Data Visualization for gaming agent of Go using deep reinforcement learning

Zhongmou Cai
Santa Cruz, CA, United States
zcai34@ucsc.edu

December 14, 2020

1 abstract

In this project, I will implement an artificial intelligent agent to play the Go game, using the deep reinforcement learning algorithm. There will be a corresponding search tree(game tree)you, where every node has an expected utility. We plan to use an interactive way to visualize the process. And I will use Bootstrap, React, Angular to visualize the search tree in every move. As we know, an game agent is an efficient algorithm that can search for the optimal strategy in the game tree. The project shows how much utility the agent will get for each move, and how the search tree is built using deep reinforcement learning in the game tree. In the meantime, there will be a board displaying the player who's move it is. The well drawn circles in the middle are the places on the game board.[1]

2 Motivation

Go originated in China over 3,000 years ago. Winning this board game requires multiple layers of strategic thinking.

Two players, using either white or black stones, take turns placing their stones on a board. The goal is to surround and capture their opponent's stones or strategically create spaces of territory. Once all possible moves have been played, both the stones on the board and the empty points are tallied. The highest number wins.

As simple as the rules may seem, Go is profoundly complex. There are an astonishing 10 to the power of 170 possible board configurations - more than the number of atoms in the known universe.

For a very long time, we had been puzzled by the interpretation of the artificial intelligence. Now I try to visualize a simple AI in the Go. Our past impression about artificial intelligence is quite vague, which we regard as a black box. Yet data visualization uses algorithms to create images from data so human can understand how AI work. And in essence, an artificial intelligent gaming agent actually work by searching the optimal solution in extremely large space. Moreover, since the entire strategic space is extremely massive, it is impossible for us to search it completely.

3 Related Work

The most famous AI in Go game is the Alpha-Go invented by DeepMind Technologies which was later acquired by Google. AlphaGo and its successors use a Monte Carlo tree search algorithm to find its moves based on knowledge previously acquired by machine learning, specifically by an artificial neural network (a deep learning method) by extensive training, both from human and computer play. A neural network is trained to identify the best moves and the winning percentages of these moves. This neural network improves the strength of the tree search, resulting in stronger move selection in the next iteration.

Deep reinforcement learning[2] is the combination of reinforcement learning (RL) and deep learning. This field of research has been able to solve a wide range of complex decision making tasks that were previously out of reach for a machine. Thus, deep RL opens up many new applications in domains such as healthcare, robotics, smart grids, finance, and many more. This manuscript provides an introduction to deep reinforcement learning models, algorithms and techniques. Particular focus is on the aspects related to generalization and how deep RL can be used for practical applications. We assume the reader is familiar with basic machine learning concepts.[3]

Reinforcement learning may be subdivided into two principal categories: model-based, and model-free . Model-based RL constructs, as an intermediate step, a model of the environment.[4] Classically, this model is represented by a Markov-decision process (MDP) consisting of two components: a state transition model, predicting the next state, and a reward model, predicting the expected reward during that transition. The model is typically conditioned on the selected action, or a temporally abstract behavior such as an option . Once a model has been constructed, it is straightforward to apply MDP planning algorithms, such as value iteration or Monte-Carlo tree search (MCTS)[5], to compute the optimal value or optimal policy for the MDP. In large or partially observed environments, the algorithm must first construct the state representation that the model should predict. This tripartite separation between representation learning, model learning, and planning is potentially problematic since the agent is not able to optimize its representation or model for the purpose of effective planning, so that, for example modeling errors may compound during planning.[6]

4 Proposed Research Approach

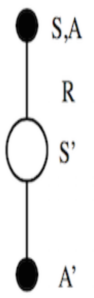
For an reinforcement learning agent, it will receive rewards(positive or negative) from the environment. since the goal of the Go game is to occupy as many points on the board as possible, we can try using the expected points as the reward for the agent. [7]

In the training process, we apply following tricks to the Go game: 1.in a limited-area find all possible legal moves 2.search in the candidate moves, find out the move with the maximum expected utility 3.in the searching process, if it is a terminal state(reach the end of the game or the depth limit of the searching process) , the utility would be the heuristic function we pick (the number of places occupied on the whole board, if it is a win, the reward would be really high) 4.collect the feature(the distribution of the grid, and the corresponding action, expected reward, actual reward) for training

the model maps the state(distribution of state) to the expected reward(utility).

To train the reinforcement learning agent, I used q-learning method and the Bellman equation to update the q-table, which stores the expected accumulative reward in the long run.

Here is the meaning of the parameters, learning rate means how much the agent consider the updated q values, and the gamma discount factor means how much the agent consider the future rewards.



$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$

And because of the state-action space are so immense that we will not be able to build such a q-table, we used deep neural network to predict the expected q value for each state. This is how we deploy deep reinforcement learning.

