Schriftenreihe CIplus, Band 3/2014

Thomas Bartz-Beielstein, Wolfgang Konen, Horst Stenzel, Boris Naujoks

Online Adaptable Learning Rates for the Game Connect-4

Samineh Bagheri, Markus Thill, Patrick Koch and Wolfgang Konen

Fachhochschule Köln
Cologne University of Applied Sciences

Online Adaptable Learning Rates for the Game Connect-4

Samineh Bagheri, Markus Thill, Patrick Koch and Wolfgang Konen

Abstract

Learning board games by self-play has a long tradition in computational intelligence for games. Based on Tesauro's seminal success with TD-Gammon in 1994, many successful agents use temporal difference learning today. But in order to be successful with temporal difference learning on game tasks, often a careful selection of features and a large number of training games is necessary. Even for board games of moderate complexity like Connect-4, we found in previous work that a very rich initial feature set and several millions of game plays are required. In this work we investigate different approaches of online-adaptable learning rates like Incremental Delta Bar Delta (IDBD) or Temporal Coherence Learning (TCL) whether they have the potential to speed up learning for such a complex task. We propose a new variant of TCL with geometric step size changes. We compare those algorithms with several other state-of-the-art learning rate adaptation algorithms and perform a case study on the sensitivity with respect to their meta parameters. We show that in this set of learning algorithms those with geometric step size changes outperform those other algorithms with constant step size changes. Algorithms with nonlinear output functions are slightly better than linear ones. Algorithms with geometric step size changes learn faster by a factor of 4 as compared to previously published results on the task Connect-4.

Index Terms

Machine learning, board games, self-play, reinforcement learning, temporal difference learning (TDL), temporal coherence, learning rates, self-adaptation, online adaptation, n-tuple systems.

1 Introduction

HUMANS are very efficient and fast in learning complicated tasks in new domains. As Sutton [1] has pointed out, information theoretic arguments suggest that the learning progress is often too rapid to be justified by the data of the new domain alone. The key to success is, as it is commonly believed, that humans bring a set of correct biases from other domains into the new domain. These biases allow to learn faster, because biases direct them to prefer certain hypotheses over others or certain features over others. If machine learning algorithms shall achieve a performance similar to humans, they probably need to acquire biases as well. Where do these biases come from?

Already in 1992 Sutton [1] suggested the Incremental Delta Bar Delta (IDBD) algorithm where the biases are understood as learning rates which can be different for different trainable parameters of any underlying algorithm. The key idea of IDBD is that these learning rates are not predefined by the algorithm designer but they are adapted as hyperparameters of the learning process themselves. Sutton [1] expected such adaptable learning rates to be especially useful for

• *All authors are with the Faculty of Engineering and Computer Science, Cologne University of Applied Sciences, Cologne, Germany. E-mail: wolfgang.konen@fh-koeln.de*

Manuscript received March 2nd, 2014; revised version accepted Oct 29th, 2014.

nonstationary tasks or sequences of related tasks and he demonstrated good results on a small synthetic nonstationary learning problem (featuring 20 weights).

A similar idea was suggested as Temporal Coherence Learning (TCL) by Beal and Smith in 1999 [2], [3], who directly modified the Temporal Difference Learning (TDL) algorithm to take into account self-tuning learning rates. Several other online learning rate adaptation algorithms have been proposed over the years (see Sec. 2) and it is the purpose of this work – as a case study in machine learning – to make a comprehensive comparison on a larger game-learning benchmark task.

Board games and learning how to play them constitute challenging tasks in machine learning (ML) and artificial intelligence (AI). They are challenging, because the action (a move) has to be taken now, but the payoff (win or loss) occurs later, at the end of the game. The most advanced method in ML to address this problem is well-known TDL, a method based on reinforcement learning (RL). TDL was applied as early as 1957 by Samuel [4] to checkers and gained more popularity through Sutton's work in 1984 and 1988 [5], [6]. It became famous in 1994 with Tesauro's TD-Gammon [7], which learned to play backgammon at expert level.

Game learning with TDL constitutes a nonstationary learning task: For most board positions the target value will change during learning (see Sec. 3.3 for details). Thus, RL and TDL for board games are expected to benefit from bias learning approaches such as the above-mentioned IDBD [1].

In this paper we consider the game Connect-4 as a specific example, which is a solved game (see Sec. 3.1) but no algorithm to learn it by self-play was known until recently. In a previous work [8] we were able to show that it is possible to learn this game with TDL and n-tuples [9] just by self-play. We achieved a playing strength close to the perfect playing agent. However, the learning process required several millions of self-play games, thus being far off human performance. In this work we investigate whether new strategies can achieve the same (or even better) strength within fewer training games. Two solution paths can be considered here:

a) Tuning, i.e. finding faster-learning solutions by optimizing the hyperparameters of the learning algorithm. The drawback of this solution is that the tuning results usually apply only to the specific learning task, i.e. Connect-4 with this TDL algorithm. Each new algorithm or each new game requires a completely new tuning.
b) Self-tuning learning algorithms like IDBD or others (see Sec. 3), which automatically adapt certain hyperparameters (here: the learning rates) as learning progresses. This avoids manual intervention and there is hope that the same self-tuning scheme can be applied to other games as well and that the insights gained are transferable to other learning tasks as well.

We follow mainly the second path in this paper and try to answer the following research questions:

1) Can online learning rate adaptation be successfully applied to problems with millions of weights?
2) How robust are online learning rate adaptation algorithms with respect to *their* meta-parameters?
3) Can online learning rate adaptation algorithms speed up learning as compared to TDL?

The rest of this paper is organized as follows: Sec. 2 briefly reviews related work. Sec. 3 introduces the methods TDL, TCL, and IDBD. It presents with TCL-EXP our new synthesis between TCL and IDBD. Sec. 4 describes our experimental setup and our results on the game Connect-4. Sec. 5 discusses the parameter sensitivity of the different methods and Sec. 6 summarizes our main findings.

2 Related work

Several online learning rate adaptation schemes have been proposed over the years: IDBD [1] from Sutton is an extension of Jacobs' [10] earlier DBD algorithm: it allows immediate updates instead of batch updates. Sutton [11] proposed some extensions to IDBD with the algorithms K1 and K2 and compares them with the Least Mean Square (LMS) algorithm and Kalman filtering. Almeida [12] discussed another method of step-size adaptation and applied it to the minimization of nonlinear functions.

Schraudolph [13] and, more recently, Li [14] extended IDBD-variants to the nonlinear case: Schraudolph's ELK1 extends K1 and performs an update with the instantaneous Hessian matrix of a suitable chosen loss function. The algorithm's complexity is $O(n^2)$ where n is the number of parameters to learn. Li's KIMEL algorithm transforms the nonlinear input data with a kernel into a high-dimensional but linear feature space where linear IDBD is applied. Sutton and Koop [15], [16] developed another nice nonlinear extension IDBD-nl of the original IDBD algorithm using the logistic sigmoid function. It was applied for learning the game Go.

Recently, Mahmood and Sutton [17], [18] proposed with Autostep an extension to IDBD which has much less dependence on the meta-step-size parameter than IDBD. In the same year, Dabney and Barto [19] developed another adaptive step-size method for temporal difference learning, which is based on the estimation of upper and lower bounds. Again, both methods are proposed only for *linear* function approximation. Schaul et al. [20] propose a method of tuning-free learning rate adaptation especially well-suited for large neural networks. RPROP [21] is another earlier version of a neural network algorithm with individual learning rates for each weight.

For the game Connect-4 – although weakly solved in 1988 by Allen [22] and Allis [23] (Sec. 3.1) – only rather few attempts to *learn* it (whether by self-play or by learning from teachers) are found in the literature: Schneider et al. [24] tried to learn Connect-4 with a neural network, using an archive of saved games as teaching information. Stenmark [25] compared TDL for Connect-4 against a knowledge-based approach from Automatic Programming and found TDL to be slightly better. Curran et al. [26] used a cultural learning approach for evolving populations of neural networks in self-play. All the above works gave no clear answer on the *true* playing strength of the agents, since they did not compare their agents with a perfect-playing Minimax agent.

