Rule-based Reinforcement Learning augmented by External Knowledge

Nicolas Bougie[1][2], **Ryutaro Ichise**[1]
[1] National Institute of Informatics
[2] The Graduate University for Advanced Studies Sokendai
nicolas-bougie@nii.ac.jp, ichise@nii.ac.jp

Abstract

Reinforcement learning has achieved several successes in sequential decision problems. However, these methods require a large number of training iterations in complex environments. A standard paradigm to tackle this challenge is to extend reinforcement learning to handle function approximation with deep learning. Lack of interpretability and impossibility to introduce background knowledge limits their usability in many safety-critical real-world scenarios. In this paper, we study how to combine reinforcement learning and external knowledge. We derive a rule-based variant version of the Sarsa(λ) algorithm, which we call Sarsa-rb(λ), that augments data with complex knowledge and exploits similarities among states. We apply our method to a trading task from the Stock Market Environment. We show that the resulting algorithm leads to much better performance but also improves training speed compared to the Deep Q-learning (DQN) algorithm and the Deep Deterministic Policy Gradients (DDPG) algorithm.

1 Introduction

Over last few years, reinforcement learning (RL) has made significant progress to learn good policies in many domains. Well-known temporal difference (TD) methods such as Sarsa [Sutton, 1996] or Q-learning [Watkins and Dayan, 1992] learn to predict the best action to take by step-wise interactions with the environment. In particular, Q-learning has been shown to be effective in solving the traveling salesman problem [Gambardella and Dorigo, 1995] or learning to drive a bicycle [Randløv and Alstrøm, 1998]. However large or continuous state spaces limit their application to simple environments.

Recently, combining advances in deep learning and reinforcement learning has proved to be very successful in mastering complex tasks. A significant example is the combination of neural networks and Q-learning, resulting in "Deep Q-Learning" (DQN) [Mnih *et al.*, 2013], able to achieve human performance on many tasks including Atari video games [Bellemare *et al.*, 2013].

Learning from scratch and lack of interpretability impose some problems on deep reinforcement learning methods. Randomly initializing the weights of a neural network is inefficient. Furthermore, this is likely intractable to train the model in many domains due to a large amount of required data. Additionally, most RL algorithms cannot introduce external knowledge limiting their performance. Moreover, the impossibility to explain and understand the reason for a decision restricts their use to non-safety critical domains, excluding for example medicine or law. An approach to tackle these problems is to combine simple reinforcement learning techniques and external knowledge.

A powerful recent idea to address the problem of computational expenses is to modularize the model into an ensemble of experts [Lample and Chaplot, 2017], [Bougie and Ichise, 2017]. The task is divided into a sequence of stages and for each one, a policy is learned. Since each expert focuses on learning a stage of the task, the reduction of the actions to consider leads to a shorter learning period. Although this approach is conceptually simple, it does not handle very complicated environments and environments with a large set of actions.

Another technique is called *Hierarchical Learning* [Tessler *et al.*, 2017], [Barto and Mahadevan, 2003] and is used to solve complex tasks, such as "simulating human brain" [Lake *et al.*, 2016]. It is inspired by human learning which uses previous experiences to face new situations. Instead of learning directly the entire task, different sub-tasks are learned by the agent. By reusing knowledge acquired from the previous subtasks, the learning is faster and easier. Some limitations are the necessity to re-train the model which is time-consuming and problems related to the catastrophic forgetting of knowledge on previous tasks. All the previously cited approaches suffer from lack of interpretation reducing their usage in critical applications such as autonomous driving.

An approach, *Symbolic Reinforcement Learning* [Garnelo *et al.*, 2016], [d'Avila Garcez *et al.*, 2018] combines a system that learns an abstracted representation of the environment and high-order reasoning. However this has several limitations, it cannot support ongoing adaptation to a new environment and cannot handle external sources of prior knowledge.