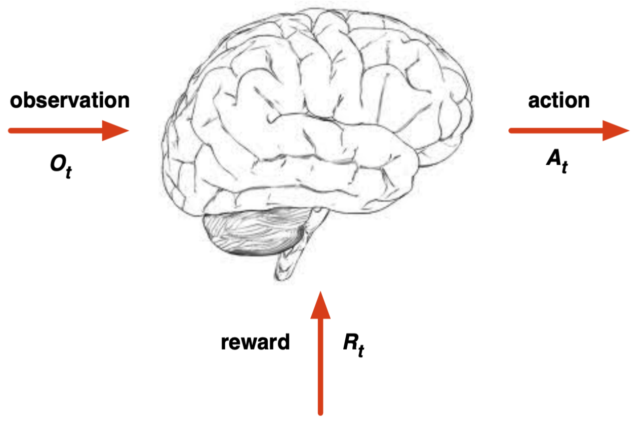
And I proposed to use D3.js to visualize the searching tree of the game agent, showing the corresponding coordinate and expected utility.

And I used tensorflow's visualization toolkit to visualize the model structure of the the deep reinforcement learning.

the training data is a tuple: (the distribution of the grid, and the corresponding action, expected reward, actual reward)

the training data will include the distribution of the grid, which is how the stones are placed on different positions of the grid, and the action that will maximize rewards, and the expected reward, the actual reward given by the environment.

and then this kind of training data will be fed into a three layer fully connected layer and a dropout layer to train a model to predict the expected reward given an action and a state.



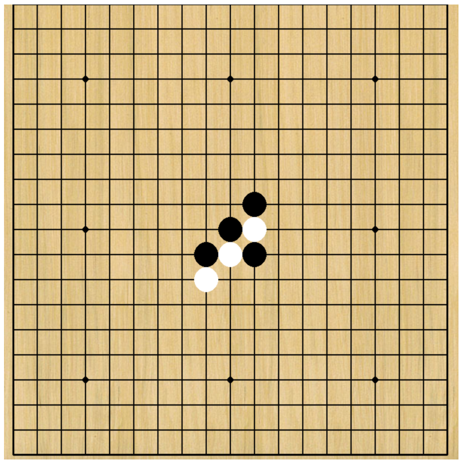
Since the entire gaming space (search space) is incredibly large, there is no way that we can search the entire space, so we can only look a few steps further. I plan to set up a few experiments to show how the number of steps we look further affect the performance of the game agent, including the time complexity and the utility. Therefore, the algorithm performs differently given different time constraints, because the steps we can look further is limited.

We can find an innovative way to display the interactive game board.

I have completed the first version of Go game.[8] Now it supports the human vs human mode. When the project is complete, it will support three modes: AI vs AI, human vs human, AI vs human.

And also I have completed the first version of AI, but it's still quite slow. I'll do some enhancement later.

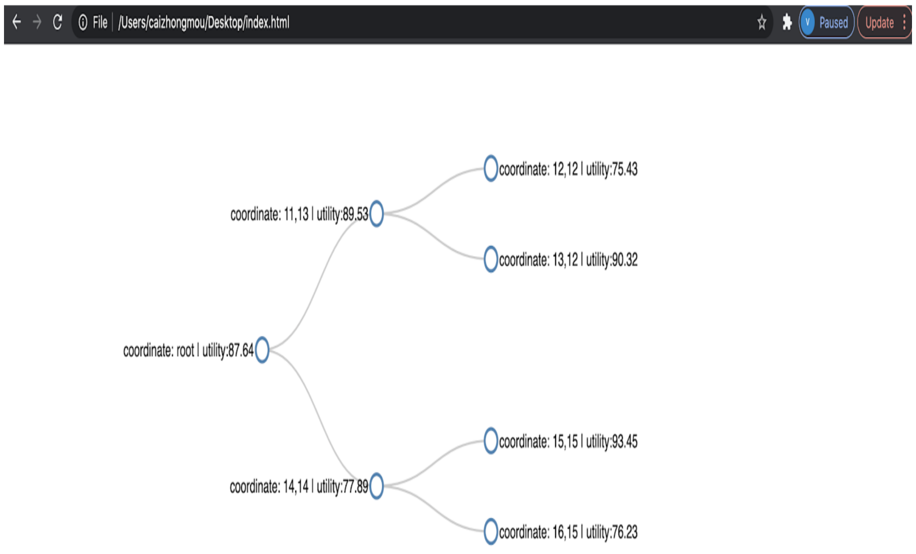
5 work completed

 I have completed the implementation of the Go game environment, now it supports human vs AI, human vs human mode as well as the basic version of Deep reinforcement learning agent for Go game.

