Lecture notes - Introduction to Reinforcement learning with David Silver

Lecture 10

By - Amal Sunny

Classical Games

Motivation for Classical Games

- Simple rules, deep concepts
- Studied and played for hundred/thousands of years
- Good IQ test for AI
- Chess was the most basic case study for RL/AI
- And... well games are fun 😃

State of the Art: AI in games

Program	Level of Play	Program to Achieve Level
Checkers	Perfect	*Chinook*
Chess	Superhuman	*Deep Blue*
Othello	Superhuman	*Logistello*
Backgammon	Superhuman	*TD-Gammon*
Scrabble	Superhuman	*Maven*
Go	Grandmaster	*MoGo*[1], *Crazy Stone*[2], *Zen*[3]
Poker[4]	Superhuman	*Polaris*

[1] 9×9
[2] 9×9 and 19×19
[3] 19×19
[4] Heads-up Limit Texas Hold'em

- Superhuman - beat the best human player
- Perfect - Solved the game
- Grandmaster/Master/International Master - levels of
- This is for all AI methods, next we showcase how far RL has gone for those games.

State of the Art: RL in games

Program	Level of Play	RL Program to Achieve Level
Checkers	Perfect	*Chinook*
Chess	International Master	*KnightCap / Meep*
Othello	Superhuman	*Logistello*
Backgammon	Superhuman	*TD-Gammon*
Scrabble	Superhuman	*Maven*
Go	Grandmaster	*MoGo*[1], *Crazy Stone*[2], *Zen*[3]
Poker[4]	Superhuman	*SmooCT*

[1] 9×9
[2] 9×9 and 19×19
[3] 19×19
[4] Heads-up Limit Texas Hold'em

Game Theory - Optimality in Games

- In a multiplayer game, we can't just have a simple fixed policy as each player's action depends very much on what the other player does.
- Our question is to find the optimal policy π^i for the i th player
- So to make this an RL problem, we fix the policy of all other players as π^{-i} and consider it part of the environment for the i th player.
- But we still need to find the best way to play the game regardless of whatever opponent we have, i.e. generally best policy.
- For that we first define Best response $\pi_*^i(\pi^{-i})$ which is the optiaml policy against those strategies π^{-i}
- And then we use Nash equilibrium as the joint policy for all players.
    - Nash's equilibrium states that every player playing this game would pick the most optimal one bearing in mind other player's strategies
    - i.e. every player's policy is the best response
    - no one would deviate from it

$$\pi^i = \pi_*^i(\pi^{-i})$$

- This might not be the most optimal for *all* opponents, but it acts as the best one against any general opponent.

Single-Agent and Self-play RL

- Best response is solution to single-agent RL problem
    - where we include other players as part of the environment, factoring in their policies into our

MDP
    - Game is reduced to an MDP
    - Solve for the MDP to get optimal response
- Nash equilibrium is fixed-point of self-play RL
    - We generate experience by playing games between 2 agents we made following the same policy
    - Each agent learns the best response to other player, based on the experience tuples generated.
    - And each agent's environment changes based on each player's policy improving
    - i.e. both players are adapting to each other
    - And we do this iterative process until we reach a point, where no further improvments are being made - the Nash equilibrium (if it exists for the problem)

Two-player Zero-Sum Games

- We focus on a special class of games:
    - Two player games - have two (alternating players)
        - P1 is taken as white and P2 as black
    - Zero sum game - Has equal and opposite rewards for Black and White
        - $$R^1 + R^2 = 0$$
- We will consider the following methods to find Nash equilibria in these games
    - Game tree search(i.e. planning)
    - Self-play RL
- We further divide the games into 2 classifications - Perfect, Imperfect information games
    - Perfect information (or Markov) games are fully observed
        - Chess
        - Checkers
        - Othello
        - Backgammon - is stochastic, but all states are observed and known
        - Go
    - Imperfect information ones are partially observed
        - Scrabble - dont know opponents hand
        - Poker - same as above
- We focus on perfect information ones first

