DAT410/DIT728

Group 54
Module 6: Game Playing Systems

Jonatan Hellgren, 960727-7010
Matematikprogrammet GU
gusheljot@student.gu.se

Ankita Rahavachari, 19960709-4605
MPDSC Chalmers University of Technology
ankita@student.chalmers.se

1st March 2021

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions.

Hours spent working

Jonatan	Ankita Rahavachari
20	17

1 Reading and Reflection

1.1 Jonatan

In the paper they go though the method they used to achieve "super human" level of go play. To achieve this they used a combination of neural networks and Monte Carlo Tree Search, all of this was trained and executed on a hpc computer designed for the purpose. This model was trained on expert level games, this data was however not enough since the model became overfitted, however with reinforcement learning (the network generating new data by self-play) they managed to overcome this.

There where two different networks used in the implementation, one that evaluates the value of the state i.e. how favourable is this position for the player. The other one estimated the probability for each next move from the current state. The benefit of having these functions as neural networks is that we are able to compress all the information in the training data into one function, this function would also be extremely complicated if one would try to construct it themselves.

After having read this I wonder if the model is the impressive part or all the work that was done to create it. Yes, the models performance is very impressive, but not really intelligent by it's own right, it is a very narrow intelligence that where put in to the model by computer scientist and professional go players.

1.2 Ankita Rahavachari

In this paper a new concept to computer Go was introduced. To evaluate the board positions 'value networks' was used and 'policy networks' to select the moves. This model is a combination of both supervised learning and reinforcement learning. These deep neural networks play Go using the MCTS programs that simulate thousands of random games of self-play. In addition to this MCTS is implemented with the new value and policy networks.

Two general principles using which the effective search space can be reduced is as follows: Position evaluation can be used to reduce the depth of the search. This approach has led to superhuman performance in many of the games. Second,by sampling actions based on a probability distribution over possible moves a in position s, the breadth of the search may be reduced.

Supervised learning of policy networks:
The policy network probability distribution alternates between convolutional layers and rectifier nonlinearities. The output probability distribution over all legal moves is given by the softmax layer. The policy network is trained based on random rollout policy and supervised learning network policy. It is therefore used to predict human expert moves in a data set of positions. Now a reinforcement learning policy network is initialized to supervised learning policy network. To maximize the outcome, the policy network is further improved using policy gradient learning.

Reinforcement learning of Policy networks:
This is the second stage of the training pipeline and it aims to improve the policy gradient reinforcement learning. The structure of RL policy networks is similar to SL policy networks. In this stage, a game between current policy network and the previously iterated policy network is played. Randomizing from the pool of opponents in this way stabilizes training by preventing overfitting of current policy. The results showed that the RL policy network won more than 80% of the games head-to-head against the SL policy networks.

Reinforcement learning of value networks:
This is the final stage of the training pipeline. It focuses on position evaluation. This is done by predicting an outcome from the positions of the games played by using policy for both the players. The

architecture of this layer is similar to the policy network. The weights of value networks are trained by using regression on state outcome pairs using stochastic gradient methods.

The MCTS algorithm is implemented along with the policy and va;ue networks. At the end of the simulation, the visit counts and action values of all the edges are updated. Once the search is completed the algorithm chooses the most visited move from the root position.

Introduction of MCTS has led to many advances in the game playing systems. Likewise, an addition of policy and value networks to the tree search has taken the game of Go to the next level. It was developed purely through general purpose supervised and reinforcement learning methods. This major improvement in the AlphaGo has allowed it to reach the professional level of the Go game.

2 Implementation

2.1 Monte Carlo Tree Search

In this implementation, we have considered each state of the game as a node, and each possible outcome from a state is considered as the nodes children. Besides parentage and games states each node contains a value representing how favourable the node is for a player, the number of times the node has been considered and an upper confidence bound (UBC) which we will discuss more about soon.

Monte-Carlo Tree search (MCTS) is tree search that uses simulation to estimate values in nodes of the game state tree. The algorithm has four steps:

- Select
- Expand
- Simulate
- Backpropagate.

