Module 6 - Game playing systems

Group: 12
Anton Claesson, 971104-3217, MPDSC, canton@student.chalmers.se 18h
Lukas Martinsson : 980203-9678, MPDSC, malukas@student.chalmers.se 20h

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

Reading and reflection: Anton

The paper starts with stating that all games with perfect information have an optimal value function for each state that determines the outcome of the game under perfect play by all players. However, in games such as chess and go the search space is infeasible to exhaustively iterate. The search space can be reduced with an approximate value function computed through monte carlo tree search which estimates the value of each state in the search tree. The policy used to select actions during search over time converges to optimal play and valuations to an optimal value function.
The paper describes using a neural network for representing the board states, a value network to estimate value and another policy network to sample actions. This way the effective depth and breadth of the search tree can be reduced.
These networks were trained using a combination of supervised learning and reinforcement learning, and were combined with monte carlo tree search rollouts to create AlphaGo. During the supervised learning phases data from expert human players was used to optimize the networks, while the reinforcement learning phases utilized self-play.
To evaluate AlphaGo they ran tournaments against several other AlphaGo variants as well as other programs. They managed to beat all other programs consistently. Finally, they played five games against a human world champion Go player and won all of them, a tremendous achievement that also showed that certain areas in AI that might seemingly be intractable today could actually be solved.

Reading and reflection: Lukas

The paper *“Mastering the game of Go with deep neural networks and tree search”* is about a new model using both Monte-Carlo tree search in tandem with value and policy networks to make it as good as possible at the board game Go. Professional standards of Go reaching to worlds best player for AI models was previously thought to be at least a decade away. This is due to Go’s complexity, since not only is it hard to evaluate each position, each position also has about 250 different possible moves with games being played for up to 150 rounds. This compared to tic-tac-toes 9 different moves that only plays for 9 maximum rounds or even chess 35 possible moves with up to 80 rounds makes this a huge leap in computational power or time if

the model uses the same ways of calculating as them. The model not only beat the previous winner of the European Go championships but also vastly outperform previous models used for playing Go, even when giving them handicaps. Given that the same time is used for each model this is an impressive feat. Lastly, through a figure it demonstrated how each combination of rollout, value network and policy network gave different elo ratings. Interesting a combination of all of them compared to only using rollouts gave almost a doubling in rating from around 1500 to almost 3000. This paper thus demonstrates the efficiency of combining neural networks with monte carlo for playing Go. How well this would translate to another complex game such as Chess would be interesting to see.

Implementation

We choose to implement our Tic-Tac-Toe game with inspiration from the first linked blog. Thus, we chose to implement Monte-Carlo tree search. This used tree-traversal to expand the nodes (which can be looked at as potential moves) and a rollout function to evaluate. To determine which node to explore we used UBT:

$\frac{score}{n} + c\sqrt{\frac{\log N}{n}}$, $c = constant$, $n = nr\ of\ visits$, $N = number\ of\ visits\ (parent\ node)$

This enables us to have an exploitation-exploration trade off based on the number of times the node and its parent have been visited. Since we used UBT we could guarantee that all possible first moves of a node will be evaluated. The UBT gives us the child node with the highest score (based on previous roll-outs and how many times it has been visited). If the child already has a score, i.e. it has already been evaluated before, the same thing occurs for the child node. If not, a roll-out for that child node occurs. Our roll-out policy was in accordance with the MCTS we found online which is to let the child node make random moves until it reaches a terminal state. Then that state is evaluated and the score of the child and its parent is updated accordingly until the root node is reached. The winning, losing, and tie state of a rollout gave an update of the score with 1, -100, and 0 respectively (more about this choice later). By iterating over this tree enough times a score for each subsequent move could be found where the child node with the highest score corresponds to the optimal move. Note that this highest score was calculated using UBT with c = 0.0 in order to utilize exploitation only when selecting the actual move to make, but during MCTS $c = 2.0$ was used to encourage exploration.
We note that the model did not have a selection policy that for example assumed that the opponent placed perfectly. However, since it could theoretically evaluate enough states (since tic-tac-toe is not a large game) to play the most optimal moves given that the opponent played randomly it should always play the move that gave it the highest score, i.e. minimized its chance to lose and draw and maximized its chance to win.

