RUDDER: Return Decomposition for Delayed Rewards

J. A. Arjona-Medina*[1 2], M. Gillhofer*[1 2], M. Widrich*[1 2], T. Unterthiner[1 2], J. Brandstetter[1 2], S. Hochreiter[1 2 3]

[1]LIT AI Lab [2]Institute for Machine Learning, JKU Linz [3]Institute of Advanced Research in Artificial Intelligence (IARAI)

Abstract. RUDDER is a novel reinforcement learning approach for delayed rewards in finite Markov decision processes (MDPs). In MDPs the Q-values are equal to the expected immediate reward plus the expected future rewards, which are related to bias problems in temporal difference (TD) learning and to high variance problems in Monte Carlo (MC) learning. Both problems are even more severe when rewards are delayed. RUDDER aims at making the **expected future rewards equal to zero**, which simplifies Q-value estimation to computing the mean of the immediate reward. We propose the following two new concepts to push the expected future rewards toward zero. **Reward redistribution** that leads to return-equivalent decision processes with the same optimal policies and, when optimal, zero expected future rewards. **Return decomposition** via contribution analysis which transforms the reinforcement learning task into a regression task at which deep learning excels. On artificial tasks with delayed rewards, RUDDER is significantly faster than MC and exponentially faster than Monte Carlo Tree Search (MCTS), TD(λ), and reward shaping approaches.

Intuition. Assume you have high reward and average reward episodes. Supervised learning identifies state-actions that are indicative for high rewards. Adjust the policy so that these state-actions are reached more often. Reward redistribution to these state-actions achieves these adjustments. For *delayed rewards* and *model-free* reinforcement learning:

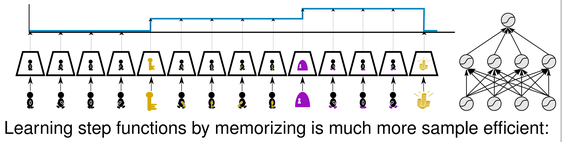
▶ Do not use value functions that are based on state-actions. The expected return has to be predicted from every state-action. A single prediction error might hamper learning. Instead *identify relevant state-actions* across the whole episode.

▶ Do not use temporal difference (TD). It suffers from vanishing information even with eligibility traces.

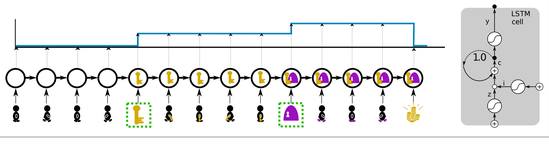
▶ Do not use Monte Carlo (MC). Averaging over all possible futures leads to high variance. Model-based MCTS worked for GO and chess.

Basic Insight: Value Functions are Step Functions. Complex tasks are hierarchical with sub-tasks or sub-goals. A step in the value function is a change in return expectation: amount or probability to obtain. Steps indicate achievements, failures, or change of environment/information. Identifying large steps is important since they speed up learning tremendously: (i) large increase of return and (ii) sampling more relevant episodes. In this example an agent has to take a key to open the door to a treasure:

Both getting the key and opening the door increases the probability of obtaining the treasure. Learning step functions by fully connected networks requires to extract the expected return from every state-action:

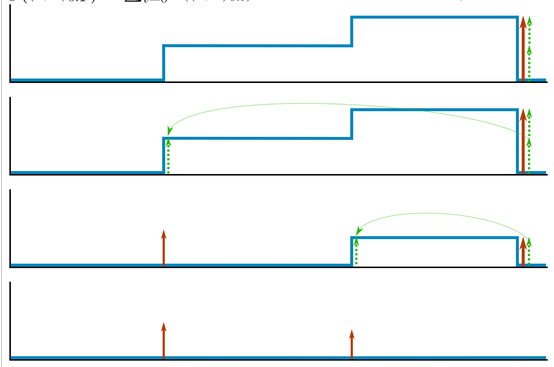


Learning step functions by memorizing is much more sample efficient:



