Proceedings of the 2022 IEEE
International Conference on Robotics and Biomimetics
December 5-9, 2022, Xishuangbanna, China

Structure-Aware Policy to Improve Generalization among Various Robots and Environments

Wei Xu[1], Yue Gao*[2] and Buqing Nie[3]

Abstract— **Recently, Deep Reinforcement Learning (DRL) has been used to solve complex robot control tasks with outstanding success. However, previous DRL methods still exist some shortcomings, such as poor generalization performance, which makes policy performance quite sensitive to small variations of the task settings. Besides, it is quite time-consuming and computationally expensive to retrain a new policy from scratch for new tasks, hence restricts the applications of DRL-based methods in the real world. In this work, we propose a novel DRL generalization method called GNN-embedding, which incorporates the robot hardware and the environment simultaneously with GNN-based policy network and learnable embedding vectors of tasks. Thus, it can learn a unified policy for different robots under different environment conditions, which improves the generalization performance of existing DRL robot policies. Multiple experiments on the Hopper-v2 robot are conducted. The experimental results demonstrate the effectiveness and efficiency of GNN-embedding on generalization, including multi-task learning and transfer learning problems.**

I. INTRODUCTION

With the growing interest and development in deep reinforcement learning (DRL), we have seen remarkable progresses and achievements of DRL in various application scenarios, including video games [1], autonomous vehicles [2], and robotics control [3]–[5]. Despite the huge success, existing DRL-based methods are still quite limited on generalization [6], which restricts real applications of DRL methods, especially in robot control tasks. The policy π highly depends on the parameter settings of the task, thus it can only learns how to control a single robot in a single environment at one time. For example, given robot control tasks with different robot hardware implementations (link length, etc.) and environment features (friction coefficient, etc.), we are required to train multiple policies, despite the similarity among these tasks, which is quite computationally expensive and time-consuming. Therefore, it is essential to design a unified DRL method, which can be utilized across various robots and environments simultaneously.

This work is supported by the National Key Research and Development Program of China (Grant No. 2021YFF0307900), National Natural Science Foundation of China (Grant No. 61903247), and Shanghai Municipal Science and Technology Major Project (Grant No. 2021SHZDZX0102).

[1]Wei Xu is with Department of Automation, Shanghai Jiao Tong University, Shanghai, 200240, P.R. China xuweirosa@sjtu.edu.cn

[2]Yue Gao is with MoE Key Lab of Artificial Intelligence and AI Institute of Shanghai Jiao Tong University, Shanghai, 200240, P.R. China yuegao@sjtu.edu.cn

[3]Buqing Nie is with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, P.R. China niebuqing@sjtu.edu.cn

*Corresponding author.

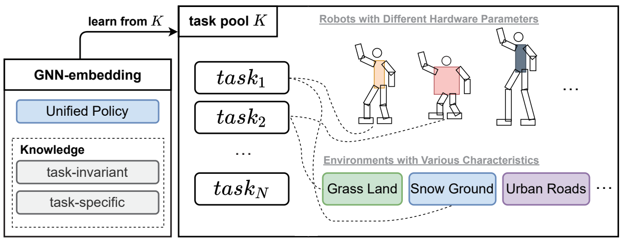

Fig. 1. The visualization of the generalization task and GNN-embedding composition. The task pool K contains robots with different hardware parameters and environments with various characteristics. Each task is a combination of a robot and an environment. GNN-embedding is a unified policy to control multiple tasks, which is able to learn both the task-invariant knowledge and the task-specific knowledge.

Recently, a number of studies have tried to address this limitation. Chen et al. [7] proposed a control method HCP across different robot structures, which represents the robot hardware as vector and can control robots with similar kinematics. Wang et al. [8] designed a novel GNN-based policy NerveNet, which leverages the structure information of the robot, thus can transfer well-trained policies to other robots with similar structures. Huang et al. [9] constructed a unified policy Shared Modular Policies (SMP) composed of a series of control modules according to robot actuators. SMP can can control robots with different morphologies.

