Reinforcement learning techniques in board games

D'Ambra Luca

Dipartimento di Scienze Statistiche
Facoltà I3S
Sapienza Università di Roma

Why board games?

- Very complex enviroment
- Easy measurement of the performance (result of the game)
- Comparison with humans
- Studied for hundreds or thousands of years
- Games are fun!

History of Reinforcement Learning in Board Games

Historical Achievements:

- **Chinook** (Univeristy of Alberta, 1989)
  Solved the game of Checkers
- **TD–Gammon** (Tesauro, 1995)
  Superhuman level in the game of Backgammon
- **KnightCap** (Tridgell, 1996)
  International Master level in the game of Chess
- **Alpha–go** (Silver et al, 2016)
  Superhuman level in the game of Go

History of Reinforcement Learning in Board Games

HISTORICAL ACHIEVEMENTS:

- **Chinook** (Univeristy of Alberta, 1989)
  Solved the game of Checkers
- **TD–Gammon** (Tesauro, 1995)
  Superhuman level in the game of Backgammon
- **KnightCap** (Tridgell, 1996)
  International Master level in the game of Chess
- **Alpha–go** (Silver et al, 2016)
  Superhuman level in the game of Go

IN THIS THESIS:

- **Ralpha–Toe**
  A program for the game of Tic–Tac–Toe
- **Ralpha–4**
  A program for the game of Connect–4

Reinforcement Learning Problem (I)

Agent-Environment Interaction

In artificial intelligence, reinforcement learning (RL) is the problem of learning from experience.

AGENT-ENVIRONMENT INTERACTION:

- Agent: the learner and the decision-maker
- Environment: the thing the agent interacts with

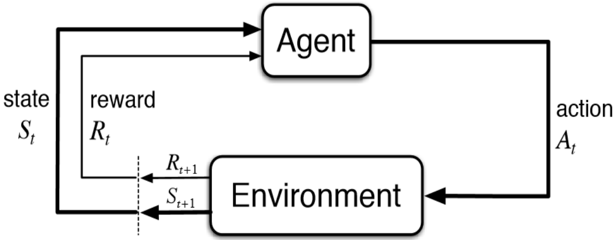
Reinforcement Learning Problem (I)

Agent-Environment Interaction

In artificial intelligence, reinforcement learning (RL) is the problem of learning from experience.

AGENT-ENVIRONMENT INTERACTION:

- Agent: the learner and the decision-maker
- Environment: the thing the agent interacts with



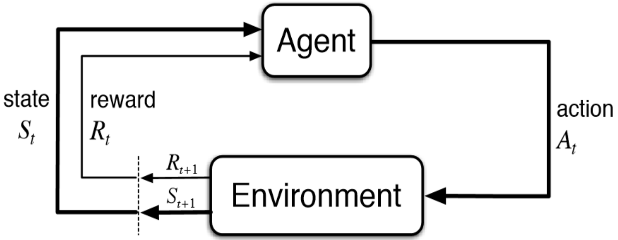
Reinforcement Learning Problem (I)

Agent-Environment Interaction

In artificial intelligence, reinforcement learning (RL) is the problem of learning from experience.

AGENT-ENVIRONMENT INTERACTION:

- Agent: the learner and the decision-maker
- Environment: the thing the agent interacts with



$$\left\{ S_0, A_0, R_1, S_1, A_1, \ldots \right\}$$

Reinforcement Learning Problem (II)

Markov Decision Process

Def. Markov Decision Process (MDP)

An MDP is given by the tuple

$$\mathcal{M} = \big(\mathcal{S}, \mathcal{A}, p(\cdot \mid \cdot, \cdot), R(\cdot, \cdot)\big),$$

where:

- $\mathcal{S}$ is a finite set of states, ($S_t \in \mathcal{S}$)
- $\mathcal{A}$ is a finite set of actions, ($A_t \in \mathcal{A}$)
- $p(s'|s, a)$ is a transition probability ($s, a \to s'$)
- $R(s, a)$ is a reward function, ($R_{t+1} \in \mathbb{R}$)

Reinforcement Learning Problem (II)

Markov Decision Process

Def. Markov Decision Process (MDP)

An MDP is given by the tuple

$$\mathcal{M} = \big(\mathcal{S}, \mathcal{A}, p(\cdot \,|\, \cdot, \cdot), R(\cdot, \cdot)\big),$$

where:

- $\mathcal{S}$ is a finite set of states, $(S_t \in \mathcal{S})$
- $\mathcal{A}$ is a finite set of actions, $(A_t \in \mathcal{A})$
- $p(s'|s, a)$ is a transition probability $(s, a \to s')$
- $R(s, a)$ is a reward function, $(R_{t+1} \in \mathbb{R})$

The agent acts according to a **policy**, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, such that:

$$\pi(a|s) = P(A_t = a \,|\, S_t = s), \qquad \forall\, s \in \mathcal{S}, a \in \mathcal{A}$$

Reinforcement Learning Problem (III)

Optimal policy & State-value function

Solution: Optimal policy (Informally...)

