Overview on Reinforcement Learning for Robotics

Anonymous authors

Paper under double-blind review

I. Introduction

Reinforcement Learning(RL) offers robotics tasks a framework and set of tools for the design of sophisticated and hard-to-engineer behaviors.[1]

Reinforcement Learning is an branch of Machine Learning inspired by behaviorist psychology**[wiki-Reinforcemenr Learning]**, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.

The basic idea of Reinforcement Learning is to obtain a policy that extract more reward from the environment by picking actions given a state.

By policy, we mean a decision maker (Agent) that decide on an action based on some parameterized rules given an input observation of environment (State). The policy can be a set of weight that linearly combine the features in a state or different structured Neural Network. The environment in Reinforcement Learning context provide the agent a new state and reward immediately after the agent takes a specific action.

From a more broad view, the Machine Learning method was mainly three folds. The Supervised Learning, Semi-supervised Learning and Unsupervised Learning. The supervised learning network was trained given a dataset including the observation data and the corresponding categorization. The latter was given a dataset that no classification is label to the observation data. For reinforcement Learning, it is more close to supervised learning, while its label is obtained by exploring the environment and get feedback (reward, r) from it. The RL algorithm marks the policy that generates the highest score as the training target and make small change of its parameters (or weights, θ) towards that policy until the policy converge.

In this report, we mainly focus on the methods of Reinforcement Learning methods for robotics.

II. RL Problem and Concepts

2.1 Markov Decision Process (MDP)

Markov Decision Process (MDP)**[18]** is a process of observing and interaction with a system (Environment). We can get system feasures (State, s) and take actions during the MDP. It includes process like this. We start the MDP from a initial observation of the environment, we take actions based on certain strategy (Policy, V) using the observed state, then we get a new state from the environment.

Markov Decision Process have the Markov Property, which basicly ensures that the future state is only related to the current state and current decision, it has nothing to do with the precious history.

A reinforcement learning problem (or optimal control problem) can also be described as a MDP.

At each time step, the process is in a state s, and the decision maker may choose any action a in state s. The process in the next time step will moving into a new state s', and giving the decision maker a corresponding reward $R(s, a, s')$. The reward is a evaluation of the action, it is always related to performance.

The decision maker may have no idea how does the environment really like, but through trial and error, the agent learn to make decisions that extract the most accumulative reward from the environment.

2.2 Bellman Equation

For policy optimizing problem like RL, if the overall reward of the new policy is no less than the previous ones, the policy is called a optimal policy (V_π). The MDP problem needs to find the optimal policy, this is where the Bellman Equation kicks in.

The Bellman Equation can be used to solve optimal control problem including the MDP. It defines the relations between the current state value and the value of the next state.

We define the V_π the state-value function for policy π.

It can be proved that for best policy π^*, we have:

$$V_\pi * (s) = max\{R(s, a) + \gamma \Sigma P(s'|s, a) V_\pi * (s')\}$$

Here $R(s, a)$ is the reward we get by taking action a in state s. γ is a discount factor which models the fact future reward is worth less than immediate reward. $P(s'|s, a)$ is the dynamics of the the system, also the *state-transition probabilities*, it basicly define how much probability it would be to get to state s' by taking action a in state s.

For each state in a MDP, we hope to update the current policy, change its parameter towawrds the direction which makes the next state value closer to V_*, so that the new policy's overall value would be higher than the current policy. For each episode, we do the same update and the policy becomes closer and closer to the optimal policy and finally it will converge to the best one and the MDP get solved.

2.3 On-policy & Off-policy

On-policy and Off-policy are two different kinds of methods adopted when updating the policy.

In on-policy methods we iteratively learn about state values at the same time that we improve our policy. In other words, the updates to our state-action values depend on the policy. In contrast, off-policy methods do not depend on the policy to update the value function. While training, off-policy method choose the max value for that state as gradient target.

If there's ever a policy π referenced in the value update part of the algorithm, then it's an on-policy method.

If a good starting policy is available, on-policy performs better, but may not explore other policies well. If more exploration is necessary, then perhaps off-policy is advisable, but may be slow.

2.4 Model free & Model based

If the agent can make predictions about what the next state and reward will be before it takes each action, then it’s a model-based RL algorithm. If it cannot, it’s a model-free algorithm.

If we know the model of our MDP problem, we can just compute the solution before ever actually executing an action in the environment. This kind of decision making is called planning and we have a bunch of conventional way to get an optimized planning result.

So why do we use RL while conventional control method is already good enough to solve the problem?

