XAI and Strategy Extraction via Reward Redistribution

Marius-Constantin Dinu[1,2,6,7], Markus Hofmarcher[1,2(✉)], Vihang P. Patil[1,2], Matthias Dorfer[4,7], Patrick M. Blies[4], Johannes Brandstetter[1,2], Jose A. Arjona-Medina[1,2,6,7], and Sepp Hochreiter[1,2,3,5,7]

[1] Institute for Machine Learning, Johannes Kepler University Linz, Linz, Austria
{dinu,hofmarcher}@ml.jku.at
[2] LIT AI Lab, Johannes Kepler University Linz, Linz, Austria
[3] ELLIS Unit Linz, Johannes Kepler University Linz, Linz, Austria
[4] enliteAI, Vienna, Austria
[5] Institute of Advanced Research in Artificial Intelligence (IARAI), Vienna, Austria
[6] Dynatrace Research, Linz, Austria
[7] AI Austria, Reinforcement Learning Community, Vienna, Austria

Abstract. In reinforcement learning, an agent interacts with an environment from which it receives rewards, that are then used to learn a task. However, it is often unclear what strategies or concepts the agent has learned to solve the task. Thus, interpretability of the agent's behavior is an important aspect in practical applications, next to the agent's performance at the task itself. However, with the increasing complexity of both tasks and agents, interpreting the agent's behavior becomes much more difficult. Therefore, developing new interpretable RL agents is of high importance. To this end, we propose to use Align-RUDDER as an interpretability method for reinforcement learning. Align-RUDDER is a method based on the recently introduced RUDDER framework, which relies on contribution analysis of an LSTM model, to redistribute rewards to key events. From these key events a strategy can be derived, guiding the agent's decisions in order to solve a certain task. More importantly, the key events are in general interpretable by humans, and are often sub-tasks; where solving these sub-tasks is crucial for solving the main task. Align-RUDDER enhances the RUDDER framework with methods from multiple sequence alignment (MSA) to identify key events from demonstration trajectories. MSA needs only a few trajectories in order to perform well, and is much better understood than deep learning models such as LSTMs. Consequently, strategies and concepts can be learned from a few expert demonstrations, where the expert can be a human or an agent trained by reinforcement learning. By substituting RUDDER's LSTM with a profile model that is obtained from MSA of demonstration trajectories, we are able to interpret an agent at three stages: First, by extracting common strategies from demonstration trajectories with MSA. Second, by encoding the most prevalent strategy via the MSA profile model and therefore explaining the expert's behavior. And third,

M.-C. Dinu and M. Hofmarcher—Equal contribution.

A. Holzinger et al. (Eds.): xxAI 2020, LNAI 13200, pp. 177–205, 2022.
https://doi.org/10.1007/978-3-031-04083-2_10

by allowing the interpretation of an arbitrary agent's behavior based on its demonstration trajectories.

Keywords: Explainable AI · Contribution analysis · Reinforcement learning · Credit assignment · Reward redistribution

1 Introduction

With recent advances in computing power together with increased availability of large datasets, machine learning has emerged as a key technology for modern software systems. Especially in the fields of computer vision [34,52] and natural language processing [14,71] vast improvements have been made using machine learning.

In contrast to computer vision and natural language processing, which are both based on supervised learning, reinforcement learning is more general as it constructs agents for planning and decision-making. Recent advances in reinforcement learning have resulted in impressive models that are capable of surpassing humans in games [39,58,73]. However, reinforcement learning is still waiting for its breakthrough in real world applications, not least because of two issues. First, the amount of human effort and computational resources required to develop and train reinforcement learning systems is prohibitively expensive for widespread adoption. Second, machine learning and in particular reinforcement learning produces black box models, which do not allow explaining model outcomes and to build trust in these models. The insufficient explainability limits the application of reinforcement learning agents, therefore reinforcement learning is often limited to computer games and simulations.

Advances in the field of explainable AI (XAI) have introduced methods and techniques to alleviate the problem of insufficient explainability for supervised machine learning [3–5,41,42,64]. However, these XAI methods cannot explain the behavior of the more complex reinforcement learning agents. Among other problems, delayed and sparse rewards or hand-crafted reward functions make it hard to explain an agent's final behavior. Therefore, interpreting and explaining agents trained with reinforcement learning is an integral component for viably moving towards real-world reinforcement learning applications.

We explore the current state of explainability methods and their applicability in the field of reinforcement learning and introduce a method, Align-RUDDER [45], which is intrinsically explainable by exposing the global strategy of the trained agent. The paper is structured as follows: In Sect. 2.1 we review explainability methods and how they can be categorized. Sect. 2.2 defines the setting of reinforcement learning. In Sect. 2.3, Sect. 2.4 and Sect. 2.5 we explore the problem of credit assignment and potential solutions from the field of explainable AI. In Sect. 2.6 we review the concept of reward redistribution as a solution for credit assignment. Section 3 introduces the concept of strategy extraction and explores its potential for training reinforcement learning agents (Sect. 3.1) as well as its intrinsic explainability (Sect. 3.2) and finally its usage for explaining

arbitrary agent behaviors in Sect. 4. Finally, in Sect. 5 we explore limitations of this approach before concluding in Sect. 6.

2 Background

2.1 Explainability Methods

The importance of explainability methods to provide insights into black box machine learning methods such as deep neural networks has significantly increased in recent years [72]. These methods can be categorized based on multiple factors [15].

First, we can distinguish local and global methods, where global methods explain the general model behavior, while local models focus on explaining specific decisions (e.g. explain the classification of a specific sample) or the influence of individual features on the model output [1,15]. Second, we distinguish between intrinsically explainable models and post-hoc methods [15]. Intrinsically explainable models are designed to provide explanations as well as model predictions. Examples for such models are decision trees [35], rule-based models [75], linear models [22] or attention models [13]. Post-hoc methods are applied to existing models and often require a second model to provide explanations (e.g. approximate an existing model with a linear model that can be interpreted) or provide limited explanations (e.g. determine important input features but no detailed explanations of the inner workings of a model). While intrinsically explainable models offer more detailed explanations and insights, they often sacrifice predictive performance. Post-hoc methods, in contrast, have little to no influence on predictive performance but lack detailed explanations of the model.

Post-hoc explainability methods often provide insights in the form of *attributions*, i.e. a measure of how important certain features are with regard to the model's output. In Fig. 1 we illustrate the model attribution from input towards its prediction. We further categorize attribution methods into *sensitivity analysis* and *contribution analysis*.

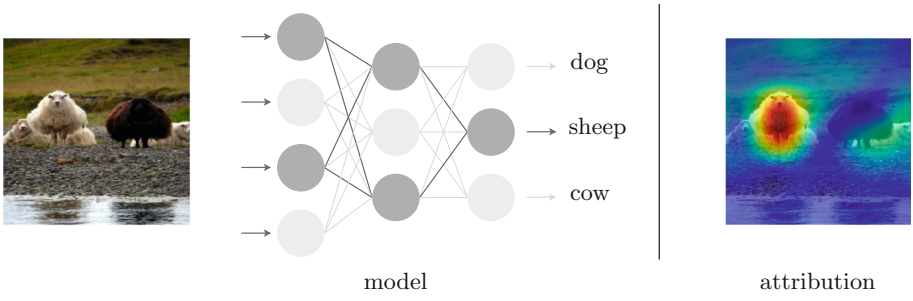

Fig. 1. Illustration of model input attributions towards its prediction [46].

Sensitivity analysis methods, or *"backpropagation through a model"* [8,43,50,51], provide attributions by calculating the gradient of the model with respect to its input. The magnitude of these gradients is then used to assign a measure of importance to individual features of the input. While sensitivity analysis is typically simple to implement, these methods have several problems such as susceptibility to local minima, instabilities, exploding or vanishing gradients and proper exploration [28,54]. The major drawback, however, is that the relevance of features can be missed since it does not consider their *contribution* to the output but only how small perturbations of features change the output. Therefore, important features can receive low attribution scores as small changes would not result in a significant change of the model's output, but removing them would completely change the output. A prominent example for sensitivity analysis methods are *saliency maps* [59].

Contribution analysis methods provide attributions based on the contribution of individual features to the model output, and therefore do not suffer from the drawbacks of sensitivity analysis methods. This can be achieved in a variety of ways, prominent examples are *integrated gradients* [64] or *layer-wise relevance propagation* (ϵ-LRP) [11].

To illustrate the differences between sensitivity analysis and contribution analysis, we can consider a model $\boldsymbol{y} = f(\boldsymbol{x})$ that takes an n-dimensional input vector $\boldsymbol{x} = \{x_1, \ldots, x_n\} \in \mathbb{R}^n$ and predicts a k-dimensional output vector $\boldsymbol{y} = \{y_1, \ldots, y_k\} \in \mathbb{R}^k$. We then define an n-dimensional attribution vector $R^k = \{R_1^k, \ldots, R_n^k\} \in \mathbb{R}^n$ for the k-th output unit, which provides the *relevance* of each input value towards its final prediction. The attribution is obtained through the model gradient:

$$R_i(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial x_i}, \tag{1}$$

although this is not the only option for attribution through gradients. Alternatively, the attribution can be defined by multiplying the input vector with the model gradient [6]:

$$R_i(\boldsymbol{x}) = x_i \frac{\partial f(\boldsymbol{x})}{\partial x_i}. \tag{2}$$

Considering Eq. 1 we answer the question of *"What do we need to change in* $\boldsymbol{x}$ *to get a certain outcome* y_k*?"*, while considering Eq. 2 we answer the question of *"How much did* x_i *contribute to the outcome* y_k*?"* [1].

