Study on Constraint Reinforcement Learning

Rui Liu

April 28, 2022

Abstract

This report aims to propose a dual network system to provide a numerical barrier to deep q learning. With carefully constructed judge module, we are able to build the barrier network. Further, we introduce trigram module to improve the performance. Finally, using the values of the barriers, we can constrain the deep q learning to the safe states and actions.

1 Introduction

Reinforcement Learning algorithms has been proved useful in many questions, especially after the Deep Q Network (DQN) [1] was proposed in 2015, which combined reinforcement learning with deep neural network. One advantage of this combination is that with the help of neural network, it is possible to estimate the action values on a continuous observation space, rather than being restricted on manually grid spaces. On the other hand, with barrier-based decomposition of the action value [2], we are able to block unsafe actions by adding $-\infty$ to the total action value. Therefore, an intuitive way of restricting the agent on safe actions is to train a separate barrier network, which returns $-\infty$ for unsafe actions and 0 for safe actions, so that the unsafe actions can by no means be chosen by the agent.

However, in practice, adding $-\infty$ directly to the loss function is not feasible, since the ∞ makes it almost impossible to calculate the gradients to update the weights. A back-off solution would be to add a negative number, which is small enough, to penalize the unsafe actions. However, this method is not generalized enough and the potential reward may be too high such that the penalty term is negligible.

In this work, we present a two-step method to solve this problem. We first train a barrier network to distinguish safe actions from unsafe actions and then, build a q network with restriction to calculate the long-term reward of each action.

2 Background

In the setting where an agent interacts with the environment, at each timestamp t, based on the current state $s_t = s \in \mathcal{S}$, the agent picks up an action $a_t = a \in \mathcal{A}$ with policy π and moves to

a new state s_{t+1}, while receiving reward $r_t \in \mathcal{R} \subset \mathbb{R}$. In addition, when an inspector observes a damage at state s_{t+1}, $d_t = 1$ will be returned and the process will be shut down immediately. Otherwise, the process continues with $d_t = 0$.

With discount factor $\gamma \in [0, 1]$, the action-value function is defined as

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \middle| s_t = s, a_t = a\right]. \tag{1}$$

If we define the hard barrier index function [2] I as

$$I(d_t) = \log(1 - d_t) = \begin{cases} 0 & d_t = 0 \\ -\infty & d_t = 1, \end{cases} \tag{2}$$

we can then define the barrier function [2]

$$B^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{t=i}^{\infty} \log(1 - D_{t+i}) \middle| s_t = s, a_t = a\right], \tag{3}$$

which is similar with the definition of the action-value function. We omit the discount factor γ, since multiplying γ cannot change the value of $-\infty$ or 0.

The barrier function suggests that a state-action pair (s, a) is safe, if there exists at least one path such that no damage is detected through that path. On the contrary, all processes starting from (s, a) are shut down at sometime, if (s, a) is unsafe with $B^*(s, a) = -\infty$

Inspired by deep q learning, we approximate the barrier function $B^*(s, a)$ with a barrier network. We notice that the value of $B^*(s, a)$ is either $-\infty$ or 0 and hence, the value of $e^{B^*(s,a)}$ is either 0 or 1, which leads us to a binary classification problem. Therefore, we update the barrier network with loss function

$$L\left(\theta^B\right) = -\mathbb{E}\left[yB\left(s, a \middle| \theta^B\right) + (1 - y)\log\left(1 - e^{B\left(s,a\middle|\theta^B\right)}\right)\right] \tag{4}$$

where

$$y = (1 - d)\max_{a'} e^{B\left(s',a'\middle|\theta^B\right)}. \tag{5}$$

With appropriate barrier, we can then train the q network with loss function

$$L\left(\theta^Q\right) = \mathbb{E}\left[\left(r + \gamma \max_{B(s',a')=0} Q\left(s', a' \middle| \theta^Q\right) - Q\left(s, a \middle| \theta^Q\right)\right)^2 \middle| B(s, a) = 0\right]. \tag{6}$$

3 Algorithm

Our method consists of two part: in the first part, we will train a barrier network, which takes state s as input and return $B^*(s, a)$ for all action a. Then, we will build the DQN, with the barrier network blocking all unsafe actions.

