Assessing Generalization in Probabilistic Ensembles for Model-Based Deep Reinforcement Learning

Michael Bayerle, Sandro Panighetti, Benjamin Tearle, Jose Luis Vazquez Espinoza

***Abstract*—Generalization in reinforcement learning refers to an agent's ability to perform outside of the environment it was trained in. Reinforcement learning (RL) algorithms are typically both trained and tested on fixed environments, which can result in over-fitting to the system under analysis. Since model-based RL (MBRL) methods attempt to learn the underlying system dynamics, they may perform better in generalization tasks compared to model-free methods, which directly learn a policy. Probabilistic Ensembles with Trajectory Sampling (PETS) [1] is a MBRL algorithm that is both sample efficient and high-performing on standard benchmark tasks. In this paper, we study how well PETS is able to generalize outside of the parameters it was trained on using a recently proposed generalization standard and modifiable gym environments from [2]. We find that PETS performs lower on these benchmarks compared to current state of the art model-free methods, and there is no apparent benefit of a standard model-based algorithm in terms of performance under model mismatch.**

I. Introduction

In reinforcement learning, an agent learns how to solve a specific task without having any prior information about the task itself or the environment it operates in. The agent explores the environment and receives a response, i.e. a reward, that indicates the quality of an action given the system state. Over time, the agent learns which actions are required to solve the assigned task. A great portion of the literature in reinforcement learning is concerned with developing algorithms that enable an agent to learn faster, more efficiently and improve scores on common benchmark tasks. However training an agent to perform a task in an environment and assessing the agent's performance by testing it on the same task and environment can result in the model overfitting to the fixed environment parameters. This is in contrast to one of the most important goals in machine learning: achieving generalization of a trained model on previously unseen test data.

Due to the inherent complexity of the problem, it remains a challenging topic in the field that is being approached with different techniques. Recently, Packer et al. [2] assessed the generalization capabilities of model-free reinforcement learning algorithms on environments that randomize the parameters of the physical system being simulated. In this paper, we study the generalization capabilities of model-based reinforcement learning algorithms following the same testing methodology as Packer et al [2]. In particular, we focus on using probabilistic ensembles with trajectory sampling (PETS) on classical control tasks, to determine if using a model-based approach is able to generalize better than model-free. Our implementation of PETS shows a lower level of performance than model-free benchmarks, and the scores in all generalization tests show no implicit generalization advantage over model-free approaches.

II. Related Work

Reinforcement learning has achieved various successes in the last decade, with recent notable breakthroughs helping the field to gain increased attention. DeepMind developed increasingly capable versions of it's AlphaGo [3] algorithm that was able to beat one of the world's best players in the complex board game Go, with followup improved versions training in far less time by playing themselves in an adversarial manner [4]. OpenAI created an open source toolkit for RL research known as the OpenAI Gym [5], which includes ATARI games, classic control tasks, and more complex locomotion tasks. This has become one of the de facto standards for benchmarking RL algorithm performance, and has helped open the field up for accessible algorithm testing.

Quite naturally, researchers have started to specifically address the generalization issue in reinforcement learning. Nagabandi et al. [6] take a meta-learning approach by learning online adaption for changes to the agent's environment. This is done by learning a prior for the dynamics model and rapidly updating the model with recent data after every time step to adapt to the local context. Rajeswaran et al. [7] propose EPOpt, an algorithm that uses ensembles of other RL models and adversarial training to create policies that are robust to environment changes. A robust policy is found by sampling trajectories from a collection of possible models among the environment parameter distribution, and then using policy gradient optimization to find a single policy that fits the different model trajectories.

Researchers in the area have applied different methodologies to assess the generalization capabilities of their models, which made standardized comparisons between different models very challenging. Witty et al. [8] provide definitions for dividing a state-space into on-policy, off-policy, and unreachable states in order to test repetition, interpolation, and extrapolation performance. Trials using a state of the art DQN network show poor generalization ability to minor changes to the environment. Packer et al. similarly propose assessing generalization performance by looking at interpolation and extrapolation ability, but also explicitly define metrics for scoring these abilities on a modified set of OpenAI Gym environments. Model-free methods, including actor-critic and proximal policy optimization, are benchmarked, where they find "vanilla" RL algorithms outperformed algorithms specifically designed to excel in generalization, like the previously mentioned EPOpt in [7]. These previous efforts have focused on assessing model-free reinforcement learning algorithms, which are typically more expressive than model-based algorithms, in that they achieve better performance on common benchmark tasks. The procedure proposed by Packer et. al is what we follow in this paper to analyze a model-based approach.

