Learning Individual Features to Decompose State Space for Robotic Skill Learning

Fengyi Zhang[1,2], Fangzhou Xiong[1,2], Zhiyong Liu[1,2,3],

1. State Key Lab of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Science, Beijing, 100190, China

2. School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS), Beijing, 100049, China

3. CAS Centre for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Shanghai, 200031, China
E-mail: zhiyong.liu@ia.ac.cn

Abstract: Due to suffering from the diversity and complexity of robotic tasks in continuous domains, robotic skill learning is the most challenging issue in this area, especially for robots with high-dimensional state spaces. To learn structured policies for continuous control, the graph neural networks (GNN) was previously applied to incorporate explicitly the robot structure into the policy network. In this work, we tackle the problem of robotic skill learning in high-dimensional state space with the help of graph neural networks. Instead of utilizing a general purpose multi-layer perceptron (MLP) as a unified controller to output actions for all joints of the robot, we construct a separate controller for each joint of the robot by using the individual features that have been extracted by GNN model. Empirical results on simulated continuous systems, including applications to PR2 task and Centipede task, demonstrate that the proposed framework can achieve satisfactory learning performance, and more importantly, it significantly reduces the parameters of the policy network.

Key Words: Robotic Skill Learning, Graph Neural Networks, State Decomposition

1 INTRODUCTION

During recent decades reinforcement learning (RL) techniques have achieved notable successes in the field of robotic control [1, 2, 3]. RL enables autonomous robots to learn large repertoires of behavioral skills with minimal human intervention making it one of the most common methods in robotic applications [4].

Most of the research in robotic control with RL typically uses multi-layer perceptron (MLP) as the agent's policy network. To be specific, MLP takes takes the state vectors of joints from the environment as input. Then the MLP policy network outputs actions to be taken by joints. With the powerful representation performance, RL using MLP as a policy network can perform complex tasks on robots such as simulated and real locomotion [5, 6] and robotic manipulation [7].

Despite the remarkable performance, the existing RL models are prone to overfitting with onerous data, especially for high dimensional data in the robot state space. Yet robots are usually formed by multiple dependent joints. As an example, the PR2 arm is composed of numerous physically linked joints. Action to be taken by each joint may thus not only depend on its states but also actions of other joints. Absolutely, RL models using MLP as a policy network have to discover the latent relationships between different joints, which typically results in longer training time, requiring considerably large number of training samples. Therefore, how to learn a representation that captures the latent relationships behind the structures of the robot is of great importance for efficiently decomposing the high dimensional state space of the robot.

By incorporating a prior on the structure, Wang et al. [8] proposed learning structured policies via graph neural networks (GNN) [9] to learn the latent independent features of each joint. Specifically, as the policy network of the agent, NerveNet[8] first propagates information through the structure of the robot and then outputs actions for different joints of the robot. By doing so, the structured policy has the ability to utilize the structure information of the robot's body, which contributes to learning the correct inductive bias, and thus less tends to overfitting.

However, an important point to note here is that work in Wang et al. [8] aimed to output actions by using a general purpose multi-layer perceptron (MLP) as a unified controller. Nevertheless, it is intuitively obvious that controllers of different joints for the robot also differ, often dramatically. The reason the model in [8] can achieve satisfactory performance we consider is because this method has the ability to learn a highly redundant feature representation. While the preceding method is attractive, complexities arise when agents have high dimensional features. GNN as a policy network which requires more parameters than traditional MLP policies tends to overfit with abundant data, especially for high dimensional data in the robot state space, which is a significant challenge for robotic control.

In this paper, considering that the GNN model has the ability to propagate information between different parts of the

This work is supported by National Key Research and Development Plan of China grant 2017YFB1300202, NSFC grants U1613213, 61375005, 61503383, 61210009, the Strategic Priority Research Program of Chinese Academy of Science under Grant XDB32050100, and Dongguan core technology research frontier pro ject (2019622101001).