Lucas showed that the game of Othello, having a somewhat greater complexity than Connect-4, could be learned by TDL within a few thousand training games with the n-tuple-approach [9]. Krawiec et al. [27] applied the n-tuple-approach in (Co-) Evolutionary TDL and outperformed TDL in the Othello League [28]. This stirred our interest in the n-tuple-approach and we applied it successfully to Connect-4 in our previous work [8]. The results against a perfect-playing Minimax agent are summarized in Sec. 4.3.

3 Methods

3.1 Connect-4

The game Connect-4 is a two-player game played on a board with 7 vertical slots containing 6 positions each (Fig. 1). Player Yellow (1st) and player Red (2nd) place one piece per turn in one of the available slots and each piece falls down under the force of gravity into the lowest free position of the slot. Each player attempts to create horizontal, vertical or diagonal piece-lines of length four. Fig. 1 shows an example position where Yellow would win if Red does not block this by placing a red piece into the right slot.

Connect-4 has a medium state space complexity of $4.5 \cdot 10^{12}$ board positions [29]. A game is said to be weakly solved if its game theoretic value and a strategy for perfect play from the initial value is known. Connect-4 was weakly solved in 1988 independently by Allen [22] and by Allis [23]: Yellow (the 1st player) wins, if she places her first piece in the middle slot. Tromp [30] solved the game Connect-4 strongly, i. e. for every intermediate position.

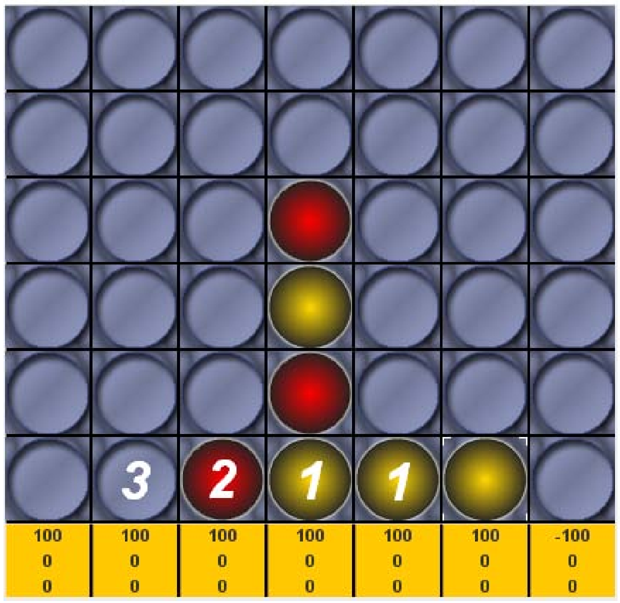


Fig. 1. Connect-4 board with an example 4-tuple '3-2-1-1' (see Sec. 3.2)

We developed a Minimax agent combined with a pre-calculated 8-ply[1] or 12-ply-opening database [8], [31] and it finds the perfect next move (or moves, if several moves are equally well) for each board position within fractions of a second. This agent will be used in our experiments, but only as reference or evaluation agent. It is by no means used for any training purpose.

3.2 N-tuples and LUTs

N-tuples in General: N-tuple systems were first introduced in 1959 by Bledsoe and Browning [32] for character recognition. Recently, Lucas [9] proposed employing the n-tuple architecture for game-playing purposes. The n-tuple approach works in a way similar to the kernel trick used in support vector machines (SVM): The low dimensional board is projected into a high dimensional sample space by the n-tuple indexing process [9].

N-tuples in Connect-4: An n-tuple T_ν is a sequence $[a_{\nu 0}, \ldots, a_{\nu n-1}]$ of n different board cells $a_{\nu j} \in \{0, \ldots, 41\}$. For example, the four white digits in Fig. 1 mark the cells of a 4-tuple. Each cell is in one of P possible states $z[a_{\nu j}] \in \{0, \ldots, P-1\}$ (the value of the digits in our example), depending on the cell's occupation. Following our earlier work [8] we use the (P=4)-encoding

0=empty and not reachable, 1=Yellow, 2=Red, 3=empty and reachable.

By *reachable* we mean an empty cell that can be occupied in the next move. The reason behind this is that it makes a difference whether e. g. three yellow pieces in a row have a reachable empty cell adjacent to them (a direct threat for Red) or a non-reachable cell (indirect threat).

An n-tuple of length n thus has P^n possible states $k_\nu \in \{0, \ldots, P^n - 1\}$ with

$$k_\nu = \sum_{j=0}^{n-1} s_t[a_{\nu j}] P^j. \tag{1}$$

Here, $s_t[a_{\nu j}]$ is the state of board cell $a_{\nu j}$ at time t. Fig. 1 shows an example board position with a 4-tuple in the numbered cells. The state of the 4-tuple is $k = 3 \cdot 4^0 + 2 \cdot 4^1 + 1 \cdot 4^2 + 1 \cdot 4^3 = 91$.

The number k_ν for the state of T_ν can be used as an index into an associated look-up table LUT_ν, whose parameters are called $w_{\nu,t}[k_\nu]$. Equivalently and more similar to standard neural networks,

1. A ply is a single move of one player, Yellow or Red

we can put all weights into one big weight vector $\vec{w}_t$ with elements $w_{i,t}$, index $i = k_\nu m + \nu$, and define a binary input vector

$$x_i[s_t] = \begin{cases} 1 & \text{if} \quad i = k_\nu m + \nu \\ 0 & \text{else} \end{cases} \tag{2}$$

for the k_ν defined in Eq. (1). For a given board position s_t at time t, the output of the n-tuple network with m n-tuples $T_\nu,\ \nu = 1, \ldots, m$ can be calculated as:

$$f(\vec{w}_t, s_t) = \sum_{i=1}^{m \cdot P^n} w_{i,t} x_i[s_t] \tag{3}$$

Vector $\vec{w}_t$ is a function of time t since it will be modified by the TD learning algorithm (Sec. 3.3). The vector $\vec{w}_t$ combines all weights from all LUTs. It can be a rather big vector, containing, e. g., 9 million weights in our standard Connect-4 implementation with 70 8-tuples. It turns out that only 600 000 - 700 000 of these weights are active during learning (the others represent non-realizable states [8]), but this is still much bigger than the 20 or 25 weights used by Sutton [1] or Beal and Smith [2].

Further details on n-tuples in Connect-4 (symmetry, n-tuple creation) are found in [8]. For all experiments reported in the following, we use the same n-tuple network which was once created by a random-walk selection process.

3.3 TDL

The goal of a game-playing agent is to predict the ideal *value function*, which returns 1.0 if the board position is a win for Yellow, and -1.0 if it is a win for Red. The TDL algorithm aims at *learning* this value function. It does so by setting up an (initially inexperienced) agent, who plays a sequence of games against itself. It learns from the environment, which gives a reward $R \in \{-1.0, 0.0, 1.0\}$ for { Red-win, Draw, Yellow-win } only at the end of each game . The main ingredient is the *temporal difference* (TD) error signal according to Sutton [6]

$$\delta_t = R(s_{t+1}) + \gamma V(\vec{w}_t, s_{t+1}) - V(\vec{w}_t, s_t). \tag{4}$$

Here, $V(\vec{w}_t, s_t) = \sigma\left(f(\vec{w}_t, s_t)\right)$ is the agent's current approximation of the value function[2] on the basis of Eq. (3) and a nonlinear sigmoid function σ (we choose $\sigma = \tanh$). The state s_{t+1} is the best successor of state s_t. In our experiments we use $\gamma = 1$ throughout.

