Emergence of Game Strategy in Multiagent Systems

Peter Lacko[*]
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
lacko@fiit.stuba.sk

Abstract
In this thesis we focused on subsymbolic approach to machine game play problem. We worked on two different methods of learning. Our first goal was to test the ability of common feed-forward neural networks and the mixture of expert topology. We have derived reinforcement learning algorithm for mixture of expert network topology. This topology is capable to split the problem into smaller parts, which are easier to be solved by an expert neural network. We have compared the quality of strategy emergence between mixture of expert networks and feed-forward networks. Our experiments demonstrate that mixture of experts is able to play a game at the same level as feed-forward networks with equal number of weights.

The second approach derived in this work is reinforcement learning with usage of extended Kalman filer. Extended Kalman filter can be used for neural network training. Its advantage is very high learning rate in terms of training cycles. We have proposed usage of extended Kalman filter for reinforcement learning with TD(0) and Monte Carlo method. We have compared the quality of strategy emergence between extended Kalman filter and TD(λ) approach. Our results show that extended Kalman filter is able to create a game strategy after playing a considerably fewer number of games.

Categories and Subject Descriptors
I.2.6 [**Learning**]: Connectionism and neural nets; I.2.1 [**Applications and Expert Systems**]: Games

Keywords
neural networks, reinforcement learning, board games, Kalman filtering

[*]Recommended by thesis supervisor: Prof. Vladimír Kvasnička
Defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on June 25, 2009.

1. Introduction
First computers where created during second world war and one of the first programmers was Alan Turing. Turing is known in field of computer science for different reasons. During and after the war Turin experimented with machine routines for chess playing. In year 1952, Arthur L. Samuel [14] wrote a program for playing checkers. The basic goal of his work was not a program which would play checkers, but program which can learn how to play it. In his work he tested two approaches – rote learning and learning procedure involving generalization. It was historically the first program which was able to improve itself without any intervention of human. The biggest challenge for artificial intelligence was the game of chess. In 70ties and 80ties the game research was concentrated mainly on chess and brute-force approach. It has been shown that it is a strong correlation between search speed and performance of the program. It resulted to speeding up of searching algorithms in opposite to finding new approaches.

By the end of 80ties the research began to advance and 90ties it brought to us many successes. First game which had a non-human champion was checkers. In year 1994 program Chinook [15] won world tournament between a human and a machine. Next success followed soon after. In year 1997 the project Deep Blue won over a world champion Garry Kasparov [1]. In this period new approaches of learning emerged, for example reinforcement learning [17], implemented in TD-gammon [19]. Quality of its play was comparable to bet players on the world.

Computerized game play has still domains in which we cannot create programs which would be comparable to human players (for example GO [11]). These are mostly games which cannot be solved by classical approaches of artificial intelligence because they have a very high branching factor (player can make many different moves). Exactly these games are then very interesting for the new approaches in artificial intelligence.

2. Thesis Objectives
In this thesis we focused on subsymbolic approach to machine game play problem using reinforcement learning. Compared to supervised learning, reinforcement learning has the advantage that it does not need learning set in form of stimulus-action. Algorithm receives only information about game outcome. We worked on two different methods of reinforcement learning. The first method is aimed at mixture of expert neural networks trained with TD(λ) rule. In the second method we used extended

Kalman filter to train neural networks with reinforcement learning approach. We can summarize thesis objectives to following points:

- To find out the algorithm of reinforcement learning for the topology of the mixture of expert neural networks. This topology is able to divide the solved problem to the sub-tasks which can be solved by the expert neural networks more effectively and accurately.
  - Verify the ability of a mixture of expert networks to divide a problem into the sub-problems.
  - Compare the rapidity and quality of the strategy creation while playing a pair symetric game, for the mixture of the expert networks and a forward neural network.
  - Analyze the way that experts are specialized.
- To derive algorithm of the reinforcement learning with utilization of the Extended Kalman Filter. The Extended Kalman Filter can be used for the training of the neural networks. Its advantage is the fast speed of learning with respect to the training cycle count.
  - To desig and implement possible modifications of the Extended Kalman Filter for the process of the reinforcement learning.
  - Compare the rapidity and quality of the ability to create a strategy while playing the symmetric pair game, for the Modified Extended Kalman Filter and the TD(λ) rule.

3. Formalization of the Game

In this section, we shall deal with a formalization of the game[18] of checkers, which can be applied to all symmetric two player games (chess, go, backgammon, etc.). Let the current position of the game be described by a variable P, this position can be changed by permitted actions – moves constituting a set $A(P)$. Using a move $\acute{a} \in A(P)$, the position P shall be transformed into a new position $\acute{P}$, $P \xrightarrow{\acute{a}} \acute{P}$. An inverse position $\bar{P}$ is obtainable from a position P by switching the color of all black pieces to white and of all white pieces to black. The term inverse position will be important for a formulation of a common algorithm, identical for both first and second player in a symmetric game. We shall use a multiagent approach, and we shall presume, that the game is played by two agents G1 and G2, which are endowed with cognitive devices, by which they are able to evaluate next positions. The formulation of the algorithm is the following (Fig. 1).

Algorithm:

1. The game is started by the first player, $G \leftarrow G1$, from a starting position, $P \leftarrow Pini$.
2. The player G generates from the position P a set of the next permitted positions $A(P) = \{P_1, P_2, ..., P_n\}$. Each such position P_i from the set of these positions is evaluated by a coefficient $0 < z_i < 1$. The player selects as his next position such $\acute{P} \in A(P)$, which is evaluated by the maximum coefficient, $P \leftarrow \acute{P}$. If the position P satisfies condition for victory, then the player G wins and the game continues with the step 4.