Conventional control method(using PID to minimize the error) only apply to linear system. By linear system, we mean that the output of the system is proportional to the input. However, most of the system we are controlling is not linear, so PID method simply do not apply to such conditions. We have to get the model of the system, linearize or decouple to simplify the problem and then use conventional PID, which is quite complex and even unsolvable. Reinforcement Learning allow us to control the system without even know about the model. We learn the policy through trial and error. As it turns out though, we don’t have to learn a model of the environment to find a good policy. That's why we use reinforcement learning in complex systems.

A large group of problem in RL can be solved by model-free method, such as Robotics Control, Game Theory and Economics, etc.

2.5 Deep Reinforcement Learning

In the case of robotics control, the state in most of the time is the position, speed, acceleration, etc. These variables are continuous variables. We need to generalize and pattern match between states.

That's where neural networks comes in. We can use a neural network, instead of a lookup table, as our Q(s,a) function. There are other type of function approximators, even a simple linear model will do the same job.

And as the model becomes more and more complex and challenging, the non-linear of the neural network should increase, which in practice, means the network gets more and more neurons and the structure goes deeper and deeper.

2.6 Action Space: Discrete or Continuous

The Reinforcement Learning tasks are divided into two kinds. Discrete Action Space(DAS) and Continuous Action Space(CAS).

In games like atari and board game, the problem is DAS. When it comes to robotics control in simulation and real world, it is always a CAS problem.

Algorithms can also be separated according to what kind of problem does the one algorithm can solve. For example, the Q-Learning and SARSA are used to handle DAS problem, while algorithms based upon Policy Iteration, like TRPO and DDPG are mostly used in CAS problem.

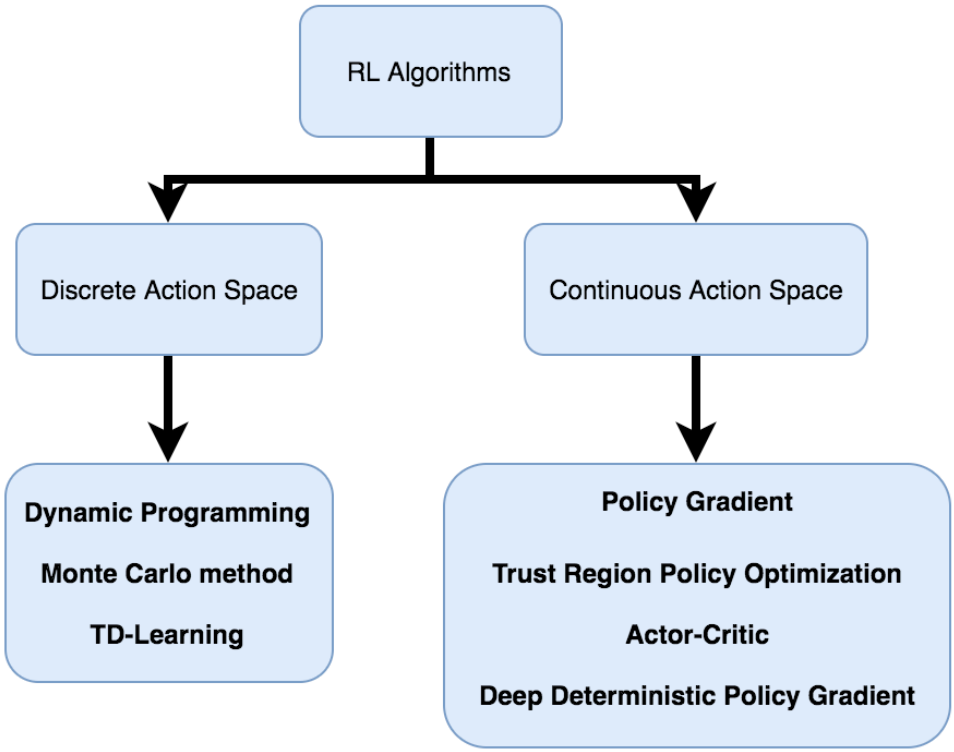

Fig. RL Topology

2.7 Markov and Episodic

Markov is the property of Markov Decision Process (MDP), it guarantees that future states of the process(conditional on both past and present states) depends only upon the present state, not on the

sequence of events that preceded it.

Episodic means that for each trial, the process always come to an end.

For example, Robot Manipulation task is both markov and episodic.[9]

We believe it to be markov because knowledge of previous arm positions and actions does not help us when attempting to pick up an object, only the current arm position and action is of use.

We believe it to be episodic due to the existence of a terminal state (picking up the object) or otherwise a cap on a number of iterations to avoid the agent getting in states far from the terminal state.

2.8 Simulation Environment

Learning on robots is often difficult due to the large number of samples needed for learning algorithms. Simulators are one way to decrease the samples time needed from the robot by incorporating prior knowledge of the dynamics into the learning algorithm**[16]**.