The weights are trained with the usual δ-rule

$$\begin{aligned} w_{i,t+1} &= w_{i,t} + \alpha \delta_t \nabla_{w_i} V(\vec{w}_t, s_t) \\ &= w_{i,t} + \alpha \left(1 - V^2(\vec{w}_t, s_t)\right) \delta_t x_i, \end{aligned} \tag{5}$$

which aims at making the current prediction match the successor prediction more closely. The complete TDL algorithm for games, including the action selection mechanism, is shown in Tab. 1.[3] This is a control algorithm, since it includes action selection (thus policy changes) and learning of the (correspondingly changing) value function.[4] More details on TDL in games can be found in [33] and references therein.[5]

2. $V(\vec{w}_t, s_{t+1})$ is set to 0 if s_{t+1} is a final state.

3. Why is the weight update in Step 17 only done in case of a non-explorative move ($q \geq \epsilon$)? – If an exploratory action (random move) is taken, it is very likely that the final reward does not reflect the true potential of the current state before the random move. E.g. if the current state is a win for Yellow, a random move is likely to turn it into a situation where Yellow looses.

4. Note that the learning of the value of state-action pairs, $Q(s_t, a_t)$, as it occurs in Q-learning or Sarsa control algorithms, can be replaced here by the value of the after state, $V(\vec{w}_t, s_{t+1})$, since in a board game all state-action pairs resulting in the same after state have the same value.

5. We note in passing that we use TDL without eligibility traces in this paper. While this article was under review, we published new research combining Connect-4, TDL, and TCL-EXP with eligibility traces [34].

TABLE 1
TDL algorithm for board games. Prior to the first game, the weight vector $\vec{w}_0$ is initialized with random values. Then the following algorithm is executed for each complete board game. During self-play, the player p switches between $+1$ (Yellow) and -1 (Red).

```
 1: Set the initial state $s_0$ (usually the empty board) and $p = 1$.
 2: Use partially trained weights $\vec{w}_0$ from previous games.
 3: **function** TDLTRAIN($s_0$, $\vec{w}_0$)
 4:     $\vec{e}_0 \leftarrow \nabla_{\vec{w}} V(\vec{w}_0, \vec{x}(s_0))$
 5:     **for** $\big(t \leftarrow 0$ ; $s_t \notin S_{Final}$ ; $t \leftarrow t+1, p \leftarrow (-p)\big)$ **do**
 6:         $V_{old} \leftarrow V(\vec{w}_t, \vec{x}(s_t))$
 7:         Draw random $q \in [0,1]$
 8:         **if** $(q < \epsilon)$ **then**                                   ▷ Explorative move
 9:             Randomly select $s_{t+1}$
10:         **else**                                                           ▷ Greedy move
11:             Select after-state $s_{t+1}$, which maximizes
12:                 $p \cdot \begin{cases} R(s_{t+1}), & \text{if } s_{t+1} \in S_{Final} \\ V(\vec{w}_t, \vec{x}(s_{t+1})), & \text{otherwise} \end{cases}$
13:         **end if**
14:         $V_{new} \leftarrow V(\vec{w}_t, \vec{x}(s_{t+1}))$
15:         $\delta_t \leftarrow R(s_{t+1}) + \gamma V_{new} - V_{old}$               ▷ TD error-signal
16:         **if** $(q \geq \epsilon$ **or** $s_{t+1} \in S_{Final})$ **then**
17:             $\vec{w}_{t+1} \leftarrow \vec{w}_t + \alpha \delta_t \vec{e}_t$              ▷ Weight-update
18:         **end if**
19:         $\vec{e}_{t+1} \leftarrow \nabla_{\vec{w}} V(\vec{w}_{t+1}, \vec{x}(s_{t+1}))$      ▷ Recompute (new $\vec{w}$!)
20:     **end for**
21: **end function**
```

Eq. (4) shows clearly why TDL imposes a nonstationary learning task: $V(\vec{w}_t, s_{t+1})$, the value for the best successor of $V(\vec{w}_t, s_t)$, is the target for $V(\vec{w}_t, s_t)$. But for most board positions (with the exception of terminal states) this target again has to be learnt, so it will probably be at the wrong value initially. Later in training this target value changes to the correct one and the weights need to be readjusted.

3.4 TCL

The TCL algorithm developed by Beal and Smith [2], [3] is an extension of TDL. It has an adjustable learning rate α_i for every weight w_i and a global constant α_{init}. The effective learning rate for each weight is $\alpha_{init}\alpha_i$. The main idea is pretty simple: For each weight two counters N_i and A_i accumulate the sum of weight changes and sum of absolute weight changes. If all weight changes have the same sign, then $|N_i|/A_i = 1$ and the learning rate stays at its upper bound. If weight changes have alternating signs, then $|N_i|/A_i \to 0$ for $t \to \infty$, and the learning rate will be largely reduced for this weight.

Tab. 2 shows the complete TCL algorithm. This algorithm is embedded in the game-playing TDL framework of Sec. 3.3. Steps 2.-6. are executed in each pass through the main TDL-loop instead of Step 17 in Tab. 1. With the help of $\alpha_i \in [0,1]$ the individual learning rate for a weight can be made smaller than the global α_{init}. This is the standard TCL formulation named *TCL[r]* in the following. We tested also a variant *TCL[δ]* where we omit the nonlinear sigmoid function for $r_{i,t}$ in Eq. (7), i. e. we use

$$r_{i,t} = \begin{cases} \delta_t & \text{if} \quad x_i = 1 \\ 0 & \text{if} \quad x_i = 0. \end{cases} \tag{8}$$

In both cases, the operational order is important, as already stated in [2]: first weight update

TABLE 2
TCL in pseudo code

```
1: Initialize: $N_i = A_i = 0$ for all weight indices $i$ and set the constant parameter $\alpha_{init}$.
2: for (every weight index $i$ ) do
3:     Set $\alpha_i = \begin{cases} 1 & \text{if} \quad A_i = 0 \\ g(|N_i|/A_i) & \text{if} \quad A_i > 0 \end{cases}$
4:     Replace TD-weight update Eq. (5) with
           $w_{i,t+1} = w_{i,t} + \alpha_{init}\alpha_i r_{i,t}$        (6)
       where $r_{i,t} = \delta_t \nabla_{w_i} V(\vec{w}_t, s_t)$ is the *recommended* weight change.
5:     Update the counters:
           $N_i \leftarrow N_i + r_{i,t}$        (7)
           $A_i \leftarrow A_i + |r_{i,t}|$
6: end for
```

Standard TCL uses for the transfer function g in Step 3 the identity function, while TCL-EXP uses $g(x)$ according to Eq. (9), see also Fig. 2.

TABLE 3
IDBD in pseudo code

```
1: Initialize: $h_i = 0,\ \beta_i = \beta_{init}$ for all weight indices $i$ and set $\theta$, the meta-learning rate.
2: for (every weight index $i$ ) do
3:     Set input $x_i$ according to Eq. (2)
4:     Set $\beta_i \leftarrow \beta_i + \theta\delta_t x_i h_i$
5:     Set $\alpha_i \leftarrow e^{\beta_i}$
6:     Replace TD-weight update Eq. (5) with
           $w_{i,t+1} = w_{i,t} + \alpha_i \delta_t x_i$
7:     Set $h_i \leftarrow h_i[1 - \alpha_i x_i^2]^+ + \alpha_i \delta_t x_i$
8: end for
```

with $[d]^+ = d$ if $d > 0$, 0 else.

using the previous values of N_i, A_i, then counter update.[6]

3.5 IDBD

Sutton's IDBD algorithm [1] introduces – similarly to TCL – an individual learning rate $\alpha_i = e^{\beta_i}$ for every weight w_i. The algorithm is shown in Tab. 3. It is again embedded in the game-playing TDL framework of Sec. 3.3. Steps 2.-8. are executed in each pass through the main TDL-loop instead of Step 17 in Tab. 1.

