PAPER

Towards Interpretable Reinforcement Learning with State Abstraction Driven by External Knowledge

Nicolas BOUGIE[†,††a)], ***Nonmember and*** **Ryutaro ICHISE**[†,††b)], ***Senior Member***

SUMMARY Advances in deep reinforcement learning have demonstrated its effectiveness in a wide variety of domains. Deep neural networks are capable of approximating value functions and policies in complex environments. However, deep neural networks inherit a number of drawbacks. Lack of interpretability limits their usability in many safety-critical real-world scenarios. Moreover, they rely on huge amounts of data to learn efficiently. This may be suitable in simulated tasks, but restricts their use to many real-world applications. Finally, their generalization capability is low, the ability to determine that a situation is similar to one encountered previously. We present a method to combine external knowledge and interpretable reinforcement learning. We derive a rule-based variant version of the Sarsa(λ) algorithm, which we call Sarsa-rb(λ), that augments data with prior knowledge and exploits similarities among states. We demonstrate that our approach leverages small amounts of prior knowledge to significantly accelerate the learning in multiple domains such as trading or visual navigation. The resulting agent provides substantial gains in training speed and performance over deep q-learning (DQN), deep deterministic policy gradients (DDPG), and improves stability over proximal policy optimization (PPO).

key words: *reinforcement learning, symbolic reinforcement learning, reasoning about knowledge, interpretable reinforcement learning*

1. Introduction

Reinforcement learning methods have led to remarkable successes in a wide variety of tasks. Well-known temporal difference (TD) methods such as Sarsa [1] or Q-learning [2] learn to predict the best action to take by step-wise interactions with the environment. Nevertheless, in many practical real-world applications, learning to control agents directly from high-dimensional observations such as images is one of the challenges of reinforcement learning. Most recent successes in deep reinforcement learning (DRL) have made it possible to master high-dimensional tasks such as autonomous vehicle control [3] or robotic [4]. For instance, "Deep Q-Learning" [5] was able to achieve human performance on many tasks including Atari video games [6]. They relied on the combination of a deep neural network and reinforcement learning.

Unfortunately, DRL methods present a number of challenges. First, they suffer from lack of interpretability. While deep neural networks have been shown to be very effective, the structure of these models makes them difficult to be interpreted, which restricts their use to non-safety critical domains, excluding for example, medicine or law. Second, they require large datasets to be efficient. To build their representation from complex data such as images, neural networks need a large amount of data which entails that they learn slowly. They typically require millions of steps to learn good control policies. As a consequence, DRL cannot be directly applicable to real-world tasks such as robots [7] or recommendation systems [8], emphasizing the need of sample-efficient RL. Third, they suffer from low generalization capability in the sense that their ability to determine similarities among previously encountered situations is limited. In deep reinforcement learning, this abstraction is achieved by a neural network. However, DRL tends to generalize poorly on seemingly minor changes in the task [9], [10].

Motivated to overcome these shortcomings, we propose a learning framework that addresses all of these issues at once by combining simple interpretable reinforcement learning and prior knowledge [11]; and with a minimal human overhead. Our algorithm leverages small amounts of prior knowledge to significantly accelerate learning without the need for demonstrations [12] or specific human engineering. Namely, we introduce prior knowledge to learn rule-based representations of the environment. We propose a new variant of the Sarsa(λ) algorithm [13], that is based on the idea of learning policies that are humanly-comprehensible. The basic concept of this method is to represent the states of our agent as understandable rules, reducing the state space as well as the amount of data needed to learn an efficient state representation. Besides, their structure can be used to maximize the benefit of past experiences to face new situations. This is the key idea of the sub-states mechanism which exploits similarities among rules. Sub-states allow a more frequent update of the Q-values thereby smoothing and speeding-up the learning. Furthermore, we adapt eligibility traces and the learning rate, which turned out to be critical in guiding the algorithm to solve tasks. Finally, we introduce extra supervision during early training by using external knowledge to initialize the parameters of our model. We find that our agent learns effective policies in a small number of iterations and exhibits higher performance than the best generally-applicable reinforcement learning methods. We also demonstrate the effectiveness of this approach across various applications such as visual

Manuscript received June 18, 2019.
Manuscript revised October 2, 2019.
Manuscript publicized July 3, 2020.

[†]The authors are with Sokendai, The Graduate University for Advanced Studies, Tokyo, 101–8430 Japan.
[††]The authors are with National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: nicolas-bougie@nii.ac.jp (Corresponding author)
b) E-mail: ichise@nii.ac.jp

DOI: 10.1587/transinf.2019EDP7170

tasks or time series.

2. Related Work

In this section, we outline the state-of-the-art methods in reinforcement learning to address each of the shortcomings presented before, namely, interpretability, data efficiency and generalization.

A key component of many reinforcement learning algorithms is a neural network. They contain a lot of implicit knowledge about the problems but need to be explainable - they should provide easy-to-interpret reasons for the choice of an action. Kim *et al.* propose to interpret neural networks using visual interpretation [14]. Specifically, a visual attention model is used to train a convolution network. A causal filtering can determine which input regions influence the output. However, this method cannot be easily interpreted by an algorithm and is only suitable for visual domains. Symbolic reinforcement learning [15] aims to solve the lack of interpretability of neural networks as well as improving their generalization capabilities. For example, Garnelo *et al.* combine a back-end deep neural network to learn a symbolic representation and a front-end interpretable reinforcement model that learns the interactions between the objects [9]. Although the agent could learn to master a navigation task, this has not yet been shown to work in rich visual environments. Instead of representing policies by neural networks, Verma *et al.* represent policies using a high-level interpretable language [16]. During the first step, a neural policy network is trained, and then a high-level policy is "extracted". This approach seems not to be directly applicable to environments in which simulations are costly.

Another active area in reinforcement learning is improving data efficiency. A line of work (Bougie *et al.*) involves external knowledge to guide the agent to focus on important features of the environment during training [11]. This allows the neural network to learn a more efficient state representation reducing training time of the agent. At present only simple knowledge can be introduced and the lack of interpretability limits the usability of this method. Another line of work [17] proposes to use deep auto-encoders as pre-processing stage of visual RL. Deep auto-encoders were shown capable to learn robust feature representations. We manage to bypass lack of interpretability by learning robust features using an auto-encoder, and then extract interpretable state representations.