Wang et. al [9] recently completed a comprehensive benchmark of various MBRL algorithms, where the performance of 18 different algorithms was analyzed in a unified setting of standard OpenAI gym environments. One of the highest performing algorithms in this study used probabalistic ensembles with trajectory sampling, or PETS, proposed by Chua et al [1]. Like other MBRL algorithms, PETS is more sample efficient in comparison to its model-free counterparts; however, it claims to also offer similar levels of performance. In this paper, we contribute to the research of generalization in deep reinforcement learning by assessing the generalization capabilities of PETS on classical control tasks. The experiments are conducted within the framework suggested by Packer et al. which allows us to compare the generalization performance of PETS to the model-free algorithms tested in their work.

III. Models and Methods

Model-based RL consists of learning the dynamics of an environment using previously acquired observations. This is in contrast with model-free RL, where the goal is to learn a policy. More precisely, let $\mathcal{S}$ denote the state space and $\mathcal{A}$ the action space. In MBRL one tries to find an approximation $\hat{f}$ of $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. Typically, the environment dynamic is a stochastic process so f is actually a probability distribution over the possible states. In our case, $\hat{f}$ is an ensemble of probabilistic neural networks.

$\hat{f}$ is used in conjunction with a model predictive controller (MPC) to decide the next action. A series of possible action sequences are generated and rolled out using the learned model to predict the potential next states of the system. Whichever action sequence results in the highest reward is deemed the best policy, and the first action is performed. This is then repeated at every time step.

A. Models

The ensemble $\hat{f}$ consists of k feed-forward neural networks $\hat{f}_i$. The output of $\hat{f}$ is the mean of the outputs of the k neural networks. Each of the neural networks takes the current state $s_t \in \mathcal{S}$ as input, together with an action $a_t \in \mathcal{A}$. In case the action space is discrete, the input action is transformed into a one-hot encoding. Our implementation of PETS consisted of an ensemble size of 3 feed-forward neural networks. An ensemble size of 3 was sufficient for the fairly simple classic control tasks that were the subject of our analysis. Each network was made up of 3 hidden layers and 500 units per layer. The SiLU activation function [10] was used as suggested in the original PETS paper.

We tested two kind of neural networks: deterministic and probabilistic. The deterministic neural network typically outputs the expected s_{t+1} and uses the standard L2 loss for training. The probabilistic neural network instead outputs a probability distribution defined by the mean, μ_θ, of the expected state, s_{t+1}, and its covariance matrix, Σ_θ. We assume a Gaussian distribution and a diagonal covariance matrix, resulting in an output layer with two neurons per observed state: one representing the mean, and one representing the variance. This network uses a Gaussian loss function:

$$\text{loss}_{\mathcal{N}}(\theta) = \sum_{n=1}^{N} [\mu_\theta - s_{n+1}]^T \Sigma_\theta^{-1} [\mu_\theta - s_{n+1}] + \log \det \Sigma_\theta \tag{1}$$

This acts like a weighted L2 loss, in that it punishes prediction errors for states that have lower uncertainty more, while also encouraging low variance values.

In our implementation, $\hat{f}_i$ actually computes the difference $\Delta_{t+1} = s_{t+1} - s_t$ instead of s_{t+1} directly. During training, these state differences are used as targets for the network. We can then simply compute $s_{t+1} = s_t + \Delta_{t+1}$ after receiving the difference output from the network. Computing delta training targets was originally suggested by Deisenroth et. al [11]. This technique encodes an implicit prior on the prediction results, such that the system state should remain constant in the case of zero value outputs, instead of suddenly dropping all states to zero.

B. MPC & Trajectory Sampling

Model predictive control is a control method that attempts to find the optimal action for the current time step using a model of the system to predict possible state trajectories. All future states are assessed by some loss or reward value, and the optimal state trajectory is selected as the one achieving the best score either in terms of minimizing the loss or maximizing the reward. Only the first action resulting in the optimal state trajectory is applied, and then the process is repeated at the next time step, this is called receding horizon control. This results in a closed loop control system that can account for disturbances from the proposed trajectory.

