Project Report

Car Racing with PPO, Learning from Raw Pixels

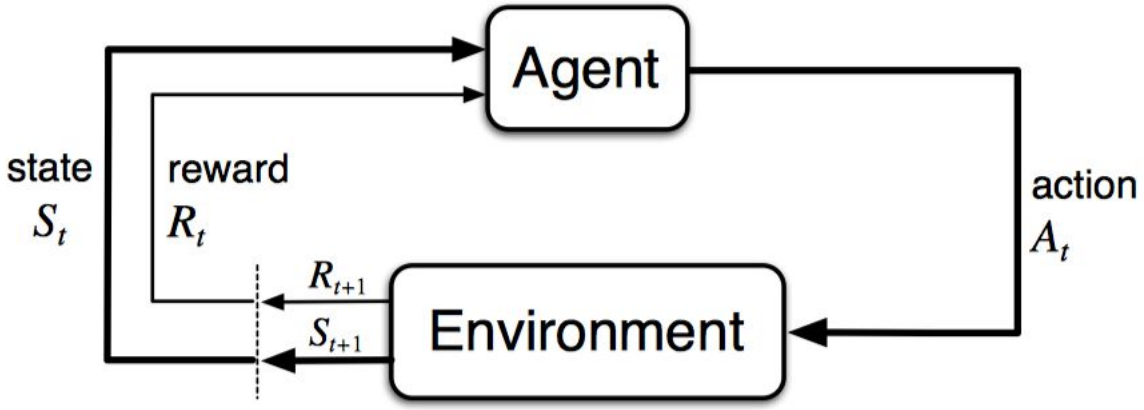
I. Members

1. *Nguyễn Minh Dương (17021227)*
2. *Nguyễn Trung Hiếu (17021246) - leader , email : bipp811@gmail.com*
3. *Nguyễn Đức Lâm (17021280)*

Contents

II. Proximal Policy Optimization (PPO)

1) Policy Gradient in RL



Principle :

We observe and act

We keep what is working and throw away what is not

$$\pi\left(u|s\right)$$

2) PPO and its concepts

It is a member of a new family of reinforcement learning methods known as Policy Gradient methods, which basically performs one policy update as per sample, however, PPO alternates between

1. Sampling data by interacting with the environment （ on-policy ）
2. Optimizing 'surrogate' objective function using **stochastic gradient ascent.**

3) Why PPO?

1. It has some benefits of Trust Region Policy Optimization [TRPO], but much simpler (in terms of implementation), more general, and has better sample complexity.
2. It outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity and simplicity, and wall-time.

3. Core

- *The Clipped Surrogate Objective*
- *Multiple epochs of stochastic gradient ascent to perform each policy update".*

4) Clipped Surrogate Objective

The Clipped Surrogate Objective is a drop-in replacement for the policy gradient objective that is designed to improve training stability by limiting the change you make to your policy at each step.

For vanilla policy gradients

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t|s_t)\hat{A}_t\right]$$

For TPRO Object function :

- *current policy / previous policy*

$$L^{TRPO}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

PPO Object function : Clipped Surrogate Objective

same objective from before

same, but r(θ) is clipped between (1 - e, 1 + e)

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right]$$

min of the same objective from before and the clipped one

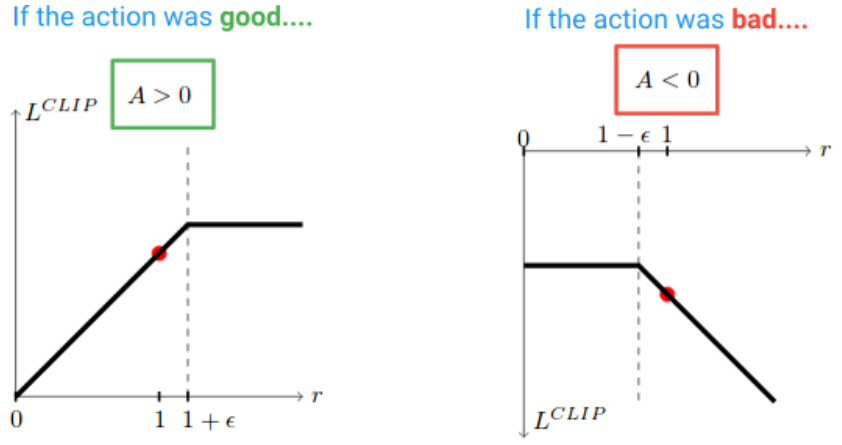


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms.

