Study of Game Strategy Emergence by using Neural Networks

Ladislav Clementis*
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 2, 842 16 Bratislava, Slovakia
clementis@fiit.stuba.sk

Abstract
In artificial intelligence systems, various machine learning algorithms are used as learning algorithms. The most used artificial intelligence approaches are symbolic rule-based systems and subsymbolic neural networks. The main objective of this work is to study the game strategy emergence by using subsymbolic approach - neural networks. From the viewpoint of artificial intelligence, games in general are interesting. The games are often complex even if their definitions, rules and goals are simple. In this work we are concerned about the Battleship game. The Battleship game is a representative of games with incomplete information. We will design and implement solutions based on using subsymbolic artificial intelligence approach to solve the Battleship game. We will use machine learning techniques as the supervised learning and the reinforcement learning for this purpose. We will compare machine learning techniques used by using simulation results and statistical data of human player.

Categories and Subject Descriptors
F.1.1 [**Theory of Computation**]: COMPUTATION BY ABSTRACT DEVICES—*Models of Computation*; G.3 [**Mathematics of Computing**]: PROBABILITY AND STATISTICS—*Markov processes, Probabilistic algorithms (including Monte Carlo), Stochastic processes*; I.2.6 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE—*Learning*; I.2.8 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE—*Problem Solving, Control Methods, and Search*; I.5.1 [**Computing Methodologies**]: PATTERN RECOGNITION—*Models*

Keywords
game strategy, neural network, probability based heuristic, reinforcement learning, supervised learning

9*Recommended by thesis supervisor: Prof. Vladimír Kvasnička
Defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on December 30, 2014.

1. Introduction
Games in general are suitable to study many *AI* approaches. Games, especially board games are simple, well defined and playing strategies are well comparable. Therefore it is simple to evaluate game strategies developed by *AI* systems, especially by *machine learning* techniques.

Popular breakthroughs have been done in *AI* system playing board games [4, 11, 16, 21, 27, 28, 33].

In our work we study the Battleship game as a *AI* problem. We will study the Battleship game by symbolic and subsymbolic *AI* approaches. We will use multiple *machine learning* techniques. We will compare these approaches by solving the Battleship game.

We define our work objectives in section 2.

2. Objectives of the work
We generalize the main objectives of our work. The main objectives are following:

- Design and describe a probability-based heuristic to the Battleship game strategy
- Use subsymbolic artificial feed-forward network adapted by supervised learning - gradient descent method to solve the Battleship game
- Use subsymbolic artificial feed-forward network adapted by reinforcement learning to solve the Battleship game
- Compare quality of game strategy learned by used subsymbolic approach
- Compare game strategy emergence effectiveness of used and subsymbolic approach.

3. Artificial Intelligence Approaches
Artificial intelligence (*AI*) [1, 2, 11, 15, 18, 20] is a wide field of study. *AI* is usually defined as *"the study and design of intelligent agents"*. *AI* problems are mainly based on natural intelligent systems and processes.

The emergence of *AI* research field is linked to the development and progress in the computer technology that enabled advanced *AI* development.

AI goal is to study and design artificial systems concerning:

- *perception*
- *deduction*
- *reasoning*
- *problem solving*
- *knowledge representation* and *knowledge transformation*
- *learning*
- *planning*
- *natural language processing*
- *motion* and *manipulation*
- *social intelligence*
- *general intelligence - strong AI*

The AI field is interdisciplinary and it overlaps many other disciplines as mathematics, computer science, psychology, philosophy, linguistics, neuroscience, cognitive science etc ...

AI uses many adjustable tools to design advanced systems:

- search and optimization algorithms
- evolutionary computation
- logic programming
- probabilistic and statistical tools
- machine learning techniques
- classifiers
- expert systems
- neural networks
- etc ...

AI use these tools to develop advanced *AI* systems. Advanced *AI* systems are used for theoretical, research and practical purposes.

We distinguish between *cybernetic*, high-level *symbolic*, low-level *subsymbolic* and *hybrid AI* approaches.

3.1 Subsymbolic Artificial Intelligence Approach

Connectionist *subsymbolic AI* uses parallel and/or sequentially processed information. In *subsymbolic* theory information is represented by a simple sequence of pulses. Transformations are realized by simple calculations performed by artificial *neurons* to solve *AI* problems.

Theoretical concepts of *subsymbolic AI* have their origin in neuroscience. Networks of *neurons*, called *neural networks* are inspired by an animal central nervous systems (in particular the brain) [3, 13, 15, 22, 24, 26, 33].

A single *neuron* (a nerve cell) is composed of:

- *dendrites* - input branched connections
- cell body which processes pulses
- *axon* - output connection

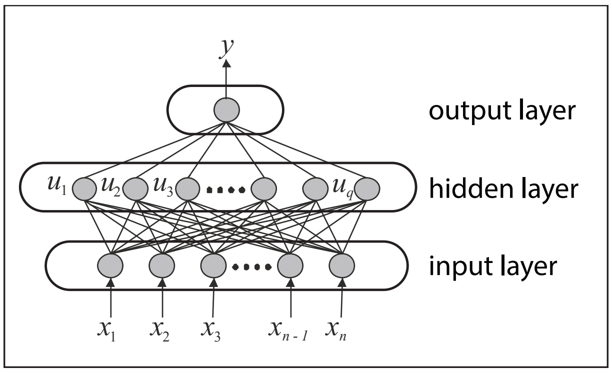

Figure 1: The simple three-layer *ANN* with the input layer *x*, the single hidden layer *u* and the output layer *y*. The output layer *y* consists of single neuron *y*.

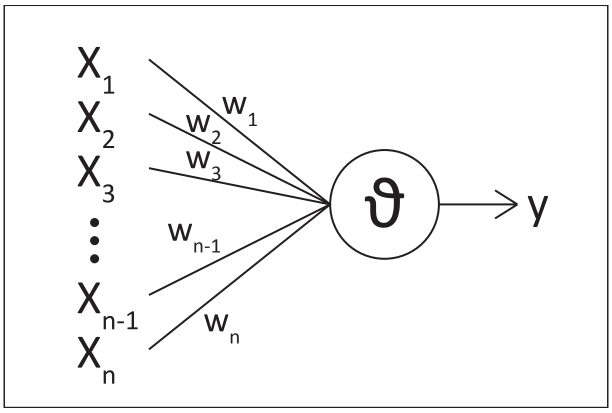

Figure 2: The simple artificial neuron with the input vector *x*, the vector of weights *w*, threshold ϑ and output activity *y*.

These biologic *neurons* and biologic *neural networks* have inspired *subsymbolic AI* approach creating computational models of *artificial neurons* and *artificial neural networks* (*ANNs*).

The simple *ANN* can be formally defined as a parametric mapping of input vector $\boldsymbol{x}$ to an output activity y, as shown by equation 1.

$$y = G(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{\vartheta}) \tag{1}$$

In the equation 1, $\boldsymbol{x}$ is the input vector, $\boldsymbol{w}$ is the vector of neural *weights* and $\boldsymbol{\vartheta}$ is the vector of neural *thresholds*.