Generalization capability field aims to facilitate transfer learning among observations, central in reinforcement learning to reduce the amount of training data. Compact state representation [18] area focuses on creating an abstract representation of the states [19]. It enables a faster learning than training the agent on the raw data without facing the drawbacks of deep learning. For instance, Andre *et al.* hierarchically abstract the states by decomposing the states into subroutines [20] but has been limited to simple domains. Another method [21] applies a state aggregation technique to reduce the number of state-action pairs that relies on estimating the similarity among the pairs of states. The main drawback is how to compute the similarity between complex objects such as pixels or time-series. All the previously cited approaches suffer from lack of interpretability which reduces their usage in critical applications such as autonomous driving. Bougie *et al.* use an abstract representation of the states to improve generalization while having decisions fully interpretable [22]. This work extends this idea; we aim to reduce the amount of human work by proposing a new deep unsupervised method to generate the rules. In addition, we increase the generalization capability of the agent with better use of sub-states. Finally, we conduct more extensive evaluations of our algorithm on two different domains.

3. Reinforcement Learning

Below we give a brief summary of the Q-learning and Sarsa algorithms, two temporal difference methods for reinforcement learning. Then, we present the eligibility trace mechanism that we improved in our algorithm.

Reinforcement learning [23] consists of an agent learning a policy π by interacting with an environment. At each time-step the agent receives an observation s_t and chooses an action a_t. The agent gets a feedback from the environment called a reward r_t. Given this reward and the observation, the agent can update its policy to improve the future rewards.

Given a discount factor γ, the future discounted rewards, called return R_t, is defined as follows:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

where T is the time-step at which the epoch terminates.

The agent learns to select the action with the maximum return R_t achievable for a given observation [24]. From Eq. (1), we can define the action value $Q^\pi(s,a)$ at a time t as the expected reward for selecting an action a for a given state s_t and following a policy π.

$$Q^\pi(s,a) = \mathbb{E}\left[R_t \mid s_t = s, a\right] \tag{2}$$

The optimal policy π^* is defined as selecting the action with the optimal Q-value, the highest expected return, followed by an optimal sequence of actions. This obeys the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right] \tag{3}$$

In temporal difference (TD) learning methods such as Q-learning or Sarsa, the Q-values are updated after each time-step instead of updating the values after each epoch, as happens in Monte Carlo learning.

3.1 Temporal-Difference Methods

3.1.1 Q-learning Algorithm

A common technique to approximate $\pi \approx \pi^*$ is Q-learning [2].

The estimation of the action value function is performed iteratively by updating $Q(s,a)$. This algorithm is considered as an off-policy method since the update rule is unrelated to the policy that is learned, as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma * \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right] \quad (4)$$

The choice of the action follows a policy derived from Q. For instance, the ϵ-greedy policy trade-off the exploration/exploitation dilemma. During exploitation, the action with the highest estimated return is selected whereas a random action is sampled during exploration. An obvious approach to adapt Q-learning to continuous domains is to discretize the state space, leading to an explosion of the number of Q-values. Therefore, a good estimation of the Q-values in this context is often intractable.

3.1.2 Sarsa Algorithm

Sarsa [25] is a variant of the Q-learning algorithm. The key difference between Q-learning and Sarsa is that Sarsa is an on-policy method, which implies that the Q-values are learned based on the action performed by the current policy instead of a greedy policy. The details are shown in Algorithm 1. Sarsa is more conservative, meaning that Sarsa tends to avoid dangerous actions that may trigger negative rewards. This may be critical in real-world tasks where mistakes are costly such as trading or autonomous driving. The update rule becomes:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \quad (5)$$

Sarsa converges with probability 1 to an optimal policy as long as all the action-value states are visited an infinite number of times. Unfortunately, it is not possible to straightforwardly apply Sarsa learning to continuous or large state spaces. Such large spaces are difficult to explore since it requires a frequent visit of each state to accurately estimate their values, resulting in an inefficient estimation of the Q-values.

3.2 Eligibility Traces

Since it takes time to back-propagate the rewards to the previous Q-values, the above model suffers from slow training in sparse reward environments. Eligibility traces [26] is a mechanism to handle the problem of delayed rewards. Many temporal-difference (TD) methods including Sarsa or Q-learning can use eligibility traces. In popular Sarsa(λ) or Q-learning(λ), λ refers to eligibility traces or n-steps returns. In the case of Sarsa(λ), this leads to the following update rule:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha[r_{t+1} + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)]E_t(s,a) \text{ for all s, a} \quad (6)$$

where

$$E_t(s,a) = \begin{cases} \lambda E_{t-1}(s,a) + 1, & \text{if } s = s_t \text{ and } a = a_t \\ \lambda E_{t-1}(s,a) & \text{otherwise} \end{cases} \quad (7)$$

The temporal difference error for a state is estimated in a bootstrapping process. Instead of looking only at the current reward, in Monte Carlo methods the prediction is made based on the successive states. The TD(λ) method is similar, the current temporal difference error is used to update all the visited states of the corresponding episode. At each step, the reward is back-propagated to the prior states according to their frequency of visit. The parameter $\lambda \in [0..1]$ controls the trade-off between one-step TD methods (TD(0)) and full-step methods (Monte Carlo).

Algorithm 1 Sarsa: Learn function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

procedure SARSA
  Initialize $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ uniformly
  while Q is not converged **do**
    Start in state $s \in \mathcal{S}$
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    while s is not terminal **do**
      Take action a, observe r, s'
      Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
      $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + \gamma \cdot Q(s',a') - Q(s,a))$
      $s \leftarrow s'$
      $a \leftarrow a'$
  return Q

4. Rule-Based Sarsa(λ)

We propose a method, rule-based Sarsa (Sarsa-rb), to enable Sarsa in continuous spaces by injecting external knowledge (Fig. 1). Sarsa-rb is divided into two stages: rule extraction, and, learning (e.g. reinforcement learning agent) (Sarsa-rb(λ)).

At the top level, the rule extraction module which extracts symbols from various sources of data and then generates rules to describe the environment. We formalize this problem as learning a mapping ϕ (i.e. a set of rules) from a state s_t to its abstract representation $\bar{s}_t$, where $\bar{s}_t = \phi(s_t)$; $\phi(s_t) \in \bar{S}$. Since the rules can be created using our prior knowledge about the task and external source of data, we refer to them as *external knowledge*. Besides the general idea that the state representation has the role of encoding and compressing essential information about the task while

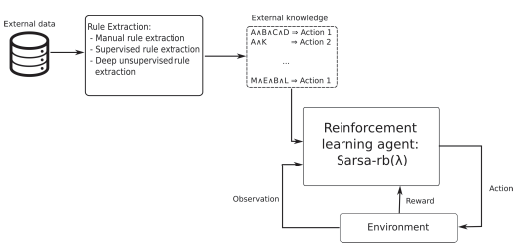


Fig. 1 The proposed rule-based Sarsa(λ) architecture. The extracted rules are fed into the reinforcement learning agent to learn efficient policies and improve state abstraction.

discarding irrelevant states, it enables to inject external knowledge.