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

  by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

5) Multiple epochs for policy updating

Unlike vanilla policy gradient methods, and *because of the Clipped Surrogate Objective function*, PPO allows you to run multiple epochs of gradient ascent on your samples without causing destructively large policy updates. This allows you to squeeze more out of your data and reduce sample inefficiency.

PPO runs the policy using *N* parallel actors each collecting data, and then it samples mini-batches of this data to train for *K* epochs using the Clipped Surrogate Objective function. See full algorithm below (the approximate param values are: *K* = 3-15, *M* = 64-4096, *T* (horizon) = 128-2048):

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

III. Project Algorithm Implementation base on paper

1) Update

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right],$$

Where c1,c2 are coefficients,and S denotes an entropy bonus,and L_t^{VF} is a squared-error loss $(V_\theta(st) - V_t targ)^2$

We must use a loss function that combines the policy surrogate and a value function error term. This objective can further be augmented by adding an entropy bonus to ensure sufficient exploration

2) Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution

However, in real-world control problems, the actions one can take are bounded by physical constraints, which introduces a bias when the standard Gaussian distribution is used as the stochastic policy. In this work, we propose to use the Beta distribution as an alternative and analyze the bias and variance of the policy gradients of both policies. We show that the Beta policy is bias-free and provides significantly faster convergence and higher scores over the Gaussian policy

3.3. Beta Policy

Let us now consider the Beta distribution

$$f(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \qquad (9)$$

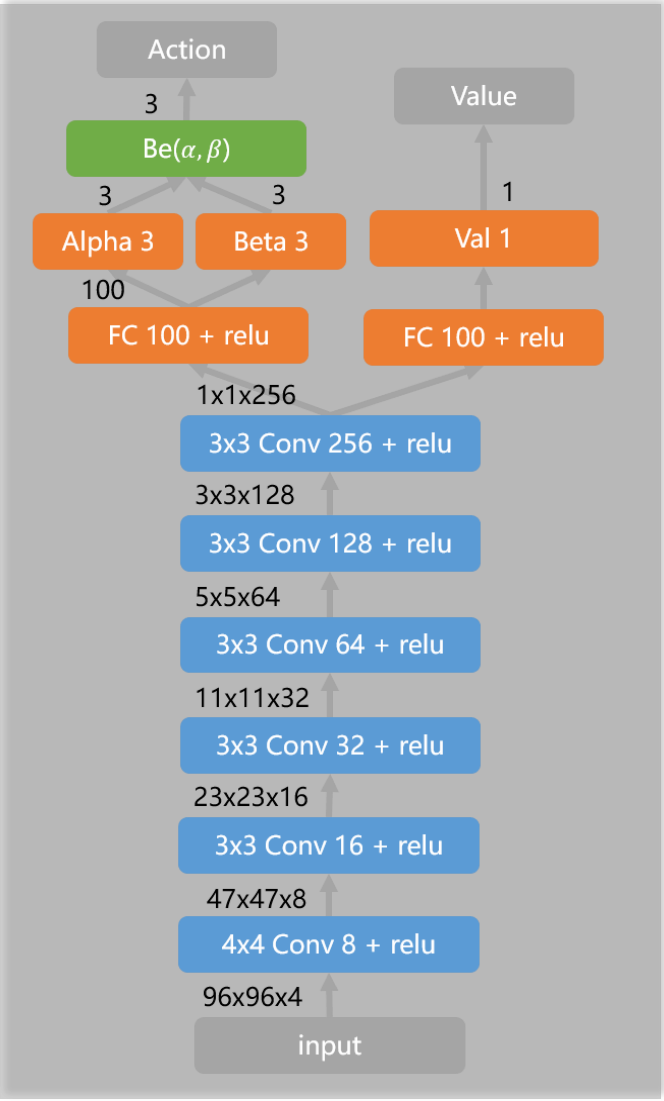
where α and β are the shape parameters and $\Gamma(\cdot)$ is the *Gamma* function that extends factorial to real numbers, i.e. $\Gamma(n) = (n-1)!$ for positive integer n. The beta distribution has a support $x \in [0,1]$ (as shown in Figure 3) and it is often used to describe the probability of success, where $\alpha - 1$ and $\beta - 1$ can be thought of as the counts of successes and failures from the prior knowledge respectively.