In reinforcement learning, we are interested in assessing the contributions of actions along a sequence which were relevant for achieving a particular return. Therefore, we are interested in contribution analysis methods rather than sensitivity analysis. We point out that this is closely related to the credit assignment problem, which we will further elaborate in the following sections.

2.2 Reinforcement Learning

In reinforcement learning, an agent is trained to take a sequence of actions by interacting with an environment and by learning from the feedback provided by the environment. The agent selects actions based on its policy, which are executed in the environment. The environment then transitions into its next state based on state-transition probabilities, and the agent receives feedback in the form of the next state and a reward signal. The objective of reinforcement learning is to learn a policy that maximizes the expected cumulative reward, also called return.

More formally, we define our problem setting as a finite Markov decision process (MDP) $\mathcal{P}$ as a 5-tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$ of finite sets $\mathcal{S}$ with states s (random variable S_t at time t), $\mathcal{A}$ with actions a (random variable A_t), and $\mathcal{R}$ with rewards r (random variable R_{t+1}) [47]. 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 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 state-value function $V^\pi(s)$ for a policy π is

$$V^\pi(s) = \mathrm{E}_\pi \left[G_t \mid S_t = s \right]$$

and its respective action-value function $Q^\pi(s, a)$ is

$$Q^\pi(s, a) = \mathrm{E}_\pi \left[G_t \mid S_t = s, A_t = a \right].$$

The goal of reinforcement 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 $\pi^* = \operatorname{argmax}_\pi [v_0^\pi]$. We consider the difficult task of learning a policy when the reward given by the environment is sparse or delayed. An integral part to facilitate learning in this challenging setting is credit assignment, i.e. to determine the contribution of states and actions towards the return.

2.3 Credit Assignment in Reinforcement Learning

In reinforcement learning, we face two fundamental problems. First, the trade-off between exploring actions that lead to promising new states and exploiting actions that maximize the return. Second, the credit assignment problem, which involves correctly attributing credit to actions in a sequence that led to a certain return or outcome [2,66,67]. Credit assignment becomes more difficult as the delay between selected actions and their associated rewards increases [2,45]. The study of credit assignment in sequences is a long-standing challenge and has been around since the start of artificial intelligence research [38]. Chess is an example of a sparse and delayed reward problem, where the reward is given at the end of the game. Assigning credit to the large number of decisions taken in a game of chess is quite difficult when the feedback is received only at the end of the game (i.e. win, lose or draw). It is difficult for the learning system to

identify which actions were more or less important for the resulting outcome. As a result, the notion of winning or losing alone is often not informative enough for learning systems [38]. This motivates the need to improve credit assignment methods, especially for problems with sparse and delayed rewards. We further elaborate on various credit assignment methods in the next section.

2.4 Methods for Credit Assignment

Credit assignment in reinforcement learning can be classified into two different classes: 1) *Structural credit assignment*, and 2) *Temporal credit assignment* [66]. Structural credit assignment is related to the internals of the learning system that lead to choosing a particular action. Backpropagation [27] is quite popular for such structural credit assignment in Deep Reinforcement Learning. In contrast, temporal credit assignment is related to the events (states and/or actions) which led to a particular outcome in a sequence. In this work, we examine temporal credit assignment methods in detail.

Temporal credit assignment methods are used to obtain policies which maximize future rewards. Temporal difference (TD) learning [67] is a temporal credit assignment method which has close ties to dynamic programming and the Bellman operator [10]. It combines policy evaluation and improvement in a single step, by using the maximum action-value estimate at the next state to improve the action-value estimate at the current state. However, TD learning suffers from high bias and slows down learning when the rewards are sparse and delayed. *Eligibility traces* and TD(λ) [60] were introduced to ameliorate the performance of TD. Instead of looking one step into the future, information from n-steps in the future or past are used to update the current estimate of the action-value function. However, the performance of the algorithm is highly dependent on how much further in the future or in the past it looks into. In TD learning, one tries to find the action-value which maximizes the future return. In contrast, there exist direct policy optimization methods like policy gradient [65] and related methods like actor-critic [40,56].

More recent attempts to tackle credit assignment for delayed and sparse rewards have been made in RUDDER: Return Decomposition for Delayed Rewards (RUDDER) [2] and Hindsight Credit Assignment (HCA) [21]. RUDDER aims to identify actions which increase or decrease the expected future return. These actions are assigned credit directly by RUDDER, which makes learning faster by reducing the delay. We discuss RUDDER in detail in Sect. 2.6.

Unlike RUDDER, HCA assigns credit by estimating the likelihood of past actions having led to the observed outcome and consequently uses hindsight information to assign credit to past decisions. Both methods have in common, that the credit assignment problem is framed as a supervised learning task. In the next section, we look at credit assignment from the lens of explainability methods.

2.5 Explainability Methods for Credit Assignment

We have established that assigning credit to individual states, actions or state-action events along a sequence, which is also known as a trajectory or episode in reinforcement learning terminology, can tremendously simplify the task of learning an optimal policy. Therefore, if a method is able to determine which events were important for a certain outcome, it can be used to study sequences generated by a policy. As explainability methods were designed for this purpose, we can employ them to assign credit to important events and therefore speed up learning. As we have explored in Sect. 2.1, there are several methods we can choose from. The choice between intrinsically explainable models and post-hoc methods depends on whether a method can be combined with a reinforcement learning algorithm and is able to solve the task. In most cases, post-hoc methods are preferable, as they do not restrict the learning algorithm and model class. Since we are mainly interested in temporal credit assignment, we will look at explainability methods with a global scope. Sensitivity analysis methods have many drawbacks (see Sect. 2.1) and are therefore not suited for this purpose. Thus, we want to use contribution analysis methods.

2.6 Credit Assignment via Reward Redistribution

RUDDER [2] demonstrates how contribution analysis methods can be applied to target the credit assignment problem. RUDDER redistributes the return to relevant events and therefore sets future reward expectations to zero. The reward redistribution is achieved through return decomposition, which reduces high variance compared to Monte Carlo methods and high biases compared to TD methods [2]. This is possible because the state-value estimates are simplified to compute averages of immediate rewards.

In a common reinforcement learning setting, one can assign credit to an action a when receiving a reward r by updating a policy $\pi(a|s)$ according to its respective Q-function estimates. However, one fails when rewards are delayed, since the value network has to average over a large number of probabilistic future state-action paths that increase exponentially with the delay of the reward [36,48]. In contrast to using a forward view, a backward view approach based on a backward analysis of a forward model avoids problems with unknown future state-action paths, since the sequence is already completed and known. Backward analysis transforms the forward view approach into a regression task, at which deep learning methods excel. As a forward model, an LSTM can be trained to predict the final return, given a sequence of state-actions. LSTM was already used in reinforcement learning [55] for advantage learning [7] and learning policies [23,24,40]. Using contribution analysis, RUDDER can decompose the return prediction (the output relevance) into contributions of single state-action pairs along the observed sequence, obtaining a redistributed reward (the relevance redistribution). As a result, a new MDP is created with the same optimal policies and, in the optimal case, with no delayed rewards (expected future rewards equal zero) [2]. Indeed, for MDPs the Q-value is equal to the expected immediate

reward plus the expected future rewards. Thus, if the expected future rewards are zero, the Q-value estimation simplifies to computing the mean of the immediate rewards.

Therefore, in the context of explainable AI, RUDDER uses contribution analysis to decompose the return prediction (the output relevance) into contributions of single state-action pairs along the observed sequence. RUDDER achieves this by training an LSTM model to predict the final return of a sequence of state-actions as early as possible. By taking the difference of the predicted returns from two consecutive state-actions, the contribution to the final return can be inferred [2].

Sequence-Markov Decision Processes (SDPs). An optimal reward redistribution should transform a delayed reward MDP into a return-equivalent MDP with zero expected future rewards. However, given an MDP, setting future rewards equal to zero is in general not possible. Therefore, RUDDER introduces *sequence-Markov decision processes* (SDPs), for which reward distributions are not required to be Markovian. An SDP is defined as a decision process which is equipped with a Markov policy and has Markov transition probabilities but a reward that is not required to be Markovian. Two SDPs $\tilde{\mathcal{P}}$ and $\mathcal{P}$ are *return-equivalent*, if (i) they differ only in their reward distribution and (ii) they have the same expected return at $t = 0$ for each policy π: $\tilde{v}_0^\pi = v_0^\pi$. RUDDER constructs a reward redistribution that leads to a return-equivalent SDP with a second-order Markov reward distribution and expected future rewards that are equal to zero. For these return-equivalent SDPs, Q-value estimation simplifies to computing the mean.

Return Equivalence. Strictly return-equivalent SDPs $\tilde{\mathcal{P}}$ and $\mathcal{P}$ can be constructed by reward redistributions. Given an SDP $\tilde{\mathcal{P}}$, a *reward redistribution* is a procedure that redistributes for each sequence $s_0, a_0, \ldots, s_T, a_T$ the realization of the sequence-associated return variable $\tilde{G}_0 = \sum_{t=0}^{T} \tilde{R}_{t+1}$ or its expectation along the sequence. The reward redistribution creates a new SDP $\mathcal{P}$ with the redistributed reward R_{t+1} at time $(t+1)$ and the return variable $G_0 = \sum_{t=0}^{T} R_{t+1}$. A reward redistribution is second-order Markov if the redistributed reward R_{t+1} depends only on $(s_{t-1}, a_{t-1}, s_t, a_t)$. If the SDP $\mathcal{P}$ is obtained from the SDP $\tilde{\mathcal{P}}$ by reward redistribution, then $\tilde{\mathcal{P}}$ and $\mathcal{P}$ are strictly *return-equivalent*. Theorem 1 in RUDDER states that the optimal policies remain the same for $\tilde{\mathcal{P}}$ and $\mathcal{P}$ [2].

