Online Adaptation using Graph Networks in Model-based RL

Laurin Paech [*1] **Mayank Mittal** [*2] **Ossama Ahmed** [*2]

Abstract

Despite the recent success of deep reinforcement learning (RL), current RL algorithms are typically unable to generalize well to unseen environment changes such as crippling or external perturbations. We hypothesize that this is due to the absence of inductive biases in these algorithms that would help transfer knowledge across tasks. In this paper, we investigate the application of graph-based relational networks to overcome this limitation. Specifically, we use a graph neural network (GNN) to learn a forward model of the environment. We use the learned model in-conjunction with a model-based controller, and compare our approach against existing model-free and model-based approaches. Further, we provide insights into whether our approach helps improve generalization to new environments or not.

1. Introduction

Reinforcement learning (RL) algorithms are broadly classified into model-free (MF) and model-based (MB) approaches. MF algorithms (Schulman et al., 2015; Lillicrap et al., 2015; Haarnoja et al., 2018) learn a policy by directly optimize over a cost. On the other hand, MB approaches (Deisenroth & Rasmussen, 2011; Chua et al., 2018) first learn a forward model of the environment and then use this *internal* model for optimal control or to train a policy.

Both of these approaches have their benefits and limitations (Wang et al., 2020). While MF methods facilitate learning complex policies, they are data-intensive and susceptible local minimum convergence. These drawbacks limit MF approaches in their applications to robotic control tasks where data collection is expensive. In contrast, MB methods are able to learn with a significantly fewer number of trials and can generalize better on unforeseen environments (Atkeson & Santamaria, 1997). By using an off-policy dataset, training a dynamics model can be performed efficiently through supervised learning. However, learning an accurate model is a bottleneck for these approaches. A MB approach may result in a suboptimal policy that exploits the deficiencies of the models. This issue is even more prominent in long-horizon prediction tasks or environments with high dimensional continuous state or action spaces (Zhang et al., 2018). Thus, the choice of the model is a critical and delicate task for MBRL.

Gaussian processes (GPs) can learn highly non-linear functions efficiently. However, they are typically limited to low dimensional dynamic systems (Calandra et al., 2014). Unlike GPs, neural networks (NNs) have a constant-time inference and scale well to large datasets with high-dimensional inputs. Recent approaches have shown application of NNs to learn non-smooth dynamics that are often present in robotics (Agrawal et al., 2016; Nagabandi et al., 2017). However, NNs typically suffer from overfitting on small datasets leading to poor predictions far in the future. In order to improve generalization, inductive biases are often incorporated into the learning algorithm by introducing domain knowledge or using pre-tuned learning parameters. These biases express assumptions about the space of solutions and allow the learning algorithm to prioritize one solution over another (Mitchell, 1980).

In the context of deep RL, inductive biases are typically included into the agent's objective or through its observations and actions (Hessel et al., 2019). In this work, we investigate an alternate way to incorporate inductive bias via the network architecture (Battaglia et al., 2018). We encode structural inductive biases of a robotic system into the dynamics model by using graph neural networks (GNNs), a class of NNs that enables learning functions on a graph (Wu et al., 2019). In our GNN-based forward model, we encode the knowledge of the body dynamics into the graph. We represent the robot's links as the graph's nodes and its joints as the edges. We use this model in-conjunction with a model-based controller, and test our approach on various continuous control benchmarks. Further, we provide insights on whether our injected inductive bias helps in improving generalization to unforeseen environments or not in comparison to existing MB-MF approaches.

Our contributions are as follows: (1) a model-based RL

[*]Equal contribution [1]Department of Computer Science, ETH Zürich, Switzerland [2]Department of Mechanical Engineering, ETH Zürich, Switzerland.

Preliminary work for the course Deep Learning, ETH Zurich.

approach that uses a GNN-based forward model which exploits the agent's morphology, and (2) transfer learning applied to the graph network to adapt to new tasks.[1]

2. Related Work

Using GNNs for RL is one of the ways to perform relational RL (Tadepalli et al., 2004). In this section, we focus on relevant literature related to using GNN for model-free and model-based RL as well as on adaptation learning.

