# Reinforcement Learning for Strategic Board Games with n-Tuple Systems

Wolfgang Konen, Samineh Bagheri and Markus Thill
TH Köln – University of Applied Sciences

## Connect-4

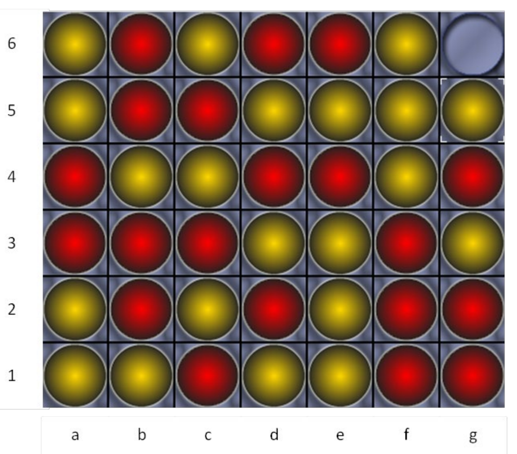


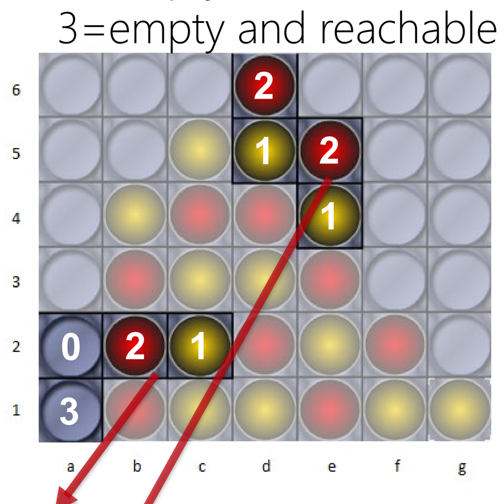
Fig. 1.: Connect-4 Position with a win for Yellow.

- Two-player board game
- Players in turn drop their stones into 1 of 7 columns
- Gravity: Stones fall to lowest empty cell in the column
- Goal: Create a row of four connected stones (horizontally, vertically or diagonally)
- State-Space Complexity: $4.5 \cdot 10^{12}$

## N-Tuple Systems

An n-tuple is a sequence of board cells where each cell can have one out of P=4 states:

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



$T_1 = (a1, a2, b2, c2)$
$\xi_1 = 3 \cdot 4^0 + 0 \cdot 4^1 + 2 \cdot 4^2 + 1 \cdot 4^3 = 99$

$T_2 = (d6, d5, e5, e4)$
$\xi_2 = 2 \cdot 4^0 + 1 \cdot 4^1 + 2 \cdot 4^2 + 1 \cdot 4^3 = 102$

Example:

- Board position $s_t$
- The system with *m* n-tuples specifies a linear function

| LUT$_1$ index | LUT$_1$ weight | LUT$_2$ index | LUT$_2$ weight | … | LUT$_m$ index | LUT$_m$ weight |
|---|---|---|---|---|---|---|
| 0 | $w_0^{(1)}$ | 0 | $w_0^{(2)}$ | | 0 | $w_0^{(m)}$ |
| 1 | $w_1^{(1)}$ | 1 | $w_1^{(2)}$ | | 1 | $w_1^{(m)}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ | ⋮ |
| 99 | $w_{99}^{(1)}$ | 102 | $w_{102}^{(2)}$ | | $\xi_m$ | $w_{\xi_m}^{(m)}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | | ⋮ | ⋮ |
| 255 | $w_{255}^{(1)}$ | 255 | $w_{255}^{(2)}$ | | 255 | $w_{255}^{(m)}$ |

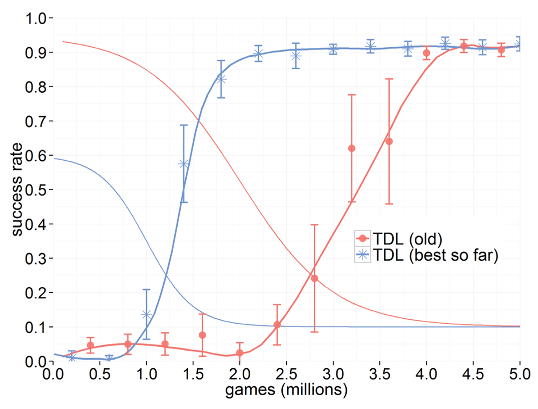
$$V(s_t) = \tanh(w_{99}^{(1)} + w_{102}^{(2)} + \cdots + w_{\xi_m}^{(m)})$$