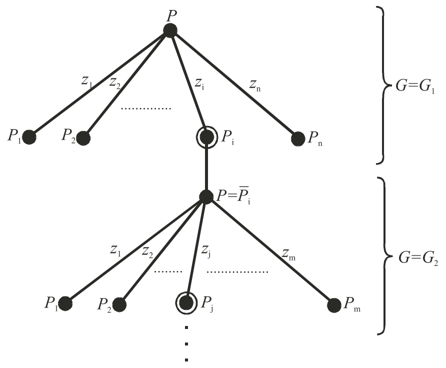

Figure 1: Diagrammatic representation of symmetric 2-player game.

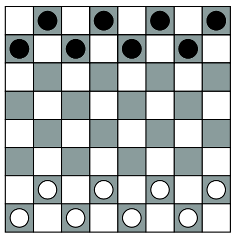

Figure 2: Initial position of simplified game of checkers

3. The other player takes turn in the game, $G \leftarrow G_2$, the position P is generated from the inverse of the current position, $P \leftarrow \bar{P}$, the game continues with the step 2.
4. End of the game.

3.1 Simplified Game of Checkers

The game of checkers is played on a square board with sixty-four smaller squares arranged in an 8×8 grid of alternating colors, see fig. 2. The starting position is with each player having 8 pieces (black, respectively white), on the 8 squares of the same color closest to his edge of the board. Each must make one move per turn. The pieces move one square, diagonally, forward. A piece can only move to a vacant square. One captures an opponent's piece by jumping over it, diagonally, to the adjacent vacant square. If a player can jump, he must. A player wins, if one of his/her pieces reaches a square on the opponent's edge of the board, or captures the last opponent's piece, or blocks all opponent's moves. These rules do not comply with standard rules of checkers. By introduction of these rules, we avoided a finish, when the board is occupied by a small number of pieces. Such finish would be very difficult for a neural network to play successfully. The difficulty for a neural network consists probably in the necessity to plan many moves ahead in such a finish.

In our experiments we used MiniMax[12] algorithm as the opponent. We have changed the search dept of MiniMax algorithm to change the strength of the player. Wwe have prune the tree in defined depth and used folloeinf heuristic function on the leafs:

If we denote the number of our pieces on a game board l and the number of opponent's pieces m, then we can denote by $y_n[i]$ the position of our ith piece along the y axis (the axis towards the opponent's starting part of game board). By $y_s[i]$ we can denote equivalent value of the opponent's pieces.

$$eval = \begin{cases} 100 & \text{player has won} \\ -100 & \text{player has lost} \\ \sum_{i=1}^{l} y_n[i] - \sum_{j=1}^{m} (8 - y_n[i]) & \text{otherwise} \end{cases} \quad (1)$$

This evaluation ensures that the MinMax will try to get its pieces toward the opponent's part of the game board and it will try to prevent the opponent's progress. The play of our player is quite offensive.

4. Adaptation of Cognitive Device of Agent with Temporal Difference TD(λ)-method

In this section we give the basic principles of the used reinforcement learning method, which currently in its temporal difference TD(λ) version belongs to effective algorithmic tools for adaptation of cognitive devices of multiagent systems [12, 17]. The basic principles of reinforcement learning are following: Agent observes the mapping of his/her input pattern to his/her output signal of his/her cognitive device (the output signal is often called an "action" or "control signal"). Agent evaluates the quality of the output signal on the basis of the external scalar signal "reward". The aim of the learning is such an adaptation of the cognitive organ of the agent, which will modify the output signals toward maximization of external "reward" signals. In many cases the "reward" signal is delayed, it arrives only at the end of a long sequence of actions and it can be understood as the evaluation of the whole sequence of actions, whether the sequence achieved the desired goal or not. In this case, the agent must solve the problem of temporal credit assignment, where learning is based on differences between temporal credit assignments for single elements of the whole sequence of actions.

In this section we outline the construction of TD(λ) method as a certain generalization of a standard method of gradient descent learning of neural networks. Let us presume, that we know the sequence of positions of the given agent – player and their evaluation by a real number z

$$P_1, P_2, ..., P_m, z_{reward} \quad (2)$$

where z_{reward} is an external evaluation of the sequence and corresponds to the fact, that the last position P_m means that the given agent won, draw, or lost

$$z_{reward} = \begin{cases} 1 & \text{sequence of positions won} \\ 0.5 & \text{sequence of positions draw} \\ 0 & \text{sequence of positions lost} \end{cases} \quad (3)$$

From the sequence 2 we shall create m couples of positions and their evaluations by z_{reward}, which shall be used as training set for the following objective function

$$E(w) = \frac{1}{2} \sum_{t=1}^{m} \left(z_{reward} - G(\mathbf{x}_t; w)\right)^2 \quad (4)$$

We shall look for such weight coefficients of the neural network – cognitive device, which will minimize the objective function. When we would find out such weight coefficients of the network that the function would be zero, then each position from the sequence $P_1, P_2, \ldots, P_m, z_{reward}$ is evaluated by a number z_{reward}. The recurrent formula for adaptation of the weight coefficients is as follows