3.1 Barrier Network

In practice, a network that outputs either $-\infty$ or 0 is hard to train. Yet, we can first train a network that outputs either 0 or 1 and then, take the logarithm. Indeed, the barrier network consists of 3 parts. A judge module, which can be regarded as a soft binary classifier, returns smaller values (close to 0), if the judge thinks the state-action pairs are unsafe, and larger values (close to 1), if the judge believes that the state-action pairs are safe. Then, the outputs are rounded to either 0 or 1. Finally, we take the logarithm for the final value.

Algorithm 1 Judge Module

Randomly initialize judge module $J\left(s, a\middle|\theta^J\right)$ weight θ^J
Initialize target module J' with weight $\theta^{J'} \leftarrow \theta^J$
Initialize replay buffer R_J
for *episode = 0, M* **do**
  Receive initial observation state s_0
  Set $t = 1$ and $d_t = 0$
  while $d_t = 0$ ***and*** $t \leq T$ **do**
    Pick up a real number x uniformly between 0 and 1
    if $x < \epsilon$ **then**
      Pick up an action a_t uniformly
    else
      Select an action a_t with largest $J\left(s_t, a_t\middle|\theta^J\right)$
    end
    Execute action a_t, get damage d_t observe new state s_{t+1}
    Store transition (s_t, a_t, d_t, s_{t+1}) in R_J
    Sample a random minibatch of N transitions (s_i, a_i, d_i, s_{i+1}) from R_J
    Set $y_i = (1 - d_i) \max_a J'\left(s_{i+1}, a\middle|\theta^{J'}\right)$
    Update judge module by minimizing the loss:

$$L = -\frac{1}{N}\sum_i \left(y_i \log J\left(s_i, a_i\middle|\theta^J\right) + (1 - y_i) \log\left(1 - J\left(s_i, a_i\middle|\theta^J\right)\right)\right)$$

    Update the target module:

$$\theta^{J'} \leftarrow \tau\theta^J + (1 - \tau)\,\theta^{J'}$$

  end
end

To train the judge module, we use three key ideas. First, we implement ϵ-greedy algorithm in the exploration part. At each step in the training process, a random step is taken with probability ϵ where all actions can be chosen with same probability. Otherwise, we greedily pick up the safest action, i.e. the action with largest J.

Second, we use a replay buffer [1] to reduce the influence of correlations between samples. At each timestamp t, a tuple (s_t, a_t, d_t, s_{t+1}) is added to the buffer, and a batch of samples is uniformly

chosen from the buffer. When the buffer is full, the oldest sample will be popped from the buffer.

Third, we do soft update [3] to the target network to improve the stability. Suppose the rate of updating is τ, during the updating process, the parameters of the target network $\theta^{B'}$ is

$$\theta^{B'} \leftarrow \tau\theta^{B} + (1-\tau)\,\theta^{B'}. \tag{7}$$

As long as we obtain the judge module, we can finalize the barrier network with

$$B(s,a) = \log\left[J(s,a)\right]. \tag{8}$$

3.2 Trigram Barrier Network

A small flaw of the barrier network is that when training the judge module, while label 0 can appear frequently due to immediate damage, the label 1 does not show up naturally. To fix this problem, we introduce the information of the state after the next state, also known as the next next state: if the next next state is very safe, meaning that the J value of the safest action of the next next state can surpass a customized bar c, the original state-action pair will be award a label 1 to update the judge module. To separate this updated version from the original one, we call it trigram barrier network.

3.3 Deep Q Network

The action of the whole training process of DQN is constrained by the barrier network. Therefore, only safe state-action pairs take part. Similar with the previous step, we use ϵ-greedy, replay buffer, and soft update to improve the performance.

