CS181 Final Project: Lux AI

Xiaohan Wu 2019533093 wuxh@
Jiankun Ni 2019533092 nijk@
Yucong Chen 2019533079 chenyc@
Jiaheng Han 2019533155 hanjh1@
Pengyu Ji 2019533156 jipy@

Abstract

In our project, we focused on Lux AI, an AI programming game competition. We proposed three models: rule based agent, reinforcement learning agent and imitation learning agent to compete with each other. We will walk through each model, introduce basic ideas and designing details of them and finally evaluate their performance.

1 Introduction

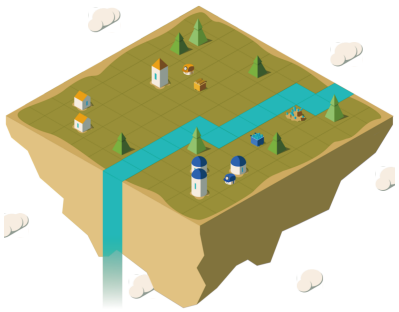
Lux AI is a turn based strategy game where both two players need to make decisions for game components-workers, carts and citytiles on the game grid of different sizes and resources distribution, as illustrated in Figure 1. The basic idea of this game is gathering resources to build citytiles during daytime and carry your team through the night. The player with more citytiles at the end of final turn wins the game.
In the daytime, units(workers and carts) are arranged to make actions according to player's policy. They can move, transfer resources, pillage or build citytiles based on cooldown mechanism and automatically receive resources from adjacent tiles at the end of each turn. As workers consume resources to build citytiles and citytiles build units as feedback, the team grows bigger but in the last 10 of every 40 turns, both units and citytiles should consume fuel converted from resources to survive the night.
As an AI programming game competition, Lux gives us the Lux design specification. We need to enhance the given baseline policy to achieve better results. Considering rules above, we take three models: Rule based Agent, Reinforcement Learning Agent and Imitation Learning Agent:

1. Rule Based Agent: Manually extract common experiences from the rules and teach the agent to win the game. The idea is simple while it does not guarantee a good performance.
2. Reinforcement Learning Agent: Due to the large state space and dynamic action space, a multi-agent version of Deep Q Network(DQN) is introduced to replace traditional Q learning.
3. Imitation Learning Agent: Collect data of replay data from experts and imitate their winning strategy. This is a model which can quickly achieve good results, but there is an upper limit of its performance.

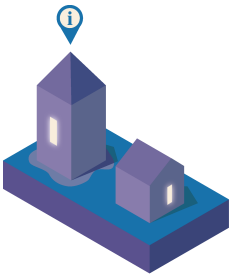
2 Rule based Learning strategy

The basic logic followed by rule based agent is collecting resources and building citytiles when it is not close to the night. Instead of an aggressive winning strategy, it is a stable one which pays more attention to self-development. We also designed several heuristic functions to make our team more tolerant to the night.

Preprint. Under review.



(a) Game Board and Units



(b) Cititiles Building

Figure 1: A Game Competition: Lux AI

2.1 Cititiles Building and Location Optimization

Since units gather resources automatically from adjacent tiles at the end of each turn, the location and moment picked by units to build citytiles are really important. Our `safe_to_build` function will determine whether it is a good opportunity to build citytiles based on total number of fuel and size of cities. Once the unit decides to build a citytile, it will choose the closet empty tile which is adjacent to a existed citytile to generate city agglomeration which is helpful for further development.

2.2 Pathfinding Optimization

Once a unit enters a citytile, the resources collected by this unit will immediately be changed into fuel for this citytile even if they don't belong to the same team. Moreover, the cooldown mechanism requires the actions of units to be efficient. In our `find_path` function, we will return the best path from start to end given a list of positions to avoid. It greatly improves units' efficiency and avoids making meaningless mistakes.

3 Reinforcement Learning

The most challenging part of LuxAI is its complex game rules and evaluation metrics. There are three kind of resources, each of which has its own distinguishing feature, and there are two kinds of units. You need to consider sustainable development as well as make your city grow bigger, otherwise your city can not survive once resources are consumed. Also, the state space of the game is huge so we need generalizing across states. These complex rules and strategies make it hard for us to hand craft features and evaluation functions, which are needed in feature based Q-learning and adversarial Search. After these considerations, we choose deep Q-learning (DQN) as our main algorithm.

