Reinforcement Learning on Neural Networks to Play Games from Experience

Gulnara Bikesheva and Arkady Borisov
Decision Support Systems Group
Technical University of Riga
1 Kalkyu St., Riga LV-1658, Latvia
Phone: +371 7089 530, Fax:+371 782 00 94
E-mail:aborisov@egle.cs.rtu.lv

ABSTRACT: The paper deals with constructing a neural network model, which is able to learn during a game depending on the win or the loss. For network learning, the Q-learning technique is applied, which allows to obtain a training set for the case when the network plays against itself. On playing the game and obtaining a training set, the network learns and the next game plays. After a sufficient number of games has been played, the trained network can be exploited for playing against a human opponent.

KEYWORDS: neural network, error back propagation algorithm, reinforcement learning, Q-learning, game of Tic-Tac-Toe

1. INTRODUCTION

The problem under consideration can be stated as follows. For the game of Tic-Tac-Toe it is necessary to construct a player that would be able to find the shortcomings in the opponent's play and to learn thus maximising his chances to win. To solve this problem, one may apply the neural net technology the main advantages of which are the presence of efficient learning algorithms and easy implementation. Error back propagation algorithm, or gradient descent method, is one of the most known learning algorithms. It is a method of supervised learning. To make application of this method possible, it is necessary to have a training set in the form of input-output pairs.

A network should be able to learn during the play when the supervisor, who could point out which move to make, is absent. Therefore, it must learn by trial and error through playing a plenty of games in order to find the winning strategy. If the game was won, the moves that led to the win should be rewarded properly and, vice versa, if the game was lost, all the moves should be "punished". Such a reward-and-punishment tactics is the essence of the approach called *reinforcement learning* (Mitchel, 1997) which has been developing rapidly during the last years. Reinforcement learning implies a wide spectrum of methods based on learning through the interaction with the environment. Among them are the method of Temporal Differences TD (λ) (Sutton, 1988) and Q-learning (Sutton and Barto, 1988) that are widely exploited.

This paper suggests to apply Q-learning for obtaining a training set for artificial neural network learning by the method of gradient descent.

2. THE NEURAL NETWORK ARCHITECTURE FOR THE GAME OF TIC-TAC-TOE

The network chosen for training how to play the game of Tic-Tac-Toe is a forward propagation neural network containing three layers: (1) input layer, (2) hidden layer and (3) output layer. The input layer consists of 18 elements. The first 9 elements encode the current board situation. All squares of the board are numbered with numbers between 1 and 9 (see Fig. 1a). At each step of the game the board position is encoded as follows:

1 - the square contains a X;
-1 - the square contains a 0;
0 - the square is empty.

For example, the board situation shown in Fig.1b is represented in Fig.1c:

1	2	3
4	5	6
7	8	9

(a) A standard Tic-Tac-Toe board

X	**O**	**O**
O	**X**	**X**
		X

(b) An example of game drawing

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

(c) Encoding of square contents

Figure 1: Encoding of the first nine network inputs

The rest 9 elements of the input vector determine valid moves that might be made from the given state. They are encoded as follows:

1 – the square is empty and a move can be made;

0 - the square already contains symbol, 0 or X.

Thus, encoding of states and moves for the same example, but one move earlier will look as shown in Fig. 2.

X	**O**	**O**
O	**X**	**X**

(a) Board position

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

(b) Encoding of states

10 **0**	11 **0**	12 **0**
13 **0**	14 **0**	15 **0**
16 **1**	17 **1**	18 **1**

(c) Encoding of possible moves

Figure 2: An example of network inputs encoding

The hidden layer consists of N nodes. The number of nodes is a system parameter and can be found experimentally.

The output layer contains 9 nodes. Each node determines the value of Q-function for each move possible for the current board situation. This means that during network learning, the evaluation function of a "state-action" pair is approximated. The outlined neural network architecture for learning to play the game of Tic-Tac-Toe is shown in Fig.3 where S $\{s_1, s_2, \ldots, s_9\}$ is a vector of states, A $\{a_1, a_2, \ldots, a_9\}$ is a vector of possible actions (moves), H $\{h_1, h_2, \ldots, h_n\}$ denotes the hidden layer nodes and O $\{o_1, o_2, \ldots, o_9\}$ is an output vector.

