Reinforcement Learning
Homework Assignment 2

s1641718

1 Actor-Critic Architecture

Actor-Critic Actor-critic methods are temporal difference reinforcement learning methods, were the policy has its own weight parameters and is independent of the value function [1]. This means that, contrary to the action-value methods, were we learn action-value pairs and make actions by using these values, in action-critic methods we may still learn the values of a state or an action (so we have a value function), but we don't use these values to choose the next action. The policy structure is the actor. The actor aims at improving the current actions. The critic is evaluating the current policy used, that is, it criticizes the action made by the actor, and is usually the value function. A temporal difference error is used for this evaluation. The error value shows whether the action taken by the actor is good or not, by computing the value:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t),$$

where V(s) is calculated with the value function used as critic.

Actor-critic methods present two main advantages. One is that, given a large or infinite action space, with these kind of methods the optimal policy can be stored, saving the time for the computations of exploring through each specific action, that would be otherwise required at each selection step with a action-value method. The other advantage is that actor-critic methods can be trained to learn the probability distributions of each action, or in other words, they can learn stochastic policies.

Application Example An application example of an actor-critic method is presented by [2], where an actor-critic reinforcement learning method is used as a autonomous learning framework for controlling a biped robot. To control the robot, neural circuits called central pattern generators (CPG) are used. The problem the authors faced was that, firstly, the controller depends on the output of the neurons, thus becomes a non-stationary controller, while most reinforcement learning algorithms learn stationary policies. Secondly, to train a recurrent neural network of this magnitude, a huge amount of back-up of the past trajectories was required, which would result in analogous computational time.

To solve these problems, the authors propose an actor-critic algorithm, and they isolate the physical system in a way that the controller can be independent of the internal states of the CPG neurons. The basic CPG system has its own connection weights, and is controlled by the actor, which is the rest of the controlling system. The internal states of the CPG are now dependent on the environment. They employ a policy gradient method to criticize the actor. The connection weights are the policy parameters, and they are updated based on the gradient's performance (critic). Overall, by definingg the basic controller/actor in this way, the learning becomes easier since the parameters are significanlty reduced.

SARSA Relation Sarsa in an action-value method, that is, the action-value pairs are learned and the policy is then exclusively dependent on these values. Similarly to the TD error used to evalutate the actor's policy, Sarsa updates the Q-value based on the current state, the current action of the agent, the reward, the previous state and the previous agent's action:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Given the fact Sarsa is a on-policy learning algorithm and the way the values are updated, it can be deduced that Sarsa can be an effective critic (not actor) in a actor-critic method.

2 RL with Function Approximation

1 We have the linear approximation of the state-action value function at time t: $Q_t(s, a) = \theta_t^T \phi_{s,a}$, where θ_t^i and ϕ_t^i denote the i^{th} component of the weights and feature vectors. To reproduce the tabular case of the Q function we need to construct the feature vectors and initialize their weights as follows:

We define the features in a way that each feature represents exactly a possible state-action pair. The feature ϕ and weight θ vectors length then is equal to the number of total state-action pairs, n. The feature vector will be: $\phi_t^i = (\phi_0, \phi_1, ..., \phi_i, ..., \phi_n)^T$. The weight vector for the i^{th} state-action pair should then be: $\theta_t^i = (0, 0, ..., 1, ..., 0)^T$, with weight 0 in any position other than

i. The resulting Q value will then be: $Q_t^i(s,a) = \theta_t^{iT}\phi_t^i = \phi_t^i$, which will be equal to the tabular Q function case.

2 We cannot possibly have as many features as the state-action pairs, so we will approximate them. We need to include features that cover collision prevention, reward moving faster and staying in the center of the road. The features we chose to implement were ones covering a basic representation of a possible state and are described in Table 1. Depending on each action, the feature vector changes accordingly.

