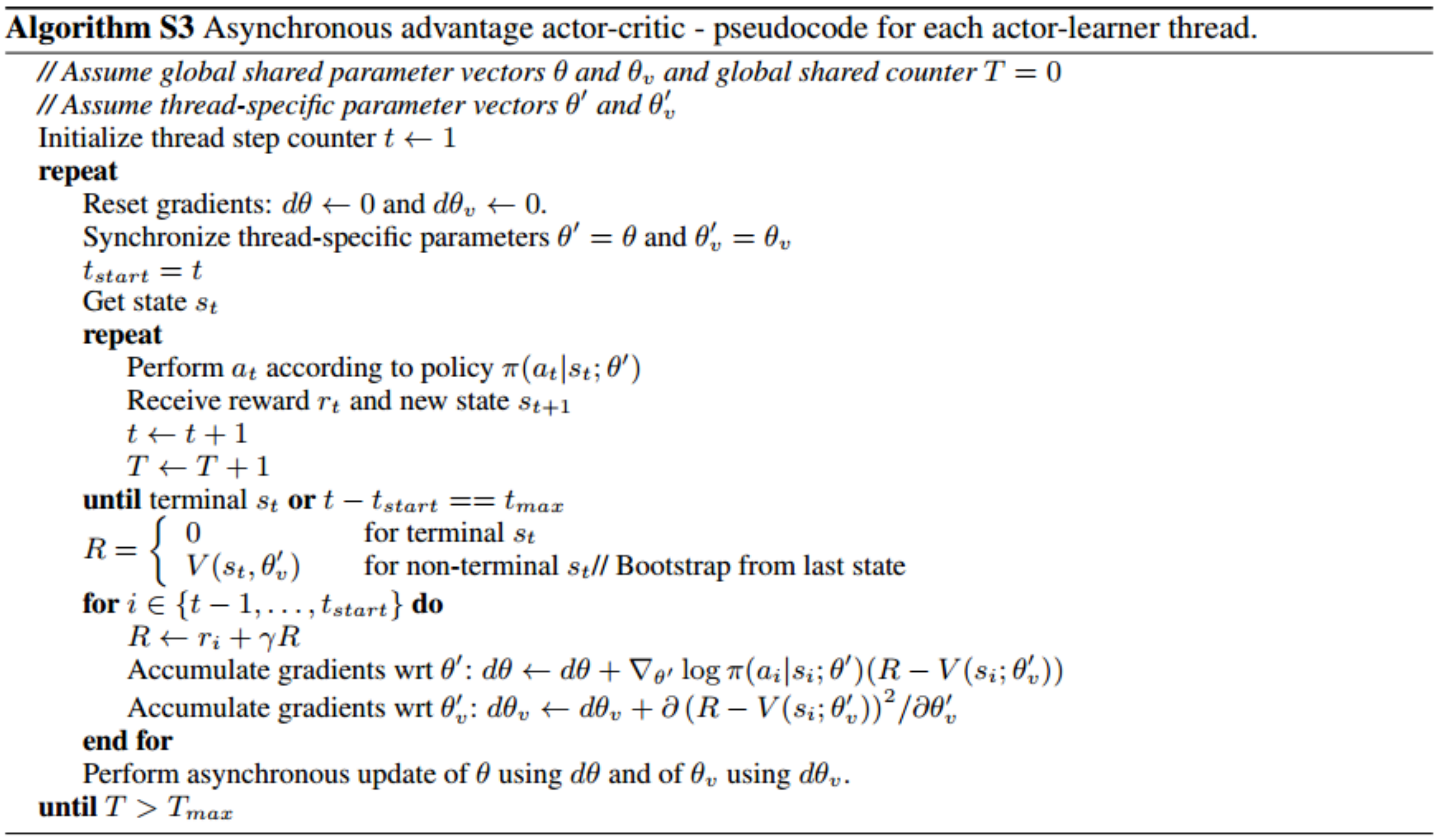
Explain to me what the RL algorithm A3C/A2C is and how it works



Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$
// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal $s_t$ or $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
until $T > T_{max}$
```

Introduction

It was previously thought that combining RL algorithms with deep neural networks were unstable. Some improvements were proposed that have to do with the observed data encountered by the online RL agent. The trajectories were highly correlated. Using experience replay memory the data can be batched and randomly sampled, thus reducing correlation. However, this required the use of off-policy algorithms.