6. The original TCL implementation [2] has the further option that the $r_{i,t}$ may be accumulated over a sequence of steps before a real weight update (Eq. (6)) takes place. This balances a tradeoff between faster changing learning rates (short sequences) and accumulation of statistic evidence (long sequences). We tested this sequence option for our Connect-4-task as well, but found the immediate update (as in the formulas above, 1-step sequence) to be better.

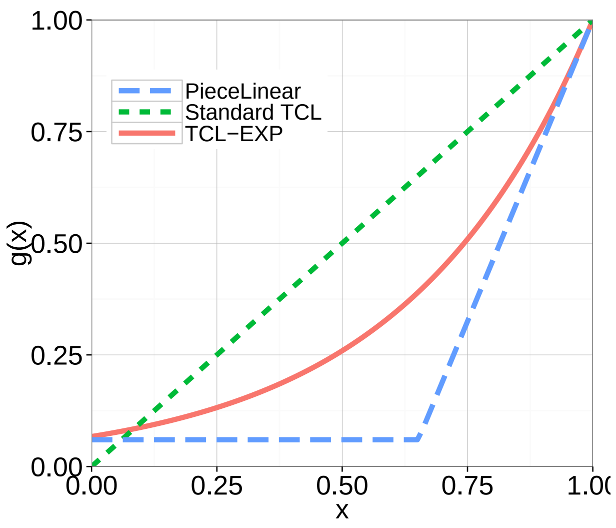


Fig. 2. Options for the transfer function $g(x)$. Standard TCL uses the identity function. TCL-EXP (Eq. (9)) is shown for $\beta = 2.7$ and PieceLinear is a piecewise linear function with the same endpoints as TCL-EXP and same slope at $x = 1$.

The main idea behind this algorithm is simple: The memory term h_i is a decaying trace of past weight changes. The increment in β_i is proportional to the product of the current weight change $\delta_t x_i$ and past weight changes h_i. Accumulated increments correspond to the *correlation* between current and recent weight changes [1]. In case of a positive correlation, the learning rate can be larger, while negative correlation indicates overshooting weight increments where the learning rate should be reduced.

Note that IDBD in its current form is only formulated for linear networks. Consequently we omit the nonlinear sigmoid function σ in Eq. (4) and in the weight update rule (Step 6 of Tab. 3) for IDBD.

3.6 Modified TCL-EXP

Our modification TCL-EXP brings a new element from IDBD into the standard TCL algorithm. Instead of the identity transfer function $g(x) = x$ we use an exponential function

$$g(x) = e^{\beta(x-1)} \tag{9}$$

in Step 3 of TCL (see Tab. 2 and Fig. 2). As pointed out by Sutton [1], an exponential function has the nice property that a fixed step-size change in x will change $g(x)$ by a fixed fraction of its current value, i.e. it allows for geometric steps. *"This is desirable because some α_i must become very small while others remain large; no fixed step-size would work well for all the α_i."* [1].

3.7 Other algorithms

The following other algorithms were used in our case study for comparison: Koop's IDBD-nl [15], [16], Sutton's K1 [11], Schraudolph's ELK1 [13], Mahmood's Autostep [17], and Dabney's α-bounds [19]. All algorithms are implemented exactly as described in their original papers (if not stated otherwise) and then applied to our Connect-4 task.

Some algorithms (IDBD, K1, Autostep) are only derived for linear units. We omit in this case the nonlinear sigmoid function $\sigma = \tanh$ for the TDL value function (Sec. 3.3). For all other algorithms we use this sigmoid function. An exception is Koop's IDBD-nl [15], [16] being derived for the logistic sigmoid function, which we use in this case instead of $\tanh$.

TABLE 4
Settings for all experiments shown in Fig. 3 – 7. Column *TCL* denotes, whether we used for TCL the recommended-weight update [r] or the δ-update [δ]. For all algorithms specifying β_{init} the relation $\alpha_{init} = e^{\beta_{init}}$ holds.

Algorithm	Fig.	α_{init}	α_{final}	ϵ_{init}	ϵ_{final}	$\epsilon_{IP}/10^6$	β	β_{init}	θ	*TCL*
Former TDL	3, 6, 7	0.004	0.002	0.6	0.1	1.0	–	–	–	–
Tuned TDL	3 – 7	0.004	0.002	0.1	0.1	–	–	–	–	–
TCL [δ]	4 – 7	0.04	–	0.1	0.1	–	–	–	–	[δ]
TCL [r]	4 – 7	0.04	–	0.1	0.1	–	–	–	–	[r]
TCL-EXP	4 – 7	0.05	–	0.1	0.1	–	2.7	–	–	[r]
IDBD	4 – 7	0.0067	–	0.1	0.1	–	–	-5.0	3.0	–
IDBD-nl	4 – 7	0.0302	–	0.1	0.1	–	–	-3.5	3.1	–
Autostep	4 – 7	0.0050	–	0.1	0.1	–	–	-5.3	10^{-4}	–
K1	4 – 7	0.0090	–	0.1	0.1	–	–	-4.7	6.0	–
ELK1	4 – 7	0.0111	–	0.1	0.1	–	–	-4.5	5.0	–
α-bounds	4 – 7	1.0	–	0.1	0.1	–	–	–	–	–

4 Results

4.1 Experimental Setup

The training of our TDL-n-tuple-agents is performed without any access to other agents. It is done without any other external information than the outcome of each game. Each agent is initialized with random weights uniformly drawn from $[-\chi/2, \chi/2]$ with $\chi = 0.001$. The agent plays a large number of games (10 millions) against itself as described in Sec. 3.3 receiving no other information from the environment than win, loss, or draw at the end of each game. Training is performed with TDL, optionally augmented by IDBD- or TCL-ingredients.[7] To explore the state space of the game, the agent chooses with a small probability ϵ the next move at random. During training, ϵ varies like a sigmoidal function ($\tanh$) between ϵ_{init} and ϵ_{final} with inflection point at game number ϵ_{IP}. Every 10 000 or 100 000 games the agent strength is measured by the procedure described in Sec. 4.2. We repeat the whole training run 20 times (if not stated otherwise) in order to get statistically sound results.

In TDL, the global learning rate α decays exponentially from α_{init} to α_{final}. TCL instead keeps the global parameter α_{init} at a constant value, but each weight has its individual learning rate α_i. In TCL-EXP we have the additional global parameter β, while for IDBD the relevant parameters are θ and β_{init}. The precise parameter values to reproduce each of our results are given in Tab. 4.

4.2 Agent Evaluation

A fair agent evaluation for board games is not trivial. There is no closed-form objective function for 'agent playing strength' since the evaluation of all board positions is infeasible and it is not clear, which relevance has to be assigned to each position. The most common approach is to assess an agent's strength by observing its *interactions* with other agents, either in a tournament

7. Our software framework in Java used for all experiments is avaiable as open source from GitHub (https://github.com/MarkusThill/Connect-Four).

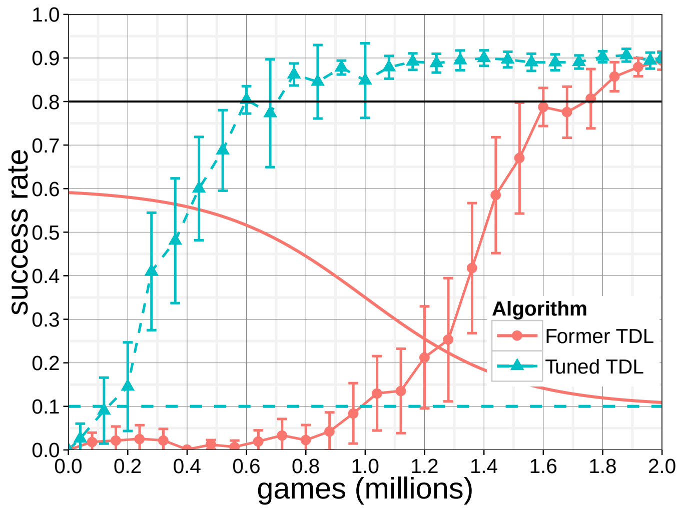


Fig. 3. *Former TDL* [8] versus *Tuned TDL*. The parameter settings are in Tab. 4. The lines without points show the exploration rate ϵ.

