RUDDER: Return Decomposition for Delayed Rewards

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

Abstract

We propose RUDDER, a novel reinforcement learning approach for delayed rewards in finite Markov decision processes (MDPs). RUDDER is based on two main ideas: (i) a backward view approach and (ii) the concept of return-equivalent MDPs. Forward view approaches, like deep Q-networks (DQNs) or Monte Carlo tree search (MCTS), have to average over a large number of probabilistic future state-action paths that increases exponentially with the delay of the reward. Backward view approaches, in contrast, identify actions and states that cause a delayed reward by analyzing already chosen paths. RUDDER's backward view transforms tasks of estimating future returns into regression tasks at which deep learning excels. RUDDER decomposes the return into new, non-delayed rewards by redistributing the original reward across the episode, thereby creating a new MDP that is return-equivalent to the original MDP. "Return-equivalent MDPs" is a new concept ensuring that both MDPs have the same optimal policies. If the return decomposition is optimal, then the new MDP will be stripped of any delayed rewards. In this case, action-value estimates are unbiased and the future expected return is always zero. On several artificial tasks with delayed rewards RUDDER significantly outperforms Monte Carlo, MCTS, temporal difference, TD(λ), and reward shaping approaches. RUDDER is even exponentially faster than the last three. RUDDER on top of a Proximal Policy Optimization (PPO) baseline improves the scores on Atari games and excels for delayed rewards. For long delayed rewards, as in Bowling, Frostbite, PrivateEye, and Venture, RUDDER yields exceptional results.

1 Introduction

Assigning credit for a received reward to performed actions is one of the central tasks in reinforcement learning [115]. One of its great challenges is long-term credit assignment, which is required for delayed rewards [88, 68, 38, 98]. Delayed rewards are associated with episodic and sparse rewards, and therefore are common [92]. Classical reinforcement learning methods like temporal difference (TD), Monte Carlo (MC), and Monte Carlo Tree Search (MCTS) take a forward view approach[2] by estimating the expected future return [115]. MCTS is the core of AlphaZero [105, 106], which learned to play Chess and Go better than human professionals. The future return is estimated by rolling out games until their end and, therefore, capturing delayed rewards. Recently, also forward world models using an evolution strategy have been successful [35]. However, forward view approaches fail at MDPs with delayed rewards if rewards, state transitions, or policies are probabilistic and if states have high transition branching factors. They have to average over the returns of all possible

*authors contributed equally

[2]Not to be confused with forward view and backward view methods for eligibility traces [115].

trajectories starting at the current state or state-action pair. Averaging becomes a crucial problem since the number of possible trajectories increases exponentially with the delay of the reward.

Instead of using a forward view, we propose to adopt a backward view approach. A backward view approach can trace back from known goal states [17] or from high reward states [29]. A recent approach to backward view uses attention-based methods to retrieve past observations, which are relevant for future rewards [51]. We propose RUDDER (RetUrn Decomposition for DElayed Rewards) as a backward view approach based on a backward analysis of a forward model that predicts the return of an episode. As RUDDER considers only already completed episodes, it avoids problems with unknown future state-action paths and detects states and actions that caused the return. Backward analysis transforms the forward view approach into regression tasks, at which deep learning methods excel. We use as forward model a Long Short-Term Memory (LSTM) network [41, 47], which predicts the return of an episode, where an episode is a state-action sequence. An LSTM can bridge long time lags and identifies reward-causing state-action pairs even if they appear early in the episode [41, 44]. Best known is backward analysis via sensitivity analysis like "backpropagation through a model" [79, 93, 94, 5]. However, sensitivity analysis leads to local minima, instabilities, exploding or vanishing gradients, and poor exploration [40, 101]. To avoid these problems, RUDDER is based on backward analysis via contribution analysis like (A) differences of return predictions, (B) layer-wise relevance propagation (LRP) [3] or (C) integrated gradients (IG) [112] (see details in fourth par. in Sec. 2). These approaches decompose a predicted return into contributions of single state-actions along the observed sequence. By substituting the prediction with the actual return, we obtain a redistributed reward. The redistributed reward creates a new MDP with the same optimal policies as the original MDP. Redistributing the reward is fundamentally different from reward shaping [80, 129], look-ahead advice, and look-back advice [130]. Since the reward shaping approaches keep the original reward, their reward redistribution does not correspond to an optimal return decomposition according to Appendix A2.3.4. Consequently, reward shaping approaches are exponentially slower than RUDDER, as we demonstrate in the experiments in Section 3.

To learn delayed rewards, there are three phases to consider: (1) discovering the delayed reward (exploration), (2) keeping information about the delayed reward (buffer), (3) learning how to receive the delayed reward for the future. Deep Q-Networks (DQNs) [73, 74], based on Q-learning [127], use an *experience replay buffer* [66] to store interesting information. Sampling from the buffer has been improved via prioritized experience replay [100] and by parallel exploration as done by Ape-X DQN [50] or double DQN (DDQN) [121, 122]. Noisy DQNs [20] explore by a stochastic layer in the policy network (see also [40, 101]), while distributional Q-learning [7] enhances exploration by using higher moments of the Q-values. Policy gradient approaches [131] explore via parallel policies, too. A2C has been improved by IMPALA through parallel actors and correction for policy-lags between actors and learners [18]. A3C with asynchronous gradient descent [72] 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 [104]. RUDDER performs a return decomposition of an LSTM forward model to obtain a redistributed reward and an MDP that is return-equivalent to the original MDP. RUDDER consists of (I) a safe exploration strategy, (II) a lessons replay buffer, and, most importantly, (III) an LSTM for return decomposition and reward redistribution.

2 Return Decomposition and Reward Redistribution

MDP Definitions. 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 $q^\pi(s, a) = \mathrm{E}_\pi [G_t \mid S_t = s, A_t = a]$. The goal of learning is to maximize the expected return at time $t = 0$, that is $v_0^\pi = \mathrm{E}_\pi [G_0]$. The optimal policy π^* is $\pi^* = \operatorname{argmax}_\pi [v_0^\pi]$.

Return-Equivalent MDPs. A Markov decision process (MDP) $\tilde{\mathcal{P}}$ is *state-enriched* compared to an MDP $\mathcal{P}$ if $\tilde{\mathcal{P}}$ has the same states, actions, transition probabilities, and reward probabilities as $\mathcal{P}$ but additional information in its 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 [89, 90]. These properties are known from state abstraction and aggregation [65] and from bisimulations [27]. For more details see Appendix A2.3.1. Two MDPs $\tilde{\mathcal{P}}$ 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 $t = 0$: $\tilde{v}_0^\pi = v_0^\pi$.

Theorem 1. *Return-equivalent decision processes have the same optimal policies.*

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

Immediate Reward to Delayed Reward. We assume to have an MDP $\mathcal{P}$ with immediate rewards which is transformed to a state-enriched MDP $\tilde{\mathcal{P}}$, where the reward is only different from zero at the end of the sequence and, therefore, delayed. The transformed MDP has reward $\tilde{r}_t = 0$ for $t \leqslant T$, 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 in the Appendix 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.

Return Decomposition, Reward Redistribution: Delayed Reward to Immediate Reward. 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 want to redistribute the final reward $\tilde{r}_{T+1}$, which is the return, to previous time steps. The *return decomposition* idea is to predict $\tilde{r}_{T+1}$ by a function g from the state-action sequence, and subsequently redistribute $\tilde{r}_{T+1}$ over the sequence with the help of g. We want to determine for each sequence element its contribution to the prediction of $\tilde{r}_{T+1}$ at the end of the sequence. Therefore, we do backward analysis through contribution analysis. Contribution analysis computes the contribution of the current input to the final prediction, i.e. the information gain by the current input on the final prediction. In principle we can use any contribution analysis method. However, we prefer three methods: (A) differences of return predictions, (B) integrated gradients (IG) [112], and (C) layer-wise relevance propagation (LRP) [3]. For contribution method (A), we must ensure that g predicts the final reward $\tilde{r}_{T+1}$ at every time step. Hence, the change in prediction is a measure of the contribution of an input to the final prediction and assesses the information gain by this input. The redistributed reward is given by the difference of consecutive predictions. In contrast to method (A), methods (B) and (C) use information later in the sequence for determining the contribution of the current input. Thus, they introduce a non-Markovian reward as it depends on later sequence elements. However, the non-Markovian reward must be viewed as probabilistic reward, which is prone to have high variance. Therefore, we prefer method (A).

A problem with contribution analysis is that the Markov property ensures that $\tilde{r}_{T+1} = \tilde{r}_{T+1}(s_T, a_T)$ is a function of the last state-action pair (s_T, a_T). To eliminate this Markov property, we define a difference $\Delta(s_{t-1}, a_{t-1}, s_t, a_t)$ between a state-action pair (s_t, a_t) and its predecessor (s_{t-1}, a_{t-1}), where (s_{-1}, a_{-1}) is introduced for starting an episode. The sequence of differences is $\Delta_{0:T} := \big(\Delta(s_{-1}, a_{-1}, s_0, a_0), \ldots, \Delta(s_{T-1}, a_{T-1}, s_T, a_T)\big)$, where the components Δ are assumed to be statistically independent from each other. The function g predicts $\tilde{r}_{T+1}$ from the sequence $\Delta_{0:T}$: $g(\Delta_{0:T}) = \tilde{r}_{T+1}$ and is decomposed into $g(\Delta_{0:T}) = \sum_{t=0}^{T} h_t$. The contributions h_t are

$$h_t \;=\; h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \;\text{ for } 0 \leqslant t \leqslant T\,. \tag{1}$$

We have to account for the facts that the reward $\tilde{R}_{T+1}$ is probabilistic and that the function g is imperfect, that is, $g(\Delta_{0:T}) \neq \tilde{r}_{T+1}$. The compensation is given by

$$\tilde{R}_{T+1} \;-\; \sum_{\tau=0}^{T} h(\Delta(s_{\tau-1}, a_{\tau-1}, s_\tau, a_\tau)) \tag{2}$$

as an extra reward R_{T+2} at time $T+2$. It is given after R_{T+1}, while remaining in the state-action pair (s_T, a_T):

$$\begin{aligned} R_{t+1} &= h_t \text{ for } 0 \leqslant t \leqslant T \,, \\ R_{T+2} &= \tilde{R}_{T+1} - \sum_{t=0}^{T} h_t \,. \end{aligned} \tag{3}$$

The next theorem states that the MDP $\mathcal{P}$ with reward redistribution does not change the optimal policies of the original $\tilde{\mathcal{P}}$ since $\mathcal{P}$ is return-equivalent to the original MDP $\tilde{\mathcal{P}}$.

Theorem 2. *The MDP $\mathcal{P}$ with redistributed reward R_t has the same optimal policies as the delayed reward MDP $\tilde{\mathcal{P}}$.*

The proof can be found after Theorem A4 in the Appendix.

Optimal Return Decomposition. An *optimal return decomposition* is defined by the fact that partial sums

$$\sum_{\tau=0}^{t} h(\Delta(s_{\tau-1}, a_{\tau-1}, s_\tau, a_\tau)) = \tilde{q}^\pi(s_t, a_t) \tag{4}$$

predict the Q-values, too. For $g(\Delta_{0:T}) = \tilde{r}_{T+1}$ and with an optimal return decomposition, the reward redistribution is

$$\begin{aligned} R_{t+1} &= h_t = h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) = \tilde{q}^\pi(s_t, a_t) - \tilde{q}^\pi(s_{t-1}, a_{t-1}) \text{ for } 0 \leqslant t \leqslant T \,, \\ R_{T+2} &= \tilde{R}_{T+1} - \tilde{q}^\pi(s_T, a_T) \,, \end{aligned} \tag{5}$$

since $\sum_{t=0}^{T} h_t = \tilde{q}^\pi(s_T, a_T)$. Again we have ensured that $\sum_{t=0}^{T+1} R_{t+1} = \tilde{R}_{T+1}$. The Q-values of an MDP with reward redistribution based on an optimal return decomposition are equal to the expected immediate reward.

Theorem 3. *If the redistributed reward relies on an optimal return decomposition, the Q-values are given by*

$$\begin{aligned} q^\pi(s_t, a_t) &= r(s_t, a_t) = \tilde{q}^\pi(s_t, a_t) - \mathrm{E}_{s_{t-1}, a_{t-1}} \left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t \right] \\ &= \tilde{q}^\pi(s_t, a_t) - \psi(s_t) \end{aligned} \tag{6}$$

and the MDP $\mathcal{P}$ with reward redistribution and the delayed reward MDP $\tilde{\mathcal{P}}$ have the same advantage function.

The proof can be found after Theorem A5 in the Appendix. For an MDP with reward redistribution based on an optimal return decomposition, the expected future rewards are zero. A similar property is achieved by reward shaping if the exact value function is known [103].

In the experiments, we also use a uniform compensation, where each reward contributes equally to the compensation:

$$R_{t+1} = h_t + \frac{1}{T+1} \left(\tilde{R}_{T+1} - \sum_{\tau=0}^{T} h_\tau \right) \,, \tag{7}$$

where $h_\tau = h(\Delta(s_{\tau-1}, a_{\tau-1}, s_\tau, a_\tau))$. We now define the expected sum of delayed rewards $\kappa(m, t-1)$, which measures the amount of delayed reward.

Definition 1. *For $1 \leqslant t \leqslant T+1$ and $0 \leqslant m \leqslant T-t+1$, the expected sum of delayed rewards at time $(t-1)$ in the interval $[t, t+m]$ is defined as*

$$\kappa(m, t-1) = \mathrm{E}_\pi \left[\sum_{\tau=0}^{m} R_{t+1+\tau} \mid s_{t-1}, a_{t-1} \right] \,. \tag{8}$$

Q-values are decomposed by κ into immediate and delayed rewards: $q(s_t, a_t) = \mathrm{E}_{R_{t+1}} \left[R_{t+1} \mid s_t, a_t \right] + \kappa(T-t, t)$. The next theorem states necessary conditions for an optimal return decomposition based on κ.

Theorem 4. *An MDP $\mathcal{P}$ with a redistributed reward of an optimal return decomposition fulfills for $1 \leqslant t \leqslant T+1$ and $0 \leqslant m \leqslant T-t+1$ the necessary conditions for optimality:*

$$\kappa(m, t-1) \;=\; 0\,. \tag{9}$$

The proof can be found after Theorem A6 in the Appendix. Equation (9) states that an MDP with reward redistribution based on an optimal return decomposition has no delayed rewards. A related approach is to ensure zero return by reward shaping if the true value function is known [103]. Redistributing the reward via reward shaping, look-ahead advice, and look-back advice is a special case of a reward redistribution giving a return-equivalent MDP (see first par. in Sec. A2.3.1 in the Appendix). However, reward redistributions leading to a return-equivalent MDP can in general not be expressed as reward shaping. Furthermore, since reward shaping keeps the original reward, the reward redistribution via reward shaping, look-ahead advice, and look-back advice does not correspond to an optimal return decomposition. Therefore, TD can still be exponentially slow for delayed rewards.

Learning with the Redistributed Reward. RUDDER can be used with $\gamma = 1$ for (A) direct Q-value estimation, (B) policy gradients, and (C) Q-learning.
(A) Q-value estimation comprises (i) direct Q-value estimates if we assume optimal return decomposition, (ii) TD-learning of the expected sum of delayed rewards κ and (iii) eligibility traces for the redistributed reward. Here, the last two approaches try to compensate for non-optimal return decomposition. For (i) we assume optimality and obtain

$$q^\pi(s_t, a_t) \;=\; r(s_t, a_t) \;=\; \mathrm{E}_{R_{t+1}}\left[R_{t+1} \mid s_t, a_t\right]\,. \tag{10}$$

Thus, it is sufficient to estimate Q-values by the expected immediate reward. For (ii) we apply TD-learning to κ:

$$\begin{aligned} q^\pi(s_t, a_t) \;&=\; r(s_t, a_t) \;+\; \kappa(T, t)\,, \\ \kappa(T, t) \;&=\; \mathrm{E}_{s_{t+1}, a_{t+1}, R_{t+2}}\left[R_{t+2} + \kappa(T, t+1) \mid s_t, a_t\right]\,, \\ \delta_\kappa(T, t) \;&=\; R_{t+2} \;+\; \kappa(T, t+1) \;-\; \kappa(T, t)\,. \end{aligned} \tag{11}$$

The TD-error δ_κ is only valid if R_{t+2} and $\kappa(T, t+1)$ are drawn together, i.e. considered as pairs, and thus have the same (s_{t+1}, a_{t+1}). For (iii) we use eligibility traces. Optimality would ensure $\mathrm{E}_{s_{t+1}}[V(s_{t+1})] = 0$, leading to a recursion for the new reward $\mathcal{G}$:

$$\mathcal{G}_t \;=\; r_{t+1} \;+\; \lambda\, \mathcal{G}_{t+1}\,, \;\; \mathcal{G}_{T+2} \;=\; 0\,. \tag{12}$$

(B) Policy gradients have as expected updates $\mathrm{E}_\pi\left[\nabla_\theta \log \pi(a \mid s; \boldsymbol{\theta}) q^\pi(s, a)\right]$, where $q^\pi(s, a) = \mathrm{E}_\pi[G_t \mid s = s_t, a = a_t]$ is replaced during learning by a sample of the return G_t or an estimation of $q^\pi(s, a)$. Policy gradients replace $q^\pi(s, a)$ by the redistributed reward $r(s, a)$ assuming an optimal return decomposition. With baseline normalization, the baseline $b(s) = \mathrm{E}_\pi[r(s, a)] = \sum_a \pi(a \mid s) r(s, a)$ is subtracted from $r(s, a)$, which gives the policy gradient $\mathrm{E}_\pi\left[\nabla_\theta \log \pi(a \mid s; \boldsymbol{\theta})(r(s, a) - b(s))\right]$. With eligibility traces using $\lambda \in [0, 1]$ for G_t^λ [115], we have the new returns $\mathcal{G}_t = r_t + \lambda \mathcal{G}_{t+1}$ with $\mathcal{G}_{T+2} = 0$. The expected updates with the new returns $\mathcal{G}$ are $\mathrm{E}_\pi\left[\nabla_\theta \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \mathcal{G}_t\right]$.

(C) Q-learning determines the Q-values via the redistributed reward. However, even for return-equivalent MDPs the Q-values for the optimal policies might differ. Instead of action-value function also the value function can be learned.

RUDDER: Return Decomposition using LSTM. We introduce RUDDER, 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 for return decomposition.

(I) Safe exploration. Exploration strategies should assure that the 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 into 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 [64] and for episodic backward update to efficiently propagate delayed rewards [63]. If RUDDER's safe exploration discovers an episode with unseen delayed reward, it is secured in a lessons replay buffer [66]. Unexpected rewards are 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 return decomposition. LSTM networks are used to predict the return from an input sequence. The LSTM solves the vanishing gradient problem by its constant error carousel [41, 44]. The vanishing gradient problem 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 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 must be one or close to one. LSTM was already used in reinforcement learning [102] for advantage learning [4], for constructing a potential function for reward shaping by representing the return by a sum of LSTM outputs across an episode [111], and learning policies [36, 72, 37]. The LSTM return decomposition is done via contribution analysis. We perform experiments with contribution analysis methods (A) differences of return predictions, (B) integrated gradients (IG), and (C) layer-wise relevance propagation (LRP). However, contribution analysis methods (B) and (C) tend to increase the variance in the redistributed reward. Therefore, we prefer contribution analysis method (A), which computes the redistributed reward by differences in predictions. This contribution analysis method is only valid if the LSTM predicts $\tilde{r}_{T+1}$ at every time step. To ensure a valid LSTM, we train it to predict the return at every time step. However, these continuous return predictions only assist as auxiliary tasks to assess the information change in the LSTM. Even if the continuous predictions are not precise, they still serve to access the information stored in the LSTM. However, if the continuous prediction are perfect, then the LSTM will predict at time step t the expectation $\mathrm{E}_\pi \left[\tilde{R}_{T+1} \mid s_t, a_t \right]$. In this case, the LSTM return decomposition is optimal since the LSTM gives the Q-values at every time step. For more details on LSTM architectures for contribution analysis see Appendix A5.4.

3 Experiments

RUDDER is evaluated on four artificial tasks with delayed rewards, and on 52 Atari games. We compare RUDDER, MC, and TD(λ) on three tasks. Additionally, the methods in the first task are complemented by MCTS. In the fourth task, RUDDER, TD(λ), and reward shaping approaches are compared. The TD representative is the Q-learning method $Q(\lambda)$ — and in the fourth task the SARSA method [97] SARSA(λ) as well — with eligibility traces using $\lambda = 0.9$ (found most favorable [115]). For reward shaping, we include the original method, look-forward advice, and look-back advice with three different potential functions. Tasks (I), (II), and (III) are designed to analyze performances for probabilistic rewards, probabilistic state transitions, and probabilistic policies, respectively. In task (IV), methods have to distinguish between an immediate (distracting) and a higher delayed reward in a deterministic environment. All experiments are based on finite time horizon or absorbing states MDPs with $\gamma = 1$ and reward at episode end. For Atari games using RUDDER the accumulated reward ρ is given at the end. For all artificial tasks, methods are compared for different delays, learn a Q-table (tabular case), and use an ϵ-greedy policy with $\epsilon = 0.2$. Their performance is evaluated by the required learning time to achieve 90% of the maximal expected return. To assess whether RUDDER is significantly faster, a Wilcoxon signed-rank test is performed between the learning time of RUDDER and those of the other methods for all artificial tasks. The performance measure for Atari games is described in paragraph "Atari games".

For the artificial tasks, RUDDER uses an LSTM network without output and forget gate and does not use a lessons buffer nor safe exploration. For tasks (I), (II), (IV), contribution analysis is performed with difference of return predictions, and for task (III) with layer-wise relevance propagation (LRP). The redistributed reward of RUDDER is used to learn a Q-table, which entries are estimated by an exponential moving average of the redistributed reward (RUDDER's Q-value estimation) or alternatively by Q-learning based on the redistributed reward. The RUDDER implementation for Atari games is described in paragraph "Atari games".

(I) Grid World is characterized by probabilistic delayed rewards. It illustrates a situation, where a time bomb explodes at episode end. The agent has to defuse the bomb and then run away as far as possible since defusing fails with a certain probability. Alternatively, the agent can immediately run away, which, however, leads to less reward on average since the bomb always explodes. The Grid World is 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*, *down*, *left*, *right*). Moves are only executed if the agent stays on the grid. At the end of the episode, after $1.5d$ steps, the agent receives a reward of 1000 with probability of 0.5, if it has visited *bomb*. At each time step, the agent receives an immediate reward of $c \cdot t \cdot h$, where c depends on the chosen action, t is the current time step, and h is the Hamming distance to *bomb*. Each move, which reduces the Hamming distance to *bomb*, is penalized by the immediate reward via $c = -0.09$. Each move, which increases the Hamming distance to *bomb*, is rewarded by the immediate reward via $c = 0.1$. The agent is forced to learn the Q-values precisely, since the immediate reward of directly running away hints at a sub-optimal policy. Figure 1(I) shows the learning time (averaged over 100 trials), required by the methods to solve the task vs. the delay of the reward. For all delays, RUDDER is significantly faster than all other methods with p-values $< 10^{-12}$ (Wilcoxon signed-rank test). RUDDER is exponentially faster with increasing delay than $Q(\lambda)$, supporting Theorem A2 in the Appendix. RUDDER also yields a speed-up vs. MC and MCTS, which suggests to be exponential with increasing delay time. In any case, **RUDDER significantly outperforms all other methods.**

(II) The Choice is characterized by probabilistic state transitions, which can be represented as a tree, with the states of the environment as nodes. The agent traverses the tree from the root node (initial state) to one of the leaf nodes (final states). At the root the agent has to choose between the left and the right subtree, where one of the two subtrees has a higher expected reward. Thereafter, it traverses the chosen subtree randomly according to given transition probabilities. Each visited node adds its fixed share to the final reward. The reward is given as accumulated shares at the leaf node and, therefore, is delayed. The task is solved when the agent has learned to always choose the subtree with higher expected reward. Figure 1(II) shows the learning time (averaged of 100 trials), required by the methods to solve the task vs. the delay of the reward. For all delays, RUDDER is significantly faster than all other methods with p-values $< 10^{-8}$ (Wilcoxon signed-rank test). RUDDER is exponentially faster with increasing delay than $Q(\lambda)$, supporting Theorem A2 in the Appendix. RUDDER also yields a speed-up vs. MC, which suggests to be exponential with increasing delay time. In any case, **RUDDER significantly outperforms all other methods.**

(III) Charge-Discharge is characterized by probabilistic policies but deterministic reward and state transitions. The environment consists of 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 $T + 1$, where T corresponds to the maximal delay of the 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. The reward redistribution provided by RUDDER serves to learn a policy by Q-learning. Figure 1(III) shows the different learning times of the methods (averaged over 100 trials) as a function of the delay of the reward. For all delays, RUDDER is significantly faster than all other methods with p-values $< 10^{-11}$ (Wilcoxon signed-rank test). RUDDER is exponentially faster with increasing delay than $Q(\lambda)$, supporting Theorem A2 in the Appendix. RUDDER also yields a speed-up vs. MC, which suggests to be exponential with increasing delay time. In any case, **RUDDER significantly outperforms all other methods.**

(IV) Trace-Back serves to investigate how fast different methods transfer information on delayed rewards back in time. We compare RUDDER, $Q(\lambda)$, SARSA(λ), and reward shaping approaches. MC estimates solve the task very fast but do not transfer back information. Therefore, they are not considered here. The agent can move in a 15×15 grid from its current position in 4 adjacent positions via actions (*up, down, left, right*). Only moves are allowed that keep the agent on the grid. The number of moves per episode is $T = 20$ and the starting position $(7, 7)$. The optimal policy moves the agent *up* in $t = 1$ and *right* in $t = 2$, which gives immediate reward of -50 at $t = 2$, and a delayed reward of 150 at the end $T = 20$. Therefore, the return is 100. For any other policy, the agent receives only an immediate reward of 50 at $t = 2$. For $t \leqslant 2$ the environment is deterministic. Afterwards, state transition probabilities are uniformly distributed independent of the actions. Thus, the return does not depend on actions after $t = 2$. The reward shaping representatives are the original reward shaping, look-ahead advice, and look-back advice. As suggested by the authors, we use SARSA

instead of Q-learning for look-back advice. Reward shaping methods require a potential function, which is based on the reward redistribution in three different ways (see Appendix for more details). At $t = 2$, the immediate reward distracts since it amounts to -50 for the optimal action and to 50 for non-optimal actions. Consequently, learning starts with a Q-value difference around -100 and ends around 50. Therefore, a positive difference of Q-values at time step $t = 2$ indicates an optimal policy based on the Q-table. For all delays, RUDDER is significantly faster than all other methods with p-values $< 10^{-17}$ (Wilcoxon signed-rank test). Figure 1(IV) shows the different learning times of the methods (averaged over 100 trials). **RUDDER is exponentially faster than all other methods and significantly outperforms them.**

In summary, RUDDER significantly outperforms TD(λ) methods, MC, MCTS, and reward shaping methods and is even exponentially faster with respect to the delay time than TD, MCTS, and reward shaping methods. For more information see Appendix A6.2.

Atari Games: RUDDER allows to tackle delayed reward problems in more complex environments, as we show on Atari games of the Arcade Learning Environment (ALE) [8] and OpenAI Gym [12]. The concept of return-equivalent MDPs ensures that the optimal policies of the MDPs are identical, which does not necessarily hold for the optimal Q-values. In other words, if the reward redistribution changes, the optimal policies do not change butwhile Q-values do. Hence, for RUDDER, Q-learning methods are less suited and we combine RUDDER with the policy gradient method PPO. The PPO baseline used for RUDDER differs in two aspects from the original PPO baseline [104]. First, the sign function of the rewards in the original version cannot distinguish small immediate rewards from large delayed rewards. However, this difference is essential for trading of large delayed rewards against small immediate rewards as demonstrated in the artificial tasks (I), (III), and (IV). Instead, we scale the rewards based on the largest return encountered so far. Secondly, the redistributed rewards have to be taken into account for learning the policy. Therefore, we introduce another output of the policy network that predicts the value function of the redistributed reward. For stability reasons, the original value function is kept but combined with the value function of the redistributed reward in the policy gradient. Besides these two aspects, we decrease the hyperparameter space of the original PPO method by removing the entropy coefficient using Proportional Control. We use differences of return predictions of the LSTM network for contribution analysis. In practice, Δ is defined as the pixel-wise difference of two consecutive frames. We augment the input with the current frame, such that static objects are made visible. In order to reduce runtime, we perform LSTM training and reward redistribution on sequence chunks rather than on complete game sequences. A coarse hyperparameter optimization is performed for our PPO baseline, where the same hyperparameters are used for RUDDER for all 52 considered Atari games. For more implementation details see Appendix A7. Source code will be made available upon publication.

Evaluation Methodology. Agents are trained for 25M timesteps, that is 100M game frames using every 4th frame, with no-op starting condition, i.e. a random number of up to 30 no-operation actions at the start of each game. Training episodes are terminated when a life is lost or a maximum of 108K frames is reached. Scoring metrics are (a) *average*, the average reward per completed game throughout training, which favors fast learning [104], (b) *maximum*, the maximum of the averages over 10 consecutive games during training, which favors exploration and the best performing model, and (c) *final*, the average over the last 10 consecutive games at the end of training, which favors consistency in learning. All scores comprise an average over 3 different random seeds for network and ALE initialization. Additionally, we report games where differences of more than 100% between PPO baseline and PPO with RUDDER are observed.

Results. Table 1 shows the number of Atari games won by either the PPO baseline and PPO with RUDDER, after averaging the scores over three random seeds. RUDDER improves the PPO baseline scores in 65% (*average*, *maximum*) and 62% (*last*) of games. Furthermore, RUDDER at least doubles the score of the PPO baseline in 11 (*average*, *maximum*) and 14 (*final*) games. A summary of results on all games can be found in Appendix A6.

