Reinforcement Learning
Homework Sheet 3

Lawrence Kurowski 2019280743

12 May 2020

1 PPO

We construct PPO using Tensorflow / Keras library.

The PPO model has 2 networks: actor and critic network. Actor network is the policy network and the loss function is the PPO loss, calculated as (pseudo-code):

```
ratio = exp( log(newpolicy_pi) - log(oldpolicy_pi) )

# following the formulas from the PPO paper
val_1 = ratio * advantages
val_2 = K.clip(ratio, min_value=1 - clipping_val, max_value=1 + clipping_val)
    * advantages

actor_loss = - mean(K.minimum(val_1, val_2))
```

Here we calculate advantages using the recursive formula for GAE (pseudo-code):

```
delta = rewards[i] + gamma * values[i + 1] * masks[i] - values[i]
gae = delta + gamma * lmbda * masks[i] * gae
```

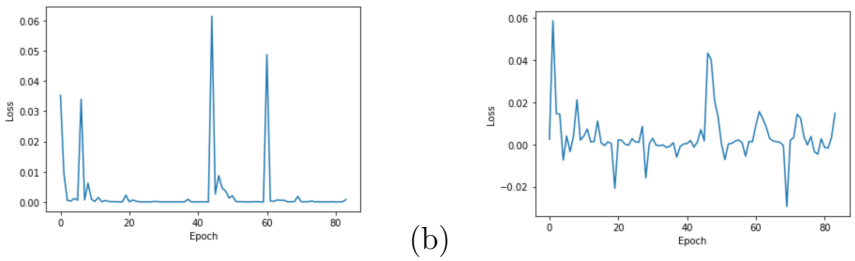
We train the PPO over 100 episodes, and let it play until "done" and for max. 100 steps inside each episode.

In each episode, after experienced is collected, the actor and critic nets are updated and environment reset.

Early-stopping is introduced when the system starts scoring positive reward (goal).

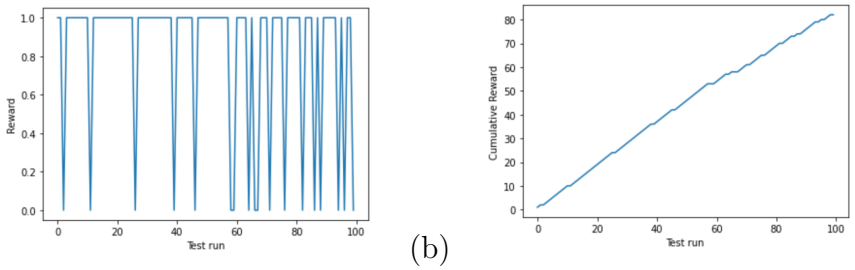
We show the value (critic) and policy (actor) loss plot as function of training episodes in figure 1 (a) and (b), respectively.

After training is completed, we run the game for 100 episodes, each time allowing for a max. of 100 "moves" before terminating the game. We record rewards and plot in figure 2 (a). The cumulative rewards over 100 testing steps is shows in figure 2 (b).



(a) (b)

Figure 1: (a) critic (value) and (b) actor (policy) PPO model training loss.



(a) (b)

Figure 2: Rewards for 100 test steps of the PPO model. (a) shows the rewards, while (b) is the cumulative reward.

2 DDPG

We construct DDPG using Tensorflow / Keras library, using similar code structure / ideas to PPO in section 1.

The DDPG model has 2 networks: actor and critic network, as well as 2 target networks actor target and critic target.

We also use the target networks to optimize the model. In particular, at each step we first update the prediction and then minimize according to the MSE loss (pseudo-code):

```
# generate critic preds
Qvals = critic.predict([next_state, action])
# generate actor-derived predictions for next action
next_actions = actor_target.predict([next_state])
# Q updates
next_Q = critic_target.predict([np.array(next_state_batch) ,
    np.array(next_actions)])
Qprime = (reward_batch) + gamma * next_Q

# MSE loss
loss = mean_squared_error(Qprime, Qvals)
```

Here the "batch" refers to samples from the "replay buffer", or in other words from the already played out scenarios.

While actor and critic networks are trained with back-propagation, the target networks are updated at each step according to (pseudo-code):

```
params = tau * actor_model_params + (1-tau) * target_actor_model_params
params = tau * critic_model_params + (1-tau) * target_critic_model_params
```

We train the DDPG over 100 episodes, and let it play until "done" and for max. 1000 steps inside each episode.

In each episode, after experienced is collected, the actor and critic nets are updated and environment reset.

Early-stopping is introduced when the system starts scoring positive reward (goal).

Unfortunately, our model did not yield satisfying results despite training for a long time, and eventually we ran out of time before the submission deadline. We hope that the attached file ("DDPG") explains our reasoning in how we designed the code.