Solving Sudoku Puzzles by using Hopfield Neural Networks

V. MLADENOV[1], P. KARAMPELAS[2], C. PAVLATOS[2], E. ZIRINTSIS[2]

[1]Department of Theoretical Electrical Engineering, Technical University of Sofia
Sofia 1000, "Kliment Ohridski" blvd. 8; BULGARIA
valerim@tu-sofia.bg

[2]IT Faculty, Hellenic American University
12 Kaplanon Str., 106 80 Athens, GREECE
pkarampelas@hau.gr, ezirintsis@hau.gr, cpavlatos@hau.gr

Abstract: - In this paper two different approaches to solve Sudoku puzzles with neural networks are presented. The first approach is proposed by J.J. Hopfield. He tries to solve the Sudoku puzzle with help of a Hopfield network and treated the problem as an integer optimization problem that is also used for the solution of the well known Traveling Salesmen Problem (TSP). Second solution uses the Hopfield network with an extension, called co-processor. Since neural networks can exactly solve linear programming problems, such a network can be used as co-processor to improve the performance of the Hopfield network. Combination of both networks, where the Hopfield network was used first, was able to solve a lot of puzzles.
Key-Words: - Sudoku puzzle, Sudoku problem, Neural Networks, Hopfield Neural Network, Traveling Salesmen Problem (TSP), Linear programming problems

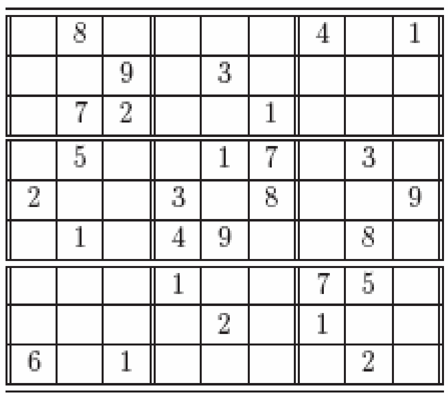
1 Introduction

In 1979 the first Sudoku puzzle was introduced by the architect Howard Garns under the name Number Place [1]. After the introduction in 1986 with the name Sudoku, the puzzle became popular in Japan. In 2005 the puzzle became an international hit.
Several simple techniques have been proposed to solve Sudoku puzzles. Some techniques show that no other numbers can be placed at a certain location, while other techniques show that a number can only be placed at a specific location. If simple techniques do not work, it is possible to make an intelligent guess, for example if a location can only have two numbers and the first one is chosen and turns out to be the wrong one, then you know that the other one has to be used. The overview of these techniques can be found in [2]. In this contribution we present a neural network approach that is based on consecutive utilization of two neural networks.
The paper is organized as follows. In the next chapter the Sudoku problem is discussed. A brief explanation how to solve Sudoku puzzles with neural networks is given in chapter 3. The combination of two neural networks that work together for solving the problem is presented there as well. Then in chapter 4, an example for application of the proposed approach is given and final conclusions are summarized in chapter 5.

2 Problem Statement

A Sudoku puzzle consists of a 9x9 board with integers in the range 1-9 placed on just a few of the empty locations. The goal of the puzzle is to fill the board with integer values in the range 1-9, with the constraints that each integer may appear only once in every: row or column or non overlapping 3x3 sub-regions. To simplify notations we will continue with the name sub-region for the non-overlapping square sub-regions of size 3. The initial puzzle has to be constructed in such a way that the puzzle has only one solution. An example of a Sudoku puzzle is given in Fig. 1.
From a mathematical point of view Sudoku puzzles can be compared to the Traveling Salesmen Problem (TSP). The problem of the traveling salesman is to find the shortest route which connects all given cities with

the smallest possible route length while every city is only passed once. TSP can be extended by not punishing the length of the route, but some other cost function over a certain route. TSP belongs to the class of integer optimization problems. These problems can be solved with Hopfield networks.



	8					4		1
		9		3				
	7	2			1			
	5			1	7		3	
2			3		8			9
	1		4	9			8	
			1			7	5	
				2		1		
6		1					2	

Figure 1: Sudoku puzzle - example

To create a Hopfield network to solve the TSP, an energy function has to be created which has a global minimum for the best solution. For TSP this could be the sum of the total costs. Also constraints can be added to the energy function in such a way that the energy increases if a constraint is violated. For the TSP this results in a constraint for every city which makes sure that every city is passed exactly once. A Hopfield network for solving Sudoku puzzles can also be obtained in similar way.

