Extending Model-Based Reinforcement Learning to Sparse Reward Environments

Taylor Henderson
Mrinal Verghese

***Abstract*—We extend a Model-Based Reinforcement Learning algorithm and a hybrid Model-Based Model-Free extension from a paper by Nagabandi et. al to sparse reward environments. We implement a critic network trained by hindsight experience replay in order to learn dense rewards that the Model-Based algorithm needs to properly train. Our results demonstrate equivalent performance in sample efficiency to that shown in the original paper in this new sparse-reward domain.**

I. Introduction

Model-Free reinforcement learning algorithms have shown great potential for learning complex behaviors and tasks. Unfortunately these algorithms often suffer from high sample complexity and are not easy to train. Model-Based reinforcement learning algorithms offer much more efficient learning at the cost of some model performance. Nagabandi et. al presented a Model-Based Reinforcement Learning algorithm that used a medium-sized neural network to learn the dynamics of an environment [1]. They then built a model-based policy using the learned dynamics model to plan and execute actions in a Model Predictive Control (MPC) formulation. This policy was able to quickly learn to complete the task, but its performance didn't match model-free methods. They then trained a hybrid model-based model-free policy by initializing a model-free learner with the model-based policy. This hybrid policy was able to match the performance of the purely model-free approach while being 3-5 times more sample efficient.

Nagabandi at. al's MPC controller has a limited horizon for which it can plan. The controller is dependent on getting a reward in this horizon in order to select an optimal action. This makes their controller unreliable for sparse reward environments where it is unlikely the sparse reward will be encountered in the controllers horizon. Our solution to this is to implement a critic network that can learn the reward in sparse reward environments and provide dense reward heuristic to the controller allowing the MPC controller to complete the task. We evaltuate our modification on both the purely model-based approach and the model-based model-free hybrid approach and demonstrate that we can matcht the results of the original paper in this model-free environment.

II. Background

A. Learning a dynamics model

In model-based reinforcement learning, a dynamics model is used to make predictions, which are then used to select actions. Let $\hat{f}_\theta(s_t, a_t)$ denote a learned discrete-time dynamics function, parameterized by θ, that takes the current state s_t and action a_t and outputs an estimate of the next state at time $t + \Delta t$.

In the paper by Nagabandi et. al, the dynamics model is represented by a multi-layer perceptron parametrized by its network weights, θ. The authors found that it was more reliable to have the network predict the change in state Δs rather than the new state s_{t+1}. As such, the model is trained by minimizing the error:

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

Training data is initially collected by sampling starting configurations $s_0 \sim p(s_0)$, executing random actions at each timestep, and recording the resulting trajectories $\tau = (s_0, a_0, ..., s_{T-2}, a_{T-2}, s_{T-1})$ of length T. The model is trained on this data before the controller starts interacting with environment. Subsequent on-policy data is later collected and used to retrain the network in order to improve the accuracy of the dynamics model for the trajectories encountered when completing the task.

B. The model-based controller

Controlling the agent involves optimizing a series of actions to maximize reward such that:

$$(a_t, ...a_{t+H-1}) = \underset{a_t, ...a_{t+H-1}}{\arg\max} \sum_{t'=t}^{t+H-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) \quad (2)$$

This optimization problem is hard to solve in practice, made even more challenging by potential inaccuracies in the dynamics model. To maximize reward for a given horizon H the authors implement a random shooting method. They generate K candidate series of actions $(a_{tk}, ..., a_{(t+H-1)k})$ and then use the learned dynamics model to roll out K corresponding trajectories $(s_{tk}, ..., s_{(t+H-1)k})$. They then pick the best trajectory based on reward, execute the first action a_t and then use and updated state s_{t+1} to plan again. Controlling the agent in this MPC formulation helps make the model-based policy robust toward inaccuracies in the dynamics model. Note that this method requires access to the underlying reward function.

The model is initially trained with randomly sampled trajectories as mentioned above. As the agent continues to explore the environment using its current learned dynamics model, further on-policy trajectories are collected. These trajectories help fine tune the dynamics model and improve the overall performance of the agent.