Currently a famous and widely used environment is the Gym. it contains a wide variaty of games and controlling tasks. It also contains an evaluation platform to compare results of different algorithms.

Another very well-known simulation environment is the Mujoco that contains some robot controlling tasks, like InvertedPendulum and Cheetah. Besides, Mujoco is very light weighted. It is also compatible with python which makes it a go-to option to test RL algorithms.

Fig. Mujoco Environment

Some Widely Used Simulation Environment：

- Gym
- Mujoco
- Gazebo (https://github.com/erlerobot/gym-gazebo/)
- Bullet (https://github.com/benelot/bullet-gym/tree/master/pybulletgym)
- Roboschool(https://blog.openai.com/roboschool/)

III. RL Related Works

3.1 Pancake Flipping Manipulation

Fig. Pancake Flipping

In order to make a pan-holding robot to toss a pancake in the air so that it rotates 180 degrees before being caught. P. Kormushev taught the robot the desired joint trajectory through grab and drag. The agent learns to control the stiffness of the arm joint while following the demonstrated path.

Finally, the robot learns that the first part of the task requires a stiff behavior to throw the pancake in the air, while the second part requires the hand to be compliant in order to catch the pancake without having it bounced off the pan.

Learning Procedure

- Demonstrate via kinesthetic teaching
- Inverse Dynamics controller with variable stiffness
- The robot follow the position while learning to use variable stiffness to adjust the behavior and improve the result

Conclusion

- year: 2009
- algorithm: Policy learning by Weighting Exploration with the Returns (PoWER)**[17]**
- input: current position
- output: softness of the joint

3.2 DEEPloco

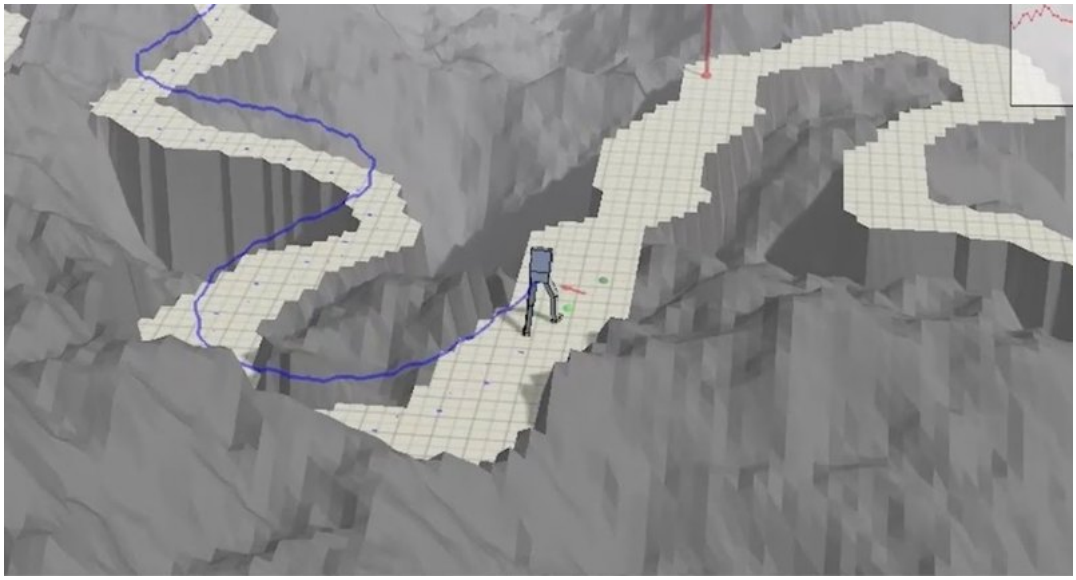

Fig. DEEPloco Simulation

DEEPloco is a robost walking locomotion generator in Mujoco environment. It is able to walk robustly towards target while avoid obstacles and external disturbance.

The whole structure of controller is splited into low-level humanoid gait control and high-level path planning.

Low-level control use Actor-Critic to train a policy that controls multi-DOF of the robot. The Low-level controller is able to substain from some external force impact.

High-level get surrounding map information, the character state and goal as input to output a footstep for the Low-lever controller to follow.