Reward Redistribution. We consider that a delayed reward MDP $\tilde{\mathcal{P}}$, with a particular policy π, can be transformed into a return-equivalent SDP $\mathcal{P}$ with an optimal reward redistribution and no delayed rewards:

Definition 1 ([2])**.** *For $1 \leqslant t \leqslant T$ and $0 \leqslant m \leqslant T - t$, the expected sum of delayed rewards at time $(t-1)$ in the interval $[t+1, t+m+1]$ is defined as* $\kappa(m, t-1) = \mathrm{E}_\pi \left[\sum_{\tau=0}^{m} R_{t+1+\tau} \mid s_{t-1}, a_{t-1}\right]$.

Theorem 2 ([2])**.** *We assume a delayed reward MDP $\tilde{\mathcal{P}}$, where the accumulated reward is given at sequence end. A new SDP $\mathcal{P}$ is obtained by a second-order Markov reward redistribution, which ensures that $\mathcal{P}$ is return-equivalent to $\tilde{\mathcal{P}}$. For a specific π, the following two statements are equivalent:*

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

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

An optimal reward redistribution fulfills for $1 \leqslant t \leqslant T$ *and* $0 \leqslant m \leqslant T-t$: $\kappa(m,t-1)=0$.

Theorem 2 shows that an optimal reward redistribution can be obtained by a second-order Markov reward redistribution for a given policy. It is an existence proof which explicitly gives the expected redistributed reward. In addition, higher-order Markov reward redistributions can also be optimal. In case of higher-order Markov reward redistribution, Equation (II) in Theorem 2 can have random variables R_{t+1} that depend on arbitrary states that are visited in the trajectory. Then Equation (II) averages out all states except s_t and s_{t-1} and averages out all randomness. In particular, this is also interesting for Align-RUDDER, since it can achieve an optimal reward redistribution. Therefore, although Align-RUDDER is in general not second-order Markov, Theorem 2 still holds in case of optimality.

For RUDDER, reward redistribution as in Theorem 2 can be achieved through return decomposition by predicting $\tilde{r}_{T+1} \in \tilde{R}_{T+1}$ of the original MDP $\tilde{\mathcal{P}}$ by a function g from the state-action sequence. RUDDER determines for each sequence element its contribution to the prediction of $\tilde{r}_{T+1}$ at the end of the sequence. Therefore, it performs 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, RUDDER could use any contribution analysis method. However, RUDDER prefers three methods: (A) differences of return predictions, (B) integrated gradients (IG) [64], and (C) layer-wise relevance propagation (LRP) [5]. For contribution method (A), RUDDER ensures 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 from later on in the sequence for determining the contribution of the current input. Thus, a non-Markovian reward is introduced, 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, RUDDER prefers method (A).

A principle insight on which RUDDER is based, is that the Q-function of optimal policies for complex tasks resembles a step function as they are hierarchical and composed of sub-tasks (blue curve, row 1 of Fig. 2, right panel). Completing such a sub-task is then reflected by a step in the Q-function. Therefore, a step in the Q-function is a change in return expectation, that is, the expected amount of the return or the probability to obtain the return changes. With return decomposition one identifies the steps of the Q-function (green arrows in Fig. 2, right panel), and an LSTM can therefore predict the expected return (red arrow, row 1 of Fig. 2, right panel), given the state-action sub-sequence to

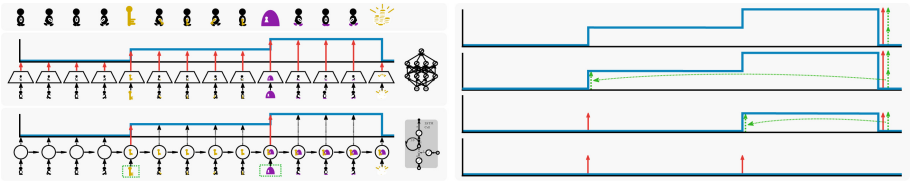

Fig. 2. Basic insight into reward redistribution [45]. **Left panel, Row 1:** An agent has to take a key to unlock a door. Both events increase the probability of receiving the treasure, which the agent always gets as a delayed reward, when the door is unlocked at sequence end. **Row 2:** The Q-function approximation typically predicts the expected return at every state-action pair (red arrows). **Row 3:** However, the Q-function approximation requires only to predict the steps (red arrows). **Right panel, Row 1:** The Q-function is the future-expected return (blue curve). Green arrows indicate Q-function steps and the big red arrow the delayed reward at sequence end. **Row 2 and 3:** The redistributed rewards correspond to steps in the Q-function (small red arrows). **Row 4:** After redistributing the reward, only the redistributed immediate reward remains (red arrows). Reward is no longer delayed. (Color figure online)

redistribute the reward. The prediction is decomposed into single steps of the Q-function (green arrows in Fig. 2). The redistributed rewards (small red arrows in second and third row of right panel of Fig. 2) remove the steps. Thus, the expected future reward is equal to zero (blue curve at zero in last row in right panel of Fig. 2). Future rewards of zero means that learning the Q-values simplifies to estimating the expected immediate rewards (small red arrows in right panel of Fig. 2), since delayed rewards are no longer present. Also, Hindsight Credit Assignment [21] identifies such Q-function steps that stem from actions alone. Figure 2 further illustrates how a Q-function predicts the expected return from every state-action pair, and how it is prone to prediction errors that hamper learning (second row, left panel). Since the Q-function is mostly constant, it is not necessary to predict the expected return for every state-action pair. It is sufficient to *identify relevant state-actions* across the whole episode and use them for predicting the expected return. This is achieved by computing the difference of two subsequent predictions of the LSTM model. If a state-action pair increases the prediction of the return, it is immediately rewarded. Using state-action sub-sequences $(s, a)_{0:t} = (s_0, a_0, \ldots, s_t, a_t)$, the redistributed reward is $R_{t+1} = g((s, a)_{0:t}) - g((s, a)_{0:t-1})$, where g is the return decomposition function, which is represented by an LSTM model and predicts the return of the episode. The LSTM model first learns to approximate the largest steps of the Q-function, since they reduce the prediction error the most. Therefore, the LSTM model extracts first the relevant state-actions pairs (events). Furthermore, the LSTM network [29–32] can store the relevant state-actions in its memory cells and subsequently, only updates its states to change its return prediction, when a new relevant state-action pair is observed. Thus, the LSTM return prediction is constant at most time points and does not have to be learned. The basic insight

that Q-functions are step functions is the motivation for identifying these steps via return decomposition to speed up learning through reward redistribution, and furthermore enhance explainability through its state-action contributions.

In conclusion, redistributed reward serves as reward for a subsequent learning method [2]: (A) The Q-values can be directly estimated [2], which is also shown in Sect. 3 for the artificial tasks and Behavioral Cloning (BC) [70] pre-training for the Minecraft environment [19]. (B) The redistributed rewards can serve for learning with policy gradients like Proximal Policy Optimization (PPO) [57], which is also used in the Minecraft experiments for full training. (C) The redistributed rewards can serve for temporal difference learning, like Q-learning [74].

3 Strategy Extraction via Reward Redistribution

A strategy is a sequence of events which leads to a desirable outcome. Assuming a sequence of events is provided, the extraction of a strategy is the process of extracting events which are important for the desired outcome. This outcome could be a common state or return achieved at the end of the sequences. For example, if the desired outcome is to construct a wooden pickaxe in Minecraft, a strategy extracted from human demonstrations might contain event sequences for collecting a log, making planks, crafting a crafting table and finally a wooden pickaxe.

Strategy extraction is useful to study policies and also demonstration sequences. High return episodes can be studied to extract a strategy achieving such high returns. For example, Minecraft episodes where a stone pickaxe is obtained will include a strategy to make a wooden pickaxe, followed by collecting stones and finally the stone pickaxe. Similarly, strategies can be extracted from low return episodes, which can be helpful in learning which events to avoid. Extracted strategies explain the behavior of underlying policies or demonstrations. Furthermore, by comparing new trajectories to a strategy obtained from high return episodes, the reward signal can be redistributed to those events that are necessary for following the strategy and therefore are important.

However, current exploration strategies struggle with discovering episodes with high rewards in complex environments with delayed rewards. Therefore, episodes with high rewards are assumed and are given as demonstrations, such that they do not have to be discovered by exploration. Unfortunately, the number of demonstrations is typically small, as obtaining them is often costly and time-consuming. Therefore, deep learning methods that require a large amount of data, such as RUDDER's LSTM model, will not work well for this task while Align-RUDDER can learn a good strategy from as few as two demonstrations.

Reward redistribution identifies events which lead to an increase (or decrease) in expected return. The sequence of *important* events is the strategy. Thus, reward redistribution can be used to extract strategies. We illustrate this on the example of profile models in Sect. 3.1. Furthermore, a strategy can be used to redistribute reward by comparing a new sequence to an already given strategy. This results in faster learning, and is explained in detail in Sect. 3.2. Finally, we study expert episodes for the complex task of mining a diamond in Minecraft in Sect. 4.2.

