Advantage Actor-Critic (A2C)

Edward Young - ey245@cam.ac.uk

Quick facts

1. A2C is an *policy gradient method*. It learns a policy (an *actor*) and a value function (a *critic*).
2. A2C is *on-policy*.
3. A2C is *model-free*.
4. A2C is applicable to any kind of state space and action space.

High-level strategy

Much like DDPG, A2C learns both a value network and a policy network. The value network is used to form **advantage estimates**. These advantage estimates are then used to critique action choices, allowing us to train the policy network. There are many different ways to use a value network to estimate advantage - we will cover a few of them here. The focus of today's tutorial is on general principles rather than a specific algorithm.

Advantage estimation

Consider a learned (stochastic) policy $\pi(a|s;\theta)$, with state-value function $V_\pi(s)$ and action-value function $Q_\pi(s,a)$. The **advantage function** is defined by

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{1}$$

The advantage function measures the value of an action a taken in state s over and above the value of the state itself. Note that these are *not* optimal value functions, but value functions for the policy. A *positive* advantage indicates that an action is *better* than the average action in that state, and a *negative* advantage indicates that it is *worse*.

There are multiple ways to estimate advantage using a learned value function. We will discuss just a few of them here. First, suppose that we learn an approximate *action*-value function $Q(s,a;\phi)$.

- *Action-value minus expected action-value*: $\hat{A}_t = Q(s_t,a_t;\phi) - \sum_a Q(s_t,a;\phi)\pi(a|s_t;\theta)$

Second, suppose that we learn an approximate *state*-value function $V(s;\phi)$.

- *Centred return*: $\hat{A}_t = G_t - V(s_t;\phi) = \sum_{k\geq 1}\gamma^{k-1}r_{t+k} - V(s_t;\phi)$
- *Temporal difference (TD) error*: $\hat{A}_t = \delta_t = r_{t+1} + \gamma V(s_{t+1};\phi) - V(s_t;\phi)$
- *Generalised Advantage Estimation (GAE) with parameter* λ: $\hat{A}_t = \sum_{k\geq 0}(\lambda\gamma)^k\delta_{t+k}$

Bootstrapping introduces *bias* into the advantage estimate, because our state-value function is only an approximation. However, it reduces *variance* because it reduces the need to sample. Accordingly, the centred return has the lowest bias and highest variance, and the TD error has the highest bias and lowest variance. The GAE interpolates between the two, with $\lambda = 0$ giving the TD error and $\lambda = 1$ giving the centred return.

Training the policy network

We train the policy network by performing gradient *ascent* on the objective:

$$J(\theta) = \frac{1}{T}\sum_{t=t_0}^{T+t_0} \hat{A}_t \ln\left(\pi(a_t|s_t;\theta)\right) \tag{2}$$

where the sum is over T *consecutive* time steps, and the advantage estimates are treated as constants. Note that here we sample across *time* rather than across a *batch*. Ascent on this objective *increases* the (log-)probability of actions with *positive* advantage and *decreases* the (log-)probability of actions with *negative* advantage. This objective is derived from **the (stochastic) policy gradient theorem**.

Training the value network

The value network used to form the advantage estimates can be trained using the same basic method that we've used to train value networks before in DQN and DDPG, namely performing gradient *descent* on a mean squared error (MSE) loss. For concreteness, suppose we are training a state-value network $V(s;\phi)$. Then the loss is given by:

$$\mathcal{L}(\phi) = \frac{1}{T}\sum_{t=t_0}^{T+t_0} [V(s_t;\phi) - y_t]^2 \tag{3}$$

where y_t are regression targets. These can be formed analogously to the advantage estimates:

- *return*, $y_t = G_t$
- *one-step truncated return*, $y_t = r_{t+1} + \gamma V(s_{t+1};\phi^-)$
- λ-*return*, $y_t = V(s_t;\phi^-) + \sum_{k\geq 0}(\lambda\gamma)^k \delta^-_{t+k}$ where $\delta^-_\tau = r_{\tau+1} + \gamma V(s_{\tau+1};\phi^-) - V(s_\tau;\phi^-)$

where $V(s;\phi^-)$ is a **target network** which uses a *lagged* set of parameters. These parameters could be updated with **Polyak averaging** or **synchronisation**. We might also use the target network to form the advantage estimates.

Trick: Parallel agents

With DQN and DDPG we used a memory replay buffer to decorrelate experience. A2C is *on-policy*, so all datapoints must be generated using the current policy. One way we can have decorrelated experience that is drawn from a single policy is by running N independent agents in parallel. Each agent interacts with their own independent copy of the environment. We can then average the objective (2) and the loss (3) across the trajectories pursued by different agents.

Conclusion: Putting it all together

Here are the key ideas behind Advantage Actor-Critic Methods:

1. Gather experience by letting N agents each interact with their own independent copy of the environment.
2. Every T time steps, form advantage estimates and regression targets using the learned value network (or target value network).
3. Aggregate experience across agents and time to form the value network loss (3) and policy network objective (2).
4. Perform gradient descent on the loss and ascent on the objective, and use Polyak averaging to update the parameters of the target network (or synchronisation).