The PETS algorithm implements an MPC style controller using a random sampling shooting method, described in Algorithm 1. At each time step, a number of action sequences of fixed horizon length are created by sampling independently and uniformly from all possible actions (both for discrete or continuous spaces). Each action sequence is then used to predict state trajectories using the learned system dynamics model $\hat{f}$ and propagating it forward. The action-state sequence that results in the state trajectory with the highest reward at the end of the horizon is chosen, and the first action is applied to the agent. The reward function r is engineered by the user and depends on the environment. For example, the Pendulum's reward function r gives a high score for keeping the pole in a vertical position. [1]

C. Training

The training is a repeated process which consists of 1) generating new data and 2) training the models.

The replay memory $\mathcal{D} = \{\tau_i\}_i$ consists of trajectories

$$\tau_i = ((s_0, a_0, r_0), (s_1, a_1, r_1), \ldots) \tag{2}$$

[1]A complete list of reward functions used for different gym environment can be found in Appendix A of [9].

Algorithm 1: MPC

```
Input  : Current state $s \in \mathcal{S}$, horizon $h$, samples $n$,
         reward function $r$, discount factor $\gamma$
Output: action $a \in \mathcal{A}$
1  for $i \leftarrow 1$ to $n$ do
2      actions$_i \sim \mathcal{U}(A^h)$
3      reward$_i = 0$
4      state$_{i,0} = s$
5      // Rollout
6      for $t \leftarrow 1$ to $h$ do
7          state$_{i,t} \leftarrow \hat{f}(\text{state}_{i,t-1}, \text{actions}_{i,t-1})$
8          reward$_i \leftarrow \text{reward}_i + \gamma^t \cdot r(\text{state}_{i,t})$
9      end for
10 end for
11 $i^* \leftarrow \arg\max_i \text{reward}_i$
12 return $actions_{i^*,0}$
```

which are generated using the MPC policy in closed loop. As the model improves, the agent is able to explore spaces closer to the goal state.

Each model $\hat{f}_i$ is trained on a bootstrapped dataset $\hat{\mathcal{D}}_i$ from the replay memory $\mathcal{D}$, which ensures that the networks are different but follow the same statistics. To further improve the training, the dataset $\mathcal{D}$ is normalized.

D. Generalization

We assess the generalization of our solution by analyzing the performance of a model tested in an environment with different parameters than it was trained in. Packer [2] provides modified OpenAI gym environments that generate randomized parameters for the physical properties of the simulated environment. For example, in the CartPole environment, three parameters can be varied: the pole length, the pole mass, and the push force magnitude [2]. The environment can be launched with three parameter settings: default (D), random (R), or extreme (E). Default maintains the original fixed parameters, while random and extreme uniformly sample the parameters from different continuous value ranges. The random distribution contains feasible parameter values in the neighborhood of the default values, while the extreme distribution contains drastically different values that represent edge cases. This provides us with different combinations of training and testing environments, which will be referred to as two-letter abbreviations of the training setting followed by the testing setting (e.g. training on default and testing on default would be DD, while training on random and testing on extreme would be RE).

Along with providing the open source modified environments, Packer [2] also proposes standardized testing metrics that can be used to create simple scores describing the different levels of generalization ability. These scores are based on the percentage of successful episodes out of all tested episodes, where each environment has a unique success function:

- CartPole: balance for at least 195 time steps
- Acrobot: swing the end-effector of the links to the fixed height within 80 time steps
- Pendulum: keep the pendulum within $\pi/3$ radians of either side of the vertical for the final 100 time steps of a 200 time step episode

This then allows the creation of three separate scores based on episode success rate in different configuration of training versus testing:

- Default: success percentage on DD
- Interpolation: success percentage on RR
- Extrapolation: geometric mean of the success percentages on DR, DE, and RE

While the Default score acts as the baseline for standard model performance, the Interpolation score shows how well the agent can generalize after being trained and tested in similar randomized environments. Finally, the Extrapolation score shows how well the agent performs when testing it in environment configurations it has never encountered during training.

IV. Results

Generalization scores were generated for three different classic control tasks using our implementation of the PETS algorithm. The modified environments of CartPole, Acrobot, and Pendulum were trained and tested according to the configuration described in the previous section. The final scores can be seen in Table I, and were the average of five runs, each measuring success over 1000 episodes. The agents were trained for 20000 total time steps, in 1000 time step increments of data generation.

While PETS was able to generally "solve" all three tasks, there was wide variation in the generalization scores obtained using the success metrics. The CartPole environment was the easiest to solve, with the agent able to adjust to normal and extreme deviations in environment parameters with the exception of our probabilistic network in the extrapolation setting. Towards the end of the RE episodes for these tests, the network adjusted to extreme variations that negatively affected model accuracy and resulted in a zero score. Since extrapolation uses a geometric mean of DE, DR, and RE, the extrapolation score becomes zero for that run. The Acrobot and Pendulum proved harder to obtain the level of performance set by the proposed success metrics, especially outside of the default test setting. There is no noticeable difference between using probabilistic networks versus deterministic networks for these three environments.