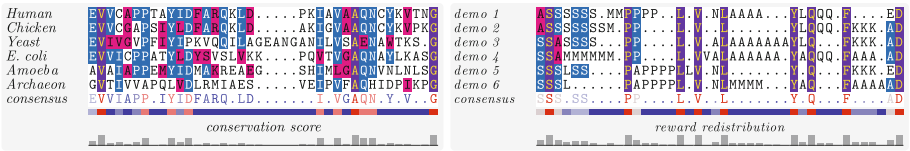

Fig. 3. The function of a protein is largely determined by its structure [45]. The relevant regions of this structure are even conserved across organisms, as shown in the left panel. Similarly, solving a task can often be decomposed into sub-tasks which are conserved across multiple demonstrations. This is shown in the right panel, where events are mapped to the letter code for amino acids. Sequence alignment makes those conserved regions visible and enables redistribution of reward to important events.

3.1 Strategy Extraction with Profile Models

Align-RUDDER introduced techniques from sequence alignment to replace the LSTM model from RUDDER by a profile model for reward redistribution. The profile model is the result of a multiple sequence alignment of the demonstrations and allows aligning new sequences to it. Both the sub-sequences $(s, a)_{0:t-1}$ and $(s, a)_{0:t}$ are mapped to sequences of events and are then aligned to the profile model. Thus, both sequences receive an alignment score S, which is proportional to the return decomposition function g. Similar to the LSTM model, Align-RUDDER identifies the largest steps in the Q-function via relevant events determined by the profile model. The redistributed reward is again $R_{t+1} = g((s, a)_{0:t}) - g((s, a)_{0:t-1})$ (see Eq. (3)). Therefore, redistributing the reward by sequence alignment fits into the RUDDER framework with all its theoretical guarantees. RUDDER is valid and works if its LSTM is replaced by other recurrent networks, attention mechanisms, or, as in case of Align-RUDDER, sequence and profile models [2].

Reward Redistribution by Sequence Alignment. In bioinformatics, sequence alignment identifies similarities between biological sequences to determine their evolutionary relationship [44,62]. The result of the alignment of multiple sequences is a profile model. The profile model is a consensus sequence, a frequency matrix, or a Position-Specific Scoring Matrix (PSSM) [63]. New sequences can be aligned to a profile model and receive an alignment score that indicates how well the new sequences agree to the profile model.

Align-RUDDER uses such alignment techniques to align two or more high return demonstrations. For the alignment, Align-RUDDER assumes that the demonstrations follow the same underlying strategy, therefore they are similar to each other analogous to being evolutionary related. Figure 3 shows an alignment of biological sequences and an alignment of demonstrations where events are mapped to letters. If the agent generates a state-action sequence $(s, a)_{0:t-1}$, then this sequence is aligned to the profile model g giving a score $g((s, a)_{0:t-1})$. The next action of the agent extends the state-action sequence by one state-action pair (s_t, a_t). The extended sequence $(s, a)_{0:t}$ is also aligned to the profile model g, giving another score $g((s, a)_{0:t})$. The redistributed reward R_{t+1} is the

difference of these scores: $R_{t+1} = g((s,a)_{0:t}) - g((s,a)_{0:t-1})$ (see Eq. (3)). This difference indicates how much of the return is gained or lost by adding another sequence element. Align-RUDDER scores how close an agent follows an underlying strategy, which has been extracted by the profile model.

The new reward redistribution approach consists of five steps, see Fig. 4: (I) Define events to turn episodes of state-action sequences into sequences of events. (II) Determine an alignment scoring scheme, so that relevant events are aligned to each other. (III) Perform a multiple sequence alignment (MSA) of the demonstrations. (IV) Compute the profile model like a PSSM. (V) Redistribute the reward: Each sub-sequence τ_t of a new episode τ is aligned to the profile. The redistributed reward R_{t+1} is proportional to the difference of scores S based on the PSSM given in step (IV), i.e. $R_{t+1} \propto S(\tau_t) - S(\tau_{t-1})$.

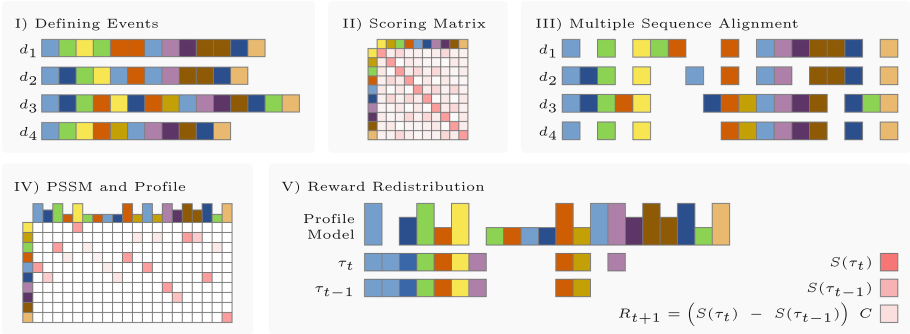

Fig. 4. The five steps of Align-RUDDER's reward redistribution [45]. **(I)** Define events and turn demonstrations into sequences of events. Each block represent an event to which the original state is mapped. **(II)** Construct a scoring matrix using event probabilities from demonstrations for diagonal elements and setting off-diagonal to a constant value. **(III)** Perform an MSA of the demonstrations. **(IV)** Compute a PSSM. Events with the highest column scores are indicated at the top row. **(V)** Redistribute reward as the difference of scores of sub-sequences aligned to the profile.

In the following, the five steps of Align-RUDDER's reward redistribution are explained in detail.

(I) Defining Events. Align-RUDDER considers differences of consecutive states to detect a change caused by an important event like achieving a subtask[1]. An *event* is defined as a cluster of state differences, where similarity-based clustering like affinity propagation (AP) [18] is used. If states are only enumerated, it is suggested to use the "successor representation" [12] or "successor features" [9]. In Align-RUDDER, the demonstrations are combined with state-action sequences generated by a random policy to construct the successor representation.

[1] Any sequence of events can be used for clustering and reward redistribution, and consequently for sub-task extraction.

A sequence of events is obtained from a state-action sequence by mapping states s to its cluster identifier e (the event) and ignoring the actions. Alignment techniques from bioinformatics assume sequences composed of a few events, e.g. 20 events. If there are too many events, good fitting alignments cannot be distinguished from random alignments. This effect is known in bioinformatics as "Inconsistency of Maximum Parsimony" [16].

(II) Determining the Alignment Scoring System. A scoring matrix $\mathbb{S}$ with entries $\mathbb{s}_{i,j}$ determines the score for aligning event i with j. A priori, we only know that a relevant event should be aligned to itself but not to other events. Therefore, we set $\mathbb{s}_{i,j} = 1/p_i$ for $i = j$ and $\mathbb{s}_{i,j} = \alpha$ for $i \neq j$. Here, p_i is the relative frequency of event i in the demonstrations. α is a hyperparameter, which is typically a small negative number. This scoring scheme encourages alignment of rare events, for which p_i is small.

(III) Multiple Sequence Alignment (MSA). An MSA algorithm maximizes the sum of all pairwise scores $S_{\text{MSA}} = \sum_{i,j,i<j} \sum_{t=0}^{L} \mathbb{s}_{i,j,t_i,t_j,t}$ in an alignment, where $\mathbb{s}_{i,j,t_i,t_j,t}$ is the score at alignment column t for aligning the event at position t_i in sequence i to the event at position t_j in sequence j. $L \geq T$ is the alignment length, since gaps make the alignment longer than the length of each sequence. Align-RUDDER uses ClustalW [69] for MSA. MSA constructs a guiding tree by agglomerative hierarchical clustering of pairwise alignments between all demonstrations. This guiding tree allows identifying multiple strategies.

(IV) Position-Specific Scoring Matrix (PSSM) and MSA Profile Model. From the alignment, Align-RUDDER constructs a profile model as a) column-wise event probabilities and b) a PSSM [63]. The PSSM is a column-wise scoring matrix to align new sequences to the profile model.

(V) Reward Redistribution. The reward redistribution is based on the profile model. A sequence $\tau = e_{0:T}$ (e_t is event at position t) is aligned to the profile, which gives the score $S(\tau) = \sum_{l=0}^{L} \mathbb{s}_{l,t_l}$. Here, $\mathbb{s}_{l,t_l}$ is the alignment score for the event e_{t_l} at position l in the alignment. Alignment gaps are columns to which no event was aligned, which have $t_l = T + 1$ with gap penalty $\mathbb{s}_{l,T+1}$. If $\tau_t = e_{0:t}$ is the prefix sequence of τ of length $t + 1$, then the reward redistribution R_{t+1} for $0 \leqslant t \leqslant T$ is

$$\begin{aligned} R_{t+1} &= \left(S(\tau_t) - S(\tau_{t-1})\right) C \\ &= g((s,a)_{0:t}) - g((s,a)_{0:t-1}), \\ R_{T+2} &= \tilde{G}_0 - \sum_{t=0}^{T} R_{t+1}, \end{aligned} \tag{3}$$

where $C = \mathrm{E}_{\text{demo}}\left[\tilde{G}_0\right] / \mathrm{E}_{\text{demo}}\left[\sum_{t=0}^{T} S(\tau_t) - S(\tau_{t-1})\right]$ with $S(\tau_{-1}) = 0$. The original return of the sequence τ is $\tilde{G}_0 = \sum_{t=0}^{T} \tilde{R}_{t+1}$, and the expectation of the return over demonstrations is E_{demo}. The constant C scales R_{t+1} to the range of $\tilde{G}_0$. R_{T+2} is the correction of the redistributed reward [2], with zero

expectation for demonstrations: $\mathrm{E}_{\mathrm{demo}}[R_{T+2}] = 0$. Since $\tau_t = e_{0:t}$ and $e_t = f(s_t, a_t)$, then $g((s,a)_{0:t}) = S(\tau_t)C$. Strict return-equivalence [2] is ensured by $G_0 = \sum_{t=0}^{T+1} R_{t+1} = \tilde{G}_0$. The redistributed reward depends only on the past: $R_{t+1} = h((s,a)_{0:t})$.