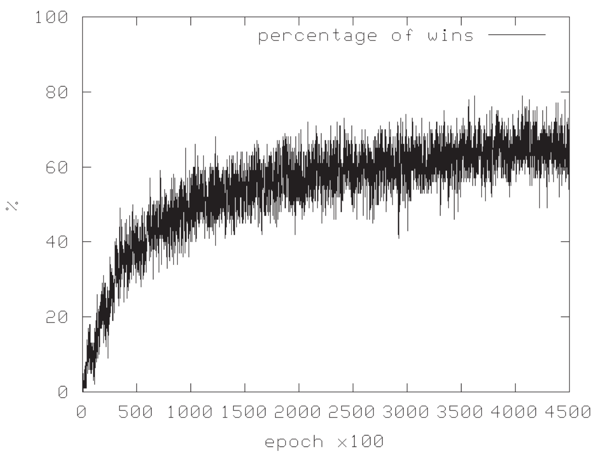

Figure 3: The progress of learning of neural network playing against the MinMax algorithm with the search depth 3

$$w = w - \alpha \frac{\partial E}{\partial w} = w + \Delta w \quad (5)$$

$$\Delta w = \alpha \sum_{t=1}^{m} \left(z_{reward} - z_t\right) \frac{\partial z_t}{\partial w}, \quad (6)$$

where $z_t = G(P_i, w)$ is an evaluation of t-th position P_t by neural network working as a cognitive device. Our goal will be, that all the positions from the sequence 2 would be evaluated by the same number z_{reward}, which specifies, if outcome of the game consisting from the sequence 2 was a win, draw, or loss for the given player. This approach can be generalized to a formula, which creates the basis of the TD(λ) class of learning methods

$$\Delta w = \sum_{t=1}^{m} \Delta w_t \quad (7)$$

$$\Delta w_t = \alpha \left(z_{t+1} - z_t\right) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial z_k}{\partial w}, \quad (8)$$

where the parameter $0 < \lambda < 1$.

4.1 Usage of TD(λ) Method on Checkers Games

In this section, we will describe the results achieved by the neural network. For training and testing we used a two-layered feed-forward network with 64 hidden neurons, the learning rate 0.01 and the coefficient λ=0.9. The network learned after each game, when it was evaluated by 1 if it won, and 0 evaluated it if it lost. After each 100 games, the ratio of won/lost games was marked on the graph. On the fig. 3 there is an evolution of success of neural network learning against algorithm MinMax (working to the search depth 3). It is evident, that the network learned more slowly and even after 450 000 matches achieved victory only in 65% of matches. Nevertheless, it is still an excellent result, since if the network would play as well as our MinMax algorithm searching to the level 3, it would win only 50% of matches.

5. Mixture of Local Experts

When solving complex task, there is almost always that opportunity to split it into smaller subtasks. Such subtask

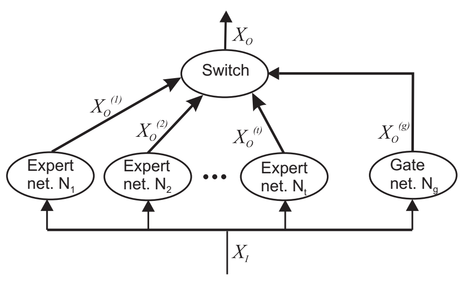

Figure 4: Mixture of local experts

may be better solved by a neural network. We can create a set of neural networks [5, 6], where every network is an expert for some subtask from the original problem. We will use t expert neural networks (fig. 4 :

$$N_i = \left(G^{(i)}, w^{(i)}, \vartheta^{(i)}\right), (i = 1, 2, ..., t) \qquad (9)$$

and one network which will be used as gating network

$$N_g = (G^{(g)}, w^{(g)}, \vartheta^{(g)}) \qquad (10)$$

The purpose of gating network is to choose which expert is best suited for solving given problem. We expect that all networks in set N_t have same number of input and output neurons. Gating network have the same number of input neurons like expert networks, but number of output neurons is t (number of expert networks).

For given combination of $\mathbf{x}_I/\hat{\mathbf{x}}_O$ are the outputs of expert neural networks described with a vector , the output of gating network is vector $\mathbf{x}_O^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_t^{(i)})$. All output activities are in interval (0,1), we will define proportional coefficients $p_i \in (0, 1)$, they describe the respond of gating network on input vector $\mathbf{x}_I$, proportional coefficients are defined as softmax function:

$$p_i = \frac{\mathbf{x}_i^{(g)}}{\sum\limits_{(j=1)}^{(p)} x_j^{(g)}}, (i = 1, 2, ..., t) \qquad (11)$$

Proportional coefficients will be interpreted as probability, that corresponding neural network will be used for clasification. Objective function is then defined as:

$$E = \frac{1}{2}\sum_{i=1}^{t} p_i \left(\mathbf{x}_O^{(i)} - \hat{\mathbf{x}}_O\right)^2 = \frac{1}{2}\sum_{i=1}^{t}\left(\sum_k \left(g_k^{(i)}\right)^2\right) \qquad (12)$$

$$g_k^{(i)} = \begin{cases} p_i\left(x_k^{(i)} - \hat{x}_k\right), k \in V_O \\ 0, k \notin V_O \end{cases}, \qquad (13)$$

where $x_k^{(i)}$ is k output activity of i-th expert neural network. The goal of mixture of experts training is to find the weights of all expert and gatening netorks which minimalize objective function. Adaptation is simmilar to adaptation process of feed-forward network, but equtions for partial derivations slightly different:

$$\frac{\partial E}{\partial w_{ij}^{(g)}} = \frac{\partial E}{\partial \vartheta_i^g} x_j^{(g)} \qquad (14)$$