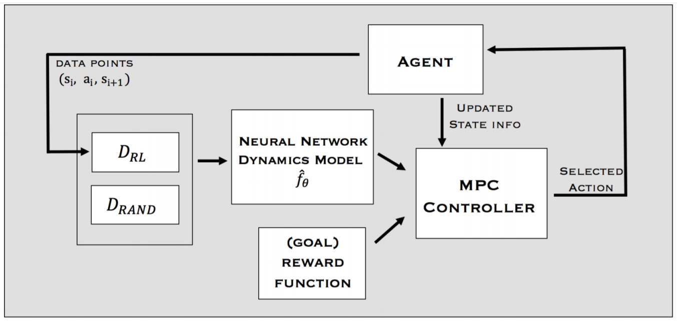

Fig. 1. Illustration of Model-Based controller. On the first iteration, random actions are performed and used to initialize D_{RAND}. On all following iterations, this iterative procedure is used to train the dynamics model, run the MPC controller for action selection, aggregate data, and retrain the model.

C. Hybrid Model-Based Model-Free Controller

The model-based reinforcement learning algorithm can learn complex tasks using a very small numbers of samples, but its final performance is worse than model-free algorithms. Thus the benefits of model-based and model-free learning can be combined using the model-based learner to initialize a model-free learner. To do this, the authors train A policy to mimic the learned model-based controller. The resulting imitation policy is used as the initialization for a model-free reinforcement learning algorithm.

We first gather example trajectories with the MPC controller which uses the learned dynamics function $\hat{f}_\theta$. We parameterize this new policy π_ϕ as a conditionally Gaussian policy $\pi_\phi(a|s) \sim N(\mu_\phi(s), \Sigma_{\pi_\phi})$, in which the mean is parameterized by a neural network and the covariance is a fixed matrix. The policy's parameters are trained using the behavioral cloning objective:

$$\min_\phi \frac{1}{2} \sum_{(s_t,a_t)\in\mathcal{D}^*} ||a_t - \mu_\phi(s_t)||_2^2 \tag{3}$$

Rather than using a randomly initialized policy like most model-free reinforcement learning algorithms use, the hybrid algorithm uses this pretrained policy π_ϕ. The agent can then be trained by any standard model-free reinforcement learning method, the authors elect to use Trust Region Policy Optimization (TRPO). With this method, the hybrid controller is able to match the performance of purely model-free methods while being 3-5 times more data efficient.

III. METHODS

While the model-based reinforcement learning algorithm presented is capable of quickly learning complex tasks, it is not compatible with sparse reward environments in its vanilla form. The algorithm's model predictive controller plans using a random shooting method with a fixed horizon H. The K generated candidate trajectories can then only look ahead to time step $t + H$. If a meaningful reward isn't encountered in this horizon, the controller will be unable to select among the K candidate trajectories and the agent will take an essential random action. To remedy this, we propose the addition of a critic network to the training method, and to utilize Hindsight Experience Replay (HER). Note that the resulting modified model-based controller behaves similarly to the original model-based controller, and as such, the same process for training the Hybrid Model-Based Model-Free controller can be used after the model-based controller is trained.

A. The Critic Network

The critic network $f_\omega(s_t)$ is a function parameterized by weights ω that, given a state s_t, predicts the long term reward starting at this state and following the policy given by the model-based controller. We implements the critic network as a multilayer perceptron with weights ω and a similar structure to the dynamics network.

We train this network using the same random and on-policy trajectories used to train the dynamic model. We first calculate the long term reward y_t for ever state s_t as:

$$y_t = \sum_{n=t}^{T} r_n * \gamma^{n-t} \tag{4}$$

where T is the horizon of the trajectory and γ is the discount factor on the reward. We then define the loss for the network as:

$$L = \frac{1}{|D|} \sum_D (y_t - f_\omega(s_t))^2 \tag{5}$$

Where D is our data set of trajectories.

