Neural Networks and ABMs

Casimiro Carrino - Alberto Vele

September 11, 2015

Abstract

This project has the aim of joining together the use of neural networks in supervised learning and an agent based model. Precisely the simulation has been done using Netlogo linked to R; an agent must learn to find the target in a 5x5 space.

In the first phase the agent randomly explores the space and produces a dataset which must be sent to a multi-layer perceptron. Then a supervised learning algorithm is used to train the neural network on the produced dataset and finally the NN is used to move the agent in the space and let it find the target.

Contents

1 Introduction **2**

2 Neural Networks and Learning **3**
2.1 Multilayer Networks 3
2.2 Supervised Learning 4

3 Simulation and Code **5**
3.1 Data Production 5
3.2 Training the Neural Network 8
3.3 Moving the Agent with the Neural Network 9

4 Results and Considerations **10**
4.1 Further Developments 10

1 Introduction

Agent based models can manage the issue of finding under what conditions and why in social and economic systems not expected emergent behaviours occur. The first aim of the research in this kind of models is to find methods that allow for building systems made of single autonomous agents that are able to make the whole system have a global behaviour, even if they are aware only of the most near environment and have only few capabilities. In other words, the approach of the agent based models in fields like social sciences or economy has the ambition to disassemble the emergent properties in the elementary constituents.

This models are obviously useful to do experiments and simulation; as simple as possible simulations are required in order to keep in consideration the fact that agents cannot have unlimited rationality and knowledge of the system in which they are immerged. Agents must answer to a precise set of rules of interaction, and in this context emergent unexpected phenomenons can occur.

The use of artificial neural networks is of great interest because of the aforesaid necessity to keep rationality of the agents out of the modelization; more precisely the agents must be able to learn from the environment in which they are and from the actions of other agents. So, neural networks allow to simulate well a wide range of typically human behaviours.

2 Neural Networks and Learning

2.1 Multilayer Networks

Generally speaking a neural network is a massive system of parallel distributed processing elements connected in a graph topology. The interest in neural network stems from the wish of understanding principles leading in some manner to the comprehension of the basic human brain functions, and to building the machines that are able to perform complex tasks. Essentially, neural network deal with cognitive tasks such as learning, adaptation, generalization and optimization. It is able to produce non-linear associations among an output vector and an input vector through certain parameters called weights. This is due to the activaction function mechanism present within each node of the network in analogy with the biological neurons in the brain. Indeed a neuron in the network represent a sort of processing unity; a vector input $\underline{x} = [x_1, x_2, ..., x_n]$ that can reach it is combined linearly with a set of weights $\underline{w} = [w_1, w_2, ..., w_n]$ and then computed using a certain activation function, tipically a sigmoid of the form

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

. The sigmoid confines the result in the interval {0,1} so working as a general threshold function. Figure 1 shows the example of a single neuron with n incoming inputs from another neuron, the related n weights, the sigmoid function f and the output y.

In the case of simple multilayer networks nodes are placed on three different layers: an input layer, an hidden layer and an output layer. The network is homegeneous in a manner that every neuron works in the same way, but the presence of an intermediate layer guarantees, after choicing opportunely the weights, a specific relation input-output necessary to solve problems of non-linear classification. It is possible to see this relation in a mathematical form for a multilayer network with 4 input neurons, 3 hidden neurons and 2 output. So, given an input vector $\underline{x} = [x_1, x_2, ..., x_n]$ and a matrix A with rows representing for each node the weigth vectors among the input and the hidden layer and analogously a matrix B for the weigth connections between the hidden and the output layer, the input-output relation is:

$$\underline{y} = f(\mathbf{B}\, f(\mathbf{A}\underline{x})]) \tag{1}$$

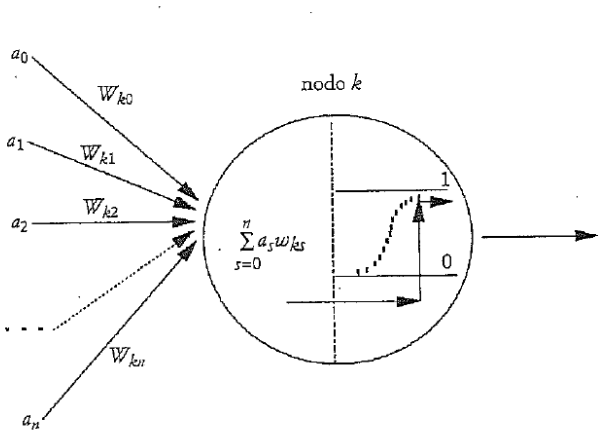

Figure 1: Single neuron computation

The above non-linear function define completely the input-output information process.

2.2 Supervised Learning

Learning in the neural network can be supervised or unsupervised. Supervised learning uses classified pattern information, while unsupervised learning uses only minimum information without preclassification. In our work a supervised approach has been applied.

The most important ability of multilayer neural networks is learning how to associate input and output; a training set is introduced for accomplish this attempt.

Multilayer perceptrons are layered feed-forward networks typically trained with back-propagation. These networks have found their way into countless applications requiring static pattern classification. Their main advantage is that they are easy to use, and that they can approximate any input/output map in a general data set called training set. Precisely, in supervised learning a spefic criteria must be defined, which indicates, for a given input in the training set, the correct output that the network must compute during the training. Realizing this exact corrispondence means that the network is able to learn. In order to account this, a certain error function $E = \sum_n (y_n - t_n)^2$

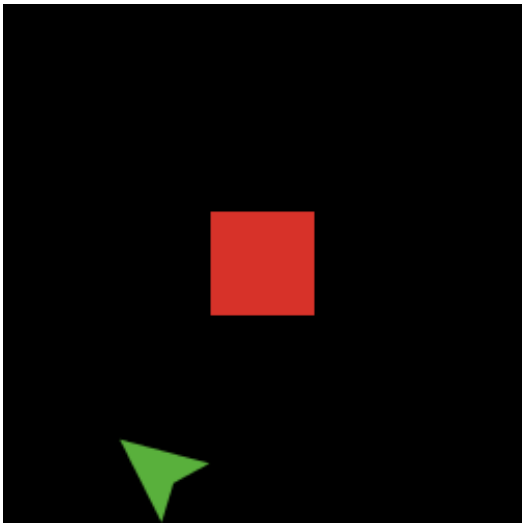

Figure 2: Space

is defined as the difference between the correct output or target t_n and the computed output y_n; then, minimization of this quantity became a following criteria with which modify the internal weights structure that codify the non-linearly relation input-output. Backpropagation algorithm is the most used algorithm in the multilayer networks operating in the previous way.

3 Simulation and Code

3.1 Data Production

The aim of the work is making an agent be able to reach a special point of the space in which it can move with less steps possible. To do this, in the first phase of the simulation the agent is left free to explore the whole space, that is composed of 5×5 patches, in a NetLogo environment. It can move from its position randomly choosing the 8 neighbour patches or stay in its current position, for a total of 9 possible movements, one step at a time; the total possible combinations are 9×25. The space has one special patch, namely target, that must be reached by the agent to realize the goal; see Figure 2.

The agent must be "conscious" of its position and movement at each step and also evaluate a sort of fitness of the action, properly the distance from the target. These informations must be collected in a suitable dataset.

In each step the distance of the agent from the target is misured; we can have a minimum distance equal to zero if the agent reaches the target and a maximum distance equal to the euclidean distance from the target if the agent stays on one of the patches in the corners.