978-1-7281-5855-6/20/$31.00 ©2020 IEEE

body based on the underlying graph structure before outputting the action for each actuator, this work devises a framework that can reduce the dimension of the feature representation. More specifically, each actuator of the agent can construct a separate controller by using the features that have been already extracted by the GNN model. By doing so, the input of the separate controller can be a low-dimensional feature, and the policy network is naturally less prone to overfitting. Moreover, the output model in the framework even can utilize the linear model to output action, rather than using more sophisticated multi-layer perceptrons (MLP).

The main contributions of this paper are summarized as follows. First, We investigate the problem of learning a structured policy for the robot via graph neural networks. Second, a novel framework where different linear models are used to output the action distribution of different joints, is proposed to address the complicated tasks that have high-dimensional state spaces.

The rest of this paper is organized as follows. Preliminaries and a brief review of related work are presented in Section 2. Section 3 shows the novel framework to learn the latent independent features of each joint. In Section 4, the main experimental result is presented and discussed. Finally, some concluding remarks are given in Section 5.

2 RELATED WORK

In spired by the huge success of deep neural networks in the supervised learning domain, recently there has been a large number of methods trying to apply neural networks on RL domain [5, 9, 10]. Due to the burden of expensive data collection for learning large repertoires of complex robotic skills, it has remained a significant challenge to decompose the high-dimensional feature space of the robot effectively. Therefore, this work proposes a novel framework to learn the latent independent features of each joint to decompose the high-dimensional feature space via graph neural network (GNN).

This paper improves on policy network by utilizing graph neural networks [11]. A graph data structure consists of a finite set of vertices (objects) and edges (relationships). It is worth noting that graphs have complex structure with rich potential information [12]. Researches of graph with machine learning methods have been receiving more and more attention, given that graph structure data is ubiquitous in the real world. GNN was introduced in [11] as a generalization of recursive neural networks that can process graph structure data. Due to its good generalization performance and high interpretability, GNN has become a widely used graph analysis method in recent years. GNN [13, 14] has been explored in a diverse range of problem domains, including supervised, semi-supervised, unsupervised, and reinforcement learning settings. GNN has been used to learn the dynamics of physical systems [15, 16, 17] and multi-agent systems [18, 19, 20]. These GNN models have also been used in both model-free [8] and model-based [21, 22] continuous control. GNN models also have potential applications in model-free reinforcement learning [23, 24], and for more classical approaches to planning [25].

In this paper, the work is based on the idea of representing a robot as a graph. Here, we define the graph structure of the robot as $G = (u, V, E)$. u is the global attribute of the graph. $V = \{v_i\}_{i=1:N_v}$ is the set of nodes(of cardinality N_v), where each v_i is the attribute of a node. $E = \{e_j, s_j, r_j\}_{j=1:N_e}$ is the set of edges (of cardinality N_e), where each e_j is the attribute of an edge, s_j is the index of the sender node and r_j is the index of the receiver node. In our tasks, the nodes correspond to the joints and the edges correspond to the bodies.

Battaglia et al. [26] presented the Graph Networks(GN) framework that unified and extended various graph neural networks. The GN framework defined a set of functions for relational reasoning on graphs and supported constructing complex structures from simple blocks. The main unit of the GN framework is the GN block which takes a graph as input and returns a graph as output. A GN block contains three "update" functions, ϕ, and three "aggregation" functions, ρ.

$$\begin{aligned} e_k^{'} &= \phi^e(e_k, v_{s_k}, v_{r_k}, u) & \overline{e}_i^{'} &= \rho^{e\to v}(E_i^{'}) \\ v_i^{'} &= \phi^v(\overline{e}_i^{'}, v_i, u) & \overline{e}^{'} &= \rho^{e\to u}(E^{'}) \\ u^{'} &= \phi^u(\overline{e}^{'}, \overline{v}^{'}, u) & \overline{v}^{'} &= \rho^{v\to u}(V^{'}) \end{aligned} \tag{1}$$

where $E_i^{'} = \{e_k^{'}, s_k, r_k\}_{r_k=i,k=1:N_e}$, $V^{'} = \{v_i^{'}\}_{i=1:N_v}$, and $E^{'} = \cup_i E_i^{'} = \{e_k^{'}, s_k,$
$r_k\}_{k=1:N_e}$

As a graph, G, is the input value of a Graph Network, the computations propagate from the edge, to the node and the global level. Algorithm 1 shows the steps of computation for details.

