Solving Nim by the Use of Machine Learning

Exploring How Well Nim Can be Played by a Computer

Mikael Nielsen Røykenes

12.2019

Origin of the Topic

- An AI challenge made by Harvey Friedman: Can a computer figure out how to play Nim perfectly?
- It includes a desire for having the AI finding proof, however this was outside the scope of this thesis.
- the letter concludes by stating that while AI can do many things, like cleaning and painting houses, the question is, can it do his math?

The Topic and Subject

- Can winning strategies for Nim be learnt by standard reinforcement learning?
- Can it play Nim efficiently, with a high winrate and reasonable timeuse?
- How can the algorithms be improved specially for this problem?

The Game Nim

- An impartial game, which consists of some number of heaps containing counters
- Two players make consecutive moves, the player who makes the last move wins.
- A legal move is a move which removes 1 or more counters from a heap.
- An algorithm exist for playing the game perfectly.

Method: Reinforcement Learning

- Reinforcement learning, using the algorithms Q-learning and Sarsa.

Q-learning and Sarsa are algorithms within reinforcement learning. They are Markov Desicion processes, which means that for each state, the computation of the expected reward can be done knowing the current state and action. This fits for Nim, because it does not matter how a state was reached, when considering your move. The main difference lies in that Q-learning assumes the best possible next move is the one that will be chosen when training, and choses only the current move on policy, while Sarsa also picks the next move based upon the policy.

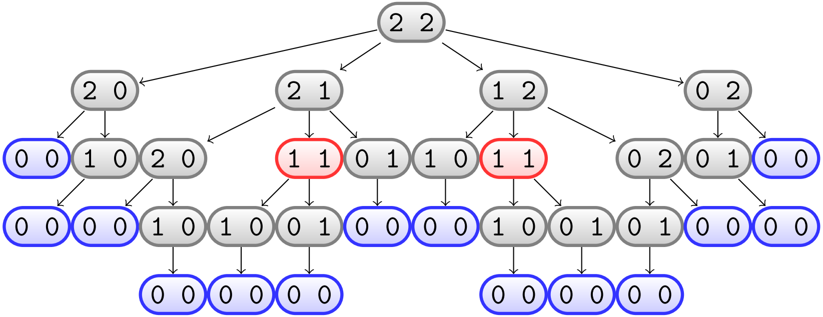
Encoding Nim for Reinforcement Learning

- The algorithms train by updating the values in a table: Q, which represents the expected reward for a move defined by the X and Y axis.
- In this case table Q has a lenght and width which is the same number as the number of nodes in the state space. Each field represents the expected reward of a move from field x in the X axis to field y in the Y axis, if it is a legal move.
- There is also a table T which has fields of value 1 or 0 depending if the move is legal or not. This is essentially a table representation of a graph, in this case the graph which is the game tree of Nim.
- The $Q(a, s)$, which is the value in a field in Q, is also known as the Action-value function. There is also the State-value Function, $V(s)$, which only takes the current state into consideration.

- The typical step in the algorithm for updating the expected reward is $Q(s,a) \leftarrow Q(s,a) + \mu(r + \gamma Q(s',a') - Q(s,a))$.
- For Nim it must be modified for the fact that making a move to a good position is bad, since one would be letting the opponent make a move from a good position. The best choice is to make a move to a position that is bad to make a move from. As follows the expected reward should be inversily related to the expected reward in the next state.
- The step needs only one change, the adjustment of the value is subtracted instead of being added: $Q(s,a) \leftarrow Q(s,a) - \mu(r + \gamma Q(s',a') - Q(s,a))$.

Game tree of Nim

This is the game tree of a game of Nim starting with the state 2 2.



- Ineffective use of space and time.
- Easily improved upon by removing duplicate nodes.

Improved Graph

$$\begin{tikzcd}
 & & 2\ 2 \arrow[dll] \arrow[dl] \arrow[dr] \arrow[drr] & & \\
0\ 2 \arrow[ddrrr] \arrow[dddrr] & 1\ 2 \arrow[l] \arrow[dr] \arrow[dd] & & 2\ 1 \arrow[r] \arrow[dl] \arrow[dd] & 2\ 0 \arrow[ddlll] \arrow[dddll] \\
 & & 1\ 1 \arrow[dl] \arrow[dr] & & \\
 & 1\ 0 \arrow[dr] & & 0\ 1 \arrow[dl] & \\
 & & 0\ 0 & &
\end{tikzcd}$$

Size of the Current State Space.

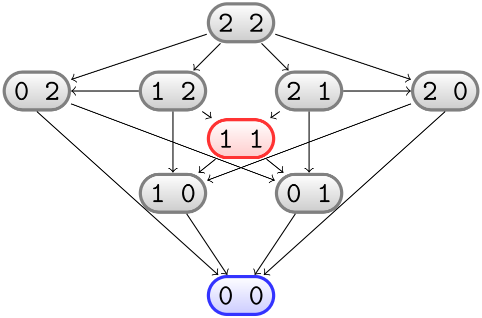
The size of the graph from any state in the game can be calculated as follows, where h_n is the size of the nth heap.