then proportional derivations $\frac{\partial E_k}{\partial x_i^{(g)}}$ are obtained recurently using:

$$\frac{\partial E_k}{\partial x_i^{(g)}} = f'(\xi_i^{\{g\}})\left(g_i^{(g)} + \sum_l \frac{\partial E_k}{\partial \vartheta_l^{(g)}} w_{li}^{(g)}\right), \qquad (15)$$

where for $g^{(g)}$ we subtitute:

$$g_i^{(g)} = \begin{cases} \left(\frac{1}{2}\left(\mathbf{x}_O^{(i)} - \hat{\mathbf{x}}_O\right)^2 - E\right) \cdot \left(\sum_j \mathbf{x}_j^{(g)}\right)^{-1}, i \in V_O \\ 0, i \notin V_O \end{cases} \qquad (16)$$

5.1 Training of Mixture of Experts Architecture with TD(λ) Rule

We ant to train the mixture of expert architecture with TD(λ) method, so objective function will be defined like this. We are summing up errors from all m moves[10]:

$$E = \frac{1}{2}\sum_{j=1}^{m}\left(\tilde{\mathbf{x}}_O^{(j)} - \hat{\mathbf{x}}_O\right)^2 \qquad (17)$$

We then have two formulas for weights change one for gate network:

$$\Delta w_j^{(i)} = \alpha\left(\tilde{\mathbf{x}}_{O,j+1} - \tilde{\mathbf{x}}_O, j\right)\sum_{k=1}^{j}\lambda^{j-k} p_{i,k}\frac{\partial \mathbf{x}_{O,k}^{(i)}}{\partial w_j^{(i)}}. \qquad (18)$$

and the second one for expert network weights change:

$$\Delta w_{j,l} = \alpha\frac{1}{2}\frac{1}{\left\|\mathbf{x}_{O,j}^{(g)}\right\|}\Delta\tilde{\mathbf{x}}_{O,j}\sum_{k=1}^{j}\lambda^{j-k}\frac{\partial \mathbf{x}_{l,k}^{(g)}}{\partial w_j}\left(\mathbf{x}_{O,k}^{(l)} - \tilde{\mathbf{x}}_{O,k}\right), \qquad (19)$$

where $\Delta\tilde{\mathbf{x}}_{O,j} = (\tilde{\mathbf{x}}_{O,j+1} - \tilde{\mathbf{x}}_{O,j})$ and i is the index of ith expert.

We have tested the mixture of expert architecture and classical feed forward network. Both used the TD(λ) learning rule while playing against MiniMax algorithm. For training and testing of the feed forward network we used a two-layered feed-forward network with different hidden neurons count, the learning rate α=0.1 and the coefficient λ=0.99. The network learned after each game, when it was evaluated by 1 if it won, and 0 evaluated it if it lost. After each 100 games, the ratio of won/lost games was marked on the graph. For training and testing of the mixture of experts architecture we used 4 experts. Every expert was a two-layered feed-forward network with 3 hidden neurons. The gating network was two-layered feed-forward network with 3 hidden neurons. In the Fig. 5 there is an evolution of success of neural networks learning against algorithm MinMax (working to the search depth 2). The graph clearly shows that mixture of experts architecture can decompose the problem of game play into smaller parts which are then better solved by local experts. Mixture of experts was able to play at the same level as feed-forward network with 10 hidden neurons.

To better study how experts are involved to the game, we took 50 winning games of best trained mixture of expert network (it winning rate was 70 %). Then we have divided each game into three equal parts, where the first third of moves is represented as opening phase of the game, second the middle phase, end the third is the ending phase of the game. For every phase we took the mean value of proportional coefficients of the gating network. Results are in table 1. It shows that first expert was used in opening phases, in the middle game, first and second network was used and the end phase of game was played by the fourth expert. Expert number three was never used during the play.

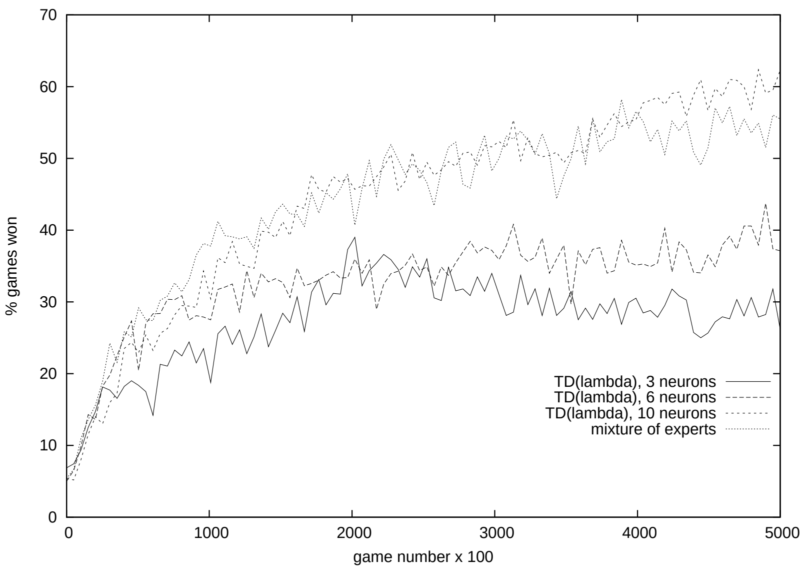

Figure 5: Comparison of learning between feed forward networks and mixture of experts architecture with 4 experts playing against algorithm MinMax (working to the search depth 2).