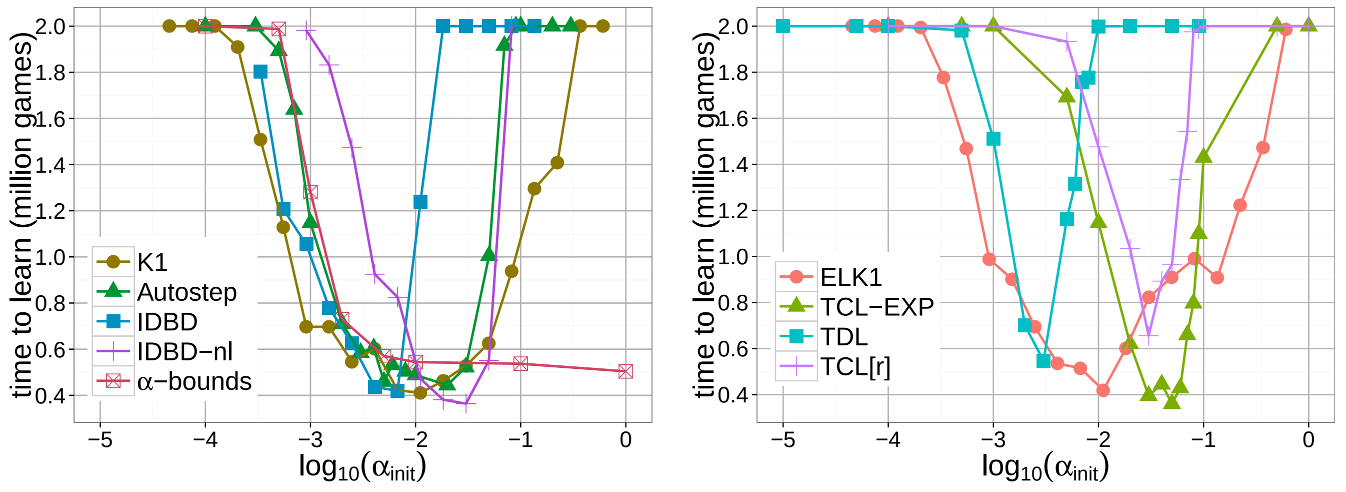


Fig. 4. Sensitivity on the initial learning rate α_{init} for all algorithms. For algorithms specifying β_{init} (like IDBD), we use the transformation $\alpha_{init} = e^{\beta_{init}}$. Parameter α_{init} is screened over a large range. Each point is the mean of 10 runs with 2 million training games. For each run we measure the success rate every 10 000 games and smooth this curve to dampen fluctuations. 'time to learn' is the number of games until this smoothed curve crosses the 80%-line for the first time. If a run never crosses this line, 'time to learn' is set to 2 millions.

or against a referee agent. We choose the Minimax agent to be the ultimate reference. Note that all the approaches to Connect-4 found in the literature (cf. Sec. 1) fail to provide a common reference point for the strength of the agents generated.

Our approach to agent evaluation in Connect-4 is as follows: TDL[8] (Yellow) and Minimax (Red) play a tournament of 50 games. (Since Minimax will always win playing Yellow, we consider only games with TDL playing Yellow.) The ideal TDL agent is expected to win every game. If both agents act fully deterministically, each game would be identical. We introduce a source of randomness without sacrificing any agent's strength as follows: If Minimax has several optimal

8. or any other of our trainable agents

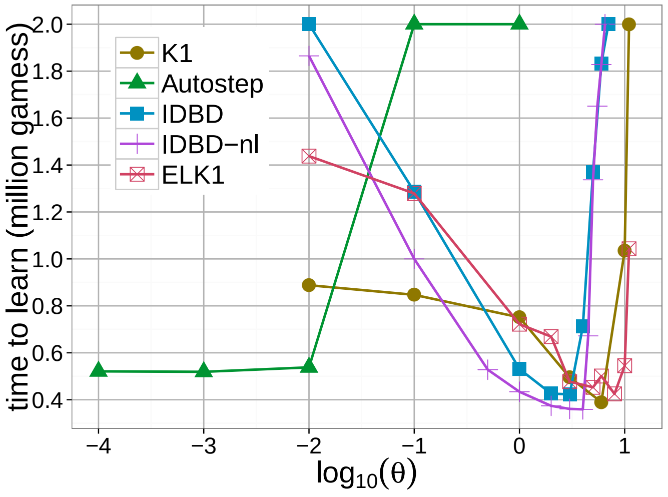


Fig. 5. Sensitivity on the meta step size parameter θ for algorithms having this parameter (μ in the case of Autostep). A large range of θ-values is screened. Other settings as in Fig. 4.

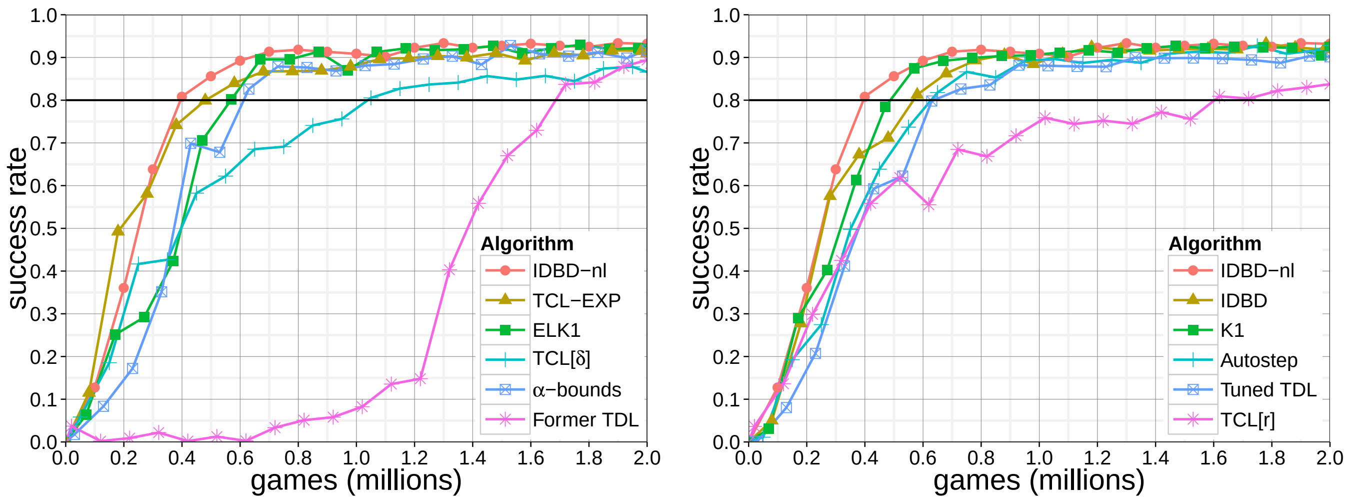


Fig. 6. Final comparison of different algorithms for Connect-4. Every setting is repeated 20 times with different random initialization and random exploration. The agent strength was measured every 10 000 games and each point is the mean of 20 runs. For better display of the curves, error bars are omitted and only a subset of all points is shown in the plots. The sampled points are connected by straight lines. Note that due to this procedure the visual crossing of the 80% line may differ somewhat from the exact numbers given in the second column of Table 5.

moves at its disposal, it chooses one randomly. For positions in which all possible moves lead to a defeat of Minimax, the agent will select that move which delays the defeat as far as possible to the future. This increases the difficulty for the TDL-agent which has to prove that it can play perfect during a longer game. The TDL agent gets a score of 1 for a win, 0.5 for a draw and 0 for a loss. The overall success rate $S \in [0.0, 1.0]$ is the mean of the 50 individual scores. A perfect TDL agent receives a success rate of 1.0.

