The Basics of Implementing Intelligent Social Networks

Stefan Cosmin Popa(✉), Marius Constantin Popescu, and Antoanela Naaji

Vasile Goldis Western University Arad, 94-96 Revolutiei Blvd., Arad, Romania
popastefancosmin@gmail.com,
{mpopescu,anaaji}@uvvg.ro

Abstract. The paper studies the implementation of a web software application in node.js language, with a view to analyzing neural networks and artificial intelligence in a virtual environment, with application in social networks. The software allows experimenting on the capacity of virtual entities to learn to act as a group or individually, avoid the margins of the environment where they are located, or obstacles, and learn to feed in order to live. The application is a novelty in the sense that, until recently, i.e. the emergence of node.js, implementing the concept of neural networks in a scripting language was not possible. This concept can be developed in practical applications such as social networks, search engines with artificial intelligence, personal assistants, e.g. Siri etc. The final aim is to obtain an efficient software solution, with application in social networks, which can evolve over time.

Keywords: Neural networks · Scripting · Node.js · Web application · Social networks

1 Introduction

The application described in the paper is based on an original design with an interface where artificial intelligence activity can be observed. There are no similar works in the literature, with a descriptive approach to artificial intelligence, but only mathematical related ones, less accessible to inexperienced programmers. The paper seeks to facilitate the process of developing applications based on artificial intelligence using *node.js*, which is a popular language, as it uses *JavaScript* on both the server and the client part; this means that developers need only program in one language, across all layers. The application uses the following *front-end* technologies: HTML5 (Hyper Text Markup Language 5) [1], CSS3 (*Cascading Style Sheets 3*) [2], JavaScript [3], HTML5 Canvas [4]. The *node.js* language is a runtime environment with libraries running JavaScript applications outside the browser. *Node.js* and the Synaptic library [5] are used in the *back-end*. The application is hosted and was tested on a *Debian 8 personal* server and runs in a modern web browser (recommended, Google Chrome v49, Microsoft Edge v25, Mozilla Firefox v45).

Usually, *node.js* is employed to run server applications in real time and excels in the performance of using *non-blocking* I/O and asynchronous events. A real advantage

V.E. Balas et al. (eds.), *Soft Computing Applications*, Advances in Intelligent Systems and Computing 634, DOI 10.1007/978-3-319-62524-9_2

is in the fact the same language can be used to construct the *backend* of the *frontend*, already constructed in JavaScript. Although still in its beta phase, the language is used by many large companies (Microsoft, Yahoo!, LinkedIn, eBay, The New York Times, etc.), on account of being stable and efficient. Few programming languages have known such rapid development as *node.js*. It can be easily seen that most of the resources found on GitHub are in JavaScript or *node.js*. As such, it is a good opportunity for developing applications based on artificial intelligence, which for the time being are not numerous. The *node.js* programming language is fairly reliable for creating services e.g. API (A*pplication* P*rogramming* I*nterface*), which would allow *multithreading* – the application's capacity to provide simultaneous responses. The beneficial part of the *node.js* language is the immense amount of code, already presented on GitHub. A developer only has to combine the resources, in order to save time when creating a new application. The following chapters will present a few basic terms and their implementation using the developed software application in intelligent social networks.

2 Neural Networks

A neuron is the fundamental unit of a neural network with inputs, processing and outputs, whose operation depends on a serious of synaptic weights, which, applied to input data, generate outputs through the activation function [6]. Neural networks consist of multiple nodes, imitating the biological neurons of the human brain, connected by links, and which interact with each other (Fig. 1). Each node can assume input data and perform simple operations with these data, the outcome being transmitted to other neurons. The output node is called value node or activation node. Each link is assigned to a synaptic weight, neurons being thus capable of learning, changing the value of the link's synaptic weight in a similar way to biological synapses.

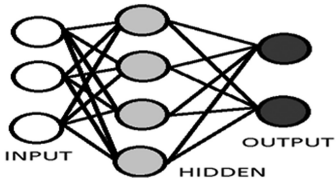

Fig. 1. Example of a simple neural network

Each line is a link between two neurons, and links can have a certain direction depending on the type of the diagram, indicating the flow of information (Fig. 1). Each link has a synaptic weight, an integer (variable) controlling the signal between two neurons. If the network generates a "good" or "desired" *output*, then there is no need to adjust the synaptic weight. If, however, it should generate an "unsatisfactory", "undesired" output or an error, then the system changes the synaptic weight, in order to

improve future outcomes. The most widely used algorithm for neural networks is the *back propagation* algorithm [7]. This is a learning algorithm in which the network's neurons learn from examples. Thus, if the algorithm is assigned a task to perform, it will change the synaptic weight of the network at the end of learning, to produce a specific *output* from *input* data. The algorithm adjusts the synaptic weight using as