Higher-Order Markov Reward Redistribution. Align-RUDDER may lead to higher-order Markov redistribution. However, Corollary 1 in the Appendix of [45] states that the optimality criterion from Theorem 2 in Arjona-Medina et al. [2] also holds for higher-order Markov reward redistribution, if the expected redistributed higher-order Markov reward is the difference of Q-values. In that case, the redistribution is optimal, and there is no delayed reward. Furthermore, the optimal policies are the same as for the original problem. This corollary is the motivation for redistributing the reward to the steps in the Q-function. Furthermore, Corollary 2 in the Appendix of [45] states that under a condition, an optimal higher-order reward redistribution can be expressed as the difference of Q-values.

3.2 Explainable Agent Behavior via Strategy Extraction

The reward redistribution identifies sub-tasks as alignment positions with high redistributed rewards. These sub-tasks are indicated by high scores $\mathbb{s}$ in the PSSM. Reward redistribution also determines the terminal states of sub-tasks, since it assigns rewards for solving the sub-tasks. As such, the strategy for solving a given task is extracted from those demonstrations used for alignment and represented as a sequence of sub-tasks. By assigning rewards to these sub-tasks with Align-RUDDER, a policy can be learned that is also able to achieve these sub-tasks and therefore high returns.

While RUDDER with an LSTM model for reward redistribution is also able to assign reward to important events, in practice it is not easy to identify sub-tasks. Changes in predicted reward from one event to the next are often small, as it is difficult for an LSTM model to learn sharp increases or decreases. Furthermore, it would be necessary to inspect a relatively large number of episodes to identify common sub-tasks. In contrast, the sub-tasks extracted via sequence alignment are often easy to interpret and can be obtained from only a few episodes. The strategy of agents trained via Align-RUDDER can easily be explained by inspecting the alignment and visualizing the sequence of aligned events. As the strategy represents the global long-term behavior of an agent, its behavior can be interpreted through the strategy.

4 Experiments

Using several examples we show how reward redistribution with Align-RUDDER enables learning a policy with only a few demonstrations, even in highly complex environments. Furthermore, the strategy these policies follow is visualized, highlighting the ability of Align-RUDDER's alignment-based approach to interpret agent behavior.

4.1 Gridworld

First, we analyze Align-RUDDER on two artificial tasks. The tasks are variations of the gridworld *rooms example* [68], where cells (locations) are the MDP states. The *FourRooms* environment is a 12×12 gridworld with four rooms. The target is in room four, and the start is in room one (from bottom left, to bottom right) with 20 portal entry locations. *EightRooms* is a larger variant with a 12×24 gridworld divided into eight rooms. Here, the target is in room eight, and the starting location in room one, again with 20 portal entry locations. We show the two artificial tasks with sample trajectories in Fig. 5.

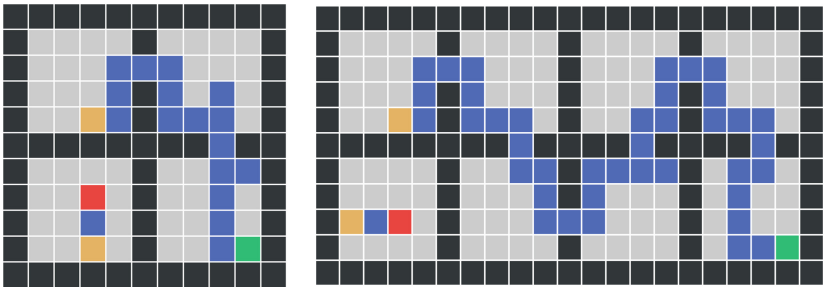

Fig. 5. Examples of trajectories in the two artificial task environments with four (left) and eight (right) rooms. The initial position is indicated in red, the portal between the first and second room in yellow and the goal in green [45]. Blue squares indicate the path of the trajectory. (Color figure online)

In this setting, the states do not have to be time-aware for ensuring stationary optimal policies but the unobserved used-up time introduces a random effect. The grid is divided into rooms. The agent's goal is to reach a target from an initial state with the lowest number of steps. It has to cross different rooms, which are connected by doors, except for the first room, which is only connected to the second room by a *portal*. If the agent is at the portal entry cell of the first room, then it is teleported to a fixed portal arrival cell in the second room. The location of the portal entry cell is random for each episode, while the portal arrival cell is fixed across episodes. The portal entry cell location is given in the state for the first room. The portal is introduced to ensure that initialization with behavioral cloning (BC) alone is not sufficient for solving the task. It enforces that going to the portal entry cells is learned, even when they are at positions not observed in demonstrations. At every location, the agent can move *up, down, left, right*. The state transitions are stochastic. An episode ends after $T = 200$ time steps. If the agent arrives at the target, then at the next step it goes into an absorbing state, where it stays until $T = 200$ without receiving further rewards. Reward is only given at the end of the episode. Demonstrations are generated by an optimal policy with an exploration rate of 0.2.

The five steps of Align-RUDDER's reward redistribution for these experiments are:

(i) Defining Events. Events are clusters of states obtained by Affinity Propagation using the successor representation based on demonstrations as similarity. Figure 6 shows examples of clusters for the two versions of the environment.
(ii) Determining the Alignment Scoring System. The scoring matrix is obtained according to (II), using $\epsilon = 0$ and setting all off-diagonal values of the scoring matrix to -1.
(iii) Multiple sequence alignment (MSA). ClustalW is used for the MSA of the demonstrations with zero gap penalties and no biological options.
(iv) Position-Specific Scoring Matrix (PSSM) and MSA profile model. The MSA supplies a profile model and a PSSM, as in (IV).
(v) Reward Redistribution. Sequences generated by the agent are mapped to sequences of events according to (I). Reward is redistributed via differences of profile alignment scores of consecutive sub-sequences according to Eq. (3) using the PSSM.

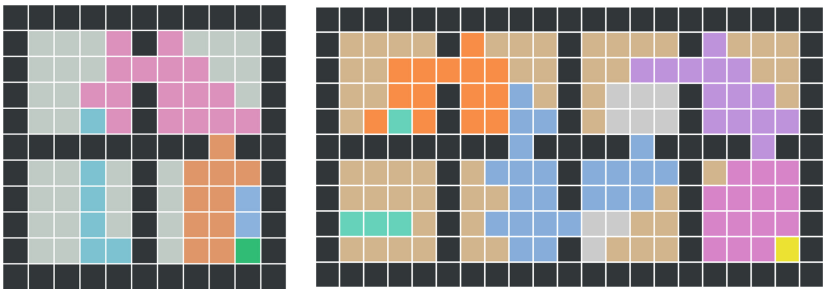

Fig. 6. Examples of different clusters in the FourRooms (left) and EightRooms (right) environment with 1% stochasticity on the transitions after performing clustering with Affinity Propagation using the successor representation with 25 demonstrations. Different colors represent different clusters [45].

The reward redistribution determines sub-tasks like doors or portal arrival. Some examples are shown in Fig. 7. In these cases, three sub-tasks emerged. One for entering the portal and going to the first room, one for travelling from the entrance of one room to the exit of the next room, and finally going to the goal in the last room. The sub-tasks partition the Q-table into sub-tables that represent a sub-agent. The emerging set of sub-agents describe the global behavior of the Align-RUDDER method and can be directly used to explain the decision-making for specific tasks.

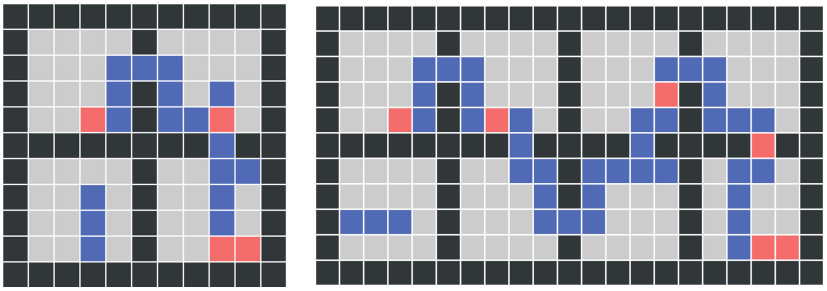

Fig. 7. Reward redistribution for the above trajectories in the FourRooms (left) and EightRooms (right) environments [45]. Here, sub-tasks emerged via reward redistribution for entering the portal, travelling from the entrance of one room to the exit of the next and finally for reaching the goal.

