CIS 680 PROJECT REPORT

Dingding Zheng(Penn ID: 62643280), Mengwei Zhang(Penn ID: 66376864), Huize Huang(Penn ID: 50132108)
University of Pennsylvania
(Dated: December 19,2019)

CarRacing-v0, a environment in OpenAI, is a typical continuous task of learning from pixels. PPO with clipped objective is used to solve the problem. By adding soft constraints to objective function, the agent state is restricted in the Trust Region which ensures better stability and model performance.

Approach: During the final project, we decide to implement PPO algorithm and make some improvements.

I. INTRODUCTION

CarRacing-v0, a environment in OpenAI, is a typical continuous task of learning from pixels. The state of this environment consists of 96*96 pixels. For each frame and track tile visited, the reward is -0.1 and +1000/N, where N is the total number of tiles in track. For Deep Reinforcement Learning, researchers have proposed several algorithms as following:

A. Deep Q-Network (DQN)

In 2015, DQN[1] beat human experts in many Atari games. In reinforcement learning, Q-learning learns the action-value function Q(s,a), which is a standard evaluating the performance of taking an action at a particular state. In Q-learning, we build a memory table Q[s,a] and sample an action from the current state. And we get an action a' from table which maximizes value of Q function. A Deep Network DQN[2] is proposed to approximate Q(s,a) to solve the memory crash issue caused by large combinations of states and actions.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

B. Trust Region Policy Optimization (TRPO)

Policy Gradient methods[3] are classical in Reinforcement Learning field. It's basic principle is to use gradient ascent to follow policies with the steepest increase in rewards. But there exists an issue that the first-order optimizer is not that accurate which may cause the agent make bad moves and ruin the whole training process. Thus, TRPO[4] is proposed to solve this problem.

For Policy Gradient method, it computes the steepest ascent direction for the rewards and update the policy towards that direction

$$g = \nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right]$$
$$\theta_{k+1} = \theta_k + \alpha g$$

But in this case, applying a large learning rate may make the algorithm suffers from convergence problem badly. Thus, TRPO[5] is proposed to constrain the policy changes so the agent may not make too aggressive moves.

TRPO updates policies by taking the largest step possible to improve performance, while satisfying a special constraint on how close the new and old policies are allowed to be. The constraint is expressed in terms of KL-Divergence, a measure of (something like, but not exactly) distance between probability distributions.

For the trust region in TRPO, we determine the maximum step size that we want to explore and then we locate the optimal point within it. And the trust region can be shrunk if the policy is changing too much.

TRPO trains a stochastic policy in an on-policy way, which means it explores by sampling actions according to the latest version of its stochastic policy. Both initial conditions and the training procedure determine the amount of randomness in action selection.

C. Proximal Policy Optimization (PPO)

PPO[6] is great for it's easily hyperparameter tuning. For traditional reinforcement learning algorithms, they requires substantial effort in tuning to get good results. The occurence of PPO strikes a balance between sampling complexity, ease of implementation and ease of parameter tuning. It tries to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

This is different from normal policy gradient, which keeps new and old policies close in parameter space. But

even seemingly small differences in parameter space can have very large differences in performance—so a single bad step can collapse the policy performance. This makes it dangerous to use large step sizes with vanilla policy gradients, thus hurting its sample efficiency. TRPO[7] nicely avoids this kind of collapse, and tends to quickly and monotonically improve performance.

PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance. Details of algorithm is shown as following:

1. Input: Initial policy parameters θ_0, initial value function parameters ϕ_0.

2. Hyperparameters: KL-divergence limit δ, backtracking coefficient α, backtracking maximum steps K

3. **for** $k = 0, 1, 2, ...$**do**

4. Collect set of trajectories $D_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$

5. Compute advantage estimate, $\hat{A}_t$ based on V_{Φ_k}

6. Estimate policy gradient

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta \left(a_t|s_t\right)_{\theta_k} \hat{A}_t$$

7. Compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k \tag{1}$$

Where $\hat{H}_k$ is the Hessian of the sample average KL-deivergence

8. Backtracking

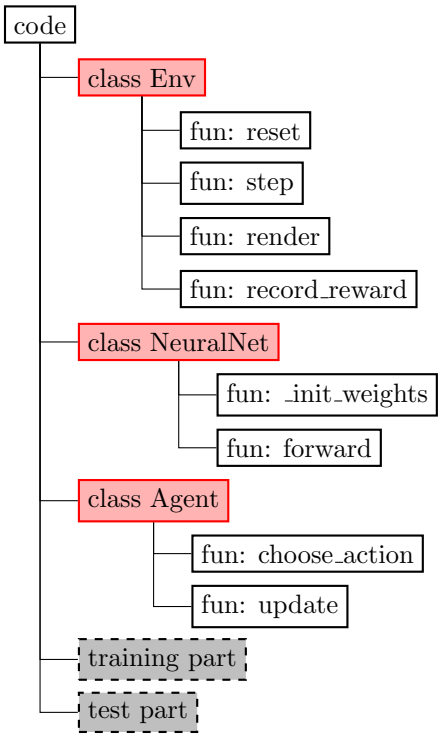
$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k \tag{2}$$

9. Fit value function

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi \left(s_t\right) - \hat{R}_t\right)^2 \tag{3}$$

10. **end for**

II. CODE STRUCTURE



III. OUR APPROACH

We decide to choose PPO algorithm to finish this project since it derives from classical Policy Gradient method and alternates between sampling data through interaction with the environment.

Compared to TRPO, PPO simply uses linear approximation. Moreover, PPO uses multiple epochs minibatches update instead of performing only one gradient update as per sample like policy gradient methods.

There are two ways of improving PPO algorithm and share something in common. The first one is "PPO with Adaptive KL Penalty". It adds a soft constraint to the objective function for optimization.

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t\right] - \beta \hat{\mathbb{E}}_t \left[\text{KL}\left[\pi_{\theta_{\text{old}}} \cdot |s_t), \pi_\theta(\cdot|s_t)\right]\right]$$

β sets weight of the penalty. it defines the severity of penalty how the new policy is different from the old one. When the KL divergence of two policies is higher than the target value, β decreases. Details of the algorithm is shown as following:

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
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

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

The second one is "PPO with Clipped Objective". In this algorithm, two policy networks: current policy and old policy are used to compute ratio which measures the difference between two policies. By importance sampling, a new policy can be evaluated with samples collected from an older policy.

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right]$$

$$r_t(\theta) = \pi_\theta(a_t|s_t) / \pi_{\theta_k}(a_t|s_t)$$

The new objective function can be modified as:

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

When the probability ratio falls outside the range (1 - $\mathcal{E}$, 1 + $\mathcal{E}$), the advantage function will be clipped.

Details of the algorithm is shown as below:

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

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

end for

These two algorithms both set constraints to parameters updating process. During the implementation, we choose "PPO with Clipped Objective" for its simple implementation. Besides, it be optimized using Adam optimizer.

IV. EXPERIMENTS

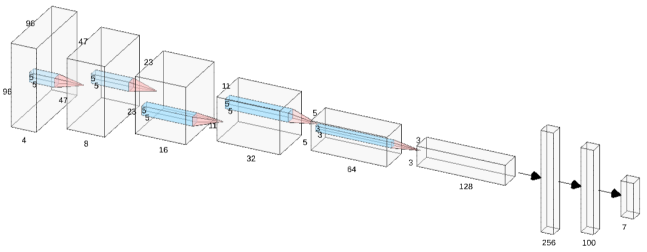
A. Model Parameter Table

The parameter values we choose during implementation is shown as below:

Discount Factor γ	0.98
Number of Image Stack	4
Learning Rate	1e-3
Number of Time Step	1000
Number of Training Epoch	20000

B. Structure of Actor-Critic Network

The figure shows the structure of our actor-critic network. It contains a CNN backbone and three headers. The CNN backbone of a CNN net which takes the image data as inputs and extract the feature of the input image.The three headers are responsible for v, α and β for PPO.



