Accelerate Reinforcement Learning with Protective Boundaries

Zongqiang Pang,Liping Bai *Member, IEEE,*

***Abstract*—Protective Boundary is a common concept for atheletic training. Gymnast, divers, figure skaters would wear harness or a coach would be present to prevent themselves from falling while training for new skills. The same is true for children, as parents caringly provide a helping hand instead of just let them try things out themselves. Reinforcement Learning is an extention of approximate dynamic programming, witch reasons backwards and a lookup table is built in the mean time. With the newly added capacity brought about by deep learning, deep reinforcement learning is proven to be a formitable force when it comes to control. However, there is a clear difference between reinforcement learning and human learning. While out joints can have all kinds of combinatorial movement, we only use a few of those even in the most rigirous circumstances. Also, the way we learn to syncronize out joints is not a exhaustive trial and error process. Immitation learning seek to address this issue by provide a model trajectory, and meta-learning see to address this by provide agents with a sense of physics. In this paper, we mimic the Protective boundary method we see in atheletic training to provide agents additional clues about the environment. There are two advantages in this proposed scheme. First, the Protective boundary prevent premature termination of an episode, which is of particular importance to environment where failure is costly. Second, the Protective boundary method accelerate data collection on states/action pair that matters. We implemented the proposed prohibitibe boundary scheme on various OpenAI Gym environments, and for all the experiments carried out, Protective boundary has shown to accelerate training. All the code and data can be found at: Code Deposite**

***Index Terms*—Reinforcement Learning, Assisted Reinforcement Leraning, Reward Engineering, Safe Exploration**

I. INTRODUCTION

REINFORCEMENT learning(RL) is the process of methotically extracting information from experiments to gradually bound policy distributions, maximizing the expected reward along a path. RL can be seen as approximate dynamic programming extended to unknow system dynamics, powered with statistical methods and neural networks.When the first batch of reasonable RL results were introduced, they were met with coldness by people in control community. The preface of Bertseka's book Neuro Dynamic Programming provides a good sample on how reinforcement learning is perceived by the control community back in 1996: "...These methods (Reinforcement Learning) were aiming to provide effective suboptimal solutions to complex problems of planning and sequential decision making under uncertainty, that for a long time were thought to be intractable. Our first impression was that the new methods were ambitious, overly optimistic, and lacked firm foundation....Three years later, after a lot of study, analysis, and experimentation, we believe that our initial impressions were largely correct." [1]

Nanjing Unversity of Posts and Telecommunications, College of Automation & College of Artificial Intelligence, Nanjing, Jiangsu,210000 China email:zqpang@njupt.edu.cn

Things are dramatically different today. RL community routinely generates results that seem untainable to traditional control methods. What has changed is not the theoretical foundation of RL, which is as leaky as it was in 1996, but its computational infrastructure. Before 2010, the predominent tools for function approximation are kernel methods, where feature spaces are used to transform nonlinear functions into linear space such that regression can be performed. Today, the default function approximator are neural networks. Another major change in computation is that of GPU based acceleration. Before the advent of CUDA, GPU programming requires PhD in computer graphics. Today, anyone who is proficient in C/C++ can programme GPU to parallelize their compuation. Software packages such as PyTorch and Tensorflow made this even easier.

However, there are clearly aspects of RL that is unreasonable yet papered over by fast computing. For instance, in order to train a humanoid to stand up, the agent need to explore all the combitions of joints movements. Reward signals would provide cues the agent as of which joints combitions are most likely to result in high scores. While children do explore plenty when they learn stand up and walk, I have never seen anyone who is malleable and flexible enough to explore all the combinatorial possibilities of joints positions, not even baby yoga master if there is one. Human must learn more efficiently than does RL agents. That is what we hope to propose in this paper: a method to accelerate reinforcement learning based on our observation of human learning process, specifically, the protective boundaries utilized during atheletic training. As is shown by our experiment results, our methods does accelerate reinforcement learning in all the experimental environments. Further research is required to analyze where to set the protective boundary and with what parameter.