We use this network to replace the underlying reward function when selecting among candidate trajectories in the MPC controller. For each candidate trajectory $(s_{tk}, ..., s_{(t+H-1)k})$ we evaluate the critic model on each state s_{tk} and average all the predicted long term rewards. We can use this method to select the best trajectory based on the highest average reward and proceed as normal with the model-based algorithm.

B. Hindsight Experience Replay

When training our critic network, we run into a classic problem with reinforcement learning in sparse reward environments. The critic network guides the agents behavior, but the agents behavior provides data to train the critic network. As such, the agent will initially only ever encounter the goal by chance in the epsisode duration, and only then will the critic network start to learn the relationship between state and reward. Note that we include various information about the goal pose for the agent in the input for the critic network, including the goal coordinate and a vector from the agent end-effector to the goal.

We employ Hindsight Experience Replay (HER) to improve sample efficiency for training the critic network [2]. For each trajectory in an epoch HER creates a duplicate trajectory and adds it to the training set. In this duplicate trajectory the terminal state S_T is treated as the new goal and the agent receives a reward of 1 at this last time step. All goal related information in the states of this duplicate trajectory is modified to reflect the new goal S_T and the final reward is discounted back along the trajectory. HER helps the critic network learn the relationship between state, specifically the goal information, and reward. As the reward is discounted along the trajectory, the critic network still learns to value states that get the agent to the reward quicker.

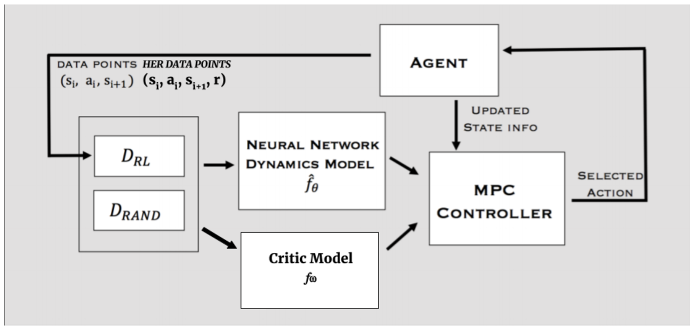

Fig. 2. Modified model-based controller for sparse reward environments. Both regular trajectories and HER trajectories are collection during episodes and used to train the critic model in addition to the dynamics model

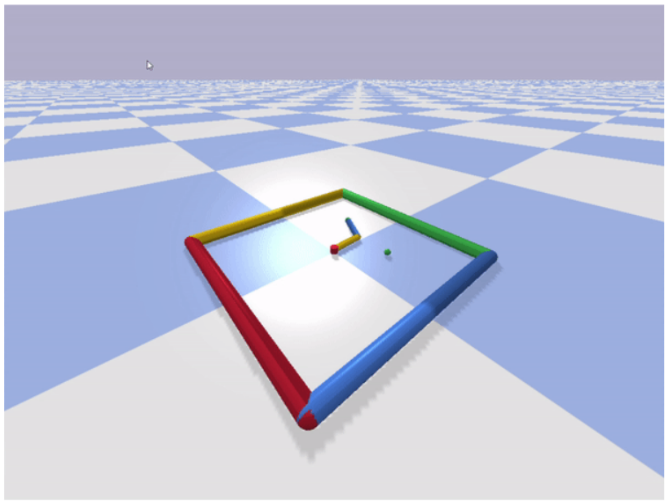

Fig. 3. Modified reacher environment with target position of end-effector specified as green dot. The central joint is always initiated in the same position at $\frac{\pi}{2}$ radians, and the elbow joint is randomly initialized anywhere between $\frac{-\pi}{2}$ and $\frac{\pi}{2}$ radians.

IV. Results

We tested our algorithm on the Modified PyBullet Reacher environment. In this environment, a 2-link arm must actuate its joints in order to achieve a fixed target end-effector position designated by a green dot. In this relatively simple environment, the state space is 10-dimensional and the action space is 2-dimensional. The agent only received sparse rewards as it moves, gaining a positive reward only when it's end-effector reaches the target position.