This paper demonstrates that a simple reinforcement learning agent can overcome these challenges to learn control policies. Our model is trained with a variant of the Sarsa(λ)

algorithm [Singh and Sutton, 1996]. We introduce external knowledge by representing the states as rules. Rules transform the raw data into a compressed and high-level representation. To deal with the problem of training speed and highly fluctuating environments [Dundar *et al.*,], we use a sub-states mechanism. Sub-states allow a more frequent update of the Q-values thereby smooth and speed-up the learning. Furthermore, we adapted eligibility traces which turned out to be critical in guiding the algorithm to solve tasks.

In order to evaluate our method, we constructed a variety of trading environment simulations based on real stock market data. Our rule-based approach, Sarsa-rb(λ), can learn to trade in a small number of iterations. In many cases, we are able to outperform the well-known Deep Q-learning algorithm in term of quality of policy and training time. Sarsa-rb(λ) also exhibits higher performance than DDPG [Lillicrap *et al.*, 2015] after converging.

The paper is organized as follows. Section 2 gives an overview of reinforcement learning. Section 3 describes the main contributions of the paper. Section 4 presents the experiments and the results. Section 5 presents the main conclusions drawn from the work.

2 Reinforcement Learning

Reinforcement learning 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 reward, 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 goal of reinforcement learning is to learn to select the action with the maximum return R_t achievable for a given observation [Sutton and Barto, 1998]. From Equation (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 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.

2.1 Q-learning algorithm

Q-learning [Watkins and Dayan, 1992] is a common technique to approximate $\pi \approx \pi^{*}$. The estimation of the action value function is iteratively performed 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:

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

The choice of the action follows a policy derived from Q. The most common policy called ϵ-greedy policy trade-off the exploration/exploitation dilemma. In case of exploration, a random action is sampled whereas exploitation selects the action with the highest estimated return. In order to converge to a stable policy, the probability of exploitation must increase over time. An obvious approach to adapting Q-learning to continuous domains is to discretize the state spaces, leading to an explosion of the number of Q-values. Therefore, a good estimation of the Q-values in this context is often intractable.

2.2 Sarsa algorithm

Sarsa is a temporal differentiation (TD) control method. The key difference between Q-learning and Sarsa is that Sarsa in an on-policy method. It implies that the Q-values are learned based on the action performed by the current policy instead of a greedy policy. The update rule becomes :

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

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

```
procedure SARSA($\mathcal{X}$, $A$, $R$, $T$, $\alpha$, $\gamma$)
    Initialize $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ uniformly
    while $Q$ is not converged do
        Start in state $s \in \mathcal{X}$
        Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        while $s$ is not terminal do
            Take action $a$, observe $r$, $s^{'}$
            Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\epsilon$-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 $Q$
```

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.

2.3 Eligibility trace

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 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 case of Sarsa(λ), this leads to the following update rule:

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

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).

3 Rule-based Sarsa(λ)

We first present the general idea of our algorithm, Sarsa-rb, a variant of the Sarsa algorithm.

We propose a simple method, Sarsa-rb, to enable Sarsa in continuous spaces boosted by injecting external knowledge. The idea behind Sarsa-rb is to enhance states representation and Q-values initialization with background knowledge to make training more efficient and interpretable. As in Sarsa, Sarsa-rb estimates the Q-values. However, each state is represented by a rule. There are various advantages of representing the states by rules. This makes possible combining reinforcement learning and complex knowledge. Furthermore, the number of Q-values is reduced, which makes the training much faster.

While Sarsa-rb provides some advantages over Sarsa in term of quality of policy, we can significantly improve their training time with a sub-states mechanism. Instead of updating one Q-value at each iteration, our model updates several Q-values which share similar information with the current state, leading to a significant speed-up. Finally, we adapt the eligibility trace λ technique to take advantage of the sub-states, Sarsa-rb(λ).

3.1 Rule-based Sarsa (Sarsa-rb)

The Sarsa algorithm maintains a parametrized Q-function which maps the states S to their Q-values. Instead of using as states the state space or a discretization of it, we enhance states representation by mapping rules and actions to Q-values. Depicted in Figure 1, states are replaced by a set of rules, R. The rules associate a pattern to an action and allow to introduce complex background knowledge.

