Exercise 8
Actor-Critic Methods

1 Advantage Actor Critic (A2C)

Remember that the policy gradient is defined as:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s)\, Q^\pi(s,a)].$$

The most basic **actor-critic** framework uses a critic to approximate the action-value function $Q_\pi(s,a)$ and applies its action-value estimates for calculating the policy gradient. In this exercise we will go one step further.

Remember that, in order to reduce variance in the estimates of the gradient, it is mathematically sound to subtract a baseline term from the policy gradient. It turns out that a reasonable choice for the baseline term is the state-value function $V^\pi(x)$, which leads to the policy gradient formulation:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s)\,(Q^\pi(s,a) - V^\pi(s))]\\ &= \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s)\, A^\pi(s,a)],\end{aligned}$$

where $A^\pi(s,a)$ is the so-called *advantage function*. This function can be interpreted as the difference in expected return when taking action a in state s compared to the expected return when following the policy π in state s. Policy gradient methods that use this advantage function in its gradient calculations belong to the family of **advantage actor-critic** methods.

Instead of approximating the two functions $Q^\pi(s,a)$ and $V^\pi(s)$ or the advantage function $A^\pi(s,a)$ directly, common approaches use the critic to approximate the state-value function $V^\pi(s)$ and estimate the advantage in one of the following ways:

1. $A^\pi(s,a) = R(s,a) - V^\pi(x)$: MC advantage estimate, where $R(s,a)$ is the return received in state s
2. $A^\pi(s,a) = r + \gamma V^\pi(s') - V^\pi(s)$: TD advantage estimate, often also in the form of a n-step TD formulation
3. $A^\pi(s,a) = \sum_{l=0}^{T} (\gamma\lambda)^l \delta_{t+l}$: Generalized advantage estimate (GAE) [1], where $\delta_t = r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$ is the TD error. GAE is the average of all n-step TD advantages weighted by a discount parameter λ.

In this exercise we will implement the advantage actor-critic method and focus on the 1. and 3. estimation approach. The VPG implementation of last exercise can be naturally extended for this.

Another benefit when using a critic is that we can change the training loop to optimize the network parameters after a fixed amount of steps performed in the environment (as compared to a fixed amount of episodes in the VPG exercise), i.e. on partial trajectories. We can do this, because we can bootstrap the value of the state the trajectory was terminated in using the critic.

Programming Tasks:

1. **Network Architecture** Similar to the last exercises it makes sense to begin with defining the neural network architecture inside `CriticNetwork`. As state values can have positive or negative sign the output layer has to be `torch.nn.Linear`.
2. **Transition Memory** Think about what additional information we need here.

[1] Schulman et al., https://arxiv.org/abs/1506.02438

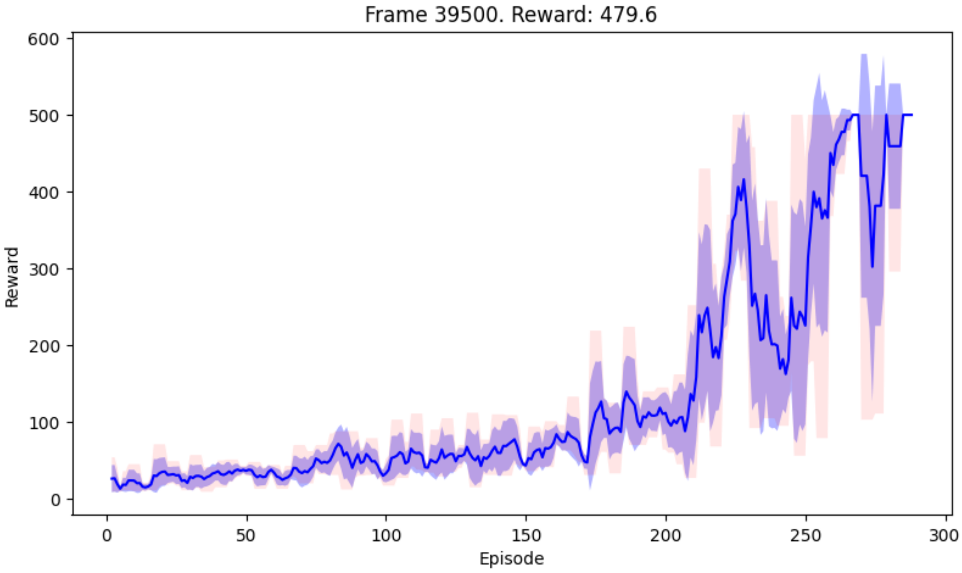

Figure 1: Sample reward plot we were able to achieve with our A2C implementation on CartPole (hyperparameters as provided in skeleton). The results are strongly hyperparameter-related, but on average you should observe an faster convergence compared to VPG.

3. **Predict Function** Expand the function to also calculate and return the value of the current observation to the training loop.
4. **MC Advantage** Write the function `compute_advantages` which estimates the MC advantage based on a list of returns and state values.
5. **Critic Loss** Write the function `calc_critic_loss`, which computes the MC error between predicted state values and returns. *Note:* This is just for simplicity. You could also use 1-step or n-step TD error here.
6. **Actor Loss** Adjust the `calc_actor_loss` function to work on advantages instead of action-value functions. Do you have to change anything?
7. **Training Loop**
   (a) As in the last exercise, sample an action from your actor, take a step inside the environment and save the transition inside the `TransitionMemory`
   (b) Write code for the case that a terminal state has been reached (*Hint:* We don't only optimize when an episode is finished in this exercise)
   (c) Write the optimization code, which should be executed when enough samples were collected to fill the training batch (compare with `self.batch_size`). You will have to call `finish_trajectory`, but this time provide the termination state value using the critic (don't forget to call `item()`, we don't want to backpropagate from these outputs). Next, calculate actor and critic loss using the function you wrote, and do the optimization step using the actor and critic optimizers.
8. **GAE** At this point your implementation should already work and be able to learn useful policies. Now as a last step, write the `compute_generalized_advantages` function and use it instead of the old MC method to estimate the state advantage function.