CS230

Re-L Catan: Evaluation of Deep Reinforcement Learning for Resource Management Under Competitive and Uncertain Environments

Chris C. Kim
Department of Computer Science
Stanford University
`chankyo@stanford.edu`

Aaron Y. Li
Department of Computer Science
Stanford University
`aaronli9@stanford.edu`

1 Introduction

In recent years, Reinforcement Learning (RL) has been increasing in popularity for their novel ability to solve large complex problems with relatively limited information. As a variation of supervised learning, RL architectures are unique models, or Agents, that learn to evaluate a given State (s) of its provided Environment and create decisions, or Actions (a), to solve problems by continuously receiving reward and penalty scores for each Action incurred in the Environment. Often modelled as a Markov Decision Process (MDP), RL models does not require information on previous States to decide on an action, which is quite practical for responding to unfamiliar environments and situations. In particular, RL models are highly interesting because they are powerful in solving dynamic problems with relatively low pre-existing knowledge of the environment. As such, popular applications of RL are found in designing artificial players that can outperform humans in games, such as Poker and GO [1, 7].

Following similar inspiration, our team seeks to design a deep neural network-based RL model that can outperform human performance in the popular board game *Settlers of Catan*. While RL models are not fundamentally required to implement a neural network architecture, many real-world problems often contain complex state spaces that result in large feature vectors that render traditional RL methods to be computationally expensive. As such, most modern RL architectures now implement a form of neural network model in addition to the basic RL framework, known as Deep Reinforcement Learning, to generate highly predictive models within computationally reasonable parameters.

In regards to the game, *Catan* is a resource driven game in which players roll dice to collect resources, trade materials with other players, and compete to be the first to build the most structures. Our group chose to explore *Catan* because its fundamental gameplay is unique compared to other conventional strategy games. In traditional turn-based games, such as Chess and Go, the game rotates actions between players who make independent decisions unaffected by other players within the same turn. In *Catan*, however, a player's turn is highly guided by both randomness and the whims of other players, who can freely intervene in the current player's turn through the game's trade mechanism. As such, this infusion of randomness, cross-player intervention, and traditional independent decision-making promotes *Catan* as a uniquely complex and interesting game to win. In particular, our group believes that developing a viable Deep RL model to play *Catan* will be a novel yet fascinating problem to evaluate how machine learning models can perform relative to human players in loosely regulated game environments, such as trading or bartering.

In this study, our group utilizes Deep RL, and specifically Deep Q-learning (DQN), as the primary neural network architecture, as the game *Catan* itself is very dynamic turn-based game that requires relatively low prerequisite knowledge to perform well. The input for our model is quite extensive and will include numerous features - including number of points per player, number of resources for each player, number of buildings created by each player, current dice roll, availability of space on the game board, number of special ability cards per player, etc. - that are extracted and compressed into a singular vector from a custom *Catan* simulation API. Once extracted, this feature vector will be inserted through the Deep Q-Learning architecture to output the next turn action the Agent should take - such as building structures, drawing ability cards, trading resources, etc.

CS230: Deep Learning, Autumn 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

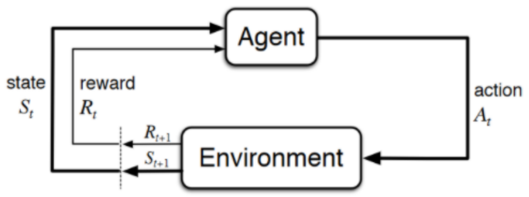

Figure 1: A graphical representation of the Agent-Environment interaction in a Markov decision process by the Agent.

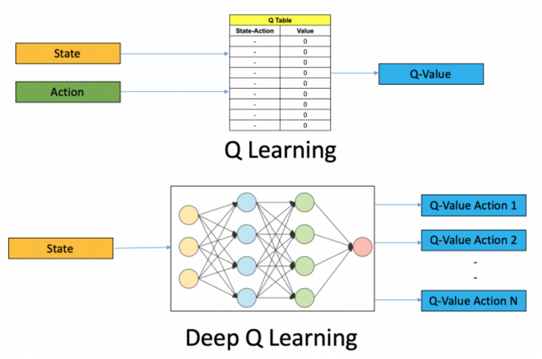

Figure 2: A graphical representation of the Q-Learning and Deep Q-Learning paradigm for each State-Action, Q-Value and State, Q-Value Action pair respectively.

