Tutor4RL: Guiding Reinforcement Learning with External Knowledge

Mauricio Fadel Argerich, Jonathan Fürst, Bin Cheng
NEC Laboratories Europe
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
{mauricio.fadel@neclab.eu, jonathan.fuerst@neclab.eu, bin.cheng@neclab.eu}

Abstract

We introduce Tutor4RL, a method to improve reinforcement learning (RL) performance during training, using external knowledge to guide the agents' decisions and experience. Current approaches of RL need extensive experience to deliver good performance, something that is not acceptable in many real systems when no simulation environment or considerable previous data are available. In Tutor4RL, external knowledge– such as expert or domain knowledge– is expressed as programmable functions that are fed to the RL agent. During its first steps, the agent uses these knowledge functions to decide the best action, guiding its exploration and providing better performance from the start. As the agent gathers experience, it increasingly exploits its learned policy, eventually leaving its tutor behind. We demonstrate Tutor4RL with a DQN agent. In our tests, Tutor4RL achieves more than 3 times higher reward in the beginning of its training than an agent with no external knowledge.

Introduction

Reinforcement Learning (RL) has achieved great success in fields such as robotics (Kober, Bagnell, and Peters 2013), recommender systems (Theocharous, Thomas, and Ghavamzadeh 2015) and video games, in which RL has even surpassed human performance (Mnih et al. 2015). However, before achieving this, RL often poor performance for an extended time—possibly for millions of iterations—until it has gathered enough experience. In practice, this problem is approached mainly with two RL techniques: (1) training via simulation and (2) learning from historical data.

Recently, researchers have applied RL to computer systems tasks, such as database management system configuration (Schaarschmidt et al. 2018) or container orchestration for Big Data systems (Fadel Argerich, Cheng, and Fürst 2019). Here, experience data for the specific task might not be available (e.g., the complexity of a data analytic task depends on the processed dataset, not known before) and the context of the agent changes drastically with each deployment (e.g., the execution environment is different). Because of this, data or simulation training are not feasible. Thus, the agent needs to gather its experience online, from the performance of a live system where each action the agent explores has a real cost that impacts the system. To apply RL in such scenarios, an agent needs to provide "good-enough" performance from the start and act safely throughout the learning (Dulac-Arnold, Mankowitz, and Hester 2019)—a fundamental problem that must be solved to make RL applicable to real world use cases.

However, in our application of RL for container orchestration in Big Data systems, we have experienced that some knowledge of the environment is usually available before the deployment of the agent. This knowledge includes common heuristics or notions, well known to domain experts (e.g., DevOps engineers). In our work, we aim to make this knowledge accessible to the RL agent. Our intuition is that just as a person prepares for a task beforehand by gathering knowledge from sources such as other human-beings, manuals, etc., we can provide external knowledge that the RL agent can use when not enough experience has been gathered.

To realize this concept, we introduce *Tutor4RL*, a method that guides RL via external knowledge. In Tutor4RL, external knowledge– such as expert or domain knowledge – is expressed as programmable functions that the agent will use during its training phase, and especially its initial steps to guide its behavior and learning. Thanks to Tutor4RL, the agent can perform in a reliable way from the start and achieve higher performance in a shorter training time. As the RL agent gathers more experience, it learns its policy and leaves the tutor behind, improving on its results thanks to its empirical knowledge. We take some inspiration from the recently proposed *Data Programming* concept for supervised learning, in which functions developed by experts, are used to encode weakly supervision sources (Ratner et al. 2017).

We test our approach with a DQN Agent in the Atari game Breakout and compare it to a plain DQN Agent. In our tests, Tutor4RL achieves more than a 3 times higher reward than the plain agent in the beginning of its training, and keeps up with its performance even after leaving the tutor behind, i.e. only using its learned policy.

Background and Motivation

As illustrated in Figure 1 (Sutton, Barto, and others 1998), in Reinforcement Learning (RL), an agent learns to control an unknown environment to achieve a certain goal while interacting with the environment. The agent interacts with the environment in discrete time steps $t = 0, 1, ...$ In each step, the agent receives a representation $s_t \in S$ of the state of environment and a numerical signal r_t called reward, and performs an action $a_t \in A$ that leads to the state s_{t+1} and reward r_{t+1}, perceived by the agent in the next time step. S and A represent the sets of states and actions respectively.

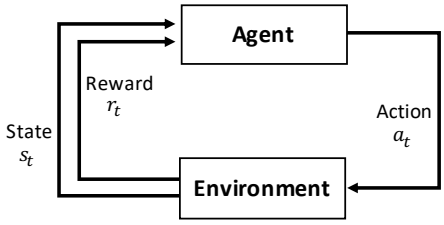

Figure 1: The RL framework and its elements.

