Deep RL Arm Manipulation

高鹏

1.Rewards

Training an agent is similar to training a pet. A pet meets your request and you give it a reward. A pet makes an unsatisfactory behavior. You give a punishment. The pet is constantly thinking about how to give the right behaves to maximize rewards, not just instant rewards. Agent training is very similar to this process, and agents are constantly weighed against temporary rewards and cumulative rewards during the training process.

There are two tasks in this project:
1. Have any part of the robot arm touch the object in front of it while preventing the gripper from touching the ground.
2. Have ONLY the gripper of the robot arm touch the object in front of it while preventing the gripper from touching the ground.

For these two tasks, we must design different reward functions. For the first task, first, the agent will be penalized (namely given a negative constant) when it touches the ground. The penalty value should be the smallest of the penalty values. That is, we hope that the agent can avoid touching the ground as much as possible, because it is very inappropriate for the task. We have chosen -50. At the same time, if the robot arm still has not made a decision after more than 100 frames, we will also Give the same penalty. Because this is also an unreasonable situation. Then, for the arm part of the arm that touches the specified object, the corresponding reward constant and penalty constant are given respectively. Since there are only two situations in the first task (the arm touches or does not touch the specified object), only the corresponding constant value is given here. (The reward is 50, and the penalty is -50.) For the second task, the agent is in touch. No decisions are made after hitting the ground and after more than 100 frames. Similarly, the second task requires that only the fixtures touch the object, so this is not the only two cases. In this task, the touch is specified for the arm section. If the object touches the specified object instead of the fixture, the agent will be given a smaller penalty than if the fixture touches the ground. For the case where the fixture only touches the specified object, the agent is given a very high reward value (in task two, the reward Set to 5, the main penalty is set to -1, and the minor penalty is set to -0.5).

The reward given above is only the final reward after completing a process, that is, the agent will only receive these punishments or rewards when certain conditions are met. However, in the process of a complete robotic arm touching the designated object, the agent also Need to receive a corresponding punishment or reward to promote the completion of the entire process and meet the conditions. Therefore, we still need to define a temporary reward to guide the agent to achieve the goal and meet the requirements, and this temporary reward needs to be based on the distance between the fixture in the target object Set (the distance of the fixture to the target object is calculated by the bounding box). Here, the Smoothed Moving Average (SMMA) algorithm is used to measure the amount of change in the distance target.

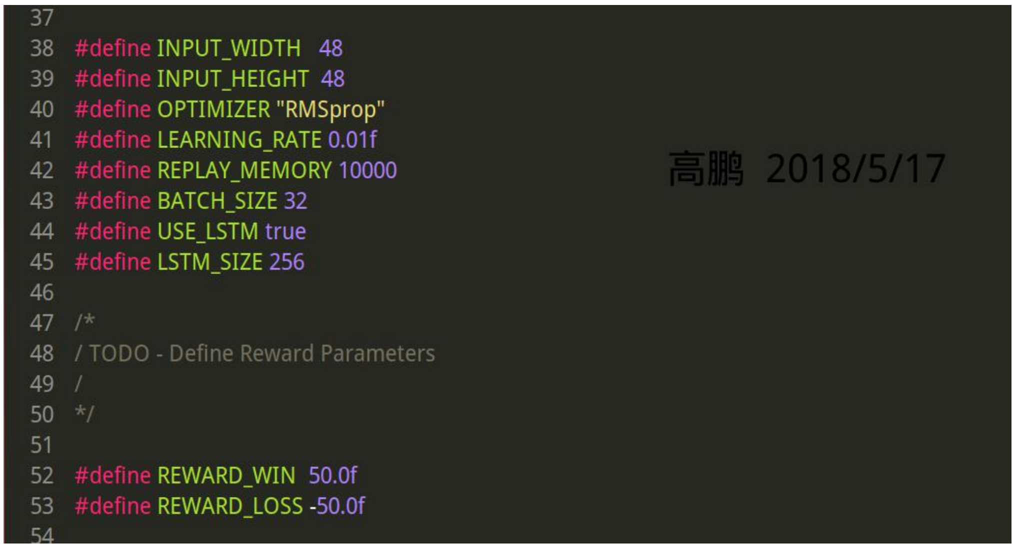
Reward functions are as follows:

average_delta = (average_delta*alpha) + (dist*(1-alpha))