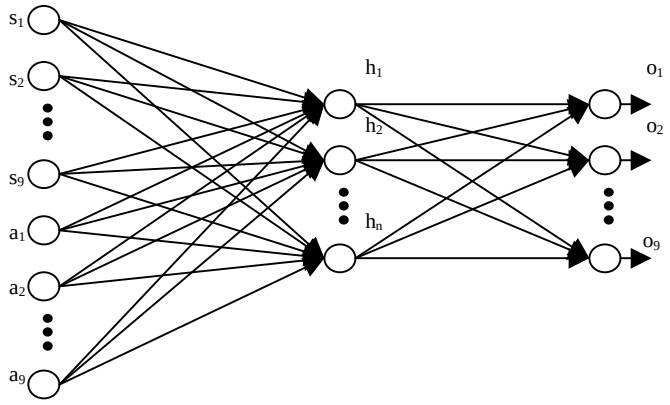


Figure 3: The neural net architecture for the game of Tic-Tac-Toe

3. GAME PLAYING

Proceeding from the assumption that a network should learn itself by repeatedly playing various games against itself, consider first in more detail network functioning in the Tic-Tac-Toe environment. As stated above, the encoded board position at the certain stage of the game and the valid moves are passed to the network input in the encoded form. At the output of the network, each move possible from the given state is evaluated with regard to how much it contributes to the win in this game. This means that each of 9 network outputs evaluates the corresponding state-action pair. A move, to which the maximum network output corresponds, is chosen as a next move. Hence, neural network learning aims at obtaining such a set of weights under which the most profitable moves, that is, the moves leading to the win, would have the largest value at the corresponding network outputs. Provided the values of network output are determined as the value of Q-function evaluation, the task under consideration may be stated as a task of this function approximation.

Before learning, a network should be initialized, that is, the number of neurons in the hidden layer and initial weight values have to be found. The initial weights values are set by random small values, whereas the optimum number of neurons in the hidden layer can only be found experimentally by varying the number of them.

The second important issue is how to exploit one and the same network so that it could play against itself. For this it was decided to consider encoding of states for 1 and -1 broader than 1 for a X and -1 for a 0. Since there is no difference for the network who places which mark, 1 will be treated as encoding for the player who is choosing a move at the moment, but for his opponent the encoding will be -1. Hence, at each moment of the play the network will be choosing a move that is more profitable for the player irrespective of whether he is playing Xs or 0s.

After the number of neurons in the hidden layer has been chosen and the initial weight values have been found, it is necessary to play the first game to obtain a training set. Initially, all the squares on the board are empty, and the first move can be made to any of them. Assume that Xs always play first. The following values will then be passed to the network input:
Placing a X

State (1-9)									Action (1 – 9)								
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

At the beginning, when the network is not trained yet, the values on the network outputs will be approximately equal. That is why the next move is being chosen out of all possible ones by random. The sigmoid function is used as an activation function for the nodes of the hidden layer and the output layer:

$$F(Y) = \frac{1}{1 + e^{-l \cdot Y}}$$

However, in order to fully exclude the possibility to choose a move to a square that is already occupied, the activation function for the output layer is modified.

$O_i = F(Sw_{ij}h_i) \times a_k$, where

O_i is the network output;

h_I is the value of the i-th node in the hidden layer;

a_k is the value of the k-th element of the possible moves vector.

Move 1. Placing a X. Suppose a move has been chosen as shown in Fig. 4a.
After this move has been made, the board position has been changed. Now it can be represented as follows:
<0 0 0 0 0 0 0 0 0> → <0 0 0 0 1 0 0 0 0>

Move 1. Placing a 0. The move is passed to the opponent. In order to choose a move for a 0, inversion of the state should be made, i.e. every 1 is replaced by -1 but every -1 is replaced by 1. Thus the representation of the board in this case will look as <0 0 0 0 1 0 0 0 0> → <0 0 0 0 -1 0 0 0 0> . From this state one can only choose a single move out of eight possible ones. Hence, there will be the following combination on the network input:
Placing a 0.

