Modern Hopfield Networks for Sample-Efficient Return Decomposition from Demonstrations

Michael Widrich[*]
ELLIS Unit Linz and LIT AI Lab
Institute for Machine Learning
Johannes Kepler University Linz, Austria

Markus Hofmarcher[*]
ELLIS Unit Linz and LIT AI Lab
Institute for Machine Learning
Johannes Kepler University Linz, Austria

Vihang Patil
ELLIS Unit Linz and LIT AI Lab
Institute for Machine Learning
Johannes Kepler University Linz, Austria

Angela Bitto-Nemling
ELLIS Unit Linz and LIT AI Lab
Institute for Machine Learning
Johannes Kepler University Linz, Austria

Sepp Hochreiter[†‡]
[†]ELLIS Unit Linz and LIT AI Lab
Institute for Machine Learning
Johannes Kepler University Linz, Austria
[‡]Institute of Advanced Research in Artificial Intelligence (IARAI)

Abstract

Delayed rewards, which are separated from their causative actions by irrelevant actions, hamper learning in reinforcement learning (RL). Especially real world problems often contain such delayed and sparse rewards. Recently, *return decomposition for delayed rewards* (RUDDER) employed pattern recognition to remove or reduce delay in rewards, which dramatically simplifies the learning task of the underlying RL method. RUDDER was realized using a long short-term memory (LSTM). The LSTM was trained to identify important state-action pair patterns, responsible for the return. Reward was then redistributed to these important state-action pairs. However, training the LSTM is often difficult and requires a large number of episodes. This can be especially problematic in real-world and offline RL settings with limited numbers of episodes. In this work, we replace the LSTM with the recently proposed continuous modern Hopfield networks (MHN) and introduce Hopfield-RUDDER. MHN are powerful trainable associative memories with large storage capacity. They require only few training samples and excel at identifying and recognizing patterns. We use this property of MHN to identify important state-action pairs that are associated with low or high return episodes and directly redistribute reward to them. However, in partially observable environments, Hopfield-RUDDER requires additional information about the history of state-action pairs. Therefore, we evaluate several methods for compressing history and introduce *reset-max history*, a lightweight history compression using the max-operator in combination with a reset gate. We experimentally show that Hopfield-RUDDER is able to outperform LSTM-based RUDDER on various 1D environments with small numbers of episodes. Finally, we show in preliminary experiments that Hopfield-RUDDER scales to highly complex environments with the Minecraft *ObtainDiamond* task from the MineRL NeurIPS challenge.

[*]Authors contributed equally

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2021

Introduction

Recent advances in reinforcement learning (RL) have resulted in impressive models that are capable of surpassing humans in games [30, 18, 27]. However, RL is still waiting for its breakthrough in real world applications, which are often characterized by delayed and sparse rewards [7]. Delayed rewards are given later than their causative action, separated by irrelevant state-action pairs [28]. This delay hampers and slows down learning [2, 21, 17]. Recently, [2] propose *return decomposition for delayed rewards* (RUDDER) to use pattern recognition to detect the causative actions, which can be used to remove or alleviate these delays. As a consequence, RUDDER dramatically simplifies and speeds up learning of the underlying RL algorithms. RUDDER can be realized in different ways, including a long short-term memory (LSTM) [10] in [2] or sequence alignment in [19].

But these realizations of RUDDER come with drawbacks. LSTMs require a large number of samples, while sequence alignment is difficult to scale to large state-action spaces. Thus, there is a need for scalable and sample efficient realization of RUDDER, especially in settings with limited numbers of episodes, such as in real world applications and offline RL.

A key idea of RUDDER is to identify important patterns in the state-action sequence. This is a task that associative memories, such as the recently introduced continuous modern Hopfield networks (MHN) [23], excel at. In this work, we introduce Hopfield-RUDDER, a realization of RUDDER based on MHN to identify important state-action pairs. We show that Hopfield-RUDDER is scalable and can reduce the reward delay even with a small number of samples.

In related work, adapted Transformer [29] architectures, which share characteristics of MHN, for RL have been proposed. In [6], RL is reformulated as a conditional sequence modelling task for *offline* RL that can be learned by a *Decision Transformer*. [14] uses a *linear Transformer* [15] as an outer-product-based fast-weight programmer [24, 25]. However, in contrast to Hopfield-RUDDER, these approaches require large amounts of data, as they are based on Transformer architectures. Furthermore, Hopfield-RUDDER can speed up any learning algorithm that relies on a delayed reward signal.

Our main contributions are: We (i) introduce modern Hopfield networks for return decomposition for delayed rewards (Hopfield-RUDDER), (ii) propose and compare different methods, such as *reset-max history*, for history compression for Hopfield-RUDDER in POMDP environments, (iii) show that Hopfield-RUDDER outperforms LSTM-based RUDDER on 1D environments with small numbers of samples, and (iv) perform preliminary experiments in the MineRL Minecraft environment indicating that Hopfield-RUDDER scales to large and complex environments.

Review

We define our setting as a finite Markov decision process (MDP) or a finite partially observable Markov decision process (POMDP) to be a 4-tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ of finite sets $\mathcal{S}$ with states s (random variable S_t at time t), $\mathcal{A}$ with actions a (random variable A_t at time t), $\mathcal{R}$ with rewards r (random variable R_t at time t), and state-reward transition distribution $p(S_{t+1} = s_{t+1}, R_{t+1} = r_t \mid S_t = s_t, A_t = a_t)$. The return of a sequence of length T at time $t = 1, \ldots, T$ is defined as $g_t = \sum_{k=0}^{T-t} r_{t+k}$.

RUDDER. Complex tasks are often hierarchically composed of sub-tasks. Hence, the Q-function of an optimal policy often resembles a step function [2]. Such steps indicate patterns like achievements, failures, accomplished sub-tasks, or changes of the environment. RUDDER reduces the delay in the rewards by identifying these patterns and moving (redistributing) the rewards to the causative state-action patterns. [2] propose an LSTM as realization of RUDDER. The LSTM predicts the return at the end of an episode as early as possible. These predictions $\widehat{g}$ are then used to redistribute the reward: $r_t = \widehat{g}_t \quad \widehat{g}_{t-1}$.

The redistributed reward serves as reward for a subsequent learning method and can be used to optimize a policy. However, training an LSTM network requires a large amount of episodes, which are often difficult or expensive to obtain.

Modern Hopfield networks. Hopfield networks are energy-based, binary associative memories, which popularized artificial neural networks in the 1980s [11, 12]. Associative memory networks have been designed to store and retrieve samples. Their storage capacity can be considerably increased by polynomial terms in the energy function [5, 20, 3, 8, 1, 13, 4, 16]. In contrast to these binary memory

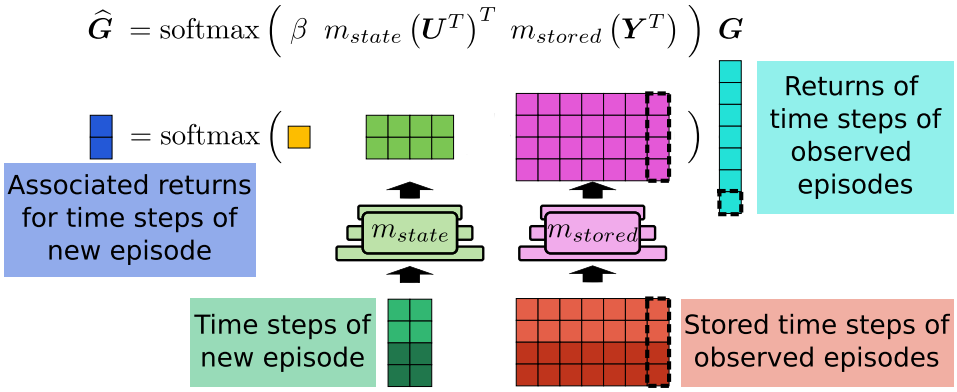

Figure 1: Illustration of modern Hopfield networks for return decomposition in Hopfield-RUDDER. The associative memory of modern Hopfield networks associates 2 time steps (raw state patterns) $\boldsymbol{U}$ of a new episode with 7 time steps (stored patterns) $\boldsymbol{Y}$ of previously observed episodes and their corresponding observed returns $\boldsymbol{G}$. Thereby, the state patterns $\boldsymbol{U}$ are associated with estimated returns $\widehat{\boldsymbol{G}}$. $\boldsymbol{U}$ and $\boldsymbol{Y}$ are transformed via m_{state} and m_{stored} before association, respectively, and can include information about previous state-action pairs, such as via reset-max history. Dotted rectangles show a single raw and transformed stored pattern with its observed return.

networks, we use continuous associative memory networks with very high storage capacity. These modern Hopfield networks for deep learning architectures have an energy function with continuous states and can retrieve samples with only one update [23, 22]. Modern Hopfield Networks have already been successfully applied to immune repertoire classification [31] and chemical reaction prediction [26]. There they serve as powerful associative memories with large memory capacity and excel at pattern-recognition as they associate inputs with similar patterns in their memory.