The second stage is a reinforcement learning system trained to maximize the reward signal. Training Sarsa in the raw state space is undesirable not only because the structure of the images makes them difficult to interpret, but also because it is hard to predict pixels directly. The rule-based representation constructed in stage one can now be jointly used to enhance state representation in Sarsa and to efficiently initialize the Q-values. As in Sarsa, the agent estimates the Q-values, however, each state is represented by a rule-based representation. At this point, the advantage of representing the states by rules becomes clear. Their compositional structure makes possible to combine and recombine the rules and interpret them. Furthermore, the present architecture maps high-dimensional raw input into a lower-dimensional rule space which reduces the number of Q-values to estimate. In addition, we propose a new technique to update the Q-values of Sarsa that relies on sub-states as generalization mechanism.

To further improve data efficiency and generalization ability of Sarsa-rb, we propose the idea of sub-states. The key insight behind our new mechanism is to exploit the similarities among the rules to help the agent to reason in similar situations. At each iteration, it takes the current observation and instead of updating only one Q-value, the Q-values sharing similarities are also updated, leading to a significant speed-up. Finally, to take advantage of the sub-states, we adapt the original learning rate update and eligibility trace λ used in Sarsa, Sarsa-rb(λ).

4.1 Rule Extraction

The purpose of this first stage is to extract the rules that can be used to represent and compress the observations of our environment. We present three methods. One consists in manually creating them according to our knowledge about the task. Supervised and deep unsupervised extraction methods retrieve patterns from external sources of annotated data. Specifically, the deep unsupervised rule generation method was designed to extract rules from visual inputs by taking advantage of deep learning [27] and follows the idea depicted by Garnelo *et al.* [9]. These sources of data can be various such as annotated datasets similar to the current environment, or any datasets containing relevant information about the task. For instance, for a trading task we can use as external sources of data several stock market datasets from other companies. The idea behind is that among other companies, we can extract patterns that are shared with the current task.

4.1.1 Manual Rule Extraction

One technique to address the rule generation relies on human or background knowledge about the domain. For instance, in a car driving task, such knowledge can be retrieved from traffic rules. Another possible application of our model is automatic trading, for which we can use expertise about time-series and stock markets. Moreover, several previous works about feature engineering can be integrated and combined such as candlestick patterns [28]. This stock-market analysis technique estimates the trend of the share price by identifying patterns into time series.

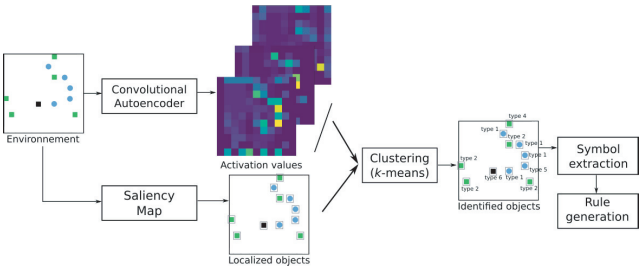


Fig. 2 Deep unsupervised rule extraction method

4.1.2 Supervised Rule Extraction

In real-world environments, the rules can be automatically captured by supervised machine learning methods. We follow a similar idea of Mashayekhi *et al.* [29]. This method extracts the rules from a random forest [30], an ensemble of decision trees [31]. A decision tree consists of several nodes that branch to two sub-trees based on a threshold value on a variable. We call leaf nodes the terminal nodes. A single decision tree has a very limited generalization capability and a high variance [32]. Several ensemble models such as random forest reduce the variance by building many trees and making prediction based on a consensus to among the decision trees. A simple tree traversal method can directly extract patterns from the trees. The recommended action associated to each pattern can be retrieved using simple heuristics such as depicted in Sect. 5.1, manually annotated, or, let empty without affecting much performance after convergence.

4.1.3 Deep Unsupervised Rule Extraction

The goal of this method (Fig. 2) is to extract symbols and then generate rules in an unsupervised manner. This technique is well-suited for visual environments. The first stage consists in extracting and recognizing the objects from images. The next stage generates symbols that represent these objects. Finally, we construct a set of rules by estimating the relevance of each symbol: the position, the color, and the relative position of the objects.

The first stage extracts the objects within an image. We use a deep unsupervised neural network in order to extract features from images. Specifically, we train a convolutional autoencoder [17] and use the compress representation (middle layer) to identify where are the objects as well as recognize them. As shown by Garnelo *et al.* [9] objects are characterized by high activation values throughout the layers of the compressed image representation. Note that with natural images, state-of-the-art methods in unsupervised

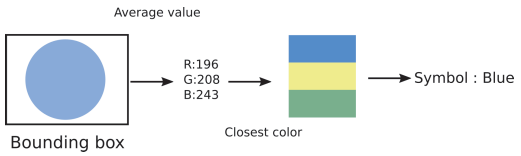


Fig. 3 Overview of the symbol color extraction method.

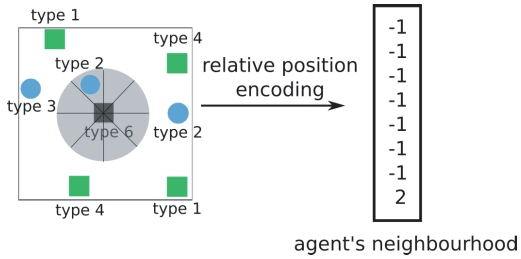


Fig. 4 Relative position extraction method. The objects within a radius r are taken into account and ther clostest objects are saved as symbols according to their angle with the considered object

learning such as PixelVAE [33] can be trained to learn a useful latent representation. In addition, similar objects share a similar activation spectra independently of their position in the image. Therefore, we can extract the position of the objects by extracting the areas with high activation values. One way to classify an area as containing an object or not relies on a fixed threshold. However, using a threshold [9] requires manual tuning to fit the environment. Instead, our technique relies on salient areas of the original image. For each pixel, the salient value is scaled between 0 and 1 and the probability that a pixel is considered as part of an object is equal to this salient value. The corresponding activation values in the compressed representation are then extracted and considered as potentially representing an object.