State (1-9)									Action (1 – 9)								
0	0	0	0	-1	0	0	0	0	1	1	1	1	0	1	1	1	1

Assume the next move has been chosen (see Fig. 4b). The board position after this move can now be represented as <0 0 0 0 -1 0 0 0 0> → <0 0 0 0 -1 0 0 0 1>. Again the move is passed to the player playng Xs. Now, to choose a move for X, inversion of the state is required: <0 0 0 0 -1 0 0 0 1> → <0 0 0 0 1 0 0 0 -1> . With this, a move can only be chosen out of 7 possible actions.

The game is then played until all the squares are occupied or until one of the players wins. Suppose the Xs won at the fifth move and the game proceeded as follows.

Move 2. Placing a X (see Fig. 4c).
Network inputs before placing a X were the following:

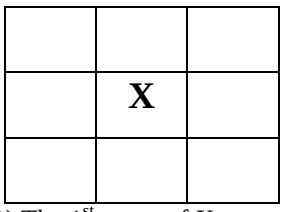
State (1-9)									Action (1 – 9)								
0	0	0	0	1	0	0	0	-1	1	1	1	1	0	1	1	1	0

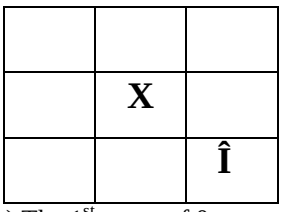
Move 2. Placing a 0 (see Fig. 4d).
Network inputs before placing a 0 might be represented as follows:

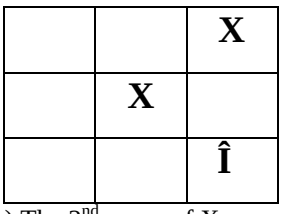
State (1-9)									Action (1 – 9)								
0	0	-1	0	-1	0	0	0	1	1	1	0	1	0	1	1	1	0

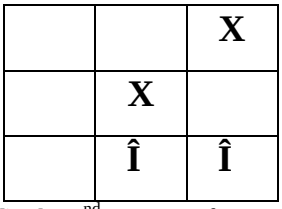
Move 3. Placing a X (see Fig. 4e).
Network inputs before placing a X are

State (1-9)									Action (1 – 9)								
0	0	1	0	1	0	0	-1	-1	1	1	0	1	0	1	1	0	0


(à) The 1st move of Xs.


(b) The 1st move of 0s.


(c) The 2nd move of Xs.


(d) The 2nd move 2 of 0s.

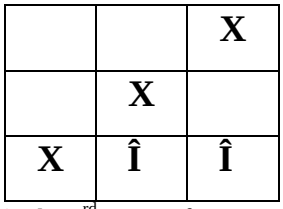

(e) The 3rd move of Xs.

Figure 4: Proceeding of the game

Until the network is not trained all network outputs corresponding to possible moves will be supposed to have equal value of 0.5 but values of all the other network outputs equal to 0. Therefore, one may compose a table of network inputs and outputs for those players who play X's and 0's (see Table 1).

Table 1: Values of network inputs and outputs obtained while playing a game

	Network inputs (<state> <action>)	Network outputs	Move
X_1	<0 0 0 0 0 0 0 0 0> <1 1 1 1 1 1 1 1 1 >	<0.5 0.5 0.5 0.5 **0.5** 0.5 0.5 0.5 0.5>	5
X_2	<0 0 0 0 1 0 0 0 -1> <1 1 1 1 0 1 1 1 0 >	<0.5 0.5 **0.5** 0.5 0 0.5 0.5 0.5 0 >	3
X_3	<0 0 1 0 1 0 0 -1 -1> <1 1 0 1 0 1 1 0 0 >	<0.5 0.5 0 0.5 0 0.5 **0.5** 0 0 >	7
O_1	<0 0 0 0 -1 0 0 0 0> <1 1 1 1 0 1 1 1 1 >	<0.5 0.5 0.5 0.5 0 0.5 0.5 0.5 **0.5** >	9
O_2	<0 0 -1 0 -1 0 0 0 1> <1 1 0 1 0 1 1 1 0 >	<0.5 0.5 0 0.5 0 0.5 0.5 **0.5** 0 >	8