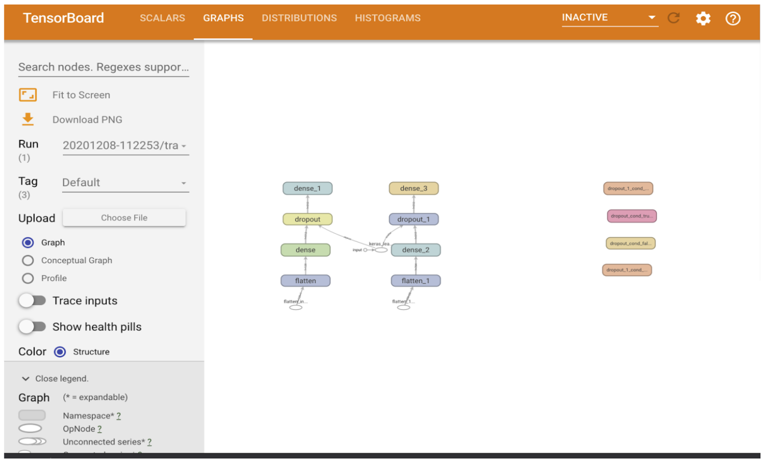
Now I use the number of points occupied by one player as the reward and the state for one agent would the distribution of its points on the board. Also I use a parameter of 0.8 for the back propagation to pass the q value back to the agent from the future.

6 visualization

For the visualization part, I've completed the visualization of the searching process and the visualization of the neural network graph structure that I used in the deep reinforcement learning. To visualize the searching tree, I saved the information needed to build a tree in a dictionary ans saved it in a json string and then dump it into a stirng, then I connect the python app with the web page using flask framework. After the web page received the json string, then it will use D3.js to visualize it and represent it as a tree.


File | /Users/caizhongmou/Desktop/index.html
Paused
Update
coordinate: root | utility:87.64
coordinate: 11,13 | utility:89.53
coordinate: 14,14 | utility:77.89
coordinate: 12,12 | utility:75.43
coordinate: 13,12 | utility:90.32
coordinate: 15,15 | utility:93.45
coordinate: 16,15 | utility:76.23

as for the visualization of the graph structure, I used the visualization toolkit in the tensorboard to visualize the graph structure of my model, which consists of three dense(fully connected) layer and one dropout layer that help avoid over fitting. As I mentioned in the paper reading about this visualization method, this visualization toolkit has following advantages: 1.the design process emphasizes understanding of both users and data. 2.the visualization tool will enable the users to define custom collapsible units by using / symbol.



Run (1) 20201208-112253/tra

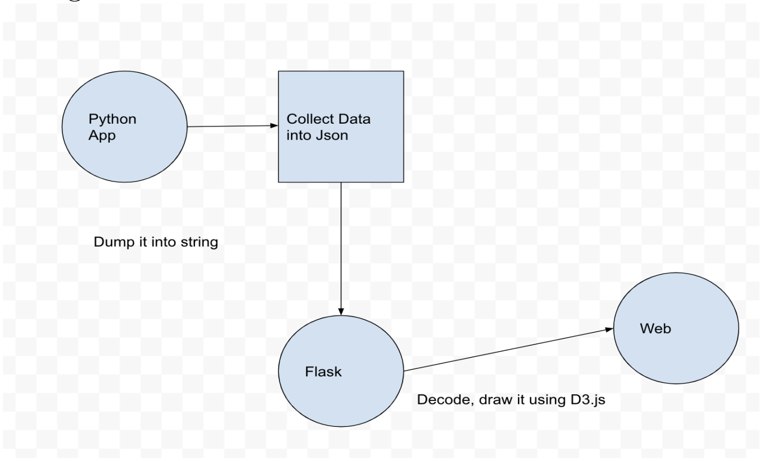
Tag (3) Default

[✓ Graph] Conceptual Graph Profile

Color [✓ Structure]

Graph

And the following framework is the structure that I used to represent the searching tree.



At first we saved the coordinate and corresponding utility and then dump it as a json string. And then I sent the json string to the web page, and then I use D3.js to visualize the searching tree.

7 conclusion

In this project, I've completed a go game app that supports human vs human, human vs AI, AI vs AI mode. And I also save completed the visualization of the searching process of the game agent which is trained using deep reinforcement learning.

In conclusion, we found that if we set the learning rate to be higher, the time that the agent's policy will be trained faster, and vice versa. Moreover, if we set the gamma parameter (discount factor) to be smaller, the agent will become more short sighted, only considering the immediate rewards. In this case, for current move the agent will consider more about how to occupy more space in the current grid instead of think more about how to will the whole game.

And I really learned a lot from this course, it make me achieve a better understanding of data visualization. It help me know more about the underlying details of reinforcement learning and some front end technology like D3.js.

References

[1] I. Srivastava, "A different take on the best-first game tree pruning algorithms," 2019.

[2] G. Brero, A. Eden, M. Gerstgrasser, D. C. Parkes, and D. Rheingans-Yoo, "Reinforcement learning of simple indirect mechanisms," 2020.

[3] S. P. Singhal and M. Sridevi, "Comparative study of performance of parallel alpha beta pruning for different architectures," 2019.

[4] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, p. 219–354, 2018. [Online]. Available: http://dx.doi.org/10.1561/2200000071

[5] M. Lanctot, A. Saffidine, J. Veness, C. Archibald, and M. H. M. Winands, "Monte carlo *-minimax search," 2013.

[6] D. Mguni, "Stochastic potential games," 2020.

[7] Y. Savas, M. Ahmadi, T. Tanaka, and U. Topcu, "Entropy-regularized stochastic games," 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.