Algorithm 1 Steps of computation in Graph Networks

Input: Graph, $G = (u, V, E)$
  for each edge $\{e_j, s_j, r_j\}$ **do**
    Compute updated edge attributes $e_k^{'} \leftarrow \phi^e(e_k, v_{s_k}, v_{r_k}, u)$
  end for
  for each node$\{n_i\}$ **do**
    Let $E_i^{'} = \{e_k^{'}, s_k, r_k\}_{r_k=i,k=1:N_e}$
    Aggregate edge attributes for each node $\overline{e}_i^{'} \leftarrow \rho^{e\to v}(E_i^{'})$
    Compute updated node attributes $v_i^{'} \leftarrow \phi^v(\overline{e}_i^{'}, v_i, u)$
  end for
  Let $V^{'} = \{v_i^{'}\}_{i=1:N_v}$, $E^{'} = \{e_k^{'}, s_k, r_k\}_{k=1:N_e}$
  Aggregate edge and node attributes globally $\overline{e}^{'} \leftarrow \rho^{e\to u}(E^{'}), \overline{v}^{'} \leftarrow \rho^{v\to u}(V^{'})$
  Compute updated global attribute $u^{'} \leftarrow \phi^u(\overline{e}^{'}, \overline{v}^{'}, u)$
Output: Graph, $G^{'} = (u^{'}, V^{'}, E^{'})$

Recent work by Wang et al. [8] modelled the structure of the reinforcement learning agents using NerveNet model. They aim at learning structured policies for continuous control to achieve transfer learning from one structure to another. Despite the powerful transfer performance, the NerveNet [8] model has more parameters than traditional MLP policy, and also needs longer training time. This sophisticated algorithm have not been applied to domains as difficult as robotic control, not to mention complex tasks

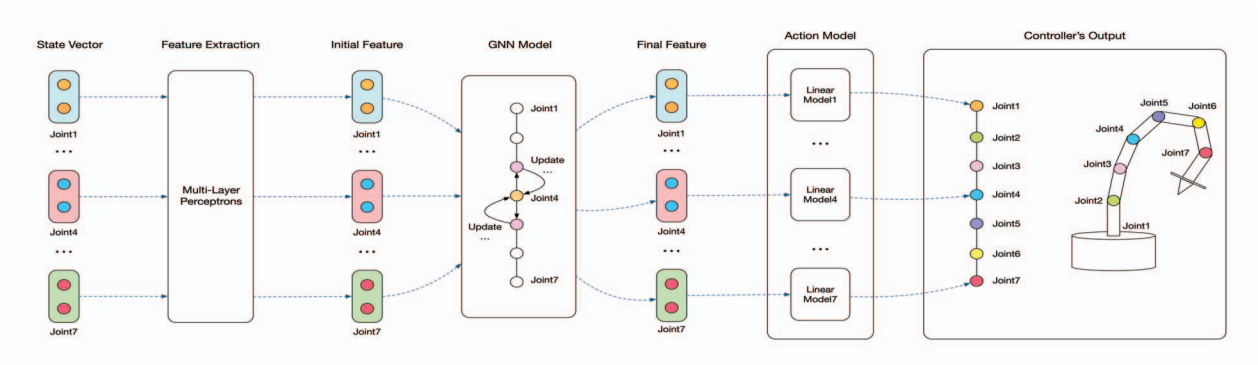

Figure 1: In this figure, we use PR2 as an example of the framework. In the feature extraction model, a MLP fetches the state vector of each joint. Then GNN model updates the initial feature of each joint to get the final feature of each joint. In the action model, the policy is produced by collecting the output from each linear model controller.

with high-dimensional state space. For robotic tasks with high-dimensional state space, the features of each joint are not sufficiently independent and will inevitably result in redundant representations. So it is of very importance to encourage independence among the learned features. To address the the problem of robotic skill learning in high-dimensional state space, this work propose a novel framework to learn features that discover the latent relationship between different joints with desired independence property.

