Convolutional DQN Agent vs 2048

Camila Roa

Introduction

2048 is a single-player puzzle game made up by tiles with numeric values, it was originally published by Gabriele Cirulli in March 2014, and it was popularized after it was developed for smartphones two months after. Due to the structure of the game, composed of simple and easy to understand rules that create a complex environment, it has gathered interest of researchers in the field of Machine Learning (ML) to find an algorithm that can learn to play the game more effectively [1]. Different Reinforcement Learning (RL) algorithms have been proposed to master the game such as Monte Carlo Tree Search [2], Deep Reinforcement Learning (DRL) [3] and ensemble methods [4], to try to increase the score that an RL agent can reach playing 2048.

Currently, the methods that achieve the best performance in 2048 are ensemble methods that combine elements from temporal difference learning with n-tuple networks and tree-based search. N-tuple networks are a predefined feature extractor that helps the RL algorithm understand the board of the game. N-tuple networks have proven to be essential if the higher value tiles are to be achieved thus achieving higher scores. Using n-tuple networks as feature extractors combined with temporal difference learning was first proposed by Szubert and Jaśkowski in 2014, which allowed the model to reach a tile of value 2048 over 97% of the time [4]. Afterwards, a more comprehensive combination of techniques including n-tuple networks was proposed by Jaśkowski that reached 32.768 valued tiles 70% of the time [5].

On the other end, deep reinforcement learning methods have consistently lagged behind ensemble and tree search methods when it comes to performance [1]. For instance, when Deep Q Learning is used by itself, the trained models tend to not even reach the 2048 tile reliably. One of the earliest attempts of tackling 2048 with Deep Q Learning was made by Guei [7], who used a convolutional neural network architecture that was previously proposed to play Atari games [9], with 168.388 parameters. The model reached the 2048 tile only 7% of the time and the average score was around 2.000. A better result was achieved later by Chan [8] increasing the size of the neural network more than 500 times to amount to 96.726.020 parameters. But even with this significant increase, it achieved an average score of 10.000 but it still reached the 2048 tile less than 5% of the time.

Agents solely based on DRL methods consistently fails at the task of achieving the 2048 tile in the game unless they are accompanied by other methods such as n-tuple networks or tree-based search. This raises the question of what kind of complexities that arise from the game structure make a DQN agent based on convolutional neural networks unsuitable. There are 2 main sources of complexity in 2048: one is the randomness introduced when new tiles are placed on the board at each turn and the other one is the fact that patterns on the grid are not translation invariant which makes it harder for a convolutional neural network to deal with. The main motivation behind this project is to explore these complexities by first trying a simpler architecture that doesn't rely on features that are translation invariant and then introducing a simple variation in the rules of the game to analyze the impact in performance of a DQN agent based on convolutional neural networks that in normal conditions would not do well at the game. The rationale behind this methodology is that by introducing modifications to the environment that in

theory would improve the performance of a DQN agent, specific features of the game that make it unsuitable for this type of agent can be uncovered.

Related Work

The 2048 game is a widely studied problem in the field of reinforcement learning, since it is non-trivial to win, it has attracted the attention of many researchers. Despite its seemingly simple rules the game itself provides a complex environment for an RL agent to learn to maximize the reward in it. Complexity arises, on the one hand, from the random component in how the tiles get added to the grid at each turn, and in the other hand from the fact that the patterns on the board are not translation invariant. Research efforts have focused on finding an RL algorithm to train an agent that can achieve ever increasing scores and higher valued tiles on the board. As mentioned in the previous section, different types of RL algorithms have been tested on the game, from Monte Carlo Tree Search variations [2] to Deep Reinforcement Learning [3], such as Deep-Q Learning, to ensemble methods that combine them both [1][4].

The game is played in a 4 by 4 grid, in which each tile holds a numeric value. The game starts with 2 random tiles holding the value 2 and the rest are empty. At each turn, the player can slide the tiles in a specific direction (Left, Right, Up, Down) and then a new tile with the value 2 or 4 randomly appears on an empty spot on the board. When the player slides the tiles, same value tiles that end up adjacent to each other combine into a single tile with the sum of their values. The goal is to get a tile of the value 2048 in the board. The score starts from 0 and increases by the sum of the values of the tiles that get combined in a single turn.

In the context of Reinforcement Learning, the game can be modeled as a Markov decision process (MDP) where the agent plays the game by choosing among 4 actions which correspond to the 4 directions the tiles can slide, the state is the grid with values on each tile at a particular turn and the reward is related to the increase in score. The state updates depending on the action taken and the rules of the game (environment). Solving the problem means finding a policy that maps a state to an action that maximizes the cumulative reward [1].

State of the art algorithms for 2048 rely on temporal difference learning with n-tuple networks as feature extractors, which was first introduced in 2014 by Szubert and Jaśkowski and the algorithm achieved a tile of value 2048 97% of the time [4]. Since then, more sophisticated techniques have been tested to push the scores even higher, past obtaining a 2048 tile and aiming to the highest theoretical tile value of 13.1072. In [5], multiple techniques were combined to improve n-tuple based algorithms to achieve an average score of 609,104, furthermore, the algorithm could reach 32.768 valued tiles 70% of the time. However, methods based on deep neural networks have not achieved near the performance of the other techniques based on n-tuple networks yet and don't reach the 2048 tile reliably. The main difference between the approaches based on n-tuple networks is that they use a previously defined feature extractor whereas DRL methods learn the features during training.