## Temporal Difference Learning

Goal: learn a value function $V(s)$ by means of Temporal Difference Learning (TDL) as follows:

**Algorithm 1** General TD(0) algorithm.

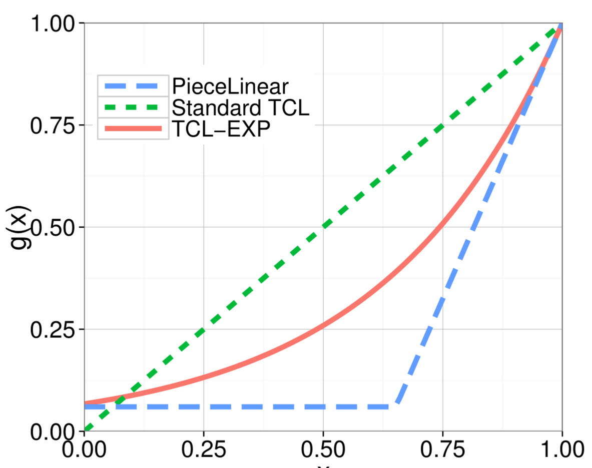
```
Initialize V(s) arbitrarily, π to the policy to be evaluated
for Each episode do
    Initialize s
    for Each step of episode do
        a := action given by π for s
        Take action a; observe reward, and next state (s_{t+1})
        V(s) := V(s) + α[r + γV(s_{t+1}) − V(s)]
        s_t := s_{t+1}
    end for                                         ▷ until s is terminal
end for
```




## Step-Size Adaptation

**Algorithm 2** General TCL algorithm in pseudo code. The counters $A_i$ and $N_i$ are initialized *once* at the beginning of the training. In the original algorithm, the individual learning rates $\alpha_i$ are calculated with the identity transfer function $g(x) = x$, but also other transfer functions may be selected, taking into consideration certain conditions.

```
1: Initialize counters A_i = 0 and N_i = 0  ∀i.
2: Set global learning rate α.
3: for (every weight index i) do
4:     α_i = { g(|N_i|/A_i),  if A_i ≠ 0
             { 1,             otherwise                 ▷ Individual step-size
5:     r_{i,t} = δ_t e_{i,t}                            ▷ recommended weight change
6:     w_i ← w_i + α α_i r_i                            ▷ TD update for each weight
7:     A_i ← A_i + |r_{i,t}|                            ▷ update accumulating counter A
8:     N_i ← N_i + r_{i,t}                              ▷ update accumulating counter N
9: end for
10: with a transfer function g(x),
11: the approximation error δ_t and,
12: the eligibility traces e_{i,t}.
```




The TCL algorithm developed by Beal and Smith is an extension of TDL. It has an adjustable learning rate $\alpha_i$ for every weight $w_i$ and a global constant $\alpha_{init}$. 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 the learning rate stays at its upper bound, otherwise the learning rate will be largely reduced.

## Eligibility Traces

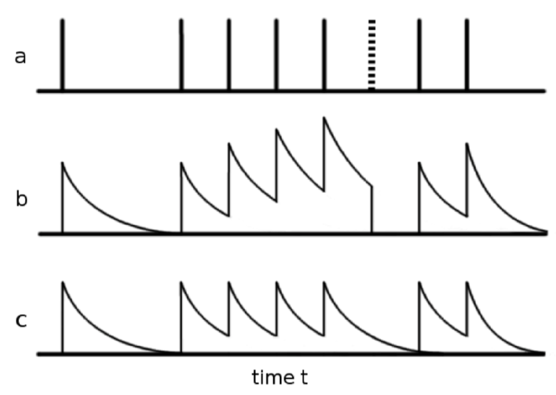



Fig. 3.: Schematic view of different eligibility trace variants.

a) Without eligibility traces: a weight is activated only at isolated time points. Dotted vertical line: random move.
b) Resetting traces: eligibility trace with reset on random move.
c) Replacing traces: a new activation replaces the old one. This time without reset on random move.