3 PROPOSED METHOD

In this section, to tackle a sequence of complicated tasks that have high-dimensional state spaces, we devise a interchangeable framework to decompose high-dimensional feature space of robots efficiently.

Robotic skill learning in RL typically use MLP as the agent's policy network. Specifically, MLP takes the observartions vector from the environment as input, and then outputs actions to be taken by every joints of the robot. Accordingly, the task of the MLP policy has to discover the latent relationships between states, which naturally results in longer training times, not to mention is prone to overfitting with high dimensional data in the robot state space. The proposed framework integrate these desiderata directly into the framework: high dimensional data is solved by GNN model, while longer training time is prevented by the Action Model.

3.1 Feature Extraction

The joint u of the robot gets an observation vector s_u at each time step in the environment. Passing the observation vector through a neural network, a fixed-size feature vector can be obtained as follows:

$$f_u = F(s_u) \tag{2}$$

where the subscript denote the joint index. Here, F is a MLP having 3 layers with 64 the number of units at the hidden layer. The caveat is that the observation vectors need to be padded with zeros if different joints have observation vectors of different dimensions.

3.2 GNN Model

In particular, for each joint u, we have access to a fixed-size feature vector f_u. Using the GNN model, joints u computes a final feature vector as below,

$$g_u = GNN(f_u) \tag{3}$$

where GNN is the propagation model which mimics the NerveNet framework [12] applied across all nodes information. The GNN model propagates nodes information between different parts of the robot based on the underlying graph structure before outputting the action for each joint. With the ability to utilize the structure information encoded by the robot's body, the GNN model is advantageous in learning the correct inductive bias for the final feature of each joint.

3.3 Action Model

Previous work in the field of RL, agents typically utilize a MLP as the policy, where the single network outputs the mean of the Gaussian distribution for all joints. Note that we do not attempt to output the action distribution of all joints by a sophisticated MLP, but instead use different linear models to predict the action distribution for each individual joints. For example, for each joints, a Linear Model takes its final feature vectors g_u as input and produces the mean of the action distribution for the corresponding joints, as shown in Fig.4. Therefore, we have the following action model:

$$\mu_u = A_u(g_u) \tag{4}$$

where μ_u is the mean value for action applied on each joint. Here, A_u is the Linear Model for joint u.

4 EXPERIMENT

In the following experiments, we first investigate the reasons why GNN with a single network can achieve satisfactory performance. Then, the proposed framework is compared with NerveNet to show its good potential in feature reduction. It is important to point out that feature reduction represents a very difficult, and more often the case, unsolved problem in RL field. Most feature reduction algorithms have not been applied to RL domains, not to mention complex robotic skill learning.