Feature	Value
$F_1(s,a)$	Car is in center, no other car in front 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_2(s,a)$	Car is not center, and is closer to the right wall 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_3(s,a)$	Car is not center, and is closer to the left wall 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_4(s,a)$	There are cars in front, and more cars to the left 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_5(s,a)$	There are cars in front, and more cars to the right 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_6(s,a)$	There are no cars ahead 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_7(s,a)$	Car just collided, there are more cars to the left 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_8(s,a)$	Car just collided, there are more cars to the right 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_9(s,a)$	Car is next to the right wall, there is no car to the left 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{10}(s,a)$	Car is next to the left wall, there is no car to the right 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{11}(s,a)$	Car is next to the right wall, there is a car to the left 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{12}(s,a)$	Car is next to the left wall, there is a car to the right 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{13}(s,a)$	Car is accelerating, there are no cars ahead, maybe to the sides 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{14}(s,a)$	Car is accelerating, there are cars in front, and more to the left 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{15}(s,a)$	Car is accelerating, there are cars in front, and more to the right 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{16}(s,a)$	Car has max speed, there are no cars ahead, left turn coming up 1 - if action a would be most likely beneficiary to the agent 0 - otherwise

$F_{17}(s,a)$	Car has max speed, there are no cars ahead, right turn coming up 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{18}(s,a)$	Car has max speed, there are cars in front 1 - if action a would be most likely beneficiary to the agent 0 - otherwise
$F_{19}(s,a)$	Car has max speed, and is in the center with no cars ahead 1 - if action a would be most likely beneficiary to the agent 0 - otherwise

Table 1: Feature vector and value assignment

Collision The features that specifically target collision avoidance are $F_4(s,a)$, $F_5(s,a)$, $F_{11}(s,a)$, $F_{12}(s,a)$, $F_{14}(s,a)$, $F_{15}(s,a)$, $F_{18}(s,a)$, since we take into account that there are cars around and try avoid the action that would result in colliding with them.

Moving faster The features that 'reward' (support) moving faster and, as a result, passing by more cars are $F_1(s,a)$, $F_6(s,a)$, $F_{11}(s,a)$, $F_{12}(s,a)$, $F_{13}(s,a)$, since we are taking into account that the car is accelerating or has max speed and by continuing in doing so it gets rewarded.

Staying in the center The features that support staying closer to the center are $F_1(s,a)$, $F_2(s,a)$, $F_3(s,a)$, $F_9(s,a)$, $F_{10}(s,a)$, $F_{16}(s,a)$, $F_{17}(s,a)$, $F_{19}(s,a)$, since we are taking into account that the car either is in the center and gets rewarded if it stays, or is not in the center and reward moving towards that position.

3 (a) The learning curve of the function approximation model is shown in Figure 1. It can be seen that, from the start, the model with these features did not perform as well as the Q-learning agent. The model is, after all, using an approximation of the Q-function. Its performance is dependent on the selection of the features. Since we picked features that did not cover all possible state-action scenarios, the model was not expected to perform as well. The agent's performance is indeed not showing Q-learning' s progress . Even so, the weight vector is updated and more important features get distinguished.

