Assignment 6: Game Playing Systems

Johan Östling - 20h
Hanna Olsson - 20h

February 28, 2023

1 Reading and Reflection

1.1 Hanna

The article "Mastering the game of Go with deep neural networks and tree search" describes the development of an algorithm called AlphaGo, which uses deep neural networks to play the game of Go at a world-class level. The article explains how the algorithm was trained using a combination of supervised learning and reinforcement learning, and how it was able to learn from a large dataset of expert games to develop its own strategies and tactics. The article also describes the use of Monte Carlo tree search to help AlphaGo make decisions during gameplay. Ultimately, AlphaGo was able to defeat the world champion in a high-profile match, demonstrating the potential of deep neural networks and machine learning to solve complex problems and learn from vast amounts of data.

1.2 Johan

This article was about AlphaGo and its success in beating a European world champion in the board game Go. They managed to develop a successful Go AI by implementing a Monte Carlo Tree search. The problem with a exhaustive search is that in Go there are too many different states in go, making the search infeasible. AlphaGo used deep neural networks to predict the roll-out policy of the opponent. And then they used reinforcement learning to come up with a value network, that estimates the probability of winning of a given state. When these two were done, they used Monte Carlo Tree search to determine which action to take which leads to a terminal state were the AI is the winner.

2 Implementation

For the simulation part of our MCTS, we choose the roll-out policy for our player, and for the opponent to be random. This makes our simulation much faster, and since the game of tic-tac-toe is a simple game, it is enough to use random roll-out policies. By simple, we mean that the game tree is small enough that a random roll-out policy is very likely to sample a diverse set of moves and positions, and therefore provide a reasonable estimate of the value of the node where the simulation takes place. Another good thing about choosing a random policy for our player and the opponent is that it can help to avoid bias in the search. Which will lead to the search being better at representing the range of possible outcomes in the game. We definitely could have looked into other roll-out policies to possibly improve our search, but since random should be efficient enough, and its computational efficiency, and its unbiasedness, we choose to stick with it. It is important to highlight that we believe that using random rollouts is not the best option for every game, especially more complex games with more choices, such as chess.

The selection policy in our search tree was based on the UCT (Upper confidence bounds applied to trees) score. This formula balances the tradeoff between exploration and exploitation by having one term that accounts for the relative wins, and one that accounts for the relative visits, making it more likely to select a node if it has not been visited. This formula also has a constant term, that can be adjusted to weigh the relative visit part more or less. In our case, we used the constant sqrt(2), since this gave us the best result. We also tried increasing it to make the UCT score value exploration higher, but in our case, it resulted in a worse search. We choose to use the UCT score, because of its property to balance exploration and exploitation, and also because it was easy to implement.

For the backpropagation part of the search we choose to update the nodes from where the simulation took place, up to the root node, by adding 1 if the simulation led to a win or tie, and 0 if it led to a loss. We tried a variety of different updates, for example, to give 1 for a win, 0 for a tie, and -1 for a loss. But after trying our search, we found that it was better to give a positive score getting a tie since a tie is sometimes the best you can get from a given node. We argued that there could be a problem with our implementation of the scores update because if there is a situation where our player could choose to make a move that leads to a win or tie, it will value these the same. But it is arguable that is should value a win higher. But since the assignment was to not lose, we wanted to focus on not losing and be satisfied as long as our player did not lose.

The implementation of the sampling is done in the simulation method where the game is played out at random. In an exhaustive search for example every possible state and outcome is evaluated, but this obviously takes a lot of computational power. MCTS uses simulation to limit the game tree, by only exploring nodes which the roll-out policy of your player and opponent follows during the simulation. This saves a lot of power since it does not evaluate every possible state. But it is still an efficient search because it is iterated many times, hence many states are still evaluated and the chance of MCTS missing important moves is not that likely.

One of the main pros of our algorithm is that it is simple and fast. When playing against it, it barely takes a second for the algorithm to make its move. We consider this to be due to the use of a random policy, as discussed above. However, there are some flaws in the learned policy. We notice that it fails to always block us from winning, so it does lose sometimes. On the other hand, it is very good at winning when it gets the opportunity, meaning that when it sees that it has two in a row and the third spot is free, it always makes the right move. We believe that the mentioned flaw also is due to the random roll-out policy. A better policy might be to always prioritise blocking three in a row, and only if there is no such situation, make a random move. Another pro of our algorithm is the structure in which we wrote our code. We use the four steps of MCTS explicitly, making it easy to read and debug. This follows the standard pattern of MCTS implementation and hence makes it interpretable. When scaling the algorithm to larger grids the search space grows exponentially and the number of

iterations would have to be increased which makes the algorithm slower. However, compared to an exhaustive search this scales much better as we are using sampling and not exploring the entire search space.

