Graph representations in Reinforcement Learning

Hannes Stärk
Technical University Munich
Munich, Germany
hannes.staerk@tum.de

Tobias Schmidt
Technical University Munich
Munich, Germany
tob.schmidt@tum.de

I. Introduction

Learning algorithms often benefit tremendously from capturing prior knowledge about the underlying tasks but encoding such inductive biases can be challenging. In this work, we explore one way to do so for reinforcement learning of robot locomotion. Namely, representing the agent as a graph which allows for explicitly capturing its structure as introduced by *Nervenet* [1]. Our first hypothesis is that this reduces the number of required parameters and could lead to faster convergence. The main advantage, however, is that learning the agent's policy via a graph neural network (GNN) yields desirable properties for transfer learning tasks such as disability transfer (disabling parts of the robot) or size transfer. The modular nature of the GNN allows for straightforward transfer to different robot structures, while a multi-layer perceptron (MLP) based policy requires workarounds like adding padding or masking out values. We show that the results of *Nervenet* [1] are hard to reproduce and make issues of the GNN-based policy approximator transparent.

II. Method

We evaluate our methods on the PyBullet [2] Ant environment and train them using Proximal Policy Optimization [3]. In our baseline, the policy is approximated by an MLP with two hidden layers of size 64 and tanh activation functions that produce the means of Gaussians with a single learned parameter for the standard deviation. The value function is represented by a similar network without parameter sharing between value and policy function.

For our GNN, we need to obtain the graph structure of a robot for any given environment. We parse an agent's structure from URDF robot specification files to extract a condensed graph representation that contains only the nodes which receive observations and generate actions (see Figure 4). The policy networks we use can be split into three parts, as visualized in Figure 5. Firstly, an input network that maps the variably sized observations belonging to a node (like 2 observations for an ankle and 3 for a hip) to fixed-size embeddings. Secondly, the propagation model for which we either use (1) a standard message passing neural network (MPNN) with an MLP for the edge updates and the embedding updates, (2) an MPNN with a gated recurrent unit (GRU) [4] for the state updates, or (3) a graph attention network [5]. The third part of the policy function is the output model, which generates the final means for the actions via an MLP. Here we employ the following approaches. We implemented three variants of this architecture that differ in how they compute the controller outputs from the embeddings produced by the GNN. *NerveNet-v0* concatenates the final embeddings of each node and uses a feed-forward layer to project to the number of required actions. Note that through the concatenation, after which the representation size depends on the robot size. On the one hand, we lose the advantageous transfer properties of the GNN approach compared to the MLP baseline, but on the other hand, the output model isn't affected as much by the information flow within the GNN because it has access to all node embeddings at once. *NerveNet-v1* instead uses mean pooling for each type of node (hip nodes, ankle nodes, root node). This allows to easily transfer to disability environments while minimizing the effect the information flow can have on the performance. *NerveNet-v2* uses an additional output network that receives only the final embedding of an individual node to produce this node's action. This approach provides full transferability to both the disability and size transfer environments but suffers from a suboptimal information flow. In line with the best results by *NerveNet* [1], the value function is approximated by the same architecture as in the baseline.

For all versions of our *NerveNet*, we report the results of the best configuration found via manual random search over the parameter space detailed in Table I.

Results and Discussion

Our quantitative results in Figure 2 reveal that our GNN approaches cannot match the performance of the MLP baseline in either speed of convergence or total reward. This is contrary to what is observed by *NerveNet* [1], who report similar performance to the MLP. The only difference we are aware of between the original *NerveNet* experiments and our *Nervenet-v2* implementation with GRU state updates is the different simulation environment. *NerveNet* uses the MuJoCo [6] Ant environment, which provides additional observations for the non-root nodes that are not available in the PyBullet simulation, such as inertia, 3D velocity, or joint friction. Because of this, the MuJoCo environment has more than twice as many observations as the PyBullet environment.

NerveNet-v2 and *NerveNet-v0* commonly run into a plateau at 1000 reward. Qualitative evaluation of video renderings of the agent's episodes reveals that for *NerveNet-v2*, this plateau often corresponds to the robot standing in place or slowly nudging one foot forward. For *NerveNet-v0*, even though it achieves a lower reward, the videos show that the agent consistently has learned to walk, albeit in a random direction or in a curve leading to the low reward. Of our GNN approaches, only the agent trained with *NerveNet-v1* is able to solve the task and walks in the correct direction, although with a lower speed than the MLP agent.

Unlike the original *NerveNet*, we additionally evaluate zero-shot transfer learning in the Ant environment. When introducing disabilities (Figure 3) for the MLP baseline by zeroing out observations and ignoring actions, the agent walks in a circle without a preference for the correct direction and reaches a reward of 810. When adding two additional legs, the robot is not able to move at all with 820 reward. Meanwhile, *NerveNet-v1* achieves 651 reward for the disability transfer and 508 for size transfer. However, qualitative evaluation of the video renderings for the disability transfer shows that the motions for the disability transfer task are much more similar to an optimal MLP agent directly trained on the disability environment than the motions of the transferred MLP agent.

