The idea behind Actor-Critics and how A2C and A3C improve them

Nov 17, 2018 8 mins read

The idea behind Actor-Critics and how A2C and A3C improve them

It’s time for some Reinforcement Learning. This time our main topic is Actor-Critic algorithms, which are the base behind almost every modern RL method from Proximal Policy Optimization to A3C. So, to understand all those new techniques, you should have a good grasp of what Actor-Critic are and how they work.

But don’t be in a hurry. Let’s refresh for a moment on our previous knowledge. As you may know, there are two main types of RL methods out there:

- Value Based: They try to find or approximate the optimal value function, which is a mapping between an action and a value. The higher the value, the better the action. The most famous algorithm is Q learning and all its enhancements like Deep Q Networks, Double Dueling Q Networks, etc

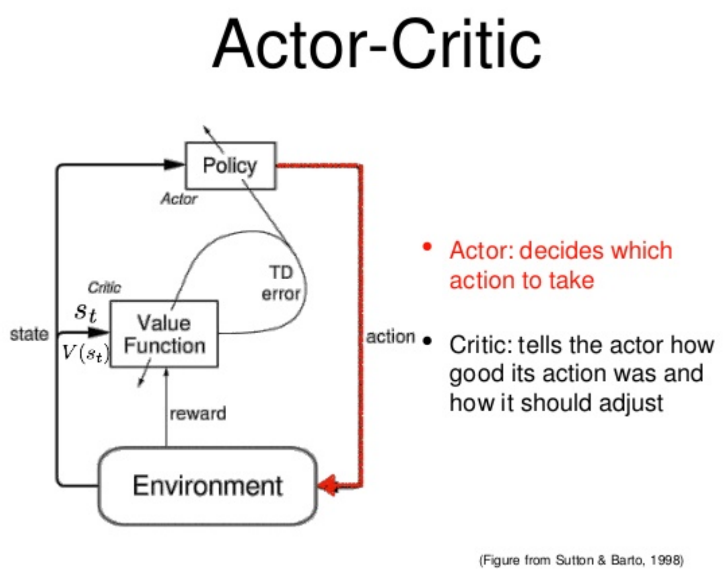
- Policy-Based: Policy-Based algorithms like Policy Gradients and REINFORCE try to find the optimal policy directly without the Q -value as a middleman.

Each method has their advantages. For example, policy-based are better for continuous and stochastic environments, have a faster convergence, while Value based are more sample efficient and steady. Check my previous posts on Reinforcement learning for more details.

When those two algorithmic families established in the scientific communities, the next obvious step is... to try to merge them. And this is how the Actor-Critic was born. Actor-Critics aim to take advantage of all the good stuff from both value-based and policy-based while eliminating all their drawbacks. And how do they do this?

The principal idea is to split the model in two: one for computing an action based on a state and another one to produce the Q values of the action.

The actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy (policy-based). The critic, on the other hand, evaluates the action by computing the value function (value based). Those two models participate in a game where they both get better in their own role as the time passes. The result is that the overall architecture will learn to play the game more efficiently than the two methods separately.


Actor-Critic
Policy
Actor
Critic
TD error
s_t
$V(s_t)$
state
Value Function
action
reward
Environment
Actor: decides which action to take
Critic: tells the actor how good its action was and how it should adjust
(Figure from Sutton & Barto, 1998)

This idea of having two models interact (or compete) with each other is getting more and more popular in the field of machine learning in the last years. Think of Generative Adversarial Networks or Variational Autoencoders for example.

But let's get back to Reinforcement Learning. A good analogy of the actor-critic is a young boy with his mother. The child (actor) constantly tries new things and exploring the environment around him. He eats its own toys, he touches the hot oven, he bangs his head in the wall (I mean why not). His mother (the critic) watches him and either criticize or compliment him. The child listen to what his mother told him and adjust his behavior. As the kid grows, he learns what actions are bad or good and he essentially learns to play the game called life. That's exactly the same way actor-critic works.