The simple *ANN*, as shown in figure 1 consists of three types of layers:

- input layer
- hidden layer(s)
- output layer

$$y = t\left(\sum_{i=1}^{n} w_i x_i - \vartheta\right) \tag{2}$$

An artificial neuron shown in figure 2 is a mathematical function shown by equation 2 based on:

- input vector $\boldsymbol{x}$
- vector of neural *weights* $\boldsymbol{w}$
- *threshold* $\boldsymbol{\vartheta}$
- transfer function $t()$
- output (activity) y

An artificial neuron function can be represented by its linear combination shown by equation 3 which is used as an input to a transfer function $t()$.

$$y = t\left(w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_{n-1}x_{n-1} + w_nx_n - \vartheta\right) \quad (3)$$

In equations 4 and 5 we are using the weighted sum of all n neuron inputs u, where $\boldsymbol{w}$ is a vector of weights and $\boldsymbol{x}$ is an input vector.

$$u = \sum_{i=1}^{n} w_ix_i \quad (4)$$

$$y = t\left(u + \vartheta\right) \quad (5)$$

There are many types of *ANN* transfer functions, most used are:

- *sigmoid function*
- *step function*
- *linear combination*

The simple *step function* usually used by perceptron is described by equation 6.

$$y = \begin{cases} 1 & \text{if } u \geq \vartheta \\ 0 & \text{if } u < \vartheta \end{cases} \quad (6)$$

Simple *ANNs* can be designed to solve simple *AI* problems. While dealing complex *AI* problems, *ANNs* are usually adapted by *machine learning* techniques:

- *Supervised learning*
- *Unsupervised learning*
- *Reinforcement learning*
- *Deep learning*

When we are dealing with complex issues, it is difficult to define symbolic rules explicitly. We can use subsymbolic approaches to develop a knowledge base by using *machine learning* techniques.

4. The Battleship Game

The Battleship game is a guessing game for two players. It is known worldwide as a pencil and paper game which dates from World War I. It was published by various companies as a pad-and-pencil game since 1931. We should mention some known commercial origins of this game:

- The Starex Novelty Co. of NY published game as *Salvo* (1931)
- The Strathmore Co. published game as *Combat, The Battleship Game* (1933)
- The Milton Bradley Company published the pad-and-pencil game *Broadsides, The Game of Naval Strategy* (1943)
- *The Battleship game* was released as a plastic board game by the Milton Bradley Company (1967)

The game has been popularized under the name "*The Battleship game*" by the Milton Bradley Company as it was published as a plastic board game in 1967.

4.1 The Original Battleship Game

We provide description of the Battleship game as an interactive game for two players; $Player_1$ and $Player_2$. Game is iterative by performing following tasks simultaneously:

1. initial deployment of $Player_1$'s and $Player_2$'s ships
2. iterative shooting to $Player_1$'s and/or $Player_2$'s battlefield until enemy ship is hit
3. complete ship destruction in $Player_1$'s and/or $Player_2$'s battlefield

Task 1 is performed just once by both players, initially at the beginning of each game, before "action" part starts. Tasks 2 and 3 are repeated sequentially until all ships in $Player_1$'s or $Player_2$'s battlefield are destroyed (i. e. one player destroys opponent's ships).

$Player_1$ wins if he reveals all ships in $Player_2$'s battlefield completely first, i. e. before $Player_2$ does the same in $Player_1$'s battlefield. Similarly, $Player_2$ wins if he reveals all ships in $Player_1$'s battlefield completely first.

Originally (the Milton Bradley Company version), ships were placed in a battlefield of size of 10×10. This battlefield included set of linear (oblong) shaped ships, with vertical and horizontal orientations allowed. Originally, ship shapes were as follows (all with the width of 1):

- Aircraft Carrier (length of 5)
- Battleship (length of 4)
- Submarine, and Cruiser or Destroyer (length of 3)
- Destroyer or Patrol Boat (length of 2)

Many definition and/or rule variations are present in the history of this game, for example:

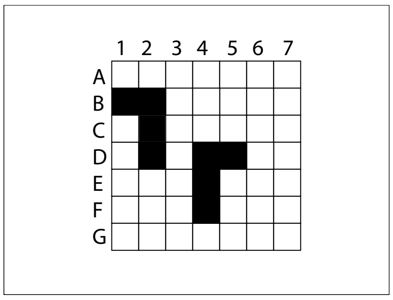

Figure 3: Example of problem instance, with pattern placements corresponding to the Battleship game rules (pattern deployment rules). This example is corresponding with our modification (environment with size of $n \times n = 7 \times 7$ and two "L"-shaped patterns with size of 2×3).

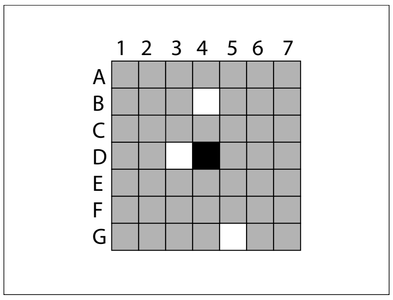

Figure 4: Figure showing the current state of environment corresponding to the problem instance shown in figure 3. Gray cells are not revealed. Black and white cells are already revealed. Each white cell does not contain a part of a pattern. Black cell contains a part of a pattern.

- after a successful hit, player gains one more hit attempt in a row
- different battlefield sizes (size of 7×7, 8×8, 8×10, 10×12, etc ...)
- many different ship shapes, etc ...

4.2 Simplified Modification of the Battleship Game as an Optimization Task

We can consider the Battleship game from the single player perspective, who is solving just single instance of "shooting" position decision problem at a time [23]. In this case, our optimization task is to minimize cumulative number of our hit attempts while revealing all opponent's ships completely.

Player, whose task is to reveal ships in enemy battlefield makes decision each time before hit attempt. Respectively, his task is to pick a position in enemy battlefield to be shot at next. This decision making can be formally described as *Markov decision process* (*MDP*), because player's decision making is independent of result of decision made in previous iteration. Since the problem space is not visible completely, this process can be described as *Partially observable Markov decision process* (*POMDP*) [8, 9, 17, 26, 32]. All information currently available is stored in a current state of environment. Therefore, no information about previous decision making is needed. Apart a current state of environment, a sequence order of previous hit attempts is irrelevant.

Player is formally modeled as an agent. Enemy battlefield (including ship placements) is taken into account as a problem instance. Current view of an agent (who's view is incomplete information represented by partially revealed problem instance) is considered as a current state of environment. Ship represents a pattern present in problem instance. Ship placement may not be completely known because of incomplete information about problem instance, represented by a current state of environment. Ship placement in an environment is formally considered as a pattern placement (rotation and position) in a problem instance.

We provide simplified modification of the Battleship game for description and simulation purposes (shown in figure 3). Our modification is based on the changed environment size, changed pattern sizes and changed pattern quantity. We can simply summarize this modification as follows:

- only single player optimization perspective
- environment size of $n \times n = 7 \times 7 = 49$ "cells" (MAX. hit attempts is 49)
- "L"-shaped patterns with size of 4 cells arranged in 2×3 shapes (with all 8 rotation permutations allowed)
- two patterns are present in environment

All other trivial rules of the Battleship game remain unchanged (including the deployment rules):

- pattern overlap (even partial) is not permitted
- patterns may not share common edge
- pattern placement configuration remains stable while solving single problem instance (to make hit attempt to same position twice is pointless)
- response from an environment to hit attempt is always truthful, undeniable (as an *oracle* which always responds "YES" or "NO" truthfully), etc ...

We provide the example of problem instance shown in figure 3. This example problem instance is in correspondence with our simplified modification of the Battleship game. We will be using this example problem instance in this work for simulations, further explanations and descriptions.

Let us define an example of environment state. In environment state we have incomplete information about problem instance. The example environment state which we will be using in this contribution is shown in figure 4.

In the current environmental state, we already performed hit attempts to cells $D3$, $B4$, $G5$ and $D4$. The first three hit attempts to cells $D3$, $B4$, $G5$ (order is irrelevant) were unsuccessful, until hit attempt to the cell $D4$ was finally successful. At this stage, we (or agent solving this problem) should perform some reasonable decision making to maximize the probability that next hit attempt will be successful. We provide description of the probabilistic heuristics in section 4.3. Note that initial hit attempts (before the first successful hit) should have at least some reasonable strategy too to increase success probability of a hit attempt.