Results. In addition to enabling an interpretation of the strategy for solving a task, the redistributed reward signal speeds up the learning process of existing methods and requires fewer examples when compared to related approaches. All compared methods learn a Q-table and use an ϵ-greedy policy with $\epsilon = 0.2$. The Q-table is initialized by behavioral cloning (BC). The state-action pairs which are not initialized, since they are not visited in the demonstrations, get an initialization by drawing a sample from a normal distribution with mean 1 and standard deviation 0.5 (avoiding equal Q-values). Align-RUDDER learns the Q-table via RUDDER's Q-value estimation (learning method (A) from above). For BC+Q, RUDDER (LSTM), SQIL [49], and DQfD [26] a Q-table is learned by Q-learning. Hyperparameters are selected via grid search with a similar computational budget for each method. For different numbers of demonstrations, performance is measured by the number of episodes to achieve 80% of the average return of the demonstrations. A Wilcoxon rank-sum test determines the significance of performance differences between Align-RUDDER and the other methods.

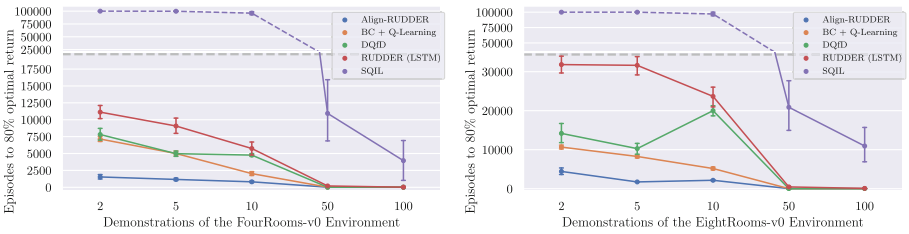

Fig. 8. Comparison of Align-RUDDER and other methods in the FourRooms (left) and EightRooms (right) environments with respect to the number of episodes required for learning on different numbers of demonstrations. Results are the average over 100 trials. Align-RUDDER significantly outperforms all other methods [45].

Figure 8 shows the number of episodes required for achieving 80% of the average reward of the demonstrations for different numbers of demonstrations. In both environments, Align-RUDDER significantly outperforms all other methods, for $\leqslant 10$ demonstrations (with p-values of $< 10^{-10}$ and $< 10^{-19}$ for Task (I) and (II), respectively).

4.2 Minecraft

To demonstrate the effectiveness of Align-RUDDER even in highly complex environments, it was applied to the complex high-dimensional problem of obtaining a diamond in Minecraft with the *MineRL* environment [19]. This task requires an agent to collect a diamond by exploring the environment, gathering resources and building necessary tools. To obtain a diamond the agent needs to collect resources (log, cobblestone, etc.) and craft tools (table, pickaxe, etc.). Every episode of the environment is procedurally generated, and the agent is placed at a random location. This is a challenging environment for reinforcement learning as episodes are typically very long, the reward signal is sparse and exploration difficult. By using demonstrations from human players, Align-RUDDER can circumvent the exploration problem and with reward redistribution can ameliorate the sparse reward problem. Furthermore, by identifying sub-tasks, individual agents can be trained to solve simpler tasks, and help divide the complex long time-horizon task in more approachable sub-problems. In complement to that, we can also inspect and interpret the behavior of expert policies using Align-RUDDER's alignment method. In our example, the expert policies are presented in the form of human demonstrations that successfully obtained a diamond. Align-RUDDER is able to extract a strategy from as few as ten trajectories. In the following, we outline the five steps of Align-RUDDER in the Minecraft environment. Furthermore, we inspect the alignment-based reward redistribution and show how it enables interpretation of both the expert policies and the trained agent.

(i) Defining Events. A state consists of a visual input and an inventory. Both inputs are normalized and then the difference of consecutive states is clustered, obtaining 19 clusters corresponding to events. Upon inspection these clusters correspond to inventory changes, i.e. gaining a particular item. Finally, the demonstration trajectories are mapped to sequences of events. This is shown in Fig. 9.

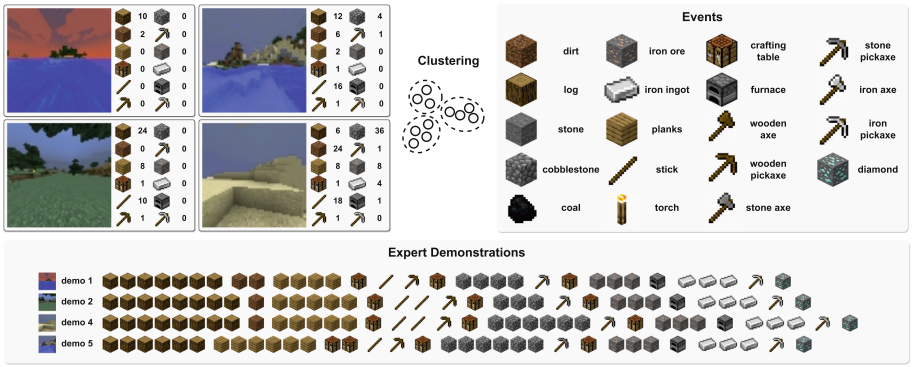

Fig. 9. Step **(I)**: Define events and map demonstrations into sequences of events.

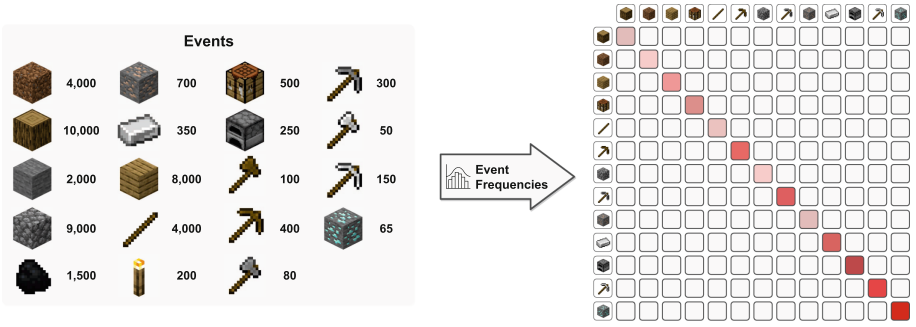

Fig. 10. Step **(II)**: Construct a scoring matrix using event probabilities from demonstrations for diagonal elements and setting off-diagonal to a constant value. Darker colors signify higher score values. For illustration, only a subset of events is shown. (Color figure online)

(ii) Determining the Alignment Scoring System. The scoring matrix is computed according to (II). Since there is no prior knowledge on how the individual events are related to each other, the scoring matrix has the inverse frequency of an event occurring in the expert trajectories on the diagonal and a small constant value on the off-diagonal entries. As can be seen in Fig. 10, this results in lower scores for clusters corresponding to earlier events as they occur more often and high values for rare events such as building a pickaxe or mining the diamond.

(iii) Multiple Sequence Alignment (MSA). The 10 expert episodes that obtained a diamond in the shortest amount of time are aligned using ClustalW with zero gap penalties and no biological options (i.e. arguments to ClustalW related to biological sequences). The MSA algorithm maximizes the pairwise sum of scores of all alignments using the scoring matrix from (II). Figure 11 shows an example of a such an alignment.

Fig. 11. Step **(III)**: Perform multiple sequence alignment (MSA) of the demonstrations.

(iv) Position-Specific Scoring Matrix (PSSM) and MSA Profile Model. The multiple alignment gives a profile model and a PSSM. In Fig. 12 an example of a PSSM is shown, resulting from an alignment of the previous example sequences. The PSSM contains for each position in the alignment the frequency of each event occurring in the trajectories used for the alignment. At this point, the strategy followed by the majority of experts is already visible.

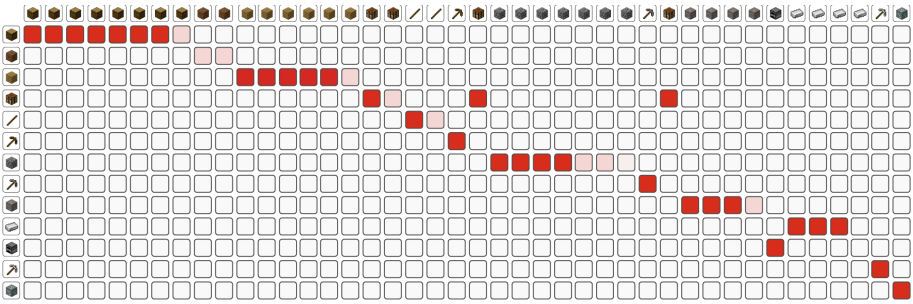

Fig. 12. Step **(IV)**: Compute a position-specific scoring matrix (PSSM). The score at a position from the MSA (column) and for an event (row) depends on the frequency of that event at that position in the MSA. For example, the event in the last position is present in all the sequences, and thus gets a high score at the last position. But it is absent in the remaining position, and thus gets a score of zero elsewhere.

(v) Reward Redistribution. The reward is redistributed via differences of profile alignment scores of consecutive sub-sequences according to Eq. (3) using the PSSM. Figure 13 illustrates this on the example of an incomplete trajectory. In addition to aligning trajectories generated by an agent, we can use demonstrations from human players that were not able to obtain the diamond and therefore highlight problems those players have encountered.