Despite the excellent methods mentioned above, previous works mainly focus on the robot hardware and make few attempt to consider the task environments, which makes DRL generalization tasks in real application scenarios still challenging. For example, as shown in Figure 1, we are given a series of walking robots with similar physical structures and several different terrain conditions, including snow, grassland, and urban roads. We are required to design a unified policy for robot walking tasks under different terrain conditions, which is quite common in the real applications. Ideally, the policy can control any robot to walk continuously from one kind of terrain to another smoothly, such as from snow to grassland.

In this paper, we propose a new DRL generalization method called **GNN-embedding** for the challenging robot control task mentioned above. Different from existing methods, our method takes variety of the robot hardware implementation and the environment into consideration simultaneously utilizing a novel GNN-based policy network. Besides, GNN-embedding adopts learnable embedding vectors to learn the latent representations of the task features, including the robot hardware implementation and the environment.

Therefore, our method enables the robot to make decisions conditioned on the current robot hardware and the environment features automatically, which is essential to improve generalization of the DRL method. Experiments on the Hopper-v2 robot [10] demonstrate that GNN-embedding can control robots with different hardware parameters in different environments with single policy only. Besides, it outperforms the baseline methods on the generalization performance.

The main contributions of this paper are as follows:

- We explore a broader generalization problem in robot control tasks: controlling different robots with similar structure under different environment settings, which is quite common in real applications but still challenging for existing methods.
- A new DRL generalization method GNN-embedding is proposed. It incorporates robot hardware and the environment simultaneously into generalization with GNN-based policy network and learnable embedding vectors. Thus, our method can improve generalization performance of DRL methods in robot control tasks.
- Experiments on the Hopper-v2 robot are conducted. The results demonstrate that GNN-embedding outperforms existing methods on the generalization performance, including multi-task and transfer learning tasks.

II. RELATED WORK

A. Reinforcement Learning

Reinforcement learning (RL) [11] is a powerful method to solve sequential decision problems, which implement learning by interacting with the task environment. RL models the environment as the Markov Decision Process (MDP) and aims to obtain the optimal control policy by maximizing the cumulative reward. DRL combines RL with deep learning and has been utilized in various application scenarios [1], [2]. Recently, DRL shows great success in robot control problems [3]–[5]. Despite the achievement, most RL methods can only handle one task each time, and are sensitive to the task parameters because of poor generalization performance. In this work, we will propose a new DRL algorithm GNN-embedding to improve generalization performance.

B. Multi-task Learning and Transfer Learning

Multi-task learning focuses on how to train one single model to accomplish multiple tasks, while transfer learning considers how to transfer a model from the training tasks to the new tasks. They are two essential problems to measure generalization performance. Chen et al. [7] propose a hardware conditioned control method called HCP-I that can control a class of robots with similar physical structure. Devin et al. [12] decompose the policy into the robot module and the task module. The well-trained modules can be grouped together and transferred to a new control problem. Research works also concern the multi-task learning and transfer learning among tasks [13], data domains [14], etc. However, previous studies pay little attention to the situation in which both the robot hardware and the environmental change in a control problem. This paper deals with the above robot control problem among various robots and environments. GNN-embedding treats the control problem as a multi-task learning challenge and trains one unified model to control the multiple tasks simultaneously.

C. Graph Neural Networks

In order to process graph-structured data, researchers proposed the graph neural networks (GNN), which could extract graph-structured features through information propagation and aggregation [15]–[17]. Besides, GNNs have a wide range of applications in various fields, such as natural language translation [18], building recommender systems [19], molecules studies in chemistry [20], etc. Recently, some works have tried to apply GNN in robot control tasks. Wang et al. [8] propose an algorithm called NerveNet, in which they innovatively apply GNN to implement the policy network in DRL. Sanchez-Gonzalez et al. [21] propose a network architecture called Graph Networks (GN) that build a graph2graph mapping and work with model predictive control (MPC) to do sequential decision in robot control problems. This paper also uses GNN to exploit physical structure information of robots. Our work applies a GNN architecture that better fits the robot control problems in order to identify latent patterns in robot control problems and improve generalization performance among various robots and environments.