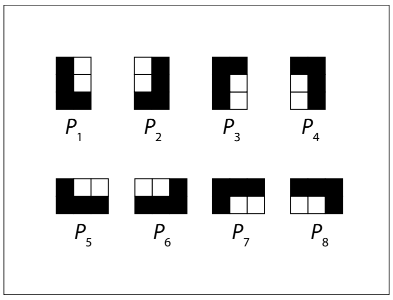

Figure 5: All 8 **possible pattern permutations,** 4 **vertical and** 4 **horizontal.**

4.3 Probability-Based Heuristic to the Simplified Battleship Game

We provide description of the probabilistic heuristic approach [5, 6, 7, 19, 29, 31] to the simplified Battleship game described in section 4.2. There was already performed the first successful hit attempt in the example environment. The current state of the environment is shown in figure 4.

Cells $D3$, $B4$, $G5$ do not contain a part of a pattern, but cell $D4$ is containing a part of a pattern. What we do not know is specific pattern rotation (rotation and flip actually) and position. Therefore, in this case we can not determine cell to be shot at next with 100% probability of successful result. But we can maximize successful hit probability by a probability-based heuristic.

According to the simplified battleship game described in section 4.2, patterns placed in an environment are "L"-shaped patterns with size of 2×3 squares, with 8 possible rotations (or permutations). All this possible permutations together form the set $\boldsymbol{P} = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8\}$. All possible pattern permutations are shown in figure 5.

In correspondence with the example environment state shown in figure 4, multiple pattern permutations do match this environment state. Furthermore, multiple pattern permutations have more than one possible position, which do match the current state of the environment. Numbers of possible pattern positions according to pattern permutations are shown in table 1.

As shown in table 1, there are totally 17 possible pattern placements (all possible pattern permutations with corresponding possible positions) that match the current state of the environment ($PosConf = 17$), so that just one pattern is considered and cell $D4$ contains it's part. Each one of 17 possible placements is covering cell $D4$ and other three cells in addition. Therefore, each cell in the environment is covered by some number of possible pattern placements. In our example, each cell has a number which is bounded by range from 0 to 17. These non-zero numbers (*Coverings*) for each cell are shown in figure 6 and in table 2.

CoveringProbability of each cell can be calculated by number of *Coverings* for the cell divided by sum of all *Coverings*. This calculation is given by equation 7.

$$CoveringProbability_i = \frac{Coverings_i}{\sum_i Coverings_i} \tag{7}$$

HitProbability of each cell is overall probability that hit attempt to this position will result into successful hit. Because in the current state of the environment we are missing three remaining parts of the current pattern ($MissingParts = 3$), *HitProbability* of each cell is three times higher than *CoveringProbability*. Calculation is given by equations 8 and 9.

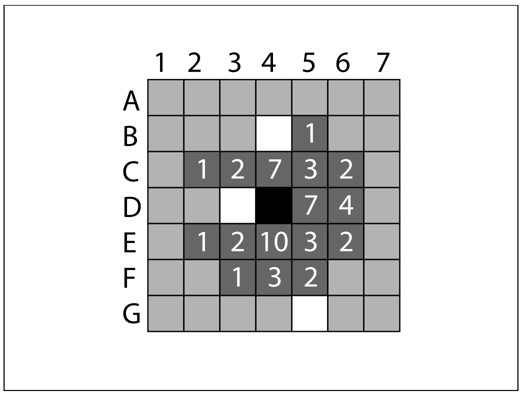

Figure 6: The current state of the environment enriched by non-zero numbers of pattern placement coverings of each cell. The white numbers placed in dark-gray cells are pattern placement coverings.

$$HitProbability_i = CoveringProbability_i \times MissingParts \tag{8}$$

$$HitProbability_i = \frac{Coverings_i}{\sum_i Coverings_i} \times MissingParts \tag{9}$$

The sum ($\sum_i Coverings_i$) is three times (*MissingParts*-times) higher than the number of possible pattern placements *PosConf* (equation 10). Therefore, *HitProbability* can be described by equation 11.

$$\frac{\sum_i Coverings_i}{PosConf} = MissingParts \tag{10}$$

$$HitProbability_i = \frac{Coverings_i}{PosConf} \tag{11}$$

For each cell i, non-zero value of $Coverings_i$, calculated $CoveringProbability_i$ and $HitProbability_i$ of all $n \times n = 7 \times 7$ cells are shown in table 2.

According to *HitProbability* values shown in table 2, the most reasonable cell for next hit attempt is cell $E4$ with *HitProbability* of 0.5882. Cells $C4$ and $D5$ are acceptable too, with *HitProbability* values of 0.4118. Shooting to these positions is reasonable according to this probability-based heuristics.

Note that if hit attempt to cell with high *HitProbability* value will result into unsuccessful hit, high number of pattern placements will be excluded from possible pattern placements in the current state of the environment, and less pattern placements will remain.

Table 1: Numbers of possible pattern positions with correspondence to the current state of the environment. Numbers correspond to all possible pattern permutations.

Pattern Permutation Option:	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	$\sum$
Numbers of Possible Placements:	2	3	2	2	2	2	2	2	17

Table 2: $CoveringProbability$ **and** $HitProbability$ **distribution in the current state of the environment. Cells** $C4$**,** $D5$ **and** $E4$ **have the highest probability values. Note that the sum of** $HitProbability$ **values equals** 3 **because there are** 3 **unrevealed parts of the current pattern.**

$Position_i identifier$	$Number of Coverings_i$	$CoveringProbability_i$	$HitProbability_i$
B1	1	$1/51 = 0.0196$	$1/17 = 0.0588$
C2	1	$1/51 = 0.0196$	$1/17 = 0.0588$
C3	2	$2/51 = 0.0392$	$2/17 = 0.1176$
C4	**7**	**7/51=0.1373**	**7/17=0.4118**
C5	3	$3/51 = 0.0588$	$3/17 = 0.1765$
C6	2	$2/51 = 0.0392$	$2/17 = 0.1176$
D5	**7**	**7/51=0.1373**	**7/17=0.4118**
D6	4	$4/51 = 0.0784$	$4/17 = 0.2353$
E2	1	$1/51 = 0.0196$	$1/17 = 0.0588$
E3	2	$2/51 = 0.0392$	$2/17 = 0.1176$
E4	**10**	**10/51=0.1961**	**10/17=0.5882**
E5	3	$3/51 = 0.0588$	$3/17 = 0.1765$
E6	2	$2/51 = 0.0392$	$2/17 = 0.1176$
F3	1	$1/51 = 0.0196$	$1/17 = 0.0588$
F4	3	$3/51 = 0.0588$	$3/17 = 0.1765$
F5	2	$2/51 = 0.0392$	$2/17 = 0.1176$
$\sum$	**51**	**51/51=1**	**51/17=3**

5. Solving the Simplified Battleship Game by Using 3-Layer Neural Network Adapted by Using Gradient Descent Method

In the current state of the environment shown in figure 4 and figure 6, area in which complete pattern is hidden ranges from $B2$ to $F6$. This area has size of $5 \times 5 = 25$ cells and contains remaining parts of the pattern.

NN input information consists of $7 \times 7 = 49$ cells. We will use information about the cells as an input for feed-forward *neural network*, shown in figure 7. Information will be enriched by successful next hit attempt performed.

We will use three-layer *neural network* shown in figure 7 as a cognitive decision-making player [14]. In the current state of the environment shown in figure 4, we will create all states that are possible next states (each possible next state with one more uncovered square, with successful hit result predicted). Area where remaining pattern parts is within $5 \times 5 = 25$ area. We will use configurations of these states as an input for *neural network*. This concept is described in figure 8.

Possible next state with the highest output activity we consider as the most probable to fit the current state of the environment, as shown by equation 12.

$$y_{opt} = arg \max_i y_i \tag{12}$$

By taking advantage of back-propagation in gradient descent method, by calculating these partial derivatives we approximate weight and threshold values. Therefore, the neural network shown in figure 7 works as a correct parametric mapping of input vector $\boldsymbol{x}$ to an output activity y when adaptation process is finished. The desired accuracy ε should be set properly to make final mapping precise enough with respect to the effectiveness of the adaptation process.