An early attempt by Guei [7] explored Q-learning with deep neural networks based on the architecture originally proposed in [9] to play Atari games, that is made up of 3 convolutional layers with 64 2x2 kernels and 128 2x2 kernels, a fully connected layer with 256 neurons and an output layer with 4 neurons. This results in 168.388 parameters. The agent was trained with 100.00 games. The average scored obtained

by the algorithm was around 2.000, reaching the 2048 tile 7% of the time and the 1024 tile at a 10% rate. At the time, it was hypothesized that using a Deep-Q convolutional network solely to build an agent that can play the game effectively is not suitable for the structure of the 2048 game, given that patterns are not translation invariant, that is, they hold different meanings if they are on different parts of the board. So, a set of tiles of a specific value in the corner of the board should be treated differently than the same set of tiles at the center of the board. The possibility that the neural network did not have enough parameters to approximate the value function and therefore it was underfitting, was also stated.

A better result was obtained by Chan [8] with a network of 96.726.020 parameters, more than 500 larger than Guei's. The network was made up of 3 convolutional blocks of varying sizes, each containing 4 convolutional layers, in addition to a fully connected layer of 1024 units and the output layer of 4 units corresponding to the 4 possible actions of the game. The convolutional layers in each block of the network are not sequentially connected but rather their results are concatenated and passed onto the next block. In this case, the agent achieved an average score of 10.000 but it still reached the 2048 tile less than 5% of the time. Given the increase in size and consequently the longer training time it could be argued that Guei's network is more efficient.

The best results that have been obtained while using Deep Reinforcement Learning with convolutional layers rely on other RL techniques used as an ensemble, for instance, in [6] an ensemble method is proposed that combines game-tree search methods with a value network, it was trained for 240 hours using game specific features and it achieved an average score of 406.927.

Previous work has shown that a DQN agent, based on a convolutional neural network, while unaccompanied with other RL methods such as n-tuple networks or tree-based search is unsuitable for the game of 2048 if the goal is to reach the winning tile reliably. So, research efforts have focused on building larger convolutional networks that could discern the features better while using DQN or combining different RL methods to get an agent that can reach ever higher scores. In this project, rather than trying to train an agent that can surpass the state of the art methods, the aim is to explore why DQN agents based on convolutional neural networks that have shown to excel at more complex games with larger observation spaces consistently fail at 2048.

For the purposes of the project, 2 main sources of complexity in 2048 are defined: one is the randomness introduced when new tiles are placed on the board at each turn and the other one is the fact that patterns on the grid are not translation invariant which makes it harder for a convolutional neural network to deal with. The main motivation behind this project is to explore these complexities by first trying a different architecture that doesn't rely on features that are translation invariant and then introducing a simple variation in the rules of the game to analyze the impact in performance of a DQN agent based on convolutional networks that would normally perform badly at the game. The rationale behind this methodology is that by introducing modifications to the environment that in theory would improve the performance of a DQN agent, specific features of the game that make it unsuitable for this type of agent can be uncovered.

RL Environment

The environment of the game is the board where it is played, that is, each tile with its position and the value it holds, as well as the rules that govern it. The game is played in a 4 by 4 grid and the goal is to get a tile of the value 2048 in the board. The player can slide the tiles to combine same value tiles into one that holds the sum of their values. Only tiles that hold the same value and are adjacent after the slide can be combined. After 2 tiles are combined, they merge into a single tile that holds the summed value of the 2 previous tiles.

The grid is initialized with 2 tiles in random locations with 2 as value. All tiles are either empty or hold values that correspond to powers of 2. The possible actions a player can take is to slide the tiles either to the RIGHT, LEFT, UP or DOWN. When the player slides the tiles, if 2 adjacent tiles have the same value, a combination happens and the tiles merge into one with double the value of one of the previous tiles. A slide moves the tiles in the direction of the action as far as possible, either they reach an edge of the board or collide with another tile. After an action is taken, the board updates to account for the possible combinations and a new tile is introduced randomly into an empty spot, with value 2 or 4 with probabilities of 0.9 and 0.1 accordingly. An action is considered invalid if none of the tiles can move in the chosen direction.

Figure 1 shows a possible action for a given state in the 2048 game and the subsequent update of the board. By sliding the tiles to the left, the player manages to merge the 2 tiles with value 8 and the 2 tiles with value 2, the score increases by the values of the merged tiles (8*2 + 2*2), that is, 20.