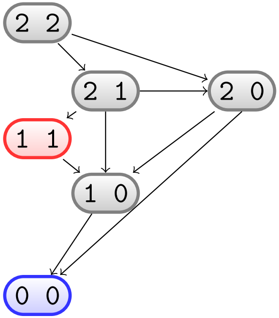
$$(h_0 + 1) * (h_1 + 1) * (h_2 + 1) * ... * (h_k + 1) = number\ of\ nodes$$

So if all heaps are the same size, the size of the graph is simply n^k where n is the size of the heaps.

- This is an improvement, however there are still duplicates.
- There is a node for each permutation reachable from the root state. The order of the heaps in Nim is not important however, so instead of permutations the graph should be trimmed to only feature combinations.

Trimmed Graph





- Since these graphs are based upon the state (2 2) any permutations removed would be mirrored, so the new graph looks as though it had been split in two.
- We keep only the nodes where the sequence of heap sizes is decreasing.

Further Improvements

- Q-learning and Sarsa use the Action-value function.
- In Nim, and many other games, it is only the state of the game that matters.
- An improvement might then be to adapt them to use the State-value function.
- Instead of a Q-table, it has a V-list, of the expected reward for a state.

A Peculiar Phenomenon

when looking at the result of running the ML algorithms a peculiar phenomena can be seen, which is that the win rate of the algorithms is higher with start states with an even number of heaps.

Untrimmed Sarsa		State
0.019s	100%	3 3
0.036s	87%	3 3 3
0.149s	96%	3 3 3 3
0.820s	40 %	3 3 3 3 3

This is because states with equal size heaps of an even number are losing states, there are no good moves from them, so the training explores more, which seems beneficial.

Adapting the Algorithms

- The algorithms might benefit from more exploration, but specifically seem to benefit from training from many states.
- This makes sense, as the measure of success is based upon the algorithm trying to win all possible game-states it has learned.
- The algorithms could be modified to start training games from all possible states in the state space, rather than just the given start state.
- This makes the algorithm train the whole graph in general, instead of just finding the best moves in games from the start state.
- The new algorithm defined like this I call the Modified Sarsa, as Sarsa is modified so that the initialized state is not the same each time.

Results

These are the results using the untrimmed graph. Total time use and winrate is measured.

Sarsa		**Q-learning**		State
0.026s	97%	0.036s	98%	2 2 2
0.036s	87%	0.050s	85%	3 3 3
0.056s	67%	0.072s	57%	4 4 4
0.105s	62%	0.121s	46%	5 5 5

For this case, and in all others, the 2 algorithms use almost the same time, though Sarsa is usually slightly faster.

Untrimmed Sarsa		**Trimmed Sarsa**		State
0.105s	62%	0.039s	66%	5 5 5
1.197s	23%	0.118s	99%	5 5 5 5
32.590s	6%	0.225s	65%	5 5 5 5 5

- Using the trimmed graph for training is a big improvement.
- How about Sarsa with the state-value function, or the modified Sarsa?

Trimmed Sarsa		**State-value Sarsa**		**Modified Sarsa**		State
0.096s	56%	0.134s	58%	0.094s	70%	8 8 8
0.367s	31%	0.667s	64%	0.350s	33%	8 8 8 8
1.492s	13%	1.808s	44%	1.537s	16%	8 8 8 8 8

State-value Sarsa is the most successful algorithm thus far.

Comparison to Prior Research

This is not in the thesis, but is included here for completeness.

- 2 papers:
  - "Reinforcement Learning and the Game of Nim" Paul Graham & William Lord, 2015
  - "Reinforcement learning in the combinatorial game of Nim" Erik Järleberg, 2011
- Both from the Royal Institute of Technology in Stockholm, Sweden.
- The paper from 2011 features a simpler variant of nim, where there is just 1 heap of size 12, and each player may only remove 1–3 counters.

Comparison to Graham & Lord 2015

- The same variant of Nim as used in my paper.
- Only uses the first untrimmed graph as a basis for training, but the Q-table is different: One axis represents all possible moves from the root-state, 1-n counters removed from state 1, from state 2,... etc.
- The training is done 3 ways, against several opponents, so the adjustment of the expected reward is only done after the consequence is known, which is different from this thesis, where the next move is done by the same agent.
- The amount of winning moves made is used to evaluate the success of a program, as opposed to winrate, though it is also recorded.

Back to the Topic.

- Can winning strategies for Nim be learnt by standard reinforcement learning?
- Yes, certainly.
- Can it play Nim efficiently, with a high win rate and reasonable time use?
- While a high win rate seems reachable, a reasonable time use might not be.
- How can the algorithms be improved specially for this problem?
- Algorithms State-value Sarsa and Modified Sarsa are examples of this.