Delayed reward games. RUDDER excels in delayed reward games. The most prominent delayed reward games are Bowling, Frostbite, PrivateEye and Venture. In Bowling, the long delay stems from the player steering a ball towards the pins and receiving reward after two rolls at the earliest. In Frostbite, the long delay stems first building an igloo which has to be entered at episode end to receive a large reward. In PrivateEye, the agent has to collect clues, navigate a map, and defeat criminals, with long delays between these actions. In Venture, the long delay is present since discovering

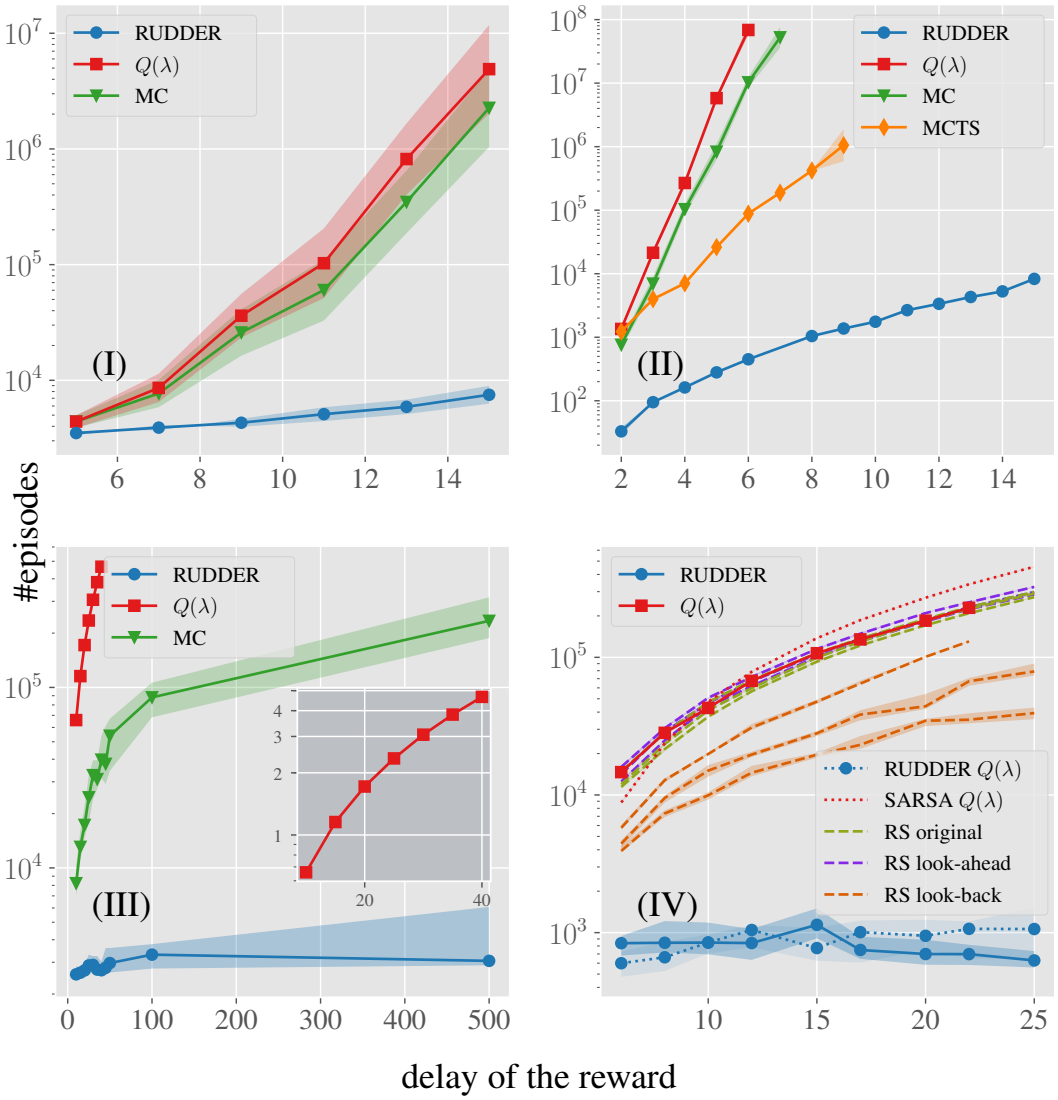


Figure 1: Comparison of RUDDER and other methods on artificial tasks with respect to the learning time versus the delay of the reward. RUDDER is compared in (I) Grid World with $Q(\lambda)$, MC, and MCTS, in (II) Choice and in (III) Charge-Discharge with $Q(\lambda)$ and MC, and in (IV) Trace-Back with $Q(\lambda)$, SARSA(λ), and reward shaping methods. The TD representatives are Q-learning and SARSA with $\lambda = 0.9$. The three reward shaping methods, original reward shaping (RS original), look-ahead advice (RS look-ahead), and look-back advice (RS look-back), each use three different potential functions. The learning time measured in number of episodes is shown as the median over 100 trials. The shadow bands indicate the 40% and 60% quantiles. In (III), the y-axis values of the inlet are scaled bygiven in 10^5. In (IV), the dashed blue line (RUDDER $Q(\lambda)$) represents RUDDER applied on top of $Q(\lambda)$ in contrast to RUDDER's direct Q-estimation. In all tasks, RUDDER significantly outperforms all other methods.

a treasure gives for the first time a reward in the game. Table 2 summarizes the results on these four delayed reward games. **RUDDER considerably outperforms the PPO baseline in prominent delayed reward games.**

Figure 2 displays how RUDDER redistributes rewards to key events in the Atari game Bowling, i.e. to actions that push the ball on a promising trajectory. Thus, RUDDER identifies exactly those actions that are essential to steer the ball in the right direction to hit all pins. Thereby, RUDDER drastically shortens the delay of the reward and effectively guides the learning algorithm towards good policies.

	average		*maximum*		*final*	
	R	B	R	B	R	B
# *won*	34	18	34	18	32	20
double	11	0	11	0	14	0

Table 1: Number of Atari games *won* by either the PPO baseline (B) or PPO with RUDDER (R). RUDDER improves the baseline scores in 65% (*average*, *maximum*) and 62% (*final*) of the games. RUDDER *doubles* the score of the PPO baseline in 11 (*average*, *maximum*) and 14 (*final*) games.

	R	B	delay	delay-event
Bowling	199	77	200	strike pins
Frostbite	2669	331	90	enter igloo
PrivateEye	1429	899	180	collect clues
Venture	910	90	150	find treasure

Table 2: PPO with RUDDER (R) and the PPO baseline (B) are compared at prominent delayed-reward games with respect to their maximum score averaged over three runs with three different random seeds. Delays refer to the number of frames between a delayed reward and its first corresponding action. RUDDER significantly outperforms the PPO baseline in all four delayed reward games.

Furthermore, RUDDER enriches sequences, which are sparse in reward, with a dense reward signal. Video demonstrations will be made available upon publication.

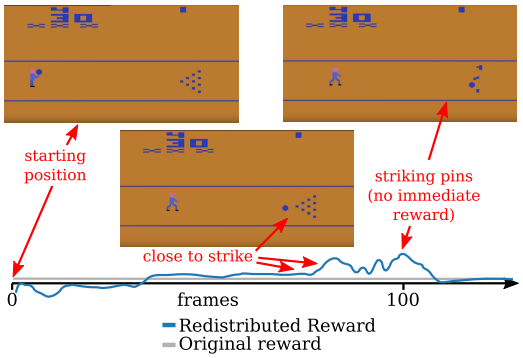


Figure 2: Return decomposition by RUDDER in the Atari game Bowling, where RUDDER redistributes rewards to key events. This game is characterized by long delayed rewards, since reward is only given after multiple rolls at the end of the episode. RUDDER identifies the actions that push the ball in the right direction to hit all pins. The first 100 frames of this 200 frames episode are depicted, where reward is given only at the end.

4 Discussion and Conclusion

Feedforward net vs. LSTM. To use feedforward neural networks instead of an LSTM does not work since they do not allow for contribution analysis. For example, differences of return predictions cannot assess the information gain via the new state-action since two consecutive predictions are not ensured to rely on the same stored information.

Exploration is very critical in RUDDER, since delayed rewards must first be discovered before they can be exploited. The exploration strategy may be too far from the actual policy which hinders learning. Methods like Retrace [78] are able to correct for being too far from the actual policy but introduce an exponential decay.

Human expert episodes are an alternative to exploration and can serve to fill the lessons replay buffer. Return decomposition will reward human key actions. Using human demonstrations in reinforcement learning has led to a huge improvement on some Atari games like Montezuma's Revenge [84, 2].

Conclusion. We have introduced RUDDER, a return decomposition method, which creates a new MDP that keeps the optimal policies but in the optimal case has no delayed rewards. On artificial tasks with delayed rewards, RUDDER significantly outperforms TD(λ), MC, MCTS and reward shaping methods. On Atari, games RUDDER improves a PPO baseline method and yields outstanding results on long delayed rewards games, such as Bowling, Frostbite, PrivateEye and Venture.

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 A8.

Contents

1 Introduction . . . 1
2 Return Decomposition and Reward Redistribution . . . 2
3 Experiments . . . 6
4 Discussion and Conclusion . . . 10
Appendix . . . 13
A1 Related Work . . . 13
A2 Reinforcement Learning and Credit Assignment . . . 14
A2.1 Finite Markov Decision Process . . . 14
A2.2 Bias-Variance of the Q-Value Estimators . . . 17
A2.2.1 Bias-Variance for MC and TD Estimates of the Expected Return . . . 18
A2.2.2 Mean and Variance of an MDP Sample of the Return . . . 20
A2.2.3 TD Corrects Bias Exponentially Slowly With Respect To Reward Delay . . . 22
A2.2.4 MC Affects Variance of Exp. Many Estimates w. Delayed Reward . . . 24
A2.3 Reward Redistribution . . . 30
A2.3.1 Return-Equivalent MDPs . . . 30
A2.3.2 State Enriched MDPs . . . 31
A2.3.3 Transforming an Immediate Reward to a Delayed Reward MDP . . . 32
A2.3.4 Reward Redistribution by Return Decomposition . . . 34
A3 Markov Decision Processes with Undiscounted Rewards . . . 44
A3.1 Properties of the Bellman Operator in MDPs with Undiscounted Rewards . . . 44
A3.1.1 Monotonically Increasing and Continuous . . . 44
A3.1.2 Contraction for Undiscounted Finite Horizon . . . 45
A3.1.3 Contraction for Undiscounted Infinite Horizon With Absorbing States . . . 46
A3.1.4 Fixed Point of Contraction is Continuous wrt Parameters . . . 46
A3.1.5 t-fold Composition of the Operator . . . 47
A3.2 Q-value Transformations: Shaping Reward, Baseline, and Normalization . . . 48
A3.3 Alternative Definition of State Enrichment . . . 49
A3.4 Variance of the Weighted Sum of a Multinomial Distribution . . . 50
A4 Backward Analysis . . . 51
A4.1 Difference of Consecutive Predictions for Sequences . . . 51
A4.2 Input Zeroing . . . 54
A4.3 Integrated Gradients . . . 55
A4.4 Layer-Wise Relevance Propagation . . . 55
A4.4.1 New Variants of LRP . . . 56
A4.4.2 LRP for Products . . . 56
A5 Long Short-Term Memory (LSTM) . . . 57
A5.1 LSTM Introduction . . . 57
A5.2 LSTM in a Nutshell . . . 58
A5.3 Long-Term Dependencies vs. Uniform Credit Assignment . . . 59
A5.4 Special LSTM Architectures for Backward Analysis . . . 59
A5.4.1 LSTM for Integrated Gradients . . . 59
A5.4.2 LSTM for LRP . . . 60
A5.4.3 LSTM for Nondecreasing Memory Cells . . . 63
A5.4.4 LSTM without Gates . . . 63
A6 Artificial Tasks . . . 66
A6.1 RUDDER compared with Reward Shaping Methods . . . 66
A6.2 RUDDER compared with TD, MC, MCTS . . . 70
A6.2.1 Grid World . . . 70
A6.2.2 The Choice . . . 71
A6.2.3 Charge-Discharge . . . 74
A7 RUDDER Implementation for Atari Games . . . 74
A7.1 Architecture . . . 75
A7.2 Lessons Replay Buffer . . . 75
A7.3 Game Processing and Update/Target Design . . . 76
A7.4 Exploration . . . 78
A7.5 Results . . . 78
A8 References . . . 82

Appendix

A1 Related Work

Delayed Reward. 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) [73, 74], which combine Q-learning with convolutional neural networks for visual reinforcement learning [59]. The success of DQNs is attributed to *experience replay* [66], which stores observed state-reward transitions and then samples from them. Prioritized experience replay [100, 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) [121, 122] which helps exploration as the overestimation bias is reduced. Noisy DQNs [20] explore by a stochastic layer in the policy network (see [40, 101]). Distributional Q-learning [7] profits from noise since means that have high variance are more likely selected. The dueling network architecture [125, 126] separately estimates state values and action advantages, which helps exploration in unknown states. Policy gradient approaches [131] explore via parallel policies, too. A2C has been improved by IMPALA through parallel actors and correction for policy-lags between actors and learners [18]. A3C with asynchronous gradient descent [72] 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 [104].
Recent approaches aim to solve learning problems caused by delayed rewards. Function approximations of value functions or critics [74, 72] 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 [114, 115] improved both DQNs and policy gradients [39, 72]. AlphaGo and AlphaZero learned to play Go and Chess better than human professionals using Monte Carlo Tree Search (MCTS) [105, 106]. 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 [35]. These forward view approaches are not feasible in probabilistic environments with a high branching factor of state transition.

Backward View. We propose learning from a backward view by "backward analysis", which either learns a separate model or analyzes a forward model. Examples of learning a separate model are to trace back from known goal states [17] or from high reward states [29]. However, learning a backward model is very challenging. When analyzing a forward model that predicts the return then either sensitivity analysis or contribution analysis may be utilized. The best known backward analysis approach is sensitivity analysis (computing the gradient) like "backpropagation through a model" [79, 93, 94, 5]. Sensitivity analysis has several drawbacks: local minima, instabilities, exploding or vanishing gradients, and proper exploration [40, 101]. The major drawback is that the relevance of actions is missed since sensitivity analysis does not consider their contribution to the output but only their effect on the output when slightly perturbing them.
We use contribution analysis since sensitivity analysis has serious drawbacks. Contribution analysis determines how much a state-action pair contributes to the final prediction. To focus on state-actions which are most relevant for learning is known from prioritized sweeping for model-based reinforcement learning [77]. Contribution analysis can be done by computing differences of return predictions when adding another input, by zeroing out an input and then compute the change in the prediction, by contribution-propagation [61], by a contribution approach [85], by excitation backprop [133], by layer-wise relevance propagation (LRP) [3], by Taylor decomposition [3, 75], or by integrated gradients (IG) [112].

LSTM. LSTM was already used in reinforcement learning [102] for advantage learning [4], for constructing a potential function for reward shaping by representing the return by a sum of LSTM outputs across an episode [111], and learning policies [36, 72, 37].

Reward Shaping, Look-Ahead Advice, Look-Back Advice. Redistributing the reward is fundamentally different from reward shaping [80, 129], look-ahead advice and look-back advice [130].

However, these methods can be viewed as a special case of reward redistribution that result in an MDP that is return-equivalent to the original MDP as is shown in Appendix A2.3.1. In contrast to these methods, reward redistribution is not limited to potential functions, where the additional reward is the potential difference, therefore it is a more general concept than shaping reward or look-ahead/look-back advice. The major difference of reward redistribution to reward shaping, look-ahead advice, and look-back advice is that the last three keep the original rewards. Both look-ahead advice and look-back advice have not been designed for replacing for the original rewards. Since the original reward is kept, the reward redistribution does not correspond to an optimal return decomposition according to Appendix A2.3.4. The original rewards may have long delays that cause an exponential slow-down of learning. The added reward improves sampling but a delayed original reward must still be transferred to the Q-values of early states that caused the reward. The concept of return-equivalence of MDPs resulting from reward redistributions allows to eliminate the original reward completely. Reward shaping can replace the original reward. However, it only depends on states but not on actions, and therefore, it cannot identify relevant actions without the original reward.

A2 Reinforcement Learning and Credit Assignment

A2.1 Finite Markov Decision Process

We consider a finite Markov decision process (MDP) $\mathcal{P}$, which is a 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 $a, a' \in \mathcal{A}$ and has a discrete probability distribution.
- $\mathcal{R}$ is a finite set of rewards; R_t is the random variable for rewards at time t with values $R_{t+1} = r \in \mathcal{R}$ 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 letters 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 states, 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 fewer 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 starting 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_*:

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

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 reward R_{T+1} at episode end 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 the 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

$$\begin{aligned} v_0^\pi \;&=\; \mathrm{E}_\pi\left[G_0\right] \;=\; \mathrm{E}_\pi\left[\sum_{t=0}^{T} R_{t+1}\right]\,, \\ \pi^* \;&=\; \operatorname*{argmax}_\pi\; v_0^\pi\,. \end{aligned} \tag{A21}$$

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 at $t = 0$. 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 returns $(G_t \mid S_t = s, A_t = a)$ in the data. When using Monte Carlo for learning a policy we use an exponentially weighted arithmetic mean, 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* [115].

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:

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

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 $(\hat{q}^\pi)^{\mathrm{new}}$. The new estimate $(\hat{q}^\pi)^{\mathrm{new}}$ is closer to the fixed point q^π of the Bellman operator, since the Bellman operator is a contraction (see Appendix A3.1.3 and Appendix A3.1.2).
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** [97] 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** [55] 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 $r(s,a)$ [95]. ***Q*-learning** [127] is an off-policy TD algorithm which is proved to converge [128, 14]. The proofs were later generalized [53, 120]. 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 action-value function q, which is learned by Q-learning, approximates q_* independently of the policy that is followed. More precisely, with Q-learning q converges 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.

A2.2 Bias-Variance of the Q-Value Estimators

Bias-variance investigations have been done for Q-learning. Grünewälder & Obermayer [34] investigated the bias of temporal difference learning (TD), Monte Carlo estimators (MC), and least-squares temporal difference learning (LSTD). Mannor et al. [69] and O'Donoghue et al. [81] 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 is a 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 $\mathrm{E}_D[.]$ 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 of 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

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

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

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

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

A2.2.1 Bias-Variance for MC and TD Estimates of the Expected Return

Monte Carlo (MC) computes the arithmetic mean $\hat{q}^\pi(s,a)$ of G_t for $(s_t = s, a_t = a)$ over the episodes given by the data.

For **temporal difference (TD)** methods, like SARSA, with learning rate α the updated estimate of $q^\pi(s_t, a_t)$ is:

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

Similar updates are used for expected SARSA and Q-learning, where only a_{t+1} is chosen differently. Therefore, for the estimation of $\hat{q}^\pi(s_t,a_t)$, SARSA and Q-learning perform an exponentially weighted arithmetic mean of $(R_{t+1}+\gamma\hat{q}^\pi(s_{t+1},a_{t+1}))$. If for the updates $\hat{q}^\pi(s_{t+1},a_{t+1})$ is fixed on some data, then SARSA and Q-learning perform an exponentially weighted arithmetic mean 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 exponentially weighted arithmetic mean 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 and Q-learning, respectively, 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 over samples with Monte Carlo or an exponentially weighted arithmetic mean over samples with TD methods. 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 exponentially weighted arithmetic mean.

We assume n samples for a state-action pair (s,a). However, the expected number of samples depends on the probabilistic 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 arithmetic mean, its bias and and its variance are:

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

The estimation variance of the arithmetic mean is determined by σ^2, the variance of the distribution the samples are drawn from.

Exponentially weighted arithmetic mean. 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 a weighted arithmetic mean with exponentially decreasing weights, since the coefficients sum up to one:

$$\begin{aligned}&\alpha\sum_{i=1}^{n}(1-\alpha)^{n-i} + (1-\alpha)^n = \alpha\,\frac{1-(1-\alpha)^n}{1-(1-\alpha)} + (1-\alpha)^n \\ &= 1 - (1-\alpha)^n + (1-\alpha)^n = 1\,.\end{aligned} \tag{A40}$$

The estimator $\hat{\mu}_n$ is biased, since:

$$\mathbf{bias}(\hat{\mu}_n) = \mathrm{E}\left[\hat{\mu}_n\right] - \mu = \mathrm{E}\left[\alpha\sum_{i=1}^{n}(1-\alpha)^{n-i}\,X_i\right] + (1-\alpha)^n\,\mu_0 - \mu \tag{A41}$$

$$\begin{aligned}&= \alpha\sum_{i=1}^{n}(1-\alpha)^{n-i}\mathrm{E}\left[X_i\right] + (1-\alpha)^n\,\mu_0 - \mu \\ &= \mu\,\alpha\sum_{i=0}^{n-1}(1-\alpha)^i + (1-\alpha)^n\,\mu_0 - \mu \\ &= \mu\,(1-(1-\alpha)^n) + (1-\alpha)^n\,\mu_0 - \mu = (1-\alpha)^n\,(\mu_0-\mu)\,.\end{aligned}$$

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

$$
\begin{aligned}
&\mathbf{var}(\hat{\mu}_n) \;=\; \mathrm{E}\left[\hat{\mu}_n^2\right] \;-\; \mathrm{E}^2\left[\hat{\mu}_n\right] \qquad \text{(A42)}\\
&=\; \mathrm{E}\left[\alpha^2 \sum_{i=1}^{n}\sum_{j=1}^{n}(1\;-\;\alpha)^{n-i}\,X_i\,(1\;-\;\alpha)^{n-j}\,X_j\right]\\
&+\; \mathrm{E}\left[2\,(1\;-\;\alpha)^n\,\mu_0\,\alpha\,\sum_{i=1}^{n}(1\;-\;\alpha)^{n-i}\,X_i\right] \;+\; (1\;-\;\alpha)^{2n}\,\mu_0^2\\
&-\; \left((1\;-\;\alpha)^n\,(\mu_0\;-\;\mu)\;+\;\mu\right)^2\\
&=\; \alpha^2\,\mathrm{E}\left[\sum_{i=1}^{n}(1\;-\;\alpha)^{2(n-i)}\,X_i^2 \;+\; \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(1\;-\;\alpha)^{n-i}\,X_i\,(1\;-\;\alpha)^{n-j}\,X_j\right]\\
&+\; 2\,(1\;-\;\alpha)^n\,\mu_0\,\mu\,\alpha\,\sum_{i=1}^{n}(1\;-\;\alpha)^{n-i} \;+\; (1\;-\;\alpha)^{2n}\,\mu_0^2\\
&-\; \left((1\;-\;\alpha)^n\,\mu_0 \;+\; (1\;-\;(1\;-\;\alpha)^n)\,\mu\right)^2\\
&=\; \alpha^2\,\left(\sum_{i=1}^{n}(1\;-\;\alpha)^{2(n-i)}\,\left(\sigma^2\;+\;\mu^2\right) + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}(1\;-\;\alpha)^{n-i}\,(1\;-\;\alpha)^{n-j}\,\mu^2\right)\\
&+\; 2\,(1\;-\;\alpha)^n\,\mu_0\,\mu\,(1\;-\;(1\;-\;\alpha)^n) \;+\; (1\;-\;\alpha)^{2n}\,\mu_0^2\\
&-\; (1\;-\;\alpha)^{2n}\,\mu_0^2 \;-\; 2\,(1\;-\;\alpha)^n\,\mu_0\,(1\;-\;(1\;-\;\alpha)^n)\,\mu \;-\; (1\;-\;(1\;-\;\alpha)^n)^2\,\mu^2\\
&=\; \sigma^2\,\alpha^2\,\sum_{i=0}^{n-1}\left((1\;-\;\alpha)^2\right)^i \;+\; \mu^2\,\alpha^2\,\left(\sum_{i=0}^{n-1}(1\;-\;\alpha)^i\right)^2 \;-\; (1\;-\;(1\;-\;\alpha)^n)^2\,\mu^2\\
&=\; \sigma^2\,\alpha^2\,\frac{1-(1\;-\;\alpha)^{2n}}{1-(1\;-\;\alpha)^2} \;=\; \sigma^2\,\frac{\alpha\,(1\;-\;(1\;-\;\alpha)^{2n})}{2-\alpha}\;.
\end{aligned}
$$

Also the estimation variance of the exponentially weighted arithmetic mean is proportional to σ^2, which is 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 [11, 118] states that for a random variable X with expected value μ and variance $\tilde{\sigma}^2$ and for any real number $\epsilon > 0$:

$$\Pr\left[|X-\mu| \;\geqslant\; \epsilon\,\tilde{\sigma}\right] \;\leqslant\; \frac{1}{\epsilon^2} \qquad \text{(A43)}$$

or, equivalently,

$$\Pr\left[|X-\mu| \;\geqslant\; \epsilon\right] \;\leqslant\; \frac{\tilde{\sigma}^2}{\epsilon^2}\;. \qquad \text{(A44)}$$

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

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

Following Grünewälder and Obermayer [34], 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 [67]):

$$\Pr\left[\left|\frac{1}{n}\sum_{i=1}^{n}X_i \;-\; \mu\right| \;\geqslant\; \epsilon\right] \;\leqslant\; 2\,\exp\left(-\,\frac{\epsilon^2\,n}{2\,\sigma^2 \;+\; \frac{2\,M\,\epsilon}{3}}\right)\;, \qquad \text{(A46)}$$

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

A2.2.2 Mean and Variance of an MDP Sample of the Return

Since the variance of the estimators of the expectations (arithmetic mean and exponentially weighted arithmetic mean) is governed by the variance of the samples, we compute mean and variance of the return estimate $q^\pi(s,a)$. We follow [108, 116, 117] 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:

$$G_0 \ = \ \sum_{k=0}^{T} \gamma^k \, R_{k+1} \ . \tag{A47}$$

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

$$G_t \ = \ \sum_{k=0}^{T-t} \gamma^k \, R_{t+k+1} \tag{A48}$$

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

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

The second moment of the return is:

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

The variance of the return is:

$$V^\pi(s,a) \ = \ \mathrm{Var}_\pi \left[G_t \mid s, a \right] \ = \ M^\pi(s,a) \ - \ \left(q^\pi(s,a) \right)^2 . \tag{A51}$$

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 analogously $\mathrm{Var}_{s',a'}$ 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*

$$q^\pi(s,a) = \sum_{s',r} p(s',r \mid s,a) \left(r + \gamma \sum_{a'} \pi(a' \mid s') q^\pi(s',a') \right) = r(s,a) + \gamma \mathrm{E}_{s',a'} \left[q^\pi(s',a') \mid s,a \right],$$
$$V^\pi(s,a) = \mathrm{Var}_r \left[r \mid s,a \right] + \gamma^2 \left(\mathrm{E}_{s',a'} \left[V^\pi(s',a') \mid s,a \right] + \mathrm{Var}_{s',a'} \left[q^\pi(s',a') \mid s,a \right] \right). \tag{A52}$$

Proof. The Bellman equation for Q-values is

$$\begin{aligned} q^\pi(s,a) \ &= \ \sum_{s',r} p(s',r \mid s,a) \ \left(r \ + \ \gamma \ \sum_{a'} \pi(a' \mid s') \ q^\pi(s',a') \right) \\ &= \ r(s,a) \ + \ \gamma \, \mathrm{E}_{s',a'} \left[q^\pi(s',a') \mid s,a \right] \ . \end{aligned} \tag{A53}$$

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

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

$$r^2(s,a) \ = \ \sum_r r^2 \ p(r \mid s,a) \ . \tag{A55}$$

For the second moment, we obtain [116]:

$$
\begin{aligned}
M^\pi(s,a) \;&=\; \mathrm{E}_\pi\left[G_t^2 \mid s,a\right] \qquad \text{(A56)}\\
&=\; \mathrm{E}_\pi\left[\left(\sum_{k=0}^{T-t}\gamma^k R_{t+k+1}\right)^2 \mid s,a\right]\\
&=\; \mathrm{E}_\pi\left[\left(R_{t+1} \;+\; \sum_{k=1}^{T-t}\gamma^k\, R_{t+k+1}\right)^2 \mid s,a\right]\\
&=\; r^2(s,a) \;+\; 2\, r(s,a)\, \mathrm{E}_\pi\left[\sum_{k=1}^{T-t}\gamma^k\, R_{t+k+1} \mid s,a\right]\\
&+\; \mathrm{E}_\pi\left[\left(\sum_{k=1}^{T-t}\gamma^k\, R_{t+k+1}\right)^2 \mid s,a\right]\\
&=\; r^2(s,a) \;+\; 2\gamma\, r(s,a) \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q^\pi(s',a')\\
&+\; \gamma^2 \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, M^\pi(s',a')\\
&=\; r^2(s,a) \;+\; 2\gamma\, r(s,a)\, \mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] \;+\; \gamma^2\, \mathrm{E}_{s',a'}\left[M^\pi(s',a') \mid s,a\right]\,.
\end{aligned}
$$

For the variance, we obtain:

$$
\begin{aligned}
V^\pi(s,a) \;&=\; M^\pi(s,a) \;-\; \left(q^\pi(s,a)\right)^2 \qquad \text{(A57)}\\
&=\; r^2(s,a) \;-\; \left(r(s,a)\right)^2 \;+\; \gamma^2\, \mathrm{E}_{s',a'}\left[M^\pi(s',a') \mid s,a\right] \;-\; \gamma^2\, \mathrm{E}^2_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\\
&=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \gamma^2\, \left(\mathrm{E}_{s',a'}\left[M^\pi(s',a') \;-\; \left(q^\pi(s',a')\right)^2 \mid s,a\right]\right.\\
&\left.-\, \mathrm{E}^2_{s',a'}\left[q^\pi(s',a') \mid s,a\right] \;+\; \mathrm{E}_{s',a'}\left[\left(q^\pi(s',a')\right)^2 \mid s,a\right]\right)\\
&=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \gamma^2\, \left(\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right)\,.
\end{aligned}
$$

□

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 [108] and as Proposition 3.1 (c) in Tamar et al. 2012 [116].
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 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. (A52) of sampled returns 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 the following 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

$$\mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] = \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s') \left(q^\pi(s',a')\right)^2 \tag{A58}$$
$$- \left(\sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q^\pi(s',a')\right)^2 .$$

This result is Equation (6) in Sobel 1982 [108]. 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. (A52). TD, 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. (A52) 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 rewards, 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 one step back via the estimates $\hat{q}$ for each sample. 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 and delayed rewards, 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 Q-values according to Eq. (A52).
Sample Distribution Used by Temporal Difference and Monte Carlo. Monte Carlo (MC) sampling uses the true mean and true variance, where the true mean is

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

and the true variance is

$$V^\pi(s,a) = \mathrm{Var}_r\left[r \mid s,a\right] + \gamma^2 \left(\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] + \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right) . \tag{A60}$$

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

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

This mean is biased by

$$\gamma \left(\mathrm{E}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right] - \mathrm{E}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right) . \tag{A62}$$

The variance used by temporal difference is

$$V^\pi(s,a) = \mathrm{Var}_r\left[r \mid s,a\right] + \gamma^2\, \mathrm{Var}_{s',a'}\left[\hat{q}^\pi(s',a') \mid s,a\right] , \tag{A63}$$

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

A2.2.3 TD Corrects Bias Exponentially Slowly With Respect To Reward Delay

Temporal Difference. We show that TD updates for delayed rewards are exponentially small, fading exponentially with the number of delay steps. 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 [6]. If for a fixed learning rate the agent always travels along the same sequence of states, then TD is superquadratic [6]. We, however, consider the general case where the agent travels along random sequences due to a random environment or due to exploration. For a 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 the Q-values of episode i as q^i and assume that the state action pairs

(s_t, a_t) are the most visited ones. We consider the update of $q^i(s_t, a_t)$ of a state-action pair (s_t, a_t) that is visited at time t in the ith episode:

$$q^{i+1}(s_t, a_t) \;=\; q^i(s_t, a_t) \;+\; \alpha\, \delta_t \;, \tag{A64}$$

$$\delta_t \;=\; r_{t+1} \;+\; \max_{a'} q^i(s_{t+1}, a') \;-\; q^i(s_t, a_t) \;\; (Q\text{-learning}) \tag{A65}$$

$$\delta_t \;=\; r_{t+1} \;+\; \sum_{a'} \pi(a' \mid s_{t+1})\, q^i(s_{t+1}, a') \;-\; q^i(s_t, a_t) \;\; (\text{expected SARSA}) \;. \tag{A66}$$

Temporal Difference with Eligibility Traces. Eligibility traces have been introduced to propagate back reward information of an episode and are now standard for TD(λ) [107]. However, the eligibility traces are exponentially decaying when propagated back. The accumulated trace is defined as [107]:

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

while the replacing trace is defined as [107]:

$$e_{t+1}(s, a) \;=\; \begin{cases} \gamma\, \lambda\, e_t(s, a) & \text{for } s \neq s_t \text{ or } a \neq a_t \;, \\ 1 & \text{for } s = s_t \text{ and } a = a_t \;. \end{cases} \tag{A68}$$

With eligibility traces using $\lambda \in [0, 1]$, the λ-return G_t^λ is [115]

$$G_t^\lambda \;=\; (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(n)} \;, \tag{A69}$$

$$G_t^{(n)} \;=\; r_{t+1} \;+\; \gamma\, r_{t+2} \;+\; \ldots \;+\; \gamma^{n-1} r_{t+n} \;+\; \gamma^{n-1}\, V(s_{t+n}) \;. \tag{A70}$$

We obtain

$$\begin{aligned} G_t^\lambda \;&=\; (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(n)} \\ &=\; (1 - \lambda) \left(r_{t+1} \;+\; \gamma\, V(s_{t+1}) \;+\; \sum_{n=2}^{\infty} \lambda^{n-1}\, G_t^{(n)} \right) \\ &=\; (1 - \lambda) \left(r_{t+1} \;+\; \gamma\, V(s_{t+1}) \;+\; \sum_{n=1}^{\infty} \lambda^{n}\, G_t^{(n+1)} \right) \\ &=\; (1 - \lambda) \left(r_{t+1} \;+\; \gamma\, V(s_{t+1}) \;+\; \lambda\, \gamma \sum_{n=1}^{\infty} \lambda^{n-1}\, G_{t+1}^{(n)} \;+\; \sum_{n=1}^{\infty} \lambda^{n}\, r_{t+1} \right) \\ &=\; (1 - \lambda) \sum_{n=0}^{\infty} \lambda^{n}\, r_{t+1} \;+\; (1 - \lambda) \gamma\, V(s_{t+1}) \;+\; \lambda\, \gamma\, G_{t+1}^\lambda \\ &=\; r_{t+1} \;+\; (1 - \lambda) \gamma\, V(s_{t+1}) \;+\; \lambda\, \gamma\, G_{t+1}^\lambda \;. \end{aligned} \tag{A71}$$

We use the naive $Q(\lambda)$, where eligibility traces are not set to zero. In contrast, Watkins' $Q(\lambda)$ [127] 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 $Q(\lambda)$ [83] which also does not zero out eligibility traces.

