KoGuN: Accelerating Deep Reinforcement Learning via Integrating Human Suboptimal Knowledge

Peng Zhang[1,2] , **Jianye Hao**[1,2,3,*] , **Weixun Wang**[1] , **Hongyao Tang**[1] , **Yi Ma**[1] , **Yihai Duan**[1] and **Yan Zheng**[1]
[1]College of Intelligence and Computing, Tianjin University
[2]Noah's Ark Lab, Huawei
[3]Tianjin Key Lab of Machine Learning
{zhangpeng123, jianye.hao, wxwang, bluecontra, mayi, duanyihai, yanzheng}@tju.edu.cn

Abstract

Reinforcement learning agents usually learn from scratch, which requires a large number of interactions with the environment. This is quite different from the learning process of human. When faced with a new task, human naturally have the common sense and use the prior knowledge to derive an initial policy and guide the learning process afterwards. Although the prior knowledge may be not fully applicable to the new task, the learning process is significantly sped up since the initial policy ensures a quick-start of learning and intermediate guidance allows to avoid unnecessary exploration. Taking this inspiration, we propose knowledge guided policy network (KoGuN), a novel framework that combines human prior suboptimal knowledge with reinforcement learning. Our framework consists of a fuzzy rule controller to represent human knowledge and a refine module to fine-tune suboptimal prior knowledge. The proposed framework is end-to-end and can be combined with existing policy-based reinforcement learning algorithm. We conduct experiments on both discrete and continuous control tasks. The empirical results show that our approach, which combines human suboptimal knowledge and RL, achieves significant improvement on learning efficiency of flat RL algorithms, even with very low-performance human prior knowledge.

1 Introduction

Deep reinforcement learning (DRL) algorithms have been applied in a range of challenging domains, from playing games [Mnih *et al.*, 2015; Silver *et al.*, 2016] to robotic control [Schulman *et al.*, 2015a]. The combination of RL and high-capacity function approximators such as neural networks holds the promise of automating a wide range of decision making and control tasks, but application of these methods in real-world domains has been hampered by the challenge of sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more.

In human's learning process, they rarely learn to master a new task from scratch. In contrast, human naturally leverage knowledge obtained in similar tasks to derive an rough policy and guide the learning process afterwards. Although these knowledge may be not completely compatible with the new task, human can adjust the policy in the following learning process. As a result, human can learn much faster than RL agents. Therefore, integrating human knowledge into reinforcement learning algorithms is promising to boost the learning process.

Combining human knowledge has been studied before in supervised learning [Collobert *et al.*, 2011; Fischer *et al.*, 2019; Garcez *et al.*, 2012; Hu *et al.*, 2016]. An important line of works that leverage human knowledge in sequential decision-making problem is imitation learning [Ho and Ermon, 2016; Pomerleau, 1991; Hester *et al.*, 2017]. Imitation learning leverages human knowledge by learning from expert trajectories that collected in the same task that we aim to solve. Demonstration data is the concrete instance of human knowledge under a certain task and can be seen as low-level representation of human knowledge. We expect to leverage high-level knowledge (e.g., common sense) to assist learning under unseen tasks (thus no demonstration data). Moreover, high-level knowledge can be easily shared with other similar tasks.

To leverage human knowledge, a major challenge is obtaining the representation of the provided knowledge. Under most circumstances, the provided knowledge is imprecise and uncertain, and even cover only a small part of the state space. As a consequence, the conventional approaches such as classical bivalent logic rules do not provide an adequate model for modes of reasoning which are approximate rather than exact [Zadeh, 1965; Zadeh, 1996]. In contrast, fuzzy Logic, which may be viewed as an extension of classical logical systems, provides an effective conceptual framework for dealing with the problem of knowledge representation in an environment of uncertainty and imprecision. In this paper, we propose a novel knowledge guided policy network (KoGuN), which can integrate human knowledge into RL algorithms in an end-to-end manner. The proposed policy framework con-

[*]Corresponding author.

sists of a trainable knowledge controller and a refine module. On one hand, the knowledge controller contains a set of fuzzy logic rules to represent human knowledge which can be continuously optimized together with the whole policy; on the other hand, the refine module undertakes the role of refining the provided suboptimal and imprecise knowledge. Finally, the combination of the knowledge controller and the refine module helps ”warm-start” the learning process and enables the agent to learn faster. Generally, the proposed framework can be combined with existing policy-based reinforcement learning algorithms to accelerate their learning processes. We evaluate our method on discrete and continuous control tasks and the experimental results show that our approach achieves significant improvement on learning efficiency of RL algorithms.

