EVALUATION OF MODEL-BASED LEARNING TECHNIQUES AND ENSEMBLE METHODS

Zian Fu
Department of Mathematics
Chongqing University
Chongqing, China
zianfu96@gmail.com

ABSTRACT

Model-based reinforcement learning (RL) methods are a promising solution to robotic control tasks. They offer higher sample efficiency than model-free RL methods. In addition, model-based RL allows for dynamic model simulation of behaviors prior to enacting them, thereby improving safety and reducing robot wear and tear. An issue with model-based RL is that the transition model must be expressive enough to ensure low model bias. We sought to address this problem by investigating the impact of the choice of dynamics model. We looked at neural network models, which are universal function approximators but require many samples to train. In comparison, we evaluated Gaussian Process (GP) regression models, which are non-parametric and provides estimates on model uncertainty. Finally, we looked at how ensemble methods could be used to further improve model fit and performance. Our experimental evaluations show that the neural networks performed better than Gaussian Process regression. We used both modeling approaches in a model predictive control framework to evaluate model performance. We explored different hyperparameters for each modeling approach: the number of sampled action sequences for the neural network, and the number of local models/clusters evaluated for the GP model. Testing it on different Mujoco tasks such as Half-Cheetah and Reacher showed that neural networks were more suitable for fitting the complex Mujoco dynamic systems models. However, we are exploring methods to tune the GP models further based on improved methods we found in the literature.

1 Introduction

Model-based reinforcement learning (RL) methods are a promising solution to robotic control tasks. Compared to model-free RL, model-based RL has higher sample efficiency, requiring smaller datasets to train. This is vital for real-world systems, where collecting sufficient demonstrations proves challenging. In addition, from a safety perspective, model-based RL allows for a better understanding of the impact that actions have on state behavior. This means that safety limits can be enforced and the amount of robot wear and tear can be reduced [Polydoros & Nalpantidis(2017)Polydoros and Nalpantidis]. On the other hand, model-free RL algorithms have better asymptotic performance, which has led to work combining both architectures for quicker learning and better asymptotic performance [Kazmi & Driesen(2015)Kazmi and Driesen].

A key issue with model-based RL is ensuring that the selected transition model is expressive enough for low model bias. To address this problem, we sought to investigate how the choice of transition model impacts the performance of the system. We compare the non-parametric Gaussian Process (GP) model with the parametric neural net (NN) model. We also investigate the use of ensembles to further improve model performance. These modeling methods are applied to the Mujoco Half-Cheetah and Reacher tasks to evaluate their effectiveness [Todorov et al.(2012)Todorov, Erez, and Tassa].

This paper is organized as follows. Section 2 provides an overview of the models investigated, as well as the implementation of the model-based RL algorithm used. Section 3 discusses the performances of the different models investigated. Finally, Section 4 outlines the key observations and provides suggestions for further improvement and future work.

2 Methodology

2.1 Models Investigated

2.1.1 Neural Networks

Neural networks are promising candidates for fitting complex dynamic systems models, as they serve as universal function approximators [Hornik(1991)]. However, as parametric models, it is vital that the network architecture be tuned appropriately to avoid high model bias or poor generalization. We decided to test our approach on two main Mujoco models such as "half-cheetah" and "reacher". Our choice was mainly affected by the different comparable results that we got from neural networks and Gaussian processes. There are several works that look at using neural networks as building blocks for controlling dynamical systems. For example, Nguyen et al. show how neural networks can be used to learn system dynamics, which they call an “emulator”. This emulator dynamics model is used in conjunction with a controller, implemented using another neural network, to model the kinematics of a truck in a backing maneuver [Nguyen & Widrow(1990)Nguyen and Widrow]. Similarly, Hagan et al. shows how neural network dynamics models can be used a model block for standard control methods like model reference adaptive control (MRAC), model predictive control (MPC), and feedback linearization. They focus on the application domain of a tank reactor, robot arm, and a magnetic levitation system [Hagan et al.(2002)Hagan, Demuth, and Jesús].

For this paper, we take inspiration from the work done by Nagabandi et al. on using neural network dynamics models to speed up learning for Mujoco tasks [Nagabandi et al.(2017)Nagabandi, Kahn, Fearing, and Levine]. Our neural network implementation and model predictive control algorithm are based primarily on the methods described in this work.

2.1.2 Gaussian Processes

Another widely used modeling approach in the literature is Gaussian Process (GP) regression. In contrast to neural networks, GPs are non-parametric (aside from fitting a kernel function). GP models place a prior over functions, and condition over training observations to determine probability distributions for new test points not in the training set. The main assumption is that given training data, a new test point is jointly Gaussian with the training points, with the covariance governed based on similarity among the points. Points that are closer to the test point have more impact on the fitted result; this is enforced using a kernel function as a similarity measure. In addition, the GP model covariance can be used to determine model uncertainty, allowing confidence intervals over predictions to be made. Full details can be found in [Rasmussen et al.(2003)Rasmussen, Kuss, et al.].

GPs have been successfully used for a variety of different control tasks, although they have mostly been low-dimensional problems. For example, Rasmussen et al. apply two GP models to model each state of Mountain Car, a 2D problem [Rasmussen et al.(2003)Rasmussen, Kuss, et al.]. Similarly, Ko et al. control a blimp using a combination of a 2nd order ODE model and a single GP to improve model fit [Ko et al.(2007)Ko, Klein, Fox, and Haehnel].