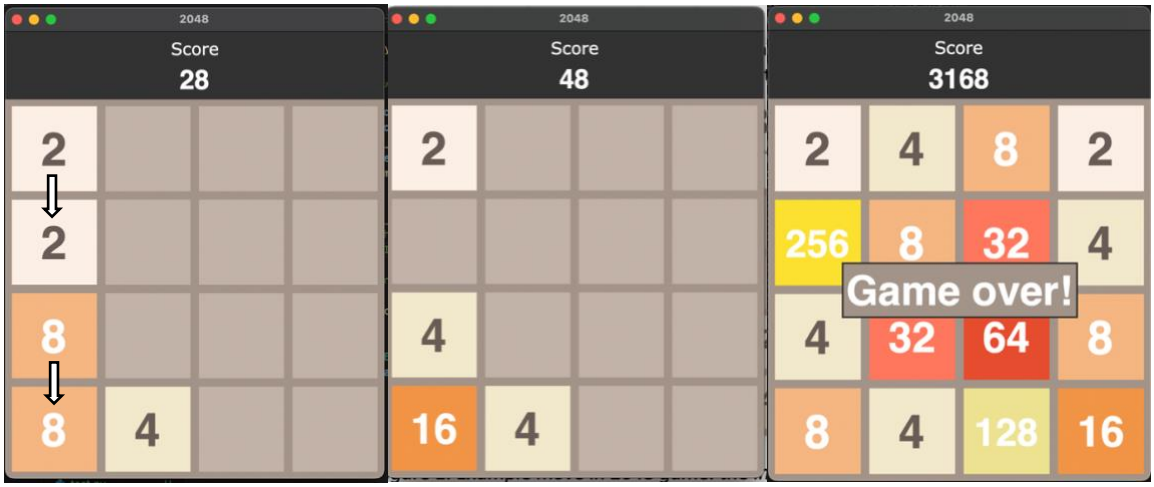

Figure 1. Example move in 2048 game: the initial state of the board is shown in the first image from left to right, then the player slides the tiles to the LEFT and the board updates to the second image, finally the third image shows the board when the game is over after losing.

Originally, the game can be played after winning to reach higher valued tiles and score, as shown in Figure 1, however, to limit training time in this implementation, a game is considered terminated when the first 2048 tile appears on the board.

Additionally, 3 different scenarios are proposed, to alter the rules of the game to evaluate the effect they have on the performance of the learning algorithm. The first one is to force the environment to always introduce a tile of value 2 at the end of each turn, eliminating the possibility of introducing a value 4 tile, therefore reducing some of the randomness of the game. The second variation introduced is to make the environment introduce the new tile at the end of each turn in one of the rightmost possible locations,

making the dynamic of the game more predictable. The third is eliminating all randomness from the game, so it adds a new tile on the rightmost topmost position.

The environment and GUI of the game were implemented in python, with the latter using tkinter. The GUI allows a person to play the game with letter keys as arrows (‘a’, ‘w’, ‘d’, ‘s’) or to choose a DQN agent to play and watch, which is stored as a Pytorch model that takes the one-hot encoded grid as an input and produces the action to take at a specific turn. It is worth to mention that the environment was implemented to be able to use the gym interface, with the step and reset methods, so that the learning algorithm was straightforward to write using Pytorch as suggested in its documentation.

Methodology

To explore why DQN agents based on convolutional neural networks that have shown to excel at more complex games with larger observation spaces but consistently fail in 2048, 3 experiments are proposed.

First, the convolutional neural network for the DQN agent proposed by Guei [7] is tested to get a baseline of performance. Comparing it as well with a random choice for an action and using sequence of Left-Down movements. Second to test the hypothesis that a DQN agent fails because convolutional neural networks cannot deal with features that are not translation invariant, the performance of a neural network with only fully connected layers is analyzed. Finally, the 2 variations mentioned in the previous section are put to the test with the convolutional network to see if decreasing the randomness in the game is enough for the network to improve performance or if the translation variance of the features is what hinders their ability to learn how to win 2048.

Action Space

The action space is discrete consisting of the 4 possible actions the agent can take at every turn of the game, that is, either sliding the tiles either to the RIGHT, LEFT, UP or DOWN.

Table 1. Action space for the 2048 game.

Action	Value	Meaning
LEFT	0	Slide tiles as far as possible to the left.
RIGHT	1	Slide tiles as far as possible to the right.
UP	2	Slide tiles as far as possible to the top.
DOWN	3	Slide tiles as far as possible to the bottom.

Observation Space

The observation space is the board itself where the game gets played, more specifically the 4 by 4 grid, in which each tile has a particular position and a value. However, it is not passed directly to the learning algorithm, it is first one-hot encoded into a tensor with multiple channels as proposed in [7]. So, the tensor has a size of 12x4x4, each of the 12 channels represent the position of a particular value on the board. The first channel has ones where there is an empty tile, the second channel has ones where there is a 2, the third channel has ones where there is a 4 and so on until the last channel which holds the position of the 2048 tiles.

This encoding is used directly as an input on the DQN agent with the convolutional neural network and flattened to use with the agent that is made up of a network with only fully connected layers.

Reward Function

The reward corresponds to the increase in the score of the game, which starts from 0 and on each turn, increases by an amount equal to the values of tiles that get combined or remains the same if there are no combinations.

To try to incentivize the algorithm to reach higher value tiles the reward was modified to take 2 conditions into account: the reward starts at 0 and if there are less tiles on the board at the end of a turn than at the start the reward is 1, if on top of that there is a new maximum tile on the board then the reward ins increased by another point and becomes 2.

