Final Project: Reinforcement Learning for Game Playing

Yun Ho Jung   
Department of Computer Science  
University of Texas at DallasRichardson, Texas  
https://orcid.org/0000-0002-1891-5642

Kippum Chang   
Department of Computer Science  
University of Texas at DallasRichardson, Texas  
Kippum.Chang@UTDallas.edu

Eojin Kim   
Department of Computer Science  
University of Texas at DallasRichardson, Texas  
Eojin.Kim@UTDallas.edu

*Abstract*—We have constructed a Reinforcement Learning system for playing a Connect-N game, which is a turn-based zero-sum Markov game with machine learning code written from scratch. We employ two methods - Q learning and Deep Learning. Keywords—Deep Learning, Reinforcement Learning, Zero-sum, Turn-based Zero-sum Markov Games.

Introduction

The usage of reinforcement learning to create systems that perform tasks is one of the well-known usages of machine learning. Reinforcement learning systems have been used in various domains, such as robotics, economics, and distributed control systems(Buşoniu et al). The methods in reinforcement learning typically involve an evolutionary algorithm where the learner improves its performance after every evolution - which is to execute through an entire iteration of a certain task(Bäck). Reinforcement learning has been explored in both continuous and discrete state spaces, where systems with continuous state spaces are more difficult to explore due to their innate characteristics of complexity but can be more useful in applying to real-life problems(Hasselt). In our work, we explore a more simplified, discrete state space reinforcement learning system representing a zero-sum game that we have defined as Connect-N. We define Connect-N as a turn-based zero-sum game where each player places their respective ‘symbol’ onto a 2-dimensional grid, in order to achieve a continuous N length sequence of their ‘symbol’ on the grid. An example of this with N=3 on a 3x3 grid is commonly known as the game of Tic-tac-toe. Such a game is a turn-based zero-sum Markov game, which is a prime system to experiment with reinforcement learning algorithms. There are numerous active researches on the topic of turn-based discrete-state zero-sum Markov games, one of the most famous being Google’s AlphaGo and AlphaGo Zero projects, where deep learning is used to effectively create a system that has effectively ‘mastered’ the game relative to any other players, machine or human(Granter et al). Our team took inspiration from the recent explosive improvement in computational algorithms in finding optimal routes in complex zero-sum games and decided to compose our own version of a deep learning reinforcement system for finding optimal routes in the aforementioned Connect-N game. We have created a robust dense sequential neural network without relying on any external libraries such as TensorFlow, Keras, or Scikit learn to be used as the backbone of the deep learning agent that is the target of reinforcement learning process. We then have integrated our neural network into a customizable and automated version of Connect-N where the deep learning agent will automatically play against a different deep learning agent, which results in two copies of trained models that make the theoretically optimal moves in both first-going position and second-going positions. A resulting model is a python object that can be either saved or be inserted into the Connect-N game directly so that a human agent can attempt to play against it in real-time.

# Theoretical and conceptual study of the technique/algorithm

The first algorithm that we have developed for our project is Q-learning which does not require a data set to predict the result. The main idea of the Q-learning algorithm is to learn from interactions with an environment like humans do.

Q-learning works similarly in real-life situations. For instance, If an untrained dog is not behaving properly in front of a person, there must be a way to train the dog. One of the ways is to give the dog a treat as a reward if the dog stays quiet and waits for the treat but otherwise does not give the treat. The dog will slowly recognize the fact that behavior is what decides the outcome of the situation, and will be more likely to behave well to get the treat.

Similarly in Q-learning, instead of using a data set, the agent learns from all consequences of the action. What we first need in using the Q-learning algorithm is the environment to run on and the agent to be trained. In our case, during the training process of an agent (or an A.I.) to achieve the ultimate goal of creating the Tic-Tac-Toe and Connect N machine with good performance, we needed to develop an environment where agents will play the game against each other.