4 Experiment

We implement our algorithm on the Pendulum question, the goal of which is to swing a pendulum to the upright position. Let $\theta_t \in [-\pi, \pi)$ be the angle to the upright position at timestamp t and ω_t be the angular velocity. Then, the observation of state s_t is $s_t = (\cos(\theta_t), \sin(\theta_t), \omega_t)$. When $\cos(\theta_t) < 0$, the inspector will detect damage $d_t = 1$.

4.1 Barrier Network

Figure 1 shows the training process of the (bigram) judge module, while Figure 2 shows its counterpart of the trigram module. We can see that by adding extra assured 1 to the sample, the safe area gradually and smoothly expands after the chaos era, rather than twists from time to time.

To quantify the results, we uniformly sample states from the whole state space. Then, we distinguish the safe states with our barrier network and run the process by selecting the safest action evaluated by the judge module at each step. The iteration is forced to be terminated at step 200. So, the maximum iteration that the agent can travel, starting from a safe state, is 200. We

Algorithm 2 Trigram Judge Module

Randomly initialize judge module $J\left(s, a\middle|\theta^J\right)$ weight θ^J
Initialize target module J' with weight $\theta^{J'} \leftarrow \theta^J$
Initialize replay buffer R_J
Initialize transition deque T_J with maximum length 3
for *episode = 0, M* **do**
  Receive initial observation state s_0
  Set $t = 1$ and $d_t = 0$
  while $d_t = 0 \ \ \& \ \ t \leq T$ **do**
    Pick up a real number x uniformly between 0 and 1
    if $x < \epsilon$ **then**
      Pick up an action a_t uniformly
    else
      Select an action a_t with largest $J\left(s_t, a_t\middle|\theta^J\right)$
    end
    Execute action a_t, get damage d_t observe new state s_{t+1}
    Store transition (s_t, a_t, d_t) in T_J
    if $t \geq 3$ **then**
      Store transition $(s_{t-1}, a_{t-1}, d_{t-1}, s_t, s_{t+1})$ in R_J
    Sample a random minibatch of N transitions $(s_i, a_i, d_i, s_{i+1}, s_{i+2})$ from R_J
    if $\max_a J'\left(s_{i+2}, a\right) \geq c \ \ \& \ \ d_i = 0$ **then**
      Set $y_i = 1$
    else
      Set $y_i = (1 - d_i) \max_a J'\left(s_{i+1}, a\middle|\theta^{J'}\right)$
    end
    Update judge module by minimizing the loss:

$$L = -\frac{1}{N}\sum_i \left(y_i \log J\left(s_i, a_i\middle|\theta^J\right) + (1 - y_i) \log\left(1 - J\left(s_i, a_i\middle|\theta^J\right)\right)\right)$$

    Update the target module:

$$\theta^{J'} \leftarrow \tau\theta^J + (1 - \tau)\,\theta^{J'}$$

  end
end

Algorithm 3 Deep Q Learner

Randomly initialize deep q network Q weight θ^Q
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$
Initialize replay buffer R_Q
for *episode = 0, M* **do**
  Receive initial observation state s_0
  for *t = 1, T* **do**
    Pick up a real number x uniformly between 0 and 1
    if $x < \epsilon$ **then**
      Pick up an action a_t uniformly where $B\left(s_t, a_t\right) = 0$
    else
      Select an action a_t with largest $Q\left(s_t, a_t \middle| \theta^Q \middle| B\left(s_t, a_t\right) = 0\right)$
    end
    Execute action a_t, get reward r_t and observe new state s_{t+1}
    Store transition $\left(s_t, a_t, r_t, s_{t+1}\right)$ in R_Q
    Sample a random minibatch of N transitions $\left(s_i, a_i, r_i, s_{i+1}\right)$ from R_Q
    Set $y_i = r_i + \gamma \max_a Q'\left(s_{i+1}, a \middle| \theta^{Q'}\right)$
    Update q network by minimizing the loss:

$$L = -\frac{1}{N} \sum_i \left(y_i - Q\left(s_i, a_i \middle| \theta^Q\right)\right)^2$$