III. METHODOLOGY

A. Problem Formulation

We formulate the multi-task robot control problem as follows. We are given N_R robots $R = \{r_i \mid 1 \leq i \leq N_R\}$ and N_M environments $M = \{m_j \mid 1 \leq j \leq N_M\}$. The robots R share similar physical structure with different hardware parameters, such as link lengths and body masses, etc. The environments M are also similar with some different parameters, such as the ground friction, etc. A robot control task can be seen as a combination of a robot $r_i \in R$ and an environment $m_j \in M$. Therefore, $N_R \times N_M$ robot control tasks are obtained and form a task pool: $K = \{k_{ij} = (r_i, m_j) \mid 1 \leq i \leq N_R, 1 \leq j \leq N_M\}$. Each task $k_{i,j} \in K$ is an individual RL task, thus can be modeled as MDP:$< S, A, P, R, \gamma >$, where S, A, P, R, γ denote state space, action space, transition function, reward function, discount factor correspondingly.

In this work, we are required to train a unified policy network $\pi_\theta : S \to A$ for all tasks K. Therefore, the optimal policy can be described as follows:

$$\pi_\theta^* \leftarrow \arg\min_\theta \mathbb{E}_\pi \mathbb{E}_{k\sim K} \left[-\sum_{\tau=0}^{\infty} \gamma^\tau R(s_\tau, a_\tau) \right] \quad (1)$$

B. GNN-embedding Model Framework

In order to build a unified policy across robot hardware and environment settings, both task-invariant knowledge (general patterns for robot tasks) and task-specific knowledge (robot-specific knowledge and environment features) are essential, corresponding to "GNN" and "embedding" designs of our method respectively.

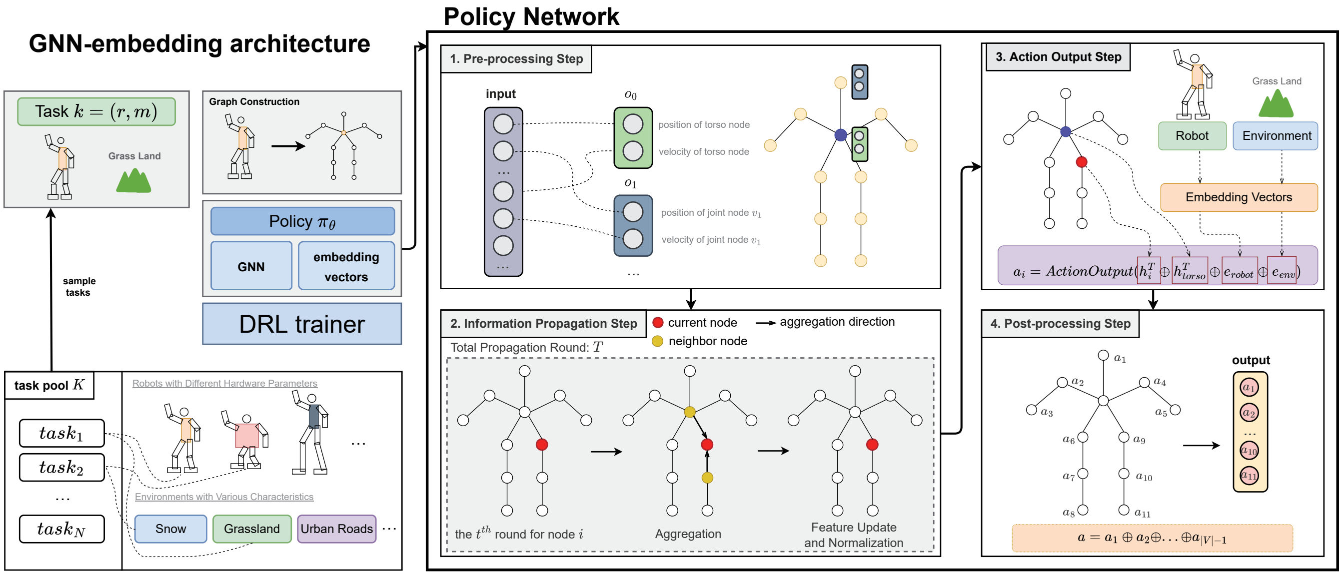

Fig. 2. The detailed illustration of our method GNN-embedding. GNN-embedding utilizes GNN-based policy network and embedding vectors. It learns a unified policy that can be applied to control multiple tasks with different robot hardware parameters and various environmental characteristics. In the policy network, the pre-processing step splits the observation vector o according to graph nodes. The information propagation step leverages GNN to update the feature vector iteratively. The action output step computes the action control signal for each joint node. The post-processing step concatenates the actions of the joint nodes and outputs the action vector a.