The challenge we sought to address is if we could tune the GP model for higher-dimensional problems. We try to address the issue that for very complex networks, there may be highly different covariance behavior in different parts of the input space. This is done by using a gating network: for a set of GPs, each individual GP model is fit to a given cluster in the training set data. At test time, the new point is fit using the closest local model (i.e. by looking at the nearest cluster centroid). The clusters were found using the K-means algorithm. We vary the number of clusters, i.e. the number of local GP models, to see if having more local models would improve performance.

2.2 Ensemble Methods

Ensemble methods have been used to speed up learning and improve performance. For example, techniques like bootstrap aggregation (bagging) have been used to do value function prediction and policy generation. Using multiple learners resulted in quicker convergence to the optimal policy compared to a single learner in maze and Tic-Tac-Toe task domains [Faußer & Schwenker(2011)Faußer and Schwenker].

Similarly, ensembles using voting techniques like majority vote, Boltzmann multiplication, and Boltzmann addition were used to combine the policies from five different reinforcement learning algorithms [Wiering & Van Hasselt(2008)Wiering and Van Hasselt].

We were interested in two main ensemble methods, averaging and Super Learner, which were implemented in [Ju et al.(2017)Ju, Bibaut, and van der Laan]. The former method simply takes the output of each learner and averages them to get an estimate. In contrast, the Super Learner uses the output of each learner as a feature and then uses a single-layer neural network to blend these outputs together. These approaches were used to make an ensemble dynamics model and try to reduce generalization error.

2.3 Implementation and Training Details

The implementation follows the approach in Section 4 of [Nagabandi et al.(2017)Nagabandi, Kahn, Fearing, and Levine]. The following mostly comes from that work, but we describe the specific parameters we change for this project.

We assume that the model dynamics are satisfied by the following discrete-time evolution:

$$s_{t+1} = s_t + f(s_t, a_t) \tag{1}$$

The goal of fitting the dynamics model is to reduce the prediction error, E, on the dataset D containing sets of observations of the form $\{s_t, a_t, s_{t+1}\}$. Note that the model predicts the change in the state, rather than the next state explicitly.

$$E = \sum_{i \in D} \|s^i_{t+1} - s^i_t - \tilde{f}(s^i_t, a^i_t)\| \tag{2}$$

The action selection is done using a MPC algorithm, using stochastic optimization to maximize the expected return. K action candidate sequences over a horizon H, $A_i = (a_{t_0}, \ldots, a_{t_0+H-1}), i = 1, 2, \ldots, K$ are chosen randomly from a uniform distribution over the action space. At each time step, t_0, the best action sequence is chosen as follows:

$$A_{max} =_{i \in \{1,\ldots,K\}} \sum_{t=t_0}^{t_0+H-1} R(\hat{s}_t, a^i_t) : \hat{s}_{t_0} = s_{t_0}, \hat{s}_t = \hat{s}_{t-1} + \tilde{f}(\hat{s}_t, a^i_t) \tag{3}$$

The first action in the sequence, $a^*_{t_0} = A_{max}(t_0)$, is then selected and taken by the agent. For more details on the Model-Based RL algorithm we used, refer to Algorithm 1 in [Nagabandi et al.(2017)Nagabandi, Kahn, Fearing, and Levine].

What we modify in our implementation is the following:

- $\tilde{f}(s, a)$ is either a neural network model, $\tilde{f}_{NN}(s, a)$ or a GP model, $\tilde{f}_{GP}(s, a)$.
- For the neural network model, $\tilde{f}_{NN}(s, a)$, we modify K, the number of candidate action sequences. This is to see how accurate the optimization must be for the neural net to get good performance.
- For the GP model, $\tilde{f}_{GP}(s, a)$ we modify the number of clusters evaluated, i.e. the number of individual GP models in the gating network. In other words, $\tilde{f}_{GP}(s, a)$ is a piecewise function over the (s,a) domain.

3 Results

3.1 Model Comparison

The following sections showcase the results we observed as we changed the hyper-parameters for the neural net and GP models, respectively.

3.1.1 Neural Network Results

For the neural network, we looked at the impact of the number of sampled action sequences used in the MPC algorithm (K in Section 2) on the model fit and performance (Figs. 3, 6).

For all neural networks investigated, we fixed the following parameters:

- MPC Horizon (H) = 15
- Number of NN Layers = 2
- Size of Each NN Layer = 500
- Batch Size = 512
- Model Training Iterations = 100
- Learning Rate = 1e-3 with Adam Optimizer