Hopfield-RUDDER

The application of MHN for RUDDER naturally follows from the idea of identifying patterns: Patterns, e.g. state-action pairs, of observed episodes with known returns are stored in the memory of the MHN. A new pattern can then be used to query this memory for similar patterns and their returns. This yields an estimated return of the new pattern, which can be used to redistribute the rewards.

Fig. 1 illustrates this in detail: The MHN associates N *stored patterns* $\boldsymbol{Y} = f\boldsymbol{y}_i g_{i=1}^N$ from previously observed episodes and their corresponding observed returns $\boldsymbol{G} = f g_i g_{i=1}^N$ with M new *state patterns* $\boldsymbol{U} = f\boldsymbol{u}_i g_{i=1}^M$. As such, it directly associates the unknown state patterns $\boldsymbol{U}$ with estimated returns $\widehat{\boldsymbol{G}} = f\widehat{g}_i g_{i=1}^M$. β is a hyper-parameter controlling the temperature of the softmax function.

The raw state patterns $\boldsymbol{U}$ and stored patterns $\boldsymbol{Y}$ may be transformed by functions m_{state} and m_{stored}, for example with linear mappings or multiple hidden layers, as in [23, 31].

This results in a return decomposition function ψ in the form of

$$\widehat{\boldsymbol{G}} = \psi(\boldsymbol{Y}, \boldsymbol{G}, \boldsymbol{U}; \beta) \ = \ \text{softmax}\left(\beta\ m_{state}\left(\boldsymbol{U}^T\right)^T\ m_{stored}\left(\boldsymbol{Y}^T\right)\right)\ \boldsymbol{G}. \tag{1}$$

The redistributed reward r_t at time t can then be obtained by computing the differences of the estimated return values as

$$r_t = \widehat{g}_t \quad \widehat{g}_{t-1}. \tag{2}$$

For state patterns $\boldsymbol{u}_t$ and $\boldsymbol{u}_{t-1}$ at times t and t 1 this results in

$$r_t \ = \ \psi(\boldsymbol{Y}, \boldsymbol{G}, \boldsymbol{u}_t; \beta) \quad \psi(\boldsymbol{Y}, \boldsymbol{G}, \boldsymbol{u}_{t-1}; \beta), \tag{3}$$

where $\boldsymbol{Y}$ contains a representation of the stored state-action pairs $f(s_i, a_i) g_{i=1}^N$ with their known corresponding returns $\boldsymbol{G}$ and $\boldsymbol{u}_t$ represents a state-action pair (s_t, a_t) without known return.

In the simplest case, a raw state pattern $\boldsymbol{u}$ or stored pattern $\boldsymbol{y}$ is a vector containing the observation s_t and action a_t at time t in an episode. The estimated return for a single state pattern $\boldsymbol{u}_t$ at time t in

this case is $\widehat{g}_t = \psi(\boldsymbol{Y}, \boldsymbol{G}, s_t, a_t; \beta)$ with the redistributed reward r_t computed from state patterns $\boldsymbol{u}_{t-1}$ and $\boldsymbol{u}_t$ as $r_t = \widehat{g}_t \quad \widehat{g}_{t-1} = \psi(\boldsymbol{Y}, \boldsymbol{G}, s_t, a_t; \beta) \quad \psi(\boldsymbol{Y}, \boldsymbol{G}, s_{t-1}, a_{t-1}; \beta)$.

History compression and reset-max history. Assume a key event at time t_1, e.g. the collection of a key, that can result in a reward at time t_2, e.g. the opening of a chest. In a POMDP, this key event might only be observable at t_1 but not at t_2, e.g. the observation space contains no information whether a key has been collected already. Assume further that observation s_t and action a_t are used as state or stored pattern: The association of the pattern at t_2 with a return, which depends on whether a key was collected at t_1, may severely suffer from the missing information.

To address this problem, the state and stored patterns can be augmented by the history of the previous state-action pairs. However, retaining the complete history of a state-action pair and including it in the state and stored patterns would not be feasible for environments with large observation spaces or long episode sequences.

As the purpose of this history is to enable the detection and storage of key events, we propose a fast and light-weight history compression method: the *reset-max history*. Assuming a vector $\boldsymbol{v}^o{}_t \in \mathbb{R}^{K \times 1}$ with K features that contains a representation of the state-action pair (s_t, a_t) at time $t = 1, \ldots, T$, the reset-max history features $\boldsymbol{v}^h{}_t \in \mathbb{R}^{J \times 1}$ with J features, and the final state or stored pattern $\boldsymbol{v}_t \in \mathbb{R}^{(J+K) \times 1}$ is computed as follows:

$$\boldsymbol{v}_t = \left[\boldsymbol{v}^h{}_t; \boldsymbol{v}^o{}_t\right] \; ; \quad \boldsymbol{v}^h{}_t = \max_{i=1,\ldots,(t-1)} \left(\boldsymbol{v}^o{}_i\right) \quad f_{reset}\left(\boldsymbol{v}^h{}_{t-1}, \boldsymbol{v}^o{}_t\right) \tag{4}$$

with $\boldsymbol{v}^h{}_0$ initialized with 1 and

$$f_{reset}\left(\boldsymbol{v}^h{}_{t-1}, \boldsymbol{v}^o{}_t\right) = \sigma\left(\boldsymbol{W}\left[\boldsymbol{v}^h{}_{t-1}; \boldsymbol{v}^o{}_t\right]\right), \tag{5}$$

where $[;]$ is the operator for vertical concatenation, the $\max$ operator is used to store the previously observed maximum feature values, and the reset gate f_{reset} utilizes the sigmoid activation function σ and learned weight matrix $\boldsymbol{W} \in \mathbb{R}^{J \times (J+K)}$ to reset the stored maximum feature values. Notably, this changes the return decomposition function ψ for a state pattern $\boldsymbol{u}_t$ to be dependent on all previous and current state-action pairs $(s_i, a_i)_{i=1}^t$ in the episode: $\widehat{g}_t = \psi(\boldsymbol{Y}, \boldsymbol{G}, s_{1,\ldots,t}, a_{1,\ldots,t}; \beta, \boldsymbol{W})$.

Experiments

In this section, we first compare different history compression methods on multiple 1D toy environments. Subsequently, we compare the performance of Hopfield-RUDDER to the originally proposed LSTM-RUDDER. Finally, we visually analyse the reward redistribution of Hopfield-RUDDER on the difficult and complex task of collecting a diamond in the MineRL Minecraft environment. We note that all experiments are performed in an offline fashion, using random policies or human demonstrations.

1D key-chest environment. We designed 16 different versions of a 1D environment, the *1D key-chest environment*, in order to evaluate the performance of Hopfield-RUDDER, different history compression methods, and to compare Hopfield-RUDDER to LSTM-RUDDER. We use a reward redistribution score rr_score, which is based on the known optimal policy, for this evaluation and comparison. For more details see App. A1.

Environment details. The 1D key-chest environment is illustrated in Fig. 2. The agent starts at position *s* in the middle. At each position it can move either one position to the right or to the left, except for the left-most and right-most position, where further movement to the left and right, respectively, is ignored. If the agent visits position *k*, it collects 1 *key*. If the agent visits position *s*, it loses all collected *keys* with a probability of p_l. If the agent visits position *c* while holding n_k *keys*, it will receive 1 reward at the end of the episode. All episodes have the same fixed length. A reward of 0 or 1 is given at the end of each episode, depending on whether the chest was opened or not.

We evaluate the performance of RUDDER on different versions of this environment with $p_l = \{0, 0.5\}$ and $n_k = \{1, 3\}$. We consider fully observable MDP and partially observable POMDP versions of the environment. Furthermore, we add $n_{rnd} = \{0, n_{obs}\}$ features that contain random values $\{0, 1\}$ to the observation space, where n_{obs} is the number of features of the original observation space. In total, we created 16 different versions of this environment (see Tab. 1 and App. A1).