Conclusion

- year: 2017
- algorithm: Actor-Critic
- input: state vector
- output: action vector
- advantage: Robust, fantastic performance in simulation environment

3.3 Autonomous Unmanned Vehicle

Fig. The quadrotor stabilizes from a hand throw

Control Position of a Quadrotor with Reinforcement Learning, Marco Hutter[20], ETH Zurich. From an initial position and speed(even thrown into air), the quadrotor learns a policy to maneuver back to the desired position

Conclusion

- year:2017
- algorithm: Actor-Critic, policy is a 64*64 nodes network with tanh activation
- input:18-dimensional state vector
- output:4-dimensional action vector
- advantage:Robust, success rate is high

3.4 Berkeley Robot Manipulator

Visual Servoing, the multi-robot system asynchronuously learns to open a door based on visual input data.

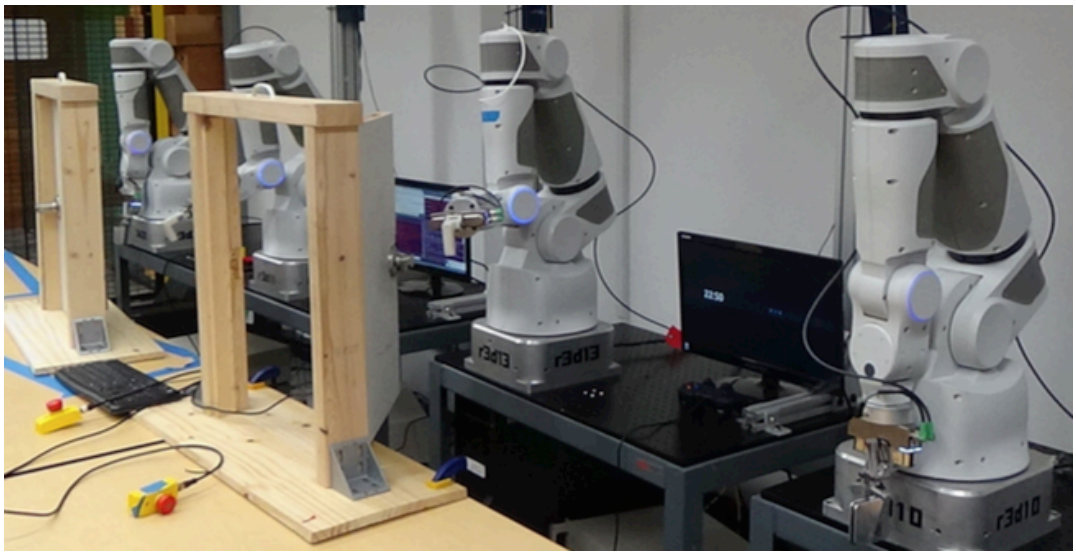

Fig. Beykeley open door manipulator

Visual Servoing, reaching a pose relative to objects in the scene, and often (though not always) rely on manually designed or specified features for feedback control.**[2]**

Conclusion

- year: 2016
- algorithm: DDPG
- input: vision data
- output: joint angle
- advantage: end-to-end, the network output the servo motion directly from video input

3.5 Helicopter Control

The agent learn to maneuvers a real helicopter in motion such as tic-tocs, chaos and auto-rotation.

Conclusion

- year: 2006 NIPS
- algorithm: DDPG
- input: integral of acceleration and angular acceleration, error
- output: controllable variable
- advantage: able to achieve the task that only exceptional human pilots can perform

3.6 Car Drifting

Fig. Drifting Car

This is the work by Aerospace Controls Lab from MIT. The agent learns to drift a car around a moving point on a terrain with variable frictions.

Conclusion

- year: 2015
- algorithm: simulation, policy + transition dynamics -> real car
- input: position, speed, drift center
- output: wheel angel
- advantage:it is hard to build the model of drifting and formulate the problem because the environment disturbance is unknown. Using RL is able to tolerate the noise because the environment noise is taken into account while training.

IV. Well Known RL Algorithms

Here we talk about the most famous RL Algorithms and divide them in different categories. There are basicly 2 large groups of Reinforcement Learning. The Value Iteration and the Policy Iteration.

4.1 Value Iteration

Value Iteration method hold a policy that record values for each state-action pair in a table. The agent update these values and choose actions based on this table.

4.1.1 Dynamic Programming(DP)

In the operations research and control literature, the field where reinforcement learning methods are studied is called dynamic programming. The term “dynamic programming” is due to Bellman (1957a), who showed how these methods could be applied to a wide range of problems. However, our interest in DP is only restricted to its use in solving MDPs.

DP is a Model based solution， it operate by sweeping through the state set, performing a full backup operation on each state. Each backup updates the value of one state based on the values of all possible successor states and their probabilities of occurring using Bellman equations. As is mentioned in part II, the Bellman Equation requires the *state-transition probabilities* of the system, which makes DP a model based strategy.

DP holds another very important feature called *bootstrapping*. They update estimates on the basis of other estimates. That is, DP updates estimates of the values of states based on estimates of the values of successor states.