Future work should explore how the performance changes when employing message passing methods that additionally capture the 3D spatial structure of the graph, such as spherical message passing [7].

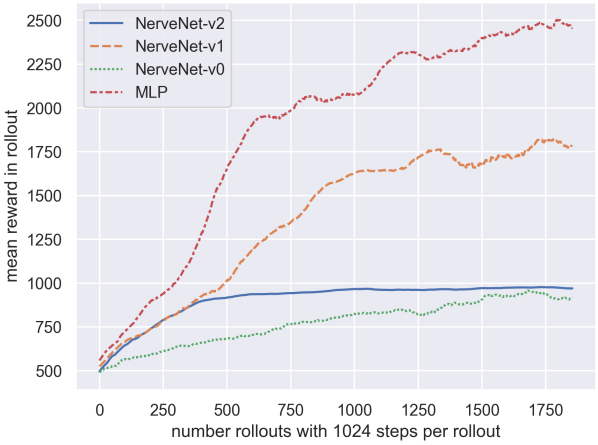

Fig. 1. Mean reward of the MLP baseline compared to NerveNet-v0, NerveNet-v1, and NerveNet-v2 which differ in their output network.

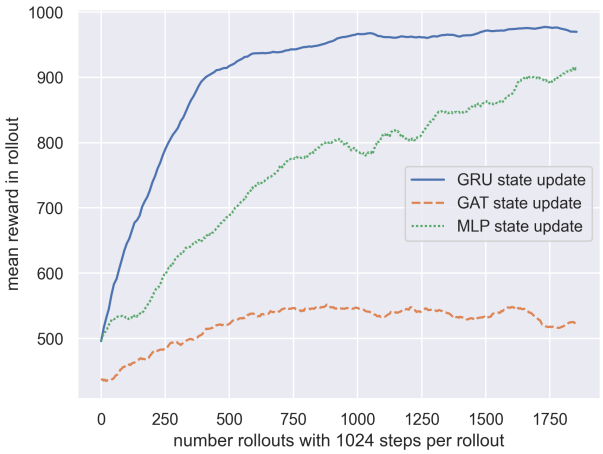

Fig. 2. Mean reward of NerveNet-v2 with either a gated recurrent unit (GRU) or an MLP for the state update or graph attention layers (GAT) with the best hyperparameter settings found.

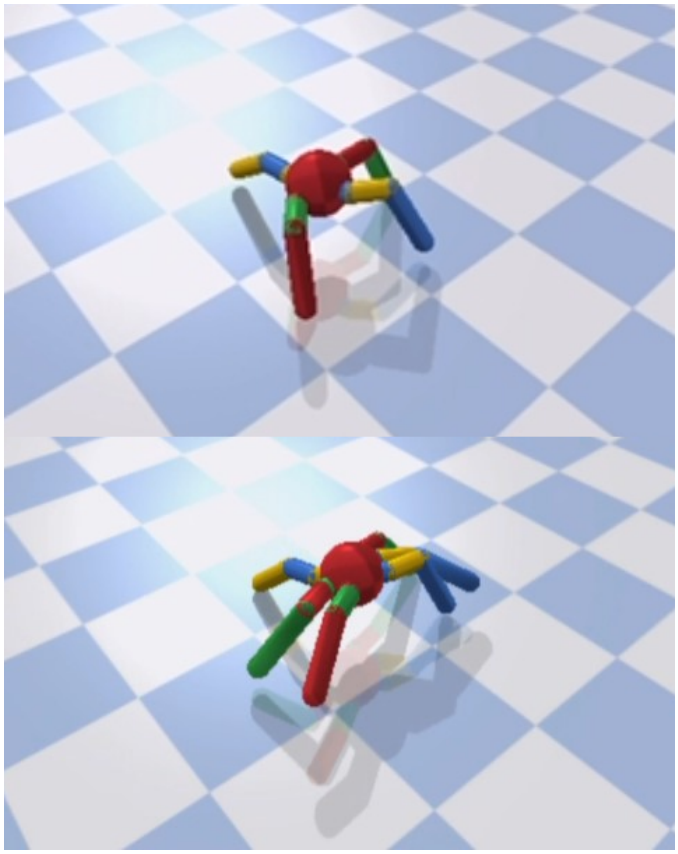

Fig. 3. Ant after disability transfer (top) where the observations of one hind leg are zeroed out and the activations for the legs are ignored. Ant after size transfer (bottom) where two additional legs are added to the middle of the ant.