In Fig. 4, we show results comparing the cumulative reward achieved by our pure model-based approach, a pure model-free approach (Trust Region Policy Optimization or TRPO), and our hybrid Model-based-Model-free approach as applied to this sparse-reward environment. We can see that our extension to sparse-reward environments achieves a similar performance to [1] in terms of sample efficiency. Even with sparse rewards, the agent can very quickly learn motions that reach the goal and can achieve qualitatively good behavior more than 20× faster than the model-free method. However, the accuracy, and thus average cumulative rewards achieved by our pure model-based approach were not sufficient to match the final performance of state-of-the-art model-free learners, motivating our combination of a sample-efficient model-based method with a high-performing model-free method. We see that oscillations occur in both the model-based and hybrid lines. We attribute most of these to the nature of our sparse-reward environment in our model-based approach, and note that the model-based controller is biased, so we expect some initial drop-off in the cumulative reward of our initialized model-free learner as it unlearns that bias. Even with these irregularities, we see that the hybrid approach in this environment is about 3x faster than the purely model-free method.

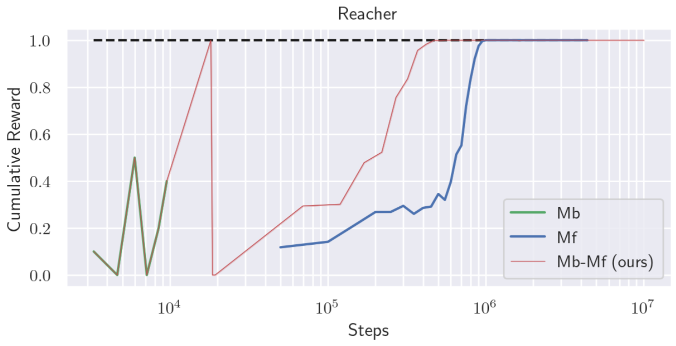

Fig. 4. Sample efficiency comparison between model-based, model-free, and hybrid Mb-Mf approaches.We see approximately 3x more sample efficiency of the hybrid Mb-Mf approach over the purely model-free method.

Fig. 5 compares the losses of both the dynamics and critic networks and how they fluctuate as more on-policy rollouts are incorporated into our original off-policy dataset. For both the dynamics and critic models, we notice some upward spikes in the loss. We attribute this to the fact as the MPC controller collects new on-policy rollouts that are different from what the networks have seen before, the loss will spike initially and then steadily decrease. Finally, Fig. 6 displays the percentage of successful trajectories completed at each aggregation iteration of out model-based algorithm, thus motivating our choice for the number of aggregation loops used. We note that even with significant oscillations, the graph trends upwards, showing the value in adding more aggregation loops. In the future, more testing will be done to fine-tune this and other hyper-parameters of our model.

Future experiments will incorporate other environments containing agents with more complex locomotion patterns in order to further validate the efficacy of our spare-reward mod-