Reward Redistribution

Reward Redistribution: Idea and Return decomposition. ▶ Reward redistribution is not potential-based reward shaping.◀ We consider reward redistributions that are obtained by return decomposition given an episodic MDP $\tilde{\mathcal{P}}$ with $\tilde{R}_{t+1} = 0$ for $t < T$. The value function is assumed to be a step function (blue curve below). **Return decomposition identifies the steps of the value function** (green arrows below). A function g that predicts the expected return $\mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right]$ (big red arrow below) given the sequence $(s,a)_{0:T} = (s_0, a_0, \ldots, s_T, a_T)$ is decomposed into steps $h_t = h((s,a)_{0:t})$ (green arrows below): $g\big((s,a)_{0:T}\big) = \sum_{t=0}^{T} h((s,a)_{0:t})$. For most t, we assume $h_t = 0$.



Explaining away problem. Return R_{T+1} is predicted from s_T without using reward causing earlier states or actions. We replace state-actions by mutually independent differences $\Delta_t = \Delta(s_{t-1}, a_{t-1}, s_t, a_t)$. With $g(\Delta_{0:T}) = \mathrm{E}\left[\tilde{R}_{T+1} \mid s_T, a_T\right]$ and contributions $h_t = h(\Delta_t)$, the redistributed rewards R_{t+1} (small red arrows above) fulfill:

$$g(\Delta_{0:T}) \;=\; \mathrm{E}_\pi\left[\tilde{G}_0\right] \;=\; \sum_{t=0}^{T} h_t \;, \quad \mathrm{E}\left[R_{t+1} \mid s_{t-1}, a_{t-1}, s_t, a_t\right] \;=\; h_t \;.$$

For an SDP $\tilde{\mathcal{P}}$ obtained from an optimal reward redistribution (later) the expected future reward is zero (blue curve at zero above). A return decomposition leads to an optimal reward redistribution (see below) if $g(\Delta_{0:t}) = \mathrm{E}_\pi\left[\tilde{R}_{T+1} \mid s_t, a_t\right] = \sum_{\tau=0}^{t} h_\tau$

Reward Redistribution: Basic Concepts. Given is a finite horizon Markov decision process (MDP) $\tilde{\mathcal{P}}$ with states s, actions a, policy π, episode length T, discount factor $\gamma = 1$, and delayed rewards $\tilde{R}_t$ with $\tilde{R}_{t+1} = 0$ for $t < T$. The return variable is $\tilde{G}_0 = \sum_{t=0}^{T} \tilde{R}_{t+1}$. Optimal reward redistribution transforms this MDP into an SDP $\mathcal{P}$ without delayed rewards but the same optimal policies as the original $\tilde{\mathcal{P}}$. The expected future rewards are zero.

Definition 1. *A* **sequence-Markov decision process (SDP)** *is an MDP where the reward distributions do not have to be Markov.*

▶ Q-values for SDPs are defined by averaging also over the past.

Definition 2. *Given an SDP $\tilde{\mathcal{P}}$, a* **reward redistribution** *is a fixed procedure that redistributes for each episode either the realization or the expectation of the return variable $\tilde{G}_0$ along the state-action sequence. For the return variable G_0 of the new SDP $\mathcal{P}$ with reward R_t holds*

$$\mathrm{E}_\pi\left[G_0 \mid s_0, a_0, \ldots, s_T, a_T\right] \;=\; \mathrm{E}_\pi\left[\tilde{G}_0 \mid s_0, a_0, \ldots, s_T, a_T\right] \;.$$

Theorem 1. *If the SDP $\mathcal{P}$ is obtained by reward redistribution from the SDP $\tilde{\mathcal{P}}$, then both SDPs have the same optimal policies.*

Reward Redistribution: Optimality.