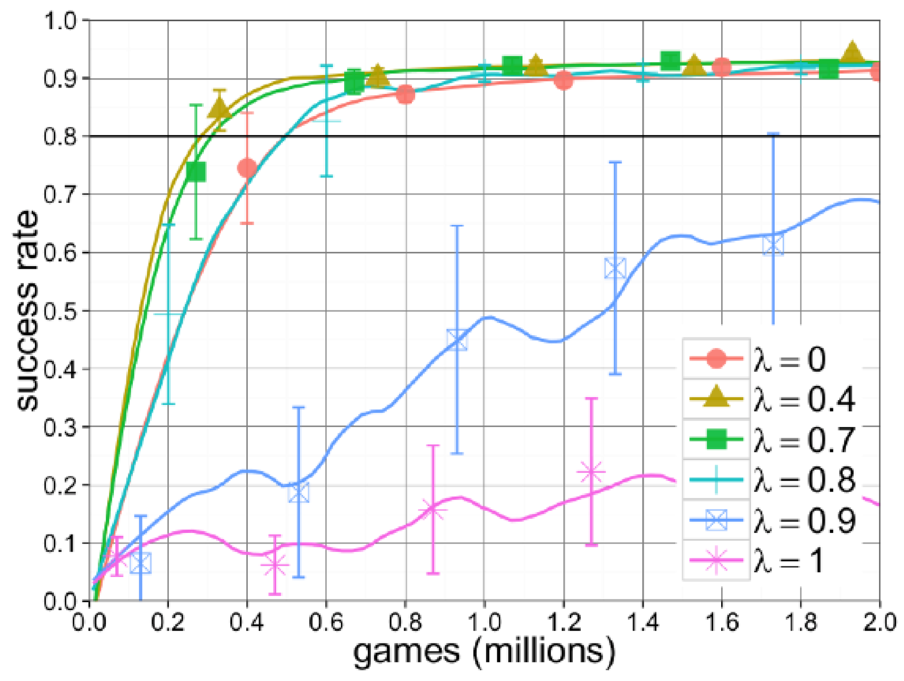



Fig. 1.: Initial results for TCL-EXP with eligibility traces, the options replacing traces and resetting traces turned off. For values λ ≥ 0.9 the system breaks down.

## Unified Algorithmic Description

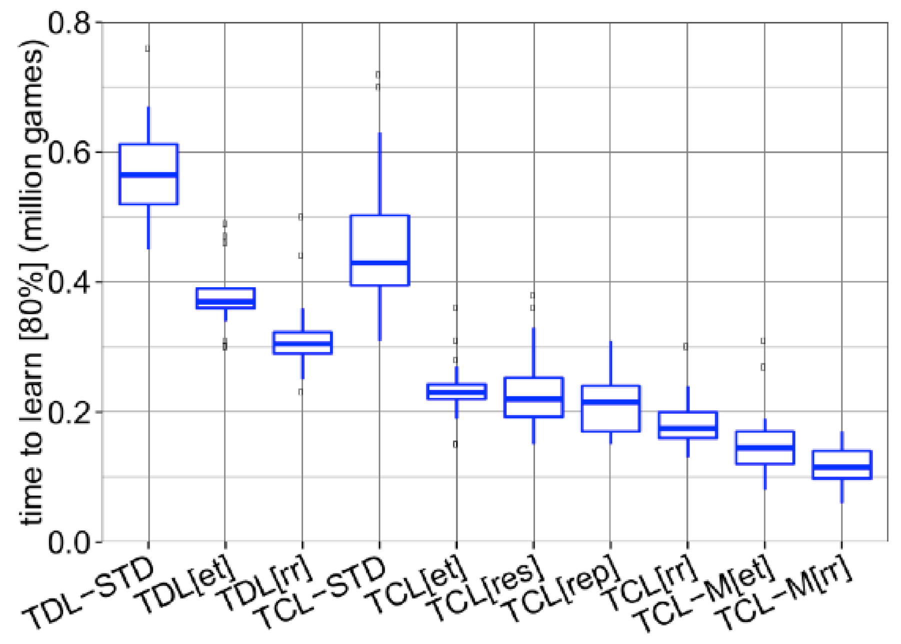
**Algorithm 3** Incremental TD(λ) algorithm for board games. Prior to the first game, the weight vector $\vec{w}$ is initialized with random values. Then the following algorithm is executed for each complete board game. During self-play the player is toggled between +1 (first player) and -1 (second player).