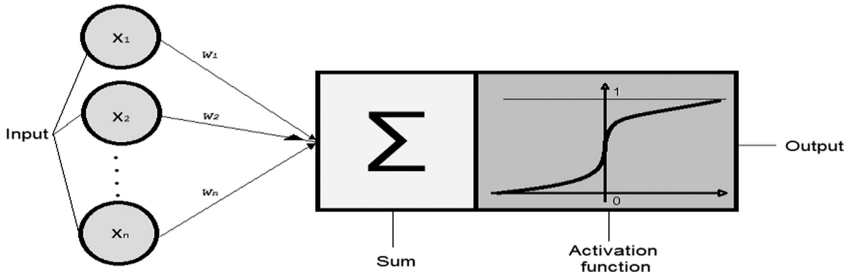

Fig. 2. Operation of a perceptron

calculus the minimization of the *Gradient Descent* function [8]. Usually, the activation function is a sigmoid function (Fig. 2).

In the case of *FeedForward* network, the propagation of information is unidirectional and, for this reason, it is one of the simplest neural networks, namely, a neuron sends information to another neuron from which it does not receive information. Such networks do not have *feedback* loops and are used to generate, recognize or classify forms having fixed inputs and outputs.

3 Implementing the Application

The *Synaptic.js* library [9] was used to implement the application. Like the other options, namely *Brain.js* and *Mind.js*, it is possible to create a complete neural network using only a few lines of code. Inside the library there are several preconfigured architectures of neural networks, which underlay the software described as follows. The application simulates a virtual set defined by the margins of the HTML5*canvas*, in which the entities created will explore a set/map, will have obstacles to avoid, will feed and will die. As follows we will detail the application's options and mode of operation. Entities are divided into two classes, one of entities learning together and behaving as a group, and the other one in which entities work individually. As such, it can be observed that, when initializing the application, all entities appear in completely random places on the map (Figs. 3 and 4). As they converge, red ones tend to form a group and move in a group (Fig. 4).

Fig. 3. Generating entities

Fig. 4. Red entities use the cohesion function

This is possible due to using the `cohesion` function, which defines the cohabitation between the members of a group, defined as follows:

```
cohesion:function(neighboors)
     { var sum=new Vector(0,0); var count=0;
       for (var i in neighboors)
       {if (neighboors[i]!=this)//&&!neighboors[i].special)
            {sum.add(neighboors[i].location);count++;
             }
            }
            sum.div(count);return sum;
       }
```

The function will find all the neighbours within its range that pertain to the same species and add them to the group. This way the neural network, which defines the entity, learns what its neighbours and what its obstacles are. The method is known as neural network *training*. In this case, the entities will be defined as follows:

```
function Creature(world,x,y)
{ this.network=new Architect.Perceptron(40,25,3);
  this.world=world;this.mass=.4;
  this.maxspeed=3;this.maxforce=.2;
  this.lookRange=this.mass*200;
  this.length=this.mass*10;
  this.base=this.length*.5;
  this.HALF_PI=Math.PI*.5;
  this.TWO_PI=Math.PI*2;
  this.location=new Vector(x,y);
  this.velocity=new Vector(0,0);
  this.acceleration=new Vector(0,0);
  this.color="#C70E4C";
}
```

Thus, entities are assigned a neural network (a perceptron), which has 40 *input*s, 25 *hidden layers* and 3 *outputs*. Likewise, each entity must be assigned a set/world. As the laws of mechanical physics will apply, each entity will have a mass, so that when a collision occurs with other objects in the map, it will move at maximum speed, therefore having maximum collision force. For an entity to be able to see, the `lookRange` variable will be assigned to it, which will be equal to its mass multiplied by 200; therefore, the entity will have a visual range of 80 pixels around it. The location of the entity will be given by the set, and color is a variable that can be set as needed. Going back to location, it is noted that it instantiates a vector function. This function is useful for calculating the position of the entity relative to coordinates, angle of impact, mass and weight. Below are a few lines of code from the function:

```
angle:function()
     { return Math.atan2(this.y,this.x);
      },
       setMag:function(m)
       { var angle=this.angle();
         this.x=m*Math.cos(angle);
         this.y=m*Math.sin(angle);
         return this;
       },
       setAngle:function(a)
       { var mag=this.mag();
         this.x=mag*Math.cos(a);
         this.y=mag*Math.sin(a);
         return this;
       },
       rotate:function(a)
       { this.setAngle(this.angle()+a);
         return this;
       },
       limit:function(l)
       { var mag=this.mag();
         if(mag>l)this.setMag(l);
         return this;
       },
```