6. Solving the Simplified Battleship Game by Using 3-Layer Neural Network Adapted by Using Reinforcement Learning

In *machine learning*, various approaches are used. While using the *supervised learning*, input/output pairs are presented (labeled examples). While using *reinforcement learning* [8, 9, 10, 25] inspired by behaviorist psychology, policy is learned on-line by performance.

Reinforcement learning is used in *symbolic* and also in *subsymbolic* approaches. Symbolic rule based system like *Learning Classifier System* (*LCS*) [5, 6, 12] uses *reinforcement learning* method to evaluate *classifiers* during their evolutionary process. *LCS* uses environment feedback as a reward response to an action performed. *Neural network* as a member of *subsymbolic* approaches is also capable of learning output policy by using feedback as a *reward* response to an *action* performed.

In general the *reinforcement learning* concept is based on:

- a set of environment states $\boldsymbol{S}$
- a set of actions $\boldsymbol{A}$
- state transition rules
- transition reward rules
- agent observation rules

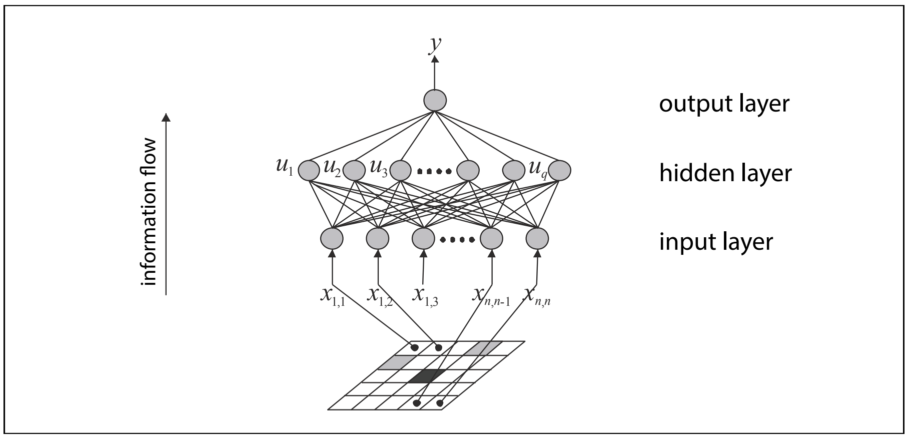

Figure 7: Figure showing the artificial neural network mapped to the of environment cells.

In the current state of the environment $s_t \in \boldsymbol{S}$, a set of possible next states $\boldsymbol{S}_{t+1} \subseteq \boldsymbol{S}$ is defined by state transition rules. Our decision problem is to choose next state $s_{t+1} \in \boldsymbol{S}_{t+1}$ with the highest reward response (predicted). For this purpose we develop policy heuristic on-line, which is built on state-action-reward history.

Performing an action $a_t \in \boldsymbol{A}$ (as an allowed action in state s_t) defines the transition between states s_t and s_{t+1} as described by equation 13.

$$s_{t+1} \leftarrow a_t(s_t); s_t \in \boldsymbol{S}; s_{t+1} \in \boldsymbol{S}_{t+1} \subseteq \boldsymbol{S}; a_t \in \boldsymbol{A} \quad (13)$$

Reward feedback is a scalar environment response value. Environment responses to an *action* performed, i.e. state transition. *Reward* is defined by transition *reward* rules, defined by equation 14.

$$r_{t+1}(s_t, a_t, s_{t+1}) \in \boldsymbol{R}; s_t \in \boldsymbol{S}; s_{t+1} \in \boldsymbol{S}_{t+1} \subseteq \boldsymbol{S}; a_t \in \boldsymbol{A} \quad (14)$$

If state space is completely observable to an agent, his next state *decision making* is described as *MDP*. If state space is partially observable by reason of observation rules, its stochasticity, complexity and/or size, decision making can be described as *POMDP*.

Agent makes decision based on a policy he develops during his history. *Reinforcement learning* itself requires clever exploration mechanisms. Decision policy can be deterministic or stochastic. From the theory of *MDP*s it is known that, the search can be restricted to the set of the so-called stationary policies. A policy is called stationary if the action-distribution returned by it depends only on the current state of the environment.

Stochastic policy, e.g. roulette selection or random decision (with small non-zero probability $1-\varepsilon$ where $0 << \varepsilon < 1$) is appropriate to ensure exploration diversity.

As while using gradient descent method, we will use three-layer *neural network* shown in figure 7 as a cognitive [14] decision-making player.

In the current state of the environment shown in figure 4, we will create all states that are possible next states (each possible next state with one more uncovered square, with successful hit result assumed). This concept is shown in figure 8.

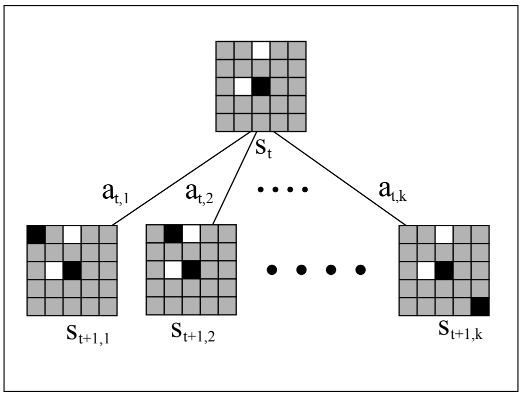

Figure 8: Diagram showing how all k possible states with successful hit prediction are created. These states are evaluated by neural network for action decision.

After an *action* is performed in the environment, environment responses by providing *reward*. This reward is directly dependent on shooting result. Transition reward rules and values depend on implementation. We choose values shown in equation 15.

$$r_{t+1}(s_t, a_t, s_{t+1}) = \begin{cases} 1 & \text{if action has resulted into hit} \\ -1 & \text{if action has not resulted into hit} \end{cases} \quad (15)$$

State s_{t+1} information we use for update of the current state $s_t \leftarrow s_{t+1}$, which will be the new current state in the next iteration.

For *neural network* adaptation we will use *Q-learning*-inspired approach [30] for updating *neural network weights*. *Q-learning algorithm* uses *Q-values*, whose are updated by equation 16.

$$\begin{aligned} Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(s_t, a_t) \times [r_{t+1}(s_t, a_t, s_{t+1}) \\ + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)] \end{aligned} \quad (16)$$

Q-learning algorithm is parameterized by *learning rate* $\alpha(s_t, a_t)$ to control *Q-value* change rate. Parameter γ called *discount factor* is used to reduce *reward* acquisition by *Q-value* update.

We use similar approach to *Q-learning algorithm* to modify *neural network* weights and thresholds. If an *action* performed resulted into hit, success information is back-propagated into *neural network* by modifying *neural network* weights and thresholds. Update starts from output layer through hidden layer, updating weights of active neurons, which have actively participated on the current decision making. Thus the *neural network* is learning environment response by reinforcing *neural network* inner preferences. Thus, if environment response is negative, activities of active neurons are suppressed.

This approach is similar to value back-propagation used in *gradient descent method approach*. The difference is that only binary (success or not success) information is used for *neural network* update. Information about the probability distribution and correct input/output pairs is unavailable. Only success or not success information is used to learn the policy.

7. Simulation Results

We provide data, presented by diagrams showing effectiveness of approaches by revealing two patterns of the simplified Battleship game. *Neural network* will be adapted by two techniques: *supervised learning* and *reinforcement learning* for comparison. We also include results of human playing the simplified Battleship game.

