Dexterous Hand Manipulation with Learning from Demonstrations

Final Project Report
EN.520.637 Foundations of Reinforcement Learning
Fall 2020

Kejia Ren (kren6), Shimin Pan (span20)

***Abstract*— Learning from demonstrations, as in the sampling based Demo Augmented Policy Gradient (DAPG) algorithm, has shown potential for solving continuous control problem of dexterous manipulators. By incorporating human demonstrations with imitation learning and augmented gradient, DAPG is able to learn the policy faster and more robust. In this project, we conducted experiments to numerically verify the effectiveness of DAPG algorithm. Furthermore, we made several extensions based on possible bottlenecks of DAPG. Firstly, we substituted this on-policy method with a simple off-policy strategy by sampling with a mixture policy combining an old policy from the previous iteration. Secondly, we used bootstrapping to reduce the variance of advantage estimation in policy improvement. Thirdly, we modeled the policy network with a Recurrent Neural Network (RNN) structure, in order to utilize histories for effective control. We conducted simulation experiments for the in-hand pen manipulation task, and evaluated the performance of proposed extension strategies with metrics including success rate and episodic total rewards.**

I. Introduction

Dexterous multi-fingered manipulator is of great value for robotics applications in human-centric environment, as it enables the robots to perform complicated tasks such as grasping, tools manipulation, etc. However, the high versatility comes at the price of challenging controlling task. Reinforcement learning, as a model agnostic approach, circumvent the issues of inaccurate dynamics model and state estimation under contact-rich scenarios, and is proved to be a prominent approach for the continuous control of dexterous manipulators. However, high dimensionality still makes the learning slow and impractical on real physical systems, due to the sampling inefficiency. By incorporating human demonstrations, the sampling complexity will dramatically reduced, and the learning will be biased towards more robust strategies.

II. Manipulation Task

In our project, we select one specific In-hand manipulation task to conduct experiments and evaluate performance of the policies. Particularly, as shown in Fig. 1, the goal of our task is controlling a high degree-of-freedom (DoF) hand manipulator to re-position the blue pen to its target orientation represented by the green pen. The frequently changing contact and interaction between the manipulator and the object makes it difficult to model the dynamics of the system. However, model-free reinforcement learning method with extensive sampling and rich function approximators can efficiently find the representations for complex solutions under different scenarios.

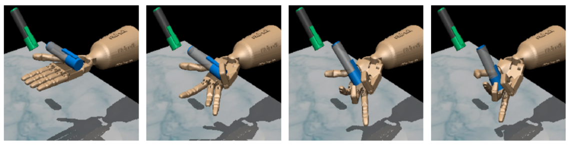
Fig. 1. In-hand manipulation task, whose goal is to re-position the blue pen to match the orientation of the green target.

A. Environment

To setup our experiments, we use an external customized OpenAI Gym environment[1] simulated with MuJoCo physics engine[2].

The platform simulates the analogue of a 24 DoF ADROIT robot hand [1] as the controlled agent for the task. As shown in Fig. 2, the blue arrows mark the 24 controllable hand joints. Besides, the platform also provides very realistic simulations of physics by carefully modeling the kinematics, dynamics and sensing details. It enables us to access the measurements of joint positions, object poses, etc, and to visualize the demonstrations of any given policies.

B. Problem Formulation

The problem is formulated as an episodic task. The episode will terminate either when a failure occurs or it exceeds the preset maximum steps (e.g. 200). Specifically, the failure state is defined that the pen falls from the hand, ie. the height of the pen position is less than a threshold.

The associated Markov Decision Process (MDP) is defined as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma\}$, where $\mathcal{S} \subset \mathbb{R}^n$

[1]https://github.com/vikashplus/mj_envs
[2]http://www.mujoco.org

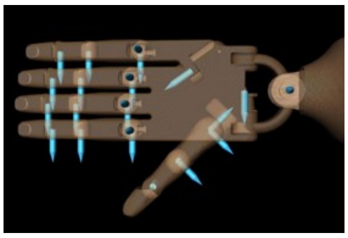
Fig. 2. Simulated ARDOIT robot hand. The blue arrows represent each of the controllable hand joints.

is the state space, $\mathcal{A} \subset \mathbb{R}^m$ is the action space, $\mathcal{R} \subset \mathbb{R}$ is the reward, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathcal{R}$ is the transition dynamics unknown to us, and $\gamma \in [0, 1)$ is the discount factor. The goal of this problem is to find the optimal stochastic policy such that it maximizes the expected discounted returns:

$$\text{maximize}_\pi \quad \mathbb{E}_{\pi,\mathcal{M}}\left[\sum_{t=0}^{T} \gamma^t R_t\right] \tag{1}$$

where R_t is the reward received at step t, and T is the length of the episode.

