The Comparison Between Different Deep Reinforcement Learning Algorithms in PyBullet and Gym Environments

Zirong Chen [1]

Abstract

Deep Reinforcement Learning techniques can be treated as ground-breaking solutions to sequential decision-making tasks. In order to show DRL's capability and potential in those kinds of tasks, in this paper, I will discuss the different performance from different Deep Reinforcement Learning Algorithms in different PyBullet and Gym environments. Then, conclusions will be drawn to better state the pros and cons of each algorithm. In the end, some personal understandings and insights will be given for reference.

1. Introduction

Deep Learning methods (Goodfellow et al., 2016; LeCun et al., 2015) can be viewed as trending techniques in real-word decision-making tasks. It shows incredible ability to deal with the caught data points can give guidance for each one of those data points. Huge potentials are shown not only in the Discriminative tasks, but in Generative tasks (Goodfellow et al., 2014) as well. However, in most of cases they are not able to give good results in sequential decision-making tasks. The main reasons can be:

- **Accumulative errors** (Zhang et al., 2016): Most Deep Learning models can not promise to be perfect, so that when encountered with sequential tasks, which requires more reliable previous decisions, Deep Learning models might suffer from the overall accumulative errors.
- **Multimodal behavior**: From the DAgger paper (Bojarski et al., 2016), especially training self-driving cars, there can be more than one 'correct' decision at each time spot in sequential decision-making tasks. Thus, only training Deep Learning models for these kind of tasks might not lead to a success.

Therefore, we need a more robust solution that take 'context decisions' into consideration. Deep Reinforcement Learning algorithms make full use of Markov Decision Process and take exceptional care of decisions within the whole sampled trajectory. Detailed introductions to Deep Reinforcement Learning algorithms will be talked about in later sections.

PyBullet[1] and Gym[2] open-sourced environments are famous for their excellent simulation of real-word sequential physics events. In order to better illustrate the better performance that Deep Reinforcement Learning algorithms are able to give, the experiments in PyBullet and Gym environments with different Deep Reinforcement algorithms will be conducted.

2. Related Work

In this paper, Deep Reinforcement Learning algorithms like Deep Q-Learning (Mnih et al., 2015), Proximal Policy Optimization (Schulman et al., 2017; 2015) and Dyna-Q Learning (Sutton & Barto, 2018; Peng et al., 2018) will be introduced and experimented with.

2.1. Markov Decision Process

In most Reinforcement Learning algorithms, problems are abstracted as Markov Decision Process (Van Otterlo & Wiering, 2012). Definitions like *state*, *action* and *reward* will not be further discussed in details. Eq.1 is the general objective function for Reinforcement Learning algorithms.

$$\theta^{\star} = arg\max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_{t} r(\mathrm{s}_t, \mathrm{a}_t) \right] \tag{1}$$

2.2. Deep Q-Learning

Deep Q-Learning (Mnih et al., 2015)(DQN, generally refers to Double Q-Learning) was first proposed to develop a game

[1]Department of Computer Science, Georgetown University, Washington, District of Columbia, USA. Correspondence to: Zirong Chen <zc157@georgetown.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

[1]View codes on Github.

[2]View introduction on Website.

agent for the Atari game. Its initial and improved versions, like Double Q-Learning(Mnih et al., 2015), Dueling Networks (Wang et al., 2016) and Prioritized Replay (Schaul et al., 2015) show DQN's fantastic capacity of dealing with discrete states. Eq.2 is the general objective function in Q-Learning.

$$\phi \leftarrow \phi - \alpha \sum_i (\mathrm{s}_i, \mathrm{a}_i) \left[Q_\phi(\mathrm{s}_i, \mathrm{a}_i) - [r(\mathrm{s}_i, \mathrm{a}_i) + \gamma Q_\phi(\mathrm{s'}_i, \mathrm{a'}_i)]\right] \tag{2}$$