3.1 Introduction

Q-learning is an efficient algorithm to solve a sequential decision problem. However, Q-learning needs to store a Q-value table, which grows linearly as the number of state-action pairs. To solve this problem, feature-based Q-learning uses weighted sum of feature functions to estimate Q-values. The weights of feature functions are optimized to minimize the bellman error. While it makes Q-learning more practical, it is usually hard to design good feature functions to make Q-learning perform well. However,compared with Q-learning which requires hand craft feature functions, deep Q-learning use a neural network to estimate Q-value, the parameters of the network are optimized to minimize the bellman error by gradient descent.

$$L_i(\theta_i) = E[(r + \gamma \max_{a^i} Q(s', a^i; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{1}$$

$$\frac{\partial L_i(\theta_i)}{\partial \theta_i} = E[(r + \gamma \max_{a^i} Q(s', a^i; \theta_i^-) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)] \tag{2}$$

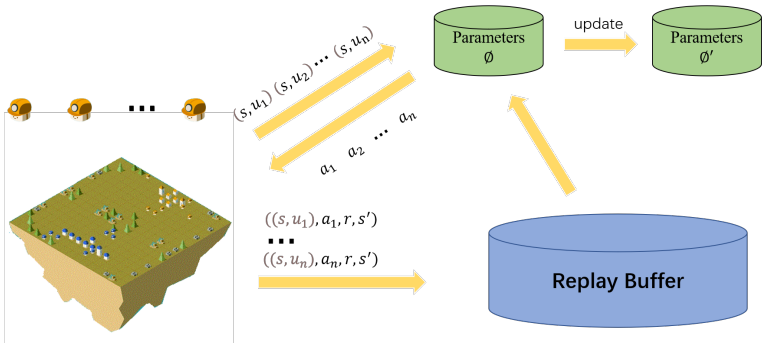


Figure 2: Reinforcement Learning

3.2 Implement Details

Our DQN agent takes a feature map as input and use a convolution network to get the Q-value of each action. During training, our agent have a possibility of ϵ to act randomly and $1-\epsilon$ to take the optimal action given estimated Q-values. We also implement a replay buffer for training efficiency. To make our DQN agent start efficient training quickly, we implement reward shaping due to the sparse reward of the game.

3.2.1 Neural Network

There are two major problems to design the neural network of our DQN. The first is that this is a multi-agent problem and the amount of action we need to take differs during the procedure of the game. Each turn new units may be created and old units may disappear if we running out of fuel and so do the cities. To handle this problem, we define our Q-value on each single unit, that is, each turn each unit will act independently given all state information.
Another problem is that in the original competition, the map of each game will have a random size from 12*12, 24*24 to 32*32. A fully convolution network can handle the various map size, but for implement simplification, our network still have some fully connect layers. So we do both training and testing on random map with size 32*32.

3.2.2 Feature Map

We will change the observation into a 16-channel feature map as the input of neural network. For 1 to 10 channel, it encodes some information directly from the observation for each specific position, such as the resource amount, the cool down and resource amount gathered by the unit, the fuel and lightkeepup of city, the level of road and so on. For 11-15 channel, it encodes some higher level information, such as the density of woods, workers, carts and citytiles around. Finally, we will encode the position of the unit which need to make an action for this iteration.

3.2.3 Designed Reward

Just like Q-learning, a DQN agent can continuously improve itself to gain larger reward and perform better strategy when it interacts with the environment. However, the environment, LuxAI game, will only give the agent a reward when the game is over. The reward is so sparse that the agent will never success to get any reward unless it survives a game. To sovle the sparce reward problem, we design a reward function that helps the agent keep making some progress. The reward function is simply a weighted sum of the game statistic information.

4 Imitation learning strategy

4.1 Introduction

Imitation learning, a widely used technique in sports analytic, racing game and basketball trajectories, aims to mimic how human behaviors given a specific task. Compared with reinforcement learning

which requires specific reward function, Imitation learning tries to learn a policy, namely a sequence of actions, given the demonstration from experts. Therefore, one of the advantage of using imitation learning seems obvious,that is, it's easier for an expert to delineate the desired behaviour instead of the specific reward function.

For this project, as all the replay data about how high-score teams play can be acquired from script provided by Kaggle, it's feasible to introduce imitation learning into our project regarding those data as experts' demonstration. To be more specific, given those replay data which includes resources, workers attributes,cities and so on, we can train a network that predicts how each worker should act at each time step. In the end, the network will mimic how those high-score teams act to play the game,and can even achieve a better performance if it learns to combine merits from several teams.