In view of model-free RL, GNN-based methods have found applications in for multi-agent systems (Malysheva et al., 2018) and network routing problems (Almasan et al., 2019). A relatively under-explored idea is representing a multi-joint system as a graph. Such a graph generally comprises of the robot's joints and links to express their dependencies on each other. Developing on this idea, Wang et al. (2018) propose an MF approach that trains a GNN which represents an agent's policy. They show that their approach allows structure transfer learning to size and disability variations in the system. Along similar lines, Pathak et al. (2019) propose a dynamical graph network policy for a self-assembling agent. On the other hand, Li et al. (2019) use graph networks and curriculum learning for block stacking in a sparse rewards setup. By introducing object-centric biasedness via GNNs, they show that their approach outperforms existing RL algorithms and exhibits zero-shot generalization.

Closer to our work is the MB approach proposed by Sanchez-Gonzalez et al. (2018). In their work, they use GNN to learn a robust and generalizable forward model of a system. They attest that during testing their model generalizes to unseen robotic systems as long as they can be denoted as a graph. However, they do not consider the scenarios where factors external to the agent varies such as the terrain or crippling of a joint.

Online adaptation has often been studied in the context of meta-learning and transfer learning. By training an agent on a distribution of tasks, meta-RL (Finn et al., 2017; Duan et al., 2016) promises to learn behaviors that can deal with the variations in the environment. On the contrary, transfer learning in RL (Donahue et al., 2014; Agrawal et al., 2016) is performed by pre-training on a source task followed by fine-tuning on a target task. This technique has shown to be very successful in reducing the sample requirements for tasks that are similar to each other. In our work, we use transfer learning to show how the learned model improves with fine tuning on tasks it has never seen before.

[1]Due to limitation of time and implementation difficulties, we deviate from our initial proposal to leverage meta-RL to learn a graph-network based model that can adapt to environmental uncertainties. Instead, we use transfer learning to provide insights for online adaption.

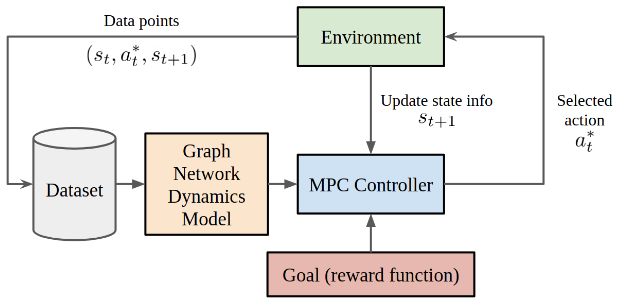


Figure 1. **Illustration of our proposed approach.** On the first iteration, random actions are performed to generate data and train the model. In the following iterations, an iterative procedure is followed for action selection by the MPC, aggregating the data and retraining the model.

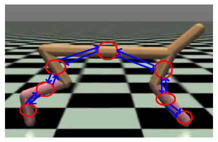
3. Methodology

In this section, we present our model-based RL algorithm. We present our GNN-based dynamics function in Section 3.1 followed by its training details in Section 3.2. In Section 3.3, we explain our model-based control, while in Section 3.4, we show how we can improve further the learned dynamics function iteratively through data aggregation. An overview of the proposed approach is shown in Figure 1.

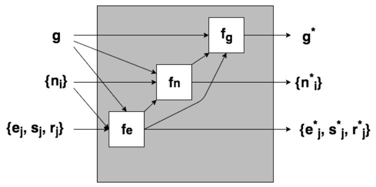
3.1. Graph Network Dynamics Function

We represent our physical system as a directed graph where the robot's links (bodies) and the joints (relations) correspond to the nodes and edges respectively, as shown in Figure 2a. We follow the GNN variation introduced in Sanchez-Gonzalez et al. (2018). The constructed directed graph is defined as $G = (g, (n_i)_{i=1,\dots N_n}, (e_j, s_j, r_j)_{j=1,\dots N_e})$, where g is a vector of global features which describes global physical attributes, $(n_i)_{i=1\dots N_n}$ is a set of nodes where each n_i is a vector of node features that describes relevant attributes to the link i in the system, and $(e_j, s_j, r_j)_{j=1\dots N_e}$ is a set of directed edges where s_j and r_j are the indices of the sender and receiver nodes respectively and e_j is a vector of edge features that describes relevant attributes to the joint j in the system. This graph contains information about the instantaneous state of the system but in a structured way so that the network would learn more about the causal relations rather than learning how to exploit correlations.