Minimax

- A value function defines the expected total reward given joint policies $\pi = \langle \pi^1, \pi^2 \rangle$

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

- A minimax value function maximizes white's expected return while minimizing black's expected return
    - i.e. it assumes each player plays alternatively to estimate value function at every point assuming players played ideally ahead

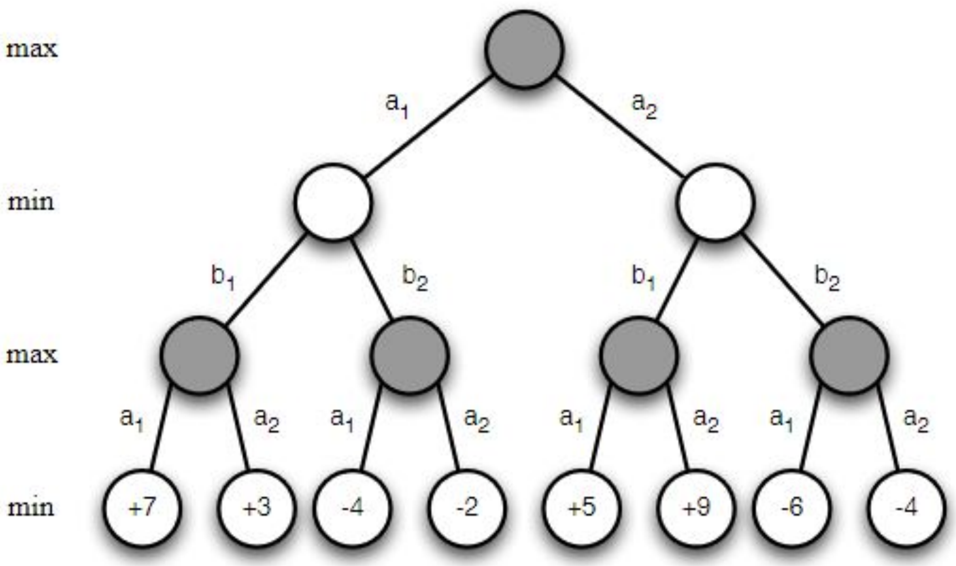
$$v_*(s) = max_{\pi^1} min_{\pi^2} v_\pi(s)$$

- A minimax policy is a joint policy $\pi = \langle \pi^1, \pi^2 \rangle$ that achieves the minimax values. We use a tree like search to evaluate that
    - For larger trees, alpha-beta search is used which cuts away parts of the tree that are not of interest.
- There is a unique minimax value function
- A minimax policy is a Nash equilibrium

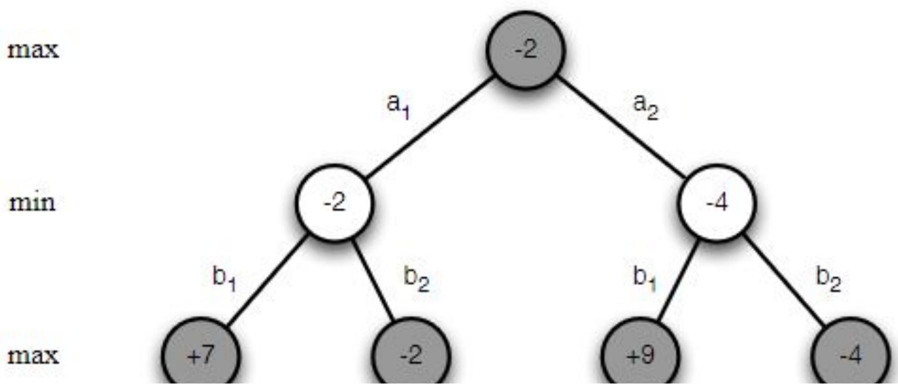
Minimax Search

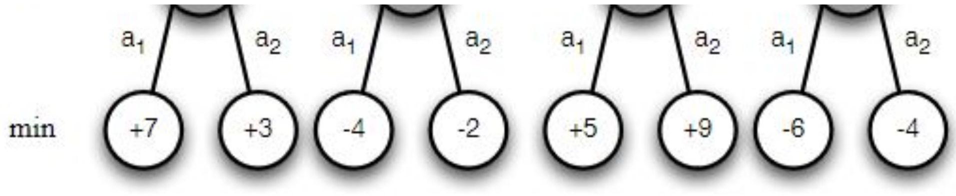
- Minimax values are evaluated by depth-firrst game tree search