4.1.2 Monte Carlo method(MC)

MC method is a random sampling algorithm which can be used in multiple fields. The algorithm averages the the results over the samples. With the sample number increases, the estimate approaches the real probability, which, in RL, means the evaluation of the model converge to the existing system.

The MC assume experiences be divided into episodes, and that all episodes eventually terminate no matter which action is selected.

For each eposode, Monte Carlo method sample returns for each state–action pair. After each sampling episode is done, it evaluate the returns by averaging all the rewards in the sample set. As more returns are observed, the average should converge to the expected value.

MC is model-free, and can be used with simulation or sample models. However, it only update the value function or policy at the end of each episodes, which makes training inefficient.

4.1.3 Temporal-Difference Learning(TD)

TD learning is the central and novel idea in the area of RL.

TD-Learning is a combination of Monte Carlo and Dynamic Programming(DP).**[3]** It learns from the environment by interaction experience without the knowledge of model's dynamics, like MC. Because the TD-Learning bases its update in part on an existing estimate, which is called the *bootstrapping* method, like DP.

TD-Learning have an advantage over DP methods in that they do not require a model of the environment, of its reward and next-state probability distributions.

TD method works like this. It stores a table(Q-table) about of the current knowledge about the environment. It is an action-value pair for every state. After each action, based on the reward and the new state value(vary for different TD-method), it update the value for the action taken in the old state. After sufficient trials, the Q-table converges. Its values indicate the estimated value for actions in states, which allow the agent to choose the action given any state in the table.

Q-Learning

Q-Learning was posted 28 years ago. This is the most conventional and commonly used method for RL. Q-Learning is an Off-Policy algorithm for Temporal Difference learning, so it is model-free and it bootstraps.

It can be proven that given sufficient training under any episilon-soft policy, the algorithm converges to a close approximation of the best policy. Q-Learning learns the optimal policy even when actions are selected according to a more exploratory or even random policy.

The Q-Learning explore the environment with a method called $\epsilon - greedy$. It makes the agent gather information from patially random moves. It choose actions based on its policy as well as a decaying noise factor ϵ to encourage exploration. As the agent learns more about the environment, the episilon gradually minish to zero, allowing the agent to adopt action completely according to the converged optimal policy.

As an off-policy algorithm, it update its action-value pair(Q-table) using the maximum of the next state value and reward.

Q Learning is a designed for Discrete Action Space(DAS). There is, however, solusions to allow Q-Learning tackle with the Continuous Action Space. Normalized Adantage Functions (NAF)**[7]** was proposed in 2016, it allows us to apply Q-learning with experience replay to continuous tasks, and substantially improves performance on a set of simulated robotic control tasks.

Deep Q-Network(DQN)

DQN is the algorithm posted by DeepMind in 2015. It showed superhuman ability in playing atari games. It offers end-to-end training process, which takes pixel image as input and returns the corresponding action should be taken by the game agent. It works so good in a large variety of games with the same network struct that brings the popularity of deep neural network into RL area.

DQN is based on the Q-Learning. The network is like a function aproximation to generalize the input data into the agent understandable state information. The network is used to preprocess the pixel image to fix the input into the meaningful data for the RL. The state together with the reward(in this case, game scores), is then fed into the Q-Learning structure to evaluate and update the policy. In case of end-to-end training, action-value pair, which also known as Q function, is also a Neural Network.

Besides, DQN used Experience Replay to avoid the instability of the nonlinear function approximator. This clears the bias of the state in a continuous episode by storing steps in groups and shuffling the steps before feeding the data.

One down side of non-linear policy is converge, there is no guarantee that the policy converge to a global optimum. Many tricks are applied to such algorithms to encourage converge, like trust-rigion method, double Learning, etc.

Double DQN(DDQN)

Due to the max in the formula for setting targets, the network suffers from maximization bias, possibly leading to overestimation of the Q function's value and poor performance. Double learning can help.**[4]**

To demonstrate this problem, let's imagine a following situation. For one particular state there is a set of actions, all of which have the same true Q value. But the estimate is inherently noisy and differs from the true value. Because of the max in the formula, the action with the highest positive error is selected and this value is subsequently propagated further to other states. This leads to positive bias – value overestimation. This severe impact on stability of our learning algorithm.

In Double DQN, two Q functions, Q_1 and Q_2, are independently learned. One function is then used to determine the maximizing action and second to estimate its value.

The Deep Reinforcement Learning with Double Q-learning paper reports that although Double DQN does not always improve performance, it substantially benefits the stability of learning. This improved stability directly translates to ability to learn much complicated tasks. When testing DDQN on 49 Atari games, it achieved about twice the average score of DQN with the same hyperparameters. With tuned hyperparameters, DDQN achieved almost four time the average score of DQN.