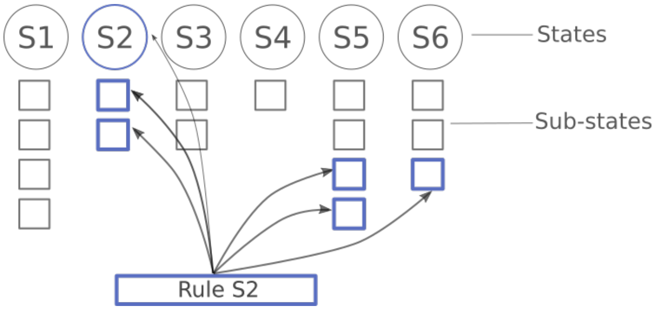


Figure 1: An illustration of the update of the Q-function. The Q-values of the states $s2$ and its sub-states are updated. The sub-states sharing similar information with $s2$, in blue, are also modified.

A pattern is a conjunction of variables which can be arbitrarily complex. 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, a rule recommends an action (e.g *brake*) for a pattern.

Given an observation obs_t, 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. In other words, this is the default state, active regardless of the input.

Our contribution here is to provide modifications to Sarsa which allow to improve states representation with background knowledge. The rules are a way to abstract the states from the environment and to deal with continuous or complex data representation. In addition, by taking advantage of the rules during the Q-values initialization the initial policy benefits from background knowledge. Moreover, in many domains $|R| << |S|$ resulting in a reduction of the number of Q-values to estimate. We filter out irrelevant rules by keeping only the most frequent ones.

In Sarsa, the Q-values are uniformly initialized. In a state s represented by a pattern p, p controls the activation of the state and we use the rule to improve the initialization of the Q-values:

$$Q(s_{t=0}, a_{t=0}) = \begin{cases} \mathcal{N}(\mu, \sigma^2), & \text{if } rule_{action} = a \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

with μ the mean and σ^2 the variance. The Q-value with the action recommended by the rule follows a normal distribution centered around μ and the other Q-values are initialized to 0.

3.2 Prior Knowledge for Rule Generation

To create the rules, we compared two methods. One consists of manually creating them according to our knowledge about the task. Automatic extraction retrieves patterns from external sources of data.

External Knowledge Based Rules

An intuitive approach to create the rules relies on human or background knowledge about domains. For example, if the task involves driving a car, background knowledge can be extracted from highway rules. The action associated with a pattern can be let empty if it cannot be predicted without much affecting the quality of the agent.

For example, we can use our expertise about time-series and stock markets. To deal with that, the rules can be based on candlestick patterns [Nison, 2001]. This stock-market technique estimates the trend of the share price by identifying patterns into the time series. Candlestick pattern analysis relies on patterns composed by the open, high, close and low prices of the previous observations.

Automatically Learned Rules

In real-world environments, the rules can be automatically captured by supervised machine learning methods. We follow a similar idea of [Mashayekhi and Gras, 2015]. The method extracts the rules from a random forest [Pal, 2005], an ensemble of decision trees [Safavian and Landgrebe, 1991]. A decision tree consists of several nodes that branch to two subtrees 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. Several ensemble models such as random forest reduce the variance by building many trees and predicting based on a consensus among decision trees. A simple tree traversal method can directly extract rules from the trees [Louppe, 2014].

3.3 Sub-states

In TD methods without eligibility traces, one Q-value of the current state s_t is updated at each iteration. Instead, we propose a technique to update the states which share similar information with s_t. We augment each Q-value with an ensemble of sub-states, sub_s. Since each state is represented by a pattern, we define the sub-states as its sub-patterns, the combinations of the variables. To avoid a too large number of sub-states, we limit the size of the sub-rules to conjunctions of at least 3 variables. The goal is to get most of the benefits of the shared information among the states while keeping the rest of the Sarsa algorithm intact and efficient. We provide modifications to Q-value estimation and update inspired by Sarsa which allow to use sub-states.