Example



- Initially we're given value functions of the bottom most nodes.
- We must find minimax policy with this





- However this tree would grow exponentially for actual games
- Impractical to search till the end - thus we use value function approximator $v(s, w) \approx v_*(s)$
  - also called evaluation function, heuristic function
- Value function is used to estimate minimax value at leaf nodes
- And search is ran to fixed depth w.r.t to leaf values.
- We'll consider an example of this - Deep Blue

Deep Blue

- First actual chess program to beat a top level human player
- Knowledge
  - 8000 handcrafted chess features
  - Binary-linear value function
  - Weights were largely tuned by human experts
- Search
  - High performance parallel alpha-beta search
  - 480 special-purpose VLSI chess processors
  - Searched 200 million positions/second
  - Looked ahead 16-40 ply
- Results
  - Defeated human champion Garry Kasparov 4-2 (1997)
  - Most watched event in internet history
- But this was not RL, this was completely handcrafted features combined with minimax searching.

Chinook

- Is a Checkers program.
- Knowledge
  - Binary-linear value function
  - 21 knowledge-based features (position, mobility, ...)
  - x4 phases of the game
- Search
  - High performance alpha-beta search
  - Retrograde analysis
    - Search backward from won positions
    - Store all winning positions in lookup tables
    - Plays perfectly from last n checkers
- Results

  - Defeated Marion Tinsley in world championship 1994
    - won 2 games but Tinsley withdrew for health reasons
  - Chinook solved Checkers in 2007
    - perfect play against God

Self-Play TD Learning

- We apply previously learnt RL methods to RL algorithms involving games with self-play
- MC: update value function towards the return G
- TD(0): update value function towards successor value $v(S_{t+1})$
- TD(λ): update value function towards the λ-return G_t^{λ}

Policy Improvement with Afterstates

- For deterministic games, we need only to estimate $v_*(s)$ for policy selection.
- This is because we can efficiently evaluate the afterstate

$$q_*(s,a) = v_*(succ(s,a))$$

- Rules of the game would deterministically give us the sucessor state succ(s,a).
- Actions are then selected - min/max-ing depending on black or white's turn.
- This improves joint policy for both.

Self-Play TD in Othello: Logistello

- Created its own features
  - Started with raw input features eg: black stone at c1
  - Constructed new features by conjuction/disjunction
  - Created 1.5m features in diff configs
  - Binary-linear value function using these features
- Logistello used generalized policy iteration
  - Generate batch of self-play games from current policy
  - Evaluate policies using Monte-Carlo (regress to outcomes)
  - Greedy policy improvement to generate new players
- Results
  - Defeated World Champion Takeshi Murukami 6-0

Self-Play TD in Backgammon: TD-Gammon

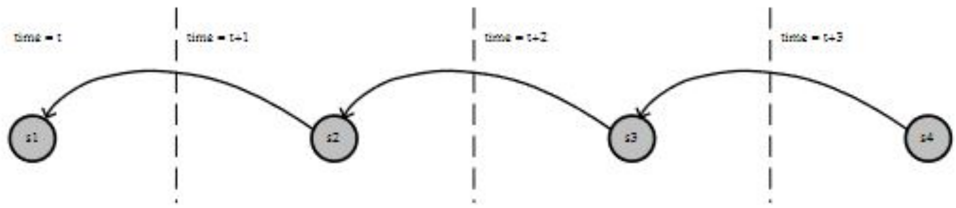
- The board is flattened into a row of stripes, and features are extracted out of the position of beads on those rows - using a neural network.
- Initialized with random weights
- Trained by games of self-play
- Using non-linear TD learning

