Teaching Computers to Play Chess Through Deep Reinforcement Learning

Dorian Van den Heede

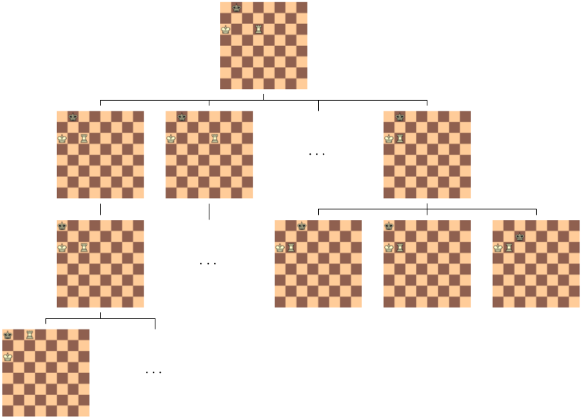
UGent

August 28, 2017

Outline

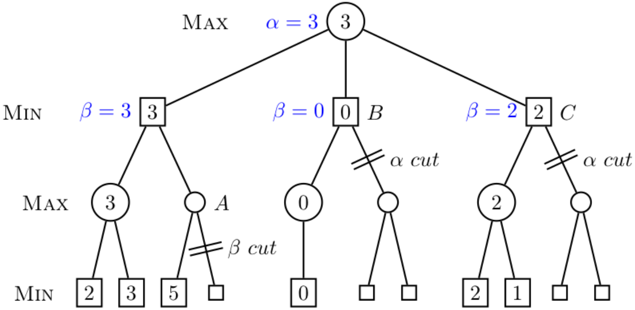
State of the Art

- static evaluation function $V(s)$



State of the Art

- static evaluation function $V(s)$
- Tree search: minimax, $\alpha\beta$-pruning
- Parallel Computing
- Databases



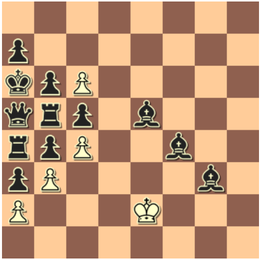
State of the Art

Rank	Name	Rating		
		Elo	+	−
1	Stockfish 8 64-bit 4CPU	3390	+17	−17
2	Houdini 5.01 64-bit 4CPU	3386	+20	−20
3	Komodo 10.3 64-bit 4CPU	3380	+21	−21
4	Deep Shredder 13 64-bit 4CPU	3287	+21	−21
5	Fire 5 64-bit 4CPU	3273	+23	−23
6	Fizbo 1.9 64-bit 4CPU	3253	+26	−26
7	Andscacs 0.89 64-bit 4CPU	3243	+25	−25
8	Chiron 4 64-bit 4CPU	3207	+26	−26
9	Gull 3 64-bit 4CPU	3196	+11	−11
10	Equinox 3.20 64-bit 4CPU	3186	+12	−12

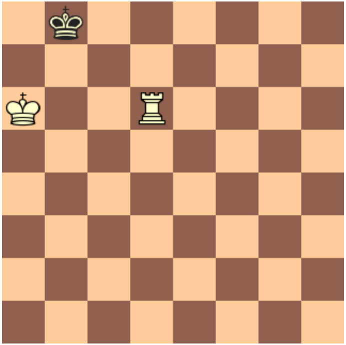
GM Magnus Carlsen (World Champion): 2822

Issues with Conventional Engines

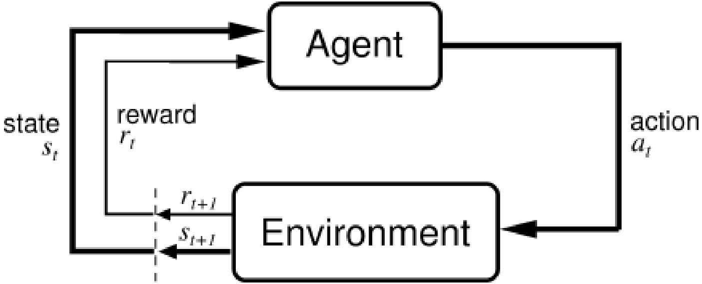
- Humanly biased
  - hand selected positional features
  - opening books
  - databases of grandmaster games
- Brute Force depth-first calculation
  - play based more on calculation than intuition
- Manual tuning and expert knowledge



Can we teach a computer to play chess in the endgame by just giving the rules of the game?



