PPO vs. DQN

Andy Piantoni, Ruben Jungius, Lisa Schneider

PPO - Proximal Policy Optimization

Developed by Schulman et al. [2], Proximal Policy Optimization (PPO) belongs to the class of on-policy algorithms. PPO introduced a new surrogate objective function to prevent excessively large policy updates by clipping the probability ratios between the new and old policies.

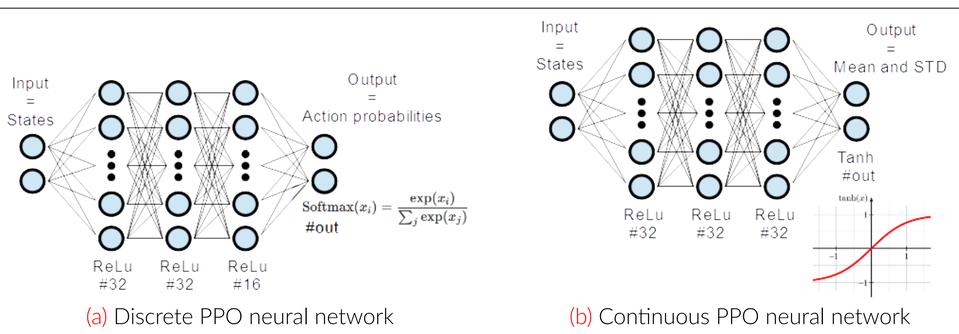
$$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{1}$$

Pseudocode

Algorithm 1 PPO implemented, Actor-Critic Style

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1$ **do**
    Run policy π_{old} in environment for T = maximum possible timesteps in environment
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  if some condition **then**
    Perform some action if condition is true
  else
    Perform another action if condition is false
  end if
  Optimize surrogate L with respect to θ, with K epochs and batch size $M = T$
  $\theta_{\text{old}} \leftarrow \theta$
end for

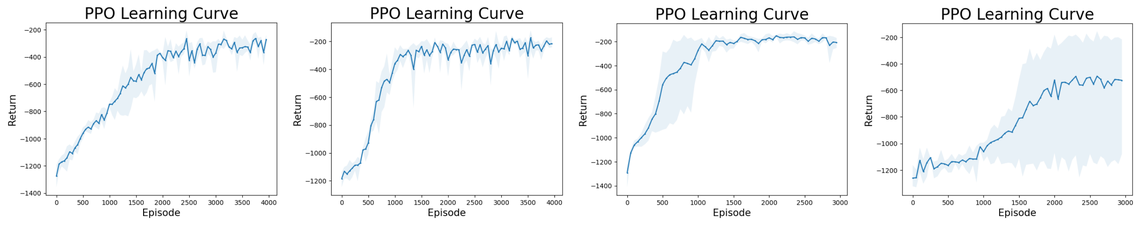
Structure of the NN



(a) Discrete PPO neural network (b) Continuous PPO neural network

Figure 1. Architecture of the PPO neural networks for discrete/discretized or a continuous action space respectively.

Pendulum-V1



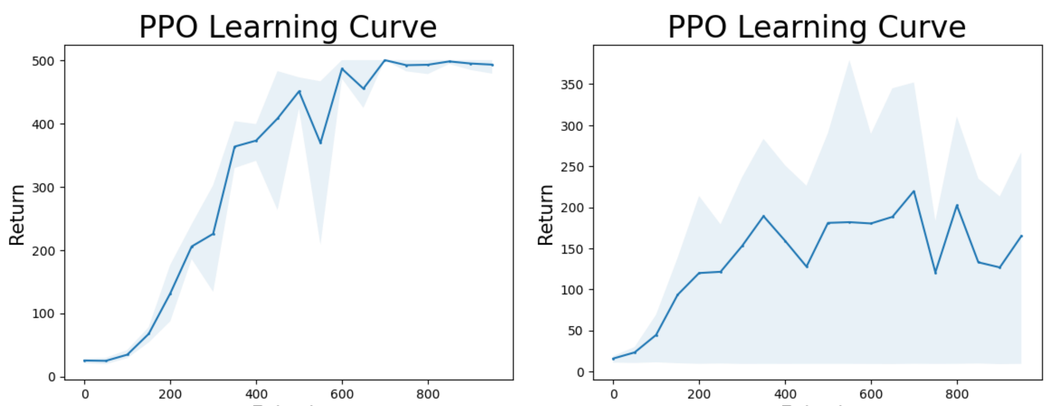
(a) Normal neural network (b) Optimized neural network (c) naction = 3 (d) naction = 21

Figure 2. PPO learning curve for the Pendulum-v1 environment with discretized action space. Evaluation is taken every 50 train episodes over 10 evaluation episodes. The plot shows the average, max and min values over three seeds.