2 Related work

2.1 Q-Learning Algorithm

Q-Learning algorithms are the most popular and commonly used model for RL implementations. The algorithm employs the Agent to interact with the Environment to learn the Q-values for each State-Action (s, a) pair [12]. These Q-values then determine the optimal Action an Agent should take given a particular State. The primary advantage of Q-Learning is that the MDP property allows the model to maximize the expected value of the total reward over all successive steps with only information on the current State [3, 8]. A significant disadvantage of Q-Learning models is that the algorithm can become intractable due to the high complexity of the targeted problem space. This results from the fact that traditional Q-Learning requires information on all possible combinations of State-Action pairs and their associated Q-values in the designated Q-Table, which can prove to be computationally expensive [3, 9]. For a high dimensional environment such as *Catan*, it would be less favorable to employ traditional Q-Learning as the primary solution.

2.2 Deep Q-Learning

Deep Q-Learning models offer a robust solution to the intractability of the traditional Q-Learning algorithm by applying the Q-Table paradigm onto a neural network model. In this version, the neural network maps the input State of the Environment to each Action-Q-Value pairs [5]. Interestingly, this model is often used in conjunction with Experience Replay, or a second DQN model, that acts as a set of memory weights and biases of previous episodes to help train the active model [6, 11, 13]. The primary disadvantage of Deep Q-Learning is that the duration for training the model is often significantly longer than those of tractable Q-Learning problems [9, 10]. For a complex problem space environment such as *Catan*, it is likely to take longer duration for training iterations.

2.3 QSettlers

Previous research work with similar objectives was endeavored by the QSettlers team led by Peter McAughan in 2019. In this work, the team attempted to similarly apply the DQN paradigm to develop an AI model to play and win *Settlers of Catan*. Unfortunately, the team was unsuccessful at implementing a working DQN model for general gameplay, but was able to train and develop a working DQN model specifically for the player trading mechanism of the game [2, 4]. Our team recognize this previous group's work as exceptionally useful for our purposes and seek to incorporate their described model as a module within our proposed model for general gameplay.

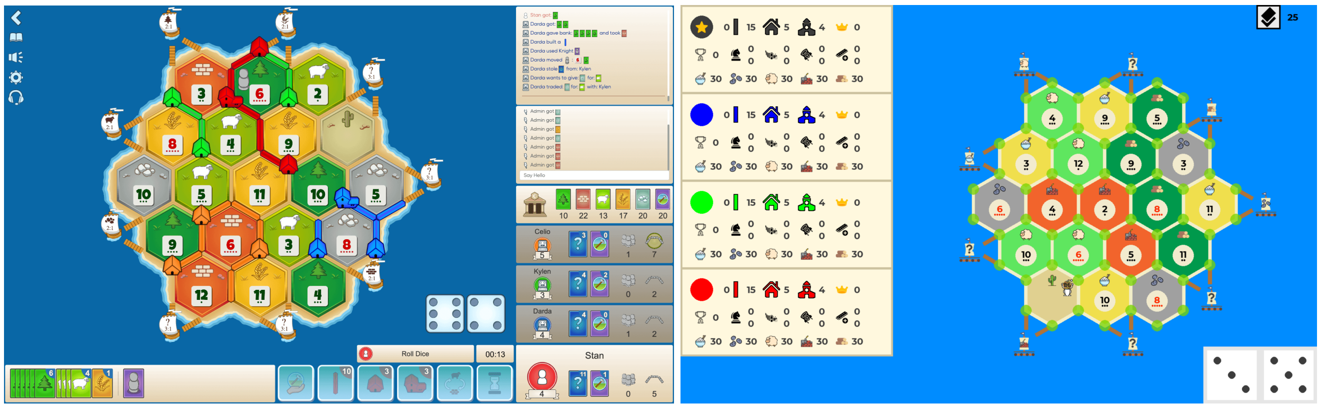

Figure 3: Left: Snapshot of *Colonist.io*, an online website-based *Catan* API. Right: Snapshot of our team's game engine of *Catan* from scratch, with a customized GUI to help train the RL model. This represents one game Environment State.