For our specific application, the detailed definitions for the components in our problem formulation are described below:

1) **State**: The state is continuous represented by a 45-dimensional vector ($n = 45$). Each of the dimensions represents one of the measurements including hand joint angles (24 DoF), pen pose (6 DoF), pen linear velocity (3 DoF), target pen orientation (3 DoF), and difference of current and target pen pose (6 DoF). These measurements can be directly observed and accessed from the simulator.
2) **Action**: The action is also continuous, represented by a 24-dimensional vector ($m = 24$). Each of the dimensions is defined as the desired angle at the next step for one of the 24 joints on the robot hand.
3) **Reward**: The reward we use is heavily shaped to help with faster learning. The shaping of the reward is adopted from the default setting of the environment we use. Basically, the reward is a function of the state, mainly related to the difference of current and target pen pose. Additionally, bonus for success and penalty for failure are both applied in shaping the reward. The reward can be explicitly expressed as

$$R_t = R_{diff} + R_{bonus} + R_{penalty}$$
$$R_{diff} = -||\boldsymbol{p}_t - \boldsymbol{p}_*||_2 + \langle \boldsymbol{\phi}_t, \boldsymbol{\phi}_* \rangle$$
$$R_{bonus} = \begin{cases} +10 & \text{if } ||\boldsymbol{p}_t - \boldsymbol{p}_*||_2 < 0.075 \\ & \text{and } \langle \boldsymbol{\phi}_t, \boldsymbol{\phi}_* \rangle > 0.9 \\ +50 & \text{if } ||\boldsymbol{p}_t - \boldsymbol{p}_*||_2 < 0.075 \\ & \text{and } \langle \boldsymbol{\phi}_t, \boldsymbol{\phi}_* \rangle > 0.95 \\ 0 & \text{otherwise} \end{cases}$$
$$R_{penalty} = \begin{cases} -5 & \text{if } \boldsymbol{p}_t^{(z)} < 0.075 \\ 0 & \text{otherwise} \end{cases}$$

where $\boldsymbol{p}_t$, $\boldsymbol{p}_*$, $\boldsymbol{\phi}_t$, $\boldsymbol{\phi}_*$ respectively denote the current position, target position, current orientation and target orientation of the pen at time t; $\boldsymbol{p}_t^{(z)}$ is the height of the pen at time t; and $\langle \cdot, \cdot \rangle$ denotes the inner product of two given vectors.

III. Methods and Results

The baseline method we use is Demo Augmented Policy Gradient (DAPG) [2] algorithm. By combining imitation learning and online fine-tuning, DAPG is proved to be one of the most successful methods for solving dexterous hand manipulation problems. We re-produce part of the experiments in their work and conduct numerical ablation study to verify the effectiveness of DAPG.

Furthermore, based on the nature of DAPG algorithm, we think about possible problems of DAPG that potentially bottleneck the efficiency of its learning, and we propose three corresponding improvements: 1) off-policy sampling, 2) bootstrapping, and 3) recurrent policy, to improve the performance of DAPG.

A. DAPG

The DAPG algorithm is derived from the classical REINFORCE with Baseline, which is a Monte-Carlo policy gradient method for optimizing the stochastic policy. It involves one neural network for the value-function baseline $\hat{v}(S, \mathbf{w})$ and another network for the policy $\pi(A|S, \boldsymbol{\theta})$. At the beginning of each training iteration, it will first sample a batch of trajectories using current policy, and estimate the advantage δ_t of each step with the actual return G_t by

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2}$$
$$\delta_t = G_t - \hat{v}(S_t, \mathbf{w}) \tag{3}$$

And then the baseline network weights $\mathbf{w}$ and the policy network weights $\boldsymbol{\theta}$ will be successively updated with a small step size $\alpha^{\mathbf{w}}$ and $\alpha^{\boldsymbol{\theta}}$ by

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha^{\mathbf{w}} \delta_t \nabla \hat{v}(S_t, \mathbf{w}) \tag{4}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha^{\boldsymbol{\theta}} \delta_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \tag{5}$$

For the major contributions of DAPG, it adopts additional training strategies which are crucial for fast convergence. These strategies can be summarized as the followings:

1) Pre-training with behavior cloning. Utilizing an expert demonstration dataset $\mathcal{D}$ which includes 25 pre-collected trajectories, the policy network will be pre-trained with supervised

learning by maximizing the likelihood of actions observed in the demonstration dataset:

$$\text{maximize}_{\boldsymbol{\theta}} \sum_{(\mathrm{S},\mathrm{A})\in\mathcal{D}} \ln \pi(\mathrm{A}|\mathrm{S},\boldsymbol{\theta}) \tag{6}$$

2) Natural policy gradient (NPG) [3]. In the fine-tuning phase of the training, instead of using the vanilla REINFORCE gradient g expressed in Eq. 7, it will make a normalized gradient ascent update to the policy pre-conditioned with the Fisher Information Matrix $F_{\boldsymbol{\theta}_t}$:

$$g = \delta_t \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) \tag{7}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \sqrt{\frac{\alpha^{\boldsymbol{\theta}}}{g^T F_{\boldsymbol{\theta}_t}^{-1} g}} F_{\boldsymbol{\theta}_t}^{-1} g \tag{8}$$

3) Augmented policy gradient from demo data. In fine-tuning, besides using the newly sampled data at each training iteration to compute the policy gradient, DAPG also keeps utilizing the demonstration data to help shape the gradient. Briefly speaking, this is done by merging the gradient computed from the demonstration data into the original gradient computed from the newly sampled data with a small decaying weight. This strategy can elegantly guide the exploration at the early stage of fine-tuning.

The results of our verifying experiments are shown in Fig. 3, where the plotted curves show how the performance evolves with respect to training iterations. For better visualization, all the curves are smoothed by averaging over successive 5 iterations. We use two evaluations to compare the performance of different methods. One metric is the success rate. It is considered as a success when the difference of the pen pose and its target is within a small tolerance at the end of the episode. Another metric is the score, which is the averaged total rewards among the evaluation episodes.

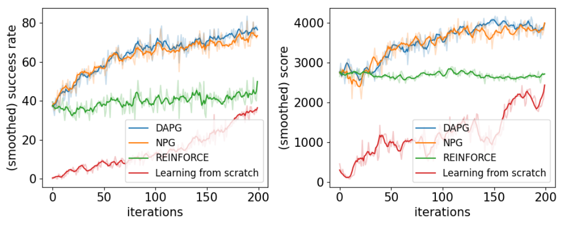

Fig. 3. Evaluations in experiments verifying effectiveness of different strategies used in DAPG.

From the experiment results, compared with learning from scratch, behavior cloning with demonstrations is able to provide a good initialization of the policy network. And compared with the vanilla REINFORCE which improves very slow, NPG is pretty effective on helping with fast convergence.

B. Off-policy Sampling

The first potential problem of the original DAPG is that it uses on-policy reinforcement learning for fine-tuning. This could be data inefficient, and more importantly, it might not be able to sufficiently explore the region near the optimal policy when the demonstration data used for pre-training is not optimal.

Therefore, we propose to encourage more off-policy exploration by sampling with a mixture policy $\beta\pi_t + (1-\beta)\pi_{t-1}$, which combines the current policy π_t and the policy from the previous iteration π_{t-1} with a weight β. The results are shown in Fig. 4. It can be observed that this simple off-policy sampling improves the performance a little in the early iterations, but it does not make a big difference afterwards.

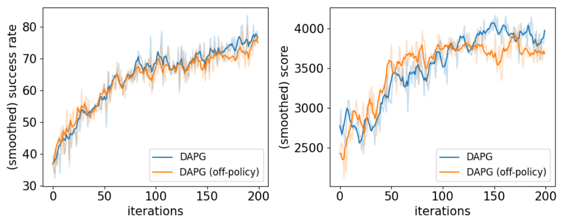

Fig. 4. Evaluations in experiments on off-policy sampling.

C. Bootstrapping

In policy gradient methods, the estimation of the advantage function is important to policy improvement since it is highly related to estimating the policy gradient. As introduced previously, DAPG estimates the advantage by Monte-Carlo, which is unbiased due to the use of actual returns. However, this estimation is presumed to be of high variance, and is thus likely to slow down the training.

To deal with this probable high variance of advantage estimation in DAPG, we propose to use Temporal Difference (TD) bootstrapping. One simple example of bootstrapping is Actor-Critic, which uses one-step TD to estimate the advantage:

$$\delta_t = R_t + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}) \tag{9}$$

This estimation in Actor-Critic is able to achieve low variance, but it is also presumably bad as verified in Fig. 5, since the bias is high due to the inaccurate approximation of the value-function baseline $\hat{v}$. However, if the estimation of the advantage is well balanced between bias and variance, the training process will potentially benefit a lot from it. Therefore, we finally apply Generalized Advantage Estimation (GAE) [4] which is a useful technique to adjust the bias-variance trade-off.

