Keys of Reinforcement Learning In Robot Visual Tasks

KAI YANG

Abstract

Reinforcement learning (RL) has emerged as a powerful tool to design navigation policies and control strategies for robots. However, how to design effective and efficient RL parameters is still an open question for various robotic tasks. In this study, we focus on the problem of teaching a robot to solve tasks with different configurations and find the keys to train the agent effectively. To achieve this, we designed a visual robot within a pushing box scenario and investigated the effects of parameters such as the visual encoder, policy network architecture, and training framework. In our work, we have fine-tuned the different hyperparameters (e.g., batch size, buffer size, learning rate) in the PPO algorithm to achieve the highest success rate for the task. We have also investigated if the training performance can be improved by using long short-term memory (LSTM), deeper convolutional neural networks, or imitation learning. Our results showed the feasibility of a learning-based controller and, therefore, served as a guideline and benchmark for implementing a learning-based controller for other similar tasks in real-life settings.

1. Introduction

There has been a growing interest in adopting artificial intelligence (AI) solutions in the perception, navigation, and control domains for robots, as they have been shown to be able to accomplish tasks better than humans. Among the AI algorithms, reinforcement learning (RL) (Kaelbling et al., 1996) is a popular method that is used in this domain. Hence, we aim to investigate the effectiveness of RL in guiding and controlling a robot when performing a specific task with visual observations. Our goal is to develop a RL-based controller that can guide a robot to find a target and push it to the desired area by using visual data. Even though there are many RL algorithms that can be suitable for this application, we only consider the Proximal Policy Optimization (PPO) algorithm (OpenAI, 2022) as the benchmark. PPO has been a popular algorithm in this domain due to its state-of-the-art performance regarding adaptation capabilities in different tasks.

However, in most cases, designing suitable training schemes can be a painful process, with too many modules and parameters to be confirmed before training. For instance, there are various visual encoders, such as simple CNN, deeper CNN, ResNet-based CNNs (He et al., 2016), etc. Besides, different policy neural network architectures also affect a lot of the performance of a RL controller. Moreover, to improve training sample efficiency, expert demonstrations are fused into some offline RL methods (Kober & Peters, 2010). Among these domains, variant combinations are still not clear to robot learning (Ravichandar et al., 2020), which hinders the further development of RL-based controllers for robots. In this study, by using PPO as our RL algorithm, we comprehensively explore policy network architectures, different deeper convolutional neural networks, and imitation learning to achieve better results. As such, these investigations provide a design framework for robot learning before starting training without considering potential optimal solutions.

Our contribution lies in exploring a synergy of diverse policy network architectures and deeper convolutional neural networks (CNNs), including ResNet-based models, to enhance the efficiency and effectiveness of RL-based controllers in visual tasks. We also investigate imitation learning compared to the PPO algorithm, boosting training sample efficiency and accelerating the learning process. This innovative combination enables our RL controller to adeptly guide robots in identifying and manipulating targets, crucially without the need for explicit task indicators during testing. Our work not only provides a robust, efficient RL solution for complex visual robotic tasks but also simplifies the preliminary design process, offering a clear framework for configuring RL controllers in real-world applications.

In the following section, we will formulate our RL problem. By using this framework, we have done the following: 1) tune the PPO's hyperparameters to achieve optimal results; 2) investigate if the results can be improved by adding LSTM layers to our models; 3) compare the results achieved by convolutional neural network variants of different architectures; 4) observe if the results can be improved by giving human demonstrations to the models (i.e., imitation learning). The rest of the article is structured as follows. Section 2 introduces robot learning with RL; Sec.3 details the robotics settings and tasks performed; Sec.4 describes the

methods utilized; Sec. 5 presents the results and discussion; Sec. 6 concludes with future insights from our simulation experiments.

2. Related Work

The integration of RL in robot learning has been a burgeoning area of research, driven by the need for adaptive and intelligent robotic systems. A significant contribution in this field has been the application of Deep RL (DRL). For instance, Levine et al. (Levine et al., 2016) demonstrated the effectiveness of DRL in end-to-end training of robotic systems for complex tasks like manipulation and grasping, using raw pixel data as input.

Another notable advancement is the use of RL in robotic navigation and control. Tai and Liu (Tai et al., 2017) explored how mobile robots could effectively navigate in complex environments using deep Q-networks, showcasing the potential of RL in real-world applications. Similarly, Ender et al. (**?**) applied RL for robot navigation, specifically focusing on obstacle avoidance using convolutional neural networks (CNNs), setting a precedent for aerial robotics.