By using the SMMA algorithm, the reward function can be given a proxy gradient feedback and let it know that as it gets closer to the target object the reward will continue to accumulate. This function can evaluate the fluctuation of the distance variation and enable the agent to move to maximize the cumulative reward. This also makes the agent learning rate faster. At the same time, you can increase the proportion of the current distance variation in the average line value by adjusting the alpha closer to zero. The original amount of change will not be completely removed, but only a smaller proportion, which will make the gradient feedback more smooth.

Finally, because both the final reward and the temporary reward depend on the positional relationship between the fixture and the specified object, the robotic control uses position control instead of speed control.

2.Hyperparameters



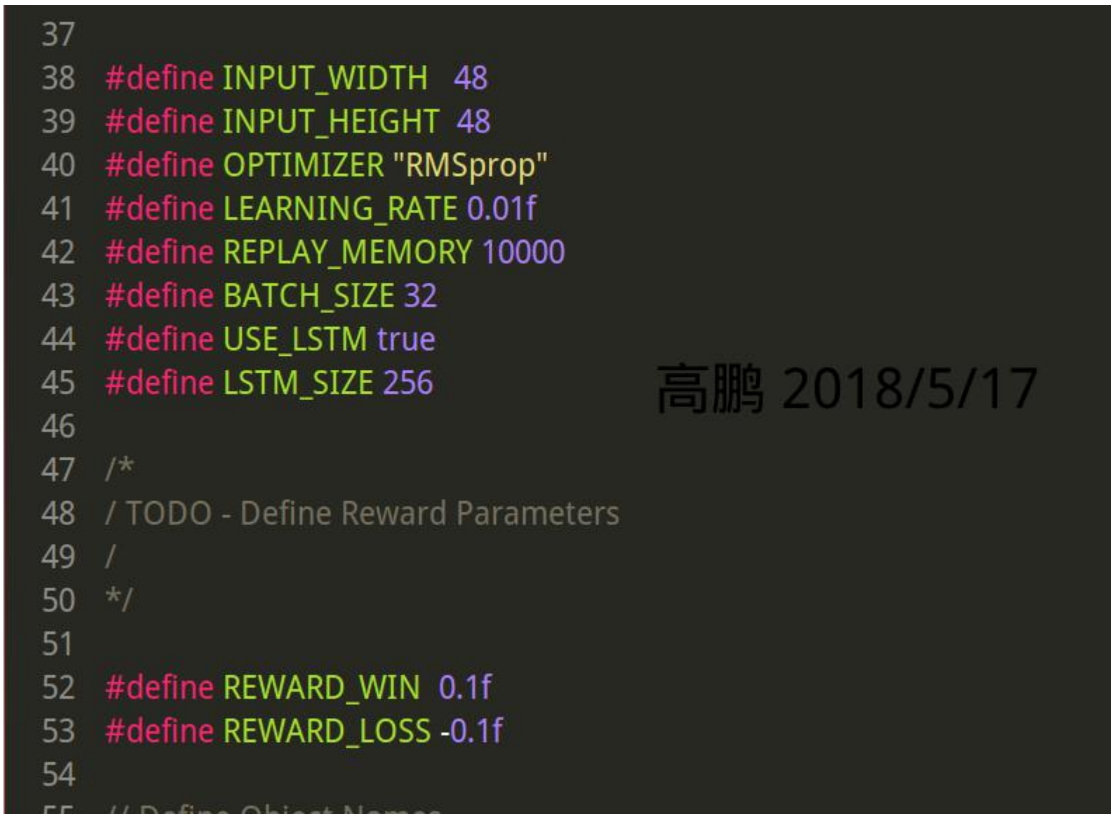
```

#define INPUT_WIDTH   48
#define INPUT_HEIGHT  48
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.01f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 32
#define USE_LSTM true
#define LSTM_SIZE 256

/*
/ TODO - Define Reward Parameters
/
*/

#define REWARD_WIN  50.0f
#define REWARD_LOSS -50.0f

```

FIgure1. Task1 Hyperparameters

In Figure 1, I show the hyperparameter settings of task 1. In task 1, in order to save computing resources, I reduce the input_size to 48, and set the batch_size to 32. This is much smaller than the previous default of 512x512. It will speed up the training. At the same time, the square image is still maintained here, which can optimize the GPU calculation, because the square matrix calculation complexity is lower.

