Exercise: Pole Balancing

Advanced Control lecture
at Ecole Centrale Paris

Anne Auger and Dimo Brockhoff
firstname.lastname@inria.fr

March 8, 2013

Abstract

We already looked at the problem of balancing a single pole on a moving cart with the help of a linear controller in a previous exercise. However, we always simulated the pole for a single (fixed) starting condition for which it was easy to find a parameter setting (by random search) that resulted in a stable pole. In this exercise, we look more carefully at the problem of finding *robust* solutions for the pole balancing problem such that the controller works for *arbitrary* starting conditions. To this end, we will use an artificial neural network (ANN) as controller and CMA-ES to optimize its weights.

If you want, you can re-use your code from the previous exercise or start from the provided solutions that can be found on the web page of the lecture at `http://researchers.lille.inria.fr/~brockhof/advancedcontrol/exercises.php` .

1 Part I: Is the linear controller robust?

Rewrite your code for simulating the cart pole with the linear controller such that the return value of the simulation is the sum of all stable simulation steps within 10 independently restarted simulations for which the starting conditions are chosen uniformly at random ($-0.1 \leq x_0 \leq 0.1$ and $-\frac{\pi}{4} \leq \theta_0 \leq \frac{\pi}{4}$).

Questions

Is it still easy to find good parameter values for $k_1, \ldots, k_4$? To answer this question, write a script that runs the simulation 100 times with randomly chosen values for $k_1, \ldots, k_4$ as in the previous exercise and report how often you find an optimal solution. What exactly is the optimal value of the simulation output? Are the solutions you find also robust? To this end, re-evaluate (some of) the solutions that you found.

2 Part II: Implementing an Artificial Neural Network

Instead of the linear controller, in the following, we will use an artificial neural network as a controller. The ANN should take the four measured values $x_t, \dot{x}_t, \theta_t$ and $\dot{\theta}_t$ at each time step t as inputs and present the force applied to the cart as output. In addition to the single output neuron, two hidden neurons should be considered here as well as so-called "shortcut" connections that connect the inputs directly to the output neuron. Since the pole balancing problem is symmetric, we do not use any bias input to the neurons. Figure 1 shows the structure of the ANN. Other variables and parameters can be found in Table 1.

Now, we can start implementing the ANN.

Recommended Procedure:

a) Write a function (in MATLAB/Scilab/Octave) that implements the transfer function $t(x) = \frac{x}{|x|+1}$ of the neurons.

b) Then implement the above artificial neural network itself as a function that takes the input values and the weights of the network's connections as input and outputs the final output of the output neuron. You do not need to keep your implementation general (i.e. for an arbitrary number of inputs or any general structure).

Table 1: System parameters and variable names for the pole balancing problem.

Symbol	Name	Description
θ	Pole Angle	measured (in radians) relatively to the upright position, random initial value in $[-\frac{\pi}{4}, +\frac{\pi}{4}]$
$\dot{\theta}$	Pole Velocity	angular velocity of the pole in rad/s, initially 0
$\ddot{\theta}$	Pole Acceleration	acceleration of the pole in rad/s^2, initially 0
x	Card Position	measured relatively to the middle of the track (in m), random initial value in $[-0.1, +0.1]$
$\dot{x}$	Card Velocity	velocity of the cart (in m/s), initially 0
$\ddot{x}$	Card Acceleration	acceleration of the cart (in m/s), initially 0
g	Gravitational Acceleration	acceleration due to gravity (g =9.81 m/s^2)
m_c	Cart Mass	1.0 kg
m_p	Pole Mass	0.1 kg
l	Pole Length	distance from pivot to the pole's center of mass (l=1.0m ! note that this parameter changed from last exercise)
t	Time	measured in s
F_t	Force	force applied to the cart at time t (in N, always $F_t \neq 0$ for a bang-bang controller)
h	Track Limit	$\pm$2.4m from track center
r	Pole Failure Angle	$\pm\frac{\pi}{2}$ from vertical (value in rad ! note that also this parameter changed from last exercise)
τ	Time Step	discrete integration time step for the simulation ($\tau = 0.02$s)
T_{sim}	Number of Simulation Steps	number of simulation steps within which the pole should be stable ($T_{\text{sim}} = 10^3$, corresponds to a simulation time of $T \cdot \tau = 20s$)
F_m	Controller Constant	"normalization" constant of controller (set to $F_m = 100$N)
k_1, k_2, k_3, k_4	Controller Constants	internal constants of linear controller (in $[0, 1]$)

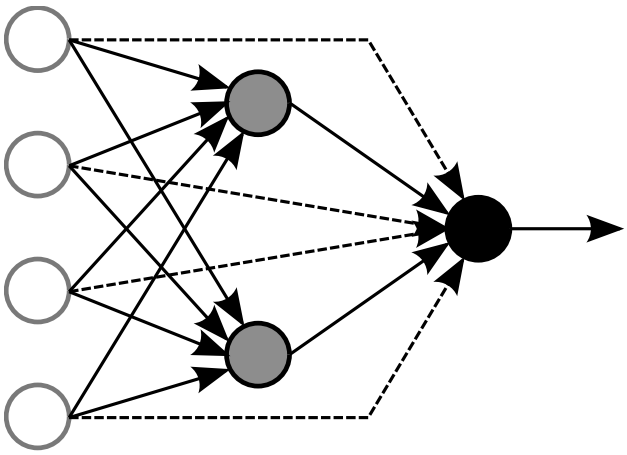

Figure 1: Layout of the artificial neural network used in this exercise. White circles: inputs; gray circles: hidden neurons; black circle: output neuron; dashed lines: shortcut connections.

3 Part III: Using CMA-ES

Last, we want to really optimize the problem of Part I with 10 independently (at random) chosen starting conditions to find a robust ANN controller.

Recommended Procedure:

a) First of all, rewrite the function of Part I to simulate the pole cart where you replace the linear controller by the ANN you implemented above. Since we want to use CMA-ES to optimize the network's weights, write your code such that your function is to be *minimized*. Call the function `myfitness` and make sure that it has all required parameters for the simulation but only takes the weights of the ANN as input. Be careful to check that all parameters are like in Table 1.

b) This function `myfitness` can then be directly plugged into CMA-ES. If not already done so, download the latest code from the web page of Nikolaus Hansen[1]. Do not forget to also get the file `plotcmaesdat.m` to look at the data. Using `fitness` as function name produces some errors because it is already defined in MATLAB.

[1] version > 3.6 can be found on `https://www.lri.fr/~hansen/cmaes_inmatlab.html`

c) To make your life easier, write a script that starts the CMA-ES algorithm with `myfitness` as the function to be optimized. Decide on adequate starting conditions for the algorithm (initialization + step size) and set the following options in your script:

```
opts = cmaes();
opts.Noise.on = 1;
opts.LBounds = 0;
opts.UBounds = 1;
```

Test

d) Start the algorithm, play with the options and starting conditions and observe the data (either with the option `opts.LogPlot=1;` or via `plotcmaesdat.m`).

Questions

a) How can we see that the optimization was successful?

b) Do the solutions, we found, result in a stable pole? What is the difference with this respect between x_{min} and x_{mean}?

c) What can we learn from the found weights? Are the solutions found in different runs of CMA-ES similar? In which sense?

Non-Mandatory Questions

a) In which way does the problem change if we delete connections in the ANN, e.g., the ones for which CMA-ES predicted a very low weight?

b) What happens if we turn off the noisy option of CMA-ES (`opts.Noise.on = 0;`)?

c) Are the results of the optimization different when we don't use random starting conditions but deterministic ones?