With the complete setup of the environment, we needed to create a reward and punishment system which guides the agent to choose optimal move every turn. Based on the result of a game, we fed reward or punishment to each action to reach the result. A reward is used to encourage the agent to perform the same action again if the action that agent has taken led to a good result. Similarly, punishment is used to prevent the agent from taking the same action if the action leads to a bad result. As the value of an action cannot be determined until the result of an action is revealed, the reward and punishment system will only be triggered at the end of an action within a state. In our case, the reward will be given to each state when the game has ended. For the reward value, we decided to give 1 point when a player wins, -1 point if a player loses (as the value of punishment increases the A.I tends to play more defensively) and 0.5 points for a tie game// change later. After the game has ended, we assigned each value to the states starting from the end state to the first state using the following equation, V(St) = V(St) + a(V(St+1) - V(St)). In the equation, V(St) represents the value of the current state, V(St+1) represents the value of the next state, and a represents value of the learning rate. The reasoning behind assigning the highest value to the most recent state is because the player’s last action contributes the most to win a game compared to the previous actions.

Using the reward and punishment given to each state, now we need to create a policy that helps A.I choose optimal moves every turn. Training policy involved two different strategies which are creativity factor and exploitation. The creativity factor is used to make the A.I explore a new path instead of taking a path based on the trained state values because if it only follows the trained path it would have a lack of ability to explore a new path that may generate a better result. Therefore, to encourage the A.I to explore a new path, setting the appropriate creativity factor rate was necessary. Another strategy is called exploitation which uses the reward and punishment value given to each state. Exploitation is a method of choosing a state with the highest value out of all the possible positions within the current state. Therefore, to make a policy that can utilize the advantage of two different strategies, an appropriate combination of two rates is required. In our case, the value that resulted in the highest A.I win rate when playing against an agent with 1 as creativity factor rate was using 0.2.

The second algorithm that we implemented was the Deep Learning based approach using neural network. The neural network is a sequential network using back-propagation that has three hidden layers where it takes the game state as the input. It returns whether or not a state is favorable to the current player or not. Hence, the neural network itself is not a generative tool, but it can act as one by generating all the possible states and comparing their values returned by the neural network and choosing the best one.

# Results and Analysis

For the reinforcement learning based on Deep Learning Algorithm, we have performed training of the model on five different learning rates and compared their results.

For the training process, we have faced off two Deep Learning Algorithm models with different symbols (X, O) but with the same learning rate, same creativity rate (the rate of making arbitrary moves to ‘learn’ from instead of using previous knowledge), and the same number of epochs.

1. The Neural Network Approach Chart, bar chart, waterfall chart

   Description automatically generated

Figure 1, lr = 0.01, creativity=0.2, training mode

Chart, bar chart, waterfall chart

Description automatically generated

Figure 2, lr = 0.05, creativity=0.2, training mode

Chart, bar chart, waterfall chart

Description automatically generated

Figure 3, lr = 0.1, creativity=0.2, training mode

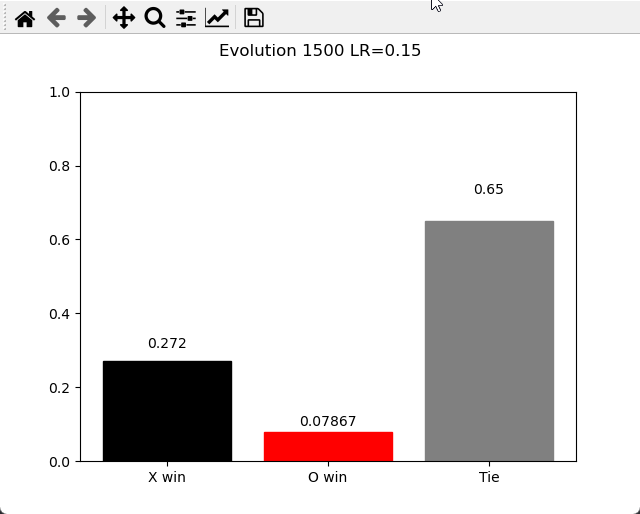


Figure 4, lr = 0.15, creativity=0.2, training mode

Chart, waterfall chart

Description automatically generated

Figure 5, lr = 0.2, creativity=0.2, training mode

One observation that can be made from the training graphs is that the rate of tie converges to around 60 percent in all the learning rate values. Connect-3, as known as Tic-tac-toe is a solved game, in which making optimal moves should always result in a tie. However, there is a 40 percent discrepancy between the optimal scenario and the training models.