2 Preliminary

2.1 Deep Reinforcement Learning

An Markov decision process (MDP) can be defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition function, $\mathcal{R}$ is the reward function and γ is the discount factor [Sutton and Barto, 2018]. An RL agent interacts with the MDP following a policy $\pi\left(a_t|s_t\right)$, which is a mapping of state space to action space. The agent aims to learn a policy that maximize the expected discounted total reward. The state value function $V_\pi(s)$ is an estimate of the expected future reward that can be obtained from state s when following policy π: $V_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|S_t = s\right]$

Several effective policy-based algorithms have been proposed in the literature. TRPO [Schulman *et al.*, 2015a] and ACKTR [Wu *et al.*, 2017] both update the policy subject a constraint in the state-action space (trust region). Proximal policy optimization (PPO) [Schulman *et al.*, 2017] designs a clipped surrogate objective that approximates the regularization. PPO aims to maximize the following objective function:

$$J(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_{\theta_{old}}}\Bigg[min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t(s_t,a_t),\right. \left. clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t(s_t,a_t)\right)\Bigg] \tag{1}$$

Here $\hat{A}_t(s_t, a_t)$ is an estimator of the advantage function at timestep t and in this paper we use generalized advantage estimation (GAE) [Schulman *et al.*, 2015b] to calculate the advantage function. When dealing with tasks with discrete action space, the policy network usually outputs the action distribution by applying $softmax$ on the logits. For continuous action space, the policy network normally outputs mean and standard deviation of a Gaussian distribution.

2.2 Fuzzy Logic and Fuzzy Rules

Fuzzy logic is based on fuzzy set theory [Zadeh, 1965]. A fuzzy set is an extension of the classical notion of crisp set. Crisp sets allow only full membership or no membership at all. Compared with crisp sets, fuzzy sets allow partial membership. An element may partially belong to a fuzzy set. The membership of an element x in a crisp set S can be described by a membership function $\mu_S(x)$, where:

$$\mu_S(x) = \begin{cases} 1 & x \in S \\ 0 & x \notin S \end{cases} \tag{2}$$

But for a fuzzy set F, the membership of x in it can be described by a membership function $\mu_F(x)$ with range from 0 to 1:

$$\mu_F : X \to [0, 1] \tag{3}$$

where X refers to the universal set defined in a specific problem.

Operations in classical crisp set theory can also be extended to fuzzy set theory. Assuming that F_1 and F_2 are two fuzzy sets. μ_{F_1} and μ_{F_2} are membership functions of F_1 and F_2 respectively. The $union$ of F_1 and F_2 is a fuzzy set whose membership function is $\mu_{F_1\cup F_2}(x) = max[\mu_{F_1}(x), \mu_{F_2}(x)]$. The $intersection$ of F_1 and F_2 is the fuzzy set with membership function $\mu_{F_1\cap F_2}(x) = min[\mu_{F_1}(x), \mu_{F_2}(x)]$.

A fuzzy rule is usually in the form of ‘IF X is A and Y is B THEN Z is C’. ‘X is A’ and ‘Y is B’ are called preconditions of the fuzzy rule and ‘Z is C’ is called conclusion of the rule. X, Y and Z are variables. A, B and C are fuzzy sets which are normally called as linguistic values. A fuzzy rule takes the observation values of X and Y as input and outputs the value of Z. To get the reasoning conclusion of a fuzzy rule, we first calculate the truth value T of each precondition. Then applying conjunction operator to these truth values, we finally get the conclusion’s strength w (w can also be seen as the satisfaction level of the rule):

$$w = min(T_1, T_2) = min[\mu_A(x_0), \mu_B(y_0)] \tag{4}$$

Here x_0 and y_0 are observation values for X and Y. T_1 and T_2 are truth values for preconditions ‘X is A’ and ‘Y is B’. The $minimum$ operator is used here as conjunction operator. Finally, to get the value of Z, we need to match the conclusion’s strength on the domain of Z. One common used matching method is applying the inverse function of the membership function of ‘C’ on the strength:

$$z = \mu_C^{-1}(w) \tag{5}$$

Note that inverse functions can be defined only for monotonic functions, and in this paper we use simple monotonic linear functions as membership functions of conclusions. Other matching methods can be found in [Yager and Zadeh, 2012].