The second stage characterizes the objects by comparing their activation spectra. This is done by using an unsupervised clustering algorithm. Given the short length of activation spectra, we applied a k-means method [34] to group the objects according to their spectra. The idea behind is that similar objects have similar activation spectra and therefore will belong to the same clusters. A k-means method is then trained using the spectra extracted from the external sources of data (stage 1). When a new image is observed by the agent, the image is processed following a similar pipeline, however, the k-means method is not retrained but only used to predict the label of each identified objects. Since we build an end-to-end model without supervision, we label the new objects with the cluster ID number.

The information extracted at the end of this stage is the position of the objects within the images, their label represented by an integer as well as their bounding boxes. In addition to the type of objects and their positions, we construct two other symbols: the color of the objects and their relative position.

The first symbol represents the colors of the objects. The color of an object is typically the average RGB value of the pixels within the bounding box. This bounding box was found by using the salient map areas. Given the average RGB values, we assign the symbol as the closest color among: *red, blue, green, yellow, red, blue, black, gray, purple*. As can be seen in Fig. 3, the number of possible colors can be augmented to fit the complexity of the task.

The second symbol describes the neighborhood of the objects. We encode the positions of objects relative to other objects within a radius r. This approach is justified by the common sense that the interactions of an object with its closest objects are more likely to have an impact on the reward than the interactions with farther objects. To do this, we divide the neighborhood of an object into areas of 45 degrees and for each one, we store the label of the closest object. In case there is no object in any direction within the radius r, we store -1. The final representation is an array, the concatenation of all the objects for each angle. Figure 4 depicts the process of encoding the relative object positions. There is only one object within the radius r represented by the light gray circle around the agent (dark gray pixel). Since its relative angle is around 330°, the only value different value of -1 in the array is 2, the label of the object.

Once the symbols are extracted, the last stage is to generate a set of rules describing the objects within frames. Note that we only take into account the observed frames followed by a reward greater than zero or negative. This method is justified by the idea that other symbols are not relevant since they don't have an immediate impact on the rewards received by the agent. Given the set of symbols, we construct the associated conjunctions of variables $symbol_1 \wedge symbol_2 \wedge \ldots \wedge symbol_n$. We refer to them as patterns since each one describes an image. The C most frequent patterns are kept and used to describe frames as rules. These generated rules are then used to train our algorithm, Sarsa-rb(λ).

4.2 Rule-Based Sarsa (Sarsa-rb)

Once extracted, we use the rules in the reinforcement learning stage. The Sarsa algorithm maintains a parametrized Q-function which maps the states to their Q-values:

$$Q : S \times A \rightarrow \mathbb{R} \tag{8}$$

Instead of extending Sarsa to continuous state space by discretizing it, our representation relies on a rule-based representation. We propose to learn a Q-function which maps the state representations $\bar{S}$ to their Q-values:

$$Q : \bar{S} \times A \rightarrow \mathbb{R} \tag{9}$$

We propose to represent abstract states $\bar{S}$ by the rules extracted in stage 1. A rule r associates a pattern p to a recommended action, which we denote as a_R

$$\{\phi(s_t) = r | r : p \rightarrow a_R\} \tag{10}$$

where the pattern p is used as the *activation function* of the

state. Specifically, a state is active when the associated pattern p satisfies the current observation. In addition, we improve the Q-value initialization $Q(\phi(s), a)_{t=0}$ by using the action recommended a_R by the pattern. The recommended action belongs to the recommended action space A_r, where we set $A_r = A$ to ensure generality. Note that recommended actions are only used to guide the agent at the start of learning, and then the Q-values are learned from agent's interactions with the environment.

As depicted before, a rule associates a pattern p to an action a, $p \rightarrow a_R$. A pattern p is an arbitrarily complex conjunction of variables. The variables represent significant events in the task. For example, in a task involving driving a car, a variable could be *(speed between 20 and 50 km/h)* and an example of pattern is (*(speed between 20 and 50 km/h)* $\wedge$ *(pedestrian crossing the road)*). Finally, the rule which links this pattern to an action (e.g *brake*, *turn left*, etc.) could be:

$$\begin{aligned} &\textit{(speed between 20 and 50 km/h)}\wedge \\ &\qquad \textit{(pedestrian crossing the road)} \implies \textit{brake} \end{aligned} \tag{11}$$

When the agent receives an observation, the active state is the state for which its associated pattern is satisfied, in other words, all its variables are active. Since no pattern is always satisfied, we added an "empty" state. This is the default state, active regardless of the input. Note that we capture only important information by filtering out irrelevant rules, the less frequent ones.

To improve generalization capability of our agent, we propose a sub-states mechanism. The sub-states are constructed by augmenting each Q-value with an ensemble of sub-states. We define the sub-states as its sub-patterns, the combinations of the variables. For example, given two patterns p_1: $A \wedge B \wedge C$ and p_2: $B \wedge C \wedge D$ the sub-patterns are $A \wedge B$, $B \wedge C$, $A \wedge C$ and $B \wedge C$, $C \wedge D$, $B \wedge D$ respectively. These two patterns are similar and share one sub-state $B \wedge C$, which indicates that the associated Q-values may be similar.

The Fig. 5 shows our model structure. To simplify the representation we show the structure for only one action, however, the agent maintains a parametrized Q-function for each possible action. In summary, the states of Sarsa are replaced by rules. Their associated patterns are used as activation functions of the states. In addition, each state is augmented with one or more sub-states represented by the squares. The state $\phi(s_2)$ represented by the pattern $p_2 : A \wedge B$

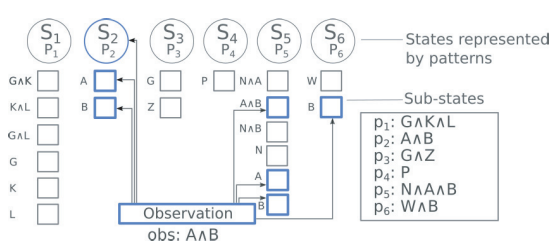


Fig. 5 An illustration of the update of the Q-function. The Q-values of the active state $s2$ and its sub-states are updated. The sub-states sharing similar information with $s2$, in blue, are also updated.

has two possible sub-states corresponding to the two possible sub-patterns A and B.

In Sarsa, the Q-values are uniformly initialized. We introduce a new method to take advantage of prior knowledge by initializing the Q-values according to the recommended actions of the rules:

$$Q(\phi(s), a)_{t=0} = \begin{cases} \mathcal{N}(\mu, \sigma^2), & \text{if } a = a_R \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where μ is the mean, σ^2 the variance, and a_R the action recommended by the rule associated with $\phi(s)$ (Eq. (10)). In order to accelerate learning by appropriately specifying initial Q-values, for a state $\phi(s)$, the initial value of the action recommended by the rule follows a normal distribution centered around μ, greater than 0, to follow our prior knowledge. For the other Q-values, the agent starts out knowing nothing, they are initialized to zero.