There are two criteria for a good agent: (a) final strength, measured as the asymptotic success rate after 2 or 10 million games and (b) speed of learning, measured as the number of games needed to reach a sufficient good success rate, e. g. 80%.

4.3 Results TDL

It was shown in [8] that n-tuple systems combined with TDL deliver strong Connect-4 agents. However, 1 565 000 training games were needed to cross the 80%-success-rate. We made a more systematic parameter tuning (latin hypercube sampling) and found that the tuned TDL-agent reaches the 80%-success-rate after 670 000 games (Fig. 3), which is faster by more than a factor of 2. This tuning result emerged as the optimal result from testing about 60 different parameter configurations. The key difference to the former TDL result is a suitably reduced exploration rate ϵ.

4.4 Sensitivity for all algorithms

All adaptive learning rate algorithms have a parameter α_{init} (or equivalently $\beta_{init} = \ln(\alpha_{init})$) and some of them have a meta step size parameter θ. For each algorithm we estimated the best pair (α_{init}, θ) by grid search. Then we assessed the sensitivity by fixing one parameter at its best value and varying the other over a broad range. The results are depicted in Fig. 4 for α_{init} and in Fig. 5 for θ.

It is clearly seen that the linear step size algorithm (TDL, TCL[r]) have only a narrow range of good α_{init} values (Fig. 4). TCL-EXP instead has a broader range. In Fig. 5 we see that Autostep performs well in a broad range of remarkably small θ values. Its α_{init}-sensitivity is however similar to the other algorithms (Fig. 4).

The results for asymptotic success rates (not shown in the figures) have somewhat broader ranges for most algorithms, but show qualitatively the same picture.

4.5 Results TCL

Initially, the results with TCL were not better or even worse than TDL. This came as a surprise, since Beal & Smith stated, that TCL would automatically self-adjust the parameters: *"The parameter α_{init} can be set high initially, and the α_i then provide automatic adjustment during the learning process."* [2]. In Fig. 4 we vary the parameter α_{init} over a broad range. We found TCL[r] *not* to have a larger area of high success rates than TDL, it is only shifted to larger values.

For the best value $\alpha_{init} = 0.04$, we got the result shown as line *TCL[r]* in Fig. 6: The agent reaches the same strength as the tuned TDL agent asymptotically, but much slower. [9] – In any case, standard TCL cannot fulfill the promise of faster learning for this complex Connect-4 task.

4.6 Results IDBD

Fig. 6 allows to compare IDBD and TDL on the Connect-4 task: IDBD is comparable to TDL, slightly better. There is some dependence on the right choice of parameters β_{init} and θ as Fig. 4 and 5 show. It has to be mentioned, however, that IDBD runs TDL without the sigmoid function, which might counteract any improvements through the individual learning rates. To make a fair comparison we include also IDBD-nl, which has a logistic sigmoid function, and this turns out to be the best algorithm in terms of learning speed.

9. It may come as a surprise that TCL[r] crosses the 80% line only at 1 600 000 games while Tab. 5 has 890 000 games instead. The reason is that we show in Fig. 6 the mean of 20 runs while Tab. 5 shows the median. Since two of the TCL[r] runs never reach the 80% line and thus are set to 2 million games, the mean is considerably higher.

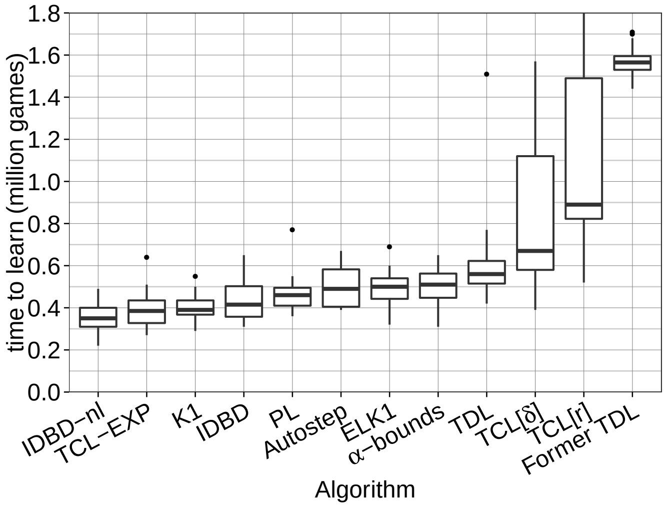


Fig. 7. Learning speed for different algorithms. The boxplots show the results from 20 runs. 'time to learn' is defined as in Fig. 4. TCL[r]: the original TCL as in [2] using recommended weight changes, TCL[δ]: using the δ-signal instead. PL: Piecewise linear, Former TDL from [8], TDL: Tuned TDL. There are one and two runs in TCL[δ] and TCL[r], resp., which never reach the 80% success rate. 'time to learn' is set to 2 million games for these runs (not shown in the figure).

4.7 Results TCL-EXP

TCL-EXP is the standard TCL algorithm with only one detail changed: the exponential transfer function for the learning rate, see Eq. (9). This brings a remarkable increase in speed of learning for the game Connect-4, as our results in Fig. 6 show: TCL-EXP reaches the 80% success rate after about 385 000 games instead of 560 000 (Tuned TDL). At the same time it reaches asymptotically a very good success rate (Table 5).

We varied the parameter β systematically between 1 and 7 and found values in the range $\beta \in [2, 3]$ to be optimal.

4.8 Overall Results

Tab. 5 and Fig. 7 measure the learning time with respect to two metrics: 'games to train' and computation time. IDBD-nl and TCL-EXP are on the first ranks for both metrics. The boxplot in Fig. 7 indicates that TCL-EXP has a much smaller variance than TCL[r] or TCL[δ].

Tab. 5 shows in addition the asymptotic success rate (ASR) after 2 and 10 million games. IDBD-nl and TCL-EXP are on the first ranks for this metric as well, but the others are not very far off. It has to be noted that a few runs for IDBD and IDBD-nl had a breakdown after more than 5 million games. ASR[10 millions] is set to 0 in these cases. The median in Tab. 5 is not influenced by these outliers. We assume that the large value of θ (being beneficial for fast learning) is responsible for the breakdown.

5 Discussion

5.1 Survey of algorithms

We conducted in this case study a comparison of several online step size adaptation algorithms on TDL for the task Connect-4 as compared to plain TDL without step size adaptation. Some general consequences can be deduced from Fig. 6 and 7:

Algorithms with geometric step size changes and individual learning rates (IDBD, IDBD-nl, K1, ELK1, TCL-EXP, Autostep) are superior to algorithms with one learning rate (TDL, α-bounds)

TABLE 5
Training times for the algorithms presented in this work. The median of 'time to learn' (Fig. 7) is given in the second column. The third column depicts the computation time in minutes (including evaluation every 10 000 games). Time is measured on a standard PC (single core of an Intel Core i7-3632QM, 2.20 GHz, 8 GB RAM). The fourth and fifth column show the asymptotic success rate ASR (median of 20 runs), after 2 and 10 million games, resp.

	time to learn		ASR	
			2	10
Algorithm	[games]	[min]	[%]	[%]
Former TDL	1 565 000	102.0	91.7	91.6
TCL[r]	890 000	67.2	83.9	90.4
TCL[δ]	670 000	50.3	89.4	91.4
Tuned TDL	560 000	39.7	90.7	92.5
α-bounds	560 000	39.2	90.8	92.4
ELK1	500 000	39.5	92.5	92.8
Autostep	490 000	38.7	91.5	92.6
PL	460 000	36.3	90.4	90.4
IDBD	415 000	33.1	92.5	92.7
K1	390 000	30.8	92.0	92.7
TCL-EXP	385 000	30.1	92.5	93.1
IDBD-nl	350 000	27.7	92.8	93.5

and both groups are superior to step size adaptation algorithms with linear step size changes (TCL[δ], TCL[r]).