Find $\pi^* \in \Pi$ that maximize the expected value of the **discounted return**:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where $\gamma \in [0, 1]$ is called discount parameter.

Reinforcement Learning Problem (III)

Optimal policy & State-value function

Solution: Optimal policy (Informally...)

Find $\pi^* \in \Pi$ that maximize the expected value of the **discounted return**:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where $\gamma \in [0, 1]$ is called discount parameter.

State-Value Function ($V_\pi : \mathcal{S} \to \mathbb{R}$):

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \,\middle|\, S_t = s \right] \end{aligned}$$

Reinforcement Learning Problem (IV)

Function approximation

Problem: $\mathcal{S}$ is too large

- Not enough memory
  (e.g. Go: 10^{170}, Chess: 10^{43})
- Slow to learn the value of each state individually

Reinforcement Learning Problem (IV)

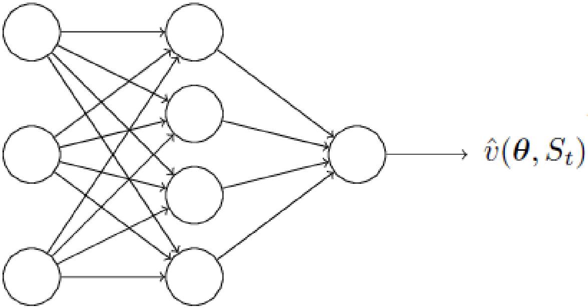
Function approximation

Problem: $\mathcal{S}$ is too large

- Not enough memory (e.g. Go: 10^{170}, Chess: 10^{43})
- Slow to learn the value of each state individually

Solution: **Function approximation**

- Linear combinations of features
- **Neural network** (Our case)
- Nearest neighbour
- $\dots$



TD(λ) Algorithm (I)

What is it?

TD(λ) ALGORITHM (PROPOSED BY R. SUTTON):

Goal: Improve the agent's policy π considering a **parameterized** and **differentiable** function

$$\underset{(s,\boldsymbol{\theta}) \rightsquigarrow \widehat{v}(s,\boldsymbol{\theta})}{\widehat{v} : \mathcal{S} \times \Theta \to \mathbb{R}},$$

in order to approximate $V_\pi : \mathcal{S} \to \mathbb{R}$, where $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^d$ and $d << |\mathcal{S}|$.

- **ON-LINE**: directly using own experience.
- **MODEL-FREE**: without knowing the dynamics of the environment; that is, $p(s' \mid a, s)$ and $R(s, a)$.

TD(λ) Algorithm (I)

What is it?

TD(λ) ALGORITHM (PROPOSED BY R. SUTTON):

Goal: Improve the agent's policy π considering a **parameterized** and **differentiable** function

$$\underset{(s,\boldsymbol{\theta}) \rightsquigarrow \widehat{v}(s,\boldsymbol{\theta})}{\widehat{v} : \mathcal{S} \times \Theta \to \mathbb{R}},$$

in order to approximate $V_\pi : \mathcal{S} \to \mathbb{R}$, where $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^d$ and $d << |\mathcal{S}|$.

- **ON-LINE**: directly using own experience.
- **MODEL-FREE**: without knowing the dynamics of the environment; that is, $p(s' \mid a, s)$ and $R(s, a)$.

How: Improving the current estimation $\widehat{v}$ ($\to$ update the parameter vector $\boldsymbol{\theta}$):

- Minimize the distance between $\widehat{v}(S_t, \boldsymbol{\theta})$ and $V(S_t)$ (...unknown...)
- Gradient Descend method

TD(λ) Algorithm (II)

Update rule

UPDATE RULE:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \Big[R_{t+1} + \gamma \widehat{v}(S_{t+1}, \boldsymbol{\theta}_t) - \widehat{v}(S_t, \boldsymbol{\theta}_t) \Big] \boldsymbol{\epsilon}_t$$

where:

- α is the **learning rate**
- $\boldsymbol{\epsilon}_t$ is the eligibility trace:

$$\boldsymbol{\epsilon}_0 = \mathbf{0}$$
$$\boldsymbol{\epsilon}_t = \nabla \widehat{v}(S_t, \boldsymbol{\theta}) + \boldsymbol{\gamma} \boldsymbol{\lambda} \boldsymbol{\epsilon}_{t-1}$$

- $\boldsymbol{\lambda} \in [0, 1]$ is the decay parameter
- $\nabla \widehat{v}(S_t, \boldsymbol{\theta})$ is the gradient of $\widehat{v}$ with respect to components of $\boldsymbol{\theta}$

TD(λ) algorithm (III)

ϵ-greedy policy & Self-play

The agent acts according to an **ϵ-greedy policy**:

- the greedy action is selected with probability $1 - \epsilon$
- a random action is selected with probability ϵ

TD(λ) algorithm (III)

ϵ-greedy policy & Self-play

The agent acts according to an **ϵ-greedy policy**:

- the greedy action is selected with probability $1 - \epsilon$
- a random action is selected with probability ϵ

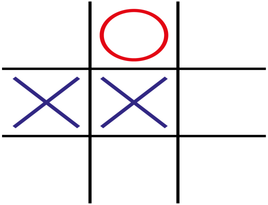
APPLICATION TO BOARD GAMES
VIA SELF-PLAY RL

TD(λ) algorithm (IV)

Algorithm 1: TD(λ) in board games

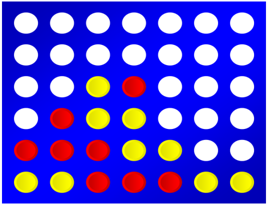
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^n \to \mathbb{R}$;
Initialize the weights $\boldsymbol{\theta}$ arbitrarily;
for *each game* **do**
    $S \leftarrow S_0$ (starting position);
    $\boldsymbol{\epsilon} \leftarrow \mathbf{0}$;
    while *the game has not ended* **do**
        $A \leftarrow$ action selected according to $\hat{v}$(e.g. ϵ-greedy);
        Perform the action A and observe R and S';
        $\boldsymbol{\epsilon} \leftarrow \gamma\lambda\boldsymbol{\epsilon} + \nabla\hat{v}(S, \boldsymbol{\theta})$;
        $\delta \leftarrow R + \gamma\hat{v}(S', \boldsymbol{\theta}) - \hat{v}(S, \boldsymbol{\theta})$;
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\delta\boldsymbol{\epsilon}$;
        $S \leftarrow S'$;
        Let the opponent perform an action;
    end
end
Output: $\pi \approx \pi^*$

Tic–Tac–Toe



- 2 players: X and O
- grid: 3x3
- complexity: $|\mathcal{S}| = 5812$
- SOLVED: teoretically draw

Connect–4



- 2 players: Yellow and Red
- grid: 6x7
- complexity: $|\mathcal{S}| \approx 4.5 \cdot 10^{12}$
- SOLVED: Yellow Wins

Experiment Setup (I)

Train & Test phase

The experiment is divided in 100 rounds. Each round is composed by two phases (Train and Test phase):

Train Phase:

- the program is trained for a certain number of games (Ralpha–Toe: 300, Ralpha–4: 10000)
- the **exploration rate** ϵ is set equal to 0.1
- the **learning rate** α is decreased according to an exponential decay rule, starting from the value 0.05.

Experiment Setup (I)

Train & Test phase

The experiment is divided in 100 rounds. Each round is composed by two phases (Train and Test phase):

Train Phase:

- the program is trained for a certain number of games (Ralpha–Toe: 300, Ralpha–4: 10000)
- the **exploration rate** ϵ is set equal to 0.1
- the **learning rate** α is decreased according to an exponential decay rule, starting from the value 0.05.

Test Phase:

- the program is tested against a fixed opponent for 200 games.
- Ralpha–Toe: Random Player
- Ralpha–4: Benchmark Player
- No updates is performed.
- the **exploration rate** ϵ is set equal to 0

Experiment Setup (II)

Hyper-parameters choice

Algorithm 2: Benchmark Player

1. Get all possible moves and randomize the order;
2. If there is an action that would win the game, perform the first action;
3. Otherwise, if the opponent has any way to win next turn, block the first one found;
4. Otherwise, do the first possible move which does not allow the opponent to win the game by placing a piece on top of it.

Experiment Setup (II)

Hyper-parameters choice

Algorithm 3: Benchmark Player

1. Get all possible moves and randomize the order;
2. If there is an action that would win the game, perform the first action;
3. Otherwise, if the opponent has any way to win next turn, block the first one found;
4. Otherwise, do the first possible move which does not allow the opponent to win the game by placing a piece on top of it.