- Greedy policy improvement (no exploration)
    - Exploration not required due to the stochastic nature of the game (die roll)
- Algorithm always converged in practice
- Not true for other games
    - That is due to the dice roll again - the randomness smooths out the value function.
- Results
    - Zero expert knowledge - strong intermediate play
    - Hand-crafted features ==⇒ advanced level of play (1991)
    - 2-ply search ==⇒ strong master play (1993)
    - 3-ply search ==⇒ superhuman play (1998)
    - Defeated world champion Luigi Villa 7-1 (1992)

Combining RL and Minimax Search

Simple TD

- TD: updates value towards successor value



- Value function approximator v(s,w) with parameter w
- Value function gets backed up from raw value at next state

$$v(S_t, w) \leftarrow v(S_{t+1}, w)$$

- First learn the entire value function by TD learning
- Then use value function in minimax search(no learning)

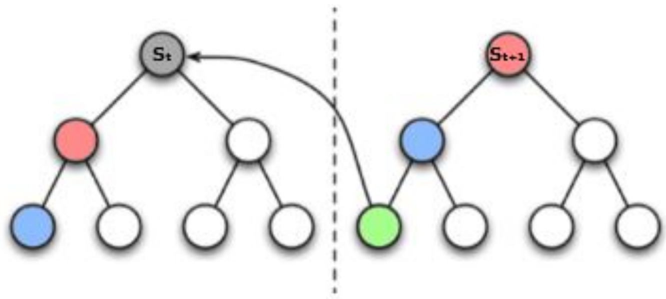
$$v_+(S_t, w) = minimax_{s \epsilon leaves(S_t)} v(s, w)$$

Results:

- Othello: superhuman performance in Logistello
- Backgammon: superhuman performance in TD-Gammon
- Chess: poor performance
- Checkers: poor performance
- In chess tactics seem necessary to find signal in position
    - i.e. we need to use search

TD root

- TD root: update value towards successor search value



- Search value is computed at root positio S_t

$$v_+(S_t, w) = minimax_{s \epsilon leaves(S_t)} v(s, w)$$

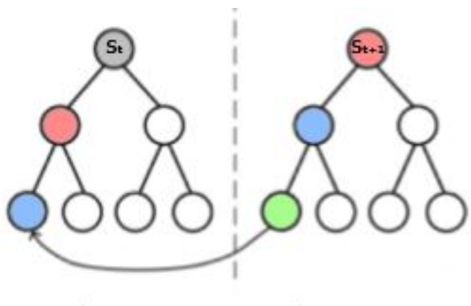
- Value function based up from search value at next state

$$v(S_t, w) \leftarrow v_+(S_{t+1}, w) = v(l_+(S_{t+1}), w)$$

  - where $l_+(s)$ is the leaf node achieving minimax value from s

TD Leaf

- TD leaf: update search value towards successor search value



- Search value computed at current and next step

$$v_+(S_t, w) = minimax_{s \epsilon leaves(S_t)} v(s, w), \qquad v_+(S_{t+1}, w) = minimax_{s \epsilon leaves(S_{t+1})} v(s, w)$$

- Search value at step t backed up from search value at t+1.

$$\begin{aligned} v_+(S_t, w) &\leftarrow v_+(S_{t+1}, w) \\ \implies v(l_+(S_t), w) &\leftarrow v(l_+(S_{t+1}), w) \end{aligned}$$

TD leaf in Chess: Knightcap

- Learning
  - Knightcap trained against expert opponent
  - Starting from standard piece values only
  - Learnt weights using TD leaf

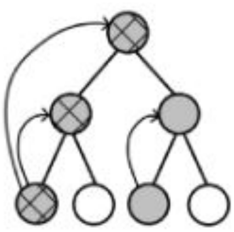
- Search
  - Alpha-beta search with standard enhancements
- Results
  - Achieved master level play after a small number of games
  - Was not effective in self-play
  - Was not effective without starting from good weights