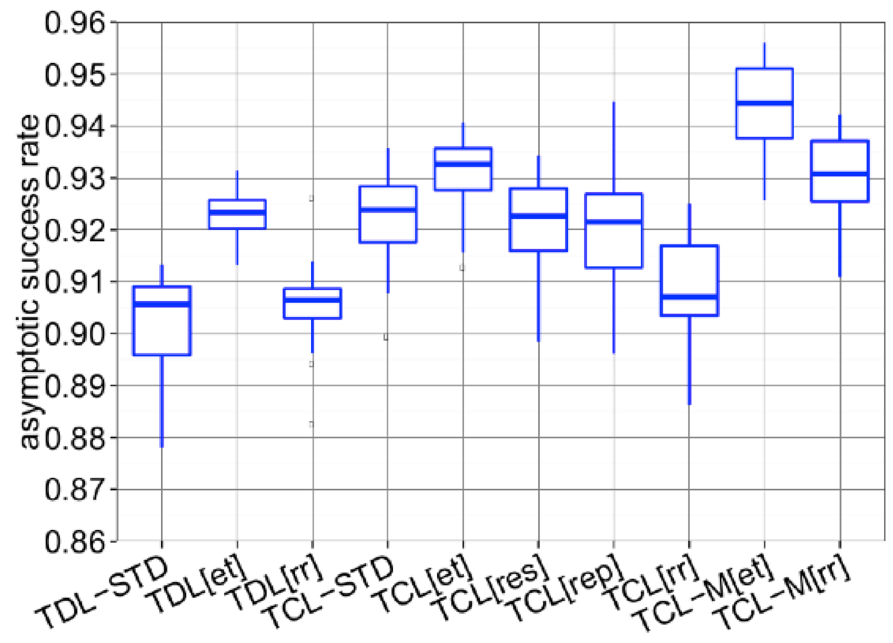
```
1: Set REP = true, if using replacing traces
2: Set RES = true, if resetting elig. traces on random moves
3: Set the initial state s_0 (usually the empty board) and p = 1
4: Use partially trained weights w⃗ from previous games
5: function TDLTRAIN(s_0, w⃗)
6:     e⃗_0 ← ∇_w⃗ ⟨w⃗, x⃗(s_0)⟩                            ▷ Initial eligibility traces
7:     for (t ← 0 ; s_t ∉ S_Final ; t ← t+1, p ← (−p)) do
8:         V_old ← ⟨w⃗_t, x⃗(s_t)⟩                          ▷ Value for s_t
9:         generate randomly q ∈ [0, 1]
10:        if (q < ε) then
11:            Randomly select s_{t+1}                      ▷ Explorative move
12:            if (RES) then
13:                e⃗_t ← 0
14:            end if
15:        else
16:            Select after-state s_{t+1}, which maximizes  ▷ Greedy move
17:                p · { R(s_{t+1}),                         if s_{t+1} ∈ S_Final
                       { ⟨w⃗_t, x⃗(s_{t+1})⟩ + R(s_{t+1}),     otherwise
18:        end if
19:        V(s_{t+1}) ← ⟨w⃗_t, x⃗(s_{t+1})⟩
20:        δ_t ← R(s_{t+1}) + γV(s_{t+1}) − V_old           ▷ TD error-signal
21:        if (q ≥ ε   s_{t+1} ∈ S_Final) then
22:            w⃗_{t+1} ← w⃗_t + α δ_t e⃗_t                    ▷ Weight-update
23:        end if
24:        for (every weight index i) do                     ▷ Update elig. traces
25:            Δe_i ← ∇_{w_i} ⟨w⃗_{t+1}, x⃗(s_{t+1})⟩
26:            e_{i,t+1} ← { Δe_i,               if x_i(s_{t+1}) ≠ 0 ∧ REP
                           { γλ e_{i,t} + Δe_i,   otherwise
27:        end for
28:    end for
29: end function                                    ▷ End of TD(λ) self-play algorithm
```