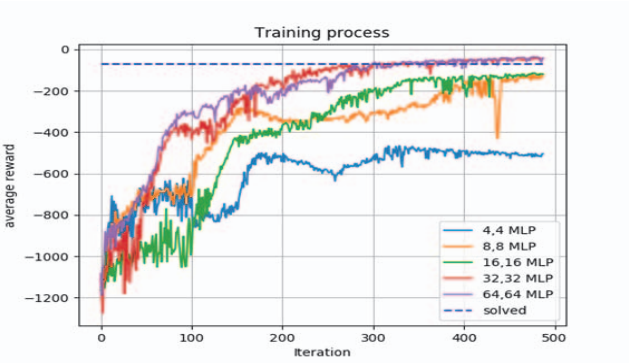

Figure 2: The performance of MLP for the feature reduction experiment

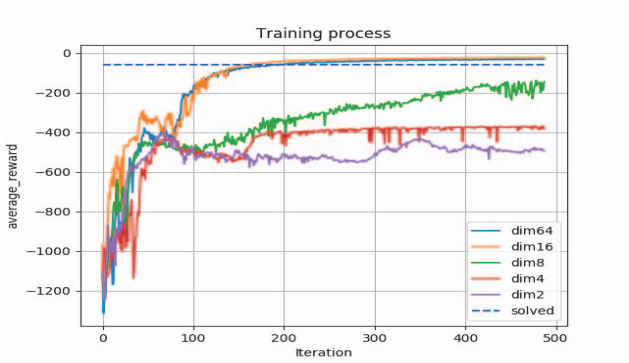

Figure 3: The performance of NerveNet for the feature reduction experiment

4.1 PR2 Task

We run experiments on a simulated continuous control task from Gym, Brockman et al. [27], which is based on MuJoCo, Todorov et al [28]. Particularly, we use a robotic arm task: PR2 arm. The maximum number of training steps is set to be 1 million. In this paper, the proximal policy optimization(PPO) [10] is used to optimize the expected reward.

Experimental settings To fully evaluate the performance of the proposed framework, the experiments are run on the simulated environments from the Gym. Similar in vein to the structure of the PR2 arm, the first environment is created. The goal of the agent is to get the end of the PR2 arm to the precise point. For each time step, the total reward is the negative value of the distance from the end of the arm to the precise point.

Results To explore the reasons why GNN with a single network can achieve good performance, we first run experiments of the NerveNet model on PR2 environment. A important point to note here is that we gradually reduce the dimension of the feature vector, but train the NerveNet in 1 million time steps for all different dimension of the feature vector, which is shown in Figure 3. On the other hand, the proposed framework and the MLP is appled on the PR2 environment while the dimension of the feature vector is reduced as before. The results of the experiments can be seen in Figure 2 and Figure 4.

As can be seen from the figure above, NerveNet is actually not able to achieve satisfactory performance when the dimension of the feature vector drops sufficiently low. One reason is that the highly redundant feature vector enable a single network to output the action distribution of all joints well. However, low-dimensional features do not have enough information to make a single network to output the action distribution of all joints. Therefore, we argue that different networks should be used to predict the action distribution for each joint. The results, as shown in Figure 4, indicate that, despite the dimension of the feature vector is very low, the proposed framework is still able to achieve satisfactory performance.

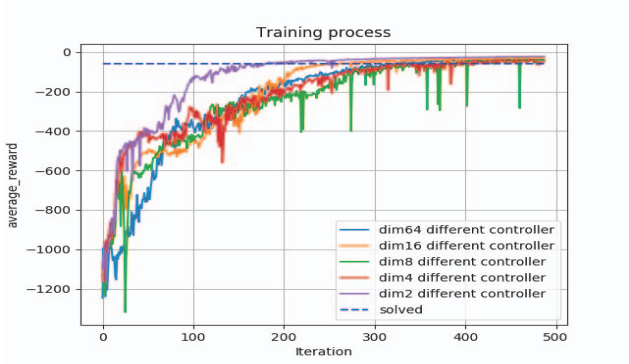

Figure 4: The performance of the proposed framework for the feature reduction experiment

We further explore that we do not utilize MLP to output the action distribution when the dimension of the feature vector drops sufficiently low, but instead attempt to use a simple linear model. It can be seen from the training curves in Figure 5 that the linear model in the proposed framework achieves comparable performance to the MLP based methods. Nevertheless, as can be seen from the Table 1, the parameters number of the proposed framework is 99.67% less than traditional MLP policy, and also the training time is reduced by 30.41%. The results, as shown in Figure 5 and Table 1, indicate that the proposed framework has the ability to achieve comparable results to NerveNet method while greatly reducing the complexity of the algorithm.