RL algorithm

Standard Deep Q Learning is used, it was implemented with Pytorch using the following parameters:

- Replay memory of size 10.000
- Batch size of 32, which corresponds to the number of transitions that are sampled from the replay memory.
- Huber Loss, wich guarantees more robustness in the presence of outliers during training.
- Discount factor of 0.99
- Exponentially decaying rate of exploration: that starts at 0.9 and ends at 0.01. With a rate of decay of 100 (implementation specific, the higher the slower the decay).
- Update rate of the target network of 0.005
- AdamW optimizer with a learning rate of 0.00001

For the DQN agent based on convolutional neural networks, the architecture proposed by Guei [7] was chosen. It consists of 2 convolutional layers with 64 and 128 2x2 kernels, a fully connected layer of 256 neurons and an output layer of 4 neurons, each with a ReLu activation function except for the output layer, as shown in Figure 2. This amounts to 168.388 parameters. The first tensor represents the one-hot encoded input, the second tensor the output of the first convolutional layer, the third tensor the output of the second convolutional layer, then there is the output of the fully connected layer and then the final output, which represents the 4 possible actions in the game.

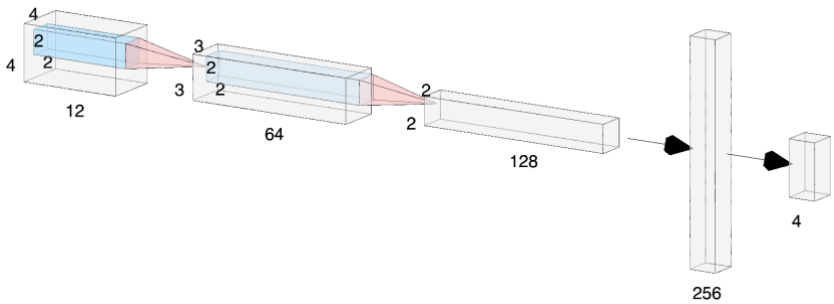

Figure 2. Architecture of the DQN agent based on a convolutional neural network.

The architecture of the DQN agent based on fully connected layers is shown on figure 3. The first tensor represents the one-hot encoded input after being flattened. Then, there are the outputs of the fully

connected layers of the network with 1024 neurons, 512 neurons, 256 neurons and 4 neurons in the output layer.

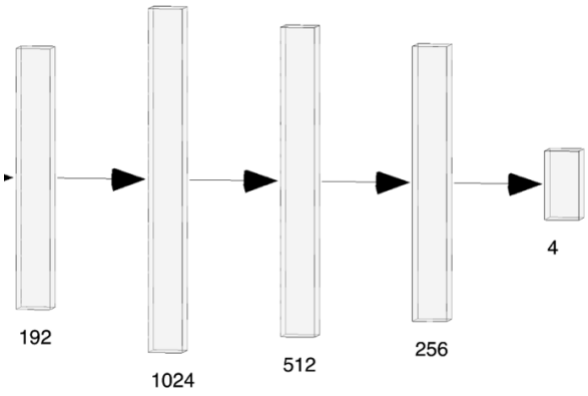

Figure 3. Architecture of the DQN agent based on a multilayer perceptron neural network.

Experiments and Results

First, the convolutional neural network for the DQN agent proposed by Guei [7] is tested to get a baseline of performance. The results for the network trained for 20.000 episodes (Figure 4) with a training time of over 11 hours show a similar but worse performance than that reported in the paper in which they trained the agent for 100.000 episodes. However, the increase in performance is negligible after 5.000 episodes, so this is not likely the reason for the degradation but more likely because they trained multiple models on different seeds and reported the best results. In this case, the average score obtained in a sequence of 1.000 games played by the trained agent is slightly higher than that reported by Guei at 2.381 but it does not reach the 2048 tile and it reaches the 512 tile less than 10% of the time (Figure 5). The results for the other experiments correspond to a training length of 5.000 episodes to reduce the execution time.

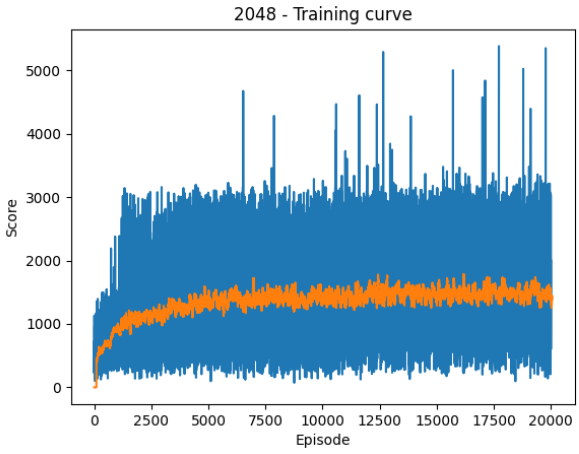

Figure 4. Learning curve of convolutional DQN agent trained for 20.000 episodes.