We use $\pi_\theta(a|s) = f(\frac{a+h}{2h};\alpha,\beta)$ to represent the stochastic policy and call it the *Beta Policy*. Since the beta distribution has finite support and no probability density falls outside the boundary, the Beta policy is bias-free. The shape parameters $\alpha = \alpha_\theta(s), \beta = \beta_\theta(s)$ are also modeled by neural networks with parameter θ. In this paper, we only consider the case where $\alpha, \beta > 1$, in which the Beta distribution is concave and unimodal.

IV) Implementation

1) Model : Convolutional Neural Network (CNN)

- CNN là một dạng đặc biệt của mạng nơ-ron được sử dụng chuyên sâu trong Computer Vision. Ứng dụng mạnh trong xe tự lái hay nhận dạng khuôn mặt.
- Kiến trúc và cách hoạt động của 1 mạng CNN bao gồm: đưa ảnh input qua các convolutional layer để trích xuất các đặc trưng của ảnh. Tiếp đó ảnh sẽ được đưa qua pooling layer nhằm giảm không gian/ dimensions để giảm độ phức tạp. Cuối cùng là thêm một mạng nơ ron với một tầng ẩn (hidden layer) ở cuối làm một fully connected layer để từ đó có thể classify ảnh
- We use the CNN model with 6 levels Conv2d (torch.nn.Conv2d) and 6 Rectified linear units ReLU (torch.nn.ReLU). The rewards, target and advatage values are calculated by the model:



2) Learning from raw pixels

Stack of 4 frames

State is defined as adjacent 4 frames in shape (4, 96, 96).

Four frames are allocated in the function reset() (class Wrapper)

self.stack = [img_gray] * 4

At every time step the first frame is discarded and one frame (img_gray) is added to the stack (of 4 frames), see function step() (class Wrapper)

stack.pop(0)

stack.append(img_gray)

3) PPO

```
for _ in range(EPOCH):
    for index in BatchSampler(SubsetRandomSampler(range(MAX_SIZE)), BATCH, False):

        alpha, beta = self.net(s[index])[0]
        dist = Beta(alpha, beta)
        a_logp = dist.log_prob(a[index]).sum(dim=1, keepdim=True)
        ratio = torch.exp(a_logp - old_a_logp[index])

        surr1 = ratio * adv[index]

        # clipped function
        surr2 = torch.clamp(ratio, 1.0 - EPS, 1.0 + EPS) * adv[index]
        action_loss = -torch.min(surr1, surr2).mean()
        value_loss = F.smooth_l1_loss(self.net(s[index])[1], target_v[index])
        loss = action_loss + 2. * value_loss

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
```

