ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Unsupervised and reinforcement learning in neural networks

2nd mini project

Lecturer: Marc-Oliver Gewaltig

Assistants: Berat Denizdurduran
Gianrocco Lazzari
He Xu
Michael Moret

Reinforcement learning

1 Project goal

The goal of this miniproject is to implement a neural network with reward modulated plasticity that learns to solve the mountain-car problem. In this problem, an underpowered car has to find its way up a steep hill (Figure 1a). Since the car does not have the sufficient power to directly climb the hill, the solution of the problem is to swing back and forth, applying the modest force provided by the car's engine in the right direction.

2 Problem description

Initially, the car finds itself in the $x < 0$ region of a double well-shaped landscape $h(x) = \frac{(x-d)^2(x+d)^2}{x^2+C}$, $C = d^4/H$. $d = 100m$ describes the position of the minima, and $H = 10m$ the height of the "pass". The random initial conditions ($x \in [-130m, -50m]$) and ($\dot{x} \in [-5\frac{m}{s}, 5\frac{m}{s}]$) are such that the car is "stuck" in the $x < 0$ side of the world. The car has only two gears, forward and reverse, and a neutral position. Applying a force $\vec{F}$ of fixed amplitude backwards or forward with the right timing, we would like to reach the $x > 0$ part of the world.
The car follows Newtonian dynamics, i.e. $m\vec{a} = \sum \vec{F}$. The dynamics of the car are implemented for you in the **mountaincar.py** file. All you have to do is tell the car which way to apply the force and read out its current position and speed. The **starter.py** file provides you an initial template for your agent.

3 Network Structure

We want to learn to solve the task using a biologically plausible neural network, implementing the Sarsa(λ) algorithm. The network you will implement has the following specifications.

- Input Layer: The input layer consists of a population of neurons with Gaussian response curves to the current state $s = (x, \dot{x})$ of the car. That is, the activity $r_j(s)$ of neuron j is

$$r_j(s) = exp(-\frac{(x_j - x)^2}{\sigma_x^2} - \frac{(\psi_j - x)^2}{\sigma_\psi^2}) \qquad (1)$$

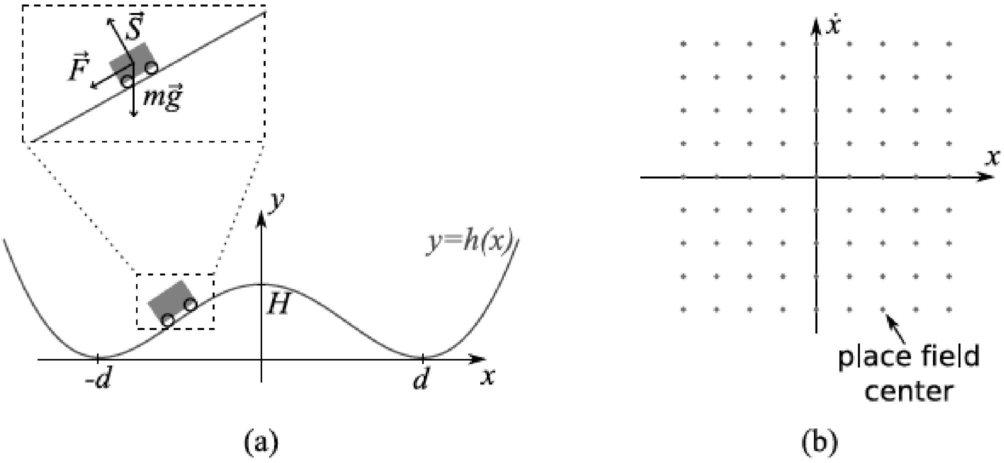

Figure 1: **a)** Schematics of the car on its double-well track. **b)** Each input neuron corresponds to a preferred point (its center) in the $x - \dot{x}$ space. The centers are arranged in a grid fashion.

where (x_j, ψ_j) is the preferred state, or center, of neuron j. The centers of the neurons should be arranged in a grid, as shown in Figure 1b, for example 20x20. The centers should be arranged so as to be able to represent the state of the car over the intervals $-150m < x < 30m$ and $-15\frac{m}{s} < \dot{x} < 15\frac{m}{s}$. The widths of the Gaussians σ_x and σ_ψ should correspond to the distance between the centers.

- Output Layer: The output layer contains 3 neurons, coding for actions ”force to the left”, ”force to the right” and ”no force” respectively. Each output neuron $\boldsymbol{a}$ receives connection from the input layer with weight w_{aj}. The activity of output neuron $\boldsymbol{a}$ when car is in state $\boldsymbol{s}$ is $Q(s,a) = \sum_j w_{aj} r_j(s)$

- Action choice: When the car is in state $\boldsymbol{s}$, the next action we take is determined by the activity of the output cells, following a soft-max rule. With a probability

$$P(a^* = a) = \frac{exp(Q(s,a)/\tau)}{\sum_{a'} exp(Q(s,a')/\tau} \tag{2}$$

it takes action $\boldsymbol{a}$, where τ is an exploration temperature parameter. Each time step of the agent corresponds to the simulation of the car for 1 second (use **n=100** timesteps with **dt=0.01** in **simulate_timesteps**).

- Rewards: the network is signaled it reached the goal $(x > 0)$ by a positive reward r = 1.

- Learning: the weights w_{aj} are updated on each time step according to the Sarsa algorithm with learning rate $\eta << 1$, reward factor $\gamma = 0.95$ and eligibity decay rate $0 < \lambda < 1$.

- Trial structure: A new trial is initiated with the car in a random position (**reset** method). A trial ends as soon as $x > 0$ (and $r > 0$).

- Exploration vs. Exploitation: The parameter τ can be either set once for all, or can be changed during learning.

4 Analyze the model

- Simulate at least 10 agents learning the task, and plot the escape latency (time to solve the task), averaged across agents, as a function of trial number (i.e., the learning curve). How long does it take the agent to learn the task?

- Visualize the behavior of the agent (the policy) by plotting a vector field (0-length vector for the neutral action) given by the direction with the highest Q-value as a function of each possible state $(x, \dot{x})$. Plot examples after different number of trials and comment what you see.

- Investigate the exploration temperature parameter τ, comparing the learning curves. Try fixed values such as, $\tau = 1, \tau = \infty, \tau = 0$, and time decaying functions. Explain its relation to exploration and exploitation.

- Compare the learning curves for different values of the eligibity trace decay rate, e.g., $\lambda = 0.95$ and $\lambda = 0$. What is the role of the eligibility trace?

- Try different initialization of the weights $w_{aj} = 0$ and $w_{aj} = 1$ What is the effect on the learning curves? Explain why.

Report

Your report must be handed in by **Friday, January 20th 2017, at 23:55**. You may work in teams of two persons but you should both upload your report. Submission takes place via the "moodle" web page. You should upload a zip file (named after your last name) containing a PDF of the report and the source code of the programs. The report must not exceed 4 pages.
The functions mentioned above are given as Python source code. All languages are accepted for the project, but we recommend Python, since you can then use the **MountainCar** module. Otherwise, you have to implement the physics yourself.
To use the Python functions from the $miniproject2.zip$ file, you need the Numpy, Scipy and Matplotlib package, which are anyway very strongly recommended for scientific computing in Python.
Feel free to add any comments or additional observations that you had while working on the mini-project.