Exercise 5

Tommaso Brumani - 100481325
ELEC-E8125 - Reinforcement Learning

October 24, 2022

Task 1

The REINFORCE algorithm was implemented by modifying **'agent.py'** as follows:

- In the **'Policy.init()'** function, after the 'TODOs':

```
# define log standard deviation
self.actor_logstd = torch.tensor(np.zeros(action_dim),
    device=device)
```

- In the **'Policy.forward()'** function, after the 'TODO':

```
# create action distribution
probs = Normal(action_mean, action_std)
```

- In the **'PG.update()'** function:

```
########## Your code starts here. ##########
# compute discounted rewards
disc_rewards = h.discount_rewards(rewards, self.gamma).
    unsqueeze(1)

# compute policy gradient loss for batch
loss = - torch.mean(disc_rewards * action_probs)

# backpropagate gradients
loss.backward()
self.optimizer.step()

# empty optimizer gradients
self.optimizer.zero_grad()
########## Your code ends here. ##########
```

- In the **'PG.get_action()'** function:

```
 ########## Your code starts here. ###########
# get action distribution following policy for given
    observation
action_distribution = self.policy.forward(x)

# select action
if evaluation == True:
    # mean if testing
    action = action_distribution.mean
else:
    # random sample otherwise
    action = action_distribution.sample()

# get log probability of chosen action
act_logprob = action_distribution.log_prob(action)
########## Your code ends here. ##########
```

And by modifying **'train.py'** as follows:

- In the **'train()'** function:

```
########### Your code starts here ###########
# get action and logprob
action, act_logprob = agent.get_action(obs)

# perform step and update observation
obs, reward, done, _ = env.step(to_numpy(action))

# update log prob and reward
agent.record(act_logprob, reward)

# increase sum and timestep
reward_sum += reward
timesteps += 1
########## Your codes ends here. ##########
```

Three versions of the algorithm were implemented and trained, using constant standard deviation $\sigma = 1$:

(a) Without baseline, the training performance of which is reported in Figure 1.

(b) With constant baseline $b = 20$, the training performance of which is reported in Figure 2, by modifying the **'PG.update()'** file as follows:

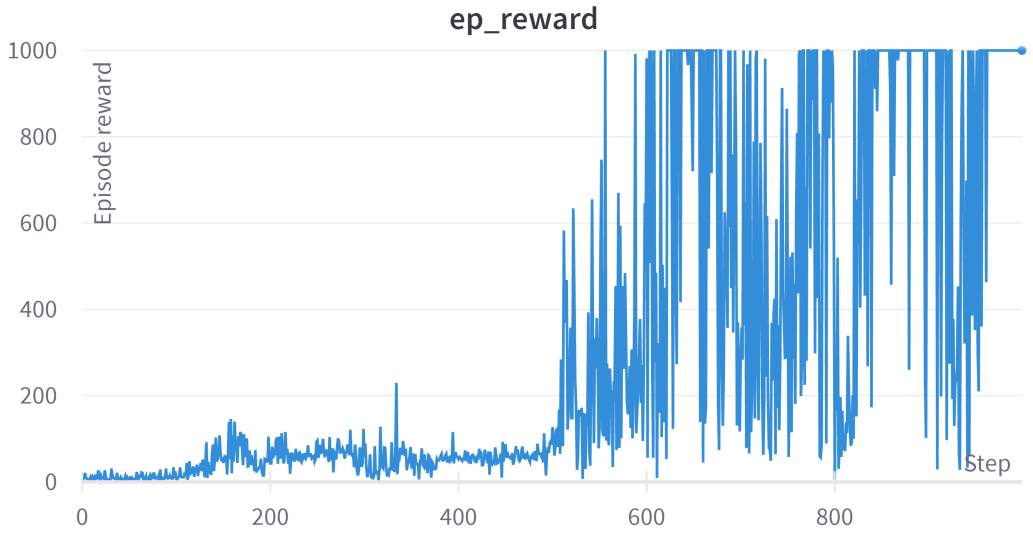

Figure 1: Training performance of REINFORCE without baseline.

```
# subtract baseline
baselined_rewards = disc_rewards - 20

# compute policy gradient loss for batch
loss = - torch.mean(baselined_rewards * action_probs)
```