In section I we introduce reinforcement learning to control resaerchers. In section II we introduce our promosed protective boundary scheme for acclerated RL agent training. In section III, we detail the results of all our experiments. While the experiments are conducted in OpenAI Gym simulated environment, we designed things in a way such that this method can be implemented in physical world as well. In the final section, we conclude on what we have learnt from our experiments and lay out directions for further research.

II. Reinforcement Learning For Control Researchers

A. A Note on Notations

The first barrier stands between control researchers and RL is the notation system. Control communities use the notation system introduced by Lev Pontryagin. State is denoted by $\mathcal{X}$, conventionally a letter representing the unknown. Action is denoted by $\mathcal{U}$, the first letter of Russian for action. The dynamics and stocasticity is captured by physical model constraints $x_{t+1} = f(x_t, u_t, e_t)$ where e denotes the noise of a system. The objective is usually to minimize the cost funtion $\mathcal{J}(.)$. Reinforcement Learning communities use the notation system introduced by Richard Bellman who studied dynamic programming. State is denoted $\mathcal{S}$, Action is denoted $\mathcal{A}$. The dynamics and stochaticity is captured via transition matrix $\mathcal{P}$ of a Markov Decision Process.The objective of RL is the maximize the reward function $\mathcal{R}(.)$. We would like to present this note to control researchers at the begining of our paper as they might find the reinforcement learning literature notation rather confusing.

B. From Approximate Dynamic Programming to RL

Reinforcement Learning is not a new subject, control researchers probably know it by the name Approximate Dynamic Programming. Dynamic Programming is simply reasoning backwards. Take trajectory optimization for instance, if we know the desired final state and system constraints then we can simply compute backwards on the penultimate step, and the computing chain flows backwards from there. The prerequisites for a successful dynamic programming based controller are two-pronged: first, the observation space and action space are reasonably small; second, the system dynamics function is known. Neither is easily met in real life scenario.

Approximate Dynamic Programming comes in when the action space and observation space are large. Instead of computing for exact one to one relationship between observation and action choice, we use a function approximator to capture all the control information. Yet even for approximate dynamic programming, the reasoning backwards requires knowing system dynamics function. RL finally provides the bridge to extend approximate dynamic program to circumstances when the system model is unknow.

In order to introduce RL to control researchers, we framed RL in the language of optimization. Broadly speaking, there are three revenues where neural network based approximation finds its way into optimization as shown in 1.

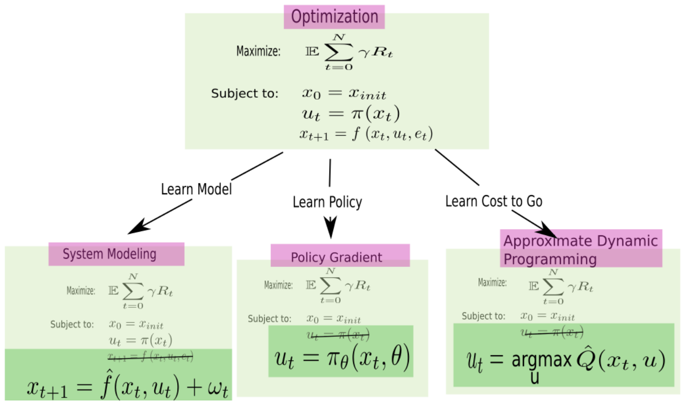
Optimization
Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
$u_t = \pi(x_t)$
$x_{t+1} = f(x_t, u_t, e_t)$
Learn Model
Learn Policy
Learn Cost to Go
System Modeling
Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
$u_t = \pi(x_t)$
~~$x_{t+1} = f(x_t, u_t, e_t)$~~
$x_{t+1} = \hat{f}(x_t, u_t) + \omega_t$
Policy Gradient
Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
~~$u_t = \pi(x_t)$~~
$u_t = \pi_\theta(x_t, \theta)$
Approximate Dynamic Programming
Maximize: $\mathbb{E}\sum_{t=0}^{N}\gamma R_t$
Subject to: $x_0 = x_{init}$
~~$u_t = \pi(x_t)$~~
$u_t = \underset{u}{\text{argmax}}\, \hat{Q}(x_t, u)$

