(a) without baseline

(b) with baseline b = 20

(c) with normalized discounted rewards

Figure 1: Training performance of policy gradient for the Cartpole environment with different baselines

1 Task 1 : Implement policy gradient for the Cartpole environment with continuous action space.

The implemented code is uploaded to the submission folder as "agent.py". Figures 1a, 1b and 1c illustrate the training process of the following three REINFORCE methods:

- **a) basic REINFORCE without baseline**
- **b) REINFORCE with a constant baseline b = 20**
- **c) REINFORCE with discounted rewards normalized to zero mean and unit variance**

Question 1 — How does the baseline affect the training? Why?

Subtracting a baseline that does not depend on the action, from the policy gradient can reduce variance, without changing the expectation of the update. Because if the baseline does not depend on the action, the following will hold:

$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)B(s)] = 0$

As it can be seen in the figure 1b subtracting a baseline improves the performance, although it is not the optimal baseline.

2 Task 2 : Implement two cases of adjusting variance during training

Figures 2a and 2b shows the training process of policy gradient with adaptive variance during training using two adjusting methods:

(a) exponentially decaying variance

(b) variance learned as a parameter of the network

Figure 2: Training performance of policy gradient for the Cartpole environment with different adjusting variances

- **a) exponentially decaying variance** $\sigma = \sigma_0 e^{-c.k}$
- **b) variance learned as a parameter of the network**

Question 2 - Compare using a constant variance, as in Task 1, to using exponentially decaying variance and to learning variance during training.

However, since the REINFORCE method tends to have a large variance in the estimation of the gradient directions, its naive implementation converges slowly. Subtraction of the optimal baseline can ease this problem to some extent, but the variance of gradient estimates is still large.

Question 2.1 - What are the strong and weak sides of each of those approaches?

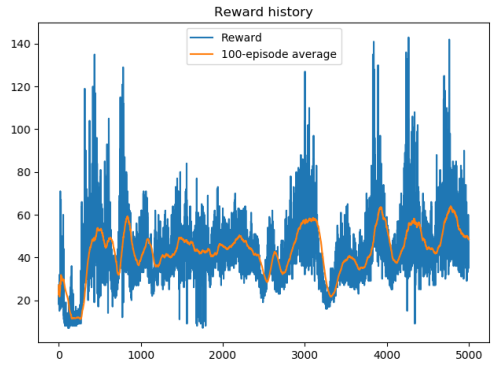
Both of these methods are used for decreasing the variance in the estimation of the gradient directions. Finding the optimal baseline in the first task is not straightforward. Also, in the task 2, the performance heavily depends on the choice of an initial policy, and appropriate initialization is not straightforward in practice.

Question 2.2 - In case of learned variance, what's the impact of initialization on the training performance?

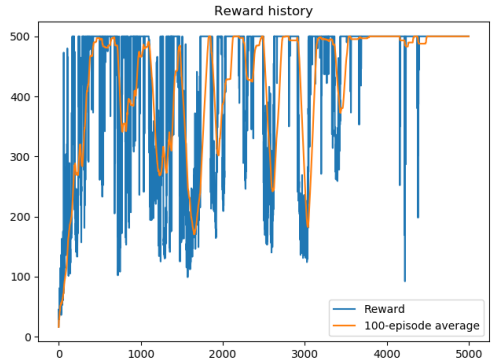
Figures 3a and 3b show the impact of initialization on the training performance of the case of learned variance with two different initialization values (1 and 5). As it can be seen by decreasing the initialize value of σ the performance will decrease. The reason could be by decreasing the initial variance of σ the agent will not explore enough.

Question 2.3 - Which takes longer to train?

In my experiences, the adjusting variance methods in task 2 took much longer time to train. Generally the REINFORCE with Baseline algorithm learns faster as a result of the reduced variance of the algorithm.



(a) $\sigma_0 = 1$



(b) $\sigma_0 = 5$

and 3b

Figure 3: Training performance with different initial values in the case of learned variance

Question 3 - Could policy gradient methods be used with experience replay? Why/why not?