So, data are appended to two lists: input and output. Input has the information related to position and movement and in output we append the distance from the target, suitably scaled in order to have always positive values less than one. Scaling the output is strictly important because the neural network works only with values in {0,1}. An interesting part of the work has been to codify position and movement in such a way to have a vector of 25 binary components for the position and a vector of 9 binary components for the movement, as can be seen in the next lines of NetLogo code:

```
to training_set

    ask turtles [
                set moves [[-1 1] [0 1] [1 1] [1 0] [1 -1] [0 -1] [-1 -1] [-1 0] [0 0]]
                let movement one-of moves
                ;setxy (xcor + item 0 movement) (ycor + item 1 movement)

                ;;codifico posizione e movimento nella lista Input da mandare alla rete neurale
                set cod_moves [0 0 0 0 0 0 0 0 0 ]
                if item 0 movement = -1 and item 1 movement = 1 [set cod_moves replace-item 0 cod_moves 1]
                if item 0 movement = 0 and item 1 movement = 1 [set cod_moves replace-item 1 cod_moves 1]
                if item 0 movement = 1 and item 1 movement = 1 [set cod_moves replace-item 2 cod_moves 1]
                if item 0 movement = 1 and item 1 movement = 0 [set cod_moves replace-item 3 cod_moves 1]
                if item 0 movement = 1 and item 1 movement = -1 [set cod_moves replace-item 4 cod_moves 1]
                if item 0 movement = 0 and item 1 movement = -1 [set cod_moves replace-item 5 cod_moves 1]
                if item 0 movement = -1 and item 1 movement = -1 [set cod_moves replace-item 6 cod_moves 1]
                if item 0 movement = -1 and item 1 movement = 0 [set cod_moves replace-item 7 cod_moves 1]
                if item 0 movement = 0 and item 1 movement = 0 [set cod_moves replace-item 8 cod_moves 1]
```

```
set pos [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
if pxcor = 0 and pycor = 0 [set pos replace-item 0 pos 1]
if pxcor = 0 and pycor = 1 [set pos replace-item 1 pos 1]
if pxcor = 0 and pycor = 2 [set pos replace-item 2 pos 1]
if pxcor = 0 and pycor = 3 [set pos replace-item 3 pos 1]
if pxcor = 0 and pycor = 4 [set pos replace-item 4 pos 1]
if pxcor = 1 and pycor = 0 [set pos replace-item 5 pos 1]
if pxcor = 1 and pycor = 1 [set pos replace-item 6 pos 1]
if pxcor = 1 and pycor = 2 [set pos replace-item 7 pos 1]
if pxcor = 1 and pycor = 3 [set pos replace-item 8 pos 1]
if pxcor = 1 and pycor = 4 [set pos replace-item 9 pos 1]
if pxcor = 2 and pycor = 0 [set pos replace-item 10 pos 1]
if pxcor = 2 and pycor = 1 [set pos replace-item 11 pos 1]
if pxcor = 2 and pycor = 2 [set pos replace-item 12 pos 1]
if pxcor = 2 and pycor = 3 [set pos replace-item 13 pos 1]
if pxcor = 2 and pycor = 4 [set pos replace-item 14 pos 1]
if pxcor = 3 and pycor = 0 [set pos replace-item 15 pos 1]
if pxcor = 3 and pycor = 1 [set pos replace-item 16 pos 1]
if pxcor = 3 and pycor = 2 [set pos replace-item 17 pos 1]
if pxcor = 3 and pycor = 3 [set pos replace-item 18 pos 1]
if pxcor = 3 and pycor = 4 [set pos replace-item 19 pos 1]
if pxcor = 4 and pycor = 0 [set pos replace-item 20 pos 1]
if pxcor = 4 and pycor = 1 [set pos replace-item 21 pos 1]
if pxcor = 4 and pycor = 2 [set pos replace-item 22 pos 1]
if pxcor = 4 and pycor = 3 [set pos replace-item 23 pos 1]
if pxcor = 4 and pycor = 4 [set pos replace-item 24 pos 1]

set input lput cod_moves input
set input lput pos input
;show patch-here

;;muovo le tartarughe dopo il movimento
setxy (xcor + item 0 movement) (ycor + item 1 movement)

;;vettore di Output che rappresenta la distanza dal target
ifelse [pcolor] of patch-here = yellow [set output lput 10  output] [set output lput distance patch 2 2 output]
```