The actor can be a function approximator like a neural network and its task is to produce the best action for a given state. Of course, it can be a fully connected neural network or a convolutional or anything else. The critic is another function approximator, which receives as input the environment and the action by the actor, concatenates them and output the action value (Q-value) for the given pair. Let me remind you for a sec that the Q value is essentially the maximum future reward.

The training of the two networks is performed separately and it uses gradient ascent (to find the global maximum and not the minimum) to update both their weights. As time passes, the actor is learning to produce better and better actions (he is starting to learn the policy) and the critic is getting better and better at evaluating those actions. It is important to notice that the update of the weights happen at each step (TD Learning) and not at the end of the episode, opposed to policy gradients.

Actor critics have proven able to learn big, complex environments and they have used in lots of famous 2d and 3d games, such as Doom, Super Mario, and others.

Are you tired? Because I now start getting excited and I plan on keep going. It's a really good opportunity to talk about two very popular improvements of Actor-critic models, A2C and A3C.

Advantage Actor-Critic (A2C)

What is Advantage? Q values can, in fact, be decomposed into two pieces: the state Value function V(s) and the advantage value A(s, a):

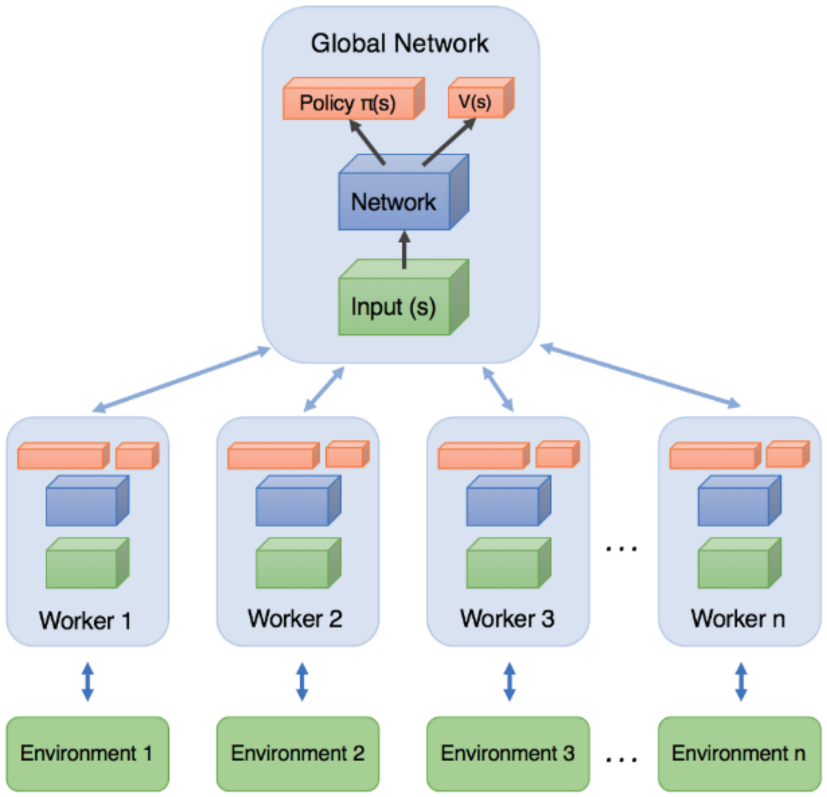
Q(s, a)= V(s)+ A(s,a) => A(s,a) =Q(s,a) -V(s) => A(s,a)= r+ γV(s_hat) -V(s)

Advantage function captures how better an action is compared to the others at a given state, while as we know the value function captures how good it is to be at this state.

You guess where this is going,right? Instead of having the critic to learn the Q values, we make him learn the Advantage values. That way the evaluation of an action is based not only on how good the action is, but also how much better it can be. The advantage of the advantage function (see what I did here?) is that it reduces the high variance of policy networks and stabilize the model.

Asynchronous Advantage Actor-Critic (A3C)

A3C's released by DeepMind in 2016 and make a splash in the scientific community. It's simplicity, robustness, speed and the achievement of higher scores in standard RL tasks made policy gradients and DQN obsolete. The key difference from A2C is the Asynchronous part. A3C consists of multiple independent agents(networks) with their own weights, who interact with a different copy of the environment in parallel. Thus, they can explore a bigger part of the state-action space in much less time.



