Project Report

Car Racing with PPO, Learning from Raw Pixels

I. Members

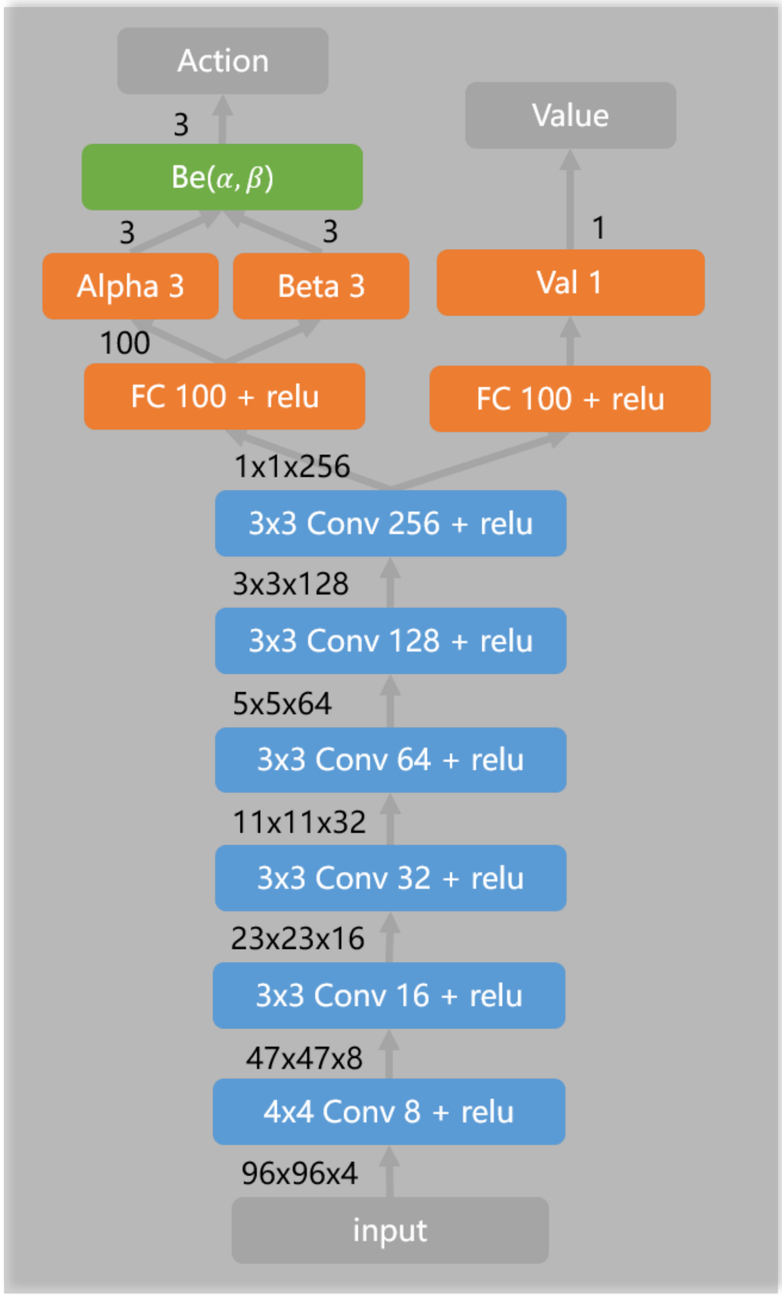
1. Nguyễn Minh Dương (17021227)
2. Nguyễn Trung Hiếu (17021246)

   - Leader , email : bipp811@gmail.com

3. Nguyễn Đức Lâm (17021280)

II. Idea , model training & update policy gradient

1. Model : Convolutional Neural Network (CNN)



Every action will be repeated for 8 frames. To get velocity information, state is defined as adjacent 4 frames in shape (4, 96, 96). Use a two heads FCN (fully convolutional network) to represent the actor and critic respectively. The actor outputs α, β for each actin as the parameters of Beta distribution.

2. Policy Gradient : Proximal Policy Optimization (PPO)

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
2. Optimizing 'surrogate' objective function using **stochastic gradient ascent.**

Why PPO?

1. It has some benefits of Trust Region Policy Optimization [TRPO], but much simpler (in terms of implementation), more general, and have 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.

PPO is a on-policy !

Online policy gradient methods are methods where

- i. Agents can pick actions on their own
- ii. Follows most obvious setup
    - a. Learn with exploration
    - b. Play with exploitation
- iii. Agent follows his own policy
    - a. Learn from experts (Imperfect)
    - b. Learn from recorded sessions (Recorded Data)

PPO Algorithm

- Uses fixed length of trajectory segments
- Each iteration : each of `N` (parallel) actors collect `T` timesteps of data
- Then use constant surrogate loss of these `NT` timesteps of data & optimize it with mini batches using **SGD** (or usually better performance using Adam) for `k` epochs.

Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1,2,... do
    for actor=1,2,...,$N$ 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 $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

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

3. Idea

+ First we have raw pixels as our input to the network - multi layer perceptron and it outputs a probability of action per step
+ Per epoch, it takes in a set of states and outputs the best policy.
+ The best stochastic policy as it allows exploration and optimization of the problem
+ We can brute force the learning approach however we need to save computations power and time; thus, we use policy gradient methods
+ We do not have labels so we simply do reinforcement learning (explore the environment, try different actions, observe what happens and exploit them later) and increase the log probability of things that worked
+ TRPO makes sure tweaking our network hyperparameters will not change our distribution of our actions for each state too much, we prefer smoothness.
+ We start random and upon stumbling into solutions increase the actions probability >> Continuously updating our distribution of our probability of actions, thus it is more and more rewarding
+ Discounted rewards is a heuristic for modulating the blame - for actions - in time sensitive < which is not very good in theory but good in practical>. It decays exponentially

$$\boxed{\sum_i A_i * \log p(y_i | x_i)}$$

Implementation

Iterate until termination condition or convergence

i. Process observations
ii. Feedforward to the NN
iii. Interact with environment
iv. Keep track of rewards, losses, states
v. Compute discounted rewards and advantage estimates
vi. Backpropagation the error
vii. perform the update for network params

III. Practice Result

Team is training agent and testing itt to get best result , we will update later .