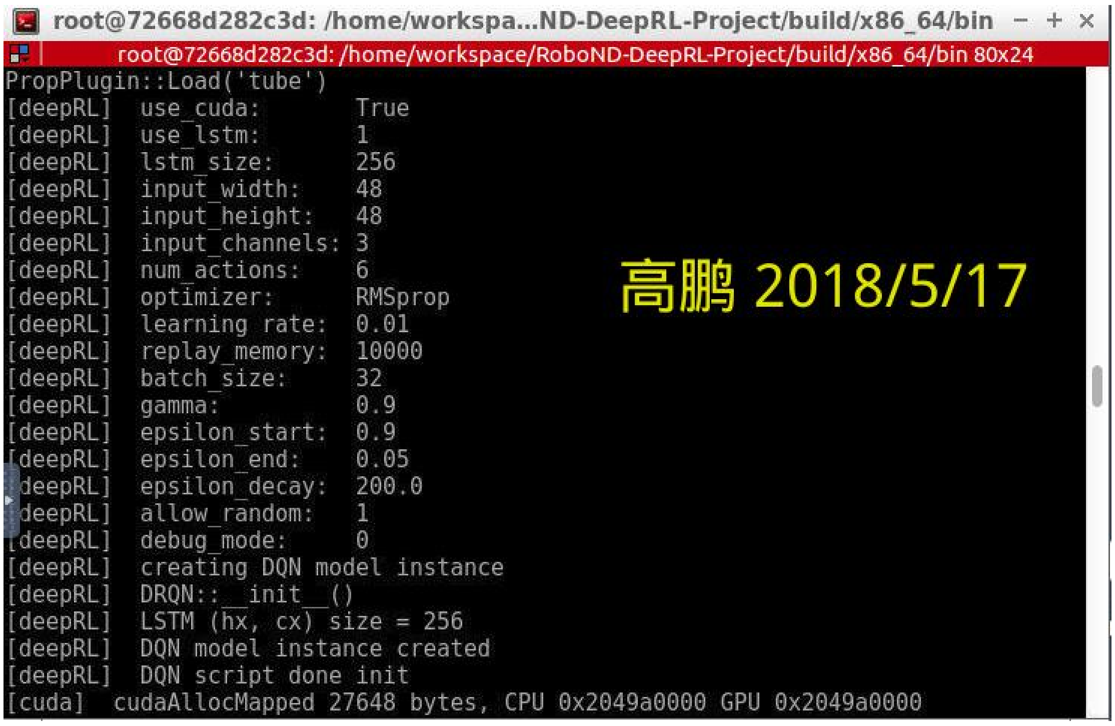
In addition, since the network input is a series of image data in time series, the introduction of RNN will better and more fully learn the data. However, RNN itself has a forgetting feature for long-term memory, so the introduction of LSTM to enhance long-term memory Learning. Using LSTM as part of a DQN will allow the network to better learn the relationships and information between the images in the sequence. This will allow the robotic arm to learn the target action more quickly than a single image. Here LSTM_size is 256.

Finally, we choose RMSprop as the optimization function. Because we introduce LSTM as part of the DQN, RMSprop is suitable for dealing with non-stationary targets, and it works well for LSTM. Here the learning efficiency is set to 0.01 in both tasks, and during actual training, This learning efficiency is continuously decreasing in order to achieve the goal of convergence to global optimum.



```

#define INPUT_WIDTH   48
#define INPUT_HEIGHT  48
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.01f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 32
#define USE_LSTM true
#define LSTM_SIZE 256

/*
/ TODO - Define Reward Parameters
/
*/

#define REWARD_WIN  0.1f
#define REWARD_LOSS -0.1f