The entity is displayed in the set using the `moveTo` function, which assumes the coordinates in the memory locations and, depending on the dangers on the map, will apply the laws of mechanical physics for each element, and will establish the angular displacement, alignment collision or coexistence. For example, `var separation` will use the `separate` function to separate the entities in the set from those of a different color, a variable which will be assigned to a `force` variable representing a vector, and then this variable will be applied to the entity. The same also happens to the other variables, defined in the previously described function.

```
moveTo:function(networkOutput){
 var force=new Vector(0,0);
 var target=new Vector(networkOutput[0]*this.world.width,
networkOutput[1]*this.world.height);
var angle=(networkOutput[2]*this.TWO_PI)-Math.PI;
var separation=this.separate(this.world.creatures);
var separation2=this.separate(this.world.obstacles);
var
alignment=this.align(this.world.creatures).setAngle(angle);
var cohesion=this.seek(target);
force.add(separation);
force.add(separation2);
force.add(alignment);
force.add(cohesion);
this.applyForce(force);
},
```

The `draw` function will access another function of the entity, namely `update`. The function calculates the location where the entity will be drawn in the `canvas`, depending on speed and current location:

```
draw:function()
 { this.update();
  var ctx=this.world.context;ctx.lineWidth=1;
  var angle=this.velocity.angle();
  x1=this.location.x+Math.cos(angle)*this.base;
  y1=this.location.y+Math.sin(angle)*this.base;
  x2=this.location.x+Math.cos(angle+this.HALF_PI)*this.base;
  y2=this.location.y+Math.sin(angle+this.HALF_PI)*this.base;
  x3=this.location.x+Math.cos(angle-this.HALF_PI)*this.base;
  y3=this.location.y+Math.sin(angle-this.HALF_PI)*this.base;
  ctx.lineWidth=2;ctx.fillStyle=this.color;
  ctx.strokeStyle=this.color;ctx.beginPath();
  ctx.moveTo(x1,y1);ctx.lineTo(x2,y2);
  ctx.lineTo(x3,y3);ctx.stroke();ctx.fill();
  },
```

The update function, as defined in the code, will access the boundaries function of the entity, which will define the limits and add extra acceleration to the speed of this entity [9]:

```
update:function()
     { this.boundaries();
       this.velocity.add(this.acceleration);
       this.velocity.limit(this.maxspeed);
       if(this.velocity.mag()<1.5)
      this.velocity.setMag(1.5);
       this.location.add(this.velocity);
       this.acceleration.mul(0);
       },
```

The seek function will establish the speed of the group, depending on the location of neighbours in the group and is defined as follows:

```
seek:function(target)
    { var seek=target.copy().sub(this.location)
      seek.normalize();seek.mul(this.maxspeed);
      seek.sub(this.velocity).limit(0.3);
      return seek;
      },
```

Two functions, blastoff and loop, are used to implement the set, or define the map. The blastoff function contains: the canvas variable, which will find the div with the *canvas* id in the html file and transform it into this set; the ctx variable, which defines the context of the application, i.e. 2-dimensional; the obs variable, which represents the number of obstacles; the num variable, which represents the number of entities in each group; the fps variable, which represents the application's display rate; the world variable, which will have width, height and an *array* of entities, an *array* of obstacles, an *array* of food and the option of context. Cycling will be conducted in order to populate the map with entities, obstacles and food, meaning one for to each variable of values, assigning to the map the entity, obstacle or food, with the key from the for at random values x and y. Likewise, this function is used to calculate the new location and the angle of incidence for each entity. The loop function contains settings for the context (genre, color, form, size), the update function for entities (obstacles or food), *foreach* for obstacles, entities and food (in which the *input* for the neural network will be set, where the learning rate and target coordinates are), the function for drawing obstacles, entities and food. The function will be recalled in order to make it recursive. Each entity, when encountering food, will assume the mass/volume of food, while it disappears from that location and reappears in a different spot on the map. To avoid an excessive increase in the number of entities the life function will be utilized, which will decrease the mass of the entity by 0.1 per minute, until it reaches 0; in this situation the die function will be called, which destroys the entity and transforms it into food. An interesting thing to observe in the entities' feeding process is that the entities which move in a group and learn together

have a higher survival rate (as they learn from the others that mass is essential for survival), whereas individual entities – if they do not find food – will die, and those that do will learn how to survive.