2.4 Settlers of Catan Bot (CS 221 Autumn 2014)

Previous research work with similar objectives was also endeavored by Sierra Kaplan-Nelson and his team in 2014 for their CS 221 final project. Their work differs from ours in that Nelson primarily used a basic neural network to train their models whereas our group intends to primarily use DQN models [14]. As such, it is unknown if their model is able to compare the expected utility of available actions, like DQN, to decide on future actions. There are two strengths of this work that we applied to our own project. First, Nelson's team would replace the weights of the three underperforming models with the weights of the highest performing model to improve training speed. Second, Nelson's team applied heuristics to the gameplay of their Environment to help reduce its dimensions and its complex problem space to help increase training time through [14].

3 Dataset and Features

Due to the Offline RL learning paradigm, our project requires our Agent to able to collect and store data on a previous turn to train and to actively interact with the *Catan* game, or the environment [10]. To clarify, while the Agent does not require information of a previous State for deciding an Action, but does require information on the previous State for network training. At first, our team attempted to contact the developers of numerous popular online versions of *Catan*, such as *Catan Universe* and *Colonist.io*, to perchance use their existing API for this project. Unfortunately, the developers of these third party APIs were unwilling to share their source code nor had available labeled data for potential supervised fine-tuning. For this reason, our team had to default to creating our own *Catan* game engine that would allow our model to directly interact with the environment as well as cache datasets on the game state of previous turns (See Figure 3).

To train our Deep RL model, our team initializes four CPU players within our custom game engine that will each use our model to generate a decision for their given turn. This will allow our team to collect four State data points within a given rotation; the number of rotations depends on the number of turns it takes to win the game. In order to use the model, the CPU must generate a feature vector containing target features from the current state of the game . Since our team is using a custom API, not much pre-processing work is required and our engine contains a pre-built code to selectively include desired features - such as number of players, number of resources per player, points per player, etc. - into a ready-to-use 1D input vector for our model. This functionality is quite important, as highlighted in the Methodology section below, as our team decided to use multiple Deep RL learning models that will each specialize in the distinct game phases of a player's turn, which may require different feature vectors that include specific variables while excluding others.

As a requisite of the RL learning paradigm, our algorithm will actively cache the previous game state vectors, weights, and biases. This will also serve as a useful back-up dataset to retrain the model. On a final note, we would like to mention the possibility of incorporating a Convolutional Neural Network (CNN) model to our current work due to the availability of a GUI in our game engine. While we did not implement a CNN in this project, we believe that a CNN with the vectorized GUI could improve performance.

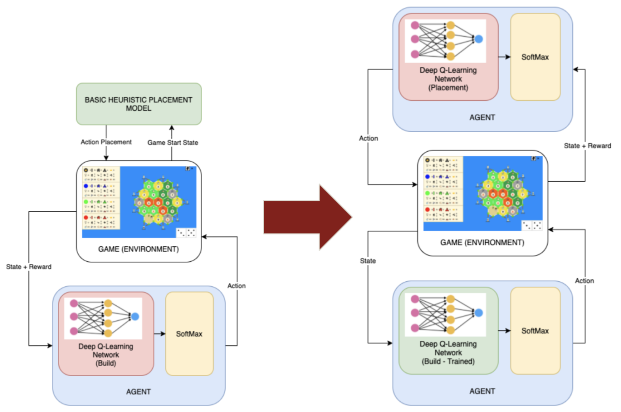

Figure 4: Left: Training model for training the DQN Build Network where initial game piece placements are made using a basic heuristic model. Right: Training model for training the DQN Placement Network in which the fully-trained DQN Build Network is used to perform actions based on placements made by the training model.