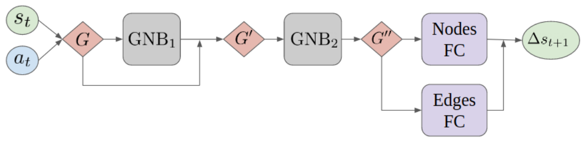
We implement a GNN architecture which can be considered as a generalization to interaction networks (Battaglia et al., 2016). Our architecture is composed of two core GNN blocks. Each of the GNN block (GNB), shown in Figure 2b, is composed of three different NNs– each for calculating new edge features e, new nodes features n and global features g. All three networks inside the GNB are multilayer perceptron (MLP) networks. Inference on this GNN variation can be essentially viewed as a two-step message-



(a) HC graph representation



(b) Graph Network Block (GNB)



(c) Proposed GNN Architecture

Figure 2. **The graph network (GN) of the physical system.** (a) The directed graph has the robot's links and the joints correspond to its nodes and edges. (b) The GN block is based on interaction networks (Battaglia et al., 2016). (c) The graph neural network used comprises of two GN blocks followed by fully connected (FC) layers over the nodes and edges of the transformed graph.

passing network, as explained in Algorithm 1. At every message passing steps: (i) f_e network is first applied to update all edges, (ii) f_n is applied to update all nodes, and (iii) f_g is applied to update the global features.

We first convert the current observations and actions to a graph G as follows:

- **Global features:** Zero vector, $g = 0_{10x1}$
- **Nodes:** Each node representation $n_i \in \mathbb{R}^{13}$, comprises of the absolute body pose, $\mathbb{R}^3 \times SO(3)$, and its absolute velocity, $\in \mathbb{R}^6$.
- **Edges:** Each edge representation $e_j \in \mathbb{R}^4$, is the magnitude of joint action, the joint angle, the joint velocity and direction of the edge (binary with $+1$: parent-to-child, -1: child-to-parent).

As shown in Figure 2c, the graph G is passed to the first GNN block, GNB$_1$, and concatenated with the input graph G to produce the embedded graph G'. This new graph is then passed to the second GNN block, GNB$_2$, which outputs the graph representation G''. This process can be viewed as a graph skip connection, which has been shown previously to alleviate the vanishing gradient problem (Orhan & Pitkow, 2017). The resulting graph G'' is used to obtain the system's predicted state by employing two fully connected (FC) layers. The node features are propagated through a FC layer to project it to $\mathbb{R}^{13}$ in order to obtain the absolute difference in the state of each link. Similarly the edge features are passed through a FC layer to project it to $\mathbb{R}^4$ to extract difference in the joint states. These are then concatenated to obtain the change in the predicted state vector of the physical system for the next time-step, i.e. $\Delta s_{t+1} = s_{t+1} - s_t$.

Algorithm 1 Inference using GNN dynamics model

```
Input: Graph $G = (g, (n_i)_{i=1,\dots N_n}, (e_j, s_j, r_j)_{j=1,\dots N_e})$
for each edge $(e_j, s_j, r_j)$ do
    Gather sender and receiver nodes $n_{sj}, n_{rj}$
    Compute output edges, $e_j^* = f_e(g, n_{sj}, n_{rj}, e_j)$
end
for each node $(n_i)$ do
    Aggregate $e_j{}^*$ per receiver, $\hat{e}_i = \sum_{j/r_j=i} e_j{}^*$
    Compute node-wise features, $n_i{}^* = f_n(g, n_i, \hat{e}_i)$
end
Aggregate all edges and nodes $\hat{e} = \sum_j e_j{}^*, \hat{n} = \sum_i n_i^*$
Compute global features, $g^* = f_g(g, \hat{n}, \hat{e})$

Output:
Graph $G^* = (g, (n_i^*)_{i=1,\dots N_n}, (e_j^*, s_j, r_j)_{j=1,\dots N_e})$
```

3.2. Training the Dynamics Function

We train the forward model using supervised learning by minimizing an L2 loss defined as follows:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}} \frac{1}{2} ||(s_{t+1} - s_t) - \hat{f}_\theta(s_t, a_t)||^2,$$

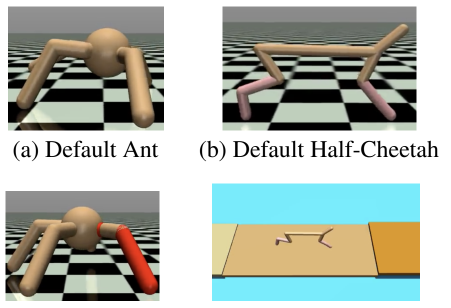
where $\hat{f}_\theta(\cdot, \cdot)$ is our graph-based dynamics model with parameters θ and $\mathcal{D}$ is a dataset containing transition tuples (s_t, a_t, s_{t+1}). We use stochastic gradient descent (Kiefer & Wolfowitz, 1952) to minimize this loss. Further, we add zero mean Gaussian noise to the training data to increase the model robustness.