3 Solution of Sudoku problems by using Neural Networks

In this chapter two different approaches to solve Sudoku puzzles with neural networks are presented. The first approach is proposed by J.J. Hopfield in [3]. He tried to solve the Sudoku puzzle with help of a Hopfield network and treated the problem as an integer optimization problem that is also used for the TSP. Second solution uses the Hopfield network with an extension, which is also described in [3]. Since neural networks can exactly solve linear programming problems [4], a neural network can be used as co-processor to improve performance of the Hopfield network. With help of the coprocessor a guess can be made about the probability which number should be placed at a certain location. Since this guess depends on initial conditions of the neurons and biases, several initial guesses can be made for the biases to obtain other optimal solutions.

3.1 Sudoku puzzle representation for neural networks

Since neural networks are constructed in such a way that only binary or bipolar outputs are accepted, Sudoku puzzles have to be transformed to another representation. Chosen representation is given in the following subsection. In next subsection the constraints of the Sudoku puzzle are transformed to the neural representation. For every number k which can be places at a location in the puzzle (i, j) a neuron exists. Now neurons are denoted by $V(i, j, k)$, where i the row number, j the column number and k the number to be placed. From the size of the puzzle it follows that i, j and k can take values in the range from 1 to 9. The output of the neurons can be interpreted as: if the neuron fires, the output of $V(i, j, k)$ is 1, then the number k should be placed at location (i, j). In order to have a good representation of the neurons in Matlab, the neurons have to be arranged in a vector form. This is chosen in the following way:

$$V_m(l)=V(i,j,k) \tag{1}$$

where $l=9^2(l-1)+9(j-1)+k$.

The total number of neurons is: $max(l)= l=9^2(9-1)+9(9-1)+9=729$

In Sudoku puzzles exactly one number has to be placed at every position, having constraints that every number (1-9) can only be placed once in every: row; column; non-overlapping square sub-region of size 3 × 3. To simplify notations we will continue with the name sub-region for the non-overlapping square sub-regions of size 3. The constraints in neuron form are given by the following equations:

$$\sum_{i=1}^{9} V(i,j,k) = 1, \quad \forall j,k \tag{2}$$

$$\sum_{j=1}^{9} V(i,j,k) = 1, \quad \forall i,k \tag{3}$$

$$\sum_{k=1}^{9} V(i,j,k) = 1, \quad \forall i,j \tag{4}$$

$$\sum_{i\in i', j\in j'}^{9} V(i,j,k) = 1, \quad \forall k \tag{5}$$

Equations (2) and (3) make sure that every number k is only placed once in every column j and row i. Equation (4) has to ensure that only one number k is placed at a location (*i, j*). Constraint 3 is taken care of by equation (5) which sums over sub-regions, which are defined above. Every number is placed only once in every sub-region. By combining all equations, all constraints of the Sudoku puzzle are given in a neural form. The total number of constraints is 9x9x4 = 324.

3.2 Hopfield network

A continuous Hopfield network is chosen with binary outputs. The continuous Hopfield network is a single layer network which contains integrators, binary sigmoid functions and multipliers. In this section the network is initialized. First the size and architecture of the network is given and finally weights are calculated.

A Hopfield network contains simple neuron models which have a binary or bipolar output value and the input to output relation of the activation function is a logsig function, which is described by:

$$V_i = g_i(x) = \frac{1}{1+\exp(-u_i)} \tag{6}$$

Designed Hopfield network contains 729 neurons as described above. These neurons are connected in such a way that constraints are not violated. The network also contains an input *Θ*, which has a certain value for the neurons which were already supposed on by the given puzzle. Last part of the network is biases. These can also be added to the neurons if it is desirable. A simple Hopfield network is shown in Fig. 2.

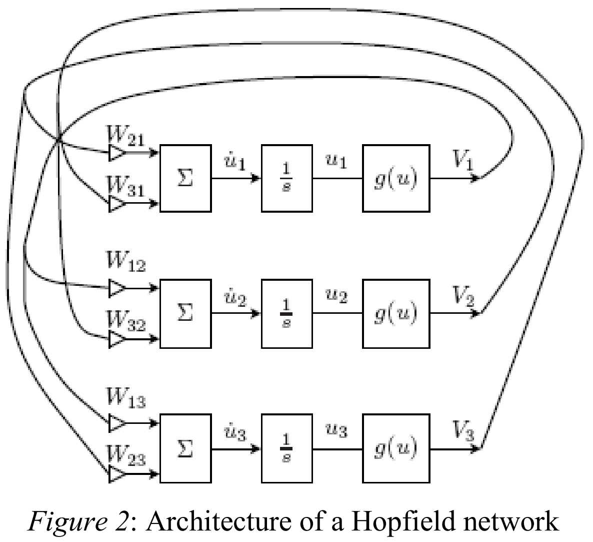


Figure 2: Architecture of a Hopfield network

This Hopfield network consists of 3 neuron models which are connected to the other neurons and not connected to themselves. The differential equation of given neuron model can be described by:

$$\dot{u}_i = \frac{du_i}{dt} = \sum_{j=1}^{n} W_{ij} V_j + \Theta_i \, , \; i = 1,2,3 \tag{7}$$