To redistribute reward in this environment successfully, RUDDER has to identify that position k needs to be visited n_k times before visiting position c. Furthermore, RUDDER has to identify that the *key*-collection process has to be restarted if the *keys* are lost at s.

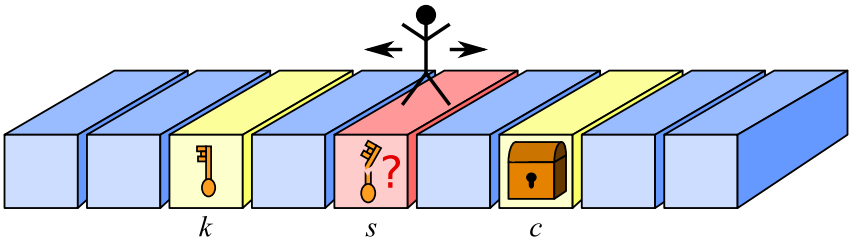

Figure 2: Illustration of 1D key-chest environment with special states k, s, c, and 9 states in total.

Reward redistribution score By design, the optimal of the two available actions at any state s in the 1D key-chest environment is known. In particular, at each state s_t at time t in an episode, the agent can either take a left or right action $a_t \in \{a_l, a_r\}$, which can be a correct, incorrect, or irrelevant action. We use this knowledge to compute a score rr_score for the quality of the reward redistribution $\psi(:)$ for an episode with time $t = 1, \ldots, T$ as follows:

$$rr\_score = 0.5 + \frac{1}{2T} \sum_{t=1}^{T} scr(s_i, a_i)_{i=1}^{t-1}, s_t), \tag{6}$$

where $scr(:)$ assigns a score $\{-1, 0, 1\}$ for the redistributed reward r_t at each time t.

In detail, action left a_l is correct if $q^*(s_t, a_l) > q^*(s_t, a_r)$. Action right a_r is correct if $q^*(s_t, a_r) > q^*(s_t, a_l)$. An action is irrelevant if $q^*(s_t, a_l) == q^*(s_t, a_r)$. Here, $q^*(s_t, a_t)$ is the Q-function of the optimal policy π^*.

The redistributed reward $r_t = \psi(:)$ is correct if r_t for the correct action is higher than for the incorrect action. If the redistributed reward is correct, $scr(:)$ assigns a score of 1, if it is incorrect a score of -1. If the action is irrelevant, the reward redistribution receives a score of 0.

Experimental setup. We sample a large test set of $1,000$ episodes and small training sets of $\{8, 16, 32, 64, 128, 256, 512\}$ episodes, using a random agent. Half of the episodes have a return of 0 and 1 in each set, with fixed sequence lengths between 32 and 148 times per episode (see App. A1).

Reset-max history outperforms other history compression methods. We first analyze the performance of Hopfield-RUDDER, where the training set is used as stored patterns. For this, we evaluate Hopfield-RUDDER without history information and with 5 different versions of history compression: (i) feature-wise max-pooling, (ii) feature-wise sum-pooling, (iii) fully-connected LSTM (LSTMf), (iv) sparsely-connected LSTM (LSTMs), and (v) reset-max history, as detailed in App. A1.2. Furthermore, we evaluate each setting with a linear mapping $m_{state} = m_{stored}$ and without any mapping. As shown in Tab. 1 and A2, reset-max history consistently outperforms all other Hopfield-RUDDER versions, except for one POMDP setting and two MDP settings, in which it is the runner-up method.

Hopfield-RUDDER with reset-max history consistently outperforms LSTM-RUDDER. Subsequently, we analyze the performance of LSTM-RUDDER with reset-max history and without history information. As shown in Tab. 1, Hopfield-RUDDER with reset-max history consistently outperforms all LSTM-RUDDER versions on all environment versions.

More details on results, history compression methods, and training can be found in App. A1.

Minecraft. In order to show that Hopfield-RUDDER can scale to complex problems, we perform preliminary experiments within the challenging MineRL Minecraft environment [9]. Specifically, we use Hopfield-RUDDER to redistribute rewards for the task `ObtainDiamond`. As the high complexity of `ObtainDiamond` makes random exploration unfeasible, [9] provide the Minecraft environment with demonstrations of human players solving this task. Players and agents are randomly placed in a

				Hopfield-RUDDER														LSTM-RUDDER			
History:				reset-max		LSTMf		LSTMs		max		sum		none		reset-max		none			
Mapping:				linear	none	linear	none	linear	none	linear	none	linear	none	linear	none	none	none	none	none		
Learning rate:				$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 4	$1e$ 3	$1e$ 4	$1e$ 3		
Environment specs																					
	n_k	p_l	n_{rnd}																		
POMDP	1	0%	0	**93.08**	82.42	72.23	70.37	76.94	66.79	91.85	89.96	78.72	41.36	58.30	58.61	65.27	72.85	57.25	74.32		
				4.87	8.80	7.12	6.40	6.65	10.29	5.40	5.74	7.41	6.09	3.24	3.85	6.69	5.97	4.41	5.93		
POMDP	1	0%	n_{obs}	87.66	81.37	66.17	61.51	75.31	62.36	**87.90**	84.31	76.91	41.92	56.90	57.54	62.23	73.37	53.84	68.23		
				5.43	4.88	5.17	5.60	7.56	6.42	6.74	5.23	6.67	4.15	3.59	2.35	8.37	4.04	5.03	7.34		
POMDP	1	50%	0	**75.41**	73.57	58.22	58.75	62.77	59.93	62.85	65.52	52.80	49.28	54.83	55.99	60.21	55.25	48.85	52.11		
				10.33	13.01	7.55	7.72	9.41	11.43	7.57	7.03	6.61	2.10	6.01	5.66	4.60	8.43	7.76	7.52		
POMDP	1	50%	n_{obs}	**69.19**	66.19	54.87	54.98	56.33	54.33	61.47	60.13	53.07	48.90	54.43	54.64	51.34	51.80	48.79	50.14		
				7.95	5.52	4.37	3.84	5.00	6.34	6.31	3.84	3.50	1.36	4.01	3.43	6.32	5.01	2.59	4.34		
POMDP	3	0%	0	**80.91**	74.06	66.61	64.94	76.06	73.80	77.02	77.59	68.16	45.17	54.88	55.31	59.50	66.62	55.37	63.68		
				7.74	8.43	8.17	6.98	8.44	13.78	8.81	6.17	12.70	5.25	4.03	4.08	5.75	7.37	6.15	9.36		
POMDP	3	0%	n_{obs}	**77.96**	74.07	63.42	61.90	73.32	68.99	75.30	75.06	66.61	45.56	54.56	54.41	53.34	65.84	51.84	58.34		
				5.95	4.84	4.95	4.01	6.65	8.06	6.29	4.19	10.50	3.73	3.33	3.42	6.20	7.19	7.31	6.18		
POMDP	3	50%	0	**74.30**	69.60	56.17	58.38	59.50	62.42	57.42	60.44	54.61	55.36	54.84	56.58	52.78	52.80	47.96	49.51		
				10.68	11.35	7.57	8.99	11.27	13.61	7.22	5.95	4.72	1.59	7.44	6.88	2.72	7.65	9.88	6.34		
POMDP	3	50%	n_{obs}	**67.30**	63.82	53.07	53.06	55.86	54.81	55.79	55.46	55.05	55.55	53.92	53.39	52.64	50.51	46.40	49.93		
				8.50	6.76	4.72	3.48	5.84	8.43	5.70	3.25	2.70	0.81	4.82	3.99	5.68	4.72	3.47	4.91		

Table 1: Comparison of different Hopfield-RUDDER and LSTM-RUDDER versions w.r.t. reward redistribution score rr_score on different versions of the 1D key-chest environment. Results show the mean rr_score over all training set sizes and a 10-fold cross-validation (CV). Error bars show mean standard deviation of 10-fold CV over all training set sizes. Hopfield-RUDDER with reset-max history consistently outperforms all LSTM-RUDDER versions. For MDP versions see Tab. A2.

procedurally generated 3D environment without any items in their possession. The objective is to gather resources and build the tools required to obtain a diamond.

For `ObtainDiamond`, auxiliary reward is given the first time the player obtains an item, even if multiple copies of this item are required. For example, agents have to gather multiple logs in order to build all necessary tools but receive reward only for the first log they acquire. However, we omit these auxiliary rewards and only use episodic return by giving a return of 1 for demonstrations that obtain a diamond and a return of 0 for those that do not manage to obtain it.

