S. Grünewälder and K. Obermayer. The optimal unbiased value estimator and its relation to LSTD, TD and MC. *Machine Learning*, 83(3):289–330, 2011.

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

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

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

[39] 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.

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

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

[42] S. Hochreiter. Recurrent neural net learning and vanishing gradient. In C. Freksa, editor, *Proc. Fuzzy-Neuro-Systeme '97*, pages 130–137, Sankt Augustin, 1997. INFIX.

[43] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Internat. J. Uncertain. Fuzziness Knowledge-Based Systems*, 6(2):107–116, 1998.

[44] 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.

[45] S. Hochreiter, M. Heusel, and K. Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736, 2007.

[46] S. Hochreiter and J. Schmidhuber. Long short-term memory. Technical Report FKI-207-95, Fakultät für Informatik, Technische Universität München, 1995.

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

[48] S. Hochreiter and J. Schmidhuber. LSTM can solve hard long time lag problems. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 473–479, Cambridge, MA, 1997. MIT Press.

[49] S. Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proc. Int. Conf. on Artificial Neural Networks (ICANN 2001)*, pages 87–94. Springer, 2001.

[50] 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).

[51] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. John Wiley & Sons, New York, 2001.

[52] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

[53] J. Jachymski. Continuous dependence of attractors of iterated function systems. *Journal Of Mathematical Analysis And Applications*, 198(0077):221–226, 1996.

[54] P. Khandelwal, E. Liebman, S. Niekum, and P. Stone. On the analysis of complex backup strategies in Monte Carlo Tree Search. In *International Conference on Machine Learning*, pages 1319–1328, 2016.

[55] E. Kirr and A. Petrusel. Continuous dependence on parameters of the fixed point set for some set-valued operators. *Discussiones Mathematicae Differential Inclusions*, 17:29–41, 1997.

[56] 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.

[57] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer, 2006.

[58] 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.

[59] M. Kwiecinski. A note on continuity of fixed points. *Universitatis Iagellonicae Acta Mathematica*, 29:19–24, 1992.

[60] 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.

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

[62] 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.

[63] 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.

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

[65] G. Lugosi. Concentration-of-measure inequalities. In *Summer School on Machine Learning at the Australian National University,Canberra*, 2003. Lecture notes of 2009.

[66] 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.

[67] S. Mannor, D. Simester, P. Sun, and J. N. Tsitsiklis. Bias and variance approximation in value function estimates. *Management Science*, 53(2):308–322, 2007.

[68] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller. Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2014)*, pages 2164–2168, Florence, May 2014.

[69] V. A. Marčenko and L. A. Pastur. Distribution of eigenvalues or some sets of random matrices. *Mathematics of the USSR-Sbornik*, 1(4):457, 1967.

[70] 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.

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

[72] 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.

[73] 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.

[74] G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2017.

[75] 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.

[76] 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.

[77] B. O'Donoghue, I. Osband, R. Munos, and V. Mnih. The uncertainty Bellman equation and exploration. *ArXiv*, 2017.

[78] S. D. Patek. *Stochastic and shortest path games: theory and algorithms*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 1997.

[79] J. Peng and R. J. Williams. Incremental multi-step q-learning. *Machine Learning*, 22(1):283–290, 1996.

[80] 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.

[81] 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.

[82] 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.

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

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

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

[86] 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.

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

[88] 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.

[89] J. Romoff, A. Piché, P. Henderson, V. Francois-Lavet, and J. Pineau. Reward estimation for variance reduction in deep reinforcement learning. *ArXiv*, 2018.

[90] M. Rudelson and R. Vershynin. Non-asymptotic theory of random matrices: extreme singular values. *ArXiv*, 2010.

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

[92] H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Proc. 15th Annual Conf. of the Int. Speech Communication Association (INTERSPEECH 2014)*, pages 338–342, Singapore, September 2014.

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

[94] 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.

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

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

[97] 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.

[98] 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.

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

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

[101] A. Soshnikov. A note on universality of the distribution of the largest eigenvalues in certain sample covariance matrices. *J. Statist. Phys.*, 108(5-6):1033–1056, 2002.

[102] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using LSTMs. *ArXiv*, 2015.

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

[104] I. Sutskever, O. Vinyals, and Q. V. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

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

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

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

[108] 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.

[109] A. Tamar, D. DiCastro, and S. Mannor. Learning the variance of the reward-to-go. *Journal of Machine Learning Research*, 17(13):1–36, 2016.

[110] P. Tchebichef. Des valeurs moyennes. *Journal de mathématiques pures et appliquées 2*, 12:177–184, 1867.

[111] P. Tseng. Solving h-horizon, stationary Markov decision problems in time proportional to $\log(h)$. *Operations Research Letters*, 9(5):287–297, 1990.

[112] J. N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202, 1994.

[113] 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.

[114] 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.

[115] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. J. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. *ArXiv*, 2014.

[116] A. Veretennikov. Ergodic Markov processes and poisson equations (lecture notes). *ArXiv*, 2016.

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

[118] 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.

[119] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[120] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[121] Michael Widrich and Markus Hofmarcher. Tensorflow Layer Library (TeLL). https://github.com/ml-jku/tensorflow-layer-library, 2018.

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

[123] 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.

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

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

[126] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *ArXiv*, 2014.

[127] 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.