2.3 Knowledge Representation Using Fuzzy Rules

Human common sense knowledge is often imprecise and uncertain. Conventional knowledge representation approaches such as hard rules, which are based on classical bivalent logic, are inappropriate to represent this type of knowledge [Zadeh, 1996]. Fuzzy logic was motivated in large measure by the need for a conceptual framework which can address the issues of uncertainty and lexical imprecision and thus are suitable for representing human imprecise knowledge.

To illustrate how human knowledge could be translated into fuzzy rules, consider an example of learning to drive a

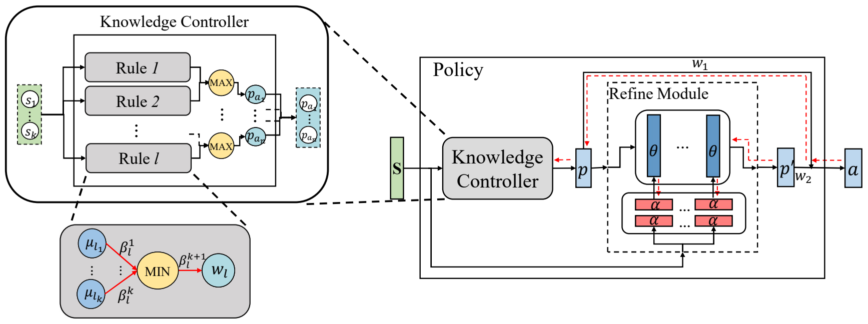


Figure 1: The overall architecture of knowledge guided policy network. It consists of a knowledge controller for human knowledge representation and a refine module (*dashed box*) for policy refinement. *red dashed lines* indicate the gradient flow.

car. When the learner sits in the driver's seat and observes that the car is off the road to the right, he would follow the common sense and turn the steering wheel to the left to keep the car in the middle of the road. Nevertheless, he is unaware of how many degrees precisely should the steering wheel be rotated. The only thing he is sure is that the farther the car is off the middle of road, the larger angle of the steering wheel should be rotated. Alongside the learning process continues, the learner gradually learns an optimal policy on how to drive a car. The prior knowledge of keeping the car in the middle of road can be expressed in the form of 'IF...THEN...', for instance, 'IF the car's deviation from the middle of road to the right is large, THEN the steering wheel should be turned to the left'. But we do not have an accurate definition of 'large'. Therefore, we can use a fuzzy set to define 'large'. Hence, the common sense mentioned before can be translated into a fuzzy rule: 'IF D_r is $large$ THEN $Action$ is $left$'. Here D_r represents the car's deviation distance from the middle of road to the right. $large$ is a fuzzy set, and its membership function can be defined as a simple linear function, for example, $y = clip[(0.5x - 1), 0, 1]$. Such a membership function means that the larger the deviation value x is, the greater the membership y is, resulting in a greater probability of turning the steering wheel to the left.

3 Knowledge Guided Policy Network

In this section, we propose a novel end-to-end framework to leverage human knowledge where the priors are continuously optimized with the whole policy. The proposed policy framework is called knowledge guided policy network (KoGuN) and the overall architecture of KoGuN is shown in Figure 1. KoGuN consists of a trainable knowledge controller and a refine module. In Section 3.1, we describe the architecture of the knowledge controller which undertakes the role of incorporating human knowledge into the policy framework in an end-to-end manner. In Section 3.2, we introduce the architecture of the refine module, which fine-tunes the prior knowledge since the rule knowledge is normally suboptimal and even cover only a small part of the state space. Finally in Section 3.3, we demonstrate how to combine the refine module and the knowledge controller, forming a complete policy framework.