The nonlinear algorithms IDBD-nl and TCL-EXP perform best, but the other algorithms with geometric step size change are not far away. Surprisingly, some purely linear algorithms (K1 and IDBD) perform nearly as good as their nonlinear variants (ELK1 and IDBD-nl)

Autostep, as expected, performs well for a large range of the meta step size parameter $\theta \in [10^{-6}, 10^{-2}]$ (Fig. 5). However, it has a sensitivity to the parameter α_{init} which is comparable to the other algorithms.

Dabney and Barto's α-bounds algorithm [19] performs poorly in its original linear version (6 out of 20 runs never pass the 80%-line and 'time to learn' has a median of 1.9 million games, not shown in the figures). A purely linear TDL would perform similarly. We extended α-bounds to the nonlinear case by making a linear expansion of the sigmoid function and deriving a slightly modified rule

$$\alpha_t = min\left(\alpha_{t-1}, |\sigma'(f)\,\vec{e}_t \cdot (\gamma\vec{x}_{t+1} - \vec{x}_t|^{-1}\right)$$

which has an additional $\sigma'(f)$ as compared to Eq. (15) in [19]. With this version and the nonlinearity $\sigma = tanh$ the results for α-bounds are slightly better than TDL (Fig. 7) but inferior to the algorithms with individual learning rates for each weight. It would be interesting to extend α-bounds to the individual learning rate case, as Dabney and Barto already suggested in their conclusion [19].

As Fig. 4 shows, α-bounds has a remarkable stability with respect to large α_{init}. It will however show a breakdown for $\alpha_{init} < 10^{-3}$.

5.2 TCL algorithms

When we started the TCL experiments, we expected that the individual and adaptable learning rates for each weight would free the user from tuning these rates. In particular, we expected that a too large α_{init} would easily be corrected by the individual α_i during the initial training phase, as pointed out by Beal & Smith [2]. After that, the training should proceed equivalently to a setting where a smaller α_{init} had been chosen directly. In contrast to this expectation, we observe for TCL[r] and all $\alpha_{init} \neq 0.04$ a rapid decrease in performance (Fig. 4). Why is this the case?

An inspection of the n-tuple network showed that for $\alpha_{init} \neq 0.04$, shortly after the initial phase, the responses to most board positions are driven into saturation of the sigmoid function. If α_{init} is too large, subsequent learning steps fall with high probability in regions of the sigmoid function with different slopes. Then, alternating weight changes will not cancel because the gradient of the sigmoid function differs. Thus N_i/A_i will not approach 0 (although it should for a too large α_{init}). If finally the network response is in saturation, learning virtually stops (due to $1 - \tanh^2(f) \approx 0$ in the gradient).

A too small $\alpha_{init} < 0.04$ leads to a decreasing TCL-performance as well. This is partly understandable since TCL can only reduce but not increase the global α_{init}. If we start with the optimal $\alpha_{init} = 0.04$, we reach with TCL[r] a good asymptotic success rate and time to learn, but the learning speed is slower than for *Tuned TDL* (Fig. 6 and 7). The reason for this is not yet fully understood. It might be due to the learning rate change proportional to N_i/A_i being suboptimal.

This is supported by our finding that *TCL-EXP* with 'geometric' step sizes is a faster learning agent than *Tuned TDL* (Fig. 6). The reason for this was explained in Sec. 3.4. To test the importance of geometric step sizes, we did another experiment: We replaced the *TCL-EXP* transfer function by a piecewise linear function (*PL*) as shown in Fig. 2 having the same endpoints and same slope at $x = 1$. The results for *PL* in Fig. 7 and Tab. 5 are worse than *TCL-EXP*. Therefore, it is not the slope at $x = 1$ but the geometric step size which is important for success.

Given the above results and discussion, it is quite natural that we would choose on a new problem among all investigated online adaptive step-size algorithms in first place the algorithm IDBD-nl, however, closely followed by TCL-EXP as a nearly equivalent choice. IDBD-nl is an algorithm with striking simplicity since the choice of nonlinearity (logistic function) leads to a simple gradient.

6 Conclusion

We investigated a complex learning task for the Connect-4 board game. It is remarkable that an agent can learn this complex game solely from self-play. Our previous work [8] has shown that a large number of features (in our case: more than half a million of n-tuple states) is a necessary prerequisite to learn such a task. The agent in [8] was slow in learning, so we studied several alternatives, namely tuning and online adaptation of learning rates (IDBD and TCL). It was hoped that IDBD and TCL with their self-adaptation capabilities could make tuning unnecessary.

Initially, we could improve our previous result [8] by tuning the exploration rate (see Sec. 4.3). This increased the learning speed by a factor of 2.

Our research question 1) from Sec. 1 was answered positively: We demonstrated that the learning algorithms IDBD, TCL, and others work for such big learning tasks with more than half a million of weights. (To the best of our knowledge, this has not been tested before.)

Research question 2) has a negative answer: We found that IDBD and TCL do *not* free the user from parameter tuning. If the meta-parameters (α_{init} and β in case of TCL; β_{init} and θ in case of IDBD) are not set to appropriate values, results are worse than with tuned TDL. The other algorithms show a similar behavior. Even Autostep, which is quite tuning-free with respect to the meta step size parameter θ, requires tuning for its parameter α_{init}.

Research question 3) received a positive answer again: Online learning rate adaptation schemes with geometric step size and individual learning rates are significantly faster than pure TDL. Our modified variant TCL-EXP is among the fastest algorithms, but not significantly faster than other algorithms in the same group. The fastest algorithms (IDBD-nl and TCL-EXP) incorporate nonlinear learning units. So we are led to the conclusion that geometric step sizes plus nonlinear units are important ingredients for fast learning. Compared to the earlier published, non-tuned TDL agent [8] (1 565 000 games to reach 80% success), these algorithms exhibit a large improvement by a factor of 4.

Thus, the route to self-adapting agents for game learning looks promising: Some of these agents learn faster than those with fixed learning rates. At the same time, it is our impression that more research is needed to better understand the interaction between self-adaptive learning rates and nonlinear output functions.

In our ongoing research we plan to investigate the role of different nonlinear output functions for online learning rate adapatation algorithms and whether it is possible to augment those learning algorithms with eligibility traces. Both aspects could increase the robustness and speed of learning.

Acknowledgments

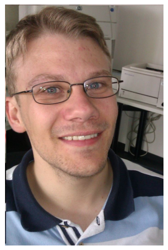
The authors would like to thank the anonymous reviewers for their helpful comments and for pointing out important adaptive learning references. They would like to thank Boris Naujoks for critical review of an earlier draft. This work has been partially supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grant SOMA (2009 - 2013).

References

[1] R. S. Sutton, "Adapting bias by gradient descent: An incremental version of delta-bar-delta." in *AAAI Conf. on Artificial Intelligence*, W. R. Swartout, Ed. AAAI Press, 1992, pp. 171–176.
[2] D. F. Beal and M. C. Smith, "Temporal coherence and prediction decay in TD learning," in *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, T. Dean, Ed. Morgan Kaufmann, 1999, pp. 564–569.
[3] ——, "Temporal difference learning for heuristic search and game playing," *Information Sciences*, vol. 122, no. 1, pp. 3–21, 2000.
[4] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.
[5] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, University of Massachusetts, Amherst, MA, 1984.
[6] ——, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.
[7] G. Tesauro, "TD-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, pp. 215–219, 1994.
[8] M. Thill, P. Koch, and W. Konen, "Reinforcement learning with n-tuples on the game Connect-4," in *PPSN'2012: Parallel Problem Solving From Nature*, C. Coello Coello, Ed. Springer, Heidlberg, 2012, pp. 184–194.
[9] S. M. Lucas, "Learning to play Othello with n-tuple systems," *Australian Journal of Intelligent Information Processing*, vol. 4, pp. 1–20, 2008.
[10] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural networks*, vol. 1, no. 4, pp. 295–307, 1988.
[11] R. S. Sutton, "Gain adaptation beats least squares," in *Proc. Yale Workshop on Adaptive and Learning Systems*, 1992, pp. 161–166.
[12] L. Almeida, T. Langlois, and J. D. Amaral, "On-line step size adaptation," INESC, 1000, Lisboa, Portugal, Tech. Rep. RT07/97, 1997.
[13] N. N. Schraudolph, "Online learning with adaptive local step sizes," in *Neural Nets WIRN Vietri-99*. Springer, 1999, pp. 151–156.
[14] C. Li, Y. Ye, Q. Miao, and H.-L. Shen, "KIMEL: A kernel incremental metalearning algorithm," *Signal Processing*, vol. 93, no. 6, pp. 1586 – 1596, 2013.
[15] R. S. Sutton, A. Koop, and D. Silver, "On the role of tracking in stationary environments," in *24th Int. Conf. on Machine Learning*. ACM, 2007, pp. 871–878.

