Homework 5
CS3316 Reinforce learning

Ng Tze Kean
Student number: 721370290002

May 9, 2024

A3C

Training agents through reinforcement learning can be a slow process, especially when dealing with complex environments. Traditional methods often rely on a single agent interacting with the environment, limiting exploration and hindering learning speed. The Asynchronous Advantage Actor-Critic (A3C) algorithm addresses these limitations by introducing several innovative approaches, making it a powerful technique in Deep Reinforcement Learning.

A key strength of A3C lies in its use of parallelization. Instead of a single agent, A3C employs multiple agents that explore the environment concurrently. This significantly speeds up the learning process compared to a single agent approach. Imagine a team of agents simultaneously exploring a maze; they can collectively gather information and learn the optimal path much faster than a single agent navigating alone.

Furthermore, A3C leverages a powerful combination of two neural networks: the actor and the critic. The actor network acts as the decision-maker, selecting actions for the agent within the environment. The critic network, on the other hand, plays the role of an evaluator, assessing the chosen actions and the resulting rewards or penalties. By combining these two networks, A3C gains a deeper understanding of the environment. It not only learns which actions to take, but also how valuable those actions are in the long run.

Another key feature of A3C is its use of asynchronous updates. As each agent gathers experiences, it asynchronously updates a central "global network." This allows for faster learning compared to waiting for a single agent to complete its exploration and update the network. Imagine multiple students working on a group project, each contributing their findings and updates to a central document simultaneously. This collaborative learning approach significantly reduces the overall learning time.

The combined effect of these approaches makes A3C a standout method in reinforcement learning. Parallel exploration and asynchronous updates significantly accelerate the learning process. Additionally, the use of multiple agents encourages broader exploration of the environment, reducing bias and leading to a more robust understanding of the available actions. Finally, A3C's ability to leverage distributed computing resources makes it well-suited for large-scale systems and complex environments.

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
  Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
  Receive new state s' and reward r
  $$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$
  Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ and $t \leftarrow t + 1$
  if $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

Figure 1: A3C pseudo code

DDPG

Deep Deterministic Policy Gradient (DDPG) is a powerful algorithm within Deep Reinforcement Learning, particularly adept at handling environments with continuous action spaces. Unlike

traditional methods that require the agent to strictly follow a specific policy during training, DDPG leverages a technique called off-policy learning. This allows the agent to learn from a broader range of experiences, even those not generated by the current policy, leading to more efficient learning.

A key strength of DDPG lies in its use of actor-critic methods. Similar to A3C, DDPG employs two neural networks: the actor and the critic. The actor network acts as the agent's brain, selecting actions within the environment. The critic network, on the other hand, plays the role of a reviewer, evaluating the chosen actions and their resulting rewards or penalties. This collaborative approach allows DDPG to not only learn which actions to take, but also how valuable those actions are in the long run.

To ensure stable learning during the training process, DDPG introduces a unique concept: separate target networks for both the actor and the critic. These target networks are periodically updated with the weights from the main networks. This separation helps to stabilize the learning process and prevent the networks from becoming overly sensitive to changes in the data.

Furthermore, DDPG utilizes a technique called experience replay. The agent stores past experiences in a replay buffer, allowing it to revisit and learn from a diverse set of data points during training. This not only improves learning efficiency but also helps to prevent the networks from overfitting on specific scenarios within the environment.

DDPG's ability to handle high-dimensional state spaces, where the environment can be described by many variables, and continuous action spaces, where actions have real-world consequences, makes it particularly suitable for complex tasks. It has achieved success in various applications, particularly in robotic control, where robots need to learn precise and continuous movements to perform various tasks. Additionally, DDPG excels in continuous control problems, such as controlling a car or a drone, where actions directly impact the physical world.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 2: A3C pseudo code

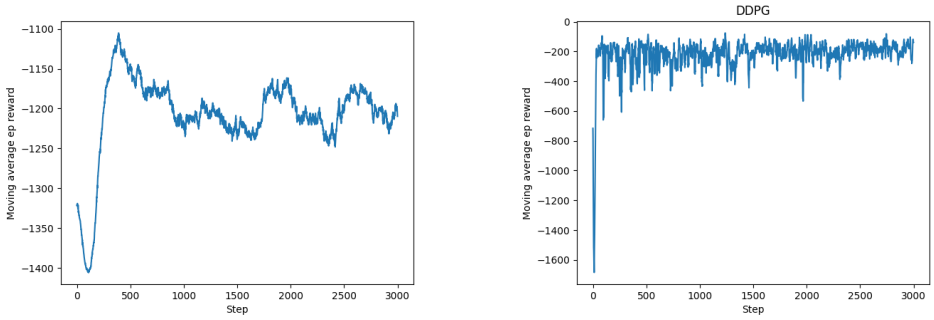
Architecture

The lab uses the Pendulum environment form OpenAI and the 2 agents are implemented in python3. We fix the other hyperparameters such as the number of hidden layers and the number of episodes. We then train the agents and compare the number of steps to reach convergence and the overall maximum reward of the agent.

We set the hidden layer for the neural network to be 64, and we try to keep all other

Results

We note that the training for A3C is much faster than DDPG. The A3C agent runs the episodes in parallel and updates the global network asynchronously. This is different from DDPG where the agent runs the episodes sequentially and updates the network after each episode. This makes A3C much faster than DDPG in terms of training. The overall time taken to train the A3C was about 1/5 the time taken to train the DDPG agent.



(a) A3C (b) DDPG

Figure 3: Results of both methods

However, as we can see from the images above the DDPG agent has a much higher reward than the A3C agent. This is because the DDPG agent is able to learn the optimal policy much better than the A3C agent. The graph of the DDPG agent is also generally more stable than the A3C agent. We can see that once the DDPG agent has converged, it is able to maintain a high reward throughout the training process. This is not the case for the A3C agent which has a more erratic reward graph.

Conclusion

Overall we can see that A3C tends to favour exploration over expliotation with its parallel exploration and asynchronous updates. This makes it faster than DDPG but it is not able to learn the optimal policy as well as DDPG. This is due to insufficient expliotation by the agent during the training. DDPG on the other hand is able to learn the optimal policy much better than A3C from its off policy learning and experience replay. But this comes at a cost of longer training time.