3.3. Model-based Control

We formulate our model-based controller using the learned model $\hat{f}_\theta(s_t, a_t)$ together with a reward function $r(s_t, a_t)$ that encodes the task we want to perform. This formulation is similar to that of a model predictive controller (MPC). Consider a finite horizon H for the controller over which we want to optimize the action sequence $A_t^H = (a_t^{,}..., a_{t+H-1})$ by maximizing the following cost function:

$$\mathcal{J}(A_t^H) = \sum_{t'=t}^{t+H} r(\hat{s}_{t'}, a_{t'}),$$
$$\text{s.t.} \quad \hat{s}_{t'+1} = \hat{f}_\theta(\hat{s}_t, a_t), \hat{s}_t = s_t.$$

Since this cost function is non-linear, obtaining an exact solution is difficult and computationally expensive. Instead we use try to obtain an approximate solution using random shooting (Rao, 2010). In this method, we sample K candidate action sequences that are generated randomly and pick the sequence that produces the maximum cumulative reward. We then apply only the first action from this sequence, a_t^*, as typically done in receding horizon control.



(a) Default Ant (b) Default Half-Cheetah

(c) Crippled Ant (d) Half-Cheetah on Blocks

Figure 3. **Environments implemented using MuJoCo physics engine.** We modify (a, b) default OpenAI gym environments for ant and half-cheetah (HC) to create environments where (c) one of their joints can be crippled and (d) perturbations are present.

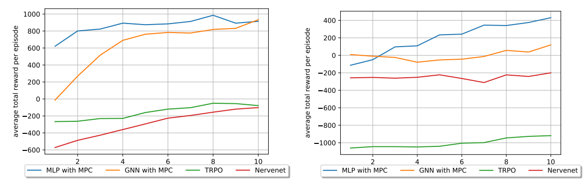
3.4. Improving the Learned Dynamics

We initially train our model by creating a dataset generated from random actions by using supervised learning. In order to further improve the performance of our model, we collect on-policy data using our model-based controller. This is achieved by running the controller to gather data, aggregating the collected transitions into the dataset $\mathcal{D}_{\text{agg}}$, and then training the model on this dataset. As shown by Nagabandi et al. (2017), this alternation between data collection and model refinement helps in dealing with mismatch between data's state-action distribution and the state-action distribution induced by the model-based controller.

4. Results

Through our experiments, we hope to find answer to the following questions: (1) How does our GNN compare to MLP for learning the dynamics models? (2) Is our GNN-based MBRL approach more efficient than other existing approaches? (3) Can our injected inductive bias help in zero-shot generalizability? (4) How much does transfer learning help in online adaptation of GNN-based models?

We implement our approach from scratch using Tensorflow 2.0 APIs (Abadi et al., 2016) and benchmark using MuJoCo (Todorov et al., 2012) physics engine.[2] For benchmarking, we use the half-cheetah (HC) ($\mathcal{S} \in \mathbb{R}^{103}, \mathcal{A} \in \mathbb{R}^{6}$) and ant environments ($\mathcal{S} \in \mathbb{R}^{133}, \mathcal{A} \in \mathbb{R}^{8}$). We modify these environments for crippling the agent and adding perturbations, as shown in Figure 3.



(a) Default Half-Cheetah (b) Default Ant

Figure 4. **Comparison against other MB-MF RL algorithms.** We compare our algorithm *(in orange)* against: (i) a MB approach *(in blue)* with an MLP dynamics model (Nagabandi et al., 2017), (ii) TRPO *(in green)* with MLP policy (Schulman et al., 2015), and (iii) Nervenet *(in red)* trained using TRPO (Wang et al., 2018).

4.1. Comparison to other MB-MF approaches on benchmark tasks

We benchmark our GNN-based approach against three baselines[3]– one MB approach and two MF approaches. To compare the performance of our GNN model against other parametrizations of the forward model, we implement the multi-layer perceptron (MLP) model proposed by Nagabandi et al. (2017). For our MF-baselines, we use trust-region policy optimization (TRPO) (Schulman et al., 2015) to learn an MLP policy and a GNN policy which follows the architecture from Wang et al. (2018). In order to have a fair comparison, we train each algorithm for $100k$ samples and use the same MLP network size of $(512, 512)$.