Table 1: Averages of the proportional coeficients during game phases

	1. ex.	2. ex.	3. ex.	4. ex.
opening of the game	0.65	0.21	0.05	0.09
midle of the game	0.51	0.30	0.05	0.15
end of the game	0.08	0.19	0.02	0.71

With next experiment we have tried to find out, if experts are really specialized or if they can play by them selves the game of simplified checkers. On figure 5 it is clear that network with 3 hidden neurons is able to play game of checkers against MiniMax algorithm working to depth 2 with 30 % winning rate. In table 2 are game outcomes for experts playing by themselves (everyone has 3 hidden neurons). It is clear, that if experts are playing by themselves they cannot play well. Best results where obtained from exert which was in mixture used in ending phase of the game (expert no. 4). Expert which was partial used for all game parts, got good result in the game. Expert which was used by game opening, was not able to play by him selves. Probably he could not recognize the winning positions, or positions which lead in the ending phase of the games to winning. The last expert which was not used in mixture architecture, was not able to play by him selves.

Table 2: Experts victory rate, if they are playing by themselves

	1. exp.	2. exp.	3. exp.	4. exp.
% won	8	15	3	19

6. Utilization of the Extended Kalman Filter for Neural Network Learning

In 1960, R. E. Kalman has published [8] his solution for recursive filtration of linear systems discrete data. In last few years, Kalman's filtration receives great attention because, in its modified version, it can be used for learning of both forward and recursive neural networks [2, 4, 21, 22].

In this section we describe the Kalman Filter and its modifications leading to the so-called Extended Kalman Filter (taken from [4] and [20, 8, 13]) which will be used for the neural network learning. We further describe our proposed modifications of the training process in the reinforcement learning. These modifications were tested on the simplified game of checkers.

6.1 The Kalman Filter

A discrete linear dynamic system can be described by 2 equations – the state equation and the equation of observations. The state of the system can be described by the vector $\mathbf{x}_k$, which is the minimal set of data that exactly describe the dynamic system in the discrete time k.

1. *The state function*

$$\mathbf{x}_{k+1} = \mathbf{F}_{k+1}\mathbf{x}_k + \mathbf{q}_k, \tag{20}$$

where $\mathbf{F}_{k+1,k}$ is the transfer matrix that describes the changes from the state $\mathbf{x}_k$ in the time k to the state $\mathbf{x}_{k+1}$ in the time $k+1$. The process noise $\mathbf{q}_k$ is additive gaussian noise with zero mean value and the covariant matrix $\mathbf{Q}_k$.

2. *The observations equation*

$$\mathbf{y}_{k+1} = \mathbf{H}_k\mathbf{x}_k + \mathbf{r}_k, \tag{21}$$

where $\mathbf{y}_k$ is the observation in the time k and $\mathbf{H}_k$ is the observation matrix. The observation noise $\mathbf{r}_k$is additive white gaussian noise with zero mean value and the covariant matrix $\mathbf{R}_k$.

The Kalman filtration is a method for finding the common optimal solution of the state equation and the observation equation for the unknown state of the system. We can define this problem as trying to find the estimation of state $\mathbf{x}_i$ that has minimal quadratic error, with utilization of all measured data $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k$ for $k \geq 1$.

The Kalman filtration consists of two phases:

1. Prediction (on-time actualization) – a priori prediction of the state and the error covariance.

$$\hat{\mathbf{x}}_k^- = \mathbf{F}_{k,k-1}\hat{\mathbf{x}}_{k-1} \tag{22}$$

$$\mathbf{P}_k^- = \mathbf{F}_{k,k-1}\mathbf{P}_{k-1}\mathbf{F}_{k,k-1}^T + \mathbf{R}_{k-1} \tag{23}$$

2. Correction (actualization of observations and correction) – adjusts the estimations in compliance with the observation $\mathbf{y}_k$.

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}_k^T[\mathbf{H}_k\mathbf{P}_k^-\mathbf{H}_k^T + \mathbf{R}_k]^{-1} \tag{24}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-) \tag{25}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^- \tag{26}$$

6.2 The Extended Kalman Filter

The Kalman filter described in the previous section has served for the problem of finding the state vector in the linear dynamic system. If we intend to use the Kalman filter for the neural network training, we need to modify it to obtain the non-linear version of the filter. (As the neural networks are non-linear systems). By extending the Kalman filter with the linearization procedure, we get the Extended Kalman Filter – EKF. This extension is possible because the original Kalman Filter is described by differential equations for the discrete-time system. The behavior of the neural network can be described as a non-linear system:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{r}_k \tag{27}$$

$$\mathbf{y}_k = \mathbf{h}(\mathbf{w}_k, \mathbf{x}_k) + \mathbf{q}_k \tag{28}$$

From the equation 27 we can see that the state of a neural network is characterized as a stationary process disturbed by the process noise $\mathbf{r}_k$. The state of the network $\mathbf{w}$ is the vector of the network's weights. The observation equation describes the output of the neural network as a linear function of the input vector $\mathbf{x}_k$ and the weights vector $\mathbf{w}_k$. This equation is extended by the random measurement noise $\mathbf{q}_k$. The measurement noise is white with zero mean value and covariance $E[\mathbf{q}_k\mathbf{q}_k^T] = \mathbf{Q}_k$. The process noise is white with zero mean value and covariance $E[\mathbf{r}_k\mathbf{r}_k^T] = \mathbf{R}_k$.