In the Selection step, we choose which node is to be searched, to do this we traverse the tree downwards from the current state of the game (also called the root node) by selecting the child node that maximizes the UCB estimation. UCB for a node i is computed with the following formula:

$$\begin{aligned} \mathrm{UCB}_i &= \frac{v_i}{n_i} s_i + c\sqrt{\frac{\log N}{n_i}} && \text{if } n_i > 0 \\ &= \infty && \text{if } n_i = 0, \end{aligned}$$

where v_i is the value of the node, n_i is the number of visits of the node and N is the total number of visits in the in the root node. As we can see we have also implemented a new varaible s_i, it is the sign of the piece that is going to be places (-1 if X and 1 if O), with variable the tree traversal assumed that the player it is playing against has the same knowledge and has a roll-out-policy to make the best known move. The first part of the equation is the average score we have gotten when simulating via this node. The second part is a exploration factor, it raises the UCB value for nodes that haven't been explored much, thus we will be able to manage the exploration-exploitation trade-off by select the value for c, we will use $c = 2$, since it worked out for us. As we can see we always select a node if it has not been visited yet, as there are many situations where there will be multiple child nodes with no visits, we have chosen to perform a random draw in this case.

In the Expand step of the algorithm, we will create the chosen nodes children if the node does not yet have any (leaf node), otherwise we move straight on to the next step. This way we keep expanding the

				X			X		X	X		X	X	O			
	O			O			O			O			O				
								O			O			O			

Table 1: An example of a scenario when the second player does not place its first piece in a diagonal and thus ends up in the check mate equivalent of tic-tac-toe.

tree the more we search, this prevents us from storing unnecessary nodes that would fill up the memory.

When we have selected a node and expanded it if necessary we will continue on to the simulation or roll-out step of the algorithm. Here we will randomly sample new state from the current state, this continues on from the newly selected state until a terminal node is reached. A terminal node is defines as a node where one player has an winning alignment on the board or the board is full. We are using a random uniform distribution to simulate each step, this is to make it not to computationally heavy and we do not need to write in any prior knowledge about each step thus keeping the algorithm simple.

In each of the terminal cases a value is generated, we implemented the following three values: -1 corresponds to a X-win likewise 1 corresponds to a O-win and if there happens to be a draw the score 0 is generated. This value is that was simulated from the node we selected by traversing the tree is then back-propagated through the same path that we took to get down to the node. The reason why we update each node from the one selected to the root node is so that we will have some knowledge saved from each turn to the next one, however if we would have kept the value all the way down to the root node, too much unnecessary information would be stored. since it is quite likely that those states wont be reached.

2.2 Evaluation

The algorithm plays very good, we stop being able to win against it at already at 200 MCTS iterations, but performs m which doesn't barely take any time at all. However the game it is playing isn't that complicated so many iterations shouldn't be necessary for the algorithm to play optimally. Now in the implementation it is only possible to be the first player, however if we just try out the function by it self on an empty board it places it's piece in the middle. As well as if we start and place our piece in the middle it places its piece in a diagonal, this is a sign that it can see a few moves ahead otherwise the scenario in Table 1. The algorithm is fairly lightweight it would be possible to use on larger grids, it does however take longer time due to the fact that the roll-out becomes longer since there are more possible states, and if we want better moves on larger grids we need more iterations so that it can see at least some steps in to the future. But with faster software, possible writing in c or c++ and parallelize it would probably only take a fraction of the time and the algorithm would be more scalable.

3 Summary & Reflection

3.1 Jonatan

3.1.1 Summary of Lecture: Game Playing Systems

- Humans have long been trying to build computers that can play against us for a long time, it started (not so seriously) in the 1700s with the mechanical Turk and became more of a real thing in the 1900s with some chess computers
- Zero-sum game is a game where if one player wins the other player loses, these games can be solved using decision trees

- Monte-carlo tree search is such a method to explore the tree, it is good since it does not search though the entire decision tree.
- When playing with a search tree we have a policy p that select the next node that we will go to, this is often to choose the maximum value, however finding the value for states can be a bit tricky and is thus often estimated with simulation of machine learning.
- When simulating a few moves ahead we also need to assume what moves the opponent is going to make, the best choice for this is that we assume the opponent is going to make the best possible move and thus we must be ready for it and maximize our changes while expecting that the opponent will minizmize them (minimax).
- When using MCTS there will be a tradeoff between exploration and exploitation.