This discrepancy can be attributed to the existence of the ‘creativity rate’ variable in each of the deep learning agents. Creativity rate is the rate at which each agent makes a completely random move instead of a move based on previous knowledge. This allows the neural network to learn rather than converge into a local minima state where more optimal weights could be possible.

Different learning rates have resulted in different performances. After training the models, we have put the models against a deep learning agent with 1.0 creativity - as known as the dumb agent. This agent will always make a random move, which allows a measurement of the performance of our trained models.

We have simulated 1000 games between the trained models with creativity set to 0, which is the test mode where the models will always make the optimal moves based on trained data, and the dumb agents.

The best performing X model: LR = 0.05 and 0.2.

Chart, bar chart

Description automatically generated

Figure 6. LR=0.05 X model against Dumb O Agent

The best performing LR for the X model in terms of X win rate was LR 0.05. It exhibited over 92 percent win rate against the dumb O agent, making it the highest of all five learning rates.

Chart, bar chart

Description automatically generated

Figure 7. LR=0.2 X model against Dumb O Agent

However, in terms of least opponent win rate, the LR=0.2 X model performed the best, only allowing one O victory against 999 X victory. Considering Connect-3, as known as Tic-tac-toe is a game where one cannot lose if playing optimally, it can be said that the LR=0.2 X model is the closest to optimal.

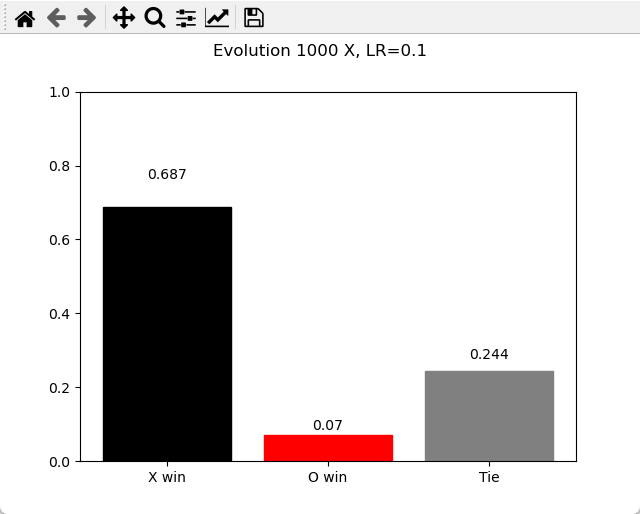


Figure 8. LR=0.1 X model against Dumb O Agent

On the contrary, LR=0.1 was the worst-performing model of all, in both the self-win rate and the opponent win rate. It can be suspected that this model converged into some local minima. It is still a good model objectively since it only allowed 70 out of 1000 O victories.

O models

For the O models perspective, we will use the X win rate as our primary metric of measurement for model performance, as in optimal Connect-3 gameplay the second going player’s goal is to tie and not lose the game.

Best O model: LR = 0.2

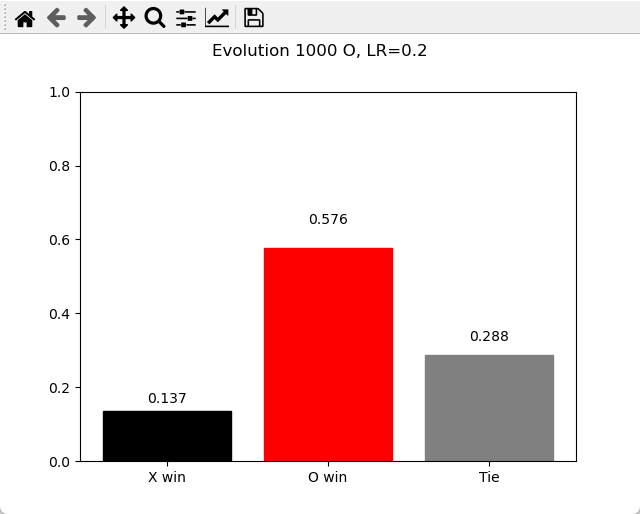


Figure 9. LR=0.2 O model against Dumb X Agent

O models did perform objectively worse than X models in terms of proximity to optimal play routes. The best performing O model was LR=0.2, and it still allowed 137 X victories compared to only one O victory in the X version of LR=0.2

It suggests that the way the training was performed with Deep Learning heavily favored the first going agent. Our future goal would be to figure out the cause of this inherent favoritism.