3.1 Knowledge Controller

Knowledge controller module plays the role of knowledge representation. However, there is no guarantee that provided human knowledge can fit perfectly into the current task, leading to knowledge mismatch problem. The proposed knowledge controller alleviates the problem by introducing trainable weights.

As shown in Figure 1 ($top\ left$), the knowledge controller $\phi(s)$, containing a few fuzzy rules translated from knowledge provided by human, takes state information as input and outputs an action preference vector $\boldsymbol{p}$, which represents the tendency of the controller in a state. Each rule corresponds to one action and has the form of:

- $Rule\ l$: IF S_1 is M_{l_1} and S_2 is M_{l_2} and ... and S_k is M_{l_k} THEN $Action$ is a_j

Here S_i are variables that describe different parts of system states. M_{l_i} are fuzzy sets corresponding to S_i. The conclusion of the rule indicates the corresponding action a_j. Now assume that we have observation values $s_1 ... s_k$ for $S_1 ... S_k$, we can get truth values for each precondition by applying membership functions to the observed values: $\mu_{l_i}(s_i)$, where μ_{l_i} are membership functions of the fuzzy set M_{l_i}. Hence, the strength of $Rule\ l$ can be calculated as described in Section 2.2. To mitigate the knowledge mismatch problem, inspired by [Berenji, 1992], we add trainable weights β to each rule, which enable the knowledge controller to learn to adapt to current task. For each rule l there are $k + 1$ weights corresponding to it, here k is the number of the preconditions. $\beta_l^1, ..., \beta_l^k$ are assigned to each precondition and β_l^{k+1} is assigned to the entire rule. The adjustment of the weights $\beta_l^1, ..., \beta_l^k$ entails the adjustment of corresponding membership functions and the weight β_l^{k+1} implies the importance of the rule. With trainable weights, the knowledge controller can be optimized like a neural network. The trainable weights are labelled as red lines in Figure 1 ($bottom\ left$). Finally, the strength of $Rule\ l$ can be calculated by:

$$w_l = \beta_l^{k+1} min[\beta_l^1 \mu_{l_1}(s_1), \beta_l^2 \mu_{l_2}(s_2), ..., \beta_l^k \mu_{l_k}(s_k)] \quad (6)$$

In this paper, these trainable weights are initialized to 1 at beginning for not disturbing the prior knowledge. Note that the trainable weights can also be set according to prior knowledge. For example, if the confidence corresponding to one of the provided rules is high, the rule then can be assigned a higher weight.

For discrete action space, the rule strength can be seen as this rule's preference for the corresponding action. Different rules cover different parts of the state space. Thus the relation of different rules for the same action is or and the preference p_{a_i} for action a_i can be calculated by using $maximum$ operator to take the largest strength value of those rules corresponding to action a_i. Finally, we get the action preference vector $\boldsymbol{p}$:

$$\boldsymbol{p} = \phi_\beta(s) = [p_{a_1}, p_{a_2}, ..., p_{a_{|\mathcal{A}|}}] \tag{7}$$

The vector $\boldsymbol{p}$ represents all the rules' preference for actions in a state.

For continuous action space, we need to further map the rule strength to a continuous action value. As described in Section 2.2, one common used method is applying the inverse membership function of the conclusion. As an example, consider an n-dimensional continuous action space. In this paper, one rule is designed only for one of the n dimensions and we assume that Rule l is designed for the dth dimension. The reasoning result of Rule l can be calculated by:

$$p_d = \mu_{a_j}^{-1}(w_l) \tag{8}$$

If there are more than one rules for the same action dimension, the weighted average of these rules' action value weighted by the rule strength is used as the final output of these rules. Finally, we get the action preference vector $\boldsymbol{p}$:

$$\boldsymbol{p} = \phi_\beta(s) = [p_1, p_2, ..., p_d, ..., p_n] \tag{9}$$

3.2 Knowledge Refine Module

The knowledge controller, representing the prior knowledge provided by human, is only a very rough policy that may not cover the whole state space. To obtain an optimal or near optimal policy, the knowledge controller needs to be extended and further refined. We import a refine module $g(\cdot)$ to refine and complete the rough rule-based policy. The refine module $g(\cdot)$ should take the preference vector $\boldsymbol{p}$ as input and output the refined action preference vector $\boldsymbol{p'}$. The refined process could be regarded as correction and complement of the rough rule-based policy, which may only cover parts of the whole state space. How to refine $\boldsymbol{p}$ should be based on current state s. Here we propose two alternative ways to approximate the refine module $g(\cdot)$. The most straightforward idea is to approximate $g(\cdot)$ using a neural network, i.e., concatenate state and preference vector as the input of the neural network:

$$\boldsymbol{p'} = g_\theta(s, \boldsymbol{p}) \tag{10}$$

Here θ is the parameters of the neural network.

However, the refinement of the knowledge controller can be different in different states, which means that the mapping from the action preference vector $\boldsymbol{p}$ to the refined preference vector $\boldsymbol{p'}$ can be different during the change of states. This requires the refine module $g_\theta(\cdot)$ to change drastically as the state s changes. The parameters θ is required to capture such complex relation. One previous work [Schmidhuber, 1992] shows that using a learning feed-forward network to generate weights for a second network is suitable for this kind of context-dependent function. Thus we proposed that approximating the refine module $g(\cdot)$ using hypernetworks [Ha *et al.*, 2016], which leverage the idea of [Schmidhuber, 1992]. Moreover, using a feed-forward network to generate weights for another netowrk, hypernetworks are more in line with the semantics of the refine module. The first network takes state as input and generates weights for the second one. The second network takes action preference vector $\boldsymbol{p}$ as input and refines it, finally outputs the refined action preference vector $\boldsymbol{p'}$. This is exactly in accordance with the semantic relationship that the refine module should refine the action preference according to the state. As shown in the dashed box in Figure 1, each hypernetwork takes the state s as input and produces the weights and biases of one layer of $g(\cdot)$. Formally:

$$\boldsymbol{p'} = g_\theta(\boldsymbol{p}) \tag{11}$$

where:

$$\theta = h_\alpha(s) \tag{12}$$

Here $h_\alpha(\cdot)$ is the hypernetwork that takes state as input and output the weights θ for the refine module $g_\theta(\cdot)$. In both methods, a $sigmoid$ function is used finally to normalize the output of the refine module $g_\theta(\cdot)$.

3.3 Integrating Knowledge Controller and Refine Module

To leverage prior knowledge, we use the rough policy given by the knowledge controller to guide the agent to interact with environments at early stage. Therefore, as demonstrated in Figure 1, a weighted sum between $\boldsymbol{p}$ and $\boldsymbol{p'}$ is used as the final preference vector. At the early stage of the training phase, $\boldsymbol{p}$ provided by the knowledge controller, accounting for a larger proportion. As the training continues, the proportion of $\boldsymbol{p'}$ gradually increases. For tasks with discrete action space, applying $softmax$ to the weighted sum, we get the final output of KoGuN:

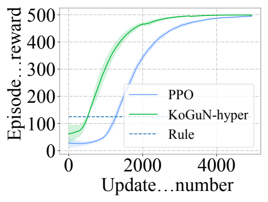
$$a \sim softmax(\frac{w_1\boldsymbol{p} + w_2\boldsymbol{p'}}{\tau}) \tag{13}$$

where $w_1 + w_2 = 1$. Here τ is the temperature to sharpen the action distribution. And for tasks with continuous action space, the mean of the action Gaussian distribution is the weighted sum of $\boldsymbol{p}$ and $\boldsymbol{p'}$:

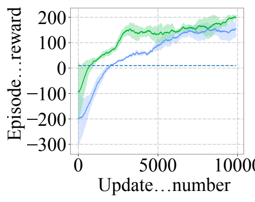
$$a \sim \mathcal{N}(w_1\boldsymbol{p} + w_2\boldsymbol{p'}, \sigma) \tag{14}$$

The weight corresponding to the knowledge controller w_1 decays linearly to a small value as the training continues. At beginning, the refine module is not well trained so the gradient flow from the refine module may harm the learning of the knowledge controller. To stabilize training process, we prohibit the gradient from the refine module back propagating to the knowledge controller and for forcing the refine module to adapt to the knowledge controller. $red\ dashed\ lines$ in Figure 1 denote the backward gradient flow.