4 Future Work

Once the previously described application has been developed, further work might include creating a platform matching people according to certain criteria, as well as a search engine, in which the input prediction will be based on the same concepts as those presented in the paper. It is worth mentioning that there is already a similar concept of using artificial intelligence to improve search, developed by Google, capable of learning and improving results, utilizing data derived from people [10]. Short-term development possibilities refer to implementing predictive search, while attempting to implement finding gender- or age-related similarities in the search engine. The medium- and long-term development possibilities refer to optimizing the source code, to developing a support service or an automated registration function, using existing accounts from the social network Facebook and Google APIs, to increase the number of users. Likewise, development refers to training neural networks and collecting comparative data from other conventional matching systems, aiming to analyze performance, or to add a satisfaction form, which will help improve the application. Another perspective consists of improving application security and preventing server overload due to robot-generated ports, or optimizing the application for mobile devices (smartphones, tablets). The *node.js* application consists of an *http* server and a *restful API*. So far, the following *node.js* modules/libraries [11] have been used to create this application:

- `express` is a module for *node.js* allowing to create a web application and post it on one of the available *http* ports;
- `body-parser` is a module transmitting post *requests* to the application;
- `ejs` is a *template manager* for *node.js*, which allows adding a special code into an *html* sheet, which can display the variables in the *template*;
- `mongoose` is a module linking the application to the `mongodb` database [12, 13];
- `passport` are modules needed to create easier and more secure login;
- `bcrypt-nodejs` is a module used to secure the application and passwords;
- `cookie-parser` is a module that helps in using `cookies` for the session;
- *Synaptic* is a library for the neural network [9].

The application has the following structure [11, 14]: `app, views, config, node-modules, routes` and `auth, models, user, index, login, register, dashboard, profile, search, database.js`. The user will be able to edit their profile, look for friends using their desired qualities for subsequent sorting, or may find already connected friends by name, using a conventional search function.

By this stage we developed the search engine, based on Hopfield Architecture, which serves as content-addressable memory. Networks are trained to “remember” patterns, and when new patterns are inputted, it returns the most similar pattern for which it was trained.

```
var hopfield = new Architect.Hopfield(10) // creation of a
network for 10-bit patterns

// training the network with two different patterns
hopfield.learn([
[0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
[1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
])
// when we input new patterns, the network will return the
most result similar with the one for which it was trained

hopfield.feed([0,1,0,1,0,1,0,1,1,1])
//  the network will return [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
hopfield.feed([1,1,1,1,1,0,0,1,0,0])
// reteaua va returna  [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
```

Using this example we created the search engine architecture by extending the Network class of Synaptic library.

We created an 80-bit Hopfield neural network. The `doTrain` function is set at 1,000 iterations and an error ratio of 0.0005. If a new username is entered into the database then the network will be automatically trained for. After converting the result back into ASCII from the binary string, the network response will be displayed in *ai.ejs* where the user will be able to view the search result (Fig. 5) and that person's data in order to contact them (Fig. 6).