Fig. 1: From Optimization to Learning

Neural Networks can be used to approximate system dynamics function, which is topic for System Identification researchers. While it is true that model based reinforcement learning and offline learning has gain tractions in recent years, in this paper, we won't focus on this line of research. Neural Netoworks can also be used to approximate control policy and the cost-to-go in approximate dynamic programming. A bootstrap structure is the most critical step in extending approximate dynamic programming to cases where the system function is unknow. Cost-to-go, which is computed based on system dynamics is used in order to choose an action. In RL, the agent just guess the cost-to-go and make choices according to its guesses. As the agent accumunate data based on our guesses, it refines the process it utilizes to come up with estimations. To put this process simply: when an agent learns to control an unknown system, it made assumptions on the steps ahead and base its action on those assumptions. After observing experiments data, the agent calibrate its ability to guess.

C. Combine Policy Approximation and Cost-to-go Approximation

1) Advantage Actor Critic(A2C):

2) Trust Region Actor Critic:

3) PPO:

D. Entropy Regularization

III. Protective Boundary

A. Protective Boundaries in Atheletic Training

Have you ever wondered how do world class athelets such as gymnists, figure skaters, divers accomplish feats that are seemingly impossible to us ordinary human? While bruises, torn ligaments, broken bones, even concausion are part of being an athelete, those injuries are by no means trivial and should be minimized at all cost. Should athletes train the same way as would reinforcement learning agents, they will be dead or so severely injured before they can pick up any skill. Protective Boundaries, implemented with either human coach or training harnesses is integral part of atheletic skill acquisition process, as examplified by the following pictures. Detailed examples of how atheletes utilizes Protective boundary can be found here.

(a) Human Protective Boundary (b) Harness Protective Boundary

Fig. 2: Protective Boundary in Atheletic Training

B. *Implement Protective Boundary in RL Setup*

Do they train as it would we train the agents in a reinforcement learning setup? The answer is clearly now, since for those athlets, failure means catestrophic consequences such as broken bones, torn ligaments, concausion etc. While being atheletes means accepting the possibility of permenent harm from training, the entire sport industry try hard to minimize the downside of training.

C. *Experiments on OpenAI Gym*

1) CartPole: fakdflaksdjflakdjflajsdlfjlakdfjlkajdsflkjlasdjf adfjakjdfhkjahdfkjhasdfkjh