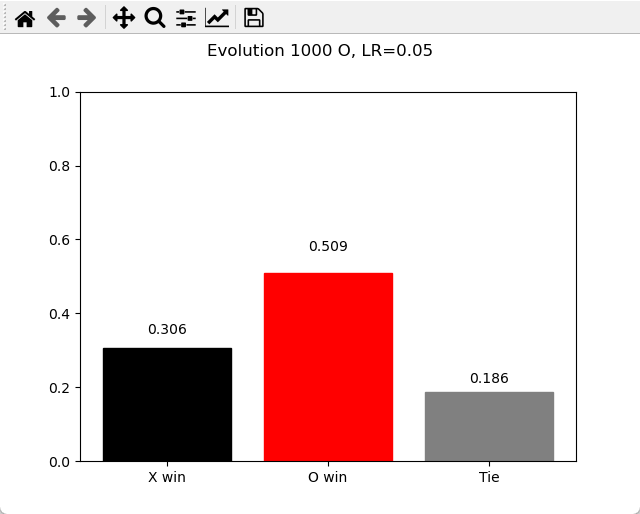


Figure 9. LR=0.2 O model against Dumb X Agent

The worst performing O model was LR=0.05, allowing almost a third of the games to be X victories. As a reminder, in an optimal scenario, X victories should not happen since there is always a way to tie the game when playing optimally. This was a surprising result because the 0.05 LR X model was the model with the highest win rate. This shows that the performance of the X models does not always correspond to the performance of the O models.

1. The Q-Learning Approach

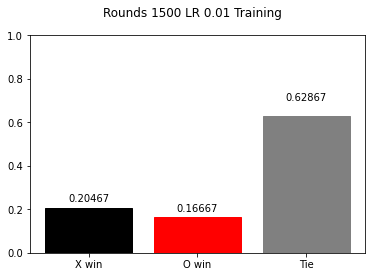


Figure 10, lr = 0.05, creativity=0.2, training mode

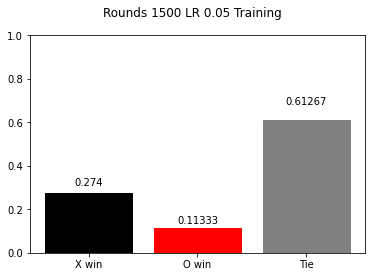


Figure 11, lr = 0.05, creativity=0.2, training mode

Chart, waterfall chart

Description automatically generated

Figure 12, lr = 0.1, creativity=0.2, training mode

Chart, waterfall chart

Description automatically generated

Figure 13, lr = 0.15, creativity=0.2, training mode

Chart, waterfall chart

Description automatically generated

Figure 14, lr = 0.2, creativity=0.2, training mode

The testing environment of this experiment is configured with the same factor values to the neural network approach, so that the observation of two different approach will have valid points in comparison.

The followings are the values set in our experiment environment:

|  |  |
| --- | --- |
| Learning Rate | 0.01 - 0.2 (0.05 increments) |
| Training Rounds | 1500 |
| Testing Rounds | 1000 |
| Creativity Rate | 20% |

As you can see in the above diagram (Figure 10-13), we ran the program with varying learning rates and a static creativity rate. Through testing our program with different combinations of learning rate and creativity rate, the optimal value of creativity rate value turns out 0.2 or 20 percent. Therefore, we conducted an experiment testing the performance of our agents with changing learning rates ranging from 0.01 to 0.2.

From the result of the training process of our agents, we have observed that player O has a lower chance of winning in the neural network algorithm. In the same sense, the chance of a tie game turned out to be very similar which happens to be around 55% of the game.

Chart, waterfall chart

Description automatically generated

Figure 15. LR=0.05 X model against Dumb O Agent

In terms of the agent O’s winning rate as the key performance indicator, the agent with a learning rate of 0.05 performed the best, only allowing the O agent to win 9 out of 1000 games.