with **W** the interconnection matrix and ***Θ*** the input vector. A fixed Hopfield network with no self-connections, $W_{ii} = 0$, and symmetric weights, $W_{ij} = W_{ji}$, has the Lyapunov function:

$$E = -\frac{1}{2} \sum_{i,j} W_{ij} V_i V_j - \sum_i \Theta_i V_i \tag{8}$$

With help of the gradient an expression for *du/dt* can be given as function of the Lyapunov function *E=E(t)*:

$$\frac{du_i}{dt} = -\frac{\partial E}{\partial V_i} \tag{9}$$

The Lyapunov function can be used as a cost function which has to be minimized. By rewriting the Sudoku problem in neural form to a Lyapunov function, the initial weights of the Hopfield network are obtained. The chosen Lyapunov function in equation (10) is used to maximize the number of neurons which fire, given the constraints of the Sudoku puzzle.

$$E = -\sum_{i,j,k} V(i,j,k) + \alpha \sum_{i,j,k,i',j',k'} V(i,j,k).V(i',j',k') \tag{10}$$

By combining (9) and (10), the following equation for weighting filters can be obtained

$$\dot{u}_i = I + \alpha . V_i(i',j',k') \tag{11}$$

Now the weighting filters are determined for the constraints, only the first part of formula (10) has to be implemented in a weighting filter. As can be seen from (11) the weighting consists of only bias terms. So a weighting filter of size [729×1] has to be created with all constant positive bias terms.
When the neural network is implemented an optimization for best performance has to be made for values of *α*, bias weighting and input weighting with respect to the *logsig* function. If the performance of this network is not sufficient, then a coprocessor can be used to improve performance.

3.3 Coprocessor

Proposed coprocessor is used to give a probability that a certain number can be placed at a certain location. With help of this information the Hopfield network can be initialized to find the right solution. If the Hopfield network still fails, it is possible to use the coprocessor again, to find new initial inputs for the Hopfield network. The coprocessor tries to solve Sudoku puzzles as linear programming problem. To explain the concept of linear programming with neural networks a simple example from [5] will be used. Then the transformation of the Sudoku puzzle problem to a linear programming problem is given and finally the neural network serving as coprocessor will be presented. The simple example on linear programming with inequality constraints is given by equation

$$\begin{aligned} &\max f(\mathbf{x}) = 3x_1 + x_2 + 2x_3 \qquad s.t. \\ &r_1(\mathbf{x}) = 3x_1 - 2x_2 + 4x_3 - 8 \geq 0;\; r_2(\mathbf{x}) = -x_1 - 2x_2 - x_3 + 9 \geq 0;\; r_3(\mathbf{x}) = -2x_1 - x_3 + 6 \geq 0 \\ &x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{aligned} \tag{12}$$

The neural network used to find the answer to this problem is a recurrent network with two layers. A first layer neuron consists of a bias, *N* multipliers and a *poslin*-function, for which holds:

$$S_m = poslin(r_m) = \begin{cases} r_m & for\ r_m > 0 \\ 0 & for\ r_m \leq 0 \end{cases} \tag{13}$$

A second layer neuron consists of a bias, *M* multipliers, an integrator and also a *poslin*-function. The differential equation for the integrator is given by:

$$\frac{dx_j}{dt} = \mu_j \left(b_j - \sum_{m=1}^{M} S_m a_{mj} \right) \tag{14}$$

where μ a growing rate, b_j the bias, S_m the output of a first layer *poslin*-function and a_{mj} the weighting for the *j*'th variable and the *m*'th constraint. For the given example the differential equations become:

$$\frac{dx_1}{dt} = \mu\left(3 - k\left(3S_1 - S_2 - 2S_3\right)\right);\ \frac{dx_2}{dt} = \mu\left(1 - k\left(-2S_1 - 2S_2 - S_3\right)\right);\ \frac{dx_3}{dt} = \mu\left(2 - k\left(4S_1 - S_2\right)\right)$$

with κ the amount of penalty for constraint violations. An alternative with the same architecture is obtained by replacing the *poslin*-function with a threshold-function. The output of this function can be written as:

$$S_m = threshold(x_m) = \begin{cases} 0 & for\ x_m \leq 0 \\ 1 & for\ x_m > 0 \end{cases} \tag{15}$$