1) GNN: learn task-invariant knowledge: GNN-embedding applies GNN to integrate structural information of the robots and aims to recognize the general decision patterns for the control problem. For the convenience of the following steps, an undirected graph $G = (V, E)$ is constructed according to the skeleton of the robot. As illustrated in Figure 3, the joints of the robot with actuators are modeled as joint nodes $V = \{v_i \mid 1 \leq i \leq |V|\}$, while the links are modeled as edges $E = \{(v_i, v_j)\}$ connecting the corresponding joint nodes. The base coordinate of the robot is modeled as the torso node v_0, which is connected to the adjacent joint nodes.

Besides, GNN-embedding employs GNN to implement the policy network and the value network in the DRL framework. The policy π_θ makes decisions for each joint node separately. The information propagation of GNN is similar to the interaction between the robot joints, thus can be utilized to simulate robot motions. More details of the implementations are given in Section III-C.

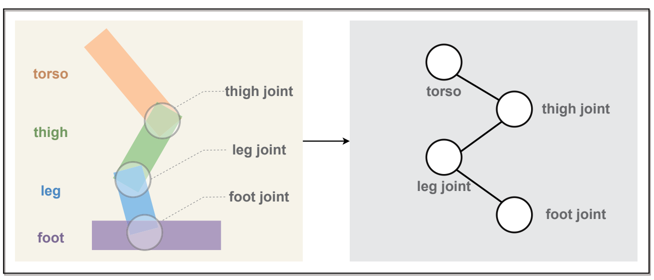

Fig. 3. The graph construction process of the Hopper-v2 robot.

2) Embedding: learn task-specific knowledge: In this problem, two task-specific knowledge are required: the hardware parameters of the robot and the environmental characteristics, which are represented by two learnable embedding vectors e_i^{robot} and e_j^{env} in our method. More specifically, each robot $r_i \in R$ is represented by a unique embedding vector e_i^{robot} to characterize its hardware properties. Similarly, each environment $m_j \in M$ is represented by an embedding vector e_j^{env}. During training in the task $k_{ij} = (r_i, m_j)$, the corresponding embedding vectors e_i^{robot} and e_j^{env} are extracted and concatenated to the observation. GNN-embedding learns two embedding vectors via gradient descent during training. More training details are shown in Section III-E.

3) GNN-embedding: As mentioned above, our method can learn task-invariant knowledge and task-specific knowledge simultaneously, thus can be trained to control several robots in several environments, i.e. solve the multi-task problem proposed in Section III-A. Besides, it is unnecessary to train the model in all the $N_R \times N_M$ tasks, while $max\{N_R, N_M\}$ tasks are sufficient. It can generalize to unseen combinations of the robots and environments. More details are given in the experiment.

C. Policy Network

As shown in Figure 2, the policy network contains the following four steps.

1) Pre-processing Step: The observation vector o received contains the position and velocity of each joint. In this step, for node i in the robot graph, we initialize the feature vector of the node i with observation vector: $h_i^0 = o_i$,where o_i denotes information of the robot joint i.

2) Information Propagation Step: This step of GNN-embedding is designed based on GraphSAGE [16] to propagate information and update the feature vector. The following two improvements are made based on the vanilla GraphSAGE for robot control problems:

1) The robot graph is quite small, thus the sampling steps are removed to reduce computational cost.
2) The single linear function in GraphSAGE is replaced with a multi-layer perceptron (MLP), which can improve the expression ability of the model.

The following are detailed descriptions. The information propagation is composed of T iterations. In the t-th iteration of the graph node i, our method aggregates information from its neighboring nodes with the aggregation function:

$$h_{N(i)}^{t} = Aggregate(\{h_j^{t-1}, \forall j \in N(i)\}) \tag{2}$$

, where $Aggregate$ function is designed as average pooling aggregator in this work, while max pooling aggregator, etc. are also suitable. Then the feature vector h_i^t is updated with the feature update function:

$$h_i^t = FeatureUpdate(h_i^{t-1}, h_{N(i)}^t) \tag{3}$$

, where $FeatureUpdate$ is implemented with MLP. Finally, the feature vector h_i^t is normalized:

$$h_i^t \leftarrow Normalize(h_i^t) = \frac{h_i^t}{\|h_i^t\|_2} \tag{4}$$

With the increment of the propagation iterations, each node i can receive information from nodes further away, making the area of the perception field become larger and larger.

