Mini project - Reinforcement Learning for the "Mountain-Car" task

Sebastien Duc

December 19, 2011

About the code

The project was implemented in `python`. To run the project, type `python starter.py`. You can change parameters in the code when the object `DummyAgent` is created at the end of file `starter.py`.

class DummyAgent

This class represent the intelligent agent that will learn to solve the problem of the mountain car using SARSA algorithm. The important parameters of this class are

- `N_input_neurons`: It's the size of the grid of the input layer of neurons. The grid has dimension `N_input_neurons`×`N_input_neurons`. By default it's equal to 20.
- `eta`: It's the learning rate used in the algorithm. By default it's equal to 0.1.
- `lambda_e`: It's the eligibility decay rate. By default it's 0.95.
- `tau`: It's the exploration temperature. By default it's equal to 1 initially and it's divided by 2 after every trial until it has reached value 2^{-7}
- `w`: If it's 0 then the initial weights are 0 and if it is 1 then they are 1.

The agent run by default 20 trials using function `learn`. Each trial is preformed by function `run_trial`. At the beginning of each trial the car is placed at random. A trial ends when the car has succeeded to climb the mountain.

class InputLayer

This class represent the input layer grid of neurons. It's instanciated in `class DummyAgent`. Neurons are equally space in $[-150, 30] \times [-15, 15]$ Neurons have gaussian response. Each of them has three weights associated to them, one for each possible action. To each weight is associated an eligibility factor, with 0 as initial value.

Analysis of the model

In this section we will analyze the impact of the different parameters of SARSA algorithm. The learning rate will be always set to 0.1. And the input layer will always be 20 by 20. The reward factor will also always be equal to 0.95. To test the parameters and have statistics, 10 agents are ran with 20 trials.

initial weights

Two different initial weights where tested, when they are all initialized to 0 or when they are all initialized to 1.

- **weights initialized to 0**: During the first trial the weigths are not updated at all during the update step of the weights in SARSA. Let w_{aj} be the weight of Neuron j for action a, let η be the learning rate, let e_{aj} be the eligibility factor of neuron j for action a, let γ be the reward factor and let $Q(s,a) = \sum_{j \in \mathcal{N}} w_{aj} r_j(s)$ where $\mathcal{N}$ is the set of neurons of the input layer. Then the weights are update using the following:

$$w_{aj}(t) \leftarrow w_{aj}(t - \Delta t) + \eta e_{aj} \left(R - (Q(s_{\text{old}}, a_{\text{old}}) - \gamma Q(s,a))\right)$$

  But while $w_{aj} = 0$, $Q(s,a) = 0, \forall s \in S, a \in A$ where S is the set of states and A the set of action (e.g. $S = [-150, 30] \times [-15, 15]$ and $A = \{-1, 0, 1\}$). Therefore the weights become non-zero when the agent gets its reward at the end of the first trial. After that, it has a normal behavior.

- **weights initialized to 1**: Unlike when weights are initialized to 0, the weights change during the first trial. Typically, the agent learns a little faster because of that just a the beginning.

eligibility trace decay rate

Two different eligibility trace decay rates where tested. When it's set to 0 and when it's set to 0.95. When the decay rate is 0, we keep no memory, thus the learning takes much more time than when it's 0.95. Let λ denote the eligibility trace decay rate.

- $\lambda = 0$: No memory is kept in that case. Therefore the agent learn very slowly since it has to wait until it gets a reward update the weights.

- $\lambda = 0.95$: In that case we keep good memory of what happens. Therefore the agent learns a lot more quickly.

exploration temperature parameter

Let τ denote the exploration temperature paramter. Several tests were done for this parameter to see the impact on exploration and exploitation. First when it's constant, $\tau = 1, \tau = 0, \tau = \infty$. Then it's a time decaying function

- **constant temperature parameter**: When $\tau = 1$ exploration is too much favorized to exploitation. Therefore the agent just learn but never really exploits what he learns. The latencies are not good with this constant.

When $\tau = 0$ (in practice τ is very small since 0 would lead to a division by 0, in our case $\tau = 2^{-7}$) exploitation is a way too much favorized to exploration, which implies that the agent has not the opportunity to learn.

When $\tau = \infty$ the agent doesn't use at all what he learned and the action are choosed uniformly at random since

$$\mathbb{P}(\{a^* = a\}) = \lim_{\tau \to \infty} \frac{\exp(Q(s,a)/\tau)}{\sum_{a' \in A} \exp(Q(s,a')/\tau)} = \frac{1}{|A|}$$

where a^* is the action randomly chosen by the agent when it is in state $s \in S$. Therefore the latencies in that case are bad.

- **time decaying function**: At the end of each trial, τ is updated as following:

$$\tau \leftarrow \begin{cases} \tau/2 & \text{if } \tau > 2^{-7} \\ \tau & \text{otherwise} \end{cases}$$

2^{-7} was choosed for technical reasons, since smaller values would lead to overflow when we compute the exponential. This good result, and a way much better that what we got with constant τ. In that case, we have a good balance between exploration and exploitation.