4 Methods

Q learning for placements requires us to have a known end state that we can assign a reward for, which means we need to know whether a placement wins or loses upon being played. That required us to have a model that builds reasonably well during the game. In order to have a model that builds reasonably well during the game, we needed a model that places reasonably. To resolve that chicken and egg situation, we first implemented a more basic heuristic model for placement. This model included a large expression, parts of which rewarded high production, parts of which rewarded resource diversity, and parts of which rewarded coordination between synergistic resources (ex. wood + brick, ore + wheat, ore + wheat + sheep). Using that heuristic placement, we were able to generate reasonable board states with which to start training a DQN for turn actions. For the turn action DQN, we represented the input state as a feature vector of important numbers in the game, like different players' victory points and development cards remaining. We assigned rewards to different (old state, new state) pairs, so that every action had some small positive or negative reward, and a winning state would get a large reward. We stored the most recent 500 of these pairs into an experience replay buffer, so that our model could recall them for training. We then wanted to feed that result back into an improved placement model, as we could now play a placement out to completion and therefore assign a placement a Q value, but we didn't have time to do so. (Refer to figure 4)

For this project, our team proposes to use the Deep Q-Learning algorithm as the framework for our learning models. Deep Q-Learning is a variation of the Q-Learning algorithm that implements a deep neural network to reduce the computational stress from high-dimensional problem states. In brief, Q-Learning is a simple algorithm that uses a Q-table to map an environment state and the agent's actions to the expected reward of the action given the state. Throughout the training process, the algorithm attempts to learn the Q-value, or the maximum expected reward an agent can receive for an action given a state, for each action and environment state, or exploration. After learning the Q-values, the agent at a given state maximizes the expected reward by taking on the action with the highest reward, or exploitation [12].

The difference between Q-Learning and Deep Q-Learning is that while Q-Learning will require the agent to search through all actions for a particular state and choose the best state and action pair, Deep Q-Learning will estimate the probability distribution of all possible actions that the agent can make, similar to a softmax algorithm, to be more computationally efficient. Furthermore, Deep Q-Learning has an onboard neural network, instead of a table, that will map the input game state to its corresponding action and Q-value pair as the output. This is especially favorable for our project because the Catan game contains a large number of features ranging from tiles, nodes, edges, development cards, resources, dice rolls, and more. As such, it is preferable to use Deep Q-Learning over traditional RL methods. In our project, our team proposes a machine learning paradigm that consists of multiple Deep Q-Learning architectures that will be specialized for specific phases of a player's turn. For instance, a turn in Catan can be theoretically split into multiple phases including Placement, Building, Trading, Development Card Usage, and Theft. We chose this design because one phase of a player's turn may require different types of variable information than another phase of the turn. For example, initial placements of buildings may require information on specific resources on the board, but stealing resources from another player will require more information on the other player's hand rather than the board itself. Thus, instead of using one Deep Q-Learning algorithm for the entirety of the game, we speculate that we will optimize performance by using a conglomeration of multiple Deep Q-Learning networks that specialize in their specific phases. Furthermore, splitting into multiple models allows each model to have smaller and more specific feature vectors for greater accuracy and faster computation.

$$Q(S_t, A_t) = (1 - \alpha)\, Q(S_t, A_t) + \alpha * (R_t + \lambda * max_a\, Q(S_{t+1}, a))$$

Figure 5: The Bellman Equation describes the primary training strategy to update the DQN neural network weights

Model	Win Percentage (%)
Basic Placement with DQN	32
Basic Placement with Random Legal Actions during Turn	18
Random Placement with DQN	10
Basic Placement with DQN, No Experience Memory	24
Basic Placement with DQN and Bad Trading	20
Basic Placement with DQN, No Dropout Regularization	18

Table 1: List of different learning architecture trained and evaluated based on game win percentage (%). Different learning architectures vary in network models and hyperparameters

5 Results and Discussion

We decided to use Adam optimization to train the DQN. We also went with a 20 percent dropout regularization after each hidden layer. That amount of dropout helped us prevent overfitting on the few thousand particular boards that our model looked at.