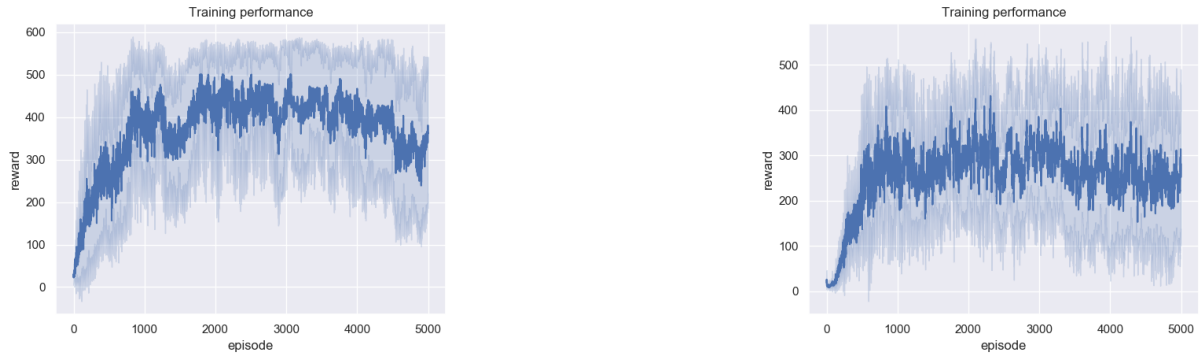
Both REINFORCE and the version of actor-critic method implemented in this exercise are on-policy: training samples are collected according to the policy that we try to optimize for (the target policy). Experience replay can be used in off-policy methods which do not require full trajectories and can reuse any past episodes ("experience replay") for much better sample efficiency. using that, the sample collection follows a behavior policy different from the target policy, bringing better exploration. There are some variant of actor-critic methods which are off-policy and can use experience replay to improve sample efficiency.

Question 4 - Can policy gradient methods be used with discrete action spaces? Why/why not? Which steps of the algorithm would be problematic to perform, if any?

The main advantage of policy gradient methods is that they are more effective in high dimensional action spaces, or when using continuous actions. However, this methods can be used for either discrete or continuous action spaces. In discrete action spaces, the algorithm will output a probability distribution over action, which means that the activation function of the output layer is a softmax. For exploration-exploitation, it samples from the actions based on their probabilities. Actions with higher probabilities have more chances to be selected.

3 Task 3 : Revisit the policy gradient solution for the continuous Cartpole from Task 1 and implement the actor-critic algorithm.

The implemented code is saved and uploaded in "ac_agent.py" and "ac_cartpole.py".



(a) updates at the end of each episode

(b) updates every 10 timesteps

Figure 4: Training performance with actor-critic methods

4 Task 4 : Update the actor-critic code to perform TD(0) updates every 10 timesteps, instead of updating your network at the end of each episode.

Figures 4a and 4b illustrate the training performance of actor-critic methods in task 3 and task 4.

In task 4, at each time step I calculated the TD error using critic value and actor value using the following equation:

$$\delta = reward + \gamma * critic\_value(new\_state) * (1 - done) + critic\_value(state) \quad (1)$$

The implemented codes are saved in "ac_agent_4.py" and "ac_cartpole_4.py". I was expecting that the training performance get better in task 4, however it gets noisier, since the gradient estimates with updates at every 10 time steps will be noisy.

Question 5 - What are the advantages of policy gradient methods compared to action-value methods such as Q-learning? When would you use Q-learning? When would you use a policy gradient method?

The first advantage of policy-based methods is that tgey have better convergence properties. The problem with value-based methods is that they can have a big oscillation while training. This is because the choice of action may change dramatically for an arbitrarily small change in the estimated action values.

On the other hand, with policy gradient, we just follow the gradient to find the best parameters. We see a smooth update of our policy at each step. Because we follow the gradient to find the best parameters, we're guaranteed to converge on a local maximum (worst case) or global maximum (best case).

The second advantage is that policy gradients are more effective in high dimensional action spaces, or when using continuous actions. The problem with action-value methods such as Q-learning is that their predictions assign a score (maximum expected future reward) for each possible action, at each time step, given the current state. This works well when we have a finite set of actions. But what if we have an infinite possibility of actions?

A third advantage is that policy gradient can learn a stochastic policy, while value functions can't. Therefore, in policy gradient we don't need to implement an exploration/exploitation trade off. A stochastic policy allows our agent to explore the state space without always taking the same action. This is because it outputs a probability distribution over actions. As a consequence, it handles the exploration/exploitation trade off without hard coding it. We also get rid of the problem of perceptual aliasing. Perceptual aliasing is when we have two states that seem to be (or actually are) the same, but need different actions.

Policy gradients have one big disadvantage. A lot of the time, they converge on a local maximum rather than on the global optimum. Instead of Q-Learning, which always tries to reach the maximum, policy gradients converge slower, step by step. They can take longer to train.