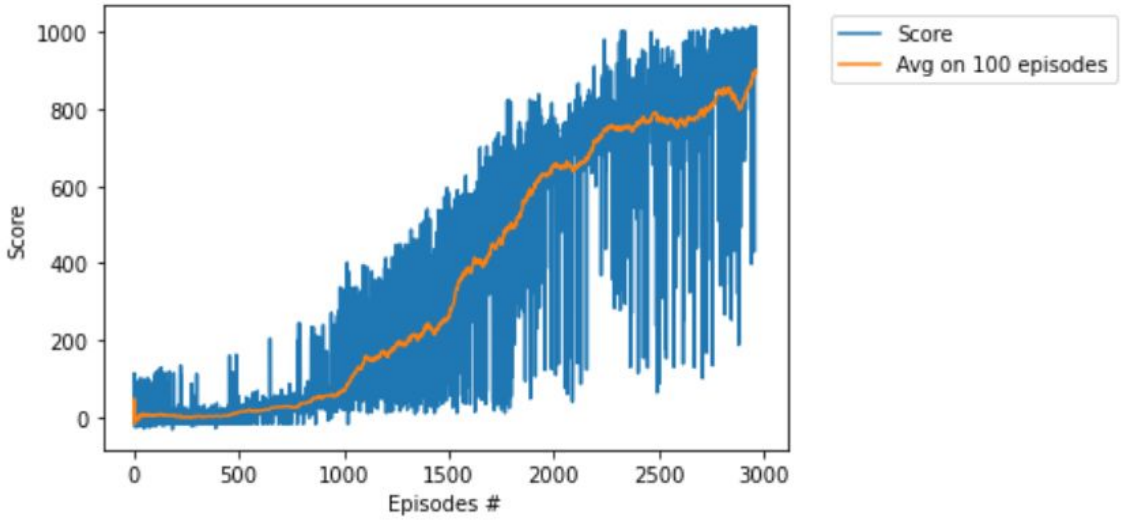
V. Practice Result

- Nhóm đã train trên 1 máy trong vòng 7 tiếng 51 phút và ghi lại các giá trị total reward (Score) của mỗi episode, giá trị trung bình của Score (Average_Score) cùng với Running Score. Tổng cộng số episode đã train là 2962

```
Ep. 2948, Ep.Timesteps 100, Score: 879.48, Avg.Score: 885.99, Run.Score 867.37, Time: 07:48:34
Ep. 2949, Ep.Timesteps 100, Score: 1000.70, Avg.Score: 891.67, Run.Score 868.70, Time: 07:48:44
updating
Ep. 2950, Ep.Timesteps 100, Score: 906.45, Avg.Score: 890.84, Run.Score 869.08, Time: 07:49:27
Ep. 2951, Ep.Timesteps 100, Score: 979.66, Avg.Score: 892.42, Run.Score 870.19, Time: 07:49:37
Ep. 2952, Ep.Timesteps 100, Score: 925.81, Avg.Score: 894.81, Run.Score 870.74, Time: 07:49:47
Ep. 2953, Ep.Timesteps 66, Score: 429.32, Avg.Score: 889.77, Run.Score 866.33, Time: 07:49:54
Ep. 2954, Ep.Timesteps 88, Score: 1012.90, Avg.Score: 889.89, Run.Score 867.79, Time: 07:50:02
Ep. 2955, Ep.Timesteps 92, Score: 1008.00, Avg.Score: 891.43, Run.Score 869.20, Time: 07:50:11
Ep. 2956, Ep.Timesteps 100, Score: 956.52, Avg.Score: 891.20, Run.Score 870.07, Time: 07:50:21
Ep. 2957, Ep.Timesteps 94, Score: 1006.30, Avg.Score: 892.09, Run.Score 871.43, Time: 07:50:30
Ep. 2958, Ep.Timesteps 100, Score: 907.41, Avg.Score: 891.07, Run.Score 871.79, Time: 07:50:40
Ep. 2959, Ep.Timesteps 97, Score: 1003.40, Avg.Score: 896.79, Run.Score 873.11, Time: 07:50:50
Ep. 2960, Ep.Timesteps 100, Score: 1000.80, Avg.Score: 899.37, Run.Score 874.38, Time: 07:50:59
Ep. 2961, Ep.Timesteps 90, Score: 1010.10, Avg.Score: 900.62, Run.Score 875.74, Time: 07:51:08
Solved environment! Running score is 875.74, Avg.Score: 900.62 !
```

-
- Đồ thị dưới là kết quả training sau 2962 episode

```
length of scores:  2962 , len of avg_scores:  2962
```



- Training after 2900 episode: model đã chạy tương đối tốt, chỉ còn một vài episode model chạy lệch khỏi đường đua. Điểm trung bình là 900

Ref :

Policy Gradient
https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146

Paper PPO : https://arxiv.org/abs/1707.06347

CNN :
https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Project github : https://github.com/xtma/pytorch_car_caring/blob/master/train.py

SPD - Beta Distribution : http://proceedings.mlr.press/v70/chou17a.html