These lists have to be converted in matrices that can be used by R to later do the training of the network. The number of rows of the matrices coincides with the number of steps the agent do in the exploring phase (number of examples of the training set). The number of columns is $34(9 + 25)$ for the input matrix and 1 for the output matrix. The R code for the creation of the matrices is reported below:

```
to connectR
   rserve:init 6311 "localhost"
end


;;creo le matrici di Input e Output da mandare ad R. Il numero di righe è il numero di prove.
to create_matrix

   ;;Creo la matrice di Input
   rserve:put "Input" input
   rserve:eval "Input_v <- c()"
   rserve:eval "for (i in Input) {for (j in i) {Input_v <- c(Input_v,j)}}"
   rserve:eval "Input_M <- matrix(Input_v, ncol = 34, byrow=T)"
   ;rserve:eval "print(Input_M)"

   ;;Creo la matrice di Output
   rserve:put "Output" output
   rserve:eval "Output_M <- matrix(Output, ncol=1, byrow=T)"
   rserve:eval "Output_M[,1] <- (Output_M[,1] - min(Output_M[,1]))/(max(Output_M[,1])-min(Output_M[,1]))"
   ;rserve:eval "print(Output_M)"

end
```

3.2 Training the Neural Network

Training the neural network requires to link NetLogo to R; precisely we embedd R in Netlogo. Hence these two programs have to communicate and this is possible through the use of a proper NetLogo extension. The Rserve-Extension of NetLogo has been developed to make the functionality of R available in NetLogo. Both extensions make it possible to send NetLogo variables to R and to get results from R back to NetLogo. They include functions for sending variable values of agents to R, which are then transformed to appropriate R data structures. Further details about how to link the two software are available in the references.

Once we have done this procedure, the phase of training can start. Matrices are obtained and their values are properly scaled using R, as already seen in figure 4. The training is accomplished through the backpropagation algorithm, and this is realized by the $library(nnet)$ of R. The arguments of the function $nnet$ are the input matrix, the output matrix, the size, which corresponds to the number of hidden neurons, and the maximum number of iterations allowed. The choice of a proper number of hidden neurons doesn't answer to a rigid rule; anyway it is important to keep in consideration the fact that too many hidden neurons could make the network overfit data.

3.3 Moving the Agent with the Neural Network

In the training phase the network has learned the corrispondence between the input vector data and the output data. In our simulation this means that for each movement and position reached after the movement, the distance from the target gets out, not by calculating it, but only using the computed output provided by the network. In the third part the agent is moved through the application of the trained network in order to exihibit the capacity of reaching the target. The agent starts in a random position in the space and for each possible movement a list of output predicted by the network is created; then it moves in the direction corresponding to the minimum value of the predicted output in the list. The part of the NetLogo code related to the prediction of the neural network and the moving procedure for the agent follows:

```
;;predizione della rete neurale sul vettore di input dataForPrediction
rserve:put "input_prediction" input_prediction
rserve:eval "input_prediction_v <- c()"
rserve:eval "for (i in input_prediction) {for (j in i) {input_prediction_v <- c(input_prediction_v,j)}} "
;rserve:eval "print(input_prediction_v)"
rserve:eval "prediction <- predict(network,input_prediction_v)"
rserve:eval "print(prediction)"

;;per ogni movimento il valore della predizione viene salvato in un vettore Predictions
set output_predictions lput rserve:get "prediction" output_predictions
]

    to moving

        ;; estraggo la posizione del minimo dalla lista Predictions
        show output_predictions
        set position_min_distance (position min output_predictions output_predictions)
        show position_min_distance

        ;; muovo le tartarughe nella direzione del movimento che minimizza la distanza dal target
        ask turtles
                    [
                     setxy (xcor + (item 0 item (position_min_distance) moves))
                           (ycor + (item 1 item (position_min_distance) moves))
                    ]
    end
```

We notice that this procedure does not involve the "renormalization" of the output list cause the predicted movement is not based on absolute values, but only the relative minimum in the list is important.