In order to show that Hopfield-RUDDER is able to redistribute reward to relevant state-action pairs in `ObtainDiamond`, we train a simple Hopfield-RUDDER model using the *reset-max history* history compression. We only use the sequence of inventory states and associated actions as input to Hopfield-RUDDER. Following [19], we use 10 successful and 10 unsuccessful demonstration episodes as training set. The state-action pairs from these 20 episodes are used as stored patterns $\boldsymbol{Y}$. We use the obfuscated version of the environment and demonstrations, therefore the observations s and actions a are the hidden representation of an unknown auto encoder (for further details see App. A2.3). We use a neural network with 2 hidden layers and ReLU activation as mapping functions $m_{state} = m_{stored}$.

Training Hopfield-RUDDER does not require massive compute resources. On the contrary, training a model for 100 episodes takes only minutes on an Nvidia A40 GPU. A small hyper-parameter search showed that the model is not overly sensitive to hyper-parameter changes and most models produce a similar reward redistribution.

Fig. 3 shows the redistribution for a demonstration episode, which was not used for training the model. We use the non-obfuscated inventory states for visualization. It visually seems that Hopfield-RUDDER is able to redistribute the reward to important state-action pairs. Furthermore, Hopfield-RUDDER appears to not redistribute rewards to irrelevant state-action pairs, resulting in a reward redistribution with little noise.

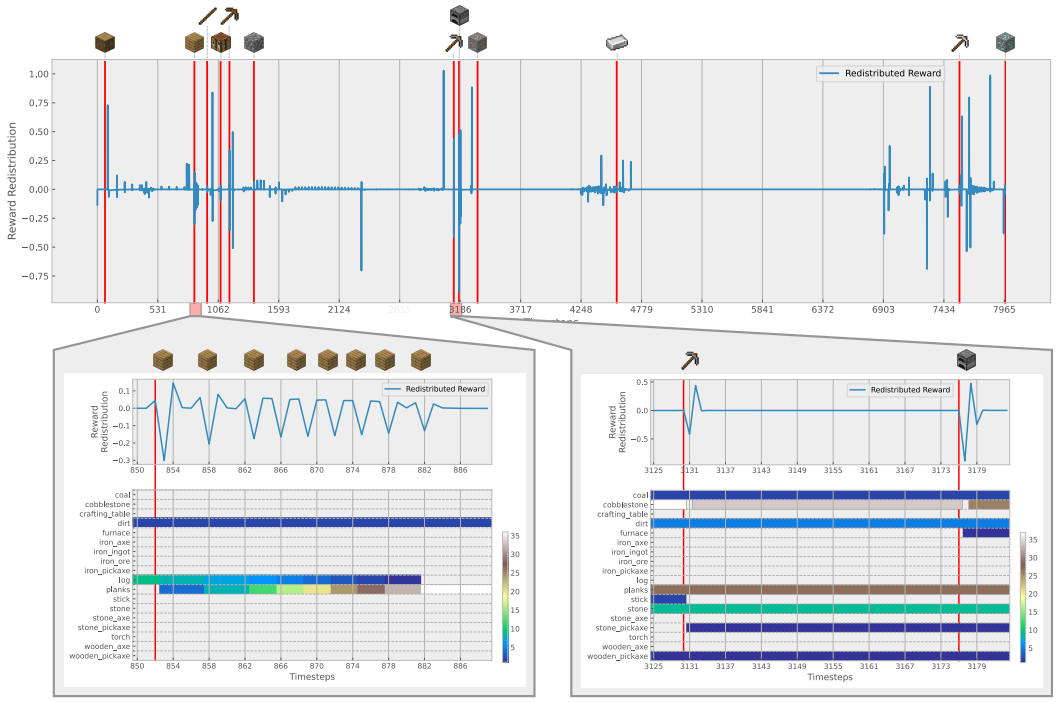

Figure 3: **Top:** example of redistributed reward of a successful demonstration using only the inventory and actions. Red vertical lines indicate the auxiliary reward an agent would receive from the environment, however the reward redistribution is trained only with reward at episode end. The reward redistribution seems to identify relevant state-action pairs. **Bottom left:** Magnified region showing acquisition of *planks* through crafting. First, logs are used up, resulting in negative redistributed reward followed by an increase in planks, which increases the reward prediction. **Bottom right:** Time steps showing the crafting of *stone pickaxe* and *furnace*. Both items require stone and show a similar behavior as crafting planks. Predicted reward decreases when the inventory state for base component decreases and increases when the crafted item appears in the inventory.

Conclusion

We have introduced Hopfield-RUDDER, a novel realization of RUDDER. Hopfield-RUDDER replaces the LSTM in the original LSTM-RUDDER with a powerful trainable auto-associative memory, the modern Hopfield network. In contrast to LSTM-RUDDER, Hopfield-RUDDER can be trained on a drastically lower number of samples, such as often the case in offline RL settings.

Acknowledgments and Disclosure of Funding

The ELLIS Unit Linz, the LIT AI Lab, the Institute for Machine Learning, are supported by the Federal State Upper Austria. IARAI is supported by Here Technologies. We thank the projects AI-MOTION (LIT-2018-6-YOU-212), DeepToxGen (LIT-2017-3-YOU-003), AI-SNN (LIT-2018-6-YOU-214), DeepFlood (LIT-2019-8-YOU-213), Medical Cognitive Computing Center (MC3), INCONTROL-RL (FFG-881064), PRIMAL (FFG-873979), S3AI (FFG-872172), DL for GranularFlow (FFG-871302), AIRI FG 9-N (FWF-36284, FWF-36235), ELISE (H2020-ICT-2019-3 ID: 951847), AIDD (MSCA-ITN-2020 ID: 956832). We thank Janssen Pharmaceutica (MaDeSMart, HBC.2018.2287), Audi.JKU Deep Learning Center, TGW LOGISTICS GROUP GMBH, Silicon Austria Labs (SAL), FILL Gesellschaft mbH, Anyline GmbH, Google, ZF Friedrichshafen AG, Robert Bosch GmbH, UCB Biopharma SRL, Merck Healthcare KGaA, Verbund AG, Software Competence Center Hagenberg GmbH, TÜV Austria, and the NVIDIA Corporation.

References

[1] L. F. Abbott and Y. Arian. Storage capacity of generalized networks. *Phys. Rev. A*, 36:5091–5094, 1987.

[2] J. A. Arjona-Medina*, M. Gillhofer*, M. Widrich*, T. Unterthiner, J. Brandstetter, and S. Hochreiter. RUDDER: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, pages 13544–13555, 2019.

[3] P. Baldi and S. S. Venkatesh. Number of stable points for spin-glasses and neural networks of higher orders. *Phys. Rev. Lett.*, 58:913–916, 1987.

[4] B. Caputo and H. Niemann. Storage capacity of kernel associative memories. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, page 51–56, Berlin, Heidelberg, 2002. Springer-Verlag.

[5] H. H. Chen, Y. C. Lee, G. Z. Sun, H. Y. Lee, T. Maxwell, and C. Lee Giles. High order correlation model for associative memory. *AIP Conference Proceedings*, 151(1):86–99, 1986.

[6] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

[7] F. Dulac-Arnold, D. J. Mankowitz, and T. Hester. Challenges of real-world reinforcement learning. *CoRR*, abs/1904.12901, 2019.

[8] E. Gardner. Multiconnected neural network models. *Journal of Physics A*, 20(11):3453–3464, 1987.

[9] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. P. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, M. Veloso, and P. Wang. The MineRL competition on sample efficient reinforcement learning using human priors. *arXiv*, 2019.

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

[11] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[12] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81(10):3088–3092, 1984.

[13] D. Horn and M. Usher. Capacities of multiconnected memory models. *J. Phys. France*, 49(3):389–395, 1988.

[14] K. Irie, I. Schlag, R. Csordás, and J. Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *arXiv preprint arXiv:2106.06295*, 2021.

[15] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[16] D. Krotov and J. J. Hopfield. Dense associative memory for pattern recognition. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 1172–1180. Curran Associates, Inc., 2016.

[17] 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.

[18] 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.

[19] V. P. Patil, M. Hofmarcher, M.-C. Dinu, M. Dorfer, P. M. Blies, J. Brandstetter, J. A. Arjona-Medina, and S. Hochreiter. Align-rudder: Learning from few demonstrations by reward redistribution. *CoRR*, abs/2009.14108, 2020.