The next Theorem A2 states that the decay of TD is exponential for Q-value updates in an MDP with delayed reward, even for eligibility traces. Thus, for delayed rewards TD requires 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 $q^i(s_{T-i}, a_{T-i}) = \alpha^{i+1} r_{T+1}^1$, where r_{T+1}^1 is the reward of episode 1. Eligibility traces with $\lambda \in [0, 1)$ lead to an exponential decay of $(\gamma\lambda)^k$ when the reward is propagated k steps back.*

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 rewards we have $r_t = 0$ for $t \leqslant T$. The Q-value $q(s_{T-i}, a_{T-i})$ at time $T - i$ can receive an update for the first time at episode i. Since all Q-values have been initialized with zero, the update is

$$q^i(s_{T-i}, a_{T-i}) \;=\; \alpha^{i+1}\, r_{T+1}^1 \;, \tag{A72}$$

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

$$q_{t+1}(s,a) \;=\; q_t(s,a) \;+\; \alpha\, \delta_t\, e_t(s,a)\,, \tag{A73}$$

$$\delta_t \;=\; r_{t+1} \;+\; \max_{a'} q_t(s_{t+1},a') \;-\; q_t(s_t,a_t)\,. \tag{A74}$$

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

$$e_{t+k}(s_t,a_t) \;=\; \big(\gamma\,\lambda\big)^k\,. \tag{A75}$$

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

$$q_{t+k+1}(s,a) \;=\; q_{t+k}(s,a) \;+\; \alpha\,\delta_{t+k}\,e_{t+k}(s,a) \;=\; q_{t+k}(s,a) \;+\; \alpha\,\big(\gamma\,\lambda\big)^k\,\delta_{t+k}\,. \tag{A76}$$

□

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

A2.2.4 MC Affects the Variance of Exponentially Many Estimates with Delayed Reward

The variance for Monte Carlo is

$$V^\pi(s,a) \;=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \gamma^2\,\left(\mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\right)\,. \tag{A77}$$

This is a Bellman equation of the variance. For undiscounted reward $\gamma = 1$, we obtain

$$V^\pi(s,a) \;=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\,. \tag{A78}$$

If we define the "on-site" variance ω as

$$\omega(s,a) \;=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right]\,, \tag{A79}$$

we get

$$V^\pi(s,a) \;=\; \omega(s,a) \;+\; \mathrm{E}_{s',a'}\left[V^\pi(s',a') \mid s,a\right]\,. \tag{A80}$$

This is the solution of the general formulation of the Bellman operator equation in Eq. (A229). The Bellman operator is defined component-wise for any variance V as

$$\mathrm{T}^\pi\left[V\right](s,a) \;=\; \omega(s,a) \;+\; \mathrm{E}_{s',a'}\left[V(s',a') \mid s,a\right]\,. \tag{A81}$$

According to the results in Appendix A3.1, for proper policies π a unique fixed point V^π exists:

$$V^\pi \;=\; \mathrm{T}^\pi\left[V^\pi\right] \tag{A82}$$

$$V^\pi \;=\; \lim_{k\to\infty} \left(\mathrm{T}^\pi\right)^k V\,, \tag{A83}$$

where V is any initial variance. In Appendix A3.1 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. We obtain component-wise for $\omega > \tilde{\omega}$:

$$\begin{aligned} &\mathrm{T}^\pi_\omega\left[q\right](s,a) \;-\; \mathrm{T}^\pi_{\tilde{\omega}}\left[q\right](s,a) \\ &=\; \left(\omega(s,a) \;+\; \mathrm{E}_{s',a'}\left[q(s',a')\right]\right) \;-\; \left(\tilde{\omega}(s,a) \;+\; \mathrm{E}_{s',a'}\left[q(s',a')\right]\right) \\ &=\; \omega(s,a) \;-\; \tilde{\omega}(s,a) \;\geqslant\; 0\,. \end{aligned} \tag{A84}$$

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

$$V^\pi(s,a) \;\geqslant\; \widetilde{V}^\pi(s,a)\,. \tag{A85}$$

Therefore if

$$\begin{aligned} \omega(s,a) \;&=\; \mathrm{Var}_r\left[r \mid s,a\right] \;+\; \mathrm{Var}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] \;\geqslant \\ \tilde{\omega}(s,a) \;&=\; \widetilde{\mathrm{Var}}_r\left[r \mid s,a\right] \;+\; \widetilde{\mathrm{Var}}_{s',a'}\left[q^\pi(s',a') \mid s,a\right] \end{aligned} \tag{A86}$$

then

$$V^{\pi}(s,a) \;\geqslant\; \widetilde{V}^{\pi}(s,a)\;. \tag{A87}$$

In Stephen Patek's PhD thesis, [82] 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 the inequality above it follows that

$$\begin{aligned} V^{\pi}(s,a) \;-\; \widetilde{V}^{\pi}(s,a) \;&=\; \left(\mathrm{T}^{\pi}_{\omega} V^{\pi}\right)(s,a) \;-\; \left(\mathrm{T}^{\pi}_{\tilde{\omega}} \widetilde{V}^{\pi}\right)(s,a) \\ &=\; \omega(s,a) \;-\; \tilde{\omega}(s,a) \;+\; \mathrm{E}_{s',a'}\left[V^{\pi}(s',a') \;-\; \widetilde{V}^{\pi}(s',a') \mid s,a\right] \\ &\geqslant\; \omega(s,a) \;-\; \tilde{\omega}(s,a)\;. \end{aligned} \tag{A88}$$

Time-Agnostic States. We defined a Bellman operator as

$$\begin{aligned} \mathrm{T}^{\pi}\left[\boldsymbol{V}^{\pi}\right](s,a) \;&=\; \omega(s,a) \;+\; \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, \boldsymbol{V}^{\pi}(s',a') \\ &=\; \omega(s,a) \;+\; \left(\boldsymbol{V}^{\pi}\right)^{T} \boldsymbol{p}(s,a)\;, \end{aligned} \tag{A89}$$

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 reads

$$\mathrm{T}^{\pi}\left[\boldsymbol{V}^{\pi}\right] \;=\; \boldsymbol{\omega} \;+\; \boldsymbol{P}\, \boldsymbol{V}^{\pi}\;, \tag{A90}$$

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 eigenvalue 1 has purely positive real components. Equation 7 of Bertsekas and Tsitsiklis, 1991, [9] states that

$$\left(\mathrm{T}^{\pi}\right)^{t}\left[\boldsymbol{V}^{\pi}\right] \;=\; \sum_{k=0}^{t-1} \boldsymbol{P}^{k}\, \boldsymbol{\omega} \;+\; \boldsymbol{P}^{t}\, \boldsymbol{V}^{\pi}\;. \tag{A91}$$

Applying the operator T^{π} recursively t times can be written as [9]:

$$\left(\mathrm{T}^{\pi}\right)^{t}\left[\boldsymbol{V}^{\pi}\right] \;=\; \sum_{k=0}^{t-1} \boldsymbol{P}^{k}\, \boldsymbol{\omega} \;+\; \boldsymbol{P}^{t}\, \boldsymbol{V}^{\pi}\;. \tag{A92}$$

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

$$\left(\mathrm{T}^{\pi}\right)^{t}\left[\mathbf{0}\right] \;=\; \sum_{k=0}^{t-1} \boldsymbol{P}^{k}\, \boldsymbol{\omega}\;. \tag{A93}$$

For finite horizon MDPs, the values $\boldsymbol{V}^{\pi} = \mathbf{0}$ are correct for time step $T+1$ since no reward for $t > T+1$ exists. Therefore, the "backward induction algorithm" [86, 87] gives the correct solution:

$$\boldsymbol{V}^{\pi} \;=\; \left(\mathrm{T}^{\pi}\right)^{T}\left[\mathbf{0}\right] \;=\; \sum_{k=0}^{T-1} \boldsymbol{P}^{k}\, \boldsymbol{\omega}\;. \tag{A94}$$

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 $\mathcal{R}(\boldsymbol{P}^{k})$ of the stochastic matrix $\boldsymbol{P}^{k}$ is: $\mathcal{R}(\boldsymbol{P}^{k}) = 1$. Furthermore, the largest eigenvalue is 1 and all eigenvalues have absolute values smaller or 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, where 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:

$$\sum_k C_{ik} \;=\; \sum_k \sum_j A_{ij}\, B_{jk} \;=\; \sum_j A_{ij} \sum_k B_{jk} \;=\; \sum_j A_{ij} \;=\; 1\,. \tag{A95}$$

□

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:*

$$\|\boldsymbol{A}\|_\infty \;=\; \max_{\|\boldsymbol{x}\|_\infty = 1} \|\boldsymbol{A}\,\boldsymbol{x}\|_\infty \;=\; \max_i \sum_j |A_{ij}|\;. \tag{A96}$$

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

$$\|\boldsymbol{A}\|_1 \;=\; \max_{\|\boldsymbol{x}\|_1 = 1} \|\boldsymbol{A}\,\boldsymbol{x}\|_1 \;=\; \max_j \sum_i |A_{ij}|\;. \tag{A97}$$

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 $\mathcal{R}$ is $\mathcal{R}(\boldsymbol{A}) = 1$. We extend the theorem to 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*

$$\|\boldsymbol{A}\|_\infty \;=\; 1\,, \quad \left\|\boldsymbol{A}^T\right\|_1 \;=\; 1\,, \text{ and for } n = m \quad \mathcal{R}(\boldsymbol{A}) \;=\; 1\,. \tag{A98}$$

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 $\|.\|$, for the spectral norm $\mathcal{R}(\boldsymbol{A})$ of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ we obtain:

$$\mathcal{R}(\boldsymbol{A}) \;=\; \lim_{k \to \infty} \left\|\boldsymbol{A}^k\right\|^{1/k}\;. \tag{A99}$$

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 $\mathcal{R}(\boldsymbol{A})$ of a row-stochastic matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is

$$\mathcal{R}(\boldsymbol{A}) \;=\; 1\,. \tag{A100}$$

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 $2\sqrt{m}$ [71]. Asymptotically we have for the maximal singular value $s_{\max}(\boldsymbol{A}) \propto \sqrt{m} + \sqrt{n}$ [96]. A bound on the largest singular value is given by [109]:

$$s_{\max}^2(\boldsymbol{A}) \;\leqslant\; (\sqrt{m} \,+\, \sqrt{n})^2 \,+\, O(\sqrt{n}\,\log(n)) \text{ a.s.} \tag{A101}$$

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 tstates are time aware, transitions only occur 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 N_t. Let c_t be the average connectivity of a state in s_t to states in s_{t-1} and $\bar{c} = \left(\prod_{t=1}^{k} c_t\right)^{1/k}$ 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 $\boldsymbol{\omega}_k$ only has one component larger than zero, that is, only one state at time $t = k$ is affected: $n_k = 1$. The number of affected 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.

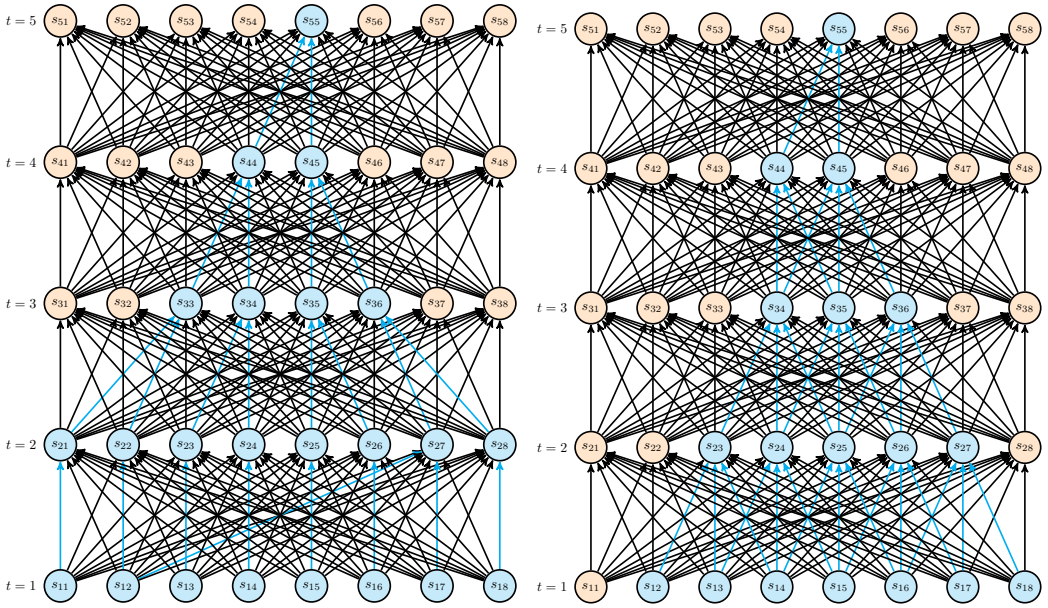


Figure A1: Examples of how affected states (cyan) affect states in a previous time step (indicated by cyan edges) starting with $n_5 = 1$ (one affected state). The left panel shows no overlap since affected states in s_{t-1} connect only to one affected state in s_t. The right panel shows some overlap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

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

$$a_k \;=\; \sum_{t=0}^{k} \left(1 \;-\; \left(1 \;-\; \frac{c_t}{N_{t-1}}\right)^{n_t}\right) \; N_{t-1} \; . \tag{A102}$$

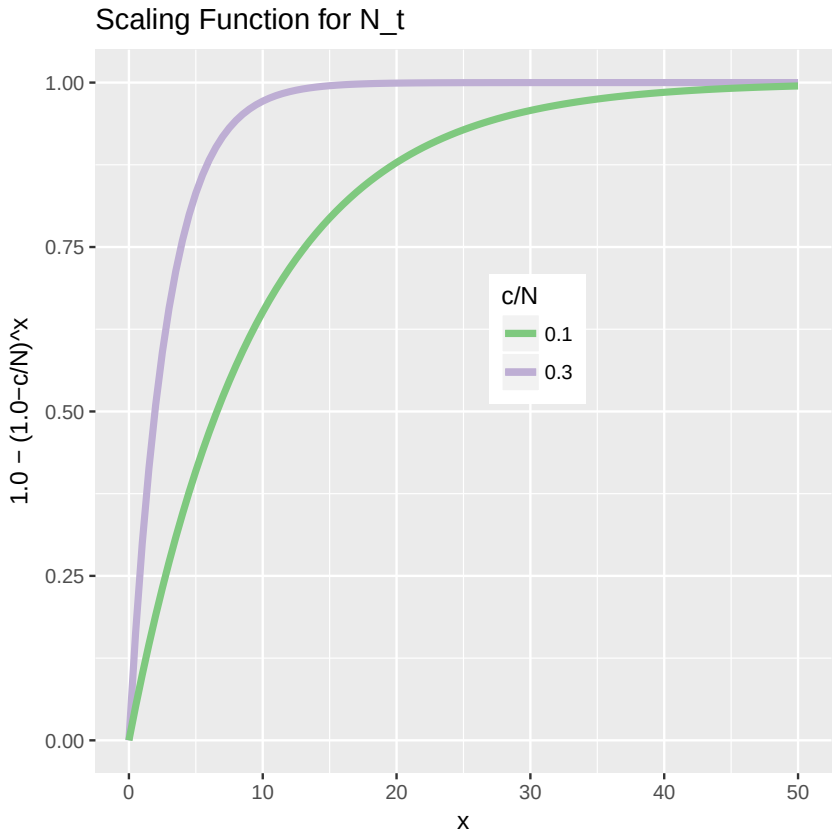


Figure A2: The function $\left(1-\left(1-\frac{c_t}{N_{t-1}}\right)^{n_t}\right)$ which scales N_{t-1} in Theorem A3. This function determines the growth of a_k, which is exponentially at the beginning, and then linearly when the function approaches 1.

Proof. The "backward induction algorithm" [86, 87] gives with $\boldsymbol{V}_{T+1}^{\pi} = \mathbf{0}$ and on-site variance $\boldsymbol{\omega}_{T+1} = \mathbf{0}$:

$$\boldsymbol{V}_t^{\pi} = \sum_{k=t}^{T} \prod_{\tau=t}^{k-1} \boldsymbol{P}_{\tau} \, \boldsymbol{\omega}_k \,, \tag{A103}$$

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 $\left\|\boldsymbol{P}_{t\leftarrow k}\right\|_{\infty} = 1$ according to Lemma A2, each on-site variance $\boldsymbol{\omega}_k$ with $t \leqslant k$ can have large effects on $\boldsymbol{V}_t^{\pi}$. Using the row-stochastic matrices $\boldsymbol{P}_{t\leftarrow k}$, we can reformulate the variance:

$$\boldsymbol{V}_t^{\pi} = \sum_{k=t}^{T} \boldsymbol{P}_{t\leftarrow k} \, \boldsymbol{\omega}_k \,, \tag{A104}$$

with $\left\|\boldsymbol{P}_{t\leftarrow k}\right\|_{\infty} = 1$. 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 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 [13]. We follow the approach of Cox et al. [13]. 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

$$\left(1 \; - \; \frac{c_t}{N_{t-1}}\right)^{n_t} \; . \tag{A105}$$

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

$$1 \; - \; \left(1 \; - \; \frac{c_t}{N_{t-1}}\right)^{n_t} \; . \tag{A106}$$

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

$$n_{t-1} \; = \; \left(1 \; - \; \left(1 \; - \; \frac{c_t}{N_{t-1}}\right)^{n_t}\right) \; N_{t-1} \; . \tag{A107}$$

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

$$a_k \; = \; \sum_{t=0}^{k} \left(1 \; - \; \left(1 \; - \; \frac{c_t}{N_{t-1}}\right)^{n_t}\right) \; N_{t-1} \; . \tag{A108}$$

□

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

$$a_k \; > \; \bar{c}^k \; . \tag{A109}$$

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

$$a_k \; \approx \; a_{\hat{t}-1} \; + \; (k - \hat{t} + 1) \; \bar{N} \; . \tag{A110}$$

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

$$\left(1 \; - \; \frac{c_t}{N_{t-1}}\right)^{n_t} \; \approx \; 1 \; - \; \frac{c_t \; n_t}{N_{t-1}} \; , \tag{A111}$$

thus

$$n_{t-1} \; \approx \; c_t \; n_t \; . \tag{A112}$$

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

$$\left(1 \; - \; \frac{c_t}{N_{t-1}}\right)^{n_t} \; = \; \left(\left(1 \; + \; \frac{-c_t}{N_{t-1}}\right)^{N_{t-1}}\right)^{n_t/N_{t-1}} \; \approx \; \exp(-(c_t \; n_t)/N_{t-1}) \; . \tag{A113}$$

We obtain

$$n_{t-1} \; = \; (1 \; - \; \exp(-(c_t \; n_t)/N_{t-1})) \; N_{t-1} \; . \tag{A114}$$

For small n_t, we again have

$$n_{t-1} \; \approx \; c_t \; n_t \; . \tag{A115}$$

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

$$n_t \; \approx \; \prod_{\tau=t}^{k} c_\tau \; \approx \; \bar{c}^{k-t} \; . \tag{A116}$$

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

$$a_k \; = \; \sum_{t=0}^{k} n_t \; \approx \; \sum_{t=0}^{k} \bar{c}^{k-t} \; = \; \sum_{t=0}^{k} \bar{c}^t \; = \; \frac{\bar{c}^{k+1} - 1}{\bar{c} - 1} \; > \; \bar{c}^k \; . \tag{A117}$$

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

$$n_t \;\approx\; N_t \,. \tag{A118}$$

Therefore

$$a_k \;-\; a_{\hat{t}-1} \;=\; \sum_{t=\hat{t}}^{k} n_t \;\approx\; \sum_{t=\hat{t}}^{k} N_t \;\approx\; (k - \hat{t} + 1)\, \bar{N} \,. \tag{A119}$$

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

Therefore, 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 rewards 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.

A2.3 Reward Redistribution

A2.3.1 Return-Equivalent MDPs

Our goal is to compare Markov decision processes (MDPs) without delayed reward to MDPs with delayed reward. Toward this end, we consider two 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.*

The definition of return-equivalence can be generalized to strict monotonic functions f for which $\tilde{v}_0^\pi = f(v_0^\pi)$. Since strict monotonic functions do not change the ordering of the returns, maximal returns stay maximal after applying the function f.

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. □

Special Cases of Return-Equivalent Decision Processes: Reward Shaping, Look-Ahead Advice and Look-Back Advice. Redistributing the reward via reward shaping [80, 129], look-ahead advice and look-back advice [130] is a special case of reward redistribution that lead to MDPs return-equivalent to the original MDP. In order to show that this is a special case, we subtract from the potential the constant $c = (\Phi(s_0, a_0) - \gamma^T \Phi(s_T, a_T))/(1 - \gamma^T)$, which is the potential of the initial state minus the discounted potential in the last state divided by a fixed divisor. Consequently, the sum of additional rewards in reward shaping, look-ahead advice, or look-back advice from 1 to T is zero. The original sum of additional rewards is

$$\sum_{i=1}^{T} \gamma^{i-1} \left(\gamma \Phi(s_i, a_i) \;-\; \Phi(s_{i-1}, a_{i-1})\right) \;=\; \gamma^T \Phi(s_T, a_T) \;-\; \Phi(s_0, a_0) \,. \tag{A120}$$

If we assume $\gamma^T \Phi(s_T, a_T) = 0$ and $\Phi(s_0, a_0) = 0$, then reward shaping does not change the return and the shaping reward is a reward redistribution leading to an MDP that is return-equivalent to the original MDP. For $T \to \infty$ only $\Phi(s_0, a_0) = 0$ is required. The assumptions can always be fulfilled by adding a single new initial state and a single new final state to the original MDP.
Without the assumptions $\gamma^T \Phi(s_T, a_T) = 0$ and $\Phi(s_0, a_0) = 0$, we subtract $c = (\Phi(s_0, a_0) - \gamma^T \Phi(s_T, a_T))/(1 - \gamma^T)$ from all potentials Φ, and obtain

$$\sum_{i=1}^{T} \gamma^{i-1} \left(\gamma(\Phi(s_i, a_i) \;-\; c) \;-\; (\Phi(s_{i-1}, a_{i-1}) \;-\; c)\right) \;=\; 0 \,. \tag{A121}$$

Therefore, the potential-based shaping function (the additional reward) added to the original reward does not change the return, which means that the shaping reward is a reward redistribution that leads

to an MDP that is return-equivalent to the original MDP. Obviously, reward shaping is a special case of reward redistribution that leads to a return-equivalent MDP, but the opposite is not true. Reward shaping does not change the general learning behavior if a constant c is subtracted from the potential function Φ. The Q-function of the original reward shaping and the Q-function of the reward shaping, which has a constant c subtracted from the potential function Φ, differ by c for every Q-value [80, 129].
For infinite horizon MDPs with $\gamma < 1$, the terms γ^T and $\gamma^T \Phi(s_T, a_T)$ vanish, therefore it is sufficient to subtract $c = \Phi(s_0, a_0)$ from the potential function.
Since the original reward is kept, the reward redistribution via reward shaping, look-ahead advice and look-back advice does not correspond to an optimal return decomposition according to Section A2.3.4. Therefore, learning by TD can be still exponentially slow for delayed rewards.
As discussed in Paragraph A2.3.4, the optimal reward redistribution does not comply to the Bellman equation. Also look-ahead advice does not comply to the Bellman equation. The return for the look-ahead advice reward $\tilde{r}_{t+1}$ is

$$G_t \;=\; \sum_{i=0}^{\infty} \tilde{r}_{t+i+1}(s_{t+i+1}, a_{t+i+1}, s_{t+i}, a_{t+i}) \tag{A122}$$

with

$$\tilde{r}_{t+1}(s_{t+1}, a_{t+1}, s_t, a_t) \;=\; \gamma \Phi(s_{t+1}, a_{t+1}) \;-\; \Phi(s_t, a_t) \; . \tag{A123}$$

The Q-value is given by

$$\tilde{q}^{\pi}(s_t, a_t) \;=\; \mathrm{E}_{\pi}\left[G_t \mid s_t, a_t\right] \; . \tag{A124}$$

However, $\tilde{r}_t(s_t, a_t, s_{t-1}, a_{t-1})$ also depends on a_t. Thus, the Q-value of the optimal policy does not solely determine the optimal action. For deriving the Bellman equation we need the split

$$\begin{aligned} G_t \;&=\; \tilde{r}_{t+1}(s_{t+1}, a_{t+1}, s_t, a_t) \;+\; \sum_{i=1}^{\infty} \tilde{r}_{t+i+1}(s_{t+i+1}, a_{t+i+1}, s_{t+i}, a_{t+i}) \\ &=\; \tilde{r}_{t+1}(s_{t+1}, a_{t+1}, s_t, a_t) \;+\; G_{t+1} \; , \end{aligned} \tag{A125}$$

however, $\tilde{r}_{t+1}(s_{t+1}, a_{t+1}, s_t, a_t)$ and $\tilde{r}_{t+1}(s_{t+2}, a_{t+2}, s_{t+1}, a_{t+1})$ are not independent as they share s_{t+1}, a_{t+1}. Consequently, the future return G_{t+1} cannot be treated separately from the immediate reward $\tilde{r}_{t+1}(s_{t+1}, a_{t+1}, s_t, a_t)$. The shaping reward can be split with respect to the actions since the current action does not influence the past rewards.

A2.3.2 State Enriched MDPs

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

Definition A4 (Ravindran and Barto [89, 90])**.** *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 $s, s' \in \mathcal{S}, a \in \mathcal{A}_s$:*

$$\tilde{p}(f(s') \mid f(s), g_s(a)) \;=\; \sum_{s'' \in [s']_f} p(s'' \mid s, a) \; , \tag{A126}$$

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

We use $[s]_f = [s']_f$ 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 [89])**.** *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 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 results 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" [27]. 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 and state aggregation for Markov decision processes, which covers homomorphic MDPs [65].
A Markov decision process $\tilde{\mathcal{P}}$ is state-enriched compared to an 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 nor the Q-values.

A2.3.3 Transforming an Immediate Reward to a Delayed Reward MDP

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 also the transformed process $\tilde{\mathcal{P}}$ is a Markov decision process.
The transformed delayed state-enriched Markov process has

- reward:

$$\tilde{R}_t \;=\; \begin{cases} 0\,, & \text{for } t \leqslant T \\ \sum_{k=0}^{T} R_{k+1}\,, & \text{for } t = T+1\,. \end{cases} \tag{A128}$$

- state:

$$\tilde{s}_t \;=\; (s_t, \rho_t)\,, \tag{A129}$$

$$\rho_t \;=\; \sum_{k=0}^{t-1} r_{k+1}\,, \quad \text{with } R_k = r_k\,. \tag{A130}$$

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 $\tilde{\pi}(a \mid \tilde{s}) = \pi(a \mid s)$*

$$\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;=\; q^{\pi}(s, a) \;+\; \sum_{k=0}^{t-1} r_{k+1}\,, \tag{A131}$$

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 $R_k = r_k$. Then, 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:

$$\mathrm{E}_\pi\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, A_t = a, \rho = \sum_{k=0}^{t-1} r_{k+1}\right] \;=\; \mathrm{E}_\pi\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, A_t = a\right]\,. \tag{A132}$$

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

We obtain

$$\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s},a) \;&=\; \mathrm{E}_{\tilde{\pi}}\left[\tilde{G}_0 \mid \tilde{S}_t = \tilde{s}, A_t = a\right] \qquad \text{(A133)}\\
&=\; \mathrm{E}_{\tilde{\pi}}\left[\sum_{k=0}^{T} R_{k+1} \mid S_t = s, \rho = \sum_{k=0}^{t-1} r_{k+1}, A_t = a\right]\\
&=\; \mathrm{E}_{\tilde{\pi}}\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, \rho = \sum_{k=0}^{t-1} r_{k+1}, A_t = a\right] \;+\; \sum_{k=0}^{t-1} r_{k+1}\\
&=\; \mathrm{E}_{\pi}\left[\sum_{k=t}^{T} R_{k+1} \mid S_t = s, A_t = a\right] \;+\; \sum_{k=0}^{t-1} r_{k+1}\\
&=\; q^{\pi}(s,a) \;+\; \sum_{k=0}^{t-1} r_{k+1}\;.
\end{aligned}$$

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

$$\tilde{q}^{*}(\tilde{s},a) \;=\; q^{*}(s,a) \;+\; \sum_{k=0}^{t-1} r_{k+1}\;. \qquad \text{(A134)}$$

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

$$\tilde{q}^{\tilde{\pi}}(\tilde{s},a) \;:=\; q^{\pi}(s,a) \;+\; \sum_{k=0}^{t-1} r_{k+1}\;, \qquad \text{(A135)}$$

$$\tilde{\pi}(a \mid \tilde{s}) \;:=\; \pi(a \mid s)\;. \qquad \text{(A136)}$$

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

$$\tilde{p}(\tilde{s}', \tilde{r} \mid s, \rho, a) \;=\; \tilde{p}(s', \rho', \tilde{r} \mid s, \rho, a) \;=\; p(s', r \mid s, a)\;. \qquad \text{(A137)}$$

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

$$\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s},a) \;&=\; q^{\pi}(s,a) \;+\; \sum_{k=0}^{t-1} r_{k+1} \qquad \text{(A138)}\\
&=\; \sum_{s',r} p(s', r \mid s, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a')\right] \;+\; \sum_{k=0}^{t-1} r_{k+1}\\
&=\; \sum_{s',\rho'} \tilde{p}(s', \rho', \tilde{r} \mid s, \rho, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a')\right] \;+\; \sum_{k=0}^{t-1} r_{k+1}\\
&=\; \sum_{\tilde{s}',\tilde{r}} \tilde{p}(\tilde{s}', \tilde{r} \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \;+\; \sum_{k=0}^{t-1} r_{k+1}\right]\\
&=\; \sum_{\tilde{s}',\tilde{r}} \tilde{p}(\tilde{s}', \tilde{r} \mid \tilde{s}, a) \left[\tilde{r} \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \;+\; \sum_{k=0}^{t} r_{k+1}\right]\\
&=\; \sum_{\tilde{s}',\tilde{r}} \tilde{p}(\tilde{s}', \tilde{r} \mid \tilde{s}, a) \left[\tilde{r} \;+\; \sum_{a'} \tilde{\pi}(a' \mid s')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right]\;.
\end{aligned}$$

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

$r = r_{T+1}$:

$$\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;&=\; q^{\pi}(s, a) \;+\; \sum_{k=0}^{T-1} r_{k+1} \qquad \text{(A139)} \\
&= \sum_{s',r} p(s', r \mid s, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \;+\; \sum_{k=0}^{T-1} r_{k+1} \\
&= \sum_{s',\rho',r} \tilde{p}(s', \rho' \mid s, \rho, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \;+\; \sum_{k=0}^{T-1} r_{k+1} \\
&= \sum_{s',\rho',r} \tilde{p}(s', \rho' \mid s, \rho, a) \left[\sum_{k=0}^{T} r_{k+1} \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \\
&= \sum_{\tilde{s}',\rho'} \tilde{p}(\tilde{s}' \mid \tilde{s}, a) \left[\rho' \;+\; \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a') \right] \\
&= \sum_{\tilde{s}',\rho'} \tilde{p}(\tilde{s}' \mid \tilde{s}, a) \; [\rho' \;+\; 0] \\
&= \sum_{\tilde{s}',\tilde{r}} \tilde{p}(\tilde{s}' \mid \tilde{s}, a) \left[\tilde{r} \;+\; \sum_{a'} \tilde{\pi}(a' \mid s')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a') \right] .
\end{aligned}$$

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

□

A2.3.4 Reward Redistribution by Return Decomposition

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 $\mathcal{P}$. The MDP $\mathcal{P}$ 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 according to their individual contribution to the final reward. Therefore, we have to decompose the return into a sum of rewards at different time steps.