2.3. Proximal Policy Optimization

Proximal Policy Optimization(PPO) was first proposed by OpenAI in 2017 (Schulman et al., 2017). PPO applied clip functions to further restrain the differences between old and new parameter sets and PPO in piratical proves its ability to give competitive results with few computational resources[3] compared with Trust Region Policy Optimization (Schulman et al., 2015).

$$\bigtriangledown_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \bigtriangledown_\theta \pi_\theta(a_{it} \mid s_{it})((\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{it'}) - V_\phi(s_{it})) \tag{3}$$

2.4. Dyna-Q Learning

Compared with Deep Q-Learning, "Dyna-Q Learning integrates planning, acting and learning" (Mnih et al., 2015; Sutton & Barto, 2018). In Dyna-Q, planning begins from an arbitrary state and model is able to learn faster and better from the 'mixture' of simulated and real experiences. In designed training process, the model learns from the planned results of simulated environments and real environments. The objective function is quite similar to the previous Eq.2. Algorithm details can be found in Figure 1.

3. Experiment Setup

3.1. Environment Selection

After exploration, CartPole-v1, MountainCar-v0, AntBulletEnv-v0 and MountainCarContinuous-v0 environments[4] are chosen. Following are the detailed dimensions of different observation space and action space.

Env	Obv_dim	Act_dim
CartPole-v1	4	2
MountainCar-v0	2	3
MountainCarContinuous-v0	2	1
AntBulletEnv-v0	28	8

3.2. Deep Q-Learning

Deep Q-Learning (Mnih et al., 2015) elegantly borrows the advantages from Reinforcement Learning and Deep Learning. It inherits strong mathematical proofs from Reinforcement Learning and strong coupling ability from Deep Learning[5]. In DQN, Q value function is abstracted as a neural network. By sampling from replay buffer, the Q network is able to be updated correspondingly to get a higher return.

However, the single Q network structure might suffer from **over-optimism** (Fan et al., 2020; Mnih et al., 2015) bought by evaluating the returns via the same network that is being updated. However, double Q-Learning, which employs two Q networks to participate 'separately' in the evaluating and updating processes. In other words, one Q network, which should be target network, is used for temporal evaluation and is kept fixed while another Q network is updating throught the evaluation results from the evaluated returns from target network. By doing this, the over-optimism issue will be addressed properly. Thus, in this experiment, DQN with double Q networks will be implemented and experimented with.

However, DQN is only famous for its capacity of handling discrete environments. When encountered with continuous environments, the initial structure is not able to deal with them. Therefore, in order to make it work, **data quantization** is implemented to convert between discrete and continuous inputs. In data quantization, several bins are employed to assign the real continuous values from the discrete indexes. By doing so, after transformation, DQN is able to fed with continuous values. During giving actions, the output from DQN is also needed be transformed back to continuous values via data quantization.

Following the detailed hyper-params settings:

Hyper Parameters	Actual Values
Num of Trajs	2000
Buffer Size	500
Epsilon Greedy	0.1
Freezing Steps	500
Update Freq	20

[3]PPO applies first-order approximation instead second-order, which, relatively speaking, theoretically and piratically lowers the computational resources.

[4]CartPole-v1 and MountainCar-v0 are discrete, AntBulletEnv-v0 and MountainCarContinuous-v0 are continuous

[5]Packages like Tensorflow(Abadi et al., 2015) and PyTorch (Paszke et al., 2019) make it even more likely to happen