The estimation of a Q-value $Q^{'}(s,a)$ in Sarsa-rb takes into account the Q-value itself and the value of the sub-states :

$$Q^{'}(s,a) = Q(s,a) + \sum_{s^{'} \in sub_s} Q(s^{'},a) \qquad (9)$$

with sub_s the sub-states of a state s.

Figure 2 shows an example of a Q-value estimation. $Q(s^{'},a)$ 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. Note that we limit the weight of the term $Q(s^{'},a)$ in the $Q^{'}(s,a)$ estimation such as $Q(s^{'},a) << Q(s,a)$ to ensure convergence towards an optimal policy. We achieved this mechanism during the update step.

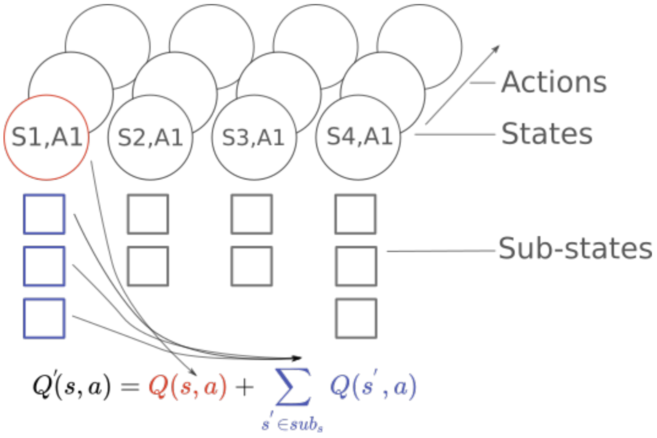


Figure 2: Estimation of a Q-value, $Q(s,a)^{'}$, with the sub-states technique. In addition to the Q-value $Q(s,a)$ itself, the sub-states values $Q(s^{'},a)$ are taken into account.

The update process propagates the reward to all similar sub-states, leading to a more frequent and early update of the states. Our approach to this problem is to increment the eligibility traces of the similar sub-states.

3.4 Eligibility Trace

Directly implementing Sarsa-rb proved to be slow to learn 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 s remains unchanged from Sarsa(λ) :

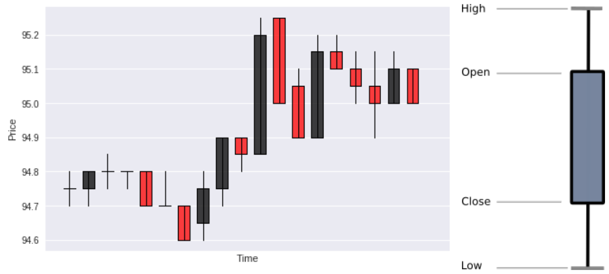
$$\begin{cases} E(s,a) = E(s,a) + 1 & \\ E(y,a) = E(y,a) + e^{-sim(y,s)}, & \text{if y is a sub-states of s} \\ E(y,a) = E(y,a) + \frac{e^{-sim(y,s)^2}}{K}, & \text{otherwise} \end{cases} \qquad (10)$$

$E(s)$ denotes the eligibility trace of the state s and $E(y)$ the eligibility trace of the sub-state y. We refer to $sim(y,s)$ as the similarity between the sub-state y and the state s. We compute the similarity score as the number of different variables between a sub-state y and a state s, $sim(y,s) = |y \cap s|$. 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 K. This constant should be positive and greater than zero. A high value leads to a small increase of the eligibility traces of the sub-states sharing only a few similar sub-patterns. Updates performed in this manner allow to estimate more accurately Q-values. Experiments also indicate that this method decreases the number of necessary visits and yield faster convergent policies.

4 Experiments

We evaluated Sarsa-rb(λ) on the OpenAI trading environment, a complex and fluctuating simulation from real stock market data. The agent observes the last stock price described by the open price, the close price, and the highest/lowest price during the one minute interval (Figure 3(b)). We limit the possible actions to *Buy*, *Hold* and *Sell*. The reward is computed according to the win/lose after buying or selling. We consider that a single agent has a limited impact on the stock market price, for this reason, the price is not influenced by the actions of the agent. 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.



