Mini project 2: Reinforcement Learning

Vidit Vidit
Dunai Fuentes Hitos
CS-434, EPFL, Switzerland

***Abstract*—Our goal is to implement a reinforcement learning algorithm based on SARSA to solve the mountain-car problem (escaping from a deep pit). We achieve so by using a biologically plausible neural structure that determines the actions of our agents through linear combination of activations of multiple neural units. We show that our proposed agent can learn to solve the task with a single reward outside of the pit in approximately 40 trials, taking advantage of eligibility traces adapted to act upon weighted activations instead of Q-values directly. We also study different configurations and present results for alternative approaches such as the exploration-exploitation dilemma or zero-initialization against optimistic-initialization. Caveats and suggestions of improvement are discussed in the last section.**

I. Introduction

A. The Mountain-car Problem

A classic problem of reinforcement learning is that of training an car-like agent to escape a pit from which it doesn't have power to go out just by pressing the accelerator at its maximum. The agent needs to learn how to interact with the basic laws of motion and embed into its policy a plan to escape that requires multiple oscillations to gain enough impulse. It also needs to be robust to small variations of the starting conditions (position inside the pit and initial velocity).

B. A Biologically Plausible Agent

The input to our agent is the position and velocity of the car. We define a square grid of evenly separated neurons (20×20 approx.) associated to different positions and velocities with a bidimentional gaussian window (maximum activation at the center, but always slightly active invent with an input on the periphery).
Three different linear combinations of the activations represent the approximated Q-values for the three actions of choice: forward, backward, and do nothing, that get translated to forces applied to the car.

C. Code and References

The code and some extra notes can be found in the complementary ipython notebook and .py files.
This introduction works as a summary. Detailed description of the project can be found in "miniproject-2.pdf".

II. Notes, Corrections and Improvements

We ventured to make some improvements over the suggested description of the project to ensure the stability and good performance of our agents.

First, we shall note that decision making follows a softmax criterion taking the three Q-values (one per action) modulated by a temperature parameter and delivering a probability distribution. The first implication is that unfortunately the learning rate will unavoidably affect decision making, making it impossible to decorrelate the two: for a zero-initialization, in the beginning smaller Q-values will produce almost random choices while in the end, with growing Q-values (actually unbounded in our implementation), may result in no exploration at all. This is a problem that doesn't occur in $\epsilon - greedy$ algorithms. However, despite the fact that we continue with the proposed softmax which is more biologically plausible, we notice that the presented formulation is prone to overflow issues during computation. Instead we implement the **normalized Softmax** which simply divides nominator and denominator of the Softmax by the biggest of the rectified Q-values, making the problem computationally stable, and often more efficient.

Another problematic aspect that we notice is that due to the eligibility trace we may fall into loops by learning from wrongly estimated Q-values, destroying previously learned good rules and even halting our simulations due to loops that become impossible to escape from (remember the just mentioned correlation between exploitation and Q-values' magnitude). To solve this, we introduce a simplistic version of **network multiplexing**. We keep track of two sets of weights, one that gets updated with every action according to SARSA and another that is used for decision making. At the end of each trial, if it was a successful one (the agent finds its way out in less than "max_step" iterations), the weights that get used to make decisions are replaced by those that we update.

All results presented in this report have benefited from these extra additions to the project's instructions and requirements.

III. Learning Curve

To evaluate how well our agent is performing we take the escape latency as a metric. This is, how many actions does the agent take before escaping from the pit. Obviously, the lower the escape latency the better our agent is.
Through the learning process, we expect to see a decreasing escape latency as the agent becomes more and more knowledgeable. However, it is not always the case that an agent that has learned for more trials will outperform one that has had less training, as the final escape latency will depend on the random starting conditions as well as in chance during decision making.

To evaluate how many trials or simulations does our agent need to learn how to solve the task (this is, arriving at a close-to-optimal and locally-optimal policy) we need to plot the learning curve until convergence.
Convergence could be defined as the moment in which our agent doesn't change its policy during training anymore. We won't perform any policy extraction to this end, but rather conclude visually that the algorithm has converged when the escape latency plateaus (stops decreasing) for incremental number of trials, averaged among several agents to mitigate the variations previously discussed.