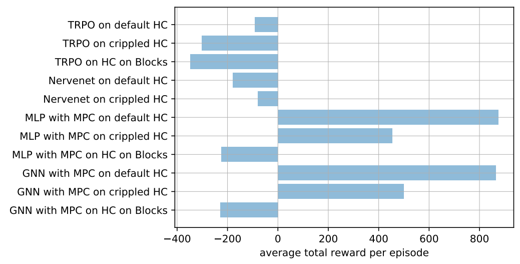
As shown in Figure 4, both MB approaches surpasses the two MF approaches proving that MF methods require several orders of magnitude more samples to learn a decent policy. Additionally, comparing MLP with MPC to our GNN with MPC, we observe that MLPs capture the dynamics earlier on with fewer iterations. We believe that this could be due to a difference in the total number of parameters in the two models. Our GNN has two GNN blocks each of which three MLPs in it (as explained in Section 3.1). This results in approximately 6 times more than the number of parameters in the MLP model.

4.2. Evaluation for zero-shot transfer

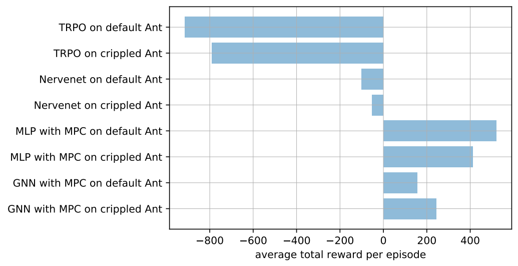
In order to verify the usefulness of structural inductive bias in generalization to new tasks, we use the pre-trained models, obtained from Section 4.1, on zero-shot transferability. As shown in Figure 5, MF approaches failed to generalize when presented a new task. On the other hand, MB approaches perform better even without any fine-tuning.

[2]Although we tried implementing our environments using pyBullet (Ellenberger), we found that there is a disparity between the reward functions and observations that the default pyBullet environments provide and those from MuJoCo environments. This difference affected the performance of our MPC. Thus, we switched to MuJoCo since it is more commonly used for RL experiments.

[3]In our proposal, we also considered the meta-RL baselines, specifically MAML (Finn et al., 2017) and GrBAL (Clavera et al., 2019). Although both these algorithms have their implementations online, we only managed to run MAML successfully. The GrBAL implementation did not work as pointed out on its GitHub issue. We were unable to fix their code. Consequently, we left leveraging meta-RL as a future work.

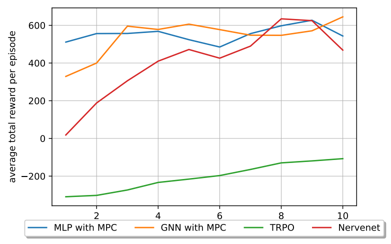


(a) Pre-trained on default Half-Cheetah

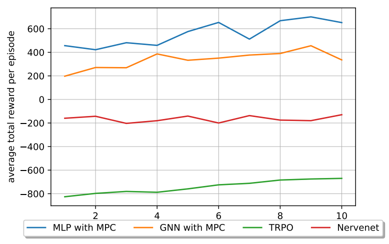


(b) Pre-trained on default Ant

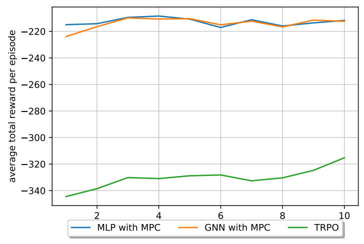
Figure 5. **Evaluation for zero-shot transfer into different environments.** We use the pre-trained models learned from training in the default half-cheetah and ant environments respectively.



(a) Crippled Half-Cheetah



(b) Crippled Ant



(c) Half-Cheetah on Blocks

Figure 6. **Affect of fine-tuning on learning.** We use the pre-trained models from Section 3.2, and transfer them to unseen environments. The plots in (a, b) are those for the crippled agents while (c) is for the HC with blocks environment where perturbations are present. We fine-tune each model after collecting $20k$ samples every time.

Contrary to our expectations, our GNN-based model performs poorly compared to an MLP. We reason this is due to weak inductive biases included in our model leading to sub-optimal performance. In the half-cheetah environments our GNN-based appraoch can at least perform on par with the MLP even with the shortcoming of our design.

4.3. Affects of fine-tuning on learning

To further evaluate where fine-tuning helps in improving our GNN to new tasks, we perform an experiment where each pre-trained model is updated after collecting $20k$ samples from the environment. We observe from Figure 6 that MF approach with MLP policy seem to barely benefit from pre-training and subsequent fine-tuning. In contrast, Nervenet, which employs a GNN policy, shows significant improvement by fine-tuning in the crippled half-cheetah environment. Our GNN-based approach performs comparable to the MLP based model although we anticipated that adding inductive bias would have helped in better adaption.