For purpose of approach comparison and quality of game strategy learned we will use these metrics:

- visualized data - graphs of success rate according to hit attempts
- average number of hit attempts performed to reveal both patterns
- our custom score

Our custom score we define as average success rate according to hit attempts evaluated by vector $\boldsymbol{k}$ shown by equation 17.

$$\boldsymbol{k} = \left(\frac{49}{49}, \frac{48}{49}, \frac{47}{49}, \ldots, \frac{2}{49}, \frac{1}{49}\right) \approx (1, 0.98, 0.96, \ldots, 0.04, 0.02) \tag{17}$$

7.1 Neural Network Results of Solving the Simplified Battleship Game By Using the Supervised Learning

After performing 40000 learning iterations the results have shown that *neural network* was adapted by the *supervised learning* properly. We have set number of learning iterations for each adaptation run to 40000, to insure proper adaptation quality.

We provide average results after 40000 learning iterations of 10000 neural network simulations, shown in figure 9. Cumulative success rate of adapted neural network is shown in figure 10.

Results have shown that adapted three-layer feedforward neural network is responding to given input almost equally to the probability-based heuristics, described in section 4.3. Outputs were in correspondence with this heuristic with precision of 96.12%. This correlation has proven that probability-based heuristic is appropriate strategy. Therefore 3-layer feedforward neural network described in section 5 is adaptable to solve the simplified Battleship game effectively.

7.2 Neural Network Results of Solving the Simplified Battleship Game By Using the Reinforcement Learning

Figure 11 is showing average success rate of *neural network* adapted by the *reinforcement learning*.

The analysis of 10000 adaptation runs has shown that *neural network* adapted by the *reinforcement learning* has:

- sufficient number of learning iterations: 160000
- average number of hit attempts performed to reveal both patterns: 20.87423
- our custom score: 6.03007

The average number of hit attempts performed to reveal both patterns was slightly higher than while using *neural network* adapted by the *supervised learning*. *RL* average number of hit attempts was 20.87423, *SL* average number of hit attempts was 19.85320. Graphical difference in average success rate of *NN* adapted by *RL* and *SL* are shown in figure 12.

7.3 Human Solving the Simplified Battleship Game

We have performed a survey, including 12 human players. We have played 100 games totally. Each human was informed about probability-based heuristics described in section 4.3. Average human results are shown in figure 13.

Average results have shown that human player has the following characteristics:

- average number of hit attempts performed to reveal both patterns: 21.02150
- our custom score: 6.02531

To compare a human player and *neural network* adapted by *supervised learning*, in figure 14 we show that *neural network* adapted by *supervised learning* is more effective than average human player.

Comparison of experiment results, including simulation and survey attributes are shown in table 3.

Table 3 compares human player and adapted neural network results. Table includes success rate of both player types until second pattern completely unrevealed. Human player success rate was 41.15% and *neural network* success rate was 47.35%, which is 6.2% more effective.

Results have shown that adapted *neural network* is more effective than human player, even if human player has information about probability-based heuristics.

Table 3 compares results of human player and adapted neural network. Table includes success rate of both player types until second pattern completely unrevealed. Human player success rate was 41.15% and neural network success rate was 47.35%, which is 6.2% more.

Results have shown that adapted neural network is more effective than human player, even if human player has information about probability-based heuristics.

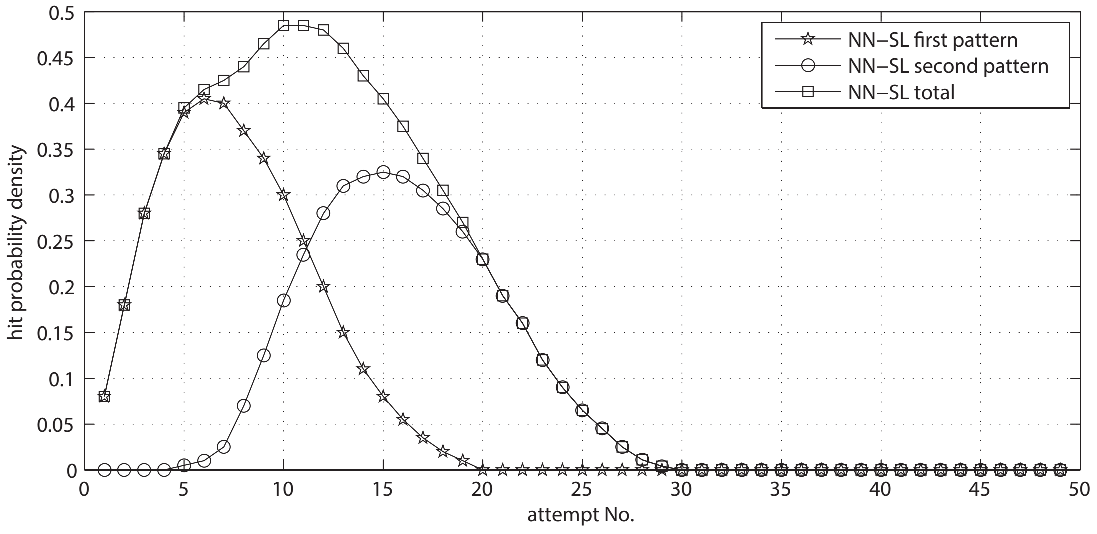

Figure 9: Average success rate of *neural network* adapted by the *supervised learning*. Diagram is showing average success rate of 10000 **adaptation runs. Each run included** 40000 **learning iterations.**

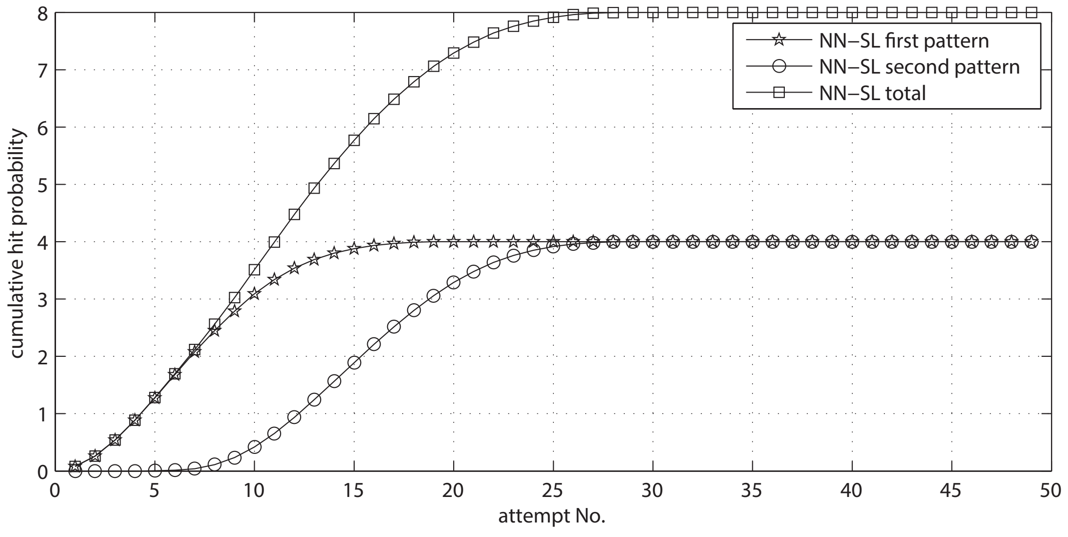

Figure 10: Diagram showing cumulative success rate of *neural network* adapted by the *supervised learning*. Diagram is showing cumulative success rate of 10000 **adaptation runs, each including** 40000 **learning iterations.**

Table 3: Table showing comparison of human players and *neural network* adapted by *supervised learning* simulation experiments, including experiment attributes and results.