```

FIgure2. Task2 Hyperparameters

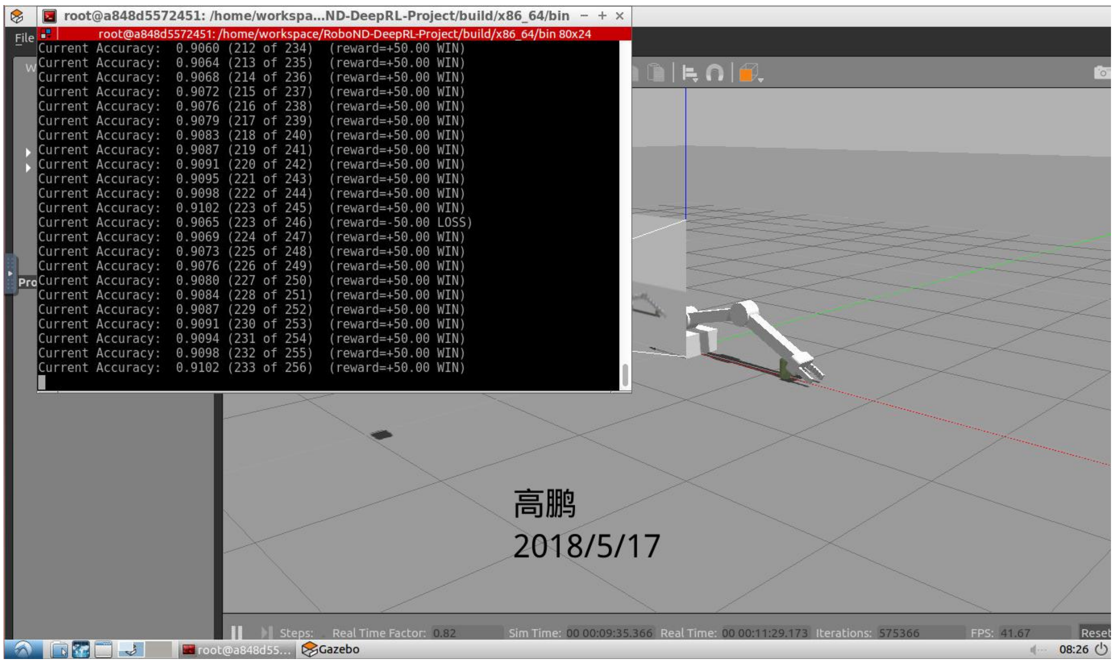

root@72668d282c3d: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@72668d282c3d: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
高鹏 2018/5/17

```
PropPlugin::Load('tube')
[deepRL]  use_cuda:       True
[deepRL]  use_lstm:       1
[deepRL]  lstm_size:      256
[deepRL]  input_width:    48
[deepRL]  input_height:   48
[deepRL]  input_channels: 3
[deepRL]  num_actions:    6
[deepRL]  optimizer:      RMSprop
[deepRL]  learning rate:  0.01
[deepRL]  replay_memory:  10000
[deepRL]  batch_size:     32
[deepRL]  gamma:          0.9
[deepRL]  epsilon_start:  0.9
[deepRL]  epsilon_end:    0.05
[deepRL]  epsilon_decay:  200.0
[deepRL]  allow_random:   1
[deepRL]  debug_mode:     0
[deepRL]  creating DQN model instance
[deepRL]  DRQN::__init__()
[deepRL]  LSTM (hx, cx) size = 256
[deepRL]  DQN model instance created
[deepRL]  DQN script done init
[cuda]  cudaAllocMapped 27648 bytes, CPU 0x2049a0000 GPU 0x2049a0000
```

FIgure3. Task2 Hyperparameters

For task two, there is no excessive adjustment in hyperparameters. The main adjustment is only the modification of the reward function. However, it may be safer to adjust the learning rate to a smaller value, for example 0.001.