The training process of the neural network with utilization of the Extended Kalman Filter can be characterized as a process of estimation of the state $\mathbf{w}$ (the network's weights) with minimal quadratic error, based on the entire observation history. As the state function of the system is linear (actually, $f(\mathbf{w}_k) = \mathbf{w}_k$), we can describe the training process like this:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_k\mathbf{H}_k^T[\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k]^{-1} \\ \hat{\mathbf{w}}_{k+1} &= \hat{\mathbf{w}}_k + \mathbf{K}_k[\mathbf{y}_k - h(\hat{\mathbf{w}}_k, \mathbf{x}_k)] \\ \mathbf{P}_{k+1} &= \mathbf{P}_k - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_k + \mathbf{Q}_k \end{aligned} \tag{29}$$

where

n_w	– number of all network's weights,
n_o	– number of the output neurons,
$\mathbf{w}[n_W \times 1]$	– the system's state (vector of all network's weights),
$\mathbf{h}(\cdot)[n_o \times 1]$	– the output function of the network (vector of all network's outputs),
$\mathbf{y}[n_o \times 1]$	– the vector of required output values,
$\mathbf{K}[n_w \times n_o]$	– the Kalman gain matrix,
$\mathbf{P}[n_w \times n_w]$	– Covariant matrix of the state error,
$\mathbf{Q}[n_w \times n_w]$	– Covariant matrix of the process noise,
$\mathbf{R}[n_o \times n_o]$	– Covariant matrix of the observation noise,
$\mathbf{H}[n_o \times n_w]$	– The observation matrix (Jacobian).

The components of the $\mathbf{P}_0$ matrix are typically chosen from 100$\mathbf{I}$ to 1000$\mathbf{I}$ (where $\mathbf{I}$ is the identity matrix). The matrix $\mathbf{Q}_0$ is initialized by values of$10^{-1}\mathbf{I}$ to 0.1$\mathbf{I}$. The components of the $\mathbf{H}$ matrix are obtained as the values of the partial differentiation of the network's outputs to its weights. In the case of the forward network we use the backpropagation algorithm.

6.3 Reinforcement Learning with Utilization of EKF

Our ambition is to modify the Extended Kalman Filter in such way that we can utilize it in the process of the reinforcement learning. In this section we will describe the possible modification of the network's training procedure with usage of the Extended Kalman Filter, for the reinforcement learning process. The proposed approach is compared with the TD(λ) algorithm, on the games Tic-Tac-Toe and simplified checkers.

In the process of reinforced learning we are trying to train the neural network to behave like an unknown value function. In the case of games playing, the value function gives us the evaluation of the game's state s_t. This evaluation represents the estimation of the concrete game's outcome, supposing that we are in a concrete state s_t. The main thought of the method is, that the whole sequence of states through one game $(s_1, \ldots, s_T)$ should be evaluated equally as the final outcome of the game. When we try to write this thought down mathematically, we get the so-called Monte Carlo learning rule [17]:

$$\begin{aligned} &\text{For each } t = 1 \ldots T: \\ &\quad V(s_t) = V(s_t) + \alpha(reward - V(s_t)), \end{aligned} \tag{30}$$

where α is the learning speed, $V(s)$ is the value function (which in our case means the network's output) and *reward* is the game's outcome.

Let us define the equations 29 as functions for `EKF(input, expected output)`. The learning algorithm is then very simple:

```
for t = 1 až T rob EKF(s[t], reward);
```

The Monte Carlo method needs not to be absolutely appropriate for our needs. For example, it can lead to evaluation of the first move by 0 (the player has lost the game) although that the first move is right and only leads to the more states with possibility of user's errorneous move. Therefore it is better to modify the equation 30 to TD(0)

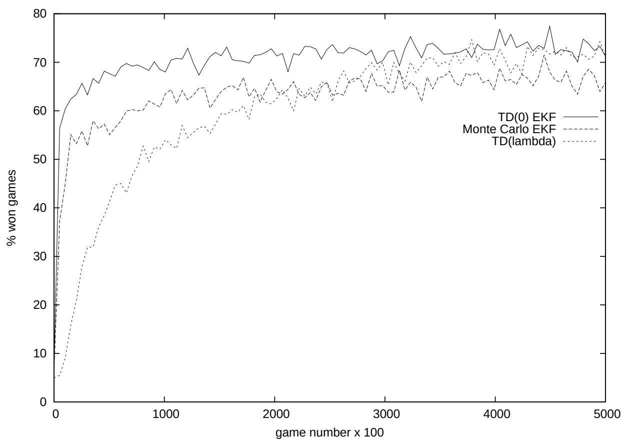

Figure 6: Learning progress of neural network (hidden layer with 20 neurons) using TD(λ) and EKF against the MinMax algorithm with the search depth 2. (Simplified checkers game)

rule [17]. This approach modifies the states' evaluation function in such way that the evaluation of the state in time t is altered according to the evaluation of the state $t+1$.

$$\begin{aligned}&\text{For each } t = 1 \dots T:\\ &\quad V(s_t) = V(s_t) + \alpha(V(s_{t+1}) - V(s_t)),\end{aligned} \tag{31}$$

where $V(s_{T+1}) = reward$. The learning algorithm can be described as:

```
pre t = 1 až T rob EKF(s[t], V(s[t+1]));
```

In the next section we confront the reinforcement learning algortihm with the utilization of EKF and the TD(λ) algorithm.