Attribute ↓ \ Player Type →	Human	Neural Network
Rounds Played	100	10000
Total Hit Attempts	1944	168952
Successful Hit Attempts	$100 \times 8 = 800$	$10000 \times 8 = 80000$
Unsuccessful Hit Attempts	$1944 - 800 = 1144$	$168952 - 80000 = 88952$
Success Rate	$800/1944 = 41.15\%$	$80000/168952 = 47.35\%$

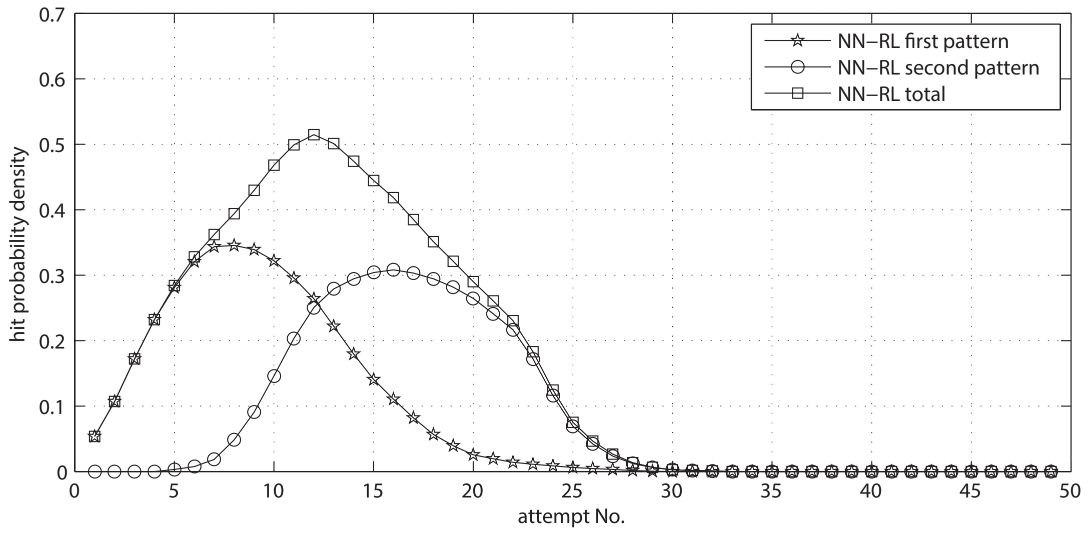

Figure 11: Average success rate of *neural network* adapted by the *reinforcement learning*. Diagram is showing average success rate of 10000 **adaptation runs. Each run included** 160000 **learning iterations.**

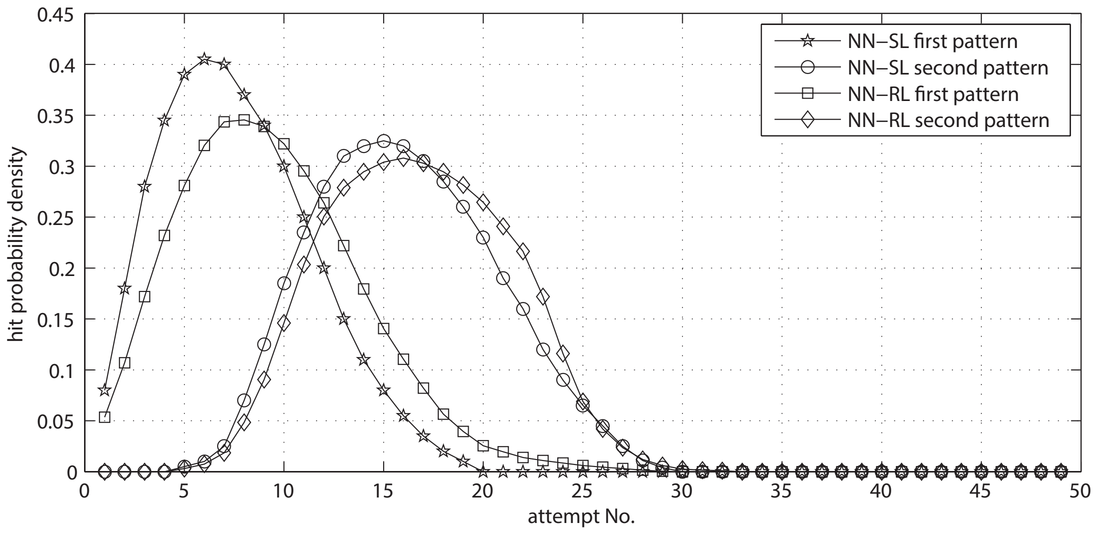

Figure 12: Diagram showing comparison of *neural network* adapted by the *reinforcement learning* and *neural network* adapted by the *supervised learning* by solving the simplified Battleship game.

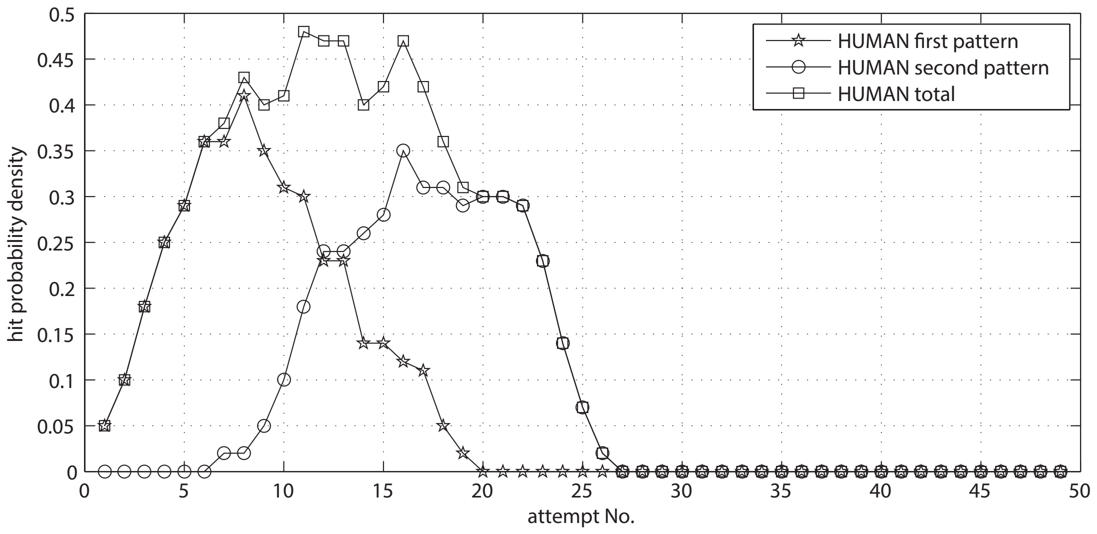

Figure 13: Diagram showing average results of human player playing the simplified Battleship game. Diagram is not quite smooth because of lower data sample amount (100 games played).

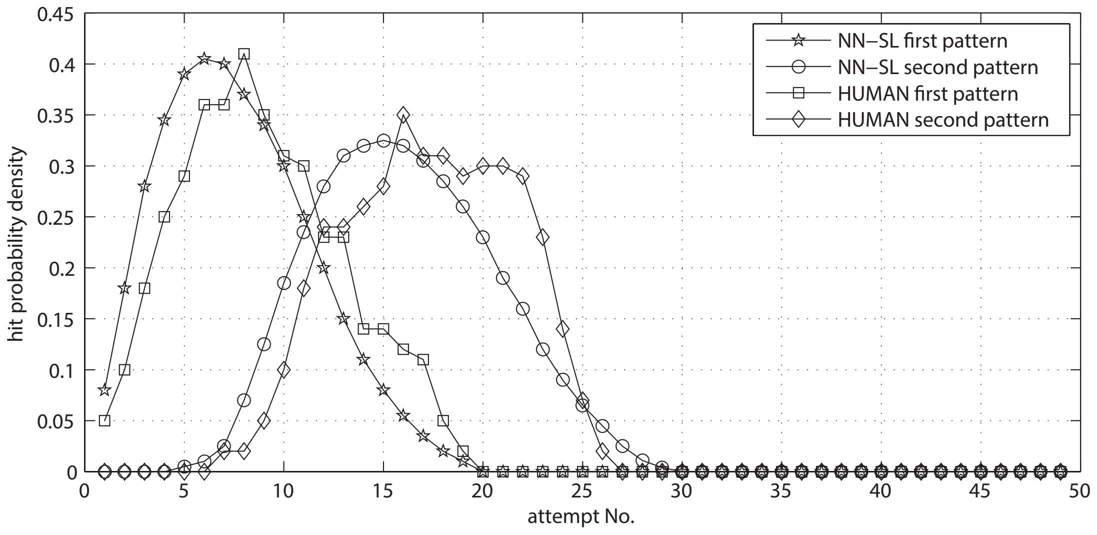

Figure 14: Diagram showing that *neural network* adapted by *supervised learning* is more effective by solving the simplified Battleship game.