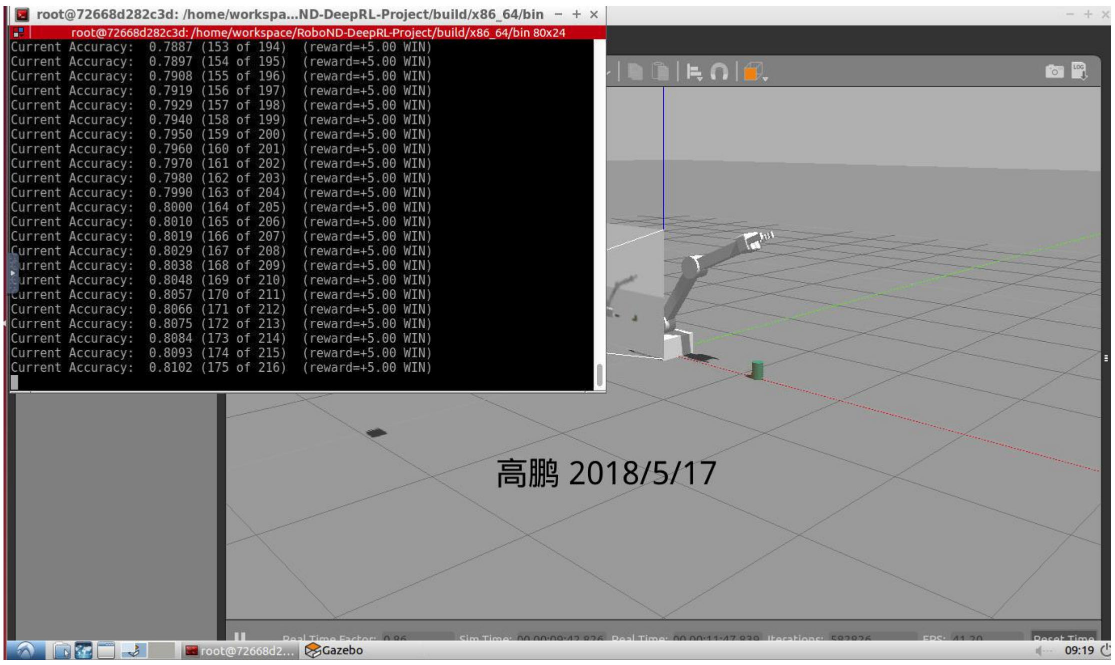
3.Results


root@a848d5572451: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@a848d5572451: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
File
高鹏
2018/5/17
Steps:
Real Time Factor: 0.82
Sim Time: 00 00:09:35.366
Real Time: 00 00:11:29.173
Iterations: 575366
FPS: 41.67
Reset
root@a848d55...
Gazebo
08:26

```
Current Accuracy:  0.9060 (212 of 234)  (reward=+50.00 WIN)
Current Accuracy:  0.9064 (213 of 235)  (reward=+50.00 WIN)
Current Accuracy:  0.9068 (214 of 236)  (reward=+50.00 WIN)
Current Accuracy:  0.9072 (215 of 237)  (reward=+50.00 WIN)
Current Accuracy:  0.9076 (216 of 238)  (reward=+50.00 WIN)
Current Accuracy:  0.9079 (217 of 239)  (reward=+50.00 WIN)
Current Accuracy:  0.9083 (218 of 240)  (reward=+50.00 WIN)
Current Accuracy:  0.9087 (219 of 241)  (reward=+50.00 WIN)
Current Accuracy:  0.9091 (220 of 242)  (reward=+50.00 WIN)
Current Accuracy:  0.9095 (221 of 243)  (reward=+50.00 WIN)
Current Accuracy:  0.9098 (222 of 244)  (reward=+50.00 WIN)
Current Accuracy:  0.9102 (223 of 245)  (reward=+50.00 WIN)
Current Accuracy:  0.9065 (223 of 246)  (reward=-50.00 LOSS)
Current Accuracy:  0.9069 (224 of 247)  (reward=+50.00 WIN)
Current Accuracy:  0.9073 (225 of 248)  (reward=+50.00 WIN)
Current Accuracy:  0.9076 (226 of 249)  (reward=+50.00 WIN)
Current Accuracy:  0.9080 (227 of 250)  (reward=+50.00 WIN)
Current Accuracy:  0.9084 (228 of 251)  (reward=+50.00 WIN)
Current Accuracy:  0.9087 (229 of 252)  (reward=+50.00 WIN)
Current Accuracy:  0.9091 (230 of 253)  (reward=+50.00 WIN)
Current Accuracy:  0.9094 (231 of 254)  (reward=+50.00 WIN)
Current Accuracy:  0.9098 (232 of 255)  (reward=+50.00 WIN)
Current Accuracy:  0.9102 (233 of 256)  (reward=+50.00 WIN)
```