Return Decomposition Idea. We want to decompose the return of a delayed reward MDP into contributions at every time step, so that the expected return is redistributed across the state-action sequence. Thus, a return decomposition supplies a reward redistribution. We assume a delayed reward MDP $\tilde{\mathcal{P}}$ with reward

$$\tilde{R}_t \;=\; \begin{cases} 0\,, & \text{for } t \leqslant T \\ \tilde{R}_{T+1}\,, & \text{for } t = T+1\,, \end{cases} \qquad \text{(A140)}$$

where $\tilde{R}_t = 0$ means that the random variable $\tilde{R}_t$ is always zero. The expected reward at the last time step is

$$\tilde{r}(s_T, a_T) \;=\; \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T \right] , \qquad \text{(A141)}$$

which is also the expected return. We want to redistribute the final expected reward $\tilde{r}_{T+1}$, which is the expected return, to previous time steps.
The *return decomposition idea* is to predict $\tilde{r}_{T+1}$ by a function g from the state-action sequence

$$(s, a)_{0:T} \;:=\; (s_0, a_0, s_1, a_1, \ldots, s_T, a_T) \qquad \text{(A142)}$$

and, subsequently, to distribute $\tilde{r}_{T+1}$ over the sequence with the help of g. We want to determine for each sequence element its contribution to the prediction of $\tilde{r}_{T+1}$. Thus, backward analysis is realized by contribution analysis. Contribution analysis computes the contribution of each input to the final prediction, that is, the information of each input about the final prediction. In principle, we can use any contribution analysis method that was mentioned in the introduction. However, we prefer three methods: (A) Differences in predictions. If we can ensure that g predicts $\tilde{r}_{T+1}$ at every time step by $g\big((s, a)_{0:t}\big) = \mathrm{E}_{\pi}\left[\tilde{R}_{T+1} \mid s_t, a_t \right]$, then the change in prediction is a measure of the

contribution of the current input to the final prediction. Thus, the current change in the prediction assesses the information gain by the current input. The difference of consecutive predictions is the redistributed reward. (B) Integrated gradients (IG) [112]. (C) Layer-wise relevance propagation (LRP) [3]. The methods (B) and (C) use information later in the sequence for determining the contribution of the current input. Therefore, they introduce a non-Markovian reward since it depends on later sequence elements. However, the non-Markovian reward can be viewed as probabilistic reward. Since probabilistic reward increases the variance, we prefer method (A).
The problem with this approach is that the Markov property ensures that $\tilde{r}_{T+1} = \tilde{r}_{T+1}(s_T, a_T)$ is a function of the last state-action pair $(s_T.a_T)$. To eliminate this Markov property, we define a difference $\Delta(s_{t-1}, a_{t-1}, s_t, a_t)$ between the state-action pair (s_t, a_t) and its predecessor (s_{t-1}, a_{t-1}), where (s_{-1}, a_{-1}) are introduced for starting an episode. The sequence of differences is defined as

$$\Delta_{0:T} \ := \ \big(\Delta(s_{-1}, a_{-1}, s_0, a_0), \ldots, \Delta(s_{T-1}, a_{T-1}, s_T, a_T)\big) \ . \tag{A143}$$

We assume that the differences Δ are mutually independent [52]:

$$\begin{aligned} &p\left(\Delta(s_{t-1}, a_{t-1}, s_t, a_t) \mid \Delta(s_{-1}, a_{-1}, s_0, a_0), \ldots, \Delta(s_{t-2}, a_{t-2}, s_{t-1}, a_{t-1}),\right. \\ &\left.\Delta(s_t, a_t, s_{t+1}, a_{t+1}) \ldots, \Delta(s_{T-1}, a_{T-1}, s_T, a_T)\right) \ = \ p\left(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)\right) \ . \end{aligned} \tag{A144}$$

The function g predicts the average delayed reward $\tilde{r}(s_T, a_T)$ from the sequence $\Delta_{0:T}$:

$$g\big(\Delta_{0:T}\big) \ = \ \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right] \ = \ \tilde{r}(s_T, a_T) \ . \tag{A145}$$

We decompose g into contributions $h_t = h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)$ at time t:

$$g\big(\Delta_{0:T}\big) \ = \ \sum_{t=0}^{T} h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \ = \ \tilde{r}(s_T, a_T) \ . \tag{A146}$$

If we can assume that g can predict the return at every time step:

$$g\big((s,a)_{0:t}\big) \ = \ \mathrm{E}_\pi\left[\tilde{R}_{T+1} \mid s_t, a_t\right] \ , \tag{A147}$$

then we can define the decomposition of g through contributions at different time steps

$$h_0 \ = \ h(\Delta(s_{-1}, a_{-1}, s_0, a_0)) \ := \ g\big((s,a)_{0:0}\big) \tag{A148}$$

$$h_t \ = \ h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \ := \ g\big((s,a)_{0:t}\big) \ - \ g\big((s,a)_{0:(t-1)}\big) \ . \tag{A149}$$

We assume that the final accumulated reward cannot be predicted from the last state. The reason is that immediate rewards are given only at sequence end or information is removed in the input. Therefore, a relevant event for predicting the final reward must be stored in the function g. The prediction errors at the end of the episode are, in general, smaller since the future is less random. Therefore, prediction errors later in the episode are up-weighted while early predictions ensure that information is captured in h_t for being used later. The prediction of $\tilde{r}(s_T, a_T)$ at time T has the largest weight. It relies on information from the past that has been stored for multiple continuous predictions.
If g does predict the return at every time step, backward analysis decomposes g via contribution analysis. For decomposing a linear g one can use the Taylor decomposition (a linear approximation) of g with respect to the h [3, 75]. A non-linear g can be decomposed by layerwise relevance propagation (LRP) [3, 76] or integrated gradients (IG) [112].

Reward Redistribution based on Return Decomposition. The optimal policies do not change if the return of the redistributed reward is equal to the original return, since both MDPs are return-equivalent. We assume a return decomposition

$$g\big(\Delta_{0:T}\big) \ = \ \sum_{t=0}^{T} h_t \ , \tag{A150}$$

with

$$h_0 \ = \ h(\Delta(s_{-1}, a_{-1}, s_0, a_0)) \ , \tag{A151}$$

$$h_t \ = \ h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \quad \text{for } 0 < t \leqslant T \ . \tag{A152}$$

To assure return-equivalent MDPs, we have to compensate for both a probabilistic reward $\tilde{R}_{T+1}$ and an imperfect function g with $g(\Delta_{0:T}) \neq \tilde{r}_{T+1}$. The compensation is given by

$$\tilde{R}_{T+1} \; - \; \sum_{\tau=0}^{T} h_t \; . \tag{A153}$$

We compensate with an extra reward R_{T+2} at time $T+2$ which is immediately given after R_{T+1} at time $T+1$ after the state-action pair (s_T, a_T). The reward redistribution based on a return decomposition is given using the random variable R_t for the reward at time t:

$$R_1 \; = \; h_0 \; , \tag{A154}$$

$$R_{t+1} \; = \; h_t \quad \text{for } 0 < t \leqslant T \; , \tag{A155}$$

$$R_{T+2} \; = \; \tilde{R}_{T+1} \; - \; \sum_{t=0}^{T} h_t \; . \tag{A156}$$

The next theorem states that the MDP $\mathcal{P}$ with reward redistribution does not change the optimal policies of the original MDP $\tilde{\mathcal{P}}$ since $\mathcal{P}$ is return-equivalent to $\tilde{\mathcal{P}}$.

Theorem A4. *The MDP $\mathcal{P}$ with redistributed reward R_t has the same optimal policies as the MDP $\tilde{\mathcal{P}}$ with delayed reward.*

Proof. The redistribution gives

$$\tilde{G}_0 \; = \; \sum_{t=0}^{T} \tilde{R}_{t+1} \; = \; \tilde{R}_{T+1} \; = \; \sum_{t=0}^{T+1} R_{t+1} \; = \; G_0 \; , \tag{A157}$$

which shows

$$v_0^\pi \; = \; \mathrm{E}_\pi \left[\sum_{t=0}^{T+1} R_{t+1} \mid s_0 \right] \; = \; \mathrm{E}_\pi \left[\tilde{R}_{T+1} \mid s_0 \right] \; = \; \mathrm{E}_\pi \left[\sum_{t=0}^{T} \tilde{R}_{t+1} \mid s_0 \right] \; = \; \tilde{v}_0^\pi \; . \tag{A158}$$

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. □

For the expected reward we obtain for $0 \leqslant t \leqslant T$

$$r(s_t, a_t) \; = \; \mathrm{E}\left[R_{t+1} \mid s_t, a_t \right] \; = \; \mathrm{E}\left[h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \mid s_t, a_t \right] \; , \tag{A159}$$

and for the last correcting reward we have

$$r_{T+1} \; = \; \mathrm{E}\left[R_{T+2} \mid s_T, a_T \right] \; = \; \mathrm{E}\left[\tilde{R}_{T+1} \; - \; \sum_{t=0}^{T} h_t \mid s_T, a_T \right] \; = \; \tilde{r}(s_T, a_T) \; - \; \sum_{t=0}^{T} \mathrm{E}\left[h_t \mid s_T, a_T \right] \; . \tag{A160}$$

In the experiments we also use a uniform compensation where each reward has the same contribution to the compensation:

$$R_1 \; = \; h_0 \; + \; \frac{1}{T+1} \left(\tilde{R}_{T+1} - \sum_{\tau=0}^{T} h(\Delta(s_{\tau-1}, a_{\tau-1}, s_\tau, a_\tau)) \right) \tag{A161}$$

$$R_{t+1} \; = \; h_t \; + \; \frac{1}{T+1} \left(\tilde{R}_{T+1} - \sum_{\tau=0}^{T} h(\Delta(s_{\tau-1}, a_{\tau-1}, s_\tau, a_\tau)) \right) \; . \tag{A162}$$

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

$$\tilde{q}^\pi(s_T, a_T) \; = \; \tilde{r}(s_T, a_T) \; . \tag{A163}$$

The decomposition Eq. (A146) becomes

$$g\big(\Delta_{0:T}\big) \; = \; \sum_{t=0}^{T} h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \; = \; \tilde{q}^\pi(s_T, a_T) \; . \tag{A164}$$

An optimal prediction must track constantly the expected final return due to the independence condition Eq. (A144). For an optimal decomposition we require Eq. (A164) to be correct for the partial sums.

Definition A6. *An return decomposition is* optimal *if it fulfills following equation the for partial sums* $0 \leqslant t \leqslant T$*:*

$$g\big(\Delta_{0:t}\big) \;=\; \sum_{\tau=0}^{t} h(\Delta(s_{\tau-1}, a_{\tau-1}, s_\tau, a_\tau)) \;=\; \tilde{q}^\pi(s_t, a_t)\;. \tag{A165}$$

We obtain for $t = 0$

$$g\big(\Delta_{0:0}\big) \;=\; h(\Delta(s_{-1}, a_{-1}, s_0, a_0)) \;=\; \tilde{q}^\pi(s_0, a_0) \tag{A166}$$

and for $t > 0$

$$g\big(\Delta_{0:t}\big) \;-\; g\big(\Delta_{0:t-1}\big) \;=\; h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1})\;. \tag{A167}$$

Therefore, for an optimal decomposition, we obtain for the reward redistribution

$$R_1 \;=\; h_0 \;=\; h(\Delta(s_{-1}, a_{-1}, s_0, a_0)) \;=\; \tilde{q}^\pi(s_0, a_0)\;, \tag{A168}$$

$$R_{t+1} \;=\; h_t \;=\; h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \quad \text{for } 0 < t \leqslant T\;, \tag{A169}$$

$$R_{T+2} \;=\; \tilde{R}_{T+1} \;-\; \tilde{q}^\pi(s_T, a_T)\;, \tag{A170}$$

since $\sum_{t=0}^{T} h_t = \tilde{q}^\pi(s_T, a_T)$. Again we have ensured that

$$\sum_{t=0}^{T+1} R_{t+1} \;=\; \tilde{R}_{T+1}\;. \tag{A171}$$

The next theorem gives the major advantage of a redistributed reward by an optimal decomposition. In this case, the Q-values of an MDP with reward redistribution based on an optimal return decomposition are equal to the expected immediate reward.

Theorem A5. *If the redistributed reward is based on an optimal return decomposition, then the Q-values are given by*

$$\begin{aligned} q^\pi(s_t, a_t) \;&=\; r(s_t, a_t) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \mathrm{E}_{s_{t-1}, a_{t-1}}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t\right] \\ &=\; \tilde{q}^\pi(s_t, a_t) \;-\; \psi(s_t) \end{aligned} \tag{A172}$$

and the MDP $\mathcal{P}$ with reward redistribution and the original MDP $\tilde{\mathcal{P}}$ with delayed reward have the same advantage function.

Proof. The expected reward is computed for $0 \leqslant t \leqslant T$, where s_{-1}, a_{-1} are states and actions, which are introduced at the beginning of an episode. The expected reward is with $\tilde{q}^\pi(s_{-1}, a_{-1}) = 0$:

$$\begin{aligned} r(s_t, a_t) \;&=\; \mathrm{E}_{R_{t+1}}\left[R_{t+1} \mid s_t, a_t\right] \;=\; \mathrm{E}_{s_{t-1}, a_{t-1}}\left[\tilde{q}^\pi(s_t, a_t) \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right] \\ &=\; \tilde{q}^\pi(s_t, a_t) \;-\; \mathrm{E}_{s_{t-1}, a_{t-1}}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right]\;. \end{aligned} \tag{A173}$$

The Q-values are defined for $0 \leqslant t \leqslant T$:

$$\begin{aligned} q^\pi(s_t, a_t) \;&=\; \mathrm{E}_\pi\left[\sum_{\tau=0}^{T-t+1} R_{t+1+\tau} \mid s_t, a_t\right] \\ &=\; \mathrm{E}_\pi\left[\tilde{R}_{T+1} \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right] \\ &=\; \mathrm{E}_\pi\left[\tilde{R}_{T+1} \mid s_t, a_t\right] \;-\; \mathrm{E}_\pi\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right] \\ &=\; \tilde{q}^\pi(s_t, a_t) \;-\; \mathrm{E}_{s_{t-1}, a_{t-1}}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right] \\ &=\; r(s_t, a_t)\;. \end{aligned} \tag{A174}$$

The posterior $p(s_{t-1}, a_{t-1} \mid s_t, a_t)$ is

$$\begin{aligned} p(s_{t-1}, a_{t-1} \mid s_t, a_t) \;&=\; \frac{p(s_t, a_t \mid s_{t-1}, a_{t-1})\, p(s_{t-1}, a_{t-1})}{p(s_t, a_t)} \\ &=\; \frac{p(s_t \mid s_{t-1}, a_{t-1})\, p(s_{t-1}, a_{t-1})}{p(s_t)} \;=\; p(s_{t-1}, a_{t-1} \mid s_t)\;, \end{aligned} \tag{A175}$$

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 $p(s_t, a_t) = \pi(a_t \mid s_t) p(s_t)$. The posterior does no longer contain a_t. We can express the mean of previous Q-values by the posterior $p(s_{t-1}, a_{t-1} \mid s_t, a_t)$:

$$\mathrm{E}_{s_{t-1}, a_{t-1}} \left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t \right] = \sum_{s_{t-1}, a_{t-1}} p(s_{t-1}, a_{t-1} \mid s_t, a_t) \, \tilde{q}^\pi(s_{t-1}, a_{t-1}) \tag{A176}$$

$$= \sum_{s_{t-1}, a_{t-1}} p(s_{t-1}, a_{t-1} \mid s_t) \, \tilde{q}^\pi(s_{t-1}, a_{t-1}) = \mathrm{E}_{s_{t-1}, a_{t-1}} \left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t \right] = \psi(s_t) \,,$$

with $\psi(s_t) = \mathrm{E}_{s_{t-1}, a_{t-1}} \left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t \right]$.
$\mathcal{P}$ and $\tilde{\mathcal{P}}$ have the same advantage function, since the value functions follow the equation $v^\pi(s_t) = \tilde{v}^\pi(s_t) + \psi(s_t)$. □

We now define the expected sum of delayed rewards $\kappa(m, t-1)$, which measures the amount of delayed reward.

Definition A7. *For $1 \leqslant t \leqslant T+1$ and $0 \leqslant m \leqslant T-t+1$, the expected sum of delayed rewards at time $(t-1)$ in the interval $[t, t+m]$ is defined as*

$$\kappa(m, t-1) = \mathrm{E}_\pi \left[\sum_{\tau=0}^{m} R_{t+1+\tau} \mid s_{t-1}, a_{t-1} \right] . \tag{A177}$$

Q-values are decomposed by κ into immediate and delayed rewards: $q(s_t, a_t) = \mathrm{E}_{R_{t+1}} \left[R_{t+1} \mid s_t, a_t \right] + \kappa(T-t, t)$. The next theorem states necessary conditions for an optimal return decomposition based on κ.

Theorem A6. *An MDP $\mathcal{P}$ with a redistributed reward of an optimal return decomposition fulfills for $1 \leqslant t \leqslant T+1$ and $0 \leqslant m \leqslant T-t+1$ the necessary conditions for optimality:*

$$\kappa(m, t-1) = 0 \,. \tag{A178}$$

Proof. We start with $m = 0$ and compute $\mathrm{E}_\pi \left[R_{t+1} \mid s_{t-1}, a_{t-1} \right]$. Since $\tilde{r}_t = 0$ for $0 < t \leqslant T$, we have

$$\tilde{q}^\pi(s_{t-1}, a_{t-1}) = \tilde{r}_t + \sum_{s_t, a_t} p(s_t, a_t \mid s_{t-1}, a_{t-1}) \, \tilde{q}^\pi(s_t, a_t) \tag{A179}$$

$$= \sum_{s_t, a_t} p(s_t, a_t \mid s_{t-1}, a_{t-1}) \, \tilde{q}^\pi(s_t, a_t) \,.$$

We use this equation in the following equation for $0 < t \leqslant T$:

$$\mathrm{E}_{s_t, a_t, R_{t+1}} \left[R_{t+1} \mid s_{t-1}, a_{t-1} \right] = \mathrm{E}_{s_t, a_t} \left[\tilde{q}^\pi(s_t, a_t) - \tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_{t-1}, a_{t-1} \right] \tag{A180}$$

$$= \sum_{s_t, a_t} p(s_t, a_t \mid s_{t-1}, a_{t-1}) \, \left(\tilde{q}^\pi(s_t, a_t) - \tilde{q}^\pi(s_{t-1}, a_{t-1}) \right)$$

$$= \tilde{q}^\pi(s_{t-1}, a_{t-1}) - \sum_{s_t, a_t} p(s_t, a_t \mid s_{t-1}, a_{t-1}) \, \tilde{q}^\pi(s_{t-1}, a_{t-1})$$

$$= \tilde{q}^\pi(s_{t-1}, a_{t-1}) - \tilde{q}^\pi(s_{t-1}, a_{t-1}) = 0 \,.$$

Next, we analyze the case $t = T+1$, therefore $m = 0$, for which we will use following equality for the expected delayed reward at sequence end:

$$\tilde{q}^\pi(s_T, a_T) = \mathrm{E}_{\tilde{R}_{T+1}} \left[\tilde{R}_{T+1} \mid s_T, a_T \right] = \tilde{r}_{T+1}(s_T, a_T) \,, \tag{A181}$$

since $\tilde{q}^\pi(s_{T+1}, a_{T+1}) = 0$. For $t = T+1$ we obtain

$$\mathrm{E}_{R_{T+2}} \left[R_{T+2} \mid s_T, a_T \right] = \mathrm{E}_{\tilde{R}_{T+1}} \left[\tilde{R}_{T+1} - \tilde{q}^\pi(s_T, a_T) \mid s_T, a_T \right] \tag{A182}$$

$$= \tilde{r}_{T+1}(s_T, a_T) - \tilde{r}_{T+1}(s_T, a_T) = 0 \,.$$

Next we consider the expectation of $\sum_{\tau=0}^{m} R_{t+1+\tau}$ for $1 \leqslant m < T-t+1$

$$
\begin{aligned}
& \mathrm{E}_\pi \left[\sum_{\tau=0}^{m} R_{t+1+\tau} \mid s_{t-1}, a_{t-1} \right] \qquad \text{(A183)} \\
&= \mathrm{E}_\pi \left[\sum_{\tau=0}^{m} (\tilde{q}^\pi(s_{\tau+t}, a_{\tau+t}) \; - \; \tilde{q}^\pi(s_{\tau+t-1}, a_{\tau+t-1})) \mid s_{t-1}, a_{t-1} \right] \\
&= \mathrm{E}_\pi \left[\tilde{q}^\pi(s_{t+m}, a_{t+m}) \; - \; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_{t-1}, a_{t-1} \right] \\
&= \mathrm{E}_\pi \left[\mathrm{E}_\pi \left[\sum_{\tau=t+m}^{T} \tilde{R}_{\tau+1} \mid s_{t+m}, a_{t+m} \right] \mid s_{t-1}, a_{t-1} \right] \\
&- \mathrm{E}_\pi \left[\mathrm{E}_\pi \left[\sum_{\tau=t-1}^{T} \tilde{R}_{\tau+1} \mid s_{t-1}, a_{t-1} \right] \mid s_{t-1}, a_{t-1} \right] \\
&= \mathrm{E}_\pi \left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1} \right] \; - \; \mathrm{E}_\pi \left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1} \right] \\
&= 0 \, .
\end{aligned}
$$

Expectations $\mathrm{E}_\pi \left[. \mid s_{t-1}, a_{t-1} \right]$ are expectations over all episodes starting in (s_{t-1}, a_{t-1}) and ending in some (s_{T+1}, a_{T+1}). We used that $\tilde{R}_{t+1} = 0$ for $t < T$.
Next we consider the expectation of $\sum_{\tau=0}^{T-t+1} R_{t+1+\tau}$, that is $m = T-t+1$

$$
\begin{aligned}
& \mathrm{E}_\pi \left[\sum_{\tau=0}^{T-t+1} R_{t+1+\tau} \mid s_{t-1}, a_{t-1} \right] \qquad \text{(A184)} \\
&= \mathrm{E}_\pi \left[\tilde{R}_{T+1} \; - \; \tilde{q}^\pi(s_T, a_T) \; + \; \sum_{\tau=0}^{T-t} (\tilde{q}^\pi(s_{\tau+t}, a_{\tau+t}) \; - \; \tilde{q}^\pi(s_{\tau+t-1}, a_{\tau+t-1})) \mid s_{t-1}, a_{t-1} \right] \\
&= \mathrm{E}_\pi \left[\tilde{R}_{T+1} \; - \; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_{t-1}, a_{t-1} \right] \\
&= \mathrm{E}_\pi \left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1} \right] \; - \; \mathrm{E}_\pi \left[\mathrm{E}_\pi \left[\sum_{\tau=t-1}^{T} \tilde{R}_{\tau+1} \mid s_{t-1}, a_{t-1} \right] \mid s_{t-1}, a_{t-1} \right] \\
&= \mathrm{E}_\pi \left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1} \right] \; - \; \mathrm{E}_\pi \left[\tilde{R}_{T+1} \mid s_{t-1}, a_{t-1} \right] \\
&= 0 \, .
\end{aligned}
$$

Expectation $\mathrm{E}_\pi \left[. \mid s_{t-1}, a_{t-1} \right]$ like $\mathrm{E}_\pi \left[\tilde{R}_{T+2} \mid s_{t-1}, a_{t-1} \right]$ are expectations over all episodes starting in (s_{t-1}, a_{t-1}) and ending in some (s_{T+1}, a_{T+1}). We used that $\tilde{R}_{t+1} = 0$ for $t < T$.

□

Thus, an MDP with reward redistribution based on an optimal return decomposition has a sum of zero expected delayed rewards. A related approach is to ensure zero return by reward shaping if the exact value function is known [103].
In particular, we have for $m = 0$ and for $m = T-t+1$ the necessary conditions for optimality

$$
\mathrm{E}_{s_t, a_t, R_{t+1}} \left[R_{t+1} \mid s_{t-1}, a_{t-1} \right] \; = \; 0 \, , \quad \mathrm{E}_\pi \left[\sum_{\tau=0}^{T-t+1} R_{t+1+\tau} \mid s_{t-1}, a_{t-1} \right] \; = \; 0 \, . \qquad \text{(A185)}
$$

Learning with Reward Redistribution. We consider for $\gamma = 1$: (i) learning if we assume optimal return decomposition, (ii) TD-learning of he expected sum of delayed rewards κ, and (iii) eligibility traces for optimal return decompositions.
(i) Assuming Optimality. According to Theorem A5, the Q-values for an optimal return decomposition are given by

$$
q^\pi(s_t, a_t) \; = \; r(s_t, a_t) \; = \; \mathrm{E}_{R_{t+1}} \left[R_{t+1} \mid s_t, a_t \right] \, , \qquad \text{(A186)}
$$

therefore it is sufficient to estimate the expected immediate reward for the Q-values. During learning we can assume optimality and only estimate the immediate reward to have an estimate of the Q-values.

(ii) TD-learning of κ and correction of the redistributed reward. If the return decomposition is, however, not optimal, then the reward can be further distributed back. In this case the expected sum of delayed rewards $\kappa(T,t)$ is not zero, and we can correct the return decomposition such that the necessary conditions for optimality are less violated. For the correction, we need an estimate of the expected sum of delayed rewards $\kappa(T,t)$.

TD-learning of κ. The expected sum of delayed rewards $\kappa(T,t)$ can be formulated as

$$\begin{aligned}
\kappa(T,t) &= \mathrm{E}_\pi\left[\sum_{\tau=0}^{T-(t+1)} R_{(t+1)+1+\tau} \mid s_t, a_t\right] \\
&= \mathrm{E}_\pi\left[R_{t+2} + \sum_{\tau=0}^{T-(t+2)} R_{(t+2)+1+\tau} \mid s_t, a_t\right] \\
&= \mathrm{E}_{s_{t+1},a_{t+1},R_{t+2}}\left[R_{t+2} + \mathrm{E}_\pi\left[\sum_{\tau=0}^{T-(t+2)} R_{(t+2)+1+\tau} \mid s_{t+1}, a_{t+1}\right] \mid s_t, a_t\right] \\
&= \mathrm{E}_{s_{t+1},a_{t+1},R_{t+2}}\left[R_{t+2} + \kappa(T,t+1) \mid s_t, a_t\right] .
\end{aligned} \tag{A187}$$

Therefore, $\kappa(T,t)$ can be estimated by R_{t+2} and $\kappa(T,t+1)$, if the last two are drawn together, i.e. considered as pairs. We can use TD-learning if the immediate reward and the sum of delayed rewards are drawn as pairs, that is, simultaneously. The TD-error δ_κ becomes

$$\delta_\kappa(T,t) = R_{t+2} + \kappa(T,t+1) - \kappa(T,t) . \tag{A188}$$

We now define eligibility traces for κ. Let the n-step return samples of κ for $1 \leqslant n \leqslant T-t$ be

$$\begin{aligned}
\kappa^{(1)}(T,t) &= R_{t+2} + \kappa(T,t+1) \\
\kappa^{(2)}(T,t) &= R_{t+2} + R_{t+3} + \kappa(T,t+2) \\
&\dots \\
\kappa^{(n)}(T,t) &= R_{t+2} + R_{t+3} + \ldots + R_{t+n+1} + \kappa(T,t+n) .
\end{aligned} \tag{A189}$$

The λ-return for κ is

$$\kappa^{(\lambda)}(T,t) = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1}\, \kappa^{(n)}(T,t) + \lambda^{T-t-1}\, \kappa^{(T-t)}(T,t) . \tag{A190}$$

We obtain

$$\begin{aligned}
\kappa^{(\lambda)}(T,t) &= R_{t+2} + \kappa(T,t+1) \\
&\quad + \lambda\, (R_{t+3} + \kappa(T,t+2) - \kappa(T,t+1)) \\
&\quad + \lambda^2\, (R_{t+4} + \kappa(T,t+3) - \kappa(T,t+2)) \\
&\dots \\
&\quad + \lambda^{T-1-t}\, (R_{T+1} + \kappa(T,T) - \kappa(T,T-1)) .
\end{aligned} \tag{A191}$$

We can reformulate this as

$$\kappa^{(\lambda)}(T,t) = \kappa(T,t) + \sum_{n=0}^{T-t-1} \lambda^n\, \delta_\kappa(T,t+n) . \tag{A192}$$

The κ error Δ_κ is

$$\Delta_\kappa(T,t) = \kappa^{(\lambda)}(T,t) - \kappa(T,t) = \sum_{n=0}^{T-t-1} \lambda^n\, \delta_\kappa(T,t+n) . \tag{A193}$$

The derivative of

$$1/2\, \Delta_\kappa(T,t)^2 = 1/2\, \left(\kappa^{(\lambda)}(T,t) - \kappa(T,t;\boldsymbol{w})\right)^2 \tag{A194}$$

with respect to $\boldsymbol{w}$ is

$$-\left(\kappa^{(\lambda)}(T,t) \;-\; \kappa(T,t;\boldsymbol{w})\right) \nabla_w \kappa(T,t;\boldsymbol{w}) \;=\; -\sum_{n=0}^{T-t-1} \lambda^n \, \delta_\kappa(T,t+n) \, \nabla_w \kappa(T,t;\boldsymbol{w}) \,. \tag{A195}$$

The full gradient of the sum of κ errors is

$$1/2 \, \nabla_w \sum_{t=0}^{T-1} \Delta_\kappa(T,t)^2 \;=\; -\sum_{t=0}^{T-1} \sum_{n=0}^{T-t-1} \lambda^n \, \delta_\kappa(T,t+n) \, \nabla_w \kappa(T,t;\boldsymbol{w}) \tag{A196}$$

$$= \; -\sum_{t=0}^{T-1} \sum_{\tau=t}^{T-1} \lambda^{\tau-t} \, \delta_\kappa(T,\tau) \, \nabla_w \kappa(T,t;\boldsymbol{w}) \;=\; -\sum_{\tau=0}^{T-1} \delta_\kappa(T,\tau) \sum_{t=0}^{\tau} \lambda^{\tau-t} \, \nabla_w \kappa(T,t;\boldsymbol{w}) \,.$$

We set $n = \tau - t$, so that $n = 0$ becomes $\tau = t$ and $n = T - t - 1$ becomes $\tau = T - 1$. The recursion

$$f(t) \;=\; \lambda \, f(t-1) \;+\; a_t \,, \qquad f(0) \;=\; 0 \tag{A197}$$

can be written as

$$f(T) \;=\; \sum_{t=1}^{T} \lambda^{T-t} \, a_t \,. \tag{A198}$$

Therefore, we can use following update rule for minimizing $\sum_{t=0}^{T-1} \Delta_\kappa(T,t)^2$ with respect to $\boldsymbol{w}$ with $1 \leqslant \tau \leqslant T-1$:

$$\boldsymbol{z}_{-1} \;=\; 0 \tag{A199}$$

$$\boldsymbol{z}_\tau \;=\; \lambda \, \boldsymbol{z}_{\tau-1} \;+\; \nabla_w \kappa(T,\tau;\boldsymbol{w}) \tag{A200}$$

$$\delta_\kappa(T,\tau) \;=\; R_{\tau+2} \;+\; \kappa(T,\tau+1;\boldsymbol{w}) \;-\; \kappa(T,\tau;\boldsymbol{w}) \tag{A201}$$

$$\boldsymbol{w}^{\text{new}} \;=\; \boldsymbol{w} \;+\; \alpha \, \delta_\kappa(T,\tau) \, \boldsymbol{z}_\tau \,. \tag{A202}$$

Correction of the reward redistribution. For correcting the redistributed reward, we apply a method similar to reward shaping or look-back advice. This method ensures that the corrected redistributed reward leads to an MDP that is return-equivalent to the original MDP. The reward correction is

$$F(s_t, a_t, s_{t-1}, a_{t-1}) \;=\; \kappa(m,t) \;-\; \kappa(m,t-1) \,, \tag{A203}$$

we define the corrected redistributed reward as

$$R^{\text{c}}_{t+1} \;=\; R_{t+1} \;+\; F(s_t, a_t, s_{t-1}, a_{t-1}) \;=\; R_{t+1} \;+\; \kappa(m,t) \;-\; \kappa(m,t-1) \,. \tag{A204}$$

We assume that $\kappa(m,-1) = \kappa(m,T+1) = 0$, therefore

$$\sum_{t=0}^{T+1} F(s_t, a_t, s_{t-1}, a_{t-1}) \;=\; \sum_{t=0}^{T+1} \kappa(m,t) \;-\; \kappa(m,t-1) \;=\; \kappa(m,T+1) \;-\; \kappa(m,-1) \;=\; 0 \,. \tag{A205}$$

Consequently, the corrected redistributed reward R^{c}_{t+1} leads to an MDP that is **return-equivalent** to the original MDP.

For a predictive reward of ρ at time $t = k$, which can be predicted from time $t = l < k$ to time $t = k - 1$, we have:

$$\kappa(m,t) \;=\; \begin{cases} 0 \,, & \text{for } t < l \,, \\ \rho \,, & \text{for } l \leqslant t < k \,, \\ 0 \,, & \text{for } t \geqslant k \,. \end{cases} \tag{A206}$$

The reward correction is

$$F(s_t, a_t, s_{t-1}, a_{t-1}) \;=\; \begin{cases} 0 \,, & \text{for } t < l \,, \\ \rho \,, & \text{for } t = l \,, \\ 0 \,, & \text{for } l < t < k \,, \\ -\rho \,, & \text{for } t = k \,, \\ 0 \,, & \text{for } t > k \,. \end{cases} \tag{A207}$$

Using κ as auxiliary task in predicting the return for return decomposition. A κ prediction can serve as additional output of the function g that predicts the return and is the basis of the return decomposition. Even a partly prediction of κ means that the reward can be distributed further back. If g can partly predict κ, then g has all information to predict the return earlier in the sequence. If the return is predicted earlier, then the reward will be distributed further back. Consequently, the return decomposition comes closer to an optimal decomposition. However, at the same time, κ can no longer be predicted. The function g must find another κ that can be predicted. If no such κ is found, then optimal return decomposition is indicated.

(iii) Eligibility Traces Assuming Optimality. We can use eligibility traces to further distribute the reward back. For a reward redistribution from an optimal return decomposition, we have $\mathrm{E}_{s_{t+1}}[V(s_{t+1})] = 0$. The new returns $\mathcal{G}_t$ are given by the recursion

$$\mathcal{G}_t \;=\; r_{t+1} \;+\; \lambda\, \mathcal{G}_{t+1}\;, \tag{A208}$$

$$\mathcal{G}_{T+2} \;=\; 0\;. \tag{A209}$$

The expected policy gradient updates with the new returns $\mathcal{G}$ are $\mathrm{E}_\pi\left[\nabla_\theta \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \mathcal{G}_t\right]$. To avoid an estimation of the value function $V(s_{t+1})$, we assume optimality, which might not be valid. However, the error should be small if the return decomposition works well. Instead of estimating a value function, we can use a correction as it is shown in next paragraph.