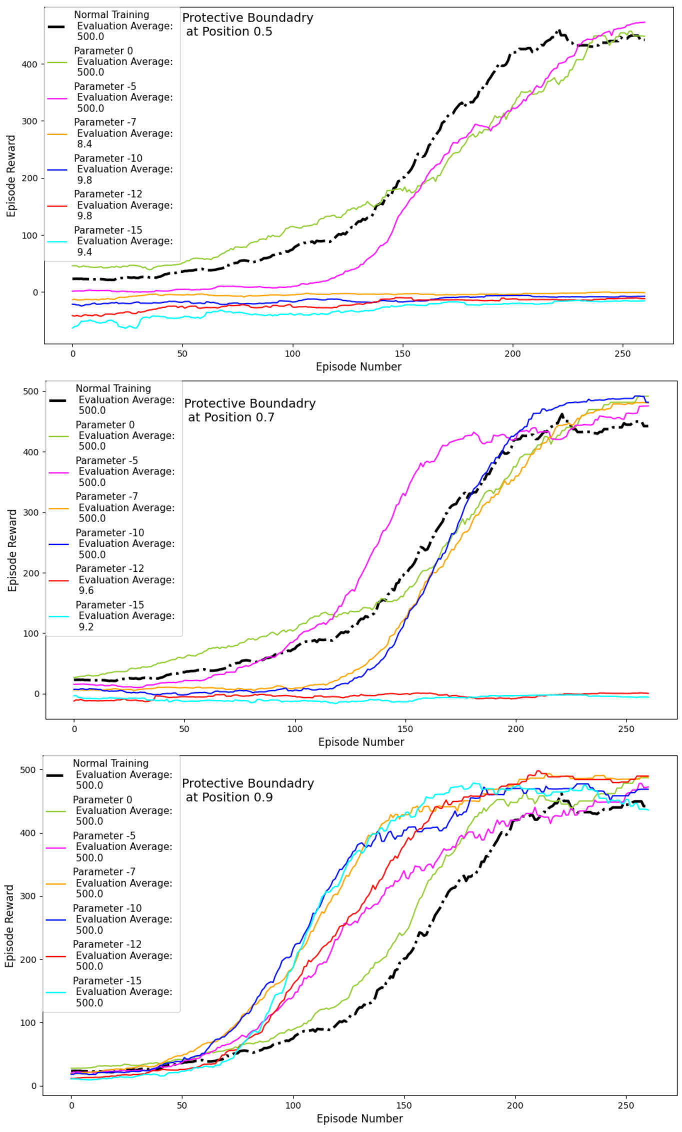

Fig. 3: CartPole Experiments

2) Inverted Pendulum: fakdflaksdjflakdjflajsdlfjlakdfjlkajdsflkjlasdjf adfjakjdfhkjahdfkjhasdfkjh

