Evolutionary Robotics Laboratory Exercises

Evolutionary Algorithms (Part2)

Davide Zappetti (davide.zappetti@epfl.ch)
Anand Bhaskaran(anand.bhaskaran@epfl.ch)

Goal. In this lab you will investigate running an EA to evolve the parameters of an artificial neural network (ANN). While there are other ways to learn the weights of an ANN, using evolution is an effective means in many circumstances. At first we will evolve the weights of a simple feed forward neural network, and then we will look at evolving the weights of more complex, recurrent neural nets.

Notes:

- **You can refer to the instructions of the first E0 lab for information regarding the setup.**
- **As a prerequisite, we expect you to complete the first five exercise of the previous lab.**

Exercise 6

Sigmoid Function: A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve. All hidden and output neurons of these neural networks use this logistic activation function:

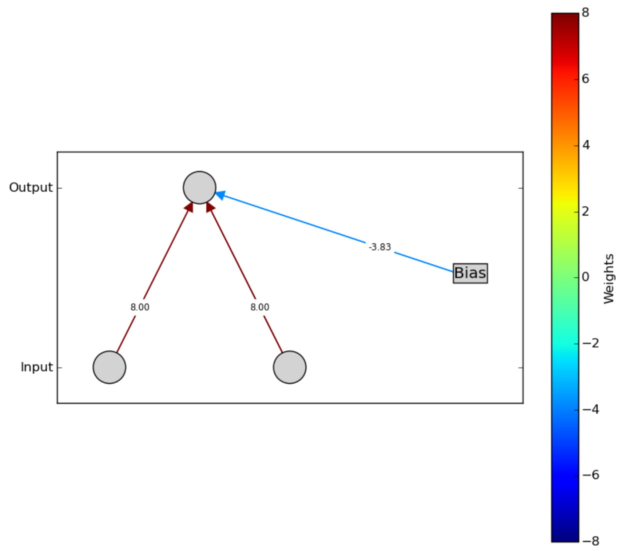
$$f(x) = \frac{1}{1 + e^{-x}}$$

Exercise 6.1

We begin by evolving the weights of a minimal neural network to solve the Or problem. That means we will use a neural network that has two inputs, and one output, which should produce the logical or function of the two input values:

Input 1	Input 2	Target Output
1	1	1
1	0	1
0	1	1
0	0	0

Run exercises_6_1.py to evolve the weights for this Or network. The fitness here is the sum of squared errors between the network's output and the target output across each of the four input patterns. If you see the best fitness approach zero (e.g. go under 0.1) then you have found a network able to solve this problem. This most likely looks similar to:



Here the neural network is depicted with its weights and biases shown by the corresponding color. You could check the value of the output neurons as: $f(\sum(input * weight))$. If you were able to solve the Or problem, look at the weights of the neural network and think about / compute how it behaves when given different input patterns. It is important to think about this now, because it will be difficult to keep track of what our neural networks are doing once we start using more complex topologies.

If you were not able to solve the Or problem, try modifying some of the parameters of the EA until you are able to do so.

Once you are able to solve Or, try solving the And problem instead (change *problem_class = Or* to be *problem_class = And*). Did the same parameters that you used for Or work for And as well? If not, modify them until you are able to solve And.

Now that we can solve Or & And, we will try something a little more challenging. Change the problem_class to be Xor, so that we are now trying to solve the Exclusive Or (Xor) function:

Input 1	Input 2	Target Output
1	1	0
1	0	1
0	1	1
0	0	0

This function has one small, but crucial difference from Or, as can be seen by comparing their truth tables.

Try running the code again after change the problem to Xor. Can you solve it? What if you change some parameters of the EA, such as the population size, crossover rate, max generations, etc.? If you are unable to solve it, why is that? Think back to what you saw in the lecture.

Here we offer you one new parameter to modify that you did not see in the previous exercises, and that is the number of hidden units of the neural network. Try changing this parameter from 0 to 1. Does this allow you to solve the problem? What if you change this value to 2 or some higher number? How many hidden neurons are required to solve this problem? Can you provide an explanation for why that is the case?

When you find a network that does compute Xor, once again see if you can understand how the network does so.

Exercise 6.2

So far we have looked at networks where you give them an input vector and they immediately produce an output. However, there are many tasks that we want to use neural networks for that should not base their output solely on one instance of time, but should instead respond to changing inputs over time. In this exercise we will begin to investigate these types of problems and you will see many more of them later in the course when we begin to use neural networks for controlling robots.

First, we start with a modified version of the Or problem, that is called "Temporal Or". While the basic Or problem involved evolving a neural network that would give a single output when provided two simultaneous inputs, in Temporal Or, there is only a single input neuron and the input values are provided in sequence. Therefore the network will have to remember the first input when seeing the second in order to output the correct value.

Try running exercise_6_2.py to solve Temporal Or. Is the EA able to solve this problem? If not try repeating some of your strategies from the previous exercise such as modifying the EA parameters or adding more hidden neurons. Can you now evolve a successful network?

If you still cannot evolve a network able to compute Temporal Or, notice that there is one new parameter that you can modify: "recurrent". This parameter is a Boolean flag, that says whether the network is recurrent (in this case an Elman network) or not (in which case it is feed forward). If you set recurrent to be True, can you now evolve a successful network? Why might recurrence be important for being able to solve a temporal problem such as this?

Once you have been able to evolve a network capable of solving Temporal Or, you can change the *problem_class* to *TemporalAnd* and repeat. Did the same parameters that solved Temporal Or also work for Temporal And? Why or why not?

Finally, change the *problem_class* to *TemporalXor* to attempt solving a temporal version of Xor. Run the code again. Are you able to find a successful network? If not, think back to what you just saw as well as the previous exercise. What combination of recurrence and number of hidden nodes is needed to solve Temporal Xor and why is this the case?

Check Points: Evolutionary Algorithms

By the end of this class you should be capable of answering the following questions! We will upload the answers for after today's class.

- Are there optimal parameters for an EA?
- What are the advantages and disadvantages of
  - Low/high mutation rates?
  - Low/high selection pressure?
- Based on these observations, do you think there is an optimal mutation rate for a biological organism? Do mutations typically improve or worsen the fitness of a biological organism? In which situations do you think low/high mutation rates are advantageous for a population of bacteria?
- What is the genotype and what is the phenotype in the problems considered in this lab?
- Why are hidden units sometimes needed for a neural network to solve a given task? What is the defining characteristic of problems that networks without hidden units are unable to solve?
- Why are recurrent connections needed to solve certain problems? What is the defining characteristic of problems that networks without recurrent connections are unable to solve? Are there problems that require recurrent connections and multiple hidden units?