Remarks on the Optimal Return Decomposition. The Bellman equation for Q-values is

$$q^\pi(s_t, a_t) \;=\; r(s_t, a_t) \;+\; \mathrm{E}_{s_{t+1}, a_{t+1}}\left[q^\pi(s_{t+1}, a_{t+1}) \mid s_t, a_t\right]\;. \tag{A210}$$

However, the Bellman equation is in general not fulfilled for the reward redistribution based on optimal return decomposition. The reward is non Markovian since it includes $\tilde{q}^\pi(s_{t-1}, a_{t-1})$, which is the information from time step $(t-1)$. Therefore, from a sequence of rewards a single reward and a corresponding state-action pair cannot be split off as done for deriving the Bellman equation. Consequently, TD-methods like Q-learning or SARSA are not justified for the reward redistribution from an optimal return decomposition. In vector and matrix notation the Bellman equation is

$$\boldsymbol{q}_t^\pi \;=\; \boldsymbol{r}_t \;+\; \boldsymbol{P}_{t \to t+1}\, \boldsymbol{q}_{t+1}^\pi\;, \tag{A211}$$

where $\boldsymbol{P}_{t \to t+1}$ is the row-stochastic matrix with $p(s_{t+1} \mid s_t, a_t)\pi(a_{t+1} \mid s_{t+1})$ at positions $((s_t, a_t), (s_{t+1}, a_{t+1}))$. Reward redistribution from an optimal return decomposition requires

$$\boldsymbol{P}_{t \to t+1}\, \boldsymbol{q}_{t+1}^\pi \;=\; \boldsymbol{0} \tag{A212}$$

and, therefore,

$$\boldsymbol{P}_{t \to t+1}\, \boldsymbol{r}_{t+1} \;=\; \boldsymbol{0}\;, \tag{A213}$$

$$\mathrm{E}_\pi\left[\sum_t R_t\right] \;=\; \tilde{r}(s_T, a_T)\;.$$

The second equation avoids that $\boldsymbol{r}_{t+1} = \boldsymbol{0}$ for some $(t+1)$ if the return is not always zero and not all reward is given at time $t = 0$. If at least as many state-action pairs (s_t, a_t) exist as pairs (s_{t+1}, a_{t+1}), the equations can only be fulfilled if $\boldsymbol{P}_{t \to t+1}$ has not full rank. However, $\boldsymbol{P}_{t \to t+1}$ is given by the environment. Consequently, simultaneously ensuring Markov properties and ensuring zero future return is in general not possible.

The local variance of Q-values can still be large. We even introduced variance in the reward by the mean of previous Q-values $\mathrm{E}_{s_{t-1}, a_{t-1}}\left[\tilde{q}^\pi(s_{t-1}, a_{t-1}) \mid s_t, a_t\right]$.

Optimal Return Decomposition for Binary Reward. A special case is a reward that indicates success or failure by giving a reward of 1 or 0, respectively. The return is equal to the final reward R, which is a Bernoulli variable. For each state s or each state-action pair (s, a) the expected return can be considered as a Bernoulli variable with success probability $p_R(s)$ or $p_R(s, a)$. The value function is $v^\pi(s) = \mathrm{E}_\pi(G \mid s) = p_R(s)$ and the action-value is $q^\pi(s) = \mathrm{E}_\pi(G \mid s, a) = p_R(s, a)$ which is in both cases the expectation of success. In this case, the optimal return decomposition tracks the success probability

$$R_1 \;=\; h_0 \;=\; h(\Delta(s_{-1}, a_{-1}, s_0, a_0)) \;=\; \tilde{q}^\pi(s_0, a_0) \;=\; p_R(s_0, a_0) \tag{A214}$$

$$\begin{aligned} R_{t+1} \;&=\; h_t \;=\; h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \tilde{q}^\pi(s_{t-1}, a_{t-1}) \\ &=\; p_R(s_t, a_t) \;-\; p_R(s_{t-1}, a_{t-1}) \;\text{ for } 0 < t \leqslant T \end{aligned} \tag{A215}$$

$$R_{T+2} \;=\; \tilde{R}_{T+1} \;-\; \tilde{r}_{T+1} \;=\; R \;-\; p_R(s_T, a_T)\;. \tag{A216}$$

The redistributed reward is the change in the success probability. A good action increases the success probability and obtains a positive reward while a bad action reduces the success probability and obtains a negative reward.

Optimal Return Decomposition reduces the MDP to a Stochastic Contextual Bandit Problem. The new MDP $\mathcal{P}$ has a redistributed reward with random variable R_t at time t distributed according to $p(r \mid s_t, a_t)$. Theorem A5 states

$$q^\pi(s_t, a_t) \;=\; r(s_t, a_t) \;. \tag{A217}$$

This equation looks like a contextual bandit problem, where $r(s_t, a_t)$ is an estimate of the mean reward for action a_t for state or context s_t. Contextual bandits [62, p. 208] are characterized by a conditionally σ-subgaussian noise (Def. 5.1 [62, p. 68]). We define the zero mean noise variable η by

$$\eta_t \;=\; \eta(s_t, a_t) \;=\; R_t \;-\; r(s_t, a_t) \;, \tag{A218}$$

where we assume that η_t is a conditionally σ-subgaussian noise variable. Therefore, η is distributed according to $p(r - r(s_t, a_t) \mid s_t, a_t)$ and fulfills

$$\mathrm{E}\left[\eta(s_t, a_t)\right] \;=\; 0 \;, \tag{A219}$$

$$\mathrm{E}\left[\exp(\lambda \eta(s_t, a_t)\right] \;\leqslant\; \exp(\lambda^2 \sigma^2 / 2) \;. \tag{A220}$$

Subgaussian random variables have tails that decay almost as fast as a Gaussian. If the reward r is bounded by $|r| < B$, then η is bounded by $|\eta| < B$ and, therefore, a B-subgaussian. For binary rewards it is of interest that a Bernoulli variable is 0.5-subgaussian [62, p. 71]. In summary, an optimal return decomposition reduces the MDP to a stochastic contextual bandit problem.

Relation to "Backpropagation through a Model". The relation of reward redistribution if applied to policy gradients and "Backpropagation through a Model" is discussed here. For a delayed reward that is only received at the end of an episode, we decompose the return $\tilde{r}_{T+1}$ into

$$g(\Delta_{0:T}) \;=\; \tilde{r}_{T+1} \;=\; \sum_{t=0}^{T} h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t)) \;. \tag{A221}$$

The policy gradient for the reward redistribution from an optimal return decomposition is

$$\mathrm{E}_\pi \left[\nabla_\theta \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \; h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t))\right] \;. \tag{A222}$$

Summing up the gradient for one episode, the gradient becomes

$$\begin{aligned} &\mathrm{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi(a_t \mid s_t; \boldsymbol{\theta}) \; h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t))\right] \\ &= \mathrm{E}_\pi \left[\boldsymbol{J}_\theta(\log \pi(\boldsymbol{a} \mid \boldsymbol{s}; \boldsymbol{\theta})) \; \boldsymbol{h}(\Delta(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{s}, \boldsymbol{a}))\right] \;, \end{aligned} \tag{A223}$$

where $\boldsymbol{a}' = (a_{-1}, a_0, a_1, \ldots, a_{T-1})$ and $\boldsymbol{a} = (a_0, a_1, \ldots, a_T)$ are the sequences of actions, $\boldsymbol{s}' = (s_{-1}, s_0, s_1, \ldots, s_{T-1})$ and $\boldsymbol{s} = (s_0, s_1, \ldots, s_T)$ are the sequences of states, $\boldsymbol{J}_\theta(\log \pi)$ is the Jacobian of the log-probability of the state sequence with respect to the parameter vector $\boldsymbol{\theta}$, and $\boldsymbol{h}(\Delta(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{s}, \boldsymbol{a}))$ is the vector with entries $h(\Delta(s_{t-1}, a_{t-1}, s_t, a_t))$.
An alternative approach via sensitivity analysis is "Backpropagation through a Model", where $g(\Delta_{0:T})$ is maximized, that is, the return is maximized. Continuous actions are directly fed into g while probabilistic actions are sampled before entering g. Analog to gradients used for Restricted Boltzmann Machines, for probabilistic actions the log-likelihood of the actions is used to construct a gradient. The likelihood can also be formulated as the cross-entropy between the sampled actions and the action probability. The gradient for "Backpropagation through a Model" is

$$\mathrm{E}_\pi \left[\boldsymbol{J}_\theta(\log \pi(\boldsymbol{a} \mid \boldsymbol{s}; \boldsymbol{\theta})) \; \nabla_a g(\Delta_{0:T})\right] \;, \tag{A224}$$

where $\nabla_a g(\Delta_{0:T})$ is the gradient of g with respect to the action sequence $\boldsymbol{a}$.
If for "Backpropagation through a Model" the model gradient with respect to actions is replaced by the vector of contributions of actions in the model, then we obtain redistribution applied to policy gradients.

A3 Markov Decision Processes with Undiscounted Rewards

We focus on Markov Decision Processes (MDPs) with undiscounted rewards, since the relevance but also the problems of a delayed reward can be considerably decreased by discounting it. Using discounted rewards both the bias correction in TD as well as the variance of MC are greatly reduced. The correction amount decreases exponentially with the delay steps, and also the variance contribution to one state decreases exponentially with the delay of the reward.
MDPs with undiscounted rewards are either finite horizon or process absorbing states without reward. The former can always be described by the latter.

A3.1 Properties of the Bellman Operator in MDPs with Undiscounted Rewards

At each time t the environment is in some state $s = s_t \in \mathcal{S}$. The agent takes an action $a = a_t \in \mathcal{A}$ according to policy π, 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:

$$\mathrm{T}^\pi [q] (s, a) \;=\; \sum_{s', r} p(s', r \mid s, a) \left[r \;+\; \sum_{a'} \pi(a' \mid s') \, q(s', a') \right] . \tag{A225}$$

We often rewrite the operator as

$$\mathrm{T}^\pi [q] (s, a) \;=\; r(s, a) \;+\; \mathrm{E}_{s', a'} [q(s', a')] \;, \tag{A226}$$

where

$$r(s, a) \;=\; \sum_r r \, p(r \mid s, a) \;, \tag{A227}$$

$$\mathrm{E}_{s', a'} [q(s', a')] \;=\; \sum_{s'} p(s' \mid s, a) \sum_{a'} \pi(a' \mid s') \, q(s', a') \;. \tag{A228}$$

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

$$\mathrm{T}^\pi [q] (s, a) \;=\; g(s, a) \;+\; \mathrm{E}_{s', a'} [q(s', a')] \;. \tag{A229}$$

In the following we show properties for this general formulation.

A3.1.1 Monotonically Increasing and Continuous

We assume the general formulation Eq. (A229) of the Bellman operator. Proposition 2.1 on pages 22-23 in Bertsekas and Tsitsiklis, 1996, [10] shows that a fixed point q^π of the Bellman operator exists and that for every q:

$$q^\pi \;=\; \mathrm{T}^\pi [q^\pi] \tag{A230}$$

$$q^\pi \;=\; \lim_{k \to \infty} (\mathrm{T}^\pi)^k \, q \;. \tag{A231}$$

The fixed point equation

$$q^\pi \;=\; \mathrm{T}^\pi [q^\pi] \tag{A232}$$

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, [124]. This form of the Poisson equation describes the Dirichlet boundary value problem. The Poisson equation is

$$q^\pi(s, a) \;+\; \bar{g} \;=\; g(s, a) \;+\; \mathrm{E}_{s', a'} [q(s', a') \mid s, a] \;, \tag{A233}$$

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

$$\bar{g} \;=\; \lim_{T \to \infty} \frac{1}{T + 1} \sum_{t=0}^{T} g(s_t, a_t) \;. \tag{A234}$$

We assume $\bar{g} = 0$ since after some time the agent does no longer receive reward in MDPs with finite time horizon or in MDPs with absorbing states that have zero reward.
T^{π} is *monotonically increasing* in its arguments [10]. For q_1 and q_2 with the component-wise condition $q_1 \geqslant q_2$, we have

$$\begin{aligned} & \mathrm{T}^{\pi}\left[q_1\right](s,a) \; - \; \mathrm{T}^{\pi}\left[q_2\right](s,a) \\ & = \; (g(s,a) \; + \; \mathrm{E}_{s',a'}\left[q_1(s',a')\right]) \; - \; (g(s,a) \; + \; \mathrm{E}_{s',a'}\left[q_2(s',a')\right]) \\ & = \; \mathrm{E}_{s',a'}\left[q_1(s',a') \; - \; q_2(s',a')\right] \; \geqslant \; 0 \; , \end{aligned} \tag{A235}$$

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

$$\|q_1 \; - \; q_2\|_{\infty} \; = \; \max_{s,a} |q_1(s,a) \; - \; q_2(s,a)| \; . \tag{A236}$$

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

$$\begin{aligned} & \|\mathrm{T}^{\pi}\left[q_1\right] \; - \; \mathrm{T}^{\pi}\left[q_2\right]\|_{\infty} \; = \; \max_{s,a} |\mathrm{T}[q_1](s,a) \; - \; \mathrm{T}[q_2](s,a)| \\ & = \; \max_{s,a} \left| \left[g(s,a) \; + \; \sum_{s'} p(s' \mid s,a) \; \sum_{a'} \pi(a' \mid s') \, q_1(s',a') \right] \; - \right. \\ & \quad \left. \left[g(s,a) \; + \; \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s') \, q_2(s',a') \right] \right| \\ & = \; \max_{s,a} \left| \sum_{s'} p(s' \mid s,a) \; \sum_{a'} \pi(a' \mid s') \; (q_1(s',a') \; - \; q_2(s',a')) \right| \\ & \leqslant \; \max_{s,a} \sum_{s'} p(s' \mid s,a) \; \sum_{a'} \pi(a' \mid s') \; |q_1(s',a') \; - \; q_2(s',a')| \\ & \leqslant \; \max_{s',a'} |q_1(s',a') \; - \; q_2(s',a')| \; = \; \|q_1 \; - \; q_2\|_{\infty} \; . \end{aligned} \tag{A237}$$

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.

A3.1.2 Contraction for Undiscounted Finite Horizon

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

$$\|q_1 \; - \; q_2\|_{\infty,t} \; = \; \max_{t=0}^{T} \eta^{T-t+1} \max_{s_t,a} |q_1(s_t,a) \; - \; q_2(s_t,a)| \; . \tag{A238}$$

T^{π} is a *contraction mapping* for q_1 and q_2 [10]:

$$\|\mathrm{T}^{\pi}[q_1] - \mathrm{T}^{\pi}[q_2]\|_{\infty,t} = \max_{t=0}^{T} \eta^{T-t+1} \max_{s_t,a} |\mathrm{T}[q_1](s_t,a) - \mathrm{T}[q_2](s_t,a)| \tag{A239}$$

$$= \max_{t=0}^{T} \eta^{T-t+1} \max_{s_t,a} \left| \left[g(s_t,a) + \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, q_1(s_{t+1},a') \right] - \left[g(s_t,a) + \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, q_2(s_{t+1},a') \right] \right|$$

$$= \max_{t=0}^{T} \eta^{T-t+1} \max_{s_t,a} \left| \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, [q_1(s_{t+1},a') - q_2(s_{t+1},a')] \right|$$

$$\leqslant \max_{t=0}^{T} \eta^{T-t+1} \max_{s_t,a} \sum_{s_{t+1}} p(s_{t+1} \mid s_t,a) \sum_{a'} \pi(a' \mid s') \, |q_1(s_{t+1},a') - q_2(s_{t+1},a')|$$

$$\leqslant \max_{t=0}^{T} \eta^{T-t+1} \max_{s_{t+1},a'} |q_1(s_{t+1},a') - q_2(s_{t+1},a')|$$

$$\leqslant \max_{t=0}^{T} \eta \, \eta^{T-(t+1)+1} \max_{s_{t+1},a'} |q_1(s_{t+1},a') - q_2(s_{t+1},a')|$$

$$= \eta \max_{t=1}^{T+1} \eta^{T-t+1} \max_{s_t,a'} |q_1(s_t,a') - q_2(s_t,a')|$$

$$= \eta \max_{t=0}^{T} \eta^{T-t+1} \max_{s_t,a'} |q_1(s_t,a') - q_2(s_t,a')|$$

$$= \eta \, \|q_1 - q_2\|_{\infty,t} \, .$$

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

$$q^{\pi} = \mathrm{T}^{T}[q] \, , \tag{A240}$$

which is correct for additionally introduced states at time $t = T+1$ since they are zero. Then, in the next iteration Q-values of states at time $t = T$ are correct. After iteration i, Q-values of states at time $t = T - i + 1$ are correct. This iteration is called the "backward induction algorithm" [86, 87]. If we perform this iteration for a policy π instead of the optimal policy, then this procedure is called "policy evaluation algorithm" [86, 87].

A3.1.3 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.
The fact 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 [119]. Also Proposition 1 in Bertsekas and Tsitsiklis, 1991, [9], Theorems 3 and 4(b) & 4(c) in Tsitsiklis, 1994, [120], and Proposition 2.2 on pages 23-24 in Bertsekas and Tsitsiklis, 1996, [10] have proved the same fact.

A3.1.4 Fixed Point of Contraction is Continuous wrt Parameters

The mean q^{π} and variance 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 [54]:

Theorem A7 (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, [21]. Corollary 4.2 in Feinstein, 2016, generalized the theorem to set valued operators, that is, these operators may have more than one fixed point [19] (see also [57]). 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 [54]:

Theorem A8 (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, [21] and Theorem 2 in Kwiecinski, 1992, [60]. The mappings $F(p, .)$ can have different Lipschitz constants.

A3.1.5 t-fold Composition of the Operator

We define the Bellman operator as

$$\begin{aligned} \mathrm{T}^\pi [q](s,a) &= g(s,a) + \sum_{s'} p(s' \mid s,a) \sum_{a'} \pi(a' \mid s')\, q(s',a') \\ &= g(s,a) + \boldsymbol{q}^T \boldsymbol{p}(s,a)\,, \end{aligned} \tag{A241}$$

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, [124]. This form of the Poisson equation describes the Dirichlet boundary value problem. The *Bellman equation* or *Poisson equation* is

$$\mathrm{T}^\pi [\boldsymbol{q}] = \boldsymbol{g} + \boldsymbol{P}\,\boldsymbol{q}\,, \tag{A242}$$

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

$$\boldsymbol{q}^\pi + \bar{g}\mathbf{1} = \boldsymbol{g} + \boldsymbol{P}\,\boldsymbol{q}\,, \tag{A243}$$

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:

$$\bar{g} = \lim_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{T} g(s_t, a_t)\,. \tag{A244}$$

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 or equal 1. Only the eigenvector to the eigenvalue 1 has purely positive real components.

Equation 7 of Bertsekas and Tsitsiklis, 1991, [9] states

$$(\mathrm{T}^\pi)^t [\boldsymbol{q}] = \sum_{k=0}^{t-1} \boldsymbol{P}^k\,\boldsymbol{g} + \boldsymbol{P}^t\,\boldsymbol{q}\,. \tag{A245}$$

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

$$\lim_{k\to\infty} \boldsymbol{P}^k = \mathbf{1}\,\boldsymbol{p}^T \tag{A246}$$

$$\lim_{k\to\infty} \boldsymbol{p}_0^T \boldsymbol{P}^k = \boldsymbol{p}^T \tag{A247}$$

then

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \boldsymbol{P}^i = \mathbf{1}\,\boldsymbol{p}^T \tag{A248}$$

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k-1} \boldsymbol{p}_0^T \boldsymbol{P}^i = \boldsymbol{p}^T\,. \tag{A249}$$

A3.2 Q-value Transformations: Shaping Reward, Baseline, and Normalization

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

$$q^\pi(s,a) \;=\; \sum_{s',r} p(s',r \mid s,a) \;\left[r \;+\; \sum_{a'} \pi(a' \mid s')\; q^\pi(s',a') \right] . \tag{A250}$$

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

$$v_0 \;=\; \sum_{s_0} p(s_0)\; v(s_0) \,. \tag{A251}$$

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 $t \leqslant T$ and $\bar{R}_{T+1} = R_{T+1} - v_0$. Consequently, $\bar{q}(s_t,a_t) = q(s_t,a_t) - v_0$ for $t \leqslant T$.
The TD update rules are:

$$q(s_t,a_t) \;\longleftarrow\; q(s_t,a_t) \;+\; \alpha \left(r_t \;+\; \sum_a \pi(a \mid s_{t+1})\; q(s_{t+1},a) \;-\; q(s_t,a_t) \right) . \tag{A252}$$

The δ-errors are

$$\begin{aligned} & R_{t+1} \;+\; \sum_a \pi(a \mid s_{t+1})\; q(s_{t+1},a) \;-\; q(s_t,a_t) \\ &= \; R_{t+1} \;+\; \sum_a \pi(a \mid s_{t+1})\; (q(s_{t+1},a) - v_0) \;-\; (q(s_t,a_t) - v_0) \\ &= \; \bar{R}_{t+1} \;+\; \sum_a \pi(a \mid s_{t+1})\; \bar{q}(s_{t+1},a) \;-\; \bar{q}(s_t,a_t) \end{aligned} \tag{A253}$$

and for the last step

$$\begin{aligned} & R_{T+1} \;-\; q(s_T,a_T) \;=\; (R_{T+1} \;-\; v_0) \;-\; (q(s_T,a_T) \;-\; v_0) \\ & =\; \bar{R}_{T+1} \;-\; \bar{q}(s_T,a_T) \,. \end{aligned} \tag{A254}$$

If we set

$$\bar{q}(s_t,a_t) \;=\; \big\{ q(s_t,a_t) \;-\; v_0 \,, \quad \text{for } t \leqslant T \,. \tag{A255}$$

$$\bar{R}_t \;=\; \begin{cases} R_t \,, & \text{for } t \leqslant T \\ R_{T+1} \;-\; v_0 \,, & \text{for } t = T+1 \,, \end{cases} \tag{A256}$$

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 [80, 129]:

$$F(s_t,a_t,s_{t+1}) \;=\; \Phi(s_{t+1}) - \Phi(s_t) \,. \tag{A257}$$

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

$$\Phi(s_t) \;=\; \begin{cases} 0 \,, & \text{for } t \leqslant m \,, \\ -c \,, & \text{for } m+1 \leqslant t \leqslant k \,, \\ 0 \,, & \text{for } t > k \,, \end{cases} \tag{A258}$$

then the shaping reward is

$$F\big(s_t,a_t,s_{t+1}\big) \;=\; \begin{cases} 0 \,, & \text{for } t < m \,, \\ -c \,, & \text{for } t = m \,, \\ 0 \,, & \text{for } m+1 \leqslant t < k \,, \\ c \,, & \text{for } t = k \,, \\ 0 \,, & \text{for } t > k \,. \end{cases} \tag{A259}$$

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 is 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 [80, 129] 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

$$q^{\text{new}}(s_t, a_t) \;=\; q(s_t, a_t) \;-\; \Phi(s_t)\;. \tag{A260}$$

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 $1 \leqslant t \leqslant T$:

$$\Phi(s_t) \;=\; \max_a q(s_t, a)\;. \tag{A261}$$

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

$$\Phi(s_t) \;=\; \min_a q(s_t, a)\;. \tag{A262}$$

A3.3 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 (MDPs) preserve the optimal action-value and action sequence properties as well as the optimal policies of the model [27]. Theorem 7 and Corollary 9.1 in Givan et al. proved these properties [27] 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 [90]. Therefore, Ravindran and Barto have also shown that state-enriched MDPs preserve the optimal action-value and action sequence properties. Li et al. give an overview over state abstraction or state aggregation for MDPs, which covers state-enriched MDPs [65].

Definition A8. *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 the 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:*

$$\tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \;=\; p(f(\tilde{s}'), r \mid f(\tilde{s}), a)\;, \tag{A263}$$

$$\tilde{p}_0(\tilde{s}_0) \;=\; p_0(f(\tilde{s}_0))\;, \tag{A264}$$

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 $\tilde{p}_0(\tilde{s}_0, r) = \tilde{p}_0(\tilde{s}_0)$.
We proof the following theorem, even if it has been proved several times as mention above.

Theorem A9. *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'.

$$\begin{aligned}
\tilde{q}^{\tilde{\pi}}(\tilde{s}, a) \;&=\; \mathrm{E}_{\tilde{\pi}}\left[\tilde{G}_t \mid \tilde{s}_t = \tilde{s}, A_t = a\right] \;=\; \sum_{\tilde{s}', r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), g(\tilde{s}'), r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \qquad \text{(A265)} \\
&= \sum_{f(\tilde{s}'), G(r, \tilde{s}, a), r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), r} \tilde{p}(\tilde{s}', r \mid \tilde{s}, a) \left[r \;+\; \sum_{a'} \tilde{\pi}(a' \mid \tilde{s}')\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), r} p(f(\tilde{s}'), r \mid f(\tilde{s}), a) \left[r \;+\; \sum_{a'} \pi(a' \mid f(\tilde{s}'))\, \tilde{q}^{\tilde{\pi}}(\tilde{s}', a')\right] \\
&= \sum_{f(\tilde{s}'), r} p(f(\tilde{s}'), r \mid f(\tilde{s}), a) \left[r \;+\; \sum_{a'} \pi(a' \mid f(\tilde{s}'))\, q^{\pi}(f(\tilde{s}'), a')\right] \\
&= q^{\pi}(f(\tilde{s}), a)\;.
\end{aligned}$$

For the 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

$$\tilde{\pi}^*(\tilde{s}) \;=\; \operatorname*{argmax}_a \;\tilde{q}^*(\tilde{s}, a) \;=\; \operatorname*{argmax}_a \;q^*(f(\tilde{s}), a) \;=\; \pi^*(f(\tilde{s}))\;. \qquad \text{(A266)}$$

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

Theorem A10. *If a 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. □

A3.4 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

$$\mathrm{E}_{s', a'}\left[q^{\pi}(s', a') \mid s, a\right] = \sum_{s'} p(s' \mid s, a) \sum_{a'} \pi(a' \mid s')\, q^{\pi}(s', a')\;, \qquad \text{(A267)}$$

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 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

$$\begin{aligned}
\text{pdf} \qquad & \frac{n!}{x_1! \cdots x_k!}\, p_1^{x_1} \cdots p_k^{x_k}\;, && \text{(A268)} \\
\text{mean} \qquad & \mathrm{E}[X_i] \;=\; n\, p_i\;, && \text{(A269)} \\
\text{variance} \qquad & \mathrm{Var}[X_i] \;=\; n\, p_i\, (1 - p_i)\;, && \text{(A270)} \\
\text{covariance} \qquad & \mathrm{Cov}[X_i, X_j] \;=\; -\, n\, p_i\, p_j\;, \;\; (i \neq j)\;, && \text{(A271)}
\end{aligned}$$

where X_i is the random variable and x_i the actual count.
A linear combination of random variables has variance

$$\begin{aligned} \operatorname{Var}\left[\sum_{i=1}^{N} a_i\ X_i\right] &= \sum_{i,j=1}^{N} a_i\ a_j\ \operatorname{Cov}\left[X_i, X_j\right] \\ &= \sum_{i=1}^{N} a_i^2\ \operatorname{Var}\left[X_i\right]\ +\ \sum_{i \neq j} a_i\ a_j\ \operatorname{Cov}\left[X_i, X_j\right]\ . \end{aligned} \tag{A272}$$

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

$$\begin{aligned} \operatorname{Var}\left[X\right] &= \operatorname{Var}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\operatorname{Var}\left[X_i\right] = \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{\sigma^2}{n}\ . \end{aligned} \tag{A273}$$

When 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:

$$\begin{aligned} \operatorname{Var}\left[\bar{y}\right] &= \frac{\sigma_y^2}{n} = \frac{1}{n}\ \left(\sum_{i=1}^{N} a_i^2\ n_y\ p_i\ (1-p_i)\ -\ \sum_{i \neq j} a_i\ a_j\ n_y\ p_i\ p_j\right) \\ &= \frac{n_y}{n}\ \left(\sum_{i=1}^{N} a_i^2\ p_i\ (1-p_i)\ -\ \sum_{i \neq j} a_i\ a_j\ p_i\ p_j\right) \\ &= \frac{n_y}{n}\ \left(\sum_{i=1}^{N} a_i^2\ p_i\ -\ \sum_{(i,j)=(1,1)}^{(N,N)} a_i\ a_j\ p_i\ p_j\right) \\ &= \frac{n_y}{n}\ \left(\sum_{i=1}^{N} a_i^2\ p_i\ -\ \left(\sum_{i=1}^{N} a_i\ p_i\right)^2\right)\ . \end{aligned} \tag{A274}$$

A4 Backward Analysis

A4.1 Difference of Consecutive Predictions for Sequences

General Approach. The idea is to assess the information gain that is induced by an input at a particular time step. This information gain is used for predicting the target at sequence end by determining the change in prediction. The input to a recurrent neural network is the sequence $\boldsymbol{x} = (x_1, \ldots, x_d)$ with target y_d, which is only given at sequence end. The prefix sequence $\boldsymbol{x}_t$ of length $t \leqslant d$ is $\boldsymbol{x}_t = (x_1, \ldots, x_t)$. F predicts the target y_d at every time step t:

$$F(\boldsymbol{x}_t)\ =\ y_d\ . \tag{A275}$$

We can define the decomposition of F through contributions at different time steps

$$h_0\ =\ F(\boldsymbol{x}_0)\ , \tag{A276}$$

$$h_t\ =\ F(\boldsymbol{x}_t)\ -\ F(\boldsymbol{x}_{t-1})\ \text{ for } t > 0\ , \tag{A277}$$

where $F(\boldsymbol{x}_0)$ is a predefined constant. We have

$$F(\boldsymbol{x}_t)\ =\ \sum_{\tau=0}^{t} h_\tau\ . \tag{A278}$$

We assume a loss function for F that is minimal if $F \equiv F_{\min}$ predicts the expected y_d

$$F_{\min}(\boldsymbol{x}_t)\ =\ \mathrm{E}\left[y_d \mid \boldsymbol{x}_t\right]\ . \tag{A279}$$

Then

$$h_0 \;=\; \mathrm{E}\left[y_d\right] \;, \tag{A280}$$

$$h_t \;=\; \mathrm{E}\left[y_d \mid \boldsymbol{x}_t\right] \;-\; \mathrm{E}\left[y_d \mid \boldsymbol{x}_{t-1}\right] \quad \text{for } t > 0 \;. \tag{A281}$$

In this case, the contributions are the change in the expectation of the target that will be observed at sequence end. The contribution can be viewed as the information gain in time step t for predicting the target. If we cannot ensure that F predicts the target at every time step, then backward analysis has to use contribution analysis. For attributing the prediction of a deep network to its input features several contribution analysis methods have been proposed. We consider Input Zeroing, Integrated Gradients (IG), and Layer-Wise Relevance Propagation (LRP).

Linear Models and Coefficient of Determination. We consider linear models and the average gain of information about the reward at sequence end if we go one time step further in the input sequence. By adding a variable, that is, another sequence element, the mean squared error (MSE) decreases, which is the amount by which the expectation improves due to new information. But by what amount does the MSE decrease in average? Here, we consider linear models. For linear models we are interested in how much the coefficient of determination increases if we add another variable, that is, if we see another input.
We consider the feature vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)^T$ from which the target y (the reward at sequence end) has to be predicted. We assume to have n pairs $(\boldsymbol{x}_i, y_i), 1 \leqslant i \leqslant n$, as training set. The prediction or estimation of y_i from $\boldsymbol{x}_i$ is $\hat{y}_i$ with $\hat{y}_i = F(\boldsymbol{x}_i)$. The vector of all training labels is $\boldsymbol{y} = (y_1, \ldots, y_n)$ and the training feature matrix is $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. We define the mean squared error (MSE) as

$$\mathrm{mse}(\boldsymbol{y}, \boldsymbol{X}) \;=\; \frac{1}{n-1} \sum_{i=1}^{n} \left(\hat{y}_i \;-\; y_i\right)^2 \;. \tag{A282}$$

The *coefficient of determination* R^2 is equal to the correlation between the target y and its prediction $\hat{y}$. R^2 is given by:

$$R^2 \;=\; 1 \;-\; \frac{\frac{1}{n-1} \sum_{i=1}^{n} \left(\hat{y}_i \;-\; y_i\right)^2}{\frac{1}{n-1} \sum_{i=1}^{n} \left(y_i \;-\; \bar{y}\right)^2} \;=\; 1 \;-\; \frac{\mathrm{mse}(\boldsymbol{y}, \boldsymbol{X})}{s_y^2} \;. \tag{A283}$$

Therefore, R^2 is one minus the ratio of the mean squared error divided by the mean total sum of squares. R^2 is a strict monotonically decreasing function of the mean squared error.
We will give a breakdown of the factors that determine how much each variable adds to R^2 [91, chapter 10.6, p. 263]. The feature vector $\boldsymbol{x}$ is expanded by one additional feature z: $\boldsymbol{w} = (x_1, x_2, \ldots, x_k, z)^T = (\boldsymbol{x}^T, z)^T$. We want to know the increase in R^2 due to adding z. Therefore, we decompose $\boldsymbol{w}$ into $\boldsymbol{x}$ and z. The difference in coefficients of determination is the difference of the according MSEs divided by the empirical variance of y:

$$R^2_{y\boldsymbol{w}} \;-\; R^2_{y\boldsymbol{x}} \;=\; \frac{\mathrm{mse}(\boldsymbol{y}, \boldsymbol{W}) \;-\; \mathrm{mse}(\boldsymbol{y}, \boldsymbol{X})}{s_y^2} \;. \tag{A284}$$

We further need definitions:

- $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)^T$.
- $\boldsymbol{w} = (x_1, x_2, \ldots, x_k, z)^T = (\boldsymbol{x}^T, z)^T$.
- The sample covariance between y and x is $s_{yx} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})/(n-1)$, where $\bar{x} = \sum_{i=1}^{n} x_i/n$ and $\bar{y} = \sum_{i=1}^{n} y_i/n$ are the sample means. The variance of x is s_{xx} often written as s_x^2, the standard deviation squared: $s_x := \sqrt{s_{xx}}$.
- The correlation between y and x is $r_{yx} = s_{yx}/(s_x s_y)$.
- The covariance matrix $\boldsymbol{S}_{xx}$ of a vector $\boldsymbol{x}$ is the matrix with entries $[\boldsymbol{S}_{xx}]_{ij} = s_{x_i x_j}$.
- The covariance matrix $\boldsymbol{R}_{xx}$ of a vector $\boldsymbol{x}$ is the matrix with entries $[\boldsymbol{R}_{xx}]_{ij} = r_{x_i x_j}$.
- The diagonal matrix $\boldsymbol{D}_x = [\mathrm{diag}(\boldsymbol{S}_{xx})]^{1/2}$ has a ith diagonal entry $\sqrt{s_{x_i}}$ and is the diagonal matrix of standard deviations of the components of $\boldsymbol{x}$.
- R^2_{yw} is the squared multiple correlation between y and $\boldsymbol{w}$.