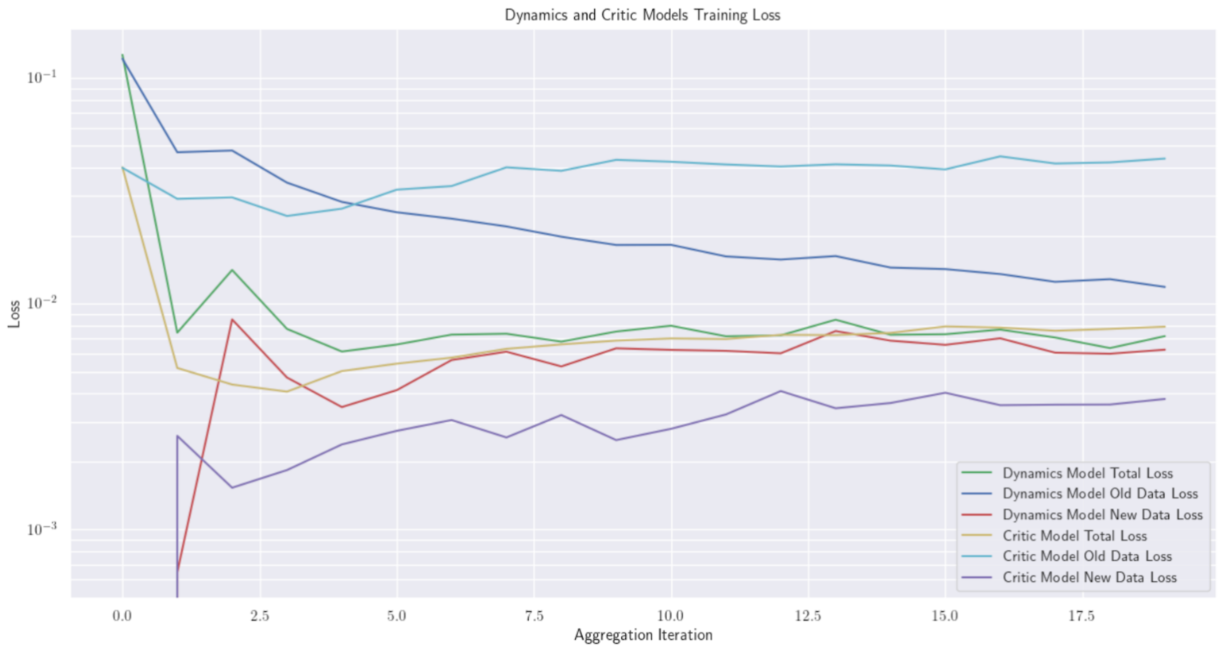

Fig. 5. These lines show the loss per aggregation loop for both the dynamics and critic models when training the model-based learner. For both models, the original off-policy (old), the newly acquired on-policy rollouts (new), and total dataset losses are plotted.

ifications. We also plan to incorporate sub-goals for trajectory-following instead of having only one single target point.

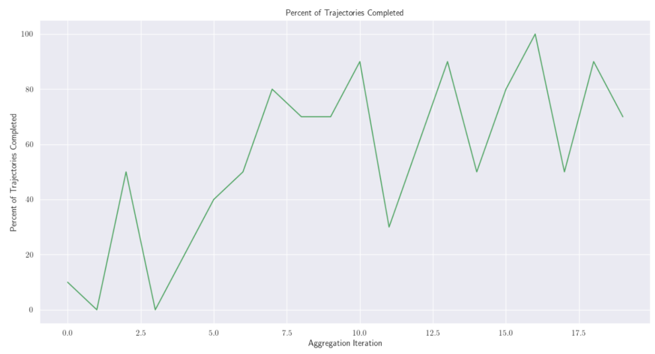

Fig. 6. Percentage of successful trajectories completed at each aggregation iteration of out model-based algorithm.

V. Discussion and Conclusion

Based on the results above, we can see that our method is a successful extension of previous work done in [1] to sparse-reward environments. Some future work and improvements to our current methods include testing other methods of evaluating the best candidate trajectory using the critic network. Currently, the expected long-term rewards are averaged across all states in the trajectory. However, we could alternatively sample the expected long-term reward from only the last state or last few states in the trajectory. Alternatively, we could evaluate the gain in reward each trajectory offers by subtracting the expected long-term reward of the initial state(s) from that of the final state(s). We can also test how the horizon of the MPC controller affects performance. This was tested in the original paper, but it would be interesting to determine whether or not the use of the critic network changes the optimal horizon. It is possible that the critic network is noisier, and thus the horizon should be longer to compensate as a result. On the other hand, it may need to be shorter to avoid propagation of error. We can also test other methods of training the critic network; for example, using state-action pairs instead of only states. Finally, as mentioned before, we want to extend these methods to other more complex environments with various agent locomotion patterns. Only then will we truly be able to evaluate our sparse-reward modifications against the original model. Overall, however, we show an effective extension of previous work in a hybrid model-based-model-free approach to spare-reward environments.

Acknowledgment

We would like to thank our professor Dr. Michael Yip and our teaching assistant Jacob Johnson for their advice and guidance through the course of this project.

References

[1] 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:1708.02596, 2017.
[2] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. In the Annual Conference on Neural Information Processing Systems (NIPS).