Definition 3. *A reward redistribution is* **optimal** *if $\kappa(T-t-1, t) = 0$ for $0 \leqslant t \leqslant T-1$, where the* **expected future reward** *$\kappa(m, t-1)$ at $(t-1)$ and $0 \leqslant m \leqslant T-t$ is $\kappa(m, t-1) = \mathrm{E}_\pi\left[\sum_{\tau=0}^{m} R_{t+1+\tau} \mid s_{t-1}, a_{t-1}\right]$.*

The Bellman equation becomes $q^\pi(s_t, a_t) = r(s_t, a_t) + \kappa(T-t-1, t)$. However, κ cannot be expressed by q^π. In general, an optimal reward redistribution violates the Markov assumptions, therefore we use SDPs. The next theorem states that an optimal reward redistribution exists and that it is only second order Markov.

Theorem 2. *We assume a delayed reward MDP $\tilde{\mathcal{P}}$ with episodic reward. A new SDP $\mathcal{P}$ is obtained by a second order Markov reward redistribution. Given π, the following two statements are equivalent:*

(I) $\kappa(T-t-1, t) = 0$ *i.e. the reward redistribution is optimal,*

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

The optimal reward redistribution is second order Markov since the expectation of R_{t+1} depends on $(s_{t-1}, a_{t-1}, s_t, a_t)$.

Theorem 3. *If the reward redistribution for the original MDP $\tilde{\mathcal{P}}$ is optimal, then the Q-values of the SDP $\mathcal{P}$ are given by*

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

The SDP $\mathcal{P}$ and the MDP $\tilde{\mathcal{P}}$ have the same advantage function. Using a behavior policy $\tilde{\pi}$ the expected immediate reward is

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

Reward Redistribution: Novel Learning Algorithms.

▶ Q-value estimation approaches:

- Estimate $q(s_t, a_t) \;=\; \tilde{q}^\pi(s_t, a_t) \;-\; \psi^\pi(s_t)$ (assumes optimality)
- Correct $q(s_t, a_t)$ by estimating κ (assumes non-optimality)
- Use eligibility traces on κ (assumes non-optimality)

▶ Policy Gradients approaches: replace $q^\pi(s,a)$ by an estimate or a sample of $r(s,a)$ in $\mathrm{E}_\pi\left[\nabla_\theta \log \pi(a \mid s;\theta) q^\pi(s,a)\right]$
The offset $\psi^\pi(s)$ reduces the variance as baseline normalization does.

▶ Q-learning is justified if immediate and future reward are drawn together, as typically done.

RUDDER — Reward Redistribution Framework:

▶ Safe exploration: Avoiding low Q-values during exploration interval.

▶ Lessons replay buffer for training LSTM: Episodes with unseen delayed rewards (large prediction errors) go to lessons replay buffer.

▶ LSTM return decomposition: An LSTM learns to predict sequence-wide return at every time step. Return decomposition uses differences of return predictions as redistributed rewards.

Demonstrations. They fill the lessons replay buffer, e.g. episodes from human experts. Reward redistribution to key actions in demonstrations.

Alternatives to LSTM.

▶ attention methods and transformers.

▶ sequence alignment techniques known from bioinformatics, if only few positive examples are available.

Limitations.

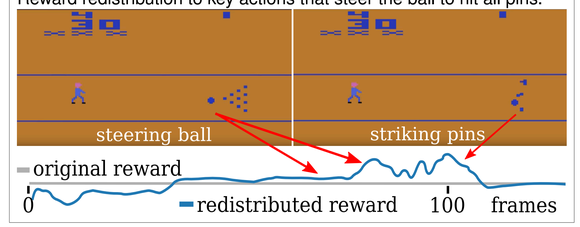
▶ ineffective if reward is not delayed since LSTM learning takes time.

▶ problems with very long sequences.

▶ reward redistribution may introduce disturbing spurious rewards.

Results

Atari game Bowling:
Reward redistribution to key actions that steer the ball to hit all pins.



Artificial Tasks.

Comparison of RUDDER and other methods on artificial tasks with respect to the learning time in episodes (median of 100 trials) vs. the delay of the reward. The shadow bands indicate the 40% and 60% quantiles. Task (I) shows that TD methods have problems with vanishing information for delayed rewards. Task (II) shows that MC methods have problems with the high variance of future unrelated rewards. The y-axis of the inlet is scaled by 10^5. Task (III) shows that potential-based reward shaping methods have problems with delayed rewards. Reward shaping (RS), look-ahead advice (look-ahead), and look-back advice (look-back) use three different potential functions. The dashed blue line is RUDDER with $Q(\lambda)$, vs. RUDDER with Q-estimation. In all tasks, RUDDER significantly outperforms all other methods.