For our experience replay buffer batch size, we decided to use 500 (old state, new state) pairs (500 turns of gameplay) because we wanted our memory to be reasonably wide in order to discourage correlation between turns that are close together (ie 1 or 2 orbits). 500 turns spans several games played on the same board, so it can cover many possible combinations of random dice rolls and the resulting actions without causing too much memory overhead.

We had our model play 50 games against bots on colonist.io, and we were able to win 16 games, a 32 percent win rate, 7 percent better than the expected win rate for a Catan game. Moreover, this is with a discontinued trading model, as the one we had was giving too many key resources away in certain instances. With an improved trading model, the win rate can only go up, as trading is almost always advantageous.

6 Conclusion/Future Work

Our current DQN for building performed well against bot players. The trading model often gives opponents too many useful cards in exchange for accomplishing a small objective, likely because our current reward function mostly prioritizes the acting player's game state rather than it's game state relative to the other players'. In the future, we would like to improve the way our model evaluates other agents' positions in the game, in order to improve the trading model. This would also open up interesting possibilities for placements and other actions like deciding what to build and where. Further, we'd like to improve the quality of the initial placement model and carry out better hyper-parameter tuning for our models across the board.

We would also like to address a few challenges we had during this project. We recognize that Deep Q Learning is excellent for single agent environment problems, whereas the problem we are attempting to tackle is multi-agent environments with each agent with different goals. Google's DeepMind Alpha GO uses a CNN + MCTS approach instead of Deep Q-Learning for this reason. We can take inspiration and do this to potentially improve our model performance [11].

7 Contributions

Each member of this team has made significant contributions to the overall project while also making specific contributions towards particular sections of the project.

Chris Kim led the development of the team's custom simulation API for *Catan* from scratch, assisted in hyperparameter research and fine tuning, management of the group GitHub repository, and was the primary author of the final research report.

Aaron Li led the development of the Deep Q-Learning and standard neural network models, code cleaning, and was the primary author of the group's video presentation.

References

[1] Anthony, T., Tian, Z., Barber, D. (2017). Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv*:1705.08439.

[2] Dabney, W., Rowland, M., Bellemare, M. G., Munos, R. (2018, April). Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

[3] Jang, B., Kim, M., Harerimana, G., amp; Kim, J. W. (2019). Q-Learning Algorithms: A comprehensive classification and applications. IEEE Access, 7, 133653–133667. https://doi.org/10.1109/access.2019.2941229

[4] McAughan P, Krishnakumar A, Hahn J, Kulkarni S. QSettlers: Deep Reinforcement Learning for Settlers of Catan, from https://akrishna77.github.io/QSettlers/

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wiestra, D., amp; Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. NIPS.

[6] Nagabandi, A., Kahn, G., Fearing, R. S., Levine, S. (2018, May). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *In 2018 IEEE International Conference on Robotics and Automation* (ICRA) (pp. 7559-7566). IEEE.

[7] Schaul, T., Quan, J., Antonoglou, I., Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv*:1511.05952

[8] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., amp; Hassabis, D. (2016). Mastering the game of go with deep neural networks and Tree Search. *Nature*, 529(7587), 484–489. https://doi.org/10.1038/nature16961

[9] Stooke, A., Abbeel, P. (2018). Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv*:1803.02811.

[10] Towards Data Science. (2018, August 5). *Applications of reinforcement learning in Real World.* Medium. Retrieved December 3, 2021, from https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12.

[11] Tucker, G. (2020, August 20). *Tackling open challenges in offline reinforcement learning*. Google AI Blog. Retrieved December 3, 2021, from https://ai.googleblog.com/2020/08/tackling-open-challenges-in-offline.html.

[12] Wang, M. (2021, October 3). *Deep Q-learning tutorial: Mindqn*. Medium. Retrieved December 3, 2021, from https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.

[13] van Hasselt, H., Guez, A., Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1). Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/10295

[14] Kaplan-Nelson, S., Leung, S., Troccoli, N., (2014) *An AI Agent for Settlers of Catan* https://github.com/skleung/cs221/blob/master/progress.pdf