SIGMA CLERMONT
ISPR

AI Memo 001 June 01, 2020

Another Banana's Collector Agent Implementation

by

Carlos M. MATEO

Abstract: This project implements the method presented in the letter [2] for resolve the problem of generate a control policy of the amazing banana's collector. Moreover implements state-of-the-art improvements like prioritized experience replay and dueling Double Deep Q-Networks DDQN. The results show that this implementation using the hyper-parameters summarized in this report obtain an average score of at least 13 points. Finally, it is done a brief discussion about which improvement and future directions can be taken to improve this repo. The use of the associated code is well explained in the README file.

Keywords: Deep Reinforcement Learning; Q-Learning; Robot Control

Special thanks to Udacity community for helping with this implementation.

1 Method

Model architecture. The Q-value function implemented for solving this problem is based on dueling q-networks [5] as it has been discussed previously. This architecture combine the result of to deep networks which share a common part. Here, the Common sequential part consists of one fully connected layer of 128 parameters plus three fully connected layer of 128 parameters and one fully connected layer of 32 parameters. While value and advantage sequential part consist of three fully connected layers of 32 parameters each one, value part has one last output layer with 4 component and advantage part has one last output layer with 1 node. After each fully connected layer is applied a rectified linear function (ReLU), except after output layers. This is due the fact that Q-value are real numbers between $[-\infty,\ \infty]$.

The error (loss) function used is MSE and ADAMS optimizer is used for updated local DQN parameters. For preventing double gradient propagation over over common part during parameter optimization, last common layer gradient is multiplied by the coefficient $\frac{1}{\sqrt{2}}$. Moreover, for reasons of stability gradients are clipped during backward pass.

Double DQN. This scheme [4] keeps update two equal deep q-networks DQN, a local and a target one, with different weights and at different rates. Here, while the local DQN is update each C steps, the target DQN is updated each less often τ steps. The weight are randomly initialize. The parameter parameters are softly update according with,

$$\theta_{target} = \alpha * \theta_{local} + (1 - \alpha) * \theta_{target}$$

Experience Replay. Instead of using an uniform sampling strategy for selecting experience samples from replay buffer, here I use a prioritized experience replay PER selection strategy following [3]. Where for each experience sample has a priority weight based on the error between Q-value predicted and the target one which is used for compute an importance sample weight for prioritize gradients. The main drawback of this approach is to keep a sorted buffer replay, because the main idea is to use those sample experiences which give more information. For handling this issue I am using the segment tree implementation of OpenAI [1].

[1] available at `https://github.com/openai/baselines/blob/master/baselines/common/segment_tree.py`

2 Experiments

Training Details Here, each new training environment is randomly initialized using a seed based on the system-clock tick. The input data used is an stack of the $k = 2$ last observations. For each training episode execution using the ε-greedy policy, it is passed a validation full episode using a greedy policy. While ε is exponentially decreased from 1.0 to 0.01, β parameter is linearly annealed from 0.4 to 1.0 during the first 1500 episodes. From the 1500 episode ε and β hyper-parameters keep constant. The rest of hyper-parameters are summarize in table 1. It is executed the agent before start training with a random policy for populate buffer replay with at least 5000 sample experiences. Stop condition is reached once as well training as validation results pass 13 points as score. Training results are shown in figure 1.

Table 1: Hyper-parameter configuration

Hyper-parameter	**Value**	**Comments**
buffer size	100000	Replay buffer size.
batch size	64	Minibatch size.
γ	0.99	Discount rate.
α	0.0025	Proportional increment used for soft updating target parameter.
τ	10	Steps to update the target network. After τ local updates, target parameters are soft updated.
learning rate	0.00012	Learning rate used in ADAM optimization.
C	4	Every 4 step is updated local parameters.
ϵ	0.000001	Minimum priority value.
α	0.4	PER alpha value [0, 1], as closer to 0 closer to uniform sampling behavior.
Clipping	10	Gradient clipping.

Evaluation Details. The agent is executed during 100 episodes using an ε-greedy policy with $\varepsilon = 0.001$. The cumulative discounted reward are averaged and obtaining 14.44 points. A detailed list of intermediate results is shown in table 2.

3 Discussions and Conclusions

Results show that this method reach an average of cumulative reward of at least 13. An interesting point that appear after observing these results is the fact that

using ε-greedy policy with $\varepsilon \approx 0.001$ it is obtained better performance than when a greedy policy $\varepsilon = 0.0$ is used. I hypothesize that this fact indicates that it is needed more training time to get the optimal solution. But note, by observing figure 1, that the cumulative discounted reward of 13 is obtained before of executing the 1800 episodes. So, also I do not get the optimal policy, it is a good trade off between expended training time and agent behaviour obtained. Also, note that the network architecture used here is quit deep but uses small layer, modeling a network with bigger layer and using dropout technique we can prevent some overfitting effects. Although, that means more training time.

Some interesting improves that can be contemplated for this task are the Rainbow [1] which extend dueling networks by adding Actor critic strategies as well as noisy layer among other improves. Also another interesting future improve is to add LSTM layer to the network architecture. Because, I have shown here that this problem is resolve using sequential input data, LSTM exploit time sequential input data. Finally, also, an important improve will be using image as input and substituting common layers by convolutional layers.

References

[1] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3215–3222, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–21, 2016.

[4] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-Learning. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 2094–2100, 2016.

[5] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Frcitas. Dueling Network Architectures for Deep Reinforcement Learning. *33rd International Conference on Machine Learning, ICML 2016*, 4(9):2939–2947, 2016.

Table 2: Evaluation results

Iteration	**Current Score**	**AVG(Score)**
10	19	11.10
20	10	12.20
30	9	13.34
40	17	13.50
50	20	13.96
60	18	14.05
70	18	14.30
80	20	14.53
90	14	14.48
100	8	14.44

Figure 1: Training results. This figure presents 3 different plots: Score evolution; Error evolution; and ε-β evolution. Top plot (score evolution) shows the curve of training score average (dark red line) over the last 100 episodes, i.e. each point in the curve represents the mean score of the last 100 episodes. Also the averaged validation score is represented (blue line) in this plot. Middle plot represent the increment applied to the local Q-network parameters. Bottom plot shows the linear and exponential evolution of the hyper-parameters β and ε.