We have also thought of suggestions to improve our algorithm. The first is to investigate the selection policy and see if there are others that may improve the results of the MCTS. There are for example selection policies such as upper confidence bounds for exploration (UCB1) which may give better results. Another suggestion is to further investigate the exploration constant in the UCT formula to find its optimal value. There are also multiple ways to improve the time efficiency of this algorithm, by for example implementing early stopping if a certain threshold for a win rate for a node is reached. Also to use hash tables to store the game state and its corresponding nodes to improve speed and reduce memory.

Finally, since our MCTS still was able to lose, and we read online that an MCTS for tic-tac-toe with 1000 iterations should be enough to never lose, we are sure that there are improvements to be made. We believe that our search happens properly and it does not skip any important steps of the MCTS algorithm, but one may question the certainty of this since our player still loses when playing against us. We have speculated that our algorithm does not explore properly and this can depend on how the selection or expansion method is implemented. Unfortunately, we have not had the time to debug this, so we leave this for future improvements.

3 Summary and Reflection

3.1 Hanna

3.1.1 Game Playing Systems 21/2

Humans began developing machines for playing games centuries ago. Although these machines may not resemble modern ones, the underlying concept remains the same: the longer you can anticipate the future, the greater your chances of winning the game.

Zero-sum games are a type of game where one player's victory results in another player's defeat. Since games are about actions, our goal is to select actions that increase our chances of winning. We can think of the game as a branching tree, and playing optimally means choosing the best path. We can formalize this structure as a search tree, where the root node represents the start of the game, and the leaf nodes represent the game's different endings. Our aim is to reach a winning ending. Each node in the tree has a value that indicates our probability of winning. We can use a search tree to explore different possible futures and backtrack from winning leaves. However, we also need to consider our opponent's actions since they influence the game's outcome.

We can assume that our opponent follows a policy, π, when selecting actions. If we knew this policy, we could compute the probability of reaching a winning state for each position and execute the actions that have the highest success rates. However, this is an ideal setting that is usually not the case in reality.

Instead of assuming a fixed opponent, we try to find a strategy that works against the best possible opponent. We want to minimize our risk of losing when playing against an opponent who maximizes our chances of losing. This approach is called minimax optimization. However, there are some limitations to this idea. The state space can be extensive, and we may not be able to find the strongest opponent. Furthermore, if we want to win quickly, it is not reasonable to assume that the opponent will play optimally.

We can model games as decision processes that describe an agent taking actions A_t according to a policy π in states S_t observed through X_t, which transitions according to dynamics $p(S_t|S_{t-1}, A_{t-1})$. We often have a designated reward function R_t as well.

We can use a Monte Carlo search tree (MCTS) that searches based on experiments with randomly selected actions instead of exhaustive searching. MCTS involves the following process: selection, expansion, simulation, and backpropagation. We use the statistics to improve the selection and simulation policies. We need to balance exploration and exploitation, also known as the exploration-exploitation problem. A greedy policy that selects the best action based on some metric can lead to no exploration. Instead, we can use upper confidence bounds to limit our exploration to actions we know are not bad.

3.1.2 Follow up 24/2

We discussed a hypothetical scenario in which a patient with cancer must choose between a black box AI surgeon with a 90% cure rate but limited transparency, and a human surgeon with an 80% cure rate. While the AI offers benefits such as higher cure rates, increased reliability, and time savings, there are also counter arguments to consider, such as insensitivity to context, uncertainty about the origin of the cure rate, and confusion regarding who bears responsibility for the surgery.

Zachary Lipton's article highlights five key desiderata that should be considered when evaluating AI models: trust, causality, transferability, informativeness, and fairness. Causality is particularly important to consider, as it helps us understand the underlying mechanisms driving observed relationships between variables. To explore causality, we may collect samples, plot graphs of relationships, and compute densities to identify patterns of conditional independence. These observations can inform a

factorization of the joint probability distribution between variables, shedding light on causal relationships. However, this approach is not sufficient on its own. Instead, we can use causal graphical models to provide a more rigorous causal interpretation of variable relationships.

3.1.3 Reflection

During my assignment on developing a rule-based classifier, a random forest, and a custom classifier to reflect the trade-off between interpretability and performance, I had the opportunity to explore the challenges of balancing these two factors in machine learning.