In the realm of multi-robot systems, Matignon et al. (Matignon et al., 2012) and Foerster et al. (Foerster et al., 2016) have made significant contributions. They explored the dynamics of cooperative and competitive behaviors among robots, using RL to facilitate effective strategies for tasks like pursuit-evasion games, highlighting the adaptability of RL in multi-agent scenarios.

The integration of imitation learning with RL has also gained traction. For example, studies by Nair et al. (Nair et al., 2018) have shown how imitation learning can be utilized to bootstrap the training process in RL, enhancing the learning efficiency in robotic manipulation tasks.

Moreover, the development of sample-efficient RL algorithms has been crucial for practical robotics applications. Haarnoja et al. (Haarnoja et al., 2018) introduced Soft Actor-Critic (SAC), an algorithm that optimizes policy in an entropy-augmented setting, demonstrating significant improvements in sample efficiency and robustness in robotic control tasks.

Finally, the concept of sim-to-real transfer in RL, where policies are trained in simulated environments and then transferred to real-world scenarios, has been a key area of research. Rusu et al. (Rusu et al., 2017) and James et al. (James et al., 2019) have made substantial contributions in this area, showing how policies developed in simulated environments can be effectively adapted to physical robots, overcoming the challenges posed by real-world complexities.

However, before transferring the trained policy to a real robot, a fine-tuned model is required, which has not been fully investigated.

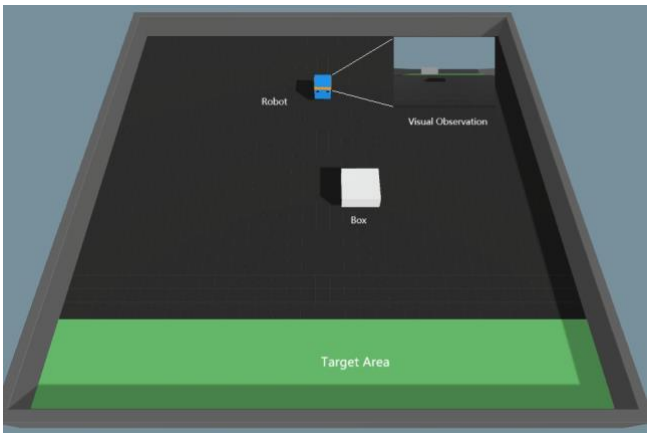

Figure 1. Pushing box task with visual perception

3. Problem Formulation

The key components in a RL problem include: the state s, the agent, the action a, the reward r, and the environment. In our system, the agent is an RL-based robot, which receives the state s as input and produces an action a following a policy π. The action space is defined by discrete movements tailored to guide the robot towards a box and subsequently push it to a designated *Target Area*, as shown in Figure 1.

The agent's objective is to learn a policy $\pi_\theta(s, a)$ that prescribes the optimal action a for each state s. In deep RL, this policy is represented by a neural network with parameters θ, which are trained to select the most favorable action for any given state.

Our simulated environment is a simplified 2D representation of a room, consisting of a robot with forward RGB camera and a box, the latter serving as the target object. The layout of the room remains unchanged across episodes, except for the box's position, which is randomized each episode. The agent is further challenged by variable lighting conditions that shift after every episode, testing its adaptability.

Interactions of the robot with the environment, such as advancing towards and maneuvering the box, induce changes within the environment, evidenced by alterations in the robot's position and orientation. These modifications are computed utilizing the robot's equations of motion, accounting for the cumulative influence of thrust and body angular rates on the robot's dynamics. The *Visual Observation* in the figure underscores the robot's perception capacity, pivotal for recognizing the environment's state and determining the subsequent action.

3.1. Observation Space

The state *s* encapsulates a representation of the current status of the environment, serving as the critical input to the agent. It comprises significant features that enable the agent to exploit the environment and make informed decisions. Within our application, the observed state is only a visual input from the robot's onboard camera.

3.2. Action Space

The action space defines the set of all possible actions the agent can take. In the context of our robotic application, this action space includes six discrete actions, namely {forward, back, turn-left, turn-right, left, right}, aimed at navigating the robot in the environment towards the achievement of its goal.