3.1.2 Reflection on the previous module: Diagnostic Systems

The previous module was quite good, we got to try out to model expert knowledge into code manually and later compare it to some of sklearns models, obviously the more frame worked models had a better performance. We also talked about the interpret-ability of the models, which is something that is very necessary in certain domain such as medical care. We also read an article that discussed what really makes a model interpret-able or not, it basically depends on what prior knowledge a person has about both the domain and the model being fitted to the data.

3.2 Ankita Rahavachari

3.2.1 Summary of Lecture: Game Playing Systems

Our goal while playing a game is to select an action that improves our chance of winning. The possible future actions are modified based on our present actions. Inorder to win, we look for good/bad features when selecting actions. For an optimal game, we should pick the best path. Once we traverse through the best path we reach the end state at some point.

Using a search tree we can enumerate the possible features. By doing this we can identify the good features and backtrack the same path after reaching the terminal node. For each position the player keeps the probability of reaching winning states can be computed based on the probability of future states.

Minimax optimization is implemented to guarantee success against the best opponent by minimizing the maximum success of the opponent. To describe game as a decision policy we first describe the actions according to a policy in the states observed through a path which transitions according to dynamics. Even though minimax optimization avoids losing to strong opponents, exhaustive search is not always feasible. There are non-terminal nodes which aren't explored using this method.

There are two problems with large state spaces, one of them is we can't explore every state. To deal with this we can implement the Monte-Carlo Tree search method as it deals with random sampling. Instead of searching exhaustively, this method works search based on randomly selected actions. MCTS process involves Selection of unexplored nodes and traverse down the path until the unexplored child node is reached. The next step is to expand the selected node. In this case an unvisited child node is selected. The third step is to simulate the game from the unvisited child node based on a simulation policy. Every node is expanded until the terminal node is reached. We know the value only at the terminal node. From the terminal node, the final step backpropagation is performed to trace the expanded nodes and mark as visited. By doing this we will be able to infer the set of good moves.

Greedy policy selects the best action according to some metric but this leads to no exploration. Bandits deals with decision making under uncertainty.

Upper confidence bounds is a popular heuristic. Application of UCB to search trees is called UCT.

The second problem is that we can't store the value of each state. We can overcome this by letting the function approximate the Q value instead of storing it for every state.

3.2.2 Summary of Follow-up Lecture: Diagnostic Systems

- A model is interpretable if there exists Trust, Causality, Transferability, Informativeness and Fairness
- If B causes A and A causes B, both are understood to have the same probabilistic independencies. But we can't tell which of the decompositions are causally sound without defining what we mean by causal. Casualty is often defined in terms of interventions.
- A casual model tells us what happens under (A = a), i.e., in $P(B|do(A = a))$ we find what happens to B when we set A. This is called interventional distribution, the non-interventional distribution is called observational. Note that they are not the same.
- A causal graphical model is not implied by the distribution.
- A mutilated graph is the one in which all the incoming edges to the intervened-on variable are removed.
- Structural causal models are those which links distributions and structure through a set of deterministic relations. An intervention replaces an equation in the SCM and mutilates the graph

3.2.3 Reflection on the previous module: Diagnostic Systems

This assignment gave an in detail view of interpretability and how these systems can be beneficial. It allows us to learn the structure of the observational data and simply used to get more information from the model. These systems should be deployed in a domain based on their long term advantages. In the implementation part we had the chance to develop a model based on a simple rule based classifier, random forest classifier and a decision tree classifier to assign the diagnosis malignant and benign. This exercise gave the insight of how well these classifiers performed on the given data. Finally, the discussion part is where I was able to describe how interpretable systems sometimes can be a pitfall in the field of medicine.