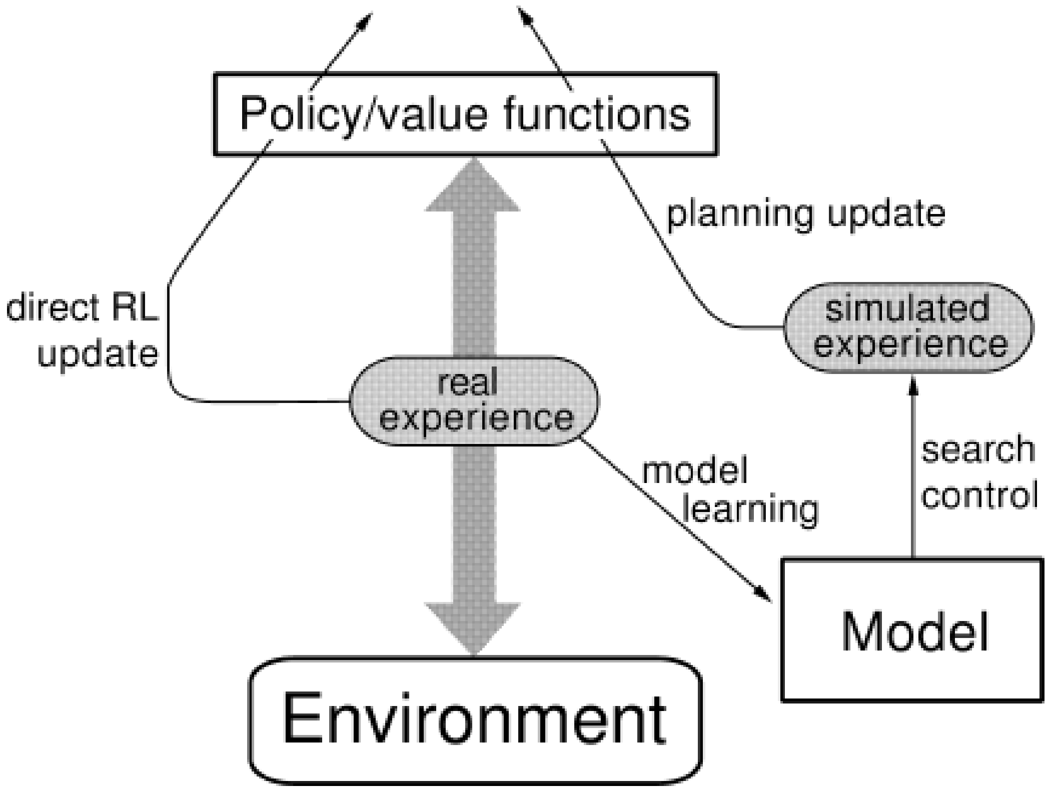

Figure 1. Dyna-Q Algorithm (Sutton & Barto, 2018; Lee, 2005)

3.3. Proximal Policy Optimization

Similar to the ideas in Trust Region Policy Optimization(TRPO) (Schulman et al., 2015), Proximal Policy Optimization(PPO) also construts a 'trust region' for updating the policy. However, instead of using second-order derivatives, PPO applied the first-order approximation and the clip function to clamp the differences(refers to the parameter sets) of the new and old policy. By doing so, PPO is able to achieve competitive(sometimes even better results) compared with TRPO with less computational resouces. In order to make it possible to happen under both continuous and discrete environments, different distributions, Categorical and Normal distributions correspondingly, are applied.

Following the detailed hyper-params settings:

Hyper Parameters	Actual Values
Epochs	500
Number of Trajs	20
Gamma	0.99
Alpha(lr)	1e-5
Beta(lr)	1e-3
Lambda	0.95
Epsilon(clamp)	0.2
c1(clamp)	0.5

3.4. Dyna-Q Learning

During experiments, the expenses brought by frequently interacting with environments are immense. However, the considerable amount of transitions are required for a better training result. Therefore, in Dyna-Q, a model to simulate the environment is employed. The training process is similar to DQN in most parts, however, Dyna-Q also needs to update its environment model and finish planning, which is to explore the simulated environments and update agent via the explored transitions.

In order to be able to handle both discrete and continuous environments, **data quantization** is also used in Dyna-Q Learning.

Following the detailed hyper-params settings:

Hyper Parameters	Actual Values
Num of Trajs	2000
Buffer Size	500
Epsilon Greedy	0.1
Freezing Steps	500
Update Freq	20
Exploration Steps	10

4. Analysis and Conclusion

4.1. Time Efficiency and Optimization

Running Reinforcement Learning experiments can be really time-consuming[6]. Here are the table of time spent to run the experiments(only parts of my experiments are recorded, and piratical time may differ according to the different parameter settings). The missing parts are marked as 'N/A', which means I failed to measure the time during training. And only the highest time spent is recorded approximately.