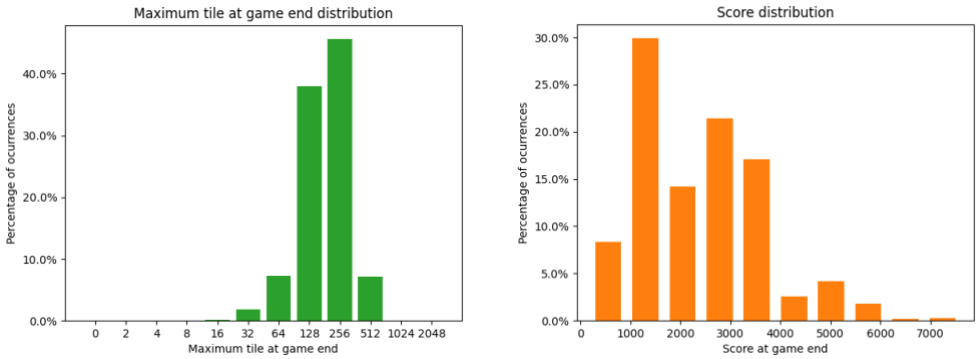

Figure 5. Rate at which the convolutional DQN agent reaches a certain value tile and a score.

Comparing these results to a policy that randomly chooses the action, shows some degree of mastery of the game by the DQN agent (Figure 6). Since the average score for this policy is 664 and the highest tile it reaches is the 256 in less than 1% of the time.

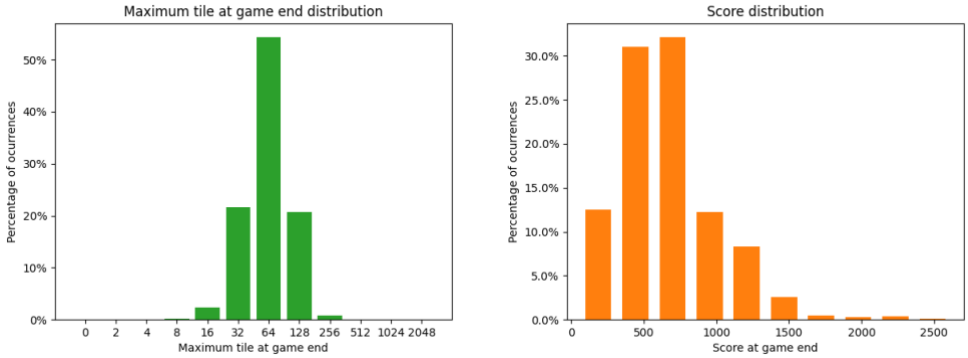

Figure 6. Rate at which a random policy reaches a certain value tile and a score.

The same can be said to a policy that chooses left and down slides continuously but to a lesser degree, since it achieves an average score of 1.668 and it reaches the 256 tile more frequently, around 5 % of the time, as shown in Figure 7.

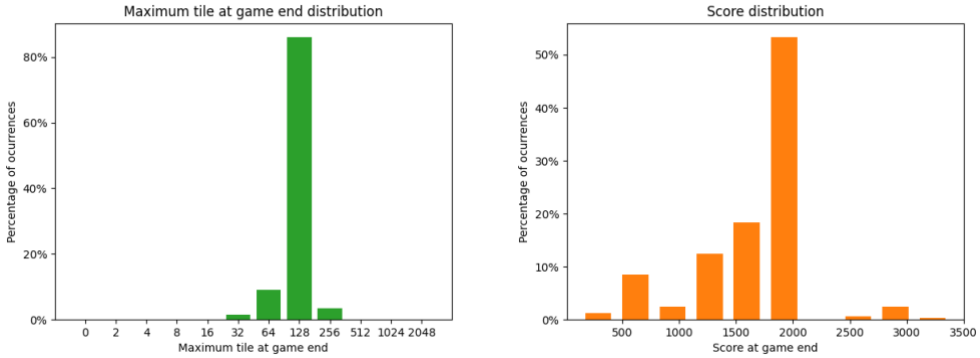

Figure 7. Rate at which a left-down policy reaches a certain value tile and a score.

As mentioned before, to test the hypothesis that a DQN agent fails because convolutional neural networks expect features that are translation invariant, a DQN agent based on a fully connected network was trained. The results are displayed on Figure 8. It achieved an average score of 1.802 and managed to reach the 512 tile less than 1% of the time.

Surprisingly, this model has a similar performance to the convolutional one while training and does slightly worse on the 1.000 games trial. Normally, this type of simpler model performs significantly worse than convolutional models when working on inputs that are grid shaped such as an image. But in this case the similarity in performance could indicate that the convolutional neural network cannot leverage the features it is extracting significantly better than the fully connected one, probably because the feature it must extract to model the game are not translation invariant. So, to a certain degree this confirms the theory that one of the aspects that make 2048 hard for convolutional neural networks is that a pattern on a particular position of the grid does not mean the same thing than the same pattern at a different location.