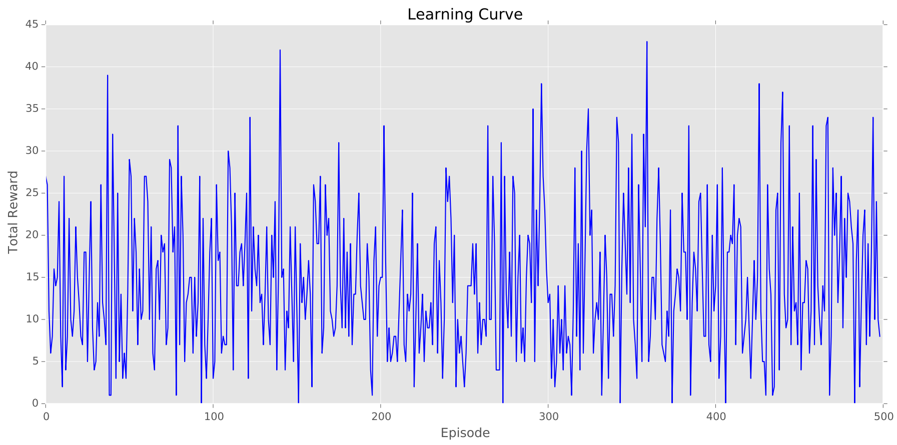

Figure 1: Learning curve - Function approximation agent - 19 features

The learning curve of the basic Q-learning agent is shown in Figure 2.

(b) During the training of the model, the weight values associated with each feature are updated according to the equation:

$$\theta_{t+1} = \theta_t + \alpha(r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t))\nabla_{\theta_t} Q_t(s_t, a_t)$$

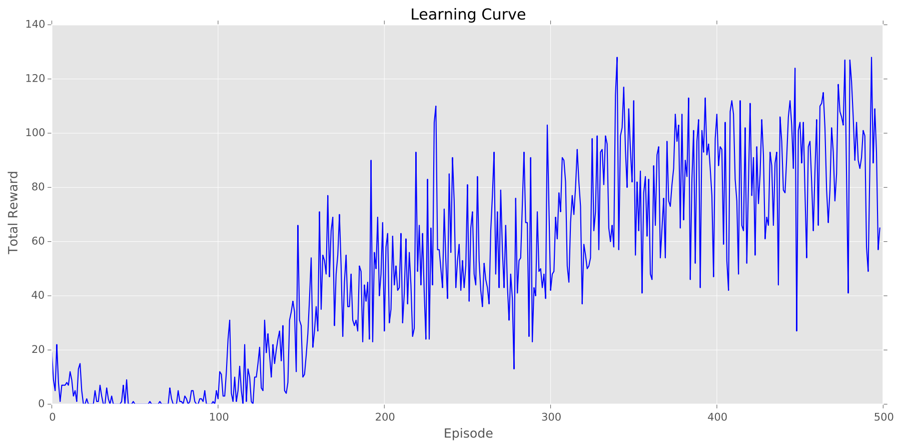

Figure 2: Learning curve - Q-learning agent

The curves of the evolution of each weight associated to each feature are plotted in Figure 3. Table 2 contains the final θ vector and are plotted in Figure 4. What can be seen is that some features have had their weights updated close to zero, while other features maintain a high weight factor. This means that the latter are more important. In other words, the features corresponding to these weights affect more what action the agent will choose than the rest. Here, these features are $F_1(s,a)$,$F_6(s,a)$,$F_{13}(s,a)$,$F_{19}(s,a)$. It is interesting to note that these features are related to whether the car is accelerating and is in the center of the road. Since, by default, this behavior possibly gives better reward, the agent has adapted to these features.