6.4 The Achieved Results of the Reinforcement Learning with the EKF Utilization

The testing of the learning algorithms has been performed on the game of Simplified Draughts. The neural network consisted of 20 hidden neurons. The learning speed for the TD(λ) algorithm has been set to 0.1 and the λ has been set to 0.99. The EKF has been initialized with values $\mathbf{P}_0 = 100\mathbf{I}$, $\mathbf{Q}_0 = 0\mathbf{I}$ a $\mathbf{R}_0 = 1\mathbf{I}$. Each algorithm was tested with 10 randomly initialized weight values that were equal for both networks. The learning of each network has run once.

In the picture 6 is shown how the algorithms were learning to play against the opponent – the MinMax algorithm of depth 2. The TD(0) EKF has converged to a good solution quite rapidly. But it is worth to notice that the TD(λ) algorithm was able to play on the comparable level in about 500 000 games. The Monte Carlo EKF method did not achieved the results of the previous two approaches.

The second chart 7 shows the first 20 000 games. It can be seen that the TD(0) EKF needed about 10000 games to converge to the configuration that is able to win 60 % of games. The Monte Carlo EKF achieves good results relatively fast but is not able to create equally good strategy as the TD(λ) and TD(0) EKF algorithms.

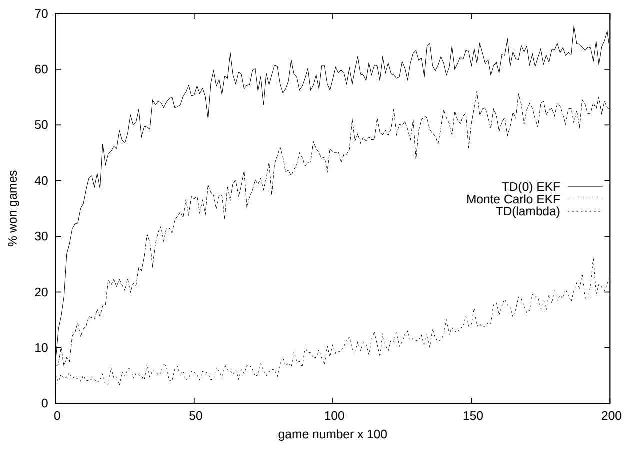

Figure 7: Learning progress of neural network (hidden layer with 20 neurons) using TD(λ) and EKF against the MinMax algorithm with the search depth 2. (Simplified checkers game – detail)

7. Conclusion

Artificial intelligence has been dealing with game play problems since the origin of computer science. In this research field we were able to develop systems which exceed humans in playing. However, there are games in which humans still overcome computers. In this space, novel approaches and algorithm design emerge.

In this thesis we focused on subsymbolic approach to machine game play problem. We worked on two different methods of learning. The first method is aimed at mixture of expert neural networks trained with TD(λ) rule. In the second method we used extended Kalman filter to train neural networks with reinforcement learning approach.

Our first goal was to test the ability of common feed-forward neural networks and the mixture of expert topology. We have derived reinforcement learning algorithm for mixture of expert network topology. This topology is capable to split the problem into smaller parts, which are easier to be solved by an expert neural network. We have compared the quality of strategy emergence between mixture of expert networks and feed-forward networks. Our experiments demonstrate that mixture of experts is able to play a game at the same level as feed-forward networks with equal number of weights. We have demonstrated that experts are specialized according to game part. Single expert can not play the whole game by itself.

The second approach derived in this work is reinforcement learning with usage of extended Kalman filer. Extended Kalman filter (EKF) can be used for neural network training. Its advantage is very high learning rate in terms of training cycles. We have proposed usage of extended Kalman filter for reinforcement learning with TD(0) and Monte Carlo method. We have compared the quality of strategy emergence between the two proposed extended Kalman filter methods and TD(λ) approach. In our experiments Monte Carlo EKF method was unable to achieve the same level of game play as the other two methods. Game play quality is same for TD(λ) and EKF with TD(0) method, but our results show that reinforcement learning with extended Kalman filter is able to create a game strategy after playing a considerably fewer number of games.

Acknowledgements. This work was supported by Scientic Grant Agency of Slovak Republic under grants No. 1/0062/03, 1/1047/04, 1/0804/08 and No. 1/4053/07 and by Science and Technology Assistance Agency under grant APVT-20-002504.

References

[1] M. Campbell, Jr. A. J. Hoane, F. Hsu. Deep Blue. *Artificial intelligence*, volume 134, No. 1-2, january 2002.

[2] M. Čerňanský, L. Benušková. Simple recurrent network trained by RTRL and extended Kalman filter algorithms. *Neural Network World*, volume 13, No. 3, pages 223–234, 2003.

[3] S. T. Hagen, B. Krsse. Linear Quadratic Regulation using Heinforcement Learning. In *Proc. BENELEARN-98, 8th Belgian-Dutch Conference on Machine Learning*, pages 39–46, 1998.

[4] S. Haykin. Kalman Filtering and Neural Networks. New York: John Wiley & Sons, Inc., ISBN: 0-471-36998-5, 2002.

[5] R. A. Jacobs, I. JordanM, S. Nowlan, G.E. Hinton. Adaptive mixtures of local experts. *Hinton Neural Computation*, 3, pages 1–12, 1991.

[6] M. Jordan, R. Jacobs. Modular and hierarchical learning systems. *The Handbook of Brain Theory and Neural Networks*, Cambridge, MA, MIT Press, 1995. aviable at `http://www.cs.berkeley.edu/~jordan/papers/handbook.ps.gz` (28.2.2006)

[7] S.J. Julier, J.K. Uhlmann. New Extension of the Kalman Filter to Nonlinear Systems. In *AProceedings of AeroSence: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls*, March 2000.