This rule-based representation can now be used to learn an effective policy. Our contribution here is to propose a technique to jointly use prior knowledge and reinforcement learning to decrease training time of the agent (rule-based representation) and to avoid learning from scratch (Q-value initialization). Since the number of abstract states is $|\bar{S}|$ (i.e. the number of extracted rules), and the total of Q-values to estimate $|\bar{S}| \times |A| << |S| \times |A|$, our algorithm can be trained in large state space domains.

4.3 Estimation of the Q-Values

Accurately estimate each Q-value would result in a very long training time due to the infrequent visit of most of the states. However, our estimation relies on a sub-states technique which aims to enable fast generalization across observations. We also provide modifications of eligibility traces and learning rate to take advantage of sub-states during the estimation of the Q-values.

4.3.1 Sub-States as Generalization Mechanism

We introduce a *sub-states* mechanism to improve data-efficiency and generalization ability of our reinforcement learning agent. The main drawback of TD algorithms is that only one Q-value is updated at each iteration, entailing that they learn slowly. These algorithms can be augmented using eligibility traces to propagate the reward to the previous states. This work introduces a novel approach to jointly update the similar states and back-propagate the reward to the previous states. The goal is to get most of the benefits of the shared information among the rules while keeping the rest of the Sarsa algorithm intact and efficient. In order to implement this mechanism, we augment each Q-value with its sub-patterns (Fig. 5). Note that to limit the number of sub-states, we limit the size of the sub-rules to conjunctions of at least 3 variables.

We provide modifications to the estimation and update of the Q-values inspired by Sarsa to incorporate sub-states.

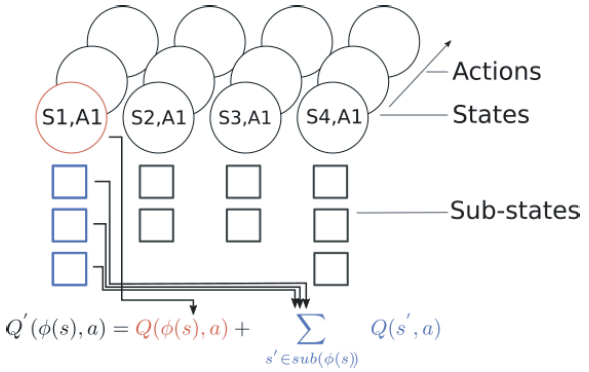


Fig. 6 Estimation of a Q-value, $Q'(\phi(s),a)$, with the sub-states technique. In addition to the Q-value $Q(\phi(s),a)$ itself, the sub-states values $Q(s',a)$ are taken into account.

Our estimation of a Q-value $Q'(\phi(s),a)$ takes into account the Q-value itself $Q(\phi(s),a)$ and the value of its sub-states. Intuitively, this estimation improves generalization among states by incorporating Q-values of similar states and sub-states encountered previously:

$$Q'(\phi(s),a) = Q(\phi(s),a) + \sum_{s' \in sub(\phi(s))} Q(s',a) \tag{13}$$

with $sub(\phi(s))$ the sub-states of a state $\phi(s)$. We modified the update rule of states to incorporate sub-states:

$$Q'_{t+1}(\phi(s),a) = Q_t(\phi(s),a) + \alpha_{Q'(\phi(s),a)t}[r_{t+1} + \gamma Q'_t(\phi(s)_{t+1}, a_{t+1}) - Q'_t(\phi(s)_t, a_t)]E_t(\phi(s),a) \tag{14}$$

and for sub-states s', is defined as:

$$Q_{t+1}(s',a) = Q_t(s',a) + \alpha[r_{t+1} + \gamma Q_t(s'_{t+1}, a_{t+1}) - Q_t(s'_t, a_t)]E_t(s',a) \tag{15}$$

where E_t refers to the eligibility trace for a state-action pair, and $\alpha_{Q'(\phi(s),a)t}$ indicates the learning rate specific to the updated Q-value (Sect. 4.3.3). Our approach to back-propagate the rewards to all similar Q-values is to increment the eligibility traces of the similar sub-states. We illustrate the Q-value estimation process for a simple example in Fig. 6. $Q(s',a)$ refers to the estimation of the value of the sub-state s' given the action a. Adding this term grounds the values of the unvisited states, and makes the value induced by the values of the similar visited states.

A frequent and early update of the sub-states turned out to be critical in fast estimation of the Q-values, inducing a much faster training.

4.3.2 Eligibility Traces

Directly implementing Sarsa-rb is proved to be slow learning in environments with sparse rewards. Our method, Sarsa-rb(λ), is derived from Sarsa(λ). Adding n-steps returns helps to propagate the current reward r_t to the earlier states. We allow a propagation of r_t to the earlier sub-states by changing their eligibility traces. The idea behind is that a sub-state similar to the current state is likely to get a similar reward by following the same action. The update of the current state $\phi(s)$ remains unchanged from Sarsa(λ):

$$\begin{cases} E_t(\phi(s),a) = \lambda E_{t-1}(\phi(s),a) + 1 & \text{if the current state is } \phi(s) \\ E_t(\phi(s),a) = \lambda E_{t-1}(\phi(s),a) & \text{otherwise} \end{cases} \tag{16}$$

Similarly for sub-states:

$$\begin{cases} E_t(s',a) = \lambda E_{t-1}(s',a) + e^{-sim(s',\phi(s))}, & \text{if } s' \text{ is a sub-states of } \phi(s) \\ E_t(s',a) = \lambda E_{t-1}(s',a) + \frac{(e^{-sim(s',\phi(s))})^2}{P}, & \text{for the sub-states sharing at least 2 variables} \\ E_t(s',a) = \lambda E_{t-1}(s',a) & \text{otherwise} \end{cases} \tag{17}$$

where $E(\phi(s),a)$ represents the eligibility trace for state $\phi(s)$, $E(s',a)$ is the eligibility trace of sub-state s' for a given action a, and $sim(s',\phi(s))$ denotes the similarity score between the sub-state s' and the state $\phi(s)$. We define the similarity function as the number of different variables between a sub-state s' and a state $\phi(s)$, $sim(s',\phi(s)) = |s' \cup \phi(s)| - |s' \cap \phi(s)|$. We bounded the score between 0 (identical) and 1. Note that we only take into account the sub-states sharing at least two variables.