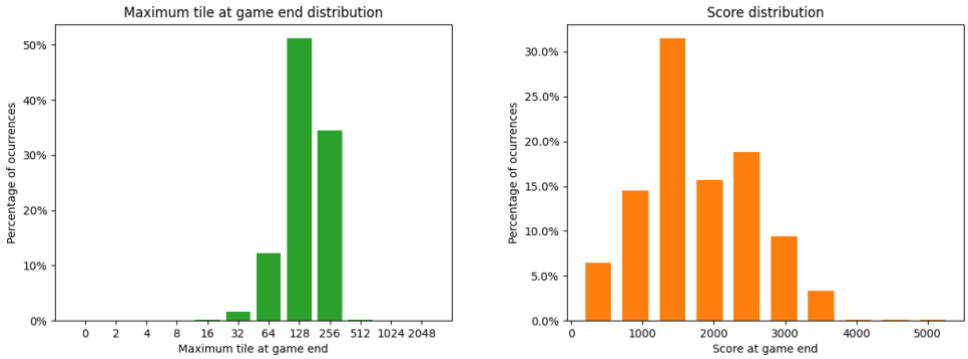

Figure 8. Rate at which the MLP DQN agent reaches a certain value tile and a score.

On top of the grid whose features are not translation invariant, randomness also adds complexity to the game which could be an aspect that hinders the ability of the convolutional DQN agent at winning the game. To check if a significant improvement could be gained by decreasing the randomness of the game, first the environment was modified to add new tiles at every turn always of value 2, eliminating the possibility of a new tile of value 4. An improvement in performance was obtained, with the agent reaching the 512 tile over 15% of the time with an average score of 3.087. However, it still does not reach the 2048 tile (Figure 10).

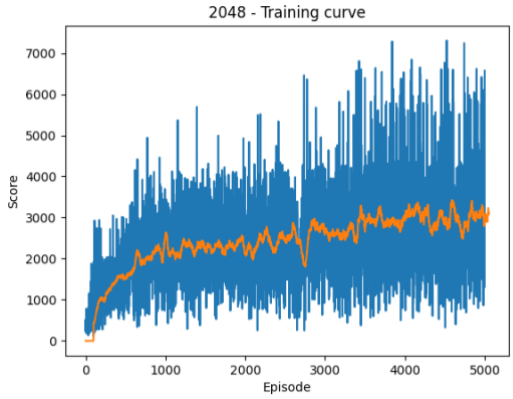

Figure 9. Learning curve of convolutional DQN agent trained for 5.000 episodes (Variation 1).

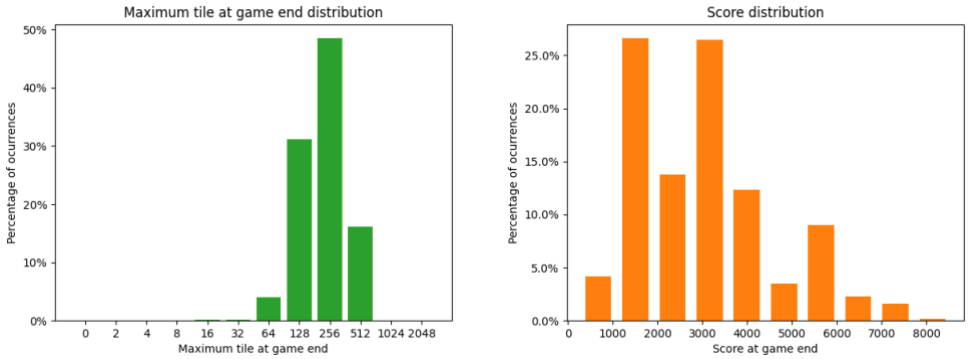

Figure 10. Rate at which the conv. DQN agent reaches a certain tile and a score (variation 1).

To minimize the randomness of the game even further, the agent was tested on an environment where the new tile at each turn was generated in the rightmost possible position. As expected, the average score during training increases bordering 4.000, which leads to an average score increase of x, yet it reaches the 512 tile with similar likelihood than the last variation.

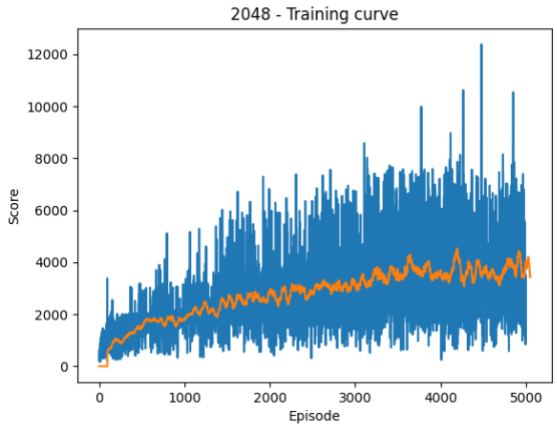

Figure 11. Learning curve of convolutional DQN agent trained for 5.000 episodes (Variation 2).