GAE is defined as the exponentially-weighted average of k-step TD estimators, which can be

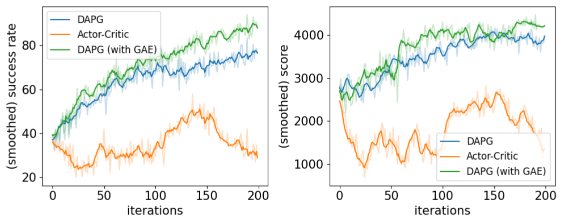

Fig. 5. Evaluations in experiments on bootstrapping.

simplified as:

$$\delta_t^{\mathrm{GAE}(\gamma,\lambda)} := (1-\lambda)\sum_{k=1}^{\infty}\gamma^{k-1}\delta_t^{(k)} \tag{10}$$

$$= \sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k}^{(1)} \tag{11}$$

where $\delta_t^{(k)}$ is the k-step TD residual:

$$\delta_t^{(k)} = \sum_{l=0}^{k-1}\gamma^l R_{t+l} + \gamma^k\hat{v}(S_{t+k},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}) \tag{12}$$

To note, $\delta_t^{\mathrm{GAE}(\gamma,0)}$ is exactly the one-step TD same as in Actor-Critic; in contrast, $\delta_t^{\mathrm{GAE}(\gamma,1)}$ is γ-just, and thus has low bias regardless of the accuracy of $\hat{v}$, but it has high variance due to the sum of terms. With carefully tuned λ, we can successfully improve the performance of DAPG by a great margin, as shown in Fig. 5.

D. Recurrent Policy

The policy used in DAPG only depends on the current state and has no memory of previous states. This assumption can become too simple to make good decisions in practice. Intuitively, a better decision could be made if some important information obtained through interactions with the environment in previous steps, such as the weight and implicit shape of the object, can propagate forwards.

Therefore, we propose to replace the original policy network modeled by a 3-layer fully-connected neural network with Long Short-Term Memory (LSTM) [5] architecture as shown in Fig. 6

The experimental results are shown in Fig. 7, it can be obviously observed that the newly applied LSTM policy eventually helps accelerate the learning.

IV. Conclusion and Discussion

By conducting verification experiments with the original DAPG algorithm, we can verify that pretraining with demonstrations is very helpful for initializing the policy. Even though merely behavior cloning is not sufficient to optimize the policy due to its overfitting to the limited demonstration data

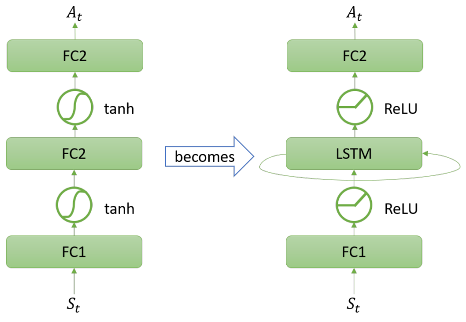

Fig. 6. Replacement of the policy network with LSTM architecture. The left is the original fully-connected policy network where the activations are $tanh$; the right is our newly applied LSTM policy network where the activations are changed to $ReLU$.

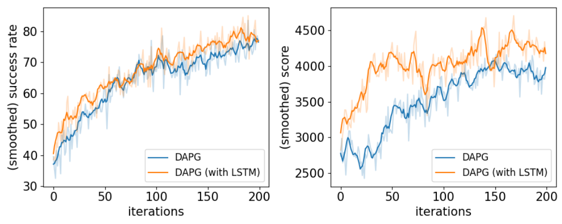

Fig. 7. Evaluations in experiments on LSTM policy.

and bad generalization to the randomized environment, it can improve the exploration efficiency at the beginning of fine-tuning stage and lead to faster convergence to the optimality.

Besides, in the context of in-hand manipulation task, three extensions from DAPG have been implemented and tested in the simulation environment we use. In summary, GAE and LSTM policy both help accelerate the learning, and outperform the original DAPG settings in terms of the success rate and total rewards. The summarized results are shown in Fig. 8.

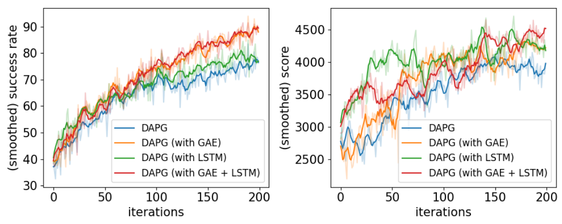

Fig. 8. Summarized results of experiments on our extensions.