Since sub-states are often updated, we avoid exploding eligibility trace values by adding an exponential decay, and a constant P which determines the scale of update signal. Intuitively, a high value decreases the update of the sub-states sharing only a few similar sub-patterns with the current state. Note that similarly to the original idea, when the eligibility trace of a sub-state $E_t(s',a) \approx 0$, the associated Q-value is not updated, but is always taken into account to estimate the Q-values of other states and sub-states. Updates performed in this manner allow estimating Q-values more accurately. Our experiments also suggest that sub-states technique decreases the number of necessary visits to accurately estimate Q-values and yields faster convergent policies.

4.3.3 Adaptive Learning Rate

The linear learning rate α in regular Sarsa assumes that the states are equally visited. Obviously, this assumption is no longer valid. One approach to this problem is to change the update rate of the states according to the frequency of visit of the states and their sub-states. We turn to a learning rate specific to each Q-value $Q'(s_t,a_t)$:

$$\alpha_{Q'(\phi(s),a)t} = \frac{1.0}{(1.0 + visit(\phi(s),a)_t + \sqrt{visit(sub(\phi(s)),a)_t})} \tag{18}$$

We add the term $visit(\phi(s),a)_t$ which refers to the number of visits of the Q-value $Q(\phi(s),a)$. The term $sub(\phi(s))$ defines the ensemble of the sub-states of the state $(\phi(s))$ and

$visit(sub(\phi(s)), a)_t$ is the total number of visits of all its sub-states. A state with its sub-states $sub(\phi(s))$ often visited will be estimated more accurately than a state without sub-states and hence doesn't need to be updated much.

5. Experiments

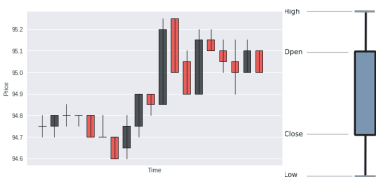
For our experiments, we use two environments that both follow the Open-AI Gym structure [35]. The first environment is a trading task from real stock market data. This task involves large continuous state spaces. We use this task to compare the two rule creation methods (manual rule generation and supervised rule generation). The second task is more challenging and consists of a visual navigation problem using images as the input of our model. This task is used to evaluate our deep unsupervised rule extraction technique. To ensure a fair evaluation, we pretrained the baseline algorithms on the datasets used to generate the rules.

5.1 Trading Task

We evaluated our agent, Sarsa-rb(λ), on the OpenAI trading environment, a complex and fluctuating simulation from real stock market data (Fig. 7 (a)). The observations (Fig. 7 (b)) are given to the agent in the form of a vector of 4 continuous variables that were recorded during a one minute interval: the open price, the close price, and the highest/lowest price. The action set consists of 3 actions: *Buy*, *Hold* and *Sell*. The reward is computed according to the win/loose after buying or selling. Each training episode is followed by a testing episode to evaluate the average reward of the agent on another subset of the same stock price. Each episode was played until the training data are consumed, approximatively 10^5 iterations.

Our system learns to trade on a minutely stock index. In total, we used 4 datasets with a duration varying between 2 years and 5 years. One stock index was used to train the agent, and the other three as external sources of knowledge to generate the rules. To fairly evaluate the model, we trained it on 80% of the training examples and evaluate the performance on the remaining 20%. We ran a grid search over the parameters to initialize the Q-values and found that μ the mean equals to 0.25 and σ equals to 0.2 were the best parameters. We use $P = 100$ as decay factor of eligibility traces. In the case of manually created rules, we first compute the percentage increase in the share price 14 days later and then estimate an optimal action associated with each pattern. In total, we took into account 40 candlestick patterns.

We follow a simplified technique used by Mashayekhi *et al.* to generate the rules from a random forest [29]. Briefly, we extract the patterns top to bottom and filter the patterns to avoid redundancy. To construct the trees, we automatically annotate 6000 samples into 3 classes. Each sample is the aggregation of the last 5 prices. We labeled the dataset according to the price p_{diff} increase 14 days later (p_{diff} >=0.5%, p_{diff} <=−0.5%, 0.5%< p_{diff} >−0.5%) to train a random forest. We compute p_{diff} as the average between the open and close price. In order to limit the number of rules and since the impact on accuracy was minimal, we build 20 trees with a maximum height of 4. In total, we retrieved 855 rules. For each pattern, the predicted class (i.e. the trend of the stock market) was used to recommended an action.



(a) An example of OHLC chart (open, high, low, close)

(b) Structure of one observation

Fig. 7 Example of a sample of data from the environment. The left plot shows the time series and the right plot is the structure of one observation.

5.1.1 Rule Creation Performance

First, we evaluate the two rule creation techniques discussed in Sect. 4.1 and their impact on the average reward. After creating the rules on 3 stock prices, we obtained between 855 and 3240 rules, resulting in a maximum of 3240×3 Q-values to estimate. Figure 8 shows the performance of Sarsa-rb with the states created using the different methods. The results are obtained by running Sarsa-rb without sub-states and eligibility traces, and the same hyper-parameters to allow a controlled experiment focused on rule effectiveness.

As can be seen in Fig. 8, the agent using background knowledge based rules achieves the highest score on average. Performance of the agent was mainly affected by the quality of the rules, we could not establish a link between the number of rules and the quality of the model. We conclude that manually creating the rules was the most efficient technique. Furthermore, the rules can be combined to take advantage of each technique.

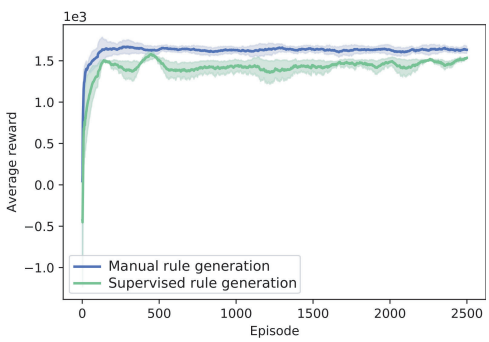


Fig. 8 Performance curves of Sarsa-rb using a selection of techniques to create the rules: background knowledge based (blue) and extracted from a random forest (green).

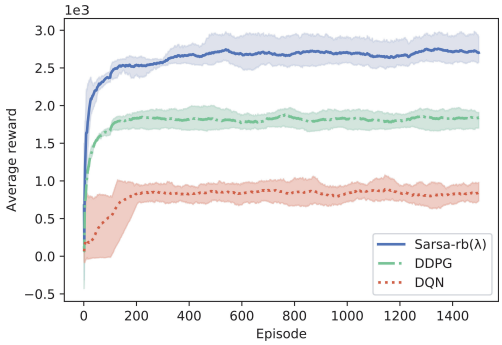


Fig. 9 Performance curves for a selection of algorithms: original Deep Q-learning algorithm (red), Deep Deterministic Policy Gradients algorithm (green) and Sarsa-rb(λ) (blue).