Reinforcement Learning



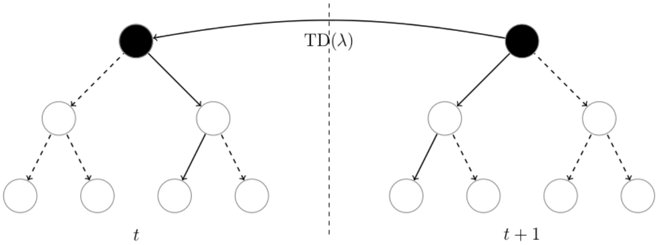
Goal: Maximize future rewards

RL Framework for Chess

- agents: white and black
  $\rightarrow$ self play
- states: board positions
- actions: moves
- episodical
- $$\text{reward}(\text{state},\text{move}) = \begin{cases} 1 & \textit{win} \\ -1 & \textit{loss} \\ 0 & \textit{else} \end{cases}$$
- value function $V(s)$: static evaluation
  $=$ what we try to approximate

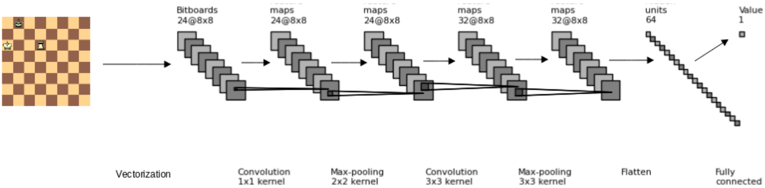
Temporal Difference Learning

- optimizes MSE cost function
- use future for estimate
- TD: $\delta_t = V(s_{t+1}) - V(s_t)$
- TD(λ): $\sum_t \lambda^{n-t}\delta_t$



– bad to spot tactics
– works best if opponent plays well

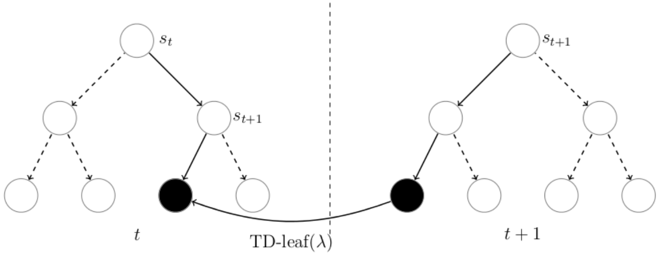
Supervised Learning



piece	piece map	mobility map	piece	piece map	mobility map
♔	$\begin{matrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\1&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{matrix}$	$\begin{matrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\1&1&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{matrix}$	♚	$\begin{matrix}0&1&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{matrix}$	$\begin{matrix}1&0&1&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{matrix}$
♖	$\begin{matrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{matrix}$	$\begin{matrix}0&0&0&1&0&0&0&0\\0&0&0&1&0&0&0&0\\0&1&1&0&1&1&1&1\\0&0&0&1&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&1&0&0&0&0\end{matrix}$	♜	$\begin{matrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{matrix}$	$\begin{matrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{matrix}$

TD-Leaf(λ)

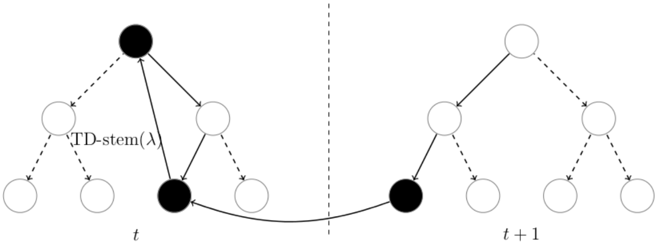
- update leaf states from PV
- TD: $\delta_t = l(V(s_{t+1})) - l(V(s_t))$