3.3. Reward Functions

The reward function provides feedback to the agent upon executing an action, resulting in environmental changes. The reward is structured as follows: a positive reward of +5 is allocated when the robot successfully reaches the target, incentivizing the discovery of the target. A negative reward of −5 is imposed if the robot move out of area, deterring the agent from engaging in collision-prone behavior. Given that these rewards are dispensed solely at the terminal step, the agent often experiences a period with no reward, culminating in a sparse reward function. To mitigate the associated learning challenges, an additional penalty given each step to encourage agent to finish task quickly, namely $-1/MaxStep$.

3.4. Learning Objective

The overarching aim for the agent is to adjust the parameters θ in the policy $\pi_\theta(s, a)$ to maximize the cumulative reward obtained within an episode, thus steering the robot efficiently towards the target.

4. Methods

4.1. PPO Baseline

Unlike many on-policy RL algorithms, PPO does not need to discard the old experiences and collect a new batch of experiences after each policy update. PPO allows the policy (i.e., the neural network) to be updated multiple times using different batches of experiences. This greatly reduces the training time as fewer data are needed to be collected each time. To further speed up the training, we have used six parallel environments that the agent can interact with. The agent can take different actions in these environments and collect different sets of experiences in parallel. Aside from speeding up the data collection, parallel environments reduce the correlation in the dataset. Besides improving the time efficiency, the PPO algorithm is designed to have a more stable training. This is done by adding a new constraint that limits how much the updated policy can differ from the old policy [1]. Due to the above-mentioned benefits, PPO is well suited for our complex application.

We have chosen Proximal Policy Optimisation (PPO) algorithm to train the agent as PPO has achieved state-of-the-art performance in many similar control problems [1]. PPO uses the actor-critic framework shown in figure 2 [2]. The critic is a value function estimator and uses a deep learning model to predict the state-value $V(s)$ from the state s. The state-value $V(s)$ is the expected value of the discounted cumulative reward by starting at state (s). In essence, the state-value $V(s)$ measures how good it is to be in state (s). The actor derives the policy $\pi_\theta(s, a)$ which maps the optimal action (a) to each state (s). For a given state (s), the actor uses a deep learning model to predict a continuous probability distribution over the action space. When the actor is trained successfully, the optimal action will receive the highest probability.

In our implementation, the actor and critic have the same deep learning model architecture, as shown in figure 3. The image component of the state is processed by a convolutional neural network (CNN) variant while the vector component of the state is fed into a 2-layer neural network (NN). The extracted features from the image and vector are then combined and fed into another 2-layer neural network for prediction. The actor predicts the probability distribution over the action space while the critic predicts the state-value $V(s)$.

At the start of the training, the actor and critic neural networks are initialised with random weights. By taking random actions, the agent collects experiences that are stored in the form: $\langle s_t, a_t, r_t, s_{t+1} \rangle$. In this form, the agent takes action a_t at state s_t. As a result, the agent receives a reward r_t and transits to a new state s_{t+1}.

For each experience $\langle s_t, a_t, r_t, s_{t+1} \rangle$, we can use the critic to find $V(s_t)$ and $V(s_{t+1})$ for s_t and s_{t+1} respectively. If the critic neural network is a perfect estimator of the state-value $V(s)$, the following equation will be true:

$$V(s_t) = r_t + \gamma V(s_{t+1}) \tag{1}$$

where γ is the discount factor and r_t is obtained from the agent's experience. By rearranging, we will obtain:

$$r_t + \gamma V(s_{t+1}) - V(s_t) = 0 \tag{2}$$

Due to estimation error in the critic neural network, the right-hand side of the equation is equal to a non-zero value δ_t. Hence,

$$r_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t \tag{3}$$

Table 1. Hyperparameters used for PPO.

Hyperparameter	Value
Discount factor	0.99
Batch size	1024
Buffer size	10240
Learning rate	0.0003
Beta	0.01
Epsilon	0.2
Lambda	0.95
Learning rate schedule	Linear schedule
Training episodes	5 million

The non-zero quantity δ_t is the error of the critic's estimation of $V(s_t)$ and is also referred to as the temporal difference error (TD error). We use the TD error as the loss function for our critic neural network and drive this error to zero using gradient descent. For the same experience $\langle s_t, a_t, r_t, s_{t+1} \rangle$, the actor neural network receives s_t as input and outputs the probability distribution over the action space. The following equation can be derived:

$$\delta_t = Q(s_t, a_t) - V(s_t) \tag{4}$$

Thus, the TD error δ_t also measures how good it is to take action a_t at state s_t. More specifically, the difference $Q(s_t, a_t) - V(s_t)$ shows how good a_t is compared to the average action in state s_t. If the TD error δ_t is positive (i.e., $Q(s_t, a_t) > V(s_t)$), the action a_t is favourable at s_t. Hence, the actor neural network will tune its weight such that the policy will increase the probability of a_t at s_t. Conversely, a negative TD error δ_t (i.e., $Q(s_t, a_t) < V(s_t)$) shows that the action a_t is not favourable at s_t. Hence, the actor neural network will tune its weight such that the policy will decrease the probability of taking action a_t at s_t.