- R^2_{yx} is the squared multiple correlation between y and $\boldsymbol{x}$.
- $R^2_{zx} = \boldsymbol{s}^T_{zx}\boldsymbol{S}^{-1}_{xx}\boldsymbol{s}_{zx}/s^2_z = \boldsymbol{r}^T_{zx}\boldsymbol{R}^{-1}_{xx}\boldsymbol{r}_{zx}$ is the squared multiple correlation between z and $\boldsymbol{x}$.
- r_{yz} is the simple correlation between y and z: $r_{yz} = s_{yz}/(s_y s_z)$.
- $\boldsymbol{r}_{yx} = (r_{yx_1}, r_{yx_2}, \ldots, r_{yx_k})^T = s_y^{-1}\boldsymbol{D}_x^{-1}\boldsymbol{S}_{yx}$ is the vector of correlations between y and $\boldsymbol{x}$.
- $\boldsymbol{r}_{zx} = (r_{zx_1}, r_{zx_2}, \ldots, r_{zx_k})^T = s_z^{-1}\boldsymbol{D}_x^{-1}\boldsymbol{S}_{zx}$ is the vector of correlations between z and $\boldsymbol{x}$.
- $\hat{\boldsymbol{\beta}}^*_{zx} = \boldsymbol{R}^{-1}_{xx}\boldsymbol{r}_{zx}$ is the vector of standardized regression coefficients (beta weights) of z regressed on $\boldsymbol{x}$.
- The parameter vector is partitioned into the constant β_0 and $\boldsymbol{\beta}_1$ via $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_m)^T = (\beta_0, \boldsymbol{\beta}_1^T)^T$. We have for the maximum likelihood estimate

$$\hat{\beta}_0 \;=\; \bar{y} \;-\; \boldsymbol{s}^T_{yx}\boldsymbol{S}^{-1}_{xx}\bar{\boldsymbol{x}}\;, \tag{A285}$$

$$\hat{\boldsymbol{\beta}}_1 \;=\; \boldsymbol{S}^{-1}_{xx}\boldsymbol{s}_{yx}\;. \tag{A286}$$

  The offset $\hat{\beta}_0$ guarantees $\bar{\hat{y}} = \bar{y}$, therefore, $\boldsymbol{y}^T\bar{\boldsymbol{y}} = \hat{\boldsymbol{y}}^T\bar{\boldsymbol{y}}$, since $\bar{\boldsymbol{y}} = \bar{y}\mathbf{1}$:

$$\begin{aligned}\bar{\hat{y}} \;&=\; \frac{1}{n}\sum_{i=1}^n \hat{y}_i \;=\; \frac{1}{n}\sum_{i=1}^n \left(\hat{\beta}_0 \;+\; \hat{\boldsymbol{\beta}}_1^T\boldsymbol{x}_i\right) \\ &=\; \bar{y} \;-\; \boldsymbol{s}^T_{yx}\boldsymbol{S}^{-1}_{xx}\bar{\boldsymbol{x}} \;+\; \frac{1}{n}\sum_{i=1}^n \hat{\boldsymbol{\beta}}_1^T\boldsymbol{x}_i \\ &=\; \bar{y} \;-\; \boldsymbol{s}^T_{yx}\boldsymbol{S}^{-1}_{xx}\bar{\boldsymbol{x}} \;+\; \boldsymbol{s}^T_{yx}\boldsymbol{S}^{-1}_{xx}\bar{\boldsymbol{x}} \\ &=\; \bar{y}\;.\end{aligned} \tag{A287}$$

- The vector of *standardized coefficients* $\hat{\boldsymbol{\beta}}_1^*$ are

$$\hat{\boldsymbol{\beta}}_1^* \;=\; \frac{1}{s_y}\,\boldsymbol{D}_x\,\hat{\boldsymbol{\beta}}_1 \;=\; \boldsymbol{R}^{-1}_{xx}\,\boldsymbol{r}_{yx}\;. \tag{A288}$$

The next theorem is Theorem 10.6 in Rencher and Schaalje [91] and gives a breakdown of the factors that determine how much each variable adds to R^2 [91, Chapter 10.6, p. 263].

Theorem 1 (Rencher Theorem 10.6)**.** *The increase in R^2 due to z can be expressed as*

$$R^2_{y\boldsymbol{w}} \;-\; R^2_{y\boldsymbol{x}} \;=\; \frac{(\hat{r}_{yz} \;-\; r_{yz})^2}{1 \;-\; R^2_{zx}}\;, \tag{A289}$$

where $\hat{r}_{yz} = (\hat{\boldsymbol{\beta}}^*_{zx})^T\boldsymbol{r}_{yx}$ is a "predicted" value of r_{yz} based on the relationship of z to the $\boldsymbol{x}$'s.

The following equality shows that $\hat{r}_{yz} = (\hat{\boldsymbol{\beta}}^*_{zx})^T\boldsymbol{r}_{yx}$ is indeed a prediction of r_{yz}:

$$\begin{aligned}\left(\hat{\boldsymbol{\beta}}^*_{zx}\right)^T\boldsymbol{r}_{yx} \;&=\; \frac{1}{s_z}\,\boldsymbol{D}_x\,\hat{\boldsymbol{\beta}}^T_{zx}\frac{1}{s_y}\,\boldsymbol{D}_x^{-1}\boldsymbol{s}_{yx} \\ &=\; \frac{1}{s_z\;s_y}\,\hat{\boldsymbol{\beta}}^T_{zx}\frac{1}{n-1}\;\sum_{i=1}^n(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(y_i - \bar{y}) \\ &=\; \frac{1}{s_z\;s_y}\;\frac{1}{n-1}\;\sum_{i=1}^n(\hat{\boldsymbol{\beta}}^T_{zx}\boldsymbol{x}_i - \hat{\boldsymbol{\beta}}^T_{zx}\bar{\boldsymbol{x}})(y_i - \bar{y}) \\ &=\; \frac{1}{s_z\;s_y}\;\frac{1}{n-1}\;\sum_{i=1}^n(\hat{z}_i - \bar{\hat{z}})(y_i - \bar{y}) \\ &=\; \frac{1}{s_z\;s_y}\;\hat{s}_{yz} \;=\; \hat{r}_{yz}\;.\end{aligned} \tag{A290}$$

If z is orthogonal to $\boldsymbol{x}$ (i.e., if $\boldsymbol{r}_{zx} = \mathbf{0}$), then $\hat{\boldsymbol{\beta}}^*_{zx} = \mathbf{0}$, which implies that $\hat{r}_{yz} = 0$ and $R^2_{zx} = 0$. In this case, Eq. (A289) can be written as

$$R^2_{y\boldsymbol{w}} \; - \; R^2_{y\boldsymbol{x}} \; = \; r^2_{yz} \; . \tag{A291}$$

Consequently, if all x_i are independent from each other, then

$$R^2_{y\boldsymbol{x}} \; = \; \sum_{j=1}^{k} r^2_{yx_j} \; . \tag{A292}$$

The contribution of z to R^2 can either be less than or greater than r_{yz}. If the correlation r_{yz} can be predicted from $\boldsymbol{x}$, then $\hat{r}_{yz}$ is close to r_{yz} and, therefore, z has contributes less to R^2 than r^2_{yz}.
Next, we compute the contribution of z to R^2 explicitly. The correlation between y and z is

$$r_{yz} \; = \; \frac{1}{s_z \; s_y} \; \frac{1}{n-1} \; \sum_{i=1}^{n} (z_i - \bar{z})(y_i - \bar{y}) \; = \; \frac{1}{s_z \; s_y} \; s_{yz} \; . \tag{A293}$$

We assume that $\bar{z} = \bar{\hat{z}}$. We want to express the information gain using the mean squared error (MSE) $1/(n-1) \sum_{i=1}^{n} (\hat{z}_i - z_i)^2$. We define the error $e_i := \hat{z}_i - z_i$ at sample i with $\bar{e} = \bar{\hat{z}} - \bar{z} = 0$. Therefore, the MSE is equal to the empirical variance $s_e^2 = 1/(n-1) \sum_{i=1}^{n} e_i^2$. The correlation r_{ey} between the target y and the error e is

$$r_{ey} \; = \; \frac{1}{s_y \; s_e} \; \frac{1}{n-1} \; \sum_{i=1}^{n} (e_i \; - \; \bar{e}) \; (y_i - \bar{y}) \; . \tag{A294}$$

Using Eq. (A290) and Eq. (A293), we can express the difference between the estimate $\hat{r}_{yz}$ and the true correlation r_{yz} by:

$$\begin{aligned} \hat{r}_{yz} \; - \; r_{yz} \; &= \; \frac{1}{s_z \; s_y} \; \frac{1}{n-1} \; \sum_{i=1}^{n} (\hat{z}_i - \bar{\hat{z}})(y_i - \bar{y}) \; - \; \frac{1}{s_z \; s_y} \; \frac{1}{n-1} \; \sum_{i=1}^{n} (z_i - \bar{z})(y_i - \bar{y}) \\ &= \; \frac{1}{s_z \; s_y} \; \frac{1}{n-1} \; \sum_{i=1}^{n} (\hat{z}_i - z_i)(y_i - \bar{y}) \; . \end{aligned} \tag{A295}$$

The information gain can now be expressed by the correlation r_{ey} between the target y and the error e:

$$\begin{aligned} R^2_{y\boldsymbol{w}} \; - \; R^2_{y\boldsymbol{x}} \; &= \; \frac{(\hat{r}_{yz} \; - \; r_{yz})^2}{1 \; - \; R^2_{zx}} \; = \; \frac{\frac{1}{s_z^2 \; s_y^2} \; \frac{1}{(n-1)^2} \; \left(\sum_{i=1}^{n} (\hat{z}_i - z_i)(y_i - \bar{y}) \right)^2}{\frac{1}{n-1} \; \sum_{i=1}^{n} (\hat{z}_i - z_i)^2 \; \frac{1}{n-1} \; \sum_{i=1}^{n} (z_i - \bar{z})^2} \\ &= \; \frac{\frac{1}{s_y^2} \; \frac{1}{(n-1)^2} \; \left(\sum_{i=1}^{n} (\hat{z}_i - z_i)(y_i - \bar{y}) \right)^2}{\frac{1}{n-1} \; \sum_{i=1}^{n} \left(\hat{z}_i \; - \; z_i \right)^2} \\ &= \; r^2_{ey} \; . \end{aligned} \tag{A296}$$

The information gain is the squared correlation r^2_{ey} between the target y and the error e. **The information gain is the information in z about y, which is not contained in $\boldsymbol{x}$.**

A4.2 Input Zeroing

The simplest contribution analysis method is Input Zeroing, where just an input is set to zero to determine its contribution to the output. Input Zeroing sets a particular input x_i to zero and then computes the network's output. For the original input $\boldsymbol{x} = (x_1, \ldots, x_d)$ and the input with $x_i = 0$, i.e. $\tilde{\boldsymbol{x}}_i = (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_d)$, 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 $\frac{1}{d} \sum_{i=1}^{d} F(\tilde{\boldsymbol{x}}_i)$:

$$F(\boldsymbol{x}) \; - \; \frac{1}{d} \; \sum_{i=1}^{d} F(\tilde{\boldsymbol{x}}_i) \; = \; \frac{1}{d} \; \sum_{i=1}^{d} \Delta x_i \; . \tag{A297}$$

The problem is that the $F(\tilde{\boldsymbol{x}}_i)$ have to be computed d-times, that is, for each input component zeroed out.
Input Zeroing does not recognize redundant inputs, i.e. each one of the inputs is sufficient to produce the output but if all inputs are missing at the same time then the output changes. In contrast, Integrated Gradients (IG) and Layer-Wise Relevance Propagation (LRP) detect the relevance of an input even if it is redundant.

A4.3 Integrated Gradients

Integrated gradients is a recently introduced method [112]. Integrated gradients decomposes the difference $F(\boldsymbol{x}) - F(\tilde{\boldsymbol{x}})$ between the network output $F(\boldsymbol{x})$ and a baseline $F(\tilde{\boldsymbol{x}})$:

$$\begin{aligned} F(\boldsymbol{x}) \ - \ F(\tilde{\boldsymbol{x}}) \ &= \ \sum_{i=1}^{d} (x_i \ - \ \tilde{x}_i) \ \int_{t=0}^{1} \frac{\partial F}{\partial x_i}(\tilde{\boldsymbol{x}} + t(\boldsymbol{x} - \tilde{\boldsymbol{x}})) \, \mathrm{d}t \\ &\approx \ \sum_{i=1}^{d} (x_i \ - \ \tilde{x}_i) \ \frac{1}{m} \ \sum_{k=1}^{m} \frac{\partial F}{\partial x_i}(\tilde{\boldsymbol{x}} + (k/m)(\boldsymbol{x} - \tilde{\boldsymbol{x}})) \ . \end{aligned} \tag{A298}$$

In contrast to previous approaches, 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

$$\begin{cases} g : [0,1] \to \mathbb{R} \\ g(t) = F(\boldsymbol{x} + t\boldsymbol{h}) \ . \end{cases} \tag{A299}$$

Consequently, we have

$$F(\boldsymbol{x} + \boldsymbol{h}) - F(\boldsymbol{x}) \ = \ g(1) - g(0) \ = \ \int_0^1 g'(t) \, \mathrm{d}t \tag{A300}$$

$$= \ \int_0^1 \left(\sum_{i=1}^{d} \frac{\partial F}{\partial x_i}(\boldsymbol{x} + t\boldsymbol{h}) \ h_i \right) \mathrm{d}t \ = \ \sum_{i=1}^{d} \left(\int_0^1 \frac{\partial F}{\partial x_i}(\boldsymbol{x} + t\boldsymbol{h}) \, \mathrm{d}t \right) \ h_i \ . \tag{A301}$$

For the final reward decomposition, we obtain

$$F(\boldsymbol{x}) \ = \ \sum_{i=1}^{d} \left((x_i \ - \ \tilde{x}_i) \ \frac{1}{m} \ \sum_{k=1}^{m} \frac{\partial F}{\partial x_i}(\tilde{\boldsymbol{x}} + (k/m)(\boldsymbol{x} - \tilde{\boldsymbol{x}})) \ + \ \frac{1}{d} \, F(\tilde{\boldsymbol{x}}) \right) \ . \tag{A302}$$

A4.4 Layer-Wise Relevance Propagation

Layer-Wise Relevance Propagation (LRP) [3] has been introduced to interpret machine learning models. LRP is an extension of the contribution-propagation algorithm [61] based on the contribution approach [85]. Recently "excitation backprop" was proposed [133], 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 is important to the network output if it has been important to its parents, and its parents have been 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 that a node obtains from its parents will be 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 transportation in general form. LRP has been used for deep neural networks (DNN) [76] 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:

$$\mathrm{net}_i \ = \ \sum_l w_{il} \, x_l \ , \tag{A303}$$

$$x_i \ = \ f_i(\mathrm{net}_i) \ = \ g(\mathrm{net}_i + b_i) \ . \tag{A304}$$

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

$$R_{i \leftarrow k} \ = \ \frac{z_{ik}}{\sum_l z_{lk}} \, R_k \ . \tag{A305}$$

The relative contributions z_{ik} are previously defined as [3, 76, 1]:

$$z_{ik} \;=\; w_{ik}\, x_k \;. \tag{A306}$$

Here, z_{ik} is the contribution of x_k to the netinput value net_i. If neuron k is removed from the network, then z_{ik} will be 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:

$$R_i \;=\; \sum_k R_{i \leftarrow k} \;. \tag{A307}$$

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

$$R_k \;=\; \sum_i R_{i \leftarrow k} \;. \tag{A308}$$

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:

$$\sum_k R_k \;=\; \sum_k \sum_i R_{i \leftarrow k} \;=\; \sum_i \sum_k R_{i \leftarrow k} \;=\; \sum_i R_i \;. \tag{A309}$$

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:

$$\sum_i R_i \;=\; R \;=\; g(\boldsymbol{x}) \;. \tag{A310}$$

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

A4.4.1 New Variants of LRP

An alternative definition of z_{ik} is

$$z_{ik} \;=\; w_{ik}\, (x_k \;-\; \bar{x}_k) \;, \tag{A311}$$

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

$$z_{ik} \;=\; f_i(\mathrm{net}_i) \;-\; f_i(\mathrm{net}_i \;-\; w_{ik}\, x_k) \;. \tag{A312}$$

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

$$z_{ik} \;=\; w_{ik}\, (x_k \;-\; x_{\min}) \;, \tag{A313}$$

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 [133]. The minimal value is considered as default off-set, which can be included into the bias.

A4.4.2 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

$$R_z \;=\; R_{x_1 \leftarrow z} \;+\; R_{x_2 \leftarrow z} \;. \tag{A314}$$

Alternative 1:

$$R_{x_1 \leftarrow z} \;=\; 0.5\, R_z \tag{A315}$$
$$R_{x_2 \leftarrow z} \;=\; 0.5\, R_z\;. \tag{A316}$$

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

$$\left.\frac{\partial z}{\partial x_1}\right|_{(a,a)} (x_1 - a) \;=\; (x_1 - a)\, a\;, \tag{A317}$$
$$\left.\frac{\partial z}{\partial x_2}\right|_{(a,a)} (x_2 - a) \;=\; a\,(x_2 - a)\;. \tag{A318}$$

We compute the relative contributions:

$$\frac{(x_1 - a)\, a}{(x_1 - a)\, a \;+\; a\,(x_2 - a)} \;=\; \frac{x_1 - a}{(x_1 \;+\; x_2 \;-\; 2\, a)}\;, \tag{A319}$$
$$\frac{(x_2 - a)\, a}{(x_1 - a)\, a \;+\; a\,(x_2 - a)} \;=\; \frac{x_2 - a}{(x_1 \;+\; x_2 \;-\; 2\, a)}\;. \tag{A320}$$

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 $[0, 1]$:

$$R_{x_1 \leftarrow z} \;=\; \frac{\frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}}{\frac{x_1 - x_{\min}}{x_{\max} - x_{\min}} \;+\; \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}}\, R_z \tag{A321}$$
$$R_{x_2 \leftarrow z} \;=\; \frac{\frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}}{\frac{x_1 - x_{\min}}{x_{\max} - x_{\min}} \;+\; \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}}\, R_z\;. \tag{A322}$$

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:

$$R_{x_1 \leftarrow z} \;=\; \frac{\ln\left(1 - \frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}\right)}{\ln\left(1 - \frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}\right) \;+\; \ln\left(1 - \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}\right)}\, R_z \tag{A323}$$
$$R_{x_2 \leftarrow z} \;=\; \frac{\ln\left(1 - \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}\right)}{\ln\left(1 - \frac{x_1 - x_{\min}}{x_{\max} - x_{\min}}\right) \;+\; \ln\left(1 - \frac{x_2 - x_{\min}}{x_{\max} - x_{\min}}\right)}\, R_z\;. \tag{A324}$$

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.

A5 Long Short-Term Memory (LSTM)

A5.1 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 [30], generation of writings [31], language modeling and identification [28, 132], automatic language translation [113], speech recognition [99, 22] analysis of audio data [70], analysis, annotation, and description of video data [16, 123, 110]. 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.

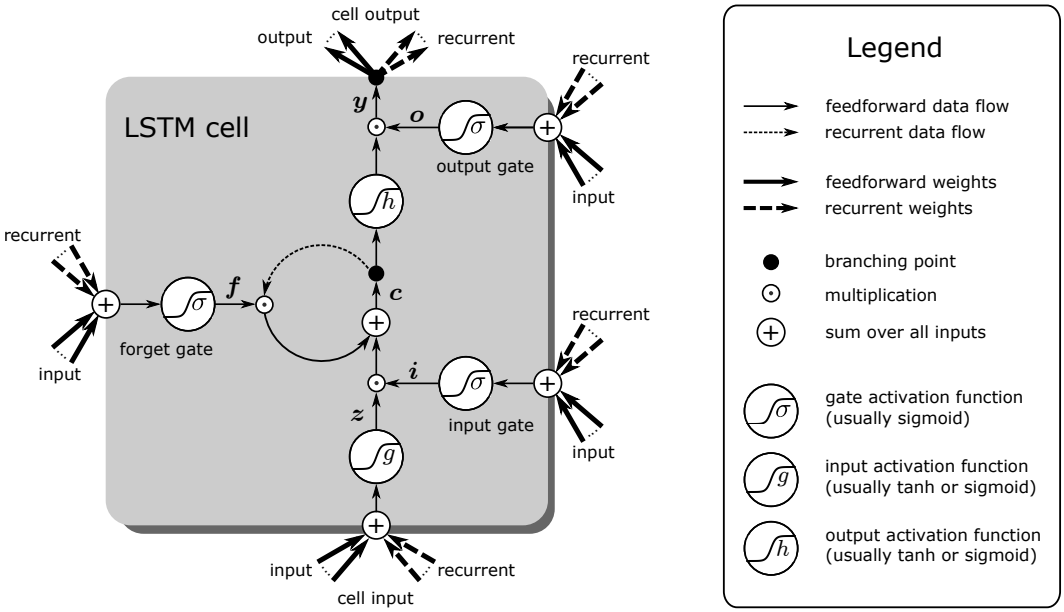


Figure A3: LSTM memory cell without peepholes. $\boldsymbol{z}$ is the vector of cell input activations, $\boldsymbol{i}$ is the vector of input gate activations, $\boldsymbol{f}$ is the vector of forget gate activations, $\boldsymbol{c}$ is the vector of memory cell states, $\boldsymbol{o}$ is the vector 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 an 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.

A5.2 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 [32, 102] contains forget gates [24, 25] and peephole connections [23]. 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 [33]. While LSTM networks are highly successful in various applications, the central memory cell architecture

was not modified since 2000 [102]. 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 vectors. Without peepholes, the LSTM memory cell forward pass rules are (see Figure A3):

$$\boldsymbol{z}^t = g\left(\boldsymbol{W_z}\,\boldsymbol{x}^t + \boldsymbol{b_z}\right) \qquad \text{cell input} \tag{A325}$$

$$\boldsymbol{i}^t = \sigma\left(\boldsymbol{W_i}\,\boldsymbol{x}^t + \boldsymbol{b_i}\right) \qquad \text{input gate} \tag{A326}$$

$$\boldsymbol{f}^t = \sigma\left(\boldsymbol{W_f}\,\boldsymbol{x}^t + \boldsymbol{b_f}\right) \qquad \text{forget gate} \tag{A327}$$

$$\boldsymbol{c}^t = \boldsymbol{i}^t \odot \boldsymbol{z}^t + \boldsymbol{f}^t \odot \boldsymbol{c}^{t-1} \qquad \text{cell state} \tag{A328}$$

$$\boldsymbol{o}^t = \sigma\left(\boldsymbol{W_o}\,\boldsymbol{x}^t + \boldsymbol{b_o}\right) \qquad \text{output gate} \tag{A329}$$

$$\boldsymbol{y}^t = \boldsymbol{o}^t \odot h\left(\boldsymbol{c}^t\right) \qquad \text{cell output} \tag{A330}$$

A5.3 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.

A5.4 Special LSTM Architectures for Backward Analysis

A5.4.1 LSTM for Integrated Gradients

For Integrated Gradients backward analysis with 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)** $\boldsymbol{o}^t = 1$ for all t. That is the output gate is always 1 and nothing is forgotten.

(A3) We set $h = a_h \tanh$ with $a_h = 1, 2, 4$.

(A4) We set $g = a_g \tanh$ with $a_g = 1, 2, 4$.

(A5) 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.

(A6) 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.

(A7) The input gate $\boldsymbol{i}$ has a negative bias, that is, $\boldsymbol{b_i} < 0$. The negative bias reduces the drift effect, that is, the memory content $\boldsymbol{c}$ either increases or decreases over time. Typical values are $\boldsymbol{b_i} = -1, -2, -3, -4, -5$.

(A8) The memory cell content is initialized with zero at time $t = 0$, that is, $\boldsymbol{c}^0 = 0$.

The resulting LSTM forward pass rules for Integrated Gradients are:

$$\boldsymbol{z}^t = a_g \, \sigma\left(\boldsymbol{W_z} \, \boldsymbol{x}^t + \boldsymbol{b_z}\right) \quad \text{cell input} \quad \text{(A331)}$$

$$\boldsymbol{i}^t = \sigma\left(\boldsymbol{W_i} \, \boldsymbol{x}^t + \boldsymbol{b_i}\right) \quad \text{input gate} \quad \text{(A332)}$$

$$\boldsymbol{c}^t = \boldsymbol{i}^t \odot \boldsymbol{z}^t + \boldsymbol{c}^{t-1} \quad \text{cell state} \quad \text{(A333)}$$

$$\boldsymbol{y}^t = a_h \, \tanh\left(\boldsymbol{c}^t\right) \quad \text{cell output} \quad \text{(A334)}$$

See Figure A4 which depicts these forward pass rules for Integrated Gradients.

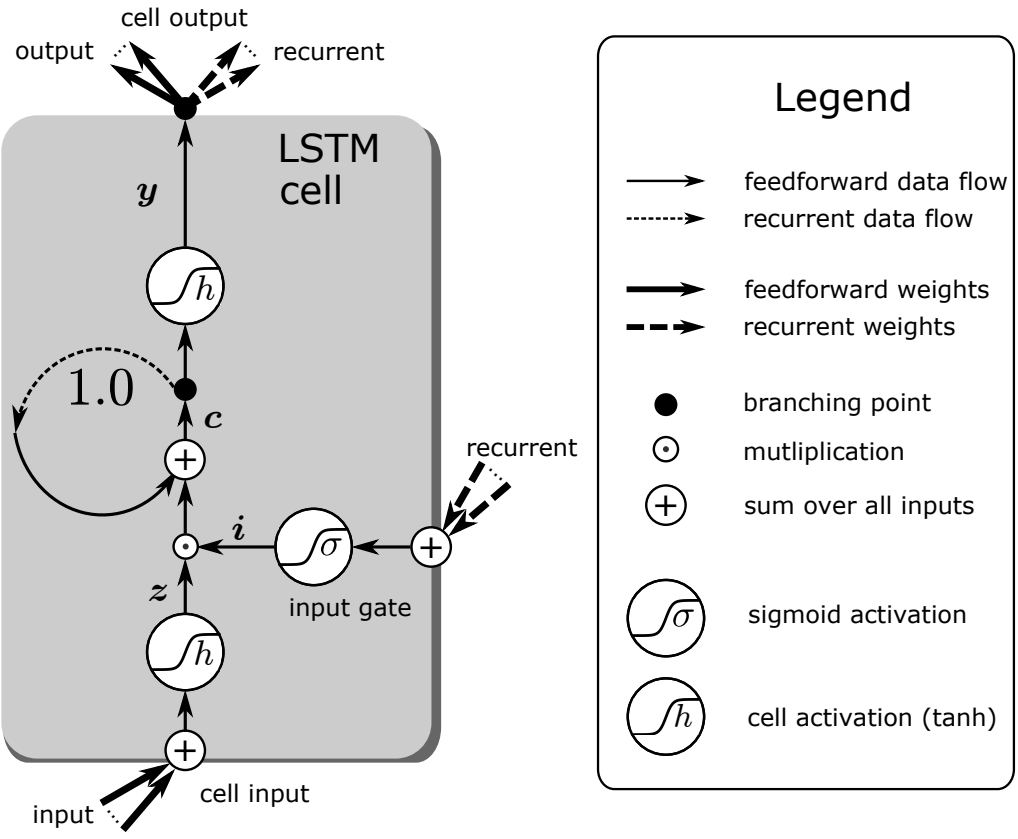


Figure A4: LSTM memory cell used for Integrated Gradients (IG). Forget gates and output gates are set to 1 since they can modify all cell inputs at times after they have been observed, which can make the dynamics highly nonlinear.

A5.4.2 LSTM for LRP

LRP has already been used for LSTM in order to identify important terms in sentiment analysis [1]. In texts, positive and negative terms with respect to the topic could be identified.
For LRP backward analysis with 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 [76]. 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 then 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, $\boldsymbol{c}^0 = 0$. The memory cell content $\boldsymbol{c}^t$ is non-negative $\boldsymbol{c}^t \geqslant 0$ since $\boldsymbol{z} \geqslant 0$ and $\boldsymbol{i} \geqslant 0$.

The resulting LSTM forward pass rules for LRP are:

$$\boldsymbol{z}^t = a_g \, \sigma\left(\boldsymbol{W_z} \, \boldsymbol{x}^t + \boldsymbol{b_z}\right) \qquad \text{cell input} \tag{A335}$$

$$\boldsymbol{i}^t = \sigma\left(\boldsymbol{W_i} \, \boldsymbol{x}^t + \boldsymbol{b_i}\right) \qquad \text{input gate} \tag{A336}$$

$$\boldsymbol{c}^t = \boldsymbol{i}^t \odot \boldsymbol{z}^t + \boldsymbol{c}^{t-1} \qquad \text{cell state} \tag{A337}$$

$$\boldsymbol{o}^t = \sigma\left(\boldsymbol{W_o} \, \boldsymbol{x}^t + \boldsymbol{b_o}\right) \qquad \text{output gate} \tag{A338}$$

$$\boldsymbol{y}^t = \boldsymbol{o}^t \odot a_h \, \tanh\left(\boldsymbol{c}^t\right) \qquad \text{cell output} \tag{A339}$$

See Figure A5 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.