[20] D. Psaltis and H. P. Cheol. Nonlinear discriminant functions and associative memories. *AIP Conference Proceedings*, 151(1):370–375, 1986.

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

[22] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, D. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter. Hopfield networks is all you need. *ArXiv*, 2008.02217, 2020.

[23] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, D. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter. Hopfield networks is all you need. In *9th International Conference on Learning Representations (ICLR)*, 2021.

[24] J. Schmidhuber. Learning To Control Fast-Weight Memories: An Alternative To Dynamic Recurrent Networks, 1991.

[25] J. Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139, 1992.

[26] P. Seidl, P. Renz, N. Dyubankova, P. Neves, J. Verhoeven, J. K. Wegner, S. Hochreiter, and G. Klambauer. Modern hopfield networks for few- and zero-shot reaction prediction. *ArXiv*, 2104.03279, 2021.

[27] 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.

[28] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1984.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[30] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing. Starcraft II: A new challenge for reinforcement learning. *ArXiv*, 2017.

[31] M. Widrich, B. Schäfl, M. Pavlović, H. Ramsauer, L. Gruber, M. Holzleitner, J. Brandstetter, G. K. Sandve, V. Greiff, S. Hochreiter, and G. Klambauer. Modern Hopfield networks and attention for immune repertoire classification. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18832–18845. Curran Associates, Inc., 2020.

Appendix

A1 1D key-chest environment

In this section we provide details on the specifications, the setup, and the results on the 1D key-chest environment described in the paper. The 16 environment versions and their settings are listed in Tab. A1.

Parameters	Environment versions							
observations	POMDP	POMDP	POMDP	POMDP	POMDP	POMDP	POMDP	POMDP
n_k	1	1	1	1	3	3	3	3
p_l	0%	0%	50%	50%	0%	0%	50%	50%
n_{rnd}	0	n_{obs}	0	n_{obs}	0	n_{obs}	0	n_{obs}
time steps	32	32	99	99	48	48	148	148
Parameters	Environment versions							
observations	MDP	MDP	MDP	MDP	MDP	MDP	MDP	MDP
n_k	1	1	1	1	3	3	3	3
p_l	0%	0%	50%	50%	0%	0%	50%	50%
n_{rnd}	0	n_{obs}	0	n_{obs}	0	n_{obs}	0	n_{obs}
time steps	32	32	99	99	48	48	148	148

Table A1: Specifications of the 16 versions of the 1D key-chest environment, as illustrated in Fig. 2. Number of time steps per episode are fixed and chosen such that a random agent generates $35\%(\ \ 2\%)$ episodes with a return of 1. Reward $\{0, 1\}$ is given only at the end of an episode. Training and test sets were sampled until 50% of episodes with 0 and 1 return were obtained. n_{rnd} is the number of random features with values $\{0, 1\}$ that are concatenated to the observation space, where n_{obs} is the number of features of the original observation space.

A1.1 MDP and POMDP observation space

For the fully observable MDP versions of the environment, the observation space includes information about the position, time step, number of currently held *keys*, if the agent is on position *k* or *c*, if the agent visited *c* while holding n_k *keys*, and, in case $p_l > 0$, if the current *keys* are being lost on *s*.

In the POMDP versions, the observation space only contains information if the agent is on position *k* or *c* and, in case $p_l > 0$, if the current *keys* are being lost on *s*.

Furthermore, we add $n_{rnd} = \{0, n_{obs}\}$ features that contain random values $\{0, 1\}$ to the observation space, where n_{obs} is the number of features of the original observation space. All features are binary, except for the position and time step feature,

A1.2 History compression methods

We evaluated 5 methods for history compression. Following the notation in the paper, they are:
(i) **feature-wise max-pooling:** $\boldsymbol{v}_t = \left[\boldsymbol{v^h}_t; \boldsymbol{v^o}_t\right]$; $\boldsymbol{v^h}_t = \max_{i=1,\dots,(t-1)} (\boldsymbol{v^o}_i)$,
(ii) **feature-wise sum-pooling:** $\boldsymbol{v}_t = \left[\boldsymbol{v^h}_t; \boldsymbol{v^o}_t\right]$; $\boldsymbol{v^h}_t = \sum_{i=1,\dots,(t-1)} (\boldsymbol{v^o}_i)$,

(iii) **fully-connected LSTM (LSTMf):** $\boldsymbol{v}_t = \left[{\boldsymbol{v}^h}_t; {\boldsymbol{v}^o}_t\right]$; ${\boldsymbol{v}^h}_t = \mathrm{LSTMf}_{i=1,\ldots,(t-1)}\left({\boldsymbol{v}^o}_i\right)$, with a layer of J fully-connected LSTM blocks,
(iv) **sparsely-connected LSTM (LSTMs):** $\boldsymbol{v}_t = \left[{\boldsymbol{v}^h}_t; {\boldsymbol{v}^o}_t\right]$; ${\boldsymbol{v}^h}_t = \mathrm{LSTMs}_{i=1,\ldots,(t-1)}\left({\boldsymbol{v}^o}_i\right)$, with a layer of J sparsely-connected LSTM blocks, where output gate, input gate, and the recurrent connections of the cell input are disabled, and
(v) **reset-max history**, as described in the main paper,
where J is the number of observation features.

A1.3 Training details

Training of all Hopfield-RUDDER, history methods, and LSTM-RUDDER versions was performed in PyTorch [34] using the Adam optimizer [33] for $10,000$ updates. Weights of the linear mappings are shared such that $m_{state} == m_{stored}$. For training of Hopfield-RUDDER, a mini-batch of 4 random samples from the training set is used as state patterns and the rest of the training set is used as stored patterns for each weight update. For LSTM-RUDDER, mini-batches of 4 random samples from the training set are used for each weight update.

For Hopfield-RUDDER, first the history features are computed from the observations. Then, if a linear mapping for Hopfield-RUDDER is used, the state and stored patterns are mapped to 16 features via a single linear layer before the association with the known episode returns is performed. The bias weights for the LSTM-based history versions and the reset-max history were initialized with random values from a normal distribution with a mean of 5.

For LSTM-RUDDER, we chose a number of 16 LSTM blocks in the network, followed by a single output layer to create the reward prediction at every time step of the episode. The LSTM loss combines the losses $L_m + 0.1\,(L_c + L_e)$, as described by [2], formula A275. Loss L_a was omitted, as there are no intermediate rewards given in the episodes.

A1.4 Detailed results and ablation study for history versions

In the following Fig. A1 and A2 , we show the detailed results for the different history versions.