[16] A. Koop, *Investigating Experience: Temporal Coherence and Empirical Knowledge Representation*, ser. Canadian theses. Univ. of Alberta (Canada), 2008.
[17] A. Mahmood, "Automatic step-size adaptation in incremental supervised learning," Ph.D. dissertation, University of Alberta, 2010.
[18] A. R. Mahmood, R. S. Sutton, T. Degris, and P. M. Pilarski, "Tuning-free step-size adaptation," in *IEEE InternationalConference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 2121–2124.
[19] W. Dabney and A. G. Barto, "Adaptive step-size for online temporal difference learning." in *26th AAAI Conference on Artificial Intelligence*, 2012.
[20] T. Schaul, S. Zhang, and Y. LeCun, "No More Pesky Learning Rates," in *International Conference on Machine Learning (ICML)*, 2013.
[21] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *IEEE Int. Conf. on Neural Networks*, 1993, pp. 586–591.
[22] J. D. Allen, "A note on the computer solution of connect-four," in *Heuristic Programming in Artificial Intelligence 1: The First Computer Olympiad*, D. Levy and D. Beal, Eds. Ellis Horwood, London, 1989, pp. 134–135.
[23] V. Allis, "A knowledge-based approach of Connect-4. The game is solved: White wins," Master's thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, 1988.
[24] M. Schneider and J. Garcia Rosa, "Neural Connect-4 - a connectionist approach," in *Brazilian Symposium on Neural Networks*, 2002, pp. 236–241.
[25] M. Stenmark, "Synthesizing board evaluation functions for Connect-4 using machine learning techniques," Master's thesis, Østfold University College, Norway, 2005.
[26] D. Curran and C. O'Riordan, "Evolving Connect-4 playing neural networks using cultural learning," National University of Ireland, Galway, NUIG-IT-081204, 2004. [Online]. Available: http://www2.it.nuigalway.ie/publications/TR/abstracts/NUIG-IT-081204.pdf
[27] K. Krawiec and M. G. Szubert, "Learning n-tuple networks for Othello by coevolutionary gradient search," in *GECCO'2011: Genetic and Evolutionary Computation Conference*. ACM, New York, 2011, pp. 355–362.
[28] S. Lucas and T. P. Runarsson, "Othello competition," http://algoval.essex.ac.uk:8080/othello/League.jsp, 2011, last access 18.02.2014.
[29] S. Edelkamp and P. Kissmann, "Symbolic classification of general two-player games," http://www.tzi.de/~edelkamp/publications/conf/ki/EdelkampK08-1.pdf, Technische Universität Dortmund, Tech. Rep., 2008, last access 18.02.2014.
[30] J. Tromp, "Solving Connect-4 on medium board sizes," *ICGA Journal*, vol. 31, no. 2, pp. 110–112, 2008.
[31] M. Thill, "Using n-tuple systems with TD learning for strategic board games (in German)," Cologne University of Applied Science, CIOP Report 01/12, 2012.
[32] W. W. Bledsoe and I. Browning, "Pattern recognition and reading by machine," in *Eastern Joint Computer Conference*, New York, 1959, pp. 225–232.
[33] W. Konen and T. Bartz-Beielstein, "Reinforcement learning: Insights from interesting failures in parameter selection," in *PPSN'2008: Parallel Problem Solving From Nature*, G. Rudolph, Ed. Springer, Berlin, 2008, pp. 478–487.
[34] M. Thill, S. Bagheri, P. Koch, and W. Konen, "Temporal difference learning with eligibility traces for the game connect four," in *IEEE International Conference on Computational Intelligence and Games*. IEEE, 2014, pp. 586–591.

Samineh Bagheri received her B.Sc. degree in electrical engineering specialized in electronics from Shahid Beheshti University, Tehran, Iran in 2011. She is currently master student and research assistant at Cologne University of Applied Sciences in Industrial Automation and IT.
Her research interests are machine learning, evolutionary computation and self adaptive learning strategies for board games or optimization tasks.



Markus Thill studied Computer Engineering at Cologne University of Applied Sciences (CUAS) and received his B.Sc. degree in 2012. Currently he is a master student in Industrial Automation and IT and research assistant at CUAS.
Markus already worked on artifical intelligence in board games for his bachelor thesis, in particular Temporal Difference Learning and tree-based algorithms. With this thesis he won the Opitz Innovation Price 2013.

Patrick Koch received his Diploma in Computer Science from the University of Paderborn, Germany in 2008 and his Ph.D. from the University of Leiden, The Netherlands in 2013. Since 2009 Patrick is a research associate at Cologne University of Applied Sciences, where he works at the research center for Computational Intelligence, Optimization & Data Mining (http://www.gociop.de). Patrick's research concerns efficient tuning in machine learning and the application of evolutionary algorithms for single and multi-criteria problems. In the Computational Intelligence in Games (CIG) area, he is especially interested in finding improved agents for board-games with Temporal Difference Learning and n-tuples.

Wolfgang Konen received his Diploma in physics and his Ph.D. degree in theoretical physics from the University of Mainz, Germany, in 1987 and 1990, resp. He worked in the area of neuroinformatics and computer vision at Ruhr-University Bochum, Germany, and in several companies. He is currently Professor of Computer Science and Mathematics at Cologne University of Applied Sciences, Germany, founding member of the Research Centers Computational Intelligence, Optimization & Data Mining (http://www.gociop.de) and CIplus (http://ciplus-research.de). His research interests include, but are not limited to: understanding how humans and computer learn, the dynamics of learning systems, computational intelligence in games, game physics, data mining, and computer vision.

Kontakt/Impressum

Diese Veröffentlichungen erscheinen im Rahmen der Schriftenreihe ”CIplus”.
Alle Veröffentlichungen dieser Reihe können unter
`www.ciplus-research.de`
oder unter
`http://opus.bsz-bw.de/fhk/index.php?la=de`
abgerufen werden.

Köln, Dezember 2014

Herausgeber / Editorship

Prof. Dr. Thomas Bartz-Beielstein,
Prof. Dr. Wolfgang Konen,
Prof. Dr. Horst Stenzel,
Prof. Dr. Boris Naujoks
Institute of Computer Science,
Faculty of Computer Science and Engineering Science,
Cologne University of Applied Sciences,
Steinmüllerallee 1,
51643 Gummersbach
url: `www.ciplus-research.de`

Schriftleitung und Ansprechpartner/ Contact editor's office

Prof. Dr. Thomas Bartz-Beielstein,
Institute of Computer Science,
Faculty of Computer Science and Engineering Science,
Cologne University of Applied Sciences,
Steinmüllerallee 1, 51643 Gummersbach
phone: +49 2261 8196 6391
url: `http://www.gm.fh-koeln.de/~bartz/`
eMail: thomas.bartz-beielstein@fh-koeln.de

ISSN (online) 2194-2870