A3C instead of using experience replay, uses multiple agents in parallel on multiple instances of the environment. This de-correlates the agent's data.

In the General Reinforcement Learning Architecture (Gorila) asynchronous training is performed in a distributed setting. Each process contains an actor that acts in its own copy of the environment and a separate replay memory. A learner samples data from the replay memory and computes the gradients of the DQN loss. The gradients are asynchronously sent to a a central server which updates the copy of its model. The updated policy parameters are sent to the learners at fixed intervals.

Background

We consider the standard RL setting where an agent interacts with an environment $\mathcal{E}$ over a discrete time steps. At each step t the agent receives a state s_t and perorms an action a_t (from a set of possible actions $\mathcal{A}$) according to a policy π (π is a mapping from states s_t to actions a_t). Then the agent receives the next state s_{t+1} and a reward r_t. This continues until the process terminates (reaches the final state).

$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

R_t is the accumulative reward from time step t until the terminal state. Where $\gamma \in (0, 1]$ is the discount factor.

The goal is to maximize the expected return from each state s_t.

The action value $Q^\pi(s, a) = \mathbb{E}\left[R_t | s_t = s, a\right]$ is the expected return from state s and and selecting action a by following polcicy π. The optimal value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ is the best action value over all possible policies.

Also the value of state s under policy π is defined by $V^\pi(s) = \mathbb{E}\left[R_t | s_t = s\right]$ and it is the expected reward from state t over any possible followed action following policy π from state s.

We represent the action value function using a function approximator (a deep Neural net).

$Q(s, a; \theta)$ where θ are the parameters of the approximator function.

The updates to θ can be derived from many RL algorithms. One of them can be Q-learning which aims to directly approximate the action value function ($Q^*(s, a) \approx Q(s, a; \theta)$)

One step Q-Learning:

i-th loss:

$L_i\left(\theta_i\right) = \mathbb{E}\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) - Q\left(s, a; \theta_i\right)\right)^2$

we aim to minimize the loss iteratively.

(s' is encountered after s)

We call it **one step** Q learning because it updates the action value funcation $Q(s,a)$ *towards* the one step return $r+\gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right)$. The drawback is that obtaining the reward r only affects directly the (s,a) pair that led to this reward. The values of the other pairs (s,a) are affected only indirectly. This will make the learning slow because there are required many updates to propagate the reward to preciding (s,a) pairs.

n**-step Q-learning**:

The action value function is updated *towards* the n-step reward $r_t+\gamma r_{t+1}+\cdots+\gamma^{n-1} r_{t+n-1}+\max_a \gamma^n Q\left(s_{t+n}, a\right)$

$$L_i\left(\theta_i\right)=\mathbb{E}\left(r_t+\gamma r_{t+1}+\cdots+\gamma^{n-1} r_{t+n-1}+\max_{a'} \gamma^n Q\left(s_{t+n}, a'\right)-Q\left(s, a ; \theta_i\right)\right)^2$$

So, a single reward directly affects the values of n-preceding (s,a) pairs making the training more efficient.

Policy based model-free RL methods directly parameterize the policy $\pi(a|s;\theta)$ and update the parameters θ by performing gradient ascent on $\mathbb{E}\left[R_t\right]$. For example the REINFORCE algorithm.

REINFORCE updates θ in the direction $\nabla_\theta \log \pi\left(a_t|s_t ; \theta\right) R_t$ which is an **unbiased** estimate of $\mathbb{E}\left[R_t\right]$.

We can reduce the variance of the estimate while keeping the bias low by subtracting it from $b_t\left(s_t\right)$ -a baseline-:

$$\nabla_\theta \log \pi\left(a_t|s_t ; \theta\right)\left(R_t-b_t\left(s_t\right)\right)$$

We commonly use the value function as the baseline:

$$b_t\left(s_t\right) \approx V^\pi\left(s_t\right)$$

When we use an approximation to calculate the value function $\left(R_t-b_t\right)$ is used to scale the policy gradient and can be seen as an estimate of the advantage of a_t at s_t:'Y

$$A\left(a_t, s_t\right)=Q\left(a_t, s_t\right)-V\left(s_t\right)$$

R_t can be seen as an estimate of Q and b_t as an estimate of the value function.

Actor critic method: Here, the actor can be seen seen as the policy and the baseline b_t as the critic.