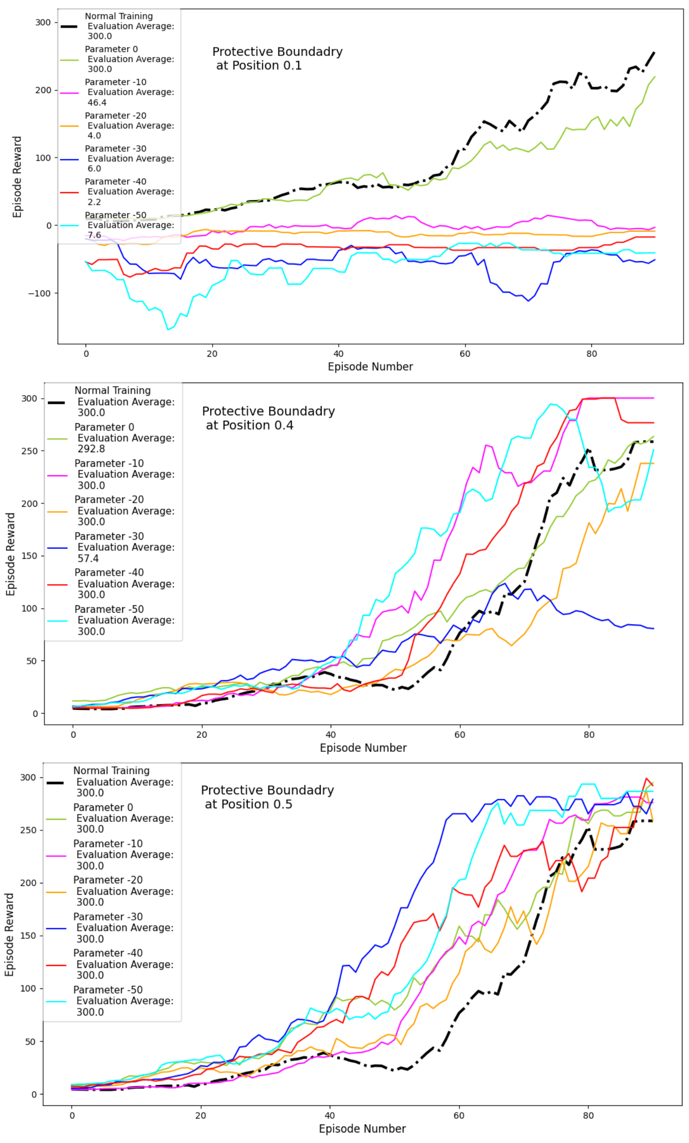

Fig. 4: Inverted Pendulum Experiments

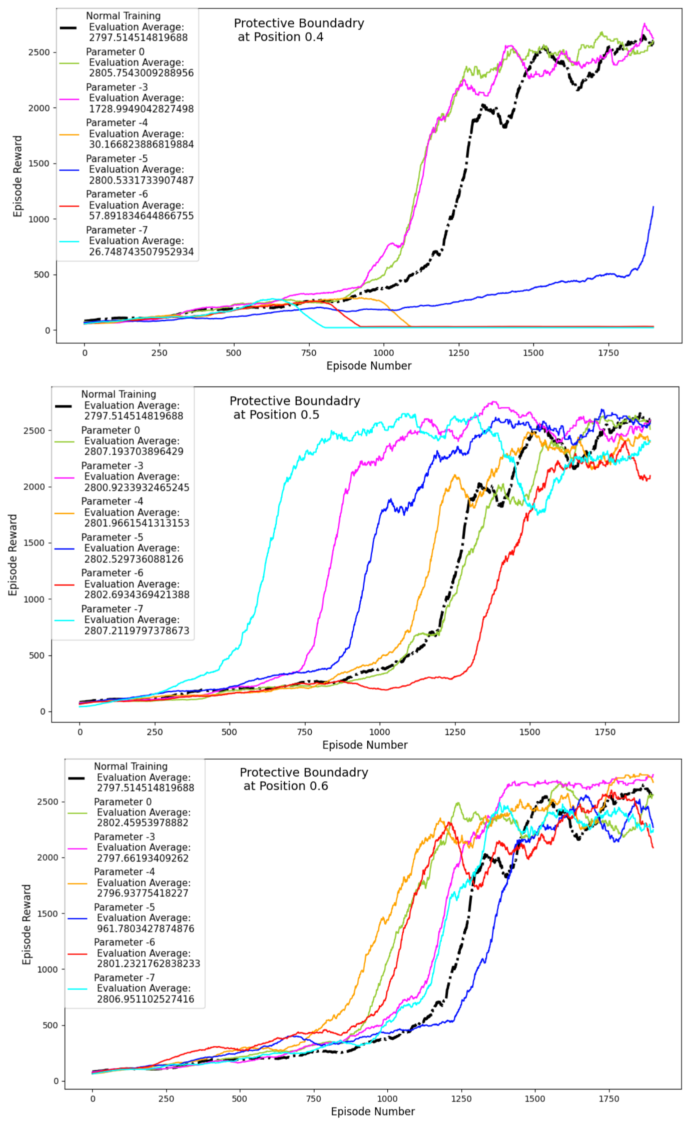

Fig. 5: Inverted Double Pendulum Experiments

3) Inverted Double Pendulum: fakdflaksdjflakdjflajsdlfjlakdfjlkajdsflkjlasdjf adfjakjdfhkjahdfkjhasdfkjh

4) Walker2d: fakdflaksdjflakdjflajsdlfjlakdfjlkajdsflkjlasdjf adfjakjdfhkjahdfkjhasdfkjh