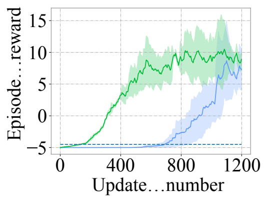
KoGuN is an end-to-end policy framework, thus it can be combined with any policy-based algorithm. In this paper, we use proximal policy optimization (PPO) [Schulman *et al.*, 2017] as our basic RL algorithm.



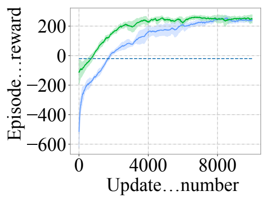
(a) $CartPole$



(b) $LunarLander$



(c) $FlappyBird$



(d) $LunarLanderCon$

Figure 2: Experimental results for PPO without KoGuN (*blue*), KoGuN with hypernetworks for refine module (*green*) and pure knowledge controller (*dashed line*) on four tasks. The shaded region denotes standard deviation of average evaluation over 5 trials.

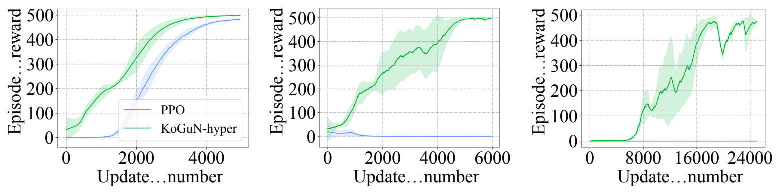


Figure 3: Experimental results in the setting of sparse reward. We set d=50 ($left$), 100 ($middle$) and 250 ($right$) in the task $CartPole$.

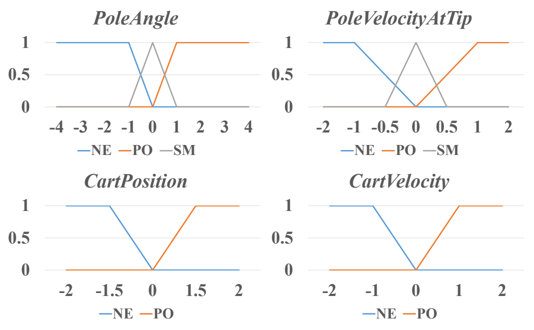


Figure 4: Membership functions used in $CartPole$.

4 Experiments

In this section, we firstly evaluate our algorithm on four tasks in Section 4.1 : $CartPole$ [Barto and Sutton, 1982], $LunarLander$ and $LunarLanderContinuous$ in OpenAI Gym [Brockman *et al.*, 2016] and $FlappyBird$ in PLE [Tasfi, 2016]. Moreover, we show the effectiveness and robustness of KoGuN in sparse reward setting in Section 4.2. Then we finally conduct ablation study to better understanding the contribution of each part of our framework in Section 4.3.

The experimental setup is as follows: for all the tasks, we use Adam optimizer [Kingma and Ba, 2014] with a learning rate of 1×10^{-4} and the temperature $\tau = 0.1$. The discounted factor γ is set to 0.99 and the GAE λ is set to 0.95. The policy is updated every 128 timesteps. For PPO without KoGuN, we use a neural network with two full-connected hidden layers as policy approximator. For KoGuN with normal network (KoGuN-concat) as refine module, we also use a neural network with two full-connected hidden layers for the refine module. For KoGuN with hypernetworks (KoGuN-hyper), we use hypernetworks to generate a refine module with one hidden layer. Each hypernetwork has two hidden layers. All hidden layers described above have 32 units. w_1 is set to 0.7 at beginning and decays to 0.1 in the end of training phase.