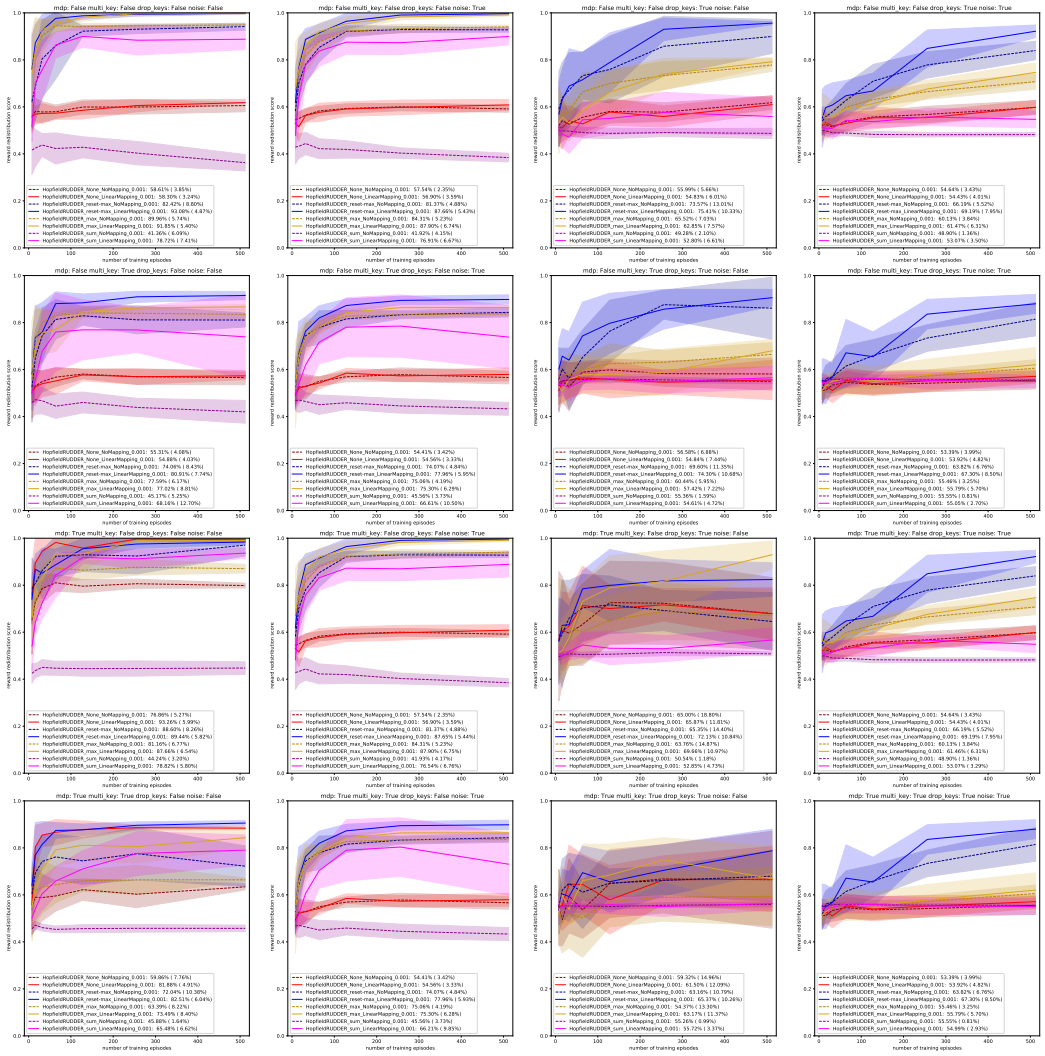

Figure A1: Comparison of Hopfield-RUDDER max-pooling, sum-pooling, and reset-max history w.r.t. reward redistribution score rr_score on different versions of the 1D key-chest environment. Results shown are the mean rr_score over 10-fold cross-validation with their corresponding standard deviations for various training set sizes.

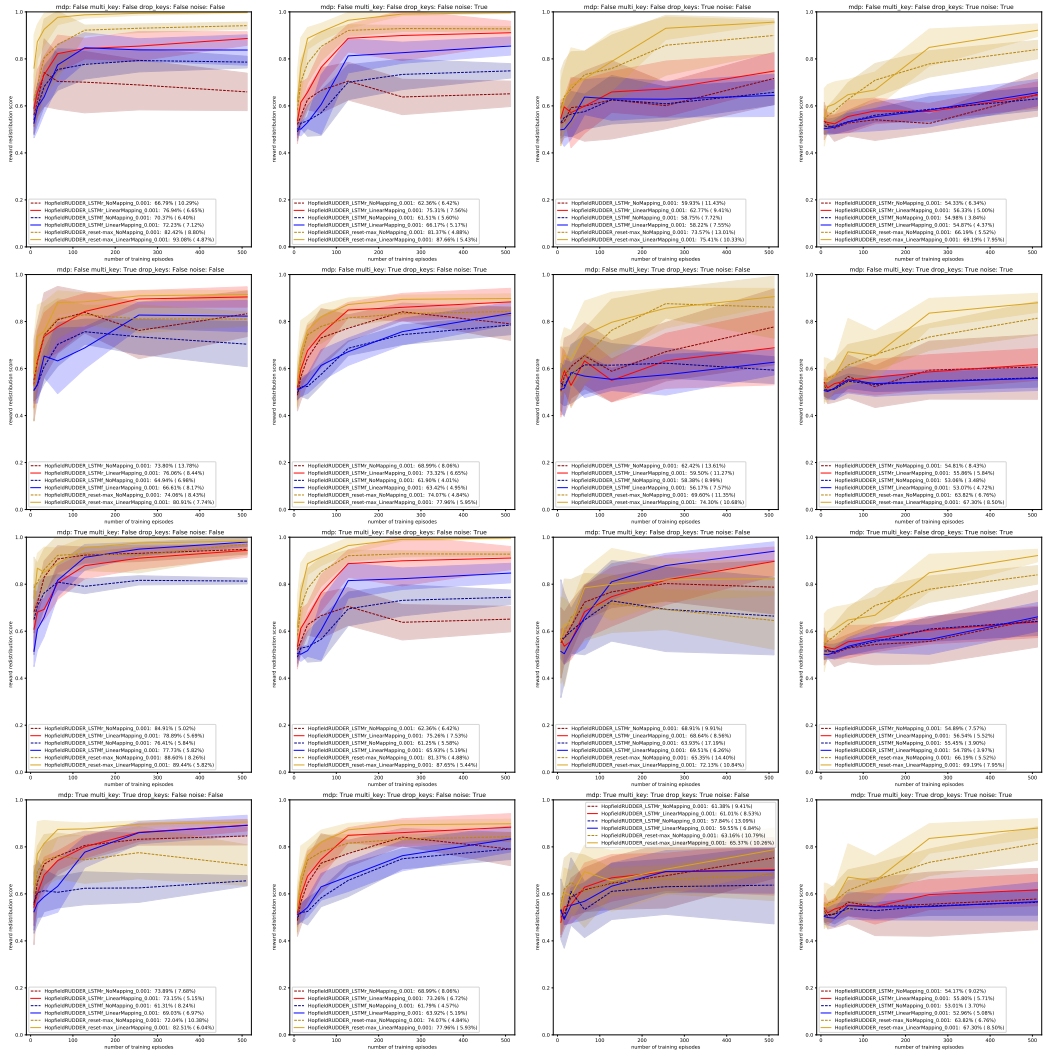

Figure A2: Comparison of Hopfield-RUDDER LSTM-based history and reset-max history w.r.t. reward redistribution score rr_score on different versions of the 1D key-chest environment. Results shown are the mean rr_score over 10-fold cross-validation with their corresponding standard deviations for various training set sizes.

A1.5 Detailed results for comparison of Hopfield-RUDDER and LSTM-RUDDER.

In the following Fig. A3, we show the detailed results for comparison of Hopfield-RUDDER with reset-max history versus the different versions of LSTM-RUDDER.