[t] 0.45

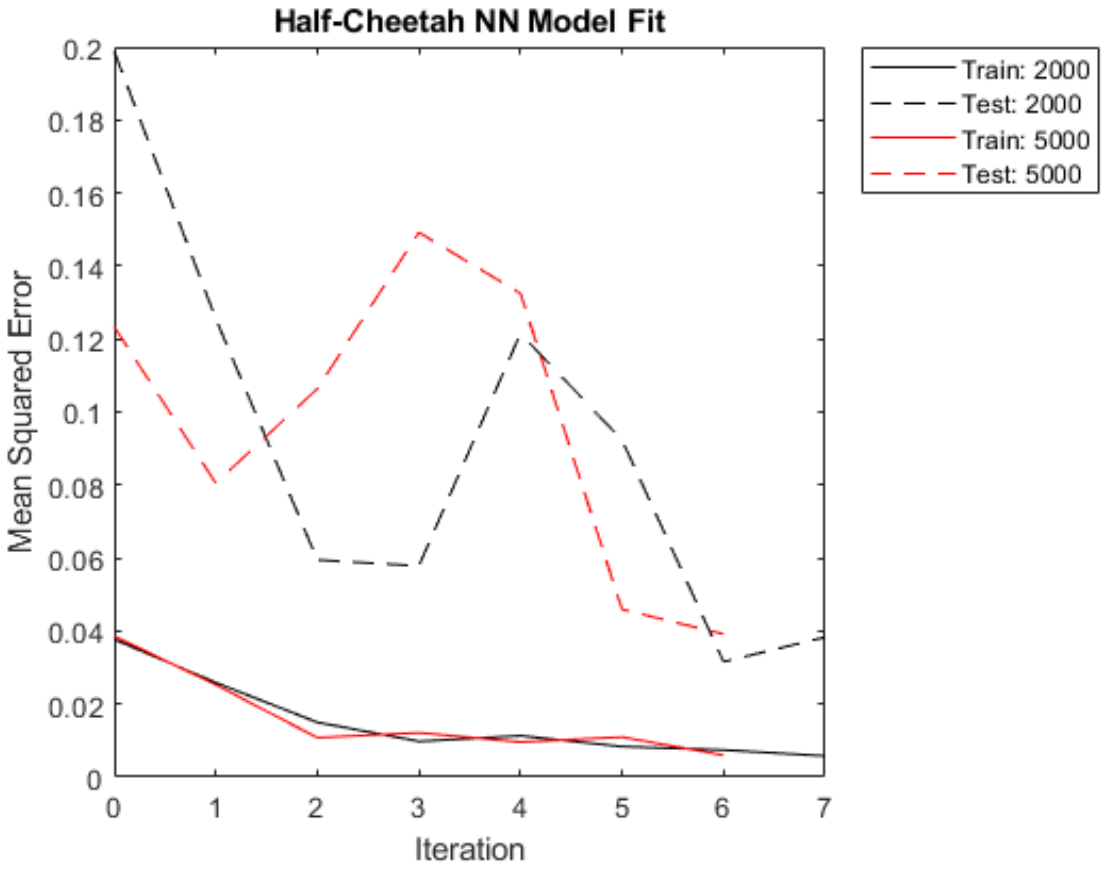
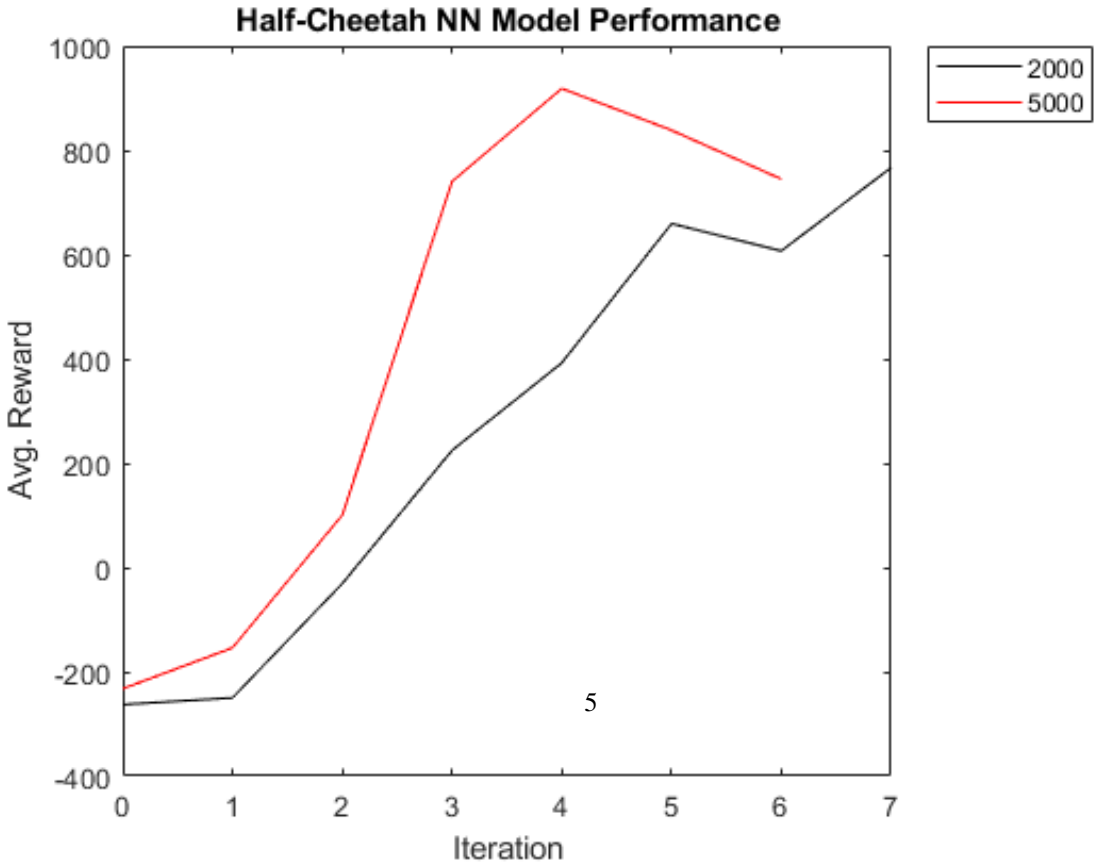