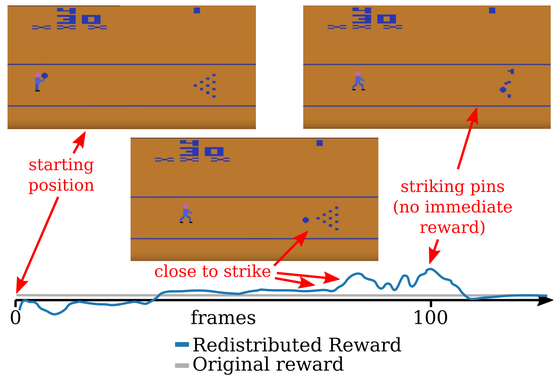
Atari Games. RUDDER is evaluated with respect to scores and learning time on Atari games of the Arcade Learning Environment (ALE) and OpenAI Gym. RUDDER is used on top of the TRPO-based policy gradient method PPO that uses GAE. We use a PPO baseline with reward scaling instead of the sign-functions and RUDDER's safe exploration for fair comparisons. A coarse hyperparameter optimization is performed for the PPO baseline. For all 52 Atari games, RUDDER uses the same architectures, losses, and hyperparameters, which were optimized for the baseline. Difference to the PPO baseline: redistributed reward instead of the original.

	RUDDER	baseline	delay	delay-event
Bowling	192	56	200	strike pins
Solaris	1,827	616	122	navigate map
Venture	1,350	820	150	find treasure
Seaquest	4,770	1,616	272	collect divers

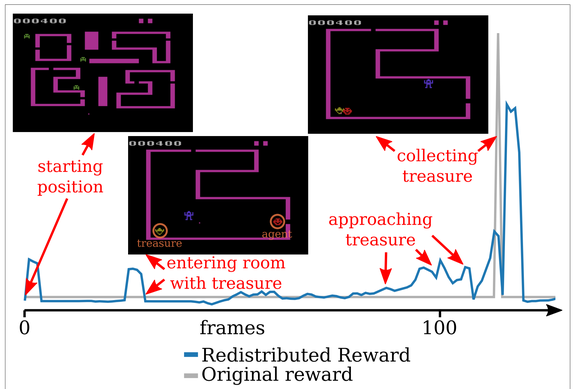
Average scores over 3 random seeds with 10 trials. "delay": frames between reward and first related action. RUDDER considerably improves the PPO baseline on delayed reward games.

▶ RUDDER blog: https://www.jku.at/index.php?id=16426
▶ Code: https://github.com/ml-jku/rudder
▶ A practical step-by-step guide for RUDDER in PyTorch:
https://github.com/widmi/rudder-a-practical-tutorial

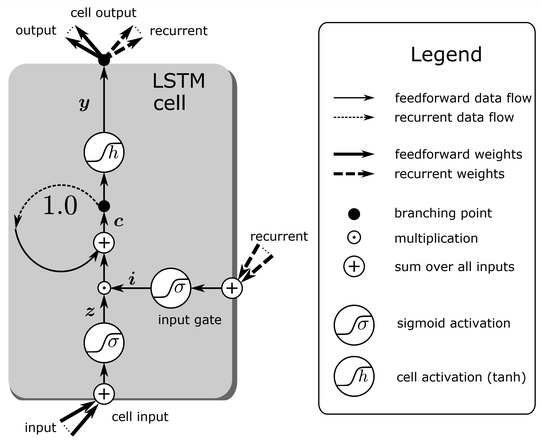
Visual Confirmation of Detecting Relevant Events by Reward Redistribution. We visually confirm a meaningful and helpful redistribution of reward in both Bowling and Venture during training. As illustrated, RUDDER is capable of redistributing a reward to key events in a game, drastically shortening the delay of the reward and quickly steering the agent toward good policies. Furthermore, it enriches sequences that were sparse in reward with a dense reward signal. Video demonstrations are available at https://goo.gl/EQerZV