4.2 Centipede Task

In order to illustrate the performance of our propsed framework, we also use a robotic task: Centipede. The maximum number of training steps is set to be 1 million for the Centipede task.

Experimental settings We use the second environment in which the agent has a similaer structure to a centipede, which is common in robotics [8]. The goal of the agent is to run as fast as possible along a certain direction in the MuJoCo envrionment. The agent is composed of three repetitive torso bodies where each one has two legs attached. Each leg of the agent has two joints, accordingly the agent

Table 1: Result of the proposed framework on PR2 environment

Model	Number of parameters	Training time (mins)
NerveNet	33591296	43.88
MLP	100352	35.87
The proposed framework	334	24.96

Figure 5: The performance of the proposed framework in linear model setting on PR2

Figure 7: The performance of the proposed framework for the feature reduction experiment

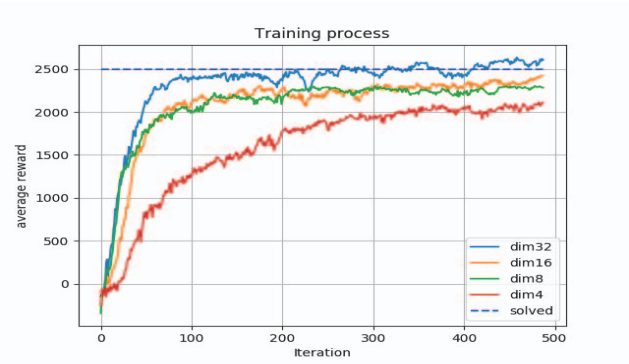

Figure 6: The performance of NerveNet for the feature reduction experiment

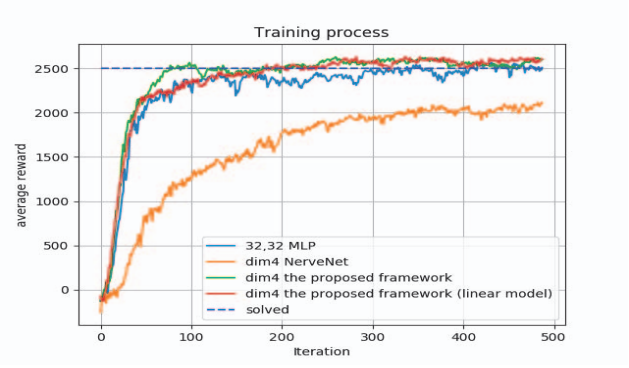

Figure 8: The performance of the proposed framework in linear model setting on Centipede

has fifteen joints totally. For Centipede task, the total reward is the speed reward minus the energy cost and force feedback from the ground.

Results The feature-reduction performance for NerveNet and the proposed framework on Centipede environment is summarized in Figure 6 and Figure 7. A more important benefit is that as the dimension of the feature vector decreases, the proposed framework is still able to achieve satisfactory performance. While for NerveNet, with the lower dimensional feature vectors, the agent can not complete the task.

We also summarize the training curves of linear model in Figure 8. Note that when the feature dimension is low enough, we try to use linear models to output the action distribution for different joints. Although the dimension of the feature is low enough, the linear models in the proposed framework can still achieve good performance. From Table 2, one can observe that the parameters number of the proposed framework is 99.86% less than MLP. At the same time, the training time of the proposed framework is reduced by 26.13% on the Centipede environmet. Hence, the proposed framework greatly reduces the complexity of the algorithm and also accomplishes the task well. One possible reason is that GNN model has the ability to discover the latent relationships between different joints. Therefore, the proposed framework can leverage different linear model to output the action distribution of different joints.

