# 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.

We start by interacting with the environment for $K$ trajectories, with $T_k$ time steps in the $k$-th trajectory. Let $(s_{t,k}, a_{t,k})$ be the state-action pair for the $t$-th time step on the $k$-th trajectory. This gives us $T_{\text{tot}} = \sum_{k=1}^{K} T_k$ time steps in this batch of data.

## Training the value network

The value network is used to form **advantage estimates**. The value network 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**. Consider training a state-value network $V(s;\phi)$. The loss is given by:

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_{(t,k)\in B} \left[V(s_{t,k};\phi) - y_{t,k}\right]^2 \tag{1}$$

The sum is over a **mini-batch** of $N$ data points, $B$. Here $y_{t,k}$ is the regression target for the $t$-th time step on the $k$-th trajectory. There are many ways to form regression targets - we give some here:

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

where $\phi^-$ is a *lagged* set of parameters. We train this loss for *multiple* (stochastic)-gradient steps, by sampling mini-batches $B$. After we have completed the training, we **synchronise** the target network parameters, $\phi^- \leftarrow \phi$.

## Training the policy network

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

$$J(\theta) = \frac{1}{K} \sum_{k=1}^{K} \sum_{t=0}^{T_k} \hat{A}_{t,k} \ln\left(\pi(a_{t,k}|s_{t,k};\theta)\right) \tag{2}$$

Where $\hat{A}_{t,k}$ is the **advantage** estimate for the $t$-th time step on the $k$-th trajectory. 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**. We perform only a single gradient step (per batch of trajectories) on this objective to update the policy network. This is why we use the entire batch to form the objective, rather than sample mini-batches. It is also why A2C tends to be quite slow and sample-inefficient.

## 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

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{3}$$

The advantage function measures the value of an action $a$ taken in state $s$ compared to the value of the state itself. Note that these are *not* optimal values, but values for the current policy (hence making A2C *on-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 many 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 $V(s;\phi)$ (as discussed above).

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

Bootstrapping introduces *bias* into the advantage estimate, because our state-value function is only an approximation. However, it reduces *variance* because it reduces the extent of sampling. 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.

## Conclusion: Putting it all together

The Advantage Actor-Critic algorithm loops through the following steps:

1. Gather $K$ trajectories of experience to form our current batch.
2. Update the value network:
   (a) Form the **regression targets** for each data point, using any of the methods discussed above.
   (b) Train the value network by performing multiple *mini-batch SGD* steps on the **mean squared error (MSE) loss**, (1)
3. Update the policy network:
   (a) Form the **advantage estimates** for each data point, using any of the methods discussed above.
   (b) Train the policy network by performing a single gradient step on the **policy objective**, (2)