(c) With discounted rewards normalized with zero mean and unit variance, the training performance of which is reported in Figure 3, by modifying the **'PG.update()'** file as follows:

```
# normalize rewards
(std, mean) = torch.std_mean(disc_rewards)
norm_rewards = (disc_rewards - mean)/std

# compute policy gradient loss for batch
loss = - torch.mean(norm_rewards * action_probs)
```

Question 1.1

A good baseline choice would be an estimate of the state value, a function of the state at time t, and a learned weight vector w. The backpropagation of cumulative discounted rewards in training means that a 'bad' action taken in an otherwise 'good' episode might result in a strong positive incentive in taking that action in further episodes, leading to a high variance in action selection. The training is rendered more stable by the use of a baseline because it reduces the variance by shifting the cumulated discounted rewards to a lower value, thereby

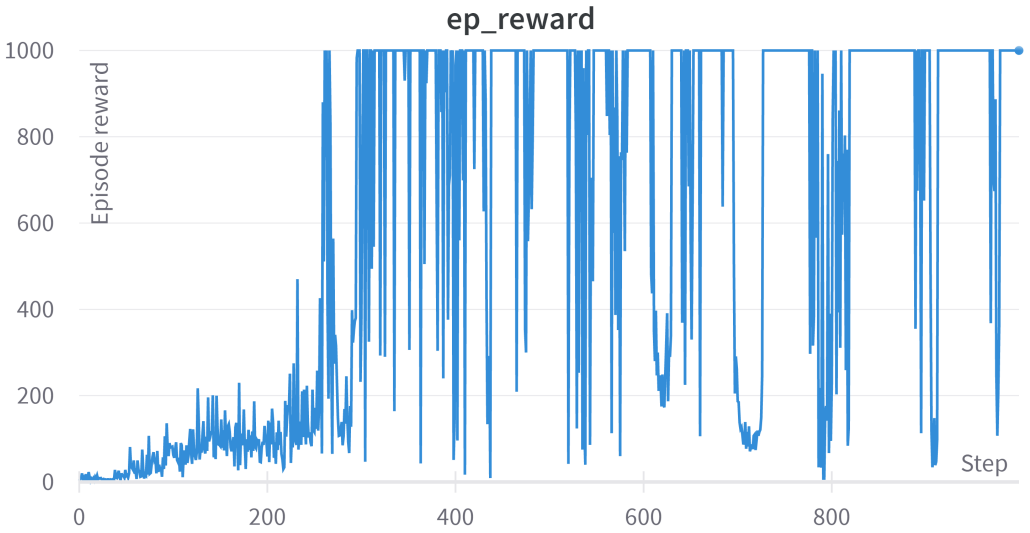

Figure 2: Training performance of REINFORCE with baseline $b = 20$.

reducing the magnitude of the gradient ascent updates, without however increasing bias (as the baseline is not a function of action a and thus its multiplication with the gradient equals 0).

Task 2

The policy's variance was implemented as a learnable parameter by modifying the **'Policy.init()'** function in the **'agent.py'** file as follows:

```
# declare log std deviation as a learnable parameter
self.actor_logstd =  torch.nn.parameter.Parameter(torch.tensor(
    np.zeros(action_dim), device=device))
```

The resulting training performance is reported in Figure 4.

Question 2.1

a) Using a constant variance during training requires no additional calculations, but maintains the randomness in the choice of the action to take constant throughout training. A poor choice of its initial value would also result in a permanently decreased performance.

b) Using a learned variance, instead, requires the learning of an additional parameter, but allows to optimize the randomness involved in the choice of actions by taking into account a smaller or greater degree of certainty over which actions are optimal.