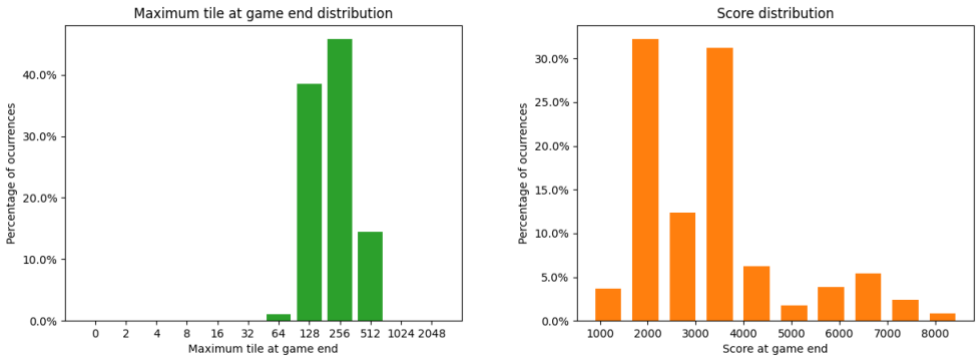

Figure 12. Rate at which the conv. DQN agent reaches a certain tile and a score (variation 2).

When the randomness of the game is fully eliminated and the game becomes predictable, the average score is higher at 4.104 but the algorithm only manages to get to a 256 random tile before the board is full and it loses the game. Even though the game is easily won by a human because it involves a predictable pattern that only evolves slightly as the game progresses (Figure 13.b). However, the training curve shows that the average score per episode keeps increasing so it raises the question of whether more training time would make the agent reach a higher valued tile (Figure 13.a).

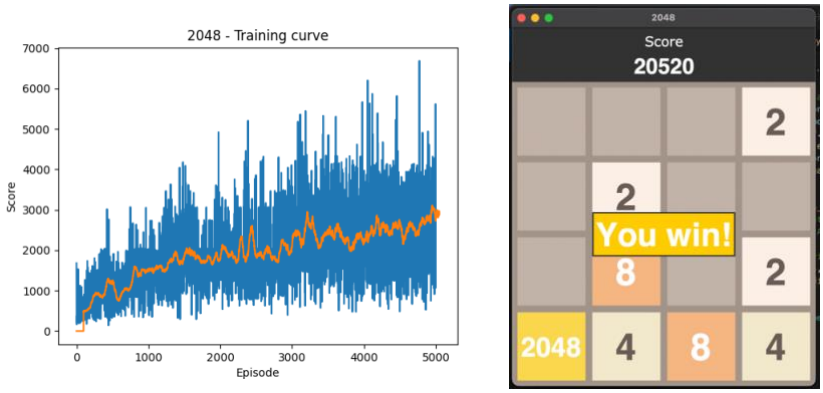

Figure 13. Learning curve of convolutional DQN agent trained for 5.000 episodes (Variation 3) and game won by a human with a predictable pattern.

In each of the cases, even though the score increases when complexity related to randomness is reduced or even eliminated, it seems like the model can still only reach up to a certain high-value tile. So, to try to incentivize the agent to reach higher value tiles the reward was modified to take 2 conditions into account: the reward starts at 0 and if there are less tiles on the board at the end of a turn than at the start the reward is 1, if on top of that there is a new maximum tile on the board then the reward ins increased by another point and becomes 2. With this modification and the fully predictable game, the highest score so far was reached at 9.096 and the agent reached a 512 tile, with the final board shown on Figure 14.

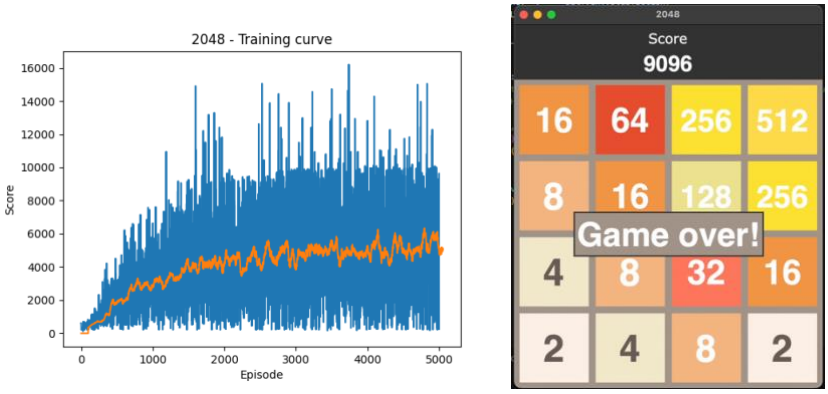

Figure 14. Learning curve of convolutional DQN agent trained for 5.000 episodes (Variation 4) and game played by AI reaching 512 tile.

Table 2 shows a summary of the results with each model, environment, or reward variation. The 3rd column contains the value of the highest tile reached by that model trained on a specific reward and environment and the 4th column shows how often it reached that tile on a trial of 1.000 games. The 5th column contains the percentage change of the average score obtained by the models with respect to the original convolutional neural network trained on the original environment.

Even while training the agent on an environment that eliminated the randomness of the game, the average score increased at most 72% while reaching a maximum tile of 256. Only by modifying the reward to try to incentivize the model to combine more tiles this emptying the board and reaching a higher valued tile, the average score increased by 282% reaching the 512 tile. So even though the complexity of the game was reduced, the agent cannot seem to understand the structure of the game because the convolutional neural network cannot use location information embedded in the features since it expects translation invariant features. So a separate feature extractor such as an n-tuple network is needed to achieve a higher value tile accompanied by a higher score.

