Unsupervised and reinforcement learning in neural networks

Prof. Wulfram Gerstner
Laboratory of Computational Neuroscience
Assistant: Christian Pozzorini, Carlos Stein

Mini-Project 2: Reinforcement Learning - SARSA(λ)

The goal of this mini-project is to implement a reinforcement learning paradigm using a rate-based neuron model. You will develop a neural network that learns to drive a car.

The position of the car $\vec{p}(t) = [p_x(t), p_y(t)]$ is controlled by modifying its velocity $\vec{v}(t) = [v_x(t), v_y(t)]$ (that is, by accelerating in a given direction $\vec{d}$). The state s of the car is defined by a 4 dimensional vector $s = \{p_x, p_y, v_x, v_y\}$. At each moment in time t, an action a is chosen that changes the velocity in a particular direction $\vec{d_a} = [d_x^{(a)}, d_y^{(a)}]$. After choosing an action a, the velocity vector is first updated $\vec{v}(t+1) = \vec{v}(t) + \vec{d_a}$. Then, the new position $\vec{p}(t+1) = \vec{p}(t) + \vec{v}(t+1)$ is computed. Each time the car hits a wall, a punishment (i.e. a negative reward) is given and the car velocity is reset to zero. Each time the car reaches the end of the circuit, a reward is given. The car should learn to complete the circuit in minimum time.

In the first part of the mini project you will design a neural network that implements SARSA(λ) with eligibility traces and using an ϵ-greedy policy. Then, you will study the performance of the algorithm as a function of some parameters. In the last part of the mini-project you will be free to change the policy and tune the neural network parameters. At the end of the semester an, amazing race will be organized by the TAs in which the algorithms proposed by the students will compete.

1 Implement the neural network

First, you should implement a neural network that controls the movement of the car in the circuit. Consider the following specifications:

- The car is moving in a rectangular arena with unit area (i.e. $p_x \in [0,1]$ and $p_y \in [0,1]$). The position $\vec{p}(t) = [p_x(t), p_y(t)]$ of the (point-like) car is encoded in the activity of a population of place cells. Let there be 31x31 place cells for which the activity of the jth cell is given by

$$r_j^p(\vec{p}) = \exp\left(-\frac{(p_x - x_j)^2 + (p_y - y_j)^2}{2\sigma_p^2}\right), \tag{1}$$

  where the centers of the place cells (x_j, y_j) are arranged on a 31x31 grid (with grid distance 1/30) and $\sigma_p = 1/30$.

- The velocity $\vec{v}(t) = [v_x(t), v_y(t)]$ of the car is encoded in a second population of neurons, called velocity neurons. Let there be 11x11 velocity cells for which the activity of the jth cell is given by

$$r_j^v(\vec{v}) = \exp\left(-\frac{(v_x - x_j)^2 + (v_y - y_j)^2}{2\sigma_v^2}\right), \tag{2}$$

  where the centers (x_j, y_j) are arranged on a 11x11 grid (with grid distance 0.2) and $\sigma_v = 0.2$. In the simulation, the maximal velocity of the car is constrained by forcing the velocity vector $\vec{v}$ to stay in the square $[-1,1] \times [-1,1]$.

- The output layer of the neural network consists of 9 neurons. The activity of the output neuron $a \in \{0, 1, 2, \ldots, 7, 8\}$ represents the Q-value of accelerating in the direction $\vec{d_a} = [\cos(-2\pi a/8 + \pi/2), \sin(-2\pi a/8 + \pi/2)]$, if $a \in [1, 8]$ or keeping a constant velocity (i.e.

$\vec{d_0} = [0,0]$), if $a = 0$. Each output neuron is connected to all neurons in the input layer with connection weights w_{aj}. Given the car position $\vec{p}$ and velocity $\vec{v}$, the activity of the output neuron a is given by

$$Q(\vec{p}, \vec{v}, a) = \sum_{j=1}^{961} w_{aj} r_j^p(\vec{p}) + \sum_{j=962}^{1083} w_{aj} r_j^v(\vec{v}). \tag{3}$$

- At each time step, an action is chosen according to the ϵ-greedy strategy. That is, the action $a^* = \arg\max_a Q(\vec{p}, \vec{v}, a)$ is chosen with probability $1 - \epsilon$ or a random action is taken with probability ϵ. For the beginning, set $\epsilon = 0.1$.
- At each time step, the weights w_{aj} are updated according to the SARSA(λ) algorithm with learning rate $\eta = 0.005$, reward discount rate $\gamma = 0.95$ and eligibility trace decay rate $\lambda = 0.95$. For the SARSA(λ) algorithm, use the formulas for the synaptic update rule and eligibility trace that were given in the lecture (see your notes or the slides on the moodle).