4. GENERATING A TRAINING SET

To train a network through the error back propagation algorithm, it is necessary to know the desired network outputs. For this one may exploit the Q-learning algorithm. Before using it, the value of discount factor γ has to be found.
Assume $\gamma = 0.5$. According to formula $Q(s,a) = r(s,a) + \gamma \max_{a'} Q(d(s,a),a')$, the value of $Q(s,a)$ depends on the value of direct reward $r(s,a)$ which is only non-zero for the last made move. Thus $r(s,a)$ is equal to 1 in the case of win and it equals -1 under the loss. In the instance under consideration where the Xs have won, the reward for the last move is 1. Since the game stops after the last move, the value of Q-function under the last move of Xs will be 1. This can be written as follows.

Step 1.

Q (<0 0 1 0 1 0 0 -1 -1>, 7) = **1**

where <0 0 1 0 1 0 0 -1 -1> is the state but 7 is the chosen move, i.e. the number of the board square were the last mark was placed.
Now one may obtain the desired network output for the input vector X_3 <0 0 1 0 1 0 0 -1 -1> <1 1 0 1 0 1 1 0 0 >. For this, the network output vector available, <0.5 0.5 0 0.5 0 0.5 **0.5** 0 0 > will be transformed to <0.5 0.5 0 0.5 0 0.5 **1** 0 0 >.

Step 2.

Let us now consider the last but one move made by Xs from state X_2. Since the move is not the last, the direct reward is 0.
$d(s,a)$ denotes the state resulting from action a. The situation obtained after one' s move and after the opponent' s move will be assumed as such a state but not the state obtained just after one's move.In this case it will be state X_3. The vector determined at the preceding step represents all values of Q-function for this state:
<0.5 0.5 0 0.5 0 0.5 **1** 0 0 >. The maximum of the function will be

$\max_{a'} Q(d(s,a),a') = \max$ (0.5 0.5 0 0.5 0 0.5 **1** 0 0) = **1**

Now one can come to Q (<0 0 0 0 1 0 0 0 -1>, 3) = 0 + 0,5 *1 = 0,5.
Hence, it is then possible to find the desired network output for the input vector
X_2 <0 0 0 0 1 0 0 0 -1> <1 1 1 1 0 1 1 1 0 >. For this, the available network output vector
<0.5 0.5 **0.5** 0.5 0 0.5 0.5 0.5 0 > will be transformed to <0.5 0.5 **0.5** 0.5 0 0.5 0.5 0.5 0 >. As can be seen, it has not changed.

Step 3.

Consider the first move by Xs made from state X_1. Similar to the preceding step, the maximum will be determined as

$\max_{a'} Q(d(s,a),a') = \max$ (<0.5 0.5 **0.5** 0.5 0 0.5 0.5 0.5 0 >) = **0.5**

Q (<0 0 0 0 0 0 0 0 0> , 5) = 0 + 0,5 *0,5 = 0,25.
The desired network output for the input vector X_1 <0 0 0 0 0 0 0 0 0> <1 1 1 1 1 1 1 1 1 > may also be found through the transformation of the available for it network output vector, <0.5 0.5 0.5 0.5 **0.5** 0.5 0.5 0.5 0.5>, to <0.5 0.5 0.5 0.5 **0.25** 0.5 0.5 0.5 0.5>.