    Update the target network:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau)\, \theta^{Q'}$$

  end
end

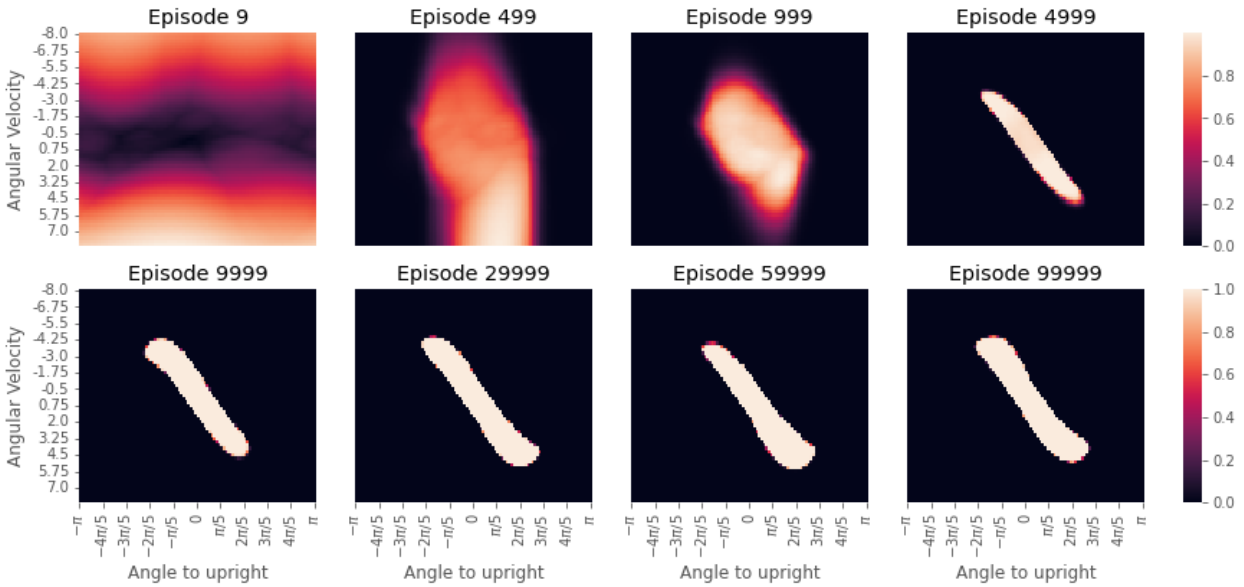

Figure 1: Judge Boundary

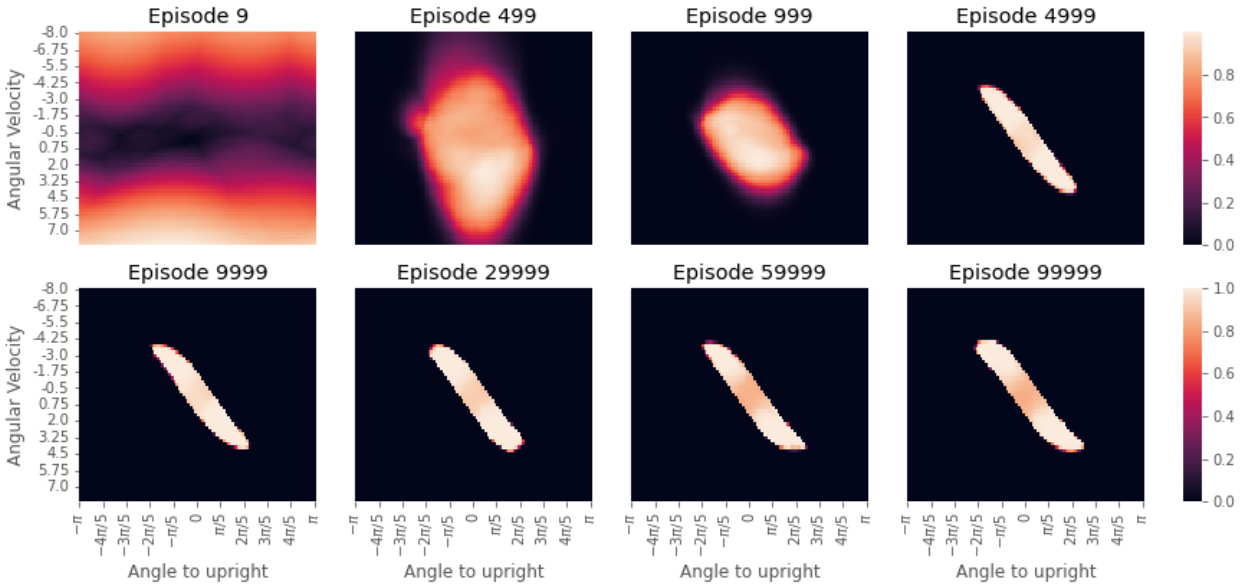

Figure 2: Trigram Judge Boundary

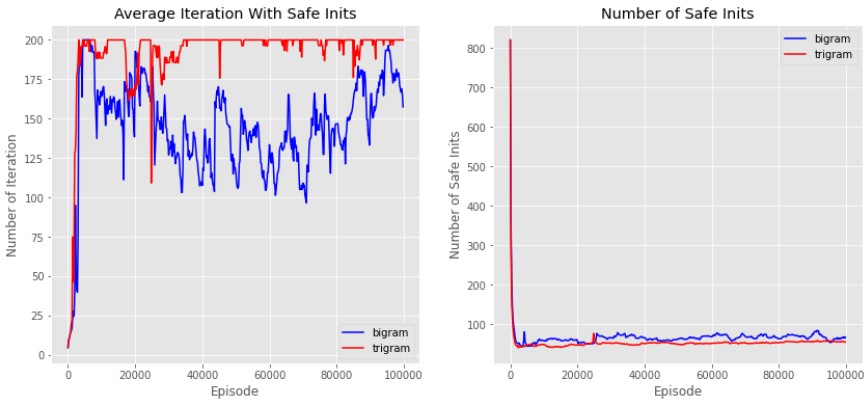

Figure 3: Evaluation of Barrier Network

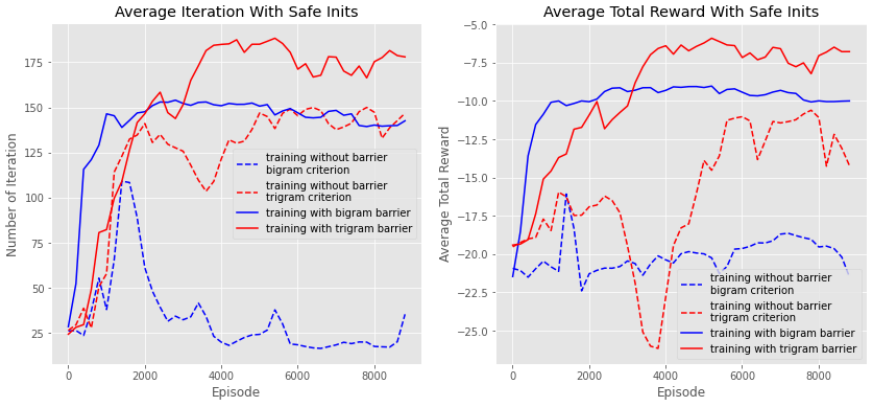

Figure 4: Evaluation of Deep Q Network

calculate the average iteration starting from safe states and count the number of safe states as in figure 3.

In total, we sampled 825 states and around 55 of them are classified safe by our exp-barrier network. We notice that the trigram barrier network provides more stable training process and safer outputs.