The agents (or workers) are trained in parallel and update periodically a global network, which holds shared parameters. The updates are not happening simultaneously and that's where the asynchronous comes from. After each update, the agents resets their parameters to those of the global network and continue their independent exploration and training for n steps until they update themselves again.

We see that the information flows not only from the agents to the global network but also between agents as each agent resets his weights by the global network, which has the information of all the other agents. Smart right?

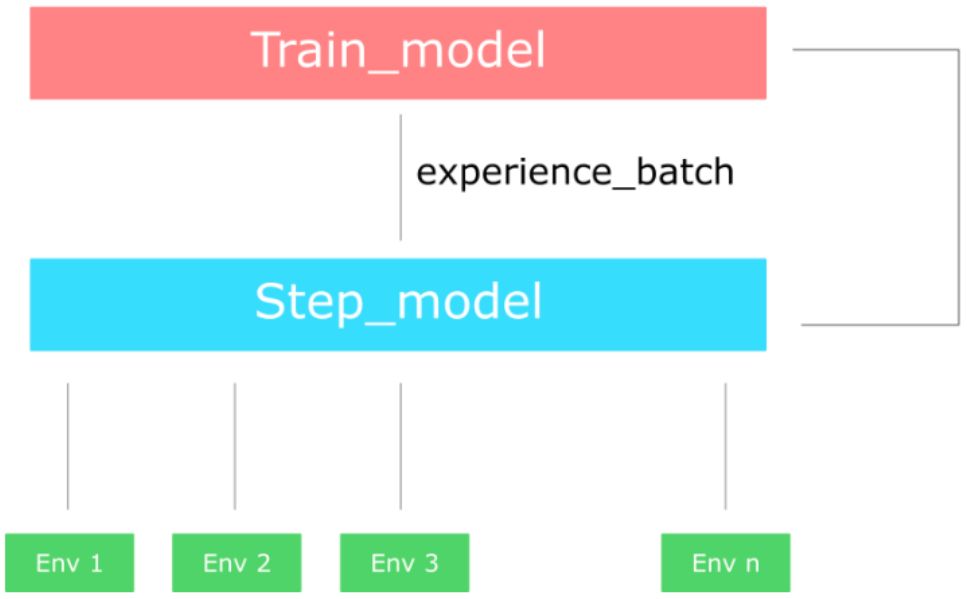
Back to A2C

The main drawback of asynchrony is that some agents will be playing with an older version of the parameters. Of course, the update may not happen asynchronously but at the same time. In that case, we have an improved version of A2C with multiple agents instead of one. A2C will wait for all the agents to finish their segment and then update the global network weights and reset all the agents.

But. There is always a but. Some argue that there is no need to have many agents if they are synchronous, as they essentially are not different at all. And I agree. In fact, what we do, is to create multiple versions of the environment and just two networks.

The first network (usually referred to as step model) interacts with all the environments for n time steps in parallel and outputs a batch of experiences. With those experience, we train the second network (train model) and we update the step model with the new weights. And we repeat the process.

If you are confused by the difference of A2C and A3C check out this Reddit post



https://medium.freecodecamp.org/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d

I tried to give you an intuitive explanation behind all these techniques without using much math and code, as things would be more complicated. However, they are not difficult models to implement as they rely on the same ideas as Policy Gradients and Deep Q Networks. If you want to build your own actor-critic model that plays Doom check this. And I think that you should. Only by building the thing ourselves , we can truly understand all the aspects, tricks and benefits of the model.

By the way, I take this opportunity to mention the recently open sourced library by Deepmind called trfl. It exposes several useful building blocks for implementing Reinforcement Learning agent, as they claim. I will try and come back to you for more details.

The idea of combining policy and value based method is now ,in 2018, considered standard for solving reinforcement learning problems. Most modern algorithms rely on actor-critics and expand this basic idea into more sophisticated and complex techniques. Some examples are : Deep Deterministic Policy Gradients(DDPG),Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO).

But don't be impatient. We'll cover them in time...

Share on:

Previous
Policy Gradients

Next
Trust Region and Proximal p...