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

(b) Structure of one observation

Figure 3: 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 data point, one observation from the environment

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. We trained the model on one stock index and we used the other datasets to generate the rules. Among the training examples, 80% are randomly selected for training the model and the remaining for testing it. We performed a grid-search to find the optimal 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 $K = 100$ as decay factor of eligibility traces. In 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. The patterns mined were filtered with C = 5, the minimum number of times a pattern occurs in the training data.

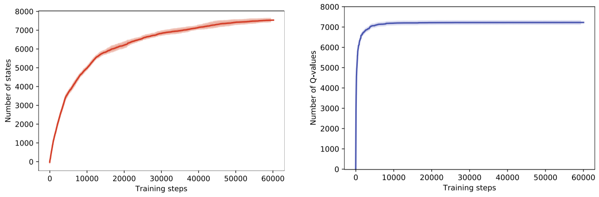
We follow a simplified technique used by [Mashayekhi and Gras, 2015] to generate rules from a random forest. Briefly, we extract the rules top to bottom (root to leaf) and filter the rules to avoid redundancy. In practice, we annotate 6000 samples into 3 classes. Each sample is the aggregation of the 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 built 20 trees with a maximum height of 4. In total, we retrieved 855 rules.

Table 1: The table compares performance in term of frequency of visit of the states. We compared Sarsa-rb with and without sub-states.

Settings	No sub-states	Sub-states
Average number of updates	376.789	5873.17
Average duration between two consecutive updates	11715.51	2189.31

We analyze the impact of the sub-states technique on the agent. Furthermore, we evaluate Sarsa-rb(λ) and compare the improvement with DQN and DDPG in terms of training speed and in terms of quality of policy.

4.1 Sub-states



(a) Sarsa-rb(λ) without sub-states

(b) Sarsa-rb(λ) with sub-states

Figure 4: Comparison of the average number of Q-values visited at least one time over 3 runs.

In order to better understand the impact of the sub-states on the learning, we analyze and compare Sarsa-rb(λ) with and without sub-states. We also investigate properties of the sub-states of the manually created rules.

Table 1 reports the number of times the Q-values are updated on average. We run the experiments 10 times for 500 episodes with the same hyper-parameters. The first row shows the average number of times states are updated and the second row shows the average number of steps between two consecutive updates of states. The states are updated $+1500\%$ with the sub-states technique and also the time between two updates is decreased. We observed that a frequent update of the sub-states leads to a faster convergence of the Q-values.

Figure 4 shows the number of states and its sub-states are updated at least once over time. At each iteration, we count the number of states or states with a sub-state visited. On average, states are updated for the first time much earlier when the sub-states technique is used. Sub-states play an important role for early updates and in the update frequency. Updating frequently the sub-states of a state improves the accuracy of estimation of its Q-values, which can significantly decrease learning time, especially when the number of states is large.

4.2 Overall Performance

We compared Sarsa-rb(λ) trained with the sub-states mechanism to a deep recurrent Q-learning model [Hausknecht and Stone, 2015] and a DDPG [Lillicrap *et al.*, 2015] model. For this evaluation, we individually tuned the hyper-parameters of each model. We decreased the learning rate from $\alpha = 0.3$ to $\alpha = 0.0001$, the eligibility trace from $\lambda = 0.9$ to $\lambda = 0.995$, and then used $\epsilon = 0.01$, $\lambda = 0.9405$ and $K = 100$ and we tuned the neural network architectures of DQN and DDPG. The results are obtained by running the algorithms with the same environment hyper-parameters. The plots are averaged over 5 runs. Finally, we used the manually created rules as the states of Sarsa-rb(λ).

We report learning curve on the testing dataset in Figure 5. Sarsa-rb(λ) always achieve a score higher than DQN and DDPG. As shown in Figure 5, Sarsa-rb(λ) clearly improves over DQN, we obtained an average reward after converging around 3.3 times higher. DDPG appears less fluctuating than Sarsa-rb(λ) but also less effective.