Models	Total Time Spent	Time Spent / Epoch/Traj
DQN	2 hours	N/A
PPO	5 hours	12 minutes
DynaQ	3.5 hours	N/A

From attempts in previous homeworks, there exist several tricks are supposed to reduce the training time.

- Shared weights in policy networks. In HW2, shared weights between policy networks and value networks are experimented. From the results, up to 5 minutes can be saved per epoch.
- Calculating accumulative rewards backwards. By calculating the reward from the end, the time spent reduced by about 3 minutes per epochs.

4.2. Hyper Parameters

Reinforcement Learning models are **super** sensitive to hyper parameter settings. One slight change might cause a disaster in the training results. For example, different number of trajectories were tried while training PPO. In my implemented version, the parameter number of trajectories is one of the parameters that effect the performance most. After attempts, 10 can be the optimal setting for the number of trajectories, however, results might differ under different environments.

Number of Trajs	Performance
10	Relatively Optimal
20	Off by 30%
5	Off by 10%

Other parameters like **learning rates** can also effect the overall performance, some unreasonable learning rates can make sense. Another finding is that the **buffer size** in DQN and DynaQ can also greatly effect the final results. If the size gets too large, the model will end up learning nothing and the performance will be trapped in a low level. If the size gets too small, the performance will vibrate violently in some interval. After attempts, the size picked was 500 or 600.

4.3. Conclusion and Insights

Experiments results can be found in Figure 2 and Figure 3. All the x-axis's in the plots measures in each 1,000 steps. According to the plots, conclusions can be drawn that:

- In AntBulletEnv-v0 environment, all three algorithms work almost well. However, PPO showed its great stability because of the restraints of the differences between the old and new polices. And Dyna-Q converged faster than DQN.
- In CartPole-v1 environment, Dyna-Q performed the worst compared with other two. At the same time, PPO is better than DQN and converged faster and more stably.
- In MountainCarContinuous-v0 environment, Dyna-Q still performs bad and bouncing within a huge interval. PPO increased stably and DQN converged faster than Dyna-Q.
- In MountainCar-v0 environment, the strange thing was that PPO got stuck around -140 no matter how many experiments were done. Dyna-Q was shaking heavily and barely learnt efficiently. For DQN, after first half, its performance boosted and turned out to be the best.

Here are my conclusions of important settings that may effect the final results from the experiments results.

- **Environment Dimension**. The actual dimensions of one environment might greatly influence the final performance of Dyna-Q. In environments with higher dimensions like AntBulletEnv-v0, it seems the environment model is able to learn better about the real environment[7]. That might be the only reason to explain why Dyna-Q is unable to stably learn in environments with lower dimensions if the implementation is correct.
- **Task Background**. The task background also matters. After running PPO in MountainCar-v0 environment, the strange performance confused me a lot. However, after exploration, I found that in MountainCar-v0 environment, the agent is asked to drive a car to cross

[6]Time is recorded on my Laptop. according to different processors and RAMs, time spent might relatively change. From HW1, Policy Gradient can cost up to 8 hours to train; From HW2, Meta-Learning (Finn et al., 2017) might need up to 1 hour to give the results of each epoch

[7]In experiment settings, the hidden size in the environment model was set to 64 so a higher input dimension might cooperate with the model better and lead to a better simulation result.

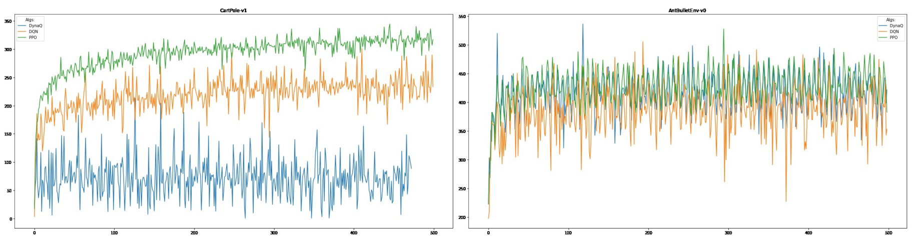

Figure 2. Model performances on CartPole-v1 env and AntBulletEnv-v0 env.