7.4 Comparison of Machine Learning Techniques by Solving the Simplified Battleship Game

The most effective was *NN* adapted by *SL*. The second was *NN* adapted by *RL*. We provide gained results of simulations in the table 4. Human has results comparable to *NN* adapted by *RL*.

We have discussed quality of learned strategy by *AI* approach. We will focus on emergence of strategy while using *NN* adapted by *SL* and *RL* techniques (figures 15 and 16). For this purpose we use our defined metrics - hit attempts performed to reveal both patterns and our custom score.

8. Conclusions

Results shown in figure 14 have proven that *neural network* adapted by the gradient descent method is more effective than human player. In table 3, it is shown that neural network has better success rate than human player (6.2% higher).

If we rank used *AI* approaches, *NN* adapted by *SL* provides slightly better strategy quality and much better learning effectiveness.

We have shown that *neural network* is capable of learning complex game strategy by using *reinforcement learning* without knowing problem model. Reinforcement feedback is sufficient for *neural network* training but it needs more iterations. It is expected that *neural network* would need more iterations while learning on reinforcement feedback.

We have completed objectives of our work, depicted in section 2:

- We have designed and described the probability-based heuristic to the Battleship game strategy
- We have studied subsymbolic artificial feed-forward network adapted by supervised learning - gradient descent method to solve the Battleship game
- We have studied subsymbolic artificial feed-forward network adapted by reinforcement learning to solve the Battleship game
- We have compared quality of game strategy learned by used subsymbolic approach
- We have compared game strategy emergence effectiveness of used subsymbolic approach.

Artificial neural networks are inspired by *natural neural networks*. Both *neural network* and human player is able to solve games as the Battleship game after learning the game strategy. This strategy is described by probability-based heuristic.

By study of *AI* behavior we can develop more advanced artificial systems to solve real-world problems. Furthermore we can study artificial systems to understand natural processes, including processes of human decision-making and learning.

Acknowledgements. Author would like to express his gratitude to his dissertation thesis supervisor prof. Vladimír Kvasnička for his valuable support and tutoring.

References

[1] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, Jan. 2002.

[2] M. Campbell, J. A. J. Hoane, and F. Hsu. Deep blue. *Artificial Intelligence*, 134(1-2):57–83, Jan. 2002.

[3] K. Chellapilla and D. B. Fogel. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, 87:1471–1496, 1999.

[4] K. Chellapilla and D. B. Fogel. Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, 5(4):422–428, 2001.

[5] L. Clementis. Model driven classifier evaluation in rule-based system. In V. Snášel, A. Abraham, and E. S. Corchado, editors, *Soft Computing Models in Industrial and Environmental Applications*, volume 188 of *Advances in Intelligent Systems and Computing*, pages 267–276. Springer Berlin Heidelberg, 2013.

[6] L. Clementis. Supervised and reinforcement learning in neural network based approach to the battleship game strategy. In I. Zelinka, G. Chen, O. E. Rössler, V. Snášel, and A. Abraham, editors, *Nostradamus 2013: Prediction, Modeling and Analysis of Complex Systems*, volume 210 of *Advances in Intelligent Systems and Computing*, pages 191–200. Springer International Publishing, 2013.

[7] L. Clementis. Global and local environment state information as neural network input by solving the battleship game. In I. Zelinka, P. N. Suganthan, G. Chen, V. Snášel, A. Abraham, and O. E. Rössler, editors, *Nostradamus 2014: Prediction, Modeling and Analysis of Complex Systems*, volume 289 of *Advances in Intelligent Systems and Computing*, pages 291–300. Springer International Publishing, 2014.

[8] L. T. Dung, T. Komeda, and M. Takagi. Mixed reinforcement learning for partially observable markov decision process. In *International Symposium on Computational Intelligence in Robotics and Automation Proceedings*, pages 7–12. IEEE, 2007.

[9] L. T. Dung, T. Komeda, and M. Takagi. Reinforcement learning for partially observable markov decision process using state clasification. *Applied Artificial Intelligence*, 22(7-8):761–779, 2008.

[10] M. Harmon and S. Harmon. Reinforcement learning: A tutorial, 1996.

[11] H. J. Herik, J. W. H. M. Uiterwijk, and J. Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, Jan. 2002.

[12] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[13] D. Kriesel. *A Brief Introduction to Neural Networks, Zeta version*. 2007. (January 2014) Online available at http://www.dkriesel.com.

[14] V. Kvasnicka. *Kognitívna Veda pre Informatikov*. (January 2014) Online available at http://www2.fiit.stuba.sk/ kvasnicka/CognitiveScience.

[15] V. Kvasnička, J. Pospíchal, and P. Tiňo. *Evolučné algoritmy*. Slovenska Technicka Univerzita, 2000.

[16] P. Lacko, V. Kvasnicka, and J. Pospichal. An emergence of game strategy in multiagent systems. *International Journal of Computational Intelligence and Applications*, 4(3):283–298, 2004.

[17] H. Li, X. Liao, and L. Carin. Region-based value iteration for partially observable markov decision processes. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 561–568. ACM, 2006.

[18] M. Müller. Computer go. *Artificial Intelligence*, 134(1-2):145–179, 2002.

[19] H. Qudrat-Ullah, J. M. Spector, and P. I. Davidsen. *Complex Decision Making: Theory and Practice*. New England Complex Systems Institute book series. Springer, 2010.

[20] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Pearson Education, 2 edition, 2003.

Table 4: Table showing results of human player, subsymbolic approach (*NN* adapted by *supervised learning* and *reinforcement learning*), random, the best and the worst possible player. Results are shown in our defined metrics - hit attempts performed to reveal both patterns and our custom score. *NN* adapted by *SL* has the best adaptation quality. The best *AI* results are marked bold.

player type	hit attempts	custom score	learning iterations
human player	21.02150	6.02531	-
NN-SL player	**19.85320**	**6.22116**	40000
NN-RL player	20.87423	6.03007	160000
random player (unadapted *NN*)	44.37491	4.09478	-
random player (calculated)	$44.\overline{44444}$	4.08163	-
the best possible player	8	7.42857	-
the worst possible player	49	0.73469	-