3) Action Output Step: This step generates actions for joint nodes with the $ActionOutput$ function. For each joint node i, the action is generated based on the final feature vector h_i^T, h_{torso}^T, the embedding vector e_{robot} representing the robot hardware properties, and the embedding vector e_{env} representing the environment features. This step is described as follows:

$$a_i = ActionOutput(h_i^T \oplus h_{torso}^T \oplus e_{robot} \oplus e_{env}) \tag{5}$$

, where $ActionOutput$ is implemented by MLP, and a_i denotes action of joint i.

4) Post-processing Step: In the post-processing step, we concatenate the actions of joint nodes, and obtain the action vector a as the final output of the policy network:

$$a = a_1 \oplus a_2 \oplus ... \oplus a_{|V|-1}. \tag{6}$$

D. Value Network

The structure of the value network is similar to the policy network. The difference lies in the post-processing step. The output of the policy network is an action vector a, while the output of the value network is a scalar. Therefore, in the value network, the value output step computes the value estimates of each joint node. The post-processing step obtains the output of the value network by averaging these value estimates.

E. Training Algorithm of GNN-embedding

GNN-embedding is a DRL-based algorithm implemented based on PPO (Proximal Policy Optimization) [4], while other DRL training algorithms can also be applied. As shown in Algorithm 1, GNN-embedding is trained on a task pool K, where each task in the task pool $k = (r, m) \in K$ is a combination of a robot r and an environment m. In each iteration, the agent first samples a task from K randomly and collects trajectories through interaction on the task. During the optimization steps, the policy network π_θ and the value network are optimized with PPO trainer with mini-batch Stochastic Gradient Descent (mini-batch SGD). Meanwhile, the embedding vectors e_{robot}, and e_{env} are also updated by gradient descent:

$$\begin{aligned} e_{robot} &\leftarrow e_{robot} - \alpha \nabla_{e_{robot}} L(e_{robot}, \theta) \\ e_{env} &\leftarrow e_{env} - \alpha \nabla_{e_{env}} L(e_{env}, \theta) \end{aligned} \tag{7}$$

,where $L(\cdot, \theta)$ is the cost function for the policy network in the PPO algorithm. Eventually, the optimal policy π_θ^* together with the well-trained embedding vectors can be used to control all the tasks in the task pool K.

Algorithm 1 Training algorithm of GNN-embedding

```
initialize embedding vectors E and replay buffer B
for each training iteration do
    sample a task k = (r, m) ~ K
    obtain embedding vectors e_robot, e_env from E
    for each environment step t do
        a_t = π_θ(o_t, e_robot, e_env)
        o_{t+1}, r_t, done = env.step(a_t)
        Store (o_t, a_t, r_t, o_{t+1}, done) into buffer B
    end for
    for each gradient step do
        sample (o, a, r, o', done) ~ B
        update policy π_θ and value network
        update embeddings e_robot, e_env via Eq. 7
    end for
    store e_robot, e_env into E
end for
```

Besides, GNN-embedding can also be utilized for transfer learning problems. Some robots and environments are trained separately during training, but they have not been grouped together as a task in the task pool K. However, GNN-embedding is able to learn corresponding embedding vectors and transferred to the unseen task directly. More details are given in Section IV-C.

IV. EXPERIMENTS

A. Experimental Settings

The experiments are conducted on the Hopper-v2 [10] in OpenAI Gym [22] with MuJoCo [23] physics engine. Hopper is a two-dimensional one-legged jumping robot with four links and three joints with actuators. In this work, the agent is required to control the joint actuators of the robot so that it can jump forward quickly without falling. 20 different hopper robots with different hardware parameters (link lengths and the body masses, etc.) are constructed as the robot set R. 4 environments with different ground parameters (friction coefficient, etc.) are applied as the environment set M. The robots and environments are randomly grouped and form a task pool K.