LRP and Contribution Propagation for LSTM. We analyze Layer-wise Relevance Propagation (LRP) and Contribution Propagation for LSTM networks. A single memory cell can be described by:

$$c^t = i^t \, z^t + c^{t-1} \, . \tag{A340}$$

Here we treat i^t like a weight for z^t and c^{t-1} has weight 1.
For positive values of i^t, z^t, and c^{t-1}, both LRP and contribution propagation leads to

$$R_{c^t \leftarrow y^t} = R_{y^t} \tag{A341}$$

$$R_{c^t} = R_{c^t \leftarrow c^{t+1}} + R_{c^t \leftarrow y^t} \tag{A342}$$

$$R_{c^{t-1} \leftarrow c^t} = \frac{c^{t-1}}{c^t} \, R_{c^t} \tag{A343}$$

$$R_{z^t \leftarrow c^t} = \frac{i^t \, z^t}{c^t} \, R_{c^t} \, . \tag{A344}$$

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 $t = 1 \ldots T$:

$$R_{c^T} = R_{y^T} \tag{A345}$$

$$R_{c^{t-1}} = \frac{c^{t-1}}{c^t} \, R_{c^t} \tag{A346}$$

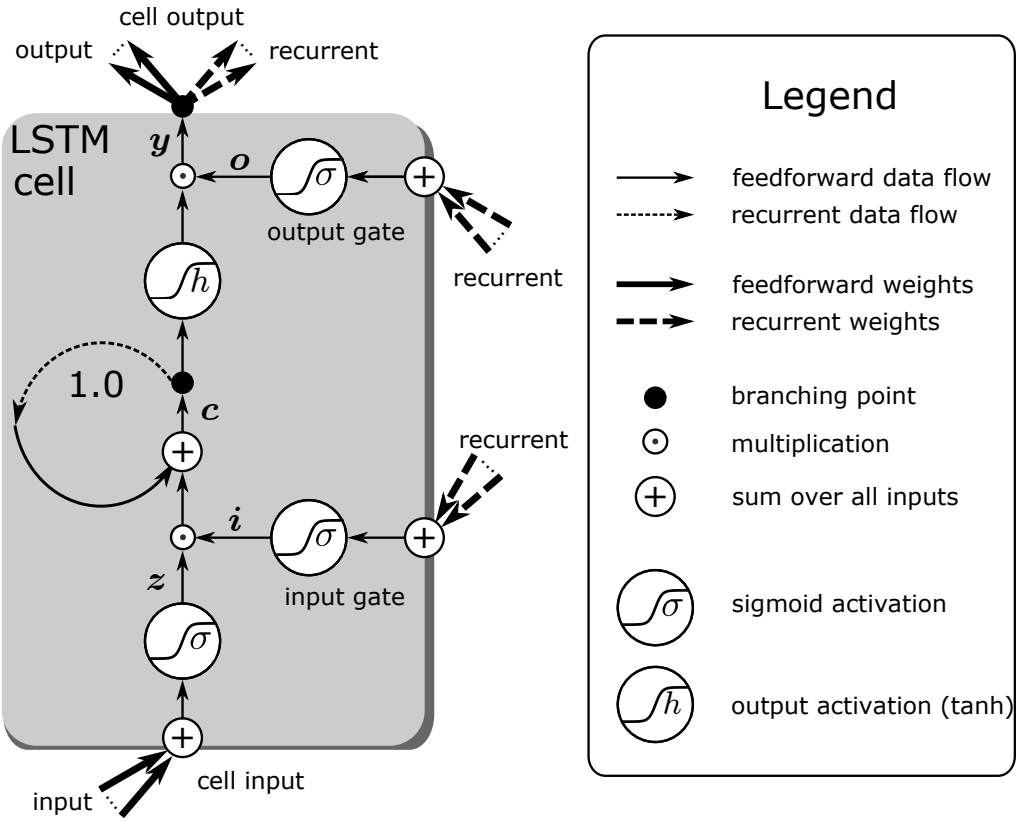


Figure A5: LSTM memory cell used for Layer-Wise Relevance Propagation (LRP). $\boldsymbol{z}$ is the vector of cell input activations, $\boldsymbol{i}$ is the vector of input gate activations, $\boldsymbol{c}$ is the vector of memory cell states, $\boldsymbol{o}$ is the vector of output gate activations, 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 an 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.

which gives

$$R_{c^t} \;=\; R_{y^T} \prod_{\tau=t+1}^{T} \frac{c^{\tau-1}}{c^\tau} \;=\; \frac{c^t}{c^T}\, R_{y^T} \tag{A347}$$

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

$$R_{c^0} \;=\; 0\,, \tag{A348}$$

$$R_{z^t} \;=\; \frac{i^t\, z^t}{c^T}\, R_{y^T}\,. \tag{A349}$$

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

$$c^T \;=\; \sum_{t=1}^{T} i^t\, z^t \tag{A350}$$

and therefore

$$R_{z^t} \;=\; \frac{i^t\, z^t}{\sum_{\tau=1}^{T} i^\tau\, z^\tau}\, R_{y^T}\,. \tag{A351}$$

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}.

A5.4.3 LSTM for Nondecreasing Memory Cells

Backward analysis is made simpler if memory cells are nondecreasing since the contribution of each input to each memory cells is well defined. The problem that a negative and a positive input cancels each other is avoided. For nondecreasing memory cells and backward analysis with 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 $a_g = 2, 3, 4$. 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 then 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.
- **(A5)** The input gate $\boldsymbol{i}$ is not connected to the input, that is, $\boldsymbol{W_i}$ has only connections to other memory cells.
- **(A6)** The output gate $\boldsymbol{o}$ is not connected to the input, that is, $\boldsymbol{W_o}$ has only connections to other memory cells.
- **(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, $\boldsymbol{c}^0 = 0$. We ensured via the architecture that $\boldsymbol{c}^t \geqslant 0$ and $\boldsymbol{c}^{t+1} \geqslant \boldsymbol{c}^t$, that is, the memory cells are positive and nondecreasing.

The resulting LSTM forward pass rules for nondecreasing memory cells are:

$$\boldsymbol{z}^t = a_g \, \sigma\left(\boldsymbol{W_z} \, \boldsymbol{x}^t + \boldsymbol{b_z}\right) \qquad \text{cell input} \qquad \text{(A352)}$$

$$\boldsymbol{i}^t = \sigma\left(\boldsymbol{W_i} \, \boldsymbol{x}^t + \boldsymbol{b_i}\right) \qquad \text{input gate} \qquad \text{(A353)}$$

$$\boldsymbol{c}^t = \boldsymbol{i}^t \odot \boldsymbol{z}^t + \boldsymbol{c}^{t-1} \qquad \text{cell state} \qquad \text{(A354)}$$

$$\boldsymbol{o}^t = \sigma\left(\boldsymbol{W_o} \, \boldsymbol{x}^t + \boldsymbol{b_o}\right) \qquad \text{output gate} \qquad \text{(A355)}$$

$$\boldsymbol{y}^t = \boldsymbol{o}^t \odot a_h \, \tanh\left(\boldsymbol{c}^t\right) \qquad \text{cell output} \qquad \text{(A356)}$$

See Figure A6 for a LSTM memory cell that is nondecreasing.

A5.4.4 LSTM without Gates

The most simple LSTM architecture for backward analysis does not use any gates. Therefore complex dynamics that have to be treated in the backward analysis are avoided. For LSTM without gates, we make following assumptions:

- **(A1)** $\boldsymbol{f}^t = 1$ for all t. That is the forget gate is always 1 and nothing is forgotten.
- **(A2)** $\boldsymbol{o}^t = 1$ for all t. That is the output gate is always 1.
- **(A3)** $\boldsymbol{i}^t = 1$ for all t. That is the input gate is always 1.
- **(A4)** $g > 0$, 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 $a_g = 2, 3, 4$. 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 $\boldsymbol{b_z} = -1, -2, -3, -4, -5$.
- **(A5)** We want to ensure that $h(0) = 0$. If the memory content is zero then nothing is transferred to the next layer. Therefore we set $h = a_h \tanh$ with $a_h = 1, 2, 4$.
- **(A6)** The memory cell content is initialized with zero at time $t = 0$, that is, $\boldsymbol{c}^0 = 0$.

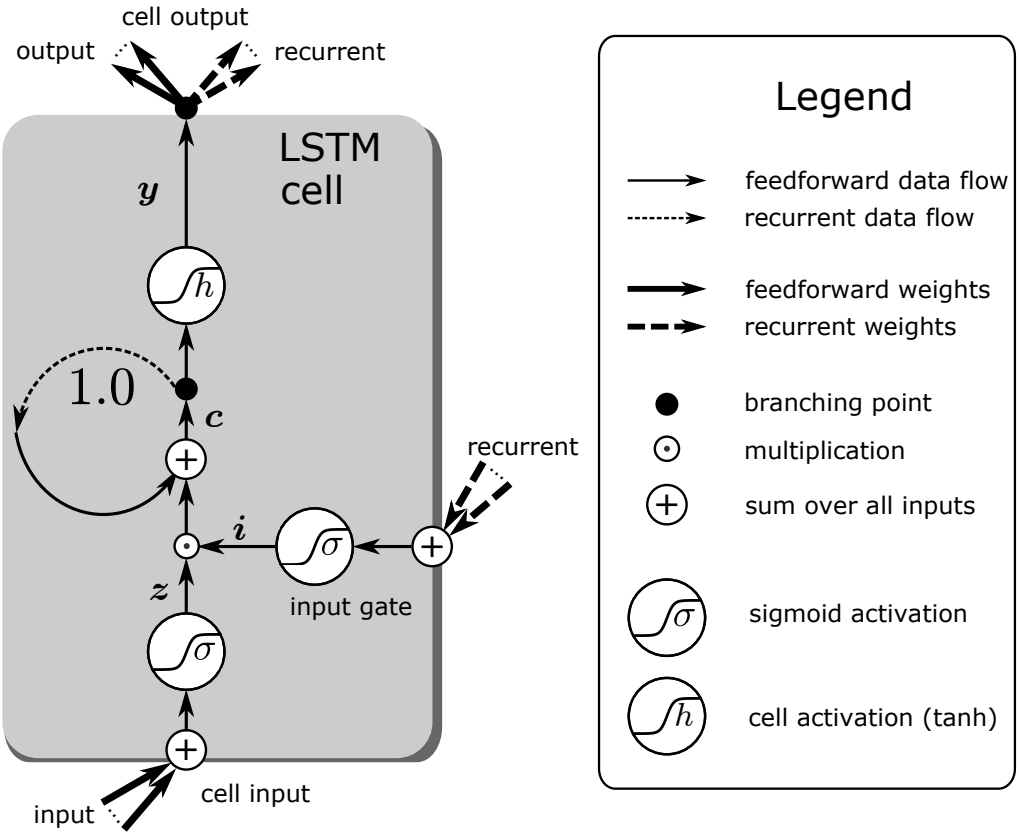


Figure A6: A nondecreasing LSTM memory cell.

The resulting LSTM forward pass rules are:

$$\boldsymbol{z}^t = a_g \, \sigma\left(\boldsymbol{W}_{\boldsymbol{z}} \, \boldsymbol{x}^t + \boldsymbol{b}_{\boldsymbol{z}}\right) \qquad \text{cell input} \qquad \text{(A357)}$$

$$\boldsymbol{c}^t = \boldsymbol{z}^t + \boldsymbol{c}^{t-1} \qquad \text{cell state} \qquad \text{(A358)}$$

$$\boldsymbol{y}^t = a_h \, \tanh\left(\boldsymbol{c}^t\right) \qquad \text{cell output} \qquad \text{(A359)}$$

See Figure A7 for a LSTM memory cell without gates which perfectly distributes the relevance across the input.

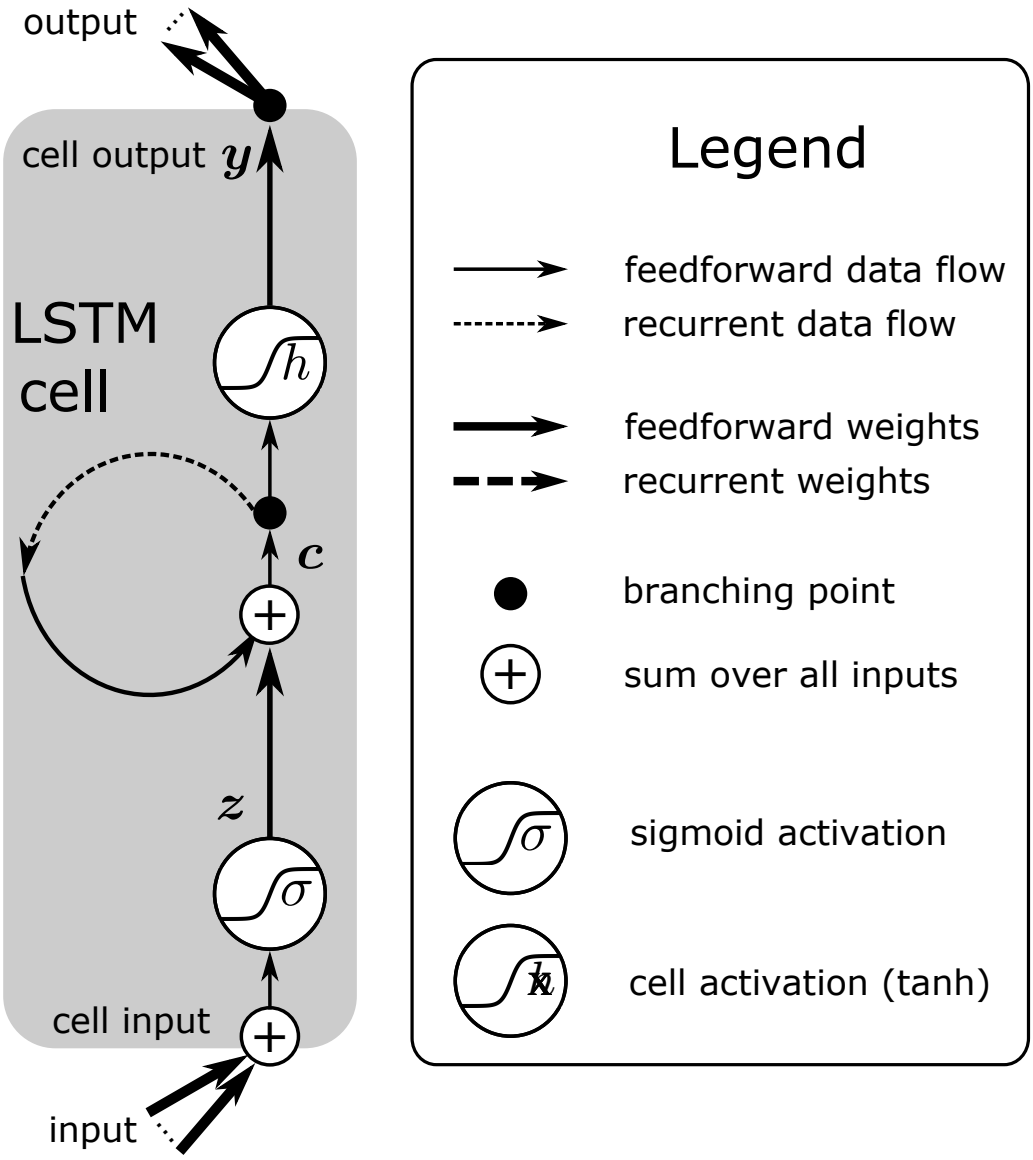


Figure A7: LSTM memory cell without gates.

A6 Artificial Tasks

This section supports the artificial tasks (I), (II), (III), (IV) in the main paper.

A6.1 RUDDER compared with Reward Shaping Methods

This section supports the artificial task (IV) – **Trace-back** – in the main paper. In this experiment, we compare reinforcement learning methods that have to transfer back information about a delayed reward. These methods comprise RUDDER, TD(λ and reward shaping approaches. For reward shaping we compare the original *reward shaping* [80], *look-forward advice*, and em look-back advice [129] with three different potential functions. Methods that transfer back reward information are characterized by low variance estimates of the value function or the action-value function, since they use an estimate of the future return instead of the future return itself. To update the estimates of the future returns, reward information has to be transferred back. The task in this experiment can be solved by Monte Carlo estimates very fast, which they do not transfer back information but use samples of the future return for estimation instead. However, Monte Carlo methods have high variance, which is not considered in this experiment. For comparisons of RUDDER with MC, see next subsection in Appendix A6.2.

The environment is a 15×15 grid, where actions move the agent from its current position in 4 adjacent positions (*up, down, left, right*), except the agent would be moved outside the grid. The number of steps (moves) per episode is $t = 20$. The starting position is $(7, 7)$ in the middle of the grid. The maximal return is a combination of negative immediate reward and positive delayed reward. To obtain the maximum return, the policy must move the agent *up* in the time step $t = 1$ and *right* in the following time step $t = 2$. In this case, the agent receives an immediate reward of -50 at $t = 2$ and a delayed reward of 150 at the end of the episode at $t = 20$, that is, a return of 100. Any other combination of actions gives the agent immediate reward of 50 at $t = 2$ without any delayed reward, that is, a return of 50. To ensure Markov properties the position of the agent, the time, as well as the delayed reward are coded in the state. The future reward discount rate γ is set to 1. The state transition probabilities are deterministic for the first two moves. For $t > 2$ and for each action, state transition probabilities are equal for each possible next state (uniform distribution), meaning that actions after $t = 2$ do not influence the return. For comparisons of long delays, both the size of the grid and the length of the episode are increased. For a delay of n, a $(3n/4) \times (3n/4)$ grid is used with an episode length of n, and starting position $(3n/8, 3n/8)$.

Compared Methods: We compare different TD(λ) and reward shaping methods. For TD(λ, the baseline is Q-learning, with eligibility traces $\lambda = 0.9$ and $\lambda = 0$ and Watkins' implementation [127]. The reward shaping methods are the original reward shaping, look-ahead advice as well as look-back advice. For look-back advice, we use *SARSA* [97] instead of Q-learning as suggested by the authors. Q-values are represented by a state-action table, that is, we consider only tabular methods. In all experiments an ϵ-greedy policy with $\epsilon = 0.2$ is used. All three reward shaping methods require a potential function ϕ, which is based on the reward redistribution ($\tilde{r}_t$) in three different ways:
(I) The Potential function ϕ is the difference of LSTM predictions, which is the redistributed reward R_t:

$$\phi(s_t) = \mathrm{E}\left[R_{t+1} \mid s_t\right] \quad \text{or} \tag{A360}$$

$$\phi(s_t, a_t) = \mathrm{E}\left[R_{t+1} \mid s_t, a_t\right] . \tag{A361}$$

(II) The potential function ϕ is the sum of future redistributed rewards, i.e. the q-value of the redistributed rewards. In the optimal case, this coincides with implementation (I):

$$\phi(s_t) = \mathrm{E}\left[\sum_{\tau=t}^{T} R_{\tau+1} \mid s_t\right] \quad \text{or} \tag{A362}$$

$$\phi(s_t, a_t) = \mathrm{E}\left[\sum_{\tau=t}^{T} R_{\tau+1} \mid s_t, a_t\right] . \tag{A363}$$

(III) The potential function ϕ corresponds to the LSTM predictions. In the optimal case this corresponds to the accumulated reward up to t plus the q-value of the delayed MDP:

$$\phi(s_t) \;=\; \mathrm{E}\left[\sum_{\tau=0}^{T} \tilde{R}_{\tau+1} \mid s_t\right] \quad \text{or} \tag{A364}$$

$$\phi(s_t, a_t) \;=\; \mathrm{E}\left[\sum_{\tau=0}^{T} \tilde{R}_{\tau+1} \mid s_t, a_t\right] . \tag{A365}$$

The following methods are compared:

1. Q-learning with eligibility traces according to Watkins ($Q(\lambda)$),
2. SARSA with eligibility traces (SARSA(λ)),
3. Reward Shaping with potential functions (I), (II), or (III) according to Q-learning and eligibility traces according to Watkins,
4. Look-ahead advise with potential functions (I), (II), or (III) with $Q(\lambda)$,
5. Look-back advise with potential functions (I), (II), or (III) with SARSA(λ),
6. RUDDER with reward redistribution for Q-value estimation and RUDDER applied on top of Q-learning.

RUDDER is implemented with an LSTM architecture without output gate and forget gate. For this experiments, RUDDER does not use lessons buffer nor safe exploration. For contribution analysis we use differences of return predictions. For RUDDER, the Q-values are estimated by an exponential moving average (RUDDER Q-value estimation) or alternatively by Q-learning.

Performance evaluation: The task is considered solved when the exponential moving average of the return is above 90, which is 90% of the maximum return. Learning time is the number of episodes required to solve the task. The first evaluation criterion is the average learning time. The Q-value differences at time step $t = 2$ are monitored. The Q-values at $t = 2$ are the most important ones, since they have to predict whether the maximal return will be received or not. At $t = 2$ the immediate reward acts as a distraction since it is -50 for the action leading to the maximal return (a^+) and 50 for all other actions (a^-). At the beginning of learning, the Q-value difference between a^+ and a^- is about -100, since the immediate reward is -50 and 50, respectively. Once the Q-values converge to the optimal policy, the difference approaches 50. However, the task will already be correctly solved as soon as this difference is positive. The second evaluation criterion is the Q-value differences at time step $t = 2$, since it directly shows to what extend the task is solved.

Results: Table A3 reports the number of episodes required by different methods to solve the task. The mean and the standard deviation over 100 trials are given. A Wilcoxon signed-rank test is performed between the learning time of RUDDER and those of the other methods. Statistical significance p-values are obtained by Wilcoxon signed-rank test. RUDDER with reward redistribution is significantly faster than all other methods with p-values $< 10^{-17}$. Tables A1,A2 report the results for all methods.

Table A1: Number of episodes required by different methods to solve the trace-back task with delayed reward. The numbers represent the mean and the standard deviation over 100 trials. RUDDER with reward redistribution significantly outperforms all other methods.

Method	**Delay 6**			**Delay 8**			**Delay 10**		
Look-back I	6073.94	± 952.03	p = 1.09E-23	13112.02	± 2024.24	p = 1.09E-23	21714.80	± 4322.59	p = 1.03E-07
Look-back II	4584.08	± 916.87	p = 3.32E-23	9897.28	± 2082.74	p = 1.09E-23	15973.00	± 4354.48	p = 1.03E-07
Look-back III	4036.48	± 1424.99	p = 5.28E-17	7812.72	± 2279.26	p = 1.09E-23	10982.40	± 2971.65	p = 1.03E-07
Look-ahead I	14469.10	± 1520.81	p = 1.09E-23	28559.32	± 2104.91	p = 1.09E-23	46650.20	± 3035.78	p = 1.03E-07
Look-ahead II	12623.42	± 1075.25	p = 1.09E-23	24811.62	± 1986.30	p = 1.09E-23	43089.00	± 2511.18	p = 1.03E-07
Look-ahead III	16050.30	± 1339.69	p = 1.09E-23	30732.00	± 1871.07	p = 1.09E-23	50340.00	± 2102.78	p = 1.03E-07
Reward Shaping I	14686.12	± 1645.02	p = 1.09E-23	28223.94	± 3012.81	p = 1.09E-23	46706.50	± 3649.57	p = 1.03E-07
Reward Shaping II	11397.10	± 905.59	p = 1.09E-23	21520.98	± 2209.63	p = 1.09E-23	37033.40	± 1632.24	p = 1.03E-07
Reward Shaping III	12125.48	± 1209.59	p = 1.09E-23	23680.98	± 1994.07	p = 1.09E-23	40828.70	± 2748.82	p = 1.03E-07
Q-learning	14719.58	± 1728.19	p = 1.09E-23	28518.70	± 2148.01	p = 1.09E-23	44017.20	± 3170.08	p = 1.03E-07
SARSA	8681.94	± 704.02	p = 1.09E-23	23790.40	± 836.13	p = 1.09E-23	48157.50	± 1378.38	p = 1.03E-07
RUDDER I	995.59	± 670.31	p = 5.00E-01	1128.82	± 741.29	p = 5.00E-01	1186.34	± 870.02	p = 5.00E-01
RUDDER II	726.72	± 399.58	p = 3.49E-04	809.86	± 472.27	p = 3.49E-04	906.13	± 514.55	p = 3.36E-02
Method	**Delay 12**			**Delay 15**			**Delay 17**		
Look-back I	33082.56	± 7641.57	p = 1.09E-23	49658.86	± 8297.85	p = 1.28E-34	72115.16	± 21221.78	p = 1.09E-23
Look-back II	23240.16	± 9060.15	p = 1.09E-23	29293.94	± 7468.94	p = 1.28E-34	42639.38	± 17178.81	p = 1.09E-23
Look-back III	15647.40	± 4123.20	p = 1.09E-23	20478.06	± 5114.44	p = 1.28E-34	26946.92	± 10360.21	p = 1.09E-23
Look-ahead I	66769.02	± 4333.47	p = 1.09E-23	105336.74	± 4977.84	p = 1.28E-34	136660.12	± 5688.32	p = 1.09E-23
Look-ahead II	62220.56	± 3139.87	p = 1.09E-23	100505.05	± 4987.16	p = 1.28E-34	130271.88	± 5397.61	p = 1.09E-23
Look-ahead III	72804.44	± 4232.40	p = 1.09E-23	115616.59	± 5648.99	p = 1.28E-34	149064.68	± 7895.48	p = 1.09E-23
Reward Shaping I	68428.04	± 3416.12	p = 1.09E-23	107399.17	± 5242.88	p = 1.28E-34	137032.14	± 6663.12	p = 1.09E-23
Reward Shaping II	56225.24	± 3778.86	p = 1.09E-23	93091.44	± 5233.02	p = 1.28E-34	122224.20	± 5545.63	p = 1.09E-23
Reward Shaping III	60071.52	± 3809.29	p = 1.09E-23	99476.40	± 5607.08	p = 1.28E-34	130103.50	± 6005.61	p = 1.09E-23
Q-learning	66952.16	± 4137.67	p = 1.09E-23	107438.36	± 5327.95	p = 1.28E-34	135601.26	± 6385.76	p = 1.09E-23
SARSA	78306.28	± 1813.31	p = 1.09E-23	137561.92	± 2350.84	p = 1.28E-34	186679.12	± 3146.78	p = 1.09E-23
RUDDER I	1121.70	± 884.35	p = 5.00E-01	1503.08	± 1157.04	p = 5.00E-01	1242.88	± 1045.15	p = 5.00E-01
RUDDER II	1065.16	± 661.71	p = 3.19E-01	972.73	± 702.92	p = 1.13E-04	1101.24	± 765.76	p = 1.54E-01

Table A2: Cont. Number of episodes required by different methods to solve the trace-back task with delayed reward. The numbers represent the mean and the standard deviation over 100 trials. RUDDER with reward redistribution significantly outperforms all other methods.

Method	**Delay 20**			**Delay 25**		
Look-back I	113873.30	± 31879.20	p = 1.03E-07			
Look-back II	56830.30	± 19240.04	p = 1.03E-07	111693.34	± 73891.21	p = 1.09E-23
Look-back III	35852.10	± 11193.80	p = 1.03E-07			
Look-ahead I	187486.50	± 5142.87	p = 1.03E-07			
Look-ahead II	181974.30	± 5655.07	p = 1.03E-07	289782.08	± 11984.94	p = 1.09E-23
Look-ahead III	210029.90	± 6589.12	p = 1.03E-07			
Reward Shaping I	189870.30	± 7635.62	p = 1.03E-07	297993.28	± 9592.30	p = 1.09E-23
Reward Shaping II	170455.30	± 6004.24	p = 1.03E-07	274312.10	± 8736.80	p = 1.09E-23
Reward Shaping III	183592.60	± 6882.93	p = 1.03E-07	291810.28	± 10114.97	p = 1.09E-23
Q-learning	186874.40	± 7961.62	p = 1.03E-07			
SARSA	273060.70	± 5458.42	p = 1.03E-07	454031.36	± 5258.87	p = 1.09E-23
RUDDER I	1048.97	± 838.26	p = 5.00E-01	1236.57	± 1370.40	p = 5.00E-01
RUDDER II	1159.30	± 731.46	p = 8.60E-02	1195.75	± 859.34	p = 4.48E-01

A6.2 RUDDER compared with TD, MC, MCTS

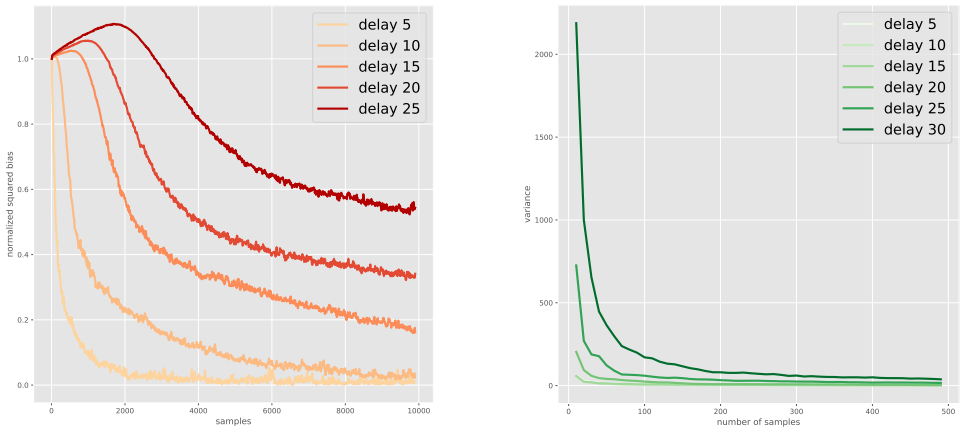
This section supports the artificial tasks (I), (II), (III) in the main paper.

A6.2.1 Grid World

This environment is characterized by probabilistic delayed rewards. It illustrates a situation, where a time bomb explodes at episode end. The agent has to defuse the bomb and then run away as far as possible since defusing fails with a certain probability. Alternatively, the agent can immediately run away, which, however, leads to less reward on average since the bomb always explodes. The Grid World is 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 are allowed that keep the agent on the grid. The episode finishes after $1.5d$ steps. At the end of the episode, with a given probability of 0.5, 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 the factor c depends on the chosen action, t is the current time step, and h is the Hamming distance to *bomb*. Each move of the agent, which reduces the Hamming distance to *bomb*, is penalized by the immediate reward via $c = -0.09$. Each move of the agent, which increases the Hamming distance to *bomb*, is rewarded by the immediate reward via $c = 0.1$. The agent is forced to learn the Q-values precisely, since the immediate reward of directly running away hints at a sub-optimal 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 the 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 experimentally 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 A5.
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 exponentially faster and for MC it has fewer number of action-values with high variance. Consequently, estimators for the new MDP learn 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 such 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 exponentially weighted arithmetic mean 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 are run for 10,000 episodes. Figure A8aa shows the bias correction for different delays, normalized by the first error.
For the MC update rule we use 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 A2.2.1, in Figure A8b the variance decreases by $1/n$, where n is the number of samples. Figure A8ab 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 apply Monte Carlo Tree Search (MCTS), Q-learning, and Monte Carlo (MC). We train until the greedy policy reaches 90% of the return of the optimal policy. The learning time is measured by the number of episodes. We use *sample updates* for Q-learning and MC [115]. For MCTS the greedy policy uses 0 for the exploration constant in UCB1 [58]. The greedy policy is evaluated in 100 episodes intervals. The MCTS selection step begins in the start state, which is the root of the game tree that is traversed using UCB1 [58] as the tree policy. If a tree-node gets 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



(a) (b)

Figure A8: (a) Experimental evaluation of bias and variance of different Q-value estimators on the Grid World. (b) Normalized bias reduction for different delays. Right: Average variance reduction for the 10th highest values.

the MCTS(1) update rule [56]. 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 A2.3.4 is used. For each *delay* and each method, 300 runs with different seeds are performed to obtain statistically relevant results.

Estimation of the median learning time and quantiles The performance of different methods is measured by the median learning time in terms of episodes. We stop training at 100 million episodes. Some runs, especially for long delays, have taken too long and have thus been stopped. To resolve this bias the quantiles of the learning time are estimated by fitting a distribution using right censored data [26] .The median is still robustly estimated if more than 50% of runs have finished, which is the case for all plotted datapoints. We find that for delays where all runs have finished the learning time follows a Log-normal distribution. Therefore, we fit a Log-normal distribution on the right censored data. We estimate the median from the existing data, and use maximum likelihood estimation to obtain the second distribution parameter σ^2. The start value of the σ^2 estimation is calculated by the measured variance of the existing data which is algebraically transformed to get the σ parameter.

A6.2.2 The Choice

In this experiment we compare RUDDER, temporal difference (TD) and Monte Carlo (MC) in an environment with delayed deterministic reward and probabilistic state transitions to investigate how reward information is transferred back to early states. This environment reveals problems of the forward-view estimators such as TD or MC, while backward-view estimators based on contribution analysis excel. To make these problems more evident, only the first action at the very beginning determines the reward at the end of the episode.

The environment is an MDP consisting of two actions $a \in \mathcal{A} = \{+, -\}$, an initial state s^0, two *charged* states s^+, s^-, two *neutral* states $s^\oplus$, $s^\ominus$, and a final state s^f. After the first action $a_0 \in \mathcal{A} = \{+, -\}$ in state s^0, the agent transits to state s^+ for action $a_0 = +$ and to s^- for action $a_0 = -$. Subsequent state transitions are probabilistic and independent on actions. With probability p_C the agent stays in the *charged* states s^+ or s^-, and with probability $(1 - p_C)$ it transits from s^+ or s^- to the *neutral* states $s^\oplus$ or $s^\ominus$, respectively. The probability to go from *neutral* states to *charged* states is p_C, and the probability to stay in *neutral* states is $(1 - p_C)$. Probabilities to transit from s^+ or $s^\oplus$ to s^- or $s^\ominus$ or vice versa are zero. Thus, the first action determines whether that agent stays in "$+$"-states or "$-$"-states. The reward is determined by how many times the agent visits *charged* states plus a bonus reward depending on the agent's first action. The accumulative reward is given at

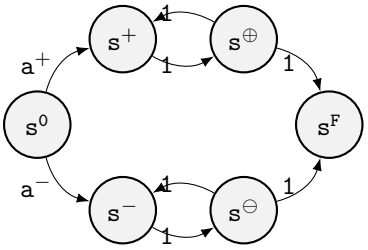


Figure A9: State transition diagram for The Choice task. The diagram is a simplification of the actual MDP.

sequence end and is deterministic. After T time steps, the agent is in the final state s^{f}, in which the reward R_{T+1} is provided. R_{T+1} is the sum of 3 deterministic terms:

1. R_0, the baseline reward associated to the first action;
2. R_C, the collected reward across states, which depends on the number of visits n to the *charged* states;
3. R_b, a bonus if the first action $a_0 = +$.