SARSA

SARSA and Q-Learning are both TD methods. They share the same features like model-free, bootstraping.

The SARSA differs by using an on-policy learning. The agent updates action-value pairs relative to the policy it follows, while off-policy Q-Learning does it relative to the greedy policy(converge to the max estimated value of the next state). This means, the action-value function converge toward the **new state value** for SARSA while it converge to the **max new state value** for Q-learning.

Although Q-learning actually learns the values of the optimal policy, its online performance is worse than that of Sarsa. However, both algorithms converge to the optimal policy.

4.2 Policy Iteration

The Policy Iteration is a main stream of Reinforcement Learning. Policy Gradient relies on optimizing parameterized policy with respect to long-term cumulative reward through gradient descent.

4.2.1 Vanilla Policy Gradient

Vanilla Policy Gradient is also referred as regular Policy Gradient(PG). Policy Gradient algorithms optimize a policy by computing noisy estimates of the gradient of the expected reward of the policy and then updating the policy in the gradient direction.

For RL problems with continuous action spaces like robot control problem, vanilla-PG is all but useless. It can run into many problems.
For example, the credit assignment problem can happen in robot control, getting one reward signal at the end of a long episode of interaction with the environment makes it difficult to ascertain exactly which actions are good ones.

4.2.2 Natural Policy Gradient(NPG)

The Natural Policy Gradient is an extension of Vanilla Policy Gradient. It makes the algorithm much more viable.

The core idea of NPG is to do just a small change $\Delta\theta$ to the policy π_θ while improving the policy. However, the meaning of *small* is ambiguous. In order to define the *small*, Natural Policy Gradient measure the closeness between the current policy and the updated policy.

In statistics, a variety of distance measures for the closeness of two distributions (e.g., the Kullback-Leibler divergence $d_K L(p_\theta, p_\theta + \Delta\theta)$, the Hellinger distance $d_H D$)

4.2.3 Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization（TRPO)**[13]** is a transformation of Policy Gradient.

The core idea of TRPO is to confine the policy update using the KL diverge factor. It is a evaluated distance between two distributions. TRPO tries to make a small change to the policy each time, if KL diverge between the updated policy and the original one is too large, TRPO will discard the update in case of diverge.

TRPO shows a better performance compared with CEM and NPG. This has become a go-to algorithm for most robotics task like Mujoco.

4.3 Policy and Value Combined Algorithm

4.3.1 Actor-Critic(AC)

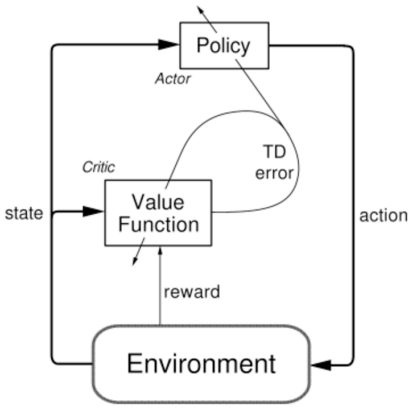

Fig. AC Topology

The Actor-Critic is the most widely used algorithm in RL for robotics. The idea of actor-critic method is to separate policy and value function into entities called actor and critic respectively. The actor stands for Policy Gradient method, it contains a policy for action choosing and policy updating. The critic means Function Approximation which contains a value function to estimate the action based on the new environment and reward.

For each step, the actor choose an action. The critic gives a value for actor according to the new state to judge whether the action is good or not. The actor update its policy based on the policy gradient and adjust the gradient step size according to the value output of the critic, while the critic update its value function based on the new state and reward.

Normally, AC is on-policy. It update the value function using the new state-action pair. There is also off-policy actor-critic**[15]** which shows an improvement in performance in Continuous Action Space tasks like *Pendulum* and *Mountain Car*. AC method bootstraps by updating its actor and critic after every step.

Continuous Action Space problems are often tackled using actor-critic methods.**(Konda & Tsitsiklis, 1999; Hafner & Riedmiller, 2011; Silver et al.,2014; Lillicrap et al., 2016)**

Another benefit for AC method is that convergence is guaranteed even for non-linear (e.g., neural network) approximations of the value function.

4.3.2 Deep Deterministic Policy Gradient(DDPG）

DeepMind proposed the DDPG which referred from the Actor-Critic structure. Deep means the algorithm is combined with Neural Network. Deterministic means it output a specific action based on the probabilistic over action space. DDPG works well on Continuous Action Space(CAS).