4.2 Deep Q Network

We run the deep q learning algorithm either with or without constraint from barrier network. We use the original definition of immediate reward in the gym package where

$$r_t = -\left(\theta_t^2 + 0.1\omega_t^2 + 0.001a_{t-1}^2\right). \tag{9}$$

Therefore, the minimum immediate reward can be achieved, when $\theta = -\pi$, $\omega = 8$, and $a = 2$, which is $r_{min} \approx -16.2736$. Since the maximum number of iteration is set to be 200, setting

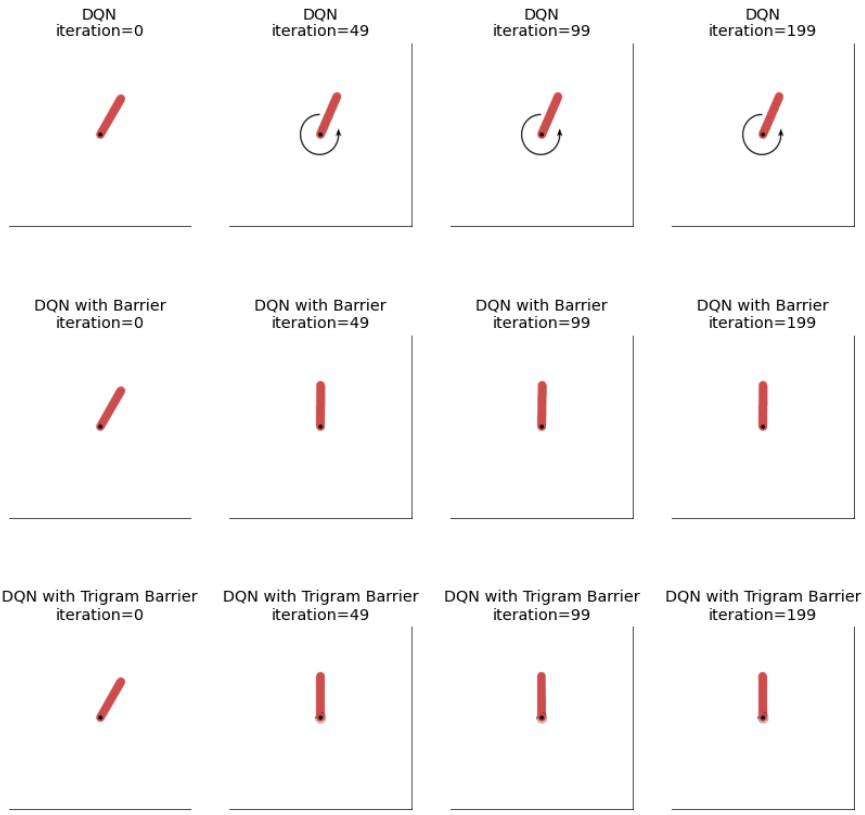

Figure 5: Simulation

$\gamma = 0.99$, the theoretical minimum long term reward is

$$Q_{min} = \sum_{t=0}^{200-1} \gamma^t r_{min} \approx -1409.3268. \tag{10}$$

We compare the average iteration the agent can travel from safe initial states, as well as the corresponding average total rewards. While the test group consists of two deep q network constraint by the bigram and trigram barrier network during training, the deep q network of the control group is barely deep q learning. The only modification we made is that we add a penalty term, which is $5Q_{min} \approx -7046.634$, when we meet an immediate damage, while the original procedure was to just terminate the episode with 0 next state value.

We notice that while constraint q network performs much better in both metrics under both criterion, with trigram barrier network, the average reward is higher with more iterations. Because our rewards are always negative, this means that while constraint with trigram barrier network is safer, its performance, in terms of rewards, is also better, comparing with DQN constraint with bigram barrier network.

As a real simulation, we run all of our three agents, starting from $[-0.5235987755982991, 1.125]$. We can see that pure DQN, even with punishment, cannot maintain the pendulum close to the up-

right position and yet, the torque applied is extremely large. DQN with barrier, however, is able to keep the pendulum almost vertical with nearly 0 torque.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[2] Agustin Castellano, Hancheng Min, Juan Bazerque, and Enrique Mallada. Learning to act safely with limited exposure and almost sure certainty. *arXiv preprint arXiv:2105.08748*, 2021.

[3] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.