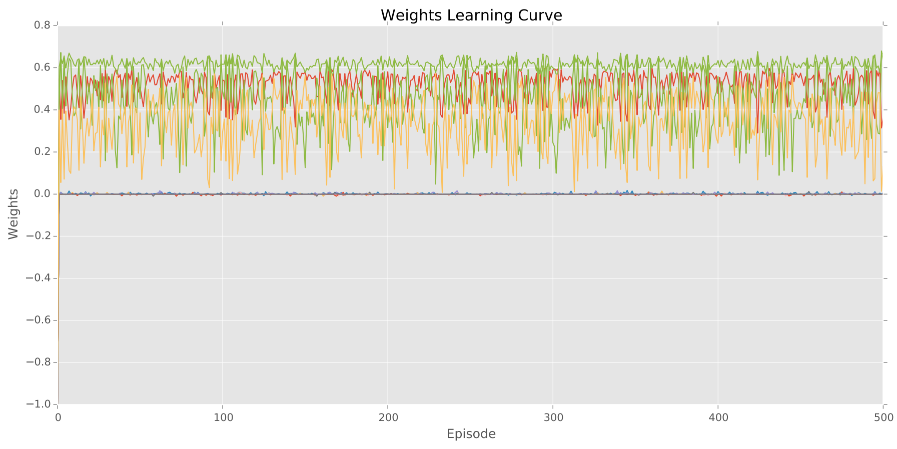

Figure 3: Weight curves - 19 features

Weight vector values									
$F_1(s,a)$	$F_2(s,a)$	$F_3(s,a)$	$F_4(s,a)$	$F_5(s,a)$	$F_6(s,a)$	$F_7(s,a)$	$F_8(s,a)$	$F_9(s,a)$	$F_{10}(s,a)$
3.142e-001	1.06e-003	-1.86e-003	-2.189e-112	-1.86e-003	6.625e-001	-4.586e-020	-1.863e-003	1.169e-006	2.714e-008
$F_{11}(s,a)$	$F_{12}(s,a)$	$F_{13}(s,a)$	$F_{14}(s,a)$	$F_{15}(s,a)$	$F_{16}(s,a)$	$F_{17}(s,a)$	$F_{18}(s,a)$	$F_{19}(s,a)$	
5.928e-323	5.928e-323	6.797e-001	5.234e-004	1.231e-007	2.937e-006	4.18e-008	-2.487e-056	1.399e-002	

Table 2: Weight vector after 500 episodes - 19 features

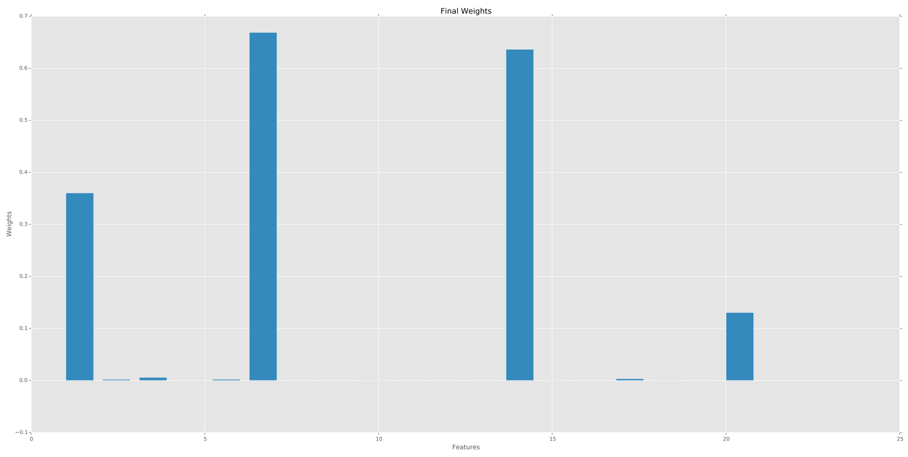

Figure 4: Weight vector after 500 episodes - 19 features

(c) We are approximating the Q learning algorithm using far too less features than the state-action pairs a Q-agent would encounter and would train on. Thus, by construction, the approximation agent was not expected to perform as good as the Q-learning one. Even so, since we had few features, the convergence would be faster. This can be seen by comparing Figures 1 and 2. The Q-learning agent shows signs of convergence after the 500 episode training. The function approximation agent did not show any signs. But it can be seen from the way the weights keep updating, in Figure 3. The learning curves keeps fluctuating, since the agent is adjusting and updating each feature at each time step. What eventually happened is that the more important features ended up with a high weight value, whereas not so beneficiary features ended up with weights close to 0. This behavior is also a result of the quality of the features we chose. What was deduced from these experiments was that these features were not enough to efficiently represent the state.

References

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.

[2] Nakamura, Yutaka, et al. "Reinforcement learning for a biped robot based on a CPG-actor-critic method." Neural Networks 20.6 (2007): 723-735.