The agents' behaviour is defined by its policy π, which provides a mapping from states S to actions A. The value function $q_\pi(s, a)$ represents the expected future reward received when taking action a at state s with a policy π. The goal of the agent is to find a policy that maximizes cumulative reward in the long run.

To do this, the agent learns from its experience for each performed action a_t, and then uses the collected observations (s_{t+1}, r_{t+1}) to optimize its policy π based on different models of the value function, such as a Tabular model (Sutton, Barto, and others 1998) or a deep neural network model (Mnih et al. 2015). Existing studies show that RL can lead to a reasonable model for determining which action to take in each state after learning from a large number of experience data. However, the biggest problem is how the RL agent can learn fast and efficiently from experience data.

There have been different approaches of addressing this problem in the state of the art. A simple approach is to explore the state space randomly, but this approach is usually time-consuming and costly when the state/action space is large. The drawback of this approach has been reported by our previous study (Fadel Argerich, Cheng, and Fürst 2019) in the case of leveraging RL to automatically decide the configuration and deployment actions of a data processing pipeline in a cloud and edge environment.

Another approach is to gain experience via simulation. With enough computational resources, we can easily produce lots of experience data in a short time, but it is difficult to ensure that the simulated experiences are realistic enough to reflect the actual situations in the observed system.

Recently, a new trend to leverage external knowledge to improve the exploration efficiency of RL agents has emerged. For example, in (Moreno et al. 2004) and (Hester et al. 2018), prior knowledge like pre-trained models and policies are used to bootstrap the exploration phase of a RL agent. However, this type of prior knowledge still originates in previous training and is limited by the availability of such data.

Instead of relying on any pre-trained model, we explore how to utilize a set of programmable knowledge functions to guide the exploration of a RL agent so that we can quickly make effective decisions, even just after a few exploration steps. We call our method Tutor4RL. Unlike existing approaches, Tutor4RL requires no previous training and is therefore, a more practical approach for the use of RL in real systems. To the best of our knowledge, Tutor4RL is the first to apply programmable knowledge functions into RL for improving training performance and sample efficiency.

Tutor4RL

Figure 2 shows our overall design of Tutor4RL. As compared to traditional RL, we add a new component called *Tutor* to guide the agent to make informed decisions during training. The reason why the tutor is able to guide the agent is because it can directly leverage a set of *knowledge functions* defined by domain experts. In this way, Tutor4RL can help the agent avoid any blind decisions at the beginning. The tutor possesses external knowledge and interacts with the agent during training. The tutor takes as input the state of the environment, and outputs the action to take; in a similar way to the agent's policy. However, the tutor is implemented as programmable functions, in which external knowledge is used to decide the mapping between states and actions, e.g., for Atari Breakout, the tutor takes the frame from the video game as input, and outputs in what direction the bar should be moved. For every time step, the tutor interacts with the agent and gives advise to the agent for making better decisions, based on all provided knowledge functions.

One issue for the agent to consider is when and how often it should ask for advise from the tutor. In a similar fashion to Epsilon greedy exploration, we define τ as the threshold parameter for the agent to control when it will take the suggested actions from the tutor instead of its own decisions. τ is a parameter of our model and the best value to initialize it depends on the use case; thus its initial value is left to be decided during implementation.

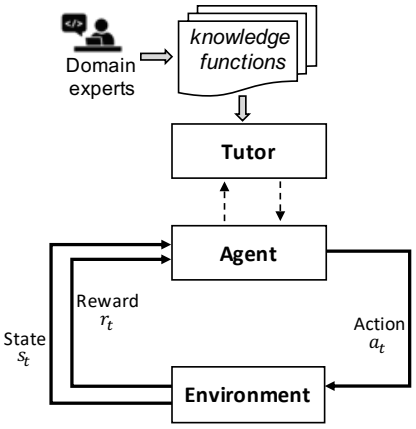

Figure 2: Overall Working of Tutor4RL

Knowledge functions must be programmed by domain experts and allow them to easily bring different types of domain knowledge into the tutor component. Currently, Tutor4RL considers the following two types of knowledge functions: *Constrain Functions* and *Guide Functions*.

Constrain Functions are programmable functions that constrain the behavior of the agent. At each time step t, a constrain function takes the state of the environment as input, and returns a vector to indicate whether an action in the action space could be taken or not using the value 1 or 0; 1 represents the action is enabled while 0 represents the action is disabled and cannot be performed for this state. Therefore, constrain functions provide a mask to avoid unnecessary actions for certain states.