2 Analyze the performance

- **Learning curve**: Simulate at least 10 independent cars that run 1000 trials each. If within a trial, the car does not reach the goal in less than $N_{\max}$ steps, abort the trial. Depending on your implementation, choose e.g. $N_{\max} = 1000$. Plot and discuss the learning curve, that is the number of time steps it takes in each trial until the goal is reached ("latency"). To obtain smooth curves, averages the results obtained with different cars. If your implementation is correct, the curve should decrease and reach a plateau after a certain number of trials.
- **Integrated reward**: Instead of looking at the time it takes to reach the target, look at the total reward that was received on each trial. Is the result consistent with the latency curve?
- **Exploration-exploitation**: The parameter ϵ controls the balance between exploration and exploitation. Why? Analyze how different values of $0 \leq \epsilon \leq 1$ affect the performance. For the comparison, plot several learning curves for different values of ϵ and quantify the performance by averaging the latencies in the last 10 trials. Try to improve the performance by letting ϵ decrease from a large value at the beginning of training to a small value at the end of training.
- **Navigation map**: Visualize the navigation map for different stages of learning. For that, set $\vec{v} = 0$ and plot for different values of $\vec{p}$ the arrow $\vec{d_{a^*}}$ corresponding to the optimal action $a^* = \arg\max_a Q(\vec{p}, \vec{v} = 0, a)$.

3 Car race (optional)

In the last part of the mini-project you may change your algorithm freely in order to optimize it. Modify the parameters of the neural network and explore different policies (e.g. soft-max). If you want, you may change the value of the reward and of the punishment. Modifying the reward in such a way as that a fast trial is more rewarded than a slow trial might for example be a good idea. For technical reasons, if you decide to manipulate the value of the reward, please do that in the class *car* and do not modify the file *track.py*. Describe which algorithm you found best.

Hand in your optimized car as an additional file, named *optimal_car_YourLastName.py*. Make sure the car class follows the specifications described below. We will train your car in a novel track and perform a race between different cars. The winner will receive a special prize.

Using the track simulator

We provide you with a python implementation of the environment, so that you can focus on the implementation of the neural network. In the moodle you will find three files, one with the track simulator (*track.py*), one with a dummy car agent (*car.py*) and one main file that trains the car (*race.py*).

The class *track* has a set of methods defining the environment. This class simulates the movements of the car in the track and, for example, manages the crashes. The file *track.py* does not need to be modified. The class has a method *reset()* that has to be called at the beginning of each trial and a method *move(action)* to tell the simulator which action was taken. The parameter action should be be an integer $action \in [0, 8]$. The method returns the new car position $\vec{p}$, the new car velocity $\vec{v}$ and a scalar value r indicating the reward obtained by taking the *action*. The attribute *finished* indicates if the race has terminated. You may find the attributes *time* and *total_reward* useful too.

Your main work is to develop the class *car* by implementing the neural network and the learning algorithm. This class has two obligatory methods:

- *setup()* : this function should reset the agent and is called the beginning of a each trial (for example this function should reset the eligibility traces).
- *choose_action(position, velocity, reward)* : the arguments of this function are the new car position $\vec{p}$, the new car velocity $\vec{v}$ and the reward r resulting from the last action (all these quantities are returned by the function *move(action)*, of the class *track*). This function should update the synaptic weights w_{aj} according to the SARSA(λ) algorithm and should return the new action $a \in [0, 8]$.

Finally, in the *race.py* file you may develop your analysis routines. You will find the remaining information about usage in the comments directly in the source files.

Report

Your report must be handed either by Wednesday, December 18th 2013 (with fraud detection on Friday, December 20th 2013) or by Wednesday, January 8th 2014, at 23:55 (with fraud detection on Friday, January 10th 2014). 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 source code itself is not part of the written report. The report must not exceed 6 pages.

Feel free to add any comments or additional observations that you had while working on the mini-project.