4 Results and Considerations

Let's now see in which conditions the agent is able to reach the target, so showing a sort of intelligent behaviour. The most important parameter in our simulation is the number of steps that the agent do in the first phase of exploration, that is the number of examples receiving by the neural network. We verify the value of the steps for which the agent realizes its goal, for any initial condition, is centered around 1000. Setting a number of steps less than 1000 the agent in mean still shows a good behaviour, but there are some initial conditions for which it seems to be "confused" in trying to reach the target. However, it is important to note that for a less number of steps the network nevertheless learns(the error function E has an acceptable value), but there is a non-zero probability that the agent doesn't reach the target because of the uncomplete exploration of the input space. Reasonably, in a 5×5 environment with an input space dimension of 9×25, the value of 1000 training examples results to be sufficient to make the simulation successful, as experimentally verified.

Furthermore the training procedure we have made up is strongly dependent on the position intially choosen for the target. In other words changing the target position after the training phase causes the failure of the simulation.

4.1 Further Developments

There is another way of performing the simulation, that is in a space 5×5 with a target patch, but in addition an obstacle that the agent must avoid to reach the target. The only improvement required is that the values of the output for the obstacle patch has to be greater than the maximum distance from the target in the space. The coded line is shown below:

```
;;vettore di Output che rappresenta la distanza dal target
ifelse [pcolor] of patch-here = yellow [set output lput 10  output]
                                       [set output lput distance patch 2 2 output]
```

Even in this variant of the simulation the agent exihibits the capability of reaching the target avoiding the obstacle with the less number steps possible. As in the case without obstacle, the 1000 collected examples in the training set are sufficient for accomplishing the aim. This entails that the presence of

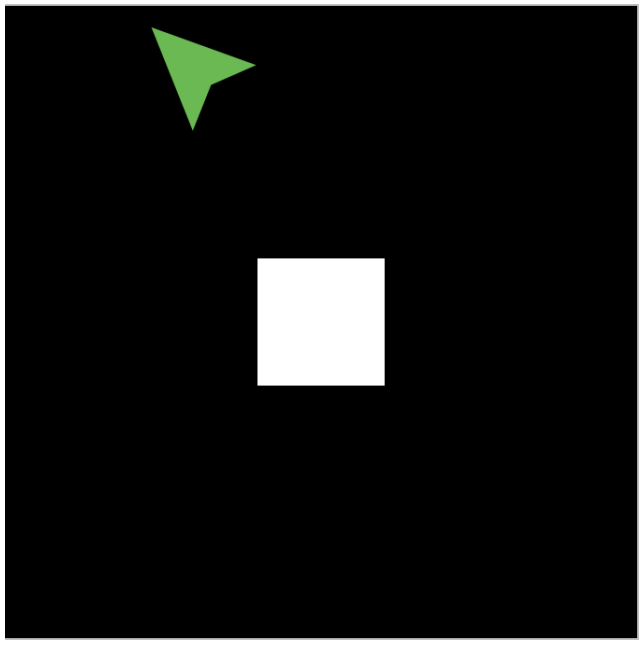

Figure 3: Initial position

an obstacle does not affect the training phase.

Finally we could try to vary the dimension of the environment to test the learning process. In this way we could find which is the suitable threshold ratio among the number of training examples and the dimension that allows for the success of the simulation.

In Figures 3-9 we can clearly see how the agent is able to reach the target both without an obstacle and with an added obstacle.