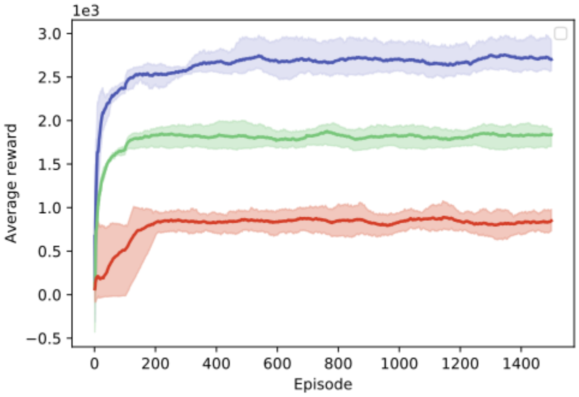


Figure 5: Performance curves for a selection of algorithms: original Deep Q-learning algorithm (red), Deep Deterministic Policy Gradients algorithm (green) and Sarsa-rb(λ) (blue).

5 Conclusion

This paper introduced a new model to combine reinforcement learning and external knowledge. We demonstrated its ability to solve complex and highly fluctuating tasks, trading in stock market. Additionally, this algorithm is fully interpretable and understandable. In a given state, we can explain the impact of each variable and the patterns on the action selection. Our central thesis is to enhance states representation of Sarsa(λ) with background knowledge and speed up learning with a sub-states mechanism. Further benefits stem from efficiently updating eligibility traces. Moreover, our approach can be easily adapted to solve new tasks with a very limited amount of human work. We have demonstrated the effectiveness of our algorithm to decrease the training time and to learn a better and more efficient policy. In the future, we are planning to evaluate our idea with other TD methods. Another challenge is how to generate the rules during the training phase and discard the useless rules to decrease learning time and improve computational efficiency. Finally, we are interested in extending our experiments to new environments such as textual or visual environments.

References

[Barto and Mahadevan, 2003] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[Bellemare *et al.*, 2013] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. 2013.

[Bougie and Ichise, 2017] N. Bougie and R. Ichise. Deep Reinforcement Learning Boosted by External Knowledge. *ArXiv e-prints*, December 2017.

[d'Avila Garcez *et al.*, 2018] A. d'Avila Garcez, A. Resende Riquetti Dutra, and E. Alonso. Towards Symbolic Reinforcement Learning with Common Sense. *ArXiv e-prints*, April 2018.

[Dundar *et al.*,] Murat Dundar, Balaji Krishnapuram, Jinbo Bi, and R Bharat Rao. Learning classifiers when the training data is not iid.

[Gambardella and Dorigo, 1995] Luca M Gambardella and Marco Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. In *Machine Learning Proceedings 1995*, pages 252–260. Elsevier, 1995.

[Garnelo *et al.*, 2016] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[Hausknecht and Stone, 2015] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. 2015.

[Lake *et al.*, 2016] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pages 1–101, 2016.

[Lample and Chaplot, 2017] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of AAAI*, pages 2140–2146, 2017.

[Lillicrap *et al.*, 2015] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[Louppe, 2014] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014.

[Mashayekhi and Gras, 2015] Morteza Mashayekhi and Robin Gras. Rule extraction from random forest: the rf+hc methods. In *Proceedings of Canadian Conference on Artificial Intelligence*, pages 223–237. Springer, 2015.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Nison, 2001] Steve Nison. *Japanese candlestick charting techniques: a contemporary guide to the ancient investment techniques of the Far East*. Penguin, 2001.

[Pal, 2005] Mahesh Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222, 2005.

[Randløv and Alstrøm, 1998] Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of ICML*, volume 98, pages 463–471, 1998.

[Safavian and Landgrebe, 1991] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[Singh and Sutton, 1996] Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[Sutton, 1996] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[Tessler *et al.*, 2017] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 1553–1561, 2017.

[Watkins and Dayan, 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.