No general rules for the hyper-parameters choice

We tried 24 different setups in both the games:

- The discount parameter γ is tested for values equal to 0.8, 0.9 and 1
- The decay rate $\boldsymbol{\lambda}$ is tested for values equal to 0, 0.2, 0.5 and 0.8
- Different neural network structures (Deep and Shallow).

Experiment Setup (III)

NN Structures

Ralpha-Toe:

MLP type	Layers size	N. of parameters
Shallow	9 - 100 - 1	1001
Deep	9 - 30 - 22 - 1	1005

Ralpha-4:

MLP type	Layers size	N. of parameters
Shallow	42 - 300 - 243 - 1	86286
Deep	42 - 250 - 250 - 50 - 1	86101

- The bias parameters are initialized with the value 0.1
- The weights of the connection are randomly initialized according to a truncated normal distribution

Ralpha–Toe (I)

Results

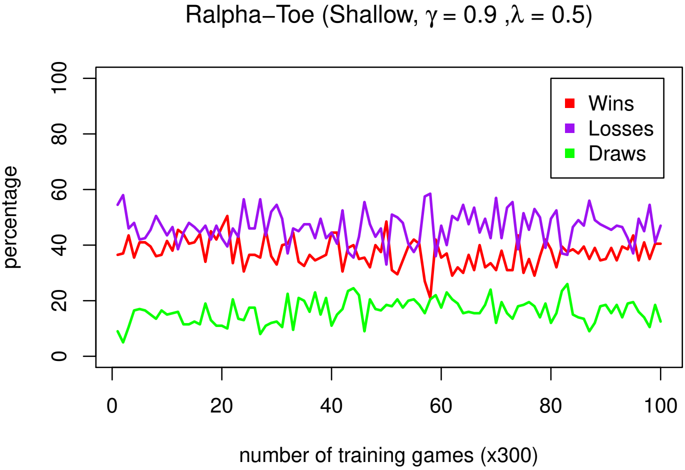
	Shallow architecture		
	$\gamma = 1$	$\gamma = 0.9$	$\gamma = 0.8$
$\lambda = 0$	93.1% (95.5)	93.62% (96.5)	93.57% (96.5)
$\lambda = 0.2$	89.22% (93.5)	90.72% (95)	91.67% (94)
$\lambda = 0.5$	74.4% (88.5)	37.7% (43.5)	90.42% (93)
$\lambda = 0.8$	57.52% (65)	55.22% (62)	81.7% (88.5)

	Deep architecture		
	$\gamma = 1$	$\gamma = 0.9$	$\gamma = 0.8$
$\lambda = 0$	93.82% (97)	91.5% (96.5)	93.1% (96.5)
$\lambda = 0.2$	90.85% (94.5)	92.15% (96.5)	92.82% (95.5)
$\lambda = 0.5$	71.8% (81.5)	88.52% (93)	91.45% (95)
$\lambda = 0.8$	58.05% (64)	56.07% (67)	75.8% (82)

Table: Ralpha-Toe average win percentage (last 20 test phases)

Ralpha–Toe (II)

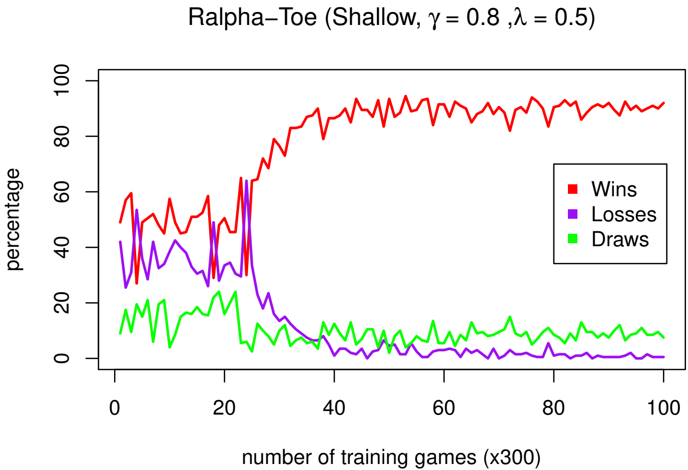
Worst Performance



Last 20 test phases: WINS (37.7%), LOSSES (46.15%) and DRAWS (16.15%)

Ralpha–Toe (III)

