COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS

Introduction to Artificial Intelligence

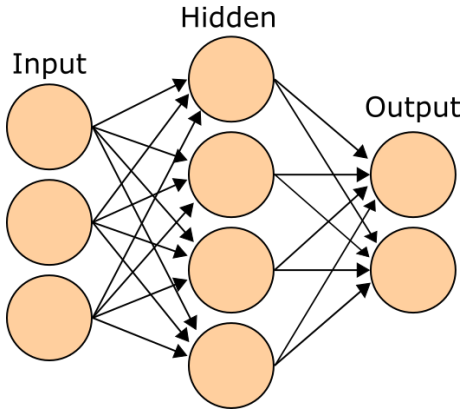
Exercise 10 - Genetically trained neural networks

November 23, 2022

10. MULTI-LAYER NEURAL NETWORK TRAINED WITH GA

We will program a multi-layer neural network, whose weights will be trained using genetic algorithm. The network will then control a car driving on a simple track.

Example of multi-layer model:



Besides input and output neurons, this model has another set of neurons in the middle, forming the so-called hidden layer. This network can be interpreted as a combination of two simple networks. First, the activation of the hidden layer is computed (output of the first layer), which is then used as input to the second layer. Its output is the output of the entire network.

Thus, this network has two weight matrices - W^{in}, connecting the input and the hidden neurons, and a W^{out}, connecting hidden neurons with the output.

To both input and hidden layer neurons, we also add bias, therefore the sizes of matrices are $(hid \times (n+1))$ and $(m \times (hid+1))$, where n is the dimensionality of the input, m is the dimensionality of output and hid is the number of neurons on hidden layer.

Computing output:

Output of two-layer network is computed in two steps. First, we compute the activation of the hidden layer:

$$\boldsymbol{h} = f_{hid}(\boldsymbol{W^{in}x'}),$$

where f_{hid} is the activation function of hidden layer, for example logistic sigmoid, and x' is input with bias. Next we can compute the network's output:

$$\boldsymbol{y} = f_{out}(\boldsymbol{W^{out}h'}),$$

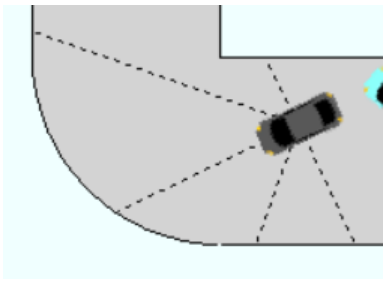
function f_{out} is the activation function of output and h' is the hidden layer activation with bias. To ensure that the output range is (-1, 1) **we use hyperbolic tangent as the output activation**.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Training:

The training using genetic algorithm consists of selecting the parents (individuals with the highest fitness), which are then used to create children (also subdued to mutation). These children along with the parents form the next generation.
In our case, right after the initialization of individuals we do not know their fitness, therefore we need to run the simulation first - i.e. we let the cars move according to their networks, until they hit the road boundary or the time runs out. After we have information about the length they travelled, we can sort them accordingly.

During this task we can expect the majority of the cars (mainly in earlier generations) to move randomly. For this reason we will not consider as parents the best half of the population (as done in exercise 4) but a much smaller portion (default is 4 out of 16 individuals, but we can tweak this number). The children are then generated by crossover of a random selection of two parents repeatedly, until we reached the original population size. This change significantly helps us to achieve better results.
But, how can we control the car during the simulation? For that purpose there is the already mentioned neural network, which takes as the input the data from sensors (along with other relevant data) and outputs the actions controlling the next movement of the car. In this task we will only consider cars with five sensors. All of them start in the middle of the car and cover the main *vision field* of the car, allowing it to collect meaningful data about the closest obstacles. The maximum length of a sensor is 200 and it is achieved only if there is no obstacle in front of the sensor. This is how the sensors look:



Task (2p):

Into the prepared skeleton *cars.py* code the key parts of the algorithm so that the cars gradually learn to drive on the road without colliding with the track boundaries.

- *MyCar (__init__)*
  - initialize the chromosome *self.chromosome* with random numbers (ideally close to zero).
  - design a neural network (by creating at least two weight matrices) and use the previously created chromosome to fill their values.
- *compute_output()*
  - create input into the network. The input consists of sensor lengths along with the current car speed and the current car turn (the angle of front wheels). Do not forget to scale these weights, so that the possible range of sensor values (<0, 200>) changes to something smaller (e.g. <0, 10>).
  - You can find the information about the sensors in the variable self.sensors, which is a list of individual sensors. Each element of this lis is in a form: $[[x1, y1], [x2, y2]]$ - the first and the second point of the sensor. From this we can easily calculate its length.
  - Another two input parameters are the car speed (self.speed) and the car turn (self.turn).
  - When creating the input to the network do not forget the bias!
  - Also in this function compute and return the output of the network. You can find the equations for computing the output in this text above. The number of output neurons in our case is 2. The first one controls the change of speed (thus the car goes slower/faster), whereas the second output neuron instructs the car to go to the left/right.
- *update_parameters()* - use the variable *instructions*, which contains the output of the network from previous step (compute_output) to update the wheel angle and the speed of the car.

Bonus (1):

Instead of two, use three layers (3 weight matrices) when creating the network to control the cars. The activation functions on the hidden layers can vary, but always use hyperbolic tangent on the output layer.
Followingly, compare the success of the cars when using two vs. three layered network. Also examine the influence of hyperparameters (mainly the number of neurons, activation functions). Other hyperparameters such as population size, number of parents, mutation probability, etc. also play a huge role in creating a fast evolving population, you can test their influence on the overall performance as well.
The key observations of this comparison should be written in a short readme, or as a comment in the submitted code.