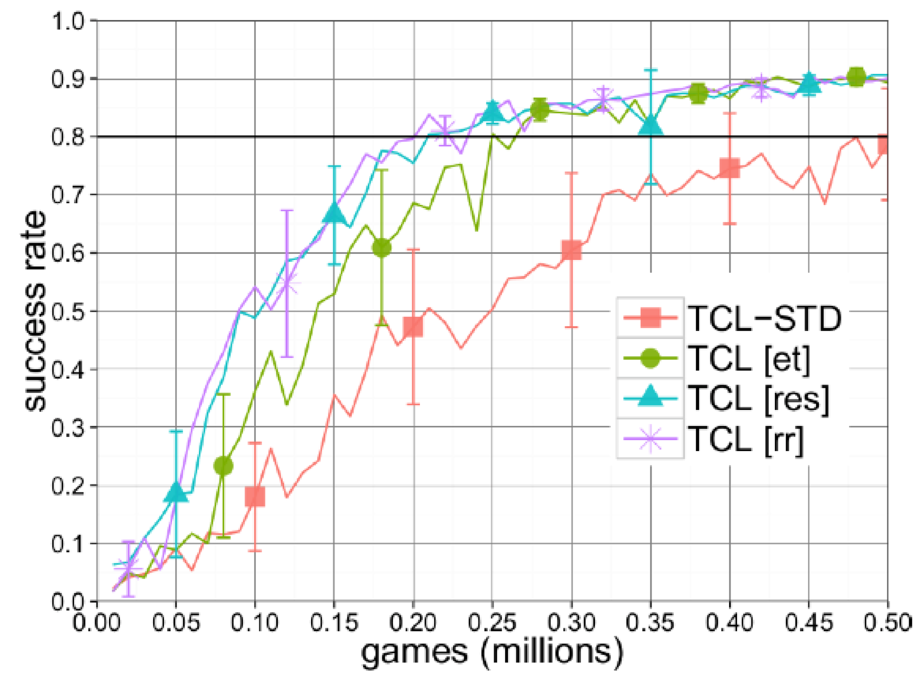
## Experimental Setup

- All agents are trained solely by self-play
- Exploration rate $\epsilon$: 0.1, discount factor $\gamma$: 1.0
- Eligibility traces factor: $0 \le \lambda \le 1$
- $\alpha_{init}$: 0.004 , $\alpha_{final}$: 0.002
- We used 70 n-tuples all of length 8 and generated by random walks on the board
- For a system with 70×8-tuple, 2 · 70 · 4^8 ≈ 6 · 10^6 weights are created
- Weights are initialized with random values uniformly drawn from $\left[-\frac{\xi}{2}, +\frac{\xi}{2}\right]$, with ξ = 0.001.
- Eligibility trace variants: [et] for standard eligibility traces without any further options, [res] for resetting traces, [rep] for replacing traces, and [rr] for resetting and replacing traces.
- Benchmark: Each evaluation point based on 200 matches against a perfect-playing Minimax agent from the empty board.

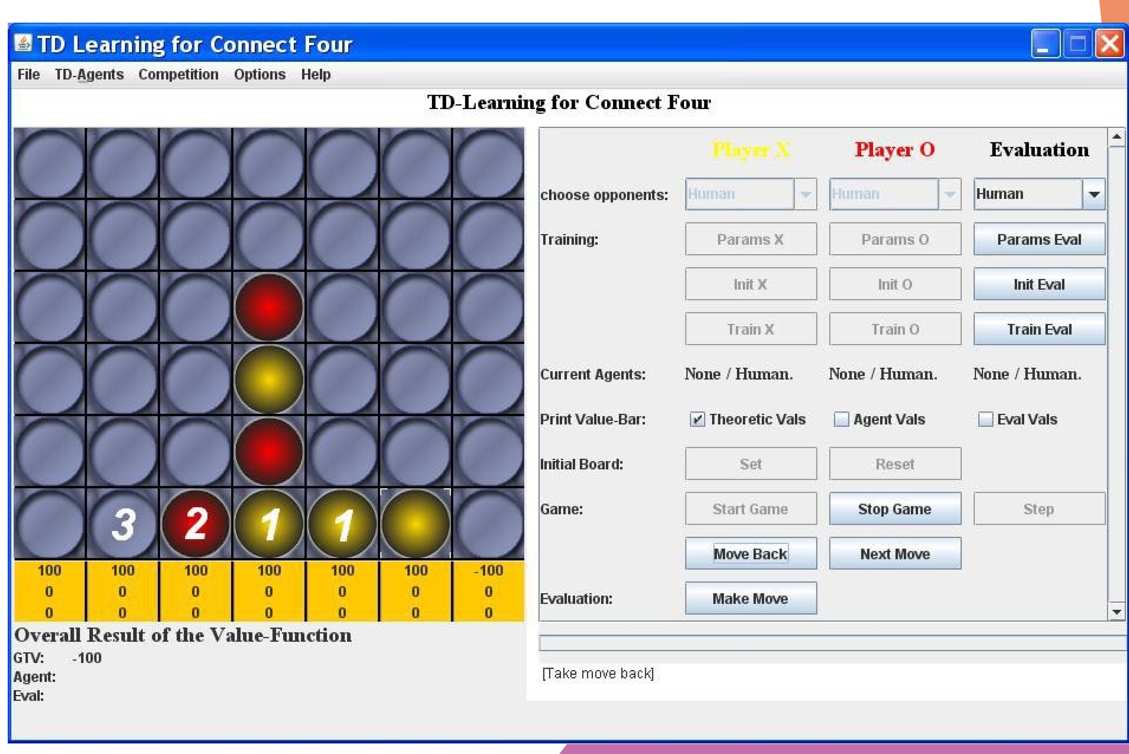
## Summary & Result




## Connect-4 Learning Framework




http://github.com/MarkusThill/Connect-Four

Technology
Arts Sciences
TH Köln

CIOP