Guide Functions are programmable functions that express domain heuristics that the agent will use to guide its decisions, especially in moments of high uncertainty, e.g. the start of the learning process. Each guide function takes the current RL state and reward as input, and outputs a vector to represent the weight of each preferred action according to the encoded domain knowledge.

The benefit of Tutor4RL is twofold:

1. During training, the tutor enables a reasonable performance, opposed to the unreliable performance from an inexperienced agent, while generating experience for training. Furthermore, the experience generated by the tutor is important because it provides examples of good behavior.
2. The knowledge of the tutor does not need to be perfect or extensive. The tutor might have partial knowledge about the environment, i.e. know what to do in certain cases only, or might not have a perfectly accurate knowledge. The tutor provides some "rules of thumb" that the agent can follow during training, and based on experience, the agent can improve upon these decisions, achieving a higher reward than the tutor.

Evaluation

We implement Tutor4RL by modifying the DQN (Mnih et al. 2015) agent, using the library Keras-RL (Plappert 2016) along with Tensorflow. In order to make our evaluation reproducible, we choose a well-known workload for RL: playing Atari games. In particular, we select the Atari game Breakout, using the environment BreakoutDeterministic-v4 from OpenAI Gym (Brockman et al. 2016). We compare our approach to a standard DQN agent as implemented by Keras-RL and we use the same set of parameters for both, the DQN agent with Tutor4RL and the one without. The parameters used for the agents are detailed in Table 1.

In the BreakoutDeterministic-v4 environment, the observation is a RGB image of the screen, which is an array of shape $(210, 160, 3)$ and four actions are available: no operation, fire (starts the game by "throwing the ball"), right and left. Each action is repeatedly performed for a duration of $k = 4$ frames. In order to simplify the state space of our agent, we pre-process each frame converting it to greyscale and reducing its resolution to $(105, 105, 1)$.

We implement a simple guide function that takes the preprocessed frame, locates the the ball and the bar in the X axis, and returns "fire" if no ball is found, or to move in the direction of the ball (left or right) if the ball is not above the bar. The simplified code for this function can be seen in Listing 1. In addition, we also include the simplified code the agent uses in each step of its training in Listing 2, choosing between the tutor decision and the policy decision.

Parameter	DQN	DQN + Tutoring
Policy	Epsilon greedy	
Epsilon	[0.3-0.1] decreasing linearly through 0 to 1.75M steps	
Gamma	0.99	
Warmup steps	50000	
Optimizer	Adam with lr=0.00025	
Tau	-	[1-0] decreasing linearly through 0 to 1.5M

Table 1: Parameters used in evaluation.

```
def guide_function(obs):
    # Find bar and ball in frame.
    bar_x_left, bar_x_right = \
                        find_bar_edges_x(obs)
    ball_x = find_ball(observation)
    if ball_x != None:
        # Where to move bar.
        if bar_x_left > ball_x:
            return [0, 0, 0, 1] # left
        elif bar_x_right < ball_x > 0:
            return [0, 0, 1, 0] # right
    return [0, 1, 0, 0] # fire
```

Listing 1: Our implementation of a guide function for Breakout. We check the position of the ball and the bar in the current frame and move the bar towards the ball.

```
def select_action(obs, tau, guide_function):
    if numpy.random.uniform() < tau:
        # Use tutor.
        tf_output = guide_function(obs)
        action = numpy.argmax(tf_output)
    else:
        # Use policy normally.
        action = policy.select_action()
    return action
```

Listing 2: Selection of action when training the agent.

Figure 3 depicts the mean reward per episode of the plain DQN agent and DQN agent with Tutor4RL during training. During the beginning of the training and until step $500,000$, the plain DQN Agent shows an expected low reward (< 15 points) because it starts with no knowledge, while the DQN Agent with Tutor4RL—thanks to the use of its tutor knowledge—manages to achieve a mean reward between 15 and 35 points, ca. double the maximum of the plain DQN Agent. From step $500,000$ we see how the plain DQN agent starts to improve, but its not until step 1.1M that the plain DQN agent shows equally good results as the tutored one. From there we see a similar reward for both agents,, with DQN Agent + Tutor4RL achieving a slightly higher mean reward in most cases. Because τ is decreased uniformly throughout training, the tutor is used less as training progresses. Finally, in step 1.5M, $\tau = 0$ and the tutor is