B. Multi-task Learning

In this section, 20 different tasks are constructed as the task pool K, where each task corresponding to one robot. The agent is required to be trained on the 20 tasks and obtain a unified policy. We choose a mainstream DRL method PPO as the baseline to verify the effectiveness and overall performance of our method. As the training curves shown in Figure 4, the x-axis denotes the number of training iterations, while y-axis denotes the average reward on the 20 robot tasks. More results are described in Figure 5. In most of the training tasks, our method GNN-embedding outperforms baseline method PPO on the learning speed and final performance, which demonstrates the effectiveness of GNN-embedding on generalization. By exploiting GNN and two embedding vectors, GNN-embedding incorporates robot hardware features and environment features, thus provides an effective solution to deal with the multi-task learning problems. More details are shown in the video attached.

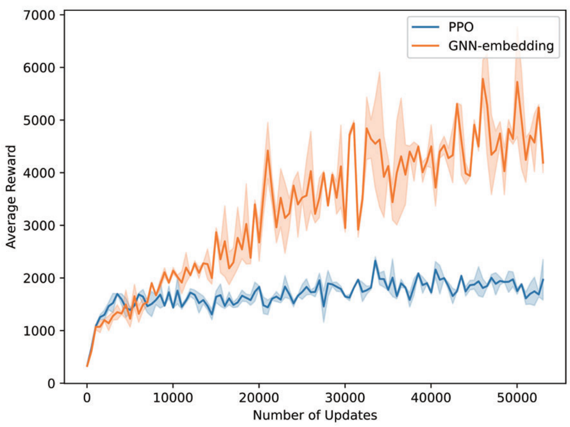

Fig. 4. The learning curves on 20 Hopper-v2 control tasks in the multi-task learning experiment.

We also conduct an experiment to measure the learning efficiency of the task-invariant knowledge. GNN-embedding utilizes GNN to learn the task-invariant knowledge, i.e. general control patterns for the tasks. During training, it can apply these knowledge to other undertrained tasks, which enables the model to achieve relatively better performance on these undertrained tasks. Therefore, we can measure generalization ability based on the performance gaps among all the tasks during training. More specifically, we find the five tasks with the highest rewards during training, and regard them as fully learned tasks (or top tasks). The average reward on the five well-trained tasks can be calculated, which is denoted as *Avg.Top*. Similarly, we find the five tasks with the lowest rewards, and calculate the average reward of them, which is symbolized as *Avg.Bottom*. Afterward, we calculate the ratio of the two average rewards:

$$Ratio.Avg = \frac{Avg.Bottom}{Avg.Top} \tag{8}$$

This metric is utilized to measure the ability to learn task-invariant knowledge. Higher *Ratio.Avg* means better ability to learn task-invariant knowledge. Three baseline methods are utilized: PPO [4], HCP-I [7], and a variant of our method GNN-embedding w/o GNN, which replaces GNN with MLP to implement the policy network and the value network.

TABLE I

PERFORMANCE ON LEARNING THE TASK-INVARIANT KNOWLEDGE

Methods	Avg.Bottom	Avg.Top	Ratio.Avg
Ours	**792.833**	2718.229	**0.291**
Ours w/o GNN	715.383	**3253.940**	0.219
PPO	642.779	2754.383	0.233
HCP-I	557.331	2937.040	0.189

We take the models of the above algorithms trained after $5,000$ iterations, which contain both fully learned tasks and undertrained tasks. The results are illustrated in Table I. The results show that GNN-embedding outperforms the other three methods in terms of *Ratio.Avg*. This demonstrates that our method can achieve better control results on the undertrained tasks. This implies that GNN-embedding can effectively learn task-invariant knowledge and implement generalization among different tasks.

Experimental results also show that the GNN-embedding w/o GNN achieves highest reward on the well learned robots, which reveals that the embedding vectors can improve generalization performance on the multi-task learning problem. However, GNN-embedding performs slightly worse than its variant in terms of the reward on the well learned robots. One possible reason is that the task-invariant knowledge affects the performance of GNN-embedding on specific tasks. GNN-embedding may prefer to make more general control decisions. Another possible reason is that the network model of GNN is too complicated, thus relatively difficult to train than MLP. Therefore, the architecture of GNN need be further optimized, which provides us with a direction for future work.