In summary, the TD error δ_t determines the direction and magnitude that the deep learning model should adjust its weights towards. In practice, we use multiple experiences (i.e., batch size > 1) to find an average δ_t which is used to update the weights in Actor and Critic. This keeps the training stable.

We have implemented the environment and trained the agent using the Unity Ml-Agents Toolkit (Juliani et al., 2020). The following table shows the hyperparameters that we have used for PPO. These hyperparameters are held constant as we examine the use of Long Short-Term Memory (LSTM), CNN variants (e.g., Resnet), and imitation learning for our deep learning model.

Table 2. Characteristics of pre-collected human experiences

Characteristic	Value
Duration of recording	1 hour
Number of steps recorded	35050
Number of episodes recorded	528
Mean reward	4.492926

4.2. Variation Studies

4.2.1. Long Short-Term Memory

In addition to memoryless neural networks (refer to Figure 3), we have explored the addition of memory to the actor and critic neural networks by integrating Long Short-Term Memory (LSTM) layers. Similar to recurrent neural networks, LSTM layers learn to combine features from both current and past inputs (Hochreiter & Schmidhuber, 1997). However, LSTM's additional gates allow for more flexible mixing of current and past features, enabling it to retain older information more effectively. This potentially leads to richer feature extraction that can be utilized by the prediction layer. In our implementation, the LSTM model processes inputs from time t to $t-63$.

4.2.2. CNN Variants

Beyond the simple 2-layer CNN used in the actor and critic model, we investigated the effectiveness of other CNN variants, namely Nature CNN and Resnet, in enhancing training performance. Nature CNN, with three convolutional layers, is a larger network compared to the simple 2-layer CNN (Mnih et al., 2015). Among these variants, Resnet is the most extensive network. Its skip connections help overcome the vanishing gradient problem associated with deeper networks (He et al., 2016). While Nature CNN and Resnet have more layers, potentially leading to richer feature extraction, their larger size also implies longer training times.

4.2.3. Imitation Learning

Our approach also considers the potential benefits of incorporating pre-collected human experiences into the training of deep learning models. The characteristics of our pre-collected human data are as follows:

In our implementation, the agent is trained using both the pre-collected human experiences and its own experiences for 1,000,000 steps. After this period, the agent continues its training solely on its experiences. This strategy allows the agent to learn beyond the provided data and potentially surpass human performance. The performance of our pre-collected human experiences also serves as a benchmark for evaluating our deep learning models, where a well-trained model should achieve a mean reward close to or exceeding

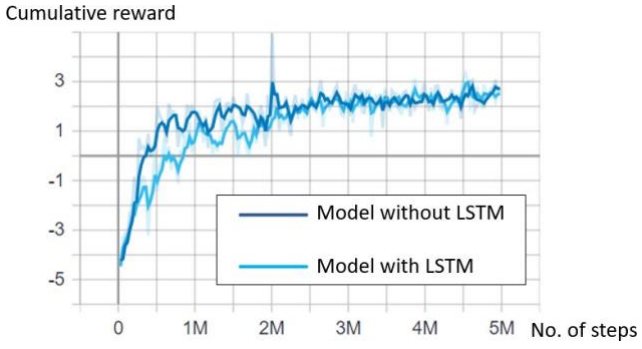

Figure 2. Comparison of training performance (cumulative reward) between model without LSTM and model with LSTM

4.492926.

5. Results and Discussions

5.1. Usefulness of LSTM

In our first experiment, we investigated the usefulness of LSTM in improving the training performance. From Figure 2, we observed that both models trained successfully. The cumulative reward for both models increased and converged at approximately 2.5, and the episode length (i.e., the number of steps taken per episode) decreased, converging at around 350. We found no significant improvement in training performance when using LSTM. In our problem formulation, we explicitly provided past information to our deep learning model, allowing the non-LSTM model to access past information. This suggests that our problem formulation already retained the required past knowledge for our deep learning models, making the use of LSTM unnecessary. Although LSTM did not worsen training performance, its significantly longer training time suggests avoiding its use.