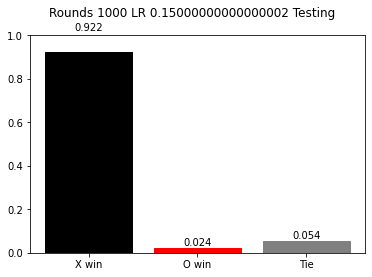


Figure 16. LR=0.15 X model against Dumb O Agent

In terms of agent X’s winning rate as the key performance indicator, the agent with a learning rate of 0.15 performed the best, winning 922 games out of 1000 games.

Chart, bar chart

Description automatically generated

The testing environment is configured in the same way as the neural network section.

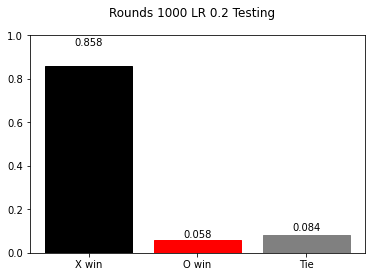
From the above bar graphs (Figure 15-16), both agents with learning rate of 0.05 and 0.15 performed well, winning more than 80 percent of the games. However, the performance is in uptrends until the rate reaches to around 0.25. After that point, the agents start to perform worse as the learning rate increases furthermore.

Overall, all our agents with learning rates below 0.25 performed very well and either won or tied against dumb O agents more than 90 percent of the time. Specifically, 0.3 learning rate was the point where the agent O showed more than 10 percent of increase in losing game.

1. Comparison of Two Approaches

Chart, bar chart

Description automatically generated



According to the graphs above, both q-learning and neural network algorithms have successfully trained the policy, however, we were able to observe slight differences between the two algorithms when training both on exact the same environment which is using 0.2 as the learning rate and 0.2 at exploration rate.

The first difference we have noticed is that the win rate of the O (the dumb agent who goes 2nd), is slightly higher in the Q-learning algorithm than NN algorithms and the second difference is that the win rate of X which is the agent we have trained is also higher when using NN algorithm. Using these differences, we were able to come up with a hypothesis that NN tends to perform better when the number of training rounds is relatively small and the reason behind this is the way each algorithm works. The Q-learning algorithm decides the next move by referring to the values of the state which out any adjustment of weight, however, in the NN algorithm we set the epochs to 10 which means that instead of deciding the next move directly we adjust the weight by traversing the neural network for 10 times to decide the most optimal next move. This does not necessarily mean the NN-algorithms is a more appropriate algorithm to train the agent for tic-tac-toe because Q-learning algorithms show a higher performance when it is trained 30000-50000, but when training rounds are relatively small, NN-algorithms performs more efficiently in terms of compile time.

# Conclusion

In our work, we have demonstrated a successful implementation of a reinforcement learning system to optimally play a Turn-based Zero-sum Markov Game. We have used two methods- Q learning and Deep Learning - and did not rely on external libraries in constructing the machine learning frameworks. We have compared the performance of the methodologies based on various learning rates and have analyzed their performances and their implications. We have observed that some parts of our implementation do follow the theoretical calculations, while some parts did not.

In the future, we would like to do the following:

1. Increase the dimension of the board and implement an optimization method because currently, the complexity of the computation increases exponentially.
2. Analyze the issue of local minima in deep learning models that occurs for certain learning rates.
3. Exploration of other algorithms for the implementation of reinforcement learning - ex) different types of neural networks.

##### References

1. Buşoniu L., Babuška R., De Schutter B. (2010) Multi-agent Reinforcement Learning: An Overview. In: Srinivasan D., Jain L.C. (eds) Innovations in Multi-Agent Systems and Applications - 1. Studies in Computational Intelligence, vol 310. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-14435-6\_7*)*
2. Bäck, Thomas. Evolutionary algorithms in theorie and practice. 1994.
3. van Hasselt H. (2012) Reinforcement Learning in Continuous State and Action Spaces. In: Wiering M., van Otterlo M. (eds) Reinforcement Learning. Adaptation, Learning, and Optimization, vol 12. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-27645-3\_
4. Granter, Scott R., Andrew H. Beck, and David J. Papke Jr. "AlphaGo, deep learning, and the future of the human microscopist." Archives of pathology & laboratory medicine 141.5 (2017): 619-621..