+ decorrelates obtained samples
+ use of minimax
− can update unseen states

TD-Stem(λ)

- update encountered states
- TD: $\delta_t = l(V(s_{t+1})) - l(V(s_t))$



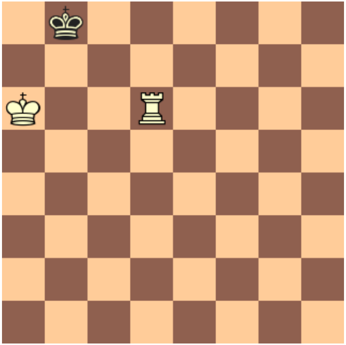
+ includes depth in value function
− no decorrelation samples
− error surface less smooth

Self-play Algorithm

1. initialization
2. self-play $\rightarrow$ replay memory
3. replay memory $\rightarrow$ mini batch SGD
4. go back to step 2 until satisfying convergence

Optimal Opponent Evaluation

- win draw loss (WDL)
- depth to mate (DTM)



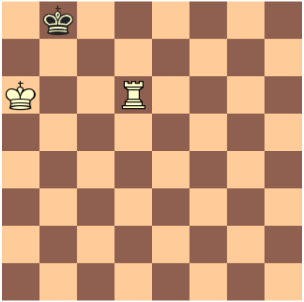
Metrics

$$\text{WCR} = \frac{\text{games model won}}{\text{games model should win}}$$

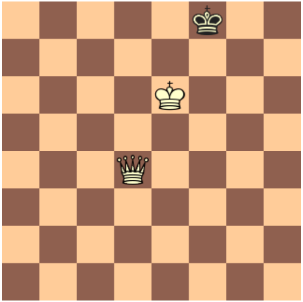
$$\text{WE} = \frac{\text{average DTM of won games}}{\text{average length of won games}}$$

$$\text{LHS} = \frac{\text{average length of lost games}}{\text{average DTM of lost games}}$$

TD-Stem vs TD-Leaf

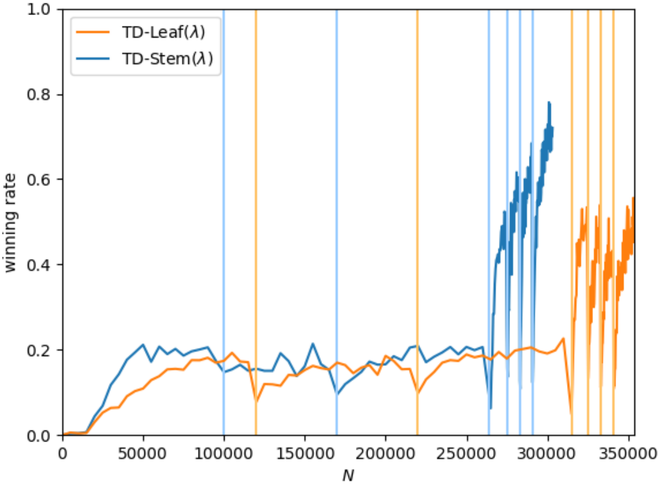


(a) krk



(b) kqk

TD-Stem vs TD-Leaf: krk learning curve

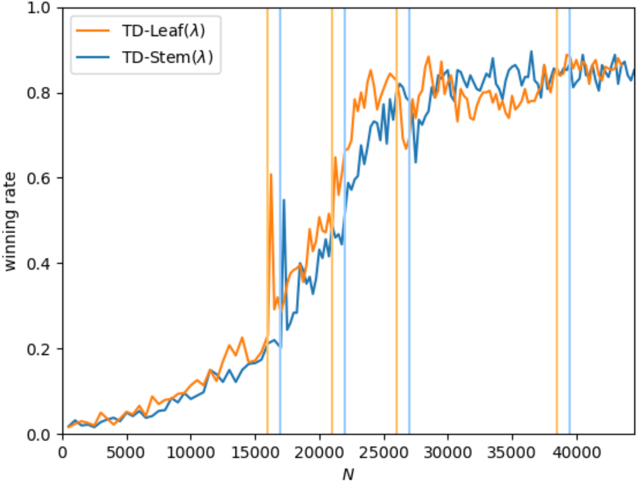


TD-Stem vs TD-Leaf: krk performance

	TD-Leaf(λ)	TD-Stem(λ)
WCR	0.48	**0.85**
WE	**0.87**	0.86
LHS	0.80	**0.91**
MPS	**228**	205
N	353 500	**304 500**

$\rightarrow$ TD-Stem better?