5. Discussion and Summary

In this work, we investigated the application of graph-based relational networks to overcome generalization issues in current state-of-the-art RL algorithms. We implemented a GNN-based forward model that exploits the agent's morphology, and use it in-conjunction with an MPC. We carry out careful experimentation to analyse the effect of adding structural inductive bias on training, zero-shot transferability to adapt to new tasks, and transfer learning.

Our experiments suggest several interesting directions for further investigations. First, our results confirmed that MB approaches are more sample efficient than MF ones. Moreover, they also need less fine-tuning to adapt to any deviation from the training environment, which can occur as perturbations in the surrounding environment or in the physical system itself. Furthermore, we showed that using a GNN with MPC performs better than existing MF methods. However, to our surprise, it didn't perform better than using an MLP dynamics model with MPC. We suspect that this is due to the over-parametrization of the GNN model, which could have led to overfitting with the comparably small dataset used for training. This can be seen in Figure 6a, where a GNN implemented in an MF approach surpassed the other MF approach as well as reaching the same performance of the MB approaches after fine tuning. Consequently, this suggests that, indeed GNNs can learn more about the causal relational model as compared to a standard MLP. We believe further experimentation with different graph networks design needs to be done in order to confirm that GNNs can help in online adaptation. Moreover, we consider that incorporating meta-learning can further improve the performance of a GNN-based forward model, as seen for MLP based representation used in Clavera et al. (2019).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016. URL https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

Agrawal, P., Nair, A., Abbeel, P., Malik, J., and Levine, S. Learning to poke by poking: Experiential learning of intuitive physics. 2016.

Almasan, P., Suárez-Varela, J., Badia-Sampera, A., Rusek, K., Barlet-Ros, P., and Cabellos-Aparicio, A. Deep reinforcement learning meets graph neural networks: An optical network routing use case, 2019.

Atkeson, C. G. and Santamaria, J. C. A comparison of direct and model-based reinforcement learning. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pp. 3557–3564, April 1997.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks, 2018.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. Manifold gaussian processes for regression, 2014.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.

Clavera, I., Nagabandi, A., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyztsoC5Y7.

Deisenroth, M. P. and Rasmussen, C. E. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of International Conference on Machine Learning*, 2011.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. In Xing, E. P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 647–655, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/donahue14.html.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. Rl^2: Fast reinforcement learning via slow reinforcement learning, 2016.

Ellenberger, B. Pybullet gymperium. URL https://github.com/benelot/pybullet-gym.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

Hessel, M., van Hasselt, H., Modayil, J., and Silver, D. On inductive biases in deep reinforcement learning, 2019. URL https://openreview.net/forum?id=rJgvf3RcFQ.

Kiefer, J. and Wolfowitz, J. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952. doi: 10.1214/aoms/1177729392. URL https://doi.org/10.1214/aoms/1177729392.

Li, R., Jabri, A., Darrell, T., and Agrawal, P. Towards practical multi-object manipulation using relational reinforcement learning, 2019.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning, 2015.

Malysheva, A., Sung, T. T., Sohn, C.-B., Kudenko, D., and Shpilman, A. Deep multi-agent reinforcement learning with relevance graphs, 2018.

Mitchell, T. M. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ..., 1980.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, 2017.

Orhan, A. E. and Pitkow, X. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

Pathak, D., Lu, C., Darrell, T., Isola, P., and Efros, A. A. Learning to control self-assembling morphologies: a study of generalization via modularity, 2019. URL `https://openreview.net/forum?id=B1lxH20qtX`.

Rao, A. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135, 01 2010.

Sanchez-Gonzalez, A., Heess, N. M. O., Springenberg, J. T., Merel, J., Riedmiller, M. A., Hadsell, R., and Battaglia, P. W. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, 2018.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization, 2015.

Tadepalli, P., Givan, R., and Driessens, K. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 workshop on relational reinforcement learning*, pp. 1–9, 2004.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, Oct 2012. doi: 10.1109/IROS.2012.6386109.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=S1sqHMZCb`.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba, J. Benchmarking model-based reinforcement learning, 2020. URL `https://openreview.net/forum?id=H1lefTEKDS`.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks, 2019.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. A study on overfitting in deep reinforcement learning. *ArXiv*, abs/1804.06893, 2018.