REFERENCES

[1] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[2] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

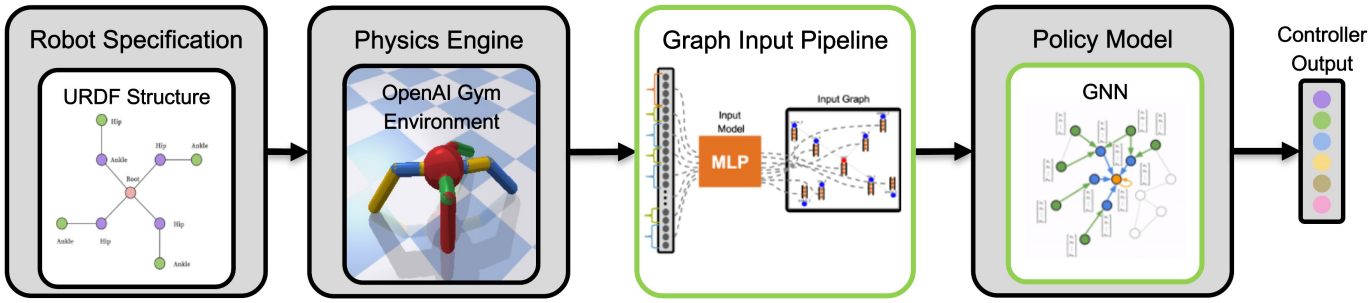

Fig. 4. The general pipeline we use to represent a robot as a graph. A URDF file specifies the structure of an agent, which is simulated in the OpenAI Gym environment. Additionally, the URDF file is parsed to obtain a graph representation and a mapping between observations and the graph nodes as it is used in the graph input pipeline. Lastly, our GNN policy model produces actions for which the correct assignment to controllers is also retrieved from the URDF robot specification.

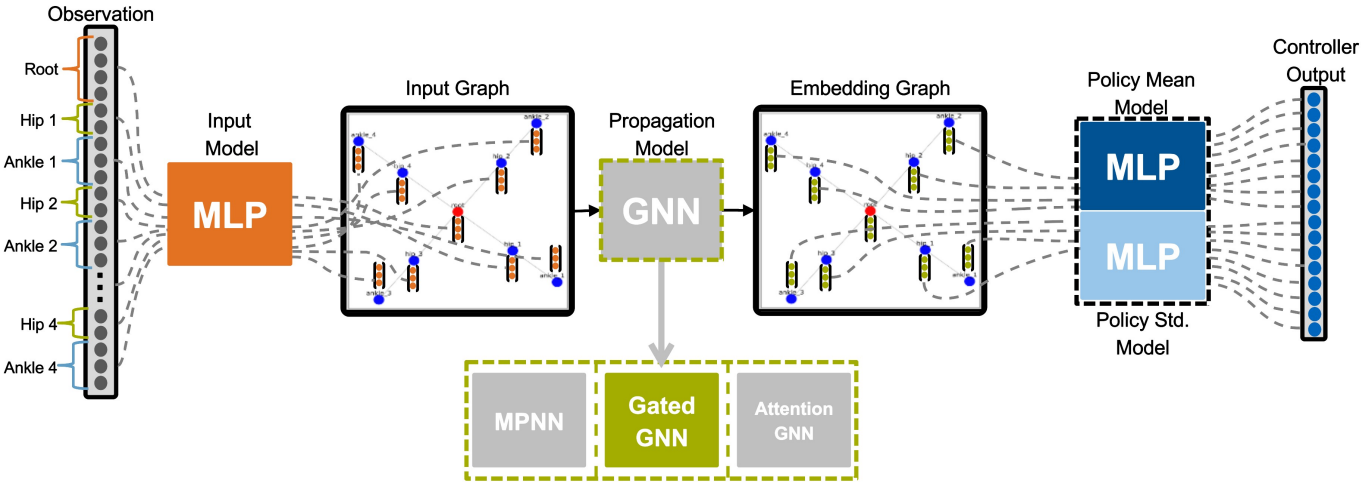

Fig. 5. The *NerveNet-v2* architecture. We first map the differently sized observations to a fixed-size vector which is assigned to the corresponding node in the graph. For the information propagation steps in the graph, we use three different options to obtain output embeddings for each node. Then two MLPs produce the mean and standard deviation for each controller from the individual node embedding of the joint that is related to the controller. The difference to *NerveNet-v0* and *NerveNet-v1* is only the readout approach, which generates the final controller output.

TABLE I
MAIN PARAMETERS TRIED IN MANUAL RANDOM SEARCH FOR OUR METHODS.

Parameter	Values tried
Propagation depth	[3, 4, 5, 6]
GNN Network sizes	[8, 16, 32, 64]
Input Net sizes	[8, 12, 16, 32]
Output Net sizes	[8, 12, 16, 32]
Learning Rate	between 3e-5 to 1e-3
GNN activations	ReLU, tanh
Steps per Rollout	[128, 512, 1024, 2048]

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[4] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014.

[5] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[6] 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*, pages 5026–5033, 2012.

[7] Yi Liu, Limei Wang, Meng Liu, Xuan Zhang, Bora Oztekin, and Shuiwang Ji. Spherical message passing for 3d graph networks, 2021.