5.1.2 Overall Performance

We evaluated the performance of the learned policy using the proposed sub-states mechanism. We compare Sarsa-rb(λ) with two baselines, a deep recurrent Q-learning model [36] and a DDPG model [37]. DDPG method was shown to perform well in trading tasks [38], [39]. Since the feasible trading actions is in a discrete set, the output of the actor is a vector of n real numbers. The action to take corresponds to the index of the maximum value.

For this evaluation, we individually tuned the hyperparameters of each model. We decreased the learning rate from $\alpha = 0.3$ to $\alpha = 0.0001$, and increased the eligibility trace from $\lambda = 0.8$ to $\lambda = 0.995$, then used $\lambda = 0.9405$ and $P = 100$. Each parameter value was sampled within the given interval, and the algorithm evaluated using those values. We use an ϵ-greedy policy as the behavior policy π. It chooses a random action with a probability ϵ and an optimal action with a probability $1 - \epsilon$. In our experiments, ϵ is set to 0.01. The plots are averaged over 5 runs. Finally, we used the external knowledge based rules as the states of Sarsa-rb(λ). Note that since Sarsa or Sarsa(λ) cannot learn from continuous state spaces, we don't report their performance.

We report the learning curve on the testing data in Fig. 9. Sarsa-rb(λ) always achieve a score higher than DQN and DDPG. From Fig. 9, it is clear that Sarsa-rb(λ) improves over DQN - we observe after converging an average reward around 3.3 times higher. DDPG appears less fluctuating than Sarsa-rb(λ) but also less effective.

The key concepts of our algorithm are sub-states and their capacity to transfer knowledge. By representing the observations with rules we can easily understand when a situation is analogous to a previously encountered set of situations. Sub-states allow to transfer knowledge from previous similar situations and even to transfer knowledge from partially encountered events. It turns out to be critical in environments with sparse rewards such as the trading environment - a reward is given only when the agent buy or sell. In the current algorithm, this capability is achieved through a simple similarity measure, but in the future we aim to develop a more efficient similarity measure.

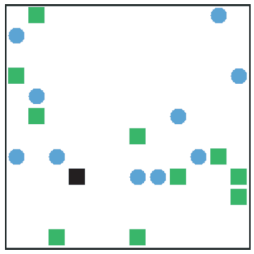


Fig. 10 A board of the visual navigation environment. The agent is represented by a gray square, the food to collect by the blue circles and the walls by the green squares.

5.2 Visual Navigation Task

As a benchmark for our visual rule extraction method, we developed a simple navigation task. The environment consists of a board with obstacles, and objects of different shapes and colors (Fig. 10). The agent learns to collect food, represented as blue circles, and to avoid walls, represented as green squares. The agent has to learn to navigate using one of the possible actions (*up*, *right*, *down*, *left*). Encountering a food results in a positive reward (+5), a wall, a negative reward (−5) while moving to an empty cell results in a negative reward (−0.2).

The goal of the agent is to collect the five "food" objects randomly positioned across the map. Eating the last food object results in a positive reward (+20) whereas going out the map gives a negative reward (−5) and restarts the game. The agent receives as input a 2D RGB image corresponding to what the agent sees around it. The raw frame is resized to 84 × 84 pixels. To decrease the storage cost of the images we convert the image scale from $0 - 255$ to $0 - 1$. Note that we limit the maximum time to 1 minute for the agent to find a strategy to get a maximum reward. The rules were extracted from a dataset of 100 000 images of the environment following the deep unsupervised rule extraction method. We trained a convolutional autoencoder on 100 000 random boards in order to extract features from images. To have an accurate representation of the environment, the position of the agent varied randomly as well as the objects whose numbers were randomly selected. For our experiments, the input of the neural network consists of a 84×84×3 RGB image. The first 3 layers convolve with the following parameters (filter: 64, 32, 32, kernel size: 3×3, 3×3, 3×3). Each one was followed by a 32×32 pooling layer. The last 4 layers are the corresponding decoding layers. We use the compressed representation, the middle layer, to recognize the objects. Since we do not assume strong prior knowledge about the task, we set the number of clusters of the k-means model to $K = 8$. Finally, We kept the 6000 most frequent patterns to build the rules.

5.2.1 Overall Performance

We compared our agent trained on the visual navigation task

Table 1 The table evaluates performance in terms of average reward. We compare Sarsa-rb(λ) with Deep Q-learning (DQN) and Proximal Policy Optimization (PPO) algorithm.

	20000 episodes	80000 episodes	160000 episodes
Sarsa-rb(λ)	2.01±0.23	9.4±1.18	16.6±1.4
DQN	−3.2±0.85	−3.3±0.83	−3.2±0.83
PPO	1.8±1.0	8.9±3.0	13.2±3.4
DQfD	5.2 ± 0.33	9.2 ± 1.06	12.1 ± 1.01

to several methods from the literature, which are considered to be effective for visual problems. We compared our algorithm against tuned versions of proximal policy optimization (PPO) [40], deep q-learning (DQN), and, deep q-learning from demonstrations (DQfD) [12]. DQfD pretrains a DQN agent with demonstrations as source of prior knowledge. For our experiments, we had a human player play the game. In total, 6237 transitions were recorded and used to pre-train DQN. We trained 5 agents for each algorithm with the same settings. The average reward is shown in Table 1. Note that for these two deep learning algorithms, we used the same policy network architecture as used by Mnih *et al.* [41].

Our approach outperforms standard DQN in term of convergence speed and quality of policy. Sarsa-rb(λ) is shown more stable than PPO. Although DQfD achieves a higher average reward at the start of learning, our model convergences towards a better policy. Besides, our technique preserves interpretability while learning a good policy, making it more suitable for real-world tasks. By representing the observations of the environment with understandable rules, we produced a humanly-comprehensible model. Furthermore, we can control the importance of knowledge transfer by giving more or less importance to sub-states.

6. Conclusion

By introducing small amounts of prior knowledge into reinforcement learning architectures, our agent can learn interpretable and compact representations of the environment. Morever, our algorithm takes advantage of such knowledge to significantly accelerate learning. Given that the states are represented by rules, we can analyze which objects, as well as symbols, are involved in the choice of an action and their importance. Specifically, the Q-values can be interpreted to evaluate the weight of each feature on the choice of the action. We have also shown its ability to solve complex tasks with continuous state spaces, and exceeds baseline agents in term of overall performance. To support these claims, we presented an exhaustive evaluation on a time series task and a visual task. We observed that our system dramatically outperforms neural network based methods in a range of different domains.