4.1 Evaluation

We use the environments mentioned above as our evaluation environments. We designed some rules for each task. Because of space limitation, here we only give the rules used in $CartPole$ as an example. In $CartPole$, a pole is attached by an un-actuated joint to a cart. The system is controlled by applying a force to the cart. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units away from the center. The state is represented by a four-dimensional vector $[CartPosition, CartVelocity, PoleAngle, PoleVelocityAtTip]$, which ranges from $[-2.4, -Inf, -15, -Inf]$ to $[2.4, Inf, 15, Inf]$. There are two available actions $[p, n]$: push the cart towards positive direction or negative direction. Totally 6 rules are used in the experiment and they are listed below:

- $Rule$ 1: IF $PoleAngle$ is NE and $PoleVelocityAtTip$ is NE THEN $Action$ is n.
- $Rule$ 2: IF $PoleAngle$ is PO and $PoleVelocityAtTip$ is PO THEN $Action$ is p.
- $Rule$ 3: IF $PoleAngle$ is SM and $PoleVelocityAtTip$ is NE THEN $Action$ is n.
- $Rule$ 4: IF $PoleAngle$ is SM and $PoleVelocityAtTip$ is PO THEN $Action$ is p.
- $Rule$ 5: IF $PoleAngle$ is SM and $PoleVelocityAtTip$ is SM and $CartPosition$ is NE and $CartVelocity$ is NE THEN $Action$ is p.

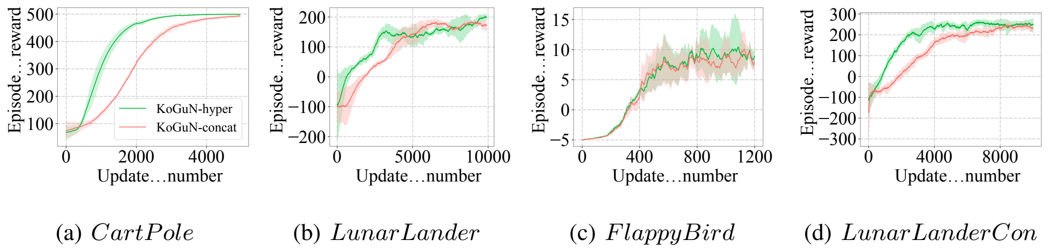


Figure 5: Experimental results for KoGuN with hypernetworks for refine module (KoGuN-hyper) and KoGuN with normal neural networks for refine module (KoGuN-concat) on four tasks.

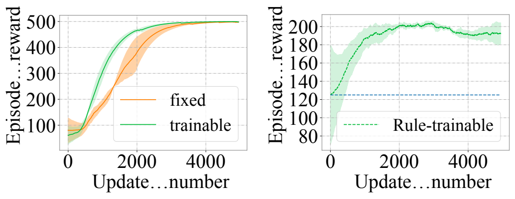


Figure 6: KoGuN with trainable knowledge controller $vs.$ KoGuN with fixed knowledge controller ($left$). Performance of knowledge controller during training ($right$).

- $Rule$ 6: IF $PoleAngle$ is SM and $PoleVelocityAtTip$ is SM and $CartPosition$ is PO and $CartVelocity$ is PO THEN $Action$ is n.

NE, PO and SM are fuzzy sets which means $negative$, $positive$ and $small$ respectively. Their membership function are shown in Figure 4. These rules are not well-crafted and are easy to understand. $Rule$ 1-$Rule$ 4 are designed to maintain the balance of the pole. $Rule$ 5 and $Rule$ 6 are designed to prevent the cart from moving out of the restricted range.

Figure 2 shows the experimental results on the four tasks. We can see that KoGuN can converge with much less update number. We also plot mean performance of pure knowledge controller (dash line in Figure 2). We can see that KoGuN can dramatically accelerate the learning process even with such poor performance rule knowledge, both in discrete control and continuous control domain.

4.2 Sparse Reward Setting

We further demonstrate the effectiveness of KoGuN in the setting of sparse reward in this section. We consider a sparse reward setting: multi-step accumulated rewards are given at sparse time steps. To simulate this setting, we deliver d-step accumulated reward every d time steps in $CartPole$. We evaluate our algorithm under different delayed steps. Figure 3 plots the results under sparse reward setting.

KoGuN shows superior performance in sparse reward setting. As the increase of delay step d, PPO without KoGuN can hardly learn a effective policies because the chance for the agent to get a reward signal is quite small. In contrast, with human knowledge, KoGuN enables the agent to avoid unnecessary exploration. As a consequence, the probability that the agent can obtain reward signals is greatly improved. Therefore, agents with KoGuN can learn effective policies even with very sparse reward signals.