4.2 Dataset generation

Kaggle provides an API called GetEpisodeReplay to download episodes from the Meta Kaggle dataset refreshed daily. We use such API and filter out teams with low score and finally got 379 episodes with 349821 samples.Then we split the dataset into training and validation dataset respectively,following the size ratio 9:1.

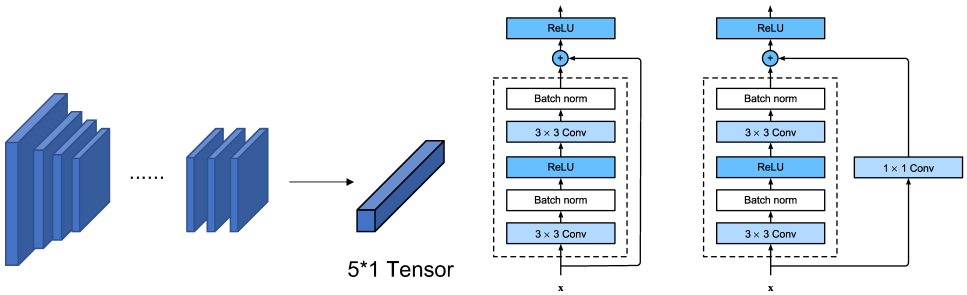
After that, we map those samples into a state space to generalize the input of the training model. Each state represents information at each time step,which is stored in a 20*32*32 matrix. Here,20 means features that we set manually,such as the existence of our(opponent) unit, the amount of wood,coal and uranium, and our(opponent) unit cool down degree etc.32*32 means the maximum map size.

4.3 Model design

For this supervised learning task, we need to generate actions for which the input is a state from the state space introduced above. There are many frameworks of machine learning to deal with such types of problem, including SVM, Decision Tree, MLP etc. However, to compress information and abstract features from such high dimensional inputs, the DNN (Figure 3(a)) takes the top spot. With the strong flexibility of DNN, it is able to model very complex mappings from state space to action space.

A typical Lux unit takes actions on the gameboard, so that the states of its neighbors will significantly influence the unit's behavior. To this end, we should make good use of this kind of locality to estimate optimal policies. CNN is naturally applicable to this scenario since convolutional kernels could help to capture local features from the high dimensional state space. For the same reasons of using CNN to deal with images, we can migrate it to the application here with no modification by treating the grid as pixels.

The model we design (LuxNN) is based on ResNet [1], which is illustrated by Figure 3(b). ResNet blocks help our model easier to train and optimize the feature represents.



(a) Use DNN to Generate Policies (b) Basic Residual Blocks

Figure 3: Imitation Learning by Neural Networks

The architecture is straight forward. We pile up 36 convolutional layers followed by one fully-connect layer and one softmax layer at the end. For each of the convolutional layers, we use kernels of 36*3*3 followed by batch normalization [2] and ReLU.

Table 1: Performance Comparison

matches \ winning rate	former winning rate(%)	latter winning rate(%)
rule based learning v.s. imitation learning	0	100
rule based learning v.s. reinforcement learning	32	68
imitation learning v.s. reinforcement learning	100	0

5 Performance

From Table 1, we can find that imitation learning always has a better performance while reinforcement learning is slightly better than rule based learning. The possible reasons are as below:

1. Due to the limitation of training time and high requirements for structural design, reinforcement learning is not as good as we expected.
2. Imitation learning is the best choice when we face the situation that we require an acceptable performance in limited training time.
3. Rule based learning is some kind of primitive, which leads to its disappointing performance. However, with more detailed rules added will rule based learning's performance be largely optimized.

It is worth mentioning that the use of supervised learning like imitation learning is not widely used in actual competitive games. This is mainly because this approach is highly dependent on the quality of the dataset and the performance of the trained model has a strict upper bound, i.e., it does not exceed the best part of the dataset.

In contrast, reinforcement learning, through a framework similar to self-supervised learning, can theoretically achieve the advanced strategies that the model may learn through continuous self-trial and error, allowing the agent to act with the optimal policy. Therefore, by combining deep learning with reinforcement learning strategies and finding a method with low training overhead will be some of the research directions for this kind of problems.

6 External Resources

1. Kaggle LuxAI Game interface, and game vitualization.
2. LuxAI competetion dataset between top-5 teams provided by Kaggle. (Dataset)
3. Pytorch.

References

[1] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[2] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456).

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Graves A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. In CoRR abs/1312.5602.

[4] Daniel, D. (2014). Reinforcement Learning and the Reward Engineering Principle. In AAAI.