Similar trends between generalization scores can be seen between our model-based method and the model-free algorithms. Default scores remain the highest as there is no deviation in parameters during testing or training, while Extrapolation scores are the lowest, as the parameter ranges being tested have never been seen during the respective training episodes.

[2]A table of each environments modifiable parameters and the different ranges can be found in Table 1 of [2].

	Default				Interpolation				Extrapolation			
	PETS-P	PETS-D	A2C	PPO	PETS-P	PETS-D	A2C	PPO	PETS-P	PETS-D	A2C	PPO
CartPole	100.0	100.0	100.0	100.0	100.0	99.86	100.0	100.0	51.71	94.19	93.63	86.20
Acrobot	59.96	61.50	88.52	87.20	36.58	30.00	72.88	72.78	17.44	16.53	66.56	64.93
Pendulum	96.28	95.86	100.0	0.0	82.20	87.06	99.86	31.80	70.86	71.20	90.27	0.0

TABLE I: Generalization scores of our implemented PETS agent, including both probabilistic (PETS-P) and deterministic (PETS-D) ensemble results, alongside the model-free algorithm benchmarks of A2C and PPO as taken from [2]. Scores presented are the average of five test runs for each of the three gym environments tested.

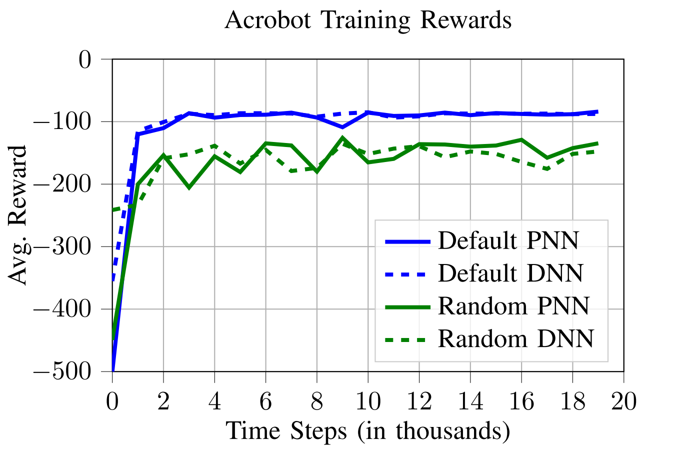

Fig. 1: The average episode reward obtained using the PNN network and DNN network on Acrobot. Rewards are shown in both Default training setting and Random training setting. We plot the average over 3 experiments of the average episode score evaluated every 1000 time steps.

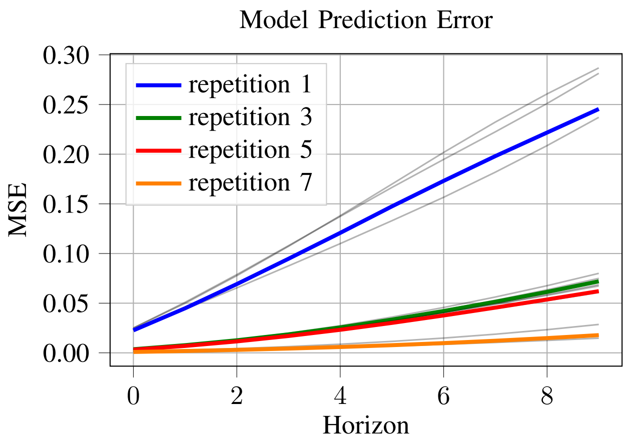

Fig. 2: The model prediction error at horizon h corresponds to mean squared error between the true observations and the state computed by rolling out h steps using the trained model and the action sequences. The grey lines correspond to the MPE of the individual models, while the colored lines correspond to the ensembles. This plot has been computed on the CartPole environment.

Figure 1 shows the progression of average reward values taken from evaluation periods performed throughout training in both default and random parameter settings for Acrobot (Appendix A contains similar CartPole and Pendulum graphs). It can be seen that the PETS algorithm is able to learn each task quickly, confirming the sample efficiency of MBRL methods. Again there is no noticeable difference between PNNs and DNNs for these environments. However the performance then plateaus at a level that does not improve with additional training data. Training with fixed default parameters results in a higher performance plateau than training on randomized parameters.

Figure 2 shows the model prediction error (MPE) which quantifies the error occurring during the rollout. This metric is useful to evaluate the quality of the predicted state with respect to the horizon length and provides information about the quality of the model itself. If the MPE does not improve, it implies that the model output is not precise and this error is accumulated over the horizon length. Hence, using a longer horizon might not provide better results. Intuitively, as the model quality improves, the MPE is reduced. At the same time, the model explores more states closer to the goal state, therefore the MPE is not guaranteed to improve with the repetitions.

V. Discussion

Our results show that there is no improvement in generalization performance for a model-based algorithm like PETS in comparison to the model-free algorithms. Although our initial hypothesis was that learning the underlying system dynamics would help a model-based approach maintain its performance under environment parameter changes, PETS demonstrated no advantage when tested in this generalization assessment.

It's generally known that model-free algorithms are able to score higher than their model-based counterparts given enough training episodes, as demonstrated more extensively in [9]. Although PETS claims to achieve similar performance to model-free algorithms, our findings show that it still plateaus at a level below model-free. PETS was successful in solving all the environments, but since the success metrics were originally derived based on model-free performance (e.g. 80 time step requirement for Acrobot), it performed worse in all three generalization scores. Compared to the Default scores, PETS Interpolation and Extrapolation scores actually fell further than A2C's and PPO's scores did (with the exception being that PPO tested in [2] was not able to solve Pendulum in Default / Extrapolation configurations).

For the simple environments tested in our experiments, there is no noticeable benefit in using a probabilistic network over a deterministic network. Both networks learn all tasks within 2000 time steps, and the Interpolation and Extrapolation scores are similar between the two types of networks. As seen in the original PETS paper, a significant improvement in a probabilistic ensemble over a deterministic ensemble only begins to show when learning more complex environments such as the MuJoCo Half-Cheetah.

Increasing the MPC horizon should allow the PETS algorithm to predict further into the future, and thus better select the optimal current action. However, the predictions become more susceptible to compounding model error, and can result in long horizon trajectories that are less reliable than short horizon trajectories. A larger MPC horizon also also greatly increases the computation time when running the algorithm, and can cause it to no longer operate in real-time when stepping through the environments (real-time meaning that the time required for action selection is less than the default time step of the environment).

Model-based reinforcement learning requires a user specified reward/cost which differs from the OpenAI gym environment reward. This hand-engineered reward function embeds a bias into the MPC policy causing more than sub optimal behavior in certain systems. While this has not been proven empirically, it is a still an open question that remains unanswered because it is incredibly difficult to set a metric with respect to the "true" reward function that allows us to quantify this bias.

VI. Summary

Our work analyzed the generalization of a particular method of model-based reinforcement learning. We confirmed in this setting that although a learning agent requires far fewer iterations to solve the task at hand compared to model-free methods, it still falls short of model-free algorithm performance in each of the generalization assessments. Based on intuition, PETS would have better generalization capabilities thanks to its decoupling of the control policy from the learned system dynamics, however our results showing the under-performance on these benchmarks demonstrate that this is not the case.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," *arXiv e-prints*, p. arXiv:1805.12114, May 2018.

[2] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song, "Assessing generalization in deep reinforcement learning," 2018.

[3] D. Silver, A. Huang, and e. a. Maddison, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, and e. a. Huang, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, Oct. 2017.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[6] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning," *arXiv e-prints*, p. arXiv:1803.11347, Mar 2018.

[7] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "EPOpt: Learning Robust Neural Network Policies Using Model Ensembles," *arXiv e-prints*, p. arXiv:1610.01283, Oct 2016.

[8] S. Witty, J. K. Lee, E. Tosch, A. Atrey, M. Littman, and D. Jensen, "Measuring and Characterizing Generalization in Deep Reinforcement Learning," *arXiv e-prints*, p. arXiv:1812.02868, Dec 2018.

[9] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," *CoRR*, vol. abs/1907.02057, 2019.

[10] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *CoRR*, vol. abs/1702.03118, 2017.

[11] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian Processes for Data-Efficient Learning in Robotics and Control," *arXiv e-prints*, p. arXiv:1502.02860, Feb 2015.

VII. Appendix

A. Training Rewards

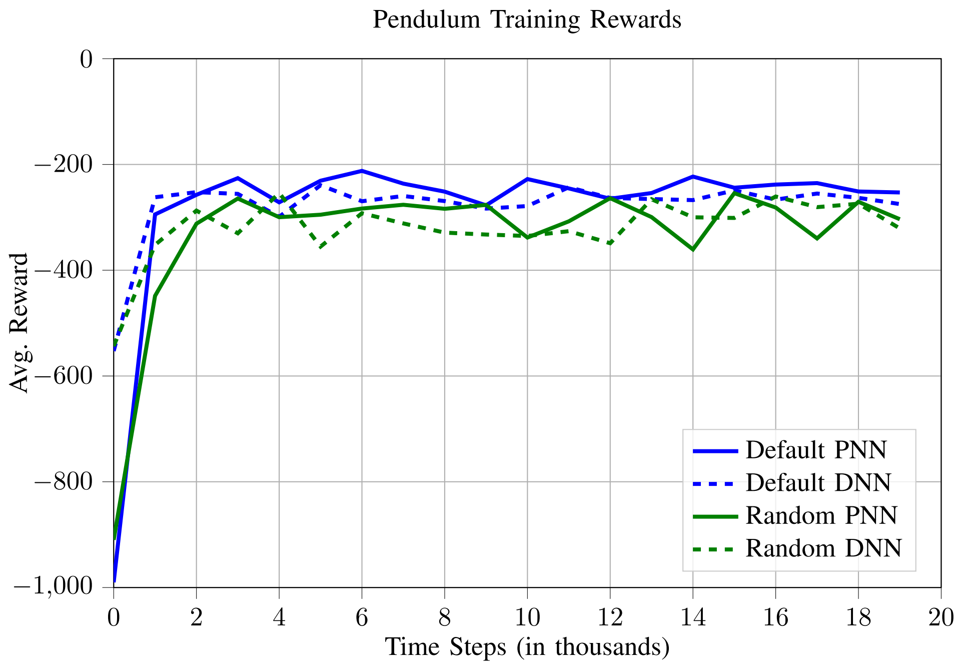

Fig. 3: The average episode reward obtained using the PNN network and DNN network on Pendulum. Rewards are shown in both Default training setting and Random training setting. We plot the average over 3 experiments of the average episode score evaluated every 1000 time steps.

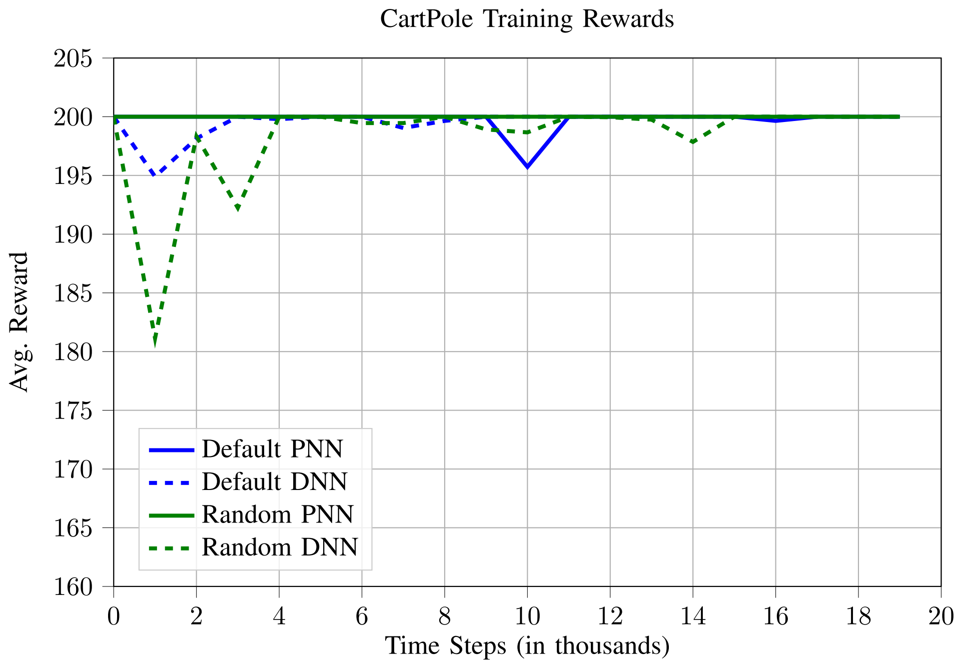

Fig. 4: The average episode reward obtained using the PNN network and DNN network on CartPole. Rewards are shown in both Default training setting and Random training setting. We plot the average over 3 experiments of the average episode score evaluated every 1000 time steps.