When determining the win-lose-tie score we had to play against the model several times. We found that when setting the score to 1 (winning), -1 (losing), 0 (tie) the

model would not always play optimally. There were instances were it would rather make a move that, given that the player does not play a winning move on he's/she's guarantees it victory, for example:

O		O
	X	
X		

It is the models turn to play (X). If the losing score is not sufficiently bad (relative to the winning and tie score) it choses (2,1) instead of (0,1). This gives the player the option for an immediate win by playing (0,1):

O		O
	X	
X	X	

Here, given that the player does not play (0,1) the model will have one or two winning moves. Therefore we assume that this issue arrives when it evaluates each possible move. Since our model assumes random play instead of optimal play by the player, it thinks that losing ¼ of the time is better since it will win the rest, instead of guaranteeing a tie since it gives a higher score. It is hard to say what the "optimal " win-lose-tie score should be for it to always make the optimal play since that would require us to evaluate several positions and compare how the model thinks. However, we theorized that by setting a very negative losing score (for example -100) we would encourage the model to play for ties or wins and discourage it massively from losing. Of course one could simulate appropriate values here by simulating games where the model for example plays against itself, but we did not have time for that. It also seemed as these chosen scores of 1, -100 and 0 respectively made the model act optimally as it always seemed to either win or play a tie when it got to start.

We tried different values for the iterations made before the model chooses the best move ranging from 1 up to 10 000. Similarly here we didn't have time to find an "optimal" time to efficiency tradeoff. However, since even with the 10 000 the model only took about 2 seconds to find the optimal action this was not really relevant. If we were to implement a similar algorithm for Go or Chess this would however be a parameter that we would have to tune more carefully. It is harder to say how well the model would scale to a larger grid like a 4x4 or 5x5 grid since the computational time would increase and additional functions would have to be defined. However, optimal play for a 4x4 or 5x5 grid would still be within the scope of what a fast computer could calculate. Nevertheless, the time to accuracy tradeoff here would be more interesting to consider since it would require significantly more than 2 seconds. Also in a 4x4 or 5x5 grid, additional functionalities similar to alpha/beta pruning could

possibly be considered to reduce the computational time whilst maintaining a similar accuracy.
Note that our implementation would also need to be slightly modified to allow a different grid size, for example our code to evaluate the result of a finished game.

Although we have not researched tic-tac-toe optimal play we can be pretty confident in thinking that our model is playing optimally with the above stated iteration numbers and win-lose-tie score. This is due to the fact that tic-tac-toe only has a limited amount of possible strategies and moves so the optimal move can always be found by even a human without much experience with the game.

Individual Summary and Reflection: Anton

Lecture summary (Game playing systems)

The lecture started with a brief overview of the history of game playing systems. Chess computers were the first examples of such systems and they first appeared in the 1900's, famously culminating in the matchup between chess world champion Kasparov and the Deep Blue computer.
Since then machines have been shown to be able to outperform humans in increasingly more complex games (at least in terms of creating a game playing system).
Next, the lecture talked about how to formalize games. First of all, we make the assumption of having a zero-sum game; if one player is winning then it is implied that the other player is losing (for example tic-tac-toe). We want to figure out the actions that best improves the chances of winning, which requires us to account for how our current action affects future actions. We also need to account for the fact that we don't really know the policy of our adversary. Therefore, we use minmax optimization where we want to minimize the maximum success of the opponent, thus "guaranteeing" success against the strongest opponent.
Thus, we represent a game as a tree where each node is a game state (example board placement) and each edge is an action available to us in that state.
The terminal nodes are states where the game ends typically associated with some kind of reward or score. From this end state one can recursively calculate a value of previous states.
In many cases, the state space of the game can be intractable to exhaustively search, meaning it is hard to estimate the value of a certain state and thus what the best policy is. In these cases we need to use an approximation of the value which can be obtained by methods such as Monte-Carlo tree search.
Finally, we talked a bit about the exploration/exploitation trade off which is something that needs to be considered such that we choose to investigate a sufficiently large state space rather than greedily choosing actions based only on what we have found so far. This can for example be managed using upper confidence trees.