5 CONCLUSION

GNN model is naturally suitable for discovering the latent relationships between different joints by incorporating a prior on the structure of the robot, making it an ideal springboard for addressing state decomposition problems. In this paper, we take a small step in this direction and propose a generic framework for learning individual features for each joint and reducing the complexity of the algorithm. In the framework, a separate controller is constructed for each joint of the agent by using the individual features that have been already extracted by GNN model. Experimental results demonstrate that the proposed framework has the

Table 2: Result of the proposed framework on Centipede environment

Model	Number of parameters	Training time (mins)
NerveNet	33636352	72.61
MLP	1152000	39.19
The proposed framework	1596	28.95

ability to significantly reduces model parameters without hurting the learning performance.

REFERENCES

[1] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies, Journal of Machine Learning Research, Vol.17, No.1, 13341373, 2015.

[2] L. Metz, J. Ibarz, N. Jaitly, J. Davidson, Discrete sequential prediction of continuous actions for deep rl, arXiv preprint arXiv:1705.05035.

[3] M. Wilson, M. W. Spong, Robot modeling and control, Industrial Robot An International Journal, Vol.17, No.5, 709737, 2006.

[4] S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, in: IEEE International Conference on Robotics and Automation, 2017.

[5] J. Schulman, S. Levine, P. Moritz, M. Jordan, P. Abbeel, Trust region policy optimization, in: International conference on machine learning, 2015.

[6] C. Devin, A. Gupta, T. Darrell, P. Abbeel, S. Levine, Learning modular neural network policies for multi-task and multi-robot transfer, in: IEEE International Conference on Robotics and Automation, 2017.

[7] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, S. Levine, (2018, Composable deep reinforcement learning for robotic manipulation, in: IEEE International Conference on Robotics and Automation, 2018.

[8] T. Wang, R. Liao, J. Ba, S. Fidler, Nervenet: Learning structured policy with graph neural networks.

[9] T. P. Lillicrap, J. J. Hunt, et al., Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347.

[11] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Transactions on Neural Networks Vol.20 No.1, 6180, 2008.

[12] A. L. Barabsi, et al, Network science, Cambridge university press, 2016.

[13] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, arXiv preprint arXiv:1511.05493.

[14] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, Computational capabilities of graph neural networks, IEEE Transactions on Neural Networks Vol.20 No.1, 81102, 2008.

[15] M. B. Chang, T. Ullman, A. Torralba, J. B. Tenenbaum, A compositional object-based approach to learning physical dynamics, arXiv preprint arXiv:1612.00341.

[16] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, P. Battaglia, Graph networks as learnable physics engines for inference and control, arXiv preprint arXiv:1806.01242.

[17] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al., Interaction networks for learning about objects, relations and physics, in: Advances in neural information processing systems, 2016.

[18] Y. Hoshen, Vain: Attentional multi-agent predictive modeling, in: Advances in Neural Information Processing Systems, 2017.

[19] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, R. Zemel, Neural relational inference for interacting systems, arXiv preprint arXiv:1802.04687.

[20] S. Sukhbaatar, A. Szlam, R. Fergus, Learning multiagent communication with backpropagation, in: Advances in Neural Information Processing Systems, 2016.

[21] J. B. Hamrick, A. J. Ballard, R. Pascanu, O. Vinyals, N. Heess, P. W. Battaglia, Metacontrol for adaptive imagination-based optimization, arXiv preprint arXiv:1705.02670.

[22] R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racani'ere, D. Reichert, T. Weber, D. Wierstra, P. Battaglia, Learning model-based planning from scratch, arXiv preprint arXiv:1707.06170.

[23] J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. McKee, J. B. Tenenbaum, P. W. Battaglia, Relational inductive bias for physical construction in humans and machines, arXiv preprint arXiv:1806.01203.

[24] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al., Relational deep reinforcement learning, arXiv preprint arXiv:1806.01830.

[25] S. Toyer, F. Trevizan, S. Thiebaux, L. Xie, Action schema networks: Generalised policies with deep learning, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[26] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, arXiv preprint arXiv:1806.01261.

[27] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540.

[28] E. Todorov, T. Erez, Y. Tassa, Mujoco: A physics engine for model-based control, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.