Advantage of this approach is that the value of κ is not needed to be very large. A drawback of this method is that the output in simulation becomes very "nervous" if the value for μ is too large.
Previous subsection showed how a linear programming problem can be solved using a neural network. This neural network can also be developed for Sudoku puzzles if in the constraints, given by equations (2) to (5), the equal sign '=' is replaced by a '≤'-sign and an optimization function like max $f(x) = \Sigma x(l)$ is chosen , where $x(l)$ is the probability that neuron $V(l)$ has to fire.
With this approach of solving Sudoku puzzles, the network consists of 729 '*V*' neurons which represent the probability of placing a certain integer at a certain position and 324 constraint neurons which give a positive value if representing constraint is violated. Every constraint neuron sums over 9 '*V*' neurons, in such a way that every constraint neuron represents one constraint. For the constraint neurons a weighting matrix ***R*** can be constructed which contains all a_{mj} elements from the differential equation as shown in (14). This weighting matrix can be used to check whether a constraint is violated. Then the differential equation for the coprocessor becomes

$$\frac{dx_j}{dt} = \mu\left(b - k.R(j,:).S\right)$$

Above suggested neural network can be used as coprocessor for the Hopfield network. Several choices have to be made to let the Hopfield network and the coprocessor work together.

4 Example

We consider a "difficult" Sudoku puzzle given in Fig. 3. The empty locations are filled with zeros. Both neural networks are realized in Matlab. Since the Hopfield network was not able to solve the difficult Sudoku puzzles, the coprocessor should be used for these puzzles. The coprocessor can be used in different ways to solve the difficult puzzles. It can be used before the Hopfield network tries to solve the puzzle, it can be utilized after the

Hopfield network has tried to solve the puzzle and then the puzzle which is processed by the coprocessor could be used again as input for the Hopfield network, or combination of these methods can be utilized. When the puzzle from Fig. 3 is first given as input to the Hopfield network, next the co-processor and finally the Hopfield network again, then the network solves the puzzle correctly as shown in Fig. 4. The following parameters are used: *mu*=10, *kappa*=10, *simulation time* = $5.10^{-1}s$, interconnection *weight* = 10, *input weight* = 10, *bias weight* = 5.

Also puzzles from newspapers and other sources are used in simulation and the best way to attack a Sudoku problem turns out to be alternating the application of both approaches until the solution is reached.

0	0	0	9	5	7	0	0	0
7	6	0	0	0	0	0	1	0
8	0	5	0	0	6	0	2	0
3	0	9	0	4	0	0	0	0
0	0	8	0	0	0	1	0	0
0	0	0	0	2	0	5	0	6
0	8	0	1	0	0	6	0	4
0	3	0	0	0	0	0	7	1
0	0	0	4	3	2	0	0	0

Figure 3: Puzzle from a two star Sudoku book

2	4	1	9	5	7	3	6	8
7	6	3	2	8	4	9	1	5
8	9	5	3	1	6	4	2	7
3	5	9	6	4	1	7	8	2
6	2	8	7	9	5	1	4	3
4	1	7	8	2	3	5	9	6
5	8	3	1	7	9	6	3	4
9	3	4	5	6	8	2	7	1
1	7	6	4	3	2	8	5	9

Figure 4: Correct solution for the puzzle in Fig. 3

5 Conclusion

In this paper a combination of two different neural network approaches to solve Sudoku puzzles are presented. The application of both approaches requires a transformation of the Sudoku problem to a neural network optimization problem and a corresponding Hopfield network is designed. This technique was inspired by the well known TSP. The Hopfield network itself was working pretty well, certainly no constraints were violated, but not always the right number was placed at the right location, which lead to zeros in the solution. This can be seen as a local minimum of the energy function that corresponds to the Hopfield network. For such a case another neural network was proposed, which was able to solve linear programming problems. The problem of Sudoku was subsequently transformed to an integer programming problem and the network was combined with the Hopfield network.

Emphasis in chapter 4 was on how to combine the networks to get satisfying results. Experiences with different puzzles lead to a recipe to solve Sudoku puzzles, but it was still not possible to solve all Sudoku puzzles. Sometimes both networks converge to the same incorrect puzzle again. A solution for this problem could be to use random initializations for the coprocessor or the Hopfield network, but this is a very tricky solution because it is impossible to know when the right initializations were used. Combination of both networks, where the Hopfield network was used first, was able to solve a lot of puzzles.

References:

1. http://www.maa.org/editorial/mathgames/mathgames_09_05_05.html.
2. http://www.sudokuoftheday.com/pages/techniques-overview.php.
3. J. J. Hopfield, Searching for memories, Sudoku, implicit check-bits, and the iterative use of not-always-correct rapid neural computation, posted at arXiv.org, q-bio.NC/0609006, September 5, 2006 *http://arxiv.org/ftp/q-bio/papers/0609/0609006.pdf.*
4. D. W. Tank, J.J. Hopfield, Simple optimization networks: an a/d converter and a linear programming circuit, *IEEE Trans. Circuits and Systems*, (CAS-33), 1986, pp. 533–541.
5. A. Cochocki and Rolf Unbehauen, *Neural Networks for Optimization and Signal Processing*, John Wiley & Sons, Inc., New York, NY, USA, 1993.