Results can be observed in Figure 1. Convergence is probably achieved somewhere before the 40th trial. However, the reader is warned to note that presented escape latencies in the learning curve are computed for a temperature τ that has been tuned to bring some exploration during training, and thus the values are bumped by the faulty runs in which our agent doesn't follow the optimal (learned) path. When we simulate this trained agent, now with a very low temperature ($\tau \to 0$ = take action max(Q)) we see escape latencies ranging between 50 and 200 timesteps for different initial conditions. We conclude those to be the minimum number of timesteps required to solve the task. In Figure 2 we the result on two low temperature trials.

Figure 1: The image shows an averaged learning curve for 15 independent runs of our agent. Parameters chosen for the simulation are: $n\_neurons = 20, \tau = 0.0001, \eta = 0.002, \gamma = 0.95, \lambda = 0.95, max\_steps = 4000$.

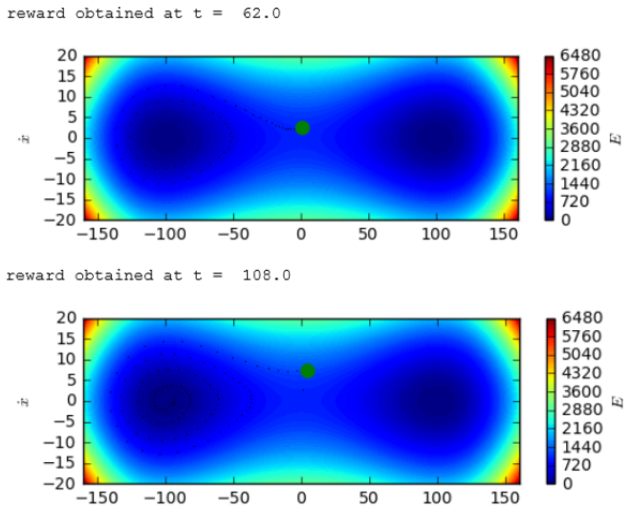

Figure 2: Two low temperature simulations for a train model with 40 trials, one finishing in 62 timesteps, and the other one (with harsher initialization down in the pit) in 108 timesteps. The trajectory as the evolution of position and velocity can be observed in the plot as a dotted path. We confirm that the agent moves back and forth in the pit to gain impulse in order to escape.

IV. Policy visualization

As we have mentioned previously, a way to define convergence when training an agent is to identify it as the convergence of the acting policy. Think of an agent that follows the path of maximum Q-value (the optimal policy for correctly estimated Q-values). In a specific state, if they are $0, 1, 3$ the actions of the agent wouldn't be any different if, after 10 more rounds of training, it estimates the Q-values to be $2, 1, 3$.

We can use this concept to study how quickly does an agent learn a task by looking into the evolution of the policy. This is exactly what it is shown in Figure 3. From the plots we can see that the agent actually learns pretty fast the basics to solve the problem. Starting with a completely random policy (weights initialized to zero), we see that after the first trial, with the help of the eligibility trace, it has already figured out a big part of what is going to be its final policy. After only 5 rounds of training, most of the previously erroneous estimations have been corrected and the boundary states between different decision are very much well identified. Not much change takes place in the policy for the following 35 training rounds. This is partly due to our strong use of exploitation instead of exploration. If we try again, other similar policies will be learned.

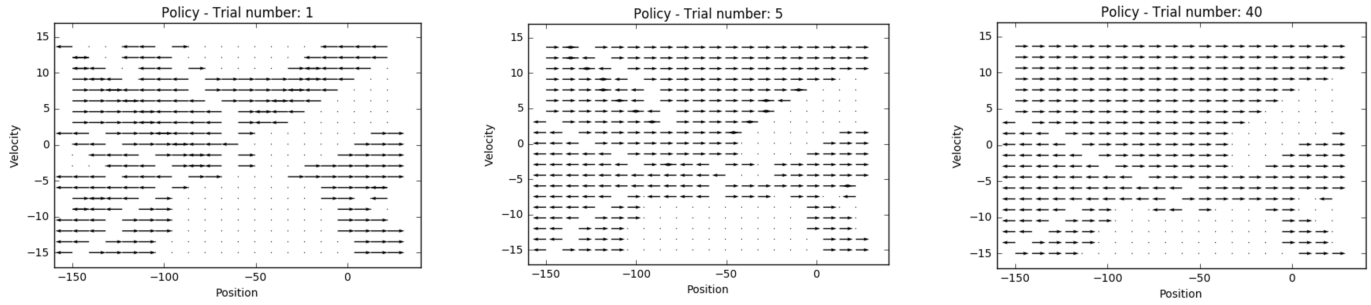

Figure 3: Evolution of the policy after a different number of training trials. Parameters chosen for the simulation are: $n\_neurons = 20, \tau = 0.0001, \eta = 0.002, \gamma = 0.95, \lambda = 0.95, max\_steps = 4000$.