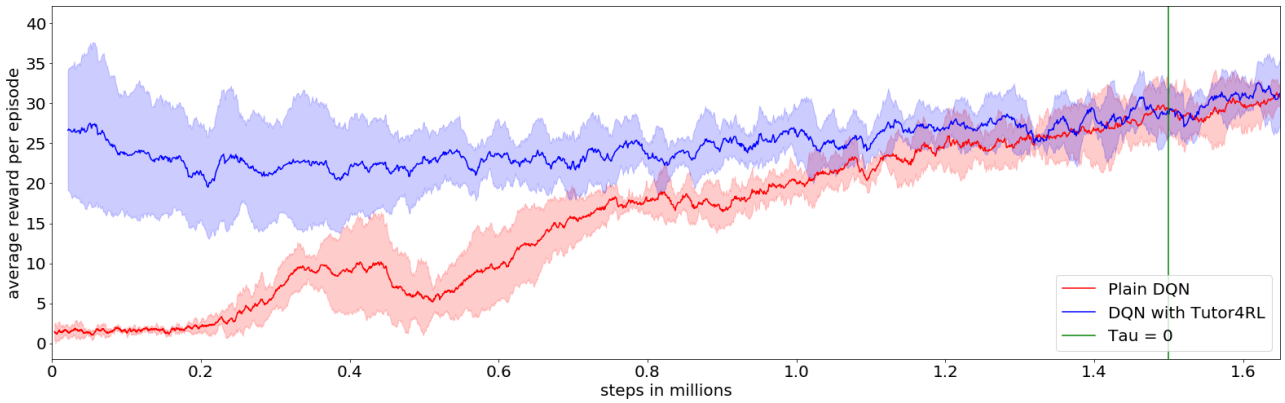

Figure 3: Average mean reward per episode achieved by plain DQN agent and DQN agent with Tutor4RL during training. Data was averaged over 4 tests for each agent and with a rolling mean of 20 episodes, bands show 0.90 confidence interval.

no longer used. It is important to note that from this point on, the reward does not decrease but it keeps improving with the agent's learning. Moreover, we test both agents after 1.75M steps: the plain DQN agent achieves an average reward of 40.75 points while Tutor4RL achieves a reward of 43. Note that this reward comes only from the learned policy of the agents and keeping $\epsilon = 0.05$, i.e. no tutor knowledge is used.

Discussion and Future Work

Tutor4RL is ongoing research and as such we plan to develop several improvements. First, the decision about when to use the tutor is an important aspect to our approach. Currently, we are using τ as a parameter that decreases as the agent gathers more experience in each step. However, this can be improved to take into account the actual learning of the agent, i.e. how "good" are the actions selected by the policy, or how certain the policy is about the action to be taken in the given state. Second, as discussed before, we plan to evaluate constrain functions to limit the behavior of the agent. This can help in situations in which an action does not make sense, e.g. in Breakout, moving the bar to the left when it's already in its most left position. Last, the accuracy of guide functions can vary and thus, the decisions of the Tutor can be improved by weighting the decisions of the functions according to their accuracy.

Conclusion

We have demonstrated Tutor4RL, a method that uses external knowledge functions to improve the initial performance of reinforcement learning agents. Tutor4RL targets deployment scenarios where historical data for training is not available and building a simulator is impractical. Our results show that Tutor4RL achieves 3 times higher reward than an agent without using external knowledge in its initial stage.

Acknowledgments

The research leading to these results has received funding from the European Community's Horizon 2020 research and innovation programme under grant agreement n° 779747.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Dulac-Arnold, G.; Mankowitz, D.; and Hester, T. 2019. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

Fadel Argerich, M.; Cheng, B.; and Fürst, J. 2019. Reinforcement learning based orchestration for elastic services. *arXiv preprint arXiv:1904.12676*.

Hester, T.; Vecerik, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Horgan, D.; Quan, J.; Sendonaris, A.; Osband, I.; et al. 2018. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11):1238–1274.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518.

Moreno, D. L.; Regueiro, C. V.; Iglesias, R.; and Barro, S. 2004. Using prior knowledge to improve reinforcement learning in mobile robotics. *Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK*.

Plappert, M. 2016. keras-rl. https://github.com/keras-rl/keras-rl.

Ratner, A.; Bach, S. H.; Ehrenberg, H.; Fries, J.; Wu, S.; and Ré, C. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment* 11(3).

Schaarschmidt, M.; Kuhnle, A.; Ellis, B.; Fricke, K.; Gessert, F.; and Yoneki, E. 2018. Lift: Reinforcement learning in computer systems by learning from demonstrations. *arXiv preprint arXiv:1808.07903*.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge.

Theocharous, G.; Thomas, P. S.; and Ghavamzadeh, M. 2015. Personalized ad recommendation systems for life-time value optimization with guarantees. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.