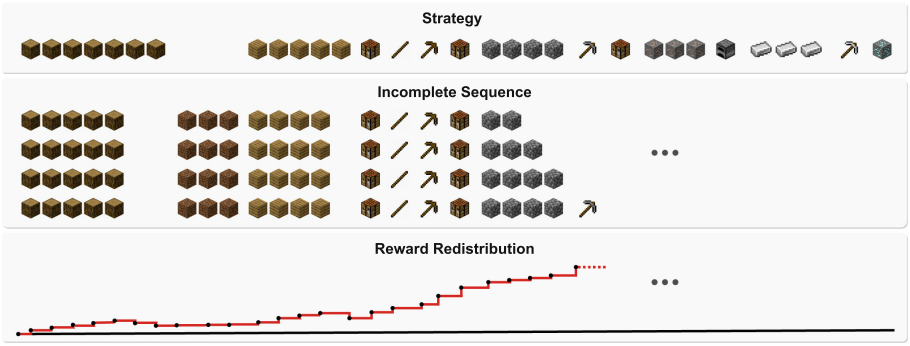

Fig. 13. Step **(V)**: A new sequence is aligned step by step to the profile model using the PSSM, resulting in an alignment score for each sub-sequence. The redistributed reward is then proportional to the difference of scores of subsequent alignments.

Interpreting Agent Behavior. The strategy for obtaining a diamond, an example of which is shown in Fig. 13, is a direct result of Align-RUDDER. If it is possible to map event clusters to a meaningful representation, as is the case here by mapping the clusters to changes in inventory states, the strategy describes the behavior of the expert policies in a very intuitive and interpretable fashion. Furthermore, new trajectories generated by the learned agent can be aligned to the strategy, highlighting differences or problems where the trained agent is unable to follow the expert strategy. Inspecting the strategy it can be seen that random events, such as collecting dirt which naturally occurs when digging, are not present as they are not important for solving the task. Surprisingly, also items that seem helpful such as torches for providing light when digging are not used by the majority of experts even though they have to operate in near complete darkness without them.

Results. Sub-agents can be trained for the sub-tasks extracted from the expert episodes. The sub-agents are first pre-trained on the expert episodes for the sub-tasks using BC, and further trained in the environment using Proximal Policy Optimization (PPO) [57]. Using only 10 expert episodes, Align-RUDDER is able to learn to mine a diamond. A diamond is obtained in 0.1% of the cases, and to the best of our knowledge, no pure learning method[2] has yet mined a diamond [53]. With a 0.5 success probability for each of the 31 extracted sub-tasks[3], the resulting success rate for mining the diamond would be 4.66×10^{-10}. Table 1 shows a comparison of methods on the Minecraft MineRL dataset by the maximum item score [37]. Results are taken from [37], in particular from Fig. 2, and completed by [33,53,61]. Align-RUDDER was not evaluated during

[2] This includes not only learning to extract the sub-tasks, but also learning to solve the sub-tasks themselves.

[3] A 0.5 success probability already defines a very skilled agent in the MineRL environment.

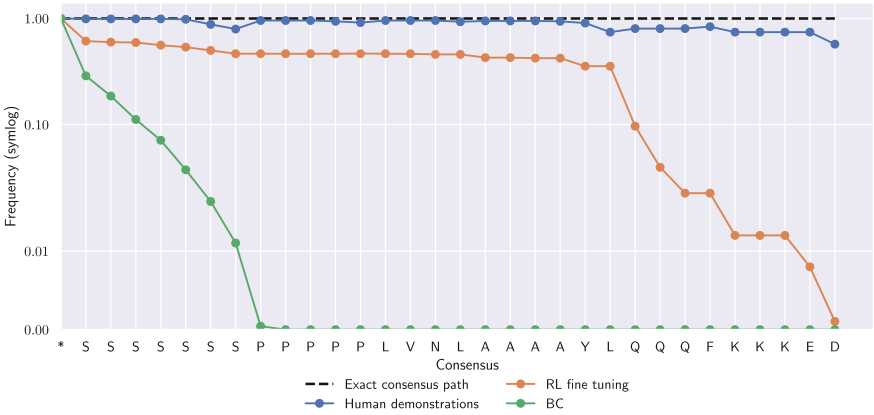

Fig. 14. Comparing the consensus frequencies between behavioral cloning (BC, green), where fine-tuning starts, the fine-tuned model (orange), and human demonstrations (blue) [45]. The plot is in symmetric log scale (symlog in matplotlib). The mapping from the letters on the x-axis to items is as follows: **S:** log, **P:** plank, **L:** crafting table, **V:** stick, **N:** wooden pickaxe, **A:** cobblestone, **Y:** stone pickaxe, **Q:** iron ore, **F:** furnace, **K:** iron ingot, **E:** iron pickaxe, **D:** diamond ore. (Color figure online)

the challenge, and may therefore have advantages. However, it did not receive the intermediate rewards provided by the environment that hint at sub-tasks, but self-discovered such sub-tasks, which demonstrates its efficient learning. Furthermore, Align-RUDDER is capable of extracting a common strategy from only a few demonstrations and train globally explainable models based on this strategy (Fig. 14).

5 Limitations

While Align-RUDDER can extract strategies and speed up learning even in complex environments, the resulting performance depends on the quality of the alignment model. A low quality alignment model can be a result of multiple factors, one of which is having many distinct events ($\gg$20). Clustering can be used to reduce the number of events, which could also lead to a low quality alignment model if too many relevant events are clustered together. While the optimal policy does not change due to a poor alignment of expert episodes, the benefit of employing reward redistribution based on such an alignment diminishes.

The alignment could fail if all expert episodes have different underlying strategies, i.e. no events are common in the expert episodes. We assume that the expert episodes follow the same underlying strategy, therefore they are similar to each other and can be aligned. However, if an underlying strategy does not exist, then the alignment may fail to identify relevant events that should receive high redistributed rewards. In this case, reward is given at sequence end, when the redistributed reward is corrected, which leads to an episodic reward without

Table 1. Maximum item score of methods on the Minecraft task. Methods: Soft-Actor Critic (SAC, [20]), DQfD, Meta Learning Shared Hierarchies (MLSH, [17]), Rainbow [25], PPO, and BC.

Method	Team Name	
Align-RUDDER	Ours	
DQfD	CDS	
BC	MC_RL	
CLEAR	I4DS	
Options&PPO	CraftRL	
BC	UEFDRL	
SAC	TD240	
MLSH	LAIR	
Rainbow	Elytra	
PPO	karolisram	

reducing the delay of the rewards and speeding up learning. This is possible, as there can be many distinct paths to the same end state. This problem can be resolved if there are at least two demonstrations of each of these different strategies. This helps with identifying events for all different strategies, such that the alignment will not fail.

Align-RUDDER has the potential to reduce the cost for training and deploying agents in real world applications, and therefore enable systems that have not been possible until now. However, the method relies on expert episodes and thereby expert decisions, which are usually strongly biased. Therefore, the responsible use of Align-RUDDER depends on a careful selection of the training data and awareness of the potential biases within those.

6 Conclusion

We have analyzed Align-RUDDER, which solves highly complex tasks with delayed and sparse rewards. The global behavior of agents trained by Align-RUDDER can easily be explained by inspecting the alignment of events. Furthermore, the alignment step of Align-RUDDER can be employed to explain arbitrary agents' behavior, so long as episodes generated with this agent are available or can be generated.

Furthermore, we have shown that Align-RUDDER outperforms state-of-the-art methods designed for learning from demonstrations in the regime of few demonstrations. On the Minecraft `ObtainDiamond` task, Align-RUDDER is, to the best of our knowledge, the first pure learning method to mine a diamond.

Acknowledgements. 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), 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, Frauscher Sensonic and the NVIDIA Corporation.

References

1. Ancona, M., Ceolini, E., Öztireli, C., Gross, M.: Gradient-based attribution methods. In: Samek, W., Montavon, G., Vedaldi, A., Hansen, L.K., Müller, K.-R. (eds.) Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. LNCS (LNAI), vol. 11700, pp. 169–191. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28954-6_9. ISBN 978-3-030-28954-6
2. Arjona-Medina, J.A., Gillhofer, M., Widrich, M., Unterthiner, T., Brandstetter, J., Hochreiter, S.: RUDDER: return decomposition for delayed rewards. In: Advances in Neural Information Processing Systems, vol. 32, pp. 13566–13577 (2019)
3. Arras, L., Montavon, G., Müller, K.-R., Samek, W.: Explaining recurrent neural network predictions in sentiment analysis. arXiv, abs/1706.07206 (2017)
4. Arras, L., et al.: Explaining and interpreting LSTMs. In: Samek, W., Montavon, G., Vedaldi, A., Hansen, L.K., Müller, K.-R. (eds.) Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. LNCS (LNAI), vol. 11700, pp. 211–238. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28954-6_11. ISBN978-3-030-28954-6
5. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLoS One **10**(7), e0130140 (2015). https://doi.org/10.1371/journal.pone.0130140
6. Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., Müller, K.-R.: How to explain individual classification decisions. J. Mach. Learn. Res. **11**, 1803–1831 (2010). ISSN 1532-4435
7. Bakker, B.: Reinforcement learning with long short-term memory. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (eds.) Advances in Neural Information Processing Systems, vol. 14, pp. 1475–1482. MIT Press (2002)
8. Bakker, B.: Reinforcement learning by backpropagation through an LSTM model/critic. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, pp. 127–134 (2007). https://doi.org/10.1109/ADPRL.2007.368179
9. Barreto, A., et al.: Successor features for transfer in reinforcement learning. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates Inc. (2017)
10. Bellman, R.E.: Adaptive Control Processes. Princeton University Press, New Jersey (1961)
11. Binder, A., Bach, S., Montavon, G., Müller, K.-R., Samek, W.: Layer-wise relevance propagation for deep neural network architectures. In: Information Science and Applications (ICISA) 2016. LNEE, vol. 376, pp. 913–922. Springer, Singapore (2016). https://doi.org/10.1007/978-981-10-0557-2_87. ISBN 978-981-10-0557-2
12. Dayan, P.: Improving generalization for temporal difference learning: the successor representation. Neural Comput. **5**(4), 613–624 (1993)