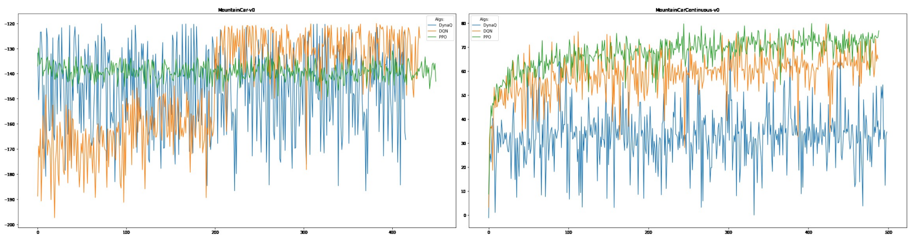

Figure 3. Model performances on MountainCar-v0 env and MountainCarContinuous-v0 env.

the mountain. The reward can only be earned once the car cross the mountain, otherwise it will get big negative results. However, in PPO, due to the existence of a Trust Region, the agent only ‘dares’ to update within a small area, which might never lead the car to cross the mountain. In other words, PPO’s caution in updating strategies damages its performance in this environment.

- **Hyper Parameters Tuning**. From personal perspective, the reason why the stability of some algorithms is not ensured is hidden with some detailed hyper parameters. However, due to time and computational limit, more experiments about hyper parameters tuning, like buffer size, learning rate and number of trajectories, were not conducted.

5. Future Work and Personal Summary

5.1. Future Work

Due to the limitation of personal computational resources, I was not able to conduct a systematical analysis of hyper parameter settings. However, nowadays, some academic works have not published their best parameter settings. Some of them even seem that they are hiding their parameters. It is also said that the most annoying things about Reinforcement Learning, other than long training time, is the hyper parameter settings. In the future, more thorough experiments should be done to find the key factors that effect the overall performance of each Reinforcement Learning algorithm implemented.

5.2. Personal Summary

After this semester’s learning, I grew the Reinforcement Learning thinking mode. I am now aware of the existence of Reinforcement Learning algorithms and the philosophy behinds them, in lots of daily events. From the way I learned how to speak, to the learning process of this course, I am seeing Reinforcement Learning. Reinforcement Learning allows me to rethink tasks not only from a stationary view, which I learned previously, but also from a motive view. There exist relations between how events happen around us. Overall, three most important things I learned from the course are:

- Viewing the world from the perspective of Reinforcement Learning algorithms;
- Self-studying ability which allows me to explore more and dive deeper in the fields that I am interested in;
- Critical thinking about current methods, which enable me to draw better academic statements.

Acknowledgements

This report was written for the final assignment for COSC-689 Deep Reinforcement Learning at Georgetown University, Fall 2020. I would especially thank Yanchen Wang, Haotian Xue, Zhiwen Tang for their insights in data quantization, network structure optimization and algorithm structure optimization. Also, I would sincerely thank Professor Grace H. Yang for her guidance in Deep Reinforcement Learning methods.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Fan, J., Wang, Z., Xie, Y., and Yang, Z. A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pp. 486–489. PMLR, 2020.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Lee, M. *9.2 Integrating Planning, Acting, and Learning*. Mark Lee, Jan 2005. URL http://incompleteideas.net/book/first/ebook/node96.html.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance
pdf.

Peng, B., Li, X., Gao, J., Liu, J., Wong, K.-F., and Su, S.-Y. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. *arXiv preprint arXiv:1801.06176*, 2018.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. 2018.

Van Otterlo, M. and Wiering, M. Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pp. 3–42. Springer, 2012.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.