In the game Bowling, reward is only given after a turn which consist of multiple rolls. RUDDER identifies the actions that guide the ball in the right direction to hit all pins. Once the ball hit the pins, RUDDER detects the delayed reward associated with striking the pins down. In the figure only 100 frames are represented but the whole turn spans more than 200 frames. In the original game, the reward is given only at the end of the turn.

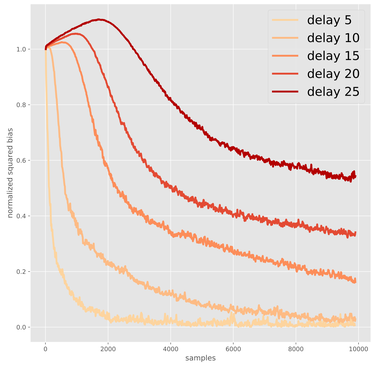


Venture: reward is only obtained after picking up the treasure. RUDDER guides the agent (red) towards the treasure (golden) via reward redistribution. Reward is redistributed to entering a room with treasure. Furthermore, reward is redistributed to step towards the treasure. The environment only gives reward at the event of collecting the treasure.

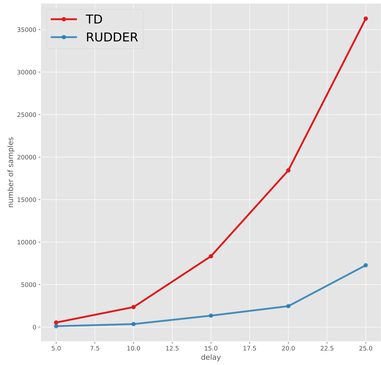


RUDDER is implemented with an LSTM architecture without output gate nor forget gate to simplify the network dynamics. Forget gates and output gates can modify all cell inputs at times after they have been observed, which can make the dynamics highly nonlinear.

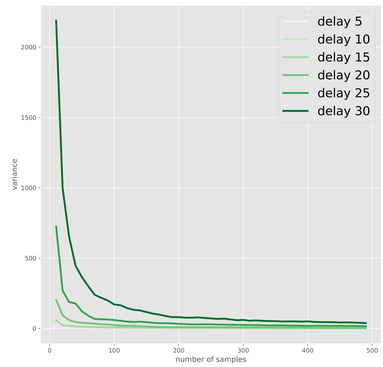
Identity output activation functions were chosen to support the development of linear counting dynamics within the LSTM layer, as is required to count the reward pieces during an episode chunk. Furthermore, the input gate is only connected recurrently to other LSTM blocks and the cell input is only connected to forward connections from the lower layer.



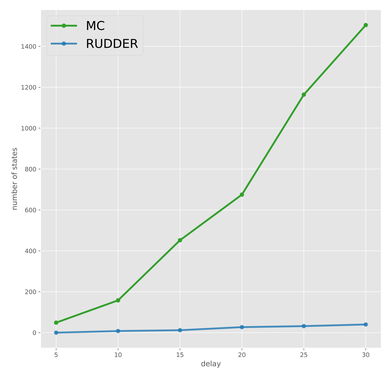
Experimental evaluation of bias and variance of different Q-value estimators on the Grid World.



Experimental evaluation of bias of TD and RUDDER Q-value estimators on the Grid World. Number of samples to reduce the bias by 80% of the original error for different delays.



Average variance reduction for the 10th highest values.