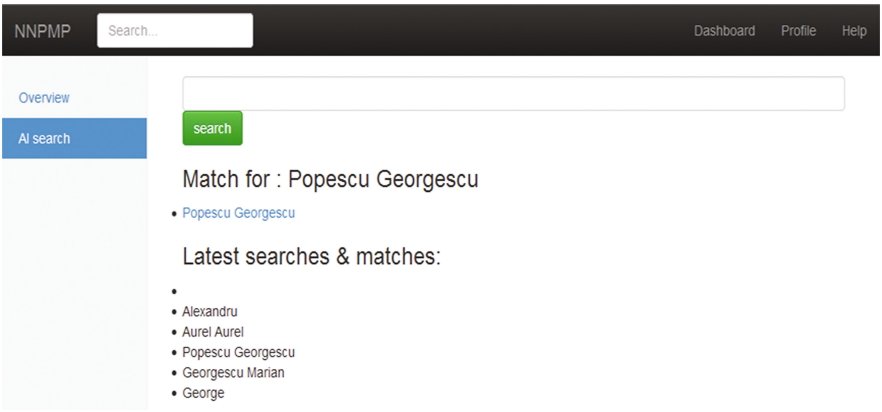

Fig. 5. Example for searching

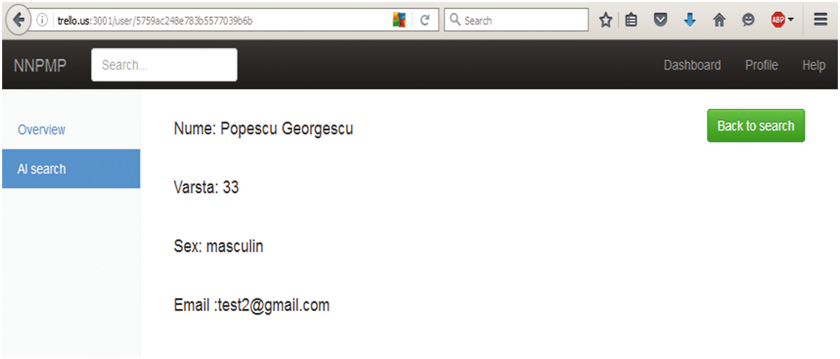

Fig. 6. Searched personal information

Each user will be able to find their friends using the search form, this information being then stored into the database and submitted to the neural network training function, according to predefined criteria (sex, age, common friends). Neural networks have yielded good results in text classification. Although instruction for complex tasks is cumbersome, we attempted to use them in this application. In theory, the application will operate by obeying a certain concept. Thus, there will be a neural network trainer, which will train the neural network according to certain formulas (e.g. age or sex). Thus, a female person will be assigned a male person, or another female person, all depending on the preferences set in the profile. Likewise, calculations will be made to determine and match optimal age. Once the neural network has been trained, when searching for people it will return as result the person it deems appropriate. This may be the beginning of a new era in social networking or dating, the innovation consisting of this very matching, which will be given by an artificial intelligence trained for this purpose. Since the search will be performed using graph technology, optimizing results with artificial intelligence will lead to a much shorter calculation time. At the same time, it is interesting to note that the selection of a partner will be made "intelligently", and not according to a set of `if-else` testing criteria.

5 Conclusions

This application illustrates the implementation of an artificial life form in a virtual environment with technologies known by most programmers, with unlimited application in the field of artificial intelligence, which may lead to faster searches, indexing and perhaps even to an artificial intelligence that could learn to communicate. By learning one programming language, namely JavaScript, with the help of Node.js and using the Synaptic library, an easy opportunity is opened to developing this kind of applications, offering JavaScript developers the possibility to do whatever they want with the same programming language. The purpose of the software is to serve an introduction to understanding neural networks, avoiding the fairly cumbersome mathematical part, with immediate application to the implementation of intelligent social networks.

References

1. HTML5 - Web developer guide, MDN (2015). https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5
2. CSS3-CSS, MDN. https://developer.mozilla.org/en/docs/Web/CSS/CSS3
3. Azat, M.: JavaScript and Node Fundamentals, Washington, DC (2014)
4. Kaufman, C., Perlman, R., Speciner, M. (eds.): Mastering Node.js. Packt Publishing, Livery Place (2013)
5. Node.js v5.9.0, Documentation. https://nodejs.org/docs/latest/api/
6. Dzitac, I. (ed.): Inteligentă artificiala. Universitatii Aurel Vlaicu, Arad (2008)
7. Popescu, M.C., Balas, V., Olaru, O., Mastorakis, N.: The backpropagation algorithm functions for the multilayer perceptron. In: Proceedings of the 11th WSEAS International Conference on Sustainability in Science Engineering, Timisoara, pp. 28–31 (2009)
8. Popescu M.C., Olaru O.: Conducerea optimala a proceselor - Proiectare asistata de calculator in Matlab si Simulink. Academiei Tehnice Militare, Bucureşti (2009)
9. Synaptic.js. https://github.com/cazala/synaptic
10. Patent Google. http://www.google.com/patents/US20050004905
11. Easy Note Authentication. https://scotch.io/tutorials/easy-node-authentication-setup-and-local
12. A Basic Introduction to Mongo DB. https://mongodb.github.io/node-mongodb-native/api-articles/nodekoarticle1.html
13. The Mongo Shell. https://docs.mongodb.org/manual/mongo/#working-with-the-mongo-s
14. An overview of Bootstrap, How to Download and Use, Basic Templates and Examples, and more. http://getbootstrap.com/getting-started/

Springer

http://www.springer.com/978-3-319-62523-2

Soft Computing Applications
Proceedings of the 7th International Workshop Soft Computing Applications (SOFA 2016), Volume 2
Balas, V.E.; Jain, L.C.; Balas, M.M. (Eds.)

2018, XXXI, 614 p. 321 illus., Softcover
ISBN: 978-3-319-62523-2