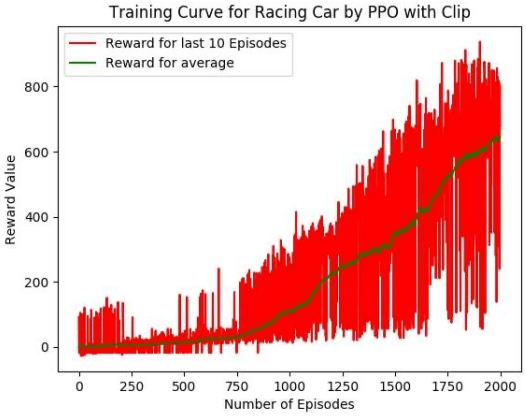
C. Structure Parameter Table

The parameter values we choose during implementation is shown as below:

conv1	(6, 6, 8), s=2, p=1, BatchNorm, ReLU
conv2	(5, 5, 16), s=2, p=1, BatchNorm, ReLU
conv3	(5, 5, 32), s=2, p=1, BatchNorm, ReLU
conv4	(3, 3, 64), s=2, BatchNorm, ReLU
conv5	(3, 3, 128), s=1, BatchNorm, ReLU
conv6	(3, 3, 256), s=1, ReLU
v_head	Linear(256, 128), ReLU, Linear(128, 32) ReLU, Linear(28, 1), ReLU
fc1	Linear(256, 128), ReLU
alpha_head	Linear(128, 3), SoftPlus
beta_head	Linear(128, 2), SoftPlus

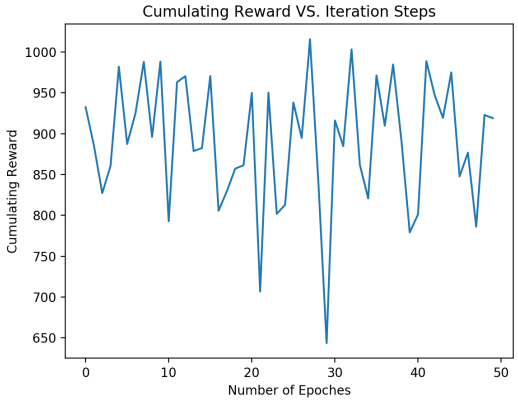
D. Training & Testing Plot

The Training curve for 2000 Epoches is shown as below for clearer view:



This is just a plot for demonstration. Our model performance is much better, whose cumulative reward reaches 893.

For testing part, we've tested 50 times. The average Cumulative Reward is 890. The Testing curve is shown as below:



V. CONCLUSION

In this project, we use Deep Reinforcement Learning to solve the problem. For the design of network architecture, we use Actor-Critic Network.

The "Critic" part estimates the value function. while the "Actor" part updates the policy distribution in the direction suggested by the Critic. Both the Critic and Actor functions are parameterized with our neural networks. The value function we choose is Advantage, which is similar to Q value. The expression of Advantage is shown as below:

$$A(x, u)=\max(Q(x,u'))+(Q(x,u)-\max(Q(x,u')))\star k/dt$$

For the training algorithm, we improve PPO by adding Clipped Objective. This constraint makes our state change stay in the trust region and the parameter change is restricted according to clip parameter $\mathcal{E}$. "PPO with Clipped Objective" is an algorithm that improves stability of training process and model performance.

During the Experiment, we run our model 50 times and get an average cumulative reward at around 890. This score guarantees a great simulation for our racing car.

VI. ACKNOWLEDGEMENT

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. A special gratitude we give to our course professor, [Dr. Jianbo Shi], whose excellent teaching skills, helped us to complete the project.

Furthermore we would also like to acknowledge with much appreciation the work from Xiaoteng Ma: `https://github.com/xtma/pytorch_car_caring` We make some improvements to the network architecture by adding BatchNorm and Regularization. Besides, we define new Advantage expression and add soft constraints to ordinary PPO algorithm. This ensures greater performance.

In the end, we want to thank all TAs of this course. They contribute a lot to the course materials and help us finishing homework and project. Merry Christmas!

VII. REFERENCES

[1] Bakker, B. 2001. Reinforcement learning with long shortterm memory. In NIPS, 1475–1482. MIT Press.

[2] Cun, Y. L. L.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. Proceedings of IEEE 86(11):2278–2324.

[3] Karpathy, A.; Johnson, J.; and Li, F.-F. 2015. Visualizing and understanding recurrent networks. arXiv preprint.

[4] Tieleman, T., and Hinton, G. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

[5] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The arcade learning environment: An evaluation platform for general agents". In: Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.

[6] S. Kakade and J. Langford. "Approximately optimal approximate reinforcement learning". In: ICML. Vol. 2. 2002, pp. 267–274.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: arXiv preprint arXiv:1602.01783 (2016).

VIII. VIDEO LINK

`https://drive.google.com/open?id=1ldHZOhVXmvp7qSOmkyvO5QQIUIsVQm7Z`