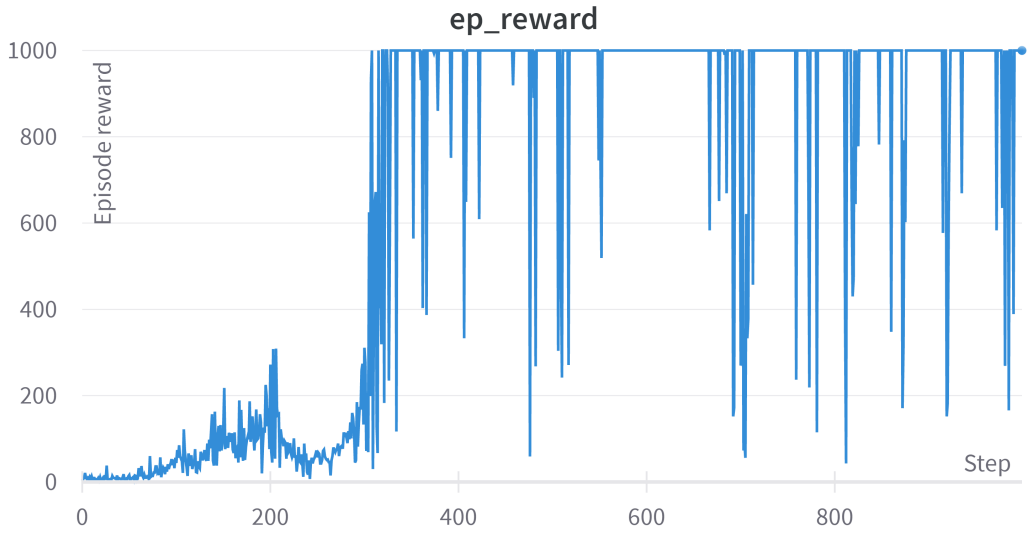

Figure 3: Training performance of REINFORCE with normalized discounted rewards.

Question 2.2

In case of learned variance, an initialization to an extremely small value might result in insufficient exploration of the action range, which might slow down or prevent learning, while an extremely high value might lead to an excessively randomize choice of actions, preventing convergence.
An appropriate choice, incorporating pre-existing knowledge on the problem, might instead lead to a faster convergence towards an optimal policy.

Question 3

Experience replay requires an Off-Policy method whereas the one implemented in this exercise is On-Policy, as the policy that is being learned by the model is the same from which the actions to be taken by the agent are drawn. Specifically, the usage of the same policy for action selection and gradient ascent improvement is problematic if we want to use experience replay. A way to solve this might be to employ two separate policies, a behavioral and target policy, for action sampling and optimization respectively.

Question 4.1

A model with an unbounded continuous action space and a reward function based solely on survival might learn to employ actions with a very high value, for example when trying to switch quickly between two courses of action. Depending on the action being taken in the real world this might have dangerous results, such as pushing machinery beyond its breaking point by asking it to exercise too much force.

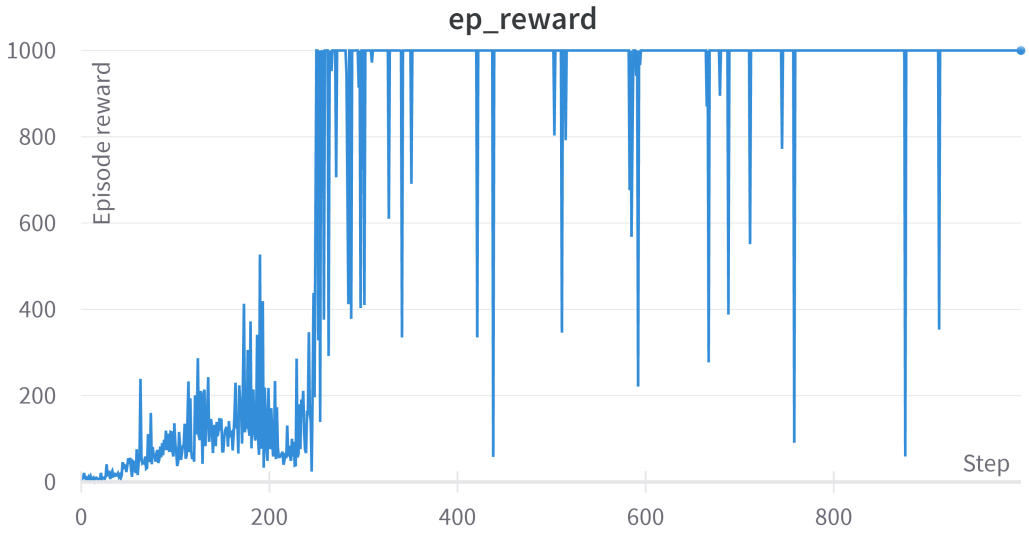

Figure 4: Training performance of REINFORCE with learnable policy variance.

Question 4.2

The problem described in the previous questions might be mitigated without employing a hard limit by, for example, penalizing high action values in the training process, as a sort of regularization technique.

Question 5

When dealing with a discrete action space, so long as the action space allows for some measure of "distance" between actions, should not be a problem: it would be sufficient to choose the action closest to the one sampled from the policy distribution, and proceed as normal. In the case of an action space with no defined topography, however, this selection might become problematic.