[8] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME, Series D, Journal of Basic Engineering*, No. 82, pages 35-–45, 1960.

[9] V. Kvasnička. Adaptive mixture of local neural networks. *Neural Network World*, volume 3, No. 2, pages 161-174, 1993.

[10] P. Lacko, V. Kvasnička. Mixture of Expert Used to Learn Game Play. In *Lecture Notes in Computer Science*, Vol. 5163, *Artificial Neural Networks - ICANN 2008, Part I, 18th International Conference*, Prague, Czech Republic, Springer Berlin/Heidelberg, pages 225–234, 2008.

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

[12] P. Návrat et al. *Artificial Intelligence.* STU, Bratislava, 2000. (in Slovak)

[13] G.S. Patel. *Modeling Nonlinear Dynamics with Extended Kalman Filter Trained Recurrent Multilayer Perceptrons.* Diploma thesis. McMaster University, 2000.

[14] A.L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development 3*, pages 211–229, July 1959.

[15] J. Schaeffer, R. Lake, P. Lu, M. Bryant. Chinook is the world checkers champion. available at `http://www.cs.ualberta.ca/events/csdays/1999/openhouse/chinook.html` (1.2.2009)

[16] S. Singh, S. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In M. C. Mozer, M. I. Jordan, and T. Petsche editors, *NIPS-9*, The MIT Press, page 974, 1997.

[17] R.S. Sutton. Learning to Predict by the Method of Temporal Difference. *Machine Learning 3*, 1988.

[18] R.S. Sutton, A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, 1998.

[19] G. Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, volume 38, No. 3, March 1995.

[20] P. Trebatický. Recurrent neural network training with the extended Kalman filter. *Diploma thesis*, FIIT STU, Bratislava, May 2004. (in Slovak)

[21] P. Trebatický. Recurrent Neural Network training with the Kalman Filter-based techniques. *Neural network world : international Journal on Neural and Mass-Parallel Computing and Information Systems*, volumen 15, No. 5, pages 471–488, 2005.

[22] P. Trebatický, J. Pospíchal. Neural Network Training with Extended Kalman Filter Using Graphics Processing Unit. *Lecture Notes in Computer Science*, volume 5164, *Artificial Neural Networks - ICANN 2008*, Prague, Czech Republic, pages 198–207, September 2008.

[23] R. Zaman, D.C. Wunsch III. TD methods applied to mixture of experts for learning 9x9 GO evaluation function. In *IJCNN 99*, volume 6, pages 3734–3739, July 1999.

Selected papers by the Author

P. Lacko, V. Kvasnička, J. Pospíchal. An Emergence of Game Strategy in Multiagent Systems. In *International Journal of Computational Intelligence and Applications.* volume 4, No. 3. Singapur: World Scientific Publishing Company, pages 283–298, 2004.

P. Lacko. Evolutionary emergence of game strategy in multiagent systems.. In *Cognition and Artificial Life IV.* Opava, Silesian University, pages 333–351, 2004. (in Slovak)

P. Lacko, V. Kvasnička. Training of game strategy with adaptive combination of local neural networks. In *Cognition and Artificial Life VIII.* Opava, Silesian University, pages 193–199, 2008. (in Slovak)

P. Lacko, V. Kvasnička. Training of neural networks with reinforcement learning Monte Carlo method using enhanced Kalman filter. In *Cognition and Artificial Life IX.* Opava, Silesian University, pages 161–168, 2009. (in Slovak)

P. Lacko. A Comparison of symbolic and subsymbolic approach in artificial inteligence. In *MENDEL 2005: 11th International Conf. on Soft Computing*, Brno: Institute of Automation and Computer Science Faculty of Mechanical Engineering Brno University of Technology, pages 163–168, 2005.

P. Lacko, V. Kvasnička. Mixture of Expert Used to Learn Game Play. In *Lecture Notes in Computer Science.* Vol. 5163, *Artificial Neural Networks – ICANN 2008*, Prague, CzR, Springer, pages 225–234, 2008.

P. Lacko. A Comparison of Symbolic and Subsymbolic Approach of the Emergence of Game Strategy. In *Cognition and Artificial Life V.* Sv. 1. Opava, Silesian University, pages 313–321, 2005. (in Slovak)

P. Lacko. Reinforcement learning. In *Cognition and Artificial Life VI.* May 28 - June 1 2006, Třešť, Czech republic. Opava, Silesian University, pages 253–258, 2006. (in Slovak)

P. Lacko. An Emergence of Game Strategy in Multiagent Systems. In *Student Research Conference 2005: Proceedings in Informatics and Information Technologies.* STU Bratislava, pages 41–48, 2005.

P. Lacko, V. Kvasnička. Comparison of Symbolic and Subsymbolic approach in Artificial Intelligence Game Play. In *Student Research Conference 2006: Proceedings in Informatics and Information Technologies.* STU Bratislava, 2006.

P. Lacko. Reinforcement Learning in Artificial Intelligence. In *Student Research Conference 2007. 3rd Student Research Conference in Informatics and Information Technologies Bratislava*, STU Bratislava, 2007.

P. Lacko, V. Kvasnička, J. Pospíchal. An emergence of game strategy in multiagent systems. In *1st. Slovak-Japanese Seminar on Intelligent Systems*, Herľany, Slovakia, 2005.