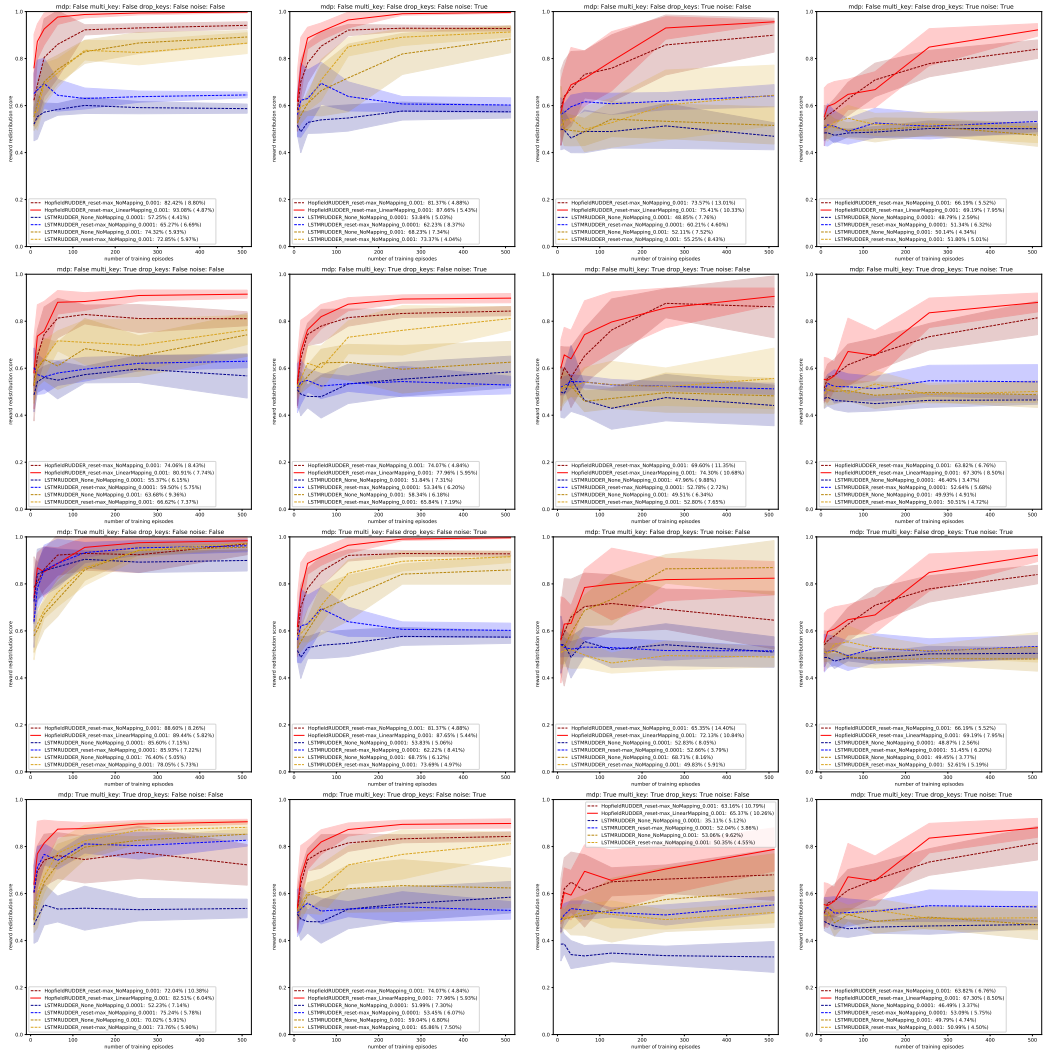

Figure A3: Comparison of Hopfield-RUDDER with reset-max history and LSTM-RUDDER versions w.r.t. reward redistribution score rr_score on different versions of the 1D key-chest environment. Results shown are the mean rr_score over 10-fold cross-validation with their corresponding standard deviations for various training set sizes.

A1.6 Detailed results for comparison of all methods.

In the following Tab. A2, we show the detailed results for comparison of all Hopfield-RUDDER and LSTM-RUDDER versions.

				Hopfield-RUDDER												LSTM-RUDDER			
History:				reset-max		LSTMf		LSTMr		max		sum		none		reset-max		none	
Mapping:				linear	none	linear	none	linear	none	linear	none	linear	none	linear	none	none	none	none	none
Learning rate:				$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 3	-	$1e$ 4	$1e$ 3	$1e$ 4	$1e$ 3
Environment specs																			
	n_k	p_l	n_{rnd}																
POMDP	1	0%	0	**93.08**	82.42	72.23	70.37	76.94	66.79	91.85	89.96	78.72	41.36	58.30	58.61	65.27	72.85	57.25	74.32
				4.87	8.80	7.12	6.40	6.65	10.29	5.40	5.74	7.41	6.09	3.24	3.85	6.69	5.97	4.41	5.93
POMDP	1	0%	n_{obs}	87.66	81.37	66.17	61.51	75.31	62.36	**87.90**	84.31	76.91	41.92	56.90	57.54	62.23	73.37	53.84	68.23
				5.43	4.88	5.17	5.60	7.56	6.42	6.74	5.23	6.67	4.15	3.59	2.35	8.37	4.04	5.03	7.34
POMDP	1	50%	0	**75.41**	73.57	58.22	58.75	62.77	59.93	62.85	65.52	52.80	49.28	54.83	55.99	60.21	55.25	48.85	52.11
				10.33	13.01	7.55	7.72	9.41	11.43	7.57	7.03	6.61	2.10	6.01	5.66	4.60	8.43	7.76	7.52
POMDP	1	50%	n_{obs}	**69.19**	66.19	54.87	54.98	56.33	54.33	61.47	60.13	53.07	48.90	54.43	54.64	51.34	51.80	48.79	50.14
				7.95	5.52	4.37	3.84	5.00	6.34	6.31	3.84	3.50	1.36	4.01	3.43	6.32	5.01	2.59	4.34
POMDP	3	0%	0	**80.91**	74.06	66.61	64.94	76.06	73.80	77.02	77.59	68.16	45.17	54.88	55.31	59.50	66.62	55.37	63.68
				7.74	8.43	8.17	6.98	8.44	13.78	8.81	6.17	12.70	5.25	4.03	4.08	5.75	7.37	6.15	9.36
POMDP	3	0%	n_{obs}	**77.96**	74.07	63.42	61.90	73.32	68.99	75.30	75.06	66.61	45.56	54.56	54.41	53.34	65.84	51.84	58.34
				5.95	4.84	4.95	4.01	6.65	8.06	6.29	4.19	10.50	3.73	3.33	3.42	6.20	7.19	7.31	6.18
POMDP	3	50%	0	**74.30**	69.60	56.17	58.38	59.50	62.42	57.42	60.44	54.61	55.36	54.84	56.58	52.78	52.80	47.96	49.51
				10.68	11.35	7.57	8.99	11.27	13.61	7.22	5.95	4.72	1.59	7.44	6.88	2.72	7.65	9.88	6.34
POMDP	3	50%	n_{obs}	**67.30**	63.82	53.07	53.06	55.86	54.81	55.79	55.46	55.05	55.55	53.92	53.39	52.64	50.51	46.40	49.93
				8.50	6.76	4.72	3.48	5.84	8.43	5.70	3.25	2.70	0.81	4.82	3.99	5.68	4.72	3.47	4.91
MDP	1	0%	0	89.44	88.60	77.73	76.41	78.89	84.91	87.66	81.16	78.82	44.24	**93.26**	76.86	85.93	78.05	85.60	76.40
				5.82	8.26	5.82	5.84	5.69	5.02	6.54	6.77	5.80	3.20	5.99	5.27	7.22	5.73	7.15	5.05
MDP	1	0%	n_{obs}	87.65	81.37	65.93	61.25	75.26	62.36	**87.90**	84.31	76.54	41.93	56.90	57.54	62.22	73.69	53.83	68.75
				5.44	4.88	5.19	5.58	7.53	6.42	6.75	5.23	6.76	4.17	3.59	2.35	8.41	4.97	5.06	6.12
MDP	1	50%	0	**72.13**	65.35	69.51	63.93	68.64	68.91	69.66	63.76	52.85	50.54	65.87	65.00	52.66	49.83	52.83	68.71
				10.84	14.40	6.26	17.19	8.56	9.91	10.97	14.87	4.73	1.18	11.81	18.80	3.79	5.91	8.05	8.16
MDP	1	50%	n_{obs}	**69.19**	66.19	54.78	55.45	56.54	54.89	61.46	60.13	53.07	48.90	54.43	54.64	51.45	52.61	48.87	49.45
				7.95	5.52	3.97	3.90	5.52	7.57	6.31	3.84	3.29	1.36	4.01	3.43	6.20	5.19	2.56	3.77
MDP	3	0%	0	**82.51**	72.04	69.03	61.31	73.15	73.89	73.49	63.39	65.48	45.88	81.88	59.86	75.24	73.76	52.23	70.02
				6.04	10.38	6.97	8.24	5.15	7.68	8.40	8.22	6.62	1.64	4.91	7.76	5.78	5.90	7.14	5.91
MDP	3	0%	n_{obs}	**77.96**	74.07	63.92	61.79	73.26	68.99	75.30	75.06	66.21	45.56	54.56	54.41	53.45	65.86	51.99	59.04
				5.93	4.84	5.19	4.57	6.72	8.06	6.28	4.19	9.85	3.73	3.33	3.42	6.07	7.50	7.30	6.80
MDP	3	50%	0	**65.37**	63.16	59.55	57.84	61.01	61.38	63.17	54.37	55.72	55.26	61.50	59.32	52.04	50.35	35.11	53.06
				10.26	10.79	6.84	13.09	8.53	9.41	11.37	13.30	3.37	0.99	12.09	14.96	3.86	4.55	5.12	9.62
MDP	3	50%	n_{obs}	**67.30**	63.82	52.96	53.01	55.80	54.17	55.79	55.46	54.99	55.55	53.92	53.39	53.09	50.99	46.49	49.79
				8.50	6.76	5.08	3.70	5.71	9.02	5.70	3.25	2.93	0.81	4.82	3.99	5.75	4.50	3.37	4.74

Table A2: Comparison of different Hopfield-RUDDER and LSTM-RUDDER versions w.r.t. reward redistribution score rr_score on different versions of the 1D key-chest environment. Results shown are the mean rr_score over all training set sizes and a 10-fold cross-validation. Error bars show mean standard deviation of 10-fold cross-validation over all training set sizes. Hopfield-RUDDER with reset-max history consistently outperforms all other Hopfield-RUDDER and LSTM-RUDDER versions.

A2 Minecraft Environment

In this section we provide details on the MineRL Minecraft environment, demonstrations, training setup and additional results.