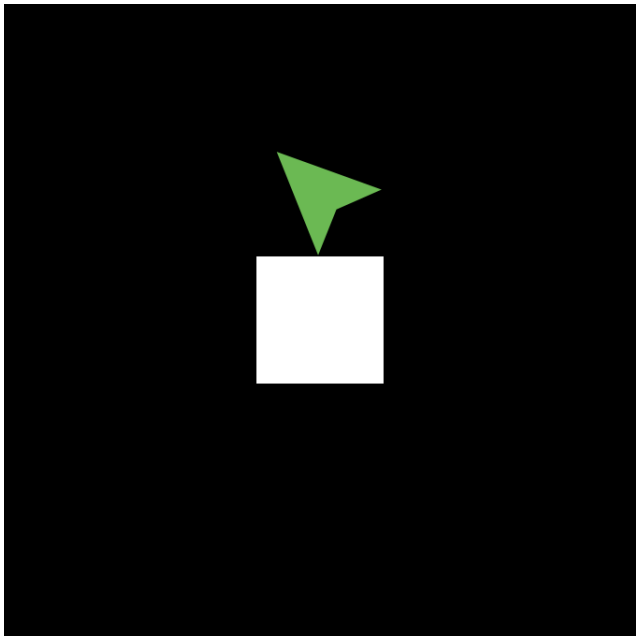

Figure 4: Second step

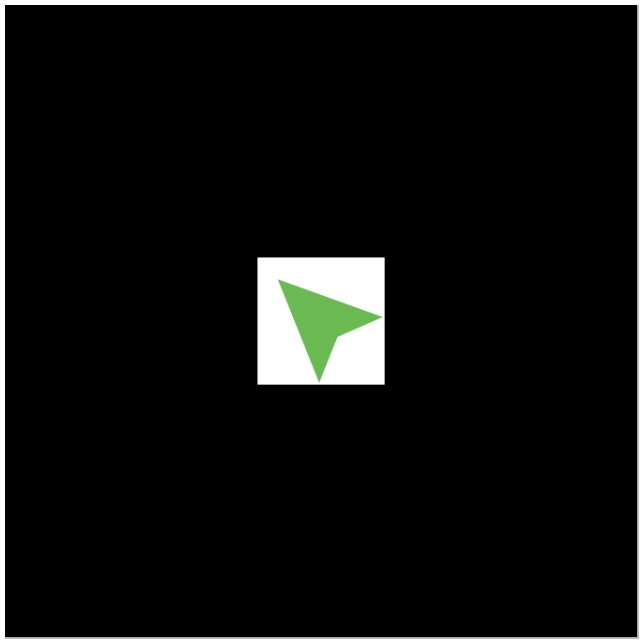

Figure 5: Target reached

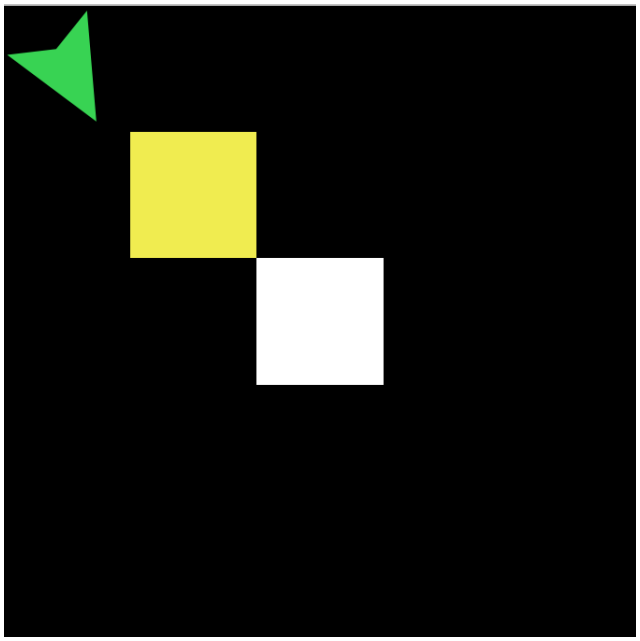

Figure 6: Initial position with obstacle

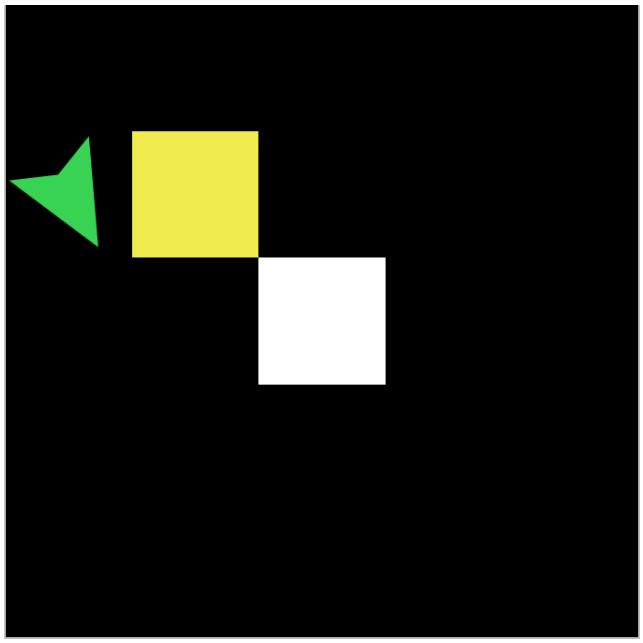

Figure 7: Second step with obstacle

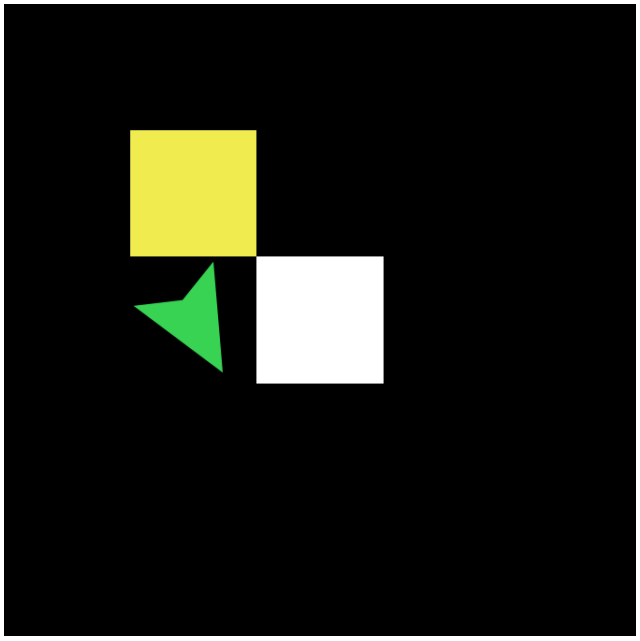

Figure 8: Third step with obstacle

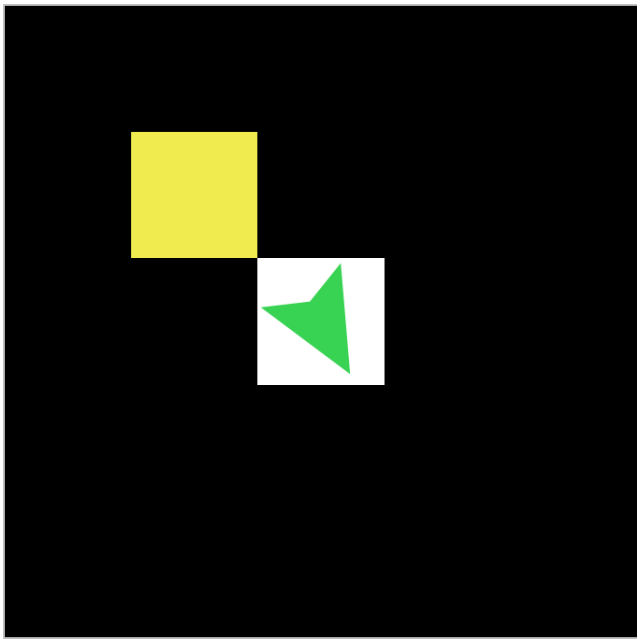

Figure 9: Target reached with obstacle

References

[1] P.Terna(1995), Reti neurali artificiali e modelli con agenti adattivi, Rivista italiana di economia, 0, pp. 71-106.

[2] Jan C. Thiele, Winfried Kurth and Volker Grimm, Agent-Based Modelling: Tools for Linking NetLogo and R, Journal of Artificial Societies and Social Simulation 15 (3) 8, 2012.

[3] Packages NNet: https://cran.r-project.org/web/packages/nnet/nnet.pdf

[4] Janglov, D. / Neural Networks in Mobile Robot Motion, pp. 15-22, International Journal of Advanced Robotic Systems, Volume 1 Number 1 (2004), ISSN 1729-8806