- Continuous action space
- Faster learning rate by favoring extreme actions
- Average final performance of about -250

- Discretized action space
- Faster learning rate when less possible action values
- Average final performance of about -200

CartPole-V1



(a) good parameter (b) bad parameters

Figure 3. PPO learning curve for the CartPole-v1 environment. Evaluation is taken every 50 train episodes over 10 evaluation episodes. The plot shows the average, max and min values over three seeds.

- Discount (γ) = 0.999
- Stabilizes around 700 episodes
- Average final performance of 500

- Discount (γ) = 0.9
- Does not find optimal solution
- Average final performance of about 150

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, D. Wierstra, A. Graves, I. Antonoglou, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

DQN - Deep Q Networks

Deep Q Networks developed by Mnih et al. [1] belongs to the class of Q-learning. The idea of DQN lies in having two neural networks. To reduce learning issues due to correlated data, DQN interacts with the environment by sampling with some policy, and then storing the results in a buffer. To train the network, another process samples randomly from the buffer, and evaluates the samples with the two networks and then backward propagates the trained network. Every so many iterations, the old network gets updated to the new network's values.

Pseudocode

Algorithm 2 Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N and action-value function Q with random weights
for episode $= 1$ to M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1$ to T **do**
    With probability ϵ select a random action a_t, otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Structure of the NN

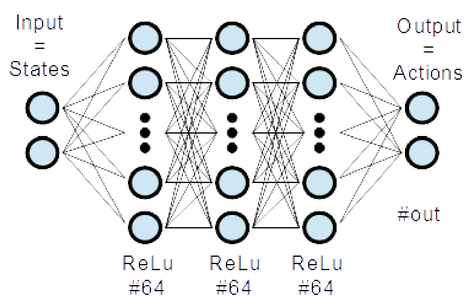


Figure 4. Architecture of the DQN neural network.

Pendulum-V1

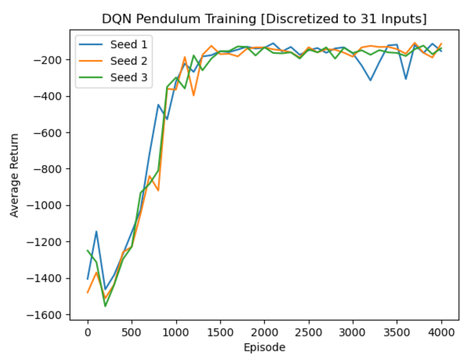


Figure 5. 3 Training iterations on Pendulum-V1

- Discretized input to 31 states
- Averages of 3 training runs
- Similar average performance
- stabilizes around 1500 episodes
- Average final performance of about [-300;-150;0] as min/avg/max
- Different discretizations led to similar performance

CartPole-V1

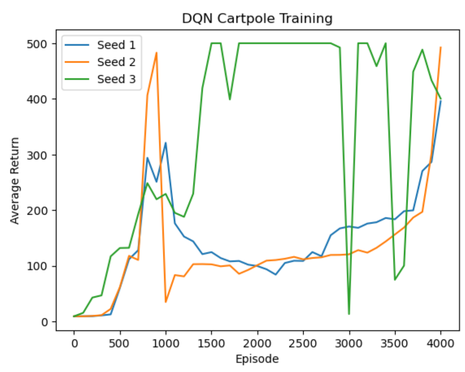


Figure 6. 3 Training iterations on Cartpole-V1

- Discrete by design
- Drops in reward probably due to not enough uncorrelated Data
- Decorrelate data by increasing buffer size and lower target network update frequency → longer training time
- Reward rescale from [0;500] to [0; 1] seems to improve training performance

Comparison

Environment	PPO	DQN	PPO discretized
CartPole-v1	0.35 s	1.8s	-
Pendulum-v1	0.36 s	1.34s	0.3 s

Table 1. Average time per episode comparison between the two algorithms.

Conclusion

PPO

- Difficult implementation
- Naturally extends to continuous action space
- Sample efficient

DQN

- Simple implementation
- Discrete action space only
- Sample inefficient