C. Transfer Learning

As illustrated in Section III-E, GNN-embedding can transfer the well-trained unified policy to some unseen tasks directly by utilizing the corresponding embedding vectors of the robot and environment. This problem is commonly called zero-shot transfer reinforcement learning in previous works. In this experiment, 20 tasks introduced in Section IV-B are chosen as source tasks, while other 20 target tasks are constructed as target tasks. The robots and environments in target tasks appears in source tasks.

As the experiment results shown in Table II, The model achieves acceptable result on the unseen tasks. Most robots

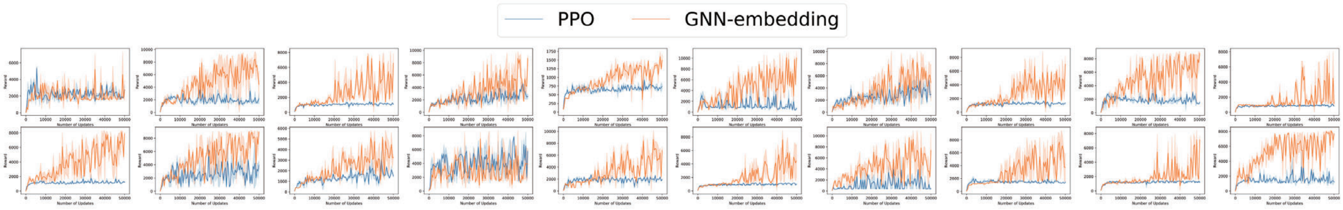

Fig. 5. The learning curves on 20 Hopper-v2 control tasks in the multi-task learning experiment. In most of the tasks, our method (orange line) outperforms baseline method (blue line) on the learning speed and the average reward.

TABLE II

THE AVERAGE REWARD ON THE UNSEEN TASKS WITH THE MODEL AT THE $50,000^{th}$ ITERATION

Methods	Indexes	Unseen Tasks	Training Tasks
GNN-embedding	Average Reward	4523.951	6272.165
	Average Steps	992	1361
PPO	Average Reward	1823.885	1825.920
	Average Steps	423	419

can successfully jump forward for a distance in the unseen tasks, while several robots even jump as far as the robots in the training set without falling. The results demonstrate that GNN-embedding can perform zero-shot transfer learning, which also reveals generalization ability of our method. More details of the result are given in the video attached.

V. CONCLUSIONS

In this work, we propose new method GNN-embedding to improve generalization ability of DRL algorithms. GNN-embedding incorporates the robot hardware and the environment into generalization with GNN-based policy network and learnable embedding vectors. Thus, it can learn a unified policy for different robots under different environment conditions, which improves the generalization performance of existing DRL robot policies. The experiments on the multi-task learning and transfer learning demonstrate the effectiveness and efficiency of our method. In future, we might improve the interpretability of the embedding vectors and design a more efficient GNN architecture for more robot tasks not only in the simulation environment but also in the real world.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[3] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[5] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[6] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey." *Journal of Machine Learning Research*, vol. 10, no. 7, 2009.

[7] T. Chen, A. Murali, and A. Gupta, "Hardware conditioned policies for multi-robot transfer learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[8] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks," in *International conference on learning representations*, 2018.

[9] W. Huang, I. Mordatch, and D. Pathak, "One policy to control them all: Shared modular policies for agent-agnostic control," in *International Conference on Machine Learning*. PMLR, 2020, pp. 4455–4464.

[10] T. Erez, Y. Tassa, and E. Todorov, "Infinite-horizon model predictive control for periodic tasks with contacts," *Robotics: Science and systems VII*, vol. 73, 2012.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[12] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2169–2176.

[13] L. Pinto and A. Gupta, "Learning to push by grasping: Using multiple tasks for effective learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2161–2168.

[14] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "Epopt: Learning robust neural network policies using model ensembles," *CoRR*, vol. abs/1610.01283, 2016. [Online]. Available: http://arxiv.org/abs/1610.01283

[15] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE international joint conference on neural networks*, vol. 2, no. 2005, 2005, pp. 729–734.

[16] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[17] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *stat*, vol. 1050, p. 20, 2017.

[18] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, "Graph convolutional encoders for syntax-aware neural machine translation," *arXiv preprint arXiv:1704.04675*, 2017.

[19] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.

[20] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[21] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4470–4479.

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

[23] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.