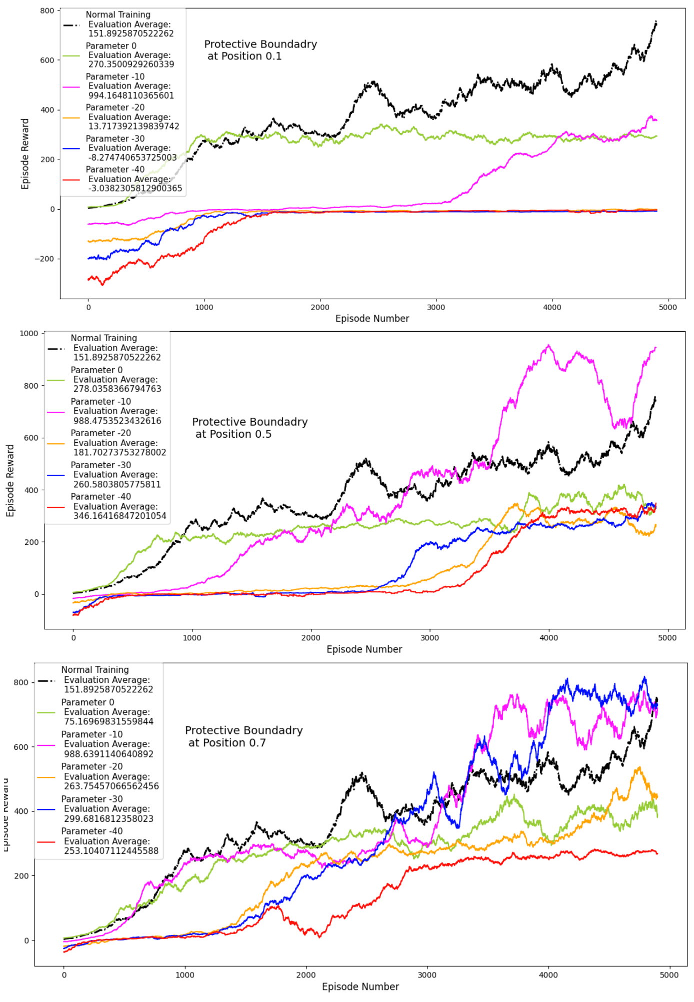

Fig. 6: Walker2D

5) Hopper: fakdflaksdjflakdjflajsdlfjlakdfjlkajdsflkjlasdjf adfjakjdfhkjahdfkjhasdfkjh

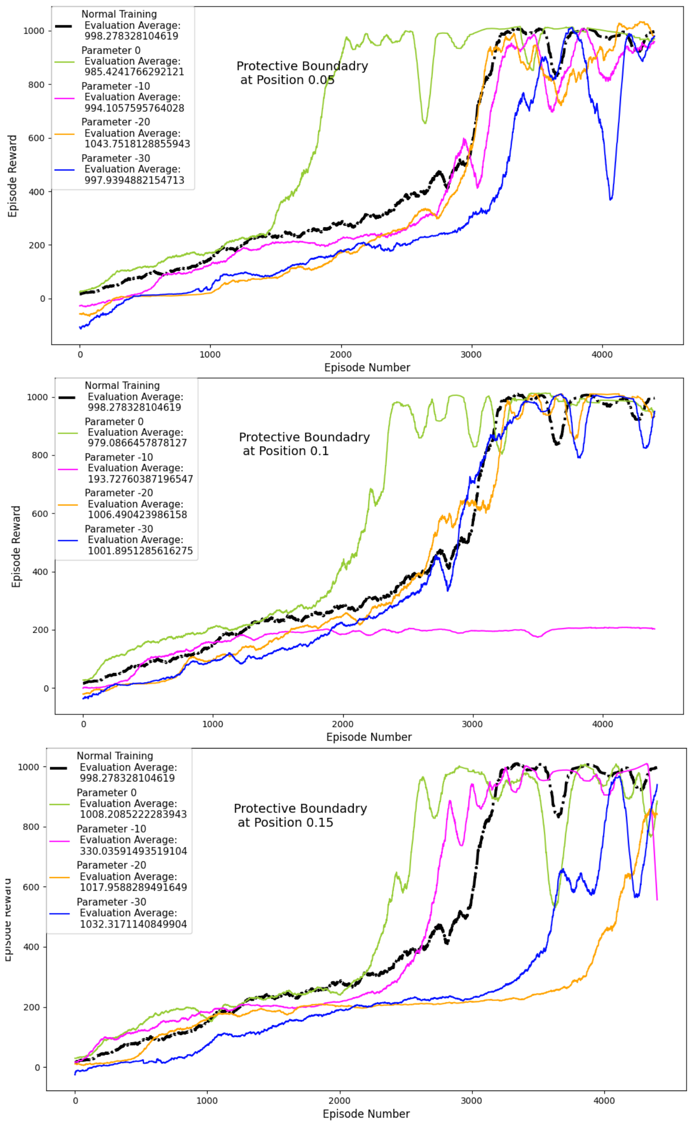

Fig. 7: Hopper

6) :

IV. Conclution

References

[1] D. Bertsekas and J. Tsitsiklis, "Neuro-dynamic programming," in *Encyclopedia of Machine Learning*, 1996.
[2] S. Levine, "Reinforcement learning and control as probabilistic inference: Tutorial and review," *ArXiv*, vol. abs/1805.00909, 2018.
[3] R. Sutton and A. Barto, "Introduction to reinforcement learning," 1998.
[4] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951.
[5] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.