DDPG used a inspiring structure in DQN, by adopting two sets of network, the evaluate network and the target network. This method is able to avoid bias, thus increase the ability to converge. The evaluate network set is use to generate action and value, the target network is where the gradient applied onto. Then evaluation network approach a step towards the target network.

DDPG requires only a straightforward actor-critic architecture and learning algorithm with very few “moving parts”, making it easy to implement and scale to more difficult problems and larger networks. DDPG can sometimes find policies that exceed the performance of the planner, in some cases even when learning from pixels

4.3.3 Asynchronous Actor-Critic Agents(A3C)

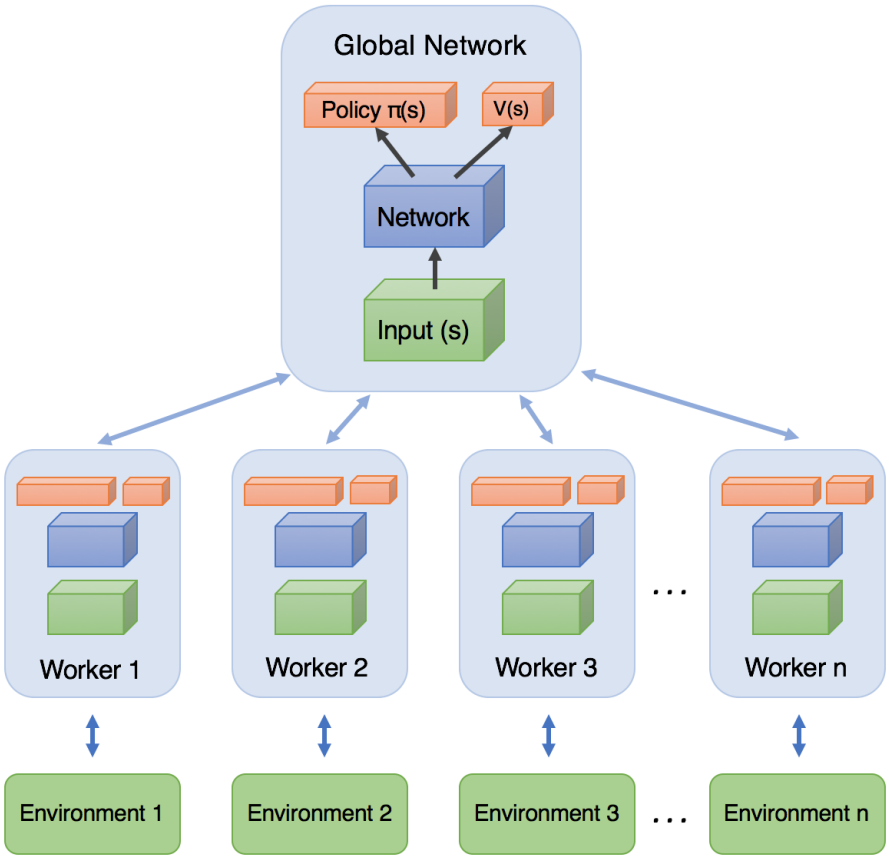

Fig. A3C Structure

The A3C algorithm was released by Google’s DeepMind group**[6]** earlier this year, and it made a splash by essentially obsoleting DQN. It was faster, simpler, more robust, and able to achieve much better scores on the standard battery of Deep RL tasks.

Compared to DQN[14], the model simultaneously learns a policy and a value function for predicting expected future rewards, and can be trained with CPUs, using multiple threads. A3C has been shown to converge faster than DQN, which makes it advantageous for research experimentation.

The A3C use the same structure as Actor-Critic(AC), it extends AC by using multi-thread. In each thread, it deploys a pair of AC to interact with the environment, after gather several steps or episodes, it calculate the gradient for value function and policy, then upload to a globle AC to apply gradient descend. After pushing the feed data, the local AC pulls the updated weights back to keep on exploring the envitonment. Each

thread goes through the process asynchronously on a multi-core cpu so as to improve the algorithm efficiency.

4.4 Derivative Free Optimization Algorithm

4.4.1Cross-Entropy method(CEM)

Cross-Entropy method randomly generate many policies based on a specified noise and test on the environment. For each episode, it picks the policies that gives best performance and average the parameters in these policies.

CEM show extraodinarily fantastic performance that was used on multiple tasks including simple robotics model control.

V. Discussion

As is delineated above, the RL task always try to find a way to adopt a best policy by exploring in the environment. The environment, in most of the time, is unknown or partially known. The trade-off between exploitation and exploration can be observed in most algorithms from the basic Q-Learning to the state-of-art A3C. How we explore in the action space determines the training time and efficiency, the convergence and whether the algorithm find a optimal policy.