5.2. Usefulness of Deeper CNN Variants

In our second experiment, we explored whether training performance could be improved using deeper convolutional neural networks. As shown in Figure 3, the deeper models (i.e., nature CNN and Resnet) worsened the training performance. The nature CNN's cumulative reward converged at a lower value (2) compared to the simple 2-layer CNN model (2.5), and its episode length converged at a higher value (450) than the simple CNN model (350). Among the three models, Resnet performed the worst with a significantly lower cumulative reward and a longer episode length. While deeper networks have the potential to learn richer features, they are more challenging to train. To improve training performance, it may be necessary to increase the number of training steps and experiment with different hyperparameters. The hyperparameters in Table 1 were insufficient for deeper models to outperform the simple CNN model.

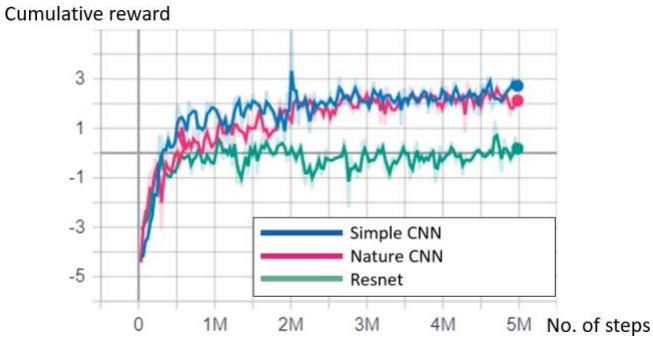

Figure 3. Comparison of training performance (cumulative reward) between CNN variants

5.3. Usefulness of Imitation Learning

In our final experiment, we assessed whether imitation learning could enhance training performance. By adjusting the number of training steps to 1,000,000, we sought optimal results with imitation learning. Figure 4 shows the training results with imitation learning. The training performance did not improve significantly with imitation learning, and the model with imitation learning underperformed compared to human demonstrations, achieving a converged cumulative reward of 2.8 against the human demonstration score of 4.492926.

Figure 4. Comparison of training performance (cumulative reward) between model without imitation learning and model with imitation learning

6. Conclusions

In our work, we have developed a RL environment using the Unity Machine Learning Agents Toolkit. We formulated the key elements (State, Agent, Action, Reward, Environment)

of our RL problem and trained our agent using the PPO algorithm. In addition to tuning the hyperparameters for PPO, we examined the use of LSTM, deeper CNN variants, and imitation learning to improve training performance.

Based on the hyperparameters in Table 1, our results indicate that neither LSTM nor deeper CNN variants improved the training performance. The relatively simpler memoryless model shown in Figure 3 achieved moderately good training performance in a significantly shorter time. Therefore, we recommend continuing with this simpler model while experimenting with other hyperparameters.

Imitation learning also showed no significant improvement using our parameters and did not outperform the human demonstrations. Further experimentation with other hyperparameters is necessary to meet or exceed this benchmark performance.

References

Foerster, J., Assael, I. A., De Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. IEEE, 2016. doi: 10.1109/CVPR.2016.90.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12627–12637, 2019.

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., and Lange, D. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020. URL https://arxiv.org/pdf/1809.02627.pdf.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Kober, J. and Peters, J. Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine*, 17(2):55–62, 2010.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Matignon, L., Jeanpierre, L., and Mouaddib, A.-I. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pp. 2017–2023, 2012.

Mnih, V., Kavukcuoglu, K., Silver, D., et al. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533, 2015. doi: 10.1038/nature14236.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

OpenAI. Proximal policy optimization algorithms. https://openai.com/blog/openai-baselines-ppo/, 2022. [Online; accessed 26-June-2022].

Ravichandar, H., Polydoros, A. S., Chernova, S., and Billard, A. Recent advances in robot learning from demonstration. *Annual review of control, robotics, and autonomous systems*, 3:297–330, 2020.

Rusu, A. A., Vecˇer´ık, M., Rotho¨ rl, T., Heess, N., Pascanu, R., and Hadsell, R. Sim-to-real robot learning from pixels with progressive nets. In *Conference on robot learning*, pp. 262–270. PMLR, 2017.

Tai, L., Paolo, G., and Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 31–36. IEEE, 2017.