V. Exploration vs. Exploitation

A common factor that has to be fine tuned in reinforcement learning problems (more as an art than science) is the relation between exploration and exploitation. An off-policy trained agent maximizing exploration will eventually learn the optimal solution, but will expend tremendous amounts of time going in parts of the state space that we can identify as completely worthless. On the other hand, if we maximize exploitation we will stick forever to the first good thing we find, missing out on possibly better rewards or sequences of actions. A tradeoff, a good balance, is mandatory.
In our case this tradeoff is controlled by a temperature (τ) that favors exploration when high ($\tau \to \infty$) and exploitation when low ($\tau \to 0$). We shall note nonetheless that SARSA, being an on-policy algorithm will not act optimally as long as the temperature is not 0 (or Q-values infinity) and its convergence is only ensured in the limit if the agent acts optimally in the limit; and won't be getting any close to *the limit* . Furthermore, after adding the eligibility trace we have no guarantee that maximum exploration will lead to anything good.

In Figure 4 we present three different learning curves for different values of τ. Note that, as it was previously explained, due to the given description of the agent, the learning rate will also play an important role along tau in this exploration/exploitation balance; we have fixed it to 0.01 during these experiments.

It seems logical that in this dilemma the best thing to do is to explore more at first and gradually shift towards exploitation as we have a better picture of the state space and its Q-values. It also helps to bring more support to the convergence of SARSA (previously discussed). This is done in our algorithm by gradually decreasing τ with each new trial. The formula in use is: $\tau_n = \tau * \mathrm{e}^{-n}$, where n is the trial number. Resulting learning curve can be seen in Figure 5

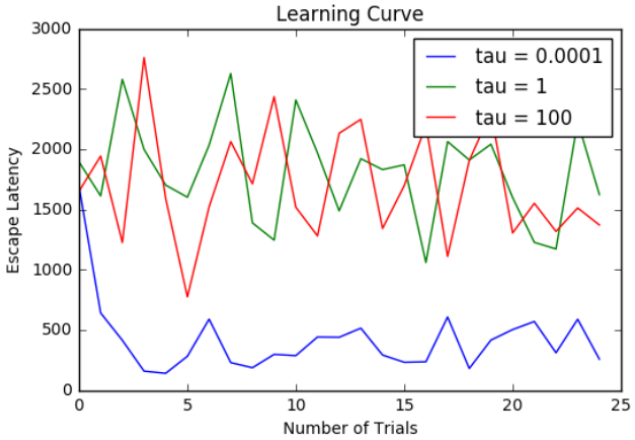

Figure 4: The image shows an averaged learning curve for 10 independent runs of our agent. Parameters chosen for the simulation are: $n\_neurons = 20, \eta = 0.01, \gamma = 0.95, \lambda = 0.95, max\_steps = 4000$. We see how, for this particular setting, a temperature $\tau = 1$ is already excessive exploration, and strong exploitation works much better (as there is a single reward there is no "missing out" beyond the eligibility trace creating bad estimations).

VI. Eligibility Trace

The use of an eligibility trace can greatly speed up learning, particularly when rewards (in the transitions of the state space) are sparse. In our particular case, with a single reward at the end, we manage to learn how to solve the task quite quickly as we have shown repeatedly in Figures 1, 3 and 4 with $\lambda = 0.95$. The contrast is presented in Figure 6 where we make no use of it ($\lambda = 0$).

VII. Optimistic Initialization

So far in all our experiments we have started the training of our agent with weights equal to zero ($w_{aj} = 0$). It is the case of Figure 1. Now we contrast it with Figure 7 where we have changed the initialization to an optimistic all ones ($w_{aj} = 1$). Without looking at the performance results we shall note that this will promote exploitation as it will raise