TD leaf in Checkers: Chinook

- Original Chinook used hand-tuned weights
- Later version was trained by self-play
- Using TD leaf to adjust weights
  - Except material weights which were kept fixed
- Self-play weights performed ≥ hand-tuned weights
- i.e. learning to play at superhuman level

TreeStrap

- TreeStrap: Updates search value towards deeper search values



- Minimax search value computed at *all* nodes s ϵ nodes (S_t)
- Value backed up from search value, at same step, for all nodes

$$v(s, w) \leftarrow v_+(s, w)$$
$$\Longrightarrow v(s, w) \leftarrow v(l_+(s), w)$$

- TreeStrap uses all the info from our search tree instead of just taking one - doesn't waste the information gained in the search, uses it all.

TreeStrap in Chess: Meep

- Binary linear value function with 2000 features
- Starting from random initial weights (no prior knowledge)
- Weights adjusted by TreeStrap
- Won 13/15 vs. international masters
- Effective in self-play
- Effective from random initial weights

Simulation-Based Search

- Self-play RL can replace search
- Instead of searching, we just simulate how a game would have gone with whatever policies we've formulated used for both players from current root state S_t
- Apply RL to simulated experience
  - MC control => MC tree search
  - Most effective varient is UCT algorithm
    - Balances exploration/exploitation in each node using UCB
  - Self-play UCT converges on minimax values

Performance of MCTS (MC Tree Search) in Games

- MCTS is best performing method in many challenging games
  - Go
  - Hex
  - Lines of Action
  - Amazons
- In many games simple Monte-Carlo search is enough
  - Scrabble
  - Backgammon

Simple MC search in Maven

- Maven is a program built to play Scrabble
  - while normally we assume a computer would be better, due to having the entire dictionary in data - human players end up memorizing that + strategizing better
- Learning
  - Maven evaluates moves by score + v (rack)
  - Binary-linear value function of rack
  - Using one, two and three letter features
    - Q??????, QU?????, III????
  - Learnt by Monte-Carlo policy iteration (cf. Logistello)
- Search
  - Roll-out moves by imagining n steps of self-play
  - Evaluate resulting position by score + v (rack)
  - Score move by average evaluation in rollouts
  - Select and play highest scoring move
  - Specialised endgame search using B*
- Results
  - Maven beat world champion Adam Logan 9-5
  - Analysis showed Maven had error rate of 3 points per game

Smooth UCT search

- Apply MCTS to information-state game tree
- Variant of UCT, inspired by game-theoretic Fictious Play
  - Agents learn against and respond to opponents average behaviour
- Extract average strategy from nodes action counts,
  - $$\pi_{avg}(a|s) = \frac{N(s,a)}{N(s)}$$
- At each node, pick actions according to

$$A \sim \begin{cases} UCT(S), & \text{with probability } \mu \\ \pi_{avg}(.|S), & \text{with probability } 1-\mu \end{cases}$$

- Empirically, in varients of Poker:
  - Naive MCTS diverged
  - Smooth UCT converged to Nash equilibrium

RL in games: Successful Recipe

Program	Input features	Value Fn	RL	Training	Search
Chess *Meep*	Binary *Pieces, pawns, ...*	Linear	TreeStrap	Self-Play / Expert	$\alpha\beta$
Checkers *Chinook*	Binary *Pieces, ...*	Linear	TD leaf	Self-Play	$\alpha\beta$
Othello *Logistello*	Binary *Disc configs*	Linear	MC	Self-Play	$\alpha\beta$
Backgammon *TD Gammon*	Binary *Num checkers*	Neural network	TD(λ)	Self-Play	$\alpha\beta$ / MC
Go *MoGo*	Binary *Stone patterns*	Linear	TD	Self-Play	MCTS
Scrabble *Maven*	Binary *Letters on rack*	Linear	MC	Self-Play	MC search
Limit Hold'em *SmooCT*	Binary *Card abstraction*	Linear	MCTS	Self-Play	-