Question 4: Diffusion Policy Policy Optimization (DPPO)

1 Literature review

Diffusion-based reinforcement learning methods. In this part, we discussed related approaches that directly train or enhance diffusion-based policies using reinforcement learning (RL) techniques. For a more comprehensive survey on diffusion models for RL, please refer to [10].

A series of works utilizes diffusion models as an action policy and generally applies Q-learning. Diffusion Q-Learning (DQL) [9] introduces a state-action critic for the final denoised actions and backpropagates the gradient from the critic through the entire Diffusion Policy (actor) denoising chain, similar to standard Q-learning. Implicit Diffusion Q-learning (IDQL) [4] proposes learning the critic to select actions at inference time for either training or evaluation while fitting the actor to all sampled actions. In [5], the authors suggest using the critic to re-weight the sampled actions for updating the actor itself, akin to the weighted regression baselines DAWR and DRWR introduced in this work.

These works on diffusion-based RL commonly rely on approximating the state-action Q function and using it to update the diffusion actor in some form — policy gradient update has been deemed challenging due to the multi-step denoising process [6]. However, inaccurate Q values may lead to biased updates to the actor, which can result in training collapse in model fine-tuning as it starts with decent pre-training performance but quickly drops to zero success rate. While Q-learning methods generally achieve better sample efficiency when they can solve the task of interest, DPPO focuses largely on challenging long-horizon robot manipulation tasks where training stability is highly desired and Q-learning methods fall short. Besides, DPPO also

addresses the challenges of the multi-step denoising process resulting from policy gradient updates.

2 Proposed Methods

Since the focus of the question requires us to replicate the results in the original DPPO paper, we do not need to propose new methods, especially for Part 1.

3 Answers of the Questions

3.1 Replicate the Results with TensorFlow Agent

We can replicate the results with Tensorflow Agent [1].

TensorFlow Agents is a library for reinforcement learning (RL), which provides a collection of RL algorithms, environments, and tools to facilitate the development and training of RL agents. It offers a wide range of RL algorithms, e.g., DQN, DDPG, PPO, and SAC. These algorithms are implemented using TensorFlow. It also includes a set of environments that can be used for training and evaluating RL agents. It supports popular environment libraries such as OpenAI Gym [2] and DeepMind Control Suite. Besides, the library provides components for building neural networks and policies used in RL agents, including common network architectures like MLPs, CNNs, and LSTMs, as well as various policy types like Gaussian and Categorical policies.

Implementation plan. To replicate the results with Tensorflow Agents, we consider the following aspects.

- **Environments.** TensorFlow Agents have provided standard RL environments, such as OpenAI Gym [2] and DeepMind Control Suite. As the paper also evaluates their DPPO on these environments, we can directly use the environment APIs in TensorFlow Agents for our replicated agents to interact with the environments.
- **Policy network.** DPPO mainly uses a multilayer perceptron (MLP) network throughout the paper, which consists of two main components: a time embedding module and an MLP network.

The time embedding module is a sequential model that processes temporal information. It starts with a SinusoidalPosEmb layer, which generates sinusoidal positional embeddings. This layer helps the model understand the relative positions or timestamps of the input data. The positional embeddings are then passed through a Linear layer to learn a dense representation of the temporal information. Another Linear layer is applied to transform the representation back to the desired dimensionality.

The MLP module is responsible for predicting the mean value based on the input features and the time embeddings. It is implemented as a ResidualMLP, which combines residual connections with multilayer perceptrons. The ResidualMLP consists of a sequence of layers defined in the ModuleList. The first layer is a Linear layer, which takes the concatenated input features and time embeddings. The second layer is a TwoLayerPreActivationResNetLinear block, which applies a residual connection and two Linear layers with ReLU activation. This block helps in learning complex representations while allowing gradients to flow easily through the network. The third layer is the Linear layer, which maps the learned representations to the desired output dimensionality. Finally, an Identity layer is used to pass the output from the previous layer unchanged.

- **Policy.** To implement DPPO with TensorFlow Agents, we need to hack its policy module to add customized code following DPPO. The policy takes the policy network into compute actions and/or distributions over actions from TimeSteps.

  Another choice is to use the policy wrapper provided by the TensorFlow Agents library. We can wrap the standard PPO policy and modify it according to DPPO. Policy wrappers are a subclass of Policy and can therefore be used just like any other standard policy in the TensorFlow Agents library.

- **Training flow.** The training flow is straightforward to implement by combining environments and policies in a loop. The loss function, optimizer can be translated from the original PyTorch code to TensorFlow counterparts.

  Notably, as DPPO mainly focuses on fine-tuning workloads and provides pretrained model weights, we need to load a PyTorch pretrained model into TensorFlow to replicate the results.

3.2 Unclear parts in the paper

It is unclear how the $V(\cdot)$ function is trained in the paper.

3.3 Any mistakes or errors?

At page 7, in the first paragraph, the original equation

$$\bar{r}\left(\bar{s}_{\bar{t}}, \bar{a}_{\bar{t}}\right) := \sum_{t' \geq t} \gamma_{\mathrm{ENV}}^{t} \bar{r}\left(\bar{s}_{\bar{t}(t',0)}, \bar{a}_{\bar{t}(t',0)}\right), \quad \bar{t} = \bar{t}(t,k), \tag{1}$$

should be

$$\bar{r}\left(\bar{s}_{\bar{t}}, \bar{a}_{\bar{t}}\right) := \sum_{t' \geq t} \gamma_{\mathrm{ENV}}^{t'-t} \bar{r}\left(\bar{s}_{\bar{t}(t',0)}, \bar{a}_{\bar{t}(t',0)}\right), \quad \bar{t} = \bar{t}(t,k), \tag{2}$$

3.4 Reviews for this paper

I recommend to accepted the paper.