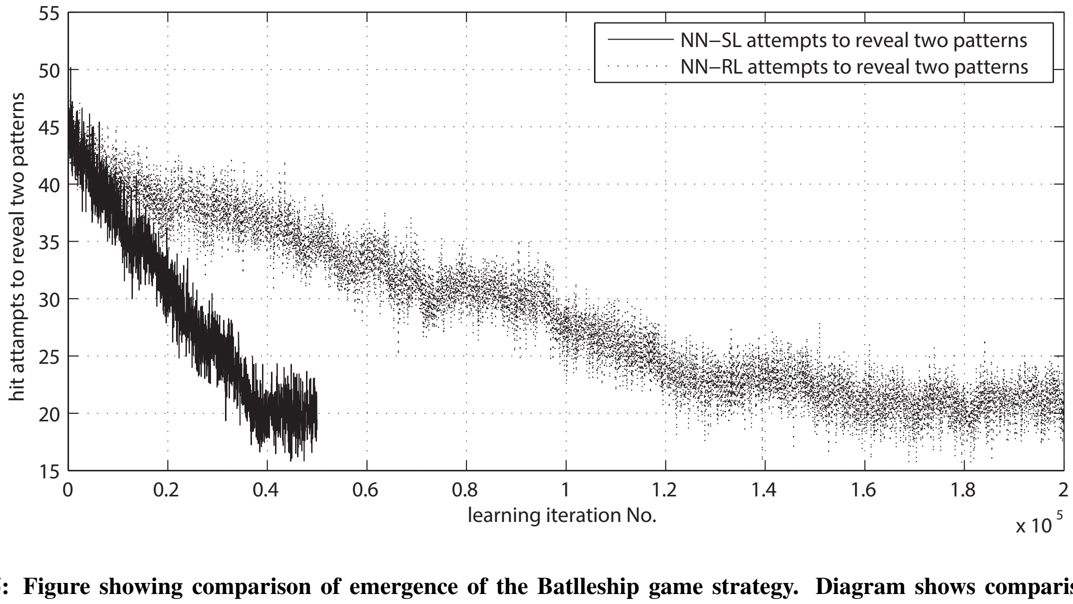

Figure 15: Figure showing comparison of emergence of the Batlleship game strategy. Diagram shows comparison of subsymbolic approach (*NN* adapted by *SL*) and subsymbolic approach (*NN* adapted by *RL*). Strategy quality is measured in hit attempts performed to reveal both patterns. It is visible that *NN* adapted by *SL* was faster (40000 learning iterations) and it has learned the better quality game strategy. The *NN* adapted by *RL* has also learned the good quality game strategy but after 160000 learning iterations.

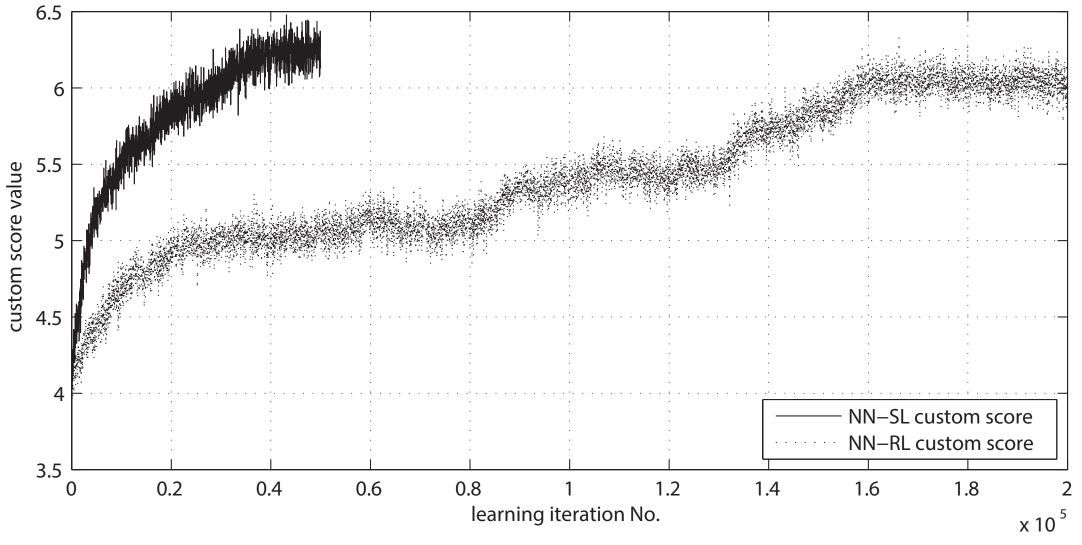

Figure 16: Figure showing comparison of emergence of the Batlleship game strategy. Diagram shows comparison of subsymbolic approach (*NN* adapted by *SL*) and subsymbolic approach (*NN* adapted by *RL*). Strategy quality is measured in our custom score. It is visible that *NN* adapted by *SL* was faster (40000 **learning iterations) and it has learned the better quality game strategy. The *NN* adapted by *RL* has also learned the good quality game strategy but after** 160000 **learning iterations.**

[21] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[22] T. D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2(6):459–473, 1989.

[23] M. Sevenster. Battleships as a decision problem. *ICGA Journal*, 27(3):142–149, 2004.

[24] M. Smith. *Neural Networks for Statistical Modeling*. Thomson Learning, 1993.

[25] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[26] K. Takita and M. Hagiwara. A pulse neural network reinforcement learning algorithm for partially observable markov decision processes. *Systems and Computers in Japan*, 36(3):42–52, 2005.

[27] G. Tesauro. Neurogammon wins computer olympiad. *Neural Computation*, 1(3):321–323, Sept. 1989.

[28] G. Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, Mar. 1994.

[29] V. Vapnik. *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science Series. Springer, 2000.

[30] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. (January 2014) Online available at http://jmvidal.cse.sc.edu/library/watkins92a.pdf.

[31] T. Weise. *Global Optimization Algorithms - Theory and Application*. Self-Published, second edition, 2009. (January 2014) Online available at http://www.it-weise.de/.

[32] M. A. Wiering and T. Kooi. Region enhanced neural q-learning for solving model-based partially observable markov decision processes. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

[33] L. Wu and P. Baldi. Learning to play go using recursive neural networks. *Neural Networks*, 21(9):1392–1400, 2008.

Selected Papers by the Author

L. Clementis. Global and local environment state information as neural network input by solving the battleship game. 2014. In: Advances in Intelligent Systems and Computing. Vol. 289 : Nostradamus 2014: Prediction, Modeling and Analysis of Complex Systems Ostrava, June 23-25, 2014 Proceedings. - Springer Berlin Heidelberg, 2014. - ISBN 978-3-319-07401-6. - S. 291-300.

L. Clementis. Advantage of Parallel Simulated Annealing Optimization by Solving Sudoku Puzzle. 2013. In: Symposium on Emergent Trends in Artifical Intelligence and Robotics (SETINAIR 2013) Košice, Slovakia, September 15-17, 2013 Proceedings. - Springer Berlin Heidelberg, 2013.

L. Clementis. Learning Classifier System Improvement Based on Probability Driven and Neural Network Driven Approaches. 2013. In: 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC 2013) Budapest, Hungary, August 29-30, 2013 Proceedings. - New York : The Institute of Electrical and Electronics Engineers (IEEE), 2013.

L. Clementis. Supervised and Reinforcement Learning in Neural Network Based Approach to the Battleship Game Strategy. 2013. In: Advances in Intelligent Systems and Computing. Vol. 210 : Nostradamus 2013: Prediction, Modeling and Analysis of Complex Systems Ostrava, June 3-5, 2013 Proceedings. - Springer Berlin Heidelberg, 2013. - ISBN 978-3-319-00542-3. - S. 191-200.

L. Clementis. Model Driven Classifier Evaluation in Rule-Based System. - , 2012. In: Soft Computing Models in Industrial and Environmental Applications. Vol. 188 : 7th International Conference on Soft Computing Models in Industrial and Environmental Applications Ostrava, September 5-7, 2012 Proceedings. - Springer Berlin Heidelberg, 2012. - ISBN 978-3-642-32922-7. - S. 267-276.

V. Kvasnička and L. Clementis and J. Pospíchal. An Extension of Hill-climbing with Learning Applied to a Symbolic Regression of Boolean Functions. 2013. In: GECCO 2013 Proceedings of the Genetic and Evolutionary Computation Conference Companion, Amsterdam, The Netherlands, July 6-10, 2013. - New York : Association for Computing Machinery (ACM), 2013. - ISBN 978-1-4503-1963-8. - S. 1129-1134.