For off-policy sampling during training, we need to take into account the distribution shift of different policies, and thus we decide to use the policy from just one iteration ahead instead of the very early policies to sample. However, the performance is almost the same with the original on-policy DAPG algorithm. The reason might be that the policy from the previous iteration is highly similar with the current policy, especially when the training starts to converge and policy update step is small.

Thus the exploration still remains insufficient compared with pure off-policy methods. In the future, we could investigate and try other more elegant off-policy strategies for promising improvement.

For the bootstrapping extension, We did several experiments to search over different values of λ with fixed γ. Particularly, we tested with $\lambda = 0$ (Actor-Critic), 0.1, 0.3, 0.5, 0.8, 0.97, and 1.0. As shown in Fig. 9, the best balance between bias and variance is the case of $\lambda = 0.97$. With relatively small λ, the performance will drastically downgrade, and it further verifies that our system is especially sensitive to the bias in the advantage estimation. In most of our cases, estimation with low bias should be prioritized relative to low variance.

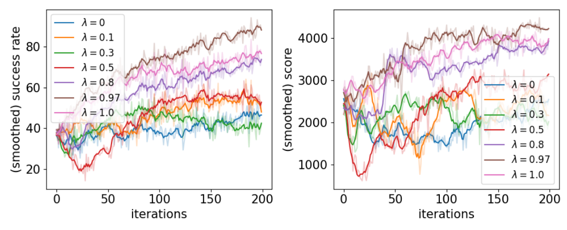

Fig. 9. Performance of GAE applied with different λ value.

For applying the recurrent policy extension, with more complex model of LSTM and more parameters to optimize compared with the original policy network in DAPG, we observed in our experiments that the pre-training of LSTM policy requires longer time. Even though applying LSTM policy significantly improved the converging speed and the converged score in the later fine-tuning, at this moment we cannot guarantee the fine-tuned LSTM policy can perform similarly better when interacting with other objects or conducting similar manipulation tasks. The complex model of LSTM might be more prone to overfitting and has worse generalization. In the future, more testing concentrated on the generalization ability of policies should be done to further study the characteristics of different types of policies.

The convergence of training will typically take very long time. With only limited time in this project, we were unable to wait until it converges every time. As you can see, we evaluate all the experiments based on the first 200 iterations of training. 200 iterations is long enough for optimizing to a sufficiently good solution, so it can provide us with fairly reliable justifications for comparing the learning efficiency of different methods, which is also our focus in this project. However, we cannot safely make conclusions about how good the solution is after convergence.

V. Contributions

Each individual member has been actively and frequently taking part in the discussions for better understanding the derivations in the relevant literature, studying the source codes in the external packages, and inspiring new ideas with each other. They both contributed intellectually into the project to make steady progress. Moreover, they also both contributed on preparing the presentation, collecting experiment results and writing the final report. Besides, some of their other individual work are listed below:

Kejia Ren:

1) Set up the simulation environment locally and re-produced part of the original DAPG experiment results;
2) Implemented and tested simple off-policy sampling methods;
3) Implemented and tested the one-step TD bootstrapping (Actor-Critic). Tested and analyzed the Generalized Advantage Estimation (GAE) technique with different experiment settings;
4) Implemented the LSTM policy network and associated pre-training, sampling and evaluation methods; Run experiments to compare the performance.

Shimin Pan:

1) Set up the simulation environment and re-produced part of the original DAPG experiment results;
2) Responsible for the interaction between the the program and the MuJoCo physics engine;
3) Extracted the state and action spaces out of the configuration files for all tasks;
4) Analyzed the Generalized Advantage Estimation (GAE) method quantitatively and implemented the corresponding simulation experiments for ablation study;
5) Theoretically comparing the DAPG algorithm with another policy gradient method with demonstrations, Deep Deterministic Policy Gradient from Demonstrations (DDPGfD) [6].

References

[1] V. Kumar, Z. Xu, and E. Todorov. Fast, strong and compliant pneumatic actuation for dexterous tendon-driven hands. In *2013 IEEE International Conference on Robotics and Automation*, pages 1512–1519, 2013.

[2] Aravind Rajeswaran*, Vikash Kumar*, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[3] Sham Kakade. A natural policy gradient. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 1531–1538, Cambridge, MA, USA, 2001. MIT Press.

[4] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan,*

Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[6] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.