It may seem that first moves that led to the win of Xs have in the long run decreased their value whereas the essence of reinforcement learning is to reward the moves leading to the win and penalise the moves leading to the loss. However, as a matter of fact, in the classic Q-learning the initial values are set as zeros. But when a neural network with random initial weght values and sigmoidal activation function are employed the value of Q-function for valid moves is usually close to 0.5. That is why it is assumed that those moves that have not been chosen yet will enable one to choose the decreased value of this function. Thus, the problem of choosing the next move which is topical for learning on the basis of reinforcement learning will be excluded. It will be reduced to choosing between the moves that have led to better results in the past and those that have never been exploited yet.
Consider now the part that relates to playing by 0s. Since the 0s have lost, their moves should be penalised. The network outputs may assume meanings between 0 and 1 that is why it is impossible to apply the negative reward. 0 cannot also be taken as a reward because it denotes invalid moves, i.e. non-empty squares. Therefore, a small positive number , say 0.1, might be used to determine the direct reward in the case of loss. The desired network ourputs can then be computed as it was performed for the Xs.

Let us reduce the data obtained for learning to a table (see Table 2).

Table 2: The training set obtained through Q-learning

Network inputs <state> <action>	Network outputs	Desired output
<0 0 0 0 0 0 0 0 0><1 1 1 1 1 1 1 1 1 >	<0.5 0.5 0.5 0.5 **0.5** 0.5 0.5 0.5 0.5>	<0.5 0.5 0.5 0.5 **0.25** 0.5 0.5 0.5 0.5>
<0 0 0 0 1 0 0 0 -1><1 1 1 1 0 1 1 1 0 >	<0.5 0.5 **0.5** 0.5 0 0.5 0.5 0.5 0 >	<0.5 0.5 **0.5** 0.5 0 0.5 0.5 0.5 0 >
<0 0 1 0 1 0 0 -1 -1><1 1 0 1 0 1 1 0 0 >	<0.5 0.5 0 0.5 0 0.5 **0.5** 0 0 >	<0.5 0.5 0 0.5 0 0.5 **1** 0 0 >
<0 0 0 0 -1 0 0 0 0><1 1 1 1 0 1 1 1 1 >	<0.5 0.5 0.5 0.5 0 0.5 0.5 0.5 **0.5** >	<0.5 0.5 0.5 0.5 0 0.5 0.5 0.5 **0.25** >
<0 0 -1 0 -1 0 0 0 1><1 1 0 1 0 1 1 1 0 >	<0.5 0.5 0 0.5 0 0.5 0.5 **0.5** 0 >	<0.5 0.5 0 0.5 0 0.5 0.5 **0.1** 0 >

The game of Tic-Tac-Toe is a symmetric game. This means that if a mirror image of the board is created or the board is rotated to 90 °, the game, in fact, will be the same. Hence, by playing a single game and properly recoding the obtained data, one can get a much greater training set.

As for the game of Tic-Tac-Toe, it is possible to obtain a training set for eight games by playing just one game. For this, 8 different techniques should be used to number squares on the board.

5. CONCLUSIONS

By using an extended training set for neural network training and after playing a sufficient number of games, a certain collection of network weights has been obtained under which the network may be exploited for playing against a real opponent. The well-known winning strategies are not represented in the network structure. For example, if there are two opponent's symbols in a row, a move should be made that would deprive the opponent of the win, given that it is impossible for the first player to win himself within the current move. Application of the proposed technique enables one to obtain such a network which would follow this principle. The principle will be encoded in the network as a certain collection of weights. Thus, reinforcement learning methodology might be employed for solving complicated tasks when it seems problematic to previously determine how to act. However, by learning step-by-step and accumulating experience by trial and error it is possible to come to the required result.

Further investigation of the reinforcement learning procedure assumes application of the outlined neural network for the determination of optimal learning parameters as well as development of learning acceleration methods.

REFERENCES

Mitchel, Tom M, 1997, “Machine learning”, McGraw – Hill.

Sutton, Richards S., 1988, “Learning to Predict by the Methods of Temporal Differences”, *Machine Learning* 3; pp. 9-44.

Sutton, Richard S.; Barto, Andrew G., 1988, “Reinforcement Learning: An Introduction”, Mit press, Cambridge, MA, A Bradford Book.