4.3 Ablation Study

Hypernetworks as refine module: To compare the performance of the two types of refine module described in Section 3.2, we conduct ablation experiments on all the four tasks. The experiment results are shown in Figure 5. As described in Section 3.2, hypernetworks can more easily capture the complex relationship between the action preference vector $\boldsymbol{p}$ and the refined action preference vector $\boldsymbol{p'}$ in different states than normal neural networks. As a result, KoGuN with hypernetworks shows overall better performance than KoGuN with normal networks.

Trainable knowledge controller: We conduct another group of experiments on $CartPole$ to demonstrate the benefits of trainable knowledge controller. We compare trainable knowledge controller and fixed knowledge controller in Figure 6(a). The performance of trainable controller during training is given in Figure 6(b), which means that the prior knowledge is constantly optimized during training. Trainable controller enables the provided knowledge to adapt as much as possible to the current task. As a result, agent can make better use of these knowledge and learns faster.

5 Conclusion

In this paper, we propose a novel policy network called KoGuN to leverage human knowledge to accelerate the learning process of RL agents. The proposed framework consists of a knowledge controller and a refine module. The knowledge controller represents human suboptimal knowledge using fuzzy rules and the refine module refines the imprecise prior knowledge. The policy framework is end-to-end and can be combined with existing policy-based algorithms to deal with tasks with both discrete action space and continuous action space. As future work, we would like to investigate the knowledge representation method of more challenging tasks, such as tasks with high-dimensional visual data as state space.

Acknowledgements

The work is supported by the National Natural Science Foundation of China (Grant Nos.: 61702362, U1836214) and the new Generation of Artificial Intelligence Science and Technology Major Project of Tianjin under grant: 19ZXZNGX00010.

References

[Barto and Sutton, 1982] Andrew G Barto and Richard S Sutton. Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research*, 4(3):221–235, 1982.

[Berenji, 1992] Hamid R Berenji. A reinforcement learning-based architecture for fuzzy logic control. *International Journal of Approximate Reasoning*, 6(2):267–292, 1992.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[Collobert *et al.*, 2011] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[Fischer *et al.*, 2019] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. Dl2: Training and querying neural networks with logic. In *Proceedings of International Conference on Machine Learning*, pages 1931–1941, 2019.

[Garcez *et al.*, 2012] Artur S d'Avila Garcez, Krysia B Broda, and Dov M Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2012.

[Ha *et al.*, 2016] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv:1609.09106*, 2016.

[Hester *et al.*, 2017] Todd Hester, Matej Vecerík, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017.

[Ho and Ermon, 2016] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[Hu *et al.*, 2016] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv:1603.06318*, 2016.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[Mnih *et al.*, 2015] V Mnih, K Kavukcuoglu, D Silver, A. A. Rusu, J Veness, M. G. Bellemare, A Graves, M Riedmiller, A. K. Fidjeland, and G Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Pomerleau, 1991] Dean Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[Schmidhuber, 1992] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[Schulman *et al.*, 2015a] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of International Conference on Machine Learning*, pages 1889–1897, 2015.

[Schulman *et al.*, 2015b] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv:1506.02438*, 2015.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[Silver *et al.*, 2016] D Silver, A. Huang, C. J. Maddison, A Guez, L Sifre, den Driessche G Van, J Schrittwieser, I Antonoglou, V Panneershelvam, and M Lanctot. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Tasfi, 2016] Norman Tasfi. Pygame learning environment. https://github.com/ntasfi/PyGame-Learning-Environment, 2016.

[Wu *et al.*, 2017] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.

[Yager and Zadeh, 2012] Ronald R Yager and Lotfi A Zadeh. *An introduction to fuzzy logic applications in intelligent systems*, volume 165. Springer Science & Business Media, 2012.

[Zadeh, 1965] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[Zadeh, 1996] L. A. Zadeh. Knowledge representation in fuzzy logic. In *Fuzzy Sets, Fuzzy Logic, And Fuzzy Systems: Selected Papers*, pages 764–774. World Scientific, 1996.