Figure 1: Test/Train Error, Half-Cheetah Neural Net

[t] 0.45



[t] 0.45

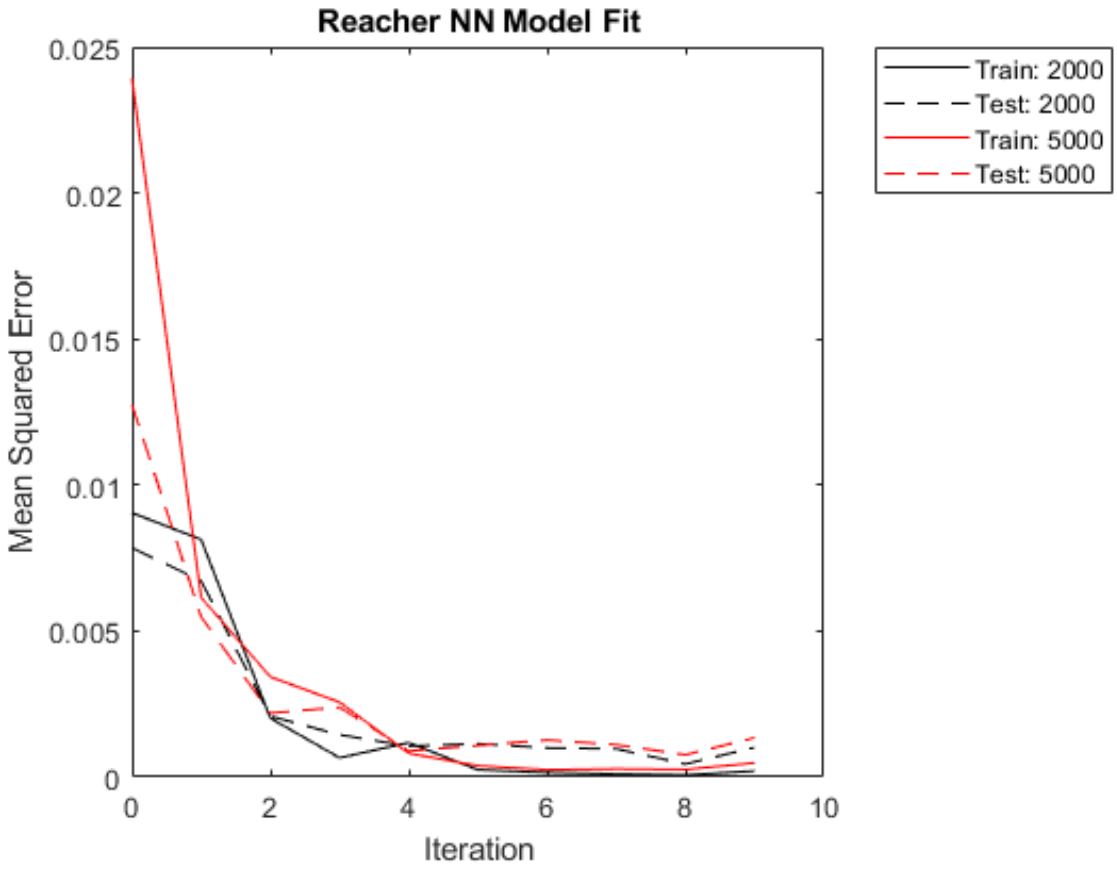
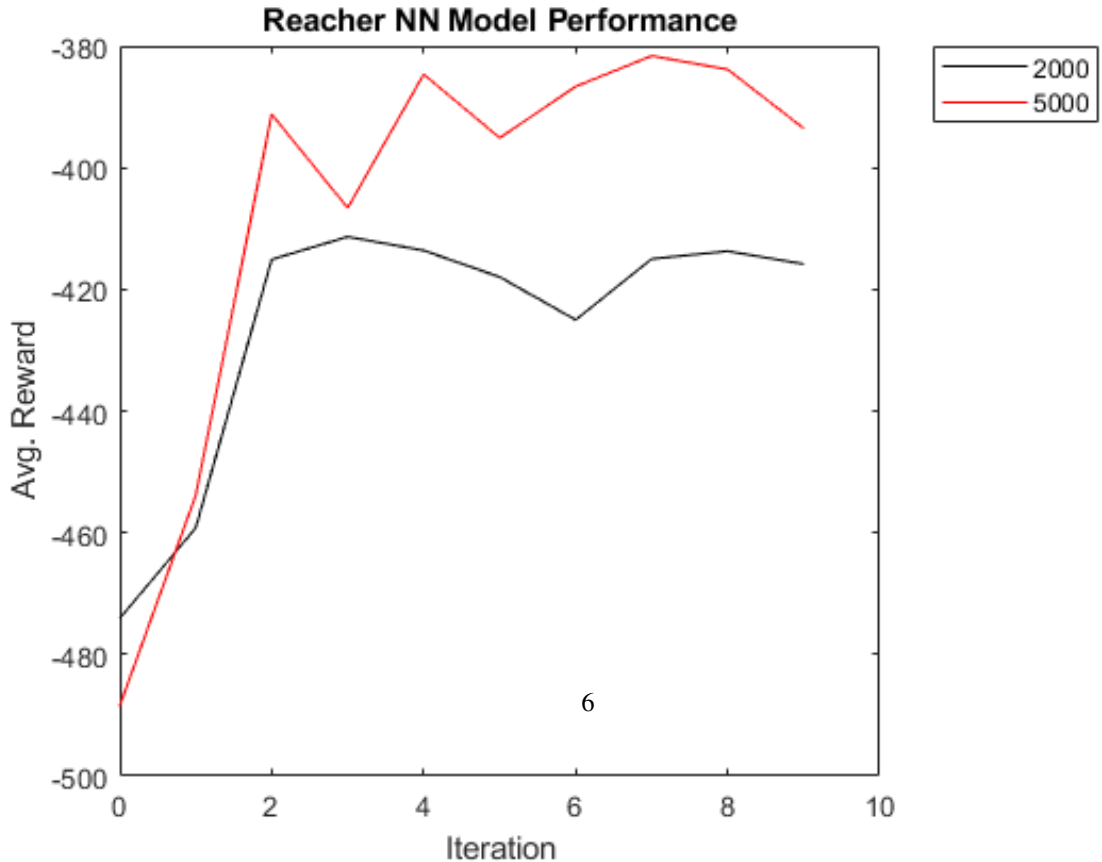


