Reinforcement Learning Implementation Project Report

Alessio Ragno

Master's Degree in Artificial Intelligence and Robotics

Department of Computer, Control and Management Engineering

Sapienza University of Rome

November 2019

Contents

1 Introduction

What follows is the report for the Reinforcement Learning Implementation Project of Machine Learning Class at Artificial Intelligence and Robotics Course.

The project consists in the implementation of an assigned Reinfocement Learning algorithm and its evaluation in two different environments.

The assigned algorithm is Trust Region Policy Optimization (TRPO) and it has to be evaluated on the following OpenAI Gym environments:

- **MountainCar-v0** is an environment in which a car has to climb a hill and reach a specific point indicated by a flag. MountainCar-v0 is a continuous state space environment that can be controlled with discrete actions. At each timestep the agent can choose between three actions: pushing the car left, right or not applying any force to the car.

- **Pong-v0** is an environment in which the agent has to play Pong game against a virtual player. The state is the image of the game and the agent acts using the Atari 2600 commands: NOOP, FIRE, RIGHT, LEFT, RIGHTFIRE, LEFTFIRE.

2 Trust Region Policy Optimization

Trust Region Policy Optimization was published by John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan and Pieter Abbeel on April 2017. TRPO is an improvement of the vanilla Policy Gradient algorithm implemented by REINFORCE with baseline. TRPO aims to make gradient step which are not "too big" to improve the algorithm stability. The length of the step is defined in terms of a hyperparameter δ which represents the size of the trust region.

Algorithm 1 TRPO

for iteration = 1,2,... **do**
    Run policy for T timesteps or N trajectories
    Estimate advantage function at all timesteps
    for $t = 1, T$ **do**
        maximize $\sum_{n=1}^{N} \frac{\pi_\theta(a_n|s_n)}{\pi_{\theta_{old}}(a_n|s_n)} A_n$
        s.t. $\overline{KL}_{\pi_{\theta_{old}}}(\pi_\theta) < \delta$
    end for
end for

Algorithm 2 TRPO

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K

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

Collect set of trakectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment

Compute rewards-to-go R_t

Compute advantage estimates, A_t (using any method of advantage estimation) based on the current value function V_{ϕ_k}

Estimate policy gradient as

$$g_k = \frac{1}{D_k} \sum_{\tau \in D_k} \sum_{t=0}^{T} \nabla_\theta log \pi_\theta(a_t|s_t)|_{\theta_k} A_t$$

Use the conjugate gradient algorithm to compute

$$x_k = H_k^{-1} g_k$$

where H_k is the Hessian of the sample average KL-divergence

Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{x_k^T H_k x_k}} x_k$$

where $j \in \{0, 1, 2, ..., K\}$ is the smallest value which improves the sample oss and satisfies the sample KL-divergence constraint

Fit the value function by regression on mean-squared-error:

$$\phi_{k+1} = argmin_\phi \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0} (V_\phi(s_t) - R_t)^2$$

tipically via some gradient descent algorithm

end for

Algorithm 1 shows the basic TRPO pseudocode.

To implement TRPO, the authors suggest to maximize the vanilla policy gradient (which they call surrogate loss) and at the same time minimize the KL divergence calculating the hessian vector product with the surrogate loss gradient.

After generating the step direction using conjugate gradients direction the algorithm uses linesearch to find the right stepsize.

Applying the suggested calculation we obtain Algorithm 2.

In the paper, the authors showed the hyperparameters used in each environment and it turned out that using $K = 10$ maximum backtracking steps for the line search is sufficient.

Increasing δ the algorithm becomes more "aggressive" and at the same time unstable, while decreasing it, the algorithm becomes slower. The authors showed that using $\delta = 0.01$ is a good tradeoff between velocity and stability.

Since the calculation of the gradients, the conjugate gradient and the linesearch make this algorithm computationally expensive, a more efficient and improved version was later published as Proximal Policy Optimization (PPO) by Schulman.

3 Model Training and Evaluation

This section shows the results obtained in the two environment mentioned above. For MountainCar-v0 using a fully connected was sufficient while for Pong-v0 convolutional layers were needed due to the fact that the state is an image.

The code was fully written in Tensorflow 2.0 using Keras APIs for the construction of the Neural Networks. In Tensorflow 2.0, contrary to Tensorflow 1.x, eager execution is enabled by default and the graph is not kept for the whole execution but only built when needed. This led to multiple nested functions for the calculation of the gradients.

3.1 MountainCar-v0

feature	clang	gcc	icc	H	L
0	300599	259228	256805	453070	363562
cmp3	53406	44174	44225	76154	65651
cwde0	22	724	0	37	709
dec2	573	173	374	802	318
leave0	3	1317	5749	11	7058
movapd3	2515	130	0	1969	676
movaps2	965	710	4316	4777	1214
movhps3	0	8	143	149	2
movlpd3	64	1594	4	748	914
xorps3	18	31	311	284	76

Table 1: Frequency of tokens over the different classes

3.2 Pong-v0