A2.1 Training details

Training of Hopfield-RUDDER was performed in PyTorch [34] using the Adam optimizer [33]. In contrast to the 1D-environment we trained the model only for 100 updates. Weights of the linear mappings are shared such that $m_{state} == m_{stored}$. For training of Hopfield-RUDDER, a mini-batch of 8 random samples from the training set is used as state patterns and the rest of the training set is used as stored patterns for each weight update.

In addition to the history compression we augment the observations with observation deltas, where we compute the delta of observation s_t and s_{t-1} and concatenate the result with the original observation. Furthermore, to reduce training time and GPU memory requirements we store only unique observations in stored patterns $\boldsymbol{Y}$.

Then, the history features are computed from the augmented observations, including actions, and the state and stored patterns are mapped to 128 features via a small neural network with 2 fully connected hidden layers with ReLU activation.

A2.2 Additional reward redistribution plots

In A4, we show additional examples for reward redistribution using Hopfield-RUDDER on the MineRL *ObtainDiamond* demonstrations. These demonstrations have not been used for training the reward redistribution model. The red vertical lines show the auxiliary sparse rewards, provided by the environment. Note that for training Hopfield-RUDDER, we only use episodic reward of 1 for successful demonstrations and 0 for unsuccessful demonstrations, without using the auxiliary rewards. As indicated by the auxiliary rewards, the reward redistribution (in blue) is able to identify sections in the episodes which are important for obtaining the diamond.

A2.3 Observation and action representation

In order to avoid hand-crafted solutions the authors of the MineRL benchmark introduced an obfuscated version of the environment and the demonstrations. In this version the inventory state and actions are encoded using an Auto Encoder. Both observations and actions are 64 dimensional vectors. The decoder of this model is not released and therefore models must learn only based on the encoded states and predict the encoded actions. We also use only the obfuscated inventory state and actions for training the Hopfield-RUDDER model. However, we use the non-obfuscated inventory states for visualizing and inspecting the reward redistribution.

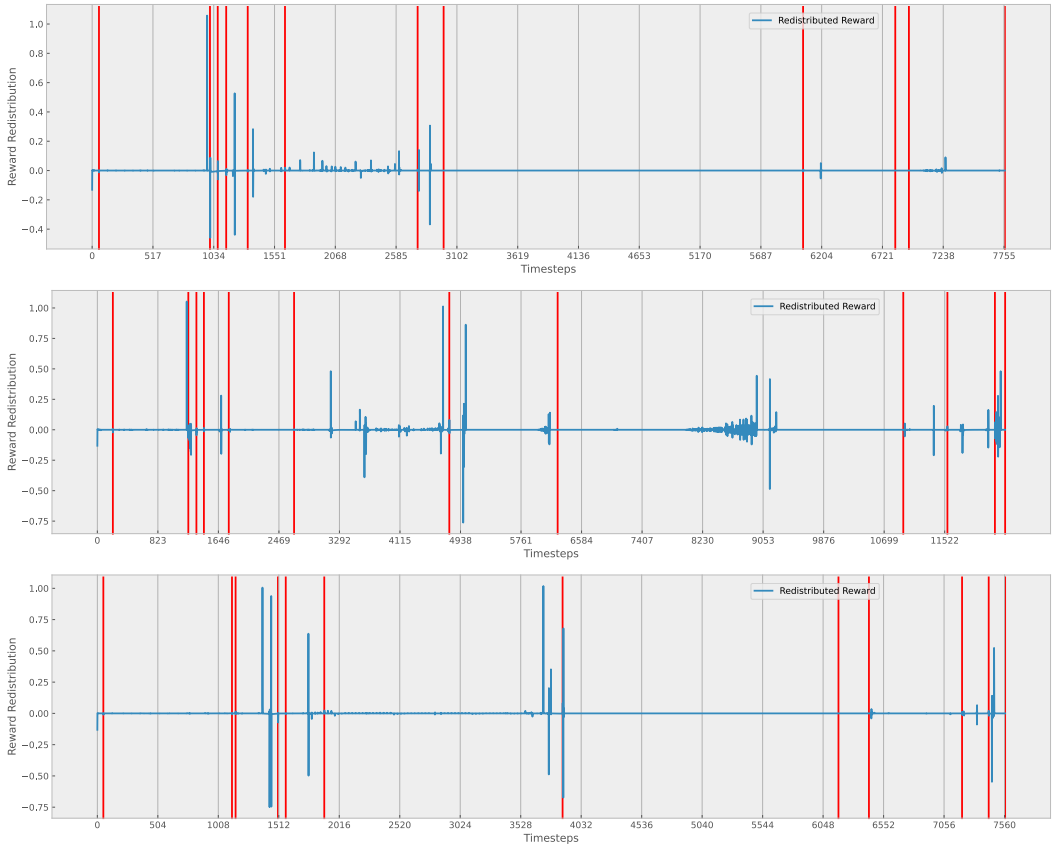

Figure A4: Reward redistribution of three demonstrations for the MineRL *ObtainDiamond* task. **Blue:** redistributed reward **Red:** sparse reward obtainable in the environment. For training we only use episodic reward of 1 for successful episodes and 0 for unsuccessful episodes.

A3 Glossary

J . . . . . .Number of features of history representation $\boldsymbol{v}^h$.
K . . . . . .Number of features of state-action pair representation $\boldsymbol{v}^o$.
$\boldsymbol{G}$ . . . . . .Return values of episodes.
$\boldsymbol{U}$ . . . . . .Raw state patterns (before optional mapping).
$\boldsymbol{W}$ . . . . . .Trainable weights of reset gate f_{reset}.
$\boldsymbol{Y}$ . . . . . .Raw stored patterns (before optional mapping).
$\boldsymbol{u}$ . . . . . .Raw state pattern (before optional mapping).
$\boldsymbol{v}^h$ . . . . . .Representation of the history as vector.
$\boldsymbol{v}^o$ . . . . . .Representation of the state-action pair as vector.
$\boldsymbol{v}$ . . . . . .Raw stored or state pattern as concatenation $[\boldsymbol{v}^o; \boldsymbol{v}^h]$.
$\boldsymbol{y}$ . . . . . .Raw stored pattern (before optional mapping).
β . . . . . .Inverse temperature of $\mathsf{softmax}$ function.
ψ . . . . . .Return decomposition function.
σ . . . . . .Sigmoid function.
$\widehat{\boldsymbol{G}}$ . . . . . .Estimated return values of episodes.
$\widehat{g}$ . . . . . .Estimated return value of an episode.
a_l . . . . . .Action to move 1 position to the left in 1D key-chest environment.

a_r . . . . . .Action to move 1 position to the right in 1D key-chest environment.

a . . . . . .a_t is the action at time t.

f_{reset} . . . . . .Reset gate function for resetting the history $\boldsymbol{v}^h$.

g . . . . . .Return value of an episode.

m_{state} . . . . . .Optional transformation function of raw $\boldsymbol{U}$.

m_{stored} . . . . . .Optional transformation function of raw $\boldsymbol{Y}$.

n_k . . . . . .Number of *keys* required to open *chest* in 1D key-chest environment.

n_{obs} . . . . . .Number of features of observation space in 1D key-chest environment.

n_{rnd} . . . . . .Number of random features in 1D key-chest environment.

p_l . . . . . .Probability of losing all *keys* in 1D key-chest environment.

rr_score . . . . . .Scoring function for reward redistribution for episode.

r . . . . . .r_t is the reward at time t.

scr . . . . . .Scoring function for reward redistribution for time step.

s . . . . . .s_t is the environment state at time t.

c . . . . . .Chest position of 1D key-chest environment.

k . . . . . .Key position of 1D key-chest environment.

s . . . . . .Mid position of 1D key-chest environment.

Hopfield-RUDDER . . .Novel RUDDER using MHN for return decomposition.

LSTM . . . . . .Long short-term memory.

LSTM-RUDDER . . .Original RUDDER using LSTM for return decomposition.

LSTMf . . . . . .Fully-connected LSTM.

LSTMs . . . . . .Sparsely-connected LSTM.

MHN . . . . . .Continuous modern Hopfield network.

RUDDER . . . . . .Return decomposition for delayed rewards.

Appendix References

[32] J. A. Arjona-Medina*, M. Gillhofer*, M. Widrich*, T. Unterthiner, J. Brandstetter, and S. Hochreiter. RUDDER: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, pages 13544–13555, 2019.

[33] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *ArXiv*, 2014.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.