Experimental evaluation of variance of MC and RUDDER Q-value estimators on the Grid World. The number of states affected by high variance grows exponentially with the delay.

	average			*final*		
	baseline	RUDDER	%	baseline	RUDDER	%
Alien	1,878	3,087	64.4	3,218	5,703	77.3
Amidar	787	724	-8.0	1,242	1,054	-15.1
Assault	5,788	4,242	-26.7	10,373	11,305	9.0
Asterix	10,554	18,054	71.1	29,513	102,930	249
Asteroids	22,065	4,905	-77.8	310,505	154,479	-50.2
Atlantis	1,399,753	1,655,464	18.3	3,568,513	3,641,583	2.0
BankHeist	936	1,194	27.5	1,078	1,335	23.8
BattleZone	12,870	17,023	32.3	24,667	28,067	13.8
BeamRider	2,372	4,506	89.9	3,994	6,742	68.8
Berzerk	1,261	1,341	6.4	1,930	2,092	8.4
Bowling	61.5	179	191	56.3	192	241
Boxing	98.0	94.7	-3.4	100	99.5	-0.5
Breakout	217	153	-29.5	430	352	-18.1
Centipede	25,162	23,029	-8.5	53,000	36,383	-31.4
ChopperCommand	6,183	5,244	-15.2	10,817	9,573	-11.5
CrazyClimber	125,249	106,076	-15.3	140,080	132,480	-5.4
DemonAttack	28,684	46,119	60.8	464,151	400,370	-13.7
DoubleDunk	-9.2	-13.1	-41.7	-0.3	-5.1	-1,825
Enduro	759	777	2.5	2,201	1,339	-39.2
FishingDerby	19.5	11.7	-39.9	52.0	36.3	-30.3
Freeway	26.7	25.4	-4.8	32.0	31.4	-1.9
Frostbite	3,172	4,770	50.4	5,092	7,439	46.1
Gopher	8,126	4,090	-49.7	102,916	23,367	-77.3
Gravitar	1,204	1,415	17.5	1,838	2,233	21.5
Hero	22,746	12,162	-46.5	32,383	15,068	-53.5
IceHockey	-3.1	-1.9	39.4	-1.4	1.0	171
Kangaroo	2,755	9,764	254	5,360	13,500	152
Krull	9,029	8,027	-11.1	10,368	8,202	-20.9
KungFuMaster	49,377	51,984	5.3	66,883	78,460	17.3
MontezumaRevenge	0.0	0.0	38.4	0.0	0.0	0.0
MsPacman	4,096	5,005	22.2	6,446	6,984	8.3
NameThisGame	8,390	10,545	25.7	10,962	17,242	57.3
Phoenix	15,013	39,247	161	46,758	190,123	307
Pitfall	-8.4	-5.5	34.0	-75.0	0.0	100
Pong	19.2	18.5	-3.9	21.0	21.0	0.0
PrivateEye	102	34.1	-66.4	100	33.3	-66.7
Qbert	12,522	8,290	-33.8	28,763	16,631	-42.2
RoadRunner	20,314	27,992	37.8	35,353	36,717	3.9
Robotank	24.9	32.7	31.3	32.2	47.3	46.9
Seaquest	1,105	2,462	123	1,616	4,770	195
Skiing	-29,501	-29,911	-1.4	-29,977	-29,978	0.0
Solaris	1,393	1,918	37.7	616	1,827	197
SpaceInvaders	778	1,106	42.1	1,281	1,860	45.2
StarGunner	6,346	29,016	357	18,380	62,593	241
Tennis	-13.5	-13.5	0.2	-4.0	-5.3	-32.8
TimePilot	3,790	4,208	11.0	4,533	5,563	22.7
Tutankham	123	151	22.7	140	163	16.3
Venture	738	885	20.1	820	1,350	64.6
VideoPinball	19,738	19,196	-2.7	15,248	16,836	10.4
WizardOfWor	3,861	3,024	-21.7	6,480	5,950	-8.2
YarsRevenge	46,707	60,577	29.7	109,083	178,438	63.6
Zaxxon	6,900	7,498	8.7	12,120	10,613	-12.4

Scores on all 52 considered Atari games for the PPO baseline and PPO with RUDDER and the improvement by using RUDDER in percent. Agents are trained for 200M 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 and (b) *final*, the average over the last 10 consecutive games at the end of training, which favors consistency in learning. Scores are shown for one agent without safe exploration.