The expectations of the accumulative rewards for R_0 and R_C have the same absolute value but opposite signs, therefore they cancel in expectation over episodes. Thus, the expected return of an episode is the expected reward R_b: $p(a_0 = +)b$. The rewards are defined as follows:

$$c_0 = \begin{cases} 1 & \text{if } a_0 = + \\ -1 & \text{if } a_0 = - \end{cases}, \tag{A366}$$

$$R_b = \begin{cases} b & \text{if } a_0 = + \\ 0 & \text{if } a_0 = - \end{cases}, \tag{A367}$$

$$R_C = c_0 \, C \, n \, , \tag{A368}$$

$$R_0 = - \, c_0 \, C \, p_C \, T \, , \tag{A369}$$

$$R_{T+1} = R_C + R_0 + R_b \, , \tag{A370}$$

where C is the baseline reward for *charged* states, and p_C the probability of staying in or transiting to *charged* states. The expected visits of charged states is $\mathrm{E}[n] = p_C T$ and $\mathrm{E}[R_{T+1}] = \mathrm{E}[R_b] = p(a_0 = +)b$.

Methods Compared: The following methods are compared:

1. Q-learning with eligibility traces according to Watkins [127],
2. Monte Carlo,
3. RUDDER with reward redistribution.

For RUDDER, we use an LSTM without lessons buffer and without safe exploration. Contribution analysis is realized by differences of return predictions. For MC, Q-values are the exponential moving average of the episode return. For RUDDER, the Q-values are estimated by an exponential moving average of the reward redistribution.

Performance evaluation and results: The task is considered as solved when the exponential moving average of the selection of the desired action at time $t = 0$ is equal to $1 - \epsilon$, where ϵ is the exploration rate. The performances of the compared methods are measured by the average learning time in the number of episodes required to solve the task. A Wilcoxon signed-rank test is performed between the learning time of RUDDER and those of the other methods. Statistical significance p-values are obtained by Wilcoxon signed-rank test. RUDDER with reward redistribution is significantly faster than all other methods with p-values $< 10^{-8}$. Table A3 reports the number of episodes required by different methods to solve the task. RUDDER with reward redistribution clearly outperforms all other methods.

Table A3: Number of episodes required by different methods to solve the grid world task with delayed reward. Numbers give the mean and the standard deviation over 100 trials. RUDDER with reward redistribution clearly outperforms all other TD methods.

Method	**Delay 10**			**Delay 15**			**Delay 20**		
RUDDER	3520.06	± 2343.79	p = 5.00E-01	3062.07	± 1278.92	p = 5.00E-01	3813.96	± 2738.18	p = 5.00E-01
MC	10920.64	± 7550.04	p = 5.03E-24	17102.89	± 12640.09	p = 1.98E-30	22910.85	± 19149.02	p = 1.25E-28
Q	66140.76	± 1455.33	p = 1.28E-34	115352.25	± 1962.20	p = 1.28E-34	171571.94	± 2436.25	p = 1.28E-34

Method	**Delay 25**			**Delay 30**			**Delay 35**		
RUDDER	4111.69	± 3768.58	p = 5.00E-01	3667.32	± 1776.48	p = 5.00E-01	3850.02	± 2874.77	p = 5.00E-01
MC	39772.34	± 47460.17	p = 2.28E-30	41922.15	± 36618.46	p = 2.78E-31	50464.40	± 60317.97	p = 1.30E-31
Q	234912.02	± 2672.87	p = 1.28E-34	305894.40	± 2928.07	p = 1.28E-34	383422.44	± 4346.46	p = 1.09E-23

Method	**Delay 40**			**Delay 45**			**Delay 50**		
RUDDER	3739.18	± 2138.87	p = 5.00E-01	4150.76	± 2582.68	p = 5.00E-01	3883.56	± 2188.41	p = 5.00E-01
MC	56945.11	± 54149.92	p = 1.01E-31	69844.73	± 79705.28	p = 5.67E-32	73242.80	± 70398.57	p = 2.67E-32
Q	466531.46	± 3514.72	p = 1.09E-23						

Method	**Delay 100**			**Delay 500**		
RUDDER	4146.55	± 2391.94	p = 4.95E-01	5768.90	± 4309.38	p = 4.97E-01
MC	119568.40	± 110048.95	p = 1.61E-12	345533.55	± 320231.79	p = 8.48E-17

A6.2.3 Charge-Discharge

The Charge-Discharge task depicted in Figure A10 is characterized by probabilistic policies, but deterministic reward and state transitions. The environment consists of 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 $T+1$,
where T corresponds to the maximal delay of the reward. The optimal policy alternates between charging and discharging to accumulate a reward of 10 every other time step. 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.

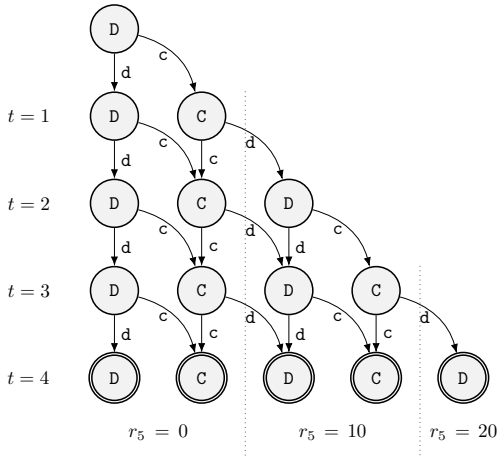


Figure A10: The Charge-Discharge task with two basic states: *charged* C and *discharged* D. In each state the actions charge c leading to the charged state C and discharge d leading to discharged state D are possible. Action d in the discharged state D leads to a small immediate reward of 1 and in the 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.

For this task, the RUDDER backward analysis is based on monotonic LSTMs as described in Section A5.4 and on layer-wise relevance propagation (LRP) as described in Section A4.4. The reward redistribution provided by RUDDER uses an LSTM which consists of 5 memory cells and is trained with Adam and a learning rate of 0.01. The reward redistribution is 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 use *sample updates* for Q-learning and MC [115]. The learning is stopped either if the agent achieves 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 are performed to obtain statistically relevant results. For delays with runs which did not finish within 100m episodes we estimate parameters like described in Paragraph A6.2.1.

A7 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 [15] for the RL components and our package for the LSTM reward redistribution model, which will be announced upon publication. If not specified otherwise, standard input processing, such as skipping 3 frames and stacking 4 frames, is performed by the *OpenAI baselines* package.
We consider the 52 Atari games that were compatible with OpenAI baselines, Arcade Learning Environment (ALE) [8], and OpenAI Gym [12]. Games are divided into episodes, i.e. the loss of a life or the exceeding of 108k frames trigger the start of a new episode without resetting the environment. Source code will be made available at upon publication.

A7.1 Architecture

We use a modified PPO architecture and a separate reward redistribution model. While parts of the two could be combined, this separation allows for better comparison between the PPO baseline with and without PPO.

PPO architecture. The design of the policy and the value network relies on the *ppo2* implementation [15], which is depicted in Figure A11 and summarized in Table A4. The network input, 4 stacked Atari game frames [74], are processed by 3 convolution layers with ReLU activation functions, followed by a fully connected layer with ReLU activation functions and 2 output units for the original and redistributed reward value function and another set of output units for the policy prediction. Striding is applied in the convolutional layers to downscale the number of features.

Reward redistribution model. Core of the reward redistribution model is an LSTM layer containing 64 memory cells with sigmoid gate activations, tanh input nonlinearities, and linear output functions, as illustrated in Figure A11 and summarized in Table A4. This LSTM implementation omits output gate and forget gate to simplify the network dynamics. Furthermore, the input gate is only connected recurrently to other LSTM blocks and the cell input is only connected to forward connections from the lower layer. For the vision system the same architecture was used as with the PPO network, with the first convolution layer being doubled to process Δ frames and full frames separately in the first layer. Additionally, the memory cell layer receives the vision feature activations of the PPO network, the the current action, and the approximate in-game time as inputs. No gradients from the reward redistribution network are propagated over the connections to the PPO network. After the memory cell layer, the LSTM has one output node for the prediction of the return G_0 and 4 output nodes for the auxiliary tasks as described in Section A7.3.

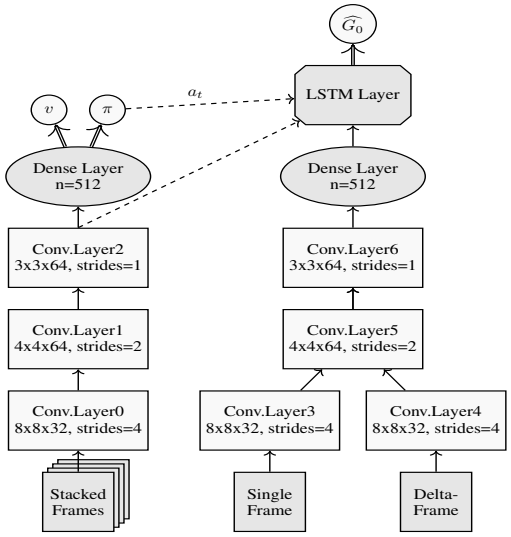


Figure A11: RUDDER models for Atari games as described in Section A7.1. Left: The *ppo2* implementation [15]. Right: LSTM reward redistribution model. The reward redistribution network has access to the PPO vision features (dashed lines) but no gradient is propagated between the networks. The LSTM layer receives the current action and an approximate in-game-time as additional input. The PPO outputs v for value function prediction and π for policy prediction each represent multiple output nodes: the original and redistributed reward value function prediction for v and the outputs for all of the available actions for π. Likewise, the reward redistribution output $\widehat{G_0}$ represents multiple outputs, as described in Section A7.3 Details on layer configuration are given in table A4.

A7.2 Lessons Replay Buffer

The lessons replay buffer is realized as a priority-based buffer containing up to 64 samples. New samples are added to the buffer if (i) the buffer is not filled or if (ii) the new sample is considered more important than the least important sample in the buffer, in which case the new sample replaces the least important sample.

Layer	Specifications		Layer	Specifications	
Conv.Layer 0	features	32	Conv.Layer 5	features	64
	kernelsize	8x8		kernelsize	4x4
	striding	4x4		striding	2x2
	act	ReLU		act	ReLU
	initialization	orthogonal, `gain`=$\sqrt{2}$		initialization	orthogonal, `gain`=$\sqrt{2}$
Conv.Layer 1	features	64	Conv.Layer 6	features	64
	kernelsize	4x4		kernelsize	3x3
	striding	2x2		striding	1x1
	act	ReLU		act	ReLU
	initialization	orthogonal, `gain`=$\sqrt{2}$		initialization	orthogonal, `gain`=$\sqrt{2}$
Conv.Layer 2	features	64	Dense Layer 1	features	512
	kernelsize	3x3		act	ReLU
	striding	1x1		initialization	orthogonal, `gain`=$\sqrt{2}$
	act	ReLU	LSTM Layer	cells	64
	initialization	orthogonal, `gain`=$\sqrt{2}$		gate act.	sigmoid
Dense Layer 0	features	512		ci act.	tanh
	act	ReLU		output act.	linear
	initialization	orthogonal, `gain`=$\sqrt{2}$		bias ig	trunc.norm., `mean`= -5
Conv.Layer 3	features	32		bias ci	trunc.norm., `mean`= 0
	kernelsize	8x8		fwd.w. ci	trunc.norm., `scale`= 0.0001
	striding	4x4		fwd.w. ig	omitted
	act	ReLU		rec.w. ci	omitted
	initialization	orthogonal, `gain`=$\sqrt{2}$		rec.w. ig	trunc.norm., `scale`= 0.001
Conv.Layer 4	features	32		og	omitted
	kernelsize	8x8		fg	omitted
	striding	4x4			
	act	ReLU			
	initialization	orthogonal, `gain`=$\sqrt{2}$			

Table A4: RUDDER models for Atari games. Specifications of layers shown in Figure A11. 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`.

Importance of samples for the buffer is determined based on a combined ranking of (i) the reward redistribution model error and (ii) the difference of the sample return to the mean return of all samples in the lessons buffer. Each of these two rankings contributes equally to the final ranking of the sample. Samples with higher loss and greater difference to the mean return achieve a higher ranking. Sampling from the lessons buffer is performed as a sampling from a softmax function on the sample-losses in the buffer. Each sample is a sequence of 512 consecutive transitions, as described in the last paragraph of Section A7.3.

A7.3 Game Processing and Update/Target Design

Reward redistribution is performed in an online fashion as new transitions are sampled from the environment. This allows to keep the original update schema of the PPO baseline, while still using the redistributed reward for the PPO updates. Training of the reward redistribution model is done separately on the lessons buffer samples from Section A7.2. These processes are described in more detail in the following paragraphs.

Reward Scaling. As described in the main paper, rewards are scaled based on the maximum return per sample R_{max} encountered during training so far:

$$r_{new} = \frac{10r}{|R_{max}|} \tag{A371}$$

Goal of this scaling is to normalize the reward r to range $[-10, 10]$, suitable for training the PPO and reward redistribution network, without losing information. Downside to this approach is that if a new R_{max} is encountered, the scaling factor is updated, and the networks have to readjust. However, since the scaling is linear, the relative differences between rewards are kept.

Reward redistribution. Reward redistribution is performed using differences of return predictions of the LSTM network. That is, the differences of the reward redistribution model prediction $\widehat{G_0}$ at timestep t and $t-1$ serve as contribution analysis and thereby give the redistributed reward $R_t = \widehat{G_{0t}} - \widehat{G_{0t-1}}$. This allows for online reward redistribution on the sampled transitions before they are used to train the PPO network, without waiting for the game sequences to be completed.

To assess the current quality of the reward redistribution model, a quality measure based on the relative absolute error is introduced:

$$quality = 1 - \frac{\left|G_0 - \widehat{G_0}\right|}{\mu} \cdot \frac{1}{1-\epsilon}, \tag{A372}$$

with ϵ as quality threshold of $\epsilon = 80\%$ and the maximum possible error μ as $\mu = 10$ due to the reward scaling applied. $quality$ is furthermore clipped to be within range $[0, 1]$.

PPO model. The *ppo2* implementation [15] samples from the environment using multiple 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 online on a batch of transitions sampled from the environment. Originally, the policy/value function network updates are adjusted using a policy loss, a value function loss, and an entropy term, each with dedicated scaling factors [104]. To decrease the number of hyperparameters, the entropy term scaling factor is adjusted automatically using Proportional Control to keep the policy entropy in a predefined range.
We use two value function output units to predict the value functions of the original and the redistributed reward. Analogous to the *ppo2* implementation, these two value function predictions serve to compute the advantages used to scale the policy gradient updates. For this, the advantages for original reward a_o and redistributed reward a_r are combined as a weighted sum $a = a_o \cdot (1 - quality) + a_r \cdot quality$. The PPO value function loss term l_v is replaced by the sum of the value function loss for the original reward l_{v_o} and the scaled value function loss for the redistributed reward l_{v_r}, such that $l_v = l_{v_o} + l_{v_r} \cdot quality$. Parameter values were taken from the original paper [104] and implementation [15], except for the value function coefficient, which was increased from 1 to 5.

Reward redistribution model. The loss of the reward redistribution model for a sample is composed of four parts. (i) The main loss l_m, which is the squared prediction loss of $\widehat{G}_0$ at the last time step T of the episode

$$l_m = \left(G_0 - \widehat{G_0}_T\right)^2, \tag{A373}$$

(ii) the continuous prediction loss l_c of $\widehat{G}_0$ at each timestep

$$l_c = \sum_0^T \left(\left(G_0 - \widehat{G_0}_t\right)^2\right)/T, \tag{A374}$$

(iii) loss of the earlier prediction of the main task output l_e at $t + 10$ at each timestep t

$$l_e = \sum_{t=0}^{T-10} \left(\left(\widehat{G_0}_{t+10} - \widehat{\widehat{G_0}_{t+10}}_t\right)^2\right)/(T-10), \tag{A375}$$

as well as (iv) the loss on 3 auxiliary tasks. At every time step t, these auxiliary tasks are (1) the prediction of the action-value function q_t, (2) the prediction of the reward r in the next 10 frames, and (3) the prediction of the reward r in the next 50 frames $\sum_{i=t}^{t+50} r_i$, resulting in the final auxiliary loss l_a as

$$l_{a_1} = \sum_{t=0}^{T} \left((q_t - \widehat{q_t}_t)^2\right)/T, \tag{A376}$$

$$l_{a_2} = \sum_{t=0}^{T-10} \left(\left(\sum_{i=t}^{t+10}(r_i) - \widehat{\sum_{i=t}^{t+10}(r_i)}_t\right)^2\right)/(T-10), \tag{A377}$$

$$l_{a_3} = \sum_{t=0}^{T-50} \left(\left(\sum_{i=t}^{t+50}(r_i) - \widehat{\sum_{i=t}^{t+50}(r_i)}_t\right)^2\right)/(T-50), \tag{A378}$$

$$l_a = \left(l_{a_1} + l_{a_2} + l_{a_3}\right)/3. \tag{A379}$$

The final loss for the reward redistribution model is then computed as

$$l = l_m + (l_c + l_e)/2 + l_a, \tag{A380}$$

where the continuous prediction and earlier prediction part $(l_c + l_e)/2$ and the auxiliary loss l_a push the reward redistribution model toward performing an optimal return decomposition. This is because they force the model to not use Markov properties in the environment, which hinder storing important events since those events are redundantly encoded in later states. Furthermore, they speed up learning by adding more information about the immediate rewards to the updates.
The reward redistribution model is only trained on the lessons buffer. Training epochs on the lessons buffer are performed every 10^4 PPO updates or if a new sample was added to the lessons buffer. For each such training epoch, 8 samples are sampled from the lessons buffer. Training epochs are repeated with new samples until the reward redistribution quality is sufficient ($quality > 0$) on all replayed samples.
Parameter values are listed in table A5.

Sequence chunking and Truncated BackPropagation Through Time (TBPTT). Ideally, RUDDER would be trained on completed game sequences, to consequently redistribute the reward within a completed game. To shorten computational time for learning the reward redistribution model, the model is not trained on completed game sequences but on sequence chunks consisting of 512 timesteps. The beginning of such a chunk is treated as beginning of a new episode for the model and ends of episodes within this chunk reset the state of the LSTM, so as to not redistribute rewards between episodes. To allow for updates on sequence chunks even if the game sequence is not completed, the PPO value function prediction is used to estimate the expected future reward at the end of the chunk.
Utilizing TBPTT to further speed up LSTM learning, gradients for the reward redistribution LSTM are cut after 128 timesteps.

learning rate	10^{-4}
l_2 weight decay	10^{-7}
gradient clipping	0.5
optimization	ADAM

Table A5: Update parameters for reward redistribution model.

A7.4 Exploration

Safe exploration to increase the likelihood to observe delayed rewards is an important feature of RUDDER. We use a safe exploration strategy, which is realized by normalizing the output of the policy network and randomly picking one of the actions that is above a threshold θ. 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 but safe off-policy trajectories within parts of the game sequences. θ is linearly increased from an agent with the strongest exploration $\theta = \theta_{\min}$ to an on-policy agent with $\theta = 1$. Only 4 of the 8 parallel actors use safe exploration but all actors sample from the softmax policy output.
To avoid policy lag during safe exploration transitions, we use those transitions only to update the reward redistribution model but not the PPO model.

A7.5 Results

Training curves for 3 random seeds for PPO baseline and PPO with RUDDER are shown in figure A14 and scores are listed in table A6 for all 52 Atari games. Training was conducted over 100M game frames (including skipped frames), as described in the experiments section of the main paper. Figures A12 and A13 display how RUDDER redistributes rewards to key events in the Atari games Bowling and Venture.

	average			*maximum*			*final*		
	B	R	%	B	R	%	B	R	%
Alien	721	1516	110	1350	2977	121	984	2400	144
Amidar	291	244	-16	594	511	-14	531	452	-15
Assault	991	1763	78	2061	3450	67	1599	3111	95
Asterix	1335	1079	-19	2707	2210	-18	2037	1618	-21
Asteroids	1216	1217	0	1792	1980	10	1503	1416	-6
Atlantis	148238	137277	-7	825613	1114150	35	259550	579847	123
BankHeist	515	495	-4	1018	822	-19	733	599	-18
BattleZone	11513	13069	14	19667	22767	16	13000	16600	28
BeamRider	927	1334	44	1631	3272	101	1019	2151	111
Berzerk	615	630	2	884	861	-3	747	722	-3
Bowling	60	149	148	77	199	158	61	165	170
Boxing	33	48	45	75	79	5	67	69	4
Breakout	44	31	-29	194	169	-13	126	25	-80
Centipede	5168	6727	30	11365	13358	18	9156	9299	2
ChopperCommand	1660	1779	7	3390	5043	49	2510	3250	29
CrazyClimber	58252	52172	-10	91103	87680	-4	72523	63597	-12
DemonAttack	437	939	115	1665	3693	122	855	2171	154
DoubleDunk	-16	-18	13	-6	-15	136	-7	-17	139
Enduro	164	138	-16	378	300	-21	327	249	-24
FishingDerby	-89	-85	-4	-78	-54	-31	-87	-69	-21
Freeway	21	23	10	29	27	-5	28	26	-8
Frostbite	261	1333	410	331	2669	706	282	2171	670
Gopher	1394	1290	-8	3481	3410	-2	2563	2976	16
Gravitar	319	462	45	800	1068	34	548	843	54
Hero	10143	9050	-11	14579	12946	-11	13371	11631	-13
IceHockey	-7	-7	3	-3	-3	-14	-6	-6	-1
Kangaroo	1258	3239	157	2290	8117	254	2143	7247	238
Krull	7401	6746	-9	9533	8714	-9	7520	7351	-2
KungFuMaster	21299	22626	6	30080	32393	8	26843	27873	4
MontezumaRevenge	0	0	138	7	17	150	0	0	0
MsPacman	1788	2317	30	4253	4868	14	2862	3704	29
NameThisGame	3293	3741	14	4854	5341	10	4126	4608	12
Phoenix	3485	4066	17	6521	8097	24	5447	6346	17
Pitfall	-22	-9	-59	0	0	0	-43	-2	-96
Pong	15	11	-26	21	19	-6	19	18	-4
PrivateEye	38	117	208	899	1429	59	67	80	20
Qbert	2544	2084	-18	6338	5254	-17	4327	4194	-3
RoadRunner	10692	12444	16	15093	16963	12	13117	15150	16
Robotank	6	12	102	16	23	47	10	20	96
Seaquest	1116	759	-32	1747	1174	-33	1647	943	-43
Skiing	-28495	-29793	5	-20406	-19642	-4	-27575	-29973	9
Solaris	1545	1629	5	3437	3689	7	843	2063	145
SpaceInvaders	456	402	-12	805	714	-11	579	537	-7
StarGunner	1043	2407	131	1343	6230	364	1057	3533	234
Tennis	-22	-20	-9	-13	-8	-39	-20	-19	-4
TimePilot	3321	3359	1	5227	5250	0	3880	3690	-5
Tutankham	139	169	21	176	251	43	159	224	41
Venture	2	268	12314	90	910	911	0	713	0
VideoPinball	19244	17873	-7	47860	41345	-14	17606	16594	-6
WizardOfWor	1177	1401	19	2820	3553	26	1540	2233	45
YarsRevenge	8987	11196	25	19083	33074	73	10153	18431	82
Zaxxon	64	4287	6650	1320	7720	485	27	6383	23838

Table A6: Scores on all 52 considered Atari games for the PPO baseline (B) and PPO with RUDDER (R) and the improvement by using RUDDER in percent (%). Agents are trained for 100M game frames (including skipped frames) with *no-op starting condition*, i.e. a random number of up to 30 no-operation actions at the start of each game. Episodes are prematurely terminated if a maximum of 108K frames is reached. Scoring metrics are (a) *average*, the average reward per completed game throughout training, which favors fast learning [104], (b) *maximum*, the maximum of the averages over 10 consecutive games during training, which favors exploration and the best performing model, and (c) *final*, the average over the last 10 consecutive games at the end of training, which favors consistency in learning.

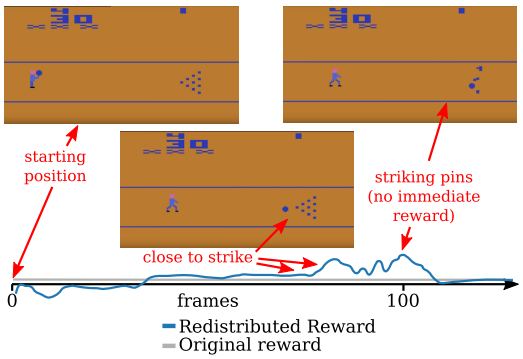


Figure A12: Return decomposition by RUDDER in the Atari game Bowling, where RUDDER redistributes rewards to key events. This game is characterized by long delayed rewards, since reward is only given after multiple rolls at the end of the episode. RUDDER identifies the actions that push the ball in the right direction to hit all pins. The first 100 frames of this 200 frames episode are depicted, where reward is given only at the end.

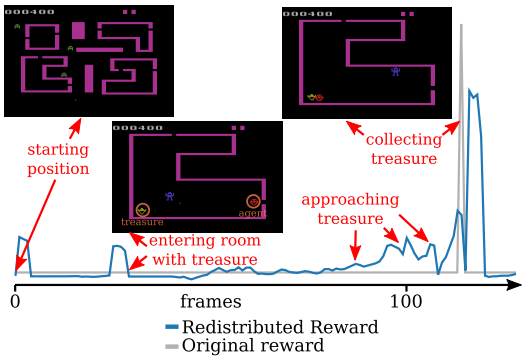


Figure A13: In the Atari game Venture, reward is only obtained after the treasure is collected. RUDDER steers the agent (red) towards the treasure (golden) via reward redistribution, where reward is redistributed to the event of entering a room with treasure. Furthermore, the redistributed reward gradually increases when the agent approaches the treasure. For illustration purposes, the blue curve shows the reward redistribution. The environment only gives reward at the event of collecting the treasure (grey).

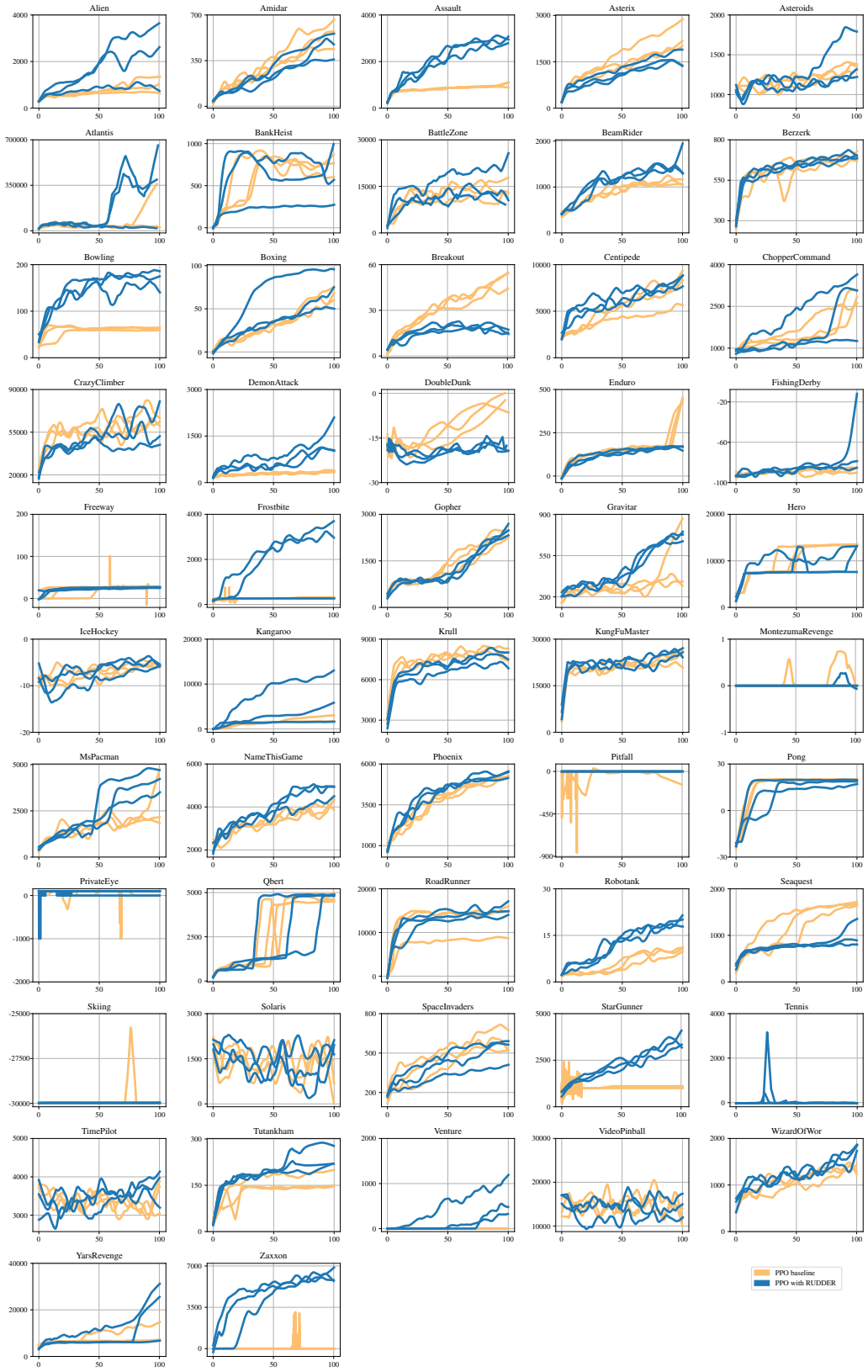


Figure A14: Training curves for PPO baseline and PPO with RUDDER over 100M game frames, 3 runs with different random seeds each. Curves show scores during training of a single agent that does not use safe exploration, smoothed using Locally Weighted Scatterplot Smoothing (y-value estimate using 10% of data with 10 residual-based re-weightings).

A8 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] F. Beleznay, T. Grobler, and C. Szepesvári. Comparing value-function estimation algorithms in undiscounted problems. Technical Report TR-99-02, Mindmaker Ltd., 1999.

[7] 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.

[8] 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.

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

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

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

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

[13] 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.

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

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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).

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

[22] 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.

[23] 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.

[24] 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.

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

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

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

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

[34] 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.

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

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

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

[38] P. Hernandez-Leal, B. Kartal, and M. E. Taylor. Is multiagent deep reinforcement learning the answer or the question? A brief survey. *ArXiv*, 2018.

[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] C. Hung, T. Lillicrap, J. Abramson, Y. Wu, M. Mirza, F. Carnevale, A. Ahuja, and G. Wayne. Optimizing agent behavior over long time scales by transporting value. *ArXiv*, 2018.

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

[53] 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.

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

[55] G. H. John. When the best move isn't optimal: q-learning with exploration. In *Proceedings of the 10th Tenth National Conference on Artificial Intelligence, Menlo Park, CA, 1994. AAAI Press.*, page 1464, 1994.

[56] 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.

[57] 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.

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

[59] 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.

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

[61] 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.

[62] T. Lattimore and C. Szepesvá. *Bandit Algorithms*. Cambridge University Press, 2018. Draft of 28th July, Revision 1016.

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

[64] 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.

[65] 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.

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

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

[68] 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.

[69] 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.

[70] 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.

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

[72] 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.

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

[74] 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.

[75] 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.

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

[77] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[78] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 30 (NIPS'16)*, pages 1054–1062, 2017.

[79] 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.

[80] 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.

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

[82] 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.

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

[84] 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.

[85] 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.

[86] 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.

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

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

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

[90] 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.

[91] A. C. Rencher and G. B. Schaalje. *Linear Models in Statistics*. John Wiley & Sons, Hoboken, New Jersey, 2 edition, 2008. ISBN 978-0-471-75498-5.

[92] M. A. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. VandeWiele, V. Mnih, N. Heess, and J. T. Springenberg. Learning by playing — solving sparse reward tasks from scratch. *ArXiv*, 2018.

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

[94] 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.

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

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

[97] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, 1994.

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

[99] 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.

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

[101] 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.

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

[103] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *ArXiv*, 2015. Fourth International Conference on Learning Representations (ICLR'16).

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

[105] 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.

[106] 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.

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

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

[109] 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.

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

[111] P.-H. Su, D. Vandyke, M. Gasic, N. Mrksic, T.-H. Wen, and S. Young. Reward shaping with recurrent neural networks for speeding up on-line policy learning in spoken dialogue systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 417–421. Association for Computational Linguistics, 2015.

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

[113] 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 (NIPS'13)*, pages 3104–3112. Curran Associates, Inc., 2014.

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

[115] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

[116] 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.

[117] 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.

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

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

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

[121] 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.

[122] 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.

[123] 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.

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

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

[126] 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.

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

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

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

[130] 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.

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

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

[133] 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.