Based on the results obtained with the reward variation, it raises the question of whether the increase of score in each turn is an appropriate reward if we want the model to prioritize clearing the board with multiple combinations at once while reaching higher value combinations.

Table 2. Summary of results for different models, environment, and reward variations.

Model	Environment	Highest tile	% of the time	Average score	% change
Convolutional	Original	512	7%	2.381	-
Random	Original	256	2%	664	-72.1%
Left-Down	Original	256	5%	1.668	-29.9%
Fully	Original	512	1%	1.802	-24%
Convolutional	Variation 1 (spawns only 2's)	512	15%	3.087	+29.7%

Convolutional	Variation 2 (1 + rightmost)	512	15%	3.663	+53.8%
Convolutional	Variation 3 (1 + 2 + topmost)	256	100%	4.104	+72.3%
Convolutional	Variation 4 (3 + Reward 2)	512	100%	9.096	+282.0%

Conclusion and Future Work

In this project, a DQN agent with model, environment and reward variations was explored for the game of 2048. Even though at first sight this RL method with a convolutional neural network seems appropriate, given that the state of the game is kept in an image like grid and this setup excels at other games with a considerably larger state space such as Atari games, its performance lags behind other techniques. This raises the question of what kind of complexities arise from the game structure that make a DQN agent based on convolutional neural networks unsuitable for 2048.

First, a known architecture proposed in [1] and used before for Atari games was tested, the results reflect a better performance than a random or left-down policy. But the agent does not reach the 2048 tile in neither of the trials, reaching at most 512. To explore why this type of agent is unsuitable for the game, 2 main sources of complexity were identified: one is the randomness introduced when new tiles are placed on the board at each turn and the other one is the fact that patterns on the grid are not translation invariant which makes it harder for a convolutional neural network to deal with.

With these complexities in mind, different experiments were performed. The first attempted to address the complexity that comes from the features not being translation invariant, so a network with just fully connected layers was evaluated and it obtained a similar performance than the convolutional model while being a simpler architecture. Afterwards, the other source of complexity was addressed, modifying the environment to reduce the randomness of the game, better scores were obtained but the increase was of 70% at the most and it still reached the 512 tile at most. The largest increase in score of 282% without randomness was obtained by modifying the reward to prioritize the act of reducing the number of tiles in each turn by combination and reaching a higher tile, rather than using the increase in score as a reward.

In conclusion, a DQN agent for 2048 without other reinforcement learning methods to accompany is nowhere near the performance of state of the art methods based on n-tuple networks. This coupled with the result obtained in this project with a fully connected network, indicates the convolutional model, even though it achieves a certain degree of mastery of the game, cannot use the features it detects effectively to win the game since it expects translation invariant features. As far as the results obtained reducing the randomness of the game, the increases in score were not significant unless the reward was modified, so even in decreased complexity the network could not learn the strategy to win the game. This raises the question of whether the increase of score in each turn is an appropriate reward which could be explored with different RL methods and network architectures, with the reward proposed by this project or different variations.

References

[1] H. Guei, “On Reinforcement Learning for the Game of 2048,” IEEE Comput. Intell. Mag., vol. 15, no. 3, pp. 60–73, Aug. 2020, doi: 10.1109/MCI.2020.2998315.

[2] P. Rodgers and J. Levine, “An investigation into 2048 AI strategies,” in 2014 IEEE Conference on Computational Intelligence and Games, Dortmund, Germany: IEEE, Aug. 2014, pp. 1–2. doi: 10.1109/CIG.2014.6932920.

[3] T. Boris and Š. Goran, "Evolving neural network to play game 2048," in 2016 24th Telecommunications Forum (TELFOR), Nov. 2016, pp. 1–3. doi: 10.1109/TELFOR.2016.7818911.

[4] M. Szubert and W. Jaśkowski, “Temporal difference learning of N-tuple networks for the game 2048,” in 2014 IEEE Conference on Computational Intelligence and Games, Aug. 2014, pp. 1–8. doi: 10.1109/CIG.2014.6932907.

[5] W. Jaśkowski, “Mastering 2048 with Delayed Temporal Coherence Learning, Multi-Stage Weight Promotion, Redundant Encoding and Carousel Shaping.” arXiv, Dec. 12, 2016. doi: 10.48550/arXiv.1604.05085.

[6] K. Matsuzaki, “Developing Value Networks for Game 2048 with Reinforcement Learning,” Journal of Information Processing, vol. 29, no. 0, pp. 336–346, 2021, doi: 10.2197/ipsjjip.29.336.

[7] Guei, H., Wei, T., Huang, J.-B. and Wu, I.-C.: An Early Attempt at Applying Deep Reinforcement Learning to the Game 2048, Workshop on Neural Networks in Games (2016).

[8] Chan: AI now plays 2048!, available from ⟨ https://github.com/qwert12500/2048_rl/tree/main ⟩.

[9]V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “ Playing Atari with Deep Reinforcement Learning,” 20 3, arXiv:1312.5602.