There are other interesting topics that waits be explored in RL. For example, use more complicated reward and policy to increase success in agent, and use penalty to restrict some bad behavior. The robotics task tends to have high dimension with continuous action space(CAS), which makes it a easily stuck in local optimum. How to encourage explore in such a high dimensional space meanwhile guarantee the policy will converge in limited time is a hard problem. Introducing penalty or complex reward is a straghtforward way to go.

In addition, mechanisms needs to be developed so that policies doesn't have to be learnt from scratch, but can be inherited**[8]**, which make the AI generally more and more smart. The same network structure of policy may be optimized with conbined efforts of different algorithms. And combine research in neuroscience, looking for a general purpose neural structure which is able to be optimized with fine tuned algorithm, that learns large-scale variaty of tasks with little overfitting.

In the robotics area, the state is, in fact, not only the data from the robot itself, but also part of the environment. How we fuse these data into our algorithm to make it a even more robust AI is still a long way to go.

Besides, as a system, how the robot handles similar but never explored states is an interesting problem. The result relies upon the generalization of the environment that the robot has explored and what the policy generally got through exploring. We hope the robot can handle more tasks but on the promise of reliability.

Will the robot perform much better with the help of optimization algorithm or under the instruction of human? It is reasonable to suppose, that after observe the expert demonstrating the task, the agent will start from a policy near the global optimum and converge efficiently without dropping into the local optima. Behind this idea, *Apprenticeship Learning* and *Guided Policy Search* is a good inspiration for us to teach robot how to perform wisely, just like we teach children to ride bike or swim.

Traditional way of control is hard to handle complex control problem so we tend to simplify the model by reducing dimension and introducing coupling in dimensions. Many RL algorithm has already adopted the method by just tuning the parameters in lower dimension. This model-based simplification will significantly cut down the time and computational expense. But it is heavily based on the validity of the simplified model.

VI. Reference

[1] Kober, Jens, et al. *Reinforcement Learning in Robotics: A Survey*. The International Journal of Robotics Research.

[2] S. Levine, et al. Learning Hand-Eye Coordination for Robotic Grasping with Large-Scale Data Collection. *Springer Proceedings in Advanced Robotics 2016 International Symposium on Experimental Robotics*, 2017

[3] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: an Introduction*. The MIT Press, 2012.

[4] Hasselt, et al. *Deep Reinforcement Learning with Double Q-learning*, arXiv:1509.06461, 2015

[5] Y. P. Pane, *Reinforcement Learning for Tracking Control in Robotics*, Master of Science Thesis, Delft Center for Systems and Control, Delft University of Technology

[6] V. Mnih, et al., *Asynchronous methods for deep reinforcement learning*. In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, pages 1928–1937, 2016.

[7] S. Gu, et al., *Continuous Deep Q-Learning with Model-based Acceleration*. arXiv:1603.00748 [cs.LG], 2016

[8] S. Amarjyoti , *Deep reinforcement learning for robotic manipulation-the state of the art*, arXiv:1701.08878v1 [cs.RO], 31 Jan 2017

[9] S James, *3D Simulated Robot Manipulation Using Deep Reinforcement Learning*, Individual Project MEng, Imperial College London, 2016

[10] Todorov, et al., *MuJoCo: A physics engine for model-based control*, Intelligent Robots and Systems (IROS), 20122012

[11] I. Zamora, et al., *Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS*

and Gazebo, arXiv:1608.05742v2, 2017

[12] J. Schulman, et al., *Proximal Policy Optimization Algorithms*, arXiv:1707.06347v2 [cs.LG], Aug 2017

[13] Schulman et al., *Trust Region Policy Optimization*, ICML, 2015

[14] V. Mnih, et al. *Human-level control through deep reinforcement learning*, Nature 518, 529–533, February 2015

[15] T. Degris, et al. *off-policy actor critic*, arXiv:1205.4839v5 [cs.LG], 2013

[16] M. Cutler and J. P. How, *Efficient Reinforcement Learning for Robots using Informative Simulated Priors*, ICRA 2015

[17] J. Kober, J. R. Peters, *Policy Search for Motor Primitives in Robotics*, In Proc *Advances in Neural Information Processing Systems 21 (NIPS)*, 2008

[18] P. Abbeel, A. Coates, M. Quigley, A. Y. Ng. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. In *Advances in Neural Information Processing Systems 19*. MIT Press.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, International Conference on Learning Representations (ICLR), 2016.

[20] J. Hwangbo, I. Sa, R. Siegwart and M. Hutter, *Control of a Quadrotor with Reinforcement Learning*, IEEE Robotics and Automation Letters, 2017.