As I was working on the different classifiers, I found myself reflecting on the article on interpretability in machine learning. This helped me gain a deeper understanding of why interpretability is important, and how it can impact the trustworthiness of machine learning models.

Through this assignment, I realized that developing a custom classifier that reflects a trade-off between interpretability and performance requires careful consideration of the features and parameters of the model. By experimenting with different approaches, I was able to strike a balance between accuracy and interpretability.

Overall, this assignment gave me a greater appreciation for the complexities of machine learning and the importance of carefully considering the trade-offs between different approaches.

3.2 Johan

3.2.1 Game Playing Systems 21/2

The lecture started out with a drawing of the mechanical turk from the 1700s to show that the concept of having a machine playing games has been around for a long time. Today there are AI systems such as OpenAI Five that can win over humans on complicated games such as DoTA.

There are zero-sum games: One player winning implies one player losing. To win at a game is about selecting actions to improve chance of winning, and to win, and good/bad futures from selected actions should be accounted for. This means to pick the best path to play optimally. Search trees can be used to find the optimal path which consists of nodes representing the different states of the game. The terminal nodes is the state of the game were there are a declared result of who won. We also need to formalize the value of a state, meaning that states/nodes that leads to a possible win is valued more. When the search tree is done, we can backtrack to find which actions we need to take for a good end.

To guarantee success against the best opponent, mimimax optimisation can be used. This means to minimize the maximum success of your opponent. Assume that the opponent will play the best possible move.

We can think of games as a decision process which describes an agent, taking action A, according to policy PI, in states S, observed through X, which transition according to dynamics p. Often we have designated reward R, a function of X that we want to optimize.

Minimax has some limitations such as enormous state space, doesn't care about winning quickly, doesn't create traps that worst-case opponent wouldn't fall into, but a weaker player might, and observability.

Two problems with large state spaces are that we can't explore every state and we can't store the value of each state. This takes too much time and a table storing all values would be too huge.

A solution to the exploration of every state problem is Monte-Carlo Tree Search [MCTS] that deals with random sampling. Instead of searching through all nodes, the search is based on experiments with

randomly selected actions. MCTS repeats the process of selection, expansion, simulation, and back-propagation. Selection refers to finding unexplored nodes. After selection, the node is then expanded and a unvisited child node is selected. This node is then evaluated by simulation. Meaning that we simulate the game starting from this node, and when we reach a terminal node, we then know the value. The last process is back-propagation that means to go back in the tree and update values of nodes along the way, and the expanded node is marked as visited.

To overcome the next issue of storing all values we can implement several solutions. In early systems values were manually inserted for non-terminal states. Today, machine learning is used. We let a function approximate the value. In modern AI, this function is a deep neural network. In Tic-Tac-Toe for example the history of a game does not influence transiaiton, meaning that it has the Markov property.

3.2.2 Follow up 24/2

We started of discussing interpreteability, using a tweet from Geoffrey Hinton, that asks the question if you would rather have a human surgeon with 80% cure rate that can explain how it works or a AI surgeon that cannon explain how it works but has 90% cure rate. Some arguments for was; more reliable, time saving, more surgeries, can use with surgeons. Some arguments against was; insensitive to context, where does the 90% come from, doctors more flexible. We then moves on to outline Zachary Lipton desiderata for model interpertability. These were, trust, causality, transferability, informativeness, and fairness.

Then we changed topic to causality. First we went through the connection between smoking and cancer, and let's say you fit a logistic regression to predict cancer, and you find a model. Even though this model can predict cancer from smoking, it does not mean that smoking is directly related to cancer. An example of a counter argument which has been used by tobacco companies, were that there is a gene that is in people who causes them to smoke and to get cancer, eliminating the cause of cancer from smoking. To give an example of how to figure out causality an example given altitude, horizon and temperature were explained.

3.2.3 Reflection

The previous module was interesting for me, mostly because of the important applications of the AI. It was interesting discussions regarding interpretability and it widened my perspective on the problems with AI in healthcare, and why it is hard to use AI models that are hard to interpretation in life and death situations. The algorithms we wrote were interesting, especially random forest since I have not done that before. From the rule-based classifier I learned to think about the tradeoff between sensitivity and specificity when choosing thresholds for the different features affecting malignancy and benign. Because if you want to minimise the number of false negatives, the number of false positives will increase. And that the distributions of the same feature for malignant and benign cells could be very overlapped, making it difficult to use them for the classification. Further learnings were on how to choose the features that determine malignancy or not, I used correlation map, but you could have chosen to take average over several features for cell size for example.