Paper summary. The paper introduces Diffusion Policy Policy Optimization (DPPO), a framework that enables fine-tuning of pre-trained diffusion-based policies using on-policy reinforcement learning (RL) methods, specifically policy gradient optimization (PPO) [7]. The authors demonstrate improved performance over baselines on challenging tasks with long horizons, sparse rewards, and pixel-based observations. DPPO leverages environment parallelization to train faster than off-policy RL methods, making it well-suited for GPU-accelerated simulators. The paper provides extensive comparisons to various RL fine-tuning methods, including novel baselines, and conducts thorough ablation studies to identify best practices for DPPO. The proposed framework enhances structured exploration, training stability, robustness, and generalization during deployment, demonstrating its efficacy in both simulated and real-world scenarios.

Strengths.

- Timely topic: Fine-tuning diffusion policy for real-world robots is an important area of research, and the experiments and findings presented in this paper are significant for the community.
- Novel framework and best practices: The paper introduces a novel dual-layer MDP framework and proposes several best practices for DPPO, such

as fine-tuning only the last few denoising steps and using DDIM sampling, leading to improved efficiency and performance.

- Improved performance on challenging tasks: DPPO achieves better results compared to baselines on long-horizon and sparse reward tasks. It demonstrates strong performance in benchmark tests, particularly in tasks involving pixel-based observations and long-horizon rollouts, and achieves highly efficient zero-shot transfer from simulation to real-world robotics tasks with minimal sim-to-real performance gap.
- Comprehensive experimentation: The paper contains extensive experiments and analysis related to fine-tuning diffusion policy via PPO, covering various aspects such as denoising steps, network architecture, GAE variants, and the effects of expert data. It also provides extensive comparisons to various RL fine-tuning methods, including novel baselines, and conducts thorough ablation studies.

Weaknesses.

- Poor presentation and organization: The authors fail to present the extensive experimental results in a concise and focused manner. The paper is somewhat hard to read and comprehend.
- Limited novelty: The diffusion policy and PPO used in this work have limited novel contributions. The advantage of diffusion policy over vanilla parameterization is already well-established and not a contribution of this paper. Previous works have proposed the idea of considering a denoising process as a multi-step MDP [3, 8, 9], and this paper simply integrates this MDP into the environmental MDP.
- Limited improvement in sample efficiency: DPPO does not seem to significantly enhance sample efficiency, manifesting a similar convergence speed as other online reinforcement learning methods.
- Potential loss of multimodality after fine-tuning: Figure 10-Right shows that after fine-tuning, the Diffusion Policy gradually converges into a deterministic policy, potentially losing the advantage of expressing multimodal action distributions.

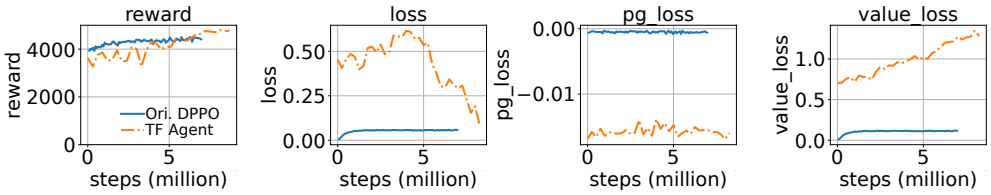

Figure 1: Current replication results on HalfCheetah-v2.

4 Testing Plan (Evaluation)

We plan to test our implementation by comparing the training performance of our agents with the original DPPO agents using the same environment. For agile development and testing, we choose the OpenAI gym environments and currently choose HalfCheetah-v2 [2]. For the hyperparameters setting, we follow the default settings in the DPPO paper.

Figure 1 presents the current replication results on HalfCheetah-v2, comparing average rewards, loss, policy gradient loss (pg_loss), and value loss (value_loss) throughout the training process of original DPPO and our replication. Though the plot of rewards present a similar pattern, the other three are different. The current replication results are not good and we will try to fix problem soon.

References

[1] TensorFlow Agents. `https://www.tensorflow.org/agents`, 2024.

[2] G Brockman. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[4] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

[5] Bingyi Kang, Xiao Ma, Chao Du, Tianyu Pang, and Shuicheng Yan. Efficient diffusion policies for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[6] Michael Psenka, Alejandro Escontrela, Pieter Abbeel, and Yi Ma. Learning a diffusion model policy from rewards via q-score matching. *arXiv preprint arXiv:2312.11752*, 2023.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[8] Dmitry Shribak, Chen-Xiao Gao, Yitong Li, Chenjun Xiao, and Bo Dai. Diffusion spectral representation for reinforcement learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[9] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[10] Zhengbang Zhu, Hanye Zhao, Haoran He, Yichao Zhong, Shenyu Zhang, Haoquan Guo, Tingting Chen, and Weinan Zhang. Diffusion models for reinforcement learning: A survey. *arXiv preprint arXiv:2311.01223*, 2023.