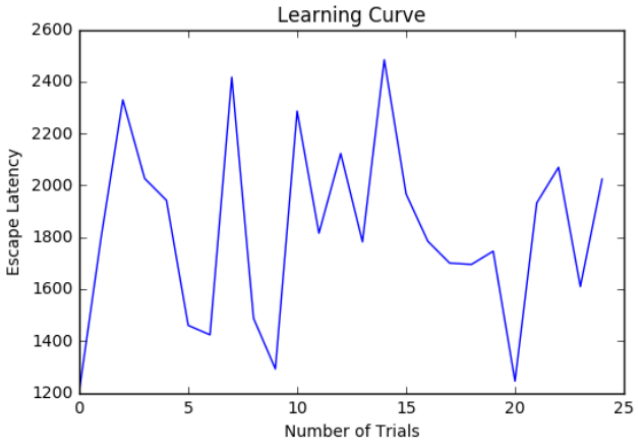

Figure 5: The image shows an averaged learning curve for 10 independent runs of our agent. Parameters chosen for the simulation are: $n\_neurons = 20, \eta = 0.01, \gamma = 0.95, \lambda = 0.95, max\_steps = 4000, \tau = 100$ follows an exponential decay with the number of trials. However, despite the value of τ being extremely low for the later stages we do not see the effect of exploitation reflected on the escape latency. This is because during the random exploration the agent has wrongly updated the weights based on deceptive values brought in by the eligibility trace.

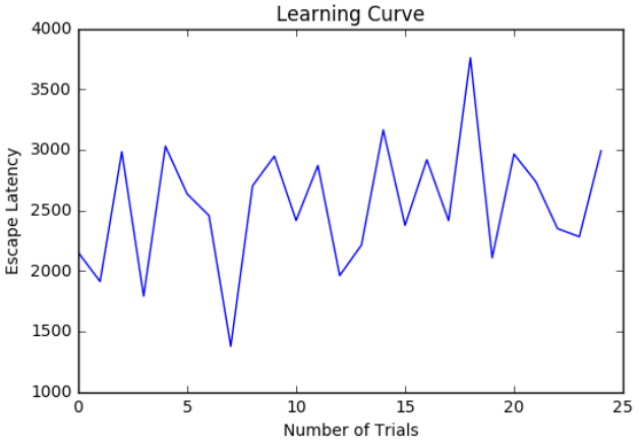

Figure 6: The image shows an averaged learning curve for 10 independent runs of our agent. Parameters chosen for the simulation are: $n\_neurons = 20, \eta = 0.01, \gamma = 0.95, \lambda = 0, \tau = 0.0001, max\_steps = 5000$. Little (basically inappreciable) progress is made without the eligibility trace.

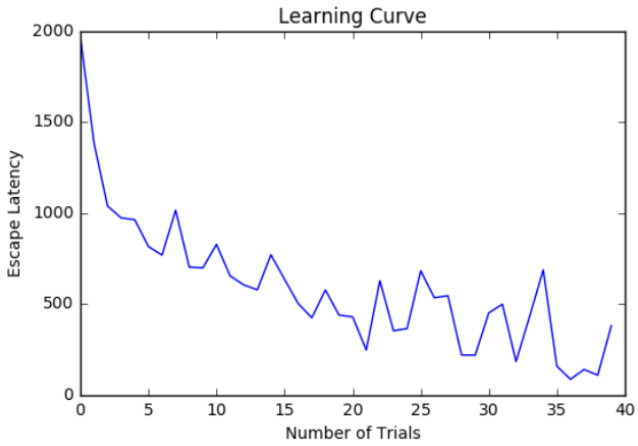

Figure 7: The image shows an averaged learning curve for 15 independent runs of our agent. Parameters chosen for the simulation are: $n\_neurons = 20, \eta = 0.01, \gamma = 0.95, \lambda = 0.95, \tau = 0.0001, max\_steps = 4000$. The results are similar to ones achieved for the zero-initialization case.

all the Q-values, maximizing their differences during the softmax in a similar way a small τ would.

VIII. Final Comments

The eligibility trace is a powerful tool that allows us to learn from yet distant rewards, unlocking the possibility of solving quickly problems with vast state spaces. However, this generous boost to the Q-values along the path to a reward can sometimes result in wrong estimations when a step taken in that path wasn't really helping to get to the reward.

On-policy methods like SARSA are simpler and biologically more plausible than Q-learning, but carry their exploration characteristics with them even when training is finished. One has to be mindful of *turning them off* before deploying the trained agent in the real world.

The evolution of the policy can be a good indicator of convergence and how fast a task is learned.

A. Crafting Good Heuristics

As a side experiment to the project we also tested agents that learned from energy differentials, instead of the single reward from escaping the pit. The heuristic *"the more energy the easier to escape"* works wonderfully and speeds up the training dramatically. Crafting good heuristics as critical as the learning algorithms in real-life scenarios.

Acknowledgements

We want to thank our professor, Marc-Oliver Gewaltig, and the team of assistants, for putting together the material and functions that allowed us to make this project.