TD-Stem vs TD-Leaf: kqk learning curve



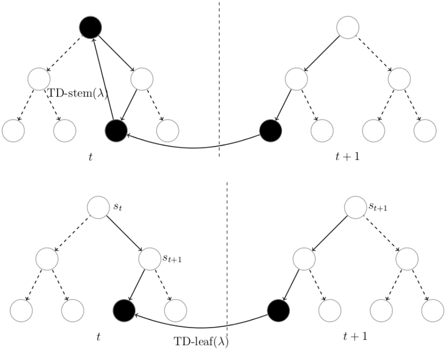
TD-Stem vs TD-Leaf: kqk performance

	3 stages		5 stages	
	TD-Leaf(λ)	**TD-Stem(λ)**	**TD-Leaf(λ)**	**TD-Stem(λ)**
WCR	0.65	**0.77**	0.90	0.90
WE	**0.67**	0.64	0.89	0.89
LHS	0.89	0.89	0.95	**0.97**
MPS	346	**359**	180	**188**
N	**26000**	27000	**43500**	44500

$\rightarrow$ TD-Stem learns faster

TD-Stem vs TD-Leaf: conclusions

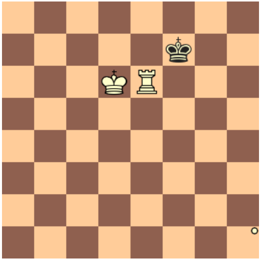
- Why is TD-Stem faster?
  - depth propagation
  - updates seen states

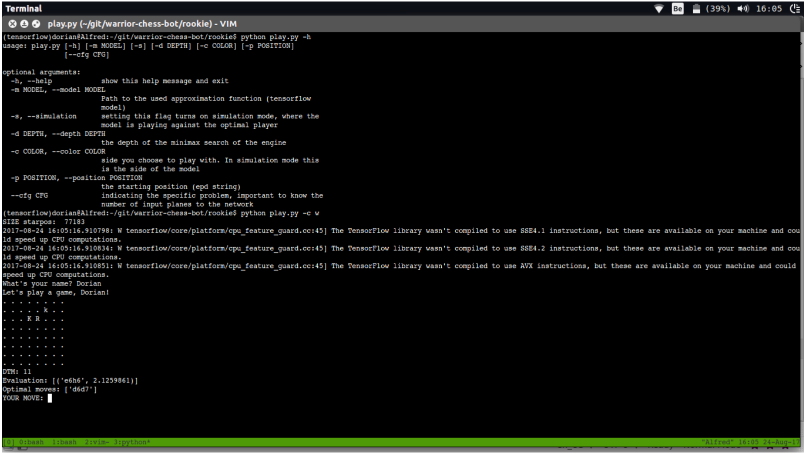


TD-Stem vs TD-Leaf: conclusions

- Experiment limitations
    - specific problems
    - initialization
    - available CPUs
    - time
    - opponent in self play
    - choice hyper-parameters