13. Correia, A.D.S., Colombini, E.L.: Attention, please! a survey of neural attention models in deep learning. arXiv, abs/2103.16775 (2021)
14. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. arXiv, abs/1810.04805 (2019)
15. Du, M., Liu, N., Hu, X.: Techniques for interpretable machine learning. Commun. ACM **63**(1), 68–77 (2019). https://doi.org/10.1145/3359786. ISSN 0001-0782
16. Felsenstein, J.: Cases in which parsimony or compatibility methods will be positively misleading. Syst. Zool. **27**(4), 401–410 (1978). https://doi.org/10.2307/2412923
17. Frans, K., Ho, J., Chen, X., Abbeel, P., Schulman, J.: Meta learning shared hierarchies. In: International Conference on Learning Representations (2018). arXiv abs/1710.09767
18. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. Science **315**(5814), 972–976 (2007). https://doi.org/10.1126/science.1136800
19. Guss, W.H., et al.: MineRL: a large-scale dataset of minecraft demonstrations. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019) (2019)
20. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Dy, J., Krause, A. (eds.) Proceedings of Machine Learning Research, vol. 80, pp. 1861–1870. PMLR (2018). arXiv abs/1801.01290
21. Harutyunyan, A., et al.: Hindsight credit assignment. In: Advances in Neural Information Processing Systems, vol. 32, pp. 12467–12476 (2019)
22. Hastie, T., Tibshirani, R.: Generalized additive models. Stat. Sci. **1**(3), 297–310 (1986). https://doi.org/10.1214/ss/1177013604
23. Hausknecht, M.J., Stone, P.: Deep recurrent Q-learning for partially observable MDPs. arXiv, abs/1507.06527 (2015)
24. Heess, N., Wayne, G., Tassa, Y., Lillicrap, T.P., Riedmiller, M.A., Silver, D.: Learning and transfer of modulated locomotor controllers. arXiv, abs/1610.05182 (2016)
25. Hessel, M., et al.: Rainbow: combining improvements in deep reinforcement learning. arXiv, abs/1710.02298 (2017)
26. Hester, T., et al.: Deep Q-learning from demonstrations. In: The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18). Association for the Advancement of Artificial Intelligence (2018)
27. Hinton, G.E., Sejnowski, T.E.: Learning and relearning in Boltzmann machines. In: Parallel Distributed Processing, vol. 1, pp. 282–317. MIT Press, Cambridge (1986)
28. Hochreiter, S.: Implementierung und Anwendung eines 'neuronalen' Echtzeit-Lernalgorithmus für reaktive Umgebungen. Practical work, Supervisor: J. Schmidhuber, Institut für Informatik, Technische Universität München (1990)
29. Hochreiter, S.: Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Technische Universität München (1991)
30. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Technical report FKI-207-95, Fakultät für Informatik, Technische Universität München (1995)
31. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
32. Hochreiter, S., Schmidhuber, J.: LSTM can solve hard long time lag problems. In: Mozer, M.C., Jordan, M.I., Petsche, T. (eds.) Advances in Neural Information Processing Systems, vol. 9, pp. 473–479. MIT Press, Cambridge (1997)

33. Kanervisto, A., Karttunen, J., Hautamäki, V.: Playing Minecraft with behavioural cloning. In: Escalante, H.J., Hadsell, R. (eds.) Proceedings of Machine Learning Research (PMLR), vol. 123, pp. 56–66. PMLR (2020)
34. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_48. ISBN 978-3-319-10602-1
35. Lundberg, S.M., et al.: From local explanations to global understanding with explainable AI for trees. Nat. Mach. Intell. **2**(1), 56–67 (2020). https://doi.org/10.1038/s42256-019-0138-9. ISSN 2522-5839
36. Luoma, J., Ruutu, S., King, A.W., Tikkanen, H.: Time delays, competitive interdependence, and firm performance. Strateg. Manag. J. **38**(3), 506–525 (2017). https://doi.org/10.1002/smj.2512
37. Milani, S., et al.: Retrospective analysis of the 2019 MineRL competition on sample efficient reinforcement learning. arXiv, abs/2003.05012 (2020)
38. Minsky, M.: Steps towards artificial intelligence. Proc. IRE **49**(1), 8–30 (1961). https://doi.org/10.1109/JRPROC.1961.287775
39. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
40. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on Machine Learning (ICML), Volume 48 of Proceedings of Machine Learning Research, pp. 1928–1937. PMLR.org (2016)
41. Montavon, G., Lapuschkin, S., Binder, A., Samek, W., Müller, K.-R.: Explaining nonlinear classification decisions with deep Taylor decomposition. Pattern Recogn. **65**, 211–222 (2017). https://doi.org/10.1016/j.patcog.2016.11.008
42. Montavon, G., Samek, W., Müller, K.-R.: Methods for interpreting and understanding deep neural networks. Digit. Signal Process. **73**, 1–15 (2017). https://doi.org/10.1016/j.dsp.2017.10.011
43. Munro, P.W.: A dual back-propagation scheme for scalar reinforcement learning. In: Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA, pp. 165–176 (1987)
44. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. **48**(3), 443–453 (1970)
45. Patil, V.P., et al.: Align-rudder: learning from few demonstrations by reward redistribution. arXiv, abs/2009.14108 (2020). CoRR
46. Petsiuk, V., Das, A., Saenko, K.: RISE: randomized input sampling for explanation of black-box models. arXiv, abs/1806.07421 (2018)
47. Puterman, M.L.: Markov Decision Processes, 2nd edn. Wiley (2005). ISBN 978-0-471-72782-8
48. Rahmandad, H., Repenning, N., Sterman, J.: Effects of feedback delay on learning. Syst. Dyn. Rev. **25**(4), 309–338 (2009). https://doi.org/10.1002/sdr.427
49. Reddy, S., Dragan, A.D., Levine, S.: SQIL: imitation learning via regularized behavioral cloning. In: Eighth International Conference on Learning Representations (ICLR) (2020). arXiv abs/1905.11108
50. Robinson, A.J.: Dynamic error propagation networks. PhD thesis, Trinity Hall and Cambridge University Engineering Department (1989)
51. Robinson, T., Fallside, F.: Dynamic reinforcement driven error propagation networks with application to game playing. In: Proceedings of the 11th Conference of the Cognitive Science Society, Ann Arbor, pp. 836–843 (1989)

52. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
53. Scheller, C., Schraner, Y., Vogel, M.: Sample efficient reinforcement learning through learning from demonstrations in Minecraft. In: Escalante, H.J., Hadsell, R. (eds.) Proceedings of Machine Learning Research (PMLR), vol. 123, pp. 67–76. PMLR (2020)
54. Schmidhuber, J.: 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
55. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015). https://doi.org/10.1016/j.neunet.2014.09.003
56. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization. In: 32st International Conference on Machine Learning (ICML), Volume 37 of Proceedings of Machine Learning Research, pp. 1889–1897. PMLR (2015)
57. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv, abs/1707.06347 (2018)
58. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). https://doi.org/10.1038/nature16961
59. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: visualising image classification models and saliency maps. arXiv, abs/1312.6034 (2014)
60. Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Mach. Learn. **22**, 123–158 (1996)
61. Skrynnik, A., Staroverov, A., Aitygulov, E., Aksenov, K., Davydov, V., Panov, A.I.: Hierarchical deep Q-network with forgetting from imperfect demonstrations in Minecraft. arXiv, abs/1912.08664 (2019)
62. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147**(1), 195–197 (1981)
63. Stormo, G.D., Schneider, T.D., Gold, L., Ehrenfeucht, A.: Use of the 'Perceptron' algorithm to distinguish translational initiation sites in E. coli. Nucleic Acids Res. **10**(9), 2997–3011 (1982)
64. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, vol. 70, pp. 3319–3328 (2017)
65. Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Solla, S., Leen, T., Müller, K. (eds.) Advances in Neural Information Processing Systems, vol. 12. MIT Press (2000)
66. Sutton, R.S.: Temporal credit assignment in reinforcement learning. PhD thesis, University of Massachusetts Amherst (1984)
67. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. MIT Press, Cambridge (2018)
68. Sutton, R.S., Precup, D., Singh, S.P.: Between MDPs and Semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**(1–2), 181–211 (1999)

69. Thompson, J.D., Higgins, D.G., Gibson, T.J.: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. **22**(22), 4673–4680 (1994)
70. Torabi, F., Warnell, G., Stone, P.: Behavioral cloning from observation (2018)
71. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30, pp. 5998–6008. Curran Associates Inc. (2017)
72. Vilone, G., Longo, L.: Explainable artificial intelligence: a systematic review. arXiv, abs/2006.00093 (2020)
73. Vinyals, O., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
74. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, King's College (1989)
75. Wei, D., Dash, S., Gao, T., Gunluk, O.: Generalized linear rule models. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, Volume 97 of Proceedings of Machine Learning Research, pp. 6687–6696. PMLR, 09–15 June 2019