Lecture summary (Diagnostics systems follow-up)

The lecture initially focused on the concept of model interpretability, and how causality connects to it. The big question when it comes to causality is if what we observe is just correlation or if we can conclude causation.

In order to find a model which is most “casually sound” (has the best explanation) we must first define what we mean by causal but also have something more than just statistics and joint distributions. We can do this by “intervening” and changing the system and seeing what happens compared to what we expect. We learn things this way by intervening rather than just observing.

Causality can thus be defined in terms of interventions- i.e. what happens when we tamper with a system.

A causal model should tell us what happens when we make an intervention, rather than what happens when we observe something without intervening.

One such model is structural causal models, which has both causal structure and probabilistic distributions.

Reflection on the previous module (Diagnostics systems)

From the diagnostic systems module, I think what we learned and discussed about model interpretability was the most interesting. As an AI engineer, it is very important to reason about these things as well rather than just trying to maximize some metric. Connecting interpretability to examples in the medical field was a very good choice as it very clearly drove the picture home for why we need to be able to interpret our models in many cases. Furthermore, learning about how AI can be used in the medical field as well as how it is bound to be used more and more to save lives and improve healthcare was very interesting as well, as it is motivating to learn about technologies that can benefit society in such a distinct and clear manner.

Individual Summary and Reflection: Lukas

Lecture summary (Game playing systems)

The lecture firstly gave a history of AI systems for board games. How first AI’s for chess were created in the 1900s and their development until deep blue which was a chess engine capable of beating the then current world champion Garry Kasparov. After this the chess engines only advanced and are nowadays far ahead of any human. Then the lecture explained how not only zero-sum perfect information games like chess and Go but also more complex games like Dota 2 could also be played by AI, the difficulty being here that the models do not have all the information as in zero sum perfect information games.

Then an explanation of zero-sum games was explained and how they, with tic-tac-toe as an example, can be demonstrated as search trees with the children of the original root node being the moves in the game. Through a then exhaustive search of the tree the optimal move could be found. However since this is time inefficient for large games, such as Go and Chess this is not viable. Further the lecturer explained how depending on different assumed policies of the person the

model is competing against influences its processes. For example if it assumes the person is playing at random or assuming they are always making the best move. Then the problem of “solving” larger games was explained. Through for example Monte-Carlo tree search a approximate score could be found for each move and if it can search long enough it will likely find the optimal or at least a good move. Monte-Carlo tree search was then explained in depth since it would be used (or could be used) for this week’s assignment. Lastly the exploitation- exploration problem was explained through one armed bandits as well as upper confidence bounds used for determining which move to make.

Lecture summary (Diagnostics systems follow-up)

The lecturer first wanted us to discuss the problem of a black box AI versus a human when it comes to surgenos. This is a difficult question if the black box is only slightly better: This was discussed in the case where a black box had an accuracy of 90% versus the humans 80%. Since most humans want to understand the reasoning behind decisions, especially in diagnoses, this is not as simple as one first thinks. Then causality was explained and its link to interpretability. To define causality in its simplest form is easy, the x value is affected y much depending on the z value. However, the reality, as usual, is far from trivial since other features could be causal to both of x and z and could be the real reason for their relationship. Further the lecture explained and went into a discussion of how to define an interpretable model. It comes mainly down things such as, trust, causality, transferability, informativeness and fairness. Then the question that can be further asked is how one defines fairness, and for whom should the model be interpretable and so on.

Reflection on the previous module (Diagnostics systems)

It was interesting to see how many different ways one can create diagnostic systems and how it has developed, especially in medicine. And even when creating simple rule-based classifiers one has to carefully find good cut-off points whilst at the same time taking specificity/sensitivity into account. And even if covid-19 is not as severe for the society as before it was still really interesting to discuss it from a more machine learning perspective whilst still using domain words from medicine to define it. Lastly, a part of the previous module I found really interesting was how one defines interpretability. Before I would think it trivial, however with examples with black box AI’s, especially in medicine this becomes really hard. And since most would agree that an AI capable of diagnosing a person with 90% accuracy compared to a doctor's 80% is better, what would the preferences be if the AI’s prediction comes from a black box? At what percentage then would individuals prefer it over a human?