FIgure4. Task1 Results

For both tasks, meeting the final goal all experienced more than 100 episodes. For task 1, 91.02% accuracy rate was achieved after 256 episodes; for task 2, 81.02% was accurate rate after 216 episodes.



```
root@72668d282c3d: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.7887 (153 of 194)  (reward=+5.00 WIN)
Current Accuracy:  0.7897 (154 of 195)  (reward=+5.00 WIN)
Current Accuracy:  0.7908 (155 of 196)  (reward=+5.00 WIN)
Current Accuracy:  0.7919 (156 of 197)  (reward=+5.00 WIN)
Current Accuracy:  0.7929 (157 of 198)  (reward=+5.00 WIN)
Current Accuracy:  0.7940 (158 of 199)  (reward=+5.00 WIN)
Current Accuracy:  0.7950 (159 of 200)  (reward=+5.00 WIN)
Current Accuracy:  0.7960 (160 of 201)  (reward=+5.00 WIN)
Current Accuracy:  0.7970 (161 of 202)  (reward=+5.00 WIN)
Current Accuracy:  0.7980 (162 of 203)  (reward=+5.00 WIN)
Current Accuracy:  0.7990 (163 of 204)  (reward=+5.00 WIN)
Current Accuracy:  0.8000 (164 of 205)  (reward=+5.00 WIN)
Current Accuracy:  0.8010 (165 of 206)  (reward=+5.00 WIN)
Current Accuracy:  0.8019 (166 of 207)  (reward=+5.00 WIN)
Current Accuracy:  0.8029 (167 of 208)  (reward=+5.00 WIN)
Current Accuracy:  0.8038 (168 of 209)  (reward=+5.00 WIN)
Current Accuracy:  0.8048 (169 of 210)  (reward=+5.00 WIN)
Current Accuracy:  0.8057 (170 of 211)  (reward=+5.00 WIN)
Current Accuracy:  0.8066 (171 of 212)  (reward=+5.00 WIN)
Current Accuracy:  0.8075 (172 of 213)  (reward=+5.00 WIN)
Current Accuracy:  0.8084 (173 of 214)  (reward=+5.00 WIN)
Current Accuracy:  0.8093 (174 of 215)  (reward=+5.00 WIN)
Current Accuracy:  0.8102 (175 of 216)  (reward=+5.00 WIN)
```

root@72668d282c3d: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
高鹏 2018/5/17
root@72668d2...
Gazebo
09:19

FIgure5. Task2 Results

In summary, we can easily find that the accuracy rate of task 2 is lower than the accuracy rate of task 1. This is because the requirements of task 2 are more severe and more severe than those of task 1. From the result graph, we can find Two tasks based on the above network model and hyperparameters have achieved the target result. However, there are still some problems. For the second task, if the agent does not fully iteratively learn the target, the network will not converge, which requires a restart process. Let the agent re-learn fully on the target object. Another drawback is also encountered in the experiment, that is, the network sometimes converges to the local optimum rather than the global optimum. This is because the network only knows how to make the robot arm touch Objects do not care about the site. The agent mistakenly believes that this is correct, and therefore rewards will be given. This will lead to an accurate and sudden increase, but this is wrong. Therefore, we hope that the agent will have a convergence trend as soon as possible, that is, the correct action will appear, allowing the robotic arm to fully learn and achieve good results.

4.Future work

Although both tasks have reached the goal, there are still problems that can be improved in this process. First, the hyperparameters can be further fine-tuned and the optimization function and learning efficiency can be optimized. The network can be implemented using optimization functions such as Adam and a small learning efficiency. Optimization, to reduce the occurrence of local optimal situation. Second, for the reward function can be further optimized, for the temporary reward function may be able to use LWMA algorithm for further optimization. For the final reward can be set without constant, so that it changes according to the training situation.

5.Reference

[1] Deep Reinforcement Learning for Robotic Manipulation with

Asynchronous Off-Policy Updates. Shixiang Gu. cs.RO. 2016