Several research directions are promising to address more drawbacks inherent to deep reinforcement learning. First, improving the transfer learning by using a more sophisticated similarity algorithm. Second, adapting over training the rules and symbols to discard the useless rules to decrease learning time and improve computational efficiency. Finally, we are interested in extending our experiments to new domains such as textual environments.

References

[1] R.S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," Proc. Advances in Neural Information Processing Systems, pp.1038–1044, 1996.
[2] C.J. Watkins and P. Dayan, "Q-learning," Machine learning, vol.8, no.3-4, pp.279–292, 1992.
[3] P. Abbeel, A. Coates, M. Quigley, and A.Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," Proc. Advances in Neural Information Processing Systems, pp.1–8, 2007.
[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," The Journal of Machine Learning Research, vol.17, no.1, pp.1334–1373, 2016.
[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.
[6] M.G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: an evaluation platform for general agents," Proc. International Conference on Artificial Intelligence, pp.4148–4152, 2015.
[7] T. Hester and P. Stone, "Texplore: real-time sample-efficient reinforcement learning for robots," Machine learning, vol.90, no.3, pp.385–429, 2013.
[8] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," IEEE Trans. Knowl. Data Eng., vol.17, no.6, pp.734–749, 2005.
[9] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards deep symbolic reinforcement learning," arXiv preprint arXiv:1609.05518, 2016.
[10] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," arXiv preprint arXiv:1804.06893, 2018.
[11] N. Bougie and R. Ichise, "Deep reinforcement learning boosted by external knowledge," Proc. ACM Symposium on Applied Computing, pp.331–338, 2018.
[12] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al., "Deep q-learning from demonstrations," Thirty-Second AAAI Conference on Artificial Intelligence, pp.3223–3230, 2018.
[13] S.P. Singh and R.S. Sutton, "Reinforcement learning with replacing eligibility traces," Machine Learning, vol.22, no.1-3, pp.123–158, 1996.
[14] J. Kim and J. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," Proc. International Conference on Computer Vision, pp.2961–2969, IEEE, 2017.
[15] A. d'Avila Garcez, A. Resende Riquetti Dutra, and E. Alonso, "Towards symbolic reinforcement learning with common sense," arXiv preprint arXiv:1804.08597, 2018.
[16] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," arXiv preprint arXiv:1804.02477, 2018.
[17] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," The 2010 International Joint Conference on Neural Networks (IJCNN), pp.1–8, IEEE, 2010.
[18] M. Rosencrantz, G. Gordon, and S. Thrun, "Learning low dimensional predictive representations," Proc. International Conference on Machine learning, p.88, 2004.
[19] S. Džeroski, L. De Raedt, and K. Driessens, "Relational reinforcement learning," Machine learning, vol.43, no.1-2, pp.7–52, 2001.
[20] D. Andre and S.J. Russell, "State abstraction for programmable

reinforcement learning agents," Proc. National Conference on Artificial Intelligence, pp.119–125, 2002.
[21] M.K. Gunady and W. Gomaa, "Reinforcement learning generalization using state aggregation with a maze-solving problem," Proc. Conference on Electronics, Communications and Computers, pp.157–162, 2012.
[22] N. Bougie and R. Ichise, "Abstracting reinforcement learning agents with prior knowledge," Proc. International Conference on Principles and Practice of Multi-Agent Systems, pp.431–439, Springer, 2018.
[23] R.S. Sutton, "Learning to predict by the methods of temporal differences," Machine learning, vol.3, no.1, pp.9–44, 1988.
[24] R.S. Sutton and A.G. Barto, Reinforcement learning: an introduction, MIT press, Cambridge, 1998.
[25] G.A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," tech. rep., University of Cambridge, Oct. 04 1994.
[26] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," Proc. International Conference on Machine Learning, pp.463–471, 1998.
[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol.521, no.7553, pp.436–444, 2015.
[28] S. Nison, Japanese candlestick charting techniques: a contemporary guide to the ancient investment techniques of the Far East, Penguin, 2001.
[29] M. Mashayekhi and R. Gras, "Rule extraction from random forest: the rf+hc methods," Proc. Canadian Conference on Artificial Intelligence, pp.223–237, Springer, 2015.
[30] M. Pal, "Random forest classifier for remote sensing classification," Proc. International Journal of Remote Sensing, vol.26, no.1, pp.217–222, 2005.
[31] S.R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," IEEE Trans. Syst., Man, Cybern., vol.21, no.3, pp.660–674, 1991.
[32] Y. Bengio, O. Delalleau, and C. Simard, "Decision trees do not generalize to new variations," Computational Intelligence, vol.26, no.4, pp.449–467, 2010.
[33] I. Gulrajani, K. Kumar, F. Ahmed, A.A. Taiga, F. Visin, D. Vazquez, and A. Courville, "Pixelvae: A latent variable model for natural images," arXiv preprint arXiv:1611.05013, 2016.
[34] S. Lloyd, "Least squares quantization in pcm," IEEE Trans. Inf. Theory, vol.28, no.2, pp.129–137, 1982.
[35] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym." https://github.com/openai/gym, 2016.
[36] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," arXiv preprint arXiv:1507.06527, 2015.
[37] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.
[38] Z. Xiong, X.Y. Liu, S. Zhong, A. Walid, et al., "Practical deep reinforcement learning approach for stock trading," arXiv preprint arXiv:1811.07522, 2018.
[39] A.R. Azhikodan, A.G. Bhat, and M.V. Jadhav, "Stock trading bot using deep reinforcement learning," in Innovations in Computer Science and Engineering, pp.41–49, Springer, 2019.
[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.
[41] 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, vol.518, no.7540, pp.529–533, 2015.

Nicolas Bougie graduated from the University of Paris Sud, France in 2017. He studied about machine learning and artificial intelligence. He is currently a PhD student at the National Institute of Informatics in Japan and a student at the Sokendai University. His research area covers reinforcement learning, deep learning and machine learning.

Ryutaro Ichise received his Ph.D. degree in computer science from Tokyo Institute of Technology, Tokyo, Japan, in 2000. From 2001 to 2002, he was a visiting scholar at Stanford University. He is currently an associate professor in Principles of Informatics Research Division at National Institute of Informatics in Japan. His research interests include machine learning, semantic web, and data mining.