TD-Stem Demo





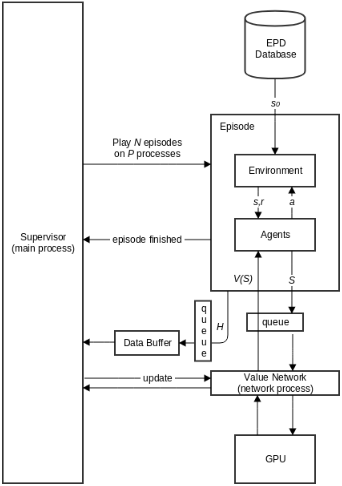

```
(tensorflow)dorian@Alfred:-/git/warrior-chess-bot/rookie$ python play.py -h
usage: play.py [-h] [-m MODEL] [-s] [-d DEPTH] [-c COLOR] [-p POSITION]
               [--cfg CFG]

optional arguments:
  -h, --help            show this help message and exit
  -m MODEL, --model MODEL
                        Path to the used approximation function (tensorflow
                        model)
  -s, --simulation      setting this flag turns on simulation mode, where the
                        model is playing against the optimal player
  -d DEPTH, --depth DEPTH
                        the depth of the minimax search of the engine
  -c COLOR, --color COLOR
                        side you choose to play with. In simulation mode this
                        is the side of the model
  -p POSITION, --position POSITION
                        the starting position (epd string)
  --cfg CFG             indicating the specific problem, important to know the
                        number of input planes to the network
(tensorflow)dorian@Alfred:-/git/warrior-chess-bot/rookie$ python play.py -c w
SIZE starpos:  77183
2017-08-24 16:05:16.910798: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and cou
ld speed up CPU computations.
2017-08-24 16:05:16.910834: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and cou
ld speed up CPU computations.
2017-08-24 16:05:16.910851: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could
speed up CPU computations.
What's your name? Dorian
Let's play a game, Dorian!
. . . . . . . .
. . . . . k . .
. . . K R . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
DTM: 11
Evaluation: [('e6h6', 2.1259861)]
Optimal moves: ['d6d7']
YOUR MOVE:
```

Future Work

- generalization
- initialization network
- policy networks
- search tree bootstrapping
- more bitboards
- different network architectures
- tree search optimizations

Experiments: architecture



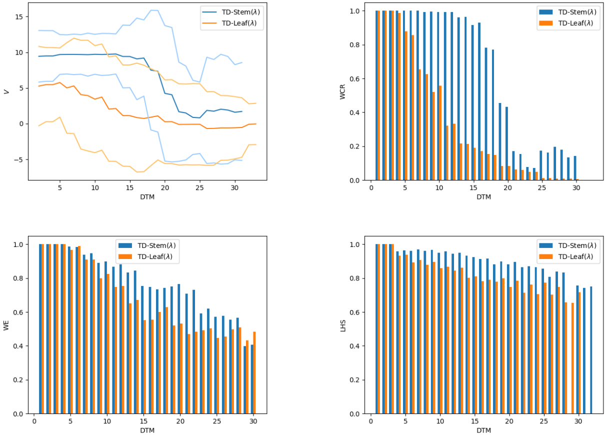
Experiments: hyper-parameters

λ	trace decay parameter for TD-learning methods
d_r	Depth at which final results are examined in search
d_V	Depth at which the value network is called
f_ϵ	Decay function for exploration parameter $\epsilon = f(i)$. i increments every iteration.
I	Number of iterations
i_0	First iteration number, to initialize ϵ with the f_ϵ
K	The number of states that are used to calculate the λ-return from an episode
M	The maximal amount of moves made in an episode
N	How many games are played during an iteration
R	The number of additional random moves played on the board position extracted from the dataset

TD-Stem vs TD-Leaf: krk (stages)

Stage	N	I	d_V	λ	i_0
1	5000	20	1	0.5	1
2	5000	20	1	0.5	2
3	5000	20	1	0.7	2
4	500	20	3	0.8	2
5	250	30	3	0.8	2
6	250	30	3	0.8	2
7	250	50	3	0.8	2

TD-Stem vs TD-Leaf: krk (performance)



TD-Stem vs TD-Leaf: kqk (stages)

Stage	N	I	d_V	λ	i_0	K
1	500	20	1	0.5	0	10
2	250	20	3	0.6	20	20
3	250	20	3	0.7	40	30
4	250	50	3	0.8	40	30
5	250	20	3	0.8	45	30

TD-Stem vs TD-Leaf: kqk (performance)