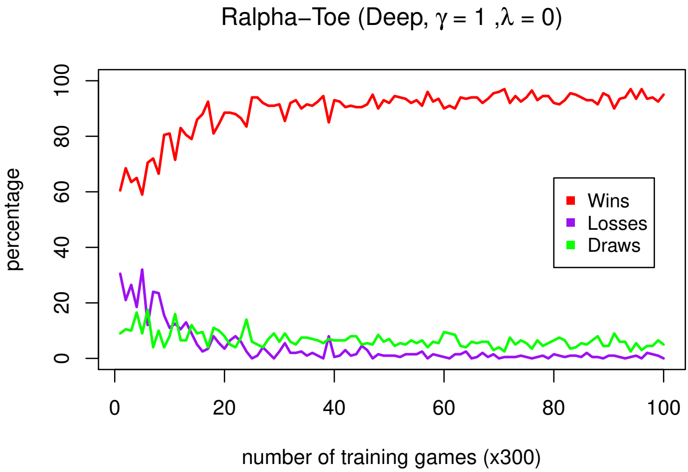
Delayed Improvement



Last 20 test phases: WINS (90.42%), LOSSES (0.8%) and DRAWS (8.78%)

Ralpha–Toe (IV)

Best Performance



Last 20 test phases: WINS (93.82%), LOSSES (0.77%) and DRAWS (5.41%)

Ralpha–4 (I)

Results

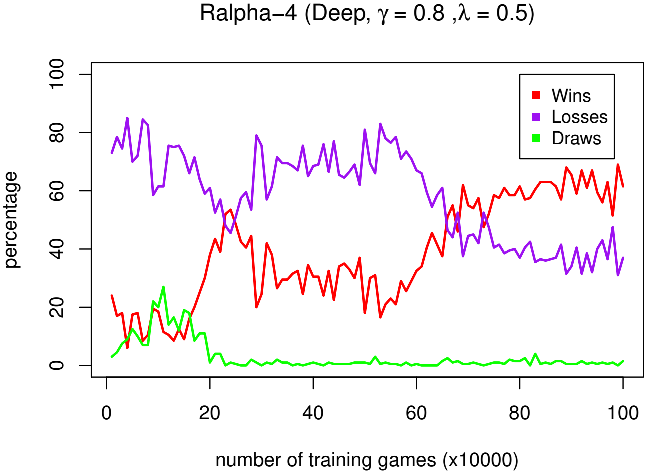
	Shallow architecture		
	$\gamma = 1$	$\gamma = 0.9$	$\gamma = 0.8$
$\lambda = 0$	66.92% (73.5)	78.22% (83.5)	66.97% (74.5)
$\lambda = 0.2$	23.3% (29.5)	80.77% (87)	78.3% (85)
$\lambda = 0.5$	71.17% (77.5)	54.07% (65)	72.7% (78)
$\lambda = 0.8$	1.47% (3)	59.4% (69)	68.8% (74)

	Deep architecture		
	$\gamma = 1$	$\gamma = 0.9$	$\gamma = 0.8$
$\lambda = 0$	75.95% (82)	73.07% (83)	73.02% (81)
$\lambda = 0.2$	75.3% (82)	69.8% (81.5)	70.67% (76.5)
$\lambda = 0.5$	67.37% (73.5)	69.17% (74)	61.52% (69)
$\lambda = 0.8$	9.07% (15)	0.65% (2)	30.52% (40.5)

Table: Ralpha-4 average win percentage (last 20 test phases)

Ralpha–4 (II)

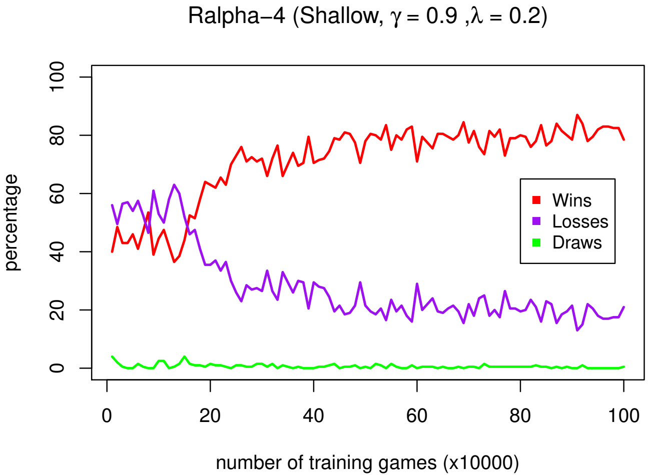
Irregular Improvement



Last 20 test phases: WINS (61.52%), LOSSES (37.45%) and DRAWS (1.03%)

Ralpha–4 (III)

Best Performance



Last 20 test phases: WINS (80.77%), LOSSES (18.95%) and DRAWS (0.28%)