Figure 4: Test/Train Error, Reacher Neural Net

[t] 0.45



3.1.2 Gaussian Process Results

For the GP model, we looked at the impact of the number of clusters (i.e. number of individual GP models fit using K-means) on the model fit and performance (Figs. 9, 12).

For all GP experiments, we fixed the following parameters:

- MPC Horizon (H) = 15
- Number of simulated action sequences (K) = 1000
- Default GP model from Scikit-Learn (RBF kernel)
- Max number of train points = 500
- Number of subsets to test = 100

To explain the last two items, the GP model was constrained so that each model could only fit on a subset of the full data. This was done since inverting the kernel matrix, which is required to do prediction, is $O(N^3)$, where N is the size of the training set. We ensured that $N < 500$ by introducing random subsampling of the data set. The idea is that 100 subsets of size 500 were sampled from the training set, and the one with the lowest training set error was kept. This procedure was used to try to find a subset of the training set that was most representative of the full training set.

[t] 0.45

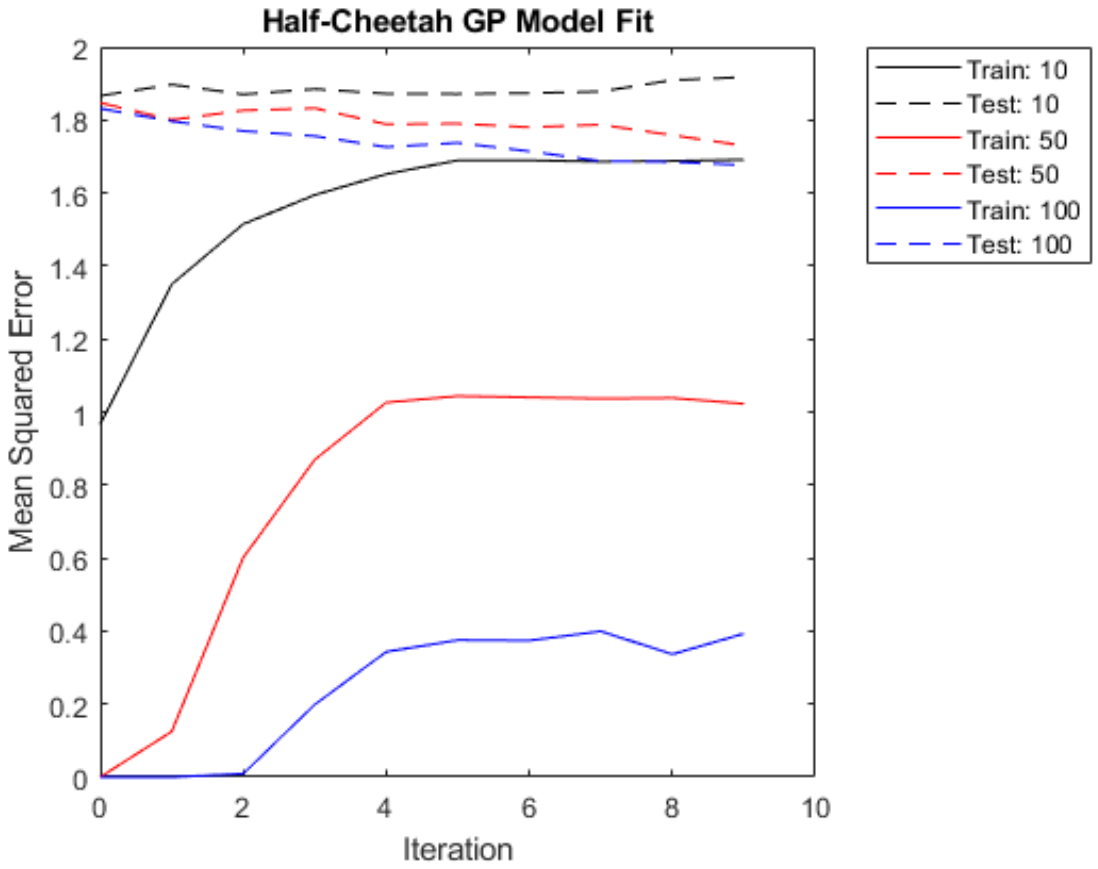
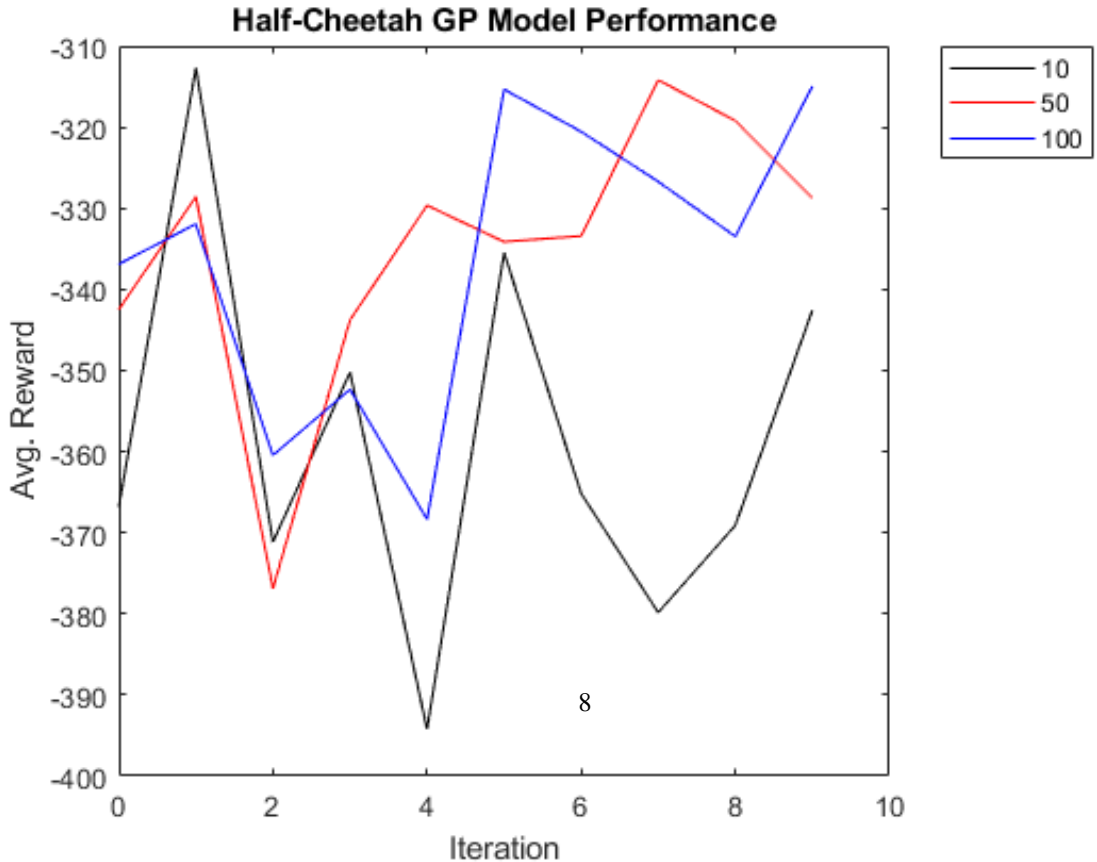


Figure 7: Test/Train Error, Half-Cheetah GP

[t] 0.45



[t] 0.45

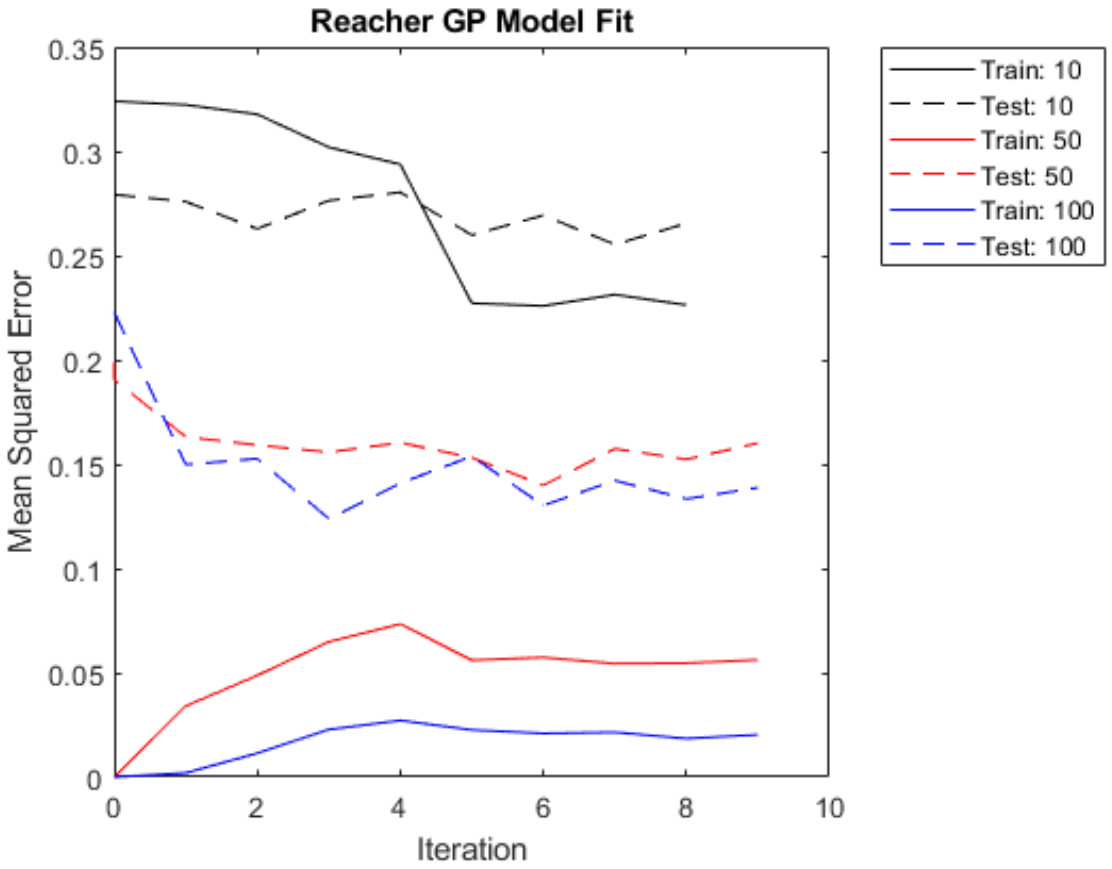
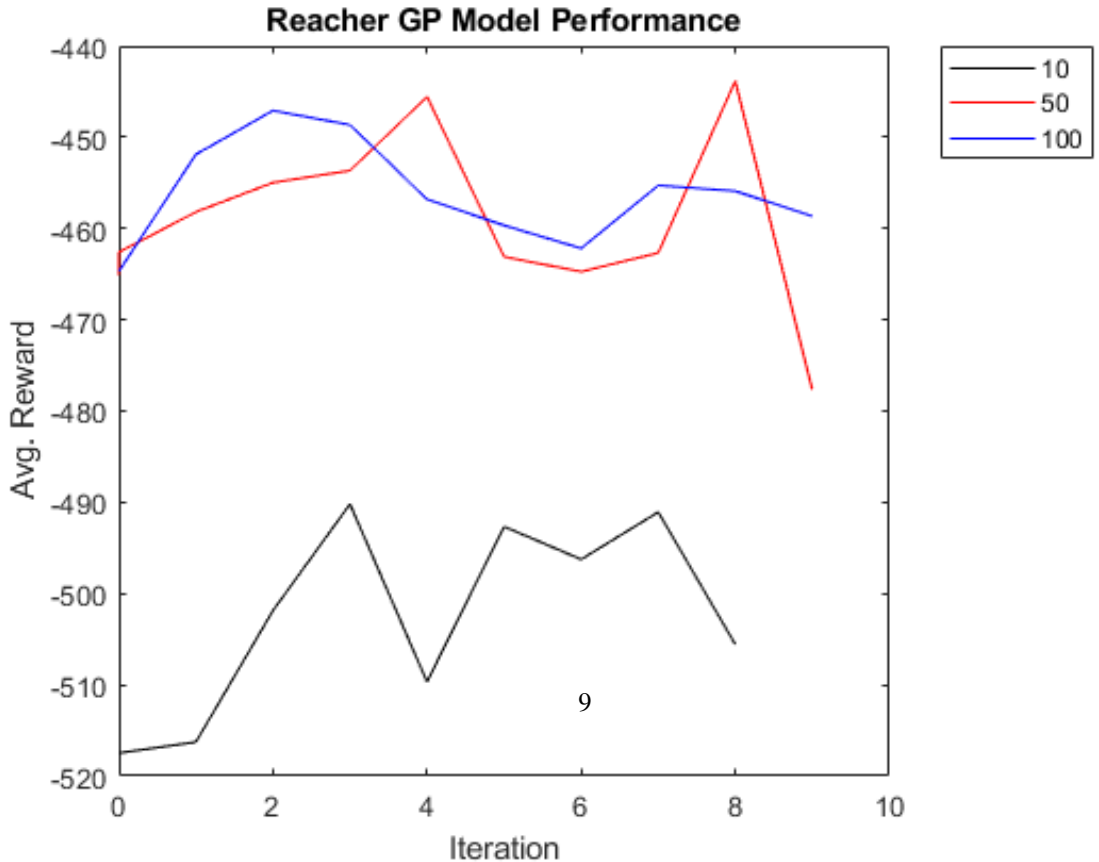


Figure 10: Test/Train Error, Reacher GP

[t] 0.45



REACHER:
PLOT 1: NNET COMPARISON (PATHS)
todo: fix mdl$_iters = 100, unnormalized, changenum_paths = 2k, 5k$

- algo iters vs. reward, 2k, 5k
- algo iters vs. MSE(train and test), 2k, 5k

PLOT 2: GP COMPARISON (Local GPs)
- algo iters vs. reward (10, 50, 100)
- algo iters vs. MSE(train and test), (10, 50, 100)

HALF CHEETAH:
PLOT 3: NNET COMPARISON (PATHS)
todo: fix mdl$_iters = 100, unnormalized, changenum_paths = 2k, 5k$

- algo iters vs. reward, 2k, 5k
- algo iters vs. MSE(train and test), 2k, 5k

PLOT 4: GP COMPARISON (Local GPs)
- algo iters vs. reward (10, 50, 100)
- algo iters vs. MSE(train and test), (1, 10, 50, 100, one per channel)

3.2 Ensemble Method Performance

From our initial experiments, it appeared that the neural network performed much better than the GP models. Due to time constraints, we decided to focus on ensemble methods to improve the neural network results. At the time of writing, we fixed a minor bug but were unable to get the full results in time for submission. The completed results, as well as a further discussion on the impact of ensemble approach, can be found at https://people.eecs.berkeley.edu/~govvijay/DRL_results.html.

4 Discussion and Conclusion

Ultimately, our experiments showed that the neural network approach performed the best, in terms of both model fit and performance. Increasing the number of sample action sequences (K) did improve the average reward as expected, without having a significant impact on the model fit. In addition, the gap between the test and train error dropped over time, showing that the neural network was generalizing well as it learned.

In contrast, the GP model did quite poorly. Increased the number of clusters (i.e. local GP models) did improve the return and reduce model error, but the average performance stayed relatively stable across iterations. In addition, the gap between the test and train error remained stable, suggesting that the GP model did not generalize well. Part of this may be due to the low data size per GP model (500), that was set up to ensure reasonable algorithm run times (under 2 hours per iteration).

While the discrepancy in performance does make sense from the standpoint that a neural network is a universal function approximator, it was disappointing to see that the GP model did poorly. We have a few ideas on what improvements could be made.

One key limitation was how we sampled the training dataset to find a representative subset to fit the GP on. We did random sub-sampling of the training data, but it would be better to look at more effective data reduction methods. Another idea would be to try approximate GP methods, which reduce the computational cost of doing predictions.

Second, it may be better to train a single GP per state dimension to simplify the model fitting process, as done in [Rasmussen et al.(2003)Rasmussen, Kuss, et al.]. The issue we saw in trying to implement this idea is that for large state-action space tasks, this is computationally very expensive and would likely require a parallelized approach. In addition, Rasmussen et al. were able to do uniform sampling over the entire (s,a) space to initial training data to fit the GP, which would be highly difficult to do with the Mujoco tasks given the large size of the (s,a) space.

Finally, we may explore more sophisticated methods to implement the gating network than K-means. Some other methods include using a Dirichlet Process to specify a gating network [Rasmussen & Ghahramani(2002)Rasmussen and Ghahramani] or employing locally weighted projection regression along with the GP model [Nguyen-Tuong et al.(2009)Nguyen-Tuong, Seeger, and Peters].

With these improvements to the GP model, it would then be worthwhile to investigate how adding the GP model could potentially improve performance in an ensemble of GP and NN models. The hope is that the increased diversity of models would help reduce variance in the dynamics model. We will explore this avenue in future work.

References

[Faußer & Schwenker(2011)Faußer and Schwenker] Stefan Faußer and Friedhelm Schwenker. Ensemble methods for reinforcement learning with function approximation. In *MCS*, pp. 56–65. Springer, 2011.

[Hagan et al.(2002)Hagan, Demuth, and Jesús] Martin T Hagan, Howard B Demuth, and Orlando De Jesús. An introduction to the use of neural networks in control systems. *International Journal of Robust and Nonlinear Control*, 12(11):959–985, 2002.

[Hornik(1991)] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[Ju et al.(2017)Ju, Bibaut, and van der Laan] Cheng Ju, Aurélien Bibaut, and Mark J van der Laan. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *arXiv preprint arXiv:1704.01664*, 2017.

[Kazmi & Driesen(2015)Kazmi and Driesen] Hussain Kazmi and Johan Driesen. Real-time model-based reinforcement learning with deep function approximation. Master's thesis, University of Alberta, 2015.

[Ko et al.(2007)Ko, Klein, Fox, and Haehnel] Jonathan Ko, Daniel J Klein, Dieter Fox, and Dirk Haehnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Robotics and Automation, 2007 IEEE International Conference on*, pp. 742–747. IEEE, 2007.

[Nagabandi et al.(2017)Nagabandi, Kahn, Fearing, and Levine] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

[Nguyen & Widrow(1990)Nguyen and Widrow] Derrick H Nguyen and Bernard Widrow. Neural networks for self-learning control systems. *IEEE Control systems magazine*, 10(3):18–23, 1990.

[Nguyen-Tuong et al.(2009)Nguyen-Tuong, Seeger, and Peters] Duy Nguyen-Tuong, Matthias Seeger, and Jan Peters. Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[Polydoros & Nalpantidis(2017)Polydoros and Nalpantidis] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[Rasmussen & Ghahramani(2002)Rasmussen and Ghahramani] Carl E Rasmussen and Zoubin Ghahramani. Infinite mixtures of gaussian process experts. In *Advances in neural information processing systems*, pp. 881–888, 2002.

[Rasmussen et al.(2003)Rasmussen, Kuss, et al.] Carl Edward Rasmussen, Malte Kuss, et al. Gaussian processes in reinforcement learning. In *NIPS*, volume 4, pp. 1, 2003.

[